Для системных администраторов UNIX

3-е издание

TCP/IP

Сетевое администрирование

Символ®

O'REILLY®

Крэйг Хант

TCP/IP

Network Administration

Third Edition

Craig Hunt

O'REILLY®

TCP/IP

Сетевое администрирование

Третье издание

Крэйг Хант

Санкт-Петербург — Москва
2004

Крэйг Хант

TCP/IP. Сетевое администрирование, 3-е издание

Перевод М. Зислиса

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*С. Маккавеев*
Редакторы	*А. Лосев, А. Петухов*
Корректор	*С. Беляева*
Верстка	*Н. Гриценко*

Крэйг Хант
TCP/IP. Сетевое администрирование, 3-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2004. – 816 с., ил.
ISBN 5-93286-056-1

Третье издание книги «TCP/IP. Сетевое администрирование» – это полноценное руководство по настройке и сопровождению сети TCP/IP, которое предназначается как системным администраторам, так и пользователям домашних компьютеров с доступом к сети Интернет. Повествование начинается с основ: зачем нужны протоколы, как они работают, как адресация и маршрутизация позволяют передавать данные по сети и как настроить сетевое соединение.

Помимо базовой настройки книга рассказывает о современных протоколах маршрутизации (RIPv2, OSPF и BGP) и пакете gated, который реализует работу с ними. Кроме того, книга является руководством по настройке многих важных сетевых служб, в том числе DNS, Apache, sendmail, Samba, PPP и DHCP. Две главы посвящены безопасности и разрешению проблем. Третье издание включает новую главу, посвященную настройке сервера Apache и раздел, в котором обсуждается настройка Samba с целью организации совместного доступа к файлам и принтерам в гетерогенной сети Unix/Windows. Справочные приложения подробно описывают синтаксис таких программ, как gated, pppd, named, dhcpd и sendmail. Книга охватывает реализации TCP/IP для систем Linux, Solaris, BSD и System V.

ISBN 5-93286-056-1
ISBN 0-596-00297-1 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 08.01.2004. Формат 70x100 1/16 . Печать офсетная.
Объем 51 печ. л. Тираж 2000 экз. Заказ N
Отпечатано с диапозитивов в Академической типографии «Наука» РАН
199034, Санкт-Петербург, 9 линия, 12.

Посвящается Алане. Ты начало новой жизни.

Оглавление

Предисловие

Первое издание книги «TCP/IP. Сетевое администрирование» было написано в 1992 году. За истекшие десять лет многое изменилось, но некоторые вещи остались все теми же. TCP/IP по-прежнему сохраняет свое лидерство среди протоколов связи, объединяющих разнотипные компьютерные системы. Он остается фундаментом для взаимодействия и обмена данными, для глобальных компьютерных сетей. Примечательно, что протоколы IP (Internet Protocol, протокол Интернета, или межсетевой протокол), TCP (Transmission Control Protocol, протокол управления передачей) и UDP (User Datagram Protocol, протокол пользовательских дейтаграмм), составляющие базу TCP/IP, не изменились. Изменились способы применения TCP/IP и управления этими протоколами.

Символичен для этих перемен тот факт, что дома у моей тещи есть подключение к сети TCP/IP, которое позволяет ей обмениваться электронной почтой, изображениями и гипертекстовыми документами с другими людьми своего поколения. Для нее это просто «выход в Интернет», но правда такова, что в ее домашней машине реализован полноценный стек протоколов TCP/IP, работает динамическое получение IP-адреса, а кроме того, используются типы данных, которые десять лет назад попросту не существовали.

В 1991 году протоколы TCP/IP были инструментом для опытных пользователей. Сетевые администраторы заведовали ограниченным числом систем и могли рассчитывать, что пользователи этих систем обладают определенным уровнем специальных знаний. Но это в прошлом. В 2002 году потребность в профессиональных сетевых администраторах выше, чем когда-либо ранее, поскольку контингент пользователей становится все более разношерстным и не столь подготовленным к самостоятельному решению технических проблем. В этой книге содержится информация для тех, кто хочет эффективно решать задачи сетевого администрирования TCP/IP.

«TCP/IP. Сетевое администрирование» стала первым сборником полезной информации для профессиональных сетевых администраторов TCP/IP и по сей день остается лучшей из подобных книг. За первым изданием последовал целый поток книг о TCP/IP и Интернете. Однако очень немногие из них сосредоточены на том, что действительно необходимо знать системному администратору об администрировании TCP/IP. Большинство книг – либо академические тексты, написанные с точки зрения архитектора протокола,

либо инструкции по использованию приложений TCP/IP. В них отсутствует практическая информация о сетях, которая необходима системным администраторам Unix. В настоящей книге сделан упор на TCP/IP и Unix, а также на поиск правильного соотношения между теорией и практикой.

Я горжусь предшествующими изданиями этой книги. Что же касается настоящего издания, я постарался сделать все возможное, чтобы не только сохранить настрой книги, но и улучшить ее. Рассмотрено динамическое назначение адресов при помощи протокола DHCP (Dynamic Host Configuration Protocol, протокол динамической настройки узлов). Материал, посвященный системе доменных имен (DNS), теперь охватывает BIND версии 8 и, в меньшей степени, BIND 9. Настройка электронной почты рассмотрена на примере текущей версии sendmail (8), а примеры, связанные с операционной системой, базируются на текущих версиях Solaris и Linux. Из протоколов маршрутизации описаны RIPv2 (Routing Information Protocol version 2, протокол маршрутной информации версии 2), OSPF (Open Shortest Path First, протокол предпочтения кратчайшего пути) и BGP (Border Gateway Protocol, протокол граничных шлюзов). Кроме того, добавлена глава, посвященная настройке веб-сервера Apache, новый материал по `xinetd`, а также информация о создании брандмауэров на базе `iptables`. Отмечу, что эти дополнительные темы не очень сильно увеличили объем книги.

TCP/IP – это набор протоколов связи, определяющих правила общения различных видов компьютеров между собой. «TCP/IP. Сетевое администрирование» – книга о том, как создать собственную сеть на базе TCP/IP. Эта книга является одновременно руководством, отвечающим на вопросы «как» и «почему» из области сетей TCP/IP, и справочником по отдельным сетевым приложениям.

Для кого эта книга

Эта книга предназначена всем владельцам Unix-машин, подключенных к сети TCP/IP.[1] Очевидно, в эту категорию попадают администраторы сетей и систем, отвечающие за настройку и сопровождение машин сети, но также и пользователи, которые желают узнать, каким образом их компьютеры общаются с другими системами. Провести границу между «системным администратором» и «конечным пользователем» довольно сложно. Человек может считать себя пользователем, но, работая на Unix-машине, ему, скорее всего, приходится заниматься и задачами системного администрирования.

В последние несколько лет, словно грибы после дождя, появляются книги для «чайников» и «идиотов». Эта книга не для людей, которые считают себя «идиотами» в отношении Unix. Кроме того, эта книга едва ли пригодится

[1] Большая часть текста применима не только к Unix-системам. Многие форматы файлов и команды, а также все описания протоколов справедливы для операционных систем Windows 98/NT/2000 и других. Администраторам NT-систем можно порекомендовать книгу «Windows NT TCP/IP Network Administration», O'Reilly.

«гениям» от сетевого администрирования. Однако читатели, не относящиеся к этим крайностям, найдут в книге немало полезной информации.

Предполагается, что читатели хорошо разбираются в работе компьютеров и знакомы с основами администрирования Unix-систем. Если это не так, изучить основы поможет книга Элин Фриш (Jeen Frisch) «Essential System Administration» (Основы системного администрирования), O'Reilly, серия Nutshell Handbook).

Структура книги

Книга состоит из трех логических частей: основные понятия, руководство, справочник. Три первых главы в общих чертах рассказывают о протоколах и службах TCP/IP. Они содержат основные понятия, необходимые для понимания последующих глав. Последующие главы содержат практические инструкции по различным темам. Главы с 4 по 7 посвящены планированию сети и настройке основных программных пакетов, необходимых для ее работы. Главы с 8 по 11 рассказывают о настройке основных сетевых служб. Главы 12 и 13 – о двух насущных вопросах, связанных с обеспечением надежной работы сети: безопасности и разрешении проблем. Завершает книгу ряд приложений-справочников, посвященных важным командам и программам.

Книга состоит из следующих глав:

Глава 1 «Обзор TCP/IP» содержит историю TCP/IP, описание архитектуры протокола, а также объясняет принципы его функционирования.

Глава 2 «Доставка данных» описывает адресацию и передачу данных по сети адресатам.

Глава 3 «Сетевые службы» посвящена отношениям между системами клиент–сервер, а также различным службам, жизненно важным для существования современной сети Интернет.

Глава 4 «Начинаем работу» открывает обсуждение установки и настройки сети, рассказывая о предварительном планировании, которое является необходимым шагом при создании сетей.

Глава 5 «Базовая настройка» посвящена вопросам настройки TCP/IP на уровне ядра Unix и настройки системы для запуска сетевых служб.

Глава 6 «Настройка интерфейса» расскажет о том, как связать сетевой интерфейс и сетевое программное обеспечение. Глава содержит примеры настройки интерфейсов Ethernet и PPP.

Глава 7 «Настройка маршрутизации» описывает, как организовать маршрутизацию, позволяющую машинам сети корректно взаимодействовать с другими сетями. В частности, освещены статические таблицы маршрутизации, распространенные протоколы маршрутизации, а также `gated` – пакет, реализующий последние версии некоторых из протоколов маршрутизации.

Глава 8 «Настройка DNS» посвящена администрированию программы сервера имен, который преобразует имена машин сети в адреса Интернета.

Глава 9 «Локальные службы сети» описывает настройку многих из распространенных сетевых серверов, в частности сервера настройки DHCP, сервера печати LPD, почтовых серверов POP и IMAP, сетевой файловой системы NFS (Network File System), серверов файлов и печати Samba, а также сетевой информационной службы NIS (Network Information System).

Глава 10 «sendmail» рассказывает о настройке sendmail – демона, отвечающего за доставку сообщений электронной почты.

Глава 11 «Настройка Apache» описывает настройку веб-сервера Apache.

В главе 12 «Сетевая безопасность» речь идет о том, как использовать современный Интернет, не подвергаясь излишнему риску. Глава посвящена угрозам безопасности, связанным с работой в сети, и возможным способам защиты от них.

Глава 13 «Разрешение проблем TCP/IP» рассказывает, какие действия можно предпринять, если что-то идет не так. Описаны методы и инструменты диагностирования проблем TCP/IP, приводятся примеры реальных проблем и их решений.

Приложение A «Инструментарий PPP» – это справочное руководство по различным программам, применяемым в настройке последовательных портов для работы по TCP/IP. Описаны программы `dip`, `pppd` и `chat`.

Приложение B «gated, справочник» – это справочное руководство по языку настройки пакета маршрутизации `gated`.

Приложение C «named, справочник» – это справочное руководство по серверу имен BIND (Berkeley Internet Name Domain).

Приложение D «dhcpd, справочник» – это справочное руководство по демону `dhcpd` (Dynamic Host Configuration Protocol Daemon).

Приложение E «sendmail, справочник» является справочным руководством по синтаксису, параметрам и ключам настройки sendmail.

В приложении F «Файл httpd.conf в Solaris» приводится содержимое файла настройки Apache, о котором идет речь в главе 11.

Приложение G «Выдержки из RFC» содержит информативные справочные фрагменты по протоколам из документов RFC, которые дополняют примеры диагностирования и разрешения проблем главы 13. Кроме того, в приложении содержатся сведения о том, где взять полноценные документы RFC.

Версии Unix

Большинство примеров книги относятся к Red Hat Linux, наиболее популярному в настоящее время дистрибутиву Linux, а также к Solaris 8, операционной системе от Sun, основанной на Unix System V. По счастью, программные средства TCP/IP достаточно стандартны в разных системах, что делает приво-

димые примеры универсальными – они должны работать в любых системах Linux, System V и BSD. Незначительные различия в выходных данных команд и параметрах командной строки не должны представлять затруднений.

Некоторые дополнительные сетевые приложения имеют собственные номера версий, не зависящие от версий операционных систем. Номера версий дополнительных пакетов упоминаются, когда это уместно. Наиболее важные из подобных пакетов:

BIND

Приводимое описание пакета BIND относится к версиям ветви 8, работающим в ОС Solaris 8. BIND 8 – это версия пакета BIND, поставляемая в составе Solaris и поддерживающая все стандартные типы записей ресурсов. В отношении базовых настроек версии BIND 8 и более новая BIND 9 не сильно различаются.

sendmail

Информация о пакете sendmail соответствует версии 8.11.3 и должна быть применима для всех других версий sendmail ветви 8.

Типографские соглашения

В книге использованы следующие типографские соглашения:

Курсив

Применяется для отображения имен файлов, каталогов, узлов, доменов, а также для выделения новых терминов.

`Моноширинный шрифт`

Применяется для отображения содержимого файлов и вывода команд, а также для выделения команд, параметров и ключевых слов в тексте.

`Моноширинный полужирный шрифт`

Применяется в примерах для выделения команд, набираемых пользователем.

`Моноширинный курсив`

Используется в примерах и в тексте для выделения переменных, значения которых должны быть подставлены в зависимости от обстоятельств. (Например, переменную *`filename`* необходимо заменять конкретным именем файла.)

`%, #`

Команды, вводимые в диалоговом режиме, отмечены приглашением стандартного интерпретатора команд C shell (`%`). Если команда должна выполняться с полномочиями администратора, она отмечается стандартным приглашением суперпользователя (`#`). В примерах, где участвует сразу несколько машин сети, приглашению может предшествовать имя той машины, на которой выполняется команда.

`[ ключ ]`

В описании синтаксиса команд необязательные аргументы заключаются в квадратные скобки. Так, запись `ls [ -l ]` означает, что ключ `-l` является необязательным.

Нам важно знать ваше мнение

Мы тщательно, насколько представляется возможным, проверили всю информацию в настоящей книге, но вы можете обнаружить, что возможности программ изменились (или наши ошибки!). Пожалуйста, сообщайте обо всех найденных ошибках, а также присылайте предложения, связанные с последующими изданиями книги по адресу:

O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (в США или Канаде)
(707) 829-0515 (международный/местный)
(707) 829-0104 (факс)

Издательством O'Reilly создана веб-страница, посвященная этой книге, на которой доступна информация о найденных ошибках и будут появляться разнообразные дополнительные сведения. Страница доступна по адресу:

http://www.oreilly.com/catalog/tcp3

Комментарии и технические вопросы, связанные с книгой, присылайте по адресу электронной почты:

bookquestions@oreilly.com

На веб-сайте издательства O'Reilly представлена дополнительная информация о книгах, конференциях, программном обеспечении, источниках информации и Сети O'Reilly (O'Reilly Network):

http://www.oreilly.com

Чтобы узнать, чем еще занимается Крэйг Хант, посетите его веб-сайт, *http://www.wrotethebook.com*.

Благодарности

Я хотел бы поблагодарить многих людей, которые помогли подготовить эту книгу. Те, кто участвовал в подготовке двух первых изданий, в первую очередь заслуживают благодарности; их вклад живет и в этом издании. Первое издание: Джон Уок (John Wack), Мэтт Бишоп (Matt Bishop), Вьетс Венема (Wietse Venema), Эрик Оллман (Eric Allman), Джефф Хониг (Jeff Honig),

Скотт Брим (Scott Brim) и Джон Дорган (John Dorgan). Второе издание: снова Эрик Оллман, Брайан Косталес (Bryan Costales), Крикет Ли (Cricket Liu), Пол Альбитц (Paul Albitz), Тед Лемон (Ted Lemon), Элизабет Цвики (Elizabeth Zwicky), Брент Чепмен (Brent Chapman), Симсон Гарфинкель (Simson Garfinkel), Джефф Седайо (Jeff Sedayao), а также Элин Фриш (Жleen Frisch).

В третьем издании книга стала лучше благодаря участию людей, многие из которых сами являются авторами. Они не только помогли мне с техническими вопросами – благодаря им я стал лучше писать. Три автора заслуживают отдельной благодарности. Многочисленные комментарии Крикета Ли, одного из авторов лучшей в мире книги о DNS, позволили улучшить раздел, посвященный системе доменных имен. Дэвид Колье-Браун (David Collier-Brown), один из авторов книги «Using Samba», написал исчерпывающую техническую рецензию на материал по Samba. Чарльз Олдс (Charles Aulds), автор бестселлера об администрировании сервера Apache, оказал помощь при написании главы о настройке Apache. Все эти люди помогли мне улучшить книгу в третьем издании. Спасибо!

Сотрудники издательства O'Reilly & Associates постоянно помогали мне. Деб Кэмерон (Deb Cameron), мой редактор, заслуживает отдельной благодарности. Ее усилий хватало и на развитие книги, и на общение со своей новорожденной красавицей, Вифанией Розой. Эмили Квилл (Emily Quill) играла роль выпускающего редактора и куратора проекта. Джефф Холкомб (Jeff Holcomb) и Джейн Эллин (Jane Ellin) выполняли проверку качества. За техническую помощь спасибо Леанне Сойлемез (Leanne Soylemez). Том Динз (Tom Dinse) создал указатель. Автором обложки является Эди Фридман (Edie Freedman), а стилевое оформление текста делала Мелани Вонг (Melanie Wang). Нил Уоллз (Neil Walls) занимался преобразованием текста из формата Microsoft Word в формат редактора Framemaker. Иллюстрации предшествующих изданий, созданные Крисом Райли (Chris Reilley) и Робертом Романо (Robert Romano), были обновлены стараниями Роберта Романо и Джессамина Рида (Jessamyn Read).

Наконец, я хочу поблагодарить мою семью – Кэти, Сару, Дэвида и Ребекку. Только их стараниями давление сроков сдачи материала до сих пор не свело меня с ума. Ребята, вы лучше всех.

1

Обзор TCP/IP

- *TCP/IP и Интернет*
- *Модель обмена данными*
- *Архитектура протоколов TCP/IP*
- *Уровень доступа к сети*
- *Уровень Internet*
- *Транспортный уровень*
- *Прикладной уровень*

Любой, кто пользуется настольной системой Unix – будь то инженер, преподаватель, ученый или деловой человек, – избрал в качестве второго занятия системное администрирование. Работа с сетями на подобных компьютерах ставит перед нами еще и задачи, связанные с сетевым администрированием.

Сетевое администрирование и системное администрирование не тождественны. Задачи системного администрирования – создание новых пользователей или резервных копий – ограничены одной независимой компьютерной системой. Дела обстоят совсем иначе в сетевом администрировании. Как только компьютер подключается к сети, он начинает взаимодействовать со многими другими системами. Качество выполнения задач сетевого администрирования оказывает определенное влияние не только на локальную систему, но и на многие системы сети, а следовательно, твердое понимание основ сетевого администрирования приносит пользу всем.

Объединение компьютеров в сети невероятно повышает их способность к общению, – а большинство компьютеров используются в основном для обмена данными, а не для вычислений. Вычислительными задачами для науки и бизнеса занимаются многочисленные суперкомпьютеры, но их число бледнеет в сравнении с миллионами систем, занятых под задачи вроде отправки почтовых сообщений или извлечения данных из удаленного хранилища. Более того, если оценить число настольных систем, используемых преимущественно для подготовки документов, позволяющих передавать мысли и идеи другим людям (речь идет о сотнях миллионов машин), станет понятно, почему компьютеры можно рассматривать как устройства для обмена информацией.

Положительное влияние компьютерных коммуникаций приводит к росту числа и видов компьютеров, участвующих в работе сетей. Одним из главных преимуществ TCP/IP является возможность прозрачного сообщения всех видов аппаратного обеспечения и операционных систем.

Название «TCP/IP» связано с целым семейством протоколов передачи данных. Оно происходит от названий двух протоколов этого семейства: протокола управления передачей (TCP, Transmission Control Protocol) и протокола Internet (IP, Internet Protocol). TCP/IP – это традиционное имя семейства протоколов, которое и используется в книге. Другое название – семейство протоколов Internet (IPS, Internet Protocol Suite) – также является приемлемым.

Настоящая книга содержит практические пошаговые инструкции по настройке и сопровождению сетевых приложений TCP/IP для компьютерных Unix-систем. TCP/IP является ведущей технологией создания локальных и корпоративных сетей, а также фундаментом всемирной сети Интернет. TCP/IP – самый важный программный комплекс для сетевого администратора Unix.

Первая часть книги рассказывает об основах TCP/IP и принципах передачи данных по сети. Вторая часть посвящена настройке и применению TCP/IP в системе Unix. Начнем с краткого исторического экскурса.

TCP/IP и Интернет

В 1969 году управление передовых исследований (Advanced Research Projects Agency, ARPA) финансировало исследования и разработку в рамках проекта по созданию экспериментальной сети на базе коммутации пакетов. Эта сеть, получившая имя *ARPAnet*, создавалась с целью изучения методов обеспечения устойчивой, надежной, не зависящей от оборудования передачи данных. Многие из применяемых сегодня методов передачи данных родились в недрах ARPAnet.

Экспериментальная сеть имела такой успех, что многие из организаций-участниц проекта начали использовать ее возможности на постоянной основе. В 1975 году ARPAnet из экспериментальной сети превратилась в рабочую, а административные полномочия были переданы управлению оборонных коммуникаций (Defense Communications Agency, DCA).[1] Однако развитие ARPAnet не прекратилось после смены статуса: базовые протоколы TCP/IP были разработаны несколько позже.

Протоколы TCP/IP были приняты в качестве военных стандартов (Military Standards, MIL STD) в 1983 году, и в этот момент всем подключенным к сети узлам предписывалось перейти на новые протоколы. Для того чтобы облегчить переход, управление DARPA[2] финансировало Болта, Беранека и Ньюмана (Bolt, Beranek, Newman; BBN), которые реализовали TCP/IP для системы Berkeley (BSD) Unix. Так начался союз систем Unix и TCP/IP.

---

[1] Управление DCA в настоящее время носит название управления оборонных информационных систем (Defense Information Systems Agency, DISA).

[2] В 80-е годы управление ARPA, входящее в состав Министерства обороны США, получило новое название: Defense Advanced Research Projects Agency (DARPA). Вне зависимости от названия, перед управлением во все времена стояла одна задача – финансирование передовых исследований.

Термин *Internet* вошел в употребление примерно в то же время, когда были приняты стандарты по TCP/IP. В 1983 году сеть ARPAnet была поделена на MILNET, рассекреченную часть оборонной информационной сети (Defense Data Network, DDN), и новую сеть ARPAnet, размерами поменьше предыдущей. Термин «Internet» использовался для обозначения сети в целом: MILNET плюс ARPAnet.

В 1985 году национальным научным фондом (National Science Foundation, NSF) была создана сеть NSFNet, которая подключилась к существовавшей в то время сети Internet. Изначально сеть NSFNet объединяла пять суперкомпьютерных центров NSF. Она уступала ARPAnet размерами, равно как и скоростью (56 Кбит/с). И все же создание NSFNet стало значительным событием в истории сети Internet, поскольку фонд NSF создал не только сеть, но и новое видение того, как можно использовать Internet. Идея NSF заключалась в том, чтобы каждый ученый и каждый инженер в США получил доступ к сети. С этой целью в 1987 году фонд NSF создал новую, трехзвенную топологию сети, которая объединяла магистральные, региональные и локальные сети. В 1990 году сеть ARPAnet перестала существовать формально, в 1995 году сеть NSFNet утратила свою роль первичной магистральной сети Internet.

Сегодня сеть Интернет имеет невообразимые размеры и состоит из сотен тысяч сетей, разбросанных по всему миру. Глобальная сеть больше не зависит от базовой (или магистральной) сети или от работы правительственных структур. Современный Интернет – это творение поставщиков услуг, работающих на коммерческой основе. Национальные поставщики сетевых услуг, то есть поставщики первого звена, а также региональные поставщики услуг создают собственно инфраструктуру. Поставщики услуг Интернета (Internet Service Providers, ISPs) обеспечивают локальный доступ и обслуживание пользователей. Сеть сетей сходится воедино на территории США, в точках доступа к сети (Network Access Points, NAPs).

Сеть Интернет вышла далеко за изначальные рамки. Сети и управления, благодаря которым создавался Интернет, утратили свою решающую роль в жизни этой сети. Интернет эволюционировал из простой магистральной сети, через трехзвенную иерархическую структуру, в гигантскую сеть взаимосвязанных, распределенных сетевых концентраторов. Начиная с 1983 года, сеть растет экспоненциально, удваиваясь в размерах ежегодно. И даже эти невероятные изменения никак не повлияли на один простой факт: основой сети Интернет является семейство протоколов TCP/IP.

Одним из признаков успеха сети является путаница, связанная с термином *internet*. Исходное его значение – сеть, построенная на протоколе IP. Сегодня *internet* – это общий термин, обозначающий целый класс сетей. Сеть internet (со строчной буквы «i») – это произвольный набор физически обособленных сетей, связанных общим протоколом с целью формирования объединяющей логической сети. Internet (с прописной буквы «I») – глобальный набор взаимосвязанных сетей, который берет начало в сети ARPAnet и объединяет физические сети в одну логическую при помощи протокола IP.

В этой книге оба термина («internet» и «Internet») относятся к сетям, построенным на основе TCP/IP.

Поскольку наличие TCP/IP является обязательным условием подключения к Интернету, рост этой сети приводит к росту интереса к TCP/IP. Распространение TCP/IP в организациях привело к осознанию того факта, что потенциал протоколов может быть использован в любых сетевых приложениях. Протоколы Интернета часто используются для построения локальных сетей, пусть даже не подключенных к Интернету. Кроме того, TCP/IP широко применяется в построении корпоративных сетей. Корпоративные сети на основе TCP/IP, в которых для распространения корпоративной информации применяются методологии Интернета и веб-инструменты, носят название *интрасетей* (*intranets*). TCP/IP является основанием для всех систем подобного рода.

Особенности TCP/IP

Популярность протоколов TCP/IP росла быстро не просто потому, что протоколы существовали, а подключение к сети Интернет требовало их применения. В нужный момент они позволили удовлетворить важную потребность (глобальную передачу данных), а кроме того, обладали рядом особенностей, необходимых для решения задачи:

- Свободно распространяемые открытые стандарты протоколов, не зависящих от конкретного аппаратного обеспечения или операционной системы. Широкая поддержка делает TCP/IP идеальным выбором для объединения разнообразных программных и аппаратных составляющих даже в случаях, когда нет необходимости работать с сетью Интернет.
- Независимость от конкретных физических сетевых устройств, позволяющая интегрировать самые разные типы сетей посредством TCP/IP. TCP/IP может работать через Ethernet, соединение DSL, коммутируемое соединение, оптоволоконный канал, то есть в качестве физического транспорта может использоваться почти любая система.
- Универсальная схема адресации, позволяющая произвольному устройству TCP/IP обращаться к любому другому устройству сети по уникальному адресу, даже во всемирной сети Интернет.
- Стандартизированные высокоуровневые протоколы, согласующие работу распространенных пользовательских служб.

Стандарты протоколов

Протокол – это свод официальных правил поведения. В международных отношениях протоколы призваны воспрепятствовать возникновению проблем, вызванных культурными различиями стран. Оговаривая ряд общих правил, которые широко известны и не привязаны к обычаям конкретной нации, дипломатические протоколы минимизируют возможные недоразумения: всем известно, как следует себя вести и как интерпретировать пове-

дение других сторон. Точно так же, чтобы компьютеры смогли общаться, необходимо определить набор правил, которым подчиняется это общение.

В обмене данными подобные наборы правил также называют *протоколами*. В гомогенных сетях правила передачи данных определяются единственным поставщиком компьютеров и направлены на эффективное использование операционной системы и аппаратной архитектуры. Гомогенная сеть в нашей аналогии – это культура отдельной страны, которую могут назвать домом лишь ее граждане. TCP/IP создает гетерогенную сеть с открытыми протоколами, не зависящими от деталей архитектуры компьютера и реализации операционной системы. Протоколы TCP/IP доступны всем, они развиваются и изменяются по единодушному решению, а не по распоряжению конкретного производителя. Кто угодно может создавать продукцию, соответствующую спецификациям этих открытых протоколов.

Открытая природа протоколов TCP/IP требует открытости процесса разработки стандарта и свободного доступа к соответствующим документам. Разработка стандартов Интернета происходит на открытых сессиях комитета по технологической поддержке сети Интернет (Internet Engineering Task Force, IETF). Разработанные таким образом протоколы публикуются в виде документов RFC (*Request for Comments*, запрос комментариев).[1] Как и следует из названия, стилистика и содержание этих документов гораздо менее строги, чем в большинстве стандартов. Документы RFC содержат широкий спектр интересных и полезных сведений и не ограничиваются формальными спецификациями протоколов обмена данными. Существует три основных типа документов RFC: стандарты (standards, STD), современные практики (best current practices, BCP), а также уведомляющие (for your information, FYI).

Документы RFC, определяющие официальные стандарты протоколов, обозначаются аббревиатурой STD и получают STD-номера в дополнение к RFC-номерам. Создание официального стандарта Интернета – строго последовательный процесс. Стандарты RFC становятся таковыми, лишь пройдя через *три уровня зрелости*:

Заявка стандарта (Proposed Standard)

Спецификация протокола, который является достаточно важным и уже получил достаточно широкую поддержку интернет-сообщества, чтобы предлагаться в качестве стандарта. Такая спецификация прозрачна и закончена, но не является стандартом, и, более того, может никогда не достигнуть статуса стандарта.

Проект стандарта (Draft Standard)

Спецификация протокола, для которой существует по меньшей мере две независимых, взаимозаменяемых реализации. Проект стандарта – это окончательная спецификация, используемая в широком тестировании.

[1] Желаете узнать, как создаются стандарты Интернета? Прочтите документ RFC 2026, *The Internet Standards Process* (Процесс разработки стандартов Интернета).

Проект изменяется лишь в том случае, когда этого требуют результаты тестирования.

Стандарт Интернета (Internet Standard)

Спецификация получает статус стандарта лишь после всеобъемлющего тестирования и лишь в том случае, когда применение определенного этой спецификацией протокола может принести значительную выгоду интернет-сообществу.

Стандарты делятся на две категории. *Техническая спецификация* (*Technical Specification*, TS) дает определение протокола. *Формулировка применимости* (*Applicability Statement*, AS) описывает случаи, когда протокол следует применять. Применимость стандарта имеет три возможных уровня:

Обязательный (Required)

Стандартный протокол, является обязательной частью любой реализации TCP/IP. Стек протоколов должен включать этот протокол, чтобы соответствовать стандарту.

Рекомендованный (Recommended)

Стандартный протокол, рекомендованный к включению во все реализации TCP/IP. Его присутствие не является обязательным условием.

Факультативный (Elective)

Факультативный стандарт. Решение по реализации принимает разработчик конкретного пакета приложений.

Два других уровня применимости (*ограниченного применения* и *не рекомендован*) связаны с документами RFC, существующими обособленно от процесса стандартизации. Протокол «ограниченного применения» используется только в особых случаях, скажем, в ходе экспериментов. Протоколы, «не рекомендованные» к применению, являются устаревшими либо имеют ограниченную функциональность. Документы RFC, *не принадлежащие процессу стандартизации*, бывают трех типов:

Экспериментальные (Experimental)

Применение экспериментальных документов RFC ограничено исследованиями и разработкой.

Исторические (Historic)

Исторические RFC являются устаревшими, их применение не рекомендуется.

Уведомляющие (Informational)

Уведомляющие документы RFC содержат информацию, представляющую интерес для широких слоев сообщества сети Интернет, но не содержат определений протоколов.

Подмножество уведомляющих RFC носит имя уведомлений FYI (For Your Information, примите к сведению). Документу FYI, помимо номера RFC, присваивается еще и номер FYI. Эти документы содержат вводный и подготови-

тельный материал по сети Интернет и сетям TCP/IP в целом. Документы FYI не упомянуты в RFC 2026 и не являются составляющей процесса стандартизации. Тем не менее некоторые документы FYI представляют интерес.[1]

Вторая группа документов RFC, лежащих за пределами задачи документирования протоколов, носит имя BCP (Best Current Practices, лучшие современные практики). Задачей документов BCP является формальное документирование методов и процедур. Некоторые из них описывают принципы, которым подчиняются действия самой организации IETF; примером такого BCP-документа может послужить RFC 2026. Прочие содержат рекомендации по функционированию сетей или служб (для примера возьмем RFC 1918, *Address Allocation for Private Internets,* выделение адресов в закрытых интернет-сетях). Документы BCP последнего типа зачастую представляют особый интерес для сетевых администраторов.

В настоящее время существует более трех тысяч документов RFC. Так или иначе, администратору систем, объединенных в сеть, придется прочесть некоторые из них. Важно не только уметь понять эти документы в процессе чтения, но и знать, какие именно следует читать. Чтобы определить, как RFC соответствуют случаю, воспользуйтесь категориями и уровнями применимости. (Для начала неплохо сосредоточить внимание на тех документах RFC, которым присвоен номер STD.) Для понимания прочитанного необходимо понимание языка, на котором происходит обмен данными. Документы RFC определяют спецификации для реализаций протоколов в терминах языка, который является уникальным для обмена данными.

Модель обмена данными

В разговоре о компьютерных сетях приходится пользоваться терминами, имеющими особые значения. Даже компьютерные специалисты из других областей знакомы далеко не с каждым термином сетевой алфавитной каши. Привычное замечание – английский язык и компьютерный жаргон не являются эквивалентными (и даже совместимыми) языками. И хотя описания и примеры должны прояснить значение терминов сетевого жаргона, некоторые из слов время от времени оказываются двусмысленными. Таким образом, для понимания терминологии обмена данными необходима общая точка отсчета.

Для описания структуры и функциональности протоколов обмена данными часто используется архитектурная модель, разработанная Международной организацией по стандартизации (International Standards Organization, ISO). Данная модель как раз является общей точкой отсчета для разговора о коммуникациях и называется *опорной моделью взаимодействия открытых систем* (*Open Systems Interconnect* (OSI) *Reference Model*). Термины, определенные в рамках модели, нашли глубокое понимание и имеют широкое хож-

[1] Более подробно о документах FYI можно узнать из RFC 1150, *FYI on FYI: An Introduction to the FYI Notes* (Введение в уведомления FYI).

дение в сетевом сообществе – настолько широкое, что обмен данными сложно обсуждать, не применяя терминологию OSI.

Опорная модель OSI состоит из семи уровней, определяющих функциональность протоколов обмена данными. Каждый из уровней модели OSI представляет функцию, выполняемую при передаче данных между приложениями, взаимодействующими по сети. Названия и краткие функциональные описания уровней приведены на рис. 1.1, где протоколы похожи на столбик из кирпичей. По этой причине структура, о которой идет речь, часто именуется *стеком*, или *стеком протоколов*.

***Рис. 1.1.** Опорная модель OSI*

Уровень не содержит определения отдельного протокола; определяемая уровнем функциональность может быть реализована любым числом протоколов. Таким образом, каждый уровень способен вмещать произвольное число протоколов, каждый из которых реализует службу, соответствующую функциональности этого уровня. Так, протоколы передачи файлов и электронной почты реализуют пользовательские службы, и оба принадлежат к прикладному уровню.

Каждый протокол реализует взаимодействие с протоколами равного положения. *Протокол равного положения* – это реализация того же протокола на эквивалентном уровне удаленной системы; так, локальный протокол пе-

редачи файлов является протоколом равного положения с удаленным протоколом передачи файлов. Успешное взаимодействие протоколов одного уровня должно следовать установленным стандартам. Теоретически работа каждого протокола направлена лишь на взаимодействие с протоколами равного уровня, и ему нет дела до соседних уровней.

Но есть и другой момент, нуждающийся в согласовании: передача данных между уровнями в пределах отдельного компьютера. Дело в том, что в передаче данных от локального приложения эквивалентному удаленному приложению участвуют все уровни. Вышележащие уровни в вопросах передачи данных по сети полагаются на нижележащие. Данные передаются вниз по стеку, с одного уровня на другой, и, в конце концов, передаются по сети протоколами физического уровня. На стороне «собеседника» данные передаются вверх по стеку приложению-адресату. Отдельный уровень может не знать, каким образом работают соседи; необходимым знанием является лишь способ передачи и получения данных. Изоляция функций сетевого взаимодействия на различных уровнях сводит к минимуму воздействие технологических изменений на все семейство протоколов. Создание новых приложений не требует внесения изменений в физическую базу сети, а установка нового сетевого оборудования не требует модификации пользовательских программ.

Несмотря на эффективность модели OSI протоколы TCP/IP не до конца следуют установленной структуре модели. Поэтому в разговоре о TCP/IP мы будем интерпретировать уровни модели OSI следующим образом:

Прикладной уровень (Application Layer)

Прикладной уровень иерархии протоколов содержит сетевые процессы, с которыми работают пользователи. В тексте книги приложение TCP/IP – это любой сетевой процесс, протекающий выше транспортного уровня. Под это определение подпадают все процессы, с которыми напрямую взаимодействуют пользователи, а также прочие процессы данного уровня, о которых пользователи могут и не знать.

Уровень представления (Presentation Layer)

Чтобы взаимодействующие приложения смогли обмениваться данными, необходимо соглашение о представлении данных. В OSI стандартная функциональность представления данных обозначена уровнем представления. Функция представления часто реализуется в приложениях TCP/IP, а также такими протоколами TCP/IP, как XDR и MIME.

Сеансовый уровень (Session Layer)

Аналогично уровню представления, сеансовый уровень не является самостоятельным в иерархии протоколов TCP/IP. Сеансовый уровень OSI отвечает за управление сеансами (соединениями) взаимодействия приложений. В TCP/IP эта функциональность заложена преимущественно на транспортном уровне, а термин «сеанс» не имеет хождения; для описания вектора взаимодействия приложений применяются термины «сокет» (socket, гнездо) и «порт».

Транспортный уровень (Transport Layer)

Большая часть информации в рассказе о TCP/IP связана с протоколами транспортного уровня. В опорной модели OSI транспортный уровень гарантирует получение адресатом данных в неизмененном виде. В TCP/IP такая функциональность возложена на *протокол управления передачей* (*Transmission Control Protocol*, TCP). При этом в TCP/IP существует вторая служба транспортного уровня – *протокол пользовательских дейтаграмм* (*User Datagram Protocol*, UDP), который не гарантирует надежной доставки данных.

Сетевой уровень (Network Layer)

Сетевой уровень управляет сетевыми соединениями и проводит границу между протоколами более высокого уровня и подробностями реализации собственно сети. Сетевой уровень TCP/IP обычно подразумевает протокол Internet (IP), который разграничивает вышележащие уровни и сеть, а также отвечает за адресацию и доставку данных.

Канальный уровень (Data Link Layer)

Надежная доставка данных по физической сети находится в ведении канального уровня. Этот уровень TCP/IP, как правило, не содержит протоколов. В большинстве документов RFC, упоминающих канальный уровень, рассматриваются вопросы интеграции IP и существующих канальных протоколов.

Физический уровень (Physical Layer)

Физический уровень определяет характеристики аппаратного обеспечения, необходимого для осуществления передачи данных, в частности такие свойства, как уровни напряжения, количество и расположение контактов интерфейсов. В качестве примеров стандартов физического уровня можно упомянуть стандарты на интерфейсные разъемы RS232C V.35, а также монтажные стандарты для локальных сетей, такие как IEEE 802.3. TCP/IP не определяет физические стандарты, но пользуется существующими.

Терминология опорной модели OSI способствует более прозрачному описанию TCP/IP, но для полного понимания системы следует воспользоваться архитектурной моделью, точнее отражающей структуру TCP/IP. Модель, которую мы используем для описания TCP/IP, представлена в следующем разделе.

Архитектура протоколов TCP/IP

Несмотря на отсутствие универсальных правил описания TCP/IP посредством многоуровневой модели, модели TCP/IP обычно содержат менее семи уровней. Большинство описаний TCP/IP определяют от трех до пяти функциональных уровней архитектуры протокола. Четырехуровневая модель, приведенная на рис. 1.2, состоит из трех уровней (прикладной, узел-узел, доступ к сети) модели DOD Protocol Model из первого тома руководства по протоколам DDN и дополнительного уровня Internet. Такая модель обеспечивает приемлемую иллюстрацию уровней иерархии протоколов TCP/IP.

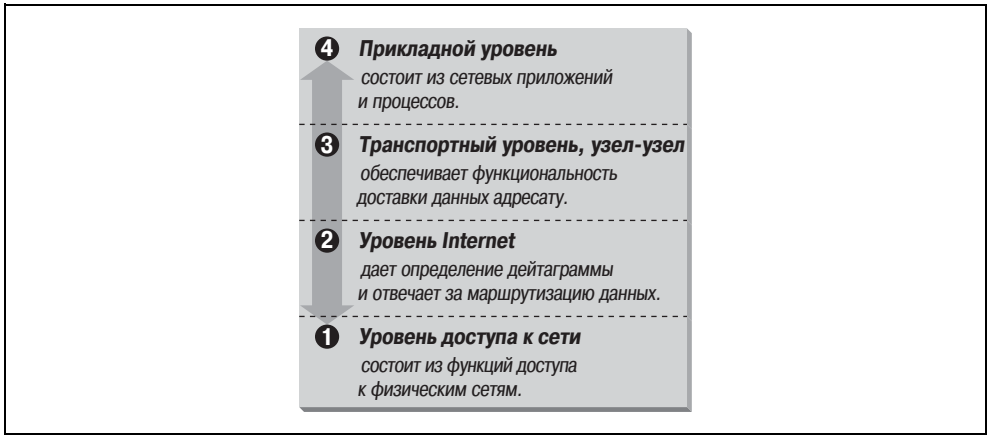


Рис. 1.2. *Архитектура TCP/IP*

Как и в модели OSI, данные передаются вниз по стеку при отправке в сеть и вверх по стеку – при получении из сети. Четырехуровневая структура TCP/IP проявляется в способе обработки данных при их прохождении вниз по стеку, от прикладного уровня непосредственно к физической сети. Каждый уровень стека добавляет управляющую информацию, гарантируя корректную доставку. Блок управляющей информации называется *заголовком* (*header*), поскольку предшествует передаваемым данным. Каждый уровень интерпретирует всю информацию, полученную от вышележащего уровня, в качестве данных и добавляет к этим данным собственный заголовок. Дополнение информации по доставке на каждом уровне носит название *инкапсуляции* (рис. 1.3).

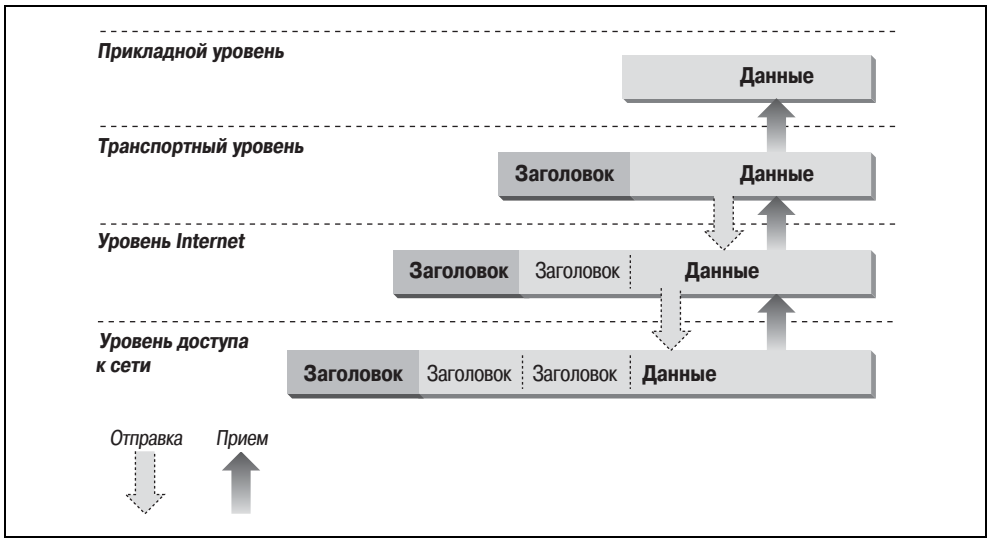


Рис. 1.3. *Инкапсуляция данных*

При получении данных происходит обратный процесс. Каждый уровень удаляет соответствующий заголовок и передает данные вышележащему уровню. При передаче вверх по стеку информация, получаемая от нижележащих уровней, интерпретируется в качестве заголовка и сопутствующих данных.

Каждому уровню соответствуют определенные структуры данных. Теоретически уровень не обязан знать о структурах данных, применяемых на соседних уровнях, однако на практике структуры данных уровня проектируются таким образом, чтобы хорошо сочетаться со структурами «соседей» в целях повышения эффективности передачи данных. Тем не менее каждому уровню соответствует собственная структура данных и специальная терминология ее описания.

На рис. 1.4 отражены термины, применяемые на различных уровнях TCP/IP в отношении передаваемых данных. Приложения TCP считают данные *потоком* (*stream*), а приложения UDP – *сообщением* (*message*). На транспортном уровне TCP данные хранятся в *сегментах* (*segment*), на транспортном уровне UDP – в *пакетах* (*packet*). Уровень Internet рассматривает данные в качестве блоков, называемых *дейтаграммами* (*datagrams*). Многочисленные типы сетей, поверх которых работает TCP/IP, также используют разнообразную терминологию в области передаваемых данных. В большинстве сетей приняты термины *пакет* (*packet*) или *фрейм* (*frame*, блок данных). На рис. 1.4 показана сеть, передающая фрагменты данных, называемые *фреймами*.

Рассмотрим более подробно функциональность каждого уровня, поднимаясь от уровня доступа к сети в направлении прикладного уровня.

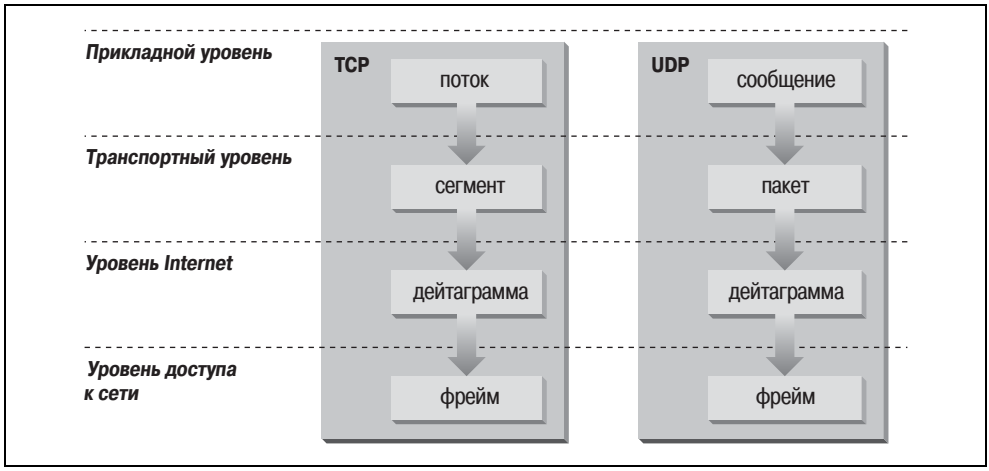


Рис. 1.4. *Структуры данных*

Уровень доступа к сети

Уровень доступа к сети является низшим уровнем иерархии протоколов TCP/IP. Протоколы этого уровня предоставляют системе средства доставки данных другим устройствам сети с прямым подключением. Этот уровень определяет способы использования сети для передачи IP-дейтаграмм. В отличие от протоколов более высоких уровней, протоколы доступа к сети должны обладать подробными сведениями о работе сети (структуре пакетов, системе адресации и т. д.), чтобы корректно форматировать передаваемые данные, следуя ограничениям, налагаемым сетью. Уровень доступа к сети TCP/IP может включать всю функциональность трех нижних уровней опорной модели OSI (сетевого, канального и физического).

Уровень доступа к сети часто остается незамеченным пользователями. Архитектура TCP/IP скрывает функциональность нижележащих уровней, а все более широко известные протоколы (IP, TCP, UDP и прочие) являются высокоуровневыми. Появление новых аппаратных возможностей сетевых устройств диктует необходимость разработки новых протоколов уровня доступа к сети. Такая разработка позволяет существующим TCP/IP-сетям использовать новое оборудование. Следствием такого положения является существование многочисленных протоколов доступа – по одному на каждый стандарт физической сети.

Функциональность данного уровня включает инкапсуляцию IP-дейтаграмм в передаваемые по сети фреймы, а также отображение IP-адресов в физические адреса, применяемые в сети. Одним из достоинств TCP/IP является универсальная схема адресации. Адрес IP должен подвергнуться преобразованию в адрес, уместный в физической сети, служащей для передачи дейтаграммы.

Протоколы уровня доступа к сети определяются следующими документами RFC:

- RFC 826, *Address Resolution Protocol* (ARP, протокол разрешения адресов); протокол выполняет отображение IP-адресов в адреса Ethernet.
- RFC 894, *A Standard for the Transmission of IP Datagrams over Ethernet Networks* (стандарт передачи IP-дейтаграмм в сетях Ethernet); документ определяет, каким образом производится инкапсуляция IP-дейтаграмм с целью передачи в сетях Ethernet.

Реализация протоколов этого уровня в системах Unix обычно представлена сочетанием драйверов устройств и сопутствующих программ. Модули, обозначаемые именами сетевых устройств, отвечают за инкапсуляцию и доставку данных по сети, а сопутствующие функции, вроде отображения адресов, возлагаются на самостоятельные программы.

Уровень Internet

Следующим в иерархии является *уровень Internet*. Наиболее важный протокол этого уровня – протокол Internet (IP). В существующей сети Интернет

применяется протокол IP версии 4 (IPv4), определенный в RFC 791. Существуют и более современные версии IP. IP версии 5 – это экспериментальный протокол потокового транспорта (Stream Transport, ST), применяемый для доставки данных в системах реального времени. IPv5 не получил рабочего распространения. IPv6 – стандарт IP, предоставляющий значительным образом расширенные диапазоны адресации. В IPv6 применяется совершенно иная структура адреса, так что протоколы IPv6 и IPv4 не способны к взаимодействию. Несмотря на существование стандарта IPv6 эта версия IP пока не получила широкого распространения в действующих коммерческих сетях. Нашей целью является изучение работы действующих сетей, поэтому мы не станем вдаваться в подробности IPv6. В тексте главы и большей части книги аббревиатура «IP» относится к IPv4. Именно протокол IPv4 является предметом настройки при необходимости организовать обмен данными между системами, и именно этому протоколу будет уделено основное внимание.

Протокол Internet – это сердце TCP/IP. Он обеспечивает работу базовой службы доставки пакетов, на которой построены сети TCP/IP. Все протоколы этого и соседствующих уровней используют протокол Internet для доставки данных. Все входящие и исходящие потоки данных TCP/IP проходят через IP независимо от пункта назначения.

Протокол Internet

Протокол Internet (IP) – это строительный элемент сетей Интернет. Он имеет следующую функциональность:

- Определяет дейтаграмму, базовую единицу передачи в сетях Интернет
- Определяет схему интернет-адресации
- Осуществляет обмен данными между уровнем доступа к сети и транспортным уровнем
- Выполняет маршрутизацию дейтаграмм, адресованных удаленным узлам
- Отвечает за разбиение и сборку дейтаграмм

Прежде чем перейти к более подробному рассмотрению этих функций, взглянем на некоторые свойства IP. Во-первых, протокол IP работает *без создания логических соединений*. Это означает, что передача данных не требует обмена управляющей информацией (подтверждения связи, известного в качестве «рукопожатия») с целью установки сквозного соединения. Напротив, протоколы, работающие *на основе соединений*, обмениваются управляющей информацией с удаленными системами, проверяя их готовность принимать данные. Успешное подтверждение связи является признаком того, что соединение установлено. Протокол Internet делегирует установку соединений протоколам других уровней для случаев, когда требуется наличие соединений.

Кроме того, в вопросах обнаружения ошибок и восстановления после ошибок протокол IP полагается на протоколы других уровней. Протокол Internet иногда называют *ненадежным протоколом*, поскольку он не реализует обнаружение и восстановление после ошибок. Однако это не означает, что на

протокол IP нельзя положиться – как раз наоборот. Можно положиться на IP в плане точной доставки данных в доступную сеть, но невозможно с его помощью проверить, что данные были корректно получены. Подобные проверки при необходимости реализуются посредством протоколов других уровней архитектуры TCP/IP.

Дейтаграмма

Протоколы TCP/IP создавались для организации передачи данных в ARPAnet, сети на базе *коммутации пакетов*. *Пакет* – это блок данных, содержащий информацию, необходимую для доставки. В этом отношении пакет похож на письмо, адрес получателя отражен на конверте. Сеть с пакетной коммутацией использует адресную информацию пакетов для коммутации пакетов из одной физической сети в другую, перемещая их ближе к пункту назначения. Пакеты путешествуют по сети независимо друг от друга.

Формат пакета, определяемый протоколом Internet, называется *дейтаграммой*. Содержимое IP-дейтаграммы наглядно показано на рис. 1.5. Первые пять или шесть 32-битных слов дейтаграммы содержат управляющую информацию, составляющую *заголовок* (*header*). По умолчанию заголовок имеет длину в пять слов, шестое является необязательным. Для указания переменной длины заголовка (в словах) используется поле *Internet Header Length* (IHL, *длина заголовка Internet*). Заголовок содержит всю необходимую для доставки пакета информацию.

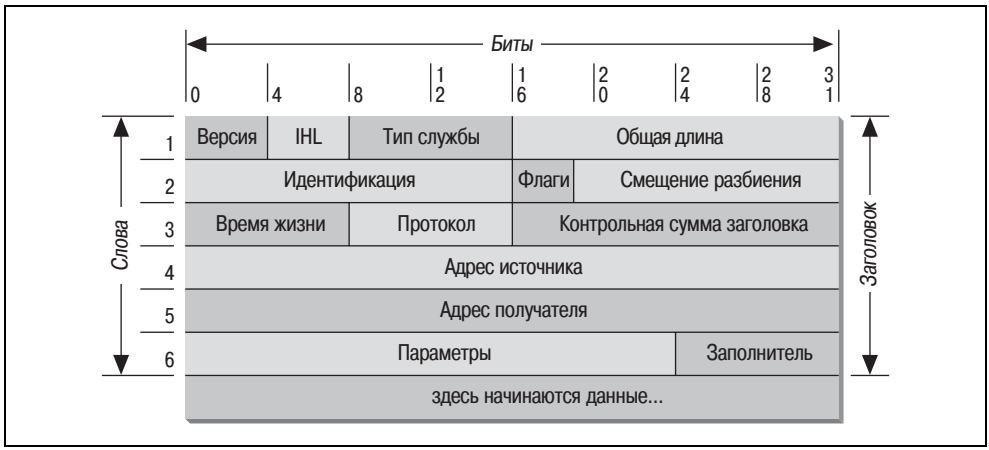


Рис. 1.5. *Формат дейтаграмм IP*

Протокол Internet выполняет доставку дейтаграммы на основе *адреса получателя* из пятого слова заголовка. Адрес получателя – это стандартный 32-битный адрес IP, соответствующий определенной сети и конкретному узлу этой сети. (Формат IP-адресов описан в главе 2.) Если адресом получателя является адрес узла локальной сети, пакет доставляется напрямую в пункт назначения. В противном случае пакет передается на шлюз (gateway) для до-

ставки. *Шлюзы* занимаются коммутацией пакетов между физически обособленными сетями. Принятие решения о том, какой именно шлюз использовать, называется *маршрутизацией*. Протокол IP принимает такое решение для каждого пакета.

Маршрутизация дейтаграмм

Шлюзы Internet применяют протокол Internet для маршрутизации пакетов по сетям, а потому их можно с большой степенью точности называть *IP-маршрутизаторами*. Традиционный жаргон TCP/IP включает лишь два типа сетевых устройств – *шлюзы* и *узлы*. Шлюзы, в отличие от узлов, передают пакеты между сетями. Однако если узел входит в несколько сетей (*многосетевой* узел), он получает возможность передавать пакеты из одной сети в другую. При пересылке пакетов многосетевой узел действует точно так же, как любой шлюз, и, вообще говоря, получает статус шлюза. Существующая сегодня терминология передачи данных различает шлюзы и маршрутизаторы[1], но мы будем считать термины *шлюз* и *IP-маршрутизатор* взаимозаменяемыми.

На рис. 1.6 отражена пересылка пакетов посредством шлюзов. Узлы (*оконечные системы*) производят обработку пакетов на всех четырех уровнях протоколов, в то время как шлюзы (*промежуточные системы*) обрабатывают пакеты лишь до уровня Internet, на котором происходит принятие решений по маршрутизации.

Система способна доставить пакет только на другое устройство, принадлежащее той же физической сети. Пакеты, адресованные узлом *A1* узлу *C1*, пересылаются через шлюзы *G1* и *G2*. Узел *A1* доставляет пакет шлюзу *G1*, с кото-

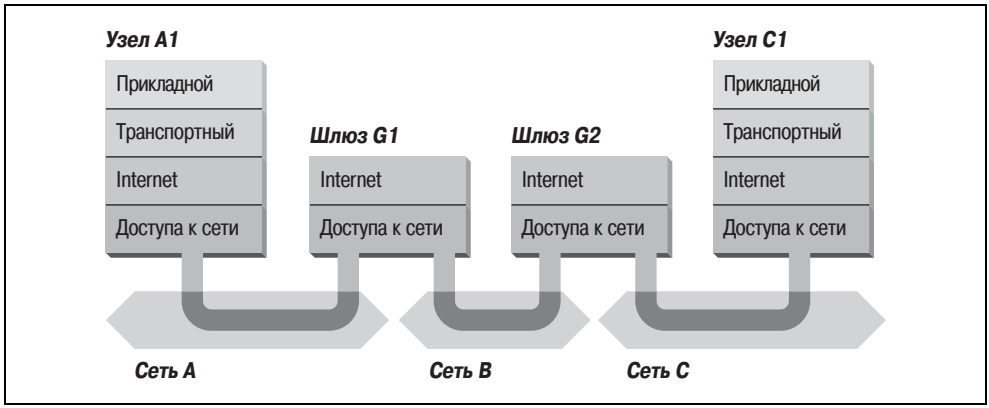


Рис. 1.6. *Маршрутизация посредством шлюзов*

[1] Согласно существующей терминологии, шлюз переносит данные, связывая различные протоколы, тогда как маршрутизатор переносит данные между различными сетями. Таким образом, система, передающая почту между сетями TCP/IP и X.400, является шлюзом, но традиционный IP-шлюз является маршрутизатором.

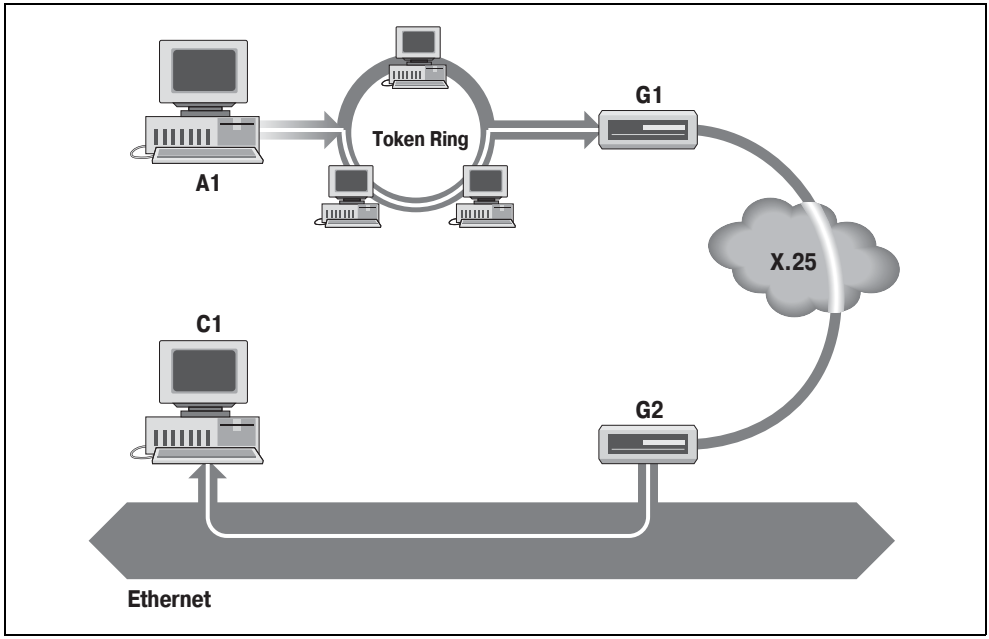


Рис. 1.7. *Сети, шлюзы и узлы*

рым вместе входит в сеть *A*. Шлюз *G1* доставляет пакет шлюзу *G2* по сети *B*. Шлюз *G2*, в свою очередь, доставляет пакет напрямую узлу *C1*, с которым вместе входит в сеть *C*. Узел *A1* понятия не имеет о существовании иных шлюзов, кроме *G1*. Он посылает пакеты, адресованные в сети *C* и *B*, этому локальному шлюзу и делегирует ему задачу корректной пересылки пакетов по нужным маршрутам. Точно так же узел *C1* отправляет свои пакеты шлюзу *G2*, чтобы обратиться к узлу сети *A* либо к узлу сети *B*.

Другой взгляд на маршрутизацию приведен на рис. 1.7. Данная иллюстрация акцентируется на том факте, что физические сети, через которые путешествует дейтаграмма, могут различаться и даже быть несовместимыми. Узел *A1* кольцевой сети передает дейтаграмму через шлюз *G1*, направляя ее узлу *C1* Ethernet-сети. Шлюз *G1* пересылает данные по сети X.25 шлюзу *G2*, который и производит доставку на *C1*. Дейтаграмма проходит через три физически разделенных сети, но рано или поздно в целости и сохранности достигает узла *C1*.

Разбиение дейтаграмм

В процессе маршрутизации дейтаграммы через различные сети на одном из шлюзов может возникнуть необходимость в разбиении дейтаграммы на более мелкие фрагменты. Дейтаграмма, полученная из одной сети, может оказаться слишком крупной, чтобы ее вместил один пакет второй сети. Такая ситуация может возникать лишь в случаях, когда шлюз связывает физически различные сети.

Для каждой сети определяется значение *MTU* (maximum transmission unit, максимальная единица передачи), которое обозначает *максимально допустимый размер пакета* в этой сети. Если полученная из первой сети дейтаграмма длиннее значения MTU второй сети, она разбивается на ряд фрагментов с целью передачи. Данный процесс называется *разбиением дейтаграммы*. Представьте поезд, везущий стальные болванки. Каждый из вагонов поезда вмещает больше стали, чем грузовики, которые повезут груз дальше, по шоссе, поэтому каждый вагон поезда разгружается на множество грузовых машин. Сеть Ethernet точно так же физически отличается от сети X.25, как железная дорога от шоссе; протокол IP должен разделить относительно крупные пакеты Ethernet на более мелкие, прежде чем их можно будет передать по сети X.25.

Формат каждого фрагмента – такой же, как для обычной дейтаграммы. Второе слово заголовка обозначает фрагмент дейтаграммы и содержит информацию о том, как производить сборку фрагментов в целях восстановления исходной дейтаграммы. Поле *Идентификация* содержит информацию о том, к какой дейтаграмме принадлежит фрагмент, а поле *Смещение разбиения* – о том, каким по счету элементом является фрагмент дейтаграммы. В поле *Флаги* присутствует бит «Другие фрагменты» (More Fragments), позволяющий протоколу IP определить, что собраны все фрагменты исходной дейтаграммы.

Передача дейтаграмм в транспортный уровень

При получении дейтаграммы, адресованной локальному узлу, протокол IP обязан передать информативную часть дейтаграммы подходящему протоколу транспортного уровня. Задача решается при помощи *номера протокола*, указанного в третьем слове заголовка дейтаграммы. Каждому протоколу транспортного уровня присвоен уникальный номер протокола. С этими номерами и работает протокол IP. Речь о номерах протоколов пойдет в главе 2.

Как можно видеть из этого краткого обзора, на протокол IP возложены многие важные функции. Разумеется, приведенного описания недостаточно для полного понимания дейтаграмм, шлюзов, маршрутизации, IP-адресации и всех прочих элементов протокола IP: в каждой из последующих глав будет уделено больше внимания этим темам. А сейчас мы переходим к другим протоколам и уровню TCP/IP Internet.

ICMP, протокол управляющих сообщений Internet

Неотъемлемой частью IP является *протокол управляющих сообщений* (*Internet Control Message Protocol,* ICMP), определенный документом RFC 792. Данный протокол принадлежит уровню Internet и использует функциональность доставки дейтаграмм для отправки собственных сообщений. Сообщения ICMP выполняют следующие информативные, управляющие и связанные с ошибками функции TCP/IP:

Управление потоками данных (Flow control)

Если скорость поступления дейтаграмм слишком высока для обработки, узел-адресат или промежуточный шлюз отправляет ICMP-сообщение по-

давления источника (Source Quench Message) передающему узлу. Сообщение предписывает источнику временно прервать посылку дейтаграмм.

Обнаружение недостижимых адресатов

В случае когда сообщение не может быть доставлено адресату, система, обнаружившая эту проблему, отправляет сообщение Destination Unreachable источнику дейтаграммы. Если недостижимый адресат является сетью или узлом, сообщение исходит от промежуточного шлюза. Если пунктом назначения является порт, сообщение исходит от конкретного узла. (О портах мы поговорим в главе 2.)

Перенаправление маршрутов

ICMP-сообщение перенаправления (Redirect Message), исходящее от шлюза, предписывает узлу воспользоваться другим шлюзом, предположительно, по той причине, что это будет более эффективный выбор. Это сообщение может применяться лишь в случаях, когда узел-источник находится в одной сети с обоими шлюзами. Взгляните на рис. 1.7. Если узел сети X.25 направляет дейтаграмму шлюзу *G1*, этот шлюз может перенаправить узел к шлюзу *G2*, поскольку узел-источник и шлюзы *G1* и *G2* входят в одну сеть. С другой стороны, если источником дейтаграммы является узел сети Token Ring, *G1* уже не сможет перенаправить его к шлюзу *G2*. Как раз по той причине, что *G2* не входит в кольцевую сеть Token Ring.

Проверка состояния удаленных узлов

Узел может отправить ICMP-сообщение эхо (Echo Message), чтобы проверить работоспособность протокола Internet удаленной системы. Получив сообщение эхо, система возвращает данные из полученного пакета узлу-источнику. На основе сообщений Echo Message построена работа команды `ping`.

Транспортный уровень

Непосредственно над уровнем Internet расположен *транспортный уровень узел-узел*, или просто *транспортный уровень*. Наиболее важными протоколами транспортного уровня являются *протокол управления передачей* TCP (*Transmission Control Protocol*) и *протокол пользовательских дейтаграмм* UDP (*User Datagram Protocol*). TCP обеспечивает надежную доставку данных со сквозным обнаружением и устранением ошибок. UDP – это нетребовательная к ресурсам служба доставки дейтаграмм, работающая без образования логических соединений. Оба протокола выполняют доставку данных между прикладным уровнем и уровнем Internet. Разработчики приложений вольны выбирать, какая из этих служб точнее отвечает задачам приложения в каждом конкретном случае.

UDP, протокол пользовательских дейтаграмм

Протокол пользовательских дейтаграмм (User Datagram Protocol, UDP) дает прикладным программам прямой доступ к службе доставки дейтаграмм,

работающей подобно службе доставки IP. Приложения получают возможность обмениваться протоколами по сети с минимальными «накладными расходами».

UDP – это ненадежный протокол доставки дейтаграмм без организации логических соединений. Повторимся, характеристика «ненадежный» говорит лишь о том, что средствами протокола невозможно убедиться, что данные корректно получены адресатом. В пределах одного компьютера UDP выполняет доставку данных безукоризненно. Для доставки данных соответствующим процессам приложений в UDP применяются 16-битные номера для *исходного* и *целевого* портов в первом слове заголовка сообщения. Формат сообщения UDP приведен на рис. 1.8.

***Рис. 1.8.** Формат сообщения UDP*

Зачем разработчикам приложений использовать UDP в качестве службы доставки данных? Есть ряд веских причин. Если объем передаваемых данных невелик, издержки на создание соединений и отслеживание надежности доставки могут оказаться больше, чем затраты на повторную передачу всех данных. В таком случае UDP становится оптимальным протоколом транспортного уровня. Другими явными кандидатами на использование UDP являются приложения, работающие по принципу *запрос-ответ*. Ответ можно считать подтверждением приема запроса. Если ответ не поступил в течение определенного промежутка времени, приложение просто повторяет запрос. Кроме того, существуют приложения, в которых реализуются собственные методы обеспечения надежной доставки данных, так что они не ищут подобной функциональности в протоколах транспортного уровня. Для каждого из упомянутых типов приложений дополнительный уровень подтверждения приема станет причиной снижения производительности.

TCP, протокол управления передачей

Для обеспечения надежной доставки данных на уровне транспортного протокола в приложениях используется протокол TCP, проверяющий факт доставки данных по сети в нужном порядке. TCP – *надежный*, *потоковый протокол*, *требующий создания логических соединений*. Рассмотрим более подробно каждую из этих характеристик.

Надежность в TCP обеспечивает механизм *подтверждения приема с повторной передачей* (*Positive Acknowledgment with Retransmission*, PAR). Система,

в которой применяется PAR, повторяет отправку данных до тех пор, пока не получит от системы-адресата подтверждение, что данные успешно получены. Единицей обмена данными для взаимодействующих модулей TCP является *сегмент* (рис. 1.9). Каждый сегмент содержит контрольную сумму, посредством которой получатель определяет целостность данных. Если сегмент данных получен в целости и сохранности, получатель отправляет источнику *подтверждение*. Поврежденные сегменты данных просто игнорируются получателем. По истечении установленного интервала ожидания модуль-источник TCP повторно выполняет передачу всех сегментов, для которых не были получены подтверждения.

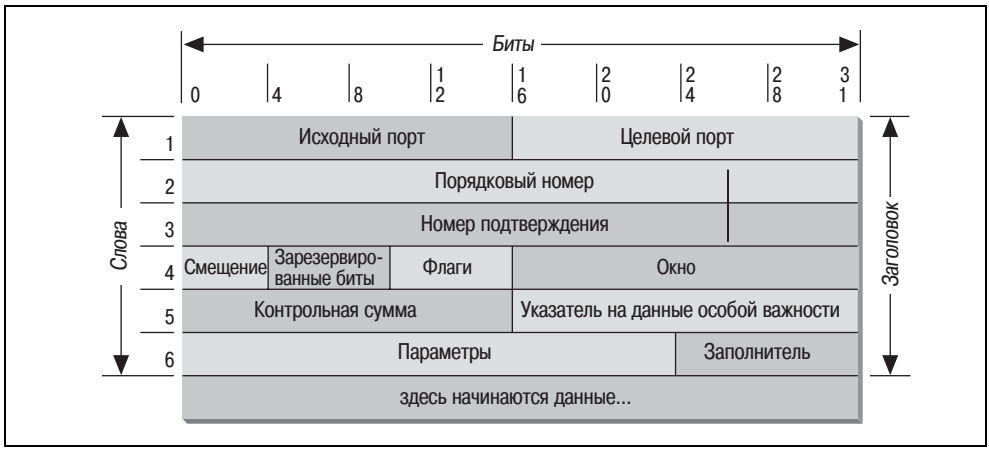


***Рис. 1.9.** Формат сегмента TCP*

TCP ориентирован на работу с соединениями. В целях обмена данными между двумя узлами образуется сквозное логическое соединение. Перед началом передачи данных (беседы) узлы обмениваются управляющей информацией (*рукопожатием*). Управляющий статус сегмента TCP отражается посредством соответствующего флага поля *Флаги* в четвертом слове заголовка сегмента.

В TCP применяется установление соединения с помощью *тройного рукопожатия*; производится обмен тремя сегментами. Простейший вариант тройного рукопожатия показан на рис. 1.10. Узел *A* открывает соединение, посылая узлу *B* сегмент с установленным битом «синхронизации порядковых номеров» (Synchronize sequence numbers, SYN). Сегмент сообщает узлу *B*, что *A* желает создать соединение и уведомляет о том, какой порядковый номер будет использоваться в качестве начального в сегментах *A*. (Порядковые номера применяются для сохранения порядка следования данных.) Узел *B* отвечает узлу *A* сегментом с установленными битами «подтверждения» (Acknowledgment, ACK) и синхронизации (SYN). Сегмент *B* подтверждает получение сегмента от *A*, а также уведомляет, какой порядковый номер станет начальным для сегментов *A*. Наконец, узел *A* передает сегмент, подтверждающий получение сегмента от *B*, а также первый блок непосредственно данных.

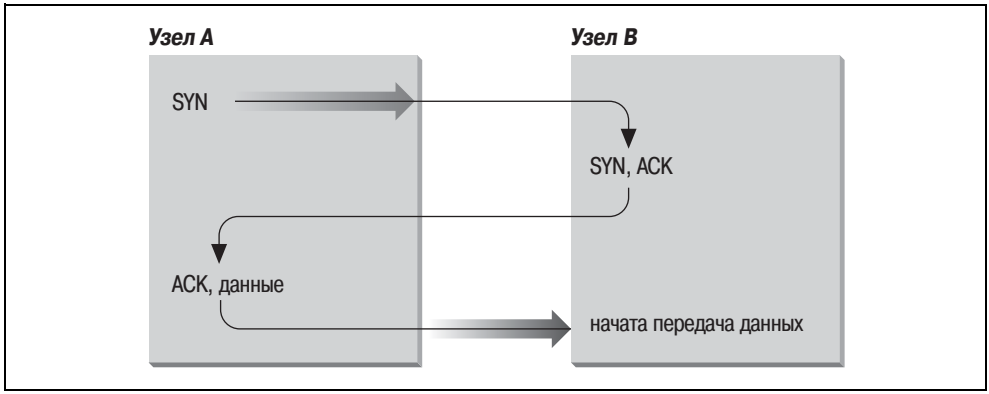


***Рис. 1.10**. Тройное рукопожатие*

После такого обмена TCP-модуль узла *A* обладает всеми свидетельствами того, что удаленный TCP-модуль функционирует и готов принимать данные. Как только соединение установлено, передача данных получает зеленый свет. После завершения передачи данных взаимодействующие модули обмениваются тройным рукопожатием, содержащим сегменты с так называемым битом FIN (No more data from sender, у источника больше нет данных), в целях закрытия соединения. Именно сквозной обмен данными становится логическим соединением между двумя системами.

В TCP данные считаются непрерывным потоком байтов, а не набором независимых пакетов. Следовательно, TCP предпринимает меры для сохранения последовательности отправки и получения байтов. Этой цели служат поля заголовка сегмента TCP – *Порядковый номер* и *Номер подтверждения*.

Стандарт TCP не определяет конкретных чисел, с которых должна начинаться нумерация; каждая система самостоятельно выбирает точку начала отсчета. Чтобы корректно отслеживать порядок в потоке данных, каждая из взаимодействующих сторон должна знать исходный номер второй стороны. Две стороны соединения синхронизируют системы нумерации байтов, обмениваясь SYN-сегментами в ходе рукопожатия. Поле *Порядковый номер* SYN-сегмента содержит *исходный порядковый номер* (*Initial Sequence Number*, ISN), который является точкой отсчета для *системы нумерации байтов*. Из соображений безопасности ISN следует быть случайным числом.

Каждый байт данных нумеруется последовательно, начиная с номера ISN, так что первый байт непосредственно данных имеет порядковый номер ISN+1. Порядковый номер в заголовке сегмента с данными указывает на порядковое положение в потоке данных первого байта данных сегмента. Например, если первый байт потока данных имел порядковый номер 1 (при ISN = 0), а 4000 байт данных уже получены адресатом, первый байт данных текущего сегмента является байтом 4001, и будет иметь порядковый номер 4001.

Подтверждающий сегмент (Acknowledgment Segment, ACK) выполняет две функции: *подтверждения приема* и *управления потоком*. Подтверждение

сообщает источнику, какой объем данных получен и сколько еще данных адресат способен принять. Номер подтверждения – это порядковый номер следующего байта, ожидаемый адресатом. Стандарт не требует создания подтверждения для каждого пакета. Номер подтверждения является подтверждением получения всех байтов, вплоть до этого номера. Например, если первый отправленный байт имел номер 1 и 2000 байт данных уже получены адресатом, номер подтверждения будет иметь значение 2001.

Поле *Окно* содержит значение *окна*, то есть количество байт, которое способен принять адресат. Если адресат способен принять еще 6000 байт, окно имеет значение 6000. Окно является указанием источнику, что можно продолжать передачу сегментов, если общий объем передаваемых байт меньше байтового окна адресата. Адресат управляет потоком байтов источника, изменяя размер окна. Нулевое окно предписывает отправителю прекратить передачу, пока не будет получено ненулевое значение окна.

На рис. 1.11 приведен поток данных TCP с нулевым значением исходного порядкового номера. Адресат получил и подтвердил получение 2000 байт, поэтому текущий номер подтверждения – 2001. Кроме того, адресат обладает возможностью принять еще 6000 байт, а потому предъявляет окно со значением 6000. Источник отправляет сегмент размером в 1000 байт с порядковым номером 4001. Для байтов 2001 и последующих еще не были получены подтверждения, однако источник продолжает передачу данных, пока не исчерпаны ресурсы окна. Если на момент заполнения окна источником для уже отправленных данных не получены подтверждения, по истечении определенного интервала ожидания источник повторно передает данные, начиная с первого неподтвержденного байта.

В отсутствие последующих подтверждений повторная передача начнется с байта 2001. Данный метод гарантирует надежность доставки данных адресату.

Кроме того, TCP отвечает за доставку полученных от IP данных соответствующему приложению. Приложение, которому предназначаются данные, обозначается 16-битным числом, *номером порта*. Значения *Исходный порт*

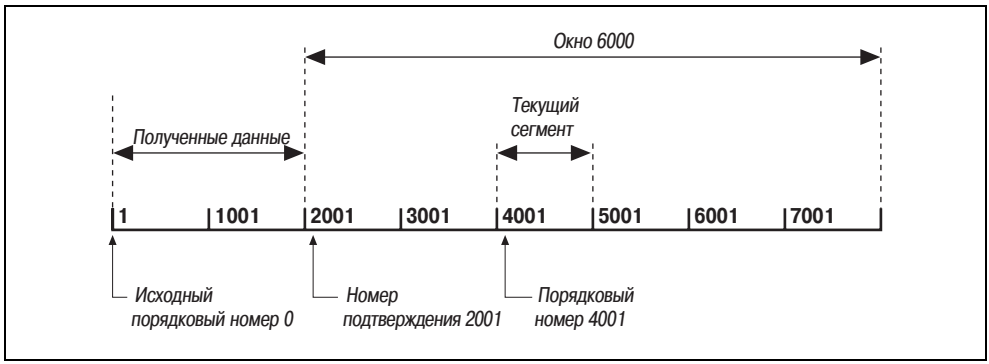


Рис. 1.11. *Поток данных TCP*

и *Целевой порт* содержатся в первом слове заголовка сегмента. Корректный обмен данными с прикладным уровнем – это важная составляющая функциональности служб транспортного уровня.

Прикладной уровень

Вершиной архитектуры протоколов TCP/IP является *прикладной уровень*, в который входят все процессы, использующие протоколы транспортного уровня в целях доставки данных. Существует большое число прикладных протоколов. Большинство из них связаны с пользовательскими службами, а число служб этого уровня постоянно растет.

Перечислим наиболее широко известные и распространенные прикладные протоколы:

Telnet

Протокол сетевых терминалов (Network Terminal Protocol) обеспечивает диалоговую работу с удаленными системами по сети.

FTP

Протокол передачи файлов (File Transfer Protocol) используется для передачи файлов в диалоговом режиме.

SMTP

Простой протокол передачи почты (Simple Mail Transfer Protocol) обеспечивает доставку сообщений электронной почты.

HTTP

Протокол передачи гипертекста (Hypertext Transfer Protocol) выполняет доставку веб-страниц по сети.

HTTP, FTP, SMTP и Telnet – наиболее широко распространенные приложения TCP/IP. При этом как пользователи, так и системные администраторы в своей работе встречаются и со многими другими. Вот некоторые из не менее широко применяемых приложений TCP/IP:

Domain Name System (DNS, Система доменных имен)

Известное также в качестве *службы имен*, это приложение ассоциирует IP-адреса с именами, назначаемыми сетевым устройствам. Система DNS подробно рассматривается в этой книге.

Open Shortest Path First (OSPF, Протокол предпочтения кратчайшего пути)

Маршрутизация является одним из несущих элементов конструкции TCP/IP. Протокол OSPF используется сетевыми устройствами для обмена информацией по маршрутизации. Маршрутизация также является одной из основных тем этой книги.

Network File System (NFS, Сетевая файловая система)

Данный протокол позволяет организовывать совместный доступ к файлам многих узлов сети.

Отдельные протоколы, такие как Telnet и FTP, требуют от пользователя определенных познаний в работе сети. Другие, например OSPF, обычно остаются скрытыми от пользователей. Системный администратор осознает существование всех этих приложений и всех протоколов уровней TCP/IP. Более того, он отвечает за их настройку!

Резюме

В этой главе мы обсудили структуру TCP/IP, семейства протоколов, составляющих основу сети Интернет. Иерархия TCP/IP состоит из четырех уровней: прикладного, транспортного, уровня Internet и уровня доступа к сети. Мы рассмотрели функциональность каждого из уровней. В следующей главе будет обсуждаться перемещение IP-дейтаграммы по сети в процессе доставки данных между узлами.

- *Адресация, маршрутизация и мультиплексирование*
- *Адрес IP*
- *Архитектура маршрутизации в Интернете*
- *Таблица маршрутизации*
- *Разрешение адресов*
- *Протоколы, порты и сокеты*

2

Доставка данных

В первой главе мы кратко изучили основы архитектуры и схему работы семейства протоколов TCP/IP и теперь знаем, что иерархия TCP/IP состоит из четырех уровней. В этой главе более пристально рассматривается перемещение данных между уровнями протоколов и компьютерами сети, уделяется внимание структуре адреса IP, в том числе маршрутизации данных на пути к адресату и локальному переопределению адресной структуры в целях образования подсетей. Кроме того, мы изучим номера протоколов и портов, которые используются для доставки данных соответствующим приложениям. Таким образом, мы перейдем от обзора TCP/IP к конкретным вопросам реализации, влияющим на настройку компьютерных систем.

Адресация, маршрутизация и мультиплексирование

Перенос данных между двумя узлами сети включает в себя передачу данных по сети на конечный узел, и уже в пределах этого узла – соответствующему пользователю или процессу. Для решения этих задач в TCP/IP применяются следующие три механизма:

Адресация

Адреса IP, уникальные «имена» узлов сети, обеспечивают возможность доставки данных нужному узлу.

Маршрутизация

Шлюзы передают данные в нужную сеть.

Мультиплексирование

Номера протоколов и портов позволяют передавать данные нужному программному модулю в пределах узла.

Каждый из механизмов – адресация узлов, маршрутизация между сетями и мультиплексирование между уровнями – необходим, чтобы взаимодействующие приложения могли обмениваться данными по сети Интернет. Рассмотрим каждый из механизмов подробнее.

Для иллюстрации понятий и организации ряда согласованных между собой примеров представим себе некоторую корпоративную сеть. В нашей воображаемой компании работают авторы компьютерных книг и преподаватели компьютерных курсов. Сеть компании состоит из нескольких сетей, объединяя центр обучения и издательскую контору, и связана с сетью Интернет. Мы отвечаем за управление сетью Ethernet из вычислительного центра. Структура сети, или ее *топология*, отражена на рис. 2.1.

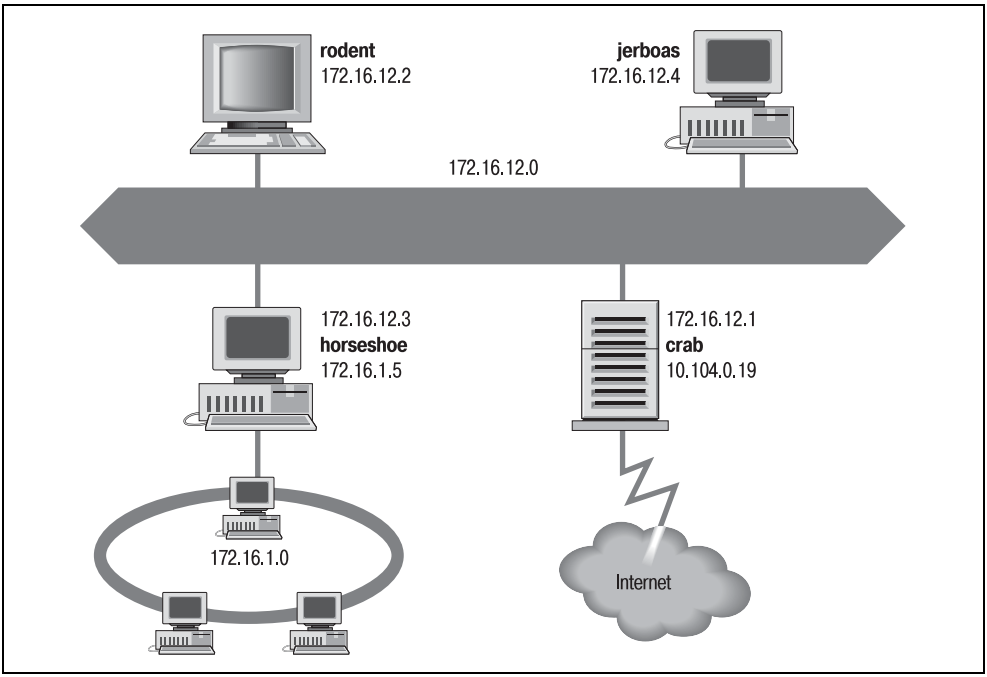


***Рис. 2.1.** Пример топологии сети*

Пиктограммы на рисунке представляют компьютерные системы. В нашей воображаемой сети есть, разумеется, и другие воображаемые компьютеры, но в большинстве наших примеров будут фигурировать узлы *rodent* (рабочая станция) и *crab* (система, выполняющая обязанности шлюза). Широкая разделительная линия обозначает Ethernet-шину компьютерного центра, овал – локальную сеть, объединяющую сети компании в одну корпоративную сеть. Облако – это сеть Интернет, а номерами представлены IP-адреса.

Адрес IP

Адрес IP – это 32-битное значение, уникальное для каждого устройства, подключенного к сети TCP/IP. Адрес IP обычно записывается в виде четырех десятичных чисел, разделенных точками.[1] Каждое десятичное число представляет один байт, состоящий из 8 бит 32-разрядного адреса, и лежит в интервале от 0 до 255 (возможные десятичные значения одного байта).

IP-адреса часто называют адресами узлов. Несмотря на широкое хождение такого названия, оно не совсем точное. Адреса IP назначаются не компьютерам, но сетевым интерфейсам. Система-шлюз, например *crab* (см. рис. 2.1), имеет разные адреса в каждой из сетей, в которую входит. Устройства определенной сети обращаются к шлюзу по его адресу в этой сети. Например, узел *rodent* обращается к узлу *crab* по адресу 172.16.12.1, тогда как внешние узлы – по адресу 10.104.0.19.

Обращение по адресу выполняется тремя различными способами. К отдельной системе обращение производится непосредственно по адресу узла, который называется *индивидуальным* (*unicast*) *адресом*. Индивидуальный пакет адресован единственному узлу. К группе систем обращение производится с помощью *групповых* (*multicast*) *адресов*, таких как 224.0.0.9. Маршрутизаторы вдоль пути от источника к адресату распознают специальные адреса и направляют копии пакетов всем членам группы.[2] Ко всем системам сети можно обратиться посредством широковещательных (broadcast) адресов, таких как 172.16.255.255. Широковещательный адрес зависит от возможностей конкретной физической сети.

Широковещательный адрес – хорошая иллюстрация того факта, что не все сетевые адреса или адреса узлов могут быть назначены сетевым устройствам. Некоторые из адресов узлов зарезервированы для служебных целей. Номера узлов 0 и 255 зарезервированы во всех сетях. Адрес IP, в котором все биты узла установлены в единицу, является *широковещательным адресом*.[3] Для сети 172.16 широковещательным адресом будет 172.16.255.255. Отправленная на этот адрес дейтаграмма будет доставлена каждому из узлов сети 172.16. Адрес IP, в котором все биты узла имеют нулевое значение, обозначает собственно сеть. Так, адрес 10.0.0.0 обозначает сеть 10, адрес 172.16.0.0 – сеть 172.16. Адреса в таком формате применяются в таблицах маршрутизации для обозначения целых сетей.

В адресе физической сети первый байт не может иметь значение, большее 223, поскольку такие адреса зарезервированы для служебных целей. Еще

---

[1] Иногда адреса записываются и в других форматах, например в виде шестнадцатеричных чисел. Независимо от представления структура и значение адреса остаются неизменными.

[2] Это лишь частично верно. Доставка по групповым адресам поддерживается не всеми маршрутизаторами. Иногда бывает необходимо передать данные через маршрутизаторы, инкапсулировав групповой пакет внутри индивидуального.

[3] В главе 5 описаны параметры настройки, позволяющие изменять стандартный широковещательный адрес.

два адреса сетей используются только в служебных целях: сеть 0.0.0.0 служит *маршрутом по умолчанию*, а сеть 127.0.0.0 является *кольцевой* (*loopback*). Маршрут по умолчанию позволяет упростить информацию по маршрутизации, с которой работает IP. Кольцевой адрес упрощает сетевые приложения, позволяя обращаться к локальному узлу таким же образом, как и к любому из удаленных. Эти специальные адреса сетей играют важную роль в настройке узла, однако не могут назначаться устройствам в реальных сетях. За исключением описанных случаев, адреса назначаются физическим устройствам и используются протоколом Internet для доставки данных на эти устройства.

Протокол Internet переносит данные между узлами в виде дейтаграмм. Каждая дейтаграмма доставляется по адресу, содержащемуся в поле *Адрес получателя* заголовка дейтаграммы (пятое слово). Адрес получателя – это стандартный 32-битный адрес, которого достаточно для однозначного определения сети и узла в этой сети.

Структура адреса

Адрес IP содержит *раздел сети* и *раздел узла*, но форматы этих разделов не одинаковы для всех IP-адресов. Число бит адреса, отведенное под обозначение сети, и, соответственно, число бит, отведенное под обозначение узла, изменяются в зависимости от длины префикса адреса. Длина префикса определяется битовой маской адреса.

Адресная маска работает следующим образом: если бит маски включен, соответствующий ему бит адреса входит в раздел адреса сети, а если бит маски сброшен, соответствующий бит адреса входит в раздел адреса узла. Например, для комбинации адреса 172.22.12.4 и сетевой маски 255.255.255.0, в которой включено 24 бита и сброшено 8 бит, первые 24 бита адреса представляют номер сети, а последние 8 бит – адрес узла. Сочетание адреса и маски позволяет определить, что речь идет об адресе узла 4 сети 172.22.12.

Указывать одновременно адрес и маску в десятичном представлении через точку не очень удобно, поэтому есть сокращенное представление. Вместо записи «сеть 172.31.26.32 с маской 255.255.255.224» можно писать просто 172.31.26.32/27. Представление имеет формат *адрес/длина_префикса*, где *длина_префикса* – число бит в разделе сети адреса. В отсутствие подобной записи существует большая вероятность, что адрес 172.31.26.32 будет интерпретирован неверно.

Организации, как правило, получают официальные адреса IP, покупая блоки адресов у поставщиков интернет-услуг. Одна организация обычно получает непрерывный блок адресов, соответствующий ее нуждам. Например, предприятие среднего размера может заказать блок адресов 192.168.16.0/20, тогда как маленькая организация ограничится блоком 192.168.32.0/24. Поскольку префикс определяет длину сетевой части адреса, количество адресов узлов, которые могут использоваться организацией (раздел адресов узлов) определяется вычитанием префикса из общего числа бит адреса (32). Таким образом, префикс 20 оставляет 12 бит для локального назначения адресов.

Такой блок адресов называют 12-битовым. Префикс 24 создает 8-битовый блок. Первый из блоков примера, 12-битовый, содержит 4096 адресов диапазона от 192.168.16.0 до 192.168.31.255, а второй, 8-битовый, содержит 256 адресов от 192.168.32.0 до 192.168.32.255.

Каждый из этих блоков адресов для внешнего мира является одним «сетевым» адресом. Внешние маршрутизаторы хранят лишь один маршрут к блоку 192.168.16.0/20 и один маршрут к блоку 192.168.32.0/24 независимо от реальных размеров этих адресных блоков. При этом внутри организации может существовать целый ряд физических сетей, объединяемых одним адресным блоком. Гибкость адресных масок позволяет поставщикам услуг выделять своим клиентам блоки адресов произвольной длины, а клиентам выполнять дальнейшее деление адресных блоков посредством префиксов различной длины.

Подсети

Структура адреса IP может изменяться на местах путем использования отдельных битов адреса узла в качестве дополнительных битов сетевого адреса. По сути дела изменяется расположение «границы» между битами адреса сети и битами адреса узла, что позволяет создавать дополнительные сети, но сокращает максимальное число адресов, допустимое для каждой сети. Эти дополнительные биты сети определяют в пределах родительского адресного блока другие блоки адресов, которые носят название *подсетей*.

В организациях подсети применяются, как правило, для разрешения топологических или организационных сложностей. Образование подсетей позволяет децентрализовать управление адресацией узлов. В случае стандартной схемы адресации один администратор отвечает за управление адресами узлов всей сети. Образование подсетей дает администратору возможность делегировать вопросы назначения адресов более мелким подразделениям организации, что может быть не только технической, но и политической необходимостью. Так, чтобы не иметь дел с отделом обработки данных, можно передать в ведение этого отдела самостоятельную подсеть.

Кроме того, подсети могут применяться в целях преодоления аппаратных различий и ограничений, накладываемых расстояниями. IP-маршрутизаторы способны связывать сети различного физического характера, но лишь в том случае, если каждой физической сети соответствует уникальный сетевой адрес. Образование подсетей – это разделение адресного блока на ряд уникальных адресов подсетей, так что каждая физическая сеть вполне может получить собственный уникальный адрес.

Подсеть создается изменением битовой маски IP-адреса. *Маска подсети* работает идентично обычной адресной маске: включенные биты интерпретируются в качестве битов адреса сети, а сброшенные принадлежат к разделу адреса узла. Различие состоит в том, что маски подсетей применяются лишь локально. Внешний мир по-прежнему интерпретирует адрес на основе известной ему маски.

Допустим, что есть небольшая компания, занимающаяся куплей-продажей недвижимости. Компании назначен блок адресов 192.168.32.0/24. Этому блоку соответствует битовая маска 255.255.255.0, и он содержит 256 адресов. Далее, предположим, что у компании есть 10 офисов и порядка 6 компьютеров в каждом. Необходимо выделить ряд адресов на каждый офис, зарезервировав часть из них на случай расширения. Блок из 256 адресов можно разделить на части при помощи маски подсети, расширив раздел адреса сети еще на несколько бит.

Чтобы разделить 192.168.32.0/24 на 16 подсетей, воспользуемся маской 255.255.255.240, то есть 192.168.32.0/28. Первые три байта содержат исходный адресный блок сети, четвертый состоит из адреса подсети и адреса узла в этой подсети. Выбранная маска предписывает использовать четыре старших бита четвертого байта в качестве раздела адреса подсети, а еще четыре бита – последние четыре бита четвертого байта – в качестве раздела адреса узла. Таким образом, мы получаем 16 подсетей, в каждой из которых существует 14 адресов узлов, что больше соответствует топологии сети нашей небольшой торговой компании. В табл. 2.1 перечислены подсети и адреса узлов, полученные посредством маски подсети для адреса сети 192.168.32.0/24.

Таблица 2.1. Действие маски подсети

Номер сети	Диапазон адресов узлов	Широковещательный адрес
192.168.32.0	192.168.32.1 – 192.168.32.14	192.168.32.15
192.168.32.16	192.168.32.17 – 192.168.32.30	192.168.32.31
192.168.32.32	192.168.32.33 – 192.168.32.46	192.168.32.47
192.168.32.48	192.168.32.49 – 192.168.32.62	192.168.32.63
192.168.32.64	192.168.32.65 – 192.168.32.78	192.168.32.79
192.168.32.80	192.168.32.81 – 192.168.32.94	192.168.32.95
192.168.32.96	192.168.32.97 – 192.168.32.110	192.168.32.111
192.168.32.112	192.168.32.113 – 192.168.32.126	192.168.32.127
192.168.32.128	192.168.32.129 – 192.168.32.142	192.168.32.143
192.168.32.144	192.168.32.145 – 192.168.32.158	192.168.32.159
192.168.32.160	192.168.32.161 – 192.168.32.174	192.168.32.175
192.168.32.176	192.168.32.177 – 192.168.32.190	192.168.32.191
192.168.32.192	192.168.32.193 – 192.168.32.206	192.168.32.207
192.168.32.208	192.168.32.209 – 192.168.32.222	192.168.32.223
192.168.32.224	192.168.32.225 – 192.168.32.238	192.168.32.239
192.168.32.240	192.168.32.241 – 192.168.32.254	192.168.32.255

Первая строка таблицы описывает подсеть с нулевым номером (первые четыре бита четвертого байта сброшены). Последняя строка описывает подсеть

с номером подсети, состоящим из единиц (первые четыре бита четвертого байта включены). Изначально документы RFC предполагали, что не следует применять номера подсетей, состоящие только из нулей или единиц. Однако из RFC 1812, *Requirements for IP Version 4 Routers* (Требования к маршрутизаторам IP версии 4), ясно следует, что применение таких подсетей совершенно законно и должно поддерживаться всеми маршрутизаторами. Некоторые из более старых маршрутизаторов не позволяли использовать такие адреса, несмотря на появление обновленных RFC. Современное программное и аппаратное обеспечение маршрутизаторов позволяет без проблем использовать любые адреса подсетей.

Нет необходимости вручную составлять таблицы, подобные приведенной, чтобы узнать, какие подсети и адреса узлов получаются в результате применения маски подсети. Все необходимые вычисления уже сделаны. Все возможные маски подсетей и допустимые для них адреса узлов перечислены в документе RFC 1878, *Variable Length Subnet Table For IPv4* (Таблица подсетей различной емкости для IPv4).

В RFC 1878 описаны все 32 значения префикса. При этом особой потребности в документации не возникает, поскольку правила для префиксов легко понять и запомнить. Запись адреса 10.104.0.19 в виде 10.104.0.19/8 показывает, что 8 бит адреса отведены под номер сети, и, следовательно, под номер узла остается 24 бита. К сожалению, не всегда все так гладко. Иногда маска адреса не указана явным образом, и в таких случаях необходимо уметь определить естественную маску, которая присваивается адресу по умолчанию.

Естественная маска

Изначально адресное пространство IP делилось на ряд структур фиксированной длины, известных в качестве *классов адресов*. Основных классов адресов было три – *класс A*, *класс B* и *класс C*. Программы IP определяли класс, а следовательно, и структуру адреса по его первым битам. Адресные классы вышли из употребления, но правила, применявшиеся для определения классов, в настоящее время используются для создания умолчаний адресных масок, называемых *естественными масками*. Правила следующие:

- Если первый бит IP-адреса сброшен, маска по умолчанию имеет длину 8 бит (префикс 8), что соответствует формату адреса сети прежнего класса A. Первые 8 бит определяют сеть, последние 24 бита определяют узел сети.
- Если первые два бита адреса имеют значения 1 0, длина маски по умолчанию – 16 бит (префикс 16), что соответствует формату адреса сети прежнего класса B. Первые 16 бит определяют сеть, последние 16 бит – узел сети.
- Если первые три бита адреса имеют значения 1 1 0, длина маски по умолчанию – 24 бита (префикс 24), что соответствует формату адреса сети прежнего класса C. Первые 24 бита определяют сеть, последние 8 бит определяют узел.

- Если первые четыре бита адреса принимают значения 1 1 1 0, адрес является групповым. Такие адреса иногда называли адресами *класса D*, но в действительности они не связаны с конкретными сетями. Групповые адреса используются для одновременного обращения к нескольким компьютерам. Они обозначают группы компьютеров, совместно использующих определенное приложение (скажем, для проведения видеоконференций), но не группы компьютеров, входящих в одну сеть. Каждый бит группового адреса имеет значение для маршрутизации, поэтому маска по умолчанию имеет длину 32 бита (префикс 32).

Когда адрес IP записывается в десятичном представлении через точку, удобнее бывает считать его набором 8-битных байтов, а не 32-битным значением. Когда речь идет о естественных масках, адрес также можно рассматривать как комбинацию цельных байтов адреса сети и адреса узла, поскольку три маски по умолчанию создают префиксы, битовые длины которых кратны числу 8. Простой способ определить маску по умолчанию – взглянуть на первый байт адреса. Если значение первого байта:

- меньше 128, маска по умолчанию имеет длину 8 бит; первый байт представляет номер сети, а три следующих байта – адрес узла;
- от 128 до 191, маска по умолчанию имеет длину 16 бит; два первых байта определяют сеть, два последних – узел сети;
- от 192 до 223, маска по умолчанию имеет длину 24 бита; три первых байта определяют адрес сети, а последний байт – номер узла;
- от 224 до 239, адрес является групповым. Весь адрес отведен под идентификацию группы машин; таким образом, длина маски по умолчанию – 32 бита;
- больше 239, адрес зарезервирован. Зарезервированные адреса интереса для нас не представляют.

Рисунок 2.2 иллюстрирует два метода определения структуры адреса по умолчанию. Первый адрес – 10.104.0.19. Первый бит данного адреса имеет значение 0; следовательно, первые 8 бит определяют сеть, а последние 24 бита определяют узел. Если говорить в терминах байтов, значение первого байта меньше 128, поэтому адрес интерпретируется как адрес узла 104.0.19 сети с номером 10. Один байт определяет сеть, три байта определяют узел.

Второй адрес – 172.16.12.1. Два старших бита имеют значения 1 и 0, значит, 16 бит определяют сеть и еще 16 бит определяют узел. В терминах байтов: первый байт имеет значение от 128 до 191, поэтому речь идет об адресе узла 12.1 сети 172.16. Два байта определяют сеть, а еще два – узел сети.

Наконец, в адресе 192.168.16.1 три старших бита имеют значения 1, 1 и 0, значит, сеть представлена 24 битами, а еще 8 бит обозначают узел. Первый байт адреса имеет значение от 192 до 223, поэтому речь идет об адресе узла 1 сети 192.168.16; три байта определяют сеть, один байт определяет узел.

Интерпретация адресов согласно приведенным правилам для классов ограничивает длины номеров сетей значениями 8, 16 или 24 бита – 1, 2 или 3 байта.

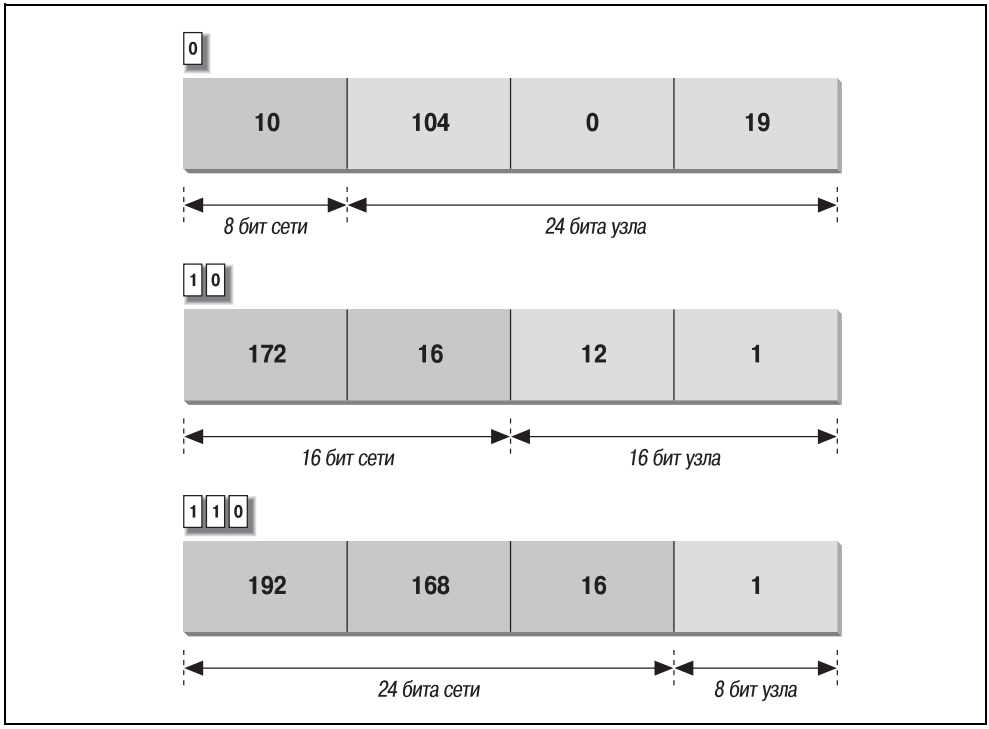


***Рис. 2.2.** Форматы адресов IP по умолчанию*

Однако в действительности IP-адрес образуют не байты, а 32 последовательных бита. *Битовая маска* адреса обеспечивает гибкий способ задания размеров разделов узла и сети в адресе. Протокол Internet использует раздел сети адреса для маршрутизации дейтаграмм между сетями. Полный адрес, включающий данные об узле, используется для обозначения конкретного узла. Такая двойственная роль адресов IP не только делает адресные маски средством более эффективного распределения адресов, но и позволяет им положительно влиять на маршрутизацию.

Блоки CIDR и объединение маршрутов

Одной из сильных сторон семейства протоколов TCP/IP являются IP-адреса, то есть универсальная схема адресации для сетей, составляющих Интернет. Тем не менее изначальная структура классов IP-адресов имела недостатки. Архитекторы TCP/IP не предвидели поразительных масштабов современной сети – во время разработки TCP/IP сети были привилегией организаций, достаточно обеспеченных для покупки солидных компьютерных систем. Никто и подумать не мог, что мощные машины под управлением системы Unix появятся на рабочих столах пользователей. На тот момент 32-битный адрес казался столь вместительным, что его разделили на классы, чтобы

сократить рабочую нагрузку маршрутизаторов, несмотря на то, что при этом резко сократилось число доступных для применения адресов. Назначение крупной сети одного класса B вместо шести классов C сокращало нагрузку на маршрутизатор, который мог хранить лишь один маршрут для целой организации. При этом в организации, получившей в свое распоряжение сеть класса B, вероятнее всего, не было шестидесяти четырех тысяч компьютеров, а потому большинство полученных адресов не использовались.

Классовая конструкция адреса подверглась критическим нагрузкам при быстром росте сети Интернет. В какой-то момент адреса класса B находились под угрозой полного исчезновения. Быстрое истощение запаса адресов класса B показало, что трех базовых адресных классов недостаточно: класс A слишком велик, а класс C имеет слишком маленькую емкость. Для большинства сетей излишеством был даже класс B, но его приходилось использовать за неимением лучшего.

Разрешить кризис с адресами класса B можно было очевидным путем – принудить организации использовать наборы адресов класса C. Свободные адреса этого класса исчислялись миллионами, и такому запасу не грозило скорое истощение. Однако, как часто случается, очевидное решение оказалось не таким простым, как на первый взгляд. Каждому адресу класса C должна соответствовать запись в таблице маршрутизации. Распространение миллионов адресов класса C привело бы к быстрому росту таблиц маршрутизации и перегрузке маршрутизаторов. Решение задачи требовало нового взгляда на функциональность адресных масок и нового метода распределения адресов.

Изначально сетевые адреса распределялись преимущественно в порядке следования, по мере поступления запросов. Метод замечательно работал для небольшой сети с централизованным управлением. Тем не менее он не учитывал топологию сети, а потому шансы того, что для передачи сообщений в сети 195.4.12.0 и 195.4.13.0 будет использоваться один набор промежуточных маршрутизаторов, определялись лишь случаем. В результате особую сложность приобретала задача минимизации размера таблицы маршрутизации. Набор адресов может рассматриваться в совокупности лишь в том случае, если адреса представлены последовательными номерами и доступны по одному маршруту. Например, для непрерывного блока адресов поставщика сетевых услуг может быть создан единый маршрут, поскольку количество подключений к сети Интернет для этой организации будет ограничено. Но если один сетевой адрес находится во Франции, а следующий за ним по порядку – в Австралии, создание общего маршрута для таких адресов становится невозможным.

В настоящее время длинные блоки последовательных адресов назначаются крупным поставщикам сетевых услуг в порядке, который более точно отражает топологию сети. Поставщики услуг выделяют фрагменты своих адресных блоков под нужды организаций, выступающих в роли клиентов. Распределение адресов, отражающее топологию сети, позволяет применять объединение маршрутов. Известно, что при такой системе сети 195.4.12.0 и 195.4.13.0 достижимы через один и тот же набор промежуточных маршру-

тизаторов. Более того, можно сказать, что оба адреса принадлежат диапазону, выделенному Европе: от 194.0.0.0 до 195.255.255.255.

Распределение адресов, отражающее топологию сети, *позволяет* применять объединение маршрутов. Чтобы схема заработала, необходима еще и реализация. Пока адреса 195.4.12.0 и 195.4.13.0 интерпретируются в качестве адресов различных сетей класса C, каждый из них требует отдельной записи в таблице маршрутизации. Развитие адресных масок не только расширило емкость адресного пространства, но и улучшило маршрутизацию.

Применение адресных масок вместо устаревших адресных классов для определения сети адресата носит название *бесклассовой междоменной маршрутизации* (*Classless Inter-Domain Routing*, CIDR[1]). CIDR требует модификации маршрутизаторов и протоколов маршрутизации. Наряду с конечными адресами, протоколы должны передавать адресные маски, позволяющие интерпретировать адреса. Маршрутизаторам и узлам необходима информация о том, как интерпретировать адреса в качестве «бесклассовых» и как применять битовую маску, сопровождающую адрес. Работа с адресными масками поддерживается всеми современными операционными системами и протоколами маршрутизации.

Предполагалось, что маршрутизация CIDR станет временным решением, но решение оказалось намного более долговечным, чем предполагали архитекторы. Система CIDR уже много лет является решением проблемы с адресацией и маршрутизацией и еще много лет останется разумным решением. Что касается долгосрочного решения проблемы истощения запасов адресов, оно заключается в обновлении системы адресации. В семействе протоколов TCP/IP за адресацию отвечает протокол IP. Комитет IETF – в целях создания новой адресной структуры – разработал новую версию IP, названную IPv6.

IPv6

IPv6 – это доработка протокола IP, основанная на двадцатилетнем опыте практического применения. Исходной причиной создания нового протокола была угроза истощения адресных емкостей. В IPv6 адрес имеет очень большой размер – 128 бит, что полностью исключает подобную опасность. Кроме того, длинные адреса позволяют использовать иерархическую адресную структуру для сокращения нагрузки на маршрутизаторы и оставляют более чем достаточный для дальнейшего расширения сети объем адресов. Среди прочих достоинств IPv6:

- Более совершенные механизмы обеспечения безопасности, встроенные в протокол
- Упрощенные заголовки фиксированной длины с выравниванием слов сокращают время на обработку заголовков и общие издержки на передачу данных
- Улучшенные методы обработки параметров заголовка

[1] Произносится «сайдр».

Несмотря на ряд полезных возможностей IPv6, протокол пока не получил широкого распространения. Отчасти это происходит потому, что развитие IPv4, повышение эффективности и производительности аппаратного обеспечения, а также изменения в способах настройки сетей сократили потребность в новых возможностях IPv6.

Критический дефицит адресов не проявился по трем причинам:

- CIDR придает распределению адресов дополнительную гибкость, что приводит к увеличению числа доступных адресов и позволяет сократить нагрузку на маршрутизаторы посредством объединения маршрутов.
- Скрытые адреса и преобразование сетевых адресов (NAT, Network Address Translation) значительным образом сократили потребность в официальных адресах. Многие организации предпочитают использовать собственную адресацию для всех систем в своих сетях, поскольку это сокращает затраты на администрирование и повышает уровень безопасности.
- Назначение фиксированных, постоянных адресов используется не столь часто, как динамическое назначение адресов. Большинство систем используют динамические адреса, временно арендуемые при помощи протокола настройки DHCP.

Появление стандарта IPsec для протокола IPv4 уменьшило потребность в более совершенных механизмах безопасности, реализованных в IPv6. Более того, многие из инструментов обеспечения безопасности, доступные для систем IPv4, не используются в полном масштабе, и потребность в средствах защиты соединений может быть попросту преувеличена.

В IPv6 устранено разбиение на транзитные участки, более эффективно спроектированы заголовки и осуществляется более совершенная обработка параметров. Эти обстоятельства делают пакеты IPv6 более эффективными в обработке, чем пакеты IPv4. Однако для большинства систем вопрос о повышении производительности не стоит, поскольку обработка дейтаграмм IP является второстепенной задачей. Подавляющее большинство систем находится на периферии сети и обрабатывает сравнительно небольшое число пакетов с данными. Процессорные мощности и объемы памяти растут с огромной скоростью – при том, что снижается уровень цен на компьютерные устройства. Мало кто из руководителей рискнет переходить на новый протокол IPv6 в целях экономии нескольких процессорных циклов, когда можно просто докупить аппаратное обеспечение, работающее с проверенным практикой протоколом IPv4. От повышения эффективности ощутимо выиграют лишь системы, расположенные близко к опорным магистралям сети, а число таких систем, пусть и весьма важных, сравнительно мало.

Все эти обстоятельства способствовали снижению востребованности протокола IPv6. Недостаток спроса ограничил число организаций, принявших IPv6 в качестве основного протокола обмена данными, а необходимым условием успешного распространения протокола является все-таки большое число пользователей. Мы используем протоколы обмена данными для общения с другими людьми. Если протоколом пользуется недостаточное число лю-

дей, мы просто не ощущаем потребности в этом протоколе. IPv6 по-прежнему находится в стадии ранней адаптации. Большинство организаций не используют IPv6 вовсе, а многие из тех, что используют, делают это исключительно в экспериментальных целях.[1] IPv6-взаимодействие между организациями выполняется путем инкапсуляции данных в дейтаграммах IPv4. Прежде чем протокол станет основным для существующих сетей, должно пройти какое-то время.

Руководителям действующих сетей не стоит проявлять чрезмерное беспокойство по поводу IPv6. Текущее поколение TCP/IP (IPv4), дополненное механизмами CIDR и других расширений, вполне соответствует нуждам существующих сетей. В таких сетях, включая Интернет, еще долго будут использоваться 32-битные IP-адреса протокола IPv4.

Архитектура маршрутизации в Интернете

В первой главе рассказывалось о развитии архитектуры сети Интернет. Изменения коснулись не только архитектуры, но и способов распространения информации маршрутизации по сети.

В исходной структуре Интернета существовала иерархия шлюзов, которая отражала факт происхождения от сети ARPAnet. При создании Интернета сеть ARPAnet стала магистралью сети, важным средством переноса трафика на большие расстояния. Магистраль получила название *стержня* (*core*), а объединяющие систему шлюзы с централизованным управлением – *стержневых шлюзов* (*core gateways*).

В описанной иерархии информация о маршрутизации для всех без исключения сетей, составлявших Интернет, передавалась стержневым шлюзам. Стержневые шлюзы обрабатывали информацию и обменивались информацией между собой – при помощи *протокола взаимодействия шлюзов* (*Gateway to Gateway Protocol*, CGP). Обработанная информация маршрутизации возвращалась внешним шлюзам. Стержневые шлюзы хранили самые точные данные по маршрутизации для всей сети Интернет.

Применение иерархической модели маршрутизации на базе стержневых маршрутизаторов имеет серьезные недостатки: любой маршрут должен быть обработан стержневой системой. Стержень подвергается невероятным нагрузкам, что и можно было наблюдать по мере роста сети. На сетевом жаргоне в таких случаях говорят, что данная модель маршрутизации «плохо масштабируется». Следствием этого стало создание новой модели.

Даже во времена единого стержня сети вне его существовали группы независимых сетей, именуемые автономными системами. В маршрутизации TCP/IP термин *автономная система* (*autonomous system*, AS) имеет особое значение. Автономная система – это не просто независимая сеть, но ряд сетей и

[1] Системы Solaris и Linux имеют поддержку IPv6, позволяя экспериментировать с применением протокола.

шлюзов с общим механизмом сбора информации маршрутизации и передачи ее другим независимым системам. Информация маршрутизации, которая передается другим сетевым системам, называется *информацией достижимости*. Информация достижимости сообщает, какие сети доступны через данную автономную систему. Во времена единого стержня автономные системы передавали информацию достижимости системам стержня – для обработки. Для обмена информацией достижимости со стержнем и другими автономными системами применялся *протокол внешних шлюзов* (*Exterior Gateway Protocol*, EGP).

Новая модель маршрутизации базируется на равенстве наборов автономных систем, называемых *доменами маршрутизации*. Домены маршрутизации обмениваются информацией маршрутизации с другими доменами при помощи *протокола граничных шлюзов* (*Border Gateway Protocol*, BGP). Каждый домен маршрутизации обрабатывает информацию, полученную от других доменов. В противоположность иерархической модели здесь нет зависимости от единственной стержневой системы при выборе «оптимальных» маршрутов. Каждый домен маршрутизации выполняет такую обработку самостоятельно, и, как следствие, модель лучше поддается масштабированию. На рис. 2.3 эта модель представлена тремя пересекающимися окружностями. Каждая окружность обозначает домен маршрутизации. Общие зоны окружностей являются пограничными, они служат для обмена информацией. Домены обмениваются данными, но источником полной информации маршрутизации уже не служит одна-единственная система.

В описанной модели существует следующая проблема: как выявлять «оптимальные» маршруты в глобальной сети в отсутствие центральной управляющей системы – такой, как стержень, – которой были бы доверены полномочия определения таких маршрутов? Во времена NSFNET для определения

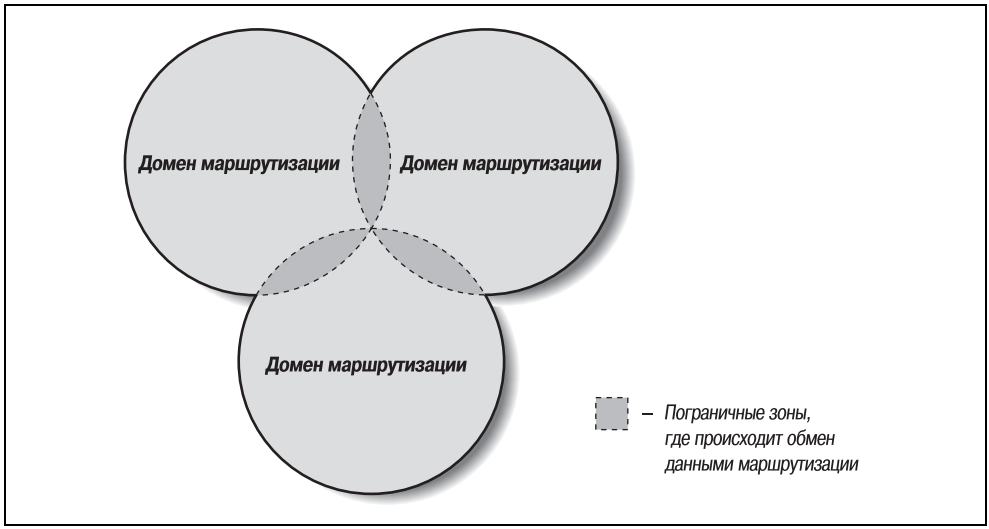


Рис. 2.3. *Домены маршрутизации*

корректности информации достижимости, предоставленной автономной системой, применялась *база данных правил маршрутизации* (*policy routing database*, PRDB). Но сегодня NSFNET уже не имеет прежних полномочий.

Чтобы заполнить пробел, NSF, наряду с *точками доступа к сети* (NAPs, *Network Access Points*), объединяющими разнообразных поставщиков сетевых услуг, создал серверы *арбитров маршрутизации* (*Routing Arbiter*, RA). Арбитр маршрутизации существует в каждой точке доступа к сети. Этот сервер обеспечивает доступ к *базе данных* RADB (*Routing Arbiter Database*), пришедшей на смену PRDB. Поставщики услуг Интернета могут обращаться к серверам RA для проверки корректности информации достижимости, предоставленной автономной системой.

База данных RADB является одной из частей *реестра маршрутизации сети Интернет* (*Internet Routing Registry*, IRR). Как и приличествует распределенной архитектуре маршрутизации, существует целый ряд организаций, отвечающих за проверку и регистрацию информации маршрутизации. Началась эта работа на европейском континенте. Европейские IP-сети обслуживаются организацией Reseaux IP Europeens (RIPE) Network Control Center (NCC). Крупные операторы сетевой связи предоставляют реестры своим клиентам. Все реестры работают с общим форматом данных, основанным на стандарте RIPE-181.

Многие поставщики услуг Интернета не используют серверы маршрутизации. Вместо этого они заключают формальные и неформальные двусторонние соглашения, определяющие, какого рода информация достижимости передается от одного поставщика услуг к другому. По сути дела, речь идет о создании частных правил маршрутизации. Небольшие компании, предоставляющие услуги Интернета, критиковали правила маршрутизации поставщиков услуг первого звена, заявляя, что эти правила ограничивают конкурентную борьбу. В результате многие поставщики услуг первого звена пообещали опубликовать правила, прояснив, таким образом, основы существующей архитектуры и открыв дорогу для возможных нововведений.

Создание эффективной архитектуры маршрутизации по-прежнему остается одной из серьезных задач, стоящих перед сетью Интернет, и эта архитектура с течением времени будет, вне всякого сомнения, эволюционировать. Но независимо от происхождения, информация маршрутизации в конечном итоге оказывается на локальном шлюзе, где и употребляется протоколом Internet для принятия решений по маршрутизации.

Таблица маршрутизации

Маршрутизацией данных между сетями занимаются шлюзы, но решения по маршрутизации приходится принимать всем участникам сети – не только шлюзам, но и обычным узлам. Для большинства узлов принятие таких решений подчиняется простому алгоритму:

- Если узел-адресат принадлежит локальной сети, данные доставляются напрямую адресату.

- Если узел-адресат принадлежит внешней сети, данные передаются на локальный шлюз.

Принятие решений по IP-маршрутизации ограничивается простым поиском в таблицах. Пакеты направляются в сторону получателя согласно данным *таблицы маршрутизации* (известной также как *таблица ретрансляции*). Таблица маршрутизации связывает пункты назначения с маршрутизаторами и сетевыми интерфейсами, которые протокол Internet должен использовать для доставки данных. Взглянем на таблицу маршрутизации Linux-системы.

В случае Linux-системы воспользуйтесь командой `route` с ключом `-n` для отображения таблицы маршрутизации.[1] Ключ `-n` предотвращает преобразование адресов IP в имена узлов, облегчая чтение результата. Таблица маршрутизации для некоторой системы Red Hat:

```
# route -n
Kernel IP routing table
Destination   Gateway      Genmask        Flags Metric Ref   Use Iface
172.16.55.0   0.0.0.0      255.255.255.0  U     0      0       0 eth0
172.16.50.0   172.16.55.36 255.255.255.0  UG    0      0       0 eth0
127.0.0.0     0.0.0.0      255.0.0.0      U     0      0       0 lo
0.0.0.0       172.16.55.1  0.0.0.0        UG    0      0       0 eth0
```

В системе Linux команда `route -n` отображает таблицу маршрутизации, состоящую из следующих полей:

`Destination`

Значение, с которым сопоставляется IP-адрес пункта назначения.

`Gateway`

Маршрутизатор, через который пролегает путь к указанному пункту назначения.

`Genmask`

Адресная маска, используемая при сопоставлении адреса IP со значением из поля Destination.

`Flags`

Определенные характеристики данного маршрута. В Linux существуют следующие значения флагов:[2]

`U` Указывает, что маршрут создан и является проходимым.

`H` Указывает на маршрут к определенном узлу (большинство маршрутов прокладывается в направлении сетей).

---

[1] Для изучения таблицы маршрутизации в системе Solaris 8 применяется команда `netstat`. Пример для системы Solaris приводится далее по тексту.

[2] Флаги `R`, `M`, `C`, `I` и `!` существуют только в Linux. Прочие флаги применяются в большинстве систем Unix.

G Указывает, что маршрут пролегает через внешний шлюз. Сетевой интерфейс системы предоставляет маршруты в сети с прямым подключением. Все прочие маршруты проходят через внешние шлюзы. Флагом G отмечаются все маршруты, кроме маршрутов в сети с прямым подключением.

R Указывает, что маршрут, скорее всего, был создан динамическим протоколом маршрутизации, работающим на локальной системе, посредством параметра `reinstate`.

D Указывает, что маршрут был добавлен в результате получения сообщения перенаправления ICMP (ICMP Redirect Message). Когда система узнает о маршруте из сообщения ICMP Redirect, маршрут включается в таблицу маршрутизации, чтобы исключить перенаправление для последующих пакетов, предназначенных тому же адресату. Такие маршруты отмечены флагом D.

M Указывает, что маршрут подвергся изменению – вероятно, в результате работы динамического протокола маршрутизации на локальной системе и применения параметра `mod`.

A Указывает на буферизованный маршрут, которому соответствует запись в таблице ARP.

C Указывает, что источником маршрута является буфер маршрутизации ядра. В большинстве систем существует две таблицы маршрутизации: база данных ретрансляции (Forwarding Information Base, FIB), представляющая для нас интерес, поскольку используется для принятия решений по маршрутизации, и буфер маршрутизации ядра, в течение некоторого времени хранящий данные по источникам и адресатам последних использовавшихся маршрутов. Флаг C упоминается в документации, но мне не приходилось встречать его в отображении таблицы маршрутизации, даже при включении в вывод буферизованных данных по маршрутизации.

L Указывает, что пунктом назначения маршрута является один из адресов данного компьютера. Такие «локальные маршруты» существуют только в буфере маршрутизации.

B Указывает, что конечным пунктом маршрута является широковещательный адрес. Такие «широковещательные маршруты» существуют только в буфере маршрутизации. В системе Solaris данный флаг назначается как широковещательным, так и сетевым адресам; так, адресам 172.16.255.255 и 172.16.0.0 системой Solaris, принадлежащей сети 172.16.0.0/16, назначается флаг B.

I Указывает, что маршрут связан с кольцевым (loopback) интерфейсом с целью иной, нежели обращение к кольцевой сети. Такие «внутренние маршруты» существуют только в буфере маршрутизации.

! Указывает, что дейтаграммы, направляемые по этому адресу, будут отвергаться системой. Linux позволяет пользователю самостоятельно создавать подобные «отрицательные» маршруты. Такие маршруты яв-

ным образом блокируют прохождение данных в определенном направлении. Данное значение флага существует только в Linux и редко применяется, хотя оно вполне допустимо.

Metric

Определяет «стоимость» маршрута. Метрика используется для сортировки дублирующих маршрутов, если таковые присутствуют в таблице. Кроме того, для использования метрики обязательно присутствие протокола динамической маршрутизации.

Ref

Зафиксированное число обращений к маршруту с целью создания соединения. Данное значение не используется в Linux-системах.

Use

Число обнаружений маршрута, выполненных протоколом IP.

Iface

Имя сетевого интерфейса[1], через который пролегает данный маршрут.

Каждая запись таблицы маршрутизации начинается с *направления*. Направление – это ключ, с которым сопоставляется IP-адрес пункта назначения при выборе маршрута, позволяющего доставить данные адресату. Значение направления обычно называют «целевой сетью», хотя диапазон значений для направления не ограничен адресами сетей. Значение направления может быть представлено адресом узла, групповым адресом, адресным блоком, представляющим объединение многих сетей, либо специальным значением – для маршрута по умолчанию или кольцевого адреса. В любом случае поле Destination содержит значение, с которым сопоставляется адрес получателя из заголовка пакета IP при выборе маршрута доставки для дейтаграммы.

Поле Genmask содержит битовую маску, применяемую протоколом IP к адресу получателя, указанному в пакете, чтобы определить, соответствует ли адрес значению направления из таблицы. Если бит маски включен, соответствующий бит адреса получателя является значимым в процессе сопоставления. Таким образом, адрес 172.16.50.183 соответствует второй записи из приведенной выше таблицы, поскольку выполнение операции логического «И» для данного адреса и маски 255.55.255.0 дает в результате 172.16.50.0.

Когда найдено соответствие адреса записи таблицы, поле Gateway позволяет протоколу IP определить, как достичь указанного направления. Если поле Gateway содержит IP-адрес маршрутизатора, используется этот маршрутизатор. Если поле Gateway содержит только нули (0.0.0.0, при выполнении команды `route` с ключом `-n`) или символ звездочки (*, при выполнении `route` без ключа `-n`), целевая сеть является подключенной напрямую сетью, а «шлюзом» оказывается сетевой интерфейс данной машины. Последнее из полей,

[1] *Сетевой интерфейс* – это устройство доступа к сети плюс программное обеспечение, используемое протоколом IP для взаимодействия с физической сетью. Более подробная информация приводится в главе 6.

заполняемых во всех записях таблицы, содержит имя сетевого интерфейса, через который пролегает маршрут. В нашем примере это первый интерфейс Ethernet (*eth0*) либо кольцевой интерфейс (*lo*). Направление, шлюз, маска и интерфейс полностью определяют маршрут.

Остальные поля (Ref, Use, Flags и Metric) содержат дополнительную информацию по маршруту. Значимость этой информации минимальна. Некоторые системы старательно отслеживают значение счетчика в поле Ref, а прочие, например Linux, вовсе не интересуются этим полем. Linux подсчитывает в поле Use, сколько раз потребовалось искать маршрут по запросу протокола IP из-за того, что он отсутствовал в буфере маршрутизации. Некоторые из прочих систем отображают в поле Use количество переданных по маршруту пакетов. Поле Flags содержит информацию, зачастую очевидную и без флагов: флаг U присутствует в определении каждого маршрута, поскольку – по определению – все маршруты таблицы являются работоспособными, а поле Gateway позволяет определить факт использования внешнего маршрутизатора без помощи флага G. Значение Metric задействовано только при работе с одним из вариантов протокола маршрутной информации (Routing Information Protocol, RIP). Эта информация не должна отвлекать внимание: главное в таблице маршрутизации – маршрут, который состоит из адресата, маски, шлюза и интерфейса.

Протокол IP использует информацию из таблицы маршрутизации (ретрансляции), чтобы строить маршруты для активных соединений. Маршруты, связанные с активными соединениями, хранятся в буфере маршрутизации. В Linux посмотреть на буфер маршрутизации можно при помощи команды `route` с ключом `-C`:

```
$ route -Cn
Kernel IP routing cache
Source          Destination     Gateway         Flags Metric Ref Use Iface
127.0.0.1       127.0.0.1       127.0.0.1       l     0      0     0 lo
192.203.230.10  172.16.55.3     172.16.55.3     l     0      0     0 lo
172.16.55.1     172.16.55.255   172.16.55.255   ibl   0      0   243 lo
172.16.55.2     172.16.55.255   172.16.55.255   ibl   0      0    15 lo
172.16.55.3     192.203.230.10  172.16.55.1           0      0     0 eth0
127.0.0.1       127.0.0.1       127.0.0.1       l     0      0     0 lo
172.16.55.3     132.163.4.9     172.16.55.1           0      0     0 eth0
172.16.55.2     172.16.55.3     172.16.55.3     il    0      0   149 lo
172.16.55.3     172.16.55.2     172.16.55.2           0      1     0 eth0
132.163.4.9     172.16.55.3     172.16.55.3     l     0      0     0 lo
```

Буфер маршрутизации отличается от таблицы маршрутизации – он отражает установленные маршруты. Таблица маршрутизации используется для принятия решений по маршрутизации, буфер маршрутизации используется *после* принятия решения. Буфер маршрутизации содержит адреса источника и получателя для сетевого соединения, а также значения для шлюза и интерфейса, через которые было установлено соединение.

Содержимое таблицы маршрутизации удобно изучать на примере Linux, поскольку в данной ОС команда `route` очень четко выводит эту таблицу. Коман-

да route системы Solaris существенно отличается в части синтаксиса. При работе с Solaris для вывода содержимого таблицы маршрутизации воспользуйтесь командой netstat -nr. Ключ -r предписывает отобразить таблицу маршрутизации, а ключ -n – использовать при этом числовой формат адресов.[1]

```
% netstat -nr
Routing Table: IPv4
Destination     Gateway        Flags  Ref    Use   Interface
-----------     -----------    -----  ----  -----  ---------
127.0.0.1       127.0.0.1      UH      1      298      lo0
default         172.16.12.1    UG      2    50360
172.16.12.0     172.16.12.2    U      40   111379      dnet0
172.16.2.0      172.16.12.3    UG      4     1179
172.16.1.0      172.16.12.3    UG     10     1113
172.16.3.0      172.16.12.3    UG      2     1379
172.16.4.0      172.16.12.3    UG      4     1119
```

Первая запись таблицы определяет *кольцевой маршрут* для локального узла. Это именно тот кольцевой адрес, о котором говорилось выше, как о зарезервированном номере сети. Каждая система использует кольцевой маршрут для отправки дейтаграмм самой себе, так что запись кольцевого маршрута присутствует в таблице маршрутизации любого узла. Флаг H говорит о том, что система Solaris создает маршрут к определенному узлу (127.0.0.1), а не к целой сети (127.0.0.0). Мы еще вернемся к кольцевому интерфейсу, когда будем говорить о настройке ядра и команде ifconfig. Сейчас же предметом нашего интереса являются внешние маршруты.

В приведенной таблице есть и другая особая запись – та, что содержит слово «default» в поле адресата. Это запись *маршрута по умолчанию*, а шлюз, указанный в записи, является *шлюзом по умолчанию*. Маршрут по умолчанию – еще один из ранее упомянутых зарезервированных номеров сетей. Шлюз по умолчанию используется в том случае, если ни один из конкретных маршрутов таблицы не соответствует указанному сетевому адресу получателя. Например, в таблице маршрутизации отсутствует запись для сети 192.168.16.0. Получив дейтаграммы, адресованные получателю в этой сети, протокол IP перешлет их через шлюз по умолчанию 172.16.12.1.

Все шлюзы, упоминаемые в таблице маршрутизации, принадлежат к сетям, напрямую подключенным к локальной системе. Для последнего примера это означает, что все адреса шлюзов начинаются с последовательности 172.16.12, вне зависимости от указанного адреса получателя. Это единственная сеть, с которой напрямую связан узел из примера, а значит – единственная сеть, в которую он может напрямую передавать данные. Шлюзы, используемые узлом для связи с остальным Интернетом, должны принадлежать к подсети этого узла.

---

[1] Система Linux при выводе таблицы маршрутизации отображает данные по адресным маскам. Solaris 8 поддерживает адресные маски, просто не отображает их в таблице маршрутизации.

На рис. 2.4 уровень IP двух узлов и шлюза нашей воображаемой сети заменены небольшими фрагментами таблицы маршрутизации, отражающими целевые сети и шлюзы, через которые осуществляется доступ к ним. Предположим, что для сети 172.16.0.0 используется адресная маска 255.255.255.0. Узел-источник (172.16.12.2), отправляя данные узлу-получателю (172.16.1.2), при помощи адресной маски определяет, что в таблице маршрутизации необходимо искать адрес целевой сети 172.16.1.0. Таблица маршрутизации узла-источника показывает, что данные, предназначенные сети 172.16.1.0, передаются через шлюз 172.16.12.3. Узел-источник передает пакеты шлюзу. Шлюз повторяет те же шаги и ищет целевой адрес в своей таблице маршрутизации. Шлюз 172.16.12.3 выполняет прямую доставку через собственный интерфейс 172.16.1.5. Обратившись к таблицам маршрутизации на рис. 2.4, можно увидеть, что на каждом узле упоминаются только шлюзы, принадлежащие к сетям, доступным с этого узла напрямую. Данное обстоятельство иллюстрирует тот факт, что шлюзом по умолчанию для адресов 172.16.12.2 и 172.16.12.3 является 172.16.12.1, но, поскольку узлу 172.16.1.2 недоступна напрямую сеть 172.16.12.0, он определяет иной маршрут по умолчанию.

Таблица маршрутизации не содержит сквозных маршрутов. Маршрут указывает лишь следующий шлюз, то есть задает ровно один *транзитный участок* (hop) на пути к целевой сети.[1] Доставку данных узел возлагает на локальный шлюз, а шлюз в доставке данных полагается на другие шлюзы. Дейтаграмма, переходя от одного шлюза к другому, рано или поздно достигает узла, который напрямую подключен к целевой сети. Именно этот последний шлюз доставляет данные узлу-получателю.

Для маршрутизации дейтаграмм между сетями протокол IP использует сетевую часть адреса. Полный адрес, включающий информацию об узле, вступает в игру в момент окончательной доставки, когда дейтаграмма достигает целевой сети.

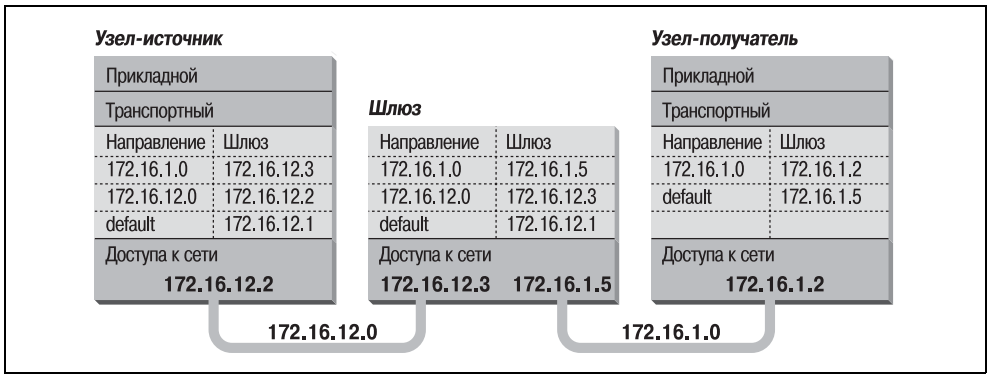


Рис. 2.4. *Табличная маршрутизация*

[1] Как мы увидим в главе 7, некоторые из протоколов маршрутизации, такие как OSPF и BGP, способны получать информацию по сквозной маршрутизации. Тем не менее пакет все так же передается следующему маршрутизатору в цепочке.

Разрешение адресов

Адрес IP и таблица маршрутизации направляют дейтаграмму в конкретную физическую сеть, но при прохождении через определенную сеть данные обязаны подчиняться протоколам физического уровня этой сети. Физическая сеть, на которой строится сеть TCP/IP, не воспринимает IP-адресацию. Физическая сеть определенного типа обычно реализует собственный, непохожий на другие, способ адресации. Одной из задач протокола доступа к сети является отображение адресов IP в адреса физической сети.

В качестве показательного примера данной функции уровня доступа к сети можно привести преобразование адресов IP в адреса Ethernet. За решение этой задачи отвечает *протокол разрешения адресов* (*Address Resolution Protocol*, ARP), определенный документом RFC 826.

Программный модуль ARP ведет таблицу соответствий между IP-адресами и адресами Ethernet. Таблица создается динамически. Когда ARP получает запрос на преобразование адреса IP, выполняется поиск адреса в таблице. Если адрес найден, протокол возвращает адрес Ethernet программе, от которой исходил запрос. В противном случае ARP выполняет широковещательную передачу всем узлам Ethernet-сети. Пакет содержит адрес IP, для которого требуется определить адрес Ethernet. Если один из узлов, получивших пакет, опознал адрес в качестве собственного, то в ответ посылает свой Ethernet-адрес узлу, от которого исходил запрос. Ответ заносится в таблицу ARP.

Команда `arp` выводит содержимое таблицы ARP. Чтобы отобразить полную таблицу ARP, воспользуйтесь командой `arp -a`. Доступ к отдельной записи можно получить, указав имя узла в качестве аргумента командной строки `arp`. Например, получить запись для узла *rodent* из ARP-таблицы узла *crab* можно с помощью команды:

```
% arp rodent
rodent (172.16.12.2) at 0:50:ba:3f:c2:5e
```

Отображение всех записей таблицы посредством ключа `-a` приводит к получению следующего результата:

```
% arp -a

Net to Media Table: IPv4
Device   IP Address               Mask      Flags   Phys Addr
------ -------------------- --------------- ----- ---------------
dnet0  rodent               255.255.255.255       00:50:ba:3f:c2:5e
dnet0  crab                 255.255.255.255 SP    00:00:c0:dd:d4:da
dnet0  224.0.0.0            240.0.0.0       SM    01:00:5e:00:00:00
```

Полученная таблица содержит информацию о том, что узел *crab*, ретранслируя дейтаграммы, адресованные узлу *rodent*, помещает их в Ethernet-фреймы и посылает их на Ethernet-адрес 00:50:ba:3f: c2:5e.

Одна из записей приведенной таблицы (*rodent*) была добавлена динамически в результате запросов, выполненных узлом *crab*. Две записи (*crab* и

224.0.0.0) являются статическими, они были созданы в процессе настройки узла *crab*. Это видно по флагу S (*static*) в поле Flags. Специальная запись *224.0.0.0* относится ко всем групповым адресам. Флаг M (*mapping*) означает *отображение* и назначается только записям для групповых адресов. В широковещательной среде Ethernet для окончательной доставки данных по групповому адресу используются широковещательные адреса.

Флаг P записи для узла *crab* означает, что запись подлежит «опубликованию». Флаг «публикации» указывает, что при получении ARP-запроса по IP-адресу узла *crab* данная система отвечает Ethernet-адресом 00:00:c0:dd:d4:da. Такое поведение вполне логично, поскольку речь идет об ARP-таблице на узле *crab*. Однако возможна публикация и Ethernet-адресов других узлов, не только локального. Обслуживание ARP-запросов, адресованных другим компьютерам, называется *представительством ARP*.

Допустим, система *24seven* является сервером для удаленной машины *clock*, которая подключается по коммутируемой телефонной линии. Чтобы не настраивать маршрутизацию в пользу удаленной системы, администратор сервера *24seven* может создать в таблице ARP статическую, публикуемую запись с IP-адресом машины *clock* и Ethernet-адресом машины *24seven*. Тогда сервер *24seven*, получая ARP-запрос для IP-адреса *clock*, отвечает собственным Ethernet-адресом. В результате другие машины сети посылают пакеты, предназначенные системе *clock*, узлу *24seven*. *24seven* передает пакеты системе *clock* по телефонной линии. Представительство ARP создается в целях обслуживания запросов к системам, которые не могут делать этого самостоятельно.

Таблицы ARP обычно не требуют внимания администратора, поскольку создаются автоматически очень надежным протоколом ARP. Но если возникают осложнения, таблицы ARP могут редактироваться вручную. Более подробная информация по теме содержится в разделе «Диагностирование при помощи команды arp» главы 13.

Протоколы, порты и сокеты

Когда маршрутизация по сети завершена, данные попадают на конечный узел, и их следует передать соответствующему пользователю или процессу. По мере продвижения данных вверх или вниз по уровням TCP/IP возникает необходимость передавать их нужному протоколу на каждом из уровней. Система должна обладать способностью передавать данные многих приложений – посредством не столь многочисленных транспортных протоколов – протоколу Internet. Объединение многих источников данных в один поток называется *мультиплексированием*.

Данные, поступающие из сети, необходимо *демультиплексировать*: разделить в целях доставки различным процессам. Для решения этой задачи в IP транспортные протоколы определяются *номерами протоколов*, а приложения определяются *номерами портов* в транспортных протоколах.

Отдельные протоколы и номера портов зарезервированы за *широко известными службами* (*well-known services*). *Широко известные службы* – это

стандартные сетевые протоколы (например, FTP и Telnet), имеющие широкое распространение в сети. Номера протоколов и номера портов назначаются широко известным службам организацией IANA (Internet Assigned Numbers Authority). Официально зарегистрированные номера задокументированы на сайте *http://www.iana.org*. Unix-системы хранят определения номеров протоколов и портов в паре текстовых файлов.

Номера протоколов

Номер протокола хранится в одном из байтов третьего слова заголовка дейтаграммы. Значение определяет протокол уровня над IP, которому следует передавать данные.

Номера протоколов для Unix-систем определяются в файле */etc/protocols*. Файл имеет простую табличную структуру и содержит пары, состоящие из имен протоколов и связанных с ними номеров. Каждая строка таблицы содержит одну запись: формальное имя протокола, отделенное от номера протокола пробельными символами. Номер протокола отделен пробельными символами от «псевдонима» протокола. Комментарии в таблице начинаются с символа #. Пример файла */etc/protocols* приводится ниже:

```
% cat /etc/protocols
#ident  "@(#)protocols  1.5     99/03/21 SMI"   /* SVr4.0 1.1   */

#
# Internet (IP) protocols
#
ip              0     IP              # pseudo internet protocol number
icmp            1     ICMP            # internet control message protocol
ggp             3     GGP             # gateway-gateway protocol
tcp             6     TCP             # transmission control protocol
egp             8     EGP             # exterior gateway protocol
pup             12    PUP             # PARC universal packet protocol
udp             17    UDP             # user datagram protocol
hmp             20    HMP             # host monitoring protocol
xns-idp         22    XNS-IDP         # Xerox NS IDP
rdp             27    RDP             # "reliable datagram" protocol

#
# Internet (IPv6) extension headers
#
hopopt          0     HOPOPT          # Hop-by-hop options for IPv6
ipv6            41    IPv6            # IPv6 in IP encapsulation
ipv6-route      43    IPv6-Route      # Routing header for IPv6
ipv6-frag       44    IPv6-Frag       # Fragment header for IPv6
esp             50    ESP             # Encap Security Payload for IPv6
ah              51    AH              # Authentication Header for IPv6
ipv6-icmp       58    IPv6-ICMP       # IPv6 internet control message protocol
ipv6-nonxt      59    IPv6-NoNxt      # IPv6No next header extension header
ipv6-opts       60    IPv6-Opts       # Destination Options for IPv6
```

Приведенная распечатка представляет содержимое файла */etc/protocols* рабочей станции под управлением Solaris 8. Данный перечень номеров никоим образом не претендует на полноту. Если заглянуть в раздел Protocol Numbers веб-сайта организации IANA, можно обнаружить гораздо больше номеров протоколов. Однако система может включать лишь те номера протоколов, которые используются в работе. Даже приведенный пример содержит больше номеров, чем на практике потребуется данной конкретной рабочей станции; так, вторая половина таблицы может использоваться только в системах, работающих по протоколу IPv6. Разумеется, если ваша система не работает с IPv6 или какими-либо вообще протоколами из перечисленных, поводов для опасений нет – лишние записи не причинят вреда.

Каков же смысл этой таблицы? Когда поступает дейтаграмма, адрес получателя которой совпадает с локальным IP-адресом, уровень IP обязан передать дейтаграмму одному из транспортных протоколов, расположенных уровнем выше. Решение о том, какому протоколу передать дейтаграмму, IP принимает на основе номера протокола, содержащегося в дейтаграмме. Из таблицы можно видеть, что если номер протокола для дейтаграммы равен 6, протокол Internet доставляет дейтаграмму протоколу TCP; если номер протокола равен 17, IP доставляет дейтаграмму протоколу UDP. TCP и UDP – это те две службы транспортного уровня, которые нас интересуют, но все протоколы, перечисленные в первой половине таблицы, напрямую используют службу доставки дейтаграмм протокола IP. Некоторые из таких протоколов – ICMP, EGP и GGP – упоминались ранее. О других речь не шла, но, по правде говоря, второстепенные протоколы не имеют значения для настройки и сопровождения сети TCP/IP.

Номера портов

Итак, протокол IP передал поступившие данные транспортному протоколу. Транспортный протокол, в свою очередь, передает данные соответствующему прикладному процессу. Прикладные процессы (известные также в качестве *сетевых служб*) определяются 16-битными номерами портов. Номер порта источника (определяет процесс, отправивший данные) и номер порта получателя (определяет процесс, которому данные предназначены) содержатся в первом слове заголовка каждого сегмента TCP и пакета UDP.

Номера портов до 1024 зарезервированы под широко известные службы (такие как FTP и Telnet) и назначаются организацией IANA. Широко известные номера портов считаются «привилегированными» и не должны ассоциироваться с пользовательскими процессами. Порты с номерами с 1024 по 49151 – «зарегистрированные». IANA по мере возможности ведет реестр служб, использующих порты указанного интервала, но данный список не имеет руководящей силы. Номера портов с 49152 по 65535 считаются «частными». Частные номера портов могут использоваться в любых целях.

Номера портов не уникальны для всего диапазона протоколов транспортного уровня, хотя уникальны для каждого протокола в отдельности. Иными словами, протоколы TCP и UDP могут использовать – и используют – одина-

ковые номера портов. Именно сочетание номеров порта и протокола точно определяет конкретный процесс, которому предназначены данные.

Номера портов для Unix-систем определяются в файле */etc/services*. Сетевых приложений существует гораздо больше, чем протоколов транспортного уровня, как можно видеть из размеров таблицы */etc/services*. Ниже приводится фрагмент файла */etc/services* рабочей станции под управлением Solaris 8:

```
rodent% head -22 /etc/services
#ident  "@(#)services   1.25    99/11/06 SMI"   /* SVr4.0 1.8   */
#
#
# Copyright (c) 1999 by Sun Microsystems, Inc.
# All rights reserved.
#
# Network services, Internet style
#
tcpmux          1/tcp
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp-data        20/tcp
ftp             21/tcp
telnet          23/tcp
```

Формат данного файла схож с форматом файла */etc/protocols*. Каждая однострочная запись начинается с формального имени службы, отделяемого пробельными символами от пары *номер порта/протокол*. Номера портов дополняются именами транспортных протоколов потому, что с одинаковыми номерами портов могут работать различные транспортные протоколы. Необязательный список псевдонимов формального имени службы может присутствовать после пары номер/протокол.

Пара файлов */etc/protocols* и */etc/services* предоставляет всю информацию, необходимую для доставки данных соответствующему приложению. Дейтаграмма передается получателю на основе адреса, указанного в пятом слове заголовка дейтаграммы. Используя номер протокола из третьего слова заголовка дейтаграммы, протокол IP выполняет доставку данных надлежащему протоколу транспортного уровня. Первое слово данных, полученных транспортным протоколом, содержит номер порта получателя, который позволяет передать данные уровнем выше, конкретному приложению. Процесс доставки отражен на рис. 2.5.

Несмотря на внушительные размеры, файл */etc/services* содержит номера портов далеко не всех важных сетевых служб. Так, в нем отсутствуют номе-

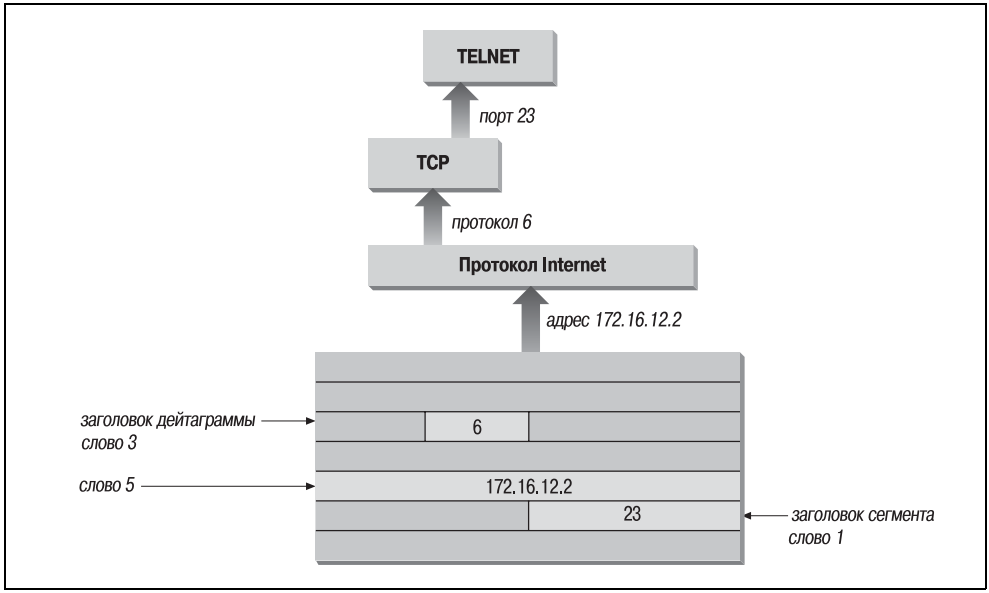


***Рис. 2.5.** Номера протоколов и портов*

ра портов для служб RPC (Remote Procedure Call, *удаленный вызов процедур*), перечисленных в файле *services*. Компания Sun разработала свой метод резервирования портов для служб RPC, не связанный с получением широко известного номера от IANA. Службы RPC обычно используют зарегистрированные номера портов, не требующие официальных процедур для присвоения. Когда запускается служба RPC, она регистрирует свой номер порта при помощи программы `portmapper`. `portmapper` отслеживает номера портов, занятых службами RPC. Когда клиент обращается к службе RPC, он посылает запрос демону `portmapper`, работающему на сервере, с целью получения номера порта, назначенного службе. Клиент может легко обнаружить службу `portmapper`, поскольку ей назначен широко известный порт с номером 111. `portmapper` делает возможной установку широко применяемых служб без необходимости получать широко известный номер официальным путем.

Сокеты

Широко известные порты – это стандартизированные номера, дающие возможность удаленным системам обращаться к сетевым службам через заранее известные порты. Такая возможность упрощает процесс подключения, поскольку и адресат и отправитель заранее знают, что данные, адресованные конкретному процессу, пройдут через определенный порт. Так, все системы, предоставляющие доступ по протоколу Telnet, используют для этой цели порт 23.

Не менее важен и второй тип номера порта, известный в качестве *динамически выделяемого порта*. Как следует из названия, динамически выделяемые

порты не назначаются заранее: они назначаются процессам по мере необходимости. Механизм гарантирует, что двум различным процессам не будут назначены одинаковые номера портов и что выделяемые порты лежат вне диапазона широко известных номеров, то есть имеют номера, большие 1024.

Динамически выделяемые порты обеспечивают гибкость, необходимую для реализации многопользовательских систем. Если пользователю `telnet` назначается порт с номером 23 как для источника, так и для получателя, какие номера портов следует назначить второму пользователю, работающему с `telnet` в то же время? В целях однозначного определения каждого соединения в качестве номера исходного порта используется динамически выделенный номер, а в качестве номера целевого порта – широко известный номер службы.

В примере с программой `telnet` первому пользователю назначается случайный номер исходного порта и номер целевого порта 23 (telnet). Второму пользователю назначается другой случайный номер исходного порта и тот же номер для целевого порта. Именно пара номеров портов, исходного и целевого, однозначно определяет каждое сетевое соединение. Узлу-получателю известен номер исходного порта, поскольку информация о нем включается как в заголовки сегментов TCP, так и в заголовки пакетов UDP. Оба узла знают номер целевого порта, поскольку он является широко известным.

Обмен номерами портов при установлении связи TCP показан на рис. 2.6. Узел-источник случайным образом выбирает исходный порт, в данном примере – 3044, и посылает сегмент с указанием исходного порта 3044 и целевого порта 23. Узел-получатель принимает сегмент и отвечает номером исходного порта 23 и номером целевого порта 3044.

Сочетание IP-адреса и номера порта называется *сокетом* (*socket*). Сокет однозначно определяет любой сетевой процесс сети Интернет. Иногда термины «сокет» и «номер порта» используются в качестве взаимозаменяемых. Более того, широко известные службы часто называют «широко известными сокетами». В контексте нашего разговора «сокет» – это сочетание адреса IP и номера порта. Пара сокетов – сокет узла-получателя и сокет узла-источника – определяет соединение для протоколов, ориентированных на работу с соединениями, таких как TCP.

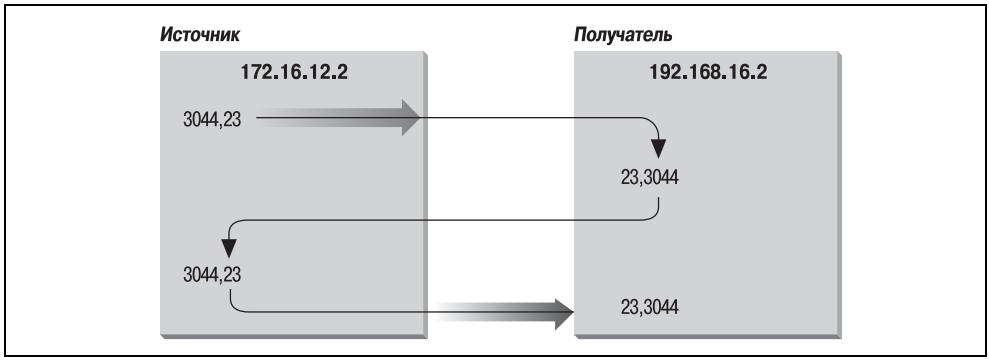


***Рис. 2.6.** Обмен номерами портов*

Разовьем пример динамически назначаемых портов и широко известных портов. Предположим, что пользователь узла 172.16.12.2 использует Telnet для подключения к узлу 192.168.16.2. Узел 172.16.12.2 является источником. Пользователю динамически назначается уникальный номер порта 3382. Выполняется подключение к службе telnet удаленного узла, которому, в соответствии со стандартом, назначен широко известный порт 23. Сокетом источника соединения будет 172.16.12.2.3382 (IP-адрес 172.16.12.2 плюс номер порта 3382). Сокетом другого конца соединения – 192.168.16.2.23 (адрес 192.168.16.2, порт 23). Порт целевого сокета известен обеим системам, поскольку является широко известным. Порт исходного сокета известен обеим системам, поскольку узел-источник сообщил второй стороне сокет источника, запрашивая соединение. Таким образом, пара сокетов известна и той, и другой машине. Сочетание двух сокетов однозначным образом определяет установленное соединение; никакое другое соединение в сети Интернет не будет иметь совпадающей пары сокетов.

Резюме

В этой главе читатели узнали, как данные передаются по глобальной сети Интернет – от конкретного процесса на исходном компьютере к взаимодействующему процессу на другом конце света. В TCP/IP для идентификации компьютеров сети применяется общая система уникальных адресов. Номера протоколов и номера портов позволяют однозначно определять пользовательские процессы, работающие на компьютерах сети.

Маршрутизация направляет дейтаграммы, предназначенные удаленному процессу, по лабиринтам глобальной сети. При этом часть адреса IP используется для определения целевой сети. В каждой системе существует таблица маршрутизации, определяющая пути отправки данных в удаленные сети. Обычно таблица маршрутизации содержит маршрут по умолчанию, используемый в случаях, когда отсутствует маршрут в конкретную удаленную сеть. Маршрут лишь указывает на следующий компьютер в пути к получателю данных. В TCP/IP применяется система транзитной маршрутизации, позволяющая шаг за шагом перемещать дейтаграммы ближе к адресату, пока они не достигнут целевой сети.

Внутри целевой сети окончательная доставка осуществляется на основе полного IP-адреса (включающего номер узла), посредством преобразования этого адреса в адрес физического уровня. В качестве примера протокола, осуществляющего преобразование IP-адресов, можно привести протокол разрешения адресов (Address Resolution Protocol, ARP). Протокол ARP преобразует адреса IP в адреса Ethernet для окончательной доставки.

Первые две главы были посвящены структуре стека протоколов TCP/IP и механизму передачи данных по сети. В следующей главе мы поднимемся по стеку протоколов, чтобы изучить сетевые службы, упрощающие настройку и работу в сети.

3

Сетевые службы

- *Имена и адреса*
- *Таблица узлов*
- *DNS*
- *Почтовые службы*
- *Серверы файлов и печати*
- *Серверы настройки*

Некоторые серверы сети предоставляют доступ к важным службам уровня взаимодействия машин сети. Такие службы не эквивалентны прикладным службам, поскольку обычные пользователи не работают с ними напрямую. Напротив, эти службы задействуются сетевыми компьютерами и упрощают создание, настройку и работу сети.

Функции серверов, описываемые в этой главе, весьма разнообразны:

- Службы имен для преобразования IP-адресов в имена узлов
- Серверы настройки, упрощающие добавление узлов сети, частично или полностью выполняя настройку TCP/IP
- Службы электронной почты, осуществляющие перемещение сообщений по сети от автора к адресату
- Файловые серверы, организующие прозрачный совместный доступ клиентов к файлам
- Серверы печати, централизующие управление печатающими устройствами и обеспечивающие совместный доступ к ним многих пользователей

Серверы сети TCP/IP не следует путать с традиционными PC-серверами локальной сети. Любой узел сети под управлением Unix-системы может быть как сервером, так и клиентом. Узлы сети TCP/IP являются «равноправными». Все системы равны, и сеть не зависит от работы единственного сервера. Службы, о которых пойдет речь далее, могут устанавливаться на одной или нескольких системах сети.

Наш рассказ начинается со службы имен. Эта жизненно необходимая служба обязательно пригодится вам в работе.

Имена и адреса

Документ, посвященный протоколу Internet[1], определяет имена, адреса и маршруты следующим образом:

> Имя показывает искомый нами объект. Адрес показывает его местонахождение. Маршрут определяет способ попасть в нужную точку.

Имена, адреса и маршруты в равной степени требуют внимания сетевого администратора. Маршрутам и адресам мы посвятили предыдущую главу. В этом разделе пойдет речь об именах и их распространении по сети. Каждый сетевой интерфейс в сети TCP/IP определяется уникальным 32-битным адресом IP. Любому устройству с IP-адресом может быть назначено имя (известное в качестве имени узла, *hostname*). Причина назначения имен устройствам заключена в том, что имена, в сравнении с числовыми адресами Internet, легче запоминать и набирать. Имена не нужны для работы сетевых программ, но они облегчают людям работу с сетью.

В большинстве случаев имена узлов и числовые адреса взаимозаменяемы. Чтобы обратиться посредством telnet к рабочей станции с IP-адресом 172.16.12.2, пользователь может набрать:

```
% telnet 172.16.12.2
```

либо воспользоваться именем узла, связанным с указанным адресом, и набрать эквивалентную команду:

```
% telnet rodent.wrotethebook.com
```

И в том и в другом случае сетевое подключение выполняется на основе адреса IP. Система преобразует имя узла в адрес перед открытием соединения. Сетевой администратор отвечает за назначение имен и адресов, сохраняя их в базах данных, используемых для преобразования.

Перевод имен в адреса нельзя назвать «частной» проблемой. Команда `telnet rodent.wrotethebook.com` должна работать корректно с любого узла, подключенного к сети. Если узел *rodent.wrotethebook.com* входит в сеть Интернет, узлы во всем мире должны иметь возможность преобразовать имя *rodent.wrotethebook.com* в соответствующий адрес. Очевидно, необходимы механизмы распространения информации по именам узлов и обеспечения этой информацией всех узлов сети.

Широкое распространение получили два метода перевода имен в адреса. Более старый сводится к поиску имени узла в таблице, известной в качестве *таблицы узлов*.[2] Более современный связан с применением механизма распределенной базы данных, получившего название *системы доменных имен* (*Domain Name System*, DNS). Сначала мы рассмотрим таблицу узлов.

---

[1] RFC 791, *Internet Protocol*, Jon Postel, ISI, 1981, с. 7.

[2] Сетевая информационная служба (Network Information Service, NIS), разработанная Sun, представляет собой более совершенный механизм доступа к таблице узлов. Речь о NIS пойдет далее в этой главе.

Таблица узлов

Таблица узлов – это обычный текстовый файл, сопоставляющий IP-адреса с именами узлов. В большинстве Unix-систем таблица хранится в файле */etc/hosts*. Каждая строка таблицы */etc/hosts* содержит адрес IP, отделенный пробелом от перечня имен узлов, связанных с этим адресом. Символ # отмечает начало комментария.

Таблица узлов системы *rodent* может содержать такие записи:

```
#
# Table of IP addresses and hostnames
#
172.16.12.2     rodent.wrotethebook.com rodent
127.0.0.1       localhost
172.16.12.1     crab.wrotethebook.com crab loghost
172.16.12.4     jerboas.wrotethebook.com jerboas
172.16.12.3     horseshoe.wrotethebook.com horseshoe
172.16.1.2      ora.wrotethebook.com ora
172.16.6.4      linuxuser.articles.wrotethebook.com linuxuser
```

Первая строка таблицы-примера относится непосредственно к системе *rodent*. IP-адресу 172.16.12.2 поставлено в соответствие имя узла *rodent.wrotethebook.com* и альтернативное имя узла (псевдоним) *rodent*. Имя узла и все его псевдонимы отображаются в один и тот же адрес IP, в данном случае – 172.16.12.2.

Псевдонимы позволяют менять имена, написание имен, а также использовать сокращенные варианты. Кроме того, псевдонимы позволяют создавать «обобщенные имена узлов». Обратите внимание на запись для адреса 172.16.12.1. Один из псевдонимов адреса – *loghost*. *loghost* – это особое имя узла, используемое в файле настройки *syslog.conf* систем Solaris. В дистрибутивах некоторых систем программы, подобные `syslogd`, настраиваются таким образом, чтобы их вывод передавался узлу с определенным именем. Вывод можно передавать любому из узлов, назначив ему соответствующее обобщенное имя в качестве псевдонима. Среди обобщенных имен встречаются такие, как *lprhost*, *mailhost* и *dumphost*.

Вторая запись из файла назначает адрес 127.0.0.1 узлу с именем *localhost*. Как говорилось ранее, сетевой адрес 127.0.0.0/8 зарезервирован под кольцевую сеть. Адрес узла 127.0.0.1 – это специальный адрес, назначаемый в качестве кольцевого локальному узлу (отсюда имя *localhost*). Такое специальное соглашение об адресации позволяет узлу обмениваться данными с самим собой таким же способом, как и с удаленными узлами. Кольцевой адрес упрощает создание программного обеспечения, поскольку позволяет использовать один и тот же код для общения с локальными и удаленными процессами. Кроме того, описанное соглашение сокращает сетевой трафик: адрес узла *localhost* связан с кольцевым устройством, возвращающим данные узлу без отправки в сеть.

На смену таблице узлов пришли механизмы DNS, но она по-прежнему широко применяется по следующим причинам:

- В большинстве систем есть небольшие файлы с таблицами узлов, содержащими имена и адреса важных узлов локальной сети. Эти небольшие таблицы используются, когда система DNS недоступна, в частности, в процессе начальной загрузки машины. Даже при наличии DNS следует создавать файлы */etc/hosts*, содержащие записи для самого узла, узла *localhost*, шлюзов и серверов локальной сети.
- Там, где применяется NIS, таблицы узлов используются в качестве исходных данных для баз данных NIS. NIS может использоваться в сочетании с DNS, но даже в этом случае на большинстве NIS-площадок создаются таблицы узлов, содержащие записи для всех узлов локальной сети. Совместное использование NIS и DNS описано в главе 9.
- Таблицы узлов иногда применяют в очень маленьких, не связанных с сетью Интернет, сетях. Если узлов в сети мало, информация об узлах изменяется редко, и нет необходимости во взаимодействии с внешним миром по TCP/IP, особых преимуществ использование DNS не даст.

Устаревший метод таблицы узлов не отвечает потребностям глобальной сети Интернет по двум причинам: из-за невозможности масштабирования и отсутствия автоматизированного процесса обновления. До появления DNS организация под названием NIC (Network Information Center, Сетевой информационный центр) занималась сопровождением огромной таблицы узлов сети Интернет (*таблица узлов NIC*). Включенные в эту таблицу узлы носили название *зарегистрированных*. Организация NIC добавляла в таблицу имена и адреса узлов для всех площадок сети Интернет.

Во времена, когда таблица узлов еще была основным средством для перевода имен узлов в IP-адреса, большинство подсетей регистрировали в таблице лишь самые важные системы. Но даже при таком ограничении таблица выросла до размеров, которые исключали эффективное преобразование имен в IP-адреса. Невозможно использовать простую таблицу для обслуживания огромного числа узлов, составляющих сегодняшний Интернет.

Вторая проблема, связанная с таблицей узлов, – отсутствие механизма автоматического распространения информации о новых узлах. К новым, зарегистрированным узлам можно обращаться по имени только после того, как будет получена новая версия таблицы узлов. Однако нет возможности гарантировать, что таблица узлов достигнет той или иной локальной сети, и нет возможности узнать, актуальна ли используемая версия таблицы. Отсутствие гарантированного единообразного распространения информации – серьезный недостаток системы таблиц узлов.

DNS

DNS исключает оба крупных недостатка таблиц узлов:

- DNS хорошо масштабируется. Система не зависит от единственной большой таблицы, в основе ее работы лежит распределенная база данных, рост объема которой практически не сказывается на эффективности. На

сегодня DNS снабжает сеть информацией о примерно 100 000 000 узлов, тогда как таблица NIC включала не более 10 000 узлов.

- DNS гарантирует распространение информации о новых узлах по мере необходимости.

Информация распространяется автоматически и только по необходимости. Система доменных имен работает следующим образом. Сервер DNS, получив запрос информации о неизвестном ему узле, передает его *компетентному* (*авторитативному*) *серверу* (*authoritative server*). Компетентный сервер – это любой сервер, в ведении которого находится точная информация по домену, о котором идет речь в запросе. Локальный сервер, получив от компетентного сервера ответ, сохраняет, или *кэширует*, его для последующего использования. Получив запрос той же информации еще раз, локальный сервер отвечает на запрос самостоятельно. Способность получать информацию об узлах из компетентных источников и автоматически распространять точные сведения дает DNS значительное преимущество перед таблицами узлов, даже в случае сетей, не подключенных к сети Интернет.

Система DNS пришла на смену не только таблице узлов, но и более старому варианту службы имен. Так сложилось, что обе службы называют *службой имен*. Обе упоминаются в файле */etc/services*. Более старой системе назначен UDP-порт 42 и имя *nameserver*, либо *name*; служба имен DNS использует порт 53 и носит имя *domain*. Естественно, это может послужить источником недоразумений. Однако не стоит беспокоиться – старая служба имен устарела. В тексте книги речь идет только о DNS, и, говоря «служба имен», мы будем иметь в виду именно ее.

Иерархия доменов

DNS – это распределенная иерархическая система, предназначенная для преобразования имен узлов в адреса IP. В DNS не существует централизованной базы данных, содержащей информацию по всем узлам Интернет. Информация распределена между многими тысячами серверов имен, связанных в единую иерархию, схожую с иерархией файловой системы Unix. Вершиной иерархии доменов DNS является *корневой домен*, обслуживанием которого занимается группа серверов имен, известных как *корневые серверы*.

Аналогично тому как каталоги файловой системы Unix описываются с помощью путей, пролегающих от корневого каталога через промежуточные, так и поиск информации о доменах осуществляется следованием по указателям: от корневого домена – через промежуточные – к целевому.

Непосредственно под корневым доменом располагаются *домены верхнего уровня*. Существует две основные категории доменов верхнего уровня – географические и организационные. Каждая страна мира получила географический домен с двухбуквенным именем-кодом. Поэтому данный тип доменов называется *доменом верхнего уровня с кодом страны* (ccTLD, *country code top-level domain*). Например, ccTLD для Великобритании – *.uk*, для Японии – *.jp*, а для США – *.us*. Обычно для домена высшего уровня .us доменом

второго уровня служит почтовый код штата США (к примеру, *.wy.us*, если речь идет о штате Вайоминг). Географические домены в США обычно используются администрациями штатов и образовательными учреждениями уровня школ, но редко когда узлами иного рода.

В пределах США наиболее популярные домены верхнего уровня – это организационные домены, членство в которых основано на типе организации (коммерческая, военная и т. д.), к которой принадлежит компьютерная система.[1] Такие домены называются *родовыми доменами верхнего уровня*, или *универсальными доменами верхнего уровня* (gTLDs, *generic top-level domains*).

Ниже перечислены официально зарегистрированные родовые домены верхнего уровня:

com

Коммерческие организации

edu

Образовательные учреждения

gov

Правительственные организации

mil

Военные организации

net

Организации, обеспечивающие работу сетевой инфраструктуры

int

Международные правительственные или околоправительственные организации

org

Организации, не попадающие ни в одну из перечисленных выше категорий, в частности некоммерческие предприятия

aero

Организации авиационной промышленности

biz

Предприятия

coop

Совместные предприятия

museum

Музеи

---

[1] В США структурные и географические домены никак не связаны. Каждая система принадлежит только к структурному либо только к географическому домену, но не к тому и другому одновременно.

pro

Профессионалы, такие как медики и адвокаты

info

Сайты, предоставляющие информацию

name

Физические лица

Итого, в настоящее время существует четырнадцать родовых доменов верхнего уровня. Первые семь доменов списка (*com*, *edu*, *gov*, *mil*, *net*, *int* и *org*) существовали в доменной системе изначально. Еще семь доменов (*aero*, *biz*, *coop*, *museum*, *pro*, *info* и *name*) были созданы в 2000 году с целью увеличения числа доменов верхнего уровня. Одной из причин для создания дополнительных gTLD послужил гигантский размер домена *.com*. Домен настолько велик, что его размеры осложняют эффективное ведение базы данных *.com*. Вопрос о том, смогут ли новые домены уменьшить регистрацию в *.com*, остается открытым.

На рис. 3.1 приводится иерархия доменов, включающая шесть изначальных организационных доменов верхнего уровня. Вершина дерева называется корнем. Непосредственно под корневым доменом располагаются домены верхнего уровня. Корневые серверы обладают полной информацией лишь о доменах верхнего уровня. Полной информацией по всем доменам не обладает ни один сервер – включая корневые, но корневые серверы предоставляют указатели на серверы доменов второго уровня.[1] Так что корневые серверы, не обладая ответом на вопрос, знают, кого следует спрашивать.

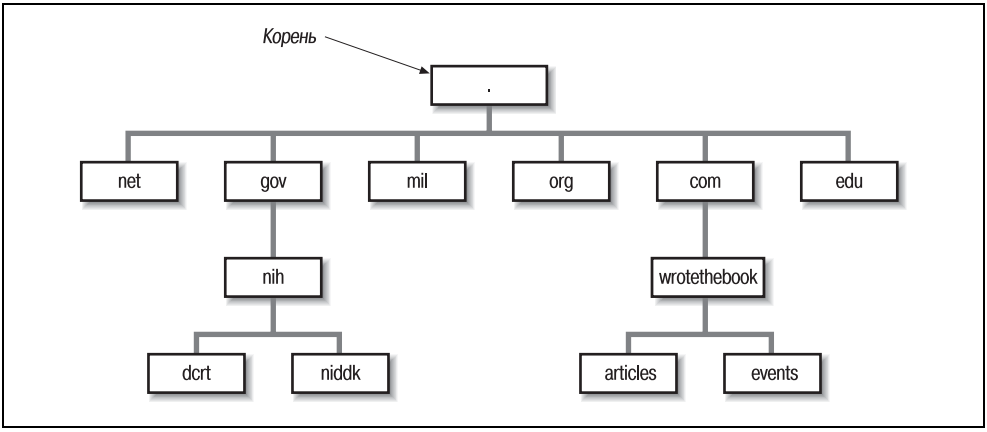


***Рис. 3.1.** Иерархия доменов*

[1] На рис. 3.1 присутствует пара доменов второго уровня: *nih* в домене *gov* и *wrotethebook* в домене *com*.

Создание доменов и поддоменов

Некоммерческая организация ICANN (Internet Corporation for Assigned Names and Numbers), созданная в целях сопровождения процесса выделения доменных имен и IP-адресов, делегирует часть своих полномочий ряду организаций-регистраторов. (Прежде процесс находился в ведении правительства США.) Регистраторы, авторизованные ICANN, имеют право выделять домены. Чтобы получить домен, необходимо обратиться к регистратору, чтобы получить право создать домен в одном из доменов верхнего уровня. (Подробности процесса получения доменного имени рассмотрены в главе 4.) Получив разрешение на создание домена, можно создавать дополнительные домены – *поддомены* – в пределах созданного. Рассмотрим процедуру на примере нашей воображаемой компании.

Наша компания – это коммерческое, приносящее прибыль (хотелось бы надеяться) предприятие. Очевидно, оно должно располагаться в домене *com*. Мы обращаемся в уполномоченную организацию, чтобы создать домен *wrotethebook* в домене *com*. Запрос нового домена содержит имена и адреса серверов, которые будут обеспечивать работу службы имен домена. Одобрив запрос, регистратор помещает в домен *com* указатели на серверы имен нового домена. Теперь корневые серверы, получая запросы по домену *wrotethebook.com*, будут перенаправлять их к новым серверам имен.

Одобрение регистратора дает нам полную власть над новым доменом. Владелец любого зарегистрированного домена имеет право делить домен на поддомены. Наша воображаемая компания может разбить домен на раздел, отвечающий за специальные события (*events.wrotethebook.com*), и раздел, координирующий подготовку журнальных статей (*articles.wrotethebook.com*), не обращаясь при этом к регистратору или в иные «вышестоящие инстанции». Решения по добавлению поддоменов принимаются на уровне локального администратора домена. Регистраторы делегируют полномочия и распределяют управление именами между отдельными организациями. Делегирование полномочий означает, что организация становится ответственной за управление выделенными ей именами.

Новый поддомен становится доступен, когда указатели на серверы нового домена размещаются в домене уровнем выше (рис. 3.1). Удаленные серверы не способны обнаружить домен *wrotethebook.com*, пока указатель на сервер этого домена не будет размещен в домене *com*. Точно так же нельзя обратиться к поддоменам *events* и *articles*, если указатели на них не размещены в домене *wrotethebook.com*. На серверы имен домена указывает запись в базе данных DNS, имеющая тип NS (*name server*). Запись этого типа содержит имя домена и имя узла, который является сервером имен домена. Конкретные примеры для базы данных DNS приводятся в главе 8. Пока что условимся считать эти записи просто указателями.

Применение NS-записей в качестве указателей отражено на рис. 3.2. Локальному серверу необходимо преобразовать имя *linuxuser.articles.wrotethebook.com* в адрес IP. В кэше сервера отсутствуют сведения о домене *wrotethe-*

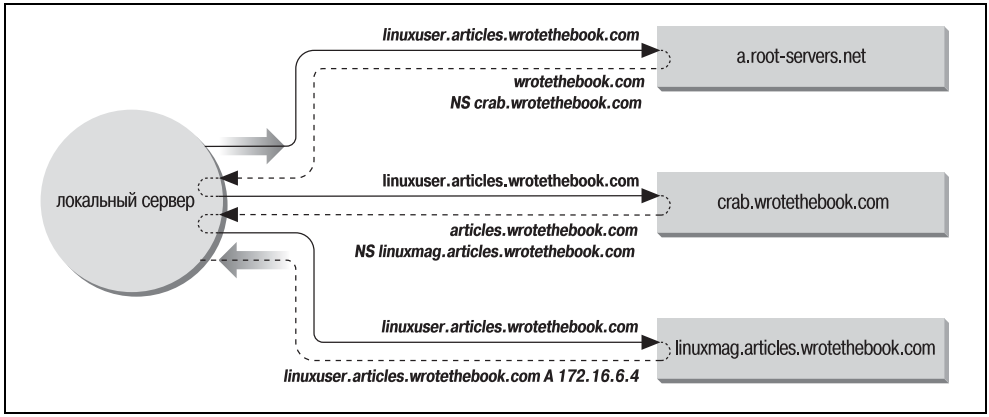


Рис. 3.2. *DNS-запрос*

book.com, поэтому происходит обращение к корневому серверу имен (в нашем примере *a.root-servers.net*) с запросом искомого адреса. Корневой сервер отвечает NS-записью, указывающей на узел *crab.wrotethebook.com* в качестве источника информации по *wrotethebook.com*. Локальный сервер посылает запрос узлу *crab*, который перенаправляет его к узлу *linuxmag.articles.wrotethebook.com*, серверу домена *articles.wrotethebook.com*. Локальный сервер обращается к *linuxmag.articles.wrotethebook.com* и получает, наконец, искомый адрес IP. Локальный сервер кэширует полученную A-запись (адресную запись) и все NS-записи. Получив следующий запрос адреса *linuxuser.articles.wrotethebook.com*, локальный сервер сможет ответить на него самостоятельно. Получив следующий запрос, связанный с доменом *wrotethebook.com*, локальный сервер обратится напрямую к узлу *crab*, не вовлекая в процесс корневой сервер.

Рисунок 3.2 содержит примеры как рекурсивного, так и нерекурсивного поиска. Удаленные серверы выполняют *нерекурсивный* поиск – они лишь указывают локальному серверу, к кому обращаться далее. Локальный сервер следует по указателям самостоятельно – и является *рекурсивным* сервером. В случае *рекурсивного* поиска сервер следует по указателям и возвращает окончательный ответ на запрос. Корневые серверы, как правило, выполняют только *нерекурсивный* поиск. Большинство других серверов выполняют рекурсивный поиск.

Доменные имена

Доменные имена отражают иерархию доменов. Запись доменных имен производится в направлении от частного (имени узла) к общему (домену верхнего уровня) и разделяется точками.[1] *Абсолютное*, или *полностью определен-*

[1] Корневой домен записывается в виде точки, то есть корневое имя представлено пустой меткой, а потому записывается как «.».

ное, *доменное имя* (FQDN, *fully qualified domain name*) начинается с имени конкретного узла и заканчивается именем домена верхнего доменного уровня. *rodent.wrotethebook.com* – это абсолютное доменное имя (FQDN) рабочей станции *rodent*, расположенной в поддомене *wrotethebook* домена *com*.

Имена доменов не всегда записываются в абсолютной форме. Они могут записываться относительно *домена по умолчанию* – точно так же, как пути в Unix записываются относительно текущего рабочего каталога. При создании запроса к серверу имен DNS добавляет ко вводу пользователя имя домена по умолчанию. Например, если доменом по умолчанию является *wrotethebook.com*, пользователь может опускать расширение *wrotethebook.com* для всех узлов этого домена. Обращаться к узлу *crab.wrotethebook.com* можно будет по имени *crab*; DNS самостоятельно добавит к имени домен по умолчанию, *wrotethebook.com*.

В большинстве систем доменное имя по умолчанию добавляется только в том случае, если в имени искомого узла отсутствуют точки. Так, имя *linuxuser.articles* не подвергнется расширению, а значит, и не будет найдено сервером имен, поскольку не существует домена высшего уровня с именем articles. Но имя узла *crab*, не содержащее точки, будет дополнено именем *wrotethebook.com*, что даст в результате корректное доменное имя *crab.wrotethebook.com*. Как читатели узнают из главы 8, поведение DNS в данном аспекте поддается настройке, что весьма в духе Unix-систем.

Использование домена по умолчанию и конструирование запросов – аспекты, зависящие от настройки программного обеспечения. По этой причине следует проявлять осторожность, встраивая имя узла в код программы. Только абсолютные доменные имена и IP-адреса полностью защищены от изменений в ПО сервера имен.

BIND, анализаторы и named

Самой распространенной реализацией DNS для систем Unix является пакет *Berkeley Internet Name Domain* (BIND). Описания в тексте основаны на реализации сервера имен из пакета BIND.

Концептуально программное обеспечение DNS делится на две категории – поисковые анализаторы (resolvers) и серверы имен. *Анализатор* (или *клиент DNS*) – это программный модуль, формирующий запрос; он задает вопросы. *Сервер имен* – это процесс, реагирующий на запросы; он дает ответы.

Поисковый анализатор не является отдельным процессом, выполняемым на компьютере. Это библиотека подпрограмм (*код поискового анализатора*), которая используется любой программой, нуждающейся в функции поиска адресов. Библиотека умеет задавать серверам имен вопросы, касающиеся информации об узлах.

В случае BIND все машины используют код поискового анализатора, но не на каждой работает процесс сервера имен. Компьютер, на котором не работает локальный процесс сервера имен и который полагается на другие узлы

для работы со службой имен, называется *чистым DNS-клиентом*. Обычно чистыми DNS-клиентами оказываются однопользовательские системы. В более крупных Unix-системах обычно существует локальный процесс сервера имен.

Сервер имен BIND представлен самостоятельным процессом демона *named* (произносится «нэйм-ди»). Классификация серверов имен основана на способах их настройки. Вот три главных категории серверов имен:

Основной (Master)

Основной сервер (также известный как *первичный* (*primary*) *сервер*) – это сервер, служащий источником всех данных по домену. Основной сервер загружает информацию о домене напрямую с диска, из файла, созданного администратором домена. Основные серверы являются *компетентными* (*authoritative*), то есть обладают полной информацией по обслуживаемым доменам и всегда дают правильные ответы. Должен существовать лишь один основной сервер для домена.

Подчиненный (Slave)

Подчиненные серверы (также известные как *вторичные*) получают полную базу данных домена от основного сервера. База данных для отдельного домена называется *файлом зоны*; копирование этого файла на подчиненный сервер называется *передачей зоны*. Подчиненный сервер гарантирует актуальность своих данных, периодически получая файл зоны домена. Подчиненные серверы также являются компетентными для обслуживаемых доменов.

Специальный кэширующий (Caching-only)

Специальные кэширующие серверы получают ответы на все запросы от других серверов имен. Получив ответ на запрос, сервер кэширует информацию и в будущем использует ее, чтобы самостоятельно давать ответы на запросы. Большинство серверов имен практикуют кэширование. Специальные кэширующие серверы используют *исключительно* этот метод для построения базы данных домена. Кэширующие серверы являются *некомпетентными* (*non-authoritative*), их информация получена из вторых рук и неполна, хотя обычно точна.

Взаимоотношения различных типов серверов – это преимущество DNS перед таблицами узлов, которое ощутимо для большинства сетей, даже очень маленьких. В DNS каждому домену требуется лишь один первичный сервер имен. Данные DNS вводятся в базу данных первичного сервера администратором домена. Следовательно, централизованное управление информацией об узлах оказывается в руках администратора. Автоматически распределяемая база данных с централизованным управлением является плюсом для сети любого размера. При добавлении новой системы в сеть не возникает необходимости редактировать файлы */etc/hosts* на всех узлах сети; достаточно изменить базу данных DNS на первичном сервере. Информация автоматически распространяется по другим серверам, которые получают зону полностью либо кэшируют отдельные ответы.

Сетевая информационная служба (NIS)

Сетевая информационная служба (Network Information Service, NIS[1]) – это административная система баз данных, разработанная компанией Sun Microsystems. Она предоставляет централизованное управление и автоматическое распространение важных административных файлов. NIS может применяться совместно с DNS либо в качестве альтернативы.

NIS и DNS обладают как сходствами, так и различиями. Подобно DNS, система NIS избавляет от проблемы точного распространения таблицы узлов, но, в отличие от DNS, является решением лишь для локальных сетей. NIS не предназначается для сети Интернет в целом. Еще одно отличие – NIS предоставляет доступ к более широкому диапазону информации, чем DNS, то есть выходит далеко за рамки простого преобразования «имя-адрес». NIS преобразует ряд стандартных файлов Unix в базы данных, доступные для запросов по сети. Такие базы данных получили название *карт NIS*.

NIS преобразует в карты такие файлы, как */etc/hosts* и */etc/networks*. Карты могут храниться на выделенном сервере, где централизованно сопровождаются, оставаясь при этом полностью доступными для клиентов NIS. Централизованное сопровождение в сочетании с автоматическим распространением позволяет NIS восполнить один из крупных недостатков таблицы узлов. Однако NIS не является альтернативой DNS для сети Интернет, поскольку таблицы узлов, а следовательно, и карты NIS содержат лишь часть информации, которой владеет DNS. По этой причине DNS и NIS обычно применяются в паре.

В этой главе представлены понятия об именах узлов и обзор различных методов преобразования имен узлов в адреса IP. Разумеется, это лишь фрагмент картины. Назначение имен узлов и управление службой имен – важные задачи сетевого администратора, и эти темы будут еще не раз подняты в книге. Подробные же сведения приводятся в главе 8.

Служба имен – не единственная служба, которая появится в ваших сетях. Наверняка пригодится и еще одна – служба электронной почты.

Почтовые службы

Пользователи считают электронную почту важнейшей из сетевых служб, поскольку используют ее для общения друг с другом. Одни приложения более современны и функциональны, другие нуждаются в существенной пропускной способности каналов сети, и еще целый ряд приложений служит для непрерывной работы в сети. Электронная почта – приложение для общения людей друг с другом – является жизненно важной составляющей, пусть и не очень изысканной.

TCP/IP обеспечивает надежную, гибкую систему электронной почты, построенную на ряде базовых протоколов. Речь идет о протоколах SMTP (*Simple Ma-*

[1] Исходное название NIS – «Желтые страницы» (Yellow Pages, или *yp*). Несмотря на смену имени, сокращение *yp* по-прежнему в ходу.

il Transfer Protocol, простой протокол передачи почты), POP (*Post Office Protocol*, протокол почтовой службы), IMAP (*Internet Message Access Protocol*, Интернет-протокол доступа к сообщениям), а также MIME (*Multipurpose Internet Mail Extensions*, многоцелевые расширения почтовой службы в Интернете). Прочие почтовые протоколы TCP/IP, обладающие привлекательными возможностями, пока не получили широкого распространения.

Ниже мы поговорим о четырех протоколах, которые почти наверняка применяются в любой локальной сети: SMTP, POP, IMAP и MIME. Начнем исследования с протокола SMTP, фундамента всех систем электронной почты TCP/IP.

Простой протокол передачи почты (SMTP)

SMTP – это TCP/IP-протокол доставки электронных сообщений. Он переносит почту по сети Интернет и по локальным сетям. Определение SMTP дается в документе RFC 821, *A Simple Mail Transfer Protocol*. Протокол функционирует на базе надежной службы логических соединений *протокола управления передачей* (*Transmission Control Protocol*, TCP) и работает через порт с широко известным номером 25.[1] В табл. 3.1 перечислены некоторые из простых команд SMTP, вполне пригодных для восприятия человеком.

Таблица 3.1. Команды SMTP

Команда	Синтаксис	Назначение
Hello	HELO <*узел-отправитель*> EHLO <*узел-отправитель*>	Источник SMTP-отправки
From	MAIL FROM:<*исходный адрес*>	Адрес отправителя
Recipient	RCPT TO:<*конечный адрес*>	Адрес получателя
Data	DATA	Начать сообщение
Reset	RSET	Прервать сообщение
Verify	VRFY <*строка*>	Проверить имя пользователя
Expand	EXPN <*строка*>	Раскрыть список рассылки
Help	HELP [*строка*]	Запросить справку
Quit	QUIT	Завершить сеанс SMTP

SMTP – протокол настолько простой, что работу по этому протоколу пользователь может в буквальном смысле осуществлять самостоятельно. Подключившись посредством `telnet` к порту 25 удаленного узла, наберите почтовое сообщение в командной строке при помощи команд SMTP. Этот способ иногда используется для тестирования SMTP-сервера удаленной системы, а мы с

[1] Большинству стандартных приложений TCP/IP назначены широко известные номера портов, чтобы облегчить удаленным системам задачу обращения к определенной службе.

его помощью проиллюстрируем процесс передачи почты между системами. В приводимом ниже примере пользователь Daniel с узла *rodent.wrotethebook.com* вручную набрал и отправил письмо пользователю Tyler узла *crab.wrotethebook.com*.

```
$ telnet crab 25
Trying 172.16.12.1...
Connected to crab.wrotethebook.com.
Escape character is '^]'.
220 crab.wrotethebook.com ESMTP Sendmail 8.9.3+Sun/8.9.3; Thu, 19 Apr 2001
16:28:01-0400 (EDT)
HELO rodent.wrotethebook.com
250 crab.wrotethebook.com Hello rodent [172.16.12.2], pleased to meet you
MAIL FROM:<daniel@rodent.wrotethebook.com>
250 <daniel@rodent.wrotethebook.com>... Sender ok
RCPT TO:<tyler@crab.wrotethebook.com>
250 <tyler@crab.wrotethebook.com>... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
Hi Tyler!
.
250 QAA00316 Message accepted for delivery
QUIT
221 crab.wrotethebook.com closing connection
Connection closed by foreign host.
```

Ввод пользователя выделен жирным шрифтом. Остальные строки представляют вывод системы. Как видите, все очень просто. Открывается TCP-соединение. Система-отправитель представляется. Указывает адреса *From* и *To*. Передача сообщения начинается командой DATA и заканчивается строкой, содержащей только точку (.). Сеанс завершается по команде QUIT. Очень просто, с использованием лишь нескольких команд.

В RFC 821 определены и другие команды (SEND, SOML, SAML, TURN), которые являются необязательными и не получили широкого распространения в реализациях SMTP. Даже некоторые из реализованных команд не находят широкого применения. Команды HELP, VRFY и EXPN предназначены скорее для диалоговой работы, нежели для обычного взаимодействия двух систем по протоколу SMTP. Следующий фрагмент сеанса SMTP демонстрирует работу этих дополнительных команд.

```
HELP
214-This is Sendmail version 8.9.3+Sun
214-Topics:
214-    HELO    EHLO    MAIL    RCPT    DATA
214-    RSET    NOOP    QUIT    HELP    VRFY
214-    EXPN    VERB    ETRN    DSN
214-For more info use "HELP <topic>".
214-For local information contact postmaster at this site.
214 End of HELP info
HELP RSET
214-RSET
```

```
214-    Resets the system.
214 End of HELP info
VRFY <jane>
250 <jane@brazil.wrotethebook.com>
VRFY <mac>
250 Kathy McCafferty <<mac>>
EXPN <admin>
250-<sara@horseshoe.wrotethebook.com>
250 David Craig <<david>>
250-<tyler@wrotethebook.com>
```

Команда HELP выводит справку по командам, реализованным в системе. Команда HELP RSET запрашивает информацию по команде RSET. Откровенно говоря, эта справочная система не очень-то полезна!

Команды VRFY и EXPN более полезны, но часто блокируются из соображений безопасности, поскольку предоставляют доступ к информации пользовательских учетных записей, которая может использоваться для атаки на систему. Команда `EXPN <admin>` запрашивает перечень адресов электронной почты из списка рассылки *admin*, и система услужливо возвращает такой список. Команда VRFY запрашивает информацию не о списке рассылки, но о конкретном пользователе. В случае команды `VRFY <mac>`, *mac* оказывается локальной учетной записью пользователя, и для этой записи отображается определенная информация. В случае `VRFY <jane>`, *jane* – это псевдоним из файла */etc/aliases*. Возвращаемое значение – электронный адрес *jane*, указанный в этом файле. Три команды последнего примера интересны, но редко используются. Свою работу SMTP выполняет при помощи других команд.

SMTP обеспечивает прямую сквозную доставку почты. Прочие почтовые системы, такие как UUCP и X. 400, используют *протоколы передачи с промежуточным хранением*, за одно действие передавая почту на шаг ближе к пункту назначения – следующей системе. Каждая из систем в конце транзитного участка сохраняет сообщение полностью, а затем передает его следующей системе. Сообщение путешествует таким образом, пока не попадет в пункт назначения. Рисунок 3.3 иллюстрирует почтовые системы с промежуточным хранением данных и с прямой доставкой. Путь UUCP позволяет четко проследить маршрут почтовых сообщений, а почтовый адрес SMTP подразумевает прямую доставку.[1]

Прямая доставка позволяет SMTP доставлять почту, не полагаясь на промежуточные узлы. Если доставка невозможна, локальная система узнает об этом немедленно. Она может уведомить пользователя, что сообщение отправлено, либо поместить сообщение в очередь для последующей отправки – все так же без помощи удаленных систем. Минус прямой доставки в том, что она требует от обеих систем полной готовности к работе с почтовыми сообщениями. Некоторые системы не пособны работать с почтой напрямую, в осо-

[1] Адрес никоим образом не определяет способ доставки в почтовой системе. Просто так получилось, что адрес в UUCP дополнительно иллюстрирует контекст.

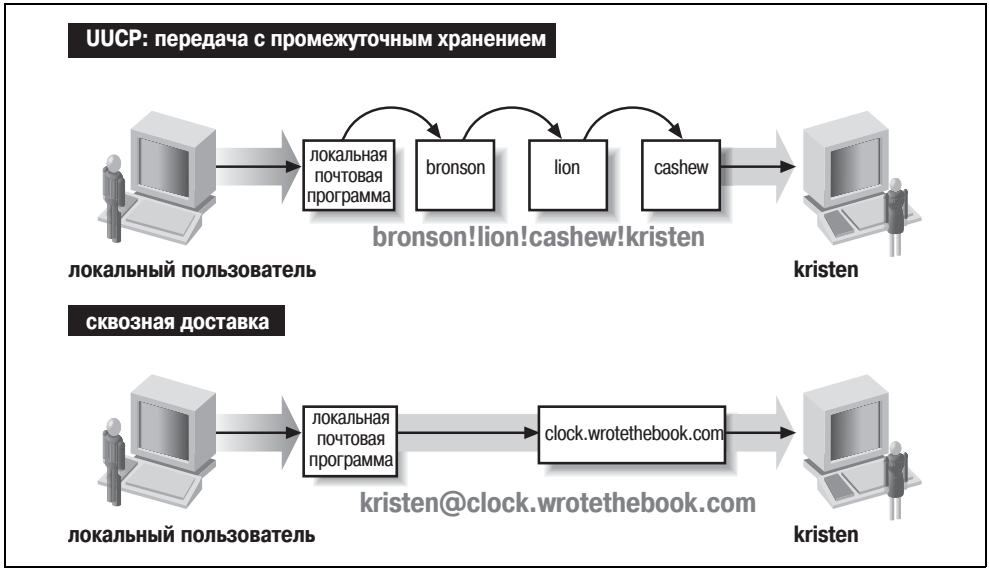


Рис. 3.3. *Системы доставки почтовых сообщений*

бенности многие персональные и мобильные компьютеры уровня ноутбуков. Такие системы часто выключаются в конце рабочего дня и могут большую часть времени не иметь подключения к сети. Отправка почты с удаленного узла дает сбой и завершается сообщением о том, что невозможно установить соединение, если локальная система выключена или не находится в сети. С целью разрешения подобных затруднений вместо прямой доставки используются возможности системы DNS для маршрутизации сообщения на почтовый сервер. Затем, когда система-клиент вновь устанавливает соединение с сетью, почта попадает к ней уже с почтового сервера. Один из протоколов, применяемых в сетях TCP/IP с этой целью, носит название протокола почтовой службы (Post Office Protocol, POP).

Протокол почтовой службы (POP)

Существует два варианта протокола почтовой службы: POP2 и POP3. POP2, определение которого дается в документе RFC 937, работает через порт 109, а POP3, определенный в документе RFC 1725, через порт 110. Несмотря на общую базовую функциональность протоколы являются несовместимыми и в работе используют различные наборы команд. Протоколы POP проверяют регистрационное имя пользователя и пароль, а затем перемещают почту пользователя с сервера в почтовый ящик локальной пользовательской программы для чтения почты. POP2 уже практически вышел из употребления, поэтому наше внимание будет сосредоточено на POP3.

Проиллюстрируем работу POP-протокола на примере сеанса POP3. POP3 – это простой протокол запросов/ответов; как и в случае SMTP, команды POP3 можно передавать напрямую через широко известный порт (110) и не-

медленно получать результаты. В данном примере ввод пользователя выделен жирным шрифтом:

```
% telnet crab 110
Trying 172.16.12.1 ...
Connected to crab.wrotethebook.com.
Escape character is '^]'.
+OK crab POP3 Server Process 3.3(1) at Mon 16-Apr-2001 4:48PM-EDT
USER hunt
+OK User name (hunt) ok. Password, please.
PASS Watts?Watt?
+OK 3 messages in folder NEWMAIL (V3.3 Rev B04)
STAT
+OK 3 459
RETR 1
+OK 146 octets
...The full text of message 1...
DELE 1
+OK message # 1 deleted
RETR 2
+OK 155 octets
...The full text of message 2...
DELE 2
+OK message # 2 deleted
RETR 3
+OK 158 octets
...The full text of message 3...
DELE 3
+OK message # 3 deleted
QUIT
+OK POP3 crab Server exiting (0 NEWMAIL messages left) Connection closed by foreign
host.
```

Команда USER позволяет указать имя пользователя, а команда PASS – пароль для учетной записи, связанной с почтовым ящиком. (Это те же самые имя и пароль, с помощью которых пользователь подключается к почтовому серверу.) В ответ на команду STAT сервер передает число сообщений в почтовом ящике и общее число байтов, занятых под хранение этих сообщений. В данном примере сервер хранит три сообщения общим объемом в 459 байт. `RETR 1` позволяет получить полный текст первого сообщения. `DELE 1` удаляет это сообщение с сервера. После извлечения с сервера каждое из сообщений удаляется. Клиент завершает сеанс командой `QUIT`. Проще некуда! В табл. 3.2 перечислены все команды POP3.

Таблица 3.2. Команды POP3

Команда	Действие
USER *username*	Указывает имя учетной записи пользователя
PASS *password*	Указывает пароль пользователя
STAT	Отображает число непрочитанных сообщений и их размер в байтах

Команда	Действие
RETR *n*	Отображает сообщение с номером *n*
DELE *n*	Удаляет сообщение с номером *n*
LAST	Отображает номер последнего сообщения, над которым выполнялись действия
LIST [*n*]	Отображает размер сообщения n либо всех сообщений
RSET	Восстанавливает все сообщения; сбрасывает номер сообщения в единицу
TOP *n l*	Выводит заголовок и *l* строк сообщения *n*
NOOP	Пустая команда; не приводит к выполнению действий
QUIT	Завершает сеанс POP3

Аргументом команды извлечения (RETR) или удаления (DELE) служит номер сообщения, что позволяет обрабатывать сообщения в произвольном порядке. Кроме того, нет прямой связи между извлечением и удалением сообщения. Можно удалить сообщение, которое вообще не было прочитано, или оставить на сервере уже прочитанное сообщение. Однако POP-клиенты обычно не пользуются подобными возможностями. В большинстве случаев содержимое почтового ящика полностью переносится с сервера на клиентскую машину, а затем удаляется с сервера либо продолжает храниться в качестве непрочитанного. Удаление отдельных сообщений на клиентской машине никак не влияет на содержимое почтового ящика на сервере, поскольку все сообщения считаются единым блоком, который либо удаляется, либо сохраняется после передачи клиенту. Клиенты электронной почты, испытывающие необходимость в удаленном управлении почтовым ящиком на сервере, скорее будут использовать протокол IMAP.

Интернет-протокол доступа к сообщениям (IMAP)

Интернет-протокол доступа к сообщениям (Internet Message Access Protocol, IMAP) является альтернативой протоколу POP. Он содержит те же базовые средства, что и POP, но обладает также функциями для синхронизации почтовых ящиков, то есть реализует чтение отдельных сообщений как на клиентской машине, так и прямо на сервере, обеспечивая при этом актуальность почтовых ящиков и той и другой системы. IMAP позволяет работать с отдельными сообщениями на клиенте и на сервере, отображая внесенные изменения в почтовых ящиках обеих систем.

Для надежной, последовательной передачи данных IMAP использует TCP. Порт протокола IMAP – TCP 143.[1] Подобно протоколу POP, IMAP работает по модели запрос/ответ и содержит небольшое число команд. Набор команд IMAP более сложен, чем применяемый в POP, поскольку IMAP обладает более

[1] В файле */etc/services* присутствует два определения портов для IMAP: 143 и 220. Порт 220 используется в IMAP 3, а 143 – в IMAP 4 и IMAP 2.

совершенной функциональностью. При этом число команд IMAP не превышает двадцати пяти. Основные команды IMAP, определенные документом RFC 2060, *Internet Message Access Protocol – Version 4rev1*, приведены в табл. 3.3.

Таблица 3.3. Команды IMAP4

Команда	Действие
CAPABILITY	Перечисляет возможности, поддерживаемые сервером
NOOP	Буквально означает «нет действия»
LOGOUT	Закрывает соединение
AUTHENTICATE	Запрашивает альтернативный метод проверки аутентичности
LOGIN	Указывает имя пользователя и пароль для аутентификации с передачей открытым текстом
SELECT	Открывает почтовый ящик
EXAMINE	Открывает почтовый ящик в режиме «только для чтения»
CREATE	Создает новый почтовый ящик
DELETE	Удаляет почтовый ящик
RENAME	Изменяет имя почтового ящика
SUBSCRIBE	Добавляет почтовый ящик в перечень активных
UNSUBSCRIBE	Удаляет почтовый ящик из перечня активных
LIST	Отображает указанные имена почтовых ящиков, выбирая из полного набора имен
LSUB	Отображает указанные имена почтовых ящиков, выбирая из набора активных
STATUS	Отображает состояние почтового ящика
APPEND	Добавляет сообщение в конец указанного почтового ящика
CHECK	Принудительное создание контрольной точки для текущего почтового ящика
CLOSE	Закрывает почтовый ящик и стирает все сообщения, отмеченные для удаления
EXPUNGE	Стирает из текущего почтового ящика все сообщения, отмеченные для удаления
SEARCH	Отображает все сообщения почтового ящика, соответствующие критерию поиска
FETCH	Извлекает сообщение из почтового ящика
STORE	Изменяет сообщение в почтовом ящике
COPY	Копирует указанные сообщения в конец указанного почтового ящика
UID	Находит сообщение по уникальному идентификатору

Приведенный набор команд четко отражает ориентированность протокола IMAP на работу с почтовыми ящиками. Протокол проектировался в качестве средства удаленного доступа к почтовым ящикам, хранимым на сервере, что видно по командам протокола. Несмотря на возросшую сложность протокола работу сервера IMAP все так же можно проверить при помощи `telnet` и небольшого числа команд.

```
$ telnet localhost 143
Trying 127.0.0.1...
Connected to rodent.wrotethebook.com.
Escape character is '^]'.
* OK rodent.wrotethebook.com IMAP4rev1 v12.252 server ready
a0001 LOGIN craig Wats?Watt?
a0001 OK LOGIN completed
a0002 SELECT inbox
* 3 EXISTS
* 0 RECENT
* OK [UIDVALIDITY 965125671] UID validity status
* OK [UIDNEXT 5] Predicted next UID
* FLAGS (\Answered \Flagged \Deleted \Draft \Seen)
* OK [PERMANENTFLAGS (\* \Answered \Flagged \Deleted \Draft \Seen)] Permanent flags
* OK [UNSEEN 1] first unseen message in /var/spool/mail/craig
a0002 OK [READ-WRITE] SELECT completed
a0003 FETCH 1 BODY[TEXT]
* 1 FETCH (BODY[TEXT] {1440}
... an e-mail message that is 1440 bytes long ...
* 1 FETCH (FLAGS (\Seen))
a0003 OK FETCH completed
a0004 STORE 1 +FLAGS \DELETED
* 1 FETCH (FLAGS (\Seen \Deleted))
a0004 OK STORE completed
a0005 CLOSE
a0005 OK CLOSE completed
a0006 LOGOUT
* BYE rodent.wrotethebook.com IMAP4rev1 server terminating connection
a0006 OK LOGOUT completed
Connection closed by foreign host.
```

Первые три и последняя строки исходят от программы `telnet`; все прочие сообщения принадлежат IMAP. Первая введенная пользователем команда IMAP – это LOGIN; она указывает имя пользователя и пароль из файла */etc/passwd* для его аутентификации. Обратите внимание, команду предваряет строка `A0001`. Это *тег*, представляющий собой уникальный идентификатор, создаваемый клиентом для каждой команды. Каждая команда должна начинаться с тега. При ручном вводе команд вы сами являетесь источником тегов.

Протокол IMAP ориентирован на работу с почтовыми ящиками. Команда SELECT выбирает почтовый ящик, с которым происходит работа. В данном примере пользователь выбрал почтовый ящик с именем «inbox». Сервер IMAP отображает состояние ящика. Можно видеть, что ящик содержит три

сообщения, с каждым из которых связан ряд флагов. Флаги позволяют управлять сообщениями в почтовом ящике, помечая их в качестве прочитанных (Seen), непрочитанных (Unseen), удаленных (Deleted) и т. д.

Команда FETCH позволяет получить сообщение из почтового ящика. В данном примере пользователь принимает текст сообщения, то есть его содержательную часть. Однако можно загрузить только заголовки сообщений или флаги.

Приняв сообщение, пользователь удаляет его, записывая флаг Deleted при помощи команды STORE. Команда DELETE используется не для удаления сообщений, а для удаления почтовых ящиков. Отдельные сообщения отмечаются для удаления установкой флага Delete. Сообщения с флагом Delete не удаляются, пока не будет выполнена команда EXPUNGE либо почтовый ящик не будет закрыт явным образом посредством команды CLOSE, как в приведенном примере. Сеанс завершается командой LOGOUT.

Очевидно, протокол IMAP сложнее протокола POP и вплотную подошел к той границе, за которой набор команд вручную становится неэффективным. Разумеется, на практике редко применяется ручной набор. Рабочие станции и серверы обмениваются командами автоматически. Приведенный пример призван лишь проиллюстрировать работу протокола IMAP. Скорее всего, единственной проверкой IMAP, выполняемой вручную, для вас станет проверка факта работоспособности демона `imapd`. Чтобы выполнить такую проверку, нет необходимости даже регистрироваться на сервере; достаточно увидеть, что сервер реагирует на соединение, установленное при помощи `telnet`. Затем остается лишь набрать команду LOGOUT, чтобы мягко закрыть соединение.

Многоцелевые расширения почтовой службы (MIME)

Последний почтовый протокол, на котором мы остановимся в этом кратком обзоре, – это протокол MIME (*Multipurpose Internet Mail Extensions*).[1] Как следует из названия, MIME является расширением существующей почтовой системы TCP/IP, а не заменой для нее. MIME больше заботит характер данных, передаваемых почтовой системой, а не механизмы доставки. Это не попытка заменить SMTP или TCP, но способ расширить определение «почтового сообщения».

Структура почтовых сообщений, передаваемых по SMTP, определена в документе RFC 822, *Standard for the Format of ARPA Internet Text Messages* (Стандарт формата текстовых сообщений сети ARPA Internet). RFC 822 содержит ряд определений для заголовков почтовых сообщений, которые получили столь широкое распространение, что применяются во многих почтовых системах, никак не связанных с протоколом SMTP. Такое положение очень выгодно для системы электронной почты, поскольку создает определенное взаимопонимание систем в вопросах перевода сообщений и доставки

[1] MIME также является составной частью среды Web и протокола HTTP.

через шлюзы в различные почтовые сети. MIME дополняет RFC 822 в двух аспектах, не затронутых исходным документом:

- Поддержка различных типов данных. Почтовая система, определяемая документами RFC 821 и RFC 822, позволяет передавать только 7-битные ASCII-данные. Этого вполне достаточно для пересылки текстовых данных, состоящих из символов ASCII, но не достаточно для пересылки сообщений на языках с другим алфавитом, а также двоичных данных.
- Поддержка сложного наполнения сообщений. В RFC 822 нет подробного описания тела электронных сообщений, хотя довольно подробно описан состав заголовков сообщений.

С целью разрешения подобных затруднений MIME определяет методы кодирования, позволяющие передавать различные виды данных, и структуру тела сообщения, позволяющую передавать в одном сообщении несколько объектов. Документ RFC 1521 *Multipurpose Internet Mail Extensions Part One: Format of Internet Message Bodies* содержит два определения заголовков, структурирующих тело почтового сообщения и позволяющих передавать различные виды данных. Речь идет о заголовках *Content-Type* и *Content-Transfer-Encoding*.

Как следует из названия, заголовок *Content-Type* определяет тип данных, передаваемых в сообщении. Поле заголовка Subtype уточняет определение. Многие подтипы появились уже после публикации исходного документа RFC. Список типов MIME, имеющих хождение, можно найти в сети Интернет.[1] Исходный документ RFC определяет семь базовых типов содержимого и ряд подтипов:

text

Текстовые данные. В RFC 1521 определены текстовые подтипы *plain* и *richtext*. Помимо определенных в стандарте, появилось еще более 30 подтипов, включая *enriched*, *xml* и *html*.

application

Двоичные данные. Основной подтип, определенный в RFC 1521, – это *octet-stream*, указывающий на поток 8-битных байтов двоичных данных. Второй определенный в стандарте подтип – *PostScript*. Помимо определенных в стандарте, появилось еще более 200 подтипов. Они отмечают двоичные данные в формате конкретных приложений. Так, например, существует application-подтип *msword*.

image

Статичные изображения в графическом формате. RFC 1521 определяет два подтипа: *jpeg* и *gif*. Помимо определенных, появилось еще более 20 дополнительных подтипов, отражающих широко распространенные стандарты хранения изображений, такие как *tiff*, *cgm* и *g3fax*.

---

[1] Медиа-типы файлов доступны по адресу *ftp://ftp.isi.edu/in-notes/iana/assignments/media-types*.

video

Анимированные изображения. Изначально был определен подтип *mpeg*, обозначающий распространенный стандарт хранения видеоданных на компьютере. Помимо этого появилось еще несколько подтипов, включая *quicktime*.

audio

Звуковые данные. Изначально был определен только один подтип – *basic*, обозначавший кодирование звука в формате PCM (pulse code modulation, импульсно-кодовая модуляция сигнала). В настоящее время существует еще около 20 дополнительных audio-типов, таких как *MP4A-LATM*.

multipart

Данные, состоящие из нескольких независимых разделов. Тело multipart-сообщения состоит из нескольких независимых частей. RFC 1521 определяет четыре подтипа. Основной подтип, *mixed*, означает, что каждая часть сообщения может представлять данные произвольного типа содержимого. Прочие подтипы: *alternative*, обозначающий повторение данных в различных форматах; *parallel*, означающий, что данные из различных частей должны просматриваться единовременно; *digest*, означающий, что данные разделов имеют тип *message*. В настоящее время уже существует ряд дополнительных подтипов, реализующих, в частности, поддержку голосовых сообщений (*voice-message*) и зашифрованных сообщений.

message

Данные, инкапсулирующие почтовое сообщение. RFC 1521 определяет три подтипа. Основной подтип, *rfc822*, указывает на данные, представляющие полное сообщение RFC 822. Подтипы *partial* и *External-body* предназначены для обработки больших сообщений. *partial* позволяет при инкапсуляции разбивать большие сообщения на ряд MIME-сообщений. *External-body* указывает на внешний источник содержимого большого сообщения, что позволяет передавать в MIME-сообщении только указатель, а не само сообщение. Два дополнительных подтипа, *news* и *http*, позволяют передавать соответственно сетевые новости и HTTP-трафик, отформатированный по требованиям типизации содержимого MIME.

Заголовок *Content-Transfer-Encoding* определяет тип кодирования данных. Консервативные SMTP-системы передают только 7-битные ASCII-данные с длиной строки не более 1000 байт. Поскольку данные, исходящие от MIME-системы, могут проходить через шлюзы, передающие только 7-битные ASCII-символы, возникает необходимость подвергать данные кодированию. RFC 1521 определяет шесть типов кодирования. Некоторые из них позволяют указывать кодировку, присущую данным. Лишь два типа связаны с конкретными методами кодирования, определенными в RFC. Шесть типов кодирования следующие:

7bit

Данные в формате ASCII. Семибитные ASCII-данные не подвергаются кодированию.

8bit

Восьмибитные данные. Кодированию не подвергаются. Данные двоичные, но строки данных достаточно короткие для SMTP-транспорта, то есть не превышают в длину 1000 байт.

binary

Двоичные данные. Кодированию не подвергаются. Данные двоичные, а длина строки может превышать 1000 байт. Между данными типа *binary* и *8bit* нет разницы, кроме ограничения на длину строки; оба типа данных представляют незакодированные потоки 8-битных байтов. MIME не изменяет незакодированные потоковые данные.

quoted-printable

Закодированные текстовые данные. Этот метод кодирования применяется для данных, состоящих преимущественно из отображаемых ASCII-символов. Текст в формате ASCII передается в незакодированном виде, а байты со значениями больше 127 или меньше 33 передаются закодированными в виде строк, состоящих из последовательностей символов. Каждая последовательность состоит из знака равенства и шестнадцатеричного значения байта. Например, ASCII-символ прогона страницы, имеющий шестнадцатеричное значение *0C*, передается в виде *=0C*. Естественно, есть и другие тонкости – так, знак равенства приходится передавать в виде *=3D*, а символ новой строки в конце строки не кодируется. Тем не менее описанный метод дает представление о передаче данных в кодировке *quoted-printable*.

base64

Закодированные двоичные данные. Данный метод кодирования может применяться для любого байтового потока данных. Три октета (8-битных байта) данных кодируются в виде четырех 6-битных символов, что увеличивает размер файла на треть. 6-битные символы являются подмножеством ASCII и выбранны так, чтобы проходить через почтовую систему произвольного типа. Максимальная длина строки для данных в формате *base64* – 76 символов. Рис. 3.4 иллюстрирует кодирование символов 3-в-4.

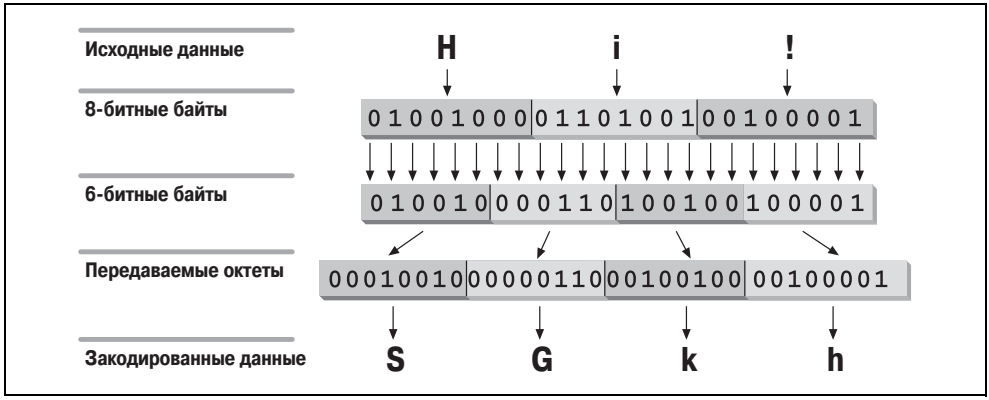


Рис. 3.4. *Кодирование base64*

x-token

Данные в специальной кодировке. Разработчики программного обеспечения могут создавать собственные методы кодирования. Имена таких методов должны начинаться символами *X-*. Однако это не приветствуется, поскольку снижает возможности взаимодействия почтовых систем.

Число применяемых типов данных и методов кодирования растет с появлением новых форматов данных, передаваемых в сообщениях. В новых документах RFC постоянно определяются дополнительные типы данных и кодировки. В поисках самой свежей информации по возможностям MIME обращайтесь к последним версиям документов RFC.

MIME определяет типы данных, перенос которых не планировался при разработке SMTP. Чтобы справиться с подобными и другими новшествами, был определен метод расширения SMTP – в документе RFC 1869, *SMTP Service Extensions*. RFC 1869 не определяет новых служб для SMTP; более того, в этом документе упоминаются лишь те расширения служб, что определены в других RFC. RFC 1869 определяет простой механизм, позволяющий системам SMTP согласовывать использование расширений. В документе определена новая команда *приветствия* (EHLO) и приемлемые ответы на эту команду. В частности, система, получившая такую команду, может вернуть список расширений, которые поддерживает. Такой ответ позволяет системе-источнику понять, какими из расширенных служб можно воспользоваться, и избежать использования тех, которые не реализованы удаленной системой. Реализации SMTP, понимающие команду EHLO, известны в качестве систем ESMTP (Extended SMTP).

Для почтовых программ MIME определен ряд дополнительных служб ESMTP. Отдельные расширения перечислены в табл. 3.4. Первая колонка таблицы содержит ключевые слова EHLO, связанные с каждым из расширений, вторая – номер руководящего документа RFC, третья – назначение расширения. Перечисленные расширения служб – просто примеры. Полная картина включает и другие усовершенствованные аспекты SMTP.

Таблица 3.4. Расширения служб SMTP

Ключевое слово	RFC	Функциональность
8BITMIME	1652	Принимает 8-битные двоичные данные
CHUNKING	1830	Принимает сообщения, разбитые на фрагменты
CHECKPOINT	1845	Перезапуск почтовых транзакций по контрольным точкам
PIPELINING	1854	Принимает наборы команд, объединенные в блоки
SIZE	1870	Отображает максимально-допустимый размер сообщения
DSN	1891	Обеспечивает передачу уведомлений о состоянии доставки
ETRN	1985	Принимает внешние запросы на обработку очереди
ENHANCEDSTATUSCODES	2034	Расширенные коды завершения операций

Ключевое слово	RFC	Функциональность
STARTTLS	2487	Использует механизм Transport Layer Security для шифрования почтового обмена
AUTH	2554	Усиленный механизм аутентификации для определения источника почтовых сообщений

Выяснить, какие расширения поддерживает сервер, можно при помощи команды EHLO. Следующий пример относится к стандартной системе Solaris 8, в состав которой входит sendmail 8.9.3:

```
> telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 crab.wrotethebook.com ESMTP Sendmail 8.9.3+Sun/8.9.3; Mon, 23 Apr 2001
11:00:35-0400 (EDT)
EHLO crab
250-crab.wrotethebook.com Hello localhost [127.0.0.1], pleased to meet you
250-EXPN
250 HELP
250-8BITMIME
250-SIZE
250-DSN
250-ETRN
250-VERB
250-ONEX
250-XUSR
QUIT
221 crab.foobirds.org closing connection
Connection closed by foreign host.
```

Система из примера в ответ на приветствие EHLO перечисляет девять команд. Две из них, EXPN и HELP, являются стандартными командами SMTP, реализованными не во всех системах (стандартные команды перечислены в табл. 3.1). 8BITMIME, SIZE, DSN и ETRN – расширения ESMTP, описанные в табл. 3.4. Еще три ключевых слова из ответа – VERB, ONEX и XUSR. Эти команды специфичны для sendmail версии 8. Их определений нет в документах RFC. VERB предписывает серверу sendmail работать в режиме подробной диагностики. ONEX ограничивает сеанс транзакциями для отдельных сообщений. XUSR – это эквивалент ключа `-U` командной строки sendmail.[1] Как видно по трем последним ключевым словам, документы RFC допускают применение частных расширений ESMTP.

Конкретный набор расширений зависит от системы. Например, в случае стандартной системы Solaris 2.5.1 в ответ на команду EHLO отображается

[1] Перечень ключей и аргументов командной строки sendmail приводится в приложении E.

лишь три ключевых слова (EXPN, SIZE и HELP). То, какие расширения доступны, зависит от применяемой версии, равно как и настройки sendmail.[1] Назначение команды EHLO – выявить конкретику расширений до начала почтового обмена SMTP.

ESMTP и MIME – важные механизмы, позволяющие передавать посредством электронной почты данные, отличные от ASCII-текста. Пользователи привыкли обмениваться данными в форматах, специфичных для конкретных приложений, и для многих людей электронная почта является основным механизмом передачи таких файлов.

Протоколы SMTP, POP, IMAP и расширения MIME – это жизненно важные составляющие почтовой системы, но в будущем их место могут занять иные протоколы электронной почты. Можно с уверенностью говорить лишь о том, что сеть постоянно меняется. Необходимо следить за изменениями и не забывать о полезных технологиях, планируя работу сети. В частности, две технологии, которые пользователи находят полезными, – совместный доступ к файлам и принтерам. В следующем разделе мы изучим серверы файлов и печати.

Серверы файлов и печати

Службы файлов и печати делают сетевую среду более комфортной для пользователей. Еще недавно дисковые накопители и высококачественные принтеры стоили относительно дорого, а бездисковые рабочие станции встречались на каждом шагу. Сегодня в каждую систему устанавливается вместительный жесткий диск, а многие системы оснащены высококачественными лазерными принтерами, однако потребность в службах совместного доступа к ресурсам продолжает возрастать.

Совместный доступ к файлам

Совместный доступ к файлам – совсем не то же самое, что передача файлов; это не просто способность систем обмениваться файлами. Корректно построенная система совместного доступа к файлам не требует копирования файлов по сети. Она организует доступ к файлам на уровне отдельных записей, позволяя клиенту прочитать запись из файла, хранимого на удаленном сервере, изменить эту запись и снова записать ее в файл – не копируя файл целиком с сервера на систему клиента и обратно.

Совместный доступ к файлам прозрачен для пользователя и прикладных программ, работающих в системе пользователя. Совместный доступ позволяет пользователям и программам обращаться к файлам так, как если бы они хранились локально. В идеальной среде совместного доступа к файлам пользователь вообще не знает и не стремится узнать, где на самом деле хранятся файлы.

---

[1] Подробности настройки sendmail приводятся в главе 10.

Организация совместного доступа к файлам не существовала в изначальном варианте семейства протоколов TCP/IP. Она появилась для поддержки бездисковых рабочих станций. Существует целый ряд протоколов TCP/IP для обеспечения совместного доступа к файлам, двум из которых досталась львиная доля этого рынка:

NetBIOS / блок серверных сообщений (NetBIOS/Server Message Block)

Авторство системы NetBIOS принадлежит IBM. Это базовая сетевая система ввода-вывода используется в ОС Microsoft Windows. Системы Unix способны действовать в качестве файловых серверов и серверов печати для клиентов Windows – при помощи программного пакета Samba, который реализует протоколы NetBIOS и SMB (Server Message Block, блок серверных сообщений).

Сетевая файловая система (Network File System)

Система NFS создана компанией Sun Microsystems в целях поддержки производимых ею бездисковых рабочих станций. NFS спроектирована для использования преимущественно в приложениях для локальной сети, а реализации NFS существуют для всех ОС Unix и многих других операционных систем.

Для организации совместного доступа к файлам Unix-систем, как правило, используется NFS – наиболее широко применяемый Unix-протокол подобного назначения. Поддержка Windows-клиентов серверами, работающими под Unix, обычно осуществляется при помощи Samba. Подробный рассказ об этих инструментах содержится в главе 9.

Службы печати

Сервер печати позволяет организовать доступ к принтерам для всех систем сети. Совместный доступ к принтерам не столь важен, как совместный доступ к файлам, но при этом остается полезной сетевой службой. Каковы же преимущества совместного доступа к печатающим устройствам?

- Появляется возможность сократить число принтеров, объемы затрат на приобретение собственно устройств и расходных материалов.
- Сокращение затрат на поддержку. Становится меньше машин и, соответственно, сокращается штат обслуживающего персонала.
- Доступ к особым принтерам. Устройства высококачественной цветной печати или высокоскоростной печати очень дороги, а используются не так часто. Организация совместного доступа к таким принтерам позволяет оптимизировать потребление дорогостоящих ресурсов.

В корпоративных сетях распространены два метода организации совместного доступа к печатающим устройствам. Первый связан с применением служб совместного доступа Samba и является предпочтительным для клиентов Windows. Второй подход – использовать традиционную команду Unix `lpr` и демон печати `lpd`. Настройка сервера печати также описана в главе 9.

Эта глава завершается рассказом о различных типах серверов настройки TCP/IP. Серверы настройки, в отличие от электронной почты, совместного доступа к файлам и серверов печати, находят применение далеко не в каждой сети. Однако потребность в упрощенной установке и повышенной мобильности делает серверы настройки важными компонентами многих сетей.

Серверы настройки

Мощные возможности, повышающие практичность и гибкость TCP/IP, повышают одновременно и его сложность. Система TCP/IP не столь проста в настройке, как некоторые другие сетевые системы. TCP/IP требует наличия информации по аппаратной части, по адресации и маршрутизации – в процессе настройки. Система работает независимо от сетевого оборудования, так что невозможно зашить в оборудование информацию настройки, как это делается в других системах. Информация должна исходить от человека, ответственного за настройку. То есть предполагается, что каждая система находится в ведении людей, достаточно осведомленных, чтобы при настройке системы использовать корректную информацию. К сожалению, такое предположение не всегда оправдано.

Серверы настройки позволяют сетевому администратору управлять настройкой TCP/IP централизованно. Это частично освобождает конечного пользователя от тягот настройки, а качество информации, используемой в настройке, повышается.

В TCP/IP существует три протокола, упрощающих задачу настройки: RARP, BOOTP и DHCP. Прежде всего, обратимся к RARP, самому старому и самому простому из инструментов настройки.

Протокол обратного разрешения адресов (RARP)

Протокол RARP (Reverse Address Resolution Protocol), определенный документом RFC 903, позволяет выполнять преобразование физического сетевого адреса в адрес IP, то есть его функция обратна функции протокола разрешения адресов (Address Resolution Protocol, ARP). Сервер RARP преобразует физический адрес в IP-адрес для клиента, которому не известен собственный IP-адрес. Клиент посылает широковещательное сообщение, пользуясь соответствующими службами физической сети.[1] Широковещательный пакет содержит адрес клиента в физической сети и запрашивает у любой компетентной системы сети IP-адрес, связанный с данным физическим адресом. Сервер RARP отвечает пакетом, содержащим IP-адрес клиента.

Клиент знает свой адрес в физической сети, поскольку этот адрес зашит в интерфейсное устройство Ethernet. В большинстве систем есть команда, с по-

[1] Подобно ARP, RARP является протоколом уровня доступа к сети, и использует службы физической сети, расположенные под уровнем Internet. Уровни протоколов TCP/IP описаны в главе 1.

мощью которой это значение можно с легкостью получить. Например, в системе Solaris 8 суперпользователь может набрать такую команду:

```
# ifconfig dnet0
dnet0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
        inet 172.16.12.1 netmask ffffff00 broadcast 172.16.12.255
        ether 0:0:c0:dd:d4:da
```

Команда `ifconfig` позволяет устанавливать и отображать значения настройки сетевых интерфейсов.[1] *dnet0* – это имя устройства интерфейса Ethernet. Адрес Ethernet отображается после метки *ether*. В приведенном примере это адрес 0:0:c0:dd:d4:da.

Сервер RARP ищет IP-адрес для ответа клиенту в файле */etc/ethers*. Файл */etc/ethers* содержит строки, состоящие из адресов Ethernet и имен узлов клиентов. Например:

```
2:60:8c:48:84:49        clock
0:0:c0:a1:5e:10         ring
0:80:c7:aa:a8:04        24seven
8:0:5a:1d:c0:7e         limulus
8:0:69:4:6:31           arthropod
```

Чтобы ответить на RARP-запрос, сервер должен также выполнить разрешение имени узла из файла */etc/ethers* в адрес IP. С этой целью используется DNS или файл таблицы узлов. Вот такие записи файла *hosts* узлов могут соответствовать приведенному выше файлу *ethers*:

```
clock           172.16.3.10
ring            172.16.3.16
24seven         172.16.3.4
limulus         172.16.3.7
arthropod       172.16.3.21
```

Имея эти файлы, сервер, получив RARP-запрос с адресом Ethernet 0:80:c7:aa:a8:04, сопоставляет адрес с именем *24seven* из файла */etc/ethers*. Затем сервер ищет IP-адрес для имени *24seven* и возвращает ответ ARP – адрес IP 172.16.3.4.

RARP – полезный инструмент, но позволяет получать только адрес IP. Остается еще целый набор значений, которые приходится указывать вручную. Есть другое, более гибкое средство настройки, – протокол инициализации BOOTP (Bootstrap Protocol), предоставляющий дополнительные значения, помимо адреса IP, и позволяющий получать эти значения по сети.

Определение BOOTP дается документами RFC 951 и RFC 1532. Протокол BOOTP описан в качестве альтернативы RARP: если используется BOOTP, отпадает необходимость в применении RARP. При этом BOOTP – в качестве протокола настройки – имеет ряд преимуществ перед RARP. Он предостав-

[1] Команда `ifconfig` описана в главе 6.

ляет гораздо больший объем информации и потенциально позволяет этот объем наращивать. В исходную спецификацию был заложен механизм развития протокола в виде возможности создания разработчиками расширений. Определение подобных расширений впервые было формализовано в RFC 1048, с ходом времени обновлялось, и на сегодняшний день в наиболее полном виде содержится в RFC 2132. Протокол BOOTP и его расширения стали фундаментом протокола динамической настройки узлов DHCP (Dynamic Host Configuration Protocol). Протокол DHCP пришел на смену BOOTP, так что именно этим протоколом вы будете пользоваться в своих сетях.

Протокол динамической настройки узлов (DHCP)

Протокол динамической настройки узлов (Dynamic Host Configuration Protocol, DHCP) определен в документах RFC 2131 и RFC 2132. Архитектура DHCP делает его совместимым с BOOTP. Взаимодействие между клиентами BOOTP и серверами DHCP, а также клиентами DHCP и серверами BOOTP описано в документе RFC 1534. DHCP – это верный выбор, если говорить о протоколе настройки для сети, поскольку превосходит BOOTP по своим возможностям, сохраняя при этом поддержку существующих BOOTP-клиентов.

В DHCP применяются те же номера UDP-портов, что и в BOOTP (67 и 68), и тот же базовый формат пакетов. При этом протокол DHCP является не просто обновлением BOOTP. Он расширяет функциональность BOOTP в двух аспектах:

- Параметры настройки, распространяемые сервером DHCP, включают все значения, перечисленные в документе RFC *Requirements for Internet Hosts* (Требования для узлов сетей Интернет). DHCP обеспечивает клиента полным набором значений настройки TCP/IP.
- DHCP позволяет производить автоматическое выделение IP-адресов.

Изначальный пакет BOOTP в DHCP расширен и включает указание типа пакета DHCP, а также полный набор данных настройки. Значения этой части пакета в DHCP называются *параметрами* (*options*). В целях охвата всего набора параметров настройки из RFC *Requirements for Internet Hosts* поле Options имеет довольно большой размер и переменный формат.

Как правило, не возникает потребности в полном наборе значений настройки. Но речь не о том, что без каких-то значений можно обойтись – все параметры участвуют в процессе настройки TCP/IP. Просто не для каждого параметра требуется задавать *определенное* значение. В большинстве реализаций TCP/IP существуют значения по умолчанию, которые следует изменять лишь в особых случаях. Расширенные параметры настройки DHCP делают этот протокол более совершенным, чем BOOTP, но не этот аспект оказывается самым привлекательным среди нововведений в DHCP.

Большинство сетевых администраторов гораздо больше интересует автоматическое выделение адресов IP. DHCP позволяет выполнять назначение адресов четырьмя путями:

Фиксированные постоянные адреса

Как и ранее, администратор может назначать адреса без помощи системы DHCP. При том, что подобное назначение происходит вне ведома DHCP, последний учитывает это, давая возможность исключать адреса из диапазонов тех, что находятся под управлением DHCP. Постоянные адреса встречаются в большинстве сетей.

Выделение вручную

Сетевой администратор сохраняет полную власть над адресами, назначая конкретные адреса конкретным клиентам при настройке DHCP. Точно так же управление адресами происходит в BOOTP. Выделение вручную не позволяет полностью задействовать потенциал DHCP, но может потребоваться для обслуживания клиентов BOOTP.

Автоматическое выделение

Сервер DHCP назначает постоянный адрес, выделяя его из резерва адресов. Администратор не участвует в процессе назначения адреса клиенту. Этот способ не позволяет применить способность сервера DHCP собирать и повторно использовать адреса.

Динамическое выделение

Сервер назначает адрес клиенту DHCP на ограниченный период времени. Выделение адреса с ограниченным временем жизни называется *арендой* (*lease*). Клиент может вернуть адрес серверу в любой момент, но обязан запросить продление аренды у сервера, чтобы сохранить его дольше предписанного времени жизни. Если клиент не запросил продление, сервер автоматически отбирает адрес. Динамическое выделение адресов – наиболее эффективное из применений DHCP.

Динамическое выделение полезно в любой сети, особенно в крупной, распределенной сети с флуктуацией (добавлением и удалением) узлов. Невостребованные адреса возвращаются серверу – без участия пользователей или системных администраторов. Адреса используются только по необходимости. Динамическое выделение позволяет максимально эффективно использовать ограниченный набор адресов и особенно удобно для мобильных систем, которые мигрируют из одной подсети в другую и постоянно нуждаются в получении новых адресов, соответствующих текущей сетевой дислокации. Даже в самых маленьких сетях динамическое выделение облегчает труд сетевого администратора.

Динамическое выделение адресов действует не для всех систем. Серверы имен, почтовые серверы, узлы регистрации и прочие системы совместного доступа постоянно находятся в сети и не являются мобильными. К ним обращаются по именам, а преобразование доменного имени должно приводить к получению верного адреса. Таким системам вручную выделяются постоянные фиксированные адреса.

Динамическое назначение адресов оказывает существенное влияние на систему DNS. Смысл DNS – в отображении имен узлов в адреса IP. Если IP-ад-

реса постоянно изменяются, а система доменных имен об этом не узнает, она не сможет выполнять свою работу. Чтобы динамическое выделение заработало для всех типов систем, необходима служба доменных имен, динамически обновляемая сервером DHCP. Существует система *Dynamic DNS* (DDNS), но пока еще она не получила широкого распространения.[1] В должное время эта система сделает динамические адреса доступными не только для клиентов, но и для систем, предоставляющих доступ к сетевым службам.

Вследствие природы динамической адресации, в большинстве локальных сетей совместно используемым серверам назначаются постоянные адреса. Это делается путем обычного администрирования, и DHCP в этом процессе не участвует. По сути дела, администратор такого сервера получает адрес и использует этот адрес при настройке. Использование DHCP для настройки отдельных систем не означает, что система должна применяться для настройки всех систем без исключения.

Серверы DHCP могут осуществлять поддержку клиентов BOOTP. Но чтобы воспользоваться всеми премуществами DHCP, необходим DHCP-клиент. Клиенты BOOTP не умеют работать с динамической арендой адресов. Они не знают, что существует срок годности адреса и что аренда адреса подлежит продлению. Клиентам BOOTP должны назначаться постоянные фиксированные адреса – вручную либо автоматически. Настоящее динамическое назначение адресов – привилегия клиентов DHCP.

Итак, для большинства локальных сетей, в которых применяется DHCP, верны следующие пункты:

- Постоянные адреса назначаются системам, которые не могут работать с DHCP
- Клиентам BOOTP адреса назначаются вручную
- Всем клиентам DHCP адреса назначаются динамически

В результате встает вопрос: каким образом клиент, не знающий собственного адреса, способен общаться с сервером? DHCP регламентирует простой обмен пакетами, позволяющий клиенту обнаружить сервер и получить настройки.

Принципы работы DHCP

Клиент DHCP посылает широковещательный пакет (сообщение *DHCPDISCOVER*), содержащий как минимум идентификатор транзакции и DHCP-идентификатор клиента – обычно его адрес в физической сети. В качестве адреса отравки широковещательного пакета выступает *ограниченный широковещательный адрес* 255.255.255.255.[2] Клиент ожидает ответ сервера. Если ответ не получен в течение заданного времени, клиент повторяет запрос.

---

[1] Дополнительная информация о DDNS содержится в главе 8.

[2] Данный адрес полезен по той причине, что, в отличие от стандартного широковещательного адреса, не требует от системы-источника знаний о том, в какой сети она расположена.

DHCP использует в качестве транспортного протокола UDP, так что, в отличие от RARP, не требует применения специальных протоколов уровня доступа к сети.

Сервер отвечает на сообщение клиента пакетом *DHCPOFFER*. В работе DHCP использует два порта с широко известными номерами. Порт UDP 67 используется сервером, а порт UDP 68 – клиентом. Такое положение дел необычно. В случае большинства приложений на стороне сервера используется широко известный номер порта, а на стороне клиента – случайный номер. (Причины и способы использования случайных номеров исходных портов описаны в главе 1.) Случайный номер порта гарантирует, что каждая пара исходного/целевого портов недвусмысленно определяет путь для обмена информацией. Однако клиент DHCP находится в процессе загрузки и, вероятнее всего, не знает своего IP-адреса. Даже если клиент сгенерирует исходный порт для пакета *DHCPDISCOVER*, ответ сервера, адресованный на этот порт и IP-адрес клиента, не будет прочитан клиентом, не опознавшим адрес. Поэтому DHCP посылает ответ на конкретный порт для всех узлов. Широковещательное сообщение, направленное на порт UDP 68, будет прочитано всеми узлами, даже системами, не знающими своего адреса. Каждая система определяет, является ли пунктом назначения сообщения, проверяя идентификатор транзакции и физический адрес, содержащиеся в пакете ответа.

Сервер заполняет пакет *DHCPOFFER* данными настройки для клиента. Корректно настроенный сервер DHCP способен предоставить клиенту любое требуемое значение настройки TCP/IP. Глава 9 является руководством по установке сервера DHCP, а приложение D содержит полный перечень параметров настройки DHCP.

Как следует из названия, пакет *DHCPOFFER* – это предложение данных настройки. Предложение действительно на ограниченный период времени – обычно 120 секунд. Клиент должен ответить на предложение, прежде чем истечет установленный временной интервал. Дело в том, что получить пакет *DHCPDISCOVER* от клиента и ответить пакетом *DHCPOFFER* могут сразу несколько серверов. Если бы сервер не требовал ответа от клиента, могла возникнуть ситуация, когда несколько серверов выделяли ресурсы одному клиенту, в то время как эти потраченные впустую ресурсы могли бы использоваться другими клиентами. Получив несколько пакетов *DHCPOFFER*, клиент реагирует лишь на один из них.

На пакет *DHCPOFFER* клиент отвечает сообщением *DHCPREQUEST*. Сообщение *DHCPREQUEST* говорит серверу, что клиент согласен на использование предложенных настроек. Сервер проверяет информацию в *DHCPREQUEST*, чтобы убедиться, что клиент все понял правильно, а предложенные параметры по-прежнему доступны. Если все верно, сервер передает клиенту сообщение *DHCPACK*, уведомляя, что теперь его настройки позволяют использовать все данные исходного пакета *DHCPOFFER*. Нормальный обмен пакетами при настройке клиента посредством DHCP показан на рис. 3.5.

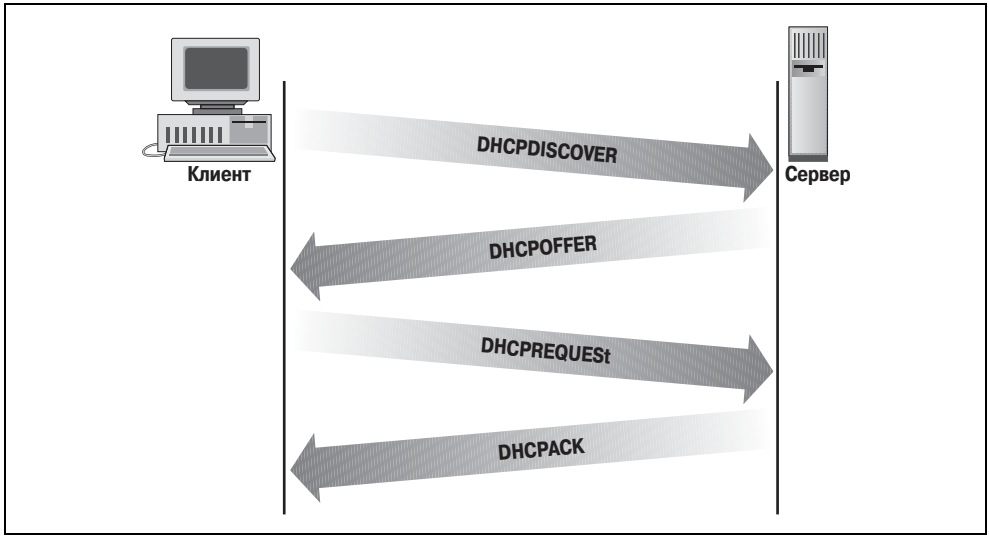


Рис. 3.5. *Клиент-серверный протокол DHCP*

Резюме

TCP/IP содержит отдельные сетевые службы, упрощающие создание, настройку и работу с сетями. Одной из таких служб является служба имен, применяемая во всех сетях TCP/IP.

Служба имен может быть построена на базе таблиц узлов, системы доменных имен DNS (Domain Name System) либо сетевой информационной системы NIS (Network Information Service). Таблица узлов – это простой набор записей, хранимых в текстовом файле */etc/hosts*. В большинстве систем существует небольшая таблица узлов, но она может применяться далеко не во всех случаях, поскольку плохо масштабируется и не имеет стандартизированного метода автоматического распространения. NIS, сервер «желтых страниц» от Sun, решает проблему автоматического распространения для таблицы узлов, но не решает проблему масштабирования. Система DNS, пришедшая на смену таблицам узлов в качестве стандарта TCP/IP, замечательно масштабируется. DNS – это иерархическая система распределенной базы данных, обеспечивающая доступ к информации по именам и адресам для всех систем сети Интернет.

Структурными элементами системы электронной почты в TCP/IP являются протоколы SMTP (Simple Mail Transfer Protocol, простой протокол передачи почты), POP (Post Office Protocol, протокол почтовой службы), IMAP (Internet Message Access Protocol, Интернет-протокол доступа к сообщениям), а также MIME (Multipurpose Internet Mail Extensions, многоцелевые расширения почтовой службы в Интернете). Простой протокол SMTP работает по модели запрос/ответ и обеспечивает сквозную доставку почтовых сообще-

ний. Иногда сквозная доставка не является оптимальным решением, и почта передается по определенному маршруту почтовому серверу. Для переноса сообщений с сервера на систему пользователя в TCP/IP применяются протоколы POP и IMAP. SMTP позволяет пересылать только 7-битные ASCII-данные. MIME расширяет почтовую систему TCP/IP, делая возможной передачу самых разнообразных видов данных.

Сетевая файловая система NFS (Network File System) – основной протокол организации совместного доступа к файлам для Unix-систем. Он позволяет серверным системам экспортировать каталоги, которые затем монтируются клиентами и используются наравне с локальными дисковыми накопителями. Для организации совместного доступа к принтерам в сети TCP/IP может применяться протокол Unix LPD/LPR. Система Samba предоставляет сходные службы совместного доступа к файлам и печатающим устройствам для клиентов Windows.

Установка TCP/IP требует присутствия многих значений настройки. Эти значения могут предоставляться сервером настройки. Для распределения настроечной информации в TCP/IP используются следующие протоколы:

RARP

> *Протокол обратного разрешения адресов* (*Reverse Address Resolution Protocol*) сообщает клиенту его адрес IP. Сервер RARP решает эту задачу путем преобразования Ethernet-адреса клиента в IP-адрес. Отображения адресов Ethernet в адреса IP хранятся на сервере в файле */etc/ethers*.

BOOTP

> *Протокол инициализации* позволяет получать целый ряд значений настройки.

DHCP

> *Протокол динамической настройки узлов* (*Dynamic Host Configuration Protocol*) заменяет BOOTP службой, предоставляющей полный набор параметров настройки, описанный в документе RFC *Requirements for Internet Hosts*. Кроме того, DHCP реализует динамическое выделение адресов, что позволяет эффективно распределять ограниченные наборы адресов.

Эта глава завершает введение в архитектуру, протоколы и службы сетей TCP/IP. В следующей главе мы займемся созданием сети TCP/IP и начнем с процесса планирования установки.

4

Начинаем работу

- *Связанные и не связанные с Интернетом сети*
- *Базовые сведения*
- *Планирование: маршрутизация*
- *Планирование: служба имен*
- *Прочие службы*
- *Что сообщить пользователям*

В этой главе мы переходим от механизмов TCP/IP к вопросам настройки. Первые три главы содержали описание протоколов TCP/IP и принципов их действия, а теперь пришло время изучить процесс настройки сети. Первый шаг этого процесса – планирование. Настройка узла для работы с TCP/IP требует наличия определенной информации. Как минимум каждый узел должен иметь имя и уникальный IP-адрес. Кроме того, следует уточнить следующие параметры:

Адрес шлюза по умолчанию

Если предполагается обмен данными по TCP/IP с системами, расположенными за пределами локальной сети, может потребоваться адрес шлюза по умолчанию. Как вариант, если в сети используется протокол маршрутизации, каждое сетевое устройство должно понимать этот протокол.

Адреса серверов имен

Каждый узел должен знать адреса серверов имен домена, чтобы иметь возможность переводить имена узлов в адреса IP.

Имя домена

Узлы, работающие с системой доменных имен, должны знать свое доменное имя.

Маска подсети

Для правильной организации связи каждая система сети должна использовать корректную сетевую маску.

Если речь идет о добавлении системы в существующую сеть, не забудьте проконсультироваться с администратором и получить ответы на все вопросы, прежде чем подключать машину. Сетевой администратор отвечает за принятие и распространение решений, касающихся настройки и конфигурации

сети в целом. В случае существующей сети TCP/IP читатели могут смело пропустить несколько разделов данной главы. Рекомендуется все же прочитать о выборе имен, планировании почтовых систем и других моментах, в равной степени важных как для сложившихся, так и для только что созданных сетей.

Если речь идет о создании сети TCP/IP с нуля, необходимо принять ряд стартовых решений. Будет ли новая сеть связана с Интернетом? Если так, то посредством какого подключения? Как выбрать номер сети? Как зарегистрировать доменное имя? Как выбирать имена узлов? В последующих разделах мы изучим информацию, необходимую для принятия таких решения.

Связанные и не связанные с Интернетом сети

Прежде всего, следует решить, будет ли новая сеть полностью интегрирована в сеть Интернет. *Связанная сеть* напрямую подключена к Интернету и полностью доступна для других сетей. *Несвязанная сеть* напрямую не подключена к сети Интернет, и доступ из нее в Интернет ограничен. Примером несвязанной сети может служить сеть TCP/IP, в которой сообщение с внешним миром происходит через прокси-сервер (посредник) или сервер, осуществляющий преобразование сетевых адресов (NAT). Пользователи несвязанной сети могут работать с внешними системами, узлами сети Интернет, но пользователи внешних систем не могут напрямую взаимодействовать с узлами несвязанной сети. Несвязанные сети не предоставляют внешнему миру доступ к службам и потому известны также в качестве *частных сетей*.

Частные сети, связующие различные подразделения одной организации, часто называют *сетями предприятий*. Сети предприятий, в которых приложения информационных служб, основанные на TCP/IP (в частности, веб-серверы и броузеры), применяются для распространения внутренней информации, называются *интрасетями* (*intranets*), или *корпоративными сетями*.

Есть ряд простых причин тому, что многие локальные сети оказываются лишь частично подключенными к сети Интернет. Первая из таких причин – безопасность. Подключение к любой сети увеличивает число людей, имеющих доступ к вашей системе. Подключение к глобальной сети с миллионами пользователей способно напугать любого специалиста по безопасности. Вне всяких сомнений – подключение к сети Интернет представляет серьезную угрозу безопасности вашей сети. В главе 12 описаны некоторые из методов противостояния этой угрозе.

Второй момент – затраты. Многие организации не видят особой пользы в том, что каждая настольная система обретет полное подключение к Интернету. Для некоторых организаций низкая потребность в доступе или ограниченные запросы (например, использование только электронной почты) не оправдывают затрат на подключение к Интернету целой сети. Иногда основной причиной подключения к Интернету оказывается необходимость предоставлять информацию о производимой продукции. Чтобы решить такую за-

дачу, нет необходимости подключать к сети Интернет всю сеть предприятия. Зачастую оказывается достаточно подключить единственный веб-сервер, обратившись к местному поставщику интернет-услуг, либо заключить соглашение с компанией, предоставляющей услуги по размещению информации на веб-узлах.

Существуют организации, для которых подключение к Интернету – насущная необходимость. Образовательные и научные учреждения используют глобальную сеть в качестве источника информации, а для многих компаний Интернет – это средство предоставления услуг и поддержки клиентов.

Оба типа сети могут присутствовать одновременно: несвязанная сеть предприятия, скрытая за барьером безопасности брандмауэра, и небольшая связанная сеть, предоставляющая доступ к службам в пользу внешних клиентов и выступающая в качестве посредника для внутрисетевых пользователей.

Без точного определения потребностей и затрат на подключение к сети Интернет не представляется возможным понять, имеет ли смысл подключать сеть целиком. Местные поставщики интернет-услуг предоставят различные варианты цены и производительности. Интересуйтесь не только ценами, но и услугами. Некоторые из подобных организаций специализируются на обеспечении домашних пользователей дешевыми услугами. При этом упор делается на цену. Однако если речь идет о подключении целой сети, может потребоваться, чтобы поставщик интернет-услуг мог предоставить IP-адреса, доступ к службе имен, веб-размещение информации и прочие возможности, в которых может нуждаться ваша сеть.

Базовые сведения

Какое бы решение не было принято, с уверенностью можно сказать одно: сеть предприятия будет построена на протоколах TCP/IP. Все сети TCP/IP, подключенные и не подключенные к сети Интернет, требуют присутствия одних и тех же базовых сведений, позволяющих настроить физические сетевые интерфейсы. Как мы увидим в главе 6, сетевому интерфейсу требуется IP-адрес, а также при необходимости маска подсети и широковещательный адрес. Решение о подключении к сети Интернет влияет на способ получения значений, позволяющих настроить интерфейс. В этом разделе мы выясним, каким путем сетевой администратор получает каждое из требуемых значений.

Определение адреса IP

Каждый интерфейс сети TCP/IP должен иметь уникальный адрес IP. Если узел принадлежит сети Интернет, его IP-адрес должен быть уникальным для всей сети Интернет. Если общение узла по протоколам TCP/IP ограничивается локальной сетью, его IP-адрес должен быть уникальным лишь в рамках локальной сети. В случае сетей, не подключаемых к сети Интернет, администраторы могут выбирать адреса из документа RFC 1918, *Address Allocation for Private Intranets* (Выделение адресов для частных интрасетей),

в котором перечислены номера сетей, зарезервированные для частного использования.[1] Номера частных сетей:

- Сеть 10.0.0.0 (с префиксом 10/8) – это 24-битный блок адресов.
- Сети с 172.16.0.0 по 172.31.0.0 (с префиксом 172.16/12) – 20-битные блоки адресов.
- Сети с 192.168.0.0 по 192.168.255.0 (с префиксом 192.168/16) – 16-битные блоки адресов.

Недостаток сетевых адресов из документа RFC 1918 в том, что может возникнуть необходимость изменить их при подключении всей сети к сети Интернет. Частные сетевые адреса обладают следующими преимуществами:

- Простота. Нет необходимости обращаться в инстанции и получать официальные адреса, равно как и одобрение.
- Вежливость. Адресное пространство сохраняется для тех сетей, которым действительно необходимо подключение к сети Интернет.
- Цена. Адреса RFC 1918 бесплатны, тогда как официально полученные адреса стоят денег.

При использовании адресов из RFC 1918 узлы локальной сети имеют возможность обращаться к системам сети Интернет, пусть на практике это и потребует определенных усилий. Понадобится прокси-сервер либо сервер *преобразования сетевых адресов* (NAT). NAT существует в виде самостоятельных устройств, а также в виде дополнительных программных модулей к некоторым маршрутизаторам и брандмауэрам. NAT осуществляет преобразование исходного адреса дейтаграмм, покидающих вашу сеть, подставляя вместо частного адреса официальный. Преобразование адресов имеет ряд преимуществ:

- Экономит адреса IP. Большинство сетевых соединений происходит между системами сети предприятия. В каждый отдельный момент времени лишь небольшому числу систем требуется взаимодействие с Интернетом. Следовательно, число необходимых официальных адресов IP значительно меньше, чем общее число систем в сети предприятия. NAT позволяет использовать емкое адресное пространство из RFC 1918 для настройки сети предприятия и максимально сократить официальное адресное пространство Интернет-соединений.
- Сокращает риск подделки адресов («спуфинга»), то есть атак злоумышленников, в ходе которых удаленная система притворяется локальной. Адреса, перечисленные в RFC 1918, не могут участвовать в маршрутизации в сети Интернет. Таким образом, дейтаграммы из локальной сети могут передаваться удаленным системам посредством маршрутизации, но

[1] В тексте книги адрес 172.16.0.0 используется в качестве официального, хотя является номером частной сети, зарезервированным для применения в несвязанных сетях предприятий. Спокойно используйте этот адрес в своей сети, если не планируется подключение к сети Интернет.

если дейтаграмма содержит конечный адрес, упомянутый в RFC 1918, любой Интернет-маршрутизатор просто игнорирует такого *пришельца*.[1]

- Избавляет от необходимости в перенумерации узлов при подключении к сети Интернет.

У преобразования сетевых адресов есть и недостатки:

Стоимость

NAT может служить источником расходов на приобретение дополнительного аппаратного и программного обеспечения. Однако эти расходы, как правило, очень невелики.

Производительность

Преобразование адресов снижает производительность, делая обязательной дополнительную обработку каждой дейтаграммы. При изменении адреса необходимо повторно вычислить контрольную сумму. Более того, некоторые из протоколов более высокого уровня передают копию IP-адреса, которая также подлежит преобразованию.

Надежность

Маршрутизаторы, в отличие от NAT-серверов, никогда не изменяют адреса в заголовках дейтаграмм. Это может привносить некоторую нестабильность. Кроме того, протоколы и приложения, встраивающие адреса в свои данные, могут некорректно работать с NAT.

Безопасность

NAT ограничивает возможности для сквозного шифрования и проверки подлинности. Механизмы проверки подлинности, включающие заголовки в свои вычисления, не работают, поскольку NAT изменяет адреса в заголовках. Шифрование не работает, если зашифрованные данные содержат адрес источника.

Прокси-серверы предоставляют во многом те же преимущества, что и механизм NAT. Более того, эти названия часто используются в качестве взаимозаменяемых. Но различия все же присутствуют. Прокси-серверы – это шлюзы приложений, которые изначально создавались в составе систем, направленных на обеспечение безопасности, а именно – брандмауэров. Внутренние системы взаимодействуют с внешним миром через прокси, а внешние системы отвечают прокси. Прокси-серверы ориентированы на работу с конкретными приложениями. В сети может присутствовать прокси веб-сервера и прокси FTP-сервера. Каждый из них обслуживает соединения единственного типа приложений. Итак, различие между серверами NAT и прокси-серверами состоит в том, что NAT выполняет преобразование IP-адресов для любых приложений, а классический прокси-сервер нацелен на единственное приложение.

---

[1] Пришелец (martian, дословно *марсианин*) – это дейтаграмма с некорректным адресом.

В прокси-серверы часто добавляются функции системы безопасности. Преобразование адресов может выполняться на уровне IP. Службы прокси требуют обработки данных вплоть до прикладного уровня. Прокси-сервер может содержать фильтры безопасности на всех уровнях стека протоколов.

Принимая во внимание описанные различия, можно решить, что серверы NAT масштабируются лучше, чем прокси-серверы, а прокси-серверы более эффективно обеспечивают безопасность. Но с течением времени происходило слияние этих технологий, и сегодня их практически невозможно различить. Прежде чем принять решение о применении служб NAT или прокси, убедитесь, что они соответствуют нуждам сети.

Сочетание механизма NAT с частными адресами сети дает каждому из узлов сети доступ ко внешнему миру, но не позволяет внешним пользователям получать доступ к частной сети. Чтобы применить подобную модель, необходимо получить официальный IP-адрес.

Получение официального сетевого адреса

Сети, полностью подключенные к сети Интернет, должны иметь официальные сетевые адреса. Каждая система, *напрямую доступная* для Интернет-узлов, должна иметь официальный адрес. Каждая сеть, вовлеченная в обмен данными с Интернетом, даже в случае использования NAT, должна иметь по меньшей мере один официальный адрес, хотя этот адрес может и не быть постоянным. Первый шаг в получении блока адресов – определить потребность в адресной емкости.

Чтобы оценить потребности и способы их удовлетворения, можно начать с выявления «типа организации». RFC 2901, *Administrative Internet Infrastructure Guide* (Руководство по инфраструктуре сети Интернет), определяет четыре типа организаций:

Конечный пользователь сети Интернет

Небольшие и среднего размера организации, которым требуется подключение к сети Интернет. Диапазон: от единственного пользователя, получающего адрес динамически при подключении к сети от DHCP-сервера поставщика интернет-услуг, до целой сети из тысяч узлов с частной адресацией в рамках сети предприятия и официальными адресами для ограниченного числа доступных извне систем. Отличительный признак организаций этого типа – желание использовать ресурсы сети Интернет, ограничивая число своих систем, доступных внешним пользователям. Конечные пользователи получают официальные адреса от поставщиков интернет-услуг. С точки зрения сети Интернет организация этого типа оказывается невелика, поскольку задействует ограниченное число официальных адресов.

Крупные конечные пользователи

Среднего размера и крупные организации, распределяющие официальные адреса системам по всей корпоративной сети. Как правило, в таких организациях применяется модель распределенного управления, и под-

разделения обладают полномочиями делать системы доступными для внешних пользователей. Крупные конечные пользователи обычно удовлетворяют адресные потребности, обращаясь к поставщикам интернет-услуг или локальным регистраторам сети Интернет. Если потребности организации превышают 8000 адресов, она может обращаться напрямую к локальному регистратору сети Интернет. И хотя в действительности сеть такой организации может уступать размерами сети конечного пользователя сети Интернет, с точки зрения сети Интернет она «крупнее», поскольку большее число ее систем доступно извне.

Поставщики интернет-услуг

Организации, предоставляющие другим организациям услуги подключения к сети Интернет и официальные адреса. Даже поставщик услуг должен каким-то образом подключаться к Сети. Поставщики интернет-услуг получают адреса от поставщиков *более высокого уровня*, либо, при подключении напрямую в точках доступа к сети (NAP, network access point), упоминавшихся в главе 2, от местного или регионального регистратора сети Интернет.

Локальные регистраторы сети Интернет

Организации, снабжающие адресами поставщиков интернет-услуг. По сути дела, локальный регистратор сети Интернет (Local Internet Registry) – первоисточник для всех других организаций, занятых в распространении адресов. Локальный регистратор получает свои адреса от регионального регистратора (Regional Internet Registry).

Эти четыре типа организаций приводятся в RFC 2901 для полноты картины, но большая часть организаций принадлежит к первым двум типам. С большой вероятностью вы работаете в одной из таких организаций, так что адреса предстоит получать от поставщика интернет-услуг.

Вашему поставщику услуг были делегированы полномочия в рамках группы сетевых адресов, так что он сможет назначить номер вашей сети. Если местный поставщик услуг не способен удовлетворить ваши нужды, имеет смысл обращаться к поставщику более высокого уровня. Поинтересуйтесь у местного поставщика услуг, чьими услугами он сам пользуется, и запросите адреса в этой организации. В самом крайнем случае придется обращаться к Интернет-регистратору. Если это случится, необходимо предпринять определенные шаги перед тем, как подавать заявление.

Потребуется подробное описание топологии сети. Топология должна включать диаграмму, отражающую физическую структуру сети и подчеркивающую каналы подключения к сети Интернет. Следует приложить также проектные документы, описывающие помимо топологии следующие аспекты:

- Планы маршрутизации, включая задействованные протоколы и упоминания ограничений, заставивших принять те или иные решения по маршрутизации.
- Планы по созданию подсетей, включая маску, а также число сетей и узлов, подключение которых планируется в течение года. Документ RFC 2050,

Internet Registry IP Allocation Guidelines (Нормы выделения IP-адресов реестрами Интернета) предлагает включить в данный план следующие элементы:

- Таблицу с перечислением всех подсетей
- Маску для каждой подсети. Весьма приветствуется использование масок подсетей переменной длины (VLSM, variable-length subnet mask). Маски VLSM описаны далее в этой главе в разделе «Создание маски подсети»
- Оценку числа узлов
- Описательный текст, определяющий назначение каждой из подсетей

Наибольшую трудность представляет прогноз потребностей в адресах. Если вы ранее уже получали адресный блок, с вас могут потребовать отчет о том, как была израсходована адресная емкость этого блока. В любом случае подобный исторический отчет полезен – если не для регистратора, то для процесса дальнейшего планирования. Помимо этого может потребоваться план развертывания сети. Обычно такой план отражает число уже существующих узлов, которым требуются официальные адреса, и прогноз числа подобных узлов на следующие шесть месяцев, год и два года.

Для определения емкостных потребностей адресного пространства можно применять, в частности, *ожидаемый коэффициент использования*. Ожидаемый коэффициент использования – это отношение числа узлов, которым присвоены официальные адреса, к максимально возможному числу узлов в сети. Планы построения должны отражать число узлов, которым будут присвоены адреса за последующие два года. Общее число узлов можно оценить, исходя из общего числа сотрудников в организации и числа машин, приходящихся на одного сотрудника. Очевидно, обращение за официальными адресами требует подробнейших познаний о структуре организации и ее нуждах.

Помимо документации, подтверждающей необходимость в выделении адресов, получение официального адреса требует формального выделения ресурсов. Как правило, в заявке на выделение адресов требуется указать двух ответственных лиц: административное и техническое. Административное лицо должно обладать полномочиями, простирающимися от случаев нарушения установленных правил до финансовых тяжб. Технический специалист должен быть в состоянии справляться с техническими затруднениями и отвечать на сопутствующие вопросы. Регистраторы требуют, чтобы ответственные лица проживали в той же стране, что и организация, которую они представляют. Необходимо предоставить имена, адреса, телефонные номера и адреса электронной почты этих людей. Не обманывайтесь – это вовсе не почетные должности. Как только возникнут проблемы – на спинах этих людей появятся мишени.

Регистратор вносит контактную информацию в базу данных `whois`, которая содержит публично-доступную контактную информацию о людях, отвечающих за функционирование сетей. Когда ваше имя внесено в базу данных, вы получаете уникальный идентификатор NIC-handle, связанный с записью в

базе данных whois. Мой идентификатор NIC-handle – cwh3. Идентификаторы NIC требуются при подаче многих официальных заявок.

Помимо кадровых ресурсов необходимо выделить и компьютерные. Обращаясь за получением официальных адресов, следует иметь готовую, работоспособную систему.

Когда с подготовительной работой покончено, можно обращаться в организацию-регистратор Интернета. Выделение адресов IP находится в ведении трехуровневой бюрократической машины:

IANA

Служба назначения числовых Интернет-адресов IANA (The Internet Assigned Numbers Authority) выделяет крупные блоки адресов региональным реестрам Интернета.

Региональный регистратор Интернета

Региональные регистраторы Интернета уполномочены IANA на выделение адресов в крупных регионах мира. Существует три региональных регистратора:

APNIC

Сетевой информационный центр APNIC (Asian Pacific Network Information Center) уполномочен выделять адреса в Азии и тихоокеанском регионе.

ARIN

Сетевой информационный центр ARIN (American Registry for Internet Numbers) уполномочен выделять адреса в обеих Америках.

RIPE

Организация Reseaux IP Europeens уполномочена выделять адреса на европейском континенте.

Локальный регистратор Интернета

Локальные регистраторы получают полномочия от IANA или региональных регистраторов на выделение адресов в пределах определенной зоны. Примером такого регистратора может служить национальный регистратор либо регистратор, созданный ассоциацией поставщиков услуг Интернета.

Независимо от уровня потребностей в адресах, следует начинать работать с основания иерархии и постепенно продвигаться к вершине. Всегда обращайтесь прежде всего к локальным поставщикам услуг. Если они не способны удовлетворить ваши нужды, поинтересуйтесь, существует ли локальный регистратор, с которым имеет смысл связаться. В качестве крайней меры обращайтесь к региональному регистратору, уполномоченному обслуживать вашу часть света.

В случае региона APNIC придется прежде всего заполнить заявление на членство. Форма заявления доступна по адресу *http://www.apnic.net/member/*

application.html. Получив членство в APNIC, можно запрашивать получение адреса.

ARIN не требует вступления в организацию для получения адреса. Крупные конечные потребители могут использовать форму *http://www.arin.net/templates/networktemplate.txt*. Поставщики услуг Интернета – форму *http://www. arin.net/templates/isptemplate.txt*. В любом варианте отправляйте заполненные формы по адресу *hostmaster@arin.net*.

Конечные пользователи в регионе RIPE должны обращаться к местным регистраторам. RIPE выделяет адреса только местным регистраторам и только при условии членства в RIPE. RIPE не занимается выделением адресов для конечных пользователей. Более подробная информация содержится в документе *ftp://ftp.ripe.net/ripe/docs/ripe-159.txt*.

Запомните один важный факт: независимо от географического расположения сети, организации, как правило, не проходят через столь сложный процесс получения адресов, предпочитая ограничивать число машин, напрямую доступных из сети Интернет. Из соображений безопасности применяются схемы частной адресации для большинства система и скромное число официальных IP-адресов. Последние, в ограниченных количествах, всегда могут быть получены у местного поставщика услуг Интернета.

Получение домена IN-ADDR.ARPA

При получении официального IP-адреса следует также озаботиться получением домена *in-addr.arpa*. Этот особый домен иногда называют *обратным*. Создание и применение домена *in-addr.arpa* описано в главе 8, что же касается сути дела – *обратный домен* реализует преобразование адресов IP в доменные имена, то есть решает задачу, обратную поиску по доменному имени, который позволяет преобразовывать доменные имена в адреса. Если поставщик услуг Интернета предоставляет в ваше распоряжение собственную службу имен либо выделяет официальный адрес из собственного блока адресов, как правило, нет необходимости самостоятельно обращаться за получением домена *in-addr.arpa*. Проконсультируйтесь со своим поставщиком услуг, *прежде чем* предпринимать дополнительные действия. Если же блок адресов получен напрямую от регионального регистратора Интернета, вероятнее всего, понадобится получить собственный домен *in-addr.arpa*. Если такой домен действительно необходим, регистрация происходит в той организации, которая выделила адреса.

- Если адресный блок выделен APNIC, заполните форму *ftp://ftp.apnic.net/apnic/docs/in-addr-request* и отправьте на адрес электронной почты *domreg@rs.apnic.net*.
- Если адресный блок получен от ARIN, заполните форму *http://www.arin.net/templates/modifytemplate.txt* и отправьте на адрес электронной почты *hostmaster@arin.net*.
- Если адресный блок получен от RIPE, объект домена должен быть помещен в базу данных RIPE. Отправьте полный объект на адрес электронной почты *auto-inaddr@ripe.net*.

Для примера предположим, что сеть находится в регионе RIPE. Следует собрать информацию, необходимую для создания объекта домена для сети. Объект домена для базы данных RIPE отражает ту информацию, которая требуется для регистрации обратного домена. Объект базы данных RIPE содержит десять полей:

domain:

Имя домена. Подробности построения обратных доменных имен приводятся в главе 8, а по сути дела такое имя является комбинацией адреса, записанного в обратном порядке, и строки *in-addr.arpa*. Для адресного блока 172.16/16 обратное доменное имя будет выглядеть как *16.172.in-addr.arpa*.

descr:

Текстовое описание домена. Например, «Блок адресов для wrotethebook.com.».

admin-c:

Идентификатор NIC административного контактного лица.

tech-c:

Дескриптор NIC технического контактного лица.

zone-c:

Дескриптор NIC администратора домена, контактного лица зоны.

nserver:

Имя или адрес основного сервера имен домена.

nserver:

Имя или адрес подчиненного сервера имен домена.

nserver:

В случае RIPE этот третий сервер всегда *ns.ripe.net*.

changed:

Адрес электронной почты лица, создавшего объект базы данных, и дата поступления объекта.

source:

Для адресов, выделенных RIPE, это поле всегда имеет значение *RIPE*.

Опять же, в разговоре о регистрации обратных адресов важно отметить, что большинство организаций обходится без таковой. Если адреса получены от поставщика услуг Интернета, вероятнее всего, не возникнет необходимости тратить время на все эти хлопоты с заявлениями. За это поставщикам услуг и платят.

Назначение адресов узлам

До сих пор мы говорили о *номерах сетей*. Сети нашей воображаемой компании был присвоен номер 172.16.0.0/16. Сетевой администратор назначает

отдельным узлам адреса из диапазона разрешенных к назначению IP-адресов, то есть определяет последние 2 байта 4-байтового адреса.[1] Часть адреса, определенная администратором, не может состоять только из одних нулей или одних единиц. Адреса 172. 16.0.0 и 172.16.255.255 не могут назначаться узлам. Таким образом, с двумя упомянутыми исключениями, адреса могут распределяться любым способом, который кажется разумным.

Сетевые администраторы обычно назначают адреса узлам одним из двух способов:

По одному адресу

Каждому узлу назначается адрес из диапазона адресов. Порядок распределения адресов, как правило, последовательный.

Группами адресов

Блоки адресов делегируются подразделениям организации, которые выполняют назначение адресов отдельным узлам.

Назначение адресов группами – обычное явление, если сеть разделена на подсети, а адресные группы разделены по границам подсетей. При этом групповое назначение адресов не требует разделения на подсети, оно может быть просто приемом распределения полномочий в рамках организации. Делегирование полномочий для группы адресов зачастую оказывается весьма удобным для крупных сетей, тогда как маленькие сети вполне обходятся назначением отдельных адресов. Независимо от способа назначения адресов, должен существовать человек, имеющий достаточно полномочий, чтобы предотвратить дублирование адресов и обеспечить правильность их записи на серверах имен домена.

Адреса могут назначаться статически или динамически. Статическое назначение выполняется в процессе создания загрузочного файла конкретного узла. В динамическом назначении адресов всегда участвует сервер, например сервер DHCP. В числе прочих преимущество динамического распределения адресов – гарантия отсутствия в сети дублирующихся адресов. Таким образом, динамическое назначение удобно не только тем, что облегчает труд администратора, но и тем, что сокращает риск ошибок.

Прежде чем устанавливать сервер динамической адресации, убедитесь, что он соответствует вашим потребностям. Динамическая адресация PPP удобна для серверов, работающих с большим числом удаленных клиентов, устанавливающих кратковременные коммутируемые соединения. Если сервер PPP объединяет различные части сети предприятия и поддерживает долгоживущие соединения, динамическая адресация, скорее всего, окажется ненужной. Точно так же, возможности динамического назначения адресов протокола DHCP приносят наибольшую пользу, если в сети существуют мобильные системы, мигрирующие из подсети в подсеть и нуждающиеся в частой смене адресов. Информация о PPP содержится в главе 6, а главы 3 и 9 подробно описывают DHCP.

---

[1] Диапазон адресов называется *адресным пространством*.

Очевидно, администратор должен принять ряд решений по получению и назначению адресов. Кроме того, он должен определить, какая битовая маска будет использоваться для адресов. В следующем разделе речь пойдет о маске подсети, которая определяет способ интерпретации адреса.

Создание маски подсети

Как следует из числа префикса, сетевой адрес назначается в сочетании с конкретной адресной маской. Например, в адресе сети 172.16.0.0/16 префикс 16 означает, что организация ARIN выделила нашей воображаемой сети блок адресов, определенный адресом 172.16.0.0 и 16-битной маской 255.255.0.0.[1] В отсутствие причин изменять интерпретацию выделенного номера сети нет надобности создавать маску подсети. В главе 2 содержится описание структуры адресов IP и кратко затронуты возможные причины создания подсетей. Решение по образованию подсетей, как правило, имеет в основе топологические или организационные соображения.

Возможные топологические причины разделения на подсети:

Снятие ограничений на расстояния

Некоторые сетевые устройства обладают довольно жесткими ограничениями на дальность работы. Изначальный Ethernet на 10 мегабит – хороший тому пример. Максимальная длина «толстого» кабеля Ethernet составляет 500 метров; максимальная длина «тонкого» кабеля – 300 метров; общая длина 10-мегабитного Ethernet-маршрута, или максимальный диаметр, составляет 2500 метров.[2] Чтобы покрыть сетью большие расстояния, можно воспользоваться IP-маршрутизаторами для объединения цепочек кабелей Ethernet. Отдельные сегменты кабеля все так же не должны превышать максимально допустимой длины, но в случае такого подхода каждый кабель становится самостоятельным Ethernet-маршрутом. Таким образом, общая длина IP-сети может превышать максимальную длину для Ethernet.

Объединение физических сетей различного строения

IP-маршрутизаторы могут использоваться для связи сетей, основанных на различных, возможно, даже несовместимых сетевых технологиях. Рис. 4.1 (см. далее в главе) отражает центральную кольцевую сеть 172.16.1.0 (Token Ring), связующую две подсети Ethernet, 172.16.6.0 и 172.16.12.0.

Межсетевая фильтрация трафика

Локальный трафик остается в пределах локальной подсети. Только трафик, предназначенный другим сетям, передается через шлюз.

---

[1] В примерах книги адрес 172.16.0.0 считается официально полученным адресом сети, хотя на деле является номером частной сети, упомянутым в RFC 1918.

[2] Чем быстрее Ethernet, тем меньше диаметр сети. По этой причине высокоскоростные технологии Ethernet строятся на коммутаторах, а не на цепочках кабельных сегментов.

Разделение на подсети – не единственный способ разрешения топологических сложностей. Сети строятся на аппаратном обеспечении и могут изменяться на том же уровне – заменой или добавлением устройств, однако разделение на подсети является эффективным способом решения подобных проблем на уровне TCP/IP.

Разумеется, причины для создания подсетей не исчерпываются техническими моментами. Подсети часто служат целям структурирования, таким как:

Облегчение сетевого администрирования

Подсети могут использоваться для делегирования управления адресами, действий по диагностированию проблем и других обязанностей сетевого администратора более мелким группам в рамках одной организации. Это очень удобный инструмент, позволяющий небольшой команде управлять крупной сетью. Ответственность за сопровождение подсети ложится на людей, которые непосредственно с ней работают.

Следование структуре организации

Структура организации (или просто ее политика) может требовать независимого управления сетевыми системами для некоторых подразделений. Создание подсетей с независимым управлением для таких подразделений более предпочтительно, чем получение каждым из подразделений выделенных номеров сетей у поставщика услуг Интернета.

Защита трафика организации

Определенные организации предпочитают, чтобы локальный трафик ограничивался подсетью, которая доступна только определенным членам организации. Это в особенности важно, если речь идет о конфиденциальных данных. Например, пакеты финансового подразделения не должны попасть в техническую подсеть, где один из неглупых сотрудников вполне способен придумать способ их перехвата.

Изоляция потенциальных проблем

Если определенный сегмент сети менее надежен, чем остальные, имеет смысл выделить этот сегмент в подсеть. Предположим, исследовательская группа время от времени включает в сеть экспериментальные системы, чтобы поэкспериментировать на собственно сети. Это приводит к снижению стабильности работы всего сегмента. Выделение в подсеть предотвращает влияние экспериментальных устройств или программ на работу всей сети.

Сетевой администратор решает, нужно ли разделение на подсети, и создает маску подсети. Маска подсети имеет тот же формат, что и адресная маска IP. Как говорилось в главе 2, она позволяет определить, какие биты адреса принадлежат к разделу сети, а какие – к разделу узла. Биты раздела сети включены (имеют значение 1), а биты раздела узла сброшены (имеют значение 0).

В нашей воображаемой сети используется маска подсети 255.255.255.0. Эта маска выделяет 8 битов на идентификацию подсетей, что позволяет задействовать 256 подсетей. Администратор *сети* решил, что 256 подсетей по 254

узла в каждой обеспечат достаточно эффективное использование адресного пространства. Приведенный далее рис. 4.1 отражает такой способ разделения на подсети. Если применить описанную маску подсети к адресам 172.16.1.0 и 172.16.12.0, они будут интерпретированы не как адреса узлов одной сети, но как адреса узлов в различных сетях.

После создания маски ее необходимо распространить по всем узлам сети. Существует два решения этой задачи: настроить сетевые интерфейсы вручную либо настроить их автоматически, при помощи протокола настройки – например DHCP. Протоколы маршрутизации способны передавать маски подсетей, но в большинстве сетей протоколы маршрутизации не задействованы на простых узлах. В последнем случае все устройства сети должны работать с одной маской подсети, поскольку каждый компьютер считает, что вся сеть разбита на подсети точно таким же образом, как локальная подсеть.

Протоколы маршрутизации передают адресную маску для каждого получателя в отдельности, и потому становится возможным применение масок подсетей переменной длины (VLSM, variable-length subnet mask). Маски переменной длины повышают гибкость и эффективность процесса образования подсетей. Предположим, необходимо разделить сеть 192.168.5.0/24 на три сети: сеть из 110 узлов, сеть из 50 узлов и сеть из 60 узлов. В случае традиционных масок подсетей все адресное пространство разбивается на сегменты одной маской. В лучшем случае придется идти на компромисс. Маски подсетей переменной длины позволяют воспользоваться маской 255.255.255.128 для создания крупных подсетей из 126 узлов и маской 255.255.255.192 для создания подсетей поменьше – из 62 узлов. Однако применение масок VLSM требует, чтобы каждый маршрутизатор сети умел хранить и использовать такие маски, а кроме того, передавать их при помощи протоколов маршрутизации. (Более подробно маршрутизация описана в главе 7.) Маршрутизация – важнейшая составляющая сети TCP/IP. Как и прочие ключевые компоненты сети, маршрутизацию следует тщательно спланировать еще до начала настройки.

Планирование: маршрутизация

В главе 2 говорилось, что узлы способны общаться напрямую только с другими системами той же сети. Общение с узлами других сетей реализуется посредством шлюзов. Чтобы такое общение стало возможным, должен существовать маршрут, пролегающий через шлюз. Существует два способа определить такой маршрут:

- Посредством *статической таблицы маршрутизации*, создаваемой администратором системы. Статические таблицы маршрутизации оказываются наиболее эффективными в ситуации, когда число шлюзов ограничено. Статические таблицы не способны автоматически подстраиваться под происходящие в сети изменения, а потому каждое изменение в таблицах должно выполняться вручную администратором сети. Сложные среды требуют иного, гибкого подхода к маршрутизации.

- Посредством *динамической таблицы маршрутизации*, реагирующей на изменения в сети. Динамические таблицы маршрутизации создаются протоколами маршрутизации. Протоколы маршрутизации выполняют обмен информацией маршрутизации, на основе которой и обновляются таблицы. Динамическая маршрутизация применяется для сетей с большим числом шлюзов; она просто необходима, если пункт назначения доступен через несколько шлюзов.

Во многих сетях применяется сочетание статической и динамической маршрутизации. Одни системы сети работают со статическими таблицами, тогда как другие – с протоколами маршрутизации и динамическими таблицами. Статические таблицы маршрутизации часто являются оптимальным выбором для узлов, а шлюзы обычно функционируют на основе протоколов маршрутизации.

Администратор сети принимает решение о том, какой вид маршрутизации использовать, и выбирает шлюзы по умолчанию для всех узлов. Прежде чем начать настройку системы, определитесь с этими моментами.

Приведенные ниже указания помогут спланировать систему маршрутизации. Если речь идет о:

Сети, не имеющей шлюзов в другие сети TCP/IP

Не требуется специальная настройка маршрутизации. В тексте книги под шлюзами понимаются IP-маршрутизаторы, связывающие сети TCP/IP. В отсутствие связуемых сетей TCP/IP не нужны и IP-маршрутизаторы. Таким образом, не нужна информация о шлюзах по умолчанию и протоколах маршрутизации.

Сети с единственным шлюзом

Не требуется использование протоколов маршрутизации. Указывайте единственный шлюз в качестве шлюза по умолчанию в статической таблице маршрутизации.

Сети, имеющей внутренние шлюзы в другие подсети и один внешний шлюз

В этой ситуации появляется возможность выбора: статически указать маршрут в каждую из подсетей и сделать маршрут через внешний шлюз маршрутом по умолчанию либо задействовать протокол маршрутизации. Принимая решение, руководствуйтесь трудоемкостью и производительностью вариантов – сопровождение статических таблиц требует усилий, протоколы маршрутизации добавляют работы узлам и сети. С ростом числа узлов применение протокола маршрутизации становится менее трудоемким.

Сети с большим числом внешних шлюзов

Если одни и те же пункты назначения доступны по ряду маршрутов, используйте протокол маршрутизации. Это позволит шлюзам приспосабливаться к изменениям в сети, обеспечивая ее избыточным доступом во внешние сети.

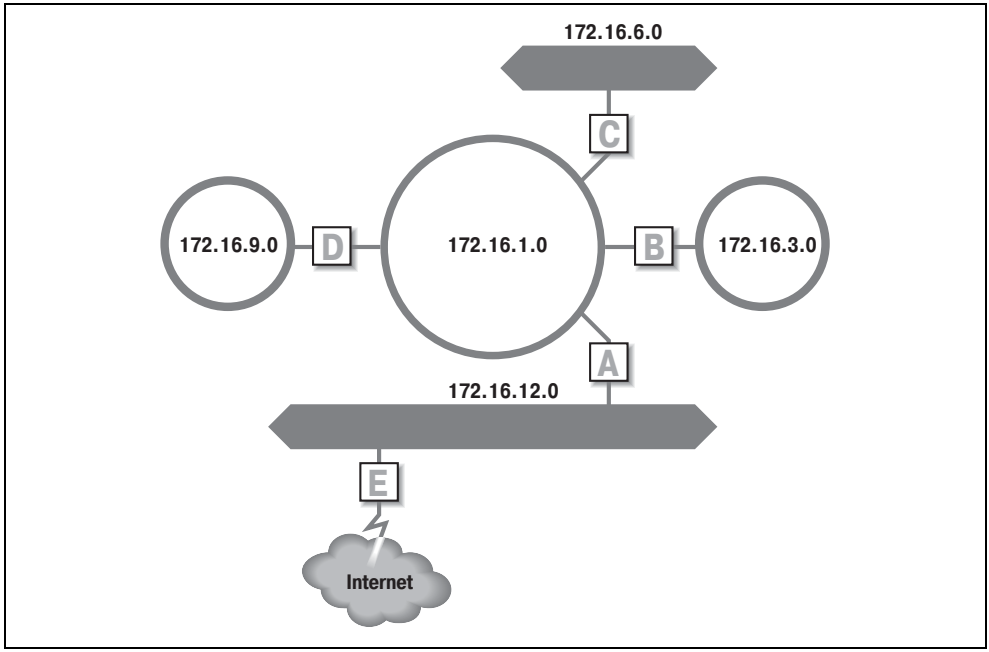


Рис. 4.1. *Маршрутизация и подсети*

На рис. 4.1 представлена разделенная на подсети сеть с пятью шлюзами, от *A* до *E*. Центральная подсеть (172.16.1.0) объединяет пять других подсетей. В одной из подсетей существует шлюз во внешнюю сеть. Администратор сети, вероятнее всего, примет решение использовать протокол маршрутизации в центральной подсети (172.16.1.0) и, возможно, в подсети 172.16.12.0, связанной с внешней сетью. Динамическая маршрутизация в этих подсетях уместна из-за многочисленных шлюзов. В отсутствие динамической маршрутизации администратор будет вынужден обновлять каждый из шлюзов вручную при любых изменениях в сети – например при создании дополнительной подсети. В таком случае достаточно одной ошибки, чтобы нарушить работу сети. Более надежным и менее трудоемким решением для этих двух подсетей является протокол маршрутизации.

С другой стороны, для остальных сетей (172.16.3.0, 172.16.6.0, 172.16.9.0) администратор, скорее всего, применит статическую маршрутизацию. У каждой из этих сетей есть только один шлюз, через который проходит любая маршрутизация. Изменения, не затрагивающие эти подсети, такие как создание дополнительных подсетей, не меняют того факта, что в трех подсетях существует лишь один маршрут. Создаваемые сети будут доступны через тот же шлюз в каждой из подсетей. Узлы этих подсетей считают шлюз подсети маршрутом по умолчанию. Иными словами, узлы подсети 172.16.3.0 считают шлюзом по умолчанию шлюз *B*, тогда как узлы подсети 172.16.9.0 считают шлюзом по умолчанию шлюз *D*, независимо от того, что происходит во внешних сетях.

Отдельные решения по маршрутизации приходится принимать при подключении ко внешним сетям. Сеть, представленная на рис. 4.1, подключена к внешней сети, требующей применения протокола пограничных шлюзов (Border Gateway Protocol, BGP). Таким образом, на шлюзе *E* должен работать протокол BGP, позволяющий обмениваться маршрутами с внешней сетью.

Получение номера автономной системы

Протокол пограничных шлюзов BGP требует наличия у шлюза специального идентификатора, *номера автономной системы* (*autonomous system number*, ASN).[1] Большинству площадок протокол BGP не требуется. А если и требуется, то работает обычно с идентификатором ASN поставщика услуг Интернета либо с одним из идентификаторов, зарезервированных для частного использования (номера с 64 512 по 65 535). Чтобы избежать конфликтных ситуаций, скоординируйте выбор идентификатора ASN со стороной, расположенной по ту сторону границы шлюза. Если подключение к сети Интернет обеспечивает один поставщик услуг, официальный номер ASN практически наверняка не понадобится. Если при обсуждении вопроса со своим поставщиком услуг вы пришли к выводу о *необходимости* получения официального идентификатора ASN, обращайтесь в региональный реестр Интернета, обслуживающий вашу страну.

- Азиатский и тихоокеанский регионы обслуживаются организацией APNIC. Заполните форму *http://ftp.apnic.net/apnic/docs/asn-request* и отправьте ее по адресу электронной почты *hostmaster@apnic.net*.
- Обе Америки обслуживаются организацией ARIN. Заполните форму *http://www.arin.net/templates/asntemplate.txt* и отправьте ее по адресу электронной почты *hostmaster@arin.net*.
- Европейские страны обслуживаются организацией RIPE. Заполните форму *ftp://ftp.ripe.net/ripe/docs/ripe-147.txt* и отправьте ее по адресу электронной почты *hostmaster@ripe.net*.

При подаче заявления требуется объяснить, зачем нужен уникальный номер автономной системы. Если вы не представляете организацию-поставщика услуг Интернета, вероятно, единственная разумная причина – наличие *многосетевой* (*multi-homed*) *площадки*. Многосетевая площадка – это любая площадка, подключенная более чем через одного поставщика услуг. Информация достижимости для площадки может быть зарегистрирована каждым поставщиком услуг в отдельности, что приведет к неоднозначностям в исполнении правил маршрутизации. Выделение площадке номера ASN делает ее ответственной за определение правил маршрутизации и регистрацию собственной информации достижимости. Это не предотвращает регистрацию некорректных маршрутов, но позволяет отследить источник некорректных сведений и обратиться к ответственному техническому лицу. (Как только вы получите номер ASN, винить, кроме себя, будет некого!)

---

[1] Автономные системы представлены в разделе «Архитектура маршрутизации сети Интернет» главы 2.

Регистрация в базе данных маршрутизации

После обретения официального номера ASN следует принять решение о регистрации в базе данных маршрутизации. Если номер ASN получен для многосетевой площадки, следует зарегистрироваться в базе данных маршрутизации. В разделе «Архитектура маршрутизации сети Интернет» главы 2 рассказано, каким образом базы данных маршрутизации используются для проверки маршрутов сети Интернет сегодня, когда уже нет централизованного стержня, которому можно довериться в выборе «лучших» маршрутов. Получив официальный номер ASN, вы становитесь частью структуры равноправных доменов маршрутизации. Принятая ответственность за небольшую долю тяжелой ноши маршрутизации подтверждается регистрацией в базе данных.

Существует ряд баз данных, составляющих реестр маршрутизации сети Интернет (Internet Routing Registry, IRR). Помимо базы данных RADB (Routing Arbiter Database, RADB), упоминавшейся в главе 2, существуют базы данных RIPE, ANS, Bell Canada и Cable & Wireless. RIPE обслуживает клиентов в регионе RIPE. ANS, Bell Canada и Cable & Wireless регистрируют только своих коммерческих клиентов. RADB доступна всем.

Для регистрирации в базе данных RADB следует сначала зарегистрировать объект службы технической поддержки (maintainer object). Служебный объект ссылается на лицо, ответственное за сопровождение ваших записей в базе данных. Укажите соответствующую информацию и заплатите взнос $200. Затем следует зарегистрировать автономную систему в качестве объекта AS. Наконец, для каждого маршрута, декларируемого вашей системой, следует создать объект маршрутизации Route. Подробная информация о регистрации этих объектов базы данных доступна по адресу *http://www.radb.net*.

Все рассмотренные вопросы (адресация, образование подсетей, маршрутизация) должны быть решены, прежде чем вы начнете настройку физической сети, на базе которой работают приложения и службы. Теперь мы приступаем к планированию служб, которые делают сеть полезной и удобной для работы.

Планирование: служба имен

Чтобы сделать сеть комфортной для пользователей, необходимо обеспечить работу службы, выполняющей преобразование имен узлов в IP-адреса. Задача решается системой доменных имен (Domain Name System, DNS) и таблицами узлов, как мы узнали в главе 3. Следует планировать применение и той и другой технологии.

Для настройки компьютера пользователю сети необходимо знать доменное имя, имя системы, а также имя узла и адрес по меньшей мере одного сервера имен. Эта информация исходит от администратора сети.

Получение доменного имени

Доменное имя – первое, что нужно для организации службы имен. Поставщик интернет-услуг может получить имя самостоятельно, если предоставляет

подобную услугу, либо выделить имя в одном из своих доменов; однако более вероятно, что вам придется самостоятельно подавать заявку на доменное имя. Купить официальное доменное имя можно у регистратора доменных имен.

Ваш домен не является частью официального пространства доменных имен, если не зарегистрирован. Определенные организации обладают полномочиями по регистрации доменных имен. Необходимо найти официального регистратора и воспользоваться его услугами для регистрации доменного имени. Можно начать с адреса *http://www.icann.org* или *http://www.internic.net*. На каждом из сайтов представлены перечни официальных регистраторов.

ICANN (Internet Corporation for Assigned Names and Numbers) – это некоммерческая организация, взявшая на себя часть функций, выполнявшихся ранее подрядчиками правительства США. ICANN осуществляет надзор за регистраторами доменных имен. На веб-сайте ICANN представлены координаты различных международных регистраторов.

http://www.internic.net – веб-сайт правительства США, задача которого – направлять пользователей к официальным регистраторам gTLD-имен и отвечать на возможные вопросы пользователей, связанные с процессом регистрации доменов. Воображаемый домен, используемый в книге, зарегистрирован в домене *.com*. Сайт InterNIC – хорошая точка старта, если речь идет о регистрации в доменах *.org*, *.com* или *.net*. На рис. 4.2 отражена часть алфавитного перечня уполномоченных регистраторов с сайта *http://www.internic.net*.

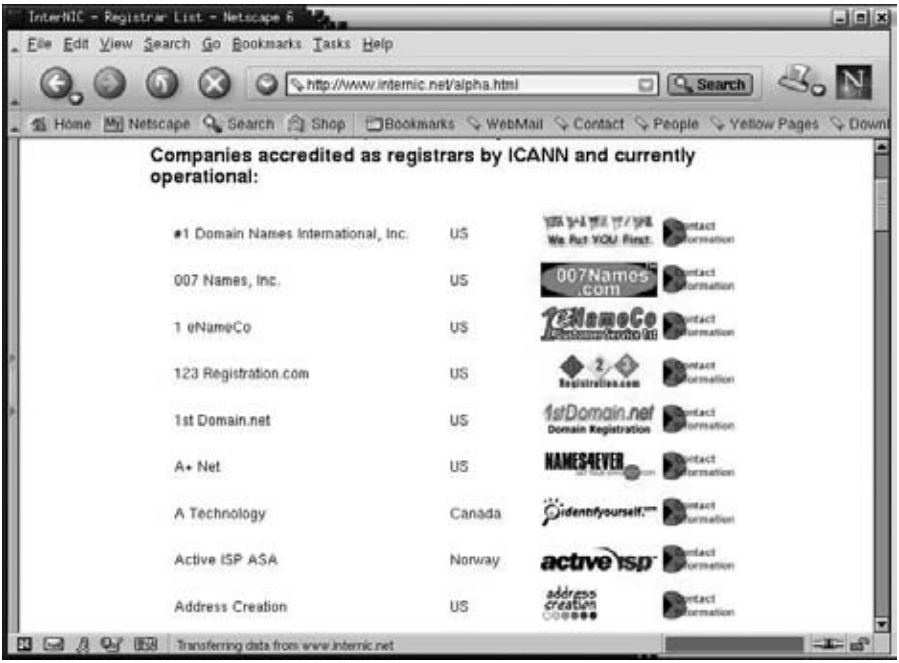


Рис. 4.2. *Перечень регистраторов*

Регистраторы мало чем различаются. Регистрация доменов стоит очень недорого, обычно не более $50 в год, так что цена особой роли не играет. Качество услуг тоже проблематично оценить, поскольку домен после регистрации, как правило, не требует сопровождения. Некоторые администраторы предпочитают организации, близкие географически, но даже это не очень важно в нашем мире, опутанном проводами. Руководствуйтесь собственным здравым смыслом. Откровенно говоря, мне не удалось найти параметры, позволяющие рекомендовать того или иного регистратора. В последующих примерах в качестве регистратора выступает компания Network Solutions – отчасти потому, что она находится в двух шагах от моего дома. Читателям же предлагаю выбрать регистраторов самостоятельно.

Регистрация домена

Выбрав регистратора, отправляйтесь на сайт организации и прочитайте инструкции по регистрации домена. Если вы находитесь на сайте *http://www.internic.net*, то можете перейти на сайт конкретного регистратора, щелкнув по его эмблеме. Большинство организаций-регистраторов предоставляют веб-формы заявлений на регистрацию доменных имен.

Например, выбрав компанию Network Solutions из перечня на *http://www.internic.net*, мы попадаем на *http://www.netsol.com*. Здесь необходимо выбрать доменное имя. Этот первый шаг обязателен – производится поиск в существующих базах данных системы доменных имен. Если имя уже занято, придется выбрать другое доменное имя. Если имя свободно, необходимо предоставить информацию о компетентных серверах имен, которые будут обслуживать новый домен. Отдельные регистраторы, включая компанию Network Solutions, предоставляют службы DNS для домена за дополнительную плату. Поскольку мы планируем создать собственный сервер для домена *wrotethebook.com* domain, то укажем информацию о собственных серверах.

Прежде всего, требуется указать имя лица, юридически ответственного за домен. Эта информация используется регистратором для выставления счетов и включается в базу данных `whois`, содержащую контактную информацию по лицам, ответственным за домены. Если у вас уже есть запись в базе данных `whois`, следует указать свой идентификатор NIC, связанный с этой записью. Мой идентификатор NIC – `cwh3`.

Новые клиенты должны указывать имена и адреса лиц, к которым можно обращаться по административным, техническим и финансовым вопросам. Это могут быть три разных человека либо один и тот же человек в зависимости от того, как организовано предприятие.

Затем система запрашивает имена и IP-адреса двух серверов, которые являются компетентными для этого домена. Введите имена основного и подчиненного серверов, которые настроены на обслуживание нового домена. На момент заполнения этой формы серверы уже должны функционировать. Если это не так, можно заплатить чуть больше и разместить домен на площадке компании Network Solutions, пока серверы не заработают. Не следует вводить имена

серверов, не готовых к работе, поскольку это приведет к некорректному делегированию – корневые серверы имен используют эти сведения и разместят в доменах высшего уровня указатели на серверы, не являющиеся в действительности компетентными. Поэтому либо заранее настройте серверы, пусть даже в самом простом варианте, либо заплатите чуть больше и зарезервируйте доменное имя до тех пор, пока серверы имен не будут готовы к работе.

Проверьте введенную информацию. Оплатите счет. Вы готовы управлять собственным доменом.

Выбор имени узла

Получив доменное имя, администратор обретает ответственность за присвоение имен узлам, которые принадлежат домену. Следует давать узлам имена, уникальные в пределах домена или поддомена, – здесь действует то же правило, что и для адресов в рамках сетей и подсетей. Но выбор имени узла не ограничивается поиском уникального сочетания символов; в этом решении, как ни удивительно, большую роль могут играть личные предпочтения. Многие люди очень скрупулезно относятся к выбору имени компьютера, поскольку склонны идентифицировать машины с собой или своей работой.

В RFC 1178 содержатся превосходные руководящие указания по выбору имен узлов. Вот некоторые из наиболее серьезных указаний:

- Используйте реально существующие слова, краткие, легко запоминаемые, с простой орфографией. Причина использования имен узлов вместо адресов IP в том, что их *легче* использовать. Имена узлов, сложные в написании и запоминании, противоречат самой цели их создания.
- Используйте тематические имена. К примеру, узлам одной группы можно дать имена, соответствующие человеческим движениям: fall, jump, hop, skip, walk, run, stagger, wiggle, stumble, trip, limp, lurch, hobble и т. д. Тематические имена зачастую легче выбирать, поскольку тема ограничивает область поиска, и, кроме того, они способствуют сплочению пользователей сети.
- Избегайте использования имен проектов, имен людей, аббревиатур, численных имен и терминов технического жаргона. Проекты и пользователи сменяются другими проектами и пользователями. Если назвать компьютер именем человека, который работает на нем в настоящее время, либо именем проекта, в котором он задействован, в будущем, весьма вероятно, компьютер придется переименовать. Используйте прозвища (псевдонимы) для указания на серверную функциональность системы: *www*, *ftp*, *nfs* и т. д. Прозвища легко переносятся в другие системы вместе с серверной функциональностью. Информация о создании псевдонимов содержится в главе 8 (записи CNAME).

Единственное требование, предъявляемое к имени узла, – оно должно быть уникальным в пределах домена. Однако тщательный выбор имени может сэкономить затраты времени в будущем и сделать работу пользователей более комфортной.

Служба имен – одна из самых нужных сетевых служб, и весьма вероятно, что она будет применяться в вашей сети. Однако существуют и другие службы, которые следует рассмотреть в процессе планирования сети.

Прочие службы

Еще три службы задействованы во многих сетях – файловые серверы, серверы печати и почтовые серверы. В главе 3 говорилось о назначении этих служб и протоколах, на основе которых они функционируют. В этом разделе мы определим, какой информацией должны обладать пользователи для успешной настройки клиентских систем и как эту информацию получает администратор сети.

Файловые серверы

Как минимум, пользователь должен знать имена узлов файловых серверов сети. При помощи имен и команды `showmount` пользователь может определить, какие файловые системы предоставляются серверами для монтирования и кому разрешено с ними работать.[1] Не имея в своем распоряжении точного имени, пользователь будет гадать, какая из систем предлагает службу совместного доступа к файлам.

Более эффективным будет снабдить пользователей информацией, включающей данные и о предоставляемых файловых системах, и о возможности использовать эти файловые системы. Например, если страницы руководства (`man`) Unix доступны для чтения на центральном сервере, следует сообщить пользователям, что нет необходимости держать локальную копию руководства на каждой машине, и снабдить их точной инструкцией о том, как получить доступ к централизованному хранилищу файлов.

Серверы печати

Независимо от способа организации совместного доступа к принтерам (`lp`, `lpd`, Samba), основная информация, необходимая для настройки клиентов службы печати, остается той же: имя узла и IP-адрес сервера печати, а также имя печатающего устройства. В случае принтера, не поддерживающего PostScript, может понадобиться информация о производителе и модели. Из соображений разграничения доступа для доступа к принтеру может требоваться имя пользователя и пароль.

Другая информация для настройки клиентских машин не требуется. Но вероятнее всего, имеет смысл сообщить пользователям дополнительные сведения о возможностях, расположении и администрировании принтеров совместного доступа.

---

[1] Команда `showmount` описана в главе 9.

Планирование почтовой системы

TCP/IP предоставляет возможности создания надежной и гибкой системы электронной почты. Одной из функций серверов является повышение надежности. Можно создать сеть равноправных почтовых систем, в которой каждая система будет посылать и получать свою почту напрямую. Однако доставка и отправка почты каждой системой потребует соответствующего администрирования, а также непрерывного либо согласованного функционирования всех систем. Это непрактично, поскольку многие небольшие системы большую часть дня не подключены к сети. В большинстве сетей используются серверы, что позволяет ограничить настройку и сопровождение почтовой службы небольшим числом систем.

Терминология серверов электронной почты сбивает с толку, поскольку все серверные функции обычно выполняются одной машиной, а все термины используются в качестве взаимозаменяемых для обозначения этой машины. В тексте этой книги проводятся различия между функциями почтового сервера, но предполагается, что все задачи будут выполняться одной системой под управлением sendmail. Итак, термины имеют следующие значения:

Почтовый сервер (Mail server)

Почтовый сервер собирает входящую почту для других компьютеров сети. Он поддерживает диалоговое взаимодействие, а также протоколы POP и IMAP, так что пользователи имеют возможность работать со своей почтой так, как им удобно.

Почтовый ретранслятор (Mail relay)

Почтовый ретранслятор – это узел, передающий почту между внутренними системами и от внутренних систем внешним узлам. Почтовые ретрансляторы упрощают настройку электронной почты для внутренних систем – специальное программное обеспечение, работающее со схемами адресации почты и псевдонимами, устанавливается только на ретрансляторах.

Почтовый шлюз (Mail gateway)

Почтовый шлюз – это система, реализующая обмен почтой для несовместимых систем. Чтобы передать почту от одного узла Интернета другому, не нужны шлюзы, поскольку обе системы работают по протоколу SMTP. Шлюз нужен для осуществления перехода от SMTP к X.400 или к фирменной почтовой программе. В однородной сети TCP/IP подобная функциональность не требуется.

Наиболее важной составляющей надежной системы является почтовый сервер, поскольку он позволяет службе работать, не прибегая к ресурсам пользовательских систем. Сервер с централизованным, компетентным управлением накапливает почту независимо от того, функционирует ли система-адресат.

Кроме того, надежность системы электронной почты основана на узлах-ретрансляторах. Если узел-ретранслятор по каким-то причинам не может доставить почту немедленно, почта помещается в очередь и обрабатывается позже. Конечная система также помещает почтовые сообщения в очередь, но ес-

ли она перестает работать, обработка очереди задерживается до следующего включения. Почтовые серверы и ретрансляторы работают 24 часа в сутки.

Архитектура большинства почтовых сетей TCP/IP основана на следующих руководящих принципах:

- Используйте почтовый сервер для накопления почты, протоколы POP и IMAP для доставки почты клиентам.
- Используйте почтовый ретранслятор для передачи почты. Выбирайте упрощенные схемы адресации для узла ретранслятора.
- Стандартизируйте применение TCP/IP и SMTP. Пользователи, настаивающие на применении фирменной почтовой системы, должны сами приобрести и настроить почтовый SMTP-шлюз для этой системы.
- Стандартизируйте применение MIME для двоичных вложений. Избегайте фирменных вариантов прикрепления файлов к письмам; это приводит к замешательству, поскольку пользователи почтовой системы фирмы X не смогут прочесть вложения, полученные от системы фирмы Y.

Для настройки клиентских систем необходимо имя узла и IP-адрес почтового сервера и почтового ретранслятора. Кроме того, почтовый сервер требует, чтобы каждый пользователь предоставлял регистрационное имя и пароль.

Что сообщить пользователям

Вся настроечная информация, собранная или созданная в процессе планирования, должна передаваться пользователям, чтобы они могли настраивать свои системы. Существует ряд методов, позволяющих облегчить участь пользователей.

Во-первых, необходимо максимально освободить пользователей от работы по настройке. В главе 3 мы обсуждали NIS, NFS и серверы настройки. Все эти компоненты позволяют упростить процесс настройки, а наиболее важную роль играет DHCP. Серверы DHCP предоставляют все параметры, необходимые для настройки клиентов TCP/IP. Все данные, о которых шла речь в этой главе – адрес IP, маска подсети, имя узла, имя домена, шлюзы по умолчанию, адреса серверов, – могут быть автоматически получены клиентом посредством DHCP. Конечный пользователь в процессе не участвует.

Одна из важных функций DHCP связана со способностью протокола перенаправлять клиентов к другим серверам сети. Серверы требуют соответствующей настройки клиента. В случае NIS и NFS клиент должен иметь полную базовую настройку, что позволяет NIS и NFS обеспечить дополнительные уровни поддержки процесса настройки. NIS организует доступ к ряду административных баз данных, содержащих многие значения базовой настройки. NIS позволяет централизованно управлять такими базами данных, не вовлекая в процесс пользователей настольных Unix-систем. NFS позволяет передавать клиентским системам заранее созданные системные файлы и документацию.

Однако даже в сочетании с другими серверами DHCP не является полноценным решением. DHCP требует от пользователя знаний о том, что DHCP вообще применяется, и ввода корректных значений в момент первоначальной установки системы. Таким образом, администратор сети должен напрямую переда, при помощи печатной документации или среды Web.

Администраторы сетей часто решают задачу при помощи кратких справочников для пользователей. В случае DHCP передаваемая пользователям информация зачастую идентична для всех клиентов Unix и для всех клиентов Windows. Например, клиенты Unix могут получить указание использовать DHCP для настройки интерфейса, установить поддержку NIS и NFS, а кроме того, смонтировать определенные файловые системы NFS. Клиенты Windows могут получить указание использовать DHCP для настройки интерфейса, а также определенные имена рабочих групп и имена NetBIOS.

Создание сети TCP/IP требует от администратора скрупулезного планирования. Когда планы готовы, их следует задокументировать и сообщить о принятых решениях людям, которые будут работать в новой сети.

Резюме

Планирование – первый шаг к настройке TCP/IP. В начале главы мы выясняли, будет ли наша сеть подключена к Интернету, и изучали последствия этого решения для всего процесса планирования. Кроме того, мы рассмотрели базовые элементы информации, необходимые для настройки физической сети: адрес IP, маску подсети, имя узла. Затем речь пошла о планировании маршрутизации, обязательной для обмена данными между сетями TCP/IP. Были кратко затронуты основные сетевые службы, включая DNS, почтовые и файловые серверы, а также серверы печати. Наконец, мы изучили различные пути передачи данных планирования от сетевого администратора к системным администраторам и пользователям.

В последующих главах мы будем воплощать этот план в жизнь и начнем с настройки сетевого интерфейса (глава 6). Однако прежде мы обратимся к строению ядра Unix и узнаем, каким образом протоколы TCP/IP внедряются в операционную систему.

5

- *Настройка ядра*
- *Загрузочные файлы*
- *Демон Internet*
- *Расширенный демон Internet*

Базовая настройка

Каждая машина под управлением Unix, работающая с TCP/IP, имеет возможность интеграции базовых служб транспорта и IP-дейтаграмм в операционную систему. В этой главе мы рассмотрим два метода базовой настройки TCP/IP в системе Unix: перекомпиляцию ядра и загрузку модулей ядра. Мы поговорим о роли этих методов в интеграции TCP/IP и Unix. Получив подобную информацию, читатели смогут понять, как принимаются решения по базовой настройке на уровне разработчиков и поставщиков систем и как изменять базовые настройки, решая собственные задачи.

Службы транспорта и дейтаграмм, интегрированные в операционную систему, используются прикладными службами, описанными в главе 3. Существует два различных метода запуска прикладных служб: в процессе загрузки системы либо по необходимости. Оба варианта описаны в этой главе. Приведена информация по настройке и управлению процессом загрузки системы. Но сначала займемся интеграцией TCP/IP в операционную систему Unix.

Настройка ядра

Вообще говоря, настройка ядра не является задачей администратора сети – это скорее одна из основных составляющих администрирования системы Unix, которая присутствует независимо от того, подключен ли компьютер к сети. Однако сетевые возможности TCP/IP, подобно прочим системным функциям, интегрируются в ядро системы.

Существует два очень разных подхода к настройке ядра. Некоторые системы спроектированы таким образом, чтобы избавить администратора от необходимости перекомпилировать ядро, тогда как другие, напротив, приветствуют создание собственного ядра. Примером последней философии может слу-

жить система Linux: в ее документации предлагается создавать ядра, отвечающие поставленным задачам. Примером первого подхода является система Solaris.

Система Solaris поставляется с общим ядром, поддерживающим все основные системные службы. В процессе загрузки система Solaris определяет все устройства и использует динамически загружаемые модули, чтобы работать с этими устройствами. Solaris может полагаться на этот метод лишь потому, что основным поставщиком аппаратного обеспечения для системы является компания Sun. Sun разрабатывает свои устройства, имея в виду ядро системы Solaris, и представляет четко определенный интерфейс для драйверов устройств, благодаря чему сторонние разработчики могут разрабатывать аппаратные средства, четко идентифицируемые ядром.

Динамически загружаемые модули

В большинстве вариантов системы Unix присутствует поддержка динамически загружаемых модулей, то есть модулей ядра, подключаемых к нему в процессе работы. Эти модули позволяют системе достигать невероятной гибкости, поскольку ядро загружает модули поддержки аппаратного обеспечения по мере его обнаружения. Динамически загружаемые модули позволяют наращивать функциональность системы, не требуя от системного администратора повторной настройки, выполняемой вручную.

Solaris работает на основе динамически загружаемых модулей. Да, в Solaris существует файл настройки ядра, а именно */etc/system*, но этот файл весьма мал, имеет ограниченные возможности и не редактируется системным администратором напрямую. В процессе установки нового программного пакета сценарий установки вносит все необходимые изменения в файл */etc/system*. И даже такая ситуация встречается нечасто. Большинство драйверов, поставляемых с устройствами сторонних разработчиков, имеют собственные файлы настройки.

В системе Solaris необязательные драйверы устройств устанавливаются при помощи команды `pkgadd`. Синтаксис команды следующий:

```
pkgadd -d device packagename
```

Здесь *device* – имя устройства, *packagename* – название пакета с программными драйверами.

Установка драйвера устройства приводит к созданию сопутствующей записи в каталоге */dev*, а также в каталоге */kernel/drv*. В качестве примера рассмотрим драйвер Ethernet для сетевых карт, созданных на основе набора микросхем DEC 21140. Имя драйвера – *dnet*.[1] В каталоге устройств существует устройство с именем */dev/dnet*. В каталоге драйверов ядра присутствует ди-

[1] dnet не является дополнительным устройством. Это стандартный компонент системы Solaris и по совместительству устройство Ethernet, которое мы используем во всех примерах по Solaris.

намически загружаемый модуль */kernel/drv/dnet*, и, кроме того, существует файл настройки драйвера – */kernel/drv/dnet.conf*. dnet является стандартным драйвером, но установка дополнительного драйвера приводит к созданию похожих файлов.

Завершив установку нового драйвера, создайте пустой файл с именем */reconfigure*. Остановите работу системы и установите новое устройство. Перезапустите систему. Файл */reconfigure* сигнализирует системе о необходимости проверить наличие новых устройств в системе. При загрузке система Solaris обнаружит новое устройство и загрузит модуль, содержащий драйвер этого устройства.

Команда Solaris `ifconfig`, описанная во всех подробностях в главе 6, имеет параметр `modlist`, приводящий к отображению модулей ядра, связанных с определенным сетевым интерфейсом TCP/IP. Например:

```
# ifconfig dnet0 modlist
0 arp
1 ip
2 dnet
```

Назначение каждого из перечисленных модулей ядра вполне очевидно. arp реализует протокол ARP для интерфейса Ethernet. ip реализует протоколы TCP/IP, применяемые в этой сети. Каждому из модулей соответствует файл настройки, хранящийся в каталоге */kernel/drv*. Существуют файлы *arp.conf*, *ip.conf* и *dnet.conf*. Однако эти файлы предоставляют весьма ограниченные возможности настройки модулей. Для управления модулями система Solaris предоставляет команду `ndd`.

Чтобы выяснить, какие параметры настройки доступны для модуля, воспользуйтесь командой `ndd` с аргументом `?`. Например, следующая команда перечисляет переменные для модуля arp:

```
# ndd /dev/arp ?
?                               (read only)
arp_cache_report                (read only)
arp_debug                       (read and write)
arp_cleanup_interval            (read and write)
arp_publish_interval            (read and write)
arp_publish_count               (read and write)
```

Модуль arp предлагает шесть значений для настройки:

`?`

Значение только для чтения, приводит к отображению данного списка.

`arp_cache_report`

Значение только для чтения, приводит к отображению постоянных записей из буфера ARP. Команда `arp` предоставляет более удобный доступ к значениям из буфера. Описание команды `arp` приводится в главе 2.

`arp_debug`

Переменная, позволяющая включить отладку для протокола ARP. По умолчанию имеет значение 0, отладка выключена. Значение 1 приводит к включению отладки. Протокол ARP – очень старый и очень надежный, так что отладка ARP никогда не требуется.

`arp_cleanup_interval`

Переменная, определяющая срок хранения в буфере временных записей.

`arp_publish_interval`

Переменная, определяющая интервал посылки широковещательных сообщений с опубликованными адресами Ethernet.

`arp_publish_count`

Переменная, определяющая число широковещательных сообщений, посылаемых в ответ на запрос опубликованного адреса.

Стандартные значения переменных модуля arp замечательно работали для всех систем Solaris, с которыми мне приходилось сталкиваться. У меня еще ни разу не возникала необходимость изменять какое-либо из них. Второй из упомянутых по параметру `modlist` модулей представляет больший интерес.

Для получения параметров настройки модуля ip воспользуйтесь командой `ndd /dev/ip ?`. Существует почти 60 параметров! Из всех этих параметров мне приходилось настраивать лишь один: `ip_forwarding`.

Переменная `ip_forwarding` определяет, должен ли модуль ip считать, что система является маршрутизатором, и пересылать пакеты другим узлам. По умолчанию системы с одним сетевым интерфейсом не пересылают пакеты, а системы более чем с одним интерфейсом являются маршрутизаторами и выполняют такую пересылку. Нулевое значение переменной `ip_forwarding` отключает пересылку пакетов, даже для системы с несколькими сетевыми интерфейсами. Значение `ip_forwarding`, равное единице, включает пересылку пакетов, даже для системы с одним сетевым интерфейсом.

Время от времени приходится иметь дело с многосетевыми узлами, то есть с узлами, подключенными к нескольким сетям. Несмотря на несколько сетевых соединений такая система является обычным узлом, а не маршрутизатором. Чтобы система не вела себя как маршрутизатор, вмешиваясь в функционирование действительной системы маршрутизации, отключите пересылку пакетов IP следующим образом:

```
# ndd /dev/ip ip_forwarding
1
# ndd -set /dev/ip ip_forwarding 0
# ndd /dev/ip ip_forwarding
0
```

В этом примере первая команда `ndd` запрашивает значение переменной `ip_forwarding` для модуля ip. Мы видим, что значение переменной равно 1 и пересылка пакетов включена. Вторая команда `ndd` при помощи параметра `-set`

присваивает переменной ip_forwarding нулевое значение. Последняя команда повторно отображает значение переменной, чтобы мы могли удостовериться, что оно изменилось.

Команда pkgadd, параметр modlist команды ifconfig, а также команда ndd специфичны для системы Solaris. В других системах также существуют динамически загружаемые модули, но управляются они иными наборами команд.

Так, поддержка загружаемых модулей присутствует в Linux. Linux получает от загружаемых модулей все те же преимущества, что Solaris, и требует столь же ограниченного вмешательства в работу этих модулей. Обычно система Linux выполняет обнаружение устройств и определение соответствующих модулей в процессе первоначальной установки, не требуя участия системного администратора. Но это не всегда так. Устройства могут не определяться в процессе установки либо могут добавляться в уже существующую систему. Чтобы справиться с подобными ситуациями, необходимо знать команды Linux, применяемые для работы с загружаемыми модулями.

Чтобы выяснить, какие модули установлены в системе Linux, воспользуйтесь командой lsmod. Вот пример для системы Red Hat:

```
# lsmod
Module                  Size  Used by
ide-cd                 26848   0  (autoclean)
cdrom                  27232   0  (autoclean) [ide-cd]
autofs                 11264   1  (autoclean)
smc-ultra               6048   1  (autoclean)
8390                    6816   0  (autoclean) [smc-ultra]
ipchains               38976   0  (unused)
nls_iso8859-1           2880   1  (autoclean)
nls_cp437               4384   1  (autoclean)
vfat                    9392   1  (autoclean)
fat                    32672   0  (autoclean) [vfat]
```

Загружаемые модули решают широкий спектр задач. Некоторые реализуют драйверы устройств (например, модуль smc-ultra является драйвером для сетевой карты SMC Ultra Ethernet), другие обеспечивают поддержку многочисленных типов файловых систем, применяемых в Linux, таких как файловая система ISO8859 для компакт-дисков или система DOS FAT с поддержкой длинных имен файлов (vfat).

Каждая запись перечня, выведенного командой lsmod, начинается с имени модуля, за которым следует размер модуля. Как можно видеть по полю Size, модули имеют небольшие размеры. Часто модули используют другие модули для решения своей задачи. Взаимосвязь модулей известна в качестве *зависимостей модулей*, и эти зависимости отражены в перечне модулей. В нашем примере драйвер smc-ultra в своей работе полагается на модуль 8390, как можно видеть из записи для 8390 – она заканчивается строкой «[smc-ultra]». В записи для 8390 модули, зависящие от него, перечислены под заголовком *Used by*. Можно наблюдать и другие зависимости: vfat зависит от fat, а cdrom зависит от ide-cd.

Большинство записей примера содержит строку «(autoclean)». Строка означает, что модуль может удаляться из памяти автоматически, если он не задействован. autoclean – это параметр. Параметры работы модулей можно указывать, вручную загружая их при помощи команды `insmod`.

Модули загружаются вручную при помощи команды `insmod`. Команда очень простая – достаточно указать имя модуля. Например, чтобы загрузить драйвер устройства 3c509, наберите команду `insmod 3c509`. В этом случае модуль загружается без параметра autoclean. Чтобы драйвер удалялся из памяти автоматически, добавьте ключ `-k` в команду `insmod`: `insmod -k 3c509`.

Серьезное ограничение команды `insmod` связано с тем, что она ничего не смыслит в зависимостях модулей. Если загрузить при помощи этой команды модуль smc-ultra, это не приведет к автоматической загрузке необходимого модуля 8390. По этой причине для ручной загрузки модулей лучше использовать команду `modprobe`. Синтаксис этой команды также очень прост. Чтобы загрузить модуль smc-ultra, наберите просто `modprobe smc-ultra`.

`modprobe` в работе использует файл зависимостей, созданный командой `depmod`. После обновления ядра или библиотеки модулей выполните команду `depmod`, чтобы создать новый файл зависимостей модулей. Команда `depmod -a` выполняет поиск всех стандартных библиотек модулей и создает соответствующий файл. По завершении этой операции воспользуйтесь `modprobe` для загрузки модулей с автоматическим отслеживанием зависимостей.

Для удаления ненужных модулей применяется команда `rmmod`. Опять же, ее синтаксис прост: `rmmod appletalk` удаляет из системы драйвер appletalk. Необходимость в ручном удалении модулей возникает редко, поскольку, как говорилось при обсуждении autoclean, неиспользуемые модули удаляются системой автоматически.

Модуль smc-ultra реализует драйвер устройства Ethernet. Именно этот драйвер устройства используется в работе сетевого интерфейса системы Linux из нашего примера. Драйверы устройств могут встраиваться в ядро – такой вариант мы рассмотрим позже, либо динамически загружаться из модулей. Большинство драйверов устройств Ethernet используются в качестве загружаемых модулей. Модули драйверов Ethernet расположены в каталоге */lib/modules*. В случае системы Red Hat 7.1 драйверы устройств Ethernet находятся в каталоге */lib/modules/2.4.7-10/kernel/drivers/net*, как можно видеть из следующего фрагмента:

```
# ls /lib/modules/2.4.7-10/kernel/drivers/net
3c501.o             atp.o       eexpress.o  ni5010.o       smc-ultra.o
3c503.o             bcm         epic100.o   ni52.o         starfire.o
3c505.o             bonding.o   eql.o       ni65.o         strip.o
3c507.o             bsd_comp.o  es3210.o    pcmcia         sundance.o
3c509.o             cipe        eth16i.o    pcnet32.o      sunhme.o
3c515.o             cs89x0.o    ethertap.o  plip.o         tlan.o
3c59x.o             de4x5.o     ewrk3.o     ppp_async.o    tokenring
8139too.o           de600.o     fc          ppp_deflate.o  tulip
82596.o             de620.o     hamachi.o   ppp_generic.o  tun.o
```

```
8390.o              defxx.o     hp100.o     ppp_synctty.o  via-rhine.o
ac3200.o            depca.o     hp.o        rcpci.o        wan
acenic.o            dgrs.o      hp-plus.o   sb1000.o       wavelan.o
aironet4500_card.o  dmfe.o      irda        shaper.o       wd.o
aironet4500_core.o  dummy.o     lance.o     sis900.o       winbond-840.o
aironet4500_proc.o  e1000.o     lne390.o    sk98lin        yellowfin.o
appletalk           e100.o      natsemi.o   skfp
arlan.o             e2100.o     ne2k-pci.o  sk_g16.o
arlan-proc.o        eepro100.o  ne3210.o    slip.o
at1700.o            eepro.o     ne.o        smc-ultra32.o
```

Здесь перечислены все загружаемые драйверы сетевых устройств. Некоторые из них, например *plip.o*, относятся вовсе не к устройствам Ethernet. Большинство модулей можно легко опознать по именам файлов, в частности драйверы 3COM, SMC, NE2000, а также Ethernet Express.

Система Linux определяет наличие устройств Ethernet в процессе первоначальной установки и в случае присутствия подходящего драйвера устанавливает его. Если карта Ethernet не определена при установке системы либо добавлена в систему уже после установки, воспользуйтесь командой `modprobe`, чтобы загрузить драйвер устройства вручную. Если подходящий драйвер не входит в комплект поставки системы, возможно, понадобится самостоятельно скомпилировать нужный модуль.

Для корректной работы драйвера устройства необходима компиляция модуля с библиотеками, соответствующими текущему ядру системы. Иногда приходится загружать исходный текст драйвера и компилировать его самостоятельно на целевой системе. Исходные тексты драйверов многих устройств Ethernet доступны на сайте *http://www.scyld.com*, в замечательном хранилище сетевых драйверов для Linux. Комментарии, сопровождающие исходный текст драйвера, обычно содержат указание по командам для компиляции модуля.

После компиляции модуля скопируйте полученный объектный файл в соответствующий каталог иерархии */lib/modules*. Воспользуйтесь командой `modprobe` для загрузки и проверки работоспособности драйвера. Следует отметить, большинство драйверов устройств сегодня доступно в формате пакетов RPM, что позволяет обойтись без компиляции.

В системах Linux часто применяются динамически загружаемые модули драйверов устройств. Однако прочие компоненты TCP/IP не загружаются во время работы, они встроены в ядро системы. В следующем разделе мы рассмотрим перекомпиляцию ядра системы Unix.

Перекомпиляция ядра

В тексте главы в качестве систем, поощряющих перекомпиляцию ядра, мы рассмотрим Linux и FreeBSD.[1] Примеры операторов настройки ядра отно-

[1] Процесс настройки ядра для других систем BSD, таких как SunOS 4.1.3, схож с процессом, описанным для системы FreeBSD.

сятся к этим двум системам. И хотя настройка ядра связана с каждым из аспектов работы операционной системы, здесь описаны только операторы, напрямую влияющие на работу TCP/IP.

Обе системы Unix, для которых приводятся примеры, поставляются с файлом настройки ядра, уже включающим все необходимое для работы с TCP/IP. В процессе первоначальной установки может возникнуть необходимость выбрать ядро с заранее определенной конфигурацией – скажем, ядро с поддержкой работы по сети, но, вероятнее всего, не будет необходимости изменять сетевые настройки ядра. Файл настройки ядра изменяется, как правило, при следующих обстоятельствах:

- Существует необходимость сократить размер ядра и повысить эффективность его работы, удалив ненужные компоненты
- Требуется добавить новое устройство
- Требуется изменить системный параметр

И хотя необходимость изменять сетевые настройки ядра возникает редко, полезно понимать, что именно означают эти настройки. Файл настройки ядра позволяет понять, как система Unix интегрируется с устройствами и программами сети.

Процедуры и файлы настройки ядра существенно различаются для каждой реализации Unix. Следствием этого обстоятельства является необходимость обращаться к документации по системе, прежде чем переходить к настройке ядра. Только в документации по конкретной системе содержатся точные, подробные инструкции, позволяющие успешно выполнить эту задачу.

Настройка ядра Linux

Исходный текст ядра Linux обычно поставляется в составе дистрибутива системы. Если в системе отсутствует исходный текст либо вы желаете воспользоваться более поздней версией ядра Linux, его можно загрузить с сайта *http://www.kernel.org* в виде сжатого tar-файла. Если каталог */usr/src/linux* уже существует, переименуйте его, прежде чем распаковывать архив:

```
# cd /usr/src
# tar -zxvf linux-2.1.14.tar.gz
```

Ядро Linux – это программа на языке C, которая компилируется и устанавливается при помощи инструмента `make`. Команда `make` позволяет изменять настройки ядра, а также генерирует файлы (включая файл сборки, Makefile), необходимые для компиляции и связывания кода ядра. Существует три варианта команды:

`make config`

Данный вариант команды – полностью текстовый. Он связан с прохождением через невероятно долгую серию вопросов по каждому из аспектов настройки ядра. Поскольку вопросы задаются строго последовательно,

данный способ – самый неудобный для перенастройки ядра, особенно если требуется изменить лишь несколько значений.

`make menuconfig`

Данный вариант использует библиотеку `curses` и предоставляет иерархию меню, посредством которой осуществляется настройка. Здесь присутствуют все возможности команды `make config`, и гораздо более удобный пользовательский интерфейс, позволяющий переходить прямо к интересующим аспектам настройки. Команда `make menuconfig` работает на любом терминале и в любой системе, даже в отсутствие поддержки X Windows.

`make xconfig`

Данный вариант основан на X Windows и предоставляет графический интерфейс для настройки ядра. Функциональность та же, что и у предыдущих вариантов, а использовать `make xconfig` очень легко.

Выбирайте вариант, пришедшийся вам по душе. Для нашего примера мы воспользуемся командой `make xconfig`.

В системе Linux исходный текст ядра хранится в иерархии */usr/src/linux*. Чтобы начать процесс настройки, следует перейти в каталог с исходным текстом и выполнить команду `make xconfig`:

```
# cd /usr/src/linux
# make xconfig
```

По команде `make xconfig` отображается приложение, показанное на рис. 5.1.

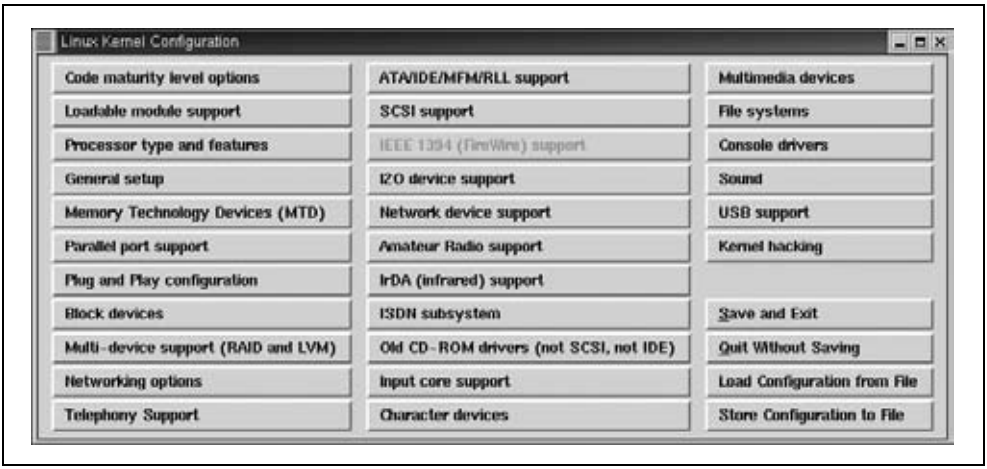


Рис. 5.1. *Главное меню xconfig в Linux*

Меню содержит более 30 кнопок, представляющих различные категории настройки. Нажмите на кнопку, чтобы просмотреть или изменить параметры настройки в определенной категории. Для нас интерес представляют те параметры настройки ядра, которые влияют на работу TCP/IP, то есть катего-

рии *Networking options* и *Network device support*. На рис. 5.2 показано окно, отображаемое по нажатию на кнопку *Network device support*.

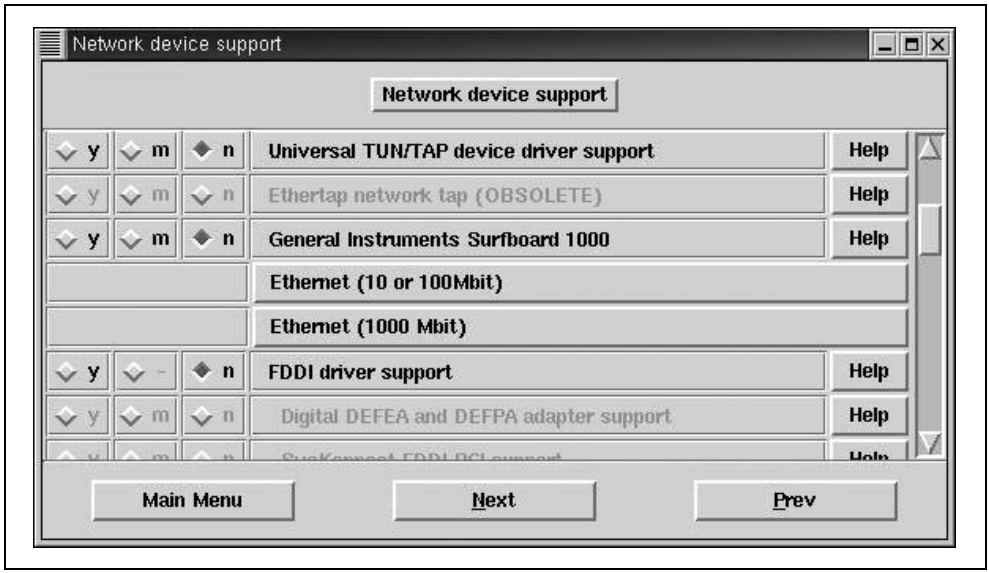


Рис. 5.2. *Поддержка сетевых устройств в ядре Linux*

В окне перечислены драйверы сетевых устройств, которые могут встраиваться в ядро либо загружаться в виде модулей. Большинство параметров настройки принимает одно из следующих значений:

y Выбор значения *y* приводит к встраиванию драйвера в ядро.

m Выбор значения *m* приводит к компиляции драйвера в модуль, динамически загружаемый ядром. Не всегда драйвер доступен в виде загружаемого модуля. Если параметр настройки требует ответа «да» или «нет», вариант с модулем недоступен. Обратите внимание на параметр *FDDI driver support*. Выбор значения *y* для этого параметра включает поддержку драйвера FDDI и делает доступным список интерфейсных карт FDDI, которые на рис. 5.2 отображены серым цветом. Часто, чтобы выбрать поддержку определенного адаптера, следует прежде включить поддержку интерфейса в целом.

n Выбор значения *n* предписывает ядру не использовать параметр настройки.

Каждому параметру настройки соответствует кнопка *Help*. Нажатие на кнопку *Help* приводит к получению дополнительной информации по параметру и сведений о том, в каких обстоятельствах его следует применять. Прежде чем изменять стандартные значения, следует обращаться к встроенной справке, даже если вы считаете, что полностью понимаете смысл параметров.

Два пункта на рис. 5.2, *Ethernet* (*10 or 100 Mbit*) и *Ethernet* (*1000 Mbit*), приводят к открытию самостоятельных окон с объемными меню – Linux поддер-

живает очень широкий спектр устройств Ethernet. Для карт Ethernet, доступных в дополнительных окнах, действуют уже описанные значения параметров *y*, *m* и *n*.

Окно *Network device support* и окна для карт Ethernet показывают, что поддержка конкретных адаптеров может встраиваться в ядро, но это не обязательный вариант. Как мы видели в предшествующем разделе, посвященном динамически загружаемым модулям, сетевые интерфейсы обычно управляются загружаемыми модулями. Всем системам Linux нужны сетевые интерфейсы для работы с TCP/IP, но поддержку интерфейсов не обязательно встраивать в ядро.

Выбор пункта *Networking options* из главного меню (см. рис. 5.1) приводит к открытию окна *Network options*, которое содержит более 60 пунктов меню – Linux поддерживает широкий спектр сетевых служб. Некоторые из служб являются экспериментальными, а некоторые относятся к протоколам, отличным от IPv4. Здесь мы ограничимся рассмотрением параметров, которые непосредственно относятся к IPv4. Тем не менее параметров все равно довольно много. Вот они:

Packet socket

Данная служба позволяет приложениям общаться с сетевыми устройствами напрямую. Это необходимо таким приложениям, как `tcpdump`, выполняющим перехват и фильтрацию пакетов. При включенном параметре *Packet socket* можно дополнительно выбрать *Packet socket: mmapped IO*, что позволит использовать ввод-вывод, отображаемый в память, для службы пакетных сокетов. Служба пакетных сокетов, как правило, включается, а поддержка отображаемого в память ввода-вывода остается отключенной.

Kernel/User netlink socket

Данная служба обеспечивает взаимодействие ядра и пользовательских приложений. Выбор данного параметра позволяет включить также параметры *Routing messages* и *Netlink device emulation*. Сокеты Netlink позволяют пользовательским приложениям взаимодействовать с маршрутизацией IPv4, таблицами ARP и кодом брандмауэра ядра.

Network packet filtering

Данная служба реализует фильтрацию IP-пакетов, необходимую для построения брандмауэра или сервера преобразования адресов. Включение параметра *Network packet filtering* позволяет включить также параметр *Network packet filtering debugging*. Фильтрация сетевых пакетов обычно включается для маршрутизаторов и отключается для обычных узлов, хотя она может использоваться для повышения защищенности сервера, как описано в главе 12 (`iptables`).

TCP/IP networking

Выбор данного параметра приводит к встраиванию в ядро поддержки для TCP/IP, то есть всех базовых транспортных протоколов TCP/IP и функцио-

нальности для работы с дейтаграммами. Включение параметра *TCP/IP networking* открывает доступ ко многим дополнительным службам TCP/IP:

IP: multicasting

Поддержка многоадресной передачи. Многоадресная передача описана в главе 2.

IP: advanced router

Выбор этого пункта меню открывает доступ к ряду параметров, позволяющих настроить ядро на работу со сложными протоколами маршрутизации. Поддержка сложной маршрутизации не требуется для маршрутизации в простейших случаях, в особенности для узлов и внутренних маршрутизаторов небольших сетей. Сложная маршрутизация применяется только в случаях, когда система Linux используется в качестве основного маршрутизатора либо внешнего маршрутизатора, связующего автономные системы. В главе 7 описано применение `gated` для работы со сложными протоколами маршрутизации в системах Unix. Дополнительные параметры, доступные в этом разделе:

IP: policy routing включает поддержку правил маршрутизации на уровне ядра, механизм, упоминаемый в главе 7 в связи с протоколом маршрутизации BGP, а также в главе 2 в связи с базой данных правил маршрутизации PRDB (Policy Routing Database). Присутствие этого параметра не требуется для работы `gated`, поскольку `gated` реализует поддержку правил маршрутизации на пользовательском уровне.

IP: equal cost multipath включает поддержку использования нескольких маршрутов к одному пункту назначения. Многолучевая маршрутизация упоминается в главе 7 в связи с протоколом маршрутизации OSPF.

IP use TOS value as routing key включает переключение тегов с применением поля IP-заголовка Type of Service (TOS) для хранения тега. Поле тега может использоваться как в OSPF, так и в RIP версии 2. В приложении B затронут синтаксис `gated`, используемый для полей тегов.

IP: verbose route monitoring увеличивает число и длину сообщений, сопровождающих обновления таблицы маршрутизации.

IP: large routing tables увеличивает объем памяти, отводимой под хранение таблицы маршрутизации.

IP: kernel level autoconfiguration

Эта служба используется бездисковыми клиентами. Выбор этого параметра открывает доступ к опциям *IP: BOOTP support* и *IP: RARP support*, которые определяют источник информации настройки – BOOTP или RARP. Описание протоколов BOOTP и RARP приводится в главе 3.

IP: tunneling

Данная служба инкапсулирует дейтаграммы IPv4 в туннели IP, что позволяет узлу создавать видимость принадлежности к сети, в которую он на самом деле физически не входит. Эта служба иногда используется на портативных компьютерах в целях обеспечения мобильности.

IP: GRE tunnels over IP

Включает поддержку протокола GRE (Generic Routing Encapsulation), позволяющего передавать дейтаграммы IPv4 и IPv6 по туннелям IPv4. Выбор этого параметра открывает доступ к параметру *IP: broadcast GRE over IP*, обеспечивающему многоадресную передачу по туннелям. GRE является предпочтительным протоколом инкапсуляции при работе с маршрутизаторами Cisco.

IP: multicast routing

Включает поддержку многоадресной маршрутизации. Параметр необходим только в случае, когда система выступает в роли многоадресного маршрутизатора, то есть использует в работе `mrouted`. Выбор параметра открывает доступ к параметрам *IP: PIM-SM version 1 support* и *IP: PIM-SM version 2 support*, определяющим уровень применяемого в системе протокола PIM-SM.

IP: TCP Explicit Congestion Notification support

Включает поддержку явных уведомлений о перегрузке (Explicit Congestion Notification, ECN). Сообщения ECN передаются маршрутизатором клиенту, чтобы уведомить о перегруженности. Параметр включается только для Linux-систем, выступающих в качестве маршрутизаторов. Поскольку многие брандмауэры несовместимы с ECN, не рекомендуется включать этот параметр в настройку.

IP: TCP syncookie support

Включает поддержку SYN-квитанций (cookies), которые применяются для противодействия атаке *SYN flooding* (ее еще называют синхронизирующей «Denial-Of-Service»-атакой или «лавинной» SYN-атакой).

IP: Netfilter Configuration

Выбор этого пункта меню приводит к открытию окна, позволяющего выбрать ряд служб для брандмауэра ядра Netfilter. В главе 12 описано применение службы Netfilter посредством `iptables`.

QoS and/or fair queueing

Здесь представлены параметры, изменяющие способ обработки сетевых пакетов сервером. Механизм является экспериментальным, так что для рабочего сервера параметр должен иметь значение *n*. Администрирование дополнительных обработчиков пакетов требует присутствия специального программного обеспечения.

Завершив настройку сетевых параметров, наберите `make dep; make clean`, чтобы создать зависимости и подготовить код к компиляции. Когда выполнение команд завершится, скомпилируйте ядро. Команда `make bzImage` создает сжатое ядро и сохраняет его в каталоге */usr/src/linux/i386/boot*.[1] Убедив-

[1] В большинстве систем Linux используется упакованное ядро, которое подвергается автоматической распаковке при загрузке.

шись, что новое ядро готово к работе, просто скопируйте файл ядра, *bzImage*, в файл *vmlinuz*, с которого начинается загрузка системы.[1]

Перечень сетевых настроек Linux весьма объемен.[2] Linux – это инь для ян Solaris: Linux позволяет системному администратору настраивать все, тогда как Solaris настраивает все за администратора. Настройка ядра BSD находится где-то между этими двумя крайностями.

Файл настройки ядра BSD

Как и в случае Linux, ядро BSD Unix является программой на языке C, компилируемой и устанавливаемой инструментом `make`. Команда `config` читает файл настройки ядра и генерирует файлы (включая файл сборки, Makefile), необходимые для компиляции и сборки ядра. В системе FreeBSD файл настройки ядра расположен в каталоге */usr/src/sys/i386/conf*.[3]

Крупный файл настройки ядра *GENERIC* поставляется в составе системы FreeBSD. Он настраивает все стандартные устройства системы, включая все необходимое для работы TCP/IP. В этом разделе мы рассмотрим лишь те элементы файла *GENERIC*, которые имеют отношение к TCP/IP. Работа основных служб TCP/IP не требует внесения изменений в ядро *GENERIC*. Причины изменения ядра BSD те же, что и для ядра Linux: получить ядро меньшего размера и более эффективное либо добавить новые возможности.

Не существует стандартного имени для файла настройки ядра BSD. Создавая файл настройки, выберите любое имя. По существующему соглашению, имена файлов настройки ядра BSD состоят из заглавных букв. Чтобы создать новый файл, скопируйте файл *GENERIC* и отредактируйте результат. Следующие команды создают новый файл настройки с именем *FILBERT*:

```
# cd /usr/src/sys/i386/conf
# cp GENERIC FILBERT
```

Если ядро системы было изменено, в каталоге */usr/src/sys/i386/conf* должен находиться новый файл настройки, созданный администратором. Файл настройки ядра содержит многочисленные команды, охватывающие все аспекты настройки системы. Здесь рассмотрены только параметры, непосредственно влияющие на работу TCP/IP. В поисках информации по другим командам настройки обращайтесь к документации, поставляемой в составе системы FreeBSD.[4]

---

1 Имя и местонахождение ядра не играют никакой роли, если загрузчик знает, где его найти; кроме того, всегда рекомендуется сохранить возможность загрузки старого ядра на случай неполадок с новым. – *Примеч. науч. ред.*

2 Перечень не просто объемный, он постоянно изменяется. Всегда сверяйтесь с документацией по системе, прежде чем начинать настройку ядра.

3 */usr/src/sys* является символической ссылкой на */sys*. Мы используем */usr/src/sys* только в качестве примера. В вашей системе это может быть совсем другой каталог.

4 Хороший источник информации по перекомпиляции ядра BSD – книга Грега Лехи (Greg Lehey) «The Complete FreeBSD», издательство Walnut Creek CDROM.

TCP/IP в ядре BSD

Для сетевого администратора больший интерес представляют операторы, необходимые для настройки TCP/IP, а не глубокое понимание структуры каждого из операторов. Для настройки TCP/IP в ядре BSD служат три типа операторов: options, pseudo-device и device.

Оператор options

Оператор options предписывает включить в ядро определенный программный параметр. Наиболее важный для TCP/IP оператор options выглядит так:

```
options INET # basic networking support--mandatory
```

В файле настройки ядра каждой BSD-системы, работающей с TCP/IP, присутствует оператор `options INET`. Оператор приводит к созданию аргумента – `DINET` для компилятора, и, как следствие, к встраиванию в ядро модулей IP, ICMP, TCP, UDP и ARP. Один этот оператор включает в систему поддержку основных транспортных служб и служб дейтаграмм IP. Никогда не удаляйте этот оператор из файла настройки.

```
options ICMP_BANDLIM #Rate limit bad replies
```

Данный параметр ограничивает использование ICMP-сообщениями об ошибках пропускающей полосы канала. Используйте этот параметр для защиты системы от атак, основанных на перегрузке сети сообщениями об ошибках.

```
options "TCP_COMPAT_43" # Compatible with BSD 4.3 [KEEP THIS!]
```

Последний параметр предотвращает зависание соединений между системами BSD 4.3 и FreeBSD, настраивая FreeBSD на игнорирование ошибок BSD 4.3. Кроме того, установка этого параметра позволяет избежать некорректной работы отдельных приложений. Учитывая все сказанное, имеет смысл оставить параметр в таком виде.

Оператор pseudo-device

Второй тип оператора, необходимый для настройки TCP/IP во всех вариантах BSD, – оператор pseudo-device. *Псевдоустройство* – это драйвер устройства, не связанный напрямую с реально существующим устройством. Оператор pseudo-device создает заголовочный файл (*.h*) с именем псевдоустройства в каталоге ядра. К примеру, следующий оператор приведет к созданию файла *loop.h*:

```
pseudo-device   loop            # loopback network--mandatory
```

Псевдоустройство loop необходимо для создания кольцевого устройства lo0. Это устройство связано с кольцевым адресом 127.0.0.1; оно определено в качестве псевдоустройства потому, что в действительности устройством не является.

Другое псевдоустройство, применяемое во многих системах TCP/IP FreeBSD:

```
pseudo-device   ether           # basic Ethernet support
```

Этот оператор необходим для поддержки устройств Ethernet. Полноценная поддержка ARP и других специальных функций Ethernet основана на псевдоустройстве ether. И хотя данный оператор может оказаться ненужным в случае системы, не имеющей интерфейсов Ethernet, обычно он присутствует в числе прочих настроек ядра.

Дополнительные распространенные псевдоустройства TCP/IP выполняют поддержку SLIP и PPP.

```
pseudo-device   sl        2     # Serial Line IP
```

Приведенный оператор определяет интерфейс для протокола SLIP (Serial Line IP). Число 2 в данном примере задает количество псевдоустройств SLIP, создаваемых ядром. Два созданных устройства получат имена sl0 и sl1.

```
pseudo-device   ppp       2     # Point-to-point protocol
```

Псевдоустройство ppp служит интерфейсом для протокола Point-to-Point. Число 2 в данном примере задает количество псевдоустройств PPP, создаваемых ядром. Два созданных устройства получат имена ppp0 и ppp1. И еще одно псевдоустройство, непосредственно связанное с PPP.

```
pseudo-device   tun         1   # Tunnel driver(user process ppp)
```

Псевдоустройство tun – это туннельный драйвер, используемый прикладными программами PPP. *Туннелирование* – это передача данных одного протокола посредством другого; tun в системе FreeBSD позволяет выполнять туннеллирование по соединениям PPP. Число 1 в данном примере определяет количество туннелей, поддерживаемых ядром.

Следующее псевдоустройство служит для отладки и тестирования.

```
pseudo-device   bpfilter   4   # Berkeley packet filter
```

Оператор bpfilter добавляет функциональность, необходимую для перехвата пакетов. Перехват пакетов – жизненная необходимость для анализаторов протоколов, таких как `tcpdump` (см. главу 13). Когда оператор bpfilter включается в ядро BSD, интерфейсы Ethernet получают возможность работать в *беспорядочном режиме* (*promiscuous mode*).[1] Интерфейс, работающий в беспорядочном режиме, передает все пакеты (а не только адресованные локальной системе) программам, расположенным уровнем выше. Эта возможность полезна для системных администраторов, занятых отладкой сети, но может использоваться и злоумышленниками для перехвата паролей и нарушения защиты системы. Используйте псевдоустройство bpfilter только по необхо-

[1] Предполагается, что устройство Ethernet способно функционировать в беспорядочном режиме. Не все карты Ethernet поддерживают эту возможность.

димости. Число 4 в данном примере определяет максимальное число интерфейсов Ethernet, которые могут находиться под наблюдением bpfilter.

Оператор device

Реально существующие устройства описываются при помощи оператора device. Каждый узел, подключенный к сети TCP/IP, должен иметь физическое аппаратное обеспечение, реализующее подключение. Это аппаратное обеспечение объявляется оператором device в файле настройки ядра. Существует большое число сетевых интерфейсов для работы с TCP/IP, но самое широкое распространение получили интерфейсы Ethernet. Вот операторы device интерфейсов Ethernet, определенные для ядра *GENERIC*:

```
device de     # DEC/Intel DC21x4x (``Tulip'')
device fxp    # Intel EtherExpress PRO/100B (82557, 82558)
device tx     # SMC 9432TX (83c170 ``EPIC'')
device vx     # 3Com 3c590, 3c595 (``Vortex'')
device wx     # Intel Gigabit Ethernet Card (``Wiseman'')
device dc     # DEC/Intel 21143 and various workalikes
device rl     # RealTek 8129/8139
device sf     # Adaptec AIC-6915 (``Starfire'')
device sis    # Silicon Integrated Systems SiS 900/SiS 7016
device ste    # Sundance ST201 (D-Link DFE-550TX)
device tl     # Texas Instruments ThunderLAN
device vr     # VIA Rhine, Rhine II
device wb     # Winbond W89C840F
device xl     # 3Com 3c90x (``Boomerang'', ``Cyclone'')
device ed0 at isa? port 0x280 irq 10 iomem 0xd8000
device ex
device ep
device wi      # WaveLAN/IEEE 802.11 wireless NIC
device an      # Aironet 4500/4800 802.11 wireless NICs
device ie0 at isa? port 0x300 irq 10 iomem 0xd0000
device fe0 at isa? port 0x300
device le0 at isa? port 0x300 irq 5 iomem 0xd0000
device lnc0 at isa? port 0x280 irq 10 drq 0
device cs0 at isa? port 0x300
device sn0 at isa? port 0x300 irq 10
```

Оператор device, позволяющий выполнять настройку интерфейса Ethernet в ядре FreeBSD, существует в двух основных формах:

```
device ed0 at isa? port 0x280 net irq 10 iomem 0xd8000
device de0
```

Формат варьируется в зависимости от типа устройства – ISA или PCI. Оператор device для устройства ed0 определяет тип шины (isa), базовый адрес ввода-вывода (port 0x280), номер прерывания (irq 10) и адрес в памяти (iomem 0xd8000). Эти значения должны соответствовать установленным на карте интерфейса и являются стандартными для настройки ISA-устройств. С другой стороны, оператор device для устройства de0 практически не требует на-

стройки, поскольку речь идет о карте, подключенной через шину PCI. Шина PCI достаточно разумна и позволяет получать настройки непосредственно от установленных устройств.

Ethernet – не единственный сетевой интерфейс TCP/IP, с которым способна работать система FreeBSD. Поддерживается и ряд других интерфейсов. Интерфейсы последовательных линий необходимы для работы SLIP и PPP:

```
device sio0 at isa? port IO_COM1 flags 0x10 irq 4
device sio1 at isa? port IO_COM2 irq 3
device sio2 at isa? disable port IO_COM3 irq 5
device sio3 at isa? disable port IO_COM4 irq 9
```

Четыре последовательных интерфейса, от sio0 до sio3, соответствуют интерфейсам MS-DOS от COM1 до COM4. Они нужны для работы по протоколам SLIP и PPP. В главе 6 рассмотрены прочие аспекты настройки PPP.

Оператор device изменяется от интерфейса к интерфейсу. Но как определить, какие аппаратные интерфейсы установлены в системе? Вспомните, что ядро *GENERIC*, поставляемое в составе системы FreeBSD, настроено на поддержку широкого спектра устройств. Простейший способ определить, какие аппаратные интерфейсы установлены в системе, – взглянуть на сообщения, отображаемые в процессе загрузки на консоли. Эти сообщения перечисляют все устройства, включая и сетевые, обнаруженные ядром при инициализации. Взгляните на вывод команды `dmesg`, которая повторяет сообщения, созданные в процессе последней загрузки. Чаще всего оказывается, что редактирование настроек ядра сводится к удалению поддержки ненужных и несуществующих устройств.

Операторы файла настройки options, pseudo-device и device предписывают системе включить в ядро поддержку определенных устройств и программ TCP/IP. Операторы вашего файла могут отличаться от тех, что мы видели в примерах, но основной их набор будет таким же. Этот базовый набор позволяет подготовить FreeBSD Unix к работе с TCP/IP.

Возможно, вам никогда не придется изменять переменные, о которых шла речь в этом разделе. Как и все прочие элементы файла настройки ядра, эти переменные по умолчанию имеют разумные значения, позволяющие корректно работать с TCP/IP. Однако вам часто придется принимать участие в управлении сетевыми службами, которые реализуются посредством TCP/IP. Мы переходим к тому, как запускать сетевые службы и управлять тем, какие из них запускаются.

Загрузочные файлы

Настройка ядра позволяет интегрировать в Unix основные транспортные службы и службы дейтаграмм IP. Но семейство протоколов TCP/IP вовсе не ограничивается базовыми службами. Каким же образом прочие протоколы настраиваются в среде Unix?

Некоторые протоколы запускаются из загрузочных файлов. Этот способ применяется, например, для протокола RIP (Routing Information Protocol) и системы доменных имен (DNS). Сетевые службы, требующие сложных процедур для запуска либо постоянно востребованные, обычно запускаются сценариями при загрузке системы и работают в качестве демонов, пока система не будет остановлена.

Все команды, пригодные для использования в командной строке интерпретатора команд, могут быть записаны в файл и работать в составе сценария. Эта возможность используется для автоматического запуска системных служб. Вызов загрузочных файлов может происходить по-разному, в зависимости от используемой модели загрузки, BSD или System V.

Модель BSD является более простой: некоторое количество стартовых сценариев выполняется в определенном порядке при каждой загрузке системы. В наиболее примитивном варианте три основных сценария, */etc/rc*, */etc/rc.boot* и */etc/rc.local* выполняются в таком порядке и обеспечивают инициализацию системы, инициализацию служб, а также вносят локальные коррективы. В системах BSD Unix сетевые службы, как правило, запускаются из файла */etc/rc.boot* или */etc/rc.local*.

Если в системе используется модель загрузки BSD, размещайте дополнительные команды настройки сети в сценарии *rc.local*. Он выполняется в конце процесса загрузки. Любые значения настройки, указанные в этом файле, замещают встреченные системой ранее.

Модель загрузки BSD применяется в системах BSD и SunOS. В Linux и Solaris применяется модель загрузки System V, в которой участвует гораздо более сложный набор стартовых файлов.[1] Здесь присутствуют целые каталоги сценариев, выполняемых процессом `init`, причем в зависимости от заданного уровня исполнения выполняются сценарии из различных каталогов.

Уровни исполнения

Чтобы понять загрузку по модели System V, необходимо сначала разобраться с уровнями исполнения, которые используются для определения состояния системы после завершения процесса `init`. Уровни исполнения никоим образом не связаны с аппаратным обеспечением системы, это исключительно программное понятие. `init` и */etc/inittab* – файл, используемый для настройки `init` – вот два элемента, реализующих влияние уровней исполнения на систему. Применение уровней исполнения мы продемонстрируем на примере системы Red Hat Linux.

Linux определяет ряд уровней исполнения, охватывающих полный спектр возможных состояний системы, от нерабочего (система остановлена) до многопользовательского режима:

---

[1] Хорошее описание лабиринта файлов инициализации System V приводится в книге Элин Фриш (Æleen Frisch) «Essential System Administration» (Основы системного администрирования), O'Reilly.

- Уровень исполнения 0 останавливает работу всех процессов, а затем и систему.
- Уровень 1 переводит систему в однопользовательский режим. Однопользовательский режим используется системным администратором для выполнения операций, которые невозможны в присутствии других работающих в системе пользователей. Этот уровень может обозначаться буквой S, а не цифрой 1. В Solaris однопользовательский режим обозначается буквой S.
- Уровень 2 – специальный многопользовательский режим, в котором не поддерживается совместный доступ к файлам.
- Уровень 3 – полноценная многопользовательская среда с поддержкой полного диапазона служб, включая совместный доступ к файлам по NFS. Это режим по умолчанию для систем Solaris.
- Уровень 4 не используется. Таким образом, существует возможность создать собственное состояние системы и реализовать его посредством уровня 4.
- Уровень 5 инициализирует систему в качестве выделенного терминала X Window. В системах Linux на этом уровне доступен вход в систему с консоли X Window. Когда система Linux загружается по уровню 3, она предоставляет пользователям для регистрации текстовую консоль. В Solaris этот уровень работы не используется. Переход на уровень 5 в Solaris приводит к останову системы.
- Уровень 6 останавливает работу всех процессов и перезагружает систему.

Теперь, когда ситуация с уровнями немного прояснилась, отметим, что в различных системах уровни исполнения трактуются по-разному. Это происходит потому, что уровни исполнения – программный компонент. Это аргументы загрузочных команд, предписывающие процессу `init` вызов определенных сценариев. Сценарии могут содержать любые допустимые команды. Отображение уровней `init` в загрузочные файлы происходит при помощи файла *inittab*.

Разбираемся с /etc/inittab

Любая строка файла *inittab*, которая начинается символом решетки (#), является комментарием. Разумное число комментариев не повредит, поскольку синтаксис строк файла *inittab* весьма насыщенный и загадочный. Запись *inittab* имеет следующий формат:

```
метка:уровень:действие:процесс
```

Поле *метка* имеет длину от одного до четырех символов и идентифицирует строку. Отдельные системы поддерживают только двухсимвольные метки, поэтому в большинстве случаев их длина ограничена двумя символами. Метки могут быть произвольной строкой; они не имеют специального значения.

Поле *уровень* определяет уровни исполнения, для которых действует эта запись. Если поле содержит цифру 3, процесс, на который указывает запись, должен быть запущен при инициализации системы по уровню исполнения 3.

Можно указывать несколько уровней одновременно. Записи с пустым полем уровня не участвуют в инициализации конкретных уровней. Например, в системах Linux существует запись *inittab*, связанная с обработкой комбинации из трех пальцев (<Ctrl>+<Alt>+<Del>); она имеет пустое поле уровня.

Поле *действие* определяет условия, при которых выполняется процесс. В табл. 5.1 перечислены значения действий, существующие в системах Red Hat, Mandrake и Caldera Linux.

Таблица 5.1. Значения действий для inittab в Linux

Действие	Значение
Boot	Выполняется при загрузке системы. Уровни не используются
Bootwait	Выполняется при загрузке системы; init ожидает завершения процесса. Уровни не используются
Ctrlaltdel	Выполняется по сочетанию клавиш <Ctrl>+<Alt>+<Del>, init получает сигнал SIGINT. Уровни не используются
Initdefault	Отсутствует выполняемый процесс. Действие устанавливает уровень исполнения по умолчанию
Kbrequest	Выполняется, когда init получает сигнал с клавиатуры. Сочетание клавиш должно быть связано с клавиатурным сигналом (KeyBoardSignal)
Off	Отключает запись, блокирует выполнение процесса
Once	Выполняется единожды для каждого уровня исполнения
Ondemand	Выполняется, когда система переходит на один из специальных уровней, A, B или C
Powerfail	Выполняется, когда init получает сигнал SIGPWR
Powerokwait	Выполняется, когда init получает сигнал SIGPWR и файл */etc/powerstatus* содержит слово OK
Powerwait	Выполняется, когда init получает сигнал SIGPWR; init ожидает завершения процесса
Respawn	Перезапускает процесс после завершения
sysinit	Выполняется до всех процессов boot и bootwait
wait	Выполняет процесс при переходе в рабочий режим, init ожидает завершения процесса

И последнее поле записи – *процесс*. Оно указывает процесс, запускаемый `init`. Процесс имеет формат команды, выполняемой в командной строке. Таким образом, поле процесса начинается с имени программы, которую следует выполнить. За именем следуют аргументы, передаваемые процессу. Например, `/sbin/shutdown -t3 -r now`, процесс, выполняемый отдельными системами Linux по сочетанию клавиш <Ctrl>+<Alt>+<Del>, – это команда, которую можно набрать в командной строке интерпретатора с целью перезагрузки системы. В большинстве записей *inittab* поле процесса содер-

жит имя стартового сценария. Существует два главных типа загрузочных сценариев: сценарии инициализации системы и сценарии инициализации уровней исполнения. В приведенном ниже фрагменте файла *inittab* системы Red Hat Linux отражены оба типа:

```
# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6
```

Эти семь строк – сердцевина файла *inittab*, они вызывают загрузочные сценарии. Первая строка предписывает `init` выполнить загрузочный сценарий */etc/rc.d/rc.sysinit* для инициализации системы. В этой записи отсутствует значение уровня исполнения. Сценарий выполняется при каждой загрузке системы. Сценарий инициализации системы выполняет ряд важных задач. Так, сценарий *rc.sysinit* системы Red Hat:

- инициализирует пространство подкачки;
- выполняет проверку файловой системы;
- монтирует файловую систему */proc*;
- монтирует корневую файловую систему в режиме чтения-записи после завершения работы `fsck`;
- загружает модули ядра.

Сценарии инициализации других систем могут выглядеть иначе, чем в Red Hat, но они решают очень похожие задачи. Например, процесс инициализации в системе Caldera начинается с загрузки модулей ядра. Затем происходит активизация пространства подкачки, проверка файловой системы и перемонтирование корневой файловой системы в режиме чтения-записи. Порядок иной, но основная функциональность не изменилась.

Выполнив сценарий инициализации системы, `init` вызывает сценарий для конкретного уровня исполнения. Остальные шесть строк примера используются для вызова загрузочных сценариев отдельных уровней. За исключением поля уровня исполнения, все эти строки идентичны.

Для примера рассмотрим строку с меткой `l3`. Данная строка запускает все процессы и службы, необходимые для полноценного многопользовательского режима. Уровень исполнения системы – 3. Действие `wait` предписывает программе `init` ожидать завершения загрузочного сценария, прежде чем переходить к прочим записям файла *inittab*, относящимся к уровню 3. `init` выполняет сценарий */etc/rc.d/rc* и передает ему аргумент командной строки – 3.

Управляющий сценарий, */etc/rc.d/rc*, в свою очередь вызывает все сценарии, соответствующие уровню исполнения, то есть все сценарии из катало-

га /etc/rc*n*.d, где *n* – указанный уровень. В нашем примере управляющий сценарий *rc* получает значение 3, а потому выполняет сценарии из каталога */etc/rc.d/rc3.d*. Просмотр каталога системы Red Hat показывает, что сценариев много:

```
$ ls /etc/rc.d
init.d  rc0.d  rc2.d  rc4.d  rc6.d     rc.sysinit
rc      rc1.d  rc3.d  rc5.d  rc.local
$ ls /etc/rc.d/rc3.d
K03rhnsd      K35smb       K74ntpd    S05kudzu     S25netfs       S85httpd
K16rarpd      K45arpwatch  K74ypserv  S06reconfig  S26apmd        S90crond
K20nfs        K45named     K74ypxfrd  S08ipchains  S28autofs      S90xfs
K20rstatd     K50snmpd     K75gated   S09isdn      S40atd         S95anacron
K20rusersd    K50tux       K84bgpd    S10network   S55sshd        S99linuxconf
K20rwalld     K55routed    K84ospf6d  S12syslog    S56rawdevices  S99local
K20rwhod      K61ldap      K84ospfd   S13portmap   S56xinetd
K28amd        K65identd    K84ripd    S14nfslock   S60lpd
K34yppasswdd  K73ypbind    K84ripngd  S17keytable  S80sendmail
K35dhcpd      K74nscd      K85zebra   S20random    S85gpm
```

Сценарии, имена которых начинаются с буквы K, используются для завершения процессов при завершении определенного уровня исполнения. В приведенном примере K-сценарии будут использованы в процессе завершения работы на уровне 3. Сценарии, имена которых начинаются с буквы S, используются при переходе на уровень 3. Однако ни один из файлов, хранимых в *rc3.d*, не является в действительности сценарием. Это логические ссылки на настоящие сценарии, расположенные в каталоге */etc/rc.d/init.d*. Так, *S80sendmail* – ссылка на файл *init.d/sendmail*. Сразу возникает вопрос – зачем выполнять сценарии из каталога *rc3.d*, когда можно вызвать их напрямую из *init.d*, где они на самом деле хранятся? Причины очень просты. Одни и те же сценарии используются в работе нескольких уровней. Логические ссылки позволяют хранить сценарии в одном месте и использовать их из каталога каждого конкретного уровня исполнения.

Сценарии исполняются в алфавитном порядке. *S10network* исполняется раньше, чем *S80sendmail*. Таким образом, можно контролировать порядок выполнения сценариев, изменяя их имена. Порядок вызова сценариев может быть различным для различных уровней исполнения, при том, что действительные сценарии в каталоге *init.d* будут по-прежнему иметь простые, осмысленные имена. Что и доказывает содержимое каталога *init.d*:

```
$ ls /etc/rc.d/init.d
amd       functions  kdcrotate  network  rarpd       rwalld    xfs
anacron   gated      keytable   nfs      rawdevices  rwhod     xinetd
apmd      gpm        killall    nfslock  reconfig    sendmail  ypbind
arpwatch  halt       kudzu      nscd     rhnsd       single    yppasswdd
atd       httpd      ldap       ntpd     ripd        smb       ypserv
autofs    identd     linuxconf  ospf6d   ripngd      snmpd     ypxfrd
bgpd      ipchains   lpd        ospfd    routed      sshd      zebra
crond     iptables   named      portmap  rstatd      syslog
dhcpd     isdn       netfs      random   rusersd     tux
```

Можно разместить специальную команду настройки прямо в соответствующем сценарии из каталога *init.d*. Более приемлемый вариант для системы Red Hat – размещать все подобные команды в сценарии *rc.local*.

Подобно системам BSD, системы Linux для внесения локальных изменений предоставляют файл *rc.local*. Общее правило: не вносите поправки в загрузочные сценарии. Исключением из этого правила является файл *rc.local*, расположенный в каталоге */etc/rc.d*. Это единственный стартовый файл, подлежащий правке, он зарезервирован для этой цели и может иметь любое содержание. После выполнения сценария инициализации системы сценарии уровней работы вызываются в алфавитном порядке. Последним из них является *S99local*, ссылка на *rc.local*. Поскольку сценарий *rc.local* вызывается в последнюю очередь, определенные в нем значения замещают встреченные системой ранее.

В системе Solaris также применяется модель загрузки System V, но дела обстоят несколько сложнее, чем в Linux. Во-первых, отсутствует сценарий *rc.local*. Чтобы воспользоваться таким сценарием, следует создать соответствующие файлы в каталогах уровней исполнения. Во-вторых, в системе Solaris не так много логических ссылок в каталогах уровней. Как следствие, отсутствует возможность централизованно изменять сценарии, задействованные на всех уровнях работы. Кроме того, каждому уровню исполнения соответствует отдельный управляющий сценарий, что может вносить различия в стартовый процесс уровней. Например, сценарий */sbin/rc2* управляет уровнем работы 2, а */sbin/rc3* – управляющий сценарий для уровня 3. Все эти отличия делают процесс загрузки Solaris более сложным для анализа.

В системе Solaris 8 уровень исполнения 3 является уровнем по умолчанию для многопользовательской среды с доступом к сетевым службам. Управляющий сценарий */sbin/rc3* выполняет сценарии из каталога */etc/rc2.d*, а затем сценарии из каталога */etc/rc3.d*. Базовая настройка сети происходит в */etc/rc2.d* и реализована сценариями *S69inet* и *S72inetsvc*. Запуск сетевых служб выполняется рядом других сценариев, расположенных в каталогах */etc/rc2.d* и */etc/rc3.d*.

В целях диагностирования и отладки очень важно понимать, что и каким образом происходит в процессе загрузки системы. Если возникли проблемы при инициализации сети, полезно знать, с чего начинать поиск. Однако в настройке сети следует придерживаться использования стандартных инструментов и методов данной конкретной системы. Прямое изменение загрузочных сценариев может вызывать проблемы при загрузке системы и наверняка запутает других людей, ассистирующих вам в сопровождении сети.

Разумеется, не все сетевые службы запускаются загрузочными сценариями. Большинство служб начинают работу по требованию. Наиболее распространенным инструментом для вызова сетевых служб по требованию является `inetd`, демон Интернета.

Демон Internet

Демон `inetd` (произносится «ай-нет-ди») запускается в процессе загрузки из файла инициализации, такого как */etc/rc2.d/S72inetsvc*. При запуске `inetd` обращается к своему файлу настройки, */etc/inetd.conf*. Файл настройки содержит имена служб, которые `inetd` слушает и запускает. Службы можно добавлять или удалять, внося изменения в файл *inetd.conf*.

Пример записи из такого файла для системы Solaris 8:

```
ftp  stream  tcp6  nowait  root  /usr/sbin/in.ftpd   in.ftpd
```

Поля записи файла *inetd.conf*, слева направо:

имя

Имя службы в том виде, в каком оно приводится в файле */etc/services*. В приведенном примере это поле имеет значение `ftp`.

тип

Тип службы доставки данных, или *тип сокета*. Распространенные типы сокетов:

stream

Служба потоковой доставки, предоставляемая TCP, то есть байтовый поток TCP.[1]

dgram

Служба доставки пакетов (дейтаграмм), предоставляемая UDP.

raw

Служба прямых дейтаграмм IP.

В примере используется потоковый сокет.

протокол

Имя протокола в том виде, в каком оно приводится в файле */etc/protocols*. Обычно принимает значение «tcp» или «udp». Значения «tcp6» и «udp6» в Solaris позволяют отразить тот факт, что служба способна работать как по протоколу IPv4, так и по протоколу IPv6. Протоколом транспортного уровня для FTP является TCP, поэтому наш пример содержит в поле протокола значение `tcp6`.

wait-status

Поле может иметь значение «wait» либо «nowait.» Обычно, но не всегда, серверы, использующие для доставки данных службы дейтаграмм, требуют значения «wait», а потоковые серверы требуют значения «nowait». Состояние «wait» предписывает inetd ожидать освобождения сокета сервером, прежде чем начать прием последующих запросов через этот сокет. Состояние «nowait» позволяет inetd немедленно начать принимать до-

[1] Речь идет о сокетах TCP/IP и потоках TCP, не о потоковом вводе-выводе AT&T или вводе-выводе через сокеты BSD.

полнительные запросы соединений, поступающие через сокет. Серверы с состоянием «nowait» задействуют для работы сокеты, отличные от тех, по которым получен запрос соединений, то есть используют динамически выделяемые сокеты.

uid

Значение uid – это имя пользователя, с полномочиями которого работает сервер. Допустимо имя любого существующего пользователя, но обычно это имя *root*. Есть и ряд исключений. Например, в стандартном варианте настройки Solaris 8 служба `finger` и сервер шрифтов `fs` (Sun Font Server) работают с полномочиями пользователя *nobody*, из соображений безопасности.

сервер

Абсолютное имя программы сервера, исполняемой демоном `inetd`. Поскольку наш пример относится к системе Solaris, использовано имя */usr/sbin/in.ftpd*. Имена и пути могут варьироваться в различных системах. Демон inetd способен самостоятельно обслуживать запросы к отдельным нетребовательным к ресурсам службам. Это более эффективно, чем запускать самостоятельные внешние серверы. Чтобы задействовать внутреннюю службу `inetd`, достаточно указать ключевое слово «internal» в поле имени сервера.

аргументы

Произвольные аргументы командной строки, передаваемые при вызове программе сервера. Перечень всегда начинается с `argv[0]` (то есть с имени вызываемой программы). Руководство по каждой конкретной программе документирует допустимые аргументы командной строки. В приведенном примере передается только имя программы, `in.ftpd`.

Существует ряд ситуаций, требующих правки файла *inetd.conf*. Например, возникла необходимость заблокировать существующую службу. Настройки по умолчанию содержат информацию о полном наборе серверов. Далеко не все эти серверы требуются на средней системе, а из соображений безопасности имеет смысл отключать невостребованные службы на некоторых машинах. Чтобы заблокировать службу, добавьте символ # в начало соответствующей записи (строка становится комментарием) и передайте сигнал принудительного завершения серверу `inetd`. Получив сигнал, демон `inetd` повторно прочитает файл настройки, и новые указания вступят в действие немедленно.

Кроме того, может возникнуть необходимость создать новые службы. Некоторые примеры этого мы еще встретим в последующих главах. Рассмотрим в подробностях процесс восстановления работы службы, которая была заблокирована. Начнем с некоторых записей и комментариев из файла Solaris */etc/inetd.conf*:

```
# Tftp service is provided primarily for booting.  Most sites run this
# only on machines acting as "boot servers."
#
#tftp  dgram  udp6 wait   root   /usr/sbin/in.tftpd  in.tftpd -s /tftpboot
```

```
#
# Finger, systat and netstat give out user information which may be
# valuable to potential "system crackers."  Many sites choose to disable
# some or all of these services to improve security.
#
finger stream tcp6 nowait nobody /usr/sbin/in.fingerd  in.fingerd
```

В этой части файла отражены две службы TCP/IP. Одна из них, `tftp`, закомментирована. Протокол TFTP – это специальный вариант протокола FTP, позволяющий выполнять передачу файлов без проверки имени пользователя и пароля. Протокол, таким образом, представляет потенциальную угрозу безопасности системы и часто блокируется в файле *inetd.conf*. Вторая служба – `finger`, комментарии для которой предполагают, что ее имеет смысл заблокировать.

В качестве примера изменения файла *inetd.conf* мы перенастроим систему на предоставление службы `tftp`, которая бывает необходима для поддержки бездисковых устройств. Во-первых, удалим символ комментария из записи `tftp` при помощи любимого текстового редактора. (В этом примере используется `sed`, любимый редактор всех пользователей!) Затем определим идентификатор процесса `inetd` и передадим ему сигнал SIGHUP. Следующие команды решают поставленную задачу:

```
# cd /etc
# mv inetd.conf inetd.conf.org
# cat inetd.conf.org | sed s/#tftp/tftp/ > inetd.conf
# ps -acx | grep inetd
  144 ?  I     0:12 inetd
# kill -HUP 144
```

В некоторых случаях может возникнуть необходимость в изменении имени сервера или аргументов его вызова. Для примера снова обратимся к записи `tftp`. Строка содержит аргументы командной строки, передаваемые `tftp`-серверу при вызове. Ключ `-s /tftpboot` закрывает самую очевидную прореху в безопасности `tftp`: запрещает пользователям получать файлы, расположенные за пределами каталога, указанного в качестве аргумента ключа `-s`. Для того чтобы воспользоваться другим каталогом, потребуется изменить файл *inetd.conf*. Серверам, вызываемым `inetd`, передаются только те аргументы командной строки, которые указаны в файле *inetd.conf*.

Расширенный демон Internet

Альтернативой `inetd` является расширенный демон Интернета – `xinetd` (Extended Internet Daemon). Настройки `xinetd` хранятся в файле */etc/xinetd.conf*, который для `xinetd` имеет такое же значение, как *inetd.conf* для `inetd`. Однако вместо позиционных параметров, значение которых определяется порядком следования в записи, в *xinetd.conf* применяются пары атрибутов и значений. Имя атрибута четко определяет смысл каждого параметра. Значение выполняет настройку параметра. Например, третье поле записи файла *in-*

etd.conf содержит название транспортного протокола. В файле *xinetd.conf* имя транспортного протокола определяется при помощи атрибута *protocol*. Например, `protocol = tcp`. Вот пример записи `tftp` в файле *xinetd.conf*:

```
# default: off
# description: The tftp server uses the trivial file transfer \
#       protocol.  The tftp protocol is often used to boot diskless \
#       workstations, download configuration files to network printers, \
#       and to start the installation process for some operating systems.
service tftp
{
        socket_type             = dgram
        protocol                = udp
        wait                    = yes
        user                    = root
        server                  = /usr/sbin/in.tftpd
        server_args             = -s /tftpboot
        disable                 = yes
}
```

Символ # отмечает строки-комментарии. Непосредственно запись начинается командой `service`. Атрибуты внутри фигурных скобок ({}) определяют характеристики конкретной службы.

Значения service, socket_type, protocol, wait, user, server и server_args соответствуют значениям, описанным в примере для файла *inetd.conf* системы Solaris. Их назначение в `xinetd` полностью соответствует назначению позиционных параметров `inetd`.

Одна из строк, `disable = yes`, требует некоторых объяснений. `disable = yes` запрещает `xinetd` запускать `tftp` по требованию. `disable = yes` является эквивалентом блокировки `tftp` в файле *inetd.conf* посредством комментария. Чтобы включить `tftp`, отредактируйте файл, изменив строку на `disable = no`.

В Red Hat 7 применяется `xinetd`. Однако в файле */etc/xinetd.conf* системы Red Hat вы не увидите перечисления сетевых служб. В системе Red Hat настройка `xinetd` выполнена таким образом, что файл *xinetd.conf* включает ссылки на все файлы каталога */etc/xinetd.d*. Приведенный выше фрагмент на самом деле отражает содержимое файла */etc/xinetd.d/tftp* системы Red Hat, использованной для создания примера. Каждой службе соответствует отдельный файл настройки.

Применение `xinetd` диктуется более совершенными механизмами обеспечения безопасности. Безопасность – одна из наиболее важных причин, по которым следует разобраться со строением файлов *inetd.conf* и *xinetd.conf*. Использование возможностей управления доступом демонов `xinetd` и `inetd` описано в главе 12.

Резюме

Базовые файлы настройки, файл настройки ядра, загрузочные файлы, файлы */etc/inetd.conf* и */etc/xinetd.conf* – необходимые элементы установки программного обеспечения TCP/IP в систему Unix. Ядра большинства систем изначально настроены на работы с TCP/IP. Архитектура некоторых систем, таких как Solaris, призвана исключить необходимость в настройке ядра, в то время как другие (Linux) поощряют подобные действия как позволяющие получить более эффективное ядро системы. В любом случае сетевой администратор должен знать, какие команды настраивают ядро на работу с TCP/IP, и не допустить, чтобы они были случайно удалены при пересборке ядра.

Сетевые службы запускаются в процессе загрузки системы из стартовых сценариев либо по требованию – демонами `xinetd` и `inetd`. В системах BSD присутствует небольшой набор загрузочных сценариев, выполняемых последовательно при каждой загрузке. Системы Unix System V содержат несколько наборов загрузочных сценариев – по одному на каждый уровень исполнения. Уровни исполнения позволяют запускать систему в различных режимах, например в однопользовательском или многопользовательском режиме. Схема загрузки System V используется как в Solaris, так и в Linux.

`inetd` и `xinetd` запускают жизненно необходимые сетевые службы. В большинстве систем Unix применяется `inetd`, а в некоторых, например в Red Hat Linux, применяется `xinetd`. Перенастройка `inetd` или `xinetd` позволяет добавлять новые службы и повысить безопасность системы. Безопасность повышается блокировкой невостребованных служб и ограничением доступа. Дополнительные сведения о применении `inetd` и `xinetd` для повышения безопасности системы приводятся в главе 12.

Настройки ядра определяют сетевой интерфейс. В следующей главе мы займемся настройкой интерфейса, обратившись к планам, составленным в главе 4.

- *Команда ifconfig*
- *TCP/IP и последовательные линии*
- *Установка PPP*

6

Настройка интерфейса

Когда сетевые протоколы взаимодействуют с однотипными физическими сетями, нет необходимости знакомить программы с интерфейсами. Программы сами знают, какие нужны интерфейсы, так что администратору здесь не остается поля для деятельности. Однако одной из сильных сторон TCP/IP является способность объединять физически различные сети. Такая гибкость усложняет задачу системного администратора: приходится объяснять TCP/IP, какими интерфейсами следует пользоваться, и определять характеристики каждого устройства.

Поскольку TCP/IP не зависит от физического строения сети, IP-адреса реализованы на уровне сетевого программного, а не аппаратного, обеспечения. В отличие от Ethernet-адресов, определяемых устройствами Ethernet, IP-адреса назначаются системным администратором всем сетевым интерфейсам.

В этой главе мы воспользуемся командой `ifconfig` (interface configure, настроить интерфейс), чтобы указать сетевой интерфейс протоколам TCP/IP и назначить этому интерфейсу IP-адрес, маску подсети, а также широковещательный адрес. Кроме того, мы выполним настройку сетевого интерфейса на работу с PPP (Point-to-Point Protocol, протокол точка-точка), стандартным протоколом уровня доступа к сети, применяемым для работы с TCP/IP по модемным соединениям.

На практике в процессе установки системный администратор абстрагируется от большинства деталей настройки сети. Программа установки запрашивает информацию, сохраняет эту информацию в файлах сценариев, а затем использует сценарии для настройки интерфейса при каждой загрузке. В этой главе мы заглянем глубже, чтобы понять, как в действительности работают подобные механизмы, и изучим работу команды `ifconfig`, а также сценариев, из которых она вызывается.

Команда ifconfig

Команда `ifconfig` позволяет устанавливать или определять значения настройки сетевых интерфейсов. Независимо от разработчика и версии Unix, команда `ifconfig` устанавливает IP-адрес, маску подсети и широковещательный адрес для каждого интерфейса. Самой существенной функцией является назначение IP-адреса.

Вот команда `ifconfig`, выполняющая настройку интерфейса Ethernet в системе Solaris:

```
# ifconfig dnet0 172.16.12.2 netmask 255.255.255.0 broadcast 172.16.12.255
```

Команда `ifconfig` может иметь и другие аргументы; некоторые из них мы обсудим позже. Основная же информация, необходимая для функционирования TCP/IP, одинакова для всех сетевых интерфейсов:

интерфейс

Имя сетевого интерфейса, подлежащего настройке для работы с TCP/IP. В приведенном примере это интерфейс Ethernet по имени dnet0.

адрес

IP-адрес, назначенный интерфейсу. Адрес может указываться в десятичной записи через точку или в виде имени узла. В последнем случае следует создать в файле */etc/hosts* запись для имени узла и соответствующего адреса. Система должна иметь возможность найти имя узла в */etc/hosts*, поскольку `ifconfig` обычно выполняется до запуска службы DNS. В примере использован численный IP-адрес 172.16.12.2.

netmask-маска

Адресная маска для интерфейса. Обойтись без этого аргумента можно только в том случае, если используется маска по умолчанию, получаемая из обычной классовой структуры адресов. Адресная маска для нашей воображаемой сети – 255.255.255.0, именно это значение и назначается интерфейсу dnet0. Адресные маски описаны в главах 2 и 4.

broadcast-адрес

Широковещательный адрес сети. В большинстве систем применяется стандартный широковещательный адрес, то есть IP-адрес, все биты раздела узла которого установлены в значение 1. В приведенном примере мы указываем широковещательный адрес явным образом (172.16.12.255), чтобы избежать путаницы, несмотря на то, что система Solaris 8 по умолчанию установит корректный широковещательный адрес. Все системы одной подсети должны пользоваться одним и тем же широковещательным адресом.

В приведенном примере мы воспользовались парами параметр/значение, чтобы упростить объяснение и сделать более прозрачным синтаксис. Однако Solaris не требует применения такого синтаксиса. Следующая (гораздо более краткая) команда делает то же самое, что и предыдущая:

```
# ifconfig dnet0 172.16.12.2/24
```

В этой команде маска сети определена при помощи адресного префикса, а широковещательный адрес определяется значением по умолчанию. Длина префикса 24 соответствует маске 255.255.255.0. Умолчание для широковещательного адреса при такой длине префикса – 172.16.12.255.

Значения адреса, маски подсети и широковещательного адреса поступают от администратора сети. Значения нашего примера взяты прямо из планов, разработанных в главе 4. Но имя интерфейса, выступающее в роли первого аргумента любой команды `ifconfig`, определяется системой во время загрузки.

Имя интерфейса

В главе 5 мы видели, что сетевых интерфейсов Ethernet существует великое множество и что различным картам Ethernet обычно соответствуют различные имена интерфейсов. Как правило, определить используемые имена интерфейсов можно, обратившись к отображаемым при загрузке системы сообщениям. Во многих системах изучить эти сообщения можно при помощи команды `dmesg`. В следующем примере показан вывод команды для двух различных систем:

```
$ dmesg | grep ether
Oct  1 13:07:23 crab gld: [ID 944156 kern.info] dnet0: DNET 21x4x:
 type "ether" mac address 00:00:c0:dd:d4:da

$ dmesg | grep eth
eth0: SMC EtherEZ at 0x240, 00 00 C0 9A 72 CA,assigned  IRQ 5 programmed-I/O mode.
```

Первая команда `dmesg` отображает сообщение, поступившее от системы Solaris 8 в момент, когда при загрузке был определен интерфейс Ethernet. Строка `type "ether"` делает очевидным тот факт, что dnet0 – интерфейс Ethernet. Кроме того, она содержит Ethernet-адрес (00:00:c0:dd:d4:da).

Второй пример `dmesg`, принадлежащий системе PC под управлением Linux, дает нам еще больше информации. В системах Linux имена интерфейсов Ethernet начинаются с подстроки «eth», поэтому мы ищем сообщение, содержащее такую подстроку. Сообщение системы Linux включает Ethernet-адрес (00:00:c0:9a:72:ca), информацию об изготовителе и модели (SMC EtherEZ) сетевой карты.

По выводу команды `dmesg` не всегда просто определить все доступные интерфейсы. Сообщения информируют лишь о физических интерфейсах обнаруженных в системе устройств. В архитектуре протоколов TCP/IP все функции, расположенные ниже уровня Internet, принадлежат к уровню доступа к сети. Этот уровень может объединять три нижних уровня стандартной модели OSI: физический, канальный и сетевой. Протоколу IP необходимы данные о конкретном интерфейсе уровня доступа к сети, чтобы передать пакеты, подлежащие доставке в определенную сеть. И этот интерфейс не ограничивается драйвером физического устройства. Он может быть представлен программным интерфейсом сетевого уровня другого семейства протоколов. Какие же методы могут применяться для определения доступных сетевых

интерфейсов? Можно воспользоваться командами `netstat` и `ifconfig`. Например, чтобы получить данные обо всех уже настроенных сетевых интерфейсах, наберите:

```
# netstat -in
Name  Mtu  Net/Dest      Address       Ipkts Ierrs Opkts Oerrs Collis Queue
lo0   8232 127.0.0.0     127.0.0.1     4504  0     4504  0     0      0
dnet0 1500 172.16.12.0   172.16.12.1   366   0     130   0     0      0
```

Ключ `-i` предписывает программе `netstat` отображать состояние всех настроенных сетевых интерфейсов, а ключ `-n` – использовать при выводе числовой формат. В приведенном примере для Solaris 8 команда `netstat -in` выводит следующие поля:

Name

Поле имени интерфейса отражает фактическое имя интерфейса. Именно это имя передается команде `ifconfig`. Символ звездочки (*) в этом поле показывает, что интерфейс не задействован, то есть «закрыт».

Mtu

Максимальный размер передаваемого блока (Maximum Transmission Unit) отражает максимальную длину фреймов (пакетов), которая не требует разбиения при передаче через этот интерфейс. Значение MTU отображается в байтах, и позже в этой главе мы поговорим о нем более подробно.

Net/Dest

Поле Network/Destination (Сеть/Пункт назначения) отражает сеть (или конечный узел), доступную посредством интерфейса. В наших примерах для Ethernet это поле содержит адрес сети. Адрес сети определяется по IP-адресу интерфейса и маске подсети. Если интерфейс настроен на работу с конкретным узлом по принципу точка-точка, поле содержит адрес узла. Конечный адрес – это адрес удаленного узла, то есть второй стороны канала точка-точка.[1] Канал точка-точка – прямое соединение между двумя компьютерами. Для создания подобного канала можно использовать команду `ifconfig`. Далее в тексте главы будет приведен соответствующий пример.

Address

Поле IP-адреса содержит Интернет-адрес, назначенный интерфейсу.

Ipkts

Поле Input Packets (Входящие пакеты) отражает число пакетов, полученных через этот интерфейс.

Ierrs

Поле Input Errors (Входящие ошибки) отражает число поврежденных пакетов, полученных через этот интерфейс.

---

[1] См. описание флага H в разделе «Таблица маршрутизации» главы 2.

Opkts

Поле Output Packets (Исходящие пакеты) отражает число пакетов, отправленных через этот интерфейс.

Oerrs

Поле Output Errors (Исходящие ошибки) отражает число пакетов, отправка которых привела к возникновению ошибок.

Collis

Поле Collisions отражает число конфликтов (collisions, коллизий) Ethernet, обнаруженных этим интерфейсом. Конфликты Ethernet – нормальное явление, вызываемое соревновательными особенностями трафика Ethernet. Для интерфейсов других типов это поле не имеет смысла.

Queue

Поле Packets Queued (Ожидающие пакеты) отражает число пакетов, помещенных в очередь и ожидающих передачи через интерфейс. Обычно поле имеет нулевое значение.

Вывод команды `netstat -in` в случае системы Linux внешне существенно отличается:

```
$ netstat -in
Kernel Interface table
Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR  TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0 1500   0  2234    280      0      0   1829      0      0      0 BRU
lo  16436   0    10      0      0      0     10      0      0      0 LRU
```

Внешние различия в данном случае – лишь повод вспомнить, что внешность бывает обманчива. Здесь мы видим имя интерфейса, значение MTU и статистику для пакетов.[1] RX-OK – общее число полученных пакетов, тогда как RX-ERR (ошибки), RX-DRP (отброшенные) и RX-OVR (переполнения) вместе дают общее число ошибок приема. Общее число отправленных пакетов – TX-OK, а счетчики TX-ERR, TX-DRP и TX-OVR в сумме дают общее число ошибок передачи. Из полей, представленных для системы Solaris, здесь отсутствуют лишь два: Net/Dest и Address. С другой стороны, вывод для Linux содержит два поля, которых не было в Solaris. Поле Met содержит метрику маршрутизации, назначенную интерфейсу. Поле Flg отражает флаги интерфейса:

- R означает, что интерфейс действует (running).
- U означает, что интерфейс функционален (up).
- B означает, что интерфейс может передавать широковещательные сообщения.
- L означает, что интерфейс является кольцевым (loopback).

Итак, мы видим, что у данной рабочей станции лишь два сетевых интерфейса. В данном случае можно легко понять, о каких интерфейсах речь. lo0 – это кольцевой интерфейс, существующий в каждой системе с TCP/IP. Имен-

[1] Статистика для пакетов, выводимая `netstat`, пригодится нам в главе 13.

но об этом кольцевом устройстве шла речь в главе 5. eth0 – интерфейс Ethernet, с которым мы тоже успели познакомиться.

В большинстве систем кольцевой интерфейс – часть стандартной настройки, так что нет необходимости его настраивать. А чтобы настроить интерфейс lo0 в системе Solaris, воспользуйтесь командой:

```
# ifconfig lo0 plumb 127.0.0.1 up
```

Данный пример верен только для Solaris, поскольку содержит параметр `plumb`. Этот параметр буквально предписывает «подвести трубы» к интерфейсу, что необходимо только при первой настройке. Последующая перенастройка интерфейса уже не требует параметр `plumb`, а в других системах, таких как Linux, этот параметр не используется.

Настройка интерфейса Ethernet требует более скрупулезного подхода. Во многих системах Unix существуют специальные сценарии установки. Сценарий запрашивает адрес узла, который используется для настройки интерфейса. Чуть позже мы изучим эти сценарии и возможные действия пользователя в ситуациях, когда сценарии установки не справляются с поставленной задачей.

Для обнаружения существующих в системе сетевых интерфейсов также может применяться команда `ifconfig`. Команда `netstat` перечисляет только настроенные интерфейсы. В некоторых системах команда `ifconfig` может использоваться для обнаружения всех интерфейсов, даже еще не прошедших настройку. В случае системы Solaris 8 результат достигается командой `ifconfig -a`; для системы Linux 2.0.0 перечень всех сетевых интерфейсов доступен по команде `ifconfig` без аргументов.

Несмотря на то что на большинстве узлов существует лишь один «физический» сетевой интерфейс, на некоторых узлах и на всех без исключения шлюзах интерфейсов несколько. Интерфейсы узла вполне могут иметь один тип: скажем, шлюз, соединяющий две Ethernet-сети, может иметь два интерфейса Ethernet. Команда `netstat` на подобном шлюзе может перечислить интерфейсы lo0, eth0 и eth1. Расшифровка вывода `netstat` для нескольких однотипных интерфейсов – задача предельно простая. Однако расшифровка для системы с многочисленными разнообразными типами сетевых интерфейсов может оказаться существенно сложнее. При выборе нужного интерфейса следует полагаться на документацию, поставляемую в составе дополнительного программного обеспечения. Прежде чем устанавливать новые сетевые программы, внимательно прочтите документацию.

Столь длинный рассказ, посвященный обнаружению сетевых интерфейсов, не должен затмить важности таких функций `ifconfig`, как назначение IP-адреса, маски подсети и широковещательного адреса. Так что мы возвращаемся к этим важным темам.

Проверка интерфейса посредством ifconfig

Как уже говорилось, сценарий установки Unix выполняет настройку сетевого интерфейса. Однако такая настройка не всегда соответствует потребнос-

тям пользователя. Проверка настроек интерфейса может быть выполнена при помощи ifconfig. Чтобы получить параметры, назначенные интерфейсу, наберите команду ifconfig с именем интерфейса в качестве единственного аргумента. Например, для интерфейса dnet0:

```
% ifconfig dnet0
dnet0: flags=1000843<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
inet 172.16.12.2 netmask ffff0000 broadcast 172.16.255.255
```

При проверке интерфейса в системе Solaris команда ifconfig выводит две строки. Первая строка содержит имя интерфейса, флаги характеристик интерфейса, а также значение максимального размера передаваемого блока (Maximum Transmission Unit, MTU) для интерфейса.[1] В этом примере имя интерфейса – dnet0, а значение MTU равно 1500 байт. Флаги отображаются одновременно в численном формате и в виде набора ключевых слов.

Флаги интерфейса имеют численное значение 1000843, которое интерпретируется следующим образом:

UP

Интерфейс функционален и может использоваться.

BROADCAST

Интерфейс поддерживает широковещательную передачу, то есть подключен к сети, поддерживающей широковещательную передачу (в данном случае Ethernet).

NOTRAILERS

Интерфейс не поддерживает инкапсуляцию завершителей (trailer encapsulation).[2]

RUNNING

Интерфейс действует.

MULTICAST

Интерфейс поддерживает многоадресную передачу.

IPv4

Интерфейс поддерживает TCP/IP версии 4, то есть стандартной версии TCP/IP, применяемой в Интернете и описанной в этой книге.

Во второй строке вывода ifconfig содержится информация, связанная непосредственно с TCP/IP. Ключевое слово inet предваряет интернет-адрес, назначенный данному интерфейсу. Ключевое слово netmask предваряет адрес-

[1] index – это характеристика интерфейса, существующая только в Solaris. Она представляет присвоенный интерфейсу внутренний уникальный номер, который не влияет на работу TCP/IP.

[2] Инкапсуляция завершителей – это оптимизация, основанная на изменении порядка передаваемых (получаемых) данных. См. RFC-893. – *Примеч. перев.*

ную маску в шестнадцатеричной записи. Наконец, ключевое слово `broadcast` предваряет широковещательный адрес.

В системе Linux команда `ifconfig` может выводить до семи строк информации по каждому интерфейсу (вместо двух строк, как в Solaris). Дополнительные сведения включают Ethernet-адрес, номер прерывания, базовый адрес ввода-вывода, адрес в памяти и статистику по пакетам. Основная информация для настроек TCP/IP одинакова в обеих системах.

```
> ifconfig eth0
eth0  Link encap:Ethernet  HWaddr 00:00:C0:9A:D0:DB
      inet addr:172.16.55.106  Bcast:172.16.55.255  Mask:255.255.255.0
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:844886 errors:0 dropped:0 overruns:0 frame:0
      TX packets:7668 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:100
      Interrupt:11 Base address:0x7c80 Memory:c0000-c2000
```

Вернемся к примеру `ifconfig dnet0` для Solaris, приведенному в начале этого раздела, и сравним полученную информацию с конфигурацией подсетей, запланированной к реализации в нашей воображаемой сети. Видно, что настройки интерфейса нужно изменить. Настройка, выполненная пользователем в процессе установки, обошла некоторые из запланированных параметров. Адрес (172.16.12.2) верен, но адресная маска (ffff0000 или 255.255.0.0) и широковещательный адрес (172.16.0.0) не соответствуют планам. Рассмотрим различные способы установки и корректировки параметров интерфейса.

Назначение адреса

IP-адрес можно назначить непосредственно в командной строке `ifconfig` либо косвенным образом в файле. В примерах использования `ifconfig`, приведенных выше в тексте, адрес IP присутствует в командной строке в десятичной записи через точку. Как вариант, можно указать в качестве аргумента `ifconfig` имя узла из файла */etc/hosts*. К примеру:

```
# ifconfig dnet0 crab netmask 255.255.255.0
```

Большинству администраторов удобнее пользоваться именами узлов, а не адресами. Однако разработчики систем, выполняющие первичную настройку, обычно добавляют еще один уровень переадресации. Команда `ifconfig` из загрузочного сценария ссылается на файл. Файл содержит имя узла, а имя узла отображается в адрес. В системах Solaris имя узла размещается в файле с именем `/etc/hostname.`*interface*, где *interface* – имя интерфейса, подлежащего настройке. Для системы нашего примера файл называется */etc/hostname.dnet0*. Файл *hostname.dnet0*, созданный при стандартной установке Solaris, содержит просто имя узла:

```
$ cat /etc/hostname.dnet0
crab
```

```
$ grep crab /etc/hosts
172.16.12.1     crab     crab.wrotethebook.com       loghost
```

Как видно из примера, система Solaris создала при настройке файл *hostname.dnet0* и соответствующую запись в файле */etc/hosts*, позволяющую выполнять отображение имени узла из файла *hostname.dnet0* в IP-адрес. В процессе загрузки Solaris берет имя узла из файла, а затем получает адрес, связанный с этим именем, из второго файла. Обе записи необходимы для процесса настройки.

В Linux также применяется косвенное указание адреса для `ifconfig`. Отдельные системы Linux, в частности Red Hat, Mandrake и Caldera, размещают значения настройки сетевого интерфейса в файле с именем `ifcfg.`*`interface`*, где *`interface`* – имя интерфейса.[1] Например, *ifcfg.eth0* содержит значения настройки для Ethernet-интерфейса eth0.

```
$ cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=none
BROADCAST=172.16.12.255
NETWORK=172.16.12.0
NETMASK=255.255.255.0
IPADDR=172.16.12.2
USERCTL=no
```

Данный файл позволяет легко интерпретировать параметры настройки.

- DEVICE определяет имя устройства, в данном случае – eth0.
- ONBOOT указывает необходимость инициализации интерфейса при загрузке системы. Обычно интерфейс Ethernet инициализируется и начинает использоваться при каждой загрузке системы.
- BOOTPROTO определяет службу настройки, посредством которой выполняется настройка интерфейса. Значение `none` означает, что интерфейс будет настроен локальными средствами. Существуют также значения `bootp` (используется устаревший сервер BootP) и `dhcp` (используется сервер DHCP). В случае применения DHCP или BootP перечисленные ниже конкретные значения настройки в файле отсутствуют.
- BROADCAST определяет широковещательный адрес для `ifconfig`.
- NETWORK определяет адрес сети.
- NETMASK определяет адресную маску для `ifconfig`.
- IPADDR определяет IP-адрес для `ifconfig`.
- USERCTL разрешает или запрещает пользователям запускать и останавливать интерфейс посредством `usernetctl`. Команда `usernetctl` существует

[1] Файл *ifcfg.eth0* в системе Red Hat нашего примера хранится в каталоге */etc/sysconfig/network-scripts*.

далеко не во всех вариантах Linux. В данном примере значение `no` запрещает пользователям отключать интерфейс.

Большинство систем использует то преимущество, что IP-адрес, маска подсети и широковещательный адрес могут устанавливаться косвенно. Последнее обстоятельство позволяет минимизировать вносимые в загрузочные файлы коррективы. Сокращение вносимых исправлений снижает вероятность зависания системы в процессе загрузки из-за внесенного в загрузочный файл некорректного изменения, а также дает возможность заранее создать такие файлы для всех систем сети. Системы Solaris обладают дополнительным преимуществом: *узлы*, *сети* и файлы *сетевых масок*, используемые в работе команды `ifconfig`, могут служить источниками карт NIS. Картами NIS можно управлять централизованно.

Недостаток косвенного указания значений для `ifconfig` заключен в потенциальном усложнении поиска неисправностей. Если все значения содержатся в загрузочном файле, их можно охватить за один прием. Когда сетевые настройки представлены ссылками, обнаружение проблемы может потребовать обращения к целому ряду файлов, а причина неработоспособности конфигурации может крыться в любом из них. В целях облегчения отладки в отдельных операционных системах параметры настройки указываются непосредственно в командной строке `ifconfig`, содержащейся в загрузочном файле.

Я рекомендую следовать стандартным методам, принятым для конкретной системы. Если используется система Solaris, указывайте адрес при помощи файлов */etc/hostname.dnet0* и */etc/hosts*. В случае системы Red Hat указывайте адрес в файле */etc/sysconfig/network-scripts/ifcfg.eth0*. В случае системы Slackware указывайте адрес непосредственно в загрузочном файле *rc.inet*. Следование принятым для системы методам облегчает отладку компьютера для всех администраторов, имеющих к нему доступ. В процессе изучения оставшихся параметров настройки интерфейса мы рассмотрим и другие альтернативы.

Назначение маски подсети

Корректная работа всех интерфейсов конкретного сегмента сети требует одинаковой адресной маски. Значение сетевой маски для узлов *crab* и *rodent* – 255.255.255.0, поскольку обе системы входят в одну подсеть. Но, хотя локальный и внешний сетевой интерфейсы узла *crab* сосуществуют на одном компьютере, они имеют разные сетевые маски, поскольку принадлежат к различным сетям.

Чтобы назначить адресную маску, укажите ее значение после ключевого слова `netmask` в командной строке `ifconfig` либо в качестве префикса, связанного с адресом. При записи в качестве префикса адресная маска представляется десятичным числом, определяющим число битов в адресной маске. Так, значение 172.16.12.2/24 задает 24-битную адресную маску. При записи в качестве аргумента ключевого слова `netmask` маска, как правило, представляется в деся-

тичной записи через точку, то есть в формате IP-адресов.[1] Следующая команда назначает корректную маску подсети интерфейсу dnet0 узла *rodent*:

```
# ifconfig le0 172.16.12.2 netmask 255.255.255.0
```

Указание значения маски непосредственно в командной строке `ifconfig` – это самый распространенный, простой и оптимальный способ ручного назначения маски для интерфейса. Однако она редко задается вручную. Подобно адресам, адресные маски используются в настройке, выполняемой в процессе первичной установки системы. `ifconfig`, в целях упрощения настройки, может читать значение сетевой маски из файла. Такой способ схож с указанием имени узла вместо IP-адреса. Администратор может разместить значение маски в файле *узлов* (*hosts*) либо в файле *сетей* (*networks*), а затем ссылаться на маску по имени. К примеру, администратор сети *books-net* может добавить следующую запись в файл */etc/networks*:

```
books-mask 255.255.255.0
```

Теперь имя *books-mask* в командной строке `ifconfig` может замещать значение маски. К примеру, так:

```
# ifconfig dnet0 172.16.5.2 netmask books-mask
```

Имя *books-mask* отображается в значение 255.255.255.0, которое представляет корректную маску для систем из нашего примера.

Лично я стараюсь избегать косвенного указания значений масок в файлах, главное назначение которых в другом. Файл *узлов* (*hosts*) – вдвойне плохой вариант для хранения значений масок. Файл узлов постоянно используется многочисленными программами, и значение маски, хранимое в файле, может смутить какие-либо из этих программ. Вероятно, оптимальным подходом является указание маски непосредственно в командной строке либо хранение ее в специальном файле.

В системах Solaris под хранение масок подсетей отведен файл */etc/inet/netmasks*.[2] Файл состоит из однострочных записей; каждая запись содержит адрес сети, отделенный пробелом от маски.[3] Если система Solaris сети *books-net* (172.16.0.0) содержит в файле */etc/inet/netmasks* такую запись:

```
172.16.0.0 255.255.255.0
```

следующая команда `ifconfig` может применяться для указания адресной маски:

```
# ifconfig dnet0 172.16.5.1 netmask +
```

---

[1] Адресная маска может записываться и в шестнадцатеричной системе. В этом случае используется единичное шестнадцатеричное значение с префиксом 0x. К примеру, в маска 255.255.255.0 в шестнадатеричной записи выглядит так: 0xffffff00. Выбирайте формат, с которым вам проще работать.

[2] */etc/netmasks* является символической ссылкой на */etc/inet/netmasks*.

[3] Используйте официальный адрес сети, а не адрес подсети.

Знак «плюс» после ключевого слова `netmask` предписывает `ifconfig` прочитать значение маски из файла */etc/inet/netmasks*. `ifconfig` ищет в файле адрес сети, соответствующий адресу настраиваемого сетевого интерфейса. Для настройки интерфейса используется маска, связанная с его адресом.

В большинстве систем Linux адресные маски также косвенно определяются в файлах. Файл *ifcfg-eth0* из предыдущего раздела содержит такую строку:

```
NETMASK=255.255.255.0
```

Эта строка четко указывает значение маски, используемое командой `ifconfig`. Чтобы изменить адресную маску в данной системе Red Hat, отредактируйте строку в файле *ifcfg-eth0*.

Указание широковещательного адреса

Документ RFC 919, *Broadcasting Internet Datagrams* (Широковещательная передача дейтаграмм Интернета) четко определяет широковещательный адрес как адрес, все биты раздела узла которого установлены в значение 1. Столь точное определение широковещательного адреса позволяет программе `ifconfig` вычислить его автоматически, так что всегда есть возможность использовать значение по умолчанию. К сожалению, в примере раздела «Проверка интерфейса посредством ifconfig» пользователь выбрал широковещательный адрес, все биты раздела узла которого имеют нулевое значение, что не позволяет использовать значение по умолчанию.

Чтобы исправить ошибку, укажем широковещательный адрес для сетевого устройства при помощи ключевого слова `broadcast` команды `ifconfig`. Следующая команда определяет широковещательный адрес интерфейса dnet0 узла *crab*:

```
# ifconfig dnet0 172.16.12.1 netmask 255.255.255.0 broadcast 172.16.12.255
```

Обратите внимание, что широковещательный адрес указывается относительно локальной подсети. *crab* считает данный интерфейс частью сети 172.16.12.0; следовательно, широковещательный адрес интерфейса – 172.16.12.255. В зависимости от реализации система Unix может интерпретировать адрес 172.16.255.255 как адрес узла 255 подсети 255 сети 172.16.0.0 либо как широковещательный адрес подсети *books-net* в целом, но ни в коем случае не как широковещательный адрес подсети 172.16.12.0.

Системы Solaris позволяют косвенно устанавливать широковещательный адрес по значению маски, определенному в файле */etc/inet/netmasks*, если таковой используется. В предыдущем разделе мы видели, что аргумент `netmask +` приводит к извлечению сетевой маски из файла. Аналогичным образом, аргумент `broadcast +` предписывает вычислить корректный широковещательный адрес на основе значения маски из файла *netmasks*:

```
# ifconfig dnet0 172.16.12.1 netmask + broadcast +
```

Предположим, что в файле *netmasks* определена маска 255.255.255.0. Система Solaris знает, что первые три байта определяют сеть, а четвертый байт опре-

деляет адрес узла. Поскольку стандартный широковещательный адрес состоит из битов сети в сочетании с битами узла, установленными в значение 1, Solaris легко вычисляет этот адрес: 172.16.12.255.

В Linux задача решается еще проще. Файл *ifcfg-eth0* системы Red Hat из примера четко определяет широковещательный адрес следующей строкой:

```
BROADCAST=172.16.12.255
```

Чтобы изменить широковещательный адрес, отредактируйте данную строку в файле *ifcfg-eth0*.

Прочие параметры команды

Мы использовали `ifconfig` для установки адреса интерфейса, маски подсети и широковещательного адреса. Однако этими важными функциями список функций `ifconfig` не исчерпывается. Команда позволяет включать и отключать протокол разрешения адресов либо собственно интерфейс. `ifconfig` дает возможность устанавливать метрику маршрутизации для протокола RIP (Routing Information Protocol, протокол маршрутной информации) и максимальный размер передаваемого блока MTU для интерфейса. В этом разделе приводятся примеры работы с каждой из этих функций.

Включение и отключение интерфейса

Аргументы `up` и `down` команды `ifconfig` позволяют включать и отключать сетевой интерфейс. Аргумент `up` включает интерфейс и отмечает его как пригодный к использованию. Аргумент `down` отключает интерфейс и запрещает передачу через него сетевого трафика.

Используйте аргумент `down` для перенастройки интерфейса вручную. Отдельные параметры настройки, например IP-адрес, могут изменяться, только когда интерфейс заблокирован. Поэтому интерфейс отключается, выполняется настройка, и затем интерфейс включается снова. Например, следующие команды изменяют адрес интерфейса:

```
# ifconfig eth0 down
# ifconfig eth0 172.16.1.2 up
```

Когда команды выполнены, интерфейс начинает работу с новыми значениями настройки. Аргумент `up` во второй команде `ifconfig` не всегда является обязательным, поскольку в некоторых системах его присутствие предполагается по умолчанию. После отключения интерфейса либо при вызове `ifconfig` из сценария обычно используется явный аргумент `up`, что позволяет избежать потенциальных проблем.

ARP

Протокол разрешения адресов (Address Resolution Protocol, ARP), описанный в главе 2, – это важный механизм преобразования IP-адресов в физические адреса Ethernet. ARP включается при помощи ключевого слова `arp` и отключается при помощи ключевого слова `-arp`. Теоретически может оказать-

ся, что узел сети не способен работать с протоколом ARP. Такое возможно только в сетях, построенных на специализированном или экспериментальном оборудовании. В таких редких случаях иногда возникает необходимость отключить ARP, чтобы обеспечить взаимодействие нестандартных систем. По умолчанию `ifconfig` включает ARP. Оставляйте ARP включенным на всех системах.

Беспорядочный режим

В главе 13 мы используем беспорядочный режим для просмотра пакетов, путешествующих в локальной Ethernet-сети. По умолчанию интерфейс Ethernet передает протоколам более высокого уровня только фреймы, адресованные локальному узлу. В беспорядочном режиме вверх по стеку передаются все фреймы, независимо от пункта назначения.

В системе Linux беспорядочный режим включается при помощи параметра `promisc` команды `ifconfig`. Например:

```
$ ifconfig eth0 promisc
```

Отключается беспорядочный режим параметром `-promisc`.[1] По умолчанию беспорядочный режим отключен. Включение беспорядочного режима навязывает локальной системе обработку многих пакетов, которые в обычной ситуации отбрасываются аппаратным интерфейсом Ethernet. Беспорядочный режим включают только для некоторых приложений диагностирования и отладки.[2]

Метрика

В некоторых системах команда `ifconfig` создает для каждого интерфейса, которому назначен IP-адрес, запись в таблице маршрутизации. Каждый интерфейс является маршрутом в определенную сеть. Даже если узел не является шлюзом, его интерфейс – это его «маршрут» в локальную сеть. `ifconfig` определяет конечную сеть маршрута, применяя адресную маску интерфейса к IP-адресу интерфейса. Например, интерфейсу dnet0 узла *crab* присвоены адрес 172.16.12.1 и маска 255.255.255.0. Применив маску к адресу, мы получим конечную сеть 172.16.12.0. Вывод команды `netstat -in` содержит конечный адрес:

```
% netstat -in
Name Mtu  Net/Dest    Address     Ipkts  Ierrs Opkts Oerrs Collis Queue
le0  1500 172.16.12.0 172.16.12.1 1125826 16   569786  0    8914   0
lo0  1536 127.0.0.0   127.0.0.1   94280   0    94280   0    0      0
```

Протокол маршрутной информации RIP (Routing Information Protocol) иногда применяется в системах Unix. RIP выполняет две функции: передает

---

[1] В системе Solaris беспорядочный режим включается программами, которые намереваются его использовать, а не командой `ifconfig`.

[2] Этот режим используется «снифферами» – программами, извлекающими конфиденциальную информацию, например пароли, из сетевых пакетов, и поэтому включение этого режима ослабляет общую защищенность сети. – *Примеч. науч. ред.*

информацию маршрутизации другим узлам, а также использует поступающую информацию маршрутизации для динамического создания таблиц маршрутизации. Маршруты, созданные программой `ifconfig`, – один из источников информации маршрутизации, распространяемой протоколом RIP, а для управления тем, как RIP использует такую информацию, может использоваться аргумент `metric` команды `ifconfig`.

RIP принимает решения по маршрутизации на основе стоимости маршрута. Стоимость маршрута определяется связанной с ним метрикой маршрутизации. Метрика маршрутизации – это просто число. Чем меньше число, тем меньше стоимость маршрута. Чем больше число, тем стоимость выше. Создавая таблицу маршрутизации, RIP отдает предпочтение более дешевым маршрутам. Сети с прямым подключением получают очень низкую стоимость маршрута. Таким образом, нулевая метрика по умолчанию отмечает маршруты, пролегающие через интерфейсы с напрямую подключенными сетями. При помощи параметра `metric` можно указать иную метрику маршрутизации для интерфейса.

Чтобы повысить стоимость интерфейса до значения 3 (в этом случае протокол RIP будет отдавать предпочтение маршрутам стоимости 0, 1 или 2), выполните такую команду:

```
# ifconfig std0 10.104.0.19 metric 3
```

Используйте параметр `metric` только в случае, когда существует иной маршрут в конечную сеть, и он должен стать основным. Мы не воспользовались подобной командой для узла *crab*, поскольку этот узел имеет лишь один интерфейс для связи с внешним миром. В присутствии второго соединения, скажем, по высокоскоростному каналу, вышеприведенная команда могла бы послужить для направления трафика через высокопроизводительный интерфейс.

Похожий параметр `ifconfig` существует и в системе Solaris. RIP создает таблицу маршрутизации, выбирая наиболее экономичные маршруты, и передает информацию маршрутизации другим узлам. Параметр `metric` позволяет менять выбор маршрутов, назначая менее предпочтительным маршрутам более высокую стоимость. Аргумент `private`, доступный в системах Solaris, управляет распределением маршрутов для RIP. Наличие аргумента `private` в командной строке `ifconfig` запрещает RIP распространять маршрут, который будет создан при выполнении данной команды. По умолчанию принимается значение `-private`, разрешающее распространение маршрута. Параметр `private` поддерживается не всеми системами.

Кроме того, не все системы обращают внимание на аргумент `metric`. В системе Linux выполнение команды `ifconfig` не приводит к созданию записи в таблице маршрутизации. При настройке системы для создания маршрутов, пролегающих через интерфейсы, используется явная команда `route`. (О команде `route` речь пойдет в следующей главе.) Система Linux отвергает аргумент `metric`, как видно из этого примера:

```
# ifconfig eth0 192.168.0.4 metric 3
SIOCSIFMETRIC: Operation not supported
```

Указывайте метрику в специальном файле настройки маршрутизации, а не в командной строке ifconfig. Это предпочтительный способ указания сведений для современных пакетов маршрутизации. Формат файлов настройки маршрутизации мы обсудим в следующей главе.

Максимальный размер передаваемого блока

Физической сети назначается максимальный размер передаваемого блока (MTU, maximum transmission unit), определяющий максимальный размер пакета, передаваемого по сети. Для Ethernet максимальный размер составляет 1500 байт и является частью стандарта Ethernet. Необходимость изменять значение MTU из командной строки ifconfig возникает крайне редко. По умолчанию ifconfig выбирает оптимальное значение, как правило, самое большое из допустимых для конкретного типа сетевых устройств. Большое значение MTU является умолчанием, поскольку обычно обеспечивает наилучшую производительность. При этом, отметим, меньшее значение MTU может способствовать достижению определенных целей:

- Избежать фрагментации. Если данные путешествуют из сети с большим значением MTU (скажем, из сети FDDI с MTU, равным 4500 байт) через сеть с меньшим значением MTU (например, Ethernet), уменьшение размера MTU может оказаться оптимальным решением для предотвращения фрагментации пакетов. Возможно, что установка значения MTU, равного 1500 байт, для интерфейса, подключенного к сети FDDI, положительно повлияет на производительность, избавив маршрутизаторы от необходимости разбивать пакеты. Разумеется, такая операция имеет смысл лишь в случае, когда в снижении производительности действительно повинна фрагментация пакетов.
- Избежать, по возможности, переполнения буферов и других проблем такого рода. Для соединения по последовательной линии может применяться оборудование с настолько низкой производительностью, что она не позволит справляться со стандартными пакетами, размер которых равен 1006 байт. В таком случае уменьшение MTU позволяет избежать переполнения буферов или SILO-переполнения. Однако такой способ можно считать лишь временным решением. Настоящее решение связано с приобретением устройств, отвечающих потребностям системы.

Чтобы изменить значение MTU, воспользуйтесь параметром mtu команды ifconfig:

```
# ifconfig fddi0 172.16.16.1 netmask 255.255.255.0 mtu 1500
```

Интерфейс FDDI с адресом 172.16.16.1 получает предписание использовать значение MTU, равное 1500 байт.

Точка-точка

Ваша система, вероятно, поддерживает еще ряд аргументов командной строки ifconfig. В Linux существуют параметры, позволяющие указать прерывания для устройства Ethernet (irq) и аппаратный адрес устройства Ethernet

(hw), а также включить многоадресную передачу (multicast) и беспорядочный режим (promisc). В Solaris существуют аргументы, позволяющие инициализировать и блокировать потоки для интерфейса (plumb/unplumb), а также использовать протокол Reverse ARP (RARP) для получения IP-адреса интерфейса (auto-revarp). Большинство подобных параметров не являются стандартными для Unix.

Еще одна функция, доступная в большинстве вариантов Unix, позволяет создавать при помощи команды ifconfig соединения типа точка-точка. Соединение точка-точка – это сетевой канал, напрямую соединяющий два компьютера. Разумеется, оба компьютера могут одновременно являться шлюзами во внешние сети, но лишь два компьютера имеют прямое подключение к этому каналу передачи данных. Примером соединения точка-точка может служить соединение двух компьютеров выделенной телефонной линией либо соединение двух офисных машин нуль-модемным кабелем.

Чтобы создать канал точка-точка в системе Solaris, воспользуйтесь командой:

```
# ifconfig zs0 172.16.62.1 172.16.62.2
```

В командной строке за именем интерфейса следует пара адресов. Первый адрес принадлежит локальному узлу. Второй адрес, известный как конечный, принадлежит второй стороне канала точка-точка. Второй адрес отображается в поле Net/Dest вывода команды netstat -ni.

В системе Linux команда настройки выглядит несколько иначе:

```
$ ifconfig sl0 172.16.62.1 point-to-point 172.16.62.2
```

Результат, несмотря на различия в синтаксисе, одинаков в обоих случаях. Интерфейс переводится в режим point-to-point (точка-точка), а кроме того, указываются адреса связываемых узлов.

Происходит ли здесь необходимая настройка протокола PPP (Point-to-Point Protocol), используемого для работы с TCP/IP по последовательным соединениям? Нет, не происходит. Эти параметры команды ifconfig иногда смущают людей, мешая им разобраться с настройкой PPP. Настройка PPP гораздо сложнее, как мы увидим позже в этой главе.

Но прежде чем обратиться к PPP, следует отметить, что настройки, сделанные из командной строки ifconfig, просуществуют лишь до следующей перезагрузки компьютера. Чтобы получить постоянные настройки, необходимо поместить команду ifconfig в загрузочный файл.

ifconfig в загрузочных сценариях

Команда ifconfig обычно вызывается при загрузке системы из загрузочного сценария. Две базовые модели загрузки систем Unix, модель BSD и модель System V, были описаны в главе 5. В системах BSD Unix команды ifconfig размещаются, как правило, в файле */etc/rc.boot* или */etc/rc.local*.

Чтобы заменить настройки, принятые в системе BSD по умолчанию, добавьте полную команду ifconfig в сценарий *rc.local*. *rc.local* выполняется в конце

процесса загрузки. Любые параметры настройки интерфейса, указанные в этом файле, замещают ранее указанные значения. Например, следующая строка в файле сценария *rc.local* выполняет настройку интерфейса eth0, игнорируя любые уже существующие настройки:

```
ifconfig eth0 172.16.12.1 broadcast 172.16.12.255 netmask 255.255.255.0
```

Модель загрузки BSD применяется в системах BSD и SunOS. В системах Linux и Solaris применяется модель загрузки System V. Однако в системах Red Hat Linux существует сценарий *rc.local* – в каталоге */etc/rc.d*. В системе Red Hat разместите собственную команду `ifconfig` в файле *rc.local*, чтобы изменить настройки по умолчанию.

В Solaris отсутствует сценарий *rc.local*, равно как и центральный каталог со сценариями для всех уровней исполнения. Чтобы воспользоваться сценарием *rc.local* в системе Solaris, необходимо самостоятельно создать файл сценария и поместить его в каталог третьего уровня исполнения. Файл должен иметь правильное имя, поскольку имена определяют порядок выполнения сценариев в Solaris. Сценарий с именем */etc/rc3.d/S99local* будет выполняться в конце загрузки стандартного третьего уровня исполнения системы Solaris. Команды, размещенные в этом файле, будут изменять уже существующие настройки.

По возможности следует выполнять настройку сети при помощи стандартных средств и процедур, принятых в конкретной операционной системе. Прямая правка загрузочных сценариев и создание нестандартных сценариев может создать много сложностей людям, которые помогают вам в поддержке систем сети.

TCP/IP и последовательные линии

Протоколы TCP/IP способны работать с многочисленными разновидностями физического транспорта. Это может быть кабельное Ethernet-подключение, как в локальных сетях, либо телефонные линии, как в территориальных сетях. В первой половине главы мы использовали `ifconfig` для настройки локального интерфейса Ethernet. В этом разделе мы используем другие команды, чтобы настроить сетевой интерфейс телефонной линии.

Практически любой обмен данными происходит через последовательные интерфейсы. *Последовательный интерфейс* – это любой интерфейс, передающий данные в виде последовательностей битов по единственному проводу (тогда как параллельные интерфейсы передают биты данных по нескольким проводам одновременно). Под такое описание последовательного интерфейса подпадают практически все интерфейсы обмена данными (включая Ethernet), но термин обычно используется в отношении интерфейса, подключенного к телефонной линии посредством модема или подобного устройства. Телефонную линию часто называют последовательной линией.

В мире TCP/IP последовательные линии применяются для создания территориальных сетей (wide area networks, WANs). К сожалению, в TCP/IP не

всегда существовал стандартный протокол физического уровня для последовательных линий. Отсутствие стандарта вынуждало проектировщиков использовать для территориальных сетей маршрутизаторы одной марки, чтобы обеспечить успешный обмен данными на физическом уровне. Распространение сетей WAN на основе TCP/IP создавало потребность в стандартизации последовательного обмена, который снял бы зависимость от производителей оборудования.

Кроме того, интерес к коммуникациям по последовательным линиям вырос с появлением небольших, недорогих систем, работающих с TCP/IP, а также высокоскоростных модемов для коммутируемых линий, обеспечивающих «приемлемую» производительность TCP/IP. На момент создания ARPAnet компьютеры стоили невероятных денег, а модемы для коммутируемых линий работали на предельно низких скоростях. В те времена тот, кто покупал компьютер, мог позволить себе приобрести и выделенную телефонную линию. Однако в последнее время стало возможным использовать системы Unix дома. В новой сетевой среде существует значительный спрос на службы, позволяющие обеспечивать коммутируемый доступ по протоколам TCP/IP. В настоящее время около 7% домашних пользователей работают через высокоскоростное DSL-подключение (Digital Subscriber Line) или кабельный модем. Большинство модемов DSL и кабельных модемов подключаются к узлу посредством Ethernet, то есть не требуют специальной настройки. Но большинство пользователей домашних компьютеров все еще работает с коммутируемыми последовательными линиями, которые требуют специальных протоколов и специальной настройки.

Эти две силы – потребность в стандартизации территориального обмена данными и потребность в коммутируемом доступе по TCP/IP – стали причиной создания двух протоколов для последовательных линий: протокола SLIP (Serial Line IP) и протокола PPP (Point-to-Point Protocol).[1]

Последовательные протоколы

Протокол Serial Line IP появился на свет первым. Это протокол, позволяющий отдельным узлам устанавливать соединения TCP/IP в телефонной сети. Протокол SLIP определяет простой механизм кадрирования дейтаграмм для передачи по последовательным линиям. SLIP передает дейтаграмму в виде последовательности байтов, а для указаний по группировке байтов в дейтаграммы использует специальные маркеры. SLIP определяет два специальных символа:

- Символ SLIP END, отдельный байт с десятичным значением 192. Отмечает конец дейтаграммы. Прием символа END в протоколе SLIP означает, что дейтаграмма полностью получена и может быть передана в уровень IP.

[1] Модемы для коммутируемых линий обычно асинхронные. Протоколы PPP и SLIP поддерживают в равной степени работу с асинхронными коммутируемыми службами и синхронными выделенными линиями.

- Символ SLIP ESC, отдельный байт с десятичным значением 219. Используется для «маскировки» управляющих символов SLIP. Если источник SLIP-передачи встречает в передаваемой дейтаграмме байт со значением, эквивалентным символу SLIP END или SLIP ESC, такой байт преобразуется в последовательность двух символов. Двухсимвольные последовательности: ESC 220 для символа END и ESC 221 для самого символа ESC.[1] Адресат SLIP-передачи, встретив такую двухбайтовую последовательность, преобразует ее в однобайтовое значение. Описанный механизм предотвращает интерпретацию байта данных в качестве маркера конца дейтаграммы SLIP-адресатом.

Протокол SLIP описан в документе RFC 1055, *A Nonstandard for Transmission of IP Datagrams Over Serial Lines: SLIP* (Нестандартная передача дейтаграмм IP через последовательные линии: SLIP). Как видно из названия RFC, SLIP не является стандартом Интернета. RFC не предлагает стандартизацию, лишь документирует существующий протокол. В документе RFC 1055 описаны недостатки протокола SLIP, разделенные на две категории:

- Протокол SLIP не содержит средств динамического управления характеристиками соединения. Следовательно, системы SLIP неявно предполагают наличие определенных характеристик канала. Вследствие этого ограничения SLIP может применяться только для передачи дейтаграмм IP и только в случае, когда каждой из сторон известен не только свой адрес, но и адрес второй стороны.
- SLIP не позволяет компенсировать зашумленность или низкую скорость телефонных линий. В протоколе отсутствуют возможности коррекции ошибок и сжатия данных.

Протокол PPP (Point-to-Point Protocol), разработанный в качестве стандарта Интернета, призван восполнить недостатки протокола SLIP. Протокол PPP регламентируется рядом документов RFC.[2] Два ключевых документа: RFC 1661, *The Point-to-Point Protocol* (PPP) и RFC 1172, *The Point-to-Point Protocol (PPP) Initial Configuration Options*.

PPP восполняет недостатки SLIP при помощи трехуровневого протокола:

Протокол канального уровня (Data Link Layer Protocol)

Протокол канального уровня, применяемый в PPP, является немного модифицированной версией протокола HDLC (Highlevel Data Link Control). PPP добавляет в HDLC поле Protocol, позволяющее PPP адресовать трафик различным протоколам сетевого уровня. HDLC – это международный стандартизированный протокол, позволяющий надежно передавать данные по синхронным, последовательным линиям. Кроме того, в PPP применяется предварительный вариант международного стандарта для HDLC-передачи по асинхронным линиям, так что PPP гарантирует надежность доставки по последовательной линии любого типа.

---

[1] В данном случае ESC обозначает управляющий символ SLIP, а не ASCII.

[2] Чтобы наверняка иметь дело с наиболее актуальной версией стандарта, обратитесь к перечню документов RFC, описанных в приложении G.

Протокол управления каналом (Link Control Protocol)

Протокол управления каналом LCP (Link Control Protocol) обеспечивает обмен управляющей информацией для последовательных каналов. LCP используется для организации соединения, согласования параметров настройки, проверки качества связи и завершения соединения. LCP разрабатывался специально для PPP.

Протоколы управления сетью (Network Control protocols)

Протоколы управления сетью – это отдельные протоколы, передающие данные настройки и управляющую информацию протоколам сетевого уровня. Вспомните, что PPP спроектирован таким образом, чтобы передавать данные широкому спектру сетевых протоколов. NCP позволяет регулировать работу PPP в этой области. Каждому из сетевых протоколов (DECNET, IP, OSI и т. д.) соответствует протокол сетевого управления. Документами RFC 1661 и RFC 1332 определен протокол управления протоколом Internet (IPCP, Internet Protocol Control Protocol), позволяющий работать с протоколом Internet.

Протокол PPP – лучший протокол TCP/IP для последовательных соединений. Дело в том, что протокол PPP является стандартом Интернета, что гарантирует корректный обмен данными для самых разнообразных систем. PPP имеет больше возможностей, чем SLIP, и является более устойчивым. Эти преимущества делают PPP оптимальным выбором открытого протокола для связи маршрутизаторов по последовательным линиям и подключения удаленных компьютеров по коммутируемым каналам.

Отдельные системы Linux включают как PPP, так и SLIP. Однако в большинстве систем Unix, таких как Solaris, присутствует PPP, а не SLIP. Ничего плохого в этом нет, поскольку следует избегать применения SLIP в пользу PPP.

Установка PPP

Процедура установки и настройки PPP зависит от конкретной реализации.[1] В этом разделе мы используем реализацию демона PPP (`pppd`), поставляемую в составе систем Linux, и рассмотрим сопутствующие команды настройки. PPP является стандартом Интернета, и в стандартных ядрах большинства систем Unix по умолчанию присутствует поддержка этого протокола. Обычно для его установки не требуется дополнительных действий. Настройка PPP в ядре Linux описана в главе 5. В системе Linux программные модули физического и канального уровня (HDLC) PPP интегрированы в ядро.

Установка PPP в ядре – лишь начало. В этом разделе мы рассмотрим применение демона `pppd` для развертывания PPP-служб в системе Linux.

---

[1] Сверьтесь с документацией на вашу систему, чтобы точно знать, как следует настраивать PPP.

Демон PPP

Протокол PPP реализован в системе Linux в виде демона PPP (pppd), созданного на основе бесплатной реализации PPP для систем BSD. pppd может работать во всех режимах: в качестве клиента, в качестве сервера, по коммутируемым и выделенным каналам. (Понятия клиента и сервера должны быть знакомы читателям из главы 3.) Выделенный канал – это прямое кабельное подключение (или выделенная линия), не требующее наличия телефона для создания соединения. Коммутируемое соединение – это модемный канал, создаваемый набором телефонного номера.

Простейший вариант настройки pppd связан с выделенными линиями. Выделенная линия или прямое подключение не требует создания сценария дозвона. Нет смысла динамически назначать адреса, поскольку выделенная линия всегда соединяет две конкретные системы. Проверка подлинности не особо востребована, поскольку выделенная линия физически соединяет две точки. Злоумышленник не может получить доступа к каналу, не считая вариантов «взлома и проникновения» или электронного прослушивания. В системе Linux настройка выделенного PPP-канала выполняется единственной командой в загрузочном сценарии:

```
pppd /dev/cua3 56000 crtscts defaultroute
```

Аргумент /dev/cua3 указывает устройство, с которым работает PPP. Разумеется, это именно тот порт, который подключен к выделенной линии. Скорость линии определяется в битах за секунду (56000). Оставшаяся часть строки команды содержит последовательность ключевых слов параметров. Параметр crtscts включает аппаратный контроль передачи. Последний параметр, defaultroute, создает маршрут по умолчанию через удаленный сервер, используя его в качестве шлюза.[1]

PPP выполняет обмен IP-адресами при первоначальном создании соединения. Если в командной строке pppd адрес не указан, демон передает второй стороне адрес локального узла, извлеченный из таблицы узлов или полученный посредством службы DNS. Аналогичным образом удаленная система передает свой адрес локальному узлу. Эти адреса используются в качестве исходного и конечного адресов канала. Адреса могут быть указаны в командной строке в формате «локальный:удаленный». Пример:

```
pppd /dev/cua3 56000 crtscts defaultroute 172.16.24.1:
```

Здесь указан локальный адрес 172.16.24.1 и пустой адрес второй стороны. В этом случае pppd передает адрес, указанный в командной строке, и ожидает, что удаленный сервер пришлет свой адрес. Локальный адрес указывается в командной строке, если он отличается от адреса, связанного с именем локального узла в таблице узлов либо на сервере DNS. Например, система может использовать интерфейс Ethernet, которому уже назначен адрес. Что-

[1] Если маршрут по умолчанию уже существует в таблице маршрутизации, параметр defaultroute игнорируется.

бы воспользоваться другим адресом для соединения PPP, мы должны указать его в командной строке pppd; в противном случае каналу PPP будет назначен существующий адрес интерфейса Ethernet.

Команда pppd предоставляет гораздо больше параметров, чем мы увидим в примерах этой главы (полный перечень параметров содержится в приложении A). В действительности параметров командной строки pppd существует столько, что иногда проще сохранить их в файле, чем набирать в командной строке. pppd читает параметры прежде всего из файла */etc/ppp/options*, затем из файла *~/.ppprc*, затем из файла */etc/ppp/options.device* (где *device* – имя устройства, скажем, cua3) и, наконец, из командной строки. Порядок обработки параметров создает определенную иерархию: параметры командной строки имеют более высокий приоритет, чем параметры файла *~/.ppprc*. Параметры файла *~/.ppprc*, в свою очередь, имеют приоритет более высокий, чем параметры файла */etc/ppp/options*. Такая система позволяет системному администратору задать общесистемные умолчания параметров в файле */etc/ppp/options*, но оставить пользователям возможность дополнительно настраивать PPP. Файл */etc/ppp/options* является удобным и гибким способом передачи параметров демону pppd.

Единственная команда pppd позволяет настроить PPP для работы по выделенному каналу. Коммутируемые соединения требуют бо́льших усилий.

Коммутируемые соединения PPP

Прямое кабельное соединение позволяет связывать лишь две системы. Приобретение третьей системы ставит нас перед вопросом – как подключить ее к сети? Именно по этой причине чаще всего применяются расширяемые технологии, в частности Ethernet, для объединения систем в локальную сеть. Кроме того, выделенные линии дороги. Они используются в основном крупными организациями для объединения сетей. Таким образом, PPP чаще применяется для создания коммутируемых соединений, нежели выделенных каналов.

Поддержка коммутируемых соединений PPP осуществляется рядом вспомогательных программ. Dial-up IP (dip) – распространенный пакет, упрощающий процесс дозвона до удаленного сервера, регистрацию на нем и передачу соединения под контроль PPP. В этом разделе мы опишем dip, поскольку пакет является популярным и бесплатно доступен для широкого спектра систем Unix, ну и, кроме того, поставляется в составе системы Red Hat Linux, которую мы используем для создания примеров применения PPP.

Одной из наиболее важных возможностей dip является автоматизация процесса создания действующего PPP-канала при помощи языка сценариев. В приложении A рассмотрены все команды языка сценариев, существующие в dip версии 3.3.7o-uri, поставляемой в составе Red Hat. Чтобы получить перечень существующих команд, запустите dip в режиме тестирования (ключ -t) и наберите команду help:

```
> dip -t
DIP: Dialup IP Protocol Driver version 3.3.7o-uri (8 Feb 96)
```

```
Written by Fred N. van Kempen, MicroWalt Corporation.

DIP> help
DIP knows about the following commands:

        beep     bootp    break    chatkey  config
        databits dec      default  dial     echo
        flush    get      goto     help     if
        inc      init     mode     modem    netmask
        onexit   parity   password proxyarp print
        psend    port     quit     reset    send
        shell    sleep    speed    stopbits term
        timeout  wait
 DIP> quit
```

Перечисленные команды позволяют настраивать интерфейс, контролировать выполнение сценария и обрабатывать ошибки. Простейшие сценарии требуют использования достаточно ограниченного подмножества команд:

```
# Запрашиваем локальный адрес IP у PPP
get $local 0.0.0.0
# Выбираем порт и устанавливаем скорость линии
port cua1
speed 38400
# Аппаратная инициализация модема и очистка буфера ввода
reset
flush
# Набираем номер PPP-сервера и ожидаем ответа CONNECT
dial *70,301-555-1234
wait CONNECT
# Делаем паузу в 2 секунды, чтобы сервер успел подготовиться
sleep 2
# Передаем символ возврата каретки, чтобы «разбудить» сервер
send \r
# Ожидаем приглашения Login> и передаем имя пользователя
wait ogin>
send kristin\r
# Ожидаем приглашения Password> и передаем пароль
wait word>
password
# Ожидаем приглашения командной строки сервера PPP
wait >
# Передаем команду, регламентированную сервером PPP
send ppp enabled\r
# Переводим интерфейс в режим PPP
mode PPP
# Завершаем работу сценария
exit
```

Команда `get` в начале сценария позволяет PPP автоматически определить локальный и удаленный адреса. `$local` – это переменная сценария. Существует несколько стандартных переменных для сценариев, все они описаны в приложении А. `$local` обычно хранит локальный адрес, который может ста-

тически определяться в сценарии. Однако сервер PPP обладает способностью динамически назначить адрес локальной системе. Этим обстоятельством мы и пользуемся, указав локальный адрес, состоящий из одних нулей. Данная команда предписывает `dip` разрешить `pppd` самостоятельно определить адреса. Клиент `pppd` может получить адреса тремя путями:

- Системы PPP могут обмениваться локальными адресами, извлеченными при помощи DNS. Такой вариант мы уже обсуждали в разговоре о настройке выделенных линий.
- Адреса могут указываться в командной строке `pppd` – и этот вариант мы также изучили.
- Клиент может разрешить серверу назначить оба адреса. Эта возможность часто используется при работе с коммутируемыми линиями. Она весьма востребована серверами, работающими с многочисленными непродолжительными соединениями. Хорошим примером является сервер поставщика услуг Интернета, предоставляющий коммутируемый доступ.

Следующие две строки сценария указывают физическое устройство, с которым связан модем, и устанавливают скорость его работы. Команда `port` подразумевает путь */dev*, поэтому в качестве аргумента указано только само имя устройства. В большинстве систем PC Unix аргумент команды `port` принимает одно из значений: cua0, cua1, cua2 и cua3. Эти значения соответствуют портам системы MS-DOS – от COM1 до COM4. Команда `speed` определяет максимальную скорость передачи данных модему через указанный порт. Скорость по умолчанию – 38400. Измените аргумент, если модем принимает данные с иной скоростью.

Команда `reset` выполняет аппаратную инициализацию, передавая модему Hayes-команду прерывания (+++) и Hayes-команду сброса (ATZ). Данный вариант `dip` использует AT-команды системы Hayes и работает только с Hayes-совместимыми модемами.[1] По счастью, это определение охватывает большинство существующих модемов. После инициализации модем генерирует ответ, отмечая свою готовность принимать данные. Команда `flush` удаляет это сообщение и все другие сообщения, которые могли быть отображены модемом, из очереди ввода. Используйте `flush`, чтобы избавиться от потенциальных проблем, связанных с наличием непредусмотренных данных в очереди.

Следующая команда осуществляет подключение к удаленному номеру. Команда `dial` передает модему стандартную Hayes-команду набора номера – ATD. Вместе с командой передается вся строка аргумента. Команда `dial` из примера передает модему строку ATD*70,301-555-1234. Модем набирает *70 (код отключения ожидания звонка[2]), а затем междугородный код 301,

---

[1] Если модем понимает лишь часть Hayes-команд, избегайте применения таких команд `dip`, как `reset` и `dial`, поскольку они генерируют Hayes-команды. Воспользуйтесь командой `send`, которая позволяет передавать модему любую желаемую строку.

[2] Отключайте ожидание звонка, прежде чем приступать к созданию соединения PPP. Различные местные телефонные операторы предлагают различные коды, позволяющие отключать ожидание звонка.

номер АТС 555 и номер 1234. Успешно установив соединение с удаленным модемом, локальный модем отображает сообщение CONNECT. Команда `wait` предписывает ожидать такого сообщения от модема.

Команда `sleep 2` вставляет в сценарий двухсекундную задержку. Зачастую бывает полезно сделать паузу перед регистрацией, чтобы позволить удаленному серверу инициализировать сеанс. Помните, что сообщение CONNECT исходит от модема, а не от удаленного сервера. Удаленному серверу может понадобиться выполнить ряд действий, прежде чем он будет готов принимать данные. Небольшая задержка позволяет избежать появления необъяснимых, невоспроизводимых ошибок.

Команда `send` передает второй стороне символ возврата каретки (`\r`). Когда установлено модемное соединение, все данные, передаваемые локальной системой, попадают напрямую в удаленную систему. Команда `send` позволяет передать любую строку. В нашем примере удаленный сервер должен получить символ возврата каретки, прежде чем отобразит первое приглашение. Символ возврата каретки записывается как `\r`, а символ новой строки – как `\n`.

Удаленный сервер отображает приглашение `Login>`, предлагая указать имя пользователя. Команда `wait ogin>` позволяет обнаружить это приглашение, а команда `send kristin` передает в ответ имя пользователя, `kristin`. Сервер запрашивает пароль, отображая приглашение `Password>`. Команда `password` предлагает локальному пользователю набрать пароль вручную. Пароль может храниться в команде `send` сценария, но такой способ потенциально опасен – если неуполномоченное лицо получит доступ к сценарию, оно получит доступ и к паролю. Команда `password` призвана повысить уровень безопасности.

Если пароль принят, удаленный сервер из нашего примера предлагает начать работу, отображая символ «больше» (`>`). Многие серверы требуют выполнения определенных команд, указывающих корректный режим работы. Сервер из нашего примера поддерживает несколько протоколов, так что мы должны предписать ему использование PPP, передав соответствующую команду при помощи `send`.

Завершается сценарий рядом команд, выполняющих настройку среды локального узла. Команда `mode` предписывает локальному узлу использовать протокол PPP для работы с этим каналом. Выбранный протокол должен соответствовать протоколу удаленного сервера. Существуют следующие значения протоколов для команды `dip mode`: SLIP, CSLIP, PPP и TERM. SLIP и CSLIP – это варианты протокола SLIP, о котором мы уже говорили. TERM – режим эмуляции терминала. PPP – протокол Point-to-Point (точка-точка). Наконец, команда `exit` завершает работу сценария, тогда как `dip` продолжает работу в фоновом режиме и обслуживает канал.

Этот простой сценарий действительно работает и дает полное представление о структуре ожидание/передача сценария `dip`. Однако ваши сценарии, вероятно, будут не столь простыми. Описанный сценарий неустойчив, поскольку не реализует проверку ошибок. Не получив ожидаемого ответа, сценарий просто повисает. Для решения проблемы можно воспользоваться интерва-

лом ожидания для каждой команды wait. К примеру, команда wait OK 10 предписывает системы ожидать ответа OK в течение 10 секунд. Когда ответ OK получен, переменная сценария $errlvl получает нулевое значение и выполнение сценария продолжается со следующей команды. Если ответ OK не получен до завершения десятисекундного интервала ожидания, $errlvl получает ненулевое значение, и сценарий переходит к следующей команде. Сочетание переменной $errlvl с командами if и goto позволяет реализовать обработку ошибок в сценариях dip. Более подробная информация по этой теме содержится в приложении А.

Готовый сценарий исполняется по команде dip. Предположим, приведенный выше сценарий хранится в файле *start-ppp.dip*. Следующая команда выполняет сценарий и создает канал PPP, соединяющий локальную систему и удаленный сервер:

```
> dip start-ppp
```

Завершается соединение PPP по команде dip -k. Команда закрывает соединение и принудительно завершает фоновый процесс dip.

Параметры pppd не хранятся в сценариях dip. dip всего лишь создает соединение PPP, но не занимается настройкой pppd. Параметры pppd хранятся в файле */etc/ppp/options*.

Имея в виду приведенный выше сценарий dip, мы можем воспользоваться следующими параметрами pppd:

```
noipdefault
ipcp-accept-local
ipcp-accept-remote defaultroute
```

Параметр noipdefault запрещает клиенту определять свой адрес при помощи локальных служб. ipcp-accept-local предписывает клиенту получить свой адрес от второй стороны. Параметр ipcp-accept-remote предписывает системе получить удаленный адрес от удаленной системы. Наконец, pppd предписывает использовать канал PPP в качестве маршрута по умолчанию. Именно этот параметр defaultroute мы уже встречали ранее в командной строке pppd. Любой параметр командной строки pppd может храниться в файле */etc/ppp/options*. В этом случае он принимается во внимание всякий раз, когда запускается демон pppd, в том числе и по вызову сценария dip.

Я использую dip на своем домашнем компьютере, чтобы настраивать коммутируемое PPP-подключение.[1] Лично я нахожу dip простым и легким в применении, отчасти потому, что знаком с языком сценариев dip. Кому-то больше понравится вспомогательная программа chat, входящая в состав пакета pppd.

---

[1] Для меня коммутируемое PPP-соединение – просто резервный вариант. Как многие другие люди, я работаю через высокоскоростное подключение. Однако DSL и кабельные модемы не требуют особой настройки, поскольку интерфейсом для большинства каналов DSL и кабельных модемов является Ethernet.

chat

Сценарий chat состоит из пар ожидание/передача, определяющих строки, ожидаемые системой, и строки, передаваемые в ответ. В действительности chat не имеет языка сценариев, но предоставляет ряд специальных символов, позволяющих создавать более сложные сценарии. Ниже приведен сценарий chat, выполняющий те же самые функции, что и сценарий dip, который мы изучали в предшествующем разделе:

```
'' ATZ
OK ATDT*70,301-555-1234
CONNECT \d\d\r
ogin> kristin
word> Wats?Wat?
> 'set port ppp enabled'
```

Каждая строка сценария начинается ожидаемой строкой, а заканчивается строкой, которая передается в качестве ответа. Модем не посылает строку, пока не получит соответствующую команду. Первая строка сценария имеет смысл «не ожидать ничего, передать модему команду инициализации». Пара одинарных кавычек (`''`) в начале строки указывает программе chat, что не следует ожидать получения какой-либо строки. Выполнив команду инициализации, сценарий ожидает получения от модема приглашения OK, а затем набирает номер удаленного сервера. Когда модем отображает сообщение CONNECT, сценарий выжидает две секунды (\d\d), а затем передает символ возврата каретки (\r). Каждый специальный символ \d предписывает паузу длительностью в одну секунду. Специальный символ \r обозначает возврат каретки. В chat существует довольно много специальных символов, которые могут использоваться в строках ожидания и передачи.[1] Сценарий завершается передачей имени пользователя, пароля, а также команды настройки удаленного сервера – в ответ на соответствующие приглашения.

Наберите сценарий в своем любимом текстовом редакторе и сохраните в файле, назвав его, например *dial-server*. Проверьте сценарий при помощи ключа -V команды chat, который предписывает комментировать выполнение сценария в стандартном потоке ошибок:

```
% chat -V -f dial-server
```

Выполнения сценария chat не достаточно для настройки канала PPP. Чтобы выполнить поставленную задачу, команду следует использовать в сочетании с pppd. Параметр командной строки connect позволяет запустить pppd и выполнить сценарий подключения в одной команде:

```
# pppd /dev/cua1 56700 connect "chat -V -f dial-server" \
     nodetach crtscts modem defaultroute
```

[1] Более подробная информация содержится в приложении A.

Команда chat, указанная в качестве аргумента параметра connect, используется для коммутируемого подключения и регистрации. Здесь может применяться любой пакет, способный выполнить подобные действия; не обязательно chat.

Команда pppd имеет и другие параметры, используемые для клиентов коммутируемых PPP-подключений. Параметр modem предписывает pppd отслеживать наличие несущей частоты (DCD) модема. Указатель DCD сообщает pppd, что соединение установлено или разорвано. pppd следит за DCD, чтобы определить, когда удаленный сервер повесит трубку. Параметр nodetach запрещает pppd отсоединение от терминала и работу в фоновом режиме. Это необходимо только в случае, когда команда chat выполняется с ключом -V. Завершив отладку сценария chat, удалите ключ V из подкоманды chat и параметр nodetach из команды pppd. Альтернативой является применение ключа -v команды chat. -V не требует изменения режима работы pppd, поскольку диагностика chat передается демону syslogd, а не в поток ошибок. Все прочие параметры этой командной строки мы уже встречали ранее.

Безопасность демона PPP

Серьезным преимуществом PPP перед SLIP являются более надежные механизмы обеспечения безопасности. Чтобы повысить безопасность, добавьте следующие параметры pppd в файл */etc/ppp/options*:

```
lock
auth
usehostname domain wrotethebook.com
```

Первый параметр, lock, предписывает pppd использовать файлы блокировки в стиле UUCP. Это предотвращает вмешательство в PPP-соединения со стороны других приложений, таких как UUCP или эмулятор терминала. Параметр auth требует авторизации второй стороны для создания канала PPP. Следуя этому указанию, локальная система запрашивает данные аутентификации у второй стороны. Вторая сторона при этом может и не запрашивать аналогичные данные у локальной системы. Если администратор удаленной системы требует аутентификации вашей системы, он должен использовать ключевое слово auth в настройках своей системы. Параметр usehostname предписывает использовать имя узла в процессе аутентификации и запрещает пользователю устанавливать произвольное имя локальной системы при помощи параметра name. (Очень скоро мы еще вернемся к вопросу проверки подлинности.) Последний параметр дополняет используемое в процессе проверки подлинности локальное имя до абсолютного – указанием доменного имени.

Вспомните, что параметры файла *~/.ppprc* и параметры командной строки pppd имеют приоритет более высокий, чем те, что содержатся в файле */etc/ppp/options*. Такая ситуация может отрицательно повлиять на безопасность. По этой причине некоторые параметры – включая только что перечисленные, будучи определенными в файле */etc/ppp/options*, уже не могут быть изменены.

pppd поддерживает два протокола проверки подлинности: протокол аутентификации с предварительным согласованием вызова CHAP (Challenge Handshake Authentication Protocol) и протокол аутентификации пароля PAP (Password Authentication Protocol). PAP – это простая система парольной безопасности, уязвимая для всех видов атак на пароли многократного применения, тогда как CHAP предоставляет более совершенные механизмы проверки подлинности, не требующие использования паролей многократного применения и периодически выполняющие аутентификацию удаленной системы.

В процессе проверки подлинности используются два файла: */etc/ppp/chap-secrets* и */etc/ppp/pap-secrets*. Приведенные выше параметры предписывают pppd прежде всего попытаться аутентифицировать удаленную систему посредством CHAP. Такая проверка требует присутствия определенной информации в файле *chap-secrets*, и удаленная система должна ответить на вызов CHAP. Если не выполнено хотя бы одно из условий, pppd пытается идентифицировать удаленную систему посредством PAP. Если отсутствует подходящая запись в файле *pap-secrets*, либо удаленная система не ответила на вызов PAP, попытка установить соединение PPP заканчивается неудачей. Такой механизм позволяет выполнять проверку подлинности удаленных систем посредством CHAP (предпочтительного протокола), если возможно, или использовать PAP в качестве запасного варианта для систем, поддерживающих только PAP. Однако чтобы этот механизм заработал, необходимо поместить корректные записи в оба файла.

Каждая запись файла *chap-secrets* содержит до четырех полей:

клиент

Имя компьютера, который должен ответить на вызов, то есть компьютера, для которого необходимо произвести аутентификацию перед созданием соединения. Речь не обязательно идет о клиенте, обращающемся к серверу PPP: в большинстве документов используется термин *клиент*, хотя на деле это просто система-респондент – та, которая отвечает на вызов. Обе стороны канала PPP могут принудительно проходить проверку подлинности. В вашем файле *chap-secrets*, вероятно, будет по две записи на каждую удаленную систему: одна запись для аутентификации этой системы, и еще одна, позволяющая проходить идентификацию по требованию этой системы.

сервер

Имя системы, от которой исходит вызов CHAP, то есть компьютера, который требует аутентификации второй стороны перед созданием соединения PPP. Речь не обязательно идет о сервере PPP. Система-клиент вполне может потребовать предоставления данных идентификации у сервера. Термин *сервер* используется в большинстве документов, хотя на деле это система, проверяющая допустимость ответа.

секрет

Секретный ключ, которым шифруется строка вызова при передаче системе, от которой исходил вызов.

адрес

Адрес, представленный именем узла или IP-адресом, приемлемый для узла, указанного в первом поле. Если узел-клиент попытается воспользоваться другим адресом, соединение будет разорвано, даже если клиент правильно зашифровал ответ на вызов. Данное поле является необязательным.

Файл *chap-secrets* узла *ring* может иметь следующий вид:

```
limulus     ring        Peopledon'tknowyou    172.16.15.3
ring        limulus     andtrustisajoke.      172.16.15.1
```

Первая запись используется для проверки узла *limulus* удаленного сервера PPP. Проверяется подлинность узла *limulus*, проверка выполняется системой ring. Секретный ключ представлен строкой «Peopledon'tknowyou». Допустимый адрес – 172.16.15.3, то есть адрес, назначенный системе limulus в таблице узлов. Вторая запись позволяет проверить узел ring, когда вызов исходит от системы *limulus*. Секретный ключ – «andtrustisajoke.». Узлу ring разрешено использовать только адрес 172.16.15.1. Пара записей, по одной на каждую сторону канала, – нормальное явление. Файл *chap-secret* обычно содержит две записи на каждый канал PPP: одну для проверки подлинности удаленной системы, и вторую – для ответов на вызовы, исходящие от удаленной системы.

Используйте PAP только по необходимости. Если речь идет о системе, не поддерживающей CHAP, создайте для этой системы запись в файле *pap-secrets*. Формат записей файла *pap-secrets* точно такой же, как формат записей *chap-secrets*. Система, не работающая с CHAP, может хранить следующие записи в файле *pap-secrets*:

```
24seven  ring       Wherearethestrong?  24seven.wrotethebook.com
ring     24seven    Whoarethetrusted?   ring.wrotethebook.com
```

Здесь снова пара записей: одна для удаленной и одна для локальной системы. Мы поддерживаем CHAP, а удаленная система – нет. Таким образом, мы должны иметь возможность ответить по протоколу PAP, если вторая сторона запросит данные идентификации.

Аутентификация PPP повышает уровень безопасности при работе по коммутируемым соединениям. Наибольшую актуальность она приобретает для серверов PPP, обслуживающих удаленные системы. В следующем разделе мы рассмотрим настройку сервера PPP.

Настройка сервера PPP

Сервер PPP может запускаться рядом способов. Первый способ – использовать `pppd` в качестве оболочки, вызываемой при регистрации пользователей, подключающихся по PPP. Замените имя командного интерпретатора в файле */etc/passwd* полным именем `pppd`, чтобы запустить сервер. Измененная запись */etc/passwd* может выглядеть так:

```
craig:wJxX.iPuPzg:101:100:Craig Hunt:/etc/ppp:/usr/sbin/pppd
```

Здесь все поля соответствуют стандартной записи файла */etc/passwd*: имя пользователя, пароль, идентификатор пользователя, идентификатор группы, код gcos, исходный каталог, оболочка (интерпретатор команд), вызываемая после регистрации. Исходным каталогом для внешнего пользователя PPP является */etc/ppp*, а в поле оболочки содержится полное имя программы `pppd`. Зашифрованный пароль должен быть указан при помощи программы `passwd`, точно так же, как для любого пользователя, и процесс регистрации в системе ничем не отличается от стандартного. Обнаружив данные, поступающие через последовательный порт, `getty` выполняет `login` в целях аутентификации пользователя. `login` проверяет введенное пользователем имя и пароль и запускает интерпретатор команд. В данном случае вместо интерпретатора запускается демон PPP.

При таком способе запуска сервера параметры настройки, как правило, размещаются в файле */etc/ppp/.ppprc*. `login` проверяет подлинность пользователя, а `pppd` выполняет аутентификацию клиента. Поэтому нужно настроить файл *chap-secrets* или *pap-secrets* для работы с теми системами, из которых пользователи соединяются с сервером.

Традиционной альтернативой запуску `pppd` посредством `login` является создание сценария, в котором `pppd` – лишь одна из команд. Например, можно создать сценарий */etc/ppp/ppplogin* следующего содержания:

```
#!/bin/sh
mesg -n
stty -echo
exec /sbin/pppd auth passive crtscts modem
```

Понятно, такой сценарий может содержать несколько команд. Команда `mesg -n` запрещает другим пользователям передавать данные на этот терминал посредством программ `talk`, `write` или других подобных. Команда `stty` отключает эхо-контроль символов. В некоторых системах символы, набираемые на терминале, отображаются эхом от удаленного узла, а не напрямую на локальном терминале. Такой режим называется полным дуплексом. В нем нет необходимости в случае канала PPP, так что мы его отключаем. Управление характеристиками физического канала – основная причина размещения команды `pppd` в отдельном файле сценария.

Ключевая строка сценария, разумеется, та, что содержит вызов `pppd`. Демон запускается с рядом параметров, но одного параметра здесь не хватает, а именно имени устройства терминала. Во всех предшествующих примерах это имя присутствовало в строке команды `pppd`. Если имя отсутствует, как в этом случае, `pppd` использует в качестве такого устройства управляющий терминал и не переходит в фоновый режим работы. Именно это нам нужно. Мы желаем воспользоваться тем устройством, которое было связано с командой `login`, когда она вызвала сценарий `ppplogin`.

Параметр командной строки `auth` предписывает `pppd` выполнять аутентификацию удаленной системы, что, разумеется, требует присутствия записи для этой системы в файле *chap-secrets* или *pap-secrets*. Параметр `crtscts` включа-

ет аппаратный контроль передачи, а параметр modem предписывает PPP отслеживать состояние индикатора DCD модема, чтобы вовремя обнаружить разрыв связи удаленной системой. Все эти параметры мы уже встречали ранее. Но один из параметров, passive, нам не знаком. Параметр passive предписывает локальной системе ожидать корректного LCP-пакета от удаленной системы, даже если удаленная система никак не отреагировала на первый пакет. В обычной ситуации локальная система разрывает соединение, не получив своевременного ответа. Данный параметр дает удаленной системе время на инициализацию собственного демона PPP.

Наконец, есть еще один вариант запуска сервера PPP – разрешить пользователям запускать сервер из приглашения интерпретатора команд. С этой целью демон pppd должен выполняться с полномочиями пользователя root (по умолчанию не так). В таком случае пользователь, вошедший в систему по стандартной учетной записи, может выполнить команду:

```
$ pppd proxyarp
```

которая запустит демон PPP. Если параметр auth существует в файле */etc/ppp/options*, то pppd выполняет аутентификацию удаленного клиента посредством CHAP или PAP. Когда подлинность клиента установлена, демон создает представительскую ARP-запись для клиента в таблице ARP сервера, что позволяет другим системам локальной сети видеть клиента в качестве равноправного узла.

Из этих трех подходов я предпочитаю создание сценария, вызываемого программой login в качестве интерпретатора команд пользователя. Это избавляет от необходимости давать pppd полномочия суперпользователя и возлагать запуск сервера на пользователя. При этом доступна вся мощь команды pppd и все возможности программирования на языке интерпретатора команд.

PPP в Solaris

dip и pppd существуют для Linux, BSD, AIX, Ultrix, OSF/1 и SunOS. Работая с иной операционной системой, вы, вероятно, столкнетесь с другими пакетами. Хорошим примером системы, в которой для настройки PPP применяется иной набор команд, является Solaris.

Протокол PPP в Solaris реализован демоном aspppd (Asynchronous PPP Daemon). Настройки aspppd хранятся в файле */etc/asppp.cf*. Файл *asppp.cf* состоит из двух разделов: раздела ifconfig и раздела path.

```
ifconfig ipdptp0 plumb ring limulus up

path
   interface ipdptp0
   peer_system_name limulus    inactivity_timeout 300
```

Команда ifconfig производит настройку интерфейса PPP (ipdptp0) как канала точка-точка с локальным адресом узла *ring* и конечным адресом узла *limulus*. В команде ifconfig не обязательно указывать конечный адрес канала. Однако если вы всегда подключаетесь к одном и тому же удаленному серве-

ру, имеет смысл указать его адрес именно в этой команде. Все присутствующие здесь параметры `ifconfig` мы обсудили ранее в этой главе.

Больший интерес представляет раздел `path`, определяющий характеристики среды PPP. Оператор `interface` указывает интерфейс, служащий для создания соединений. Здесь требуется указать один из интерфейсов PPP, определенных в разделе `ifconfig`. В нашем примере определен только один интерфейс – `ipdptp0`. Оператор `peer_system_name` определяет систему, выступающую в роли второй стороны. Адрес может совпадать с конечным адресом из оператора `ifconfig`, но это не обязательно. Конечный адрес в команде `ifconfig` может вообще отсутствовать, а разделов `path` может быть несколько, если необходимо работать с рядом удаленных систем. Как мы увидим далее, имя узла из оператора `peer_system_name` используется в процессе подключения.

Раздел `path` завершается оператором `inactivity_timeout`. Команда примера устанавливает интервал ожидания в 300 секунд. Одно из достоинств системы Solaris в действии. Solaris автоматически подключается к удаленной системе при обнаружении данных, которые должны быть ей доставлены, а затем автоматически разрывает соединение PPP, если отсутствует активность канала в течение указанного интервала времени. Такой механизм избавляет от необходимости запускать программу набора номера вручную, а затем сбрасывать телефонную линию, если канал не используется.

Подобно `pppd`, `aspppd` не имеет встроенной функции для набора номеров и полагается на внешние программы. `aspppd` использует средства дозвона пакета UUCP следующим образом.

Последовательный порт, подключенный к нему модем и скорость их работы определяются в файле */etc/uucp/Devices*. Например, ниже мы создаем автоматическое вызывающее устройство (Automatic Call Unit, ACU – всего лишь другое название модема), подключенное к последовательному порту B (cua/b), работающему на любой скорости из указанных в файле *Systems*. Характеристики модема определяются записью «hayes» в файле *Dialers*.

```
ACU cua/b - Any hayes
```

Затем мы определяем характеристики модема, такие как строка инициализации и команда набора, в файле */etc/uucp/Dialers*. Команды инициализации и набора определяются в качестве `chat`-сценария по стандартной схеме ожидания/передачи и посредством стандартного набора специальных символов `chat`. Например:

```
hayes =,-, "" \dA\pTE1V1X1QOS2=255S12=255\r\c OK\r \EATDT\T\r\c CONNECT
```

Файлы *Devices* и *Dialers* уже существуют в стандартной установке системы. Существующие записи, вероятно, совместимы с модемом вашей системы. Правка настроек в таком случае может ограничиваться лишь файлом */etc/uucp/Systems*. В файле *Systems* необходимо указать имя удаленной системы, выбрать модем, указать телефонный номер, а также создать сценарий `chat`, выполняющий регистрацию. Например:

```
limulus Any ACU 56700 5551234 "" \r ogin> kristin word> Wats?Watt? >      set ppp on
```

В единственной строке мы указываем *limulus* в качестве удаленной системы, объявляем, что разрешены входящие и исходящие соединения для этого канала в любое время суток (Any), выбираем запись ACU из файла *Devices*, указывая таким образом порт и модем, устанавливаем скорость линии в значение 56 700, передаем программе набора номера телефонный номер, а также определяем регистрационный сценарий `chat`.

Поскольку эта книга посвящена совсем не UUCP, мы не будем вдаваться в детали этих файлов. В поисках более подробной информации по UUCP и `aspppd` предлагаю обращаться к книгам Solaris AnswerBook и Solaris *TCP/IP Network Administration Guide* (*Администрирование сетей TCP/IP* Solaris – как им удалось придумать такое классное имя?).

Диагностирование последовательных соединений

Отладка соединений PPP затруднена проблемами, возникающими на нескольких уровнях. Установка PPP требует настройки последовательного порта, модема, настройки PPP и TCP/IP. Ошибка на любом из уровней может приводить к проблемам на другом. Любой из уровней может скрыть истинную причину проблемы. Лучший способ диагностирования проблем для последовательных линий – применять отладку на каждом из уровней по очереди. Обычно к настройке очередного уровня следует переходить после отладки предыдущего.

Физические последовательные порты должны настраиваться системой в процессе загрузки. Проверьте корректность настроек, обратившись к каталогу */dev*. Для системы Linux с четырьмя последовательными портами должны существовать входящие последовательные порты с */dev/ttyS0* по */dev/ttyS3* и исходящие последовательные порты с */dev/cua0* по */dev/cua3*. Имен устройств tty и cua существует гораздо больше. Однако прочие устройства связаны с реально существующими только в случаях присутствия в системе многопортовой последовательной карты. В большинстве систем используются имена tty и cua, даже если эти имена являются всего лишь символическими ссылками на реальные устройства. Например, в Solaris:

```
% ls -l /dev/tty?
lrwxrwxrwx 1 root root 6 Sep 23  2001 /dev/ttya -> term/a
lrwxrwxrwx 1 root root 6 Sep 23  2001 /dev/ttyb -> term/b
% ls -l /dev/cua/*
lrwxrwxrwx 1 root root 35 Sep 23 2001 /dev/cua/a ->
     /devices/obio/zs@0,100000:a,cu
lrwxrwxrwx 1 root root 35 Sep 23 2001 /dev/cua/b ->   /devices/obio/zs@0,100000:b,cu
```

Если последовательные устройства не обнаружены в каталоге */dev*, их можно добавить вручную при помощи команды `mknod`. Так, следующая команда создает последовательные устройства для первого последовательного порта системы Linux:

```
# mknod -m 666 /dev/cua0 c 5 64
# mknod -m 666 /dev/ttyS0 c 4 64
```

При добавлении последовательных устройств пользователем могут возникать проблемы с настройкой ядра. По умолчанию последовательные устройства должны устанавливаться системой при загрузке – в процессе обнаружения аппаратных устройств. Следующее загрузочное сообщение отражает обнаружение одного последовательного интерфейса в системе Linux:

```
$ dmesg | grep tty
ttyS00 at 0x03f8 (irq = 4) is a 16550
```

При загрузке системы должны наблюдаться аналогичные сообщения для всех обнаруженных интерфейсов. В противном случае возможно наличие аппаратных проблем, связанных с картой последовательного интерфейса.

Модем, позволяющий установить соединение, подключается к одному из последовательных портов. Прежде чем приступить к созданию сценария подключения, убедитесь, что модем работает и отзывается через последовательный порт. Воспользуйтесь простым коммуникационным пакетом, таким как `minicom`, `kermit` или `seyon`. Прежде всего, программу следует настроить на использование модема. Необходимо указать корректный порт, скорость, четность, число битов данных и т. д. Сверьтесь с документацией своего модема, чтобы определить эти значения.

Для создания примеров мы используем `minicom` и систему Linux. Чтобы настроить `minicom`, станьте суперпользователем при помощи команды `su` и выполните `minicom` с ключом `-s`, предписывающим вывод меню настройки. По необходимости измените значения параметров. Можно заметить, что значение порта в настройках – */dev/modem*. Это имя устройства иногда является символической ссылкой на порт подключения модема. Если вы точно не знаете, существует ли такая ссылка в системе, введите в этом поле корректное имя устройства, например */dev/cua1*. Убедившись в точности настроек, покиньте меню и воспользуйтесь эмулятором терминала `minicom`, чтобы проверить работоспособность модема:

```
Welcome to minicom 1.83.1

OPTIONS: History Buffer, F-key Macros, Search History Buffer, I18n
Compiled on Feb 23 2001, 07:31:40.

Press CTRL-A Z for help on special keys

AT S7=45 S0=0 L1 V1 X4 &c1 E1 Q0
OK
atz
OK
atdt555-1234
CONNECT 26400/LAPM-V
^M
Enter login> kristin
Enter user password> Wats?Watt?

   Welcome to the PPP MODEM POOL
```

```
PORT-9> set port ppp enabled
+++
OK
ath
OK
atz
OK
^A
CTRL-A Z for help | 57600 8N1 | NOR | Minicom 1.83.1 | VT102 | Offline
X
```

В этом примере `minicom` выводит несколько строк заголовка, а затем посылает модему Hayes-команду (AT). Мы ее не вводили, это часть стандартной настройки программы `minicom`. (Если выполнение команды вызывает проблемы, удалите ее при помощи вышепомянутого меню настройки.) Затем мы выполняем аппаратную инициализацию модема (`atz`) и набираем номер удаленного сервера (`atdt`). Когда модемы установили соединение, регистрируемся на сервере и настраиваем его. (Процедура регистрации своя для каждого удаленного сервера, это просто пример.) Похоже, что все замечательно работает, так что мы завершаем соединение – привлекаем внимание модема (+++) и вешаем трубку (`ath`), а затем заново инициализируем модем. Работа с `minicom` завершается нажатием <Ctrl>+<A> и затем <X>. Порт и модем нашей системы отлично работают. Если вы не можете передать модему даже простую команду, проверьте, что:

- модем правильно подключен к порту;
- использованы нужные кабели;
- модем включен в сеть питания;
- модем корректно настроен на исходящие звонки и эхо-отображение команд.

Если модем реагирует на простые команды, наберите номер удаленного сервера, как мы сделали в приведенном выше примере. Если модем отказывается набирать номер либо отображает сообщение NO DIALTONE, убедитесь, что телефонный кабель соединяет верный порт модема и телефонную розетку. Для проверки телефонной розетки можно использовать обычный аналоговый телефон. Если его нет под рукой, попробуйте заменить соединительный кабель. Если модем набирает номер, но не соединяется с удаленным модемом, убедитесь, что настройки локального модема соответствуют требованиям удаленной системы, которые необходимо знать для успешной отладки соединения. Ниже приводится перечень советов по отладке, в котором упомянут ряд моментов, подлежащих проверке. Успешно произведя подключение к удаленной системе, зафиксируйте все свои действия, а также весь вывод своего модема и удаленного сервера. Затем переведите удаленный сервер в режим PPP или SLIP и запишите, как этого удалось добиться. Все эти шаги необходимо будет воспроизвести в сценарии `dip`.

Начните с примитивного сценария, похожего на *start-ppp.dip*, чтобы иметь возможность убедиться в работоспособности базовых настроек, прежде чем

наращивать сложность сценария, добавляя обработку ошибок. Выполните сценарий посредством `dip` с ключом подробной диагностики (`-v`). Строки сценария будут отображаться по мере их выполнения. Обращайте внимание на следующие возможные проблемы:

- Модем не реагирует на команды сценария. Убедитесь, что в команде `port` используется корректное имя устройства. Убедитесь, что аргументы команд `databits`, `parity`, `speed` и `stopbits`, если таковые присутствуют в сценарии, являются допустимыми для установленного модема. Еще раз проверьте, что модем понимает команды системы Hayes, в особенности если настройка модема выполняется при помощи ключевых слов `dip`, а не команды `send`.
- Модем не может соединиться с удаленным узлом. Убедитесь, что настройки модема идентичны тем, что использовались при подключении вручную. Параметры модема, такие как `databits` и `parity`, должны соответствовать настройкам удаленной системы. Вполне возможно, что для регистрации в удаленной системе требуется временное переключение на сочетание 7-бит/контроль четности. После регистрации следует переключиться обратно на сочетание 8-бит/без контроля, которое требуется для работы PPP и SLIP. Не забудьте убедиться, что верен номер телефона, указанный в команде набора, особенно если модем выводит строки VOICE, RING – NO ANSWER либо BUSY вместо ожидаемого CONNECT.
- Сценарий зависает. Вероятно, он находится в ожидании ответа. Убедитесь, что корректны строки команд `wait`. Помните, что в качестве ожидаемой строки необходимо указывать лишь часть ответа. Лучше использовать строку «`>`», чем «`Port9>`», в случае, когда вы не уверены в номере порта, отображаемого удаленной системой. Используйте подстроку, которой заканчивается предполагаемый ответ, чтобы сценарий не начинал передачу данных прежде, чем будет готов к приему сервер. Кроме того, пробуйте вставлять задержку в сценарий, непосредственно перед первой командой серверу: к примеру, `sleep 2`, чтобы создать двухсекундную задержку. Задержка бывает необходима, чтобы сервер успел инициализировать порт, когда установлено модемное соединение.
- Удаленный сервер отображает сообщение об ошибке. Вероятно, сценарий посылает неверное значение. Проверьте строки всех команд передачи. Убедитесь, что они завершаются именно той комбинацией символов возврата каретки и новой строки, которая ожидается сервером.

Если возникли проблемы со сценарием, попробуйте выполнить `dip` в режиме тестирования (ключ `-t`), который позволяет вводить команды вручную, одну за другой. Повторяйте процедуру до тех пор, пока не получите четкой последовательности команд, необходимых для регистрации на удаленном сервере. Затем возвращайтесь к отладке сценария. Скорее всего, свежий взгляд на процесс регистрации позволит вам быстро выявить ошибку в сценарии.

Когда сценарий запущен, а соединение успешно установлено, ничто не должно препятствовать нормальной работе. Удаленный сервер должен отвечать на команду `ping`. Если здесь возникают проблемы, дело, вероятно, в на-

стройке IP-интерфейса или неверном маршруте по умолчанию. Сценарий должен создать последовательный интерфейс. Команда `netstat -ni` выводит статистику по уже настроенным интерфейсам:

```
# netstat -ni
Name  Mtu  Net/Dest      Address      Ipkts Ierrs Opkts Oerrs Collis Queue
dnet0 1500 172.16.15.0   172.16.15.1      1     0     4     0       0     0
lo0   1536 127.0.0.0     127.0.0.1     1712     0 1712     0       0     0
ppp0  1006 172.16.15.26 172.16.15.3       0     0     0     0       0     0
```

Интерфейс, в данном случае – `ppp0`, установлен. Команда сценария `default` создает маршрут по умолчанию. Воспользуйтесь `netstat` для просмотра содержимого таблицы маршрутизации:

```
# netstat -nr
Routing tables
Destination       Gateway           Flags  Refcnt  Use Interface
127.0.0.1         127.0.0.1           UH       1    28       lo0
default           172.16.25.3          U       0     0      ppp0
172.16.15.0       172.16.15.1          U      21 1687       le0
```

О содержимом таблиц маршрутизации мы подробно поговорим в следующей главе. Пока просто обратите внимание, что маршрут по умолчанию проходит через интерфейс `ppp0` и что маршрут по умолчанию является маршрутом к удаленному серверу PPP (в данном случае – 172.16.25.3).

Если сценарий создал соединение, установил интерфейс, и таблица маршрутизации содержит маршрут по умолчанию, все должно работать как часы. Если по-прежнему возникают проблемы, они могут быть связаны с другими подсистемами TCP/IP. В главе 13 подробно описан процесс диагностирования и отладки.

Резюме

TCP/IP работает с широким спектром физических сетей. Протоколы TCP/IP не могут угадывать возможности сетей – следует явно указывать сетевые интерфейсы и их характеристики. В этой главе мы рассмотрели ряд примеров, связанных с настройкой физического сетевого интерфейса, с которым работает TCP/IP.

Чаще всего для настройки интерфейсов применяется команда `ifconfig`. Она назначает интерфейсу IP-адрес, устанавливает маску подсети, широковещательный адрес, а также выполняет ряд других функций.

TCP/IP может работать и по коммутируемым телефонным каналам. Задача решается при помощи протоколов Serial Line IP (SLIP) и Point-to-Point Protocol (PPP). Предпочтительным вариантом является PPP – стандартный протокол Internet, обладающий повышенной надежностью, производительностью и безопасностью.

Создание соединения PPP состоит из ряда этапов: настройки протокола последовательного порта, настройки порта и модема, установки коммутируемого соединения, регистрации на удаленном сервере. Отдельные программы, такие как `dip`, объединяют все перечисленные этапы. Другие программы, такие как `pppd` и `chat`, напротив, делят функциональность.

Настройка сетевого интерфейса позволяет нам общаться с локальной сетью, в то время как настройка маршрутизации позволяет общаться с целым миром. Мы кратко затронули маршрутизацию в главе 2, а в этой главе – в разговоре о метриках маршрутизации для `ifconfig` и маршрутах по умолчанию в PPP. В следующей главе мы изучим маршрутизацию гораздо более подробно.

- *Варианты настройки маршрутизации*
- *Простейшая таблица маршрутизации*
- *Создание статической таблицы маршрутизации*
- *Протоколы внутренней маршрутизации*
- *Протоколы внешней маршрутизации*
- *Демон шлюзовой маршрутизации*
- *Настройка gated*

7

Настройка маршрутизации

Маршрутизация связывает Интернет воедино. Без маршрутизации трафик TCP/IP ограничивался бы единственной физической сетью. Маршрутизация позволяет данным из локальной сети достигать адресатов во всех концах света, проходя через многочисленные промежуточные сети.

Важность маршрутизации и сложность взаимосвязей сетей Интернета делают проектирование протоколов маршрутизации серьезным испытанием для разработчиков сетевых программ. Как следствие, в большинстве материалов по маршрутизации говорится об архитектуре протоколов. Очень немного написано о важной задаче правильной настройки протоколов маршрутизации. И это при том, что большая часть возникающих проблем может быть обусловлена неверной настройкой маршрутизаторов, а вовсе не плохо спроектированными алгоритмами маршрутизации. Задача системного администратора – убедиться в том, что маршрутизация на вверенных ему системах настроена правильно. Именно этой задаче посвящена настоящая глава.

Варианты настройки маршрутизации

Прежде всего, следует провести черту между *маршрутизацией* и *протоколами маршрутизации*. Все системы выполняют маршрутизацию данных, но далеко не на каждой функционирует протокол маршрутизации. *Маршрутизация* – это действие по пересылке дейтаграммы, основанное на информации из таблицы маршрутизации. *Протоколы маршрутизации* – это программы, которые обмениваются информацией для построения таблиц маршрутизации.

Настройка маршрутизации сети не обязательно требует присутствия протокола маршрутизации. В случаях, когда информация маршрутизации не изме-

няется – например, если существует лишь один возможный маршрут, – администратор системы, как правило, создает таблицу маршрутизации вручную. Некоторые сети не имеют доступа к другим сетям TCP/IP, а следовательно, не требуют от администратора даже создания таблицы маршрутизации. Вот три наиболее распространенных варианта настройки маршрутизации.[1]

Примитивная маршрутизация

Сеть, полностью изолированная от всех других сетей TCP/IP, требует лишь примитивной маршрутизации. Простейшая таблица маршрутизации обычно создается при настройке сетевых интерфейсов: на каждый из интерфейсов добавляется по одному маршруту. Если отсутствует прямой доступ к другим сетям TCP/IP и разделение на подсети, других таблиц маршрутизации может не потребоваться.

Статическая маршрутизация

В сети с ограниченным числом шлюзов в другие сети TCP/IP имеет смысл применять статическую маршрутизацию. Для сети с единственным шлюзом статическая маршрутизация станет идеальным выбором. Статическая таблица маршрутизации создается вручную администратором – при помощи команды `route`. Статические таблицы маршрутизации не способны реагировать на изменения в сети, поэтому они оптимальны для случаев, когда маршруты постоянны.

Динамическая маршрутизация

В сетях, где существует несколько путей к одному пункту назначения, следует применять динамическую маршрутизацию. Динамическая таблица маршрутизации создается на основе сведений, которыми обмениваются протоколы маршрутизации. Задача этих протоколов – распространение информации, позволяющей автоматически настраивать маршруты в случае изменений в сети. Протоколы маршрутизации справляются со сложными задачами быстрее и точнее, чем при всем желании способен это делать системный администратор. Протоколы маршрутизации позволяют не только переключаться на резервный маршрут, если становится непроходимым основной, но также выбирать «лучший» маршрут из нескольких доступных. Протоколы маршрутизации следует применять во всех сетях, где существуют альтернативные маршруты.

Маршруты создаются вручную системным администратором или динамически – протоколами маршрутизации. Но, независимо от способа создания, они, в конечном итоге, оказываются в таблице маршрутизации.

Простейшая таблица маршрутизации

Взглянем на содержимое таблицы маршрутизации, созданной в процессе настройки сетевых интерфейсов системы Solaris 8 посредством `ifconfig`:

[1] В главе 4 представлены руководящие указания по выбору варианта настройки маршрутизации для ваших сетей.

```
% netstat -rn
Routing Table: IPv4
  Destination           Gateway           Flags  Ref   Use   Interface
-------------------- -------------------- ----- ----- ------ ---------
172.16.12.0          172.16.12.15          U         1      8  dnet0
224.0.0.0            172.16.12.15          U         1      0  dnet0
127.0.0.1            127.0.0.1             UH       20   3577  lo0
```

Первая запись определяет маршрут в сеть 172.16.12.0, пролегающий через интерфейс dnet0. 172.16.12.15 – это не адрес внешнего шлюза, но адрес, назначенный интерфейсу dnet0 на данном узле. Оставшиеся две записи не являются определениями маршрутов; они отражают принятые для программного обеспечения соглашения. 224.0.0.0 – групповой адрес. Данная запись предписывает Solaris выполнять доставку групповых сообщений через интерфейс 172.16.12.15. Последняя запись определяет кольцевой маршрут к узлу *localhost*, созданный при настройке интерфейса lo0.

Обратите внимание на поле Flags этих записей. Для всех записей установлен флаг U (up), показывающий, что маршруты готовы к использованию, но ни разу не встречается флаг G (gateway). Флаг G отмечает внешние шлюзы. Флаг G в данном случае отсутствует потому, что все маршруты пролегают через локальные интерфейсы и ни один не проходит через внешний шлюз.

Кроме того, для кольцевого маршрута установлен флаг H (host), который указывает, что лишь один узел доступен по этому маршруту. Значение флага становится ясным, если взглянуть на поле Destination записи кольцевого маршрута. Оно содержит адрес конкретного узла, а не адрес сети. Адрес кольцевой сети – 127.0.0.0. Данный адрес пункта назначения (127.0.0.1) является адресом конкретного узла – *localhost*. В одних системах используется маршрут в кольцевую сеть, в других – маршрут к локальному узлу, но во всех системах в таблице маршрутизации существует некоторый маршрут для кольцевого интерфейса.

В таблице из примера существует один маршрут к узлу, но в большинстве случаев маршруты прокладываются к сетям. Первая причина тому – необходимость сократить размер таблицы маршрутизации. Единственная сеть организации может состоять из сотен узлов. Сеть Интернет состоит из тысяч сетей и из миллионов узлов. Таблица маршрутизации, содержащая маршрут для каждого из узлов, была бы неуправляема.

Приведенная таблица содержит лишь один маршрут в физическую сеть, 172.16.12.0. Следовательно, данная система может общаться только с узлами, расположенными в указанной сети. Ограниченные возможности этой таблицы маршрутизации легко увидеть при помощи команды `ping`. `ping` вынуждает удаленный узел вернуть пакет локальному узлу – при помощи ICMP-сообщения эхо (Echo Message). Прохождение пакетов до удаленного узла и обратно означает, что два узла могут успешно обмениваться данными.

Чтобы проверить возможности таблицы маршрутизации этой системы, прежде всего, выполните команду `ping` для другого узла локальной сети:

```
% ping -s crab
PING crab.wrotethebook.com: 56 data bytes
64 bytes from crab.wrotethebook.com (172.16.12.1): icmp_seq=0. time=11. ms
64 bytes from crab.wrotethebook.com (172.16.12.1): icmp_seq=1. time=10. ms
^C
----crab.wrotethebook.com PING Statistics----
2 packets transmitted, 2 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 10/10/11
```

ping выводит по одной строке на каждый полученный ответ ICMP ECHO_RESPONSE.[1] Когда выполнение ping прерывается, программа отображает сводную статистику. Мы наблюдаем успешный обмен данными с узлом *crab*. Но если мы обратимся к узлу, который не принадлежит сети 172.16.12.0, скажем, к узлу издательства O'Reilly, результат будет другим.

```
 % ping 207.25.98.2
sendto: Network is unreachable
```

В данном случае сообщение «sendto: Network is unreachable» показывает, что локальный узел не знает, как отправить данные в сеть узла 207.25.98.2. В таблице маршрутизации системы – лишь три маршрута, и ни один из них не ведет в сеть 207.25.98.0.

Данная таблица маршрутизации не позволяет общаться даже с машинами другой подсети *books-net*. Подтверждает сказанное команда ping, выполненная для узла другой подсети:

```
% ping 172.16.1.2
sendto: Network is unreachable
```

Эти тесты показывают, что простейшая таблица маршрутизации, созданная в процессе настройки сетевых интерфейсов, позволяет обмениваться данными лишь с другими узлами локальной сети. Если сети не требуется доступ к другим сетям TCP/IP, вполне можно обойтись и таким вариантом. В противном случае следует добавить прочие маршруты в таблицу маршрутизации.

Создание статической таблицы маршрутизации

Как мы видели, простейшая таблица маршрутизации позволяет работать лишь с узлами, расположенными в напрямую подключенных физических сетях. Обращение к удаленным узлам требует добавления в таблицу маршрутизации маршрутов, пролегающих через внешние шлюзы. Одним из решений задачи является создание статической таблицы маршрутизации при помощи команд route.

---

[1] Команда ping системы Sun отобразит лишь сообщение «*crab* is alive», если не использовать ключ -s. Большинство реализаций ping не требуют наличия ключа -s.

С помощью команды Unix route можно вручную добавить или удалить записи таблицы маршрутизации. Так, чтобы добавить маршрут 207.25.98.0 в таблицу маршрутизации системы Solaris, наберите:

```
# route add 207.25.98.0 172.16.12.1 1
add net 207.25.98.0: gateway crab
```

Первый аргумент команды route – ключевое слово add. Первым ключевым словом в командной строке route может быть add либо delete, и оно предписывает route соответственно добавить новый маршрут либо удалить существующий. Значения по умолчанию нет – в отсутствие ключевых слов route просто отображает таблицу маршрутизации.

Следующее значение – адрес пункта назначения, то есть адрес, доступный по этому маршруту. Конечный адрес может быть представлен IP-адресом, именем сети из файла */etc/networks*, именем узла из файла */etc/hosts* либо ключевым словом default. Поскольку добавление большинства маршрутов происходит достаточно рано в процессе загрузки системы, численные IP-адреса используются чаще, чем имена. Такое положение снимает зависимость настройки маршрутизации от доступности и состояния серверов имен. Всегда используйте полные численные адреса, состоящие из четырех байтов. route выполняет расширение адреса, содержащего меньше четырех байтов, и полученный результат может достаточно сильно отличаться от желаемого.[1]

Если конечный адрес представлен ключевым словом default, route создает *маршрут по умолчанию*.[2] Маршрут по умолчанию используется в случаях, когда отсутствует явный маршрут в конечную сеть. Зачастую маршрут по умолчанию – единственный необходимый маршрут. Если сеть работает с одним шлюзом, используйте маршрут по умолчанию, чтобы передавать через этот шлюз весь трафик, предназначенный внешним сетям.

Далее в командной строке route следует адрес шлюза[3], а именно IP-адрес внешнего шлюза, через который передаются данные. Шлюз должен располагаться в сети с прямым подключением. Маршруты TCP/IP определяют следующий транзитный участок в пути к пункту назначения. Этот следующий транзитный участок должен быть напрямую доступен локальному узлу; следовательно, он должен быть в сети с прямым подключением.

И последний аргумент в командной строке – метрика маршрутизации. Аргумент метрики отсутствует при удалении маршрутов, но в некоторых более старых системах его присутствие обязательно для добавления маршрута. В Solaris 8 метрика является необязательной. Системы, требующие наличия аргумента метрики, используют его, только чтобы определить, пролегает

---

1 Некоторые реализации route преобразуют «26» в 0.0.0.26, хотя «26» вполне может означать сеть Milnet (26.0.0.0).

2 С маршрутом по умолчанию связан адрес сети 0.0.0.0.

3 В Linux значения командной строки route предваряются ключевыми словами. Например route add -net 207.25.98.0 netmask 255.255.255.0 gw 172.16.12.1. Сверьтесь с документацией своей системы.

маршрут через напрямую подключенный интерфейс или же через внешний шлюз. В случае нулевой метрики маршрут создается как проходящий через локальный интерфейс, и флаг G, который мы наблюдали в выводе команды `netstat -i`, отсутствует. Если значение метрики больше нуля, для маршрута устанавливается флаг G, и адрес считается адресом внешнего шлюза. В статической маршрутизации метрики не используются по назначению. Меняющиеся значения метрики имеют смысл только в динамической маршрутизации.

Создание статических маршрутов

В качестве примера займемся настройкой статической маршрутизации гипотетической рабочей станции *rodent*. На рис. 7.1 представлена подсеть 172.16.12.0. В подсети существует два шлюза, *crab* и *horseshoe*. *crab* – это шлюз в тысячи сетей Интернета; *horseshoe* обеспечивает доступ к другим подсетям *books-net*. Используем узел *crab* в качестве шлюза по умолчанию, поскольку через него пролегают тысячи маршрутов. Число маршрутов через узел *horseshoe* ограничено, их можно создать вручную. Выбор шлюза по умолчанию диктуется именно числом маршрутов, пролегающих через шлюз, а вовсе не объемом трафика. Даже если большая часть трафика с узла *rodent* уходит через *horseshoe* к другим узлам *books-net*, шлюзом по умолчанию должен быть *crab*.

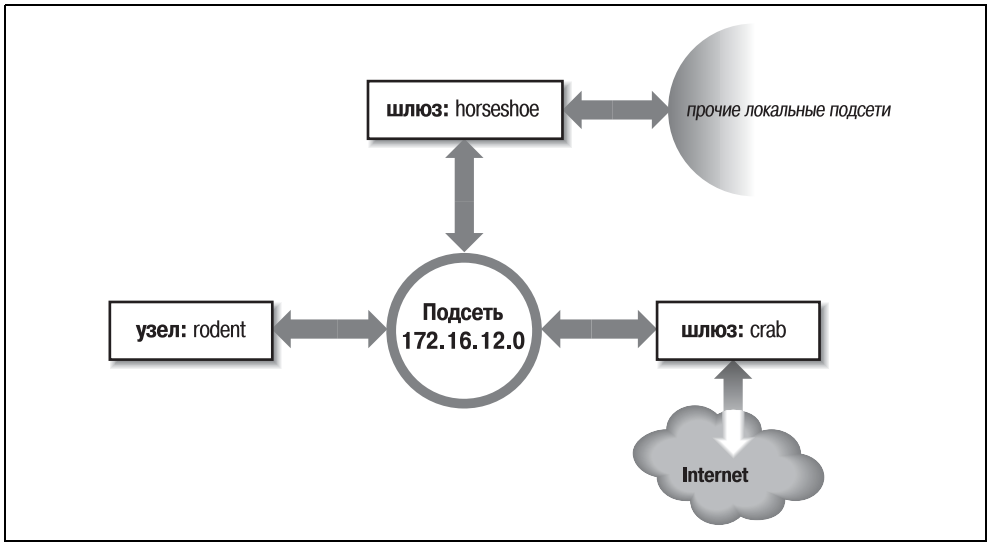


***Рис. 7.1.** Маршрутизация в подсети*

Чтобы создать маршрут по умолчанию на узле *rodent*, наберите:

```
# route add default gw 172.16.12.1
```

Конечный адрес представлен ключевым словом `default`, а в качестве адреса шлюза (172.16.12.1) выступает адрес узла *crab*. Теперь *crab* является шлю-

зом по умолчанию для узла *rodent*. Обратите внимание, что синтаксис команды отличается от описанного в предшествующем примере по route системы Solaris. Система *rodent* работает под управлением Linux. В Linux большинству аргументов команды route предшествуют ключевые слова. В данном случае адресу шлюза предшествует ключевое слово gw.

Создав маршрут по умолчанию, изучим таблицу маршрутизации, чтобы убедиться, что все в порядке:[1]

```
# route -n
Kernel IP routing table
Destination   Gateway       Genmask         Flags Metric Ref Use Iface
172.16.12.0   0.0.0.0       255.255.255.0   U     0      0     0 eth0
127.0.0.0     0.0.0.0       255.0.0.0       U     0      0     0 lo
0.0.0.0       172.16.12.1   0.0.0.0         UG    0      0     0 eth0
```

Снова воспользуемся командой ping, чтобы выяснить, может ли теперь *rodent* общаться с удаленными узлами. Если нам повезет[2], удаленный узел ответит, и мы увидим:

```
% ping 207.25.98.2
PING 207.25.98.2: 56 data bytes
64 bytes from ruby.ora.com (207.25.98.2): icmp_seq=0. time=110. ms
64 bytes from ruby.ora.com (207.25.98.2): icmp_seq=1. time=100. ms
^C
----207.25.98.2 PING Statistics----
2 packets transmitted, 2 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 100/105/110
```

Такой вывод свидетельствует об успешном обмене данными с удаленным узлом, то есть у нас теперь есть действующий маршрут в сеть Интернет.

Однако мы еще не создали маршруты к другим сегментам сети *books-net*. Если мы выполним прозвонку для узла из другой подсети, нас ожидает нечто интересное:

```
% ping 172.16.1.2
PING 172.16.1.2: 56 data bytes
ICMP Host redirect from gateway crab.wrotethebook.com (172.16.12.1)
 to horseshoe.wrotethebook.com (172.16.12.3) for ora.wrotethebook.com (172.16.1.2)
64 bytes from ora.wrotethebook.com (172.16.1.2): icmp_seq=1. time=30. ms
^C
----172.16.1.2 PING Statistics----
1 packets transmitted, 1 packets received, 0% packet loss round-trip (ms)  min/avg/
max = 30/30/30
```

---

[1] В Solaris для изучения таблиц маршрутизации всегда применяется netstat. В Linux можно использовать netstat или route, но обычно используется route.

[2] Возможно, что удаленный узел неисправен или не работает. В таком случае ping не получит ответа. Не отчаивайтесь, повторите попытку с другим узлом.

rodent считает, что все пункты назначения достижимы через маршрут по умолчанию. Следовательно, даже данные, предназначенные другим подсетям, передаются через шлюз *crab*. Если *rodent* передает узлу *crab* данные, которые должны пройти через *horseshoe*, *crab* отвечает узлу *rodent* сообщением ICMP Redirect, которое предписывает обращаться к узлу *horseshoe*. (Сообщение ICMP Redirect описано в главе 1.) Команда `ping` показывает сообщение ICMP Redirect в действии. Перенаправление непосредственно влияет на таблицу маршрутизации:

```
# route -n
Kernel IP routing table
Destination   Gateway       Genmask         Flags Metric Ref Use Iface
172.16.12.0   0.0.0.0       255.255.255.0   U     0      0     0 eth0
127.0.0.0     0.0.0.0       255.0.0.0       U     0      0     0 lo
0.0.0.0       172.16.12.1   0.0.0.0         UG    0      0     0 eth0
172.16.1.2    172.16.12.3   255.255.255.0   UGHD  0      0   514 eth0
```

Маршрут с установленным флагом D был создан сообщением ICMP Redirect.

Некоторые администраторы при проектировании сетей пользуются сообщениями ICMP Redirect. Все узлы настраиваются на работу с маршрутом по умолчанию, даже в сетях с многочисленными шлюзами. Шлюзы обмениваются информацией маршрутизации посредством соответствующих протоколов и перенаправляют узлы к лучшим шлюзам для каждого конкретного маршрута. Такой вид маршрутизации, зависимой от сообщений ICMP Redirect, получил широкое распространение благодаря персональным компьютерам. Многие персональные компьютеры неспособны работать с протоколами маршрутизации; отдельные ранние модели не имели команды `route` и были ограничены маршрутом по умолчанию. Сообщения ICMP Redirect оказались подходящим способом поддержки таких клиентов. Кроме того, этот вид маршрутизации прост в настройке и может эффективно претворяться в жизнь при помощи сервера настройки, поскольку каждый узел использует один-единственный маршрут (по умолчанию). По этим причинам некоторые руководители сетей поощряют многочисленные перенаправления ICMP.

Прочие сетевые администраторы предпочитают избегать перенаправлений ICMP и управлять содержимым таблицы маршрутизации напрямую. Чтобы обойтись без перенаправлений, мы можем создать конкретные маршруты для каждой конкретной подсети при помощи команды `route`:

```
# route add -net 172.16.1.0 netmask 255.255.255.0 gw 172.16.12.3
# route add -net 172.16.6.0 netmask 255.255.255.0 gw 172.16.12.3
# route add -net 172.16.3.0 netmask 255.255.255.0 gw 172.16.12.3
# route add -net 172.16.9.0 netmask 255.255.255.0 gw 172.16.12.3
```

Узел *rodent* имеет прямое подключение только к подсети 172.16.12.0, поэтому адреса всех шлюзов в таблице маршрутизации начинаются со значения 172.16.12. Вот так выглядит полученная таблица маршрутизации:

```
# route -n
Kernel IP routing table
```

```
Destination     Gateway         Genmask         Flags Metric Ref Use Iface
172.16.6.0      172.16.12.3     255.255.255.0   UG    0      0     0 eth0
172.16.3.0      172.16.12.3     255.255.255.0   UG    0      0     0 eth0
172.16.12.0     0.0.0.0         255.255.255.0   U     0      0     0 eth0
172.16.1.0      172.16.12.3     255.255.255.0   UG    0      0     0 eth0
172.16.9.0      172.16.12.3     255.255.255.0   UG    0      0     0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0     0 lo
0.0.0.0         172.16.12.1     0.0.0.0         UG    0      0     0 eth0
172.16.1.2      172.16.12.3     255.255.255.0   UGHD  0      0   514 eth0
```

Маршрут по умолчанию (через узел *crab*) позволяет работать с внешними сетями, а конкретные маршруты (через узел *horseshoe*) – достигать прочих подсетей в пределах *books-net*. Повторное выполнение тестов командой `ping` дает устойчивые положительные результаты. Однако при добавлении новых подсетей придется вручную добавить и соответствующие маршруты в таблицу маршрутизации. Кроме того, при перезагрузке системы все записи статической таблицы маршрутизации будут утрачены. Таким образом, применение статической маршрутизации требует создания маршрутов в процессе загрузки системы.

Создание статических маршрутов при загрузке

Приняв решение использовать статическую маршрутизацию, администратор должен внести два изменения в загрузочные файлы:

1. Добавить необходимые команды `route` в один из загрузочных сценариев.
2. Удалить из загрузочных файлов все команды, запускающие протоколы маршрутизации.

Чтобы добавить команды статической маршрутизации в загрузочный сценарий, следует, прежде всего, выбрать, в какой именно сценарий их поместить. В системах BSD и Linux под локальные дополнения процесса загрузки отведен файл *rc.local*. Он выполняется последним из загрузочных, а потому прекрасно подходит для внесения изменений в стандартный процесс загрузки. В системе Red Hat Linux, которую мы используем в примерах, полное имя файла *rc.local* – */etc/rc.d/rc.local*. В системе Solaris добавьте в файл */etc/init.d/inetinit* следующие команды:

```
route -n add default 172.16.12.1 > /dev/console
route -n add 172.16.1.0 172.16.12.3 > /dev/console
route -n add 172.16.6.0 172.16.12.3 > /dev/console
route -n add 172.16.3.0 172.16.12.3 > /dev/console
route -n add 172.16.9.0 172.16.12.3 > /dev/console
```

Ключ `-n` предписывает `route` отображать численные адреса в информационных сообщениях. Внося команды `route` в загрузочный файл системы Solaris, используйте ключ `-n`, чтобы запретить `route` тратить время, обращаясь к серверу имен, который в этот момент, вполне возможно, еще не функционирует. Ключ `-n` не требуется в случае системы Linux, поскольку Linux не сопровождает создание маршрутов информационными сообщениями.

Создав команды `route`, проверьте, не запускает ли сценарий модуль протокола маршрутизации, и при необходимости заблокируйте выполнение соответствующих строк сценария при помощи комментариев. Присутствие протокола маршрутизации ни к чему, если используется статическая маршрутизация. В системе Solaris из нашего примера программный модуль маршрутизации запускается только в случае, если в системе существует более одного сетевого интерфейса (то есть если она является маршрутизатором) либо если создан файл */etc/gateways*. (Об этом файле поговорим чуть позже.) Не выполнено ни одно из условий, значит, демон маршрутизации не будет запущен в процессе загрузки и от нас не требуются какие-либо дополнительные действия, помимо создания команд `route`.

Прежде чем вносить изменения в рабочую систему, сверьтесь с документацией. Возможно, следует изменить другой сценарий либо скорректировать полное имя демона маршрутизации. Только документация на конкретную систему содержит точные указания.

Несмотря на возможные отличия в именах загрузочных файлов, процедура в целом одинакова для всех систем. Описанные простые шаги – все, что нужно для настройки статической маршрутизации. Собственно говоря, проблема со статической маршрутизацией заключается не в ее настройке, но в сопровождении, если речь идет о меняющейся сетевой среде. Протоколы маршрутизации обладают достаточной гибкостью, чтобы справляться как с простыми, так и со сложными средами маршрутизации. Именно поэтому некоторые из последовательностей загрузки автоматически инициируют работу протоколов маршрутизации. Однако большинству систем Unix требуется лишь статический маршрут по умолчанию. Протоколы маршрутизации обычно нужны только маршрутизаторам.

Протоколы внутренней маршрутизации

Протоколы маршрутизации делятся на две базовых категории: протоколы *внутренней маршрутизации* и протоколы *внешней маршрутизации*. Протокол внутренней маршрутизации используется в рамках независимой сетевой системы. В терминологии TCP/IP такие независимые сетевые системы называются автономными системами.[1] В пределах автономной системы информация маршрутизации циркулирует на основе протокола маршрутизации, выбранного при администрировании этой автономной системы.

Все протоколы внутренней маршрутизации выполняют одни и те же основные функции: определяют «лучший» маршрут в каждый пункт назначения и распространяют информацию маршрутизации среди систем сети. То, как они выполняют эти функции (в частности, как определяют лучшие маршруты), – критерий, по которому различаются протоколы маршрутизации. Протоколов внутренней маршрутизации существует несколько:

[1] Автономные системы описаны в главе 2.

- *Протокол маршрутной информации* (RIP, *Routing Information Protocol*) – протокол внутренней маршрутизации, наиболее широко распространенный на платформах Unix. Реализации RIP поставляются в составе большинства систем Unix. Протокол адекватен в локальных сетях (LAN) и прост в настройке. RIP считает лучшим маршрут с минимальным числом транзитных участков (*метрикой маршрутизации*). Число транзитных участков в случае RIP – это число шлюзов, через которые должны пройти данные, прежде чем достигнут пункта назначения. RIP предполагает, что лучший маршрут проходит через минимальное число шлюзов. Такой подход к выбору маршрута носит название *алгоритма вектора расстояния* (*distance-vector algorithm*).
- *Hello* – протокол, в котором выбор лучшего маршрута выполняется на основе анализа задержек. *Задержка* – это время, за которое дейтаграмма проходит от источника к адресату и обратно. Пакет Hello содержит отметку времени отправки. Когда пакет доходит до адресата, получившая его система вычисляет время путешествия пакета. Hello используется достаточно редко. В свое время он использовался для внутренней маршрутизации исходной магистрали NSFNET (56 Кбит) и, пожалуй, больше практически нигде.
- *Протокол общения промежуточных систем* IS-IS (*Intermediate System to Intermediate System*) – протокол внутренней маршрутизации из набора протоколов OSI. Протокол IS-IS работает на основе *алгоритма состояния канала* и является *протоколом кратчайшего пути* (Shortest Path First, SPF). Данный протокол использовался для внутренней маршрутизации магистрали NSFNET T1 и сегодня все еще применяется некоторыми из крупных поставщиков услуг.
- *Протокол предпочтения кратчайшего пути* OSPF (*Open Shortest Path First*) – другой протокол состояния канала, разработанный для TCP/IP. Он подходит для применения в очень крупных сетях и имеет ряд преимуществ перед RIP.

Из перечисленных протоколов в подробностях мы рассмотрим RIP и OSPF. OSPF широко применяется на маршрутизаторах, а RIP – в системах Unix. Мы начнем с протокола RIP.

Протокол маршрутной информации (RIP)

В поставку многих систем Unix протокол Routing Information Protocol входит в качестве демона маршрутизации `routed` (произносится «рут-ди»). При запуске `routed` генерирует запрос на обновление маршрутов и ожидает получения ответов на этот запрос. Получив запрос, система, настроенная на распространение информации RIP, отвечает пакетом обновлений, созданным на основе информации из локальной таблицы маршрутизации. Пакет обновлений содержит конечные адреса из таблицы маршрутизации и связанные с ними метрики маршрутизации. Пакеты обновлений генерируются в ответ на запросы, а также в целях периодического уточнения информации маршрутизации.

Для создания таблицы маршрутизации routed использует информацию из пакетов обновлений. Если обновление содержит маршрут к пункту назначения, не существующий в локальной таблице маршрутизации, происходит добавление нового маршрута. Если обновление описывает маршрут, конечный пункт которого уже есть в локальной таблице, новый маршрут будет использован только в случае, если является лучшим из двух. Как уже говорилось, RIP считает лучшим маршрут с меньшим числом транзитных участков, а само это число в терминологии RIP называется *стоимостью* маршрута, или *метрикой маршрутизации*. Ранее мы видели, что метрика маршрутизации в локальной таблице поддается ручной корректировке посредством аргумента metric команды route. Чтобы выбрать лучший маршрут, протокол RIP должен сначала определить стоимость маршрута. Стоимость маршрута определяется суммой стоимости маршрута до шлюза, сгенерировавшего обновление, и метрики, содержащейся в пакете обновлений RIP. Если совокупная стоимость меньше стоимости уже существующего в таблице маршрута, используется новый маршрут.

Кроме того, RIP удаляет маршруты из таблицы маршрутизации. Во-первых, маршрут удаляется, если шлюз, через который пролегает маршрут, утверждает, что стоимость маршрута превышает 15. Во-вторых, RIP считает шлюз, не присылающий обновления, неработоспособным. Удаляются все маршруты, пролегающие через этот шлюз, если за определенный период времени не было получено ни одного обновления. Как правило, RIP генерирует обновления раз в 30 секунд. Во многих реализациях молчание шлюза в течение 180 секунд является поводом для удаления всех пролегающих через этот шлюз маршрутов из локальной таблицы маршрутизации.

RIP и routed

Чтобы запустить службу RIP при помощи демона маршрутизации (routed)[1], наберите такую команду:

```
# routed
```

Команда routed часто фигурирует без аргументов командной строки, но ключ -q может вам пригодиться. Ключ -q запрещает routed распространять маршруты и разрешает демону только принимать маршруты, распространяемые другими системами. Если машина не является шлюзом, имеет смысл использовать ключ -q.

В разделе, посвященном статической маршрутизации, мы не стали блокировать команду routed в файле *inetinit*, поскольку Solaris запускает routed только в случае, когда в системе установлено более одного сетевого интерфейса либо когда существует файл */etc/gateways*. Если система Unix запускает routed в любом случае, для запуска службы RIP не требуется дополнительных действий – достаточно просто загрузить систему. Иначе необходимо убедиться, что команда routed присутствует в одном из загрузочных файлов

[1] В некоторых системах демон маршрутизации имеет имя in.routed.

и что выполнены все условия ее запуска. Простейший способ запустить routed в системе Solaris – создать файл *gateways*, пусть даже пустой.

routed читает файл */etc/gateways* при запуске и добавляет информацию из файла в таблицу маршрутизации. routed может создать работоспособную таблицу маршрутизации на основе только обновлений RIP, полученных от прочих RIP-систем. Но иногда бывает полезно дополнить эту информацию, скажем, начальным маршрутом по умолчанию либо сведениями о шлюзе, который не распространяет данные о своих маршрутах. Такие дополнительные сведения хранятся в файле */etc/gateways*.

Чаще всего файл */etc/gateways* содержит определение активного маршрута по умолчанию, и это обстоятельство мы используем в наших примерах. Одного этого примера вполне достаточно, поскольку все записи файла */etc/gateways* имеют однородный формат. Следующая запись определяет систему *crab* в качестве шлюза по умолчанию:

```
net 0.0.0.0 gateway 172.16.12.1 metric 1 active
```

Запись начинается ключевым словом net. Все записи начинаются ключевым словом net либо ключевым словом host: первое предшествует адресу сети, второе – адресу узла. Конечный адрес 0.0.0.0 – это адрес маршрута по умолчанию. Применяя команду route, для обозначения этого маршрута мы использовали ключевое слово default, но в файле */etc/gateways* маршрут по умолчанию обозначается адресом сети 0.0.0.0.

Далее следует ключевое слово gateway и IP-адрес шлюза. В данном случае – адрес узла *crab* (172.16.12.1).

Затем следуют ключевое слово metric и численное значение метрики маршрутизации. Метрика определяет стоимость маршрута. В статической маршрутизации метрика почти не востребована, но в случае RIP метрики используются для принятия решений. Метрика RIP определяет число шлюзов, через которые должны пройти данные, чтобы попасть в пункт назначения. Но, как мы видели при изучении ifconfig, на деле метрика – это произвольное значение, используемое администратором для расстановки приоритетов маршрутов. (Системный администратор волен назначать маршруту любое значение метрики.) Однако имеет смысл варьировать метрику для нескольких маршрутов, ведущих к одному пункту назначения. У нас есть только один шлюз в сеть Интернет, так что верной метрикой для узла *crab* будет 1.

Все записи */etc/gateways* заканчиваются ключевым словом passive либо active. Первое означает, что от указанного шлюза локальная система не ожидает RIP-обновлений. Используйте ключевое слово passive, чтобы запретить RIP удалять маршруты в случае, когда шлюз не присылает пакеты обновлений и не должен их присылать. Пассивные маршруты добавляются в таблицу маршрутизации и существуют в течение всего времени работы системы. По сути дела, они становятся постоянными статическими маршрутами.

С другой стороны, ключевое слово active создает маршруты, обновляемые протоколом RIP. Ожидается, что активный шлюз предоставляет информа-

цию маршрутизации, которая удаляется из таблицы, если пакеты обновлений не поступают в течение заранее определенного интервала времени. Активные маршруты используются для «стимулирования» на этапе запуска RIP: предполагается, что они будут обновляться после перехода протокола в фазу активного существования.

Приведенная запись завершается ключевым словом `active`, то есть данный маршрут по умолчанию будет удален в случае отсутствия обновлений от узла *crab*. Умолчания маршрутов удобны, в особенности для статической маршрутизации. Однако при динамической маршрутизации их следует использовать осторожно, особенно если речь идет о нескольких шлюзах, предоставляющих маршруты одного направления. Пассивный маршрут по умолчанию не позволяет протоколу маршрутизации выполнять динамическое обновление и подстраиваться под изменения условий сетевой среды. Используйте активные маршруты по умолчанию, которые могут обновляться протоколом маршрутизации.

RIP легок в установке и настройке. Идеальный вариант? Не совсем так. RIP имеет три серьезных недостатка:

Ограниченный диаметр сети

Максимальная длина маршрута RIP – 15 транзитных участков. Маршрутизатор RIP не способен создать полную таблицу маршрутизации для сети, работающей с более длинными маршрутами. Число транзитных участков не может быть увеличено из-за следующего недостатка.

Медленная сходимость

Удаление неверного маршрута иногда требует многократного обмена пакетами обновлений, прежде чем стоимость маршрута достигнет значения 16. Это называется «счетом до бесконечности», поскольку RIP продолжает увеличивать стоимость маршрута, пока она не превысит максимально допустимого значения метрики в RIP. (В данном случае бесконечность представлена числом 16.) Кроме того, в RIP отсрочка удаления маршрута может достигать 180 секунд. Говоря на сетевом жаргоне, эти условия замедляют «сходимость маршрутизации», то есть требуется значительное время для того, чтобы таблица маршрутизации полностью отразила изменившееся состояние сети.

Классовая маршрутизация

RIP интерпретирует все адреса по правилам для классов, приведенным в главе 2. С точки зрения RIP все адреса принадлежат классам A, B и C, что делает протокол RIP несовместимым с современной практикой интерпретации адресов на основе битовых адресных масок.

Обойти ограничение диаметра сети невозможно. Небольшие значения метрик – насущная необходимость, позволяющая сократить воздействие счета до бесконечности. Однако ограниченный размер сети – наименьший из недостатков протокола RIP. Настоящая задача по улучшению RIP связана с решением двух других проблем – медленной сходимости и классовой маршрутизации.

В RIP были добавлены возможности, позволяющие бороться с медленной сходимостью. Прежде чем перейти к изучению этих возможностей, мы должны понять, как возникает проблема «счета до бесконечности». На рис. 7.2 отражена сеть, в которой может возникнуть такая проблема.

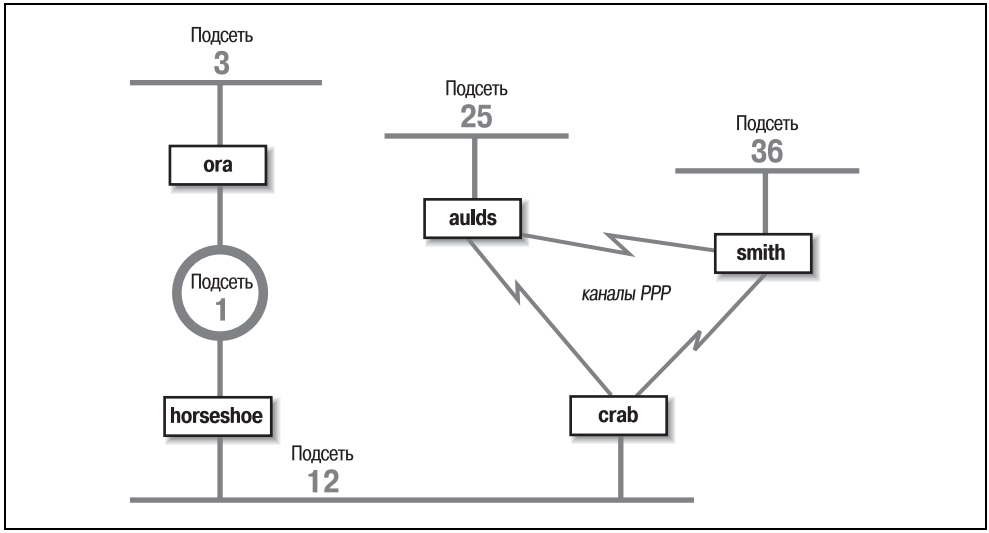


Рис. 7.2. *Пример сети*

Узел *crab* обращается к подсети 3 через *horseshoe* и далее через узел *ora*. Подсеть 3 удалена на два транзитных участка от узла *crab* и на один транзитный участок от узла *horseshoe*. Следовательно, *horseshoe* афиширует стоимость 1 для подсети 3, а *crab* – стоимость 2, и маршрутизация трафика через *horseshoe* продолжается. До тех пор пока не возникнут проблемы. Если неожиданно перестает работать узел *ora*, *horseshoe* ожидает обновлений от *ora* в течение 180 секунд. В процессе ожидания *horseshoe* продолжает посылать узлу *crab* обновления, и маршрут в подсеть 3 сохраняется в таблице маршрутизации *crab*. Когда интервал ожидания *horseshoe* наконец истекает, *horseshoe* удаляет все маршруты, пролегающие через *ora*, из своей таблицы маршрутизации, включая и маршрут в сеть 3. Затем *horseshoe* получает от узла *crab* обновление, уведомляющее, что *crab* находится в двух транзитных участках от подсети 3. *horseshoe* создает этот маршрут и объявляет, что находится в трех транзитных участках от подсети 3. *crab* получает это обновление, создает маршрут и объявляет, что находится в четырех транзитных участках от подсети 3. И так по кругу, пока стоимость маршрута в подсеть 3 не достигнет 16 в обеих таблицах маршрутизации. Если интервал обновления равен 30 секундам, процесс может затянуться надолго!

Механизмы *Split horizon* (Расщепленные горизонты) и *Poison reverse* (Отравленный обратный путь) – вот те две технологии, которые позволяют во многих случаях избежать счета до бесконечности:

Split horizon

Данный механизм не позволяет маршрутизатору афишировать маршруты через канал, по которому эти маршруты были получены, и решает описанную выше проблему счета до бесконечности. Следуя этому правилу, *crab* не станет уведомлять подсеть 12 о маршруте в подсеть 3, поскольку узнал этот маршрут из обновлений, полученных от узла *horseshoe*, расположенного в подсети 12. Механизм работает для приведенного выше примера, но не работает для всех случаев счета до бесконечности. Мы еще остановимся на этом вопросе чуть позже.

Poison reverse

Данный механизм является усовершенствованием механизма *Split horizon*. Идея та же: «Не афишировать маршруты через канал, по которому они получены». Однако к этому, по существу, негативному правилу добавляется позитивное действие. Маршрутизатору предписывается объявлять бесконечное расстояние для маршрутов такого канала. В результате узел *crab* должен сообщать, что стоимость пролегающих через него маршрутов в подсеть 3 равна 16. Стоимость 16 означает, что доступ к подсети 3 нельзя получить через шлюз *crab*.

Эти две технологии решают описанную выше проблему. Но что будет, если произойдет сбой в работе узла *crab*? Взгляните на рис. 7.2. Следуя правилу «split horizon», узлы *aulds* и *smith* не объявят маршрут в подсеть 12 шлюзу *crab*, поскольку сами узнали этот маршрут от узла *crab*. Однако они обмениваются маршрутом в подсеть 12 друг с другом. Если *crab* перестает работать, *aulds* и *smith* начинают свой счет до бесконечности, который заканчивается удалением маршрута в подсеть 12. Эту проблему призвана решить технология *triggered updates* (обновления по условию, или мгновенные обновления).

Triggered updates – большой шаг вперед, поскольку обновления посылаются немедленно, а не по истечении стандартного 30-секундного интервала. Таким образом, если происходит сбой маршрутизатора более высокого уровня или локального канала, маршрутизатор передает своим соседям обновления сразу после того, как внесет их в собственную таблицу маршрутизации. Без обновлений по условию счет до бесконечности может занять до восьми минут! Обновления по условию позволяют уведомить соседей за несколько секунд. Кроме того, данный механизм позволяет более эффективно использовать сетевые каналы. Обновления по условию не содержат полных таблиц маршрутизации – лишь сведения об изменившихся маршрутах.

Обновления по условию позволяют предпринимать четкие действия по уничтожению непроходимых маршрутов. Маршрутизатор объявляет маршруты, удаленные из таблицы маршрутизации, с бесконечной стоимостью, что вынуждает прочие маршрутизаторы также удалить эти маршруты. Взгляните еще раз на рис. 7.2. При сбое шлюза *crab* узлы *smith* и *aulds* выжидают 180 секунд, прежде чем удалить маршруты в подсети 1, 3 и 12 из своих таблиц маршрутизации. Затем они обмениваются обновлениями по условию, содержащими метрику 16 для подсетей 1, 3 и 12. Таким образом они сообщают друг другу, что не способны общаться с этими сетями, а необходимость в сче-

те до бесконечности исчезает. Технологии split horizons, poison reverse и triggered updates играют важную роль в уничтожении счета до бесконечности.

Последний недостаток – несовместимость RIP с сетями CIDR и подсетями переменной длины – привел к тому, что в 1996 году протокол RIP получил статус «исторического». RIP несовместим с существующим стеком протоколов TCP/IP, равно как с планами по его развитию. Для решения этой последней проблемы была разработана новая версия RIP.

RIP Version 2

Протокол RIP версии 2 (RIP-2), определенный в RFC 2453, является новой версией RIP. Протокол разрабатывался не с нуля – он лишь определяет расширения формата пакетов RIP. К адресу пункта назначения и метрике, существовавшим в пакетах RIP, RIP-2 добавляет маску сети и адрес следующего транзитного участка.

Маска сети снимает с маршрутизаторов RIP-2 ограничение, связанное с интерпретацией адресов по устаревшим правилам адресных классов. Теперь маска применяется к адресу пункта назначения, чтобы определить способ его интерпретации. Маска дает маршрутизаторам RIP-2 возможность работать с подсетями переменной длины и надсетями CIDR.

Адрес следующего транзитного участка – это IP-адрес шлюза, через который проходит маршрут. Если это адрес 0.0.0.0, источник пакета обновлений является шлюзом для маршрута. Транзитный адрес позволяет источнику данных RIP-2 распространять информацию маршрутизации о шлюзах, которые не говорят на языке протокола RIP-2. Функциональность транзитных адресов схожа с функциональностью сообщений ICMP Redirect, они указывают на лучшие шлюзы для маршрутов и сокращают число транзитных участков.

RIP-2 содержит и другие нововведения. Протокол передает обновления на групповой адрес 224.0.0.9, чтобы сократить нагрузку на системы, не способные обрабатывать пакеты RIP-2. Кроме того, RIP-2 предоставляет механизм проверки подлинности пакетов, позволяющий сократить возможность приема ошибочных обновлений от некорректно настроенных систем.

Несмотря на все изменения протокол RIP-2 совместим с RIP. Исходные спецификации RIP закладывали возможности такого развития протокола. В заголовке пакета RIP присутствует номер версии и несколько пустых полей для потенциальных расширений. Новые значения RIP-2 не потребовали переработки структуры пакета; они передаются в пустых полях, которые в исходном протоколе были зарезервированы для использования в будущем. Корректные реализации маршрутизаторов RIP способны принимать пакеты RIP-2 и извлекать из пакетов данные, не обращая внимания на новую информацию.

Split horizon, poison reverse, triggered updates, а также протокол RIP-2 решают большинство проблем изначального протокола RIP. Однако RIP-2 остается протоколом вектора расстояния. Существуют современные технологии маршрутизации, которые считаются более приемлемыми для крупных сетей. В частности – протоколы маршрутизации, выполняющие *анализ со-*

стояния каналов, поскольку они обеспечивают быструю сходимость и сокращают риск возникновения петель маршрутизации.

Протокол предпочтения кратчайшего пути

Протокол предпочтения кратчайшего пути OSPF (*Open Shortest Path First*), определенный документом RFC 2328, является протоколом состояния канала и в корне отличается от протокола RIP. Маршрутизатор, использующий RIP, делится информацией обо всей сети со своими соседями. Напротив, маршрутизатор, использующий OSPF, делится информацией о своих соседях со всей сетью. «Вся сеть» означает максимум одну автономную систему. RIP не пытается получить полные сведения о сети Интернет, а OSPF не пытается распространить информацию по всей сети Интернет. Задача этих протоколов в другом. Протоколы внутренней маршрутизации призваны решать вопросы маршрутизации в рамках отдельных автономных систем. OSPF подходит к задаче более скрупулезно, определяя иерархию областей маршрутизации автономной системы:

Области (Areas)

Область – это произвольный набор взаимосвязанных сетей, узлов и маршрутизаторов. Обмен информацией маршрутизации между областями одной автономной системы происходит посредством *пограничных маршрутизаторов областей*.

Магистраль (Backbone)

Магистраль – это особая область, объединяющая все прочие области автономной системы. Каждая область должна быть связана с магистралью, поскольку магистраль отвечает за распространение информации маршрутизации между областями.

Оконечная область (Stub area)

Оконечная область имеет лишь один пограничный маршрутизатор, то есть из области существует единственный маршрут. В данном случае пограничный маршрутизатор области может не сообщать о внешних маршрутах прочим маршрутизаторам оконечной области. Достаточно заявить о себе как о точке, через которую пролегает маршрут по умолчанию.

Деление на области необходимо лишь для крупных автономных систем. Сеть, показанная на рис. 7.2, невелика, нет смысла делить ее на области. Тем не менее она может послужить иллюстрацией различных областей. Мы можем разделить данную автономную систему на любые области – как нам заблагорассудится. Предположим, мы разделили ее на три области: область 1 содержит подсеть 3; область 2 содержит подсети 1 и 12; область 3 содержит подсети 25 и 36, а также каналы PPP. Далее, мы можем определить область 1 в качестве оконечной, поскольку она имеет лишь один пограничный маршрутизатор – *ora*. Кроме того, мы можем определить область 2 в качестве магистральной, поскольку она объединяет две оставшиеся области и передает всю информацию маршрутизации между областями 1 и 3. Область 2 содержит два пограничных маршрутизатора, *crab* и *ora*, плюс один внутренний

маршрутизатор – *horseshoe*. Область 3 содержит три маршрутизатора: *crab*, *smith* и *aulds*.

Очевидно, OSPF обеспечивает высокую гибкость в плане разграничения автономной системы. Для чего же нужна такая гибкость? Одной из проблем протоколов, анализирующих состояние каналов, является большой объем данных, накапливаемых в *базе данных состояний каналов*, и объем времени, необходимый для вычисления маршрутов на основе этих данных. Сейчас станет ясно, почему возникает такая проблема.

Каждый OSPF-маршрутизатор выполняет построение *ориентированного графа* всей сети при помощи алгоритма Дейкстры, служащего для обнаружения кратчайшего пути (Shortest Path First, SPF). Ориентированный граф – это карта сети с точки зрения маршрутизатора. То есть корнем графа является маршрутизатор. Построение графа выполняется на основе данных из базы данных состояния каналов, содержащей информацию о каждом маршрутизаторе сети и обо всех соседях каждого маршрутизатора. Эта база данных для автономной системы, представленной на рис. 7.2, содержит пять маршрутизаторов и десять соседей: у *ora* один сосед, *horseshoe*; у *horseshoe* два соседа, *ora* и *crab*; у *crab* три соседа – *horseshoe*, *aulds* и *smith*; у *aulds* – два соседа, *crab* и *smith*; у *smith* – два соседа, *aulds* и *crab*. Граф этой автономной системы в представлении маршрутизатора *ora* отражен на рис. 7.3.

Алгоритм Дейкстры создает карту следующим образом:

1. Локальная система устанавливается в качестве корня карты и получает нулевую стоимость.

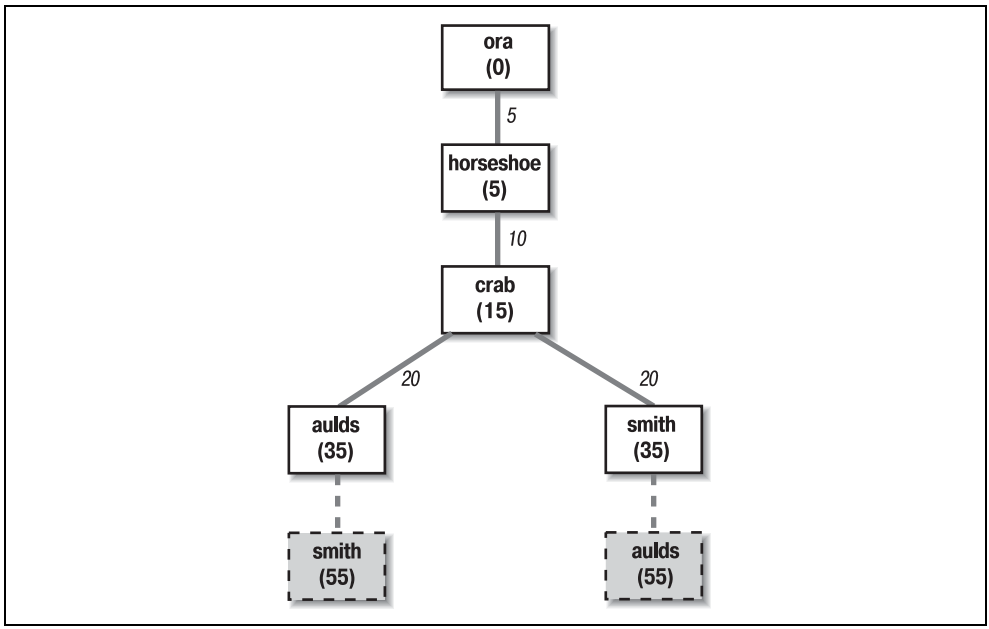


Рис. 7.3. *Граф сети*

2. В карту добавляются соседи только что установленной системы. Стоимость сообщения с соседями представлена суммой стоимости сообщения с только что установленной системой и стоимостью, которую эта система афиширует для каждого из соседей. Например, предположим, что *crab* афиширует стоимость 20 для *aulds*, а стоимость сообщения с *crab* – 15. Тогда стоимость *aulds* на карте *ora* – 35.
3. Выполняется обход карты с выбором самых дешевых маршрутов для всех направлений. Например, когда в карту добавляется *aulds*, среди его соседей находится *smith*. Путь к узлу *smith* через *aulds* временно добавляется в карту. На третьем шаге алгоритма стоимость сообщения с узлом *smith* через *crab* сравнивается со стоимостью сообщения с узлом *smith* через *aulds*. Выбор делается в пользу более дешевого пути. На рис. 7.3 отброшенные маршруты представлены пунктирными линиями. Шаги 2 и 3 алгоритма повторяются для каждой системы из базы данных.

Информация базы данных состояния каналов накапливается и распространяется по простым и эффективным правилам. OSPF-маршрутизатор выполняет обнаружение своих соседей при помощи пакетов Hello.[1] Он отправляет пакеты Hello и ожидает получения пакетов Hello от соседствующих маршрутизаторов. Пакет Hello идентифицирует локальный маршрутизатор и перечисляет соседние маршрутизаторы, от которых были получены пакеты. Получив пакет Hello с информацией о себе в качестве соседствующего, маршрутизатор понимает, что обнаружил соседа. И это вполне логично – ведь он может получать пакеты от этого соседа, а сосед считает его своим соседом – и, значит, может получать ответные пакеты. Обнаруженные соседи добавляются в соответствующий локальный список системы.

Затем OSPF-маршрутизатор передает сведения обо всех своих соседях, а именно выполняет веерную *рассылку* (*flooding*) по сети пакетов LSA (Link-State Advertisement). Пакет LSA содержит адрес каждого соседа и стоимость сообщения с этим соседом для локальной системы. Веерная рассылка означает, что маршрутизатор передает пакет LSA с каждого интерфейса и что каждый маршрутизатор, получивший пакет, передает его с каждого интерфейса – за исключением того, через который пакет был изначально получен. Чтобы предотвратить распространение дубликатов пакетов LSA, маршрутизаторы хранят экземпляры полученных пакетов и удаляют дубликаты.

Обратимся к рис. 7.2 за очередным примером. Когда протокол OSPF запускается на *horseshoe*, то посылает пакет Hello в подсеть 1 и еще один – в подсеть 12. *ora* и *crab* получают приветствие и отвечают пакетами Hello, в которых *horseshoe* указан соседствующим маршрутизатором. *horseshoe*, получив эти пакеты Hello, добавляет *ora* и *crab* в список своих соседей. Затем *horseshoe* создает пакет LSA, в котором каждому из соседей (*ora* и *crab*) поставлена в соответствие стоимость. Например, *horseshoe* может присвоить *ora* стоимость 5, а *crab* – стоимость 10. *horseshoe* рассылает пакеты LSA в подсетях 1 и 12. *ora* получает LSA и рассылает его в подсети 3. *crab* получает LSA и рас-

[1] Не путайте пакеты Hello с протоколом Hello. Здесь речь идет о пакетах OSPF Hello.

сылает его через оба своих канала PPP. *aulds* рассылает LSA по каналу к *smith*, а *smith* – по тому же каналу *aulds*. *aulds* и *smith*, получив вторую копию LSA, удаляют ее, поскольку она дублирует уже полученную от узла *crab*. Таким образом, каждый маршрутизатор всей сети получает LSA-пакеты всех других маршрутизаторов.

OSPF-маршрутизаторы отслеживают состояние своих соседей, принимая пакеты Hello. Пакеты Hello генерируются всеми маршрутизаторами периодически. Если маршрутизатор перестал генерировать пакеты, он, или связанный с ним канал, переходит в разряд неработоспособных. Соседи этого маршрутизатора обновляют свои записи LSA и посылают их в сеть. Новые LSA включаются в базу данных состояний каналов на каждом маршрутизаторе сети, и каждый маршрутизатор заново выполняет построение карты сети, исходя из новых сведений. Очевидно, ограничение размера сети и, как следствие, количества маршрутизаторов снижает нагрузки, связанные с построением карт. Для одних сетей вся автономная система оказывается достаточно невелика. Другим требуется разделение автономной системы на области.

Еще одной чертой OSPF, благоприятно влияющей на производительность, является возможность определить *назначенный маршрутизатор*. Назначенный маршрутизатор – это один из маршрутизаторов сети, который считает соседями все остальные маршрутизаторы, тогда как все остальные маршрутизаторы сети считают соседом только его. Назначенный маршрутизатор позволяет сократить размер базы данных состояний каналов и повышает скорость работы алгоритма вычисления кратчайшего пути. Рассмотрим для примера широковещательную сеть с пятью маршрутизаторами. Пять маршрутизаторов – по четыре соседа на каждого – являются источником базы данных с двадцатью записями. Но если один из маршрутизаторов является назначенным, тогда у него четыре соседа, а у каждого из его соседей – всего по одному. В общей сложности получается десять записей базы данных. И хотя в столь маленькой сети назначенный маршрутизатор не нужен, чем крупнее сеть, тем выше экономия. Например, широковещательная сеть с 25 маршрутизаторами, один из которых является назначенным, имеет базу данных состояний каналов из пятидесяти записей, тогда как в отсутствие обозначенного маршрутизатора размер базы данных – шестьсот записей.

OSPF дает маршрутизатору полную картину маршрута из конца в конец – сравните с ограниченным видом следующего транзитного участка в RIP. Рассылка LSA позволяет быстро распространить информацию по сети. Ограничение размера базы данных при помощи разделения на области и отметки маршрутизаторов ускоряет вычисление кратчайших путей. В целом, OSPF оказывается весьма эффективным протоколом маршрутизации.

OSPF предоставляет также функции, которых нет в RIP. Простая аутентификация по паролю, состоящему из восьми символов и передаваемому открытым текстом, позволяет убедиться, что обновление исходит от доверенного маршрутизатора. Кроме того, реализован более надежный механизм аутентификации на основе контрольных сумм MD5 (Message Digest 5).

OSPF поддерживает *многолучевую маршрутизацию для лучей равной стоимости* (*equal-cost multi-path routing*). Эта неудобоваримая фраза означает, что маршрутизаторы OSPF способны работать более чем с одним маршрутом определенного направления. При соблюдении определенных условий такая возможность может использоваться для распределения нагрузки по ряду сетевых каналов. Однако многие системы не смогут воспользоваться этим вариантом из-за своих собственных недостатков. Чтобы определить, поддерживает ли ваш маршрутизатор распределение нагрузки посредством равноценных маршрутов OSPF, обратитесь к соответствующей документации.

С учетом описанных возможностей, OSPF является предпочтительным протоколом внутренней маршрутизации TCP/IP для выделенных маршрутизаторов.

Протоколы внешней маршрутизации

Протоколы внешней маршрутизации реализуют обмен информацией маршрутизации между автономными системами. Такая информация маршрутизации известна как *информация достижимости*. Информация достижимости – это сведения о том, какие сети доступны через конкретную автономную систему.

RFC 1771 дает определение протокола пограничных шлюзов BGP (Border Gateway Protocol), ведущего протокола внешней маршрутизации, и содержит следующее описание функции маршрутизации автономной системы:

> Классическое определение автономной системы (АС): набор маршрутизаторов, подчиненных единому техническому управлению, использующих протокол внутренних шлюзов и общие метрики для маршрутизации пакетов в пределах АС, а также протокол внешних шлюзов для маршрутизации пакетов в другие АС... Одна АС видится другой как имеющая четкое внутреннее планирование маршрутизации и представляющая ясную картину достижимых через нее сетей. С точки зрения внешней маршрутизации АС можно считать монолитной конструкций...

Обмен информацией с такими монолитными конструкциями как раз и является функцией протоколов внешней маршрутизации. Протоколы внешней маршрутизации называют также протоколами внешних шлюзов. Не надо путать понятие протокола внешних шлюзов с Протоколом Внешних Шлюзов EGP (Exterior Gateway Protocol). EGP – не общее название; это имя собственное конкретного протокола внешней маршрутизации, причем довольно старого.

Протокол внешних шлюзов (EGP)

Шлюз, работающий с протоколом EGP, сообщает, что способен передавать информацию в сети, которые входят в его автономную систему. Шлюз не говорит, что способен обращаться к сетям за пределами своей автономной системы. Например, внешний шлюз гипотетической автономной системы *book-as*

способен обращаться ко всей сети Интернет через свой внешний канал, но в его автономной системе существует только одна сеть (172.16.0.0), доступ к которой он и будет афишировать, работая с EGP.

Прежде чем передавать информацию маршрутизации, системы обмениваются EGP-сообщениями *Hello* и *I-Heard-You* (I-H-U). Эти сообщения начинают диалог между парами шлюзов EGP. Компьютеры, выполняющие обмен данными по EGP, называются *EGP-соседями*, а обмен сообщениями Hello и I-H-U – *знакомством с соседом*.

Когда соседи познакомились, информация маршрутизации доступна в результате *опроса* (*poll*). Сосед отвечает пакетом информации достижимости, или *обновлением* (*update*). Локальная система включает маршруты из обновления в локальную таблицу маршрутизации. Если сосед не ответил на три запроса подряд, система делает вывод, что сосед недоступен, и удаляет поступившие от него маршруты из таблицы маршрутизации. Получив запрос от EGP-соседа, система отвечает собственным пакетом обновлений.

В отличие от протоколов внутренней маршрутизации, описанных выше, EGP не пытается выбрать «лучший» маршрут. Обновления EGP содержат данные векторных расстояний, но EGP не производит вычислений на основе этой информации. Метрики маршрутизации из различных автономных систем не поддаются прямому сравнению, поскольку в различных АС могут использоваться различные критерии вывода таких значений. Таким образом, EGP оставляет выбор «лучшего» маршрута кому-то другому.

Когда проектировался протокол EGP, функционирование сети зависело от группы доверенных стержневых шлюзов, которые обрабатывали и распространяли маршруты, полученные от всех автономных систем. Предполагалось, что стержневые шлюзы обладают всей необходимой для выбора лучших внешних маршрутов информацией. Информация достижимости EGP передавалась стержневым шлюзам, обрабатывалась и возвращалась автономным системам.

Структура маршрутизации, находящаяся в зависимости от одной группы шлюзов с централизованным управлением, не очень хорошо масштабируется, а потому является неадекватным решением, учитывая темпы роста сети Интернет. По мере роста числа автономных систем и сетей, подключенных к Интернету, центральным шлюзам становилось все труднее справляться с рабочей нагрузкой. В том числе это обстоятельство послужило причиной перехода сети Интернет на более эффективную распределенную архитектуру, возлагающую нагрузку обработки маршрутов на отдельные автономные системы. Вторая причина – отсутствие выделенного управляющего органа коммерциализированной сети Интернет. Интернет состоит из многих равноправных сетей. В распределенной архитектуре автономной системе требуются протоколы маршрутизации, как внутренней, так и внешней, позволяющие принимать осмысленные решения в выборе маршрутов. По этой причине протокол EGP вышел из моды.

Протокол пограничных шлюзов (BGP)

Протокол пограничных шлюзов BGP (*Border Gateway Protocol*) является ведущим протоколом внешней маршрутизации сети Интернет. Протокол BGP основан на *OSI-протоколе междоменной маршрутизации* (*InterDomain Routing Protocol*, IDRP). BGP поддерживает маршрутизацию, подчиненную правилам, позволяющим принимать решения по маршрутизации, исходя из нетехнических причин (политических, структурных, из соображений безопасности и т. д.). Таким образом, BGP совершенствует способность автономной системы выбирать маршруты и приводить в исполнение правила, не обращаясь к высшему авторитету. В отсутствие центральных шлюзов, выполняющих эти задачи, наличие подобных механизмов чрезвычайно важно.

Правила маршрутизации не являются частью протокола BGP. Правила имеют внешние источники и выступают в роли данных настройки. Как уже говорилось в главе 2, в точках доступа к сети (NAPs, Network Access Points), объединяющих крупных поставщиков услуг Интернета, существуют арбитры маршрутизации (RAs, Routing Arbiters). Правила маршрутизации могут быть получены от арбитров. Кроме того, большинство поставщиков услуг Интернета создают частные наборы правил – двусторонние соглашения с другими поставщиками услуг. BGP может применяться для реализации таких соглашений: управления афишируемыми маршрутами и принимаемыми маршрутами. Позже в этой главе – в разделе, посвященном `gated`, – мы обсудим команды `import` и `export`, которые определяют принимаемые (`import`) и афишируемые (`export`) маршруты. Администратор сети приводит правила маршрутизации в исполнение путем соответствующей настройки маршрутизатора.

BGP работает поверх TCP, что обеспечивает его надежной службой доставки. BGP использует широко известный порт TCP 179 и знакомится с соседями посредством стандартного тройного рукопожатия TCP. Соседи BGP известны в качестве *равных* (*peers*). Установив соединение, BGP-равные обмениваются сообщениями OPEN в целях согласования параметров сеанса, таких как версия протокола BGP.

Сообщение UPDATE содержит перечень пунктов назначения, доступных через определенное направление, а также свойства этого направления. BGP является *протоколом векторного пути* и называется так потому, что предоставляет информацию о сквозном пути в виде последовательности номеров автономных систем. Наличие полного пути AC исключает появление петель маршрутизации и проблем счета до бесконечности. Сообщение BGP UPDATE содержит один вектор пути и перечень всех пунктов назначения, доступных через этот вектор. Для создания таблицы маршрутизации могут передаваться множественные пакеты UPDATE.

BGP-равные обмениваются полными обновлениями таблиц маршрутизации в первом сеансе связи. После этого передаются только изменения. Если изменений нет, передается небольшое (19 байт) сообщение KEEPALIVE, уведомляющее, что BGP-система и канал по-прежнему функционируют. BGP весьма экономно расходует ресурсы систем и полосы пропускания сетевых каналов.

Самое важное, что следует помнить о протоколах внешней маршрутизации, – большинство систем прекрасно обходятся без них. Протоколы внешней маршрутизации требуются лишь для переноса информации между автономными системами. Большинство маршрутизаторов в пределах автономной системы работают на основе протокола внутренней маршрутизации, такого как OSPF. И только шлюзам, связующим различные автономные системы, необходимы протоколы внешней маршрутизации. Ваша сеть, скорее всего, входит в качестве независимой составляющей в автономную систему, которая управляется кем-то иным. Хорошим примером автономных систем, состоящих из многочисленных независимых сетей, являются поставщики услуг Интернета. И если ваша организация не предоставляет услуги подобного уровня, то и протокол внешней маршрутизации, вероятно, не понадобится.

Выбор протокола маршрутизации

Несмотря на разнообразие вариантов выбрать протокол маршрутизации обычно легко. Мотивацией разработки большинства описанных протоколов внутренней маршрутизации служила необходимость разрешения конкретных проблем маршрутизации в крупных сетях. Некоторые из протоколов использовались лишь в крупных национальных и региональных сетях. Для территориальных сетей обычным выбором по-прежнему является RIP, тогда как в более крупных сетях выбор делается в пользу OSPF.

В случае необходимости работать с протоколом внешней маршрутизации его тип зачастую не приходится выбирать. Чтобы две автономные системы могли обмениваться информацией маршрутизации, они должны работать с одним протоколом. Если вторая автономная система уже функционирует, ее администраторы, скорее всего, уже решили, какой протокол использовать, и вам останется лишь согласиться с этим выбором. Чаще всего выбор падает на BGP.

На выбор протоколов влияет и тип оборудования. Маршрутизаторы работают с широким спектром протоколов, хотя отдельные производители могут делать акцент на каких-то конкретных. Простые узлы обычно обходятся без протоколов маршрутизации, а в состав большинства систем Unix входит только RIP. Таким образом, допуск простых узлов на поле динамической маршрутизации может серьезно ограничить варианты выбора. Но `gated` позволяет работать в Unix-системе со многими протоколами маршрутизации. И хотя производительность специального аппаратного обеспечения маршрутизаторов обычно выше, `gated` позволяет использовать в качестве маршрутизатора любую систему Unix.

В последующих разделах мы изучим программный пакет `gated` (Gateway Routing Daemon, демон шлюзовой маршрутизации), сочетающий протоколы как внутренней, так и внешней маршрутизации, и рассмотрим примеры работы с RIP, RIPv2, OSPF и BGP.

Демон шлюзовой маршрутизации

Разработка ПО маршрутизации для систем Unix общего назначения выглядит весьма скромно. На большинстве площадок системы Unix используются для решения лишь самых простых задач маршрутизации, и в этих случаях обычно адекватен протокол RIP. В случаях сложной маршрутизации, требующей применения более совершенных протоколов, используются специализированные аппаратные маршрутизаторы, предназначенные для решения именно этой задачи. Многие из более совершенных протоколов маршрутизации доступны системам Unix только в пакете `gated`. `gated` сочетает несколько различных протоколов маршрутизации в одном программном пакете.

Кроме того, `gated` предоставляет и другие возможности, обычно присущие только специализированным маршрутизаторам:

- Одна система может работать с целым набором протоколов маршрутизации. `gated` объединяет информацию маршрутизации, полученную по различным протоколам, и выбирает «лучшие» маршруты.
- Маршруты, полученные посредством протоколов внутренней маршрутизации, могут распространяться через протокол внешней маршрутизации, что делает возможным динамическое изменение информации достижимости, распространяемой внешним образом, отражающее результаты изменения внутренних маршрутов.
- Правила маршрутизации позволяют создавать фильтры для приема и распространения маршрутов.
- Все протоколы настраиваются при помощи единственного файла (*/etc/gated.conf*). Для настройки применяется единый синтаксис команд.
- Пакет `gated` постоянно дорабатывается. Применение `gated` гарантирует, что ваш пакет маршрутизации шагает в ногу со временем.

Значение предпочтения в gated

Каждая реализация протокола маршрутизации имеет две стороны. Первая сторона – внешняя – реализует обмен информацией маршрутизации с удаленными системами. Вторая сторона – внутренняя – обновляет таблицу маршрутизации, используя сведения, полученные от удаленных систем. Например, когда протокол OSPF обменивается пакетами Hello в поисках соседей, то выполняет внешнюю функцию протокола. Когда протокол OSPF добавляет маршрут в таблицу маршрутизации, то выполняет внутреннюю функцию.

Внешние функции протоколов, реализованные в `gated`, соответствуют внешним функциям в других реализациях протоколов. Однако внутренняя сторона `gated` присуща только системам Unix. `gated` обрабатывает информацию маршрутизации от различных протоколов, каждый из которых имеет собственную метрику для определения лучшего маршрута, и объединяет эту информацию перед тем, как внести обновления в таблицу маршрутизации. До появления `gated` сосуществующие в системе Unix протоколы обновляли таб-

лицу маршрутизации совершенно независимо друг от друга. В итоге в таблице оказывался маршрут, внесенный последним, – не обязательно лучший.

В случае нескольких протоколов маршрутизации и нескольких сетевых интерфейсов система может получать маршруты одного направления по различным протоколам. `gated` сравнивает такие маршруты и пытается выбрать лучший. Однако метрики различных протоколов не подлежат прямому сравнению. Каждый протокол использует собственную метрику. Это может быть счетчик транзитных участков, задержка для маршрута либо произвольное значение, указанное администратором. Для выбора лучшего маршрута `gated` требуется больше, чем метрики протоколов. Демон использует собственную оценку, позволяющую выбирать между маршрутами, полученными от нескольких протоколов или интерфейсов. Данное значение известно как *значение предпочтения*.

Значения предпочтения позволяют `gated` объединить сведения, полученные из нескольких источников, в одной таблице маршрутизации. В табл. 7.1 перечислены источники маршрутов `gated` и значения предпочтения, присвоенные каждому из них по умолчанию. Диапазон значений предпочтения – от 0 до 255, причем наименьшее значение обозначает наиболее предпочтительный маршрут. По этой таблице можно понять, что `gated` предпочитает маршруты, полученные по OSPF, тем же маршрутам, полученным от BGP.

Таблица 7.1. Значения предпочтения по умолчанию

Тип маршрута	Предпочтение по умолчанию
Прямой маршрут	0
OSPF	10
IS-IS Level 1	15
IS-IS Level 2	18
Внутренний маршрут по умолчанию	20
Перенаправление ICMP	30
Маршрут, полученный из сокета маршрутов	40
Статический маршрут	60
Маршруты SLSP	70
RIP	100
Маршруты интерфейса точка-точка	110
Маршруты, пролегающие через неработающий интерфейс	120
Объединенные и порожденные маршруты	130
Маршруты OSPF ASE	150
BGP	170
EGP	200

Значение предпочтения может фигурировать в целом ряде операторов настройки. Оно может использоваться для расстановки приоритетов маршрутов, полученных через различные сетевые интерфейсы, от различных протоколов маршрутизации либо от различных внешних шлюзов. Значения предпочтения не передаются и не изменяются протоколами, они встречаются только в файле настройки. В следующем разделе мы изучим файл настройки `gated` (*/etc/gated.conf*) и хранимые в нем команды настройки.

Настройка gated

Получить `gated` можно по адресу *http://www.gated.org*. В приложении В содержится информация о том, как скопировать и собрать пакет. В данном разделе мы будем работать с `gated` release 3.6, версией `gated`, доступной в настоящее время без каких-либо ограничений. Существуют и другие версии `gated`, доступные членам консорциума Gated. Если вы планируете создавать продукты, основанные на `gated`, либо проводить исследования протоколов при помощи `gated`, вам следует вступить в консорциум. Для целей этой книги отлично подойдет и версия 3.6.

`gated` читает свои настройки из файла */etc/gated.conf*. Команды настройки в этом файле напоминают код на языке C. Все операторы заканчиваются точкой с запятой, а связанные операторы группируются фигурными скобками. Такая структура позволяет быстрее разобраться во взаимосвязях фрагментов файла настройки, что очень важно, если выполняется настройка нескольких протоколов в одном файле. Помимо структуры языка, структура присуща и файлу */etc/gated.conf* в целом.

Различные операторы настройки, а также порядок, в котором они должны следовать, делят файл *gated.conf* на разделы: *операторы параметров*, *операторы интерфейсов*, *операторы определений*, *операторы индивидуальных* и *групповых протоколов*, *статические операторы*, *управляющие операторы* и *объединяющие операторы*. Нарушение порядка следования операторов приводит к возникновению ошибки в процессе разбора файла.

Есть еще два типа операторов, которые не входят в перечисленные категории. Это операторы инструкций и операторы трассировки. Такие операторы могут встречаться в любой точке файла *gated.conf* и не связаны напрямую с настройкой какого-либо протокола. Операторы инструкций предписывают определенное поведение модулю разбора файла настройки, а операторы трассировки позволяют управлять трассировкой из файла настройки.

В табл. 7.2 содержится сводка команд настройки `gated`. Для каждой команды указано имя, тип оператора и приведено краткое резюме ее функциональности. Полностью язык команд описан в приложении В.

Как видите, команд в языке настройки `gated` много. Язык содержит рычаги управления для ряда различных протоколов, плюс дополнительные команды настройки функциональности собственно демона `gated`. Читатели могут испытать легкое замешательство.

Таблица 7.2. Операторы настройки gated

Оператор	Тип	Назначение
%directory	инструкция	Указывает каталог, в котором расположены включаемые файлы
%include	инструкция	Включает содержимое указанного файла в *gated.conf*
Traceoptions	трассировка	Указывает, какие события необходимо отслеживать
Options	параметр	Задает параметры работы `gated`
Interfaces	интерфейс	Задает параметры интерфейсов
Autonomoussystem	определение	Задает номер АС
Routerid	определение	Задает исходный маршрутизатор для BGP или OSPF
Martians	определение	Задает недопустимые адреса пунктов назначения
Multicast	протокол	Задает параметры протокола групповой передачи
Snmp	протокол	Включает отчетность по SNMP
Rip	протокол	Включает RIP
Isis	протокол	Включает протокол IS-IS
kernel	протокол	Задает параметры взаимодействия с ядром системы
ospf	протокол	Включает протокол OSPF
redirect	протокол	Удаляет маршруты, созданные сообщениями ICMP
egp	протокол	Включает EGP
bgp	протокол	Включает BGP
icmp	протокол	Выполняет настройку обработки пакетов ICMP в целом
pim	протокол	Включает протокол групповой передачи PIM
dvmrp	протокол	Включает протокол групповой передачи DVMRP
msdp	протокол	Включает протокол групповой передачи MSDP
static	статический	Определяет статические маршруты
import	управление	Указывает, какие маршруты приемлемы
export	управление	Указывает, какие маршруты афишируются
aggregate	объединение	Управляет объединением маршрутов
generate	объединение	Управляет созданием маршрута по умолчанию

Во избежание этого не пытайтесь понять до конца и в деталях все возможности gated. В вашей среде маршрутизации едва ли будет востребована вся функциональность пакета. Даже при создании шлюза на границе двух автономных систем вам, скорее всего, придется использовать лишь два протокола маршрутизации: один для внутренней маршрутизации, а второй – для внешней. Файл настройки должен содержать только команды, которые непосредственно относятся к востребованным протоколам. В процессе чтения этого раздела пропускайте информацию, которая не понадобится. К примеру, если не будет использоваться протокол BGP, зачем изучать оператор bgp? В то же время, более подробная информация по операторам содержится в приложении B. Учитывая сказанное, обратимся к примерам настройки.

Примеры настройки, gated.conf

Подробности, приводимые в приложении B, могут создать впечатление, будто настройка gated сложнее, чем на самом деле. Богатый язык команд gated может ввести в заблуждение, поскольку предназначен для работы со многими протоколами, а кроме того, позволяет достичь одних и тех же результатов различными способами. Но, как мы увидим из реалистичных примеров, настройка в каждом конкретном случае совсем необязательно оказывается сложной.

Основой для примеров станет рис. 7.4. Мы установили новый маршрутизатор, который обеспечивает нашей магистрали прямой доступ к сети Интернет, и приняли решение установить новые протоколы маршрутизации. Один из узлов будет принимать обновления RIP-2, внутренний шлюз – работать с протоколами RIP-2 и OSPF, а внешний шлюз – с протоколами OSPF и BGP.

Шлюз *limulus* связывает подсети 172.16.9.0 и 172.16.1.0. Узлам подсети 9 он представляется в качестве шлюза по умолчанию, поскольку является шлюзом во внешний мир. Для передачи маршрутов в подсеть 9 *limulus* применяет RIP-2. Подсети 1 шлюз *limulus* представляется в качестве шлюза в подсеть 9 – посредством OSPF.

Шлюз *chill* обеспечивает подсеть 1 доступом к сети Интернет через автономную систему 164. Поскольку шлюз *chill* выходит в сеть Интернет, он представляется в качестве шлюза по умолчанию всем остальным системам подсети 1 – посредством OSPF. Внешней автономной системе, посредством BGP, он представляется путем ко внутренним сетям, о которых узнает при помощи OSPF.

Итак, рассмотрим настройку маршрутизации для узла *minasi*, шлюза *limulus* и шлюза *chill*.

Настройка узла

Настройка маршрутизации узла очень проста. Оператор rip yes включает RIP, а для работы RIP, вообще говоря, больше ничего не требуется. В присутствии этого оператора протокол RIP должен работать в любой системе. Дополнительные операторы, заключенные в фигурные скобки, изменяют базо-

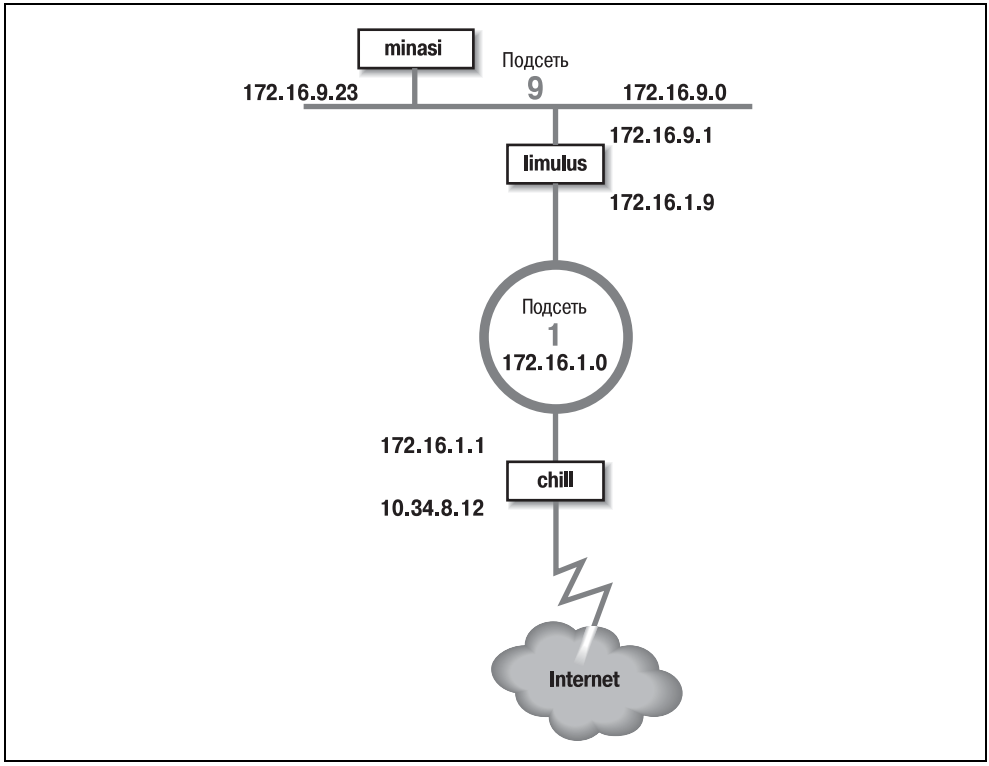


***Рис. 7.4.** Пример топологии маршрутизации*

вые настройки RIP. Мы используем ряд операторов, чтобы сделать приведенный ниже пример настройки RIP-2 для узла *minasi* чуть более интересным:

```
#
#  включить rip, не производить широковещательную
#  передачу обновлений,
#  принимать обновления RIP-2 на групповой адрес,
#  проверять подлинность обновлений.
#
rip yes {
        nobroadcast ;
        interface 172.16.9.23
             version 2
             multicast
             authentication simple "REAL stuff" ;
 } ;
```

Этот пример отражает общее строение операторов настройки, хранимых в файле *gated.conf*. Символ решетки (#) начинает строки комментариев.[1] Каж-

[1] Комментарии могут также заключаться между парами символов * и *\.

дый оператор заканчивается точкой с запятой. Связанные с оператором настройки уточнения могут быть многострочными и заключаются в фигурные скобки ({}). В этом примере уточнения nobroadcast и interface относятся непосредственно к оператору rip. Ключевые слова version, multicast и authentication входят в состав уточнения interface.

Ключевое слово nobroadcast запрещает узлу распространять собственные обновления RIP. По умолчанию принимается режим nobroadcast, если в системе всего один сетевой интерфейс, и broadcast, если интерфейсов несколько. Ключевое слово nobroadcast выполняет ту же функцию, что и ключ -q командной строки routed. gated, впрочем, способен на большее, чем routed, как видно из следующего уточнения.

Уточнение interface определяет параметры интерфейса для RIP. Параметры, связанные с уточнением в нашем примере, предписывают принимать обновления RIP-2 через групповой адрес RIP-2 интерфейса 172.16.9.23 и что подлинные обновления должны содержать пароль REAL^stuff. В RIP-2 механизм примитивной проверки подлинности построен на открытых текстовых паролях до 16 байтов длиной. Задача механизма – не защитить систему от злоумышленников, но предотвратить неожиданности в процессе настройки маршрутизаторов. Если какой-то пользователь по ошибке настроит свою систему на распространение данных RIP, маловероятно, что он также случайно укажет в своих настройках правильный пароль. Чтобы воспользоваться серьезным механизмом проверки подлинности – алгоритмом вычисления криптографической контрольной суммы MD5 (Message Digest 5) – следует указать значение md5 в уточнении authentication.

Настройки внутренних шлюзов

Настройки шлюзов сложнее, чем настройки рядового узла. Шлюзы всегда имеют несколько интерфейсов и время от времени работают с целыми наборами протоколов маршрутизации. Первый пример настройки связан с внутренним шлюзом, связующим подсеть 9 и несущую магистраль, подсеть 1. В подсети 9 шлюз использует RIP-2, чтобы сообщать о маршрутах Unix-системам. В подсети 1 – OSPF, чтобы обмениваться маршрутами с другими шлюзами. Вот настройки шлюза *limulus*:

```
#  Подсеть 9 не блокируется по времени
interfaces {
    interface 172.16.9.1 passive ;
} ;
# Идентификатор маршрутизатора OSPF
routerid 172.16.1.9 ;
# Включаем RIP-2; передаем OSPF-маршруты
# в подсеть 9, стоимость маршрутов - 5.
rip yes {
    broadcast ;
    defaultmetric 5 ;
    interface 172.16.9.1
         version 2
```

```
          multicast
          authentication simple "REAL stuff" ;
    } ;
    # Включаем OSPF; подсеть 1 является магистральной областью;
    # парольная проверка подлинности.
    ospf yes {
         backbone {
              interface 172.16.1.9 {
                  priority 5 ;
                  auth simple "It'sREAL" ;
                  } ;
              } ;
    } ;
```

Оператор `interfaces` определяет характеристики маршрутизации для сетевых интерфейсов. Ключевое слово `passive` в уточнении `interface` используется в данном случае (как и ранее) для создания статического маршрута, который не будет удален из таблицы маршрутизации. Постоянный маршрут в данной ситуации пролегает через напрямую подключенный сетевой интерфейс. Обычно, если по мнению `gated` работоспособность интерфейса нарушена, демон увеличивает стоимость интерфейса, присваивая ему высокое значение предпочтения (120), чтобы сократить вероятность маршрутизации данных через неработоспособный интерфейс. `gated` начинает считать интерфейс неработоспособным, если не получает обновлений маршрутизации через этот интерфейс. Мы не хотим, чтобы по вине `gated` интерфейс 172.16.9.1 был снижен в категории, даже если демон уверовал в существование проблем: наш маршрутизатор является единственным путем в подсеть 9. Именно поэтому настройки содержат уточнение `interface 172.16.9.1 passive`.

Оператор `routerid` определяет идентификатор маршрутизатора для протокола OSPF. Если идентификатор явным образом не задан в файле настройки, `gated` использует в качестве адреса-идентификатора маршрутизатора адрес первого из найденных интерфейсов. В данном случае мы указываем в качестве идентификатора адрес интерфейса, который действительно говорит на языке OSPF.

В предшествующем примере мы обсудили все уточнения оператора `rip`, за исключением одного `defaultmetric`. Уточнение `defaultmetric` определяет метрику RIP, используемую при распространении маршрутов, полученных от других протоколов маршрутизации. Данный шлюз работает как с OSPF, так и с RIP-2. Мы хотим передавать маршруты, полученные от OSPF, клиентам RIP, а для этого требуется метрика. Мы выбрали метрику RIP, равную 5. В отсутствие уточнения `defaultmetric` маршруты, полученные по протоколу OSPF, не передаются RIP-клиентам.[1] Данный оператор жизненно необходим для нашей сети.

Оператор `ospf yes` включает OSPF. Первое уточнение в этом операторе – `backbone`. Оно показывает, что маршрутизатор входит в состав магистральной

[1] Строго говоря, не совсем так. Маршруты афишируются со стоимостью 16, которая обозначает недостижимые пункты назначения.

области OSPF. Каждый оператор `ospf yes` должен включать, по меньшей мере, одно уточнение `area`. Уточнение `area` определяет конкретную область, например `area 2`, но, по меньшей мере, один маршрутизатор должен существовать в магистральной области. И хотя магистраль OSPF – это область 0, ее нельзя указать как `area 0`; для этой цели следует воспользоваться ключевым словом `backbone`. В нашей сети магистралью является подсеть 1, и все связанные с ней маршрутизаторы принадлежат магистральной области. Один маршрутизатор может принадлежать нескольким областям, и для каждой области могут быть определены собственные параметры. Обратите внимание на группировку уточнений вложенными парами фигурных скобок. Оставшиеся уточнения файла настройки связаны непосредственно с ключевым словом `backbone`.

Уточнение `interface` определяет интерфейс, связывающий маршрутизатор с магистральной областью. Здесь мы наблюдаем два дополнительных уточнения, `priority` и `auth`.

Уточнение `priority 5 ;` указывает приоритет данного маршрутизатора в процессе выбора магистралью отмеченного маршрутизатора. Чем больше номер приоритета, тем менее вероятно, что данный маршрутизатор будет выбран в качестве отмеченного. Расстановка приоритетов способствует выбору наиболее подходящего маршрутизатора.

Уточнение `auth simple "It'sREAL" ;` предписывает использовать простую парольную проверку подлинности в магистральной зоне и содержит текстовый пароль. В Gated 3.6 проверка подлинности сводится к трем вариантам: `none`, `simple` и `md5`. `none` означает отсутствие проверки подлинности. `simple` означает, что должен быть указан верный пароль из восьми символов, в противном случае обновление будет отвергнуто. Парольная проверка подлинности защищает лишь от случайностей, она не предназначена для защиты от злоумышленников. `md5` предписывает выполнять проверку подлинности по алгоритму MD5.

Настройка внешнего шлюза

Самое сложное – настройка шлюза *chill*, работающего с OSPF и BGP. Вот файл настройки шлюза *chill*:

```
# Определяем номер нашей AC для BGP
autonomoussystem 249;

# Идентификатор маршрутизатора OSPF
routerid 172.16.1.1;

# Отключаем RIP
rip no;

# Включаем BGP
bgp yes {
   group type external peeras 164 {
        peer 10.6.0.103 ;
        peer 10.20.0.72 ;
```

```
        };
};

# Включаем OSPF; подсеть 1 является магистральной областью;
# парольная проверка подлинности.
ospf yes {
     backbone {
         interface 172.16.1.1 {
              priority 10 ;
              auth simple "It'sREAL" ;
              } ;
          } ;
};

# Распространять маршруты, полученные от OSPF,
# выполнять маршрутизацию в сеть с прямым доступом
# через BGP в AS 164
export proto bgp as 164 {
     proto direct ;
     proto ospf ;
};

# Распространять маршруты, полученные от BGP из
# AS 164, в области OSPF.
export proto ospfase type 2  {
     proto bgp autonomoussystem 164  {
          all ;
          };
};
```

Такие настройки включают BGP и OSPF и устанавливают определенные параметры того и другого протокола. BGP для работы необходим номер АС, который равен 249 для *books-net*. OSPF необходим адрес-идентификатор маршрутизатора. Последний мы установили в адрес интерфейса маршрутизатора, который работает с OSPF. Номер АС и идентификатор маршрутизатора встречаются раньше прочих настроек, поскольку `autonomoussystem` и `routerid` – операторы определений, которые должны предшествовать первому оператору протокола. Различные типы операторов мы рассматривали выше в табл. 7.2.

Первый оператор протокола – тот, что отключает RIP. Мы не собираемся работать с RIP, но в `gated` RIP по умолчанию включается. Следовательно, его необходимо отключить явным образом, при помощи оператора `rip no;`.

Включение BGP выполняется оператором `bgp yes`, который содержит несколько дополнительных параметров BGP. Уточнение `group` устанавливает параметры для всех BGP-равных узлов группы и определяет тип создаваемого соединения BGP. Пример ориентирован на классическое соединение внешнего протокола маршрутизации, а внешняя автономная система, к которой мы подключаемся, имеет номер АС 164. `gated` позволяет создавать сеансы BGP пяти различных типов, но лишь один из них, тип `external`, используется для прямого обмена данными со внешней автономной системой. Еще четыре

типа групп используются для внутреннего BGP (*internal* BGP, IBGP).[1] IBGP – это просто обозначение протокола BGP, когда он применяется для распространения информации маршрутизации в пределах автономной системы. В нашем примере протокол применяется для передачи информации между автономными системами.

Соседи BGP, от которых принимаются обновления, обозначены уточнениями `peer`. Каждый *равный* (*peer*) является членом группы. Все параметры группы, такие как номер АС, применяются ко всем системам группы. Чтобы разрешить прием обновлений от любой системы АС 164, воспользуйтесь ключевым словом `allow` вместо списка равных.

Протокол OSPF включается оператором `ospf yes`. Настройка OSPF на этом маршрутизаторе ничем не отличается от настройки любого из маршрутизаторов магистральной области. Единственное, что изменилось относительно предыдущего примера, – число приоритета. Поскольку на данный маршрут ложится особенно большая нагрузка, мы решили сделать его маршрутизатор менее интересным для процесса выбора обозначенного маршрутизатора.

Оператор `export` управляет маршрутами, которые `gated` передает другим маршрутизаторам. Первый оператор `export` предписывает `gated` использовать BGP (`proto bgp`) для передачи сведений в автономную систему 164 (`as 164`) обо всех напрямую подключенных сетях (`proto direct`) и маршрутах, полученных от OSPF (`proto ospf`). Обратите внимание, что в этом операторе указан отнюдь не номер АС для *books-net*, а номер АС внешней системы. Первая строка оператора `export` определяет адресатов информации. Уточнения `proto` в фигурных скобках определяют состав передаваемой информации.

Второй оператор `export` распространяет маршруты, полученные от внешней автономной системы. Маршруты принимаются по протоколу BGP и передаются по протоколу OSPF. Поскольку эти маршруты получены от внешней автономной системы, они афишируются в качестве ASE-маршрутов (*autonomous system external*, внешние для автономной системы). Поэтому оператор `export` предписывает использовать для распространения маршрутов протокол `ospfase`. Параметр `type 2` задает тип распространяемых внешних маршрутов. `gated` поддерживает два типа. Маршруты типа 2 – это маршруты, полученные по протоколу внешних шлюзов, не имеющего метрики маршрутизации, совместимой с метрикой OSPF. Эти маршруты афишируются со стоимостью сообщения с пограничным маршрутизатором. В данном случае маршруты афишируются с OSPF-стоимостью сообщения со шлюзом *chill*. Маршруты типа 1 – это маршруты, полученные от внешнего протокола, имеющего метрику, сравнимую с метрикой OSPF. В этом случае при распространении маршрутов метрика внешнего протокола прибавляется к стоимости сообщения с граничным маршрутизатором.

Источником маршрутов, распространяемых по второму оператору `export`, является соединение BGP (`proto bgp`) с автономной системой 164 (`autonomous-`

[1] Все типы групп описаны в приложении В.

system 164). Уточнение proto может дополняться *фильтром маршрутов*. Фильтр маршрутов используется для отбора маршрутов из конкретного источника. Фильтр может перечислять сети и связанные с ними маски для отбора конкретных направлений. В нашем примере ключевое слово all предписывает выбирать все маршруты, полученные по протоколу BGP, что в действительности является умолчанием. Таким образом, мы могли бы опустить ключевое слово all. Однако вреда в его присутствии нет, зато оно четко документирует наши намерения.

Все маршруты, полученные от внешних автономных систем, могут привести к созданию очень большой таблицы маршрутизации. Отдельные маршруты полезны, если существует несколько пограничных маршрутизаторов, реализующих сообщение с внешним миром. Однако если пограничный маршрутизатор всего один, может оказаться, что достаточно маршрута по умолчанию. Чтобы экспортировать маршрут по умолчанию, добавьте оператор options gendefault ; в начало файла настройки.[1] Такой оператор предписывает gated генерировать маршрут по умолчанию, если происходит сообщение с равными BGP-соседями. Кроме того, замените второй оператор export в файле примера на следующий:

```
# Афишировать маршрут по умолчанию при сообщении
# с BGP-равными.
export proto ospfase type 2  {
     proto default ;
};
```

Данный оператор export предписывает gated афишировать пограничный маршрутизатор в качестве шлюза по умолчанию, но только при наличии активных соединений с внешней системой.

Как можно видеть из приведенных примеров, файлы *gated.conf* имеют небольшой размер и легко читаются. Если вам необходимо работать с протоколом маршрутизации на своем компьютере, воспользуйтесь gated. gated — это единое программное обеспечение и единый язык настройки для всех ваших узлов, внутренних и внешних шлюзов.

Тестирование настройки

Проверьте файл настройки перед использованием; синтаксис настройки gated очень сложен, в нем легко ошибиться. Сохраните настройки в тестовом файле, проверьте их, а затем перенесите в файл */etc/gated.conf*. Ниже мы приведем пример того, как это можно сделать.

Предположим, файл настройки называется *test.conf*, и мы его уже создали. Для проверки воспользуемся ключами командной строки -f и -c:

```
% gated -c -f test.conf trace.test
```

[1] Альтернативным способом создания маршрута по умолчанию является оператор generate. За подробностями обращайтесь к приложению B.

Ключ `-f` предписывает `gated` читать настройки из указанного файла (в данном случае – *test.conf*), а не из файла */etc/gated.conf*. Ключ `-c` предписывает `gated` проверить файл настройки на наличие ошибок синтаксиса. Завершив чтение файла, `gated` прекращает работу: таблица маршрутизации при этом не изменяется. Ключ `-c` включает трассировку, поэтому указывайте имя файла для данных трассировки, если не желаете, чтобы они отображались на терминале. В данном случае мы указали имя файла трассировки – *trace.test*. Кроме того, ключ `-c` создает снимок состояния демона после чтения файла настройки и записывает образ в файл */usr/tmp/gated_dump*.[1] Для выполнения `gated` с ключом `-c` нет необходимости в статусе администратора системы, равно как и в принудительном завершении процесса `gated`.

Файлы образа и трассировки (*trace.test*) могут содержать дополнительные данные, включая сведения об ошибках. Убедившись, что настройки верны, поместите содержимое нового файла настройки (*test.conf*) в файл */etc/gated.conf* (для этого необходимы полномочия администратора).

Существует альтернативная программа – `gdc`, которая должна выполняться пользователем `root` либо с полномочиями суперпользователя. Она предоставляет функциональность для проверки и ввода в эксплуатацию новых настроек. `gdc` использует три различных файла настройки. Текущие настройки хранятся в */etc/gated.conf*. Предыдущий вариант настройки хранится в файле */etc/gated.conf–*, а «следующий» вариант настройки хранится в файле */etc/gated.conf+*, для которого как раз и проводится тестирование. Вот так `gdc` проверяет настройки:

```
# cp test.conf /etc/gated.conf+
# gdc checknew
configuration file /etc/gated.conf+ checks out okay
# gdc newconf
# gdc restart
gated not currently running
gdc: /etc/gated was started
```

В данном примере настройки для проверки копируются в */etc/gated.conf+* и проверяются посредством команды `gdc checknew`. Если в файле найдены ошибки синтаксиса, отображается предупреждение, а подробная диагностика записывается в файл */usr/tmp/gated_parse*. В данном примере ошибок нет, так что мы делаем файл-претендент текущим файлом настройки при помощи команды `gdc newconf`. Данная команда перемещает текущие настройки в файл *gated.conf–*, а новые настройки (*gated.conf+*) – в файл текущих. Команда `gdc restart` – ее не было в нашем примере – принудительно завершает `gated`, если демон запущен, и запускает новый экземпляр `gated` с новыми настройками.

---

[1] */usr/tmp* – каталог по умолчанию для записи этого файла и файла *gated_parse*, описанного ниже; однако некоторые системы сохраняют эти файлы в каталоге */var/tmp*.

Запуск gated при загрузке

Как и многие другие программы маршрутизации, демон gated следует включить в процесс загрузки системы. В некоторых системах код для запуска gated уже существует в загрузочных файлах. Если у вас не такая система, код придется создать вручную. Если в загрузочном файле уже существует код для запуска routed, замените его кодом для запуска gated. gated и routed не должны работать одновременно.

Наш воображаемый шлюз *crab* работает под управлением Solaris, и в файле */etc/init.d/inetinit* содержится код для запуска routed. Закомментируем соответствующие строки и добавим такие:

```
if [ -f /usr/sbin/gated -a -f /etc/gated.conf ]; then
     /usr/sbin/gated;       echo -n 'gated' > /dev/console
fi
```

Данный фрагмент кода предполагает, что gated установлен в каталог */usr/sbin* и что файл настройки называется */etc/gated.conf*. Код проверяет присутствие gated и существование файла настройки */etc/gated.conf*. Если существуют оба файла, выполняется запуск gated.

Наличие файла настройки проверяется потому, что демон gated обычно работает с файлом настройки. Если запустить gated и не указать файл настройки, демон проверит наличие в таблице маршрутизации маршрута по умолчанию. Будет использован маршрут по умолчанию, если он существует, и запущен протокол RIP в противном случае. Создавайте файл */etc/gated.conf*, даже если собираетесь работать только с RIP. Файл настройки документирует конфигурацию подсистемы маршрутизации и защищает от возможных изменений стандартных настроек демона в будущем.

Резюме

Маршрутизация – это связующее звено, объединяющее различные сети в сеть Интернет. Без маршрутизации сети не смогут обмениваться данными. Настройка маршрутизации – важная задача сетевого администратора.

Чтобы обмениваться данными по сетевому интерфейсу с напрямую подключенной сетью, необходима примитивная маршрутизация. Такие маршруты можно встретить в таблице маршрутизации – им соответствуют записи без флага G (*gateway*, шлюз). В некоторых системах минимально необходимые маршруты создаются посредством команды ifconfig в процессе настройки интерфейса. В системах Linux маршрут через интерфейс должен быть явно создан командой route.

Команда route используется для создания статических таблиц маршрутизации. Статическая маршрутизация – это маршрутизация, которая требует внимания и сопровождения со стороны сетевого администратора. Маршруты добавляются и удаляются из таблицы маршрутизации при помощи команды route. Чаще всего статическая маршрутизация применяется для создания маршрутов по умолчанию.

В динамической маршрутизации применяются протоколы маршрутизации, осуществляющие выбор лучших маршрутов и обновление таблиц маршрутизации. Существует много различных протоколов динамической маршрутизации. В большинстве систем Unix существует протокол маршрутной информации RIP (Routing Information Protocol). Работа RIP осуществляется посредством демона `routed`. `routed` создает таблицу маршрутизации на основе информации, полученной по сети, а также информации из файла */etc/gateway*.

`gated` – это программный пакет, позволяющий работать с целым рядом протоколов маршрутизации в системах Unix, включая такие сложные протоколы, как OSPF (Open Shortest Path First) и BGP (Border Gateway Protocol). Настройка `gated` выполняется посредством файла */etc/gated.conf*. Команды настройки `gated` описаны в приложении B.

Данная глава – последняя из посвященных созданию физических сетевых соединений. После настройки маршрутизации система приобретает способность к обмену данными. В следующей главе мы начнем изучение различных приложений и служб, делающих сеть действительно полезной.

- *BIND: служба имен Unix*
- *Настройка DNS-клиента*
- *Настройка демона named*
- *Работа с nslookup*

8

Настройка DNS

Поздравляю! Вы установили TCP/IP на уровне ядра, настроили сетевой интерфейс и маршрутизацию. Таким образом, решены все задачи настройки, позволяющие работать с TCP/IP в системе Unix. И хотя материал, который мы изучим далее, не является строго необходимым для работы по TCP/IP, он позволяет сделать сеть более полезной и удобной в работе. В следующих двух главах мы рассмотрим настройку основных сетевых служб TCP/IP. Вероятно, наиболее важной из таких служб является служба имен.

Это, как и следует из названия, действительно служба – говоря точнее, сервис, облегчающий конечным пользователям работу с сетью. Компьютеры вполне могут обойтись IP-адресами, но люди предпочитают иметь дело с именами. Значение службы имен можно оценить по объему материала, отведенного ей в этой книге. В главе 3 шла речь о том, *для чего нужна* служба имен, в этой главе описана ее *настройка*, а в приложении C содержится *подробное описание* команд настройки. Материала данной главы достаточно для настройки пакета BIND[1] и работы с ним. При желании выяснить, почему выполняется та или иная операция, либо получить подробные инструкции, обращайтесь к главе 3 и приложению C.

BIND: служба имен Unix

В Unix система DNS реализована в виде пакета Berkeley Internet Name Domain (BIND). BIND – это программный пакет, работающий по модели кли-

[1] BIND 8 – это версия пакета для работы с доменными именами, поставляемая в составе большинства вариантов Linux и Solaris 8. Доступна и более новая версия пакета DNS – BIND 9. Синтаксис файлов настройки BIND 8 и BIND 9, по существу, совпадает. Приводимые примеры должны работать как в BIND 8, так и в BIND 9.

ент–сервер. Клиентская часть BIND называется *DNS-клиентом* (*resolver*).[1] DNS-клиент генерирует запросы информации, связанной с доменными именами, и передает их серверу. Сервер DNS отвечает на эти запросы. Серверная сторона BIND реализована в виде демона `named` (произносится «нэйм-ди»).

В этой главе мы изучим три основных задачи настройки BIND:

- Настройка DNS-клиента BIND
- Настройка сервера имен BIND (`named`)
- Создание файлов базы данных сервера имен, известных как *файлы зон*

Зона – это сегмент пространства доменных имен, который входит в компетенцию определенного сервера имен. Зона не может содержать домен, делегированный другому серверу. В данном случае мы используем термин «зона» для обозначения файла базы данных DNS, а термин «домен» обозначает более широкий контекст. В данной книге домен – часть иерархии доменов, обозначенная именем домена. Зона – пакет информации о домене, хранящейся в файле базы данных DNS. Файл, содержащий информацию о домене, называется файлом зоны.

Документ RFC 1033, *Domain Administrators Operations Guide* (Руководство администратора домена), описывает базовый набор стандартных записей, из которых состоят файлы зон. Во многих документах RFC предлагаются новые записи DNS, которые не получили широкого распространения. В этой главе и в приложении C я старался не выходить за рамки основных, часто используемых записей ресурсов. Мы используем их для создания файлов зон. Однако способ и сама необходимость создания файлов зон определяются типом настройки BIND, примененным в системе.

Варианты настройки BIND

Варианты настройки BIND определяются типом службы, доступ к которой необходимо обеспечить. Четыре уровня службы для BIND: *чистый клиент*, *специальный кэширующий сервер*, *основной сервер*, *подчиненный сервер*.

DNS-клиент – это программный код, запрашивающий доменную информацию у сервера. В системах Unix DNS-клиент реализуется в виде библиотеки, а не самостоятельного клиентского приложения. Некоторые системы, называемые *чистыми DNS-клиентами*, используют только DNS-клиент (*resolver*), и на них не выполняется сервер имен (`named`). Такие системы очень легко настраивать: достаточно создать и настроить файл */etc/resolv.conf*.

Три других варианта настройки BIND требуют работы на локальной системе программы сервера имен `named`. Приведем эти варианты:

Основной сервер имен (Master)

Основной сервер имен является компетентным (авторитетным, авторитативным) источником любой информации о конкретной зоне. Он загружа-

[1] В литературе могут встретиться и другие переводы термина *resolver* – поисковый анализатор, распознаватель, разрешитель, resolver-библиотека. – *Примеч. ред.*

ет информацию в память из локального дискового файла, создаваемого администратором домена. Этот файл (файл зоны) содержит наиболее актуальную информацию о сегменте иерархии доменов, подчиненном данному серверу имен. Основной сервер имен является компетентным, поскольку он может ответить на любой запрос относительно зоны с полной уверенностью.

Настройка основного сервера требует полного набора файлов: файлов зон для зон прямого и обратного отображения, файла *conf*, файла корневых указателей и кольцевого файла. Ни один другой вариант настройки не требует столь обширного набора файлов.

Подчиненный сервер имен (Slave)

Подчиненный, или вторичный, сервер имен получает всю информацию зоны от основного сервера. Данные зоны передаются основным сервером и сохраняются подчиненным сервером в локальном файле. Такая передача называется соответственно *передачей зоны*. Подчиненный сервер хранит полную копию всех данных зоны и может компетентно отвечать на запросы относительно этой зоны. Таким образом, подчиненный сервер также считается компетентным сервером.

Настройка подчиненного сервера имен не требует создания локальных файлов зон, поскольку эти файлы копируются с основного сервера. Однако другие файлы – файл загрузки, кэширования и кольцевой – должны обязательно присутствовать.

Специальный кэширующий сервер имен (Caching-only)

Специальный кэширующий сервер имен выполняет ту же программу, но не хранит файлы зон. Он узнает ответы на все запросы от других серверов. Получив определенный ответ, сервер кэширует его и использует впоследствии для ответа на идентичные запросы. В принципе, все серверы имен использую кэшированную информацию таким образом, но только специальный кэширующий сервер имен полностью полагается на этот метод. Кэширующий сервер не считается компетентным, поскольку получаемая от него информация – не из первых рук. Данный вариант настройки требует присутствия только загрузочного файла и файла кэширования, но в большинстве случаев присутствует еще и кольцевой файл. Данный вариант настройки является, вероятно, наиболее распространенным, и после варианта чистого DNS-клиента – самым простым.

Сервер имен может действовать по любому из вариантов либо, как очень часто бывает, сочетать несколько вариантов одновременно. При этом на всех системах работают DNS-клиенты, поэтому мы начнем с изучения настройки клиентской стороны программного обеспечения DNS.

Настройка DNS-клиента

Настройка DNS-клиента производится посредством файла */etc/resolv.conf*. DNS-клиент (*resolver*) не является отдельным самостоятельным процессом:

это библиотека подпрограмм, которые вызываются сетевыми процессами. Чтение файла *resolv.conf* происходит в момент, когда начинает работу процесс, нуждающийся в DNS-клиенте, и на время жизни этого процесса кэшируется. Если файл настройки не найден, DNS-клиент пытается подключиться к серверу `named`, работающему на локальном узле. И хотя такой подход может сработать, я не рекомендую им пользоваться. Применение настроек по умолчанию лишает администратора контроля над DNS-клиентом и делает его уязвимым для различий в умолчаниях, принятых в различных системах. По этим причинам следует создавать файл настройки DNS-клиента на любой системе, работающей с BIND.

Файл настройки DNS-клиента

Файл настройки ясно документирует настройки DNS-клиента. Администратор может указать до трех серверов имен, два из которых являются резервными – на случай, если не ответит первый сервер. Кроме того, файл содержит имя домена по умолчанию и прочие параметры работы. Файл *resolv.conf* – важнейшая часть настройки службы имен.

resolv.conf – это простой файл, подходящий для чтения людьми. Существуют некоторые вариации команд файла, зависящие от системы. Ниже перечислены записи, поддерживаемые большинством систем:

`nameserver` *адрес*

Записи `nameserver` указывают IP-адреса серверов, опрашиваемых DNS-клиентом на предмет получения доменной информации. Опрос серверов имен происходит в порядке следования записей `nameserver`. Если от сервера не получен ответ, DNS-клиент посылает запрос следующему серверу по списку, и так до тех пор, пока не будет достигнут последний из серверов.[1] Если файл *resolv.conf* не существует либо в файле отсутствуют записи `nameserver`, все запросы направляются локальному узлу. Однако если файл *resolv.conf* существует и содержит записи `nameserver`, обращение к локальному узлу происходит только в том случае, если присутствует соответствующая запись `nameserver`. Указывайте официальный IP-адрес локального узла (или адрес 0.0.0.0), но не кольцевой адрес. Официальный адрес позволяет избежать проблем, возникающих в некоторых вариантах Unix при использовании кольцевого адреса. Вариант *resolv.conf* для чистого клиента DNS никогда не содержит записи `namerserver`, указывающей на локальный узел.

`domain` *имя*

Запись `domain` определяет доменное имя по умолчанию. DNS-клиент добавляет доменное имя по умолчанию к любому имени узла, не содержащему точки.[2] Дополненное таким образом имя узла используется в запросе к

[1] В большинстве реализаций BIND максимальное число опрашиваемых серверов – 3.

[2] Таков наиболее распространенный способ применения доменных имен по умолчанию, но этот механизм поддается настройке.

серверу имен. Например, если DNS-клиент получает имя *crab* (не содержащее точки), он добавляет доменное имя по умолчанию в процессе конструирования запроса. Предположим, значение имени в записи `domain` – `wrotethebook.com`, тогда DNS-клиент создает запрос для *crab.wrotethebook.com*. Переменная среды LOCALDOMAIN, будучи установленной, имеет приоритет более высокий, чем запись `domain`, и значение переменной используется для дополнения имени узла.

`search` *домен* ...

Запись `search` определяет ряд доменов, в которых производится поиск, если имя узла не содержит точки. Предположим, файл содержит запись `search essex.wrotethebook.com butler.wrotethebook.com`. Запрос для узла по имени *cookbook* будет преобразован сначала в запрос для имени *cookbook.essex.wrotethebook.com*. Если поиск для такого имени не принес положительных результатов, DNS-клиент создает запрос для *cookbook.butler.wrotethebook.com*. Вновь получив отрицательный результат, клиент DNS прервет процесс поиска для имени узла. Используйте запись `search` либо запись `domain`. Предпочтение отдавайте команде `search`. Никогда не используйте обе команды одновременно. Переменная среды LOCALDOMAIN имеет более высокий приоритет, чем запись `search`.

`sortlist` *сеть*[*/маска сети*] ...

Адреса, принадлежащие перечисленным в команде `sortlist` сетям, являются для DNS-клиента предпочтительными. Если DNS-клиент получает несколько адресов в ответ на запрос по многосетевому узлу или маршрутизатору, адреса сортируются таким образом, что адреса сетей `sortlist` предшествуют адресам всех прочих сетей. В ином случае адреса возвращаются приложению в порядке их получения от сервера имен.

Команда `sortlist` используется редко, поскольку затрудняет работу таких серверных механизмов, как распределение нагрузки. Основным исключением является ситуация, когда список сортировки настраивается для предпочтения адресов локальной сети всем прочим адресам. В последнем случае, если клиент DNS состоит в сети 172.16.0.0/16 и один из адресов, полученных в многоадресном ответе, также принадлежит этой сети, адрес сети 172.16.0.0 будет предшествовать всем прочим адресам.

`options` *параметр* ...

Запись `options` позволяет устанавливать необязательные параметры настройки клиента DNS. Доступны следующие параметры:[1]

`debug`

Включает отладку – печать отладочных сообщений на стандартный вывод. `debug` работает только в случае, если библиотека DNS-клиента была собрана с ключом –DDEBUG (в большинстве случаев это не так).

[1] Приведенный перечень отражает параметры BIND 8 для системы Linux. В версии BIND 8 для системы Solaris отсутствуют параметры `rotate`, `no-check-names` и `inet6`.

`ndots:n`

Устанавливает число точек в имени узла, наличие которого служит критерием необходимости использования списка поиска перед отправкой запроса серверу имен. По умолчанию имеет значение 1. Таким образом, к имени узла с одной точкой не добавляется доменное имя перед отправкой серверу имен. Если указать параметр `ndots:2`, к имени узла с одной точкой добавляется домен из списка поиска перед отправкой запроса, но не к имени с двумя или более точками.

Параметр `ndots` может пригодиться, если одну из составляющих имени домена можно спутать с доменом высшего уровня, и пользователи постоянно усекают имена из этого домена. В таком случае запросы будут передаваться для разрешения прежде всего корневым серверам имен для поиска в домене верхнего уровня, прежде чем, в конечном итоге, вернуться на локальный сервер имен. Беспокоить корневые серверы по пустякам – очень плохой тон. Используйте `ndots`, чтобы обязать DNS-клиент принудительно дополнять проблемные имена локальным доменным именем, чтобы разрешение происходило без обращения к корневым серверам.

`timeout:n`

Устанавливает начальный интервал ожидания ответа DNS-клиентом. По умолчанию интервал ожидания равен 5 секундам для первого запроса к каждому серверу. В пакете BIND для системы Solaris 8 данный параметр имеет синтаксис `retrans:n`.

`attempts:n`

Задает число повторных попыток получить ответ на запрос. По умолчанию имеет значение 2, то есть DNS-клиент дважды повторяет попытку получить ответ для каждого из серверов в списке серверов, прежде чем вернуть приложению сообщение об ошибке. В пакете BIND для системы Solaris 8 данный параметр имеет синтаксис `retry:n` и значение по умолчанию 4.

`rotate`

Включает циклический механизм «round-robin» выбора серверов имен. В обычной ситуации DNS-клиент посылает запрос первому серверу из списка, а следующему серверу – лишь в том случае, если первый сервер не ответил на запрос. Параметр `rotate` предписывает DNS-клиенту распределить нагрузку поровну между всеми серверами имен.

`no-check-names`

Отключает проверку доменных имен на соответствие документу RFC 952, *DOD Internet Host Table Specification* (Спецификация таблицы узлов сети Интернет Министерства обороны). По умолчанию доменные имена, содержащие подчеркивание (_), символы не из таблицы ASCII либо управляющие символы ASCII, считаются ошибочными. Воспользуйтесь этим параметром, если существует необходимость работать с именами, содержащими подчеркивание.

`inet6`

Предписывает DNS-клиенту создавать запросы адресов IPv6. В настоящее время в Интернете применяется четвертая версия протокола Internet (IP) – IPv4. Длина адреса в IPv4 составляет 32 бита. Длина адреса в IPv6 – 128 бит.

Чаще всего файл настройки *resolv.conf* содержит в списке поиска локальное доменное имя, указывает локальный узел в качестве первого сервера имен, а также один или два резервных сервера имен. Пример такой настройки:

```
# Файл настройки клиента DNS
#
search wrotethebook.com
# обратись, прежде всего, к себе
nameserver 172.16.12.2
# затем к узлу crab
nameserver 172.16.12.1
# и, наконец, к узлу ora
nameserver 172.16.1.2
```

Пример основан на воображаемой сети, поэтому по умолчанию для доменного имени указано имя *wrotethebook.com*. Файл взят с узла *rodent*, который и обозначен в качестве первого сервера имен. В качестве резервных серверов выступают *crab* и *ora*. Настройка не содержит параметров и списка сортировки, поскольку они применяются нечасто. Так выглядит файл настройки обычного DNS-клиента.

Настройка чистого DNS-клиента

Настройки чистого DNS-клиента очень просты. Они идентичны настройкам обычного клиента, но не указывают локальную систему в качестве сервера имен. Вот пример файла *resolv.conf* для системы чистого DNS-клиента:

```
# Файл настройки DNS-клиента
#
search wrotethebook.com
# обратись к узлу crab
nameserver 172.16.12.1
# затем к узлу ora
nameserver 172.16.1.2
```

Данные настройки предписывают DNS-клиенту передавать все запросы узлу *crab*, а если *crab* не ответил – узлу *ora*. Ни при каких обстоятельствах запросы не разрешаются локально. Столь простой файл настройки – все, что требуется для работы чистого DNS-клиента.

Настройка демона named

Для настройки DNS-клиента необходим всего один файл; однако настройка демона `named` требует присутствия нескольких файлов. Полный набор файлов настройки `named` выглядит следующим образом:

Файл настройки

Определяет общие аспекты работы `named` и указывает источники данных базы DNS, используемой этим сервером. Источниками могут служить локальные дисковые файлы либо удаленные серверы. Файл настройки обычно называется *named.conf*.

Файл корневых указателей

Указывает на серверы корневых зон. Файл корневых указателей обычно называют *named.ca*, *db.cache*, *named.root* или *root.ca*.

Кольцевой файл

Используется для локального разрешения кольцевого адреса. Обычно называется *named.local*.

Файл прямого отображения зоны

Файл зоны, содержащий отображения имен узлов в IP-адреса. Именно этот файл содержит основной массив информации о зоне. Чтобы упростить обсуждение этого файла, в тексте книги он называется просто *файлом зоны*. Файл зоны обычно имеет наглядное имя, позволяющее понять, данные какой зоны хранятся в файле, например *wrotethebook.com.hosts*.

Файл обратного отображения зоны

Файл зоны, содержащий отображения IP-адресов в имена узлов. Чтобы упростить обсуждение этого файла, в тексте книги он называется *файлом обратной зоны*. Файл обратной зоны обычно имеет наглядное имя, позволяющее понять, какие IP-адреса он обслуживает, например *172.16.rev*.

Всем этим файлам администратор волен давать любые имена. Однако для того чтобы облегчить коллегам сопровождение системы, следует использовать наглядные имена для файлов зон, имена *named.conf* и *named.local* для загрузочного файла и файла кольцевого адреса и одно из широко распространенных имен для файла корневых указателей. В последующих разделах мы изучим каждый из описанных файлов, а начнем с файла *named.conf*.

Файл named.conf

Файл *named.conf* указывает демону `named` источники информации DNS. Некоторые из источников представлены локальными файлами; другие – удаленными серверами имен. Создавать следует только файлы, на которые ссылаются операторы master и cache. Мы рассмотрим примеры создания файлов всех типов.

Структура команд настройки файла *named.conf* похожа на структуру программ языка C. Оператор заканчивается точкой с запятой (`;`), константы заключаются в кавычки (`""`), а родственные элементы группируются при помощи фигурных скобок (`{}`). Комментарий может ограничиваться парами символов `/*` и `*/`, как в языке C; может начинаться парой символов `//`, как в языке C++, либо символом `#`, как в сценариях командного интерпретатора. В приводимых примерах использованы комментарии в стиле C++, но, разумеется, можно использовать любой из трех стилей – по вкусу.

В табл. 8.1 описаны основные операторы настройки *named.conf*. Приводимой информации как раз достаточно, чтобы разобраться в примерах. Однако в примерах использованы далеко не все команды настройки *named.conf*, и не все команды вы будете применять в работе. Набор команд спроектирован таким образом, чтобы охватить полный диапазон вариантов настройки, даже настройки корневых серверов. Подробная информация о каждом операторе настройки *named.conf* содержится в приложении C.

Таблица 8.1. Команды настройки named.conf

Команда	Назначение
acl	Определяет список управления доступом, состоящий из адресов IP
include	Включает содержимое внешнего файла в файл настройки
key	Определяет ключи проверки подлинности
logging	Определяет состав журнала сообщений и способ хранения
options	Определяет глобальные параметры настройки и умолчания
server	Определяет свойства удаленного сервера
zone	Определяет зону

Содержимое файла *named.conf* определяет роль сервера: основной сервер зоны, подчиненный сервер зоны или специальный кэширующий сервер. Лучший способ разобраться с различными вариантами настройки – рассмотреть конкретные примеры файлов *named.conf*. В следующих разделах описаны все варианты настройки.

Настройка специального кэширующего сервера

Настройка специального кэширующего сервера проста. Требуются только файлы *named.conf* и *named.ca*, хотя обычно присутствует еще и *named.local*. Для специального кэширующего сервера *named.conf* может выглядеть так:

```
$ cat /etc/named.conf
options {
        directory "/var/named";
};

//
// настройка специального кэширующего сервера
//
zone "." {
        type hint;
        file "named.ca";
};

zone "0.0.127.in-addr.arpa" {
        type master;
        file "named.local";
};
```

Оператор options определяет каталог по умолчанию для демона named. В данном примере – */var/named*. Все последующие ссылки на файлы в *named.conf* интерпретируются относительно этого каталога.

Два оператора zone, присутствующие в данных настройках специального кэширующего сервера, есть во всех вариантах настройки. Первый оператор zone определяет файл указателей, который помогает серверу имен обнаружить корневые серверы имен после начального запуска. Второй оператор zone делает сервер основным для его собственного кольцевого адреса, а также уведомляет, что информация кольцевого домена хранится в файле *named.local*. Кольцевой домен – это домен *in-addr.arpa*[1], отображающий адрес 127.0.0.1 в имя *localhost*. Идея преобразования собственного кольцевого адреса многим кажется вполне разумной, и следует создавать такую запись в файле *named.conf*. Файл указателей и файл локального узла, наряду с файлом *named.conf*, используются во всех вариантах настройки сервера.[2]

Приведенными операторами zone и options ограничивается большинство случаев настройки специального кэширующего сервера имен, однако оператор options может быть более сложным. Время от времени применяются параметры forwarders и forward only. Параметр forwarders предписывает кэширующему серверу передавать все запросы, на которые он не может ответить самостоятельно, указанным серверам. Например:

```
options {
        directory "/var/named";
        forwarders { 172.16.12.1; 172.16.1.2; };
};
```

Данный параметр forwarders приводит к пересылке всех запросов, ответы на которые отсутствуют в локальном кэше, серверам 172.16.12.1 и 172.16.1.2. Параметр forwarders создает внушительный кэш данных DNS на избранных серверах локальной сети. Это сокращает количество запросов, проходящих по территориальной сети, что особенно полезно, если ограничена ширина канала пропускания между локальной и территориальной сетью либо оплачивается использование канала.

Когда сетевой обмен с внешним миром жестко ограничен, используйте параметр forward only, который предписывает локальному серверу всегда использовать один из серверов пересылки:

```
options {
        directory "/var/named";
        forwarders { 172.16.12.1; 172.16.1.2; };
        forward only;
};
```

---

[1] Домены *in-addr.arpa* описаны в главе 4.

[2] BIND 8 требует присутствия файла корневых указателей, тогда как в BIND 9 встроены указатели, которые используются, если файл корневых указателей не существует.

В присутствии такого параметра в файле настройки локальный сервер не будет пытаться самостоятельно ответить на запрос, даже если не может получить ответ на этот запрос от серверов пересылки.

Добавление параметров в оператор `options` не меняет роли сервера. Изменить роль сервера можно только добавлением зон типа `master` или `slave`.

Варианты настройки основного и подчиненного сервера имен

Основой для примеров настройки основного и подчиненного серверов имен станет воображаемый домен *wrotethebook.com*. Вот файл *named.conf*, обозначающий узел *crab* в качестве основного сервера имен домена *wrotethebook.com*:

```
options {
        directory "/var/named";
};

// настройка основного сервера имен
//
zone "." {
        type hint;
        file "named.ca";
};

zone "0.0.127.in-addr.arpa" {
        type master;
        file "named.local";
};

zone "wrotethebook.com" {
        type master;
        file "wrotethebook.com.hosts";
};

zone "16.172.in-addr.arpa" {
        type master;
        file "172.16.rev";
};
```

Параметр `directory` позволяет экономить время при наборе имен файлов. Он сообщает `named`, что все относительные имена файлов (то есть имена, которые не начинаются символом /) в данном файле должны интерпретироваться относительно каталога */var/named*. Данный параметр также сообщает `named`, куда записывать различные файлы, в частности файл образа.

Первые два оператора `zone` связаны с кольцевым адресом и файлом корневых указателей. Эти операторы мы уже обсуждали в процессе настройки специального кэширующего сервера. Их функциональность неизменна, и они встречаются практически в каждом файле *named.conf*.

Первый из новых операторов `zone` гласит, что данный сервер является основным сервером имен для домена *wrotethebook.com* и что данные домена загружаются из файла *wrotethebook.com.hosts*.

Второй из новых операторов zone указывает на файл, в котором хранятся отображения IP-адресов сети 172.16.0.0 в имена узлов. Оператор гласит, что локальный сервер является основным сервером имен для обратного домена *16.172.in-addr.arpa* и что данные домена загружаются из файла *172.16.rev*.

Настройка подчиненного сервера имен отличается от настройки основного лишь строением операторов zone. Оператор zone подчиненного сервера в качестве источника доменной информации обозначает удаленный сервер, а не локальный дисковый файл, а тип такой зоны – slave. В отличие от инструкции file в операторе zone основного сервера, инструкция file в операторе zone подчиненного сервера содержит имя локального файла, в котором будет храниться информация, полученная от удаленного сервера, а вовсе не имя файла, из которого загружается домен. Следующий файл *named.conf* настраивает узел *ora* как подчиненный сервер домена *wrotethebook.com*:

```
options {
        directory "/var/named";
};

// настройка подчиненного сервера имен
//
zone "." {
        type hint;
        file "named.ca";
};

zone "0.0.127.in-addr.arpa" {
        type master;
        file "named.local";
};

zone "wrotethebook.com" {
        type slave;
        file "wrotethebook.hosts";
        masters { 172.16.12.1; };
};

zone "16.172.in-addr.arpa" {
        type slave;
        file "172.16.rev";
        masters { 172.16.12.1; };
};
```

Первый оператор zone (тип slave) гласит, что данный сервер является подчиненным сервером имен для домена *wrotethebook.com*. Оператор предписывает named загружать данные домена *wrotethebook.com* с сервера, IP-адрес которого – 172.16.12.1, и сохранять полученные данные в файле */var/named/wrotethebook.hosts*. Если файл *wrotethebook.hosts* не существует, named создает его, получает данные зоны от удаленного сервера и записывает данные в созданный файл. Если файл существует, named сверяется с удаленным сервером и выясняет, изменились ли данные. Если данные изменились, named загружает обновленные данные и перезаписывает содержимое файла. Если дан-

ные не изменились, named загружает в память содержимое дискового файла и благополучно пропускает передачу зоны.[1] Хранение копии базы данных в локальном файле позволяет избежать передачи зоны при каждой перезагрузке локального узла. Передача зоны должна производиться лишь в том случае, когда данные изменились.

И последний оператор zone гласит, что локальный сервер является также подчиненным сервером имен обратного домена *16.172.in-addr.arpa* и что данные этого домена также должны загружаться с сервера по адресу 172.16.12.1. Данные обратного домена хранятся в локальном файле *172.16.rev*, для которого выполняются все те же правила, что и для файла *wrotethebook.hosts*.

Стандартные записи ресурсов

Описанные выше и перечисленные в табл. 8.1 команды настройки используются только в файле *named.conf*. Все прочие файлы, задействованные в настройке демона named (файл зоны, файл обратной зоны, *named.local* и *named.ca*), хранят информацию базы данных DNS. Эти файлы имеют схожее строение и состоят из записей одних и тех же типов, а именно стандартных записей ресурсов (resource records), известных как RR-записи. RR-записи определены документом RFC 1033, *Domain Administrators Operations Guide*, а также другими RFC. В табл. 8.2 перечислены все стандартные записи ресурсов, используемые в примерах этой главы. Подробные описания этих записей даны в приложении C.

Таблица 8.2. Стандартные записи ресурсов

Текстовое название RR-записи	Тип записи	Назначение
Начало компетенции (Start of Authority)	SOA	Отмечает начало данных зоны и определяет параметры, влияющие на зону в целом
Сервер имен (Nameserver)	NS	Указывает сервер имен домена
Адрес (Address)	A	Обеспечивает преобразование имени узла в адрес
Указатель (Pointer)	PTR	Обеспечивает преобразование адреса в имя узла
Почтовый ретранслятор (Mail Exchange)	MX	Указывает, куда следует доставлять почту, предназначенную определенному доменному имени
Каноническое имя (Canonical Name)	CNAME	Определяет псевдоним для имени узла
Текст (Text)	TXT	Хранит произвольные текстовые строки

[1] В приложении C (раздел «Запись начала компетенции») рассказано, как named определяет, что данные изменились.

Синтаксис записей ресурсов описан в приложении C, но для прочтения примеров настройки, приводимых далее, необходим определенный уровень понимания структуры этих записей.

RR-запись DNS имеет следующий формат:

```
[имя] [ttl] IN тип данные
```

имя

Имя доменного объекта, с которым связана запись. Это может быть конкретный узел либо целый домен. Строка в поле имени интерпретируется относительно текущего домена, за исключением случая, когда заканчивается точкой. Пустое поле имени (то есть содержащее исключительно пробелы) означает, что запись относится к последнему из упомянутых доменных объектов. Например, если за адресной (A) записью для узла *rodent* следует MX-запись с пустым полем имени, считается, что обе записи относятся к узлу *rodent*.

ttl

Время жизни (time-to-live) определяет продолжительность хранения записи в кэше удаленной системы в секундах. Как правило, это поле оставляют пустым, и в таком случае используется время жизни по умолчанию, установленное для всей зоны в целом при помощи директивы $TTL.[1]

IN

Указывает, что RR-запись имеет класс Internet. Существуют и другие классы, но они редко применяются. Любопытствуете? В приложении C описаны прочие классы записей.

тип

Указывает тип RR-записи. Типы записей перечислены в табл. 8.1, в колонке *Тип записи*. Укажите в этом поле одно из этих значений.

данные

Информация, присущая данному типу RR-записи. Например, в случае адресной (A) записи поле данных содержит IP-адрес.

Позже в этой главе мы рассмотрим каждый из оставшихся файлов настройки. В процесс чтения примеров помните, что все стандартные записи ресурсов в этих файлах следуют описанному формату.

Основу файла зоны составляют стандартные записи ресурсов. Кроме того, в BIND существует ряд директив файла зон, которые используются при создании базы данных DNS.

Директивы файла зон

В BIND существует четыре директивы, упрощающих создание файла зон либо определяющих параметры RR-записей. Эти четыре директивы – две ко-

[1] Директива $TTL описана далее в этой главе.

манды, упрощающие создание файла зоны ($INCLUDE и $GENERATE), и две – определяющие значения для записей ресурсов – $ORIGIN и $TTL.

Директива $TTL

Директива $TTL задает значение времени жизни по умолчанию для RR-записей, не содержащих явного указания параметра TTL. Значение времени может определяться в секундах либо сочетаниями чисел и букв. Недельное время жизни по умолчанию определяется в численном формате так:

```
$TTL 604800
```

Одна неделя эквивалентна 604800 секундам. Посредством алфавитно-цифрового формата одну неделю можно задать гораздо проще:

```
$TTL 1w
```

Допустимые значения алфавитно-цифрового формата:

- `w` – неделя
- `d` – день
- `h` – час
- `m` – минута
- `s` – секунда

Директива $ORIGIN

Директива $ORIGIN устанавливает текущую зону, то есть доменное имя, которым дополняются все относительные доменные имена. Относительным доменным именем считается любое имя, которое не заканчивается точкой. По умолчанию $ORIGIN принимает значение доменного имени, указанного в операторе `zone`. Используйте директиву $ORIGIN, чтобы изменить это значение.

Директива $INCLUDE

Директива $INCLUDE включает содержимое внешнего файла в качестве фрагмента файла зоны. Внешний файл включается в файл зоны в точке присутствия директивы $INCLUDE.

Директива $GENERATE

Директива $GENERATE используется для создания серий RR-записей. Записи ресурсов, создаваемые посредством $GENERATE, практически идентичны и различаются только числовыми индексами. Например:

```
$ORIGIN 20.16.172.in-addr.arpa.
$GENERATE 1-4 $ CNAME $.1to4
```

За ключевым словом $GENERATE следует диапазон записей, которые необходимо создать. В данном случае диапазон охватывает индексы от 1 до 4.

За диапазоном следует шаблон RR-записей. В данном случае шаблон выглядит так: `$ CNAME $.1to4`. Символ `$` в шаблоне заменяется текущим значением индекса. В примере значение индекса изменяется в диапазоне от 1 до 4. Таким образом, данная директива $GENERATE приводит к созданию следующих записей:

```
1 CNAME 1.1to4
2 CNAME 2.1to4
3 CNAME 3.1to4
4 CNAME 4.1to4
```

Учитывая, что текущей зоной является *20.16.172.in-addr.arpa.*, данные записи ресурсов идентичны следующим:

```
1.20.16.172.in-addr.arpa. CNAME 1.1to4.20.16.172.in-addr.arpa.
2.20.16.172.in-addr.arpa. CNAME 2.1to4.20.16.172.in-addr.arpa.
3.20.16.172.in-addr.arpa. CNAME 3.1to4.20.16.172.in-addr.arpa.
4.20.16.172.in-addr.arpa. CNAME 4.1to4.20.16.172.in-addr.arpa.
```

Эти странного вида записи полезны для делегирования обратных поддоменов. О делегировании доменов мы поговорим чуть позже.

За исключением *named.conf*, все файлы настройки BIND состоят из стандартных записей ресурсов и директив. Все четыре оставшихся файла настройки – файлы базы данных. Два из них, *named.ca* и *named.local*, существуют на каждом сервере независимо от его типа.

Файл инициализации кэша

Оператор `zone` в файле *named.conf*, имеющий тип `hints`, указывает на файл инициализации кэша. Каждый сервер, кэширующий данные, имеет такой файл. Файл содержит информацию, позволяющую начать построение кэша данных домена после запуска сервера имен. Корневой домен обозначается в операторе `zone` одной точкой в поле имени домена, поскольку файл инициализации кэша содержит имена и адреса корневых серверов.

Файл *named.ca* называют файлом «указателей» потому, что он содержит указания, позволяющие демону `named` инициализировать кэш. Указатели предоставляют данные об именах и адресах корневых серверов имен. Файл указателей помогает локальному серверу обнаружить корневой сервер во время запуска. Когда найден один корневой сервер, локальный сервер получает от него официальный перечень корневых серверов. Указатели используются вновь только в случае принудительного перезапуска локального сервера. Информация из файла *named.ca* используется нечасто, но она критична для начального этапа работы сервера `named`.

Простейший файл *named.ca* содержит NS-записи с именами корневых серверов и A-записи с адресами корневых серверов. Пример такого файла:

```
;
.                       3600000  IN  NS   A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.     3600000  IN  A    198.41.0.4
;
```

```
.                       3600000      NS    B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.     3600000  IN  A     128.9.0.107
;
.                       3600000      NS    C.ROOT-SERVERS.NET.
C.ROOT-SERVERS.NET.     3600000  IN  A     192.33.4.12
;
.                       3600000      NS    D.ROOT-SERVERS.NET.
D.ROOT-SERVERS.NET.     3600000  IN  A     128.8.10.90
;
.                       3600000      NS    E.ROOT-SERVERS.NET.
E.ROOT-SERVERS.NET.     3600000  IN  A     192.203.230.10
;
.                       3600000      NS    F.ROOT-SERVERS.NET.
F.ROOT-SERVERS.NET.     3600000  IN  A     192.5.5.241
;
.                       3600000      NS    G.ROOT-SERVERS.NET.
G.ROOT-SERVERS.NET.     3600000  IN  A     192.112.36.4
;
.                       3600000      NS    H.ROOT-SERVERS.NET.
H.ROOT-SERVERS.NET.     3600000  IN  A     128.63.2.53
;
.                       3600000      NS    I.ROOT-SERVERS.NET.
I.ROOT-SERVERS.NET.     3600000  IN  A     192.36.148.17
;
.                      3600000       NS   J.ROOT-SERVERS.NET.
J.ROOT-SERVERS.NET.    3600000   IN  A    198.41.0.10
;
.                      3600000       NS   K.ROOT-SERVERS.NET.
K.ROOT-SERVERS.NET.    3600000   IN  A    193.0.14.129
;
.                      3600000       NS   L.ROOT-SERVERS.NET.
L.ROOT-SERVERS.NET.    3600000   IN  A    198.32.64.12
;
.                      3600000       NS   M.ROOT-SERVERS.NET.
M.ROOT-SERVERS.NET.    3600000   IN  A    202.12.27.33
```

Данный файл содержит только NS- и A-записи. Каждая NS-запись определяет сервер имен для корневого домена (.). Соответствующая A-запись содержит адрес для каждого корневого сервера. Значение TTL для всех записей равно 3600000, что примерно соответствует 42 дням.

Копию файла *named.ca* можно сделать с файла *domain/named.root*, хранящегося на анонимном FTP-сервере *ftp.rs.internic.net*. Данный файл имеет формат, подходящий для применения в Unix-системе. Приводимый ниже листинг отражает копирование файла корневых указателей администратором системы в локальный файл *named.ca*. Полученный файл можно даже не редактировать, он готов к применению.

```
# ftp ftp.rs.internic.net
Connected to rs.internic.net.
220-*****Welcome to the InterNIC Registration Host  *****
    *****Login with username "anonymous"
```

```
     *****You may change directories to the following:
       policy            - Registration Policies
       templates         - Registration Templates
       netinfo           - NIC Information Files
       domain            - Root Domain Zone Files
220 And more!
Name (ftp.rs.internic.net:craig): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password: craig@wrotethebook.com
230 Guest login ok, access restrictions apply.
Remote system type is Unix.
Using binary mode to transfer files.
ftp> get /domain/named.root /var/named/named.ca
local: /var/named/named.ca remote: /domain/named.root
200 PORT command successful.
150 Opening BINARY mode data connection for /domain/named.root (2769 bytes).
226 Transfer complete.
2769 bytes received in 0.998 secs (2.7 Kbytes/sec)
ftp> quit
221 Goodbye.
```

Обновляйте файл *named.root* раз в несколько месяцев, чтобы информация о корневых серверах имен всегда была актуальной. Некорректные сведения о корневых серверах имен могут служить источником проблем для локального сервера. Приведенные выше данные являются актуальными на момент публикации книги, но могут и измениться.

Если ваша система не подключена к Интернету, она не сможет обмениваться данными с корневыми серверами. В таком случае инициализация файла указателей перечисленными выше данными серверов будет бесполезна, и следует хранить в этом файле указатели на крупные серверы имен локальной сети. Эти серверы должны быть настроены на обработку запросов по «корневому» домену. Однако этот корневой домен содержит только NS-записи доменных серверов, расположенных в локальной сети. Например, предположим, что домен *wrotethebook.com* не подключен к Интернету, а узлы *crab* и *horseshoe* функционируют в качестве корневых серверов этого изолированного домена. Файл *named.conf* узла *crab* обозначает этот сервер в качестве основного для корневого домена. *horseshoe* – подчиненный сервер корневого домена. Они загружают данные корневого домена из файла зоны, начинающегося с SOA-записи, обозначающей *crab* в качестве сервера и предоставляющей внутреннюю точку контакта. За SOA-записью в файле следуют NS-записи и A-записи, обозначающие узлы *crab* и *horseshoe* в качестве компетентных серверов корневого домена и делегирующие домены *wrotethebook.com* и *16.172.in-addr.arpa* локальным серверам имен, на которые возложена задача обслуживания этих доменов. (О том, как делегируются домены, мы поговорим позже в этой главе.) Подробности данного варианта настройки рассмотрены в книге Альбитца и Ли «DNS и BIND» (издательство O'Reilly & Associates).[1]

[1] П. Альбитц и К. Ли «DNS и BIND» – Пер. с англ. – СПб: Символ-Плюс, 2002.

Файл named.local

Файл *named.local* позволяет осуществлять преобразование адреса127.0.0.1 («кольцевого адреса») в имя *localhost* и является файлом зоны обратного домена *0.0.127.IN-ADDR.ARPA*. Поскольку адрес 127.0.0.1 используется в качестве «кольцевого» во всех системах, содержание этого файла идентично практически для всех серверов. Вот пример файла *named.local*:

```
$TTL    86400
@       IN  SOA     crab.wrotethebook.com. alana.crab.wrotethebook.com. (
                    1               ; Порядковый номер
                    360000          ; Обновление каждые 100 часов
                    3600            ; Повторение попытки через 1 час
                    3600000         ; Устаревание через 1000 часов
                    3600            ; TTL кэша 1 час
                    )
        IN  NS      crab.wrotethebook.com.
0       IN  PTR     loopback.
1       IN  PTR     localhost.
```

Большинство файлов зон начинаются подобно этому – директивой $TTL. Данная директива устанавливает значение времени жизни по умолчанию для всех записей ресурсов данной зоны. Значение может переопределяться для каждой в отдельности записи – явным указанием TTL.

Записи SOA и NS определяют зону и сервер имен зоны. Первая запись PTR связывает сеть 127.0.0.0 с именем *loopback*, что является альтернативой созданию подобной связи в файле */etc/networks*. Вторая запись PTR – самая главная в этом файле. Она связывает узел с адресом 1 в сети 127.0.0 с именем *localhost*.

Поля данных записи SOA и запись NS с именем конкретного компьютера меняются от системы к системе. В данном примере запись начала компетенции (SOA) обозначает узел *crab.wrotethebook.com*. в качестве сервера, распространяющего данные зоны, а почтовый адрес *alana.crab.wrotethebook.com*. – в качестве контактных координат, которыми можно воспользоваться, если возникнут какие-либо вопросы относительно зоны. (Обратите внимание, что в SOA-записи адрес электронной почты записывается в формате пользователь.узел: пользователь – *alana*, узел – *crab.wrotethebook.com*. Доменные имена заканчиваются точками, то есть являются абсолютными и не подлежат дополнению доменным именем по умолчанию.) Запись NS также содержит имя узла. Изменив эти три поля данных, вы сможете воспользоваться полученным файлом на любом узле.

Файлы, которые мы уже обсудили – *named.conf*, *named.ca* и *named.local*, позволяют полностью настроить кэширующий сервер или подчиненный сервер. В большинстве ваших серверов будут использоваться только эти файлы, причем их содержание для различных серверов будет отличаться незначительно. Простейший способ создать файлы – скопировать примеры и изменить их под нужды своих систем. В составе большинства систем поставляют-

ся примеры таких файлов. Если в вашей системе их нет, возьмите файлы настройки с работающего сервера имен.

Оставшиеся файлы настройки named более сложны, но и менее востребованы, если брать общее число серверов имен. Полный набор файлов настройки нужен лишь основному серверу, а в каждой зоне основной сервер только один.

Файл обратной зоны

Файл обратной зоны весьма схож строением с файлом *named.local*. Оба файла преобразуют IP-адреса в имена узлов, поэтому оба содержат записи PTR.

Файл *172.16.rev* в нашем случае является файлом обратной зоны для домена *16.172.in-addr.arpa*. Администратор домена создает этот файл на узле *crab*, а все прочие узлы, нуждающиеся в хранимой информации, обращаются к серверу *crab*.

```
$TTL 86400
;
;        Отображения адресов в имена узлов.
;
@       IN      SOA     crab.wrotethebook.com. jan.crab.wrotethebook.com. (
                                2001061401   ;   Порядковый номер
                                21600        ;   Обновление
                                1800         ;   Повтор
                                604800       ;   Устаревание
                                900 )        ;   TTL кэша
                IN      NS      crab.wrotethebook.com.
                IN      NS      ora.wrotethebook.com.
                IN      NS      bigserver.isp.com.
1.12            IN      PTR     crab.wrotethebook.com.
2.12            IN      PTR     rodent.wrotethebook.com.
3.12            IN      PTR     horseshoe.wrotethebook.com.
4.12            IN      PTR     jerboas.wrotethebook.com.
2.1             IN      PTR     ora.wrotethebook.com.
6               IN      NS      linuxuser.articles.wrotethebook.com.
                IN      NS      horseshoe.wrotethebook.com.
```

Как и в прочих файлах зон, в качестве первой RR-записи в файле обратной зоны выступает запись SOA. Символ @ в поле имени SOA-записи является ссылкой на текущую зону. Поскольку данный файл зоны не содержит директивы $ORIGIN, явным образом задающей текущую зону, текущей зоной является домен *16.172.in-addr.arpa*, обозначенный оператором zone этого файла в файле *named.conf*:

```
zone "16.172.in-addr.arpa" {
        type master;
        file "172.16.rev";
};
```

Символ @ в SOA-записи позволяет оператору zone определять домен для файла зоны. Точно такая же запись SOA используется для всех зон; она всег-

да ссылается на верное имя домена, поскольку ссылается на домен, определенный для текущей зоны в файле *named.conf*. Измените имя узла (*crab.wrotethebook.com.*) и почтовый адрес руководителя (*jan.crab.wrotethebook.com.*), и записью можно будет пользоваться в любом файле зоны.

NS-записи, следующие за SOA-записью, определяют серверы имен домена. Как правило, серверы имен перечисляются сразу после SOA-записи с пустыми полями имен. Вспомните, что пустое поле имени означает, что запись находится в области действия последнего упомянутого доменного имени. Таким образом, последующие NS-записи относятся к тому же домену, что и запись SOA.

Основу файла обратной зоны составляют записи PTR, поскольку они используются для преобразования адресов в имена узлов. Записи PTR в нашем примере обеспечивают преобразование адреса в имя для узлов 12.1, 12.2, 12.3, 12.4 и 2.1 сети 172.16. Поскольку значения в полях имен PTR-записей не заканчиваются точкой, они интерпретируются относительно текущего домена. Например, значение 3.12 интерпретируется как *3.12.16.172.in-addr.arpa*. В поле данных PTR-записи содержится абсолютное имя узла, что предотвращает интерпретацию имени в качестве относительного для текущего имени домена и потому оканчивается точкой. Используя информацию в этой PTR-записи, `named` преобразует *3.12.16.172.in-addr.arpa* в *horseshoe.wrotethebook.com*.

Последние две строки файла содержат дополнительные NS-записи. Как любой другой домен, *in-addr.arpa* может содержать поддомены. Две последние NS-записи реализуют как раз создание поддомена; они указывают узлы *horseshoe* и *linuxuser* в качестве серверов имен для поддомена *6.16.172.in-addr.arpa*. Любой запрос, касающийся информации поддомена *6.16.172.in-addr.arpa*, перенаправляется к этим серверам. NS-записи, указывающие на серверы поддомена, должны быть размещены в домене более высокого уровня, прежде чем к поддомену можно будет обращаться.

Доменные имена – не то же самое, что IP-адреса, и структура этих сущностей различна. Когда IP-адрес преобразуется в доменное имя *in-addr.arpa*, четыре байта адреса интерпретируются в качестве четырех компонентов одного имени. В действительности IP-адрес – это 32 последовательных бита, а не четыре отдельных байта. Подсети разделяют адресное пространство IP, а маски подсетей – побитовые, то есть не ограничены рамками отдельных байтов. Ограничение поддоменов рамками байтов делает их менее гибкими, чем подсети, которые эти поддомены должны поддерживать. В нашем примере домен *in-addr.arpa* делегирует поддомен по границе байта, то есть каждый байт адреса считается самостоятельным «именем». Таков простейший механизм делегирования обратных поддоменов, но в отдельных случаях он может оказаться недостаточно гибким.

Приведенный ранее пример использования директивы $GENERATE позволяет сделать делегирование обратных доменов более гибким. Директива $GENERATE создала CNAME-записи, отображающие диапазон адресов домена *in-addr.arpa* в другой домен, с более гибкими правилами для доменных имен. Доменные имена *in-addr.arpa* должны состоять из четырех числовых

полей, соответствующих четырем байтам IP-адреса, и строки *in-addr.arpa*. Мы связали эти имена с более длинными и более гибкими именами посредством директивы $GENERATE. Вот еще один (более серьезный) пример работы $GENERATE:

```
$ORIGIN 30.168.192.in-addr.arpa.
$GENERATE 0-63    $ CNAME $.1ST64
$GENERATE 63-127  $ CNAME $.2ND64
$GENERATE 128-191 $ CNAME $.3RD64
$GENERATE 192-255 $ CNAME $.4TH64
```

Эти четыре команды $GENERATE выполняют отображение 256 численных имен домена *30.168.192.in-addr.arpa* в четыре других домена, по 64 численных имени в каждом. Когда удаленный сервер запрашивает PTR-запись для *52.30.168.192.in-addr.arpa*, он получает уведомление, что каноническое имя этого узла – *52.1st64.30.168.192.in-addr.arpa*, и что следует запрашивать запись-указатель этого узла у сервера домена *1st64.30.168.192.in-addr.arpa*. По сути дела, директива $GENERATE позволяет разделить один домен *30.168.192.in-addr.arpa* на ряд доменов. После разделения каждый из фрагментов может быть делегирован произвольному серверу имен.

Делегирование поддоменов может усложнить работу с обратными доменами.[1] Однако в большинстве случаев файл обратной зоны проще, чем файл прямого отображения зоны.

Файл прямого отображения зоны

Файл прямого отображения зоны содержит большую часть доменной информации. Данный файл обеспечивает преобразование имен узлов в IP-адреса, поэтому содержит преимущественно A-записи, но также записи типов MX, CNAME и других. Файл зоны, как и файл обратной зоны, создается только для основного сервера имен, все прочие серверы получают информацию от основного.

```
$TTL 86400
;
;       Адреса и прочие сведения об узлах.
;
@       IN      SOA     crab.wrotethebook.com. jan.crab.wrotethebook.com. (
                                2001061401   ;   Порядковый номер
                                21600        ;   Обновление
                                1800         ;   Повтор
                                604800       ;   Устаревание
                                900 )        ;   TTL кэша
;       Определения серверов имен и почтовых серверов
                IN      NS      crab.wrotethebook.com.
```

[1] Еще более сложные примеры приводятся в упоминавшейся ранее книге Альбитца и Ли «DNS и BIND», 4-е издание.

```
                IN      NS      ora.wrotethebook.com.
                IN      NS      bigserver.isp.com.
                IN      MX      10 crab.wrotethebook.com.
                IN      MX      20 horseshoe.wrotethebook.com.
;
;       Определение localhost
;
localhost       IN      A       127.0.0.1
;
;       Определения узлов зоны
;
crab            IN      A       172.16.12.1
loghost         IN      CNAME   crab.wrotethebook.com.
rodent          IN      A       172.16.12.2
                IN      MX      5 crab.wrotethebook.com.
mouse           IN      CNAME   rodent.wrotethebook.com.
horseshoe       IN      A       172.16.12.3
jerboas         IN      A       172.16.12.4
ora             IN      A       172.16.1.2
;       В таблице узлов для адреса 10.104.0.19 существует запись как узла, так и шлюза
wtb-gw          IN      A       10.104.0.19
;
;    Связующие записи для серверов в пределах домена
;
linuxmag.articles  IN      A       172.16.18.15
24seven.events     IN      A       172.16.6.1
;
;       Определения поддоменов
;
articles           IN      NS      linuxmag.articles.wrotethebook.com.
                   IN      NS      horseshoe.wrotethebook.com.
events             IN      NS      24seven.events.wrotethebook.com.
                   IN      NS      linuxmag.articles.wrotethebook.com.
```

Подобно файлу обратной зоны, файл зоны начинается с записи SOA и нескольких записей NS, которые определяют домен и серверы домена; но при этом файл зоны содержит более представительный набор RR-записей. Используя приведенный пример, рассмотрим каждую из этих записей в порядке их следования.

Первая MX-запись обозначает почтовый сервер домена. Ее смысл: узел *crab* является почтовым сервером домена *wrotethebook.com* и имеет значение предпочтения 10. Почтовые сообщения с адресом вида *user@wrotethebook.com* передаются узлу *crab* для доставки. Разумеется, успешная доставка почты узлом *crab* подразумевает его настройку в качестве почтового сервера. Запись MX – лишь часть процесса настройки. Настройка sendmail описана в главе 10.

Вторая запись MX обозначает узел *horseshoe* в качестве почтового сервера домена *wrotethebook.com* со значением предпочтения 20. Значения предпочтений позволяют создавать указатели на альтернативные почтовые серверы.

Чем ниже значение предпочтения, тем более предпочтителен сервер. Следовательно, две приведенные MX-записи сообщают следующее: «Почту домена *wrotethebook.com* доставлять через узел *crab*; если *crab* не доступен, попробовать доставить через *horseshoe*». Значения предпочтений позволяют создавать указатели на резервные серверы, а не полагаться на единственный почтовый сервер. Если основной почтовый сервер по каким-либо причинам недоступен, почта домена передается одному из резервных серверов.

Приведенные MX-записи осуществляют перенаправление почты, адресованной в домен *wrotethebook.com*, но почта для *user@jerboas.wrotethebook.com* по-прежнему будет передаваться напрямую узлу *jerboas.wrotethebook.com*, а не *crab* или *horseshoe*. Данный вариант настройки позволяет использовать упрощенную адресацию вида *user@wrotethebook.com*, но по-прежнему разрешает выполнять прямую доставку почты отдельным узлам.

Первая A-запись в данном примере определяет адрес узла *localhost*, выполняя задачу, противоположную той, что возложена на соответствующую запись PTR из файла *named.local*. Адресная запись позволяет пользователям домена *wrotethebook.com* набрать имя *localhost* и получить доступ к системе по адресу 127.0.0.1 при помощи локального сервера имен.

Следующая A-запись определяет IP-адрес узла *crab*, который является основным сервером этого домена. За этой адресной записью следует запись CNAME, определяющая имя *loghost* в качестве псевдонима узла *crab*.

За адресной записью для узла *rodent* следует MX-запись и CNAME-запись. (Обратите внимание: записи, относящиеся к одному узлу, сгруппированы. Такое строение файла зоны встречается чаще всего.) MX-запись узла *rodent* направляет всю почту, адресованную *user@rodent.wrotethebook.com*, узлу *crab*. Наличие этой записи объясняется тем, что первая MX-запись перенаправляет почту только в том случае, если она адресована *user@wrotethebook.com*. Если требуется перенаправление почты, адресованной узлу *rodent*, для него следует создать дополнительную MX-запись.

Поле имени CNAME-записи содержит псевдоним официального имени узла. Официальное, или *каноническое*, имя содержится в поле данных этой записи. Приведенные CNAME-записи позволяют обращаться к узлу *crab* по имени *loghost*, а к узлу *rodent* – по имени *mouse*. Псевдоним *loghost* – это обобщенное имя узла, позволяющее направлять вывод демона `syslogd` узлу *crab*.[1] Псевдонимы узлов *не следует* использовать в других RR-записях.[2] Например, не используйте псевдоним в качестве имени почтового сервера в MX-записи. Напротив, используйте только каноническое (официальное) имя, определенное в адресной записи.

В каждом конкретном случае файл зоны может быть гораздо больше, чем только что рассмотренный пример, но будет содержать в большинстве случа-

[1] Более подробно обобщенные имена описаны в главе 3.

[2] Дополнительные сведения о применении записи CNAME в файлах данных зоны содержатся в приложении C.

ев преимущественно подобные записи. Если доступны имена и адреса узлов домена, вы обладаете достаточным объемом информации для создания файлов настройки named.

Управление процессом named

После создания файла *named.conf* и необходимых файлов зон запустите named. named обычно запускается в процессе загрузки системы – из загрузочного сценария. В системе Solaris 8 named запускается сценарием */etc/init.d/inetsvc*. В системе Red Hat Linux – сценарием */etc/rc.d/init.d/named*. Сценарий Red Hat может выполняться из командной строки и имеет ряд необязательных аргументов. Например, в системе Red Hat для останова сервера имен может применяться следующая команда:

```
# /etc/rc.d/init.d/named stop
```

Чтобы возобновить работу службы имен, выполните команду:

```
# /etc/rc.d/init.d/named start
```

Загрузочные сценарии замечательно работают, но более эффективным инструментом управления процессом named является программа ndc (named control). Она поставляется в составе BIND 8 и предоставляет набор функций, облегчающих управление демоном службы имен. Аналогичный инструмент в BIND 9 называется rndc. В табл. 8.3 перечислены параметры ndc.[1]

Таблица 8.3. Параметры ndc

Параметр	Назначение
status	Отображает состояние процесса named
dumpdb	Записывает образ кэша в файл *named_dump.db*[1]
reload	Перезагружает сервер имен
stats	Записывает статистику в файл *named.stats*
trace	Включает запись отладочной информации в файл *named.run*
notrace	Отключает запись отладочной информации и закрывает файл *named.run*
querylog	Переключает регистрацию запросов. При включенной регистрации запросов поступающие запросы фиксируются посредством демона syslogd
start	Запускает named
stop	Останавливает named
restart	Останавливает текущий процесс named и запускает новый

[1] Данный файл сохраняется в каталоге, определенном параметром directory в файле *named.conf*.

[1] На момент написания книги команды status, trace и restart еще не реализованы в rndc.

Параметры ndc легко понять и применить. Следующие команды останавливают, а затем запускают процесс named:

```
# ndc stop
# ndc start
new pid is 795
```

В данной последовательности команд предполагается наличие некоторой паузы между остановом старого процесса named и запуском нового. Чтобы перезапустить процесс named действительно быстро, воспользуйтесь параметром restart:

```
# ndc restart
new pid is 798
```

При первом запуске named обращайте внимание на сообщения об ошибках. named заносит сообщения об ошибках в файл *messages*.[1] Как только демон named заработает, воспользуйтесь командой nslookup, чтобы обратиться к серверу и убедиться, что он предоставляет корректные сведения.

Работа с nslookup

nslookup – это инструмент отладки, который входит в состав пакета BIND. Программа позволяет пользователю напрямую обращаться к серверу имен с запросами и получать любую информацию, хранимую в распределенной базе данных DNS. Команда nslookup помогает определить факт работоспособности сервера и корректность его настройки, а также запросить информацию, которой владеют удаленные серверы.

Программа nslookup позволяет выполнять запросы в диалоговом режиме либо при помощи параметров командной строки. В следующем примере nslookup применяется для создания запроса IP-адреса определенного узла:

```
% nslookup crab.wrotethebook.com
Server:  rodent.wrotethebook.com
Address:  172.16.12.2

Name:    crab.wrotethebook.com
Address:  172.16.12.1
```

Пользователь просит программу nslookup отобразить адрес узла *crab.wrotethebook.com*. nslookup отображает имя и адрес сервера, который использовался для получения ответа, а затем и собственно ответ. Удобная возможность, однако nslookup чаще используется в диалоговом режиме.

В диалоговом режиме работы заключена истинная сила nslookup. Чтобы приступить к работе в диалоговом режиме, выполните команду nslookup без ар-

[1] Данный файл хранится в каталоге */usr/adm/messages* нашей системы Solaris и в каталоге */var/log/messages* нашей системы Red Hat. В вашей системе он может храниться в ином месте: сверьтесь с документацией.

гументов. Завершается диалоговый сеанс сочетанием клавиш <Ctrl>+<D> (^D) или набором команды `exit` в приглашении `nslookup`. В качестве диалогового сеанса предшествующий запрос выглядит следующим образом:

```
% nslookup
Default Server:  rodent.wrotethebook.com
Address:  172.16.12.2

> crab.wrotethebook.com
Server:  rodent.wrotethebook.com
Address:  172.16.12.2

Name:    crab.wrotethebook.com
Address:  172.16.12.1
 > ^D
```

По умолчанию `nslookup` выполняет запрос A-записей, но при помощи команды `set type` можно запросить любой тип RR-записей либо выполнить специальный запрос с типом ANY. ANY позволяет получить все доступные RR-записи для конкретного узла.[1]

В следующем примере выполняется поиск MX-записей для узлов *crab* и *rodent*. Обратите внимание, что указание типа запроса действует на все последующие запросы – автоматического возврата к типу A не происходит. Чтобы выполнить сброс типа запроса, необходима еще одна команда `set type`.

```
% nslookup
Default Server:  rodent.wrotethebook.com
Address:  172.16.12.2

> set type=MX
> crab.wrotethebook.com
Server:  rodent.wrotethebook.com
Address:  172.16.12.2

crab.wrotethebook.com    preference = 5, mail exchanger = crab.wrotethebook.com
crab.wrotethebook.com    inet address = 172.16.12.1

> rodent.wrotethebook.com
Server:  rodent.wrotethebook.com
Address:  172.16.12.2

rodent.wrotethebook.com    preference = 5, mail exchanger = rodent.wrotethebook.com
rodent.wrotethebook.com    inet address = 172.16.12.2
> exit
```

Сервер, к которому обращены запросы, можно изменить при помощи команды `server`. Данная возможность особенно полезна в случае необходимости об-

[1] Смысл словосочетания «все доступные» изменяется в зависимости от типа узла, отвечающего на запрос. Компетентный сервер зоны, хранящий все записи узла, вернет полный набор записей. Некомпетентный сервер, кэширующий информацию об узле, вернет все кэшированные записи – и этот набор может оказаться далеко не полным.

ратиться за информацией напрямую к компетентному серверу. В следующем примере выполняется именно такая операция. Фактически он содержит ряд интересных команд:

- Во-первых, мы выполняем команду `set type=NS` и получаем NS-записи домена *zoo.edu*.
- Исходя из полученной информации, мы выбираем сервер и при помощи команды `server` предписываем `nslookup` обращаться к этому серверу.
- Затем при помощи команды `set domain` определяем домен по умолчанию – *zoo.edu*. `nslookup` использует доменные имена по умолчанию для дополнения имен узлов в запросах точно так же, как это делает клиент DNS для доменного имени по умолчанию, определенного в файле *resolv.conf*.
- Мы переустанавливаем тип запроса в ANY. Если этого не сделать, `nslookup` будет по-прежнему искать NS-записи.
- Наконец, мы запрашиваем информацию об узле *tiger.zoo.edu*. Поскольку значение домена по умолчанию – *zoo.edu*, мы набираем в приглашении `nslookup` только имя узла – *tiger*.

А вот и сам пример:

```
% nslookup
Default Server:  rodent.wrotethebook.com
Address:  172.16.12.2

> set type=NS
> zoo.edu
Server:  rodent.wrotethebook.com
Address:  172.16.12.2

Non-authoritative answer:
zoo.edu nameserver = NOC.ZOO.EDU
zoo.edu nameserver = NI.ZOO.EDU
zoo.edu nameserver = NAMESERVER.AGENCY.GOV
Authoritative answers can be found from:
NOC.ZOO.EDU     inet address = 172.28.2.200
NI.ZOO.EDU      inet address = 172.28.2.240
NAMESERVER.AGENCY.GOV inet address = 172.21.18.31
> server NOC.ZOO.EDU
Default Server:  NOC.ZOO.EDU
Address:  172.28.2.200

> set domain=zoo.edu
> set type=any
> tiger
Server:  NOC.ZOO.EDU
Address:  172.28.2.200

tiger.zoo.edu   inet address = 172.28.172.8
tiger.zoo.edu   preference = 10, mail exchanger = tiger.ZOO.EDU
tiger.zoo.edu   CPU=ALPHA OS=Unix
tiger.zoo.edu   inet address = 172.28.172.8, protocol = 6
         7 21 23 25 79
```

```
tiger.ZOO.EDU   inet address = 172.28.172.8
> exit
```

Последний пример демонстрирует, как загрузить целый домен с компетентного сервера и изучить его в локальной системе. Команда `ls` запрашивает передачу зоны и отображает содержимое полученной зоны.[1] Если в файле зоны много строк, перенаправьте вывод в файл и воспользуйтесь командой `view`, чтобы изучить его содержимое. (`view` сортирует файл и отображает его строки посредством Unix-команды `more`.) Сочетание `ls` и `view` полезно при поиске определенного удаленного узла. В данном примере команда `ls` инициирует передачу зоны *big.com* и сохраняет информацию в файле *temp.file*. Для просмотра *temp.file* применяется `view`.

```
rodent% nslookup
Default Server:  rodent.wrotethebook.com
Address:  172.16.12.2

> server minerals.big.com
Default Server:  minerals.big.com
Address:  192.168.20.1

> ls big.com > temp.file
[minerals.big.com]
########
Received 406 records.
> view temp.file
 acmite                          192.168.20.28
 adamite                         192.168.20.29
 adelite                         192.168.20.11
 agate                           192.168.20.30
 alabaster                       192.168.20.31
 albite                          192.168.20.32
 allanite                        192.168.20.20
 altaite                         192.168.20.33
 alum                            192.168.20.35
 aluminum                        192.168.20.8
 amaranth                        192.168.20.85
 amethyst                        192.168.20.36
 andorite                        192.168.20.37
 apatite                         192.168.20.38
 beryl                           192.168.20.23
--More-q
> exit
```

Приведенные примеры показывают, что `nslookup` позволяет:

- Запрашивать RR-записи любого из стандартных типов
- Напрямую обращаться к компетентным серверам доменов

[1] Из соображений безопасности многие серверы имен не реагируют на команду `ls`. Реализация ограничений на передачу зоны описана в приложении C (параметр `allow-transfer`).

- Записывать полное содержимое домена в файл для последующего просмотра

Чтобы узнать о других возможностях `nslookup`, воспользуйтесь командой `help`. Включите отладку (команда `set debug`) и изучите дополнительную информацию. По мере изучения инструмента `nslookup` вы обнаружите много полезных возможностей.

Резюме

Система доменных имен (Domain Name System, DNS) является важной пользовательской службой, необходимой каждой системе, подключенной к Интернету. Подавляющее большинство реализаций DNS для Unix основано на пакете BIND (Berkeley Internet Name Domain). BIND содержит приложения клиента и сервера DNS.

Клиент BIND, или DNS-клиент (resolver), генерирует запросы и реализован в виде набора библиотечных подпрограмм. Его настройка выполняется посредством файла *resolv.conf*. Клиент DNS существует на всех системах.

Сервер BIND отвечает на запросы и существует в качестве демона. Демон называется `named`. Настройка `named` выполняется посредством файла *named.conf*, определяющего источники доменной информации для сервера и тип сервера. Серверы бывают трех типов: основные, подчиненные и специальные кэширующие. Поскольку кэширование выполняют серверы всех типов, один и тот же вариант настройки может подходить для серверов различных типов.

Исходные записи базы данных DNS хранятся в файлах на основном сервере. Файл базы данных DNS называется файлом зоны. Файл зоны состоит из стандартных записей ресурсов (RR-записей), формат которых определен документами RFC. RR-записи имеют общую структуру и являются средством определения любой информации базы данных DNS.

Тестирование сервера DNS может выполняться при помощи `nslookup`. Данный инструмент входит в комплект поставки BIND.

В данной главе мы научились настраивать и тестировать службу DNS. В следующей главе мы займемся другими службами.

- *Сетевая файловая система (NFS)*
- *Совместный доступ к принтерам Unix*
- *Samba и Windows: совместный доступ к ресурсам*
- *Сетевая информационная служба (NIS)*
- *DHCP*
- *Управление распределенными серверами*
- *Серверы почтовой службы*

9

Локальные службы сети

Переходим к настройке локальных серверов сети. Подобно службе имен, эти серверы не являются строго необходимыми для функционирования сети, но предоставляют службы, позволяющие использовать сеть по назначению.

Сетевых служб существует великое множество – больше, чем мы сможем рассмотреть в этой главе. Поэтому наше внимание будет сосредоточено на службах, наиболее востребованных локальными клиентами:

- Сетевая файловая система NFS (Network File System)
- Демон печати LPD (Line Printer Daemon) и служба печати LP (Line Printer)
- Службы печати и файлов Windows (Samba)
- Сетевая информационная служба NIS (Network Information Service)
- Протокол динамической настройки узлов DHCP (Dynamic Host Configuration Protocol)
- Протокол почтовой службы POP (Post Office Protocol)
- Интернет-протокол доступа к сообщениям IMAP (Internet Message Access Protocol)

Все перечисленные программные пакеты проектировались с прицелом на обслуживание пользователей в пределах организации, но не на внешних пользователей. Основные службы, востребованные как внешними пользователями, так и пользователями локальной сети, – это электронная почта, веб-служба (сервер Apache) и служба имен. Их описанию посвящены отдельные главы.

Разговор о локальных сетевых службах мы начнем с NFS – сервера, обеспечивающего совместный доступ к файлам в сетях Unix.

Сетевая файловая система (NFS)

Сетевая файловая система NFS (Network File System) позволяет организовать совместный доступ пользователей к каталогам и файлам по сети. Авторство системы принадлежит компании Sun Microsystems, но сегодня она поддерживается практически всеми операционными системами Unix, а также многими другими ОС. Посредством NFS пользователи и программы могут обращаться к файлам, хранящимся на удаленных системах, так, как будто это локальные файлы. В идеальном случае пользователь NFS не знает и не испытывает необходимости знать, где на самом деле хранятся файлы.

NFS имеет ряд преимуществ:

- Сокращает потребность в локальном дисковом пространстве – единственная копия каталога, доступного всем пользователям сети, может храниться на сервере.
- Упрощает реализацию централизованной поддержки – обновленные файлы немедленно становятся доступны всей сети.
- Позволяет пользователям использовать для работы с удаленными файлами знакомые команды Unix и не требует изучения новых команд. NFS не принуждает пользователя обращаться к `ftp` или `rcp`, если требуется скопировать файл; `cp` оказывается более чем достаточно.

Существует две стороны NFS: клиент и сервер. Клиент – это система, использующая удаленные каталоги в качестве каталогов локальной файловой системы. Сервер – это система, которая делает такие каталоги доступными для клиентов. Включение удаленного каталога в локальную файловую систему (функция клиента) называется *монтированием* каталога. Предоставление удаленного доступа к каталогам (функция сервера) называется *организацией совместного доступа* (*sharing*) или *экспортированием* (*exporting*) каталога.[1] Достаточно часто система выступает в качестве сервера и клиента одновременно. В этом разделе мы изучим настройку системы на экспортирование и монтирование каталогов посредством NFS.

Если в вашем ведении находится сервер NFS крупной сетевой площадки, следует тщательно планировать создание среды NFS. В этой главе описана настройка NFS для клиента и сервера, но могут потребоваться и дополнительные сведения о проектировании оптимальной среды NFS. Подробная информация по теме приводится в книге Хэла Стерна (Hal Stern) «Managing NFS and NIS» (O'Reilly & Associates).

Демоны NFS

Работа сетевой файловой системы (Network File System) основана на ряде демонов, выполняющих функции клиента или сервера. Прежде чем перейти к

[1] Термин «организация совместного доступа» (*sharing*) используется в Solaris, в большинстве других систем – «экспортирование» (*exporting*).

настройке NFS, рассмотрим функциональность и назначение демонов NFS в системе Solaris 8:

`nfsd [nservers]`

Демон NFS `nfsd` работает на серверах NFS. Этот демон обслуживает запросы клиентов NFS. Параметр *nservers* определяет число запускаемых демонов.

`mountd`

Демон монтирования NFS `mountd` обрабатывает запросы клиентов на монтирование каталогов. Демон `mountd` работает на серверах NFS.

`nfslogd`

Демон журналов NFS `nfslogd` фиксирует активность для экспортированных файловых систем. Демон журналов работает на серверах NFS.

`rquotad`

Сервер удаленных квот `rquotad` предоставляет информацию о квотах пользователей в удаленных файловых системах. Информация отображается по команде `quota`. Демон квот может работать как на серверах, так и на клиентах.

`lockd`

Демон блокировки `lockd` обрабатывает запросы на блокировку файлов. Демон блокировки работает как на серверах, так и на клиентах. Клиенты запрашивают блокировку файлов, а серверы ее разрешают.

`statd`

Демон наблюдения за сетевым состоянием `statd` необходим для работы `lockd`. В частности, он позволяет корректно отменять блокировку после сбоя. Демон `statd` работает как на серверах, так и на клиентах.

В системе Solaris 8 демоны, необходимые для работы NFS, располагаются в каталоге */usr/lib/nfs*. Большая часть этих демонов запускается при загрузке системы из двух сценариев, *nfs.client* и *nfs.server*, расположенных в каталоге */etc/init.d*. Сценарий *nfs.client* запускает программы *statd* и *lockd*.[1] На серверах NFS работают эти два демона плюс демон сервера NFS (`nfsd`), демон журналов NFS (`nfslogd`) и демон монтирования (`mountd`). В системе Solaris сценарий *nfs.server* запускает `mountd`, `nfslogd`, а также 16 экземпляров `nfsd`. Системы Solaris обычно не запускают демон `rquotad` в процессе загрузки. Вместо этого `rquotad` запускается демоном `inetd`, как можно видеть из следующей команды `grep`:

```
$ grep rquotad /etc/inetd.conf
rquotad/1   tli   rpc/datagram_v  wait root /usr/lib/nfs/rquotad  rquotad
```

[1] Как вариант имена демонов может предварять префикс *rpc*. Например, в системе Slackware Linux демон NFS называется *rpc.nfsd*. Сверьтесь с документацией по своей системе.

В каждой системе существуют свои способы запуска этих демонов. Если отдельные демоны не запускаются, проверьте правильность загрузочных сценариев и файла *inetd.conf*.

Совместный доступ к файловым системам Unix

При настройке сервера необходимо, прежде всего, определить, для каких файловых систем будет организован совместный доступ и какие ограничения будут действовать для этих файловых систем. Клиентам следует предоставлять только те файловые системы, которые могут быть полезны. Прежде чем открывать доступ к файловой системе, подумайте о том, каким целям это послужит. Вот некоторые из распространенных причин организации совместного доступа к файловым системам:

- Предоставить дисковое пространство бездисковым клиентам
- Избежать ненужного дублирования данных на многих системах
- Обеспечить централизованную поддержку программ и данных
- Организовать совместный доступ к данным для группы пользователей

Для всех выбранных файловых систем необходимо провести настройку при помощи соответствующих команд. В следующем разделе данный процесс описан для системы Solaris. Для Linux процесс значительно отличается и будет описан позже. Обращайтесь к документации по своей системе за точными сведениями о том, как организован совместный доступ к файлам по NFS.

Команда share

В системах Solaris каталоги экспортируются при помощи команды `share`. Упрощенный синтаксис этой команды выглядит следующим образом:

```
share -F nfs [-o параметры] путь
```

где *путь* – путь к каталогу на сервере, экспортируемому в пользу клиентов, а *параметры* управляют доступом клиентов к данному каталогу. Доступные параметры:

`rw`

Параметр `rw` разрешает чтение и запись для файловой системы. Может иметь вид `rw=`*список_доступа*, что позволяет указывать системы, которым разрешен такой доступ. В последнем случае доступ к файловой системе предоставляется только системам из списка доступа. Если список доступа отсутствует, все узлы получают полномочия чтения/записи в данной файловой системе.

`ro`

Данный параметр ограничивает доступ только чтением. Он также позволяет указать список доступа – в формате `ro=`*список_доступа*. В последнем случае доступ предоставляется только системам из списка, и этот доступ разрешает только чтение данных. В отсутствие списка доступа все узлы

получают в искомой файловой системе полномочия только чтения – что является умолчанием для команды `share`.

`aclok`

Предоставляет полный доступ всем клиентам. Может стать причиной несанкционированного проникновения в систему. Этот параметр описан в документации системы Solaris, но его не следует использовать ни при каких обстоятельствах. Он нужен только для обратной совместимости с более не существующей версией NFS.

`anon=`*`uid`*

Указывает, какой идентификатор пользователя (UID) следует использовать, если пользователь не представил корректный идентификатор.

`index=`*`file`*

Предписывает NFS использовать файл индекса в стиле индекса веб-сервера – вместо стандартного списка файлов каталога для данной файловой системы.

`log[=`*`tag`*`]`

Включает регистрацию событий в журнале. Если присутствует дополнительный тег (*`tag`*), он должен совпадать с тегом, определенным в файле */etc/nfs/nfslog.conf*.

`nosub`

Запрещает клиентам монтирование подкаталогов. По умолчанию монтирование подкаталогов разрешено.

`nosuid`

Запрещает клиентам создавать в этой файловой системе файлы *setuid* и *setgid*. По умолчанию создание таких файлов разрешено.

`public`

Предписывает использовать открытый файловый дескриптор для этой файловой системы.

`root=`*`список_доступа`*

Разрешает root-доступ к файловой системе пользователям root с указанных систем.

`sec=`*`type`*

Определяет вид аутентификации, прохождение которой требуется для доступа к данной файловой системе. Вид (*`type`*) – это список режимов безопасности NFS, элементы которого разделяются двоеточиями. Чтобы получить доступ, клиент должен поддерживать по меньшей мере один режим безопасности из перечисленных в списке. Существуют следующие режимы:

`sys`

Управление доступом к файловой системе осуществляется на основе идентификаторов пользователей и групп, передаваемых открытым

текстом. Таким же образом работают традиционные права доступа в Unix-системах – на основе идентификаторов UID и GID. В NFS UID и GID передаются по сети, и сервер вынужден доверять удаленному источнику, от которого поступили данные.

`dh`

Для аутентификации используется криптографический метод с открытым ключом Диффи-Хеллмана.

`krb4`

Проверка подлинности по протоколу Kerberos версии 4.

`none`

Не использовать аутентификацию. В этом случае все пользователи получают доступ к файловой системе в качестве пользователя *nobody*.

`window=`*`seconds`*

Указывает максимальное время действия данных идентификации для методов `dh` и `krb4`. Сервер NFS отказывает в доступе клиентам, предоставляющим данные идентификации с большим временем действия. Число секунд (*seconds*) по умолчанию равно 30 000.

Некоторые из параметров принимают в качестве аргумента список доступа, элементы которого разделяются двоеточиями. Каждый элемент обозначает либо конкретный компьютер посредством имени узла или IP-адреса, либо домен, сеть или сетевую группу NIS, к которым принадлежат узлы. Синтаксис элементов списка доступа следующий:

имя узла

Произвольное имя узла, которое преобразуется в IP-адрес. Допустимы абсолютные имена, равно как и локальные имена узлов, но в любом случае должна существовать возможность превратить имя в адрес. Если имя упоминается в локальной таблице узлов, можно использовать краткую форму. Если разрешение имен выполняется при помощи DNS, следует указывать абсолютное доменное имя. Однако не следует использовать абсолютные имена, если ваша система не использует DNS, а полагается в работе исключительно на NIS.

адрес

IP-адрес в десятичной записи через точку.

сетевая группа (netgroup)

Если указано имя сетевой группы NIS, параметр действует для всех систем этой сетевой группы. Имена сетевых групп похожи на локальные имена узлов, что вносит некоторую путаницу. Имена сетевых групп следует использовать, только если применяется NIS.

.домен

Доменное имя позволяет применить параметр ко всем системам домена. Имена доменов предваряются точкой (`.`). Таким образом, указание имени *.wrotethebook.com* распространяет действие параметра на все системы до-

мена *wrotethebook.com*. Доменные имена следует использовать, только если сервер взаимодействует с системой DNS.

`@network[/prefix]`

Адрес сети позволяет распространить действие параметра на все системы определенной сети. Адреса сетей предваряются символом `@`. Необязательный префикс сети (*prefix*) позволяет четко указать маску для сети.

Сочетание параметров `rw` и `ro` может использоваться для предоставления различного уровня доступа различным клиентам. Например:

```
share -F nfs -o rw=crab:horseshoe ro  /usr/man
share -F nfs -o rw=rodent:crab:horseshoe:jerboas  /export/home/research
```

Первая команда `share` разрешает чтение и запись узлам *crab* и *rodent* и предоставляет доступ только для чтения всем прочим. С другой стороны, вторая команда `share` разрешает чтение/запись узлам *rodent*, *crab*, *horseshoe* и *jerboas*, но запрещает доступ любого рода всем остальным клиентам.

Команда `share` утрачивает силу при перезагрузке системы. Поместите команды `share` в сценарий */etc/dfs/dfstab* – это позволит клиентам получать доступ к соответствующим файловым системам и после перезагрузки. Вот пример файла *dfstab*, содержащего две приведенные выше команды:

```
% cat /etc/dfs/dfstab
#   place share(1M) commands here for automatic execution
#   on entering init state 3.
#
#   share [-F fstype] [ -o options] [-d "<text>"] <pathname> [resource]
#   .e.g.,
#   share  -F nfs  -o rw=engineering  -d "home dirs"  /export/home2
share -F nfs -o rw=crab:horseshoe ro  /usr/man
share -F nfs -o rw=rodent:crab:horseshoe:jerboas  /export/home/research
```

Команда `share`, файл *dfstab* и даже собственно термин «share» присущи только системе Solaris. В большинстве систем Unix, если речь заходит о предоставлении клиентам доступа к файлам по NFS, то для файлов выполняется *экспортирование*, а не *организация совместного доступа*. Более того, в этих системах не используется команда `share` и файл *dfstab* – режимы доступа к файловым системам устанавливаются в файле */etc/exports*. Примером такой системы является Linux.

Файл /etc/exports

Файл */etc/exports* содержит настройки сервера NFS для систем Linux. Он определяет, какие файлы и каталоги экспортируются и в каких режимах доступа. Например, файл */etc/exports* может содержать следующие записи:

```
/usr/man        crab(rw) horseshoe(rw)  (ro)
/usr/local      (ro)
/home/research  rodent(rw) crab(rw) horseshoe(rw) jerboas(rw)
```

Здесь сказано, что:

- */usr/man* доступен для монтирования любому клиенту, но запись в каталог разрешена только узлам *crab* и *horseshoe*. Прочим клиентам разрешено только чтение файлов.
- */usr/local* может монтироваться любым клиентом, разрешено только чтение файлов.
- */home/research* может монтироваться только узлами *rodent*, *crab*, *horseshoe* и *jerboas*. Этим узлам разрешены чтение и запись.

Параметры каждой из записей файла */etc/exports* определяют режимы доступа. Записи подчиняются следующему формату:

```
каталог [узел(параметр)]...
```

каталог указывает экспортируемый каталог или файл. *узел* – имя узла или клиента, которому разрешен доступ к экспортированному каталогу. *параметр* определяет режим доступа для клиента.

В приведенном файле */etc/exports* в качестве узла фигурирует имя отдельного клиента. В одном случае имя узла не указано. Если указано имя одного узла, доступ получает отдельный клиент. Если узел не указан, каталог экспортируется для всех клиентов. Подобно Solaris, Linux позволяет использовать домены, сети и сетевые группы, хотя синтаксис немного отличается. Допустимые значения для поля `узел`:

- Имена отдельных узлов, такие как *crab* или *crab.wrotethebook.com*.
- Маски доменов. Например, маска **wrotethebook.com* включает все узлы домена *wrotethebook.com*.
- Пары вида адрес IP/маска. К примеру, 172.16.12.0/255.255.255.0 включает все узлы, адреса которых начинаются с 172.16.12.
- Сетевые группы, такие как *@group1*.

Обратите внимание, что в Linux имена доменов начинаются с символа звездочки (*), а не с точки (.), как в Solaris. Кроме того, символ @ предваряет имя сетевой группы, а не адрес сети, как в Solaris.

Параметры, использованные в примере файла */etc/exports*:

`ro`

Режим только для чтения запрещает клиентам NFS запись в каталог. Попытки записи в такой каталог завершаются неудачно сообщением «Read-only filesystem» (Файловая система доступна только для чтения) или «Permission denied» (Недостаточно полномочий). Если указан параметр `ro`, но отсутствует имя узла клиента, все клиенты получают доступ только для чтения.

`rw`

Режим чтения/записи разрешает клиентам чтение и запись в данный каталог. Если не указано имя узла, всем клиентам разрешается доступ в режиме чтение/запись. Если указано имя узла, только этот узел получает полномочия чтения/записи.

Итак, отдельные узлы получают доступ в режиме чтения/записи к некоторым из каталогов, однако доступ конкретных пользователей этих узлов регламентируется стандартными для Unix правами уровня пользователя, группы и глобального доступа, вычисляемыми на основе идентификатора пользователя (UID) и группы (GID). NFS считает, что удаленный узел выполнил проверку подлинности пользователей и назначил им корректные идентификаторы UID и GID. Экспортирование файлов дает пользователям системы-клиента такой же доступ к этим файлам, как если бы они регистрировались напрямую на сервере. Разумеется, предполагается, что клиент и сервер назначили одинаковые идентификаторы UID и GID каждому из пользователей. Но это далеко не всегда так. Если клиент и сервер назначили один и тот же идентификатор UID определенному пользователю, например идентификатор пользователя Craig равен 501 на обеих системах, тогда обе системы корректно определят пользователя Craig, и он в результате получит доступ к своим файлам. Если же клиентская система назначила пользователю Craig идентификатор UID, равный 501, а сервер назначил тот же UID пользователю Michael, сервер предоставит пользователю Craig доступ к файлам пользователя Michael – так, словно они принадлежат пользователю Craig. В NFS существует ряд механизмов, позволяющих справляться с проблемами, возникающими вследствие такого несовпадения идентификаторов UID и GID.

Первая из очевидных проблем – работа с учетной записью администратора. Маловероятно, что администратор разрешит доступ на сервер пользователям, обладающим root-полномочиями на клиентских системах. По умолчанию NFS блокирует такой доступ параметром `root_squash`, который выполняет отображение UID и GID пользователя *root* в соответствующие идентификаторы пользователя *nobody*. Таким образом, пользователь *root* с системы-клиента может получить только общие полномочия на сервере. Чтобы изменить это поведение, можно воспользоваться установкой `no_root_squash`, однако `no_root_squash` делает сервер потенциально уязвимым.

Отображение прочих идентификаторов UID и GID в идентификаторы пользователя *nobody* достигается при помощи параметров `squash_uids`, `squash_gids` и `all_squash`. `all_squash` выполняет такое отображение для всех пользователей клиентской системы. `squash_uids` и `squash_gids` выполняют отображение конкретных идентификаторов UID и GID. Пример:

```
/pub            (ro,all_squash)
/usr/local/pub (squash_uids=0-50,squash_gids=0-50)
```

Первая запись экспортирует каталог */pub* и делает его доступным только для чтения всем клиентам. Все пользователи клиентских систем получают общие полномочия, присущие пользователю *nobody*, то есть могут читать только те файлы, которые доступны для чтения всем пользователям, а не только владельцу или группе.

Вторая запись экспортирует каталог */usr/local/pub* и делает его доступным для чтения/записи (режим по умолчанию) всем клиентам. Параметры `squash_uid` и `squash_gid` в данном примере показывают, что для некоторых пара-

метров допустимо указание диапазонов идентификаторов UID и GID.[1] Параметры позволяют указывать отдельные идентификаторы UID или GID, но часто бывает удобно охватить диапазон значений одной командой. В этом примере мы блокируем доступ к каталогу пользователей, идентификаторы UID и GID которых меньше либо равны 50. Столь низкие значения обычно присваиваются служебным учетным записям. К примеру, на нашей системе Linux UID 10 назначен записи *uucp*. Попытка записать файл от пользователя *uucp* приведет к записи файла от имени пользователя *nobody*. Таким образом, пользователь *uucp* сможет выполнить запись в каталог */usr/local/pub*, только если в этот каталог разрешена запись всем пользователям.

Кроме того, существует возможность связать всех пользователей клиентской системы с конкретным идентификатором пользователя или группы. Задача решается при помощи параметров `anonuid` и `anongid`. Они наиболее полезны, если доступ к клиентской системе имеет только один пользователь и этому пользователю не назначаются идентификаторы. Например, такова ситуация с персональным компьютером под управлением Microsoft Windows, на котором работает NFS. У персонального компьютера обычно только один пользователь, и к нему неприменимы понятия идентификаторов UID и GID. Чтобы связать пользователя персонального компьютера с корректными идентификаторами пользователя и группы, создайте подобную строку в файле */etc/exports*:

```
/home/alana  giant(all_squash,anonuid=1001,anongid=1001)
```

В данном примере имя узла компьютера Аланы – *giant*. Запись разрешает клиенту доступ к каталогу */home/alana* в режиме чтения/записи. Параметр `all_squash` преобразует все запросы от клиента, подставляя конкретный идентификатор UID, однако на этот раз вместо идентификаторов пользователя *nobody* подставляются идентификаторы, указанные в параметрах `anonuid` и `anongid`. Разумеется, чтобы прием сработал, сочетанию 1001:1001 должна соответствовать пара идентификаторов UID/GID, назначенная пользователю *alana* в файле */etc/passwd*.

Единичного отображения достаточно для персонального компьютера, однако его явно не хватит для полноценной работы с Unix-клиентом. Клиенты Unix назначают своим пользователям идентификаторы UID и GID. Если тем же пользователям на сервере NFS назначены другие идентификаторы, начинаются сложности. Воспользуйтесь параметром *map_static*, чтобы указать файл, в котором хранятся отображения идентификаторов UID и GID для конкретного клиента. Пример:

```
/export/oscon oscon(map_static=/etc/nfs/oscon.map)
```

[1] Из восьми параметров, описанных в этом разделе, три позволяют выполнять отображение диапазонов идентификаторов UID и GID, а именно `squash_uid`, `squash_gid` и `map_static`. Эти три параметра недоступны в NFS уровня ядра (kernel-level NFS, `knfsd`), применяемой в некоторых Linux-системах. Отображение в `knfsd` должно выполняться при помощи других параметров.

Данная запись экспортирует каталог */export/oscon* для клиента *oscon* в режиме чтения/записи. Параметр `map_static` указывает, что в файле */etc/nfs/oscon.map* на сервере хранятся отображения идентификаторов, позволяющие пользователям узла *oscon* корректно работать с сервером. Файл *oscon.map* может содержать записи следующего характера:

```
# UID/GID mapping for client oscon
# remote    local    comment
uid 0-50    -    #squash these
gid 0-50    -    #squash these
uid 100-200    1000    #map 100-200 to 1000-1100
gid 100-200    1000    #map 100-200 to 1000-1100
uid 501    2001    #map individual user
gid 501    2001    #map individual user
```

Первые две строки преобразуют идентификаторы UID и GID от 0 до 50 в идентификаторы пользователя nobody. Следующие две строки преобразуют все идентификаторы клиентов из диапазона от 100 до 200 в соответствующие числа из диапазона от 1000 до 1100 на сервере. Иначе говоря, пользователь с идентификатором 105 на клиенте получает идентификатор 1005 на сервере. Записи такого рода встречаются чаще всего. В большинстве систем присвоение существующих идентификаторов UID и GID происходило последовательно. Зачастую различные системы назначают пользователям идентификаторы, начиная с 101 последовательно, но не согласованно. Данная запись выполняет отображение для пользователей узла *oscon*, назначая им идентификаторы начиная с 1000. В другом файле может выполняться отображение идентификаторов с 100 по 200 уже для другого клиента – в идентификаторы сервера, начинающиеся с 2000. В третьем файле идентификаторы третьего клиента могут получить уже порог 3000. Таким образом, существует возможность согласовывать на сервере идентификаторы в случаях, когда никакой согласованности не существует. Последние две строки выполняют отображение идентификаторов UID и GID отдельного пользователя. Такие записи требуются реже, но все же встречаются время от времени.

Команда exportfs

Создав необходимые записи для каталогов в файле */etc/exports*, выполните команду `exportfs`, чтобы обработать файл экспорта и создать файл */var/lib/nfs/xtab*. Файл *xtab* содержит информацию об экспортированных каталогах – именно из этого файла извлекает сведения `mountd`, обрабатывая запросы клиентов на монтирование. Чтобы обработать все записи файла */etc/exports*, выполните `exportfs` с ключом командной строки `-a`:

```
# exportfs -a
```

Данная команда экспортирует все каталоги, упомянутые в файле *exports*, и обычно выполняется в процессе загрузки из загрузочного сценария. Чтобы привести в исполнение изменения, внесенные в файл */etc/exports*, не перезагружая систему, воспользуйтесь ключом `-r`:

```
# exportfs -r
```

Ключ `-r` синхронизирует содержимое файлов *exports* и *xtab*. Новые элементы файла *exports* добавляются в файл *xtab*, а удаленные элементы удаляются из файла *xtab*.

Команда `exportfs` позволяет экспортировать каталог, не упомянутый в файле */etc/exports*. Например, чтобы временно экспортировать каталог */usr/local* для клиента *fox* в режиме чтения/записи, выполните такую команду:

```
# exportfs fox:/usr/local -o rw
```

Когда клиент завершил работу с временно экспортированной файловой системой, каталог удаляется из списка доступных при помощи ключа `-u`, как показано ниже:

```
# exportfs -u fox:/usr/local
```

Сочетание ключей `-u` и `-a` позволяет отменить экспорт всех каталогов, не завершая работу демона NFS:

```
# exportfs -ua
```

Как только сервер организовал совместный доступ или экспортировал каталоги, клиенты могут начинать монтировать и использовать файловые системы. В следующем разделе описана настройка клиента NFS.

Монтирование удаленных файловых систем

Принятие решения о том, какие каталоги NFS монтировать, требует наличия некоторых начальных сведений. Необходимо знать, какие серверы NFS существуют в вашей сети и какие каталоги они предоставляют для монтирования. Каталог не может быть смонтирован, если он не экспортирован сервером.

Верным источником таких сведений станет ваш системный администратор. Администратор может сообщить, какие системы предоставляют доступ к службе NFS, какие каталоги экспортируют и что содержат эти каталоги. Администратору сервера NFS следует предварительно создать сводку такой информации для пользователей. В главе 4 приводятся советы по планированию и распространению информации о сети.

Системы Solaris и Linux позволяют получить сведения о каталогах еще и напрямую от сервера – при помощи команды `showmount`. Как правило, серверы NFS работают на тех же выделенных системах, которые предоставляют доступ к другим службам, таким как DNS и электронная почта. Выберите сервер, который подходит под данное описание, и выполните для него запрос `showmount -e` *имя_узла*. В ответ на эту команду сервер перечисляет экспортированные каталоги и условия доступа к ним.

Например, запрос `showmount -e`, адресованный узлу *jerboas*, дает следующий результат:

```
% showmount -e jerboas
export list for jerboas:
/usr/man          (everyone)
```

```
/home/research      rodent,crab,limulus,horseshoe
/usr/local          (everyone)
```

Данный список отражает каталоги NFS, экспортированные узлом *jerboas*, а также перечисляет клиентов, которым разрешен доступ к каждому конкретному каталогу. Исходя из этих сведений администратор узла *rodent* может смонтировать любой из каталогов, экспортированных сервером *jerboas*. Наш воображаемый администратор решил:

1. Смонтировать каталог */usr/man* с *jerboas* и избавиться от необходимости локального хранения страниц руководства.
2. Смонтировать */home/research*, чтобы облегчить задачу обмена файлами с другими системами исследовательской группы.
3. Смонтировать каталог */usr/local*, чтобы пользоваться приложениями с централизованным сопровождением.

Такой выбор диктуется наиболее распространенными соображениями по монтированию каталогов NFS:

- Экономия дискового пространства
- Общий доступ к файлам для различных систем
- Централизованное сопровождение общих файлов

Масштабы использования NFS – дело вкуса. Одним нравится персональный контроль при локальном хранении файлов, а другим – удобства, предлагаемые NFS. Для вашей сетевой площадки могут существовать специальные правила использования NFS, регламентирующие монтирование определенных каталогов и централизованное управление определенными файлами. Точные сведения о том, как используется NFS, всегда можно получить от системного администратора вашей сетевой площадки.

Команда mount

Чтобы получить возможность использовать каталог NFS, клиент должен его смонтировать. «Монтирование» каталога означает его закрепление в иерархии файловой системы клиента. Монтироваться могут только каталоги, экспортируемые серверами NFS, однако из этих каталогов можно отдельно монтировать любые самостоятельные элементы, такие как подкаталоги и файлы.

Монтирование каталогов NFS выполняется посредством команды `mount`. Общий вид команды:

```
mount имя узла:удаленный каталог локальный каталог
```

Имя узла определяет сервер NFS, *удаленный каталог* – каталог на сервере либо его фрагмент. Команда `mount` закрепляет удаленный каталог в файловой системе клиента, используя имя *локальный каталог*. Локальный каталог клиента, известный в качестве точки монтирования, должен существовать на момент выполнения команды `mount`. После завершения монтирования файлы, расположенные в удаленном каталоге, становятся доступными в локальном каталоге наравне с прочими локальными файлами.

Например, предположим, что *jerboas.wrotethebook.com* является сервером NFS и позволяет клиентам работать с каталогами, упомянутыми в предшествующих разделах. Далее, предположим, что администратор узла *rodent* желает получить доступ к каталогу */home/research*. Он просто создает локальный каталог */home/research* и монтирует удаленный каталог */home/research*, экспортированный сервером *jerboas*, в этой новой точке:

```
# mkdir /home/research
# mount jerboas:/home/research /home/research
```

В данном примере локальная система способна смонтировать файловую систему NFS потому, что имени удаленного каталога предшествует имя узла, а для данного клиента сетевой файловой системой по умолчанию является NFS. NFS – это наиболее распространенная сетевая файловая система, используемая по умолчанию. Если ваша система-клиент по умолчанию работает с другой сетевой файловой системой, следует явно упомянуть NFS в командной строке `mount`. В системе Solaris 8 для этой цели служит ключ `-F`:

```
# mount -F nfs jerboas:/home/research /home/research
```

В системе Linux задачу решает ключ `-t`:

```
# mount -t nfs jerboas:/home/research /home/research
```

После монтирования удаленный каталог остается закрепленным в локальной файловой системе, пока не будет явным образом размонтирован либо пока не произойдет перезагрузка локальной системы. Чтобы размонтировать каталог, воспользуйтесь командой `umount`. В командной строке `umount` укажите локальное или удаленное имя каталога. Например, администратор узла *rodent* может размонтировать удаленную файловую систему *jerboas:/home/research* и освободить таким образом локальную точку монтирования */home/research* командой

```
# umount /home/research
```

либо

```
# umount jerboas:/home/research
```

При перезагрузке размонтируются все каталоги NFS. Часто существует необходимость работать с одними и теми же файловыми системами даже после перезагрузки, поэтому в Unix существует система автоматического повторного монтирования после перезагрузки.

Файлы vfstab и fstab

Для повторного монтирования файловых систем всех типов, включая NFS, в Unix используется информация из специальной таблицы. Следует очень осторожно вносить изменения в эту таблицу, поскольку от нее во многом зависит доступ пользователей к программам и файлам. В различных вариантах Unix используется один из двух форматов. В системах Linux и BSD при-

меняется файл */etc/fstab*, а в Solaris, точнее в нашей подопытной системе типа System V, – файл */etc/vfstab*.

Формат записей NFS в файле Solaris *vfstab* следующий:

```
файловая система - точка монтирования nfs - yes параметры
```

Поля записей расположены в строго определенном порядке и разделяются пробелами. Поля, не выделенные курсивом (оба дефиса, слова `nfs` и `yes`), являются ключевыми словами и должны фигурировать в записях точно в таком виде. *файловая система* – это имя каталога на сервере, *точка монтирования* – путь к локальной точке монтирования, а о *параметрах* монтирования мы поговорим ниже. Пример записи NFS файла *vfstab*:

```
jerboas:/home/research  -  /home/research  nfs  -  yes  rw,soft
```

Данная запись монтирует файловую систему NFS *jerboas:/home/research* в локальной точке монтирования */home/research*. Файловая система монтируется с параметрами `rw` и `soft`. Мы уже обсуждали распространенные параметры режимов `rw` (read/write, чтение/запись) и `ro` (read-only, только для чтения), однако параметров NFS существует гораздо больше. В системах Solaris доступны следующие параметры монтирования NFS:

`remount`

Если файловая система уже смонтирована в режиме только для чтения, перемонтировать ее в режиме чтения/записи.

`soft`

Если сервер не отвечает, сообщить об ошибке и не повторять попытку монтирования.

`timeo=`*n*

Определяет длительность интервала ожидания, по истечении которого выдается сообщение об ошибке (в секундах).

`hard`

Если сервер не отвечает, повторять попытки монтирования, пока не будет получен ответ. Режим по умолчанию.

`bg`

Повторять попытки в фоновом режиме, чтобы не блокировать процесс загрузки системы.

`fg`

Повторять попытки в приоритетном режиме. Данный параметр может заблокировать процесс загрузки системы повторениями попыток монтирования. По этой причине параметр `fg` используется преимущественно при отладке.

`intr`

Разрешает принудительное завершение процесса, ожидающего ответа от сервера, прерыванием с клавиатуры. Жестко смонтированные файловые

системы могут приводить к «зависанию» процессов, поскольку клиент бесконечно повторяет попытки монтирования, независимо от работоспособности сервера. Параметр `intr` принимается по умолчанию.

`nointr`

Запрещает прерывания с клавиатуры. Как правило, это плохая идея.

`nosuid`

Запрещает запуск исполняемых файлов из смонтированных файловых систем в режиме `setuid`. Повышает защищенность, но может ограничивать полезность приложений.

`acdirmax=`*n*

Ограничивает продолжительность кэширования атрибутов каталогов временем в *n* секунд. По умолчанию значения хранятся не более 60 секунд. Частое повторение запросов по атрибутам файловой системы – одна из основных причин роста трафика NFS. Кэширование этой информации помогает сократить объем передаваемых данных.

`acdirmin=`*n*

Устанавливает минимальное время кэширования атрибутов каталога. По умолчанию 30 секунд.

`acregmax=`*n*

Устанавливает максимальное время кэширования атрибутов файлов. По умолчанию 60 секунд.

`acregmin=`*n*

Устанавливает минимальное время кэширования атрибутов файлов. По умолчанию 3 секунды.

`actimeo=`*n*

Устанавливает общее значение для параметров *acdirmax*, *acdirmin*, *acregmax* и *acregmin*.

`grpid`

Предписывает использовать идентификатор группы родительского каталога при создании файлов. В отсутствие данного параметра используется действующий идентификатор группы вызывающего процесса.

`noac`

Запрещает кэшировать информацию. По умолчанию кэширование включено и может включаться принудительно при помощи параметра `ac`.

`port=`*n*

Указывает номер IP-порта сервера.

`posix`

Предписывает работать с файловой системой по стандарту POSIX. POSIX – это широкоплановый стандарт взаимодействия систем Unix, включающий многочисленные стандарты на файловые системы. В частности, определяет максимальную длину имен файлов и методы блокировки файлов.

`proto=`*`protocol`*

Позволяет указать транспортный протокол для NFS.

`public`

Предписывает использовать открытый дескриптор файла при подключении к серверу NFS.

`quota`

Предписывает использовать ограничительные квоты пользователей для данной файловой системы.

`noquota`

Запрещает использовать ограничительные квоты пользователей для данной файловой системы.

`retrans=`*`n`*

Определяет число повторных попыток передачи при использовании транспортного протокола, не предусматривающего создание соединений.

`retry=`*`n`*

Определяет число повторных попыток монтирования. По умолчанию попытка повторяется 10 000 раз.

`rsize=`*`n`*

Определяет размер буфера чтения в байтах. Для NFS версии 3 размер по умолчанию равен 32 768 байтам.

`sec=`*`type`*

Указывает тип защиты для транзакций NFS. Значения типа для команды `mount` в Solaris 8 идентичны описанным для команды `share`: `sys`, `dh`, `krb4` и `none`.

`wsize=`*`n`*

Определяет размер буфера записи в байтах. Для NFS версии 3 размер по умолчанию равен 32 768 байтам.

`vers=`*`version`*

Указывает версию NFS для данной операции монтирования. По умолчанию система автоматически выбирает наиболее современную версию NFS, реализованную как клиентом, так и сервером.

В Solaris файловые системы, определенные в файле *vfstab*, монтируются командой `mountall`, выполняемой из загрузочного сценария. В системе Linux загрузочный файл содержит команду `mount` с ключом `-a`, который предписывает смонтировать все файловые системы, обозначенные в *fstab*.[1] Формат записей NFS в файле */etc/fstab* следующий:

файловая система точка монтирования `nfs` *параметры*

[1] В Red Hat Linux используется специальный сценарий, */etc/init.d/netfs*, который отвечает только за монтирование разнообразных сетевых файловых систем, включая NFS.

Порядок следования полей должен соответствовать указанному. Поля разделяются пробелами. Ключевое слово `nfs` является обязательным для файловых систем NFS. *файловая система* – это имя каталога, который необходимо смонтировать. *точка монтирования* – путь к локальной точке монтирования. В качестве параметров могут фигурировать любые параметры монтирования Linux.

Параметры монтирования в Linux в значительной степени повторяют параметры монтирования NFS в Solaris. Параметры `rsize`, `wsize`, `timeo`, `retrans`, `acregmin`, `acregmax`, `acdirmin`, `acdirmax`, `actimeo`, `retry`, `port`, `bg`, `fg`, `soft`, `hard`, `intr`, `nointr`, `ac`, `noac`, а также `posix` являются общими для Linux и Solaris. Кроме того, в Linux существуют:

`mountport=`*n*

Указывает порт для использования `mountd`.

`mounthost=`*name*

Указывает, на каком сервере работает `mountd`.

`mountprog=`*n*

Указывает номер программы RPC для `mountd` на удаленном узле.

`mountvers=`*n*

Указывает номер версии RPC для `mountd` на удаленном узле.

`nfsprog=`*n*

Указывает номер программы RPC для `nfsd` на удаленном узле.

`nfsvers=`*n*

Указывает номер версии RPC для `nfsd` на удаленном узле.

`namlen=`*n*

Указывает, какая максимальная длина имен файлов поддерживается удаленным сервером.

`nocto`

Запрещает получение атрибутов при создании файла. По умолчанию атрибуты копируются. Принудительно такого поведения можно добиться при помощи параметра `cto`.

`tcp`

Предписывает NFS использовать в качестве транспортного протокола TCP.

`udp`

Предписывает NFS использовать в качестве транспортного протокола UDP.

`nolock`

Предотвращает запуск демона `lockd`. По умолчанию демон `lockd` запускается и может запускаться принудительно при помощи параметра `lock`.

Наконец, существует ряд параметров, действующих не только для NFS. Их можно использовать с командой `mount` при монтировании файловых систем

всех типов. В табл. 9.1 перечислены распространенные параметры монтирования для систем Linux.

Таблица 9.1. Распространенные параметры монтирования

Параметр	Назначение
async	Использовать асинхронный файловый ввод/вывод. Операции записи подтверждаются непосредственно после получения информации в целях повышения производительности
auto	Монтировать файловую систему, когда используется ключ -a
dev	Разрешить использование файлов символьных и блочных специальных устройств в файловой системе
exec	Разрешить исполнение файлов из файловой системы
noauto	Не монтировать файловую систему, когда используется ключ -a
nodev	Запретить использование файлов символьных и блочных специальных устройств в файловой системе
noexec	Запретить исполнение файлов из файловой системы
nosuid	Запретить исполнение программ из файловой системы в режимах setuid и setgid
nouser	Разрешить монтирование файловой системы только пользователю root
remount	Перемонтировать файловую систему с новыми параметрами
ro	Монтировать данную файловую систему в режиме только чтения
rw	Монтировать данную файловую систему в режиме чтения/записи
suid	Разрешить исполнение программ в режимах setuid и setgid
sync	Использовать синхронный файловый ввод/вывод. Операции записи подтверждаются только после записи данных на диск в целях повышения надежности
user	Разрешить монтирование файловой системы обычным пользователям
atime	Обновлять отметку времени доступа к узлу *inode* при всех операциях доступа
noatime	Не обновлять отметку времени доступа к узлу *inode*
defaults	Установить параметры rw, suid, dev, exec, auto, nouser и async

Команда grep для файла *fstab* позволяет просмотреть примеры записей NFS.[1]

```
% grep nfs /etc/fstab
jerboas:/usr/spool/mail   /usr/spool/mail   nfs rw     0 0
jerboas:/usr/man          /usr/man          nfs rw     0 0
jerboas:/home/research    /home/research    nfs rw     0 0
```

[1] Мы использовали grep потому, что файл *fstab* содержит и другую информацию, никак не связанную с NFS.

Мы видим, что в файле */etc/fstab* существуют записи для трех файловых систем NFS. Команда загрузочного сценария `mount -a` повторно монтирует эти каталоги при каждой загрузке системы.

Файлы *vfstab* и *fstab* реализуют наиболее распространенный способ монтирования файловых систем в процессе загрузки. Существует и другой механизм автоматического монтирования файловых систем NFS – при необходимости. Речь идет об автомонтировщике.

Автомонтировщик NFS

Автомонтировщик – это механизм, доступный большинству клиентов NFS. Широкое распространение получили две разновидности автомонтировщика: autofs и amd. Реализация autofs (Automounter Filesystem) входит в состав Solaris и Linux, именно ее мы и рассмотрим в этом разделе. Демон amd (Automounter Daemon) доступен во многих вариантах Unix и входит в состав Linux, но не в Solaris. Подробная информация по amd содержится в книге «Администрирование NFS и автомонтировщика в Linux» (Linux NFS and Automounter Administration), автором которой является Эрец Задок (Erez Zadok), текущий руководитель проекта amd. В этом разделе слова «автомонтировщик» и «демон автомонтировщика» относятся к версии autofs, поставляемой в составе Solaris 8.

Файлы настройки автомонтировщика называются *картами*. Для определения файловой системы в автомонтировщике используются три основных вида карт:

Главная карта

Файл настройки, читаемый командой `automount`. В нем перечислены все другие карты, определяющие файловую систему autofs.

Прямая карта

Файл настройки, хранящий сведения о точках монтирования, путях и параметрах файловых систем, подлежащих монтированию демоном `automountd`.

Косвенная карта

Файл настройки, хранящий сведения о путях и «относительных» точках монтирования. Точки монтирования в данном случае определяются относительно пути, указанного в главной карте. Смысл применения косвенных карт станет понятен из приводимых далее примеров.

В системах Solaris запуск демона автомонтировщика (`automountd`) и выполнение команды `automount` происходит в сценарии */etc/init.d/autofs*. Для запуска автомонтировщика служит аргумент сценария `start`, то есть выполняется команда `autofs start`. Аргумент `stop` позволяет завершить работу автомонтировщика. Команда `automount` и демон `automountd` – две самостоятельные программы. Демон `automountd` динамически монтирует файловые системы, когда возникает необходимость. `automount` обрабатывает файл *auto_master*, чтобы определить, какие системы могут монтироваться динамически.

Чтобы воспользоваться автомонтировщиком, необходимо создать файл */etc/auto_master*. Записи файла *auto_master* имеют следующий формат:

```
точка-монтирования     имя-карты        параметры
```

В системе Solaris по умолчанию существует работоспособный файл *auto_master*. Измените этот файл в соответствии с имеющейся конфигурацией. Закомментируйте запись `+auto_master`, если не используется NIS+ или NIS и ваши серверы не отвечают за распространение карты *auto_master*. Кроме того, не обращайте внимания на запись `/xfn`, относящуюся к интегрированным (составным) службам имен. Добавьте запись для собственной прямой карты. В данном примере эта карта называется *auto_direct*. Вот так выглядит файл */etc/auto_master* с нашими исправлениями:

```
# Master map for automounter
#
#+auto_master
#/xfn            -xfn
/net             -hosts          -nosuid
/home            auto_home
/-               auto_direct
```

Символ решетки (#) начинает строки комментариев, в частности строки `+auto_master` и `/xfn`, которые мы закомментировали. Первая рабочая запись в данном файле указывает, что файловые системы, предлагаемые каждым из серверов NFS, перечисленных в файле */etc/hosts*, монтируются автоматически в иерархии каталога */net*. Для каждого сервера в каталоге */net* создается подкаталог – в качестве имени используется имя узла сервера. Предположим, что сервер *jerboas* упоминается в файле *hosts* и экспортирует каталог */usr/local*. Описанная запись *auto_master* автоматически делает этот удаленный каталог доступным на локальном узле под именем */net/jerboas/usr/local*.

Вторая запись автоматически монтирует исходные каталоги, перечисленные в карте */etc/auto_home*, в иерархии каталога */home*. Файл */etc/auto_home*, используемый по умолчанию, поставляется с системой Solaris. Закомментируйте запись `+auto_home` в стандартном файле. Она используется только в случаях, когда применяется NIS+ или NIS и на ваших серверах доступна централизованно обновляемая карта *auto_home*. Добавьте записи исходных каталогов отдельных пользователей либо всех исходных каталогов отдельных серверов. Измененная карта *auto_home* выглядит так:

```
# Home directory map for automounter
#
#+auto_home
craig            crab:/export/home/craig
*                horseshoe:/export/home/&
```

Первая запись монтирует файловую систему */export/home/craig* с узла *crab* в локальной точке */home/craig*. Карта *auto_home* является косвенной, поэтому указанная точка монтирования (*craig*) определяется относительно точки

монтирования */home*, определенной в карте *auto_master*. Вторая запись монтирует все исходные каталоги из файловой системы */export/home* с узла *horseshoe* в «аналогичных» точках монтирования на локальном узле. Предположим, что на узле *horseshoe* существует два исходных каталога, */export/home/daniel* и */export/home/kristin*. Автомонтировщик делает эти каталоги доступными на локальном узле под именами */home/daniel* и */home/kristin*. Звездочка (*) и амперсанд (&) – специальные символы, используемые для достижения описанных целей в картах autofs.

С картой *auto_home* мы закончили. Взглянем снова на карту *auto_master*. Третья и последняя запись файла */etc/auto_master* выглядит так:

```
/-              auto_direct
```

Эту запись мы добавили для прямой карты. Специальная точка монтирования /- говорит о том, что имя, указанное в этой строке, является именем прямой карты. Следовательно, действительные точки монтирования обозначены в файле прямой карты. Файл прямой карты мы назвали */etc/auto_direct*. По умолчанию такого файла нет, так что его придется создать с нуля. Мы создали следующий файл:

```
# Direct map for automounter
#
/home/research  -rw        jerboas:/home/research
/usr/man        -ro,soft   horseshoe,crab,jerboas:/usr/share/man
```

Формат записей файла прямой карты:

```
точка-монтирования     параметры   удаленная файловая система
```

Созданный файл содержит две типичных записи. Первая запись монтирует удаленную файловую систему */home/research* с сервера *jerboas* в локальной точке */home/research*. При монтировании устанавливается режим доступа чтение/запись. Вторая запись монтирует страницы руководства в режиме только чтения с «мягким» интервалом ожидания.[1] Обратите внимание, что для страниц руководства представлен целый список из трех серверов, имена в котором разделены запятыми. Если сервер недоступен либо не ответил в установленный интервал ожидания, клиент обращается к следующему серверу из списка. Это одна из приятных возможностей автомонтировщика.

Автомонтировщик имеет четыре ключевых возможности: карта -hosts, создание масок, автоматическое монтирование, опрос серверов по списку. Карта -hosts делает доступными локальному пользователю все файловые системы, экспортированные всеми серверами, упомянутыми в файле */etc/hosts*. Специальные символы максимально упрощают монтирование целых наборов каталогов с удаленного сервера в каталоги локальной файловой системы с похожими именами. Автоматическое монтирование идеально подходит к этим двум возможностям, поскольку монтируются только те файловые сис-

[1] См. описание параметров монтирования NFS выше по тексту.

темы, которые используются. Возможности -hosts и специальных символов позволяют получить доступ к очень большому числу файловых систем, однако автоматическое монтирование позволяет монтировать только файловые системы, непосредственно участвующие в работе. Последняя возможность – списки серверов – повышает надежность NFS, снимая зависимость от единственного сервера.

Совместный доступ к принтерам Unix

Для организации совместного доступа к файлам в системах Unix используется NFS; для предоставления доступа к службе печати локальным и удаленным пользователям используются различные инструменты – демон `lpd` (*Line Printer Daemon*) и служба `lp` (*Line Printer*). `lpd` используется в системах BSD и в большинстве систем Linux. `lp` используется в системах System V, включая Solaris 8. В следующих подразделах мы изучим оба инструмента.

Демон печати lpd

Демон `lpd` управляет буферной зоной (спулингом) и очередями заданий печати. Он запускается в процессе загрузки системы из загрузочного сценария. Обычно демон по умолчанию запускается в процессе загрузки систем Linux и BSD, так что во многих случаях его вызов не требуется добавлять в сценарии загрузки. В системе Red Hat Linux, например, запуск демона происходит в сценарии */etc/init.d/lpd*.

Файл printcap

При запуске `lpd` обращается к файлу */etc/printcap* в поисках информации о доступных для использования печатающих устройствах. Файл *printcap* определяет принтеры и их характеристики. Создание файла *printcap* – самый жуткий этап настройки Unix-сервера печати. (Не пугайтесь: как мы позже увидим на примере инструмента Solaris `admintool`, в большинстве систем существуют графические интерфейсы, облегчающие настройку принтеров.) Файл *printcap* внушает страх системным администраторам потому, что синтаксический анализатор, выполняющий разбор файла, весьма придирчив, а синтаксис записей – лаконичен и загадочен. Большинства осложнений, связанных с синтаксисом, можно избежать, следуя приводимым ниже правилам:

- Начинайте каждую запись с имени принтера в первой колонке. Пробелов перед первым именем принтера быть не должно. Допустимо использование списков имен, разделенных символом конвейера (|). Одна из записей должна быть связана с принтером по имени *lp*. Если к серверу подключено несколько принтеров, назначайте имя *lp* принтеру «по умолчанию».
- Для разбиения записей на строки используйте символ обратного слэша \ в конце строки и табуляцию в начале строки продолжения. Следите за тем, чтобы после символа \ не было пробелов. Следующим символом обязательно должен быть символ новой строки.

- Каждое поле, за исключением поля имени принтера, начинается и заканчивается двоеточием (:). В записи, разбитой на строки, двоеточие предшествует символу обратного слэша и оно же следует за символом табуляции в строке продолжения.
- Начинайте комментарии символом решетки (#).

Параметры настройки в файле *printcap* определяют характеристики принтера. Эти характеристики в документации *printcap* называются возможностями (capabilities), но в действительности это всего лишь характеристики принтера, которые нужны демону `lpd`, чтобы общаться с печатающим устройством. Параметры обозначаются двухсимвольными именами и обычно имеют значения. Синтаксис параметров немного варьируется в зависимости от типов значений. Параметры бывают трех видов:

Логические

Все логические параметры *printcap* по умолчанию имеют значение «false» (то есть «ложь»). Упоминание логического параметра в записи принтера включает соответствующую функцию. Достаточно просто добавить в запись двухсимвольное имя параметра. Так, `:rs:` включает ограничение доступа для удаленных пользователей.

Численные

Некоторым из параметров назначаются численные значения. Например, `:br#9600:` устанавливает скорость обмена данными для последовательного принтера.

Строковые

Некоторым из параметров назначаются строковые значения. К примеру, `:rp=laser:` определяет имя удаленного принтера.

Взгляд на страницу руководства, посвященную `lpd`, позволяет понять, что параметров *printcap* существует множество. К счастью, большинство из них вам никогда не придется использовать. Чаще встречаются простые определения принтеров и небольшие файлы *printcap*.

К серверу печати обычно напрямую подключен один или два принтера; все прочие принтеры, определенные в файле *printcap*, скорее всего, являются удаленными принтерами. Большинство определений (если не все) клиентского файла *printcap* относятся к удаленным принтерам.

```
#
# Remote LaserWriter
#
lw:\
    :lf=/var/adm/lpd-errs:\
    :lp=:rm=horseshoe:rp=lw:\
    :sd=/var/spool/lpd-lw:
```

В данном примере файла *printcap* принтер *lw* является удаленным. Параметр `lf` указывает файл журнала, в который записываются сообщения о состоянии и ошибках. Удаленная машина, к которой подключен принтер,

определена параметром `:rm=horseshoe:`, а имя удаленного принтера на этой машине – параметром `:rp=lw:`. Один файл журнала может использоваться для нескольких принтеров. Последний параметр, `sd`, определяет каталог буферной зоны печати. Каждому принтеру соответствует отдельный каталог буферной зоны. Для настройки клиента LPD достаточно создать определение удаленного принтера в файле *printcap* клиента.

Совершенно необязательно создавать файл *printcap* с нуля. Самое большее может потребоваться отредактировать файл, чтобы он лучше соответствовал вашим нуждам. Во всех современных системах Unix существуют инструменты настройки принтеров, которые самостоятельно создают основу файла *printcap*. Примером такого инструмента может служить приложение `printconf-gui` в системе Red Hat.

Запустите инструмент настройки принтеров в системе Red Hat 7.2, с рабочего стола Gnome обратившись к элементу Printer Configuration меню System. Когда откроется окно `printconf-gui`, щелкните по кнопке New, чтобы добавить определение принтера в файл *printcap*. Кнопка New вызывает мастер настройки принтера. На первой странице мастера укажите имя принтера и тип очереди печати. Наберите имя принтера, например *lw*, в поле ввода Queue Name. Затем выберите тип очереди (Queue Type). В Red Hat 7.2 существует пять вариантов для типа очереди:

Local Printer (Локальный принтер)

Используйте данный тип для принтера, подключенного напрямую. При выборе типа Local Printer мастер предлагает указать порт, к которому подключен принтер. По умолчанию это порт */dev/lp0*.

Unix Printer (Принтер Unix)

Используйте данный тип для принтера, подключенного к удаленному серверу, если доступ осуществляется по протоколу LPD. При выборе типа Unix Printer мастер предлагает указать имя удаленного сервера и имя принтера на этом сервере.

Windows Printer (Принтер Windows)

Используйте данный тип для принтера, подключенного к удаленному серверу, если доступ осуществляется по протоколу SMB. При выборе типа Windows Printer мастер предлагает указать IP-адрес удаленного сервера, рабочую группу SMB, а также имя удаленного принтера (мастер обозначает его как *имя ресурса*, share name). Кроме того, мастер позволяет указать имя пользователя и пароль, если они требуются для доступа к принтеру. Организация совместного доступа к принтерам по протоколу SMB описана далее в этой главе.

Novell Printer (Принтер Novell)

Используйте данный тип для принтера, подключенного к удаленному серверу, если доступ осуществляется по протоколам NetWare. При выборе типа Novell Printer мастер предлагает указать имя сервера и принтера на сервере, а также необязательное имя пользователя и пароль, если они

требуются для доступа к принтеру. Чтобы работать с принтером Novell, необходимо установить реализацию протоколов NetWare.

JetDirect Printer (Принтер JetDirect)

Используйте данный тип для принтера, подключенного к сети, если доступ осуществляется по протоколу JetDirect. В основном этот протокол используется в принтерах HP со встроенным интерфейсом Ethernet. Такие принтеры подключаются к сети напрямую по кабелю Ethernet, без участия сервера. При выборе типа JetDirect мастер предлагает указать IP-адрес принтера и позволяет ввести номер порта в случае, когда принтер работает не через стандартный порт JetDirect.

Наконец, мастер предлагает выбрать один из сотен драйверов для принтера. В большинстве систем Unix используются стандартные принтеры PostScript. Системы Linux, напротив, чаще создаются на основе PC-компонентов широкого потребления. С персональными компьютерами используется сборная солянка из самых разных принтеров. Мастер позволяет выбрать подходящий драйвер на основе информации о марке и модели принтера. Когда драйвер выбран и настройка завершена, происходит установка нового принтера.

Данный инструмент для Red Hat – просто пример. В новых версиях Red Hat появятся более совершенные инструменты аналогичной функциональности, а в других системах Unix существуют свои подобные приложения. Суть дела не в том, как использовать такой инструмент, а в том, что файлы *printcap* обычно не создаются вручную. Для этой цели используются приложения настройки.

Работа с LPD

После того как принтер настроен, задания печати передаются демону `lpd` при помощи программы `lpr` (*Line Printer Remote*). Программа `lpr` создает управляющий файл и передает его демону `lpd` вместе с файлом для печати. Существует множество различных аргументов командной строки `lpr`, но обычно в команде просто указывается принтер и файл, подлежащий печати:

```
% lpr -Plj ch09
```

Данная команда передает файл *ch09* принтеру *lj*. Принтер может быть локальным или удаленным – значения это не имеет, если принтер определен в файле *printcap*, и `lpd`, как следствие, знает его характеристики.

Программный модуль клиента предоставляет команды, позволяющие пользователю проверить состояние задания печати. Эти команды, их синтаксис и назначение описаны в табл. 9.2.

В синтаксисе описанных команд *принтер* – это имя принтера, определенное в файле */etc/printcap*, *пользователь* – имя учетной записи владельца задания печати, а *задание* определяется номером задания, назначенным на время его хранения в очереди печати. Вместо имени принтера в любой команде `lpc` может использоваться ключевое слово `all`, обозначающее все принтеры.

Таблица 9.2. Команды работы с принтером

Команда	Назначение
`lpc start [`*принтер*`]`	Запускает новый процесс демона печати
`lpc status [`*принтер*`]`	Отображает состояние принтера и его очереди печати
`lpq -Pпринтер [`*пользователь*`] [`*задание*`]`	Перечисляет задания в очереди печати указанного принтера
`lprm -P`*принтер задание*	Удаляет задание из очереди печати

`lpc` в основном предназначается для использования системным администратором, однако команды `status` и `start` доступны всем пользователям. Все команды, перечисленные в табл. 9.2, доступны для пользователей.

Команда `lpq` перечисляет задания, находящиеся в очереди принтера. Аргументы командной строки позволяют пользователю выбирать очередь определенного принтера и фильтровать выводимые сведения – просматривать задания конкретного пользователя либо просто отдельные задания. Вот пример перечисления заданий очереди для принтера *lp*:

```
$ lpq -Plp
Printer: lp@crab 'Canon'
 Queue: 4 printable jobs
 Server: pid 1459 active
 Unspooler: pid 1460 active
 Status: waiting for subserver to exit at 14:17:47.120
 Rank   Owner/ID             Class Job Files                 Size Time
active alana@crab+458         A   458 /usr/share/printconf 18043 14:16:53
2      micheal@crab+477       A   477 /usr/share/printconf/t 193 14:17:38
3      james@crab+479         A   479 /usr/share/printconf 18259 14:17:43
4      daniel@crab+481        A   481 /usr/share/printconf 18043 14:17:46
```

Владелец задания печати может удалить его из очереди при помощи команды `lprm`. Предположим, пользователь *daniel* хочет удалить задание с номером 481. Он выполняет такую команду:

```
$ lprm -Plp 481
Printer lp@crab:
  checking perms 'daniel@crab+481'
  dequeued 'daniel@crab+481'
```

`lpd` и `lpr` одними из первых вошли в инструментарий Unix-систем, нацеленный на использование потенциала сетей TCP/IP. Управление принтерами – в большой степени задача системного администрирования. Здесь же описаны только аспекты `lpd`, связанные печатью на удаленных устройствах.

Служба LP

Служба печати LP (*Line Printer*) используется в большинстве систем Unix System V. LP предоставляет те же возможности, что и LPD.

Традиционно в системах Unix System V файлы настройки LP хранятся в каталоге */etc/lp*. Основное назначение этих файлов – такое же, как у файла */etc/printcap* в LPD. Однако файлы */etc/lp* не редактируются системным администратором напрямую. Они создаются и изменяются при помощи команд System V – `lpadmin` и `lpsystem`.

Команда `lpadmin` добавляет локальные принтеры в каталог */etc/lp/printers*, делая их доступными службе LP. Базовый синтаксис команды для добавления принтера прост. Ключ `-p` определяет локальное имя принтера и используется совместно с ключом `-v`, определяющим интерфейс локального принтера, либо совместно с ключом `-s`, определяющим имена сервера и принтера для удаленного принтера. Например, следующая команда добавляет локальный принтер с именем `lp`, подключенный к параллельному порту принтера */dev/lp1*:

```
# lpadmin -plp -v /dev/lp1
```

А эта команда добавляет принтер *lj*, подключенный к удаленному серверу *crab*, в качестве принтера с локальным именем *lj*:

```
# lpadmin -llaser -s crab!lj
```

Конкретными характеристиками принтера, добавленного командой `lpadmin`, заведует файл *terminfo*. Файл *terminfo* практически идентичен файлу *printcap*. Подобно *printcap*, он имеет массу всевозможных параметров. За более подробной информацией по файлу *terminfo* обращайтесь к соответствующей странице руководства (man).

Команда `lpsystem` управляет доступом к принтерам в системах System V. По умолчанию в большинстве систем System V клиентам предоставляется доступ ко всем локальным принтерам. Параметры удаленного доступа к принтерам определяются в файле */etc/lp/Systems*, и по умолчанию в нем существует следующая запись:

```
+:x:-:s5:-:n:10:-:-:Allow all connections
```

Как видно из комментария в конце строки, эта запись разрешает всем удаленным системам доступ к локальным принтерам. Первое поле содержит имя узла, которому разрешен доступ. Знак сложения (+) в этом поле означает все узлы.

Поля записей файла */etc/lp/Systems* разделяются двоеточием (`:`). Поле, содержащее букву `x`, а также все поля, содержащие дефис (`-`), можно смело игнорировать, поскольку они не используются.

Четвертое поле определяет тип операционной системы, под управлением которой работает внешний клиент. Оно содержит значение `s5` для машин под управлением System V, использующих LP для передачи заданий печати, либо `bsd` для систем BSD, использующих LPD.

Буква `n` в шестом поле указывает, что данное «соединение» никогда не должно завершаться по истечении интервала ожидания. В этом поле может указываться длительность периода ожидания в минутах, но обычно такая воз-

можность не используется. Сохраняйте соединение, пока работает локальный сервер. Значение 10 является относительным. Оно указывает, что повторная попытка подключиться к удаленной системе должна выполняться через 10 минут. Это разумное значение: достаточно большое, чтобы удаленная система успела загрузиться после сбоя. `n` и 10 – значения по умолчанию, и обычно нет необходимости их изменять.

Не изменяйте файл */etc/lp/Systems* напрямую. Используйте для этой цели команду `lpsystem`. Чтобы удалить систему из файла *Systems*, воспользуйтесь командой `lpsystem` с ключом `-r` *hostname*, где *hostname* – значение первого поля записи, которую следует удалить. К примеру, чтобы удалить запись со знаком сложения (+) из стандартного файла */etc/lp/Systems*, наберите:

```
# lpsystem -r +
```

Чтобы добавить запись в файл *Systems*, используйте команду `lpsystem` без ключа `-r`. Например, чтобы добавить систему BSD по имени *clock*, наберите:

```
# lpsystem -t bsd -y "Linux PC in room 820" clock
```

В результате выполнения этой команды в файл *Systems* добавляется такая строка:

```
clock:x:-:bsd:-:n:10:-:-:Linux PC in room 820
```

Ключ `-t` определяет тип операционной системы. Ключ `-y` позволяет добавить комментарий; а `clock`, как можно догадаться, – имя узла. Для интервалов ожидания и повторения попытки мы оставили значения по умолчанию. Эти значения можно изменить при помощи ключей командной строки `-T` *timeout* и `-R` *retry*. За более подробными сведениями обращайтесь к странице руководства `lpsystem` (man).

Команды `lpadmin` и `lpsystem` существуют в большинстве систем System V, включая Solaris. Однако Solaris 8 не полагается целиком и полностью на эти две команды и каталог */etc/lp* в настройке LP. В системе Solaris настройка принтеров выполняется посредством файла */etc/printers.conf*. Команда `lpadmin` добавляет новые принтеры в файл */etc/printers.conf*, тогда как настройка принтеров обычно производится в окне Printer Manager (Менеджер печати) (рис. 9.1) приложения `admintool`.

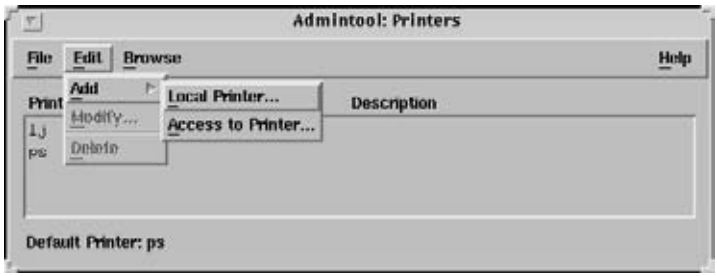


Рис. 9.1. *Менеджер печати*

Клиенты выбирают Add (Добавить), затем Access to Printer (Доступ к принтеру) из меню Edit (Правка) и вводят имя удаленного принтера и сервера в открывшемся окне. Серверу, чтобы организовать совместный доступ к принтеру, достаточно выбрать Add (Добавить), затем Local Printer (Локальный принтер) из того же меню и выполнить настройку локального принтера.

Все ОС Unix предоставляют какие-либо механизмы организации совместного доступа к принтерам. Задача администратора сети – убедиться, что принтеры доступны по сети и соответствующим образом защищены.

Samba и Windows: совместный доступ к ресурсам

NFS и `lpd` – это службы совместного доступа к файлам и принтерам для систем Unix, причем обе являются «родными» приложениями TCP/IP. Службы совместного доступа к принтерам и файлам в Microsoft Windows работают на основе протокола NetBIOS (Network Basic Input Output System). Samba соединяет эти два мира, обеспечивая совместный доступ к ресурсам систем под управлением Unix и Windows. Samba – ключ к интеграции систем Unix и Windows, позволяющий системе Unix быть сервером файлов и печати для клиентов Windows либо выступать в качестве клиента сервера Windows.

Для обмена данными клиенты и серверы NetBIOS используют протокол SMB (Server Message Block) – протокол блоков серверных сообщений. Изначально NetBIOS представлял собой монолитный протокол, самостоятельно переносивший данные от приложения на уровень физической сети. Сегодня NetBIOS работает поверх TCP/IP, что позволяет исполнять приложения NetBIOS в системах Unix, реализующих семейство протоколов TCP/IP.

Для работы NetBIOS в сети TCP/IP необходимы две вещи: протокол для передачи данных NetBIOS через стек TCP/IP и способ отображения адресов NetBIOS в адреса TCP/IP. Протокол транспортировки данных NetBIOS носит название NBT (NetBIOS over TCP/IP), его определение содержится в документах RFC 1001 и RFC 1002. Отображение адресов выполняет специальный сервер имен NetBIOS. Samba реализует обе функции.

Службы Samba реализованы в виде пары демонов. Демон SMB (`smbd`), сердце Samba, обеспечивает совместный доступ к файлам и принтерам. Демон сервера имен NetBIOS (`nmbd`) реализует преобразование имен NetBIOS в адреса IP. NBT требует способа преобразования имен компьютеров NetBIOS, являющихся адресами сети NetBIOS, в IP-адреса сети TCP/IP.

Samba входит в состав большинства дистрибутивов Linux и устанавливается в процессе начальной установки системы. В системе Red Hat в процессе загрузки системы выполняется сценарий */etc/rc.d/init.d/smb*, запускающий `smbd` и `nmbd`. Samba не входит в поставку Solaris 8, однако пакет можно получить в сети Интернет. Отправляйтесь на сайт *http://www.samba.org* и выберите территориально ближайший сервер для загрузки файла.

Настройка сервера Samba

Настройки сервера Samba хранятся в файле *smb.conf*. Взгляните на загрузочный сценарий, чтобы выяснить, где, по мнению `smbd`, находится файл настройки. В системе Red Hat этот файл называется */etc/samba/smb.conf*. В системе Caldera – */etc/samba.d/smb.conf*. Имя по умолчанию, которое фигурирует в большинстве документов по Samba, – */usr/local/samba/lib/smb.conf*. Воспользуйтесь командой `find` или сверьтесь с загрузочным сценарием, чтобы определить расположение этого файла в своей системе.

Файл *smb.conf* состоит из разделов. За исключением раздела *global*, содержащего параметры настройки сервера в целом, названия разделов соответствуют именам совместно используемых ресурсов (shares), которые сервер предоставляет клиентам. Таким ресурсом может быть файловая система либо печатающее устройство.

Лучший способ изучить файл *smb.conf* – рассмотреть конкретный пример. Если не считать ресурса printers, о котором мы поговорим позже, файл *smb.conf* системы Red Hat содержит следующие активные строки:

```
[global]
   workgroup = MYGROUP
   server string = Samba Server
   printcap name = /etc/printcap
   load printers = yes
   printing = lprng
   log file = /var/log/samba/%m.log
   max log size = 0
   security = user
   encrypt passwords = yes
   smb passwd file = /etc/samba/smbpasswd
   socket options = TCP_NODELAY SO_RCVBUF=8192 SO_SNDBUF=8192
  dns proxy = no
[homes]
   comment = Home Directories
   browseable = no
   writable = yes
   valid users = %S
   create mode = 0664
   directory mode = 0775
```

В приведенном фрагменте мы наблюдаем два раздела файла настройки: *global* и *homes*. Раздел *global* содержит параметры, влияющие на работу сервера в целом:

`workgroup`

Определяет иерархическую группу узлов (рабочую группу, workgroup), к которой принадлежит сервер. Замените имя MYGROUP из примера осмысленным именем рабочей группы не длиннее 15 символов. Убедитесь, что имя содержательно. Никогда не используйте имена вроде MYGROUP или WORKGROUP.

`server string`

Определяет описательный комментарий для данного сервера. Комментарий отображается по команде `net view` на DOS-клиентах. Замените строку из примера на осмысленное описание своей системы.

`printcap name`

Определяет расположение файла *printcap*. Файл *printcap* позволяет определить, для каких принтеров может осуществляться совместный доступ. Имя по умолчанию – */etc/printcap*.

`load printers`

Указывает необходимость предоставлять доступ ко всем принтерам, определенным в файле *printcap*. По умолчанию имеет значение `yes`, которое предписывает использовать все принтеры. Значение `no` запрещает чтение определений из файла *printcap*. В случае когда указано значение `no`, все доступные клиентам принтеры должны быть определены в индивидуальном порядке.

`printing`

Определяет систему печати Unix, используемую сервером. В примере принимает значение `lprng` (LPR Next Generation, LPR нового поколения), то есть указывает на реализацию стандартной системы LPR/LPD, описанной ранее в этой главе.

`log file`

Определяет расположение файла журнала. В нашем примере имя содержит переменную `%m`[1], значением которой является NetBIOS-имя клиента. Для каждого клиента создается отдельный файл журнала с именем клиента в качестве расширения. Для клиента с NetBIOS-именем *crab* файл журнала получит имя */var/log/samba/log.crab*. Для клиента с NetBIOS-именем *rodent* – имя */var/log/samba/log.rodent*.

`max log size`

Определяет максимальный размер файла журнала. По умолчанию этот размер равен 5 Мбайт. (При превышении максимального размера демон `smbd` закрывает журнал и переименовывает файл, добавляя расширение *.old*.) В примере данный параметр принимает значение 0, что означает отсутствие ограничений на размер файлов журналов.

`security`

Определяет используемый механизм безопасности. Возможны 4 варианта:

`share`

Безопасность уровня совместно используемого ресурса. Самый низкий уровень безопасности. Ресурс доступен всем. С ним может быть связан пароль, однако этот пароль одинаков для всех клиентов.

---

[1] В Samba существует около 20 различных переменных. Полный их список приводится на соответствующих страницах руководства (man).

`user`

Безопасность уровня пользователя. Каждый пользователь должен указать имя и соответствующий пароль. По умолчанию используются имена пользователей и пароли из файла */etc/passwd*. Значения паролей по умолчанию могут изменяться. Более подробная информация о паролях приводится ниже в тексте раздела.

`server`

Безопасность уровня сервера. Работает аналогично безопасности уровня пользователя, но для авторизации пользователя используется внешний сервер. Внешний сервер необходимо указать при помощи параметра `password server`.

`domain`

Безопасность уровня домена. Сервер Linux присоединяется к домену Windows NT/2000 и использует контроллер домена Windows NT/2000 в качестве сервера, проверяющего имена пользователей и пароли. Чтобы указать первичный контроллер домена Windows NT/2000 (PDC, primary domain controller), воспользуйтесь параметром `password server`. На сервере PDC необходимо создать учетную запись для системы Linux. Наконец, добавьте следующие строки в раздел настройки *global* на системе Linux:

```
domain master = no
local master = no
preferred master = no
os level = 0
```

`encrypt passwords`

Значение `yes` данного параметра предписывает Samba шифровать пароли при передаче их по сети. Это повышает совместимость сервера с клиентами Windows, начиная с системы Windows 98, которые по умолчанию используют шифрованные пароли, и затрудняет перехват паролей злоумышленниками. По умолчанию в Samba используются пароли Unix, передаваемые открытым текстом.

`smb passwd file`

Данный параметр определяет расположение файла *smbpasswd*, хранящего зашифрованные пароли Samba. При использовании зашифрованных паролей серверу Samba приходится вести два файла паролей: *passwd* и *smbpasswd*. Используйте сценарий *mksmbpasswd.sh* для создания начальной версии файла *smbpasswd* из существующего файла *passwd*.

`socket options`

Определяет параметры настройки производительности. Не является обязательным, хотя установка размера буферов передачи и приема в 8 Кбайт может немного повысить производительность. В данном примере настройки Red Hat параметр `TCP_NODELAY`, предписывающий Samba посылать наборы пакетов при каждой передаче, не имеет никакого смысла, по-

скольку таково поведение системы по умолчанию для версий Samba начиная с 2.0.4. Настройка производительности Samba достаточно хорошо описана в приложении B книги «Using Samba» Питера Келли (Peter Kelly), Роберта Экштейна (Robert Eckstein) и Дэвида Колье-Брауна (David Collier-Brown), изданной O'Reilly.

`dns proxy`

Указывает, следует ли демону `nmbd` передавать неразрешенные запросы NBNS системе DNS.

Помимо описанных параметров, существуют и другие, часто используемые в разделе *global*: они описаны в табл. 9.3.

Таблица 9.3. Прочие параметры раздела global

Параметр	Назначение
deadtime	Определяет интервал ожидания для неактивных соединений
debug level	Устанавливает уровень диагностики для сообщений журнала
keepalive	Использует keepalive-сообщения для выяснения состояния клиентов
lock directory	Определяет каталог, где хранятся файлы *wins.dat*, файлы состояния, а также файлы блокировки
message command	Определяет способ обработки сообщений WinPopup демоном `smbd`
name resolve order	Определяет порядок обращения к службам при разрешении имен NetBIOS. Допустимые значения: lmhosts, hosts, wins и bcast
netbios aliases	Определяет дополнительные имена, на которые отзывается сервер
netbios name	Определяет NetBIOS-имя сервера
syslog	Отображает уровни отладки в уровни syslog
syslog only	Предписывает использовать syslog вместо файлов журналов Samba
time server	Предписывает серверу объявлять о своей способности выступать в роли сервера времени Windows
wins support	Включает сервер имен WINS

Как можно видеть из примера настройки системы Red Hat, многие серверы поставляются с разумными глобальными настройками, позволяющими начать работу простой серверной системы. Помимо готового раздела *global*, в стандартном файле Red Hat существует еще и готовый раздел *homes*.

Раздел homes файла smb.conf

Раздел *homes* – это специальный раздел ресурса. Он предписывает `smbd` разрешить пользователям доступ к своим исходным каталогам посредством SMB. В отличие от других разделов, рассмотренных ниже, он не содержит имени конкретного каталога, для которого организуется совместный до-

ступ. smbd использует домашний каталог пользователя из файла */etc/passwd*, основываясь на имени пользователя, обратившегося к ресурсу. Параметры настройки в разделе *homes* для системы Red Hat следующие:

comment

Содержит описание ресурса, отображаемое в поле комментария окна Network Neighborhood при просмотре ресурса с компьютера, работающего под управлением Windows.

browseable

Указывает, всем ли пользователям разрешено просматривать содержимое данного ресурса. no означает, что просмотр сегмента разрешен только пользователям с корректным идентификатором пользователя. yes разрешает просмотр всем пользователям, независимо от значений UID. Параметр контролирует только просмотр; непосредственный доступ к содержимому ресурса управляется стандартными файловыми правами доступа Linux.

writable

Определяет, разрешена ли запись файлов в этот ресурс. Значение yes разрешает запись. Значение no делает ресурс доступным только для чтения. Данный параметр определяет действия, разрешенные сервером Samba. Непосредственная работа с содержимым каталога, связанного с ресурсом, по-прежнему управляется стандартными файловыми правами доступа Linux.

valid users

Указывает пользователей, которым разрешено использовать данный ресурс. В примере переменная %S принимает значение имени пользователя, которому разрешен доступ к ресурсу.

create mode

Определяет права доступа к файлу, устанавливаемые при создании пользователем файла в ресурсе *homes*.

directory mode

Определяет права доступа к каталогу, устанавливаемые при создании клиентом каталога в ресурсе *homes*.

Организация совместного доступа к каталогам

Чтобы сделать каталог доступным посредством Samba, создайте в файле *smb.conf* раздел ресурса, описывающий каталог и условия доступа к нему. Чтобы организовать совместный доступ к каталогам */usr/doc/pcdocs* и */home/research* (последний встречался нам в примерах по NFS), добавьте следующие два раздела в файл *smb.conf*:

```
[pcdocs]
     comment = PC Documentation
     path = /usr/doc/pcdocs
     browseable = yes
     writable = no
```

```
        public = yes

[research]
        comment = Research Deptment Shared Directory
        path = /home/research
        browseable = no
        writable = yes
        create mode = 0750
        hosts allow = horseshoe, jerboas, crab, rodent
```

Каждый ресурс обозначен содержательным именем. Это имя отображается как отдельная папка в окне Network Neighborhood на машине пользователя. Приведенный фрагмент содержит команды, которые мы уже изучили, и несколько новых команд. Первая из новых команд – `path`, она определяет путь к каталогу сегмента.

Кроме того, сегмент *pcdocs* содержит команду `public`, разрешающую доступ всем пользователям, даже в отсутствие корректного имени пользователя и пароля. Такие пользователи получают «гостевой вход» для работы с ресурсом. Применительно к системе Linux это означает, что они работают с полномочиями пользователя *nobody* и ограничены в доступе к файлам.

В ресурсе *research* запись файлов разрешена. Команда `create mode` определяет права доступа, устанавливаемые для файлов, создаваемых клиентом в этом ресурсе. В нашем примере значение 0750 определяет следующие права для созданных файлов: чтение/запись/исполнение владельцем, чтение/исполнение группой, запрет любых действий для прочих пользователей. Родственная команда `directory mode` определяет права доступа для создаваемых в пределах ресурса каталогов. Пример:

```
directory mode = 0744
```

Для новых каталогов устанавливаются такие права доступа: чтение/запись/исполнение владельцем, чтение/исполнение группой, чтение/исполнение для прочих пользователей. Это разумное значение, позволяющее командам `cd` и `ls` работать предсказуемым образом.

Раздел *research* содержит команду `hosts allow`, определяющую клиентов, которым разрешен доступ к ресурсу. Даже если пользователь представил корректное имя и правильный пароль, доступ к ресурсу ему разрешен только с конкретного узла. По умолчанию доступ разрешен со всех узлов, а более точное управление доступом осуществляется при помощи имен пользователей и паролей.

Помимо команды `hosts allow` существует команда `hosts deny`, явным образом запрещающая доступ к ресурсу перечисленным клиентам. Ее синтаксис идентичен синтаксису команды `hosts allow`.

Эти два новых раздела ресурсов вместе с теми, что по умолчанию существовали в настройках Red Hat, создают сервер, позволяющий пользователям обращаться к своим исходным каталогам, открытым каталогам, а также к закрытым каталогам, доступным только пользователям групп. Результат –

те же возможности, что предоставляет NFS, но более простая интеграция для клиентов, работающих под управлением Microsoft Windows. Samba также может использоваться для предоставления клиентам Windows доступа к принтерам.

Организация совместного доступа к принтерам

Настройка доступа к принтерам также выполняется в файле *smb.conf*. В составе системы Red Hat поставляется файл *smb.conf*, уже содержащий настройки для обеспечения совместного доступа к принтерам. Следующие строки следуют в файле *smb.conf* системы Red Hat непосредственно за разделами *global* и *homes*:

```
[printers]
   comment = All Printers
   path = /var/spool/samba
   browseable = no
   guest ok = no
   writable = no
   printable = yes
```

Строки `printcap` и `load printers` раздела *global* готовят сервер к организации совместного доступа к принтерам, определенным в файле *printcap*. Приведенный раздел *printers* делает эти принтеры доступными клиентам, подобно тому, как раздел *homes* делает доступными пользователям их исходные каталоги. В Red Hat раздел ресурса *printers* содержит пять параметров.

С тремя из них – `comment`, `browseable` и `path` – мы уже встречались. Однако в случае принтеров параметр `path` определяет не путь к файлам, но путь к каталогу буферной зоны печатающих устройств SMB.

Здесь мы познакомимся еще с двумя параметрами. Первый из них, `printable`, указывает, что данный ресурс является принтером. По умолчанию этот параметр имеет значение `no`, то есть ресурсы по умолчанию считаются файловыми. Чтобы создать ресурс принтера, необходимо присвоить этому параметру значение `yes`. Использование `printable = yes` разрешает клиентам запись файлов для печати в каталог буферной зоны, обозначенный параметром `path`. Чтобы ограничить права доступа к файлам, создаваемым клиентами в каталоге буферной зоны, используйте команду `create mode`. К примеру, такую: `create mode = 0700`.

Вторая новая строка, `guest ok`, определяет, разрешен ли доступ к ресурсу «гостям». Действие данного параметра абсолютно идентично действию параметра `public`, о котором говорилось выше, так что эти два параметра взаимозаменяемы. Значение `no` запрещает пользователю *nobody* отправку заданий на принтер. Пользователю нужна действующая учетная запись, чтобы воспользоваться принтером. По замыслу такой подход предотвращает злоупотребление принтером со стороны пользователей с гостевым доступом, однако корректное имя пользователя позволяет еще и выполнять сортировку заданий печати, если используются титульные листы и ведется учет потребления ресурса.

Как правило, сервер печати предлагает всем клиентам доступ ко всем своим принтерам. При этом для каждого принтера может быть создан отдельный совместно используемый ресурс – точно так же, как в случае файлов. Если вы не хотите открывать доступ ко всем принтерам, удалите раздел *printers*, назначьте параметру `load printers` значение `no`, а затем создайте отдельные разделы ресурсов только для тех принтеров, которые должны быть доступны клиентам.

Файл *smb.conf* с разделом ресурса для отдельного принтера может выглядеть следующим образом:

```
[global]
   workgroup = BOOKS
   server string = Print Server
   load printers = no
   security = user
[homes]
   comment = Home Directories
   browseable = no
   writable = yes
[hp5m]
   comment = PostScript Laser Printer
   path = /var/spool/samba
   browseable = no
   public = no
   create mode = 0700
   printable = yes
   printer = lp
```

В данном файле отсутствует раздел *printers*. Вместо него мы добавили раздел ресурса с именем `hp5m` – и этот ресурс обеспечивает доступ к принтеру по имени `lp`. Чтобы схема заработала, принтеру должно соответствовать определение в файле *printcap*. Для параметра *printcap* можно использовать значение по умолчанию – */etc/printcap*.

Демон `smbd` в Samba обеспечивает совместный доступ к файлам и принтерам. Второй компонент Samba – демон `nmbd`.

Служба имен NetBIOS

Демон сервера имен NetBIOS (`nmbd`) входит в состав базового программного дистрибутива Samba и превращает сервер Unix в сервер имен NetBIOS (NBNS). Демон `nmbd` умеет обрабатывать запросы от клиентов LanManager, а кроме того, может быть настроен на работу в качестве сервера WINS (Windows Internet Name Server).

Настройка демона `nmbd` выполняется в разделе *global* файла *smb.conf*. Следующие параметры относятся к работе сервера WINS:

`wins support`

Допустимые значения: `yes` и `no`. Определяет, следует ли демону `nmbd` работать в качестве сервера WINS. По умолчанию принято значение `no`, поэто-

му по умолчанию nmbd предоставляет средства просмотра, но не службу WINS.

dns proxy

Допустимые значения: yes и no. Данный параметр предписывает nmbd использовать DNS для разрешения запросов WINS, которые не могут быть разрешены другим путем. Параметр имеет смысл только в случае, когда nmbd выступает в роли сервера WINS. Значение по умолчанию – yes. Система DNS может быть полезна в разрешении имен NetBIOS, только если имена NetBIOS и имена узлов DNS совпадают.

wins server

Позволяет указать IP-адрес внешнего сервера WINS. Параметр имеет смысл, только если на локальной системе Linux не установлен сервер WINS. Параметр сообщает системе Samba адрес внешнего сервера WINS, которому следует направлять запросы по именам NetBIOS.

wins proxy

Допустимые значения: yes и no. По умолчанию – no. Значение yes предписывает nmbd при разрешении широковещательных запросов по именам NetBIOS преобразовывать их в индивидуальные запросы и передавать напрямую серверу WINS. Если wins support имеет значение yes, эти запросы обрабатываются непосредственно демоном nmbd. Если же указано значение параметра wins server, такие запросы передаются внешнему серверу. Параметр wins proxy требуется только в случаях, когда клиенты не знают адреса сервера либо не понимают протокола WINS.

Сервер имен NetBIOS обычно запускается в процессе загрузки системы при помощи следующей команды:

```
nmbd -D
```

При запуске с ключом -D nmbd работает непрерывно в качестве демона и принимает запросы к службе имен NetBIOS через порт 137. Сервер отвечает на запросы, исходя из сведений, предоставленных клиентами при регистрации, а также соответствия имен NetBIOS и адресов, полученного от других серверов.

Файл *lmhosts* используется для создания отображений адресов вручную, когда возникает такая необходимость. Большинству серверов WINS файл *lmhosts* не требуется, поскольку отображения поступают динамически от клиентов и других серверов. Имена NetBIOS регистрируются автоматически; клиенты регистрируют свои имена NetBIOS на сервере в процессе загрузки. Адреса и имена хранятся в базе данных WINS, *wins.dat*. Файл *lmhosts* является лишь небольшой составляющей полной базы данных.

Файл *lmhosts* схож с файлом *hosts*, описанным в главе 4. Каждая запись начинается с адреса IP и содержит имя узла. Однако на этот раз вместо имени узла фигурирует имя NetBIOS. Вот пример файла *lmhosts*:

```
$ cat /etc/lmhosts
172.16.12.3       horseshoe
172.16.12.1       crab
```

```
172.16.12.2       rodent
172.16.12.4       jerboas
```

В соответствии с этим файлом *lmhosts* имя NetBIOS *rodent* отображается в адрес IP 172.16.12.2. Обратите внимание, что эти имена NetBIOS совпадают с именами узлов TCP/IP, назначенными клиентам. Используйте одинаковые имена в NetBIOS и TCP/IP. Иная политика выбора имен ограничивает возможности настройки и создает путаницу.

Сетевая информационная служба (NIS)

Сетевая информационная служба NIS (Network Information Service)[1] – это административная база данных, реализующая централизованное сопровождение и автоматическое распространение важных административных файлов. NIS позволяет преобразовать ряд стандартных файлов Unix в базы данных, к которым можно обращаться по сети. Такие базы данных называются картами NIS. Некоторые карты создаются из файлов, с которыми вы знакомы по системному администрированию, – в частности, из файла паролей (*/etc/passwd*) и файла групп (*/etc/group*). Прочие являются производными от файлов, связанных с сетевым администрированием:

/etc/ethers

Источник NIS-карт *ethers.byaddr* и *ethers.byname*. Файл */etc/ethers* используется в работе RARP (см. главу 2).

/etc/hosts

Источник карт *hosts.byname* и *hosts.byaddr* (см. главу 3).

/etc/networks

Источник карт *networks.byname* и *networks.byaddr* (см. главу 3).

/etc/protocols

Источник карт *protocols.byname* и *protocols.byaddr* (см. главу 2).

/etc/services

Источник одной карты – *services.byname* (см. главу 2).

/etc/aliases

Определяет псевдонимы для адресов электронной почты и служит источником карт *mail.aliases* и *mail.byaddr* (см. главу 10).

Определить, какие карты доступны на сервере, можно при помощи команды `ypcat -x`. Данная команда выводит одинаковый список для наших тестовых систем Solaris и Linux. Список для вашего сервера может отличаться. Вот список, полученный в системе Linux:

```
% ypcat -x
Use "passwd"    for map "passwd.byname"
```

[1] Изначально NIS носила название «Желтые страницы» (Yellow Pages, или *yp*). Название изменилось, но аббревиатура *yp* по-прежнему в ходу.

```
Use "group"     for map "group.byname"
Use "networks"  for map "networks.byaddr"
Use "hosts"     for map "hosts.byname"
Use "protocols" for map "protocols.bynumber"
Use "services"  for map "services.byname"
Use "aliases"   for map "mail.aliases"
Use "ethers"    for map "ethers.byname"
```

NIS позволяет централизованно хранить на сервере важные административные файлы, предоставляя доступ к ним всем рабочим станциям сети. Все карты хранятся на головном сервере, выполняющем серверный процесс NIS – `ypserv`. Клиентские системы обращаются к картам по сети. Клиенты используют `ypbind` для обнаружения сервера.

Сервер NIS и его клиенты составляют *домен NIS*. Этот термин NIS делит с DNS. Домен NIS обозначается доменным именем NIS. Единственное предъявляемое к именам требование заключается в том, что в пределах одной локальной сети различные домены NIS должны иметь разные имена. Хотя домены NIS и DNS – различные сущности, но Sun рекомендует использовать в качестве имен NIS-доменов имена DNS-доменов, чтобы упростить администрирование и избежать путаницы.

NIS использует имя своего домена при создании подкаталога в иерархии */var/yp* для хранения карт. Например, домен DNS нашей воображаемой сети носит имя *wrotethebook.com*, и такое же имя мы используем для нашего домена NIS. NIS создает каталог */var/yp/wrotethebook.com* и хранит в нем карты NIS.

Протоколы и команды NIS изначально были определены компанией Sun Microsystems, однако сама служба в настоящее время получила широкое распространение в различных реализациях. Многочисленные примеры этого раздела относятся к системе Linux, а не Solaris, и замечательно иллюстрируют этот факт. Синтаксис команд в различных системах весьма схож.

Команда `domainname` позволяет узнать либо задать имя домена NIS. Суперпользователь может сделать *wrotethebook.com* именем домена NIS, выполнив следующую команду:

```
# domainname wrotethebook.com
```

Имя домена NIS обычно указывается в процессе загрузки системы – с этой целью команда `domainname` помещается в один из загрузочных сценариев. Во многих системах имя домена NIS, выступающее в качестве аргумента команды `domainname`, хранится в отдельном файле. Так, в системах Solaris значение имени домена NIS извлекается из файла */etc/defaultdomain*. Как видно из следующего примера, `defaultdomain` содержит только имя домена NIS:

```
% cat /etc/defaultdomain
wrotethebook.com
```

В системах Red Hat Linux имя домена NIS – лишь одно из значений, хранящихся в файле */etc/sysconfig/network*:

```
$ cat /etc/sysconfig/network
NETWORKING=yes
```

```
HOSTNAME=jerboas.wrotethebook.com
NISDOMAIN=wrotethebook.com
```

Инициализация сервера NIS и создание первых вариантов карт выполняется при помощи `make`. Файл */var/yp/Makefile* содержит инструкции, необходимые для сборки карт. Как уже говорилось, каталог для карт получает имя, совпадающее с именем домена NIS. *Makefile* читает файлы из каталога */etc* и сохраняет созданные на их основе карты в новом каталоге. Чтобы инициализировать систему Linux в качестве сервера NIS, выполните подобную последовательность команд:

```
# domainname wrotethebook.com
# cd /var/yp
# make
make[1]: Entering directory '/var/yp/wrotethebook.com'
Updating hosts.byname...
Updating hosts.byaddr...
Updating networks.byaddr...
Updating networks.byname...
Updating protocols.bynumber...
Updating protocols.byname...
Updating rpc.byname...
Updating rpc.bynumber...
Updating services.byname...
Updating passwd.byname...
Updating passwd.byuid...
Updating group.byname...
Updating group.bygid...
Updating netid.byname...
make[1]: Leaving directory '/var/yp/wrotethebook.com'
```

После инициализации карт запустите процесс сервера NIS `ypserv` и связующий процесс NIS – `ypbind`:[1]

```
# ypserv
# ypbind
```

Наша система теперь выступает в роли как NIS-сервера, так и NIS-клиента. Быстрая проверка при помощи `ypwhich` показывает, что мы связаны с нужным сервером. Воспользуйтесь `ypcat` или `ypmatch`, чтобы убедиться, что данные с сервера действительно доступны. В следующем примере мы используем `ypcat`:

```
# ypwhich
localhost
# ypcat hosts
172.16.55.105           cow cow.wrotethebook.com
172.16.55.106           pig pig.wrotethebook.com
```

[1] Если в процессе первой сборки карт NIS `make` сообщит, что сервер `ypserv` не зарегистрирован, запустите `ypserv`, прежде чем выполнять `make`.

```
172.16.26.36            island.wrotethebook.com island
127.0.0.1               localhost
```

Клиентам остается только указать корректное доменное имя и выполнить приложение связующего модуля – `ypbind`:

```
# domainname wrotethebook.com
# ypbind
```

Большинство клиентов NIS для обнаружения сервера используют `ypbind`. Исходя из имени домена NIS, `ypbind` передает широковещательный запрос к серверу этого домена. К первому из ответивших серверов и «привязывается» клиент. Теория гласит, что сервер, ответивший быстрее прочих, имеет наименьшую загрузку. Как правило, гипотеза работает хорошо. Тем не менее существует вероятность привязки клиента к неподходящей системе, а именно к системе, на которой случайно оказался установлен сервер `ypserv`, либо к системе, которую специально настроили в качестве ложного сервера. Принимая во внимание такую возможность, некоторые системы позволяют явным образом указать сервер, с которым общается клиент. Этой цели в Linux служит файл */etc/yp.conf*. Синтаксис записей файла варьируется от версии к версии, поэтому обратитесь к документации по своей системе, прежде чем использовать файл.

Поместите имя домена NIS в подходящий загрузочный файл, чтобы настройки NIS продолжали действовать и после перезагрузки. Команды `ypbind` и `ypserv`, вероятно, уже присутствуют в одном из загрузочных сценариев. В системе Red Hat Linux программам `ypbind` и `ypserv` соответствуют специальные сценарии в каталоге */etc/init.d*. Поместив значение `NISDOMAIN` в */etc/sysconfig/network*, не забудьте при помощи команды `chkconfig` убедиться, что сценарии `ypbind` и `ypserv` выполняются в процессе загрузки системы.

NIS является возможной альтернативой DNS, однако в большинстве систем применяется как NIS, так и DNS. Преобразование имен узлов в IP-адреса может быть возложено на DNS, NIS и таблицу узлов. Порядок опроса различных источников определяется в файле *nsswitch.conf*.

Файл nsswitch.conf

Файл коммутации службы имен (Name Service Switch) *nsswitch.conf* определяет порядок поиска информации в различных источниках. Вопреки названию, действие файла распространяется не только на службу имен. *nsswitch.conf* позволяет работать с любыми базами данных NIS, как можно видеть из следующего примера:

```
hosts:      dns  nis  files
networks:   nis  [NOTFOUND=return]  files
services:   nis  files
protocols:  nis  files
```

Первая запись файла гласит, что поиск информации по имени узла выполняется прежде всего в DNS; если DNS не находит соответствия, поисковый

запрос передается NIS, и только в последнюю очередь выполняется поиск в файле *hosts*. Вторая запись предписывает осуществлять поиск сетевых имен в NIS. Строка `[NOTFOUND=return]` предписывает использовать файл *networks* только в случае, если служба NIS недоступна. В случае когда NIS не может найти имени указанной сети, поиск завершается. Последние две записи организуют поиск портов служб и номеров протоколов в NIS, а затем в файлах каталога */etc*.

NIS+

Прежде чем завершить рассказ о NIS, следует упомянуть NIS+. Этот раздел будет коротким, потому что я не использую NIS+ и знаю об этой системе не очень много.

NIS+ приходит на смену NIS в системах от Sun. NIS+ является не новой версией NIS, но совершенно самостоятельным программным продуктом, предоставляющим полноценную функциональность NIS плюс некоторые новые возможности. Вот эти возможности:

- Повышенная безопасность. NIS не позволяет проверять подлинность серверов (как отмечалось в разговоре об `ypbind`) или клиентов. В NIS+ существует механизм авторизации пользователей, работающий на базе шифрования DES. Кроме того, NIS+ реализует различные уровни доступа, позволяющие дифференцировать доступ пользователей к данным. NIS позволяет предоставить всем пользователям домена NIS только одинаковые права доступа.
- Иерархическая, децентрализованная архитектура. Подобно DNS, NIS+ является распределенной, иерархической системой баз данных. Это позволяет создавать очень вместительные пространства имен, а также распределить управление информационными структурами, сохраняя согласованность доступа к данным. NIS имеет линейную структуру. Вся информация о домене NIS поступает от единственного головного сервера, и домены NIS никак не связаны между собой.
- Более совершенные структуры данных. NIS преобразует ASCII-файлы в простые ассоциативные массивы, которые в документации NIS+ названы «картами в две колонки». *Таблицы* баз данных NIS+ состоят из многих колонок и позволяют производить поиск самой разнообразной информации по записи. Кроме того, для таблиц NIS+ может выполняться связывание с целью обеспечения дополнительной информации по определенной записи.

Очевидно, NIS+ имеет ряд превосходных новых возможностей и преимуществ перед NIS. Почему же я не использую NIS+? Хороший вопрос! Иерархическая архитектура и более совершенные структуры данных – важные элементы для большой сети, хранящей серьезные объемы данных в своем сегменте пространства имен. Однако многие сетевые площадки развивались с применением NIS в локальных подсетях и не испытывают необходимости в переводе предприятия в целом под управление NIS+. Улучшенная безопасность – аргу-

мент, похоже, железный, однако площадки с низкими требованиями к безопасности опять же не испытывают необходимости в ее улучшении, а площадки с высокими требованиями к безопасности, скорее всего, уже защищены брандмауэром, блокирующим внешние запросы к NIS. Кроме того, реализации NIS охватывают большее число операционных систем, нежели реализации NIS+. И наконец, вместо NIS+ все чаще используются другие службы каталогов, предоставляющие подобный уровень обслуживания и более доступные, например LDAP. Сумма этих причин замедлила переход на NIS+.

Чтобы узнать больше о NIS+ и ее установке в своей системе, прочтите документы *NIS+ Transition Guide* (Руководство по переходу на NIS+), *Name Service Configuration Guide* (Руководство по настройке службы имен) и *Name Service Administration Guide* (Руководство администратора службы имен). Все они доступны в составе набора руководств *Solaris System and Network Administration* (Системное и сетевое администрирование Solaris) от Sun.

NIS и NIS+ обеспечивают своих клиентов широким диапазоном сведений, относящихся к настройке системы. При этом они не позволяют получать информацию, необходимую для настройки подсистемы TCP/IP. В следующих двух разделах мы изучим серверы настройки, позволяющие решать эту задачу в комплексе.

DHCP

Протокол инициализации BOOTP (Bootstrap Protocol) стал первым всесторонним протоколом настройки. Он предоставляет всю информацию, обычно необходимую для настройки TCP/IP, начиная с адреса IP клиента и заканчивая сервером печати. BOOTP оказался настолько простым и эффективным протоколом, что лег в основу протокола динамической настройки узлов DHCP (Dynamic Host Configuration Protocol). DHCP работает через те же порты UDP, 67 и 68, что и BOOTP. Этот протокол предоставляет все возможности BOOTP, а также включает важные расширения. Три главные особенности протокола DHCP:

Обратная совместимость с протоколом инициализации BOOTP

Сервер DHCP способен поддерживать клиентов BOOTP. Правильно настроенный сервер DHCP может обслуживать всех ваших клиентов.

Полноценная настройка

Сервер DHCP предоставляет клиентам полный набор параметров настройки TCP/IP. (Полный перечень содержится в приложении D.) Администратор сети получает возможность управлять всеми настройками пользователей.

Динамическое назначение адресов

Сервер DHCP позволяет выделять постоянные адреса вручную либо автоматически, а временные адреса – динамически. Администратор сети выбирает механизм назначения адресов, исходя из нужд сети и систем клиентов.

В этом разделе мы создадим сервер DHCP, который поддерживает клиентов BOOTP, динамически выделяет адреса, а также представляет клиентам широкий диапазон параметров настройки.

Для систем Unix существует несколько реализаций DHCP. Одни пакеты являются платными, другие работают только с определенными вариантами Unix. Мы воспользуемся демоном `dhcpd`, разработанным консорциумом ISC (Internet Software Consortium). Он свободно доступен в Интернете и работает с широким спектром систем Unix, включая наши тестовые системы Linux и Solaris. (Приложение D содержит информацию о том, где взять и как скомпилировать `dhcpd`.) Обратившись к другому серверному модулю DHCP, вы, вероятнее всего, столкнетесь с иным набором команд настройки, однако основная функциональность отличаться не будет.

dhcpd.conf

Настройки `dhcpd` хранятся в файле */etc/dhcpd.conf*. Файл настройки содержит инструкции, которые определяют, какие подсети и узлы обслуживает сервер и какую информацию настройки он им предоставляет. *dhcpd.conf* – обычный текстовый файл, сходный с файлом исходного текста на языке C. Проще всего изучить файл *dhcpd.conf* на примере:

```
# Определения значений, действующих для всех систем.

default-lease-time 86400;
max-lease-time 604800;
get-lease-hostnames true;
option subnet-mask 255.255.255.0;
option domain-name "wrotethebook.com";
option domain-name-servers 172.16.12.1, 172.16.3.5;
option lpr-servers 172.16.12.1;
option interface-mtu 1500;

# Определения подсетей, параметров для отдельных подсетей,
# диапазона адресов, доступных для динамического распределения.

subnet 172.16.3.0 netmask 255.255.255.0 {
    option routers 172.16.3.25;
    option broadcast-address 172.16.3.255;
    range 172.16.3.50 172.16.3.250;
}

subnet 172.16.12.0 netmask 255.255.255.0 {
    option routers 172.16.12.1;
    option broadcast-address 172.16.12.255;
    range 172.16.12.64 172.16.12.192;
    range 172.16.12.200 172.16.12.250;
}

# Определения клиентов BOOTP - оператор host

group {
    use-host-decl-names true;
```

```
    host 24seven {
        hardware ethernet 00:80:c7:aa:a8:04;
        fixed-address 172.16.3.4;
    }
    host rodent {
        hardware ethernet 08:80:20:01:59:c3;
        fixed-address 172.16.12.2;
    }
    host ring {
        hardware ethernet 00:00:c0:a1:5e:10;
        fixed-address 172.16.3.16;
    }
}
```

Это пример настройки сервера, объединяющего и обслуживающего две различные подсети. Сервер динамически назначает IP-адреса DHCP-клиентам обеих подсетей и осуществляет поддержку нескольких клиентов BOOTP. Все строки, начинающиеся символом решетки (#), являются комментариями. Первые несколько активных строк файла определяют ряд параметров и режимов, действующих для всех обслуживаемых сервером подсетей и клиентов. Первые три строки содержат параметры для сервера. Каждый из трех параметров в данном примере определяет один из аспектов работы `dhcpd` при динамическом назначении адресов.

`default-lease-time`

Указывает серверу длительность аренды адреса по умолчанию (в секундах). Клиент может запросить аренду адреса на определенный период времени, и в этом случае его запрос выполняется. Однако могут действовать определенные ограничения. Часто клиенты не указывают конкретную длительность аренды при получении адреса. В такой ситуации используется параметр `default-lease-time`. В нашем примере длительность аренды по умолчанию составляет один день (86 400 секунд).

`max-lease-time`

Определяет максимально допустимое время аренды. Независимо от длительности аренды, фигурирующей в запросе клиента, это самый длительный срок, на который `dhcpd` выдаст клиенту адрес. Время аренды определяется в секундах. В данном примере максимальное время аренды составляет одну неделю.

`get-lease-hostnames`

Предписывает `dhcpd` предоставлять каждому клиенту наряду с динамически назначенным адресом имя узла. Имя узла должно быть получено в DNS. Данный параметр – логический. При значении `false` (принятом по умолчанию) клиенту назначается адрес, но не имя узла. Поиск имен узлов для всех динамически выделяемых адресов значительно увеличивает время, требуемое на запуск демона. Присваивайте этому параметру значение `false`. Значение `true` используйте, только если сервер работает с очень небольшим количеством адресов, назначаемых динамически.

В файле настройки содержатся и другие параметры – их мы изучим в ходе дальнейшего изложения. Полный перечень всех параметров DHCP приведен в приложении D.

Следующие четыре строки представляют ключи настройки. Они начинаются с ключевого слова `option`. За ним следует имя ключа и назначенное ему значение. Ключи определяют значения настройки, используемые клиентом.

О назначении ключей из примера легко догадаться, поскольку их имена носят описательный характер. Мы передаем клиентам маску подсети, имя домена, адреса сервера доменных имен, а также адрес сервера печати. Эти значения аналогичны тем, что могли распространяться посредством более старой службы BOOTP.

Однако возможности DHCP шире, чем у BOOTP. Чтобы проиллюстрировать это утверждение примером, мы также указали максимальную длину передаваемого блока – MTU (maximum transmission unit). В примере ключ `interface-mtu` уведомляет клиента, что значение MTU равно 1500 байт. В данном случае это избыточное действие (поскольку значение 1500 принимается по умолчанию для интерфейсов Ethernet), но оно ясно показывает, что DCHP позволяет передавать клиентам самую разнообразную информацию о настройках.

Операторы `subnet` определяют подсети, обслуживаемые `dhcpd`. Идентификация сетей выполняется на основе адреса и адресной маски – оба элемента являются обязательными для оператора `subnet`. `dhcpd` передает информацию о настройках только клиентам этих сетей. Для каждой подсети, с которой физически связан сервер, должен присутствовать отдельный оператор `subnet`, даже если в некоторых из подсетей нет клиентов. Информация о подсетях требуется `dhcpd` при запуске.

Действие ключей и параметров внутри оператора `subnet` распространяется только на эту подсеть и находящиеся в ней клиенты. Назначение ключей в примере понятно. Они указывают клиентам, какой маршрутизатор и широковещательный адрес использовать. Параметр `range` более интересен, поскольку напрямую связан с одной из ключевых возможностей DHCP.

Параметр `range` определяет диапазон адресов для динамического выделения. Он всегда существует в контексте оператора `subnet`, и диапазон адресов должен принадлежать адресному пространству этой подсети. Диапазон в параметре `range` определяется парой указанных адресов. Первый адрес определяет наименьший адрес, который может быть назначен автоматически, а второй – наибольший из таких адресов. Первый параметр `range` нашего примера обозначает непрерывную группу адресов с 172.16.12.50 по 172.16.12.250, доступных для динамического назначения. Обратите внимание, что во втором операторе `subnet` два параметра `range`. Так создаются раздельные последовательности динамически выделяемых адресов. Когда это может пригодиться? К примеру, если некоторые из адресов уже были назначены вручную до установки сервера DHCP. Главное, что независимо от причин можно определять области динамически выделяемых адресов, содержащие разрывы, при помощи нескольких операторов `range`.

Если параметр range фигурирует в операторе subnet, любой клиент DHCP из этой подсети, запросивший адрес, получит один из адресов диапазона, если свободные адреса есть. В отсутствие параметра range механизм динамического назначения адресов не действует.

Чтобы включить автоматическое назначение адресов клиентам BOOTP, добавьте аргумент dynamic-bootp в параметр range. Например, так:

```
range dynamic-bootp 172.16.8.10 172.16.8.50;
```

По умолчанию клиентам BOOTP назначаются постоянные адреса. Поведение системы можно изменить при помощи параметра dynamic-bootp-lease-cutoff либо при помощи параметра dynamic-bootp-lease-length. Однако клиенты BOOTP не умеют арендовать адреса и не знают, что должны их обновлять. Поэтому параметры dynamic-bootp-lease-cutoff и dynamic-bootp-lease-length используются только в особых обстоятельствах. Если вам нужны именно эти параметры, обратитесь к приложению D.

Каждому из клиентов BOOTP должен быть сопоставлен оператор host, позволяющий указать параметры и ключи настройки для клиента. Он может использоваться для назначения клиентам постоянных, фиксированных адресов. Приведенный выше файл настройки заканчивается тремя операторами host – для узлов *24seven*, *rodent* и *ring* соответственно. Каждый оператор host содержит параметр hardware, определяющий тип сетевого аппаратного обеспечения (ethernet) и физический сетевой адрес (например, 08:80:20:01:59:c3), используемый клиентом. Параметр hardware является обязательным для операторов host клиентов BOOTP. Адрес Ethernet используется dhcpd для идентификации клиентов BOOTP. Клиентам DHCP также могут соответствовать операторы host. Для клиентов DHCP параметр hardware является необязательным, поскольку идентификацию клиента DHCP можно осуществить по значению dhcp-client-identifier. Однако клиент DHCP, подключенный по интерфейсу Ethernet, проще идентифицировать по его адресу Ethernet.

Оператор host может содержать широкий спектр различных параметров и ключей. Например, добавление в каждый оператор host ключа следующего типа назначает каждому клиенту определенное имя узла:

```
option host-name 24seven;
```

Часто оказывается проще определять значения параметров и ключей на более высоком уровне. Глобальные настройки действуют для всех систем. Настройки подсети действуют для всех ее клиентов, а настройки, принадлежащие оператору host, действуют лишь для одного узла. Ключ host-name, приведенный выше, необходимо повторить с различными именами узлов в каждом операторе host. Проще определить параметр или ключ для группы узлов, воспользовавшись оператором group.

Оператор group группирует любые другие операторы. Единственным назначением оператора group является распространение действия параметров и ключей на всех участников группы. Именно это мы сделали в примере. Оператор group объединил все операторы host. Параметр use-host-decl-names в

операторе `group` действует на все узлы в группе. Этот параметр предписывает `dhcpd` присвоить каждому клиенту имя узла, указанное в операторе `host` клиента, что делает ключ `host-name` ненужным в данном файле настройки.

Исходя из приведенного выше файла *dhcpd.conf*, мы знаем, что, когда `dhcpd` получает запрос от клиента с адресом Ethernet 08:80:20:01:59:c3, клиенту передается следующая информация:

- Адрес 172.16.12.2
- Имя узла *rodent*
- Адрес маршрутизатора по умолчанию 172.16.12.1
- Широковещательный адрес 172.16.12.255
- Маска подсети 255.255.255.0
- Имя домена *wrotethebook.com*
- Адреса серверов имен домена: 172.16.12.1 и 172.16.3.5
- Адрес сервера печати 172.16.12.1
- Значение MTU для интерфейса Ethernet

Клиент получает все глобальные значения, все значения своей подсети, а также все значения, определенные только для этого узла. Очевидно, DHCP позволяет обеспечить клиент полноценными настройками.

Ваши настройки DHCP вероятно проще, чем в приведенном примере, хотя реализуют поддержку большего числа систем. Некоторые из команд включены в пример только ради наглядности. Основное различие заключается в том, что на большинстве площадок единственный сервер настройки не обслуживает более одной подсети. В каждой подсети существует свой сервер: это сокращает нагрузку на сервер, в особенности нагрузку, вызванную неполадками с электропитанием, затронувшими сеть в целом; снимает необходимость передавать загрузочные пакеты через маршрутизаторы. Кроме того, распределение адресов на уровне подсети делает более логичным и размещение сервера на том же уровне. В следующем разделе мы рассмотрим вопрос обновлений для распределенных серверов.

Управление распределенными серверами

В крупной сети существует достаточно много серверов. Как говорилось выше, серверы часто распределены по сети – и в каждой подсети существует сервер. Такая архитектура повышает эффективность работы сети, но противоречит общей цели централизации управления настройкой. Чем больше в сети серверов, тем более рассредоточено управление и тем более вероятно возникновение ошибок в настройках. Работа с распределенными серверами требует механизма централизованного управления и распространения информации среди серверов. В TCP/IP таких механизмов существует несколько.

Для переноса настроек с центральной системы на группу распределенных систем может использоваться любой протокол передачи файлов. Подойдет

FTP или TFTP, однако подобное использование протоколов связано с некоторыми сложностями. FTP и TFTP – протоколы диалоговые, они требуют выполнения многих команд для получения файла, что затрудняет написание соответствующих сценариев. Кроме того, FTP требует парольной идентификации для получения доступа к файлам, а большинство специалистов по безопасности реагируют на хранение паролей в сценариях недовольными гримасами. По этим причинам мы не будем рассматривать эти протоколы в качестве средства распространения файлов настройки. Кроме того, если вы знакомы с FTP (а вы должны быть знакомы с этим протоколом!), то знаете, как использовать его для передачи файла настройки.

Второй вариант – использовать для распространения информации NFS. NFS позволяет клиентам использовать файлы с сервера так, словно это локальные файлы. NFS – очень мощный инструмент, однако в качестве средства распространения информации, необходимой для загрузки серверов, он имеет определенные ограничения. Неполадки с электропитанием, влияющие на распределенные серверы, могут стать причиной сбоя и центрального сервера. Загрузка распределенных серверов и их клиентов будет отложена до того времени, пока центральный сервер не начнет нормальную работу в сети. Организация доступа к единственной копии файла настройки противоречит усилиям, направленным на распределение служб загрузки, поскольку возникает слишком большая зависимость от центрального сервера.

Одним из способов избежать проблемы является периодическое копирование распределенными серверами файла настройки из смонтированной файловой системы на локальный диск. Это решение легко автоматизировать, но оно создает возможную ситуацию, когда распределенные серверы в определенные моменты времени будут отставать в синхронизации – копируя файл настройки по расписанию, а не тогда, когда изменилась основная копия файла. Есть, разумеется, и иной подход: все удаленные серверы экспортируют файловые системы, монтируемые центральным сервером. Тогда центральный сервер может копировать файл настройки непосредственно в удаленные файловые системы, когда изменяется основная копия файла. Тем не менее существуют и более простые пути.

`r`-команды Unix `rcp` и `rdist` реализуют наиболее популярные методы распространения файлов настройки.

rcp

Программа удаленного копирования (*remote copy*, `rcp`) – это обычный протокол передачи файлов. Применительно к поставленной задаче она имеет два преимущества перед FTP: легко позволяет автоматизировать процессы и не требует использования паролей. Автоматизация с `rcp` проста потому, что для передачи файла требуется одна-единственная строка. В следующем примере выполняется передача файла *dhcpd.conf* с головного сервера на удаленный сервер *arthropod.wrotethebook.com*:

```
# rcp /etc/dhcpd.conf arthropod.wrotethebook.com:/etc/dhcpd.conf
```

Для каждого удаленного сервера, которому необходимо передать файл, добавьте подобную строку в процедуру обновления основной копии файла настройки.

`rcp` – лишь один из вариантов решения задачи о распространении основной копии файла настройки. Программа `rdist`, будучи не столь простой в применении, часто оказывается более предпочтительным выбором, поскольку имеет ряд возможностей, делающих ее особенно подходящей для решения подобных задач.

rdist

Программа удаленного распространения файлов (*Remote File Distribution Program*, `rdist`) задумывалась как средство сохранения идентичности файлов, хранимых на нескольких узлах. Единственная команда `rdist` позволяет передать несколько различных файлов на ряд узлов. В основе действий команды лежат инструкции, хранимые в файле настройки `rdist` – *Distfile*.

Назначение *Distfile* аналогично назначению файла *Makefile*, используемого совместно с командой `make`, а синтаксис и структура этих файлов схожи. Стоп, без паники! Все не настолько плохо. Начальная настройка для команды `rdist` сложнее, чем простой синтаксис команды `rcp`, однако `rdist` обеспечивает лучшее управление и в долгосрочной перспективе проще в сопровождении.

Файл *Distfile* состоит из *макроопределений* и *примитивов*. Макроопределению может быть назначено одно значение или список значений. Список значений заключается в скобки, например `macro = ( значение значение )`. После того как значение присвоено, можно обратиться к макроопределению, используя синтаксис `${macro}`, где `macro` – имя макроопределения. Примитивы описаны в табл. 9.4.[1]

Таблица 9.4. Примитивы rdist

Примитив	Назначение
install	Рекурсивно обновлять файлы и каталоги
notify *address*	Передать почтовое сообщение, уведомляющее об ошибке или состоянии, по указанному адресу (*address*)
except *file*	Исключает указанный файл из обновления
except_pat *pattern*	Исключает из обновления файлы, имена которых соответствуют шаблону (*pattern*)
special «*command*»	Исполняет указанную команду после обновления каждого файла

Проще всего понять, как сочетаются макроопределения и примитивы в рабочем файле *Distfile*, взглянув на пример. Следующий файл настройки

[1] Более подробная информация содержится на страницах руководства (man) по `rdist`.

распространяет текущую версию dhcpd и наиболее свежий файл настройки *dhcpd.conf*, копируя их на удаленные серверы *horseshoe*, *arthropod* и *limulus*:

```
HOSTS = ( horseshoe root@limulus arthropod )
FILES = ( /usr/sbin/dhcpd /etc/dhcpd.conf )

${FILES} -> ${HOSTS}
       install ;
       notify craig@crab.wrotethebook.com
```

Рассмотрим каждую из строк файла:

```
HOSTS = ( horseshoe root@limulus arthropod )
```

Эта строка содержит HOSTS, макроопределение, включающее имя каждого из удаленных серверов. Обратите внимание на запись, соответствующую узлу *limulus*. В ней содержится указание rdist выполнять обновление *limulus* в качестве пользователя *root*. На *horseshoe* и *arthropod* rdist выполняется с правами того пользователя, которому принадлежит на локальном узле.

```
FILES = ( /usr/sbin/dhcpd /etc/dhcpd.conf )
```

Макроопределение FILES перечисляет файлы, которые необходимо синхронизировать.

```
${FILES} -> ${HOSTS}
```

Последовательность символов -> имеет специальное значение для rdist. Программа rdist должна скопировать файлы, перечисленные в левой части выражения, на узлы, перечисленные в правой части. В этом случае FILES – макроопределение, содержащие имена файлов */usr/sbin/dhcpd* и */etc/dhcpd.conf*, а HOSTS – макроопределение, содержащее имена узлов *horseshoe*, *limulus* и *arthropod*. Таким образом, данная команда предписывает rdist скопировать два файла на три различных узла. Действие всех последующих примитивов распространяется на это отображение файлы-узлы.

```
install ;
```

Примитив install явным образом предписывает rdist скопировать указанные файлы на указанные узлы, если на удаленном узле какой-либо из файлов устарел. Файл считается устаревшим, если его дата создания или размер не совпадают с соответствующими характеристиками основной копии. Точка с запятой в конце строки показывает, что следом идет еще один примитив.

```
notify craig@crab.wrotethebook.com
```

Сообщения о состоянии и ошибках должны передаваться по электронной почте на адрес *craig@crab.wrotethebook.com*.

Дополнительные файлы и узлы могут легко добавляться в этот файл настройки. В итоге большинство приходит к выводу, что rdist дает простейший способ распространения нескольких файлов по многим узлам.

И последнее замечание: файл настройки не обязательно называть именем *Distfile*. В командной строке rdist при помощи ключа -f может указываться

любое имя. Таким образом, приведенный выше файл можно сохранить под именем *dhcp.dist* и использовать в такой команде:

```
% rdist -f dhcp.dist
```

Серверы почтовой службы

В данном разделе мы займемся настройкой системы в целях создания *сервера почтовой службы*. Сервер почтовой службы, или почтовый сервер, – это компьютер, который хранит почту для компьютера-клиента – до тех пор, пока клиент не будет готов получить ее и прочитать в специальной почтовой программе. Такого рода служба очень важна для поддержки мобильных пользователей и небольших систем, не находящихся в сети постоянно. Последние часто не способны получать почту в реальном времени. Мы рассмотрим два механизма создания почтового сервера: протокол POP (Post Office Protocol), который изначально использовался для этих целей, и протокол IMAP (Internet Message Access Protocol), представляющий собой популярную альтернативу. Начнем с протокола POP.

Сервер POP

Компьютер под управлением Unix превращается в сервер протокола почтовой службы, если на нем работает демон POP. Обратитесь к документации по своей системе, чтобы определить, входит ли демон POP в комплект поставки. Если из документации это определить невозможно, обратитесь к файлу *inetd.conf* или *xinetd.conf* либо проведите простой эксперимент при помощи `telnet` (см. описание в главе 4). Если сервер отзывается на команду `telnet`, значит, демон не только присутствует в системе, он установлен и готов к работе.

```
% telnet localhost 110
Trying 127.0.0.1 ...
Connected to localhost.
Escape character is ' ]'.
+OK POP3 crab Server (Version 1.004) ready
quit
+OK POP3 crab Server (Version 1.001) shutdown
Connection closed by foreign host.
```

Данный пример относится к системе, в поставку которой входит готовый к работе демон POP3. В Red Hat Linux существует поддержка POP3, но для работы с ней необходимо включить ее в файле */etc/xinetd.d/pop3*. В системе Solaris, с другой стороны, POP2 и POP3 по умолчанию отсутствуют. Не волнуйтесь, если в системе нет такого программного модуля. Программное обеспечение POP3 доступно на ряде сайтов и обычно представлено архивом *popper17.tar* или *pop3d.tar*. Я пользовался и тем и другим вариантом, оба работают отлично.

Если в системе отсутствует POP3, загрузите исходные тексты. Извлеките их при помощи команды `tar`. Из архива *pop3d.tar* создается подкаталог с име-

нем *pop3d* в текущем каталоге, однако того же нельзя сказать о *popper17.tar*. Приняв решение воспользоваться демоном `popper`, создайте новый каталог и поместите в него архив, прежде чем выполнять команду `tar`. Внесите в *Makefile* изменения, необходимые для вашей системы, и выполните `make`, чтобы скомпилировать модуль демона POP3. Если компиляция прошла без ошибок, установите демон в один из системных каталогов.

В системе Solaris POP3 запускается демоном интернет-служб `inetd`. Чтобы запустить POP3 под управлением `inetd`, поместите следующую строку в файл *inetd.conf*:

```
pop3   stream  tcp     nowait  root    /usr/sbin/pop3d             pop3d
```

Данная запись предполагает, что используется демон `pop3d`, его исполняемый файл находится в каталоге */usr/sbin*, а порт демона указан в файле */etc/services* и связан с именем `pop3`. Если какое-либо из утверждений неверно, отредактируйте запись соответствующим образом.

Убедитесь, что определение службы POP3 действительно присутствует в файле */etc/services*. Если это не так, добавьте в файл строку:

```
pop3          110/tcp              # Post Office Version 3
```

Когда все нужные строки добавлены в файлы *services* и *inetd.conf*, небходимо послать демону `inetd` сигнал SIGHUP, вызывающий чтение новых настроек, как показано в следующем примере:

```
# ps -ef | grep inetd
  root  109  1  0   Jun 09 ?   0:01 /usr/sbin/inetd -s
# kill -HUP 109
```

Теперь, когда служба POP3 установлена, повторите тест с командой `telnet localhost pop3`. Если демон POP3 отвечает, дело сделано. Теперь все пользователи, имеющие действующие учетные записи в системе, могут получать почту по протоколу POP3 или читать почтовые сообщения непосредственно на сервере.

Сервер IMAP

Протокол IMAP (Internet Message Access Protocol) представляет собой альтернативу протоколу POP. Он предоставляет все те же базовые возможности, что и POP, и дополнительную функциональность, связанную с *синхронизацией почтовых ящиков*, позволяющей читать почту на машине клиента или на сервере. Почтовые ящики клиента и сервера при этом содержат все последние обновления. Для типичного сервера POP содержимое почтового ящика полностью передается клиенту и удаляется с сервера либо сохраняется как непрочитанное. Удаление отдельных сообщений на стороне клиента не отражается на сервере, поскольку все сообщения обрабатываются единым блоком, который при первой передаче клиенту сохраняется либо удаляется. IMAP позволяет работать с отдельными сообщениями в системе клиент либо на сервере. Изменения отражаются в почтовых ящиках обеих систем.

IMAP – протокол достаточно старый; его возраст сравним с возрастом POP3. Известны четыре самостоятельные версии протокола: IMAP, IMAP2, IMAP3 и современная IMAP4, определенная документом RFC 2060. Популярность IMAP объясняется значимостью электронной почты как средства сообщения. Эта значимость сохраняется, даже когда люди покидают офис, так что существует необходимость в почтовых ящиках, с которыми можно работать откуда угодно.

IMAP не поставляется в составе Solaris 8. Исполняемые файлы IMAP для Solaris доступны по адресу *http://sunfreeware.com*. Исходные тексты IMAP доступны для анонимного FTP-копирования на сервере *ftp.cac.washington.edu*. Скопируйте файл */mail/imap.tar.Z* с сервера *ftp.cac.washington.edu* в режиме передачи двоичных образов. Распакуйте и извлеките файлы из архива. Полученный каталог содержит исходные тексты и файл сборки *Makefile*.[1] Внимательно прочитайте файл *Makefile*. Файл сборки реализует поддержку многих вариантов систем Unix. Если ваша система упомянута в списке, воспользуйтесь соответствующим трехсимвольным типом операционной системы из списка. Для системы Solaris и компилятора `gcc` наберите:

```
# make gso
```

Если компиляция прошла без ошибок, что справедливо для системы Solaris, создается следующий набор демонов: `ipop2d`, `ipop3d` и `imapd`. С установкой POP3 мы уже знакомы. Осталось разобраться с `imapd`. Установите его в файл */etc/services*:

```
imap      143/tcp       # IMAP version 4
```

И добавьте его к */etc/inetd*:

```
imap  stream  tcp  nowait  root  /usr/sbin/imapd  imapd
```

Теперь базовая служба IMAP доступна всем пользователям, имеющим действующие учетные записи на сервере.

Приятная особенность пакета Вашингтонского университета – он включает реализации протоколов POP2 и POP3 наряду с IMAP. Это важно, поскольку многие почтовые клиенты работают с POP3. Доступ к серверу IMAP может получить только IMAP-клиент. Установка POP3 вместе с IMAP дает возможность работать с полным диапазоном клиентов.

В состав большинства систем Linux входит IMAP, поэтому компиляция не является обязательным шагом. Достаточно убедиться, что служба упомянута в файле */etc/services* и доступна через `inetd` или `xinetd`. В Red Hat Linux 7 файл */etc/xinetd.d/imap* по умолчанию не используется. Чтобы клиенты получили доступ к службе, файл необходимо активировать.

POP и IMAP – важные компоненты почтовой службы. Тем не менее, как мы увидим в следующей главе, настройка полноценной системы электронной почты требует более серьезных действий.

---

[1] Название каталога отражает номер версии программы. На момент написания книги каталог называется *imap-2001*.

Резюме

В этой главе мы рассмотрели ряд важных служб TCP/IP.

Сетевая файловая система NFS (Network File System) – это ведущий протокол для организации совместного доступа к файлам в системах Unix. В рамках NFS клиенты получают возможность обращаться к отдельным каталогам сервера как к локальным дискам. В целях аутентификации и авторизации в NFS используются механизмы доверенных узлов, идентификаторов UID и GID.

Совместный доступ к Unix-принтерам в сетях TCP/IP организуется при помощи демона LPD (Line Printer Daemon) или сервера LP (Line Printer). Демон `lpd` происходит из системы BSD Unix, но в настоящее время получил широкое распространение. Программа `lpd` извлекает определения принтеров из файла *printcap*. Программа LP родом из Unix стандарта System V. Возможности печатающих устройств она извлекает из файла *terminfo*, а настройки отдельных принтеров – из каталога */etc/lp*. В Solaris 8 совместный доступ к принтерам основан на программном модуле LP, однако настройка принтеров выполняется в единственном файле – */ etc/printers.conf*.

Персональные компьютеры под управлением Windows задействуют NetBIOS и протокол SMB (Server Message Block) для организации совместного доступа к файлам и принтерам. Системы Unix могут выступать в роли серверов SMB при помощи программного пакета Samba. Samba реализует службы совместного доступа к файлам и принтерам, за настройку которых отвечает файл *smb.conf*.

Сетевая информационная служба NIS (Network Information Service) – это сервер, распространяющий ряд административных баз данных. Он позволяет централизованно хранить, изменять и автоматически распространять важные настройки систем.

Расширение протокола BOOTP, протокол динамической настройки узлов DHCP (Dynamic Host Configuration Protocol), обеспечивает клиентов полным набором параметров настройки, определенных в документе RFC *Requirements for Internet Hosts* (Требования к узлам сети Интернет). Кроме того, протокол реализует *динамическое выделение адресов*, позволяющее максимально эффективно использовать ограниченные адресные пространства.

В крупных сетях, для того чтобы не возлагать чрезмерную нагрузку на один сервер и избежать передачи параметров инициализации через IP-маршрутизаторы, используются распределенные серверы загрузки. Файлы настройки на таких серверах синхронизируются посредством передачи файлов, совместного доступа к файлам NFS или при помощи программы `rdist` (Remote File Distribution Program).

Серверы POP (Post Office Protocol) и IMAP (Internet Message Access Protocol) позволяют хранить почтовые сообщения на сервере до тех пор, пока пользователь не будет готов прочесть их. В следующей главе мы более пристально рассмотрим настройку подсистемы электронной почты и изучим программу sendmail.

10

sendmail

- *Назначение sendmail*
- *sendmail в роли демона*
- *Псевдонимы sendmail*
- *Файл sendmail.cf*
- *Язык настройки sendmail.cf*
- *Переписывание почтового адреса*
- *Изменение файла sendmail.cf*
- *Тестирование sendmail.cf*

Электронная почта – заклятый друг пользователей: они обожают ее использовать, но ненавидят, когда она не работает. И задача системного администратора – сделать так, чтобы почта работала. Решению данной задачи и посвящена эта глава.

sendmail – не единственная программа транспортировки почты; популярны также smail и qmail. Однако sendmail – наиболее широко распространенная программа транспортировки. Вся эта глава отведена под рассказ о sendmail, а вообще этой теме можно с легкостью посвятить целую книгу.[1] Отчасти причина кроется в значимости электронной почты, но также и в сложности настройки sendmail.

Как ни странно, сложность sendmail в некоторой степени проистекает из попыток упростить поддержку электронной почты, вместив всю возможную функциональность в одну программу. В определенный момент времени для работы с электронной почтой использовалось большое число программ и протоколов. Такое разнообразие программ затрудняло настройку и сопровождение. Даже сегодня существует несколько четко различимых схем доставки электронной почты. Протокол SMTP осуществляет передачу почты в сетях TCP/IP; другая программа передает почтовые сообщения между пользователями одной системы; и еще одна передает почту между системами сетей UUCP. Каждой из этих почтовых систем – SMTP, UUCP и локальной почте – соответствует собственная программа доставки и схема почтовой адресации. Такое положение дел может приводить в замешательство пользователей электронной почты и системных администраторов.

---

[1] Полноценное описание содержится в книгах: Косталес (Costales) и Оллман (Allman) «sendmail», O'Reilly и Крэйг Хант (Craig Hunt) «Linux Sendmail Administration», Sybex.

Назначение sendmail

sendmail уничтожает путаницу, вызванную большим числом приложений для доставки почты. sendmail самостоятельно передает почту нужной программе доставки, исходя из адреса получателя. sendmail принимает почтовые сообщения от почтовой программы пользователя, интерпретирует почтовый адрес, переписывает его в формате, понятном соответствующей программе доставки, и маршрутизирует почту через эту программу. sendmail изолирует конечного пользователя от подобных деталей работы. Если почта адресована верно, sendmail проконтролирует передачу сообщений для доставки. Так же и для входящей почты – sendmail интерпретирует адрес и доставляет сообщение почтовой программе пользователя либо передает другой системе.

На рис. 10.1 представлена особая роль sendmail в маршрутизации почты между различными почтовыми программами, существующими в системах Unix.

Помимо маршрутизации почты между пользовательскими программами и программами доставки, sendmail имеет следующие функции:

- Получает и доставляет почту SMTP (Internet)
- Реализует работу с общесистемными почтовыми псевдонимами, что позволяет создавать списки рассылки

Настройка системы на корректное выполнение всех этих функций – задача сложная. В этой главе мы разберем каждую из функций, изучим процесс их настройки, а также способы упрощения этой задачи. Прежде всего, мы рассмотрим функционирование sendmail в части получения почты SMTP. Затем вы научитесь использовать почтовые псевдонимы и настраивать sendmail с целью маршрутизации почтовых сообщений на основе почтового адреса.

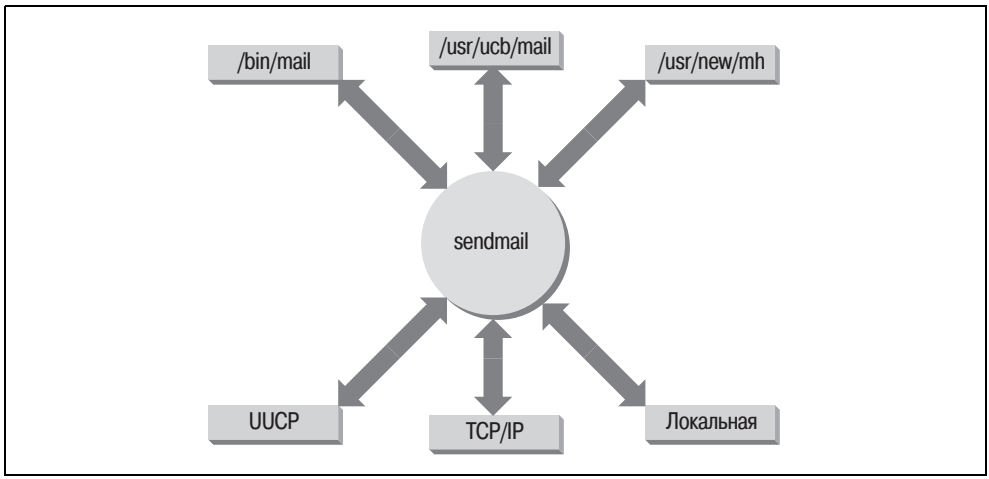


Рис. 10.1. *Маршрутизация почты через sendmail*

sendmail в роли демона

Чтобы получать почту SMTP по сети, запустите sendmail в качестве демона в процессе загрузки системы. Демон sendmail прослушивает порт TCP с номером 25 и обрабатывает входящую почту. В большинстве случаев код, реализующий запуск sendmail, уже существует в одном из загрузочных сценариев. Если это не так, добавьте его. Следующая строка запускает sendmail в качестве демона:

```
# /usr/lib/sendmail -bd -q15m
```

В этой командной строке мы используем два ключа. Ключ `-q` указывает sendmail, как часто следует обрабатывать очередь почтовых сообщений. В данном случае очередь обрабатывается раз в 15 минут (`-q15m`), что является разумным значением для частой обработки очереди. Не стоит слишком занижать это значение. Слишком частая обработка очереди может привести к проблемам, если очередь станет слишком большой из-за сложностей с доставкой – скажем, из-за недоступности сети. Для средней рабочей станции разумным вариантом является обработка очереди раз в час (`-q1h`) или раз в полчаса (`-q30m`).

Второй параметр относится непосредственно к приему почты SMTP. Ключ `-bd` предписывает sendmail работать в качестве демона и принимать входящую почту через порт TCP с номером 25. Используйте этот параметр, если ваша система должна принимать входящую почту TCP/IP.

Вот такая простая командная строка. Большинство системных загрузочных сценариев не столь просты. Они, в общем случае, не ограничиваются запуском. В системе Solaris 8 для запуска sendmail используется сценарий */etc/init.d/sendmail*. Прежде всего, этот сценарий Solaris проверяет наличие каталога почтовой очереди. Если каталог не существует, sendmail создает его. В сценарии Solaris 8 параметры командной строки устанавливаются при помощи переменных. В переменной MODE хранится ключ `-bd`, а в переменной QUEUEINTERVAL – интервал обработки очереди. По умолчанию в сценарии Solaris 8 QUEUEINTERVAL имеет значение `15m`; чтобы изменить интервал обработки очереди, измените значение QUEUEINTERVAL. Не изменяйте значение переменной MODE, если нет необходимости запретить прием входящей почты. Чтобы sendmail работала в качестве демона и принимала входящую почту, эта переменная должна иметь значение `-bd`. Чтобы добавить другие ключи в командную строку sendmail, выполняемую сценарием Solaris 8, поместите эти ключи в значение переменной OPTIONS.

Сценарий Red Hat */etc/rc.d/init.d/sendmail* еще сложнее, чем его версия для Solaris. Он принимает аргументы `start`, `stop`, `restart`, `condrestart` и `status`, что позволяет эффективно использовать сценарий для управления процессом демона sendmail. Аргументы `start` и `stop` пояснений не требуют. Аргумент `restart` приводит к останову работающего процесса sendmail и запуску нового процесса. Аргумент `condrestart` схож по действию с `restart`, с той разницей, что действия выполняются только в случае, если существует работающий процесс sendmail. Если демон sendmail не запущен на момент выполне-

ния сценария с аргументом condrestart, сценарий просто завершает работу. Аргумент status выводит состояние демона, а именно – численный идентификатор процесса, если демон запущен, либо сообщение о том, что sendmail остановлен, если он не запущен.

Когда сценарий Red Hat вызывается с аргументом start, то начинает работу с повторной сборки всех файлов баз данных sendmail. Затем запускается демон sendmail с параметрами командной строки, определенными в файле */etc/sysconfig/sendmail*. Подобно сценарию для Solaris, сценарий Red Hat использует переменные для определения значений параметров командной строки, но сами переменные инициализируются косвенно значениями из файла */etc/sysconfig/sendmail*. Файл */etc/sysconfig/sendmail* системы Red Hat по умолчанию содержит всего две строки:

```
$ cat /etc/sysconfig/sendmail
DAEMON=yes
QUEUE=1h
```

Если DAEMON имеет значение yes, sendmail запускается с ключом -bd. Интервал обработки очереди определяется значением переменной QUEUE. В данном примере очередь почтовых сообщений обрабатывается каждый час (1h). Дополнительный код, присутствующий в большинстве загрузочных сценариев, полезен, однако не является строго необходимым для запуска sendmail в качестве демона. Все что в действительности нужно – командная строка sendmail с ключом -bd.

Псевдонимы sendmail

Практически невозможно преувеличить важность почтовых псевдонимов. Без них система sendmail не смогла бы функционировать в качестве центрального почтового сервера. Почтовые псевдонимы позволяют:

- Создавать альтернативные имена (прозвища) для отдельных пользователей
- Пересылать почту на другие узлы
- Создавать списки рассылки

Почтовые псевдонимы sendmail хранятся в файле *aliases*.[1] Базовый формат записей файла *aliases*:

```
alias: recipient[, recipient,...]
```

alias – это имя, на которое адресуется почта, а *recipient* – имя, по которому выполняется ее доставка. Значение *recipient* может быть именем пользователя, именем другого псевдонима или полноценным адресом электронной почты, содержащим имя пользователя и имя узла. Имя узла в составе адреса позволяет выполнять пересылку почты удаленному узлу. Кроме того, с одним псевдонимом может быть связано любое число получателей. Почта, ад-

[1] Местоположение файла определяется параметром ALIAS_FILE в файле m4-макроопределений для sendmail.

ресованная на этот псевдоним, доставляется всем получателям – фактически так создается список рассылки.

Псевдонимы, определяющие прозвища отдельных пользователей, могут использоваться для работы с именами, в которых люди часто делают ошибки. Псевдонимы могут также использоваться для доставки почты, адресованной специальным именам, таким как *postmaster* или *root*, на адреса реальных пользователей, выполняющих соответствующую работу. Псевдонимы могут использоваться и для создания системы упрощенной почтовой адресации, что особенно эффективно в сочетании с MX-записями.[1] Следующий файл *aliases* с узла *crab* демонстрирует все описанные применения:

```
# специальные имена
postmaster: clark
root: norman
# разрешить адреса firstname.lastname@wrotethebook.com
rebecca.hunt: becky@rodent
jessie.mccafferty: jessie@jerboas
anthony.resnick: anthony@horseshoe
andy.wright: andy@ora
# список рассылки
admin: kathy, david@rodent, sara@horseshoe, becky@rodent, craig,
       anna@rodent, jane@rodent, christy@ora
owner-admin: admin-request
admin-request: craig
```

Первые два псевдонима являются специальными именами. В их присутствии почта, адресованная на имя *postmaster*, доставляется локальному пользователю *clark*, а почта, адресованная пользователю *root*, доставляется пользователю *norman*.

В формате второго набора псевдонимов используется сочетание имени и фамилии. Первый псевдоним этой группы – *rebecca.hunt*. Почта, адресованная на *rebecca.hunt*, передается с узла *crab* и доставляется пользователю *becky@rodent*. Сочетание этого псевдонима с MX-записью, определяющей *crab* в качестве почтового сервера домена *wrotethebook.com*, приводит к доставке почты, адресованной пользователю *rebecca.hunt@wrotethebook.com*, по адресу *becky@rodent.wrotethebook.com*. Такая схема адресации позволяет всем пользователям иметь постоянный почтовый адрес, который не меняется только потому, что учетная запись пользователя перенесена на другой узел. Кроме того, если удаленный пользователь в курсе, что в домене *wrotethebook.com* используется схема адресации имя.фамилия, он может написать письмо Ребекке Хант на адрес *rebecca.hunt@wrotethebook.com*, не зная ее действительного адреса электронной почты.

Последние два псевдонима относятся к списку рассылки. Псевдоним *admin* определяет собственно список. Если почта адресована *admin*, копия сообщения передается каждому из получателей (*kathy*, *david*, *sara*, *becky*, *craig*, *anna*, *jane* и *christy*). Обратите внимание, что список рассылки занимает не-

[1] MX-записи описаны в главе 8.

сколько строк. Строка продолжения должна начинаться пробелом или символом табуляции.

Псевдоним *owner-admin* – специальное имя, обладающее смыслом для sendmail. Формат специального псевдонима: `owner-`*список*, где *список* – имя (название) списка рассылки. Лицо, указанное в строке этого псевдонима, отвечает за указанный список рассылки. Если sendmail сталкивается с проблемами при доставке сообщений любому из получателей списка *admin*, сообщение об ошибке передается по адресу *owner-admin*. Псевдоним *owner-admin* указывает, что за сопровождение списка *admin* отвечает *admin-request*. Псевдонимы в формате *список*-`request` обычно используются в административных запросах, например при подписке на рассылку либо для списков рассылки, сопровождение которых осуществляется вручную. Обратите внимание, что наш псевдоним указывает на другой псевдоним, и это совершенно законно. Псевдоним *admin-request* отображается в *craig*.

sendmail не использует файл *aliases* напрямую. Файл *aliases* необходимо обработать командой `newaliases`. Выполнение `newaliases` эквивалентно вызову `sendmail` с ключом `-bi`, который приводит к созданию базы данных псевдонимов sendmail. `newaliases` создает файлы баз данных, используемые программой sendmail при поиске псевдонимов. Применяйте `newaliases`, когда вносите изменения в файл *aliases*, чтобы новые псевдонимы становились доступны sendmail.[1]

Пересылка личной почты

Помимо пересылки почты на основе псевдонимов, sendmail позволяет пользователям системы создавать собственные правила пересылки. Такие правила пользователь определяет в файле *.forward*, в своем исходном каталоге. sendmail сверяется с этим файлом после обработки по файлу *aliases* и перед окончательной доставкой пользователю. Если файл *.forward* существует, sendmail доставляет почту, следуя правилам из этого файла. Например, предположим, что пользователь *kathy* создала в своем исходном каталоге файл *.forward*, содержащий адрес *kathy@podunk.edu*. Почта, которую в обычной ситуации sendmail доставляет локальному пользователю *kathy*, пересылается в таком случае на учетную запись *kathy* в системе *podunk.edu*.

Используйте файл *.forward* для создания временных правил пересылки. Изменение файла *aliases* и компиляция базы данных требует бульших усилий, нежели изменение файла *.forward*, в особенности если изменение в правилах пересылки носит кратковременный характер. Кроме того, файл *.forward* позволяет пользователю самостоятельно управлять пересылкой своей почты.

Почтовые псевдонимы и пересылка почты управляются файлами *aliases* и *.forward*. Все прочие вопросы, связанные с настройкой sendmail, решаются при помощи файла *sendmail.cf*.

---

[1] Параметр `AutoRebuildAliases` приводит к автоматическому созданию базы данных псевдонимов, даже если не выполнена команда `newaliases`. См. приложение E.

Файл sendmail.cf

Настройки sendmail хранятся в файле *sendmail.cf*.[1] Файл содержит большую часть настроек sendmail, включая сведения, необходимые для маршрутизации почты между пользовательскими почтовыми программами и программами доставки почты. Файл *sendmail.cf* имеет три основных функции:

- Определяет среду sendmail
- Преобразует адреса в формат принимающей почтовой программы
- Связывает адреса с инструкциями, необходимыми для доставки почтовых сообщений

Выполнение всех этих функций требует определенного набора команд. Команды макроопределений и параметров создают среду. Правила подстановки преобразуют адреса электронной почты. Почтовые определения указывают инструкции, выполнение которых позволяет доставлять почту. Из-за излишне лаконичного синтаксиса этих команд системные администраторы весьма неохотно читают файл *sendmail.cf*, не говоря уже о том, чтобы его писать! К счастью, как мы скоро увидим, можно обойтись и без написания собственного файла *sendmail.cf*.

Пример файла sendmail.cf

Едва ли можно назвать вескую причину для создания *sendmail.cf* с нуля. Примеры файлов настройки поставляются в комплекте большинства систем. Некоторые системные администраторы используют стандартный файл настройки *sendmail.cf* и вносят в него небольшие изменения, отражающие локальные потребности настройки. Этот подход к настройке sendmail мы рассмотрим позже в этой главе.

Системные администраторы в общем случае предпочитают использовать исходные файлы `m4` для создания файла *sendmail.cf*. Создание настроек при помощи `m4` рекомендовано разработчиками sendmail как простейший способ создания и сопровождения настроек. Однако в состав некоторых систем исходные файлы `m4` не входят, а даже если и входят, они пригодны для использования совместно только с исполняемым файлом sendmail из дистрибутива системы. После обновления sendmail следует использовать исходные файлы `m4`, совместимые с обновленной версией sendmail. Если вы намереваетесь использовать `m4` или наиболее свежую версию sendmail, загрузите дистрибутив с исходными текстами sendmail с сайта *http://www.sendmail.org*. Пример установки дистрибутива sendmail описан в приложении Е.

Каталог sendmail *cf/cf* содержит несколько примеров файла настройки. В ряде файлов представлены предварительные базовые настройки для раз-

[1] До выхода версии sendmail 8.11 по умолчанию файл настройки хранился в каталоге */etc*. Теперь по умолчанию он хранится в */etc/mail*, однако часто располагается в других каталогах, таких как */usr/lib*.

личных операционных систем. Подкаталог *cf/cf* каталога sendmail.8.11.3 содержит общие настройки для систем BSD, Solaris, SunOS, HP Unix, Ultrix, OSF1 и Next Step, а также несколько файлов прототипов, предназначенных для использования в других операционных системах после внесения небольших изменений. Мы изменим файл *tcpproto.mc*, предназначенный для систем с прямыми подключениями к сетям TCP/IP и без подключений UUCP, в целях использования в нашей системе Linux.

Построение sendmail.cf при помощи макроопределений m4

Файлы прототипов из архива sendmail не готовы к мгновенному применению. Их необходимо отредактировать и обработать макропроцессором `m4`, чтобы получить собственно файлы настройки. Например, файл *tcpproto.mc* содержит такие макроопределения:

```
divert(0)dnl
VERSIONID(`$Id: ch10,v 1.3 2002/03/01 21:02:23 sue Exp emily $')
OSTYPE(`unknown')
FEATURE(`nouucp', `reject')
MAILER(`local')
MAILER(`smtp')
```

Эти макроопределения не являются командами sendmail; они представляют собой исходные данные для макропроцессора `m4`. Приведенные строки файла прототипа активны, их предваряет блок комментариев (не отраженный в листинге), который `m4` игнорирует, поскольку этот блок следует за командой `divert(-1)`, перенаправляющей вывод в «никуда». Данный блок файла начинается командой `divert(0)`, то есть команды обрабатываются, а результаты направляются на стандартный вывод.

Команда `dnl` в конце строки `divert(0)` предотвращает появление в файле результата нежелательных строк. `dnl` удаляет все вплоть до следующего символа новой строки. Она влияет на внешний вид, но не на функциональность конечного файла. `dnl` может фигурировать в конце любой макрокоманды, а также может использоваться в начале строки, и в таком случае строка интерпретируется как комментарий.

Макроопределение VERSIONID используется для управления версиями. Как правило, значение данного макровызова является номером версии в формате RCS (Release Control System) или SCCS (Source Code Control System). Данное макроопределение – необязательное, его можно смело игнорировать.

OSTYPE определяет специфичную для операционной системы информацию настройки. Каталог *cf/ostype* содержит почти 50 файлов макроопределений для различных операционных систем. Макроопределение OSTYPE – обязательное, и передаваемое в его вызове значение должно совпадать с именем одного из файлов упомянутого каталога. Примеры значений: `bsd4.4`, `solaris8`, `linux`.

FEATURE определяет необязательные возможности, включаемые в файл *sendmail.cf*. В данном примере значение `nouucp` указывает, что адреса UUCP

в этой системе не используются. Аргумент `reject` предписывает отвергать локальные адреса в стиле UUCP, то есть содержащие символ `!` в локальной части. Вспомним, что в предшествующем разделе мы описали *tcpproto.mc* как файл прототипа для систем без соединений UUCP. В другом файле прототипа будут иные значения FEATURE.

Файл прототипа завершается макроопределениями почтовой программы. В исходном файле они должны занимать последние места. Приведенный пример содержит макроопределения локальной почтовой программы и почтовой программы SMTP.

В макроопределении MAILER(local) фигурирует почтовая программа *local*, выполняющая доставку локальной почты между пользователями системы, а также программа *prog*, передающая почтовые файлы программам, работающим в системе. Все базовые файлы с макроопределениями содержат определение MAILER(local), поскольку почтовые программы *local* и *prog* реализуют необходимые службы локальной доставки почты.

Определение MAILER(smtp) включает все почтовые программы, необходимые для передачи SMTP-почты по сети TCP/IP. В данный набор включены следующие почтовые программы:

smtp

Обрабатывает традиционную почту SMTP в формате ASCII 7-бит. Программа устарела, поскольку современные почтовые сети умеют работать с разнообразными типами данных.

esmtp

Поддерживает протокол Extended SMTP (ESMTP). Понимает расширения протокола ESMTP и способна работать со сложносоставными сообщениями и более совершенными типами данных MIME-почты. Для SMTP-почты это почтовая программа, используемая по умолчанию.

smtp8

Передает удаленному серверу 8-битные данные, даже если удаленный сервер не подтвердил свою способность принимать такие данные. Обычно сервер, поддерживающий работу с 8-битными данными, поддерживает также ESMTP и способен сообщить о поддержке 8 битов в ответ на команду EHLO. (Описание протокола SMTP и команды EHLO содержится в главе 3.) Однако возможен случай подключения к удаленному серверу, который поддерживает 8-битные данные, но не поддерживает ESMTP. И для таких редких случаев существует почтовая программа smtp8.

dsmtp

Позволяет конечной системе получать почту из очереди на сервере. Обычно система-источник сама является инициатором передачи почты, как бы «выталкивает» ее на конечную систему. dsmtp, напротив, позволяет конечной системе получить почту с сервера по запросу, как бы «вытянуть» ее. Данная почтовая программа реализует команду ETRN, разрешающую доставку по запросу. (Команда ETRN описана в документе RFC 1985.)

relay

Эта почтовая программа используется, если требуется пересылка SMTP-почты через промежуточный почтовый сервер. Узлов промежуточной пересылки почты может быть несколько.

Каждый сервер, подключенный к Интернету или обменивающийся с ним данными, использует набор почтовых программ MAILER(smtp), как и большинство систем изолированных сетей, поскольку в сетях предприятий также используется TCP/IP. Большинство систем sendmail требуют присутствия этих почтовых программ, однако их установка не происходит по умолчанию. Поддержка почты SMTP требует присутствия макроопределения MAILER(smtp) в настройках, почему данное макроопределение и включается в файл прототипа.

Кроме этих двух важных наборов почтовых программ, команда MAILER предоставляет доступ еще к девяти наборам. Описания наборов содержатся в приложении E. Большая их часть не представляет интереса для средней конфигурации. Двумя наборами почтовых программ, включенными в настройки *tcpproto.mc*, ограничивается большинство администраторов.

Чтобы создать пример файла *sendmail.cf* на основе файла прототипа *tcpproto.mc*, скопируйте файл прототипа в рабочий файл. Отредактируйте рабочий файл: необходимо подставить в строку OSTYPE корректное для вашей операционной системы значение вместо `unknown` (к примеру, `solaris8` или `linux`). В следующем примере мы использовали `sed`, чтобы заменить `unknown` на `linux`. Результат мы сохраняем в файле под именем *linux.mc*:

```
# sed 's/unknown/linux/' < tcpproto.mc > linux.mc
```

Теперь выполним команду `m4`:

```
# m4 ../m4/cf.m4 linux.mc > sendmail.cf
```

Файл *sendmail.cf*, результат выполнения команды `m4`, имеет формат, подходящий для чтения программой sendmail. За исключением вопросов обработки адресов UUCP, полученный файл схож с файлом примера *generic-linux.cf* из дистрибутива sendmail.

OSTYPE – не единственный элемент файла макроопределений, поддающийся изменению с целью создания частного случая настройки. Существует большое число параметров настройки, и все они описаны в приложении E. Для примера мы изменим некоторые параметры с целью создания настроек, приводящих к преобразованию почтовых адресов вида *user@host*, исходящих с нашей машины, в формат *firstname.lastname@domain*. С этой целью мы создадим два новых файла настройки: файл макроопределений с конкретными значениями для домена – под именем *wrotethebook.com.m4*, а также измененный файл управления макроопределениями, *linux.mc*, включающий новый файл *wrotethebook.com.m4*.

Файл *wrotethebook.com.m4*, который мы сохранили в каталоге *cf/domain*, содержит такие строки:

```
$ cat domain/wrotethebook.com.m4
MASQUERADE_AS(wrotethebook.com)
```

```
FEATURE(masquerade_envelope)
FEATURE(genericstable)
```

Эти строки указывают, что мы хотим скрыть действительное имя узла и для исходящих писем в адресе электронной почты заменять его на *wrotethebook.com*. Помимо адресов из заголовков сообщений, мы предписываем то же действие для адресов «конвертов» (envelope). Первые две строки отвечают за преобразование имени узла в электронных адресах исходящих сообщений. Последняя строка указывает, что мы намереваемся использовать базу данных преобразования адресов, отображающую регистрационные имена пользователей в любые значения, которые мы сочтем подходящими для пользовательской части адреса. Мы должны сгенерировать базу данных: создать текстовый файл с необходимыми данными и обработать его при помощи команды `makemap` из дистрибутива sendmail.

Формат базы данных может быть очень простым:

```
dan Dan.Scribner
tyler Tyler.McCafferty
pat Pat.Stover
willy Bill.Wright
craig Craig.Hunt
```

Каждая строка данного файла состоит из двух полей: первое поле содержит ключ, то есть регистрационное имя пользователя, а второе поле – имя и фамилию пользователя, разделенные точкой. В качестве разделителей полей выступают пробелы. Запрос по имени `dan` для этой базы возвращает результат `Dan.Scribner`. Небольшую базу данных вроде этой можно легко создать вручную. В системе с большим числом пользовательских учетных записей процесс лучше автоматизировать извлечением регистрационного имени, имени и фамилии пользователя из файла */etc/passwd*. Поле `gcos` файла */etc/passwd* часто содержит настоящее имя пользователя.[1] После сохранения данных в текстовом файле их необходимо преобразовать в базу данных посредством команды `makemap`. Программа `makemap` входит в состав дистрибутива sendmail. Ее синтаксис выглядит следующим образом:

```
makemap type name
```

`makemap` читает стандартный ввод и записывает базу данных в файл `name`. Поле *type* определяет тип базы данных. Наиболее распространенные типы баз данных для sendmail: `dbm`, `btree` и `hash`.[2] Базы данных всех этих типов могут создаваться при помощи команды `makemap`.

---

[1] Пример сценария, создающего базу данных *realnames* на основе */etc/passwd*, содержится в приложении Е.

[2] В системах Solaris карты NIS и таблицы NIS+ создаются при помощи стандартных команд из состава операционной системы. В sendmail используется иной синтаксис для работы с этими картами (см. табл. 10.3).

Предположим, что приведенные выше данные мы сохранили в файле *realnames*. Следующая команда преобразует этот файл в базу данных:

```
# makemap hash genericstable < realnames
```

`makemap` читает текстовый файл и создает файл базы данных с именем *genericstable*. Эта база данных связывает регистрационные имена пользователей с их настоящими именами, то есть по ключу `willy` возвращается значение `Bill.Wright`.

После создания базы данных следует создать файл настройки, где она будет задействована. Все макроопределения, связанные с использованием базы данных, хранятся в файле *wrotethebook.com.m4*. Нам необходимо включить этот файл в настройки. С этой целью мы добавим строку `DOMAIN(wrotethebook.com)` в управляющий файл (*linux.mc*), а затем обработаем файл *linux.mc* процессором `m4`. Следующая команда `grep` показывает, как выглядят макроопределения в файле после внесения изменений:

```
# grep '^[A-Z]' linux.mc
VERSIONID(`$Id: ch10,v 1.3 2002/03/01 21:02:23 sue Exp emily $')
OSTYPE(`linux')
DOMAIN(`wrotethebook.com')
FEATURE(`nouucp', `reject')
MAILER(`local')
MAILER(`smtp')
# m4 ../m4/cf.m4 linux.mc > sendmail.cf
```

При установке sendmail из `tar`-архива используйте *mc*-файл прототипа в качестве отправной точки настройки. Чтобы использовать наиболее актуальную версию sendmail, необходимо создать совместимый файл *sendmail.cf* при помощи макроопределений `m4`. Не пытайтесь использовать старый файл *sendmail.cf* с новой версией sendmail; этим вы только наживете неприятности. Как можно видеть из приведенных выше примеров, файлы настройки `m4` весьма лаконичны и могут состоять из небольшого числа макроопределений. Используйте `m4` для создания новых настроек при каждом обновлении sendmail.

И наоборот, не следует использовать *sendmail.cf*, созданный на основе файлов прототипов из дистрибутива sendmail старой версии. Возможности в этих файлах требуют присутствия совместимой версии sendmail, а значит – перекомпиляции sendmail при необходимости использовать новый файл настройки.[1] На такую операцию решится далеко не всякий системный администратор, поскольку в некоторых системах нет нужных библиотек, в других и вовсе не установлен компилятор языка C. Если вы отказываетесь от повторной компиляции sendmail, то можете использовать в качестве отправной точки стандартный файл примера *sendmail.cf* из собственной системы. Однако если запланированы серьезные изменения в настройках, вероятно,

[1] Информация о компиляции sendmail содержится в приложении E.

легче перекомпилировать sendmail и создать новые настройки при помощи m4, чем вносить серьезные изменения непосредственно в файл *sendmail.cf*.

В следующей части главы мы воспользуемся одним из примеров файла *sendmail.cf* из дистрибутива Linux. Если точнее, мы начнем с файла *generic-linux.cf*, расположенного в каталоге *cf/cf*, дистрибутива sendmail. Все сведения, приводимые на протяжении оставшейся части главы, применимы в равной степени и к файлам *sendmail.cf*, созданным при помощи m4. Структура файла *sendmail.cf*, хранимых в нем команд, а также инструменты его отладки одинаковы во всех случаях.

Общее строение файла sendmail.cf

Структура большинства файлов *sendmail.cf* более-менее одинакова, поскольку они создаются на основе стандартных определений m4. Поэтому файлы из вашей системы, скорее всего, схожи с теми, что фигурируют в наших примерах. В некоторых системах структура может отличаться, но функциональность описанных здесь разделов так или иначе присутствует в большинстве файлов *sendmail.cf*.

Файл Linux *generic-linux.cf* послужит нам примером для изучения структуры файла *sendmail.cf*. Названия разделов из файла примера использованы ниже, в обзоре структуры *sendmail.cf*. Эти разделы будут описаны более подробно, когда мы займемся изменением настроек. Итак, разделы:

Локальная информация (Local Information)

Определяет информацию, специфичную для отдельного узла. В файле *generic-linux.cf* раздел Local Information содержит определения имени узла, имен узлов пересылки почты, а также почтового домена. В этом же разделе определено имя, которым представляется sendmail при передаче сообщений об ошибках, сообщение, отображаемое sendmail при SMTP-регистрации, а также номер версии файла *sendmail.cf*. (Увеличивайте номер версии каждый раз, когда вносите изменения в настройки.) Этот раздел обычно подвергается изменениям при настройке.

Параметры (Options)

Определяет параметры sendmail. Обычно не требует изменений.

Старшинство сообщений (Message Precedence)

Определяет значения старшинства для различных сообщений, используемые sendmail. Не подвергается изменениям.

Доверенные пользователи (Trusted Users)

Определяет пользователей, которым разрешено переопределять адрес отправителя при отправке почты. Не подвергается изменениям. Добавление пользователей в этот раздел – потенциальная угроза безопасности.

Формат заголовков (Format of Headers)

Определяет формат заголовков, добавляемых sendmail к сообщениям. Не подвергается изменениям.

Правила подстановки (Rewriting Rules)

Определяет правила перезаписи почтовых адресов. Раздел содержит общие правила перезаписи, используемые sendmail, а также другие правила подстановки. В ходе начальной настройки sendmail данный раздел не подвергается изменениям. Правила подстановки обычно редактируются только с целью разрешения проблемы или добавления новой службы.

Определения почтовых программ (Mailer Definitions)

Определяет инструкции, используемые sendmail для вызова программ доставки почты. Также раздел содержит правила подстановки, связанные с конкретными почтовыми программами. Определения почтовых программ обычно не подвергаются изменениям, однако связанные с ними правила подстановки могут редактироваться в целях разрешения проблем или добавления новых служб.

Метки заголовков в файле примера из вашей системы могут отличаться от приводимых здесь, но структура файла, скорее всего, схожа с описанной в следующих аспектах:

- Информация частной настройки узла, вероятно, располагается в начале файла.
- Команды одного типа (настройки параметров, заголовков и т. д.) обычно группируются.
- Большую часть файла составляют правила подстановки.
- Последняя часть файла, вероятно, содержит определения почтовых программ вперемешку с правилами подстановки для отдельных почтовых программ.

Изучите комментарии своего файла *sendmail.cf*. Иногда эти комментарии содержат сведения, позволяющие проникнуть в структуру файла и разобраться с элементами настройки системы.

Важно осознавать, насколько малы изменения, вносимые в *sendmail.cf* при настройке средней системы. Выбрав правильный исходный файл примера, вы сможете обойтись изменением всего нескольких строк в первом разделе. С этой точки зрения настройка sendmail начинает казаться примитивной задачей. Так почему же она пугает системных администраторов? Преимущественно из-за сложного синтаксиса языка настройки *sendmail.cf*.

Язык настройки sendmail.cf

При каждом запуске sendmail выполняет чтение файла *sendmail.cf*. По этой причине синтаксис команд *sendmail.cf* разработан с целью облегчения их разбора программой, но совсем не обязательно людьми. Как следствие, команды sendmail очень кратки, даже по стандартам Unix.

Команда настройки и ее переменные части или аргументы не разделяются пробелами. Такой «потоковый» формат затрудняет чтение команд. Формат

***Рис. 10.2.** Команда настройки sendmail.cf*

команды иллюстрирует рис. 10.2. Команда макроопределения присваивает значение *wrotethebook.com* макропеременной D.

Начиная с sendmail версии 8 длина имен переменных перестала ограничиваться одним символом. Теперь допустимы длинные имена переменных, заключенные в фигурные скобки. Например, макроопределение define из рис. 10.2 можно записать так:

```
D{Domain}wrotethebook.com
```

Длинные имена переменных проще для чтения и предоставляют больший выбор, чем ограниченный набор односимвольных имен. При этом старомодные, краткие имена переменных по-прежнему в ходу. Столь краткий синтаксис может быть весьма сложно расшифровать, но полезно помнить, что первый символ в строке всегда является командой. Он позволяет определить, о какой команде идет речь, а значит, и какова структура команды. В табл. 10.1 описаны команды *sendmail.cf* и их синтаксис.

Таблица 10.1. Команды настройки sendmail

Команда	Синтаксис	Назначение
Version Level	V*level*[*/vendor*]	Указывает уровень версии
Define Macro	D*x value*	Устанавливает *x* в значение *value*
Define Class	C*cword1*[*word2*] …	Связывает класс *c* со значениями: *word1 word2…*
Define Class	F*cfile*	Загрузка класса *c* из файла *file*
Set Option	O*option=value*	Устанавливает для параметра *option* значение *value*
Trusted Users	T*user1*[*user2* …]	Указывает доверенных пользователей: *user1 user2…*
Set Precedence	P*name=number*	Устанавливает для имени *name* значение приоритета *number*
Define Mailer	M*name*, {*field=value*}	Определяет имя (*name*) почтовой программы
Define Header	H[?*mflag*?]*name*:*format*	Устанавливает формат заголовка
Set Ruleset	S*n*	Начинает набор правил с номером *n*
Define Rule	R*lhs rhs comment*	Перезаписывает образец *lhs* в формат *rhs*
Key File	K*name type* [*argument*]	Определяет имя базы данных (*name*)

В следующих разделах каждая из команд настройки описана подробнее.

Команда Version Level (Уровень версии)

Команда уровня версии является необязательной и присутствует не во всех файлах *sendmail.cf*. Администратор не добавляет команду `V` в файл *sendmail.cf* и не изменяет уже существующую команду. Команда `V` фигурирует в файле настройки, созданном поставщиком системы либо макропроцессором `m4`.

Число *`level`* в строке команды `V` указывает уровень версии синтаксиса настройки. V1 – самая старая версия синтаксиса настройки, а V9 – версия, которую поддерживает sendmail 8.11.3. Появление промежуточных уровней связано с расширением возможностей. Значение *`vendor`* в строке команды `V` указывает, какие специальные расширения поддерживаются. Для дистрибутива sendmail значение *`vendor`* по умолчанию – Berkeley.

Команда `V` сообщает исполняемому модулю sendmail, какой уровень синтаксиса и команд необходим для работы с данным файлом настройки. Если sendmail не может предоставить затребованную функциональность команд и синтаксиса, то отображает следующее сообщение об ошибке:

```
# /usr/lib/sendmail -Ctest.cf
Warning: .cf version level (9) exceeds sendmail version 8.9.3+Sun functionality (8):
Operation not permitted
```

Это сообщение показывает, что данная версия sendmail поддерживает файлы настройки уровня 8 с расширениями синтаксиса от Sun.[1] Данный пример был получен в системе Solaris 8 для версии sendmail из дистрибутива системы. Мы попытались прочитать файл настройки, созданный на основе макроопределений `m4` из комплекта sendmail 8.11.3. Синтаксис и функциональность, необходимые для этого файла настройки, не доступны в существующей программе sendmail. Чтобы использовать данный файл настройки, нам придется скомпилировать более новую версию sendmail. Пример компиляции sendmail приводится в приложении E.

Вам никогда не придется изменять значения в строке команды `V`. Однако может возникнуть необходимость в уточнении некоторых команд `D`.

Команда Define Macro (Макроопределение)

Команда макроопределения (`D`) определяет переменную и сохраняет в ней значение. Значение макроопределения используется в других командах *sendmail.cf* и непосредственно sendmail. Это позволяет многим системам использовать один файл настройки, внося изменения в небольшое число макроопределений.

В качестве имени макроопределения может выступать любой одиночный символ ASCII либо слово, заключенное в фигурные скобки. Используйте длинные имена для пользовательских макроопределений. Внутренние мак-

[1] Синтаксис для систем Sun описан в табл. 10.4.

роопределения sendmail используют большую часть доступных букв и специальных символов. Кроме того, по умолчанию существует большое число макроопределений с длинными именами. Это не означает, что вы избавлены от необходимости давать имена макроопределениям, но означает, что придется проявлять осторожность, чтобы избежать конфликтов с существующими именами. Внутренние макроопределения иногда отражены в файле *sendmail.cf*. В приложении E приводится полный перечень внутренних макроопределений sendmail. При создании пользовательских макроопределений обращайтесь к этому перечню, чтобы избежать конфликтов имен. Чтобы получить значение, хранимое макроопределением, обратитесь к определению по ссылке `$x`, где `x` – имя макроопределения. Подстановка макроопределений происходит при чтении файла *sendmail.cf*. Для подстановки макроопределений в ссылках используется специальный синтаксис, `$&x`. Синтаксис `$&x` используется только для некоторых внутренних макроопределений, которые изменяются во время работы sendmail.

Приведенный ниже код содержит макроопределения `{our-host}`, `M` и `Q`. После выполнения этого кода ссылка `${our-host}` возвращает значение *crab*, `$M` возвращает значение *wrotethebook.com*, а `$Q` возвращает *crab.wrotethebook.com*. Переменная `Q` хранит значение макроопределения `{our-host}` (а именно `${our-host}`), постоянную точку плюс значение M (`$M`).

```
D{our-host}crab
DMwrotethebook.com
DQ${our-host}.$M
```

Изменение файла *sendmail.cf*, вероятно, потребует изменения некоторых макроопределений. Обычно изменения касаются макроопределений, содержащих специфичную для сервера информацию, такую как имена узлов и доменов.

Условные зависимости в определениях

Макроопределение может содержать условную зависимость. Вот пример такого определения:

```
DX$g$?x ($x)$.
```

Буква `D` – это команда макроопределения; `X` – имя макроопределения; а ссылка `$g` предписывает использовать значение, хранимое в макроопределении `g`. Но что означает последовательность `$?x ($x)$.`? Конструкция `$?x` является условным выражением: она проверяет, установлено ли значение макроопределения `x`. Если значение установлено, интерпретируется текст, следующий за условным выражением. Конструкция `$.` завершает условное выражение.

Итак, макроопределение для `X` интерпретируется следующим образом: `X` присваивается значение `g`; и если значение `x` установлено, к `X` добавляется буквальный пробел, левая скобка, значение `x` и правая скобка.

Поэтому если `g` содержит *chunt@wrotethebook.com*, а `x` содержит *Craig Hunt*, `X` получит значение:

```
chunt@wrotethebook.com (Craig Hunt)
```

Условные выражения могут использоваться в сочетании с конструкцией «иначе», а именно $|. Полный синтаксис условного выражения:

```
$?x text1 $| text2 $.
```

Интерпретируется он следующим образом:

- если ($?) значение x установлено;
- использовать *text1*;
- иначе ($|);
- использовать *text2*;
- конец условного выражения ($.).

Создание классов

За создание классов sendmail отвечают команды C и F. Класс подобен массиву значений. Классы используются во всех случаях, когда необходимо одинаковым образом обрабатывать однородные значения, такие как различные имена локального узла или имена узлов uucp. Классы позволяют sendmail выполнять сравнение сразу со списком значений, а не с отдельными значениями. С классами используются специальные символы шаблонов. Символ $= соответствует любому значению в классе, тогда как символ $~ является маской любого значения, не входящего в класс. (Подробнее о поиске по шаблонам поговорим позже.)

Подобно макроопределениям, классы имеют односимвольные имена либо длинные имена, заключенные в фигурные скобки. В пользовательских классах используются длинные имена, не конфликтующие с внутренними именами sendmail. (Полный перечень имен внутренних классов sendmail приводится в приложении E.) Значения класса могут определяться в одной строке, в нескольких строках либо загружаться из файла. Например, для определения всех имен узлов, под которыми известен локальный узел, используется класс w. Чтобы поместить в класс w значения *goober* и *pea*, можно указать значения в одной строке следующим образом:

```
Cwgoober pea
```

Или в нескольких строках:

```
Cwgoober
Cwpea
```

Кроме того, можно использовать команду F для загрузки значений класса из файла. Команда F читает файл и записывает найденные слова в переменную класса. Например, чтобы создать класс w и присвоить ему все строки из файла */etc/mail/local-host-names*, воспользуйтесь такой командой:[1]

```
Fw/etc/mail/local-host-names
```

[1] В sendmail 8.11 класс w загружается из файла */etc/mail/local-host-names*. В более ранних версиях sendmail файл назывался */etc/sendmail.cw*. Изменилось только имя, файл по-прежнему содержит перечень имен узлов.

При создании файла *sendmail.cf* вам может понадобиться изменить некоторые определения классов. В операторах классов часто фигурирует информация, связанная с uucp, псевдонимами узлов, а также специальными доменами для маршрутизации почты. Если в вашей системе наряду с подключением TCP/IP задействовано подключение uucp, обратите особое внимание на определения классов. В любом случае внимательно проверьте определения классов и убедитесь, что они соответствуют вашей конфигурации.

Вот результат поиска строк команд C и F в примере файла настройки для Linux:

```
% grep '^[CF]' generic-linux.cf
Cwlocalhost
Fw/etc/mail/local-host-names
CP.
CO @ % !
C..
C[[
FR-o /etc/mail/relay-domains
C{E}root
CPREDIRECT
```

Из результатов видно, что в файле *generic-linux.cf* определены классы w, P, O, ., [, R и E. w содержит псевдонимы узла. Обратите внимание, что в класс w значения добавляются при помощи команд C и F одновременно. В отличие от команды D, перезаписывающей значение макроопределения, команды записи в массивы классов являются накопительными. Команды C и F в начале приведенного фрагмента добавляют значения в класс w. Другая иллюстрация накопительной природы команды C – класс P. В P хранятся псевдодомены, используемые для маршрутизации почты. Первая команда C для класса P добавляет в массив точку. Последняя команда списка добавляет REDIRECT в класс P.

В классе O хранятся операторы, недопустимые в именах пользователей. Классы . (точка) и [представляют интерес в основном тем, что показывают, что имена переменных не обязательно представлены алфавитными символами и что допустим массив, состоящий из единственного значения. E перечисляет имена пользователей, которые всегда должны ассоциироваться с абсолютным доменным именем локального узла, даже если для всех прочих пользователей задействованы упрощенные адреса электронной почты (о которых мы еще поговорим позже.) Обратите внимание, что даже односимвольное имя класса, в данном случае E, может заключаться в фигурные скобки.

Помните, что ваша система отличается и классам с такими именами могут назначаться значения совсем иного рода. Приведенная выдержка является только примером. Внимательно прочтите комментарии в своем файле *sendmail.cf* и извлеките информацию о том, как в настройках используются классы и макроопределения.

Многие имена классов зарезервированы для внутреннего использования sendmail. Все внутренние классы, определенные в sendmail версии 8.11, перечислены в приложении E. Системные администраторы, изменяющие не-

посредственно файл *sendmail.cf*, редактируют, как правило, только класс `w`, определяющий все имена узлов, на которые система будет реагировать как на свои собственные.

Установка параметров

Команда установки параметров (`O`) используется для настройки среды sendmail. Используйте команду `O` для указания значений, уместных для вашей системы. Значение, присвоенное параметру, может быть представлено строкой, целым числом, логическим значением либо временным интервалом, в зависимости от назначения параметра. Все параметры хранят значения, непосредственно используемые sendmail.

Параметры не могут создаваться пользователем. Смысл каждого из параметров sendmail зафиксирован в самой программе. Назначение и применение всех параметров, число которых велико, описаны в приложении E.

Приводимый ниже фрагмент файла *generic-linux.cf* содержит несколько примеров параметров. Параметр `AliasFile` определяет имя файла псевдонимов sendmail – */etc/mail/aliases*. Если вы намереваетесь хранить файл *aliases* в другом месте, измените значение этого параметра. Параметр `TempFileMode` определяет умолчание режима создания файлов **0600** для временных файлов, создаваемых программой sendmail в каталоге */var/spool/mqueue*. Параметр `Timeout.queuereturn` устанавливает интервал ожидания для сообщений, не поддающихся доставке. В данном случае ожидание длится пять дней (`5d`). Данные параметры отражают общий характер параметров настройки, устанавливаемых командой `O`.

```
# location of alias file
O AliasFile=/etc/mail/aliases
# temporary file mode
O TempFileMode=0600
# default timeout interval
O Timeout.queuereturn=5d
```

Синтаксис команды `O`, используемый в данном примере и в приложении E, появился в sendmail версии 8.7.5. До этого синтаксис команды походил на синтаксис прочих команд sendmail. Прежний синтаксис: `O`*o*`value`, где `O` – команда, *o* – односимвольное имя параметра, а *value* – значение, присваиваемое параметру. С использованием прежнего синтаксиса приведенные выше параметры будут выглядеть так:

```
# location of alias file
OA/etc/aliases
# temporary file mode
OF0600
# default timeout interval OT5d
```

Если ваша система определяет параметры в старом формате, она опасно устарела и подлежит модернизации. Информация по загрузке, компиляции и установке современной версии sendmail приведена в приложении E.

Большинство параметров в файле *sendmail.cf* из дистрибутива вашей системы не требуют изменений. Настройки изменяют потому, что хотят изменить среду sendmail, а не потому, что это необходимо. Параметры в вашем файле настройки практически наверняка соответствуют потребностям системы.

Определение доверенных пользователей

Команда `T` определяет список пользователей, которым разрешено переопределение адреса отправителя при помощи ключа почтовой программы `-f`.[1] Список доверенных пользователей обычно выглядит так: *root*, *uucp*, *daemon*. Доверенные пользователи могут перечисляться в одной строке либо в нескольких строках. Имена пользователей должны существовать в файле */etc/passwd*.

Общеупотребительные определения доверенных пользователей:

```
Troot
Tdaemon
Tuucp
```

Не изменяйте этот список. Добавление доверенных пользователей повышает риск возникновения проблем безопасности.

Приоритеты почтовых сообщений

Значение предпочтения является для sendmail одним из факторов, учитываемых при расстановке приоритетов для сообщений, помещаемых в очередь. Команда `P` определяет значения приоритетов, доступные пользователям sendmail. Чем выше значение, тем выше приоритет сообщения. По умолчанию сообщению присваивается значение предпочтения 0. Отрицательные значения предпочтения обозначают сообщения с очень низким приоритетом. Для сообщений с отрицательными значениями предпочтения не генерируются ошибки, что делает низкие приоритеты привлекательными для массовых рассылок. Вот некоторые из распространенных значений:

```
Pfirst-class=0
Pspecial-delivery=100
Plist=-30
Pbulk=-60
Pjunk=-100
```

Чтобы указать желаемый приоритет, добавьте заголовок `Precedence` в исходящее сообщение. Используйте текстовое название приоритета из команды `P`. С учетом приведенных выше определений, пользователь, желающий избежать получения сообщений об ошибке при попытке массовой рассылки, может выбрать значение предпочтения для своего письма, равное –60, включив в письмо следующий заголовок:

```
Precedence: bulk
```

[1] Флаги почтовых программ перечислены в приложении E.

Маловероятно, что в какой-либо ситуации пяти значений предпочтения, приведенных выше, вам окажется недостаточно.

Определение почтовых заголовков

Команда `H` определяет формат строк заголовка, вставляемых программой sendmail в сообщения. Формат команды заголовка: команда `H`, необязательные флаги заголовка, заключенные в знаки вопроса, имя заголовка, двоеточие, шаблон заголовка. Шаблон заголовка – это сочетание литералов и макроопределений, включаемых в строку заголовка. Перед вставкой заголовка в сообщение выполняется подстановка макроопределений в шаблоне. Синтаксис условных выражений, описанный для макроопределений, может использоваться и в шаблонах заголовков, и действует точно таким же образом: позволяет проверять, установлено ли значение макропеременной, и указывать альтернативные варианты.

Поле шаблона заголовка может содержать конструкцию `$>`*name*, используемую в правилах подстановки. В шаблоне заголовка конструкция `$>`*name* позволяет вызывать набор правил *name* для обработки заголовков входящих сообщений. Такая возможность полезна для фильтрации по заголовкам нежелательных писем (спама). Наборы правил, правила подстановки, конструкцию `$>`*name*, равно как и применение всех этих сущностей, мы обсудим позже в этой главе.

Флаги заголовков часто вызывают больше вопросов, чем заслуживают. Назначение флагов очень простое: они определяют, должен ли тот или иной заголовок добавляться в сообщение, предназначенное определенной почтовой программе. В отсутствие флагов заголовок вставляется во все сообщения. Если указан флаг, заголовок используется только для сообщений, передаваемых через программу доставки, в определении которой присутствует тот же флаг. (Флаги почтовых программ перечислены в приложении Е.) Флаги заголовков управляют *только вставкой* заголовков. Если заголовок изначально присутствовал в исходных данных, он передается в конечном сообщении независимо от того, какие флаги установлены.

Вот некоторые определения заголовков из файла *generic-linux.cf*:

```
H?P?Return-Path: <$g>
HReceived: $?sfrom $s $.$?_($?s$|from $.$_)
H?D?Resent-Date: $a
H?D?Date: $a
H?F?Resent-From: $?x$x <$g>$|$g$.
H?F?From: $?x$x <$g>$|$g$.
H?x?Full-Name: $x
H?M?Resent-Message-Id: <$t.$i@$j>
H?M?Message-Id: <$t.$i@$j>
```

Заголовков, определенных в файле *sendmail.cf* вашей системы, вполне достаточно для большинства ситуаций. Маловероятно, что понадобится их изменять.

Определения почтовых программ

Команды `M` определяют программы доставки почты, с которыми работает sendmail. Синтаксис команды:

```
Mname, {field=value}
```

name – произвольное внутреннее имя, используемое sendmail для описания данной почтовой программы. Выбор имени не имеет значения, если в файле *sendmail.cf* это имя используется последовательно для ссылки на одну почтовую программу. Так, почтовая программа для доставки SMTP-почты в пределах локального домена может называться *smtp* в одной системе и *ether* в другой. Назначение обеих программ одинаково, различаются только имена.

Свобода выбора имени ограничивается рядом исключений. Почтовая программа, осуществляющая доставку локальной почты пользователей той же машины, должна называться *local*, и определение почтовой программы *local* должно присутствовать в файле *sendmail.cf*. Еще три специальных имени почтовых программ:

- *prog*

  Доставляет почту программам.
- **file*

  Записывает почту в файлы.
- **include*

  Направляет почту в списки `:include:`.

Из перечисленных почтовых программ только *prog* имеет определение в файле *sendmail.cf*. Двум другим соответствуют внутренние определения sendmail.

Имя почтовой программы может быть любым, но обычно имена одинаковы во всех системах, поскольку определения почтовых программ в *sendmail.cf* генерируются из стандартных макроопределений `m4`. В созданной ранее настройке *linux.mc* макроопределение MAILER(local) отвечает за определения почтовых программ *prog* и *local*, а макроопределение MAILER(smtp) – за определения *smtp*, *esmtp*, *smtp8*, *dsmtp* и *relay*. В любой системе, с которой вам придется работать, будет, вероятно, тот же набор имен почтовых программ.

За именем почтовой программы следует список пар *field*=*value*, разделенных запятыми. Эти пары определяют характеристики почтовой программы. Односимвольные идентификаторы полей (*field*) и связанные с ними значения описаны в табл. 10.2. Большинству почтовых программ требуются не все описанные поля.

Таблица 10.2. Поля определений почтовых программ

Поле	Значение	Содержимое	Пример
P	Path	Путь к почтовой программе	P=/bin/mail
F	Flags	Флаги sendmail для данной почтовой программы	F=lsDFMe
S	Sender	Наборы правил для адресов отправителей	S=10

Поле	Значение	Содержимое	Пример
R	Recipient	Наборы правил для адресов получателей	R=20
A	Argv	Вектор аргументов почтовой программы	A=sh -c $u
E	Eol	Символ конца строки для почтовой программы	E=\r\n
M	Maxsize	Максимальная длина сообщения	M=100000
L	Linelimit	Максимальная длина строки	L=990
D	Directory	Рабочий каталог почтовой программы *prog*	D=$z:/
U	Userid	Идентификаторы пользователя и группы, используемые для исполнения почтовой программы	U=uucp:wheel
N	Nice	Значение приоритета, используемое при исполнении почтовой программы	N=10
C	Charset	Тип содержимого для 8-битных символов MIME	C=iso8859-1
T	Type	Информация типа для ошибок MIME	T=dns/rfc822/smtp

Поле Path (P) содержит полное имя программы доставки почты либо строку `[IPC]`. Определение, в котором указано `P=[IPC]`, предписывает использовать sendmail для доставки почты по SMTP.[1] Путь к программе доставки почты для каждой конкретной системы зависит от того, где хранится та или иная программа. Прежде чем вносить изменения в поле Path, уточните, где хранятся программы. Если вы используете файл *sendmail.cf* с другой машины, убедитесь, что пути к почтовым программам верны для данной системы. Если вы используете `m4` для создания настроек, пути верны изначально.

Поле Flags (F) содержит флаги sendmail для данной почтовой программы, а именно флаги почтовых программ, упоминавшиеся ранее в этой главе, в разделе «Определение почтовых заголовков». Однако флаги почтовых программ не просто управляют вставкой заголовков. Флагов существует очень много. Описания флагов содержатся в приложении E.

Поля Sender (S) и Recipient (R) указывают наборы правил, используемые для переписывания адресов отправителя и получателя для данной почтовой программы. Каждый набор правил обозначается по номеру. Мы еще обсудим наборы правил более подробно в этой главе, а к значениям S и R будем обращаться при отладке настроек sendmail.

Поле Argv (A) определяет вектор аргументов, передаваемых почтовой программе. Оно содержит, помимо прочего, расширения макроопределений: имя пользователя-получателя (а именно `$u`)[2], имя узла получателя (`$h`), а также адрес From отправителя (`$f`). Подстановка значений макроопределений выполняется до передачи вектора аргументов почтовой программе.

---

[1] Значения `[TCP]` и `[IPC]` взаимозаменяемы, как в поле P, так и в поле A.

[2] В определении почтовой программы *prog* `$u` в действительности передает имя программы в векторе аргументов.

Поле End-of-line (E) определяет символы, используемые для обозначения конца строки. Комбинация возврата каретки и перевода строки (CRLF) по умолчанию используется для почтовых программ SMTP.

Поле Maxsize (M) определяет максимальный размер сообщения, которое может быть обработано данной почтовой программой, в байтах. Это поле используется чаще всего в определениях почтовых программ UUCP.

Поле Linelimit (L) определяет максимальную длину строки в байтах, допустимую в письме, передаваемом через данную почтовую программу. Данное поле появилось в sendmail V8. Предыдущие версии sendmail ограничивали длину строки 80 символами, поскольку таково было ограничение почты SMTP до появления расширений MIME.

Поле Directory (D) определяет рабочий каталог для почтовой программы *prog*. Поле может содержать несколько имен каталогов, разделенных двоеточиями. Пример в табл. 10.2 предписывает *prog* использовать исходный каталог получателя (таково значение внутреннего макроопределения `$z`). Если указанный каталог недоступен, программе следует использовать корневой (/) каталог.

Поле Userid (U) используется для указания умолчаний идентификаторов пользователя и группы, с полномочиями которых выполняется почтовая программа. Например, `U=uucp:wheel` указывает, что почтовая программа должна выполняться с полномочиями пользователя *uucp* и группы *wheel*. В отсутствие значения в поле Userid используется значение параметра DefaultUser.

Используйте Nice (N), чтобы изменять значение `nice`, с которым работает почтовая программа. Значение `nice` определяет приоритет почтовой программы для планировщика задач. Поле используется редко. Более подробные сведения о допустимых значениях вы можете почерпнуть из страниц руководства (man), посвященных `nice`.

Последние два поля используются только для почты MIME. Charset (C) определяет набор символов; значение используется в заголовке Content-type при преобразовании 8-битных сообщений в формат MIME. В отсутствие определения Charset используется значение параметра DefaultCharSet. Если и этот параметр не определен, по умолчанию используется значение `unknown-8bit`.

Поле Type (T) определяет информацию типа, используемую в MIME-сообщениях об ошибках. Информация MIME-типа определяет тип агента передачи почтовой программы, тип почтового адреса и тип сообщений об ошибках. По умолчанию принимает значение *dns/rfc822/smtp*.

Общеупотребительные определения почтовых программ

Следующие определения почтовых программ взяты из файла *generic-linux.cf*:

```
Mlocal,    P=/usr/bin/procmail, F=lsDFMAw5:/|@qSPfhn9,
           S=EnvFromL/HdrFromL, R=EnvToL/HdrToL, T=DNS/RFC822/X-Unix,
           A=procmail -Y -a $h -d $u
Mprog,     P=/bin/sh, F=lsDFMoqeu9, S=EnvFromL/HdrFromL,
```

```
            R=EnvToL/HdrToL, D=$z:/, T=X-Unix/X-Unix/X-Unix,
            A=sh -c $u
Msmtp,      P=[IPC], F=mDFMuX, S=EnvFromSMTP/HdrFromSMTP, R=EnvToSMTP,
            E=\r\n, L=990, T=DNS/RFC822/SMTP, A=TCP $h
Mesmtp,     P=[IPC], F=mDFMuXa, S=EnvFromSMTP/HdrFromSMTP, R=EnvToSMTP,
            E=\r\n, L=990, T=DNS/RFC822/SMTP, A=TCP $h
Msmtp8,     P=[IPC], F=mDFMuX8, S=EnvFromSMTP/HdrFromSMTP, R=EnvToSMTP,
            E=\r\n, L=990, T=DNS/RFC822/SMTP, A=TCP $h
Mdsmtp,     P=[IPC], F=mDFMuXa%, S=EnvFromSMTP/HdrFromSMTP, R=EnvToSMTP,
            E=\r\n, L=990, T=DNS/RFC822/SMTP, A=TCP $h
Mrelay,     P=[IPC], F=mDFMuXa8, S=EnvFromSMTP/HdrFromSMTP, R=MasqSMTP,
            E=\r\n, L=2040, T=DNS/RFC822/SMTP,A=TCP $h
```

Приведенный пример содержит следующие определения:

- Локальной доставки почты, имя которой всегда *local*. Определение является обязательным для sendmail.
- Определение для доставки почты программам, всегда с именем *prog*. Данное определение присутствует в большинстве файлов настройки.
- Определение для доставки почты по TCP/IP, под именем *smtp*.
- Определение почтовой программы Extended SMTP, под именем *esmtp*.
- Определение почтовой программы SMTP, работающей с незакодированными 8-битными данными, под именем *smtp8*.
- Определение SMTP по требованию, под именем *dsmtp*.
- Определение почтовой программы, осуществляющей пересылку почты TCP/IP через внешний узел пересылки почты, под именем *relay*.

Пристальное изучение полей одного из определений, например определения почтовой программы *smtp*, показывает следующее:

`Msmtp`

Определение относится к почтовой программе с пользовательским именем *smtp*.

`P=[IPC]`

Путь к почтовой программе – `[IPC]`, то есть доставкой почты занимается непосредственно sendmail.

`F=mDFMuX`

Флаги sendmail для данной почтовой программы: программа способна отправлять сообщение нескольким получателям одновременно; требует присутствия заголовков Date, From и Message-Id; заглавные буквы в именах узлов и пользователей должны сохраняться; в начало строки, начинающейся с точки, добавляется дополнительная точка. Более подробная информация содержится в приложении E.

`S =EnvFromSMTP/HdrFromSMTP`

Адрес отправителя на «конверте» сообщения обрабатывается набором правил `EnvFromSMTP`, адрес отправителя в заголовке сообщения обрабатывается набором правил `HdrFromSMTP`. Подробнее об этом позже.

`R= EnvToSMTP`

Все адреса получателей обрабатываются набором правил `EnvToSMTP`.

`E=\r\n`

Строки завершаются возвратом каретки и символом перевода строки.

`L=990`

Данная почтовая программа способна обрабатывать строки до 990 байтов длиной.

`T=DNS/RFC822/SMTP`

Информация MIME-типа для данной почтовой программы: для работы с именами узлов используется DNS, адреса электронной почты подчиняются RFC 822, используются коды ошибок SMTP.

`A=TCP $h`

Значение каждого из параметров в векторе аргументов определяется руководством по команде; в качестве иллюстрации изучите почтовую программу *local*. Однако для почтовой программы *smtp* аргумент ссылается на внутренний процесс sendmail, осуществляющий доставку почты SMTP по соединению TCP. Адрес узла-получателя возникает в результате подстановки макроопределения `$h`.

Несмотря на объем приводимых сведений, не стоит слишком беспокоиться об определениях почтовых программ. Файл настройки, созданный макропроцессором `m4` для вашей операционной системы, содержит верные определения почтовых программ, позволяющие sendmail функционировать в среде сети TCP/IP. Вы не должны испытывать необходимости в изменении каких-либо определений почтовых программ.

Переписывание почтового адреса

Правила подстановки – сердце файла *sendmail.cf*. Наборы правил – это группы отдельных правил подстановки, используемые для разбора адресов электронной почты, полученных от пользовательских почтовых программ, и переписывания их в формате, доступном программам доставки почты. Каждое правило подстановки определяется командой `R`. Синтаксис команды `R`:

```
Rшаблон преобразование комментарий
```

Поля команды `R` разделяются символом табуляции. Система игнорирует поле комментариев, но хорошие комментарии жизненно необходимы, если нужно разобраться в том, что происходит. Поля шаблона и преобразования – вот основа команды `R`.

Шаблоны поиска

Правила подстановки сопоставляют исходный адрес с шаблоном, и если обнаружено соответствие, адрес переписывается в новом формате по правилам преобразования. Адрес может обрабатываться правилом подстановки не-

сколько раз, поскольку после переписывания адрес снова сравнивается с шаблоном. Если соответствие вновь обнаружено, адрес переписывается повторно. Цикл сопоставления с шаблоном и переписывания продолжается, пока адрес не перестанет соответствовать шаблону.

Шаблон определяется посредством макроопределений, классов, литералов и специальных метасимволов. Макроопределения, классы и литералы используются в качестве значений, с которыми сравниваются исходные данные, а метасимволы задают правила сравнения с шаблоном. В табл. 10.3 описаны метасимволы, используемые для поиска по шаблонам.

Таблица 10.3. Метасимволы шаблонов

Символ	Значение
`$@`	Нулевое количество лексем
`$*`	Нуль или более лексем
`$+`	Одна или более лексем
`$-`	Ровно одна лексема
`$=`*x*	Любая лексема из класса *x*
`$~`*x*	Любая лексема не из класса *x*
`$`*x*	Все лексемы из макроопределения *x*
`$%`*x*	Любая лексема из карты NIS, обозначенной макроопределением *x*[1]
`$!`*x*	Любая лексема не из карты NIS, обозначенной макроопределением *x*
`$%y`	Любая лексема из карты NIS *hosts.byname*

[1] Этот символ действует только в операционных системах Sun.

Каждый метасимвол предписывает поиск определенного числа лексем. Лексема – это строка символов в адресе электронной почты, ограниченная оператором. Операторами являются символы, указанные в параметре OperatorChars. Операторы также считаются лексемами при разборе адреса. Следующий адрес электронной почты:

```
becky@rodent.wrotethebook.com
```

содержит семь лексем: becky, @, rodent, ., wrotethebook, ., а также com. Этот адрес соответствует шаблону поиска:

```
$-@$+
```

Адрес соответствует шаблону, потому что:

- в нем ровно одна лексема перед оператором `@`, то есть выполняется требование метасимвола `$-`;
- в нем присутствует оператор `@`, соответствующий литералу шаблона `@`;
- после оператора `@` в адресе присутствует одна или более лексем, то есть выполняется требование метасимвола `$+`.

Многие адреса, такие как *hostmaster@apnic.net* и *craigh@ora.com*, соответствуют этому шаблону, однако далеко не все. Так, адрес *rebecca.hunt@wrotethebook.com* шаблону не соответствует, поскольку оператору @ предшествуют три лексемы: rebecca, ., а также hunt. Следовательно, не выполняется требование наличия ровно одной лексемы, связанное с метасимволом `$-`. Метасимволы, макроопределения и литералы позволяют конструировать шаблоны для поиска адресов электронной почты любого типа.

Если адрес соответствует шаблону, подстроки адреса, найденные метасимволами, сохраняются в *неопределенных лексемах* (*indefinite tokens*). Найденные строки называются неопределенными лексемами, поскольку могут содержать по нескольку значений лексем. Неопределенные лексемы нумеруются последовательно в соответствии с положением в шаблоне метасимволов, которому они соответствуют. Иначе говоря, неопределенная лексема, полученная по соответствию первого метасимвола, называется $1; соответствие для второго символа называется $2; для третьего – $3 и т. д. Сопоставление адреса *becky@rodent.wrotethebook.com* с шаблоном `$-@$+` привело к созданию двух неопределенных лексем. Первая из них обозначается как $1 и содержит одну лексему, becky, соответствующую символу `$-`. Вторая неопределенная лексема носит имя $2 и содержит пять лексем – rodent, ., wrotethebook, ., а также com, – соответствующих символу `$+`. На неопределенные лексемы, созданные при поиске по шаблону, можно ссылаться по именам ($1, $2 и т. д.) при переписывании адреса.

Некоторые из символов, описанных в табл. 10.3, используются только в особых случаях. Символ `$@` используется обычно отдельно для поиска пустого адреса. Символы сравнений с картами NIS могут использоваться только в операционной системе Sun с программой sendmail из дистрибутива системы. В следующем разделе мы увидим, что системы, использующие стандартную версию sendmail, могут работать с картами NIS, но только в части преобразований, а не поиска по шаблонам.

Преобразование адреса

Поле преобразования в правой части правила подстановки определяет формат, в который переписывается адрес. Формат определяется при помощи тех же элементов, что и шаблон: литералов, макроопределений и специальных метасимволов. Литералы в преобразованиях попадают в новый адрес без изменений. Для макроопределений выполняется подстановка. Метасимволы решают специальные задачи. Метасимволы преобразований и их функциональность описаны в табл. 10.4.

Таблица 10.4. Метасимволы преобразований

Символ	Значение
`$`*n*	Подстановка неопределенной лексемы *n*
`$[`*name*`$]`	Подстановка канонической формы имени *name*

Символ	Значение
$*map key*$@*argument* $:*default*$)	Подстановка значения из карты *map* по индексу *key*
$>*n*	Вызов набора правил *n*
$@	Завершение набора правил
$:	Завершение правила подстановки

Символ *$n*, где *n* – число, используется для подстановки неопределенных лексем, обсуждавшихся выше. После подстановки неопределенная лексема записывается в «новый» адрес. Подстановка неопределенных лексем – важнейшая составляющая гибкого переписывания адресов. Она дает возможность простого переноса значений из исходного адреса в конечный адрес. Следующий пример иллюстрирует сказанное.

Адрес всегда обрабатывается рядом правил подстановки. Универсальных правил не бывает. Предположим, исходный адрес *mccafferty@rodent* подвергся некоторой предварительной обработке и принял вид:

```
kathy.mccafferty<@rodent>
```

Предположим, что текущее правило подстановки:

```
R$+<@$-> $1<@$2.$D> user@host -> user@host.domain
```

Адрес соответствует шаблону, поскольку содержит одну или более лексем перед литералом <@, ровно одну лексему после литерала <@, а также литерал >. Сопоставление с шаблоном при переписывании адреса приводит к созданию двух неопределенных лексем, используемых для преобразования.

Поле преобразования содержит неопределенную лексему $1, литерал <@, неопределенную лексему $2, литерал точки (.), ссылку на макроопределение D, а также литерал >. После сопоставления с шаблоном $1 содержит *kathy.mccafferty*, а $2 содержит *rodent*. Предположим, что макроопределение D было создано ранее в файле *sendmail.cf* и хранит значение *wrotethebook.com*. В результате исходный адрес переписывается следующим образом:

```
kathy.mccafferty<@rodent.wrotethebook.com>
```

Рисунок 10.3 иллюстрирует данное конкретное преобразование адреса. Показаны лексемы, выделенные из исходного адреса, а также их сопоставление с шаблоном. Представлены также неопределенные лексемы, созданные при сопоставлении с шаблоном, и их использование наряду с другими значениями поля преобразования для создания конечного, измененного адреса. После подстановки адрес снова сравнивается с шаблоном. На этот раз с отрицательным результатом, поскольку адрес уже не содержит одну и только одну лексему между литералами <@ и >. Итак, адрес не подвергается дальнейшей обработке данным правилом подстановки, но передается следующему по очереди правилу. Правила набора выполняются последовательно, хотя для изменения порядка их обработки могут использоваться некоторые метасимволы.

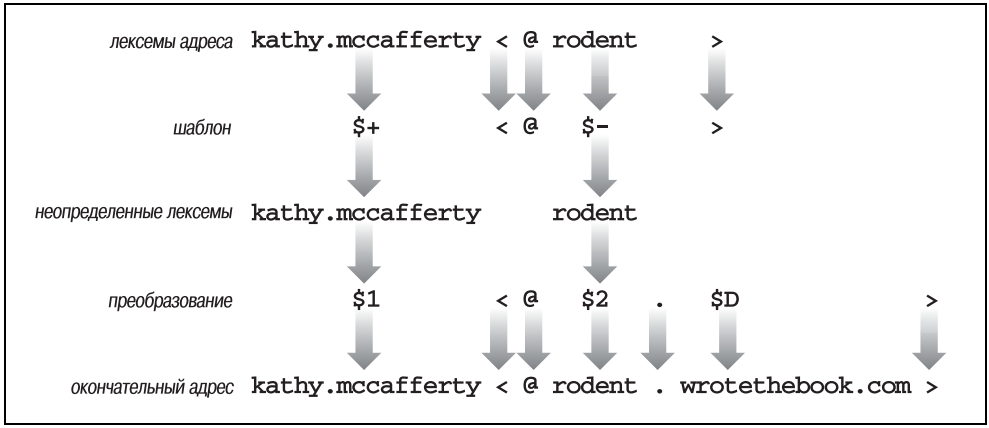


Рис. 10.3. *Переписывание адреса*

Символ `$>`*n* вызывает набор правил *n* и передает этому набору адрес, созданный оставшимися инструкциями поля преобразования. Например:

```
$>9 $1 % $2
```

Это преобразование вызывает набор правил 9 (`$>9`) и передает этому набору для обработки содержимое лексемы $1, символ %, а также содержимое лексемы $2. Когда набор правил 9 заканчивает работу, он возвращает обработанный адрес вызвавшему его правилу. Этот адрес снова сравнивается с шаблоном вызывающего правила. Если соответствие найдено, набор правил 9 вызывается повторно.

Рекурсия, присущая правилам подстановки, потенциально позволяет создавать бесконечные циклы. sendmail, как может, старается обнаружить возможные циклы, однако ответственность за создание правил, ведущих к зацикливанию, лежит на их авторе. Символы `$@` и `$:` используются для управления обработкой и предотвращения зацикливания. Если преобразование начинается символом `$@`, весь набор правил завершается и остаток поля преобразования возвращается набором правил. Если преобразование начинается символом `$:`, данное конкретное правило выполняется лишь единожды. Используйте `$:` для предотвращения рекурсии и зацикливания при вызове других наборов правил. Используйте `$@` для прерывания обработки набора правил на конкретном правиле.

Символ `$[`*name*`$]` преобразует псевдоним узла или его адрес IP в каноническое имя узла. С этой целью значение *name* передается для разрешения серверу имен. К примеру, при обращении к серверам имен домена *wrotethebook.com* `$[mouse$]` возвращает *rodent.wrotethebook.com*, а `$[[172. 16.12.1]$]` возвращает *crab.wrotethebook.com*.

Подобно тому как имя узла или адрес используется для поиска канонического имени в базе данных сервера имен, ключ (*key*) может использоваться для извлечения информации из базы данных (*map*) при помощи конструк-

ции $(*map key*$). Здесь мы имеем дело с более сложным в применении и более общим синтаксисом работы с базой данных, чем в случае извлечения канонических имен. Прежде чем перейти к подробностям создания и использования баз данных для sendmail, закончим разбираться с синтаксисом правил подстановки.

Существует специальный синтаксис правил подстановки, используемый в наборе правил 0. В наборе правил 0 определена тройка значений (*mailer*, *host*, *user*), указывающая программу доставки почты, узел-получатель, а также адресата почты.

Специальный синтаксис преобразования, решающий задачу, выглядит следующим образом:

```
$#mailer$@host$:user
```

Пример использования такой конструкции из файла *generic-linux.cf*:

```
R$*<@$*>$*   $#esmtp $@ $2 $: $1 < @ $2 > $3 user@host.domain
```

Предположим, что данным правилом обрабатывается адрес электронной почты *david<@ora.wrotethebook.com>*. Этот адрес соответствует шаблону $*<@$+>$* по следующим причинам:

- Нуль или более лексем (david) соответствуют первому символу $*
- Адрес содержит литерал <@
- Нуль или более лексем (пять лексем фрагмента ora.wrotethebook.com) соответствуют требованиям второго символа $*
- Адрес содержит литерал >
- Нуль или более (в данном случае нуль) лексем соответствуют требованиям последнего символа $*

Сопоставление адреса с шаблоном приводит к созданию двух неопределенных лексем. Лексема $1 содержит *david*, а лексема $2 содержит *ora.wrotethebook.com*. Других соответствий не было, поэтому лексема $3 пуста. Эти неопределенные лексемы используются для переписывания адреса в следующую тройку значений:

```
$#smtp$@ora.wrotethebook.com$:david<@ora.wrotethebook.com>
```

Компоненты результата:

`$#smtp`

`smtp` — внутреннее имя почтовой программы, доставляющей сообщение.

`$@ora.wrotethebook.com`

`ora.wrotethebook.com` — узел-получатель.

`$:david<@ora.wrotethebook.com>`

`david<@ora.wrotethebook.com>` — пользователь-адресат.

Существует ряд вариаций синтаксиса тройки значений почтовой программы, которые также используются в шаблонах некоторых правил. Две из таких вариаций содержат только составляющую «почтовой программы».

`$#OK`

Указывает, что исходный адрес прошел проверку безопасности. К примеру, адресу разрешена пересылка почтовых сообщений.

`$#discard`

Указывает, что исходный адрес не прошел тот или иной тест безопасности и сообщение электронной почты должно быть удалено.

Ни одно из значений — `OK`, `discard` и `error` — не фигурирует в командах `M` наряду с реальными почтовыми программами. Однако в документации по sendmail эти значения названы «почтовыми программами»; этой терминологии мы здесь и следуем.

Почтовые программы `$#OK` и `$#discard` используются в управлении пересылкой и в безопасности. Почтовая программа `$#discard` молча удаляет почту и не возвращает отправителю сообщение об ошибке. Почтовая программа `$#error` также работает с почтовыми сообщениями, не подлежащими доставке, однако, в отличие от `$#discard`, возвращает отправителю сообщение об ошибке. Конструкция шаблона для почтовой программы `$#error` сложнее, чем синтаксис `$#OK` и `$#discard`. Выглядит она следующим образом:

```
$#error $@dsn-code $:message
```

В качестве значения почтовой программы должно выступать `$#error`. Поле `$:`*`message`* содержит текст сообщения об ошибке, который необходимо передать отправителю. Поле `$@`*`dsn-code`* необязательное. Если оно присутствует, то предшествует сообщению *`message`* и должно содержать корректный код DSN (Delivery Status Notification, уведомление о состоянии доставки). Коды DSN определены в RFC 1893, *Mail System Status Codes* (Коды состояний почтовой системы).

Код DSN состоит из трех компонентов, разделенных точками:

`класс (class)`

Всеобъемлющая классификация состояния. В RFC для класса определены такие значения: `2` означает успешное завершение, `4` означает временный сбой, а `5` — постоянный сбой.

`предмет (subject)`

Соотносит сообщения об ошибках с одной из восьми категорий:

`0` (*Неопределенная*)

Невозможно причислить ошибку к определенной категории.

`1` (*Адресация*)

Проблема, связанная с адресом.

2 (*Почтовый ящик*)

Проблема, связанная с почтовым ящиком доставки.

3 (*Почтовая система*)

Проблема, связанная с системой доставки почты второй стороны.

4 (*Сеть*)

Проблема, связанная с сетевой инфраструктурой.

5 (*Протокол*)

Проблема, связанная с протоколом.

6 (*Содержимое*)

Содержимое сообщения вызвало ошибку преобразования.

7 (*Безопасность*)

Проблема, связанная с безопасностью.

подробности (detail)

Содержит подробные сведения по конкретной ошибке. Значение `detail` осмысленно только в контексте кода `subject`. К примеру, `x.1.1` означает несуществующее имя пользователя-адресата, а `x.2.1` означает, что почтовый ящик заблокирован, тогда как `x.2.2` – что почтовый ящик полон. Значений `detail` слишком много, чтобы перечислять их здесь. Полный список содержится в RFC 1893.

Сообщение об ошибке, записанное в целевом представлении DSN, может выглядеть следующим образом:

```
R<@$+> $#error$@5.1.1$:"user address required"
```

Это правило возвращает код DSN `5.1.1` и сообщение `"user address required"`, если адрес соответствует шаблону. Код DSN состоит из значения `5` в поле класса (постоянная ошибка), значения `1` в поле предмета (сбой адресации) и значения `1` в поле подробностей (с учетом, что subject имеет значение `1`, речь идет о некорректном адресе пользователя).

Коды ошибок и синтаксис ошибок – составная часть сложных параметров настройки, используемых для управления пересылкой и безопасностью. Эти значения генерируются макроопределениями `m4`, предназначенными для работы со сложными механизмами. Очень редко эти значения добавляются в файл *sendmail.cf* напрямую системным администратором.

Преобразование по базе данных

Для преобразования адресов в правилах подстановки могут использоваться базы данных. База данных включается в часть правила, отвечающую за преобразование, посредством следующей конструкции:

```
$(map key [$@argument...] [$:default] $)
```

Здесь *map* – имя, назначенное базе данных в файле *sendmail.cf*. Имена карт *map* не ограничены теми соглашениями, которым подчиняются имена макро-

определений. Подобно именам почтовых программ, имена карт используются только в файле *sendmail.cf* и могут быть произвольными. Выбирайте простые описательные имена, такие как «users» или «mailboxes». Имя карты назначается при помощи команды `K`, о которой мы скоро поговорим.

key – это значение индекса записи из базы данных. Значение, возвращаемое базой данных по этому ключу, используется для переписывания исходного адреса. Если значение отсутствует, исходный адрес не изменяется, если только не указано значение *default*.

argument – дополнительное значение, передаваемое подпрограмме работы с базой данных наряду с ключом. Аргументов может быть несколько, но каждый аргумент должен начинаться сочетанием символов `$@`. Аргумент может использоваться подпрограммой для изменения значений, возвращаемых в sendmail. В базе данных ссылки на аргументы имеют формат `%`*n*, где *n* – цифра, указывающая порядковый номер аргумента в правиле подстановки – `%1`, `%2` и т. д. – для случаев, когда аргументов несколько. (Аргумент `%0` – это ключ, *key*.)

Следующий пример прояснит использование аргументов. Рассмотрим такой исходный адрес:

```
tom.martin<@sugar>
```

Предположим, что существует база данных, хранящая внутренние имена sendmail для узлов пересылки:

```
oil     %1<@relay.fats.com>
sugar   %1<@relay.calories.com>
salt    %1<@server.sodium.org>
```

Кроме того, предположим, что есть следующее правило подстановки:

```
R$+<@$-> $(relays $2 $@ $1 $:$1<@$2> $)
```

Исходный адрес *tom.martin*<*@sugar*> соответствует шаблону, поскольку содержит одну или более лексем (tom.martin) перед литералом <@ и ровно одну лексему (sugar) после этого литерала. Сопоставление с шаблоном приводит к созданию двух неопределенных лексем и передаче их в часть преобразования. В преобразовании происходит обращение к базе данных (relays) с передачей лексемы $2 (sugar) в качестве ключа и лексемы $1 (tom.martin) в качестве аргумента. Если ключ не найден в базе данных, используется значение по умолчанию ($1<@$2>). В данном случае ключ в базе данных существует. Подпрограмма базы данных использует ключ для поиска значения «%1@relay.calories.com», подставляет аргумент %1 и окончательно возвращает sendmail значение «tom.martin@relay.calories.com». sendmail использует полученное значение для замены исходного адреса.

Прежде чем использовать базу данных в sendmail, ее необходимо определить. Этой цели служит команда `K`. Синтаксис команды `K`:

```
Kname type [arguments]
```

name – это имя, по которому можно ссылаться на базу данных из настроек sendmail. В приведенном выше примере фигурировало имя «relays».

type – это класс базы данных. Значение *type* в команде K должно соответствовать возможностям работы с базами данных, встроенным в sendmail при компиляции. Большинство исполняемых файлов sendmail поддерживает не все типы баз данных, однако ряд основных типов поддерживается достаточно широко. Общеупотребительные типы: hash, btree, nis. Типов существует гораздо больше, и все они описаны в приложении E.

Аргументы (*arguments*) являются необязательными. Обычно единственным аргументом оказывается путь к файлу базы данных. Время от времени аргументы включают флаги, интерпретируемые подпрограммой базы данных. Полный перечень флагов команды K содержится в приложении E.

Чтобы определить базу данных «relays» из приведенного выше примера, мы можем использовать следующую команду в файле *sendmail.cf*:

```
Krelays hash /etc/mail/relays
```

Имя *relays* – просто имя, выбранное пользователем в качестве описательного. Тип базы данных *hash* – это тип, поддерживаемый вашей версией sendmail, он использовался и для создания файла базы данных. Наконец, аргумент */etc/mail/relays* – полное имя файла базы данных.

Не переживайте, если вам не очень понятно, как создавать и использовать файлы баз данных в sendmail. Мы еще вернемся к теме позже в этой главе и проясним на примерах практическое применение таких файлов.

Создание набора правил

Набор правил – это группа связанных правил подстановки, на которую можно ссылаться по номеру или по имени. Команда S отмечает начало набора правил и дает ему имя. В конструкции S*name* поле *name* содержит идентификатор набора правил. Кроме того, с набором может быть связан необязательный номер – при помощи полной конструкции, S*name=number*. В этом случае ссылаться на набор правил можно как по имени, так и по номеру. Можно даже обозначить набор правил номером вместо имени – посредством старого синтаксиса S*number*. Подобные конструкции встречаются преимущественно в старых файлах настройки, поскольку в прежних версиях sendmail для идентификации наборов правил использовались номера.

Набор правил можно считать подпрограммой, или функцией, предназначенной для обработки адресов электронной почты. Наборы вызываются из определений почтовых программ, отдельных правил подстановки либо напрямую sendmail. Шесть наборов правил имеют специальное назначение и вызываются напрямую sendmail. Вот эти наборы:

- Набор canonify (3) – первый из наборов, применяемых к адресам. Он преобразует адрес в канонический вид: *local-part@host.domain*.

- Набор parse (0) применяется к адресам, используемым для доставки почты. Набор parse применяется после набора canonify, и только для адресов получателей, фактически используемых при доставке почты. parse преобразует адрес в тройку значений (*mailer*, *host*, *user*), состоящую из имени почтовой программы, осуществляющей доставку почты, имени узла-адресата, имени получателя сообщения.
- Набор sender (1) применяется ко всем адресам отправителя в сообщении.
- Набор recipient (2) применяется ко всем адресам получателя в сообщении.
- Набор final (4) применяется ко всем адресам в сообщении и используется для преобразования адреса из внутреннего формата во внешний формат.
- Набор localaddr (5) применяется к локальным адресам после их обработки sendmail по файлу aliases. Набор 5 применяется только для локальных адресов, не имеющих псевдонимов.

На рис. 10.4 представлен порядок обработки сообщения и адресов перечисленными наборами правил. Символы S и R обозначают дополнительные наборы правил. Дополнительные наборы также имеют имена, однако эти имена не являются фиксированными, как в случае описанных выше наборов. Имена наборов S и R указываются в полях S и R определений почтовых программ. С каждой почтовой программой могут быть связаны собственные наборы правил S и R, выполняющие специальную обработку адресов отправителя и получателя непосредственно перед доставкой сообщения.

В большинстве файлов *sendmail.cf* наборов правил, разумеется, гораздо больше. Прочие наборы правил осуществляют дополнительную обработку адресов и вызываются существующими наборами правил при помощи конструкции `$>`*n*. (См. табл. 10.5 далее в этой главе.) Наборы правил, существующие в файле *sendmail.cf*, поставляемом с вашей системой, адекватны для целей доставки почты SMTP. Маловероятно, что вам придется добавлять

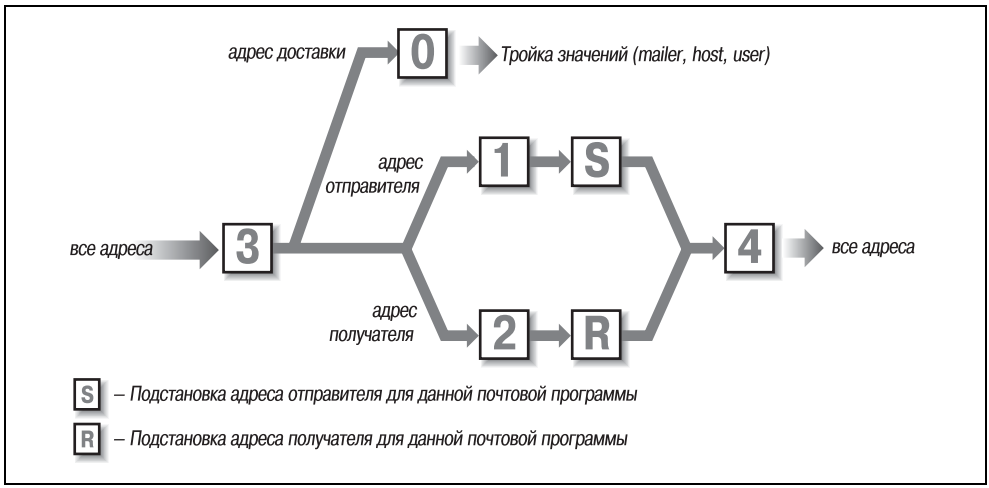


Рис. 10.4. *Последовательность наборов правил*

что-либо к этим наборам, если только вы не описываете новые возможности своей почтовой программы.

Изменение файла sendmail.cf

В этом разделе мы применим на практике все, что узнали о файлах настройки sendmail – их структуре и командах создания. Мы изменим файл настройки *generic-linux.cf* с целью использования его на узле *rodent.wrotethebook.com*. Почему именно этот файл? Потому что его настройки ближе всего к тем, что нужны нам для *rodent.wrotethebook.com.rodent* – рабочей станции под управлением Linux, в сети TCP/IP Ethernet, работающей с почтой SMTP и системой DNS.

Заголовки последующих разделов повторяют названия разделов файла, а сами разделы описывают изменения, вносимые нами в файл. Помните, что в других файлах *sendmail.cf*, вероятно, будут иные заголовки разделов, хотя базовая информация, хранимая в файле настройки, останется прежней.

Изменение локальной информации

Первая строка раздела локальной информации файла настройки содержит определение класса `w`.[1] Класс `w` хранит полный набор имен узлов, для которых принимает почту эта система. Используйте команду `C` или `F` для добавления имен узлов к этому набору. sendmail инициализирует этот класс значением макроопределения `w` (`$w`), которое соответствует имени данного компьютера. Для многих систем этого достаточно. Но иногда сервер sendmail выступает в роли сервера почтовых ящиков: принимает и хранит почту для клиентов, не получающих SMTP-почту напрямую. В классе `w` должны быть указаны системы, для которых этот узел принимает почту. Для каждого клиента почтового ящика следует добавить имя узла в класс `w`.

В рассматриваемом примере мы принимаем команду `Cw` в ее исходном виде и разрешаем sendmail определять значение `w` самостоятельно. Это наиболее распространенное решение для рабочих станций вроде *rodent*. В системе *crab*, известной также под именем *wtb-gw*, мы добавим значения в класс `w` следующим образом:

```
Cwlocalhost wtb-gw wtb-gw.wrotethebook.com
```

Теперь почта, адресованная пользователям *user@wtb-gw.wrotethebook.com*, будет принята узлом *crab*, а не отвергнута как адресованная не тому узлу.

Некоторые почтовые серверы требуют настройки для приема почты, адресованной узлам с различными именами. В таком случае может быть удобно загружать класс `w` из файла, содержащего имена всех узлов, при помощи команды `F`. В файле *generic-linux.cf* команда `F` уже существует, поэтому достаточно будет поместить имена узлов-клиентов в файл */etc/mail/local-host-names*.

[1] Команды `C` и `F` из файла *generic-linux.cf* приводятся выше в данной главе.

Макроопределение j не требует изменений, поскольку в данной системе sendmail получает абсолютное доменное имя для j из DNS. Так же обстоят дела в большинстве систем; в других системах sendmail извлекает имя узла без доменного расширения. Если j не содержит полного имени, инициализируйте макроопределение именем узла ($w) и именем домена. В файле примера с этой целью мы раскомментировали бы команду Dj и изменили бы строку домена на *wrotethebook.com*. Впрочем, делать это нет необходимости, поскольку j хранит верное значение.

Чтобы проверить, верно ли значение j для вашей системы, выполните sendmail с ключом -bt и уровнем отладки 0.4. В результате sendmail выводит несколько строк информации, включая и значение j. В следующем примере sendmail отображает для j значение *rodent.wrotethebook.com*. Получив результат *rodent*, мы отредактировали бы *sendmail.cf* с целью корректировки значения j.

```
# sendmail -bt -d0.4
Version 8.11.3
 Compiled with: LOG MATCHGECOS MIME8TO7 NAMED_BIND NDBM
                NETINET NETUNIX NEWDB SCANF USERDB XDEBUG
canonical name: rodent.wrotethebook.com
 UUCP nodename: rodent
        a.k.a.: rodent.wrotethebook.com
        a.k.a.: [172.16.12.2]

============ SYSTEM IDENTITY (after readcf) ============
      (short domain name) $w = rodent
  (canonical domain name) $j = rodent.wrotethebook.com
         (subdomain name) $m = wrotethebook.com
              (node name) $k = rodent
========================================================

ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address> > ^D
```

Следующая строка раздела локальной информации содержит определение класса P. В нашем примере файла настройки класс P хранит имена двух псевдодоменов: «.» и REDIRECT. Псевдодомен «точка» (.) используется для идентификации канонических доменных имен. Псевдодомен REDIRECT используется возможностью redirect, описанной в приложении E. Класс P может содержать и другие псевдодомены, позволяющие обращаться к пользователям, не подключенным к сети Интернет, по адресам электронной почты в интернет-стиле. Например, можно добавить UUCP в класс P, что позволит адресовать почту посредством старого UUCP-синтаксиса с восклицательным знаком, например *ora!los!craig*, либо в формате псевдо-интернет-адреса: *craig@los.ora.uucp*. Эти домены маршрутизации почты упрощают адрес, вводимый пользователем, и передают почту нужному узлу пересылки. Однако потребность в дополнительных псевдодоменах возникает редко, поскольку сегодня большинство почтовых программ поддерживают стандартные адреса в интернет-стиле. Определение класса P в файле *generic-linux.cf* не требует изменений.

Файл настройки содержит макроопределения ряда узлов пересылки почты. Ни одному из определений не присвоено значение в нашем файле примера. Узел пересылки требуется только в том случае, если ваша система неспособна доставить почту из-за отсутствия подключения или нужной функциональности. Функциональность в Unix-системах присутствует, однако сетевые взаимодействия могут ограничиваться брандмауэрами. Некоторые сетевые площадки используют узел пересылки, чтобы минимизировать число систем, требующих полноценной настройки в файле *sendmail.cf*. Прочие узлы площадки просто передают свою почту «умному» узлу для доставки. Если такова архитектура вашей площадки, укажите имя узла пересылки. Например, так:

```
DSrelay.wrotethebook.com
```

Для узла *rodent* мы оставляем все настройки узлов пересылки без изменений. Данная рабочая станция самостоятельно будет работать со своей почтой. Собственно, ради этого мы и работаем с Unix!

Раздел локальной информации в нашем файле содержит также пять определений файлов ключей. Две из этих K-команд определяют псевдобазы данных – внутренние подпрограммы sendmail используют их в правилах подстановки, как если бы это были реально существующие базы данных. База данных *arith* – это внутренняя подпрограмма, выполняющая определенные арифметические функции. База данных *dequote* – внутренняя подпрограмма sendmail, удаляющая кавычки из адресов электронной почты. Три других команды K определяют реальные базы данных: *mailertable*, *virtuser*, *access*. Базы данных реальные, однако файлы баз данных существуют только в случае, если администратор создаст их. База *mailertable* используется для отправки почты, адресованной в определенный домен, посредством определенной почтовой программы конкретному удаленному узлу. База *virtuser* реализует маршрутизацию почты для виртуальных почтовых доменов, существующих только в пределах сервера sendmail. База данных *access* реализует управление доступом для пересылки почты и контроля за спамом.

Номер версии *не требует* изменений, однако неплохо бы отслеживать изменения, внесенные в файл *sendmail.cf*, и номер версии позволяет это делать. Каждый раз, внося изменения в настройки, изменяйте номер версии. В то же время создавайте сопутствующий версии комментарий, описывающий внесенные изменения. Обычно правка номера версии производится после всех прочих изменений, так что комментарий отражает все внесенные изменения. Например, исходный раздел номера версии в файле *generic-linux.cf* выглядит так:

```
######################
#   Version Number   #
######################
 DZ8.11.3
```

После всех внесенных нами изменений он примет вид:

```
######################
#   Version Number   #
######################
```

```
#  R1.0 - изменения для rodent, автор Craig
#       - уточнение комментариев в разделе локальной информации
#  R1.1 - изменилось макроопределение M: в исходящей почте используется
#      wrotethebook.com вместо имени узла
#  R2.0 - добавлено правило в SEnvFromSMTP и S HdrFromSMTP с целью переписывания
          в исходящей почте идентификатора пользователя  в формат имя.фамилия
 DZ8.11.3R2.0
```

Наконец, необходимо разобраться с назначением ряда других классов и макроопределений из этого раздела. Макроопределение `M` используется для изменения адреса узла отправителя. Определите значение `M`, чтобы скрыть имя локального узла в исходящих сообщениях. Классы `E` и `M` связаны с макроопределением `M`. Класс `E` указывает имена пользователей, для которых не производится замена имени узла даже в присутствии макроопределения `M`. Например, *root@rodent.wrotethebook.com* не переписывается в виде *root@wrotethebook.com*, даже если `M` определено как `DMwrotethebook.com`. Класс `M` определяет прочие имена узлов, помимо локального, для которых следует выполнять подстановку значения макроопределения `M`. Это используется на почтовых серверах при необходимости переписывать адреса отправителей для клиентов. Например:

```
# who I masquerade as (null for no masquerading) (see also $=M)
DMwrotethebook.com

# class M: domains that should be converted to $M CM24seven.wrotethebook.com
brazil.wrotethebook. com ora.wrotethebook.com
```

С учетом приведенных выше определений макроопределения `M` и класса `M`, данный узел будет изменять адреса вида *user@brazil.wrotethebook.com* и *user@24seven.wrotethebook.com* на адреса вида *user@wrotethebook.com*. *rodent* сервером не является, и потому мы не используем класс `M`. Однако макроопределение `M` мы еще задействуем в настройках.

Мы потратили массу времени на изучение раздела локальной информации потому, что практически все действия по настройке системы могут выполняться в этом разделе. Мы быстро обсудим прочие разделы, прежде чем перейти к действительно серьезной задаче работы с правилами подстановки.

Изменение параметров

Раздел параметров «Options» определяет параметры работы среды sendmail. Некоторые из параметров определяют пути к файлам, используемым sendmail, как в следующих строках *generic-linux.cf*:

```
# location of alias file
O AliasFile=/etc/mail/aliases
# location of help file
O HelpFile=/etc/mail/helpfile
# status file
O StatusFile=/etc/mail/statistics
# queue directory
O QueueDirectory=/var/spool/mqueue
```

Если пути верны для вашей системы, не меняйте их. На *rodent* мы оставим файлы на своих местах, как обычно и бывает при использовании *sendmail.cf*, разработанного для вашей операционной системы. В действительности вам навряд ли понадобится изменять какие-либо из параметров в файле настройки, созданном для установленной операционной системы. Если же вас интересуют параметры sendmail, загляните в приложение E.

В последующих разделах файла *generic-linux.cf* содержатся определения приоритетов сообщений, доверенных пользователей и заголовков. Ни один из этих разделов не подвергается изменениям. Затем следуют разделы с правилами подстановки и определениями почтовых программ. Они и занимают большую часть объема файла и являются основой настроек. Пример файла настройки разрешает доставку почты SMTP на системе Linux с работающей DNS, поэтому мы предположим, что изменения не требуются. Но прежде чем копировать настройки в файл *sendmail.cf*, мы протестируем их. Сохранив настройки во временном файле *test.cf*, мы воспользуемся возможностями отладки sendmail.

Тестирование sendmail.cf

sendmail предоставляет мощные инструменты для тестирования и отладки настроек. Эти инструменты доступны из командной строки sendmail посредством некоторых из многочисленных ключей. Все существующие ключи командной строки sendmail описаны в приложении E, а в табл. 10.5 приведены ключи, относящиеся к тестированию и отладке.

Таблица 10.5. Ключи sendmail для тестирования и отладки

Ключ	Функциональность
`-t`	Передача по всем адресам из полей To:, Cc: и Bcc:
`-bt`	Работа в режиме тестирования
`-bv`	Проверка адресов без сборки и доставки почты
`-bp`	Вывод почтовой очереди
`-C`*`file`*	Использование указанного файла *`file`* в качестве файла настройки
`-d`*`level`*	Указание уровня отладки
`-O`*`option=value`*	Установка параметра *`option`* в значение *`value`*
`-e`	Указание способа возврата ошибок
`-v`	Работа в режиме подробной диагностики

Некоторые ключи командной строки используются для проверки механизмов обработки адресов и подтверждения работоспособности новых настроек. Когда вы будете уверены в работоспособности настроек, отправьте себе почту на различные адреса; необходимость в тестировании – отличная причина завести несколько почтовых адресов в различных бесплатных почтовых

службах. Используйте ключ -C для работы с определенным файлом настройки и ключ -v для подробного диагностирования процесса доставки почты. -v полностью выводит обмен SMTP между двумя узлами.

Наблюдая, как почтовая программа подключается к удаленной почтовой программе и форматирует адреса, можно получить представление о том, насколько хорошо работают настройки. Следующий тест выполнялся с узла *rodent* с использованием только что созданного нами файла *test.cf*:

```
rodent# sendmail -Ctest.cf -t -v
To: craigh@ora.com
From: craig
Subject: Sendmail Test
Ignore this test.
^D
craigh@ora.com... Connecting to ora.com. via esmtp...
220-ruby.ora.com ESMTP Sendmail 8.9.3+Sun/8.9.3; Wed, 23 May 2001
>>> EHLO rodent.wrotethebook.com
250-ruby.ora.com Hello craig@rodent.wrotethebook.com [172.16.12.2],
pleased to meet you
250-EXPN
250-VERB
250-8BITMIME
250-SIZE
250-DSN
250-ONEX
250-ETRN
250-XUSR
250 HELP
>>> MAIL From:<craig@rodent.wrotethebook.com> SIZE=64
250 <craig@rodent.wrotethebook.com>... Sender ok
>>> RCPT To:<craigh@ora.com>
250 <craigh@ora.com>... Recipient ok
>>> DATA
354 Enter mail, end with "." on a line by itself
>>> .
250 SAA27399 Message accepted for delivery
craigh@ora.com... Sent (SAA27399 Message accepted for delivery)
Closing connection to ora.com.
>>> QUIT
221 ruby.ora.com closing connection
```

Весь текст до символа ^D (<Ctrl>+<D>) мы набрали самостоятельно. Весь текст после символа ^D является выводом sendmail. На рис. 10.5 выделены некоторые важные элементы вывода и отмечены макроопределения sendmail, связанные с выделенными фрагментами.

В результате тестирования сообщение успешно получено удаленным узлом сети Интернет. Вывод sendmail показывает, что узел *rodent* отправил почту в домен *ora.com* посредством программы доставки *smtp*. Приветствие sendmail показывает, что удаленным узлом, принимающим данное SMTP-соединение, является *ruby.ora.com*. Следовательно, узел *ruby* должен являться поч-

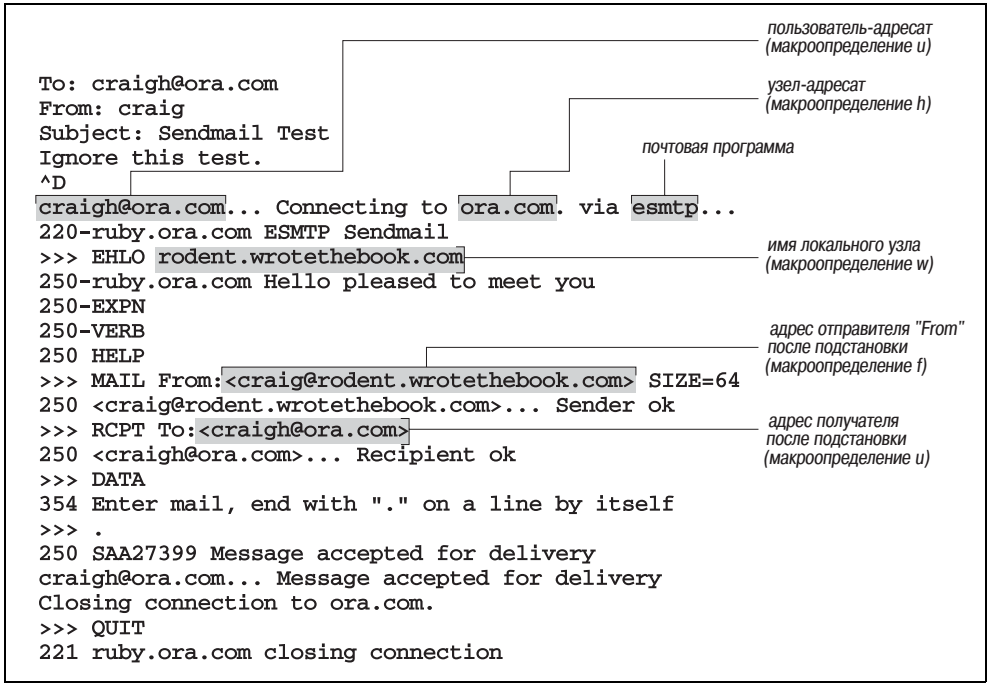


Рис. 10.5. *Подробная диагностика*

товым сервером домена *ora.com*; то есть MX-запись для *ora.com* указывает на *ruby.ora.com*.

Сообщения EHLO указывают, что узлы *rodent* и *ruby* используют протокол ESMTP (Extended Simple Mail Transfer Protocol).

Все замечательно работает! Мы могли бы завершить тестирование прямо сейчас и использовать текущие настройки. Но, как и большинство компьютерщиков, мы не можем остановиться в своем стремлении «сделать лучше».

Адрес From:, *craig@rodent.wrotethebook.com*, очевидно, является допустимым, но его формат нас не совсем устраивает. Мы хотим, чтобы люди могли присылать сообщения на адрес вида *firstname.lastname@domain*, а не *user@host.domain*, то есть получить такие же настройки, как созданные при помощи нескольких строк кода `m4` ранее в этой главе. Здесь мы создадим те же настройки, чтобы привести пример использования различных инструментов отладки, встроенных в sendmail. Однако если вы действительно намереваетесь существенно изменить настройки sendmail, воспользуйтесь макроопределениями `m4`.

Изменения, вносимые в *sendmail.cf*, обычно незначительны и располагаются ближе к началу файла, в разделе локальной информации (Local Information). Изучив внимательно этот раздел, мы получим сведения, необходимые для разрешения части нашей проблемы настройки.

Пусть мы и не знаем, что такое «masquerading» (сокрытие), комментарии классов E и M, а также макроопределения M позволяют нам догадаться, что значение макроопределения M будет использовано при подстановке имени узла.[1] В частности, комментарий «names that should be exposed as from this host, even if we masquerade» позволил мне предположить, что сокрытие (masquerading) скрывает имя узла. Исходя из этой догадки мы установили значение M следующим образом:

```
# who I masquerade as (null for no masquerading) (see also $=M) DMwrotethebook.com
```

Есть ли у нас уверенность, что установка значения макроопределения M скроет имя узла? Нет. Однако изменение значения в файле *test.cf* и еще один раунд тестирования ничем не повредит. Запуск программы для тестирования с тестовыми настройками никак не влияет на работу демона sendmail, запущенного командой `sendmail -bd -q1h` из загрузочного сценария. Только экземпляр sendmail, запущенный с аргументом `-Ctest.cf`, будет использовать в качестве файла настройки *test.cf*.

Тестирование правил подстановки

В начальном тесте адрес From: поступил на вход sendmail в виде *craig* и был преобразован в *craig@rodent.wrotethebook.com*. Очевидно, произошла подстановка. Теперь мы протестируем собственно правила подстановки и выясним, повлияли ли на процесс подстановки изменения, внесенные в макроопределение M. Прежде всего, необходимо выяснить, какие правила использовались для переписывания адреса. Чтобы получить дополнительную информацию, выполним sendmail с ключом `-bt`.

В результате выполнения команды выводится приглашение ввода – символ «больше» (>). В ответ на приглашение наберите одну из команд тестирования, описанных в табл. 10.6.

Таблица 10.6. Команды тестирования sendmail

Команда	Назначение
ruleset[, *ruleset*…] *address*	Обрабатывает адрес *address* указанными наборами правил
.D*mvalue*	Присваивает значение *value* макропеременной *m*
.C*cvalue*	Добавляет значение *value* в класс *c*
=S*ruleset*	Выводит правила из набора *ruleset*
=M	Выводит определения почтовых программ
-d*value*	Устанавливает значение *value* для флага отладки
$*m*	Выводит значение макроопределения *m*

[1] В исходном файле m4 мы указали на необходимость сокрытия имени при помощи команды `MASQUERADE_AS(wrotethebook.com)`.

Команда	Назначение
`$=`*c*	Выводит содержимое класса *c*
`/mx` *host*	Отображает MX-записи для узла *host*
`/parse` *address*	Возвращает тройку значений *почтовая программа*/*узел*/*пользователь* для указанного адреса *address*
`/try` *mailer address*	Обрабатывает адрес *address* для почтовой программы *mailer*
`/tryflags` *flags*	Устанавливает адрес для обработки `/parse` или `/try` в `H` (Header), `E` (Envelope), `S` (Sender) или `R` (Recipient)
`/canon` *hostname*	Выполняет приведение имени узла *hostname* к каноническому виду
`/map` *mapname key*	Выводит значение для ключа *key* из базы данных *mapname*
`/quit`	Завершает работу в режиме тестирования адресов

Простейший тест – имя набора правил и следующий за ним адрес электронной почты. Адрес представляет тестовые данные, а имя – набор правил, который следует протестировать. Адрес выбрать просто – тот, что был неправильно переписан. Но как определить, какой набор правил использовать?

Используйте рис. 10.4 для выбора наборов правил. Набор правил *canonify* применяется ко всем адресам. За ним следуют наборы для различных типов адресов (адреса доставки, адреса отправителя либо адреса получателя). Более того, наборы для адресов отправителя и получателя изменяются в зависимости от почтовой программы, выполняющей доставку почты. Окончательно все адреса обрабатываются набором *final*.

В определении наборов правил, используемых для обработки адреса, участвуют две переменные: тип адреса и почтовая программа, через которую передается письмо. Адреса бывают трех типов: адреса доставки, адреса получателей и адреса отправителей. Тип адреса известен, поскольку мы сами выбираем адрес для тестирования. В тестовых письмах нас интересует адрес отправителя. Какая используется почтовая программа – определяет адрес доставки. Один из способов определить, какая почтовая программа выполняет доставку тестового письма, – выполнить sendmail с ключом `-bv` и адресом доставки в качестве аргумента:

```
# sendmail -bv craigh@ora.com
craigh@ora.com... deliverable: mailer esmtp, host ora.com.,
      user craigh@ora.com
```

Определив почтовую программу, мы можем использовать sendmail с ключом `-bt` для обработки адреса отправителя, From:. Существует два типа адреса отправителя: адрес отправителя на «конверте» и адрес отправителя в заголовке сообщения. Адрес заголовка сообщения фигурирует в строке From:, передаваемой с сообщением в сеансе передачи SMTP DATA. Этот адрес вы можете видеть в заголовках почтовых сообщений при просмотре в программе чтения почты. Адрес «конверта» используется во взаимодействиях уров-

ня протокола SMTP. sendmail позволяет нам наблюдать обработку адресов и того и другого типа:

```
# sendmail -bt -Ctest.cf
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
> /tryflags HS
> /try esmtp craig
Trying header sender address craig for mailer esmtp
canonify           input: craig
Canonify2          input: craig
Canonify2        returns: craig
canonify         returns: craig
1                  input: craig
1                returns: craig
HdrFromSMTP        input: craig
PseudoToReal       input: craig
PseudoToReal     returns: craig
MasqSMTP           input: craig
MasqSMTP         returns: craig < @ *LOCAL* >
MasqHdr            input: craig < @ *LOCAL* >
MasqHdr          returns: craig < @ wrotethebook . com . >
HdrFromSMTP      returns: craig < @ wrotethebook . com . >
final              input: craig < @ wrotethebook . com . >
final            returns: craig @ wrotethebook . com
Rcode = 0, addr = craig@wrotethebook.com
> /tryflags ES
> /try esmtp craig
Trying envelope sender address craig for mailer esmtp
canonify           input: craig
Canonify2          input: craig
Canonify2        returns: craig
canonify         returns: craig
1                  input: craig
1                returns: craig
EnvFromSMTP        input: craig
PseudoToReal       input: craig
PseudoToReal     returns: craig
MasqSMTP           input: craig
MasqSMTP         returns: craig < @ *LOCAL* >
MasqEnv            input: craig < @ *LOCAL* >
MasqEnv          returns: craig < @ rodent . wrotethebook . com . >
EnvFromSMTP      returns: craig < @ rodent . wrotethebook . com . >
final              input: craig < @ rodent . wrotethebook . com . >
final            returns: craig @ rodent . wrotethebook . com
Rcode = 0, addr = craig@rodent.wrotethebook.com
> /quit
```

Команда `/tryflags` определяет тип адреса для обработки командой `/try` или `/parse`. Для команды `/tryflags` доступно четыре флага: `S` (sender) – отправитель, `R` (recipient) – получатель, `H` (header) – заголовок и `E` (envelope) – конверт. Сочетанием двух из этих флагов в первой команде `/tryflags` мы обраба-

тываем адрес отправителя из заголовка (HS). Команда /try предписывает sendmail обработать адрес для передачи через конкретную почтовую программу. В данном примере адрес электронной почты *craig* обрабатывается для почтовой программы *esmtp*. Сначала мы обрабатываем его как адрес отправителя из заголовка, а затем как адрес отправителя с конверта. По этому тесту можно определить, что значение из макроопределения M используется для подстановки адреса отправителя в заголовке сообщения, но не адреса отправителя на конверте.

В результате проведенных тестов мы видим, что значение макроопределения M заменяет имя узла в адресе отправителя из заголовка сообщения – что нам и требовалось. Имя узла в адресе отправителя на конверте не изменяется. Обычно это приемлемо. Однако мы хотим создать в точности такие настройки, как в примере m4. Команда FEATURE(masquerade_envelope) из примера m4 приводит к изменению адреса отправителя на конверте. Следовательно, в наших настройках также должна происходить подстановка.

Разница в обработке адресов сообщения и конверта состоит в том, что первые обрабатываются набором правил HdrFromSMTP, тогда как вторые – набором правил EnvFromSMTP. Тесты показывают, что оба набора правил используют пересекающиеся наборы правил. Есть одно расхождение: набор правил HdrFromSMTP вызывает набор правил MasqHdr, а набор правил EnvFromSMTP вызывает набор MasqEnv. Тесты также показывают, что набор MasqHdr выполняет для адреса отправителя сообщения подстановку в соответствии с нашими пожеланиями, тогда как MasqEnv не выполняет такую подстановку для адреса отправителя с конверта. Ниже приводится код *test.cf* для наборов правил MasqEnv:

```
######################################################################
###  Ruleset 94 -- convert envelope names to masquerade form    ###
######################################################################
SMasqEnv=94
R$* < @ *LOCAL* > $*    $: $1 < @ $j . > $2
```

Очевидно, набор правил MasqEnv не делает то, что нам нужно, в отличие от набора MasqHdr. Быстрое изучение набора MasqEnv позволяет понять, что он не содержит ни единой ссылки на макроопределение M. Но при этом в самом его начале присутствует комментарий «do masquerading», благодаря которому можно догадаться, что здесь должно выполняться сокрытие для адреса. Решение нашей задачи – добавить к набору правил MasqEnv строку, вызывающую набор правил MasqHdr, который в действительности и производит необходимую обработку. Измененный набор правил показан ниже:

```
######################################################################
###  Ruleset 94 -- convert envelope names to masquerade form    ###
######################################################################
SMasqEnv=94
R$+                     $: $>93 $1          do masquerading
R$* < @ *LOCAL* > $*    $: $1 < @ $j . > $2
```

Отладка файла *sendmail.cf* – это скорее искусство, нежели наука. Решение о добавлении в набор правил MasqEnv строки вызова набора MasqHdr – немногим более чем догадка. И тестирование – единственный способ проверить ее. Мы повторно выполнили `sendmail -bt -Ctest.cf`, чтобы протестировать адреса *craig*, *craig@rodent* и *craig@localhost* при помощи команды `/try esmtp`. Все тесты завершились успешно – переписыванием различных исходных адресов в конечный адрес *craig@wrotethebook.com*. Затем мы повторили тест отправки почты командой `sendmail -v -t -Ctest.cf`. Только когда все тесты отработали успешно, мы окончательно уверились в своей догадке и можем переходить к следующей задаче – подстановке имени и фамилии пользователя вместо регистрационного имени пользователя в адресе электронной почты.

Использование файлов ключей в sendmail

Последняя из добавленных нами в исходный файл `m4` возможностей – FEATURE (genericstable) – включает процесс работы с базой данных в задачу преобразования электронного адреса (регистрационного имени пользователя в имя и фамилию пользователя). Чтобы повторить здесь те же действия, создайте текстовый файл регистрационных имен и имен/фамилий пользователей. На основе этого файла создается база данных посредством `makemap`.[1]

```
# cd /etc/mail
# cat realnames
dan Dan.Scribner
tyler Tyler.McCafferty
pat Pat.Stover
willy Bill.Wright
craig Craig.Hunt
# makemap hash realnames < realnames
```

Когда база данных создана, создайте определения для sendmail. С этой целью воспользуйтесь командой `K`. Чтобы воспользоваться только что созданной нами базой данных, добавьте следующие строки в раздел локальной информации (Local Information) файла *sendmail.cf*:

```
# define a database to map login names to firstname.lastname
Krealnames hash /etc/mail/realnames
```

Команда `K` определяет *realnames* в качестве внутреннего имени sendmail для этой базы данных. Кроме того, она указывает, что база данных имеет тип *hash*, а ее полное имя – */etc/realnames*. sendmail добавляет к именам файлов баз данных нужные расширения исходя из их типа, так что о расширениях можете не беспокоиться.

Наконец, мы добавляем новое правило, в котором база данных используется для перезаписи адресов. Добавляем мы его в наборы правил EnvFromSMTP и HdrFromSMTP – сразу после строк в этих наборах, вызывающих набор

[1] Более подробные сведения о `makemap` приводятся в разделе, посвященном `m4`.

MasqHdr. В этом случае наше новое правило обрабатывает адрес сразу после того, как закончится его обработка набором правил MasqHdr.

```
# when masquerading convert login name to firstname.lastname
R$-<@$M.>$*     $:$(realnames $1 $)<@$M.>$2     user=>first.last
```

Данное правило обрабатывает вывод набора правил MasqHdr, отвечающего за перезапись раздела узла в адресе. Адреса, соответствующие условиям перезаписи раздела узла, также должны подвергаться перезаписи раздела пользователя. Взгляните на вывод набора правил MasqHdr из приведенного ранее теста. Адрес *craig*<@*wrotethebook.com*.> соответствует шаблону `$-<@$M.>$*`. В адресе ровно одна лексема (*craig*) перед литералом <@, за которым следует значение `M` (wrotethebook.com), литерал .>, а затем нулевое число лексем.

Секция преобразования этого правила использует первую лексему ($1) исходного адреса в качестве ключа базы данных *realnames*, на что указывает синтаксис `$:$(realnames $1 $)`. Для адреса *craig*<@*wrotethebook.com*> $1 имеет значение *craig*. Запрос по этому индексу, обращенный к базе данных *realnames*, приводимой в начале данного раздела, возвращает значение *Craig.Hunt*. Это значение предваряет литерал <@, значение макроопределения `M` ($M), литерал .>, а также значение $2, как предписывается секцией преобразования `<@$M.>$2`. В результате действия этого нового правила регистрационное имя пользователя в адресе преобразуется в настоящее имя и фамилию пользователя.

Когда это правило добавлено в наборы EnvFromSMTP и HdrFromSMTP, тест дает следующие результаты:

```
# sendmail -bt -Ctest.cf
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
> /tryflags HS
> /try esmtp craig
Trying header sender address craig for mailer esmtp
canonify           input: craig
Canonify2          input: craig
Canonify2        returns: craig
canonify         returns: craig
1                  input: craig
1                returns: craig
HdrFromSMTP        input: craig
PseudoToReal       input: craig
PseudoToReal     returns: craig
MasqSMTP           input: craig
MasqSMTP         returns: craig < @ *LOCAL* >
MasqHdr            input: craig < @ *LOCAL* >
MasqHdr          returns: craig < @ wrotethebook . com . >
HdrFromSMTP      returns: Craig . Hunt < @ wrotethebook . com . >
final              input: Craig . Hunt < @ wrotethebook . com . >
final            returns: Craig . Hunt @ wrotethebook . com
Rcode = 0, addr = Craig.Hunt@wrotethebook.com
```

```
> /tryflags ES
> /try esmtp craig
Trying envelope sender address craig for mailer esmtp
canonify           input: craig
Canonify2          input: craig
Canonify2        returns: craig
canonify         returns: craig
1                  input: craig
1                returns: craig
EnvFromSMTP        input: craig
PseudoToReal       input: craig
PseudoToReal     returns: craig
MasqSMTP           input: craig
MasqSMTP         returns: craig < @ *LOCAL* >
MasqEnv            input: craig < @ *LOCAL* >
MasqHdr            input: craig < @ *LOCAL* >
MasqHdr          returns: craig < @ wrotethebook . com . >
MasqEnv          returns: craig < @ wrotethebook . com . >
EnvFromSMTP      returns: Craig . Hunt < @ wrotethebook . com . >
final              input: Craig . Hunt < @ wrotethebook . com . >
final            returns: Craig . Hunt @ wrotethebook . com
Rcode = 0, addr = Craig.Hunt@wrotethebook.com
> /quit
```

Если тесты не приводят к получению нужных результатов, убедитесь, что вы правильно набрали новые правила подстановки и корректно создали базу данных. Может возникнуть также следующее сообщение об ошибке:

```
test.cf: line 116: readcf: map realnames: class hash not available
```

Сообщение свидетельствует о том, что ваша система не поддерживает ассоциативные базы данных. Можно попробовать изменить тип базы данных в строке команды `K` и повторно выполнить `sendmail -bt`, пока не будет найден тип, с которым работает sendmail. В этом случае необходимо также повторно выполнить `makemap` для каждого нового типа базы данных. Если ваша версия sendmail не поддерживает никакие типы баз данных, обратитесь к приложению E за информацией о перекомпиляции sendmail с поддержкой баз данных.

Обратите внимание, что все изменения, вносимые непосредственно в *sendmail.cf* во второй половине этой главы (сокрытие адреса отправителя, сокрытие адреса конверта, преобразование имен пользователей), в исходном файле `m4` отражены всего лишь тремя строками. Эти примеры демонстрируют использование инструментов тестирования sendmail. Если вам действительно нужны новые, специальные настройки, воспользуйтесь `m4`. Настройки sendmail проще всего сопровождать и развивать посредством исходного файла `m4`.

Резюме

sendmail передает и получает SMTP-почту, обрабатывает почтовые псевдонимы, выступает в качестве интерфейса между пользовательскими почто-

выми программами и агентами доставки почты. Демон sendmail запускается в процессе загрузки системы и обрабатывает входящую почту SMTP. Определения псевдонимов sendmail хранятся в файле *aliases*. Правила взаимодействия пользовательских программ и агентов доставки почты могут быть сложными; для их определения в sendmail используется файл *sendmail.cf*.

Настройка *sendmail.cf* – самый трудный этап в создании сервера sendmail. Синтаксис команд файла очень сжатый и затрудняет чтение. Задачу упрощают доступные примеры файлов *sendmail.cf*. В составе большинства систем присутствует файл настройки, созданный разработчиком, а в дистрибутив sendmail входят дополнительные примеры. Примеры файлов настройки sendmail необходимо обрабатывать макропроцессором `m4`. Изменения, вносимые в подходящий файл примера, минимальны. Практически все изменения, требуемые для завершения настройки, производятся в начале файла и связаны с указанием сведений о локальной системе, таких как имя узла и имя сервера пересылки почты. sendmail предоставляет инструмент для диалогового тестирования, позволяющий проверить настройки перед вводом их в эксплуатацию.

sendmail – это большая, сложная служба, настолько важная, что заслуживает отдельной главы. Еще одна важная служба – веб-служба, реализованная в большинстве систем Unix на основе сервера Apache. Сложный синтаксис настройки Apache – тема следующей главы.

11

Настройка Apache

- *Установка сервера Apache*
- *Настройка сервера Apache*
- *Постигаем файл httpd.conf*
- *Безопасность веб-сервера*
- *Шифрование*
- *Управление веб-сервером*

Веб-серверы предлагают основной способ доставки информации в сети IP. Среда Web известна более всего за распространение информации в глобальной сети Интернет, однако она может эффективно применяться для передачи информации не только внешним клиентам, но и сотрудникам организации. Во всех сетях, за исключением самых маленьких, хорошо организованный веб-сервер может приносить пользу: предоставлять информацию о продукции и предлагать услуги поддержки внешним клиентам, а также координировать и распространять информацию между пользователями внутри организации. Web-среда – непревзойденное средство доставки информации по требованию конечных пользователей.

В системах Unix создание веб-серверов выполняется в основном при помощи пакета Apache. Apache – это распространяемый бесплатно веб-сервер, ведущий свое начало от веб-сервера NCSA (National Center for Supercomputer Applications, Национальный суперкомпьютерный центр), первого веб-сервера, получившего широкое распространение. Столь «древние» корни сервера Apache означают годы и годы тестирования и разработки. Сегодня Apache – самый распространенный веб-сервер сети Интернет, и, вероятно, именно его вы будете использовать для создания сервера на базе Unix.

В этой главе мы сосредоточимся на установке и настройке сервера Apache. Настройка Apache может показаться более сложной, чем она является в действительности, из-за большого числа доступных параметров. Эта глава – пример того, как быстро и просто настроить сервер Apache и ввести его в строй.

Мы изучим настройки и администрирование службы, однако оформление информационного наполнения выходит за пределы настоящей книги. Если вам повезло, оформлением займутся ваши коллеги веб-дизайнеры, а в противном случае не исключено, что придется самостоятельно решать эту твор-

ческую задачу. Издательство O'Reilly выпустило ряд полезных книг по этой теме: Чак Муссиано (Chuck Musciano) и Билл Кеннеди (Bill Kennedy) «HTML and XHTML: The Definitive Guide»[1] и Дженнифер Нидерст (Jennifer Niederst) «Web Design in a Nutshell».[2]

Установка сервера Apache

Сервер Apache входит в состав многих систем Unix. Часто Apache устанавливается уже в процессе начальной установки операционной системы. Например, начальная установка системы Red Hat представляет пользователю возможность установить сервер, щелкнув по пиктограмме, обозначенной надписью *Apache Web Server*.

Пользователи часто выбирают установку сервера Apache, даже если не планируют создавать веб-сервер. Вы можете с удивлением обнаружить, что сервер Apache установлен и работает на обычной рабочей станции. Проверьте этот факт командой `ps`:

```
$ ps ax | grep httpd
  321  ?  S    0:00 httpd
  324  ?  S    0:00 httpd
  325  ?  S    0:00 httpd
  326  ?  S    0:00 httpd
  329  ?  S    0:00 (httpd)
  330  ?  S    0:00 (httpd)
  331  ?  S    0:00 (httpd)
  332  ?  S    0:00 (httpd)
  333  ?  S    0:00 (httpd)
  334  ?  S    0:00 (httpd)
  335  ?  S    0:00 (httpd)
 2539  p1 D    0:00 grep http
```

Демон Apache, реализующий работу веб-служб, носит название Hypertext Transport Protocol daemon (`httpd`, демон транспортного протокола гипертекста). Воспользуйтесь `ps` – командой состояния процессов (process status), чтобы получить список всех процессов системы, а также командой `grep`, чтобы вывести только процессы с именем `httpd`. Выполнение такой проверки на только что установленной операционной системе может показать, что сервер Apache тоже установлен и функционирует.

В таком случае запустите веб-броузер и наберите «localhost» в строке адреса. Результат для системы Red Hat 7 представлен на рис. 11.1. Сервер Apache не только установлен и функционирует, он еще и настроен, и даже отвечает веб-страницей. Пользователи настольных машин под управлением Linux

---

[1] Муссиано Ч., Кеннеди Б. «HTML и XHTML. Подробное руководство» – Пер. с англ. – СПб: Символ-Плюс, 2002.

[2] Нидерст Дж. «Веб-мастеринг для профессионалов» – Пер. с англ. – СПб: Питер, 2001.

Рис. 11.1. *Стандартная веб-страница сервера Apache*

иногда с удивлением обнаруживают на своих компьютерах полностью функциональный веб-сервер. Разумеется, если вы администратор системы веб-сервера, именно такой результат и представляет интерес – Apache установлен, запущен, работает.

Если сервер Apache не установлен в системе, необходимо установить соответствующий пакет. Простейший способ установить дополнительный пакет в системе Linux – воспользоваться менеджером пакетов. Существует несколько качественных менеджеров пакетов. Большинство Linux-систем поддерживают менеджер пакетов Red Hat Package Manager (`rpm`), которым мы и воспользуемся в следующем примере.

Работа с менеджером пакетов Red Hat

Используйте менеджер пакетов Red Hat для установки нужных программных пакетов, удаления ненужных программных пакетов, а также для получения перечня установленных пакетов. `rpm` предлагает многочисленные ключи для разработчиков, создающих пакеты, однако для администратора сети работа с `rpm` может ограничиваться тремя основными командами:

`rpm --install` *пакет*

Ключ `--install` устанавливает программный пакет.

`rpm --uninstall` *пакет*

Ключ `--uninstall` удаляет программный пакет.

`rpm --query`

Ключ `--query` выводит сведения об уже установленном программном пакете. Используйте ключ `--all` совместно с ключом `--query`, чтобы получить список всех установленных пакетов.

Чтобы установить пакет при помощи `rpm`, необходимо знать его имя. Чтобы узнать полное имя пакета Apache, смонтируйте компакт-диск дистрибутива Linux и загляните в каталог *RPMS*. Вот пример для системы Red Hat 7.2:

```
$ cd /mnt/cdrom/RedHat/RPMS
$ ls *apache*
apache-1.3.20-16.i386.rpm
apacheconf-0.8.1-1.noarch.rpm
```

В данном примере предполагается, что компакт-диск смонтирован в точке */mnt/cdrom*. Мы видим, что в состав дистрибутива Red Hat входит два пакета Apache: пакет веб-сервера и средство настройки для Red Hat. Чтобы добавить в систему веб-сервер, установите пакет *apache-1.3.20-16.i386.rpm* следующей командой:

```
# rpm --install apache-1.3.20-16.i386.rpm
```

После завершения операции проверьте, что пакет установлен:

```
$ rpm --query apache
apache-1.3.20-16
```

Теперь следует убедиться, что демоны `httpd` запускаются при загрузке системы. В системе Red Hat это действие выполняется в сценарии */etc/init.d/httpd*. Воспользуйтесь командой `chkconfig` или аналогичной, чтобы включить сценарий в процесс загрузки. В следующем примере сценарий запуска *httpd* включается в процесс загрузки для уровней исполнения 3 и 5:

```
# chkconfig --list httpd
httpd           0:off   1:off   2:off   3:off   4:off   5:off   6:off
# chkconfig --level 35  httpd on
# chkconfig --list httpd
httpd           0:off   1:off   2:off   3:on    4:off   5:on    6:off
```

Первая команда `chkconfig` перечисляет состояние сценария *httpd* для всех уровней исполнения. Полученный результат показывает, что *httpd* отключен на всех семи уровнях, то есть сценарий не выполняется ни при каких обстоятельствах. Мы хотим, чтобы веб-сервер запускался на уровне 3, то есть на уровне многопользовательского режима, а также на уровне 5, который для данной системы Red Hat является уровнем работы по умолчанию. Вторая команда `chkconfig` решает поставленную задачу. Ключ `--level` предписы-

вает изменить настройки для уровней 3 и 5 – обратите внимание, что номера уровней не разделяются пробелами. Аргумент `httpd on` предписывает выполнять сценарий *httpd* на этих уровнях исполнения. Последняя команда `chkconfig` снова перечисляет состояние сценария *httpd* для всех уровней исполнения. На этот раз мы видим, что сценарий *httpd* будет выполнен на уровнях 3 и 5.

При следующей перезагрузке данной системы Red Hat будет запущен веб-сервер. Чтобы запустить веб-сервер без перезагрузки, необходимо выполнить сценарий *httpd* из командной строки:

```
# /etc/init.d/httpd start
Starting httpd:                                            [  OK  ]
```

Установка Apache в системе Linux очень проста. Часто сервер устанавливается в процессе начальной установки операционной системы, а в противном случае может быть установлен с диска дистрибутива системы. Установка Apache в Solaris не менее проста, поскольку в Solaris 8 сервер Apache является частью операционной системы. Если же в вашей системе Unix отсутствует пакет Apache, его можно получить из сети Интернет.

Как получить пакет Apache

Сервер Apache доступен в виде исходных текстов и готовых исполняемых файлов по адресу *http://www.apache.org*. Исходный текст Apache доступен для систем Unix в сжатых tar- или zip-архивах. Можно загрузить и скомпилировать исходный текст, но проще всего использовать готовую исполняемую версию Apache. На рис. 11.2 отражены лишь некоторые из версий Unix, для которых доступны собранные демоны `httpd`.

Дистрибутивы рассортированы по операционным системам. Предположим, нужен дистрибутив для системы FreeBSD. Перейдите по ссылке *freebsd*, чтобы получить список архивов. Каждый архив предназначен для совершенно определенной версии системы FreeBSD и содержит исполняемые файлы Apache. Выберите архив, который подходит используемой версии FreeBSD, и сохраните его в рабочий каталог. Если в системе уже есть более старая версия Apache, то создайте резервную копию существующего демона. Извлеките новую версию при помощи команды `tar`. Пакет установлен и готов к работе с существующими файлами настройки.

Настройка сервера Apache

В настройке Apache обычно участвуют следующие три файла:

httpd.conf

Главный файл настройки. Традиционно содержит параметры настройки протокола HTTP и работы сервера. Этот файл обрабатывается в первую очередь.

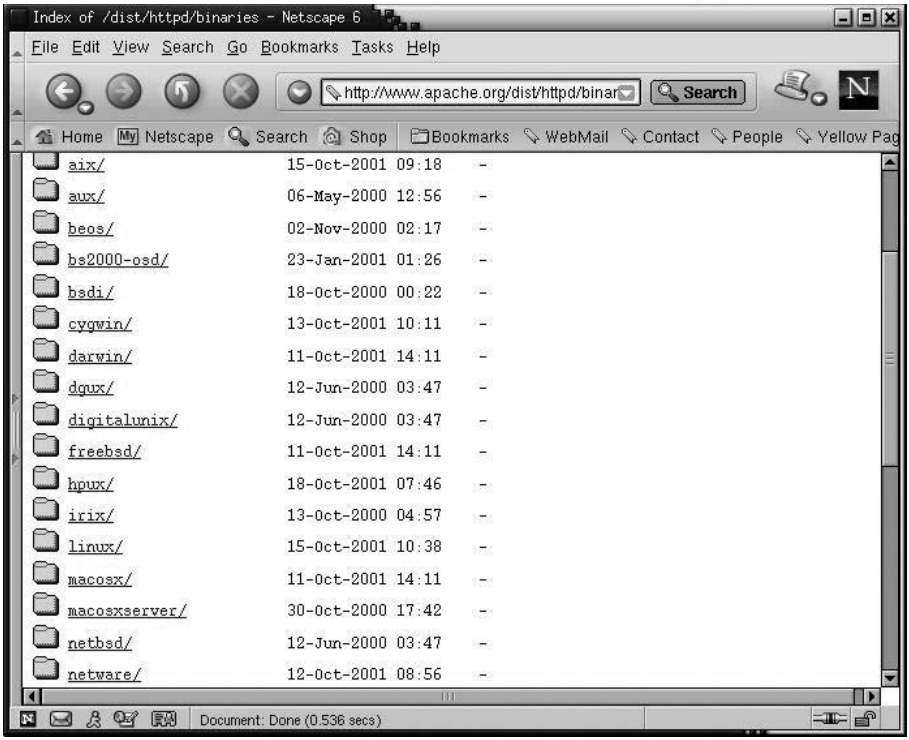


Рис. 11.2. *Дистрибутивы исполняемых файлов на веб-сайте проекта Apache*

srm.conf

Традиционно содержит настройки, помогающие серверу отвечать на запросы клиентов. Настройки позволяют учесть различные MIME-типы, способы оформления вывода, расположение документов HTTP и сценариев CGI (Common Gateway Interface). Этот файл обрабатывается вторым.

access.conf

Данный файл традиционно регулирует управление доступом к серверу и хранимой сервером информации. Обрабатывается последним.

Все три файла имеют сходное строение: состоят из обычного ASCII-текста, комментарии начинаются символом # и довольно обширны. Большинство инструкций представлено парами параметр/значение.

Мы говорим, что эти файлы *традиционно* используются для настройки Apache, однако сегодня часто встречается и другая практика. Функциональность трех описанных файлов частично совпадает. Пребывая в полной уверенности относительно того, где хранится определенное значение, вы можете с удивлением обнаружить его в другом файле. В действительности любая инструкция настройки Apache может фигурировать в любом из файлов на-

стройки – изначальное деление файлов по функциям сервера, данных и безопасности было, в большой степени, необязательным. Некоторые администраторы следуют традициям, но чаще всего настройки в полном объеме хранятся в файле *httpd.conf*. Это предпочтительный подход, именно его мы используем в данной главе.

Расположение файла *httpd.conf* зависит от операционной системы. В системе Solaris он хранится в каталоге */etc/apache*; в системе Red Hat – в каталоге */etc/httpd/conf*; а в системах Caldera – в каталоге */etc/httpd/apache/conf*. Страница руководства, посвященная Apache, должна содержать сведения о том, где в данной системе хранится файл *httpd.conf*; в противном случае просто обратитесь к сценарию, запускающему `httpd` при загрузке системы. Расположение файла *httpd.conf* в этом файле определено переменной сценария либо аргументом ключа `-f` в командной строке `httpd`. Разумеется, есть еще один способ найти этот файл – при помощи команды `find`, как в следующем примере для Caldera Linux:

```
# find / -name httpd.conf -print
/etc/httpd/apache/conf/httpd.conf
```

Найдите файл *httpd.conf* и отредактируйте его в соответствии с вашей системой. Файл настройки Apache имеет большой размер и сложен для восприятия; однако он создается заранее, так что от администратора требуется лишь изменение отдельных значений. В файле *httpd.conf* укажите адрес электронной почты администратора веб-сервера при помощи параметра ServerAdmin, а также имя узла сервера при помощи параметра ServerName. Столь простые изменения, скорее всего, подготовят файл настройки `httpd`, поставляемый с операционной системой, к работе. Взглянем на пример для Solaris 8.

Настройка Apache в Solaris

Первый шаг в настройке Apache для системы Solaris – скопировать файл *httpd.conf-example* в *httpd.conf*:

```
# cd /etc/apache
# cp httpd.conf-example httpd.conf
```

При помощи редактора укажите корректные значения ServerAdmin и ServerName. В данном примере для Solaris мы изменим строку ServerAdmin:

```
ServerAdmin you@your.address
```

На такую:

```
ServerAdmin webmaster@www.wrotethebook.com
```

А параметр ServerName:

```
#ServerName new.host.name
```

следующим образом:

```
ServerName www.wrotethebook.com
```

После сохранения изменений можно запускать сервер. Простейший способ решения этой задачи в Solaris – выполнить сценарий */etc/init.d/apache*. Сценарий принимает аргументы `start`, `restart` и `stop`. Поскольку `httpd` еще не работает, демон не может быть остановлен (`stop`) или перезапущен (`restart`), так что воспользуемся командой `start`:

```
# /etc/init.d/apache start
httpd starting.
# ps -ef | grep '/httpd'
  nobody   474   473  0 12:57:27 ?        0:00 /usr/apache/bin/httpd
  nobody   475   473  0 12:57:27 ?        0:00 /usr/apache/bin/httpd
  nobody   476   473  0 12:57:27 ?        0:00 /usr/apache/bin/httpd
    root   473     1  0 12:57:26 ?        0:00 /usr/apache/bin/httpd
  nobody   477   473  0 12:57:27 ?        0:00 /usr/apache/bin/httpd
  nobody   478   473  0 12:57:27 ?        0:00 /usr/apache/bin/httpd
    root   501   358  0 13:10:04 pts/2    0:00 grep /httpd
```

Выполнив сценарий запуска *apache*, воспользуйтесь командой `ps`, чтобы убедиться, что демон `httpd` запущен.[1] В данном примере работают несколько копий демона, точно так же, как в вышеприведенном примере для Linux. Эта группа демонов называется пулом (swarm), и позже мы изучим инструкции настройки Apache, управляющие размером пула.

Теперь, когда демоны работают, запустите броузер. Набрав «localhost» в строке адреса, вы должны увидеть картину, подобную представленной на рис. 11.3.

Наш сервер Solaris Apache запущен, работает и предоставляет данные. Разумеется, своим клиентам мы намереваемся предоставлять совсем не эти данные. Решить проблему можно двумя способами: поместить нужные данные в каталог, задействованный в работе сервера, либо настроить сервер на использование каталога, в котором хранятся нужные данные.

Инструкция DocumentRoot указывает серверу каталог, в котором хранятся содержательные веб-страницы. По умолчанию сервер Solaris читает веб-страницы из каталога */var/apache/htdocs*, в чем можно убедиться, сверившись со значением DocumentRoot из файла *httpd.conf*:

```
# grep '^DocumentRoot' httpd.conf
DocumentRoot "/var/apache/htdocs"
# ls /var/apache/htdocs
apache_pb.gif  index.html
```

Каталог */var/apache/htdocs* содержит лишь два файла. Графический GIF-файл – это изображение пера Apache, расположенное внизу страницы (рис. 11.3). В файле *index.html* содержится HTML-документ, интерпретация которого и приводит к отображению такой веб-страницы. По умолчанию

[1] Если в системе Solaris работает сервер *AnswerBook2*, `ps` может отражать присутствие демона DynaWeb (`dwhttpd`), который применяется для организации доступа к документации *AnswerBook*.

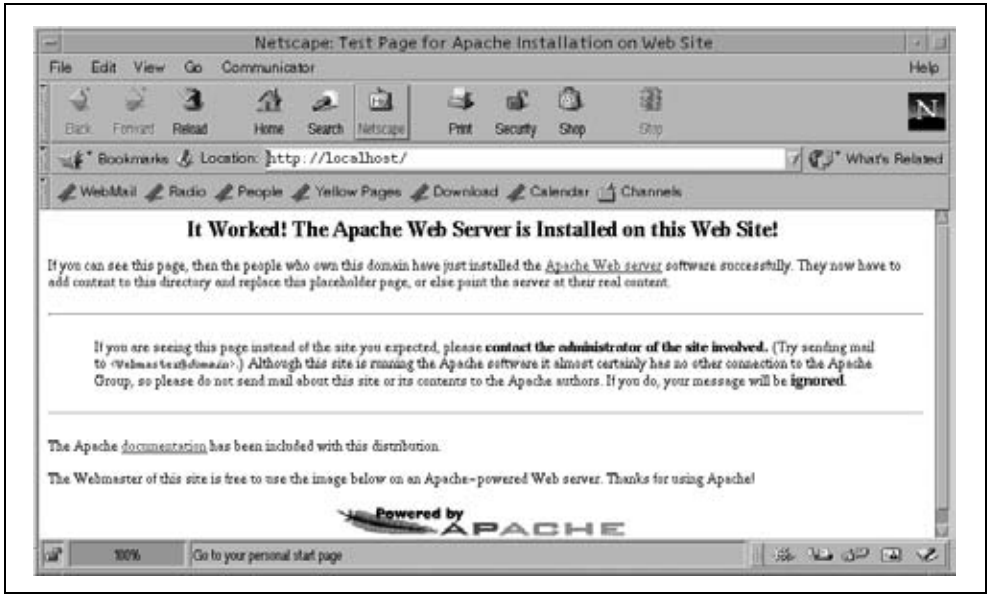


Рис. 11.3. *Стандартная веб-страница для сервера Solaris*

Apache ищет файл с именем *index.html* и использует его в качестве «домашней страницы», если в запросе не фигурирует конкретная страница сайта. Вы можете разместить собственный файл *index.html* в этом каталоге – вместе с прочими дополнительными файлами и каталогами, и сервер Apache начнет распространять ваши данные. Как вариант можно отредактировать файл *httpd.conf* – изменить значение инструкции DocumentRoot, указав каталог, где хранятся данные. Выбор за вами. В любом случае необходимо создать HTML-документы, которые будут отображаться веб-сервером.

Сервер Solaris может функционировать уже после изменения двух или трех инструкций настройки, однако администратор должен разбираться во всех без исключения параметрах настройки Apache. Учитывая важность веб-служб для большинства сетей, невозможно игнорировать тонкости работы сервера Apache. Эффективная отладка некорректно настроенного сервера подразумевает полное понимание содержимого файла *httpd.conf*. Следующие разделы подробно рассказывают об этом файле.

Постигаем файл httpd.conf

Бывает полезно помнить настройки по умолчанию, когда необходимо исправить настройки незнакомой системы. В данном разделе мы изучим значения, присутствующие в стандартном файле настройки системы Solaris 8. (Стандартный файл настройки для Solaris 8 приведен в приложении F.)

Здесь мы сосредоточим свое внимание на инструкциях, которые на практике используются при настройке Apache в Solaris 8, а также ряде инструк-

ций, связанных с важными возможностями сервера. Существуют и другие инструкции, о которых мы не будем говорить. Дополнительную информацию по любой инструкции можно получить из многих источников. Полный файл *httpd.conf* содержит многочисленные замечательные комментарии, поясняющие назначение каждой инструкции. В сети документация по серверу доступна на веб-сайте Apache (*http://www.apache.org*). Что касается книг, можно порекомендовать два превосходных издания, посвященных настройке Apache: Бен и Питер Лори (Ben and Peter Laurie) «Apache: The Definitive Guide», издательство O'Reilly и Чарльз Олдс (Charles Aulds) «Linux Apache Web Server Administration», издательство Sybex. Если же говорить о повседневных вариантах настройки Apache, в этой главе вы, вероятнее всего, найдете больше сведений о файле *httpd.conf*, чем реально используете.

Файл *httpd.conf*, поставляемый в составе Solaris, включает 160 активных строк настройки. Чтобы нам было проще в них ориентироваться, в следующих подразделах инструкции настройки разбиты на группы. Обратите внимание, что собственно файл настройки имеет другую организацию – по областям действия инструкций: глобальные инструкции среды, основные инструкции настройки сервера, инструкции настройки виртуальных узлов. (О виртуальных узлах мы поговорим позже в этой главе.) Такая организация замечательно подходит для обработки файла демоном `httpd`, но вовсе не столь удобна для тех, кто его читает. Поэтому родственные параметры здесь сгруппированы по назначению, чтобы проще было понять отдельные инструкции. Разобравшись с инструкциями по отдельности, вы сможете понять и настройку в целом.

Мы начнем с изучения инструкций файла *httpd.conf*, отвечающих за загрузку динамически загружаемых модулей. Такие модули должны быть загружены, чтобы их инструкции могли использоваться при настройке, поэтому имеет смысл обсудить сначала сами модули и только затем – функциональность, которую они предоставляют. Динамически загружаемые модули – хорошая отправная точка для изучения настройки сервера Apache.

Загрузка динамических разделяемых объектов

Чаще других в файле Solaris *httpd.conf* встречаются инструкции LoadModule и AddModule. В общей сложности они занимают более 60 из 160 строк файла *httpd.conf*. И все 60 строк выполняют настройку модулей динамических разделяемых объектов (Dynamic Shared Object, DSO), задействованных в работе сервера Apache.

Apache состоит из многочисленных программных модулей. Подобно модулям ядра, модули DSO могут встраиваться в исполняемый файл Apache, либо загружаться во время работы сервера. Ключ `-l` командной строки `httpd` позволяет получить список всех модулей, встроенных в Apache. Следующий пример взят с системы Solaris 8:

```
$ /usr/apache/bin/httpd -l
Compiled-in modules:
  http_core.c
  mod_so.c
```

В некоторых системах число модулей, встроенных в демон Apache, может быть достаточно большим. В системах Solaris и Red Hat по умолчанию таких модулей всего два:

http_core.c

Главный модуль. Он всегда связывается с ядром Apache статически и предоставляет базовую функциональность, необходимую каждому веб-серверу Apache. Данный модуль является обязательным, состав прочих может меняться.

mod_so.c

Данный модуль предоставляет поддержку времени выполнения для модулей динамических разделяемых объектов. Он требуется, если вы намереваетесь динамически подключать другие модули во время работы сервера. Если модули загружаются посредством инструкций в файле *httpd.conf*, наличие данного модуля обязательно. По этой причине он часто встраивается в ядро Apache.

Помимо двух этих встроенных модулей, в Solaris используется большое число динамически загружаемых модулей. Для загрузки объектов DSO в файле *httpd.conf* применяются инструкции LoadModule и AddModule. Прежде всего, каждый модуль должен быть указан в инструкции LoadModule. Например, следующая строка из файла Solaris *httpd.conf* определяет модуль, отслеживающий пользователей посредством cookie-квитанций:

```
LoadModule usertrack_module /usr/apache/libexec/mod_usertrack.so
```

Инструкция LoadModule содержит имя модуля и путь к файлу разделяемого объекта.

Прежде чем модуль можно будет использовать, его следует добавить в список модулей, доступных серверу Apache. Первый шаг на пути к новому списку модулей – очистка старого списка. Она выполняется посредством инструкции ClearModuleList. У инструкции ClearModuleList нет аргументов или параметров. Она фигурирует в файле *httpd.conf* за последней инструкцией LoadModule и до первой инструкции AddModule.

Инструкция AddModule добавляет имя модуля в список модулей. Список модулей должен содержать все необязательные модули, как встроенные в сервер, так и загружаемые динамически. Для нашей тестовой системы Solaris это означает, что инструкций AddModule в файле *httpd.conf* на одну больше, чем инструкций LoadModule. Дополнительная инструкция AddModule относится к модулю *mod_so.c*, который на нашей тестовой системе является единственным встроенным из необязательных модулей.[1] Однако инструкции LoadModule и AddModule по большей части встречаются парами: одна инструкция AddModule на каждую инструкцию LoadModule. Например, следующая инструкция AddModule в файле Solaris *httpd.conf* добавляет мо-

[1] Модуль *http_core.c* является интегральной частью Apache. Он не устанавливается командами LoadModule и AddModule.

дуль usertrack_module, определенный инструкцией LoadModule, которую мы ранее уже встречали, в список модулей:

```
AddModule mod_usertrack.c
```

Инструкция AddModule требует указания имени исходного файла загружаемого модуля. Обратите внимание – это имя исходного файла, компиляция которого привела к получению объектного модуля, а не имя модуля из инструкции LoadModule. Это имя, если не считать расширения, полностью совпадает с именем объектного файла. В инструкции LoadModule, которая использует расширение разделяемых объектов *.so*, именем объектного файла является *mod_usertrack.so*. Инструкция AddModule работает с расширением файлов исходного текста – *.c*, поэтому в ней фигурирует имя *mod_usertrack.c*.

В табл. 11.1 перечислены все модули, упомянутые в инструкциях AddModule файла *httpd.conf* системы Solaris 8.

Таблица 11.1. Модули DSO, загружаемые при настройке сервера в Solaris

Модуль	Назначение
mod_access	Реализует управление доступом
mod_actions	Позволяет использовать специальные обработчики для отдельных типов MIME или методов доступа
mod_alias	Позволяет ссылаться на документы и сценарии, существующие за пределами корневого каталога документа
mod_asis	Определяет типы файлов, передаваемых без заголовков
mod_auth	Реализует аутентификацию пользователей
mod_auth_anon	Реализует анонимный доступ
mod_auth_dbm	Реализует работу с файлом данных аутентификации в формате DBM
mod_autoindex	Реализует автоматическое создание указателя
mod_cern_meta	Реализует совместимость со старыми веб-серверами CERN
mod_cgi	Реализует исполнение программ CGI
mod_digest	Реализует аутентификацию по алгоритму MD5
mod_dir	Управляет форматом перечней файлов в каталогах
mod_env	Разрешает сценариям CGI и серверным инструкциям SSI (server-side includes) наследовать все переменные среды интерпретатора
mod_expires	Позволяет указывать дату для заголовка Expires:
mod_headers	Реализует настраиваемые заголовки ответов
mod_imap	Реализует обработку сенсорных карт
mod_include	Реализует обработку SSI-файлов
mod_info	Позволяет использовать информацию о сервере

Таблица 11.1 (продолжение)

Модуль	Назначение
mod_log_config	Позволяет настраивать формат записей журнала
mod_mime	Реализует поддержку файлов MIME
mod_mime_magic	Позволяет определять MIME-тип файла по его содержимому
mod_negotiation	Реализует согласование MIME-содержимого
mod_perl	Реализует поддержку языка Perl
mod_proxy	Позволяет использовать веб-кэширование
mod_rewrite	Позволяет использовать отображения URI в имена файлов
mod_setenvif	Позволяет выполнять настройку переменных среды на основе сведений, полученных от клиента
mod_so	Обеспечивает поддержку динамических разделяемых объектов (DSO) во время работы сервера
mod_speling	Автоматически исправляет мелкие ошибки орфографии
mod_status	Обеспечивает веб-доступ к отчету серверной информации
mod_unique_id	Позволяет генерировать уникальный идентификатор для каждого запроса
mod_userdir	Определяет каталог, в котором пользователи могут создавать свои веб-страницы
mod_usertrack	Позволяет отслеживать пользователей при помощи уникальных идентификаторов, известных как coookie-квитанции
mod_vhost_alias	Реализует поддержку для именованных виртуальных узлов

Добавляя модули в настройки сервера, будьте очень осторожны. Порядок следования инструкций LoadModule и AddModule в файле *httpd.conf* невероятно важен. Не изменяйте настройки, если не уверены в своих действиях. Прежде чем устанавливать новый модуль, прочтите документацию к нему, а также документацию по модулям, хранимую в каталоге *manual/mod* дистрибутива Apache. Более подробные рекомендации по установке новых модулей приводятся в книге «Linux Apache Web Server Administration», упомянутой выше.

Когда загружены модули DSO, их инструкции можно использовать в файле настройки. Продолжим изучение файла *httpd.conf* для Solaris и обратимся к некоторым из основных инструкций настройки.

Основные инструкции настройки

В этом разделе рассмотрены шесть различных инструкций в том виде, в каком они существуют в примере настройки нашей системы Solaris:

```
ServerAdmin webmaster@www.wrotethebook.com
```

```
ServerName www.wrotethebook.com
UseCanonicalName On
ServerRoot "/var/apache"
ServerType standalone
Port 80
```

О двух из них, ServerAdmin и ServerName, уже шла речь ранее. ServerAdmin позволяет указать адрес электронной почты администратора веб-сервера. В стандартной настройке Solaris используется фиктивное значение *you@your.host*. Его следует заменить полным адресом электронной почты действительного администратора, прежде чем запускать веб-сервер.

ServerName определяет имя узла, которое передается клиентам, когда они запрашивают данные с этого сервера. В стандартной настройке Solaris инструкция ServerName закомментирована, то есть клиентам передается «настоящее» имя узла. Таким образом, если первому сетевому интерфейсу присвоено имя *crab.wrotethebook.com*, это имя передается клиентам. Многие специалисты по Apache рекомендуют явным образом указывать значение ServerName в целях документирования настройки и гарантии нужного результата. Ранее мы установили ServerName в значение *www.wrotethebook.com*, поэтому, хотя веб-сервер и работает на узле *crab*, он будет известен под именем *www.wrotethebook.com* для всех веб-взаимодействий. Разумеется, имя *www.wrotethebook.com* должно быть правильно зарегистрировано в DNS. (В главе 8 можно найти определение имени *www* в качестве псевдонима узла *crab* в файле зоны *wrotethebook.com*.)

Связанной с ServerName инструкцией является UseCanonicalName, которая определяет, каким образом `httpd` создает URL, «ссылающиеся на себя» («self-referencing»). Имеются в виду такие URL, в которых в качестве имени хоста указано имя самого сервера. Например, для сервера *www.wrotethebook.com* URL, начинающийся со строки *http://www.wrotethebook.com*, будет ссылаться на себя. Имя узла в URL должно быть *каноническим*, то есть именем, которое посредством DNS может быть преобразовано в действительный IP-адрес. Когда UseCanonicalName принимает значение on (включено), как в стандартной настройке для Solaris, в URL, ссылающихся на себя, сервер задается значением ServerName. В большинстве случаев следует оставить инструкцию включенной. Если отключить UseCanonicalName, используется значение из запроса клиента.

Параметр ServerRoot указывает каталог, в котором хранятся важные файлы `httpd`, включая журналы ошибок, журналы обращений, а также три файла настройки: *httpd.conf*, *srm.conf* и *access.conf*. В Solaris ServerRoot указывает на каталог */var/apache*. Это неожиданно, потому что файлы настройки для демона `httpd` в Solaris на самом деле расположены в каталоге */etc/apache*. Очевидно, действует еще какой-то механизм.

В Solaris используется ключ `-f` командной строки `httpd` – для переопределения месторасположения файла *httpd.conf* во время работы сервера. `httpd` запускается во время загрузки системы из сценария */etc/init.d/apache*. Этот сценарий определяет переменную `CONF_FILE`, в которой хранится значение

/etc/apache/httpd.conf. Переменная используется в команде `httpd`, выполняющей запуск веб-сервера, и именно эта переменная определяет расположение файла настройки в системе Solaris.

Параметр ServerType определяет способ запуска сервера. Если сервер запускается из загрузочного сценария, параметру присваивают значение `standalone`. Если сервер запускается по необходимости – посредством `inetd`, параметру присваивают значение `inetd`. В стандартной настройке для Solaris ServerType имеет значение `standalone`, которое является предпочтительным, поскольку веб-сервер обычно пользуется высоким спросом, и лучше всего запускать его при загрузке системы. Разумеется, вполне возможна ситуация, когда пользователь создает небольшой, мало востребованный сайт на своей рабочей станции, и в таком варианте может быть более предпочтительным запуск сервера посредством `inetd`. Однако для веб-сервера вашей сети следует использовать значение `standalone`.

Инструкция Port определяет номер порта TCP, используемого сервером. Стандартный порт имеет номер 80. Иногда частные веб-серверы работают через порты с другими номерами. К примеру, сервер *AnswerBook2* в Solaris работает через порт 8888. Прочие популярные номера портов для веб-сайтов специального назначения – 8080 и 8000. Если используется нестандартный номер порта, обязательно уведомите пользователей. К примеру, URL веб-сайта, доступного через порт TCP 8080, выглядит следующим образом: *http://jerboas.wrotethebook.com:8080*.

Когда ServerType принимает значение `inetd`, желательно присвоить параметру Port значение, отличное от стандартного (80). Причиной тому – «привилегированность» портов ниже 1024. Если используется порт 80, демон `httpd` должен выполняться из `inetd` в качестве процесса администратора (пользователя *root*). Это создает потенциальную уязвимость, поскольку злоумышленник может найти способ через веб-сайт получить root-доступ к системе. Использование порта с номером 80 вполне допустимо при значении ServerType `standalone`, поскольку начальный процесс `httpd` не обслуживает клиентов напрямую. Вместо этого он порождает набор других демонов HTTP, называемый *пулом*, и эти демоны предоставляют доступ клиентам. Демоны пула не работают с полномочиями администратора.

Управление пулом

Изначально веб-сервер был спроектирован таким образом, что создавал отдельный процесс для обработки каждого запроса. Высокая нагрузка на веб-сервер при этом неизбежно влекла высокую загрузку процессора системы и значительно снижала скорость ее реакции. Процессы `httpd` были способны свести к минимуму жизнедеятельность всей системы.

В Apache применяется иной подход. Пул процессов запускается при загрузке (приведенная ранее команда `ps` отражает набор процессов `httpd`, работающих в системе Solaris), и рабочая нагрузка распределяется между процессами пула. Если все долгоживущие процессы `httpd` заняты обработкой, сервер

порождает дополнительные процессы, принимающие на себя часть нагрузки. За управление пулом порожденных процессов в Apache отвечают пять инструкций настройки.

MinSpareServers

Определяет число постоянных свободных процессов сервера. В настройках Solaris принято значение 5, то есть значение по умолчанию в дистрибутиве Apache. Если число свободных процессов опускается ниже пяти, создается дополнительный процесс, и число таких процессов восстанавливается. Пять процессов – разумное значение для среднего сервера; оно позволяет обрабатывать серии из пяти запросов подряд, не заставляя клиента ожидать, пока запустится порожденный процесс. Для сервера с небольшой загрузкой можно использовать меньшее число, а серверу с высокой загрузкой может быть полезно более высокое значение. Однако нет смысла держать большое число свободных процессов, если они никогда не будут востребованы.

MaxSpareServers

Определяет максимально допустимое число свободных процессов сервера. Данная инструкция предотвращает бездействие слишком большого числа процессов. Если число свободных процессов превышает значение MaxSpareServers, лишние свободные процессы принудительно завершаются. В настройке для Solaris MaxSpareServers получает значение 10, то есть значение по умолчанию в дистрибутиве Apache. Значение MaxSpareServers рекомендуется выбирать таким образом, чтобы оно превышало значение MinSpareServers примерно в два раза.

StartServers

Определяет число демонов `httpd`, запускаемых при загрузке. В настройке для Solaris StartServers получает значение 5. Действие этой инструкции напрямую отражается на выводе команды `ps`, приводившемся ранее в этой главе и содержавшем сведения о шести процессах демона `httpd`. Один из этих процессов – родительский, он управляет пулом; пять других процессов – порожденные, они и занимаются обработкой запросов данных, поступающих от клиентов.

MaxClients

Определяет максимальное число обращений от клиентов, обслуживаемых единовременно. После превышения значения MaxClients поступающие запросы на HTTP-соединения не обслуживаются. В Solaris данный параметр принимает значение 150, которое наиболее широко распространено. MaxClients не позволяет серверу использовать все ресурсы системы при получении чрезмерно большого числа запросов. Значение MaxClients можно увеличивать лишь на сверхпроизводительной системе с быстрыми дисковыми накопителями и большим объемом памяти. Как правило, справляться с ростом числа клиентов лучше путем увеличения числа серверов. Верхний порог для MaxClients определяется значением HARD_SER-

VER_LIMIT, которое указывается при компиляции Apache. По умолчанию HARD_SERVER_LIMIT равно 256.

MaxRequestsPerChild

Определяет, сколько запросов может обработать порожденный процесс, прежде чем будет завершен. В Solaris MaxRequestsPerChild принимает значение 0, «неограниченное», то есть порожденный процесс может обрабатывать поступающие от клиентов запросы, пока работает система. В данной инструкции всегда следует указывать значение 0, за исключением случаев, когда достоверно известно, что библиотека, использовавшаяся для компиляции Apache, подвержена утечкам памяти.

Инструкции User и Group определяют идентификатор пользователя (UID) и группы (GID), с полномочиями которых выполняется пул процессов `httpd`. Когда `httpd` запускается при загрузке системы, то, обладая полномочиями суперпользователя, устанавливает связь с портом 80 и запускает группу порожденных процессов, которые занимаются обслуживанием запросов. Именно этим порожденным процессам присваиваются идентификаторы UID и GID, определенные в файле *httpd.conf*. UID и GID следует выбирать таким образом, чтобы веб-сервер получил минимальные права доступа к системе. В операционной системе Solaris устанавливается уровень доступа пользователя *nobody* и группы *nobody*. Это ясно видно из вывода последней команды `ps`. Один процесс `httpd` принадлежит пользователю *root*, а еще пять процессов `httpd` – пользователю *nobody*. Альтернативой использованию пользователя/группы *nobody* является создание идентификатора пользователя и группы специально для `httpd`. Выбрав этот путь, будьте особенно внимательны при создании прав доступа к файлам для учетной записи демона. Преимущество создания специального пользователя и группы для `httpd` в том, что можно использовать групповые права для повышения защищенности, а также снимается зависимость от общесистемных полномочий *nobody*.

Хранение данных

Инструкция DocumentRoot определяет каталог, в котором хранятся документы веб-сервера. Из соображений безопасности файлы настройки хранятся в другом каталоге. Как мы уже видели, инструкция DocumentRoot в Solaris представлена следующим образом:

```
DocumentRoot "/var/apache/htdocs"
```

Чтобы применить инструкции к определенным каталогам, необходимо создать для этих инструкций *контейнер* (*container*). Следующие три инструкции *httpd.conf* применяются для создания контейнеров:

`<Directory` *pathname*`>`

Инструкция Directory создает контейнер для инструкций, относящихся к каталогу *pathname*. Любая инструкция настройки, следующая за инструкцией Directory и предшествующая следующему оператору `</Directory>`, применяется только к указанному каталогу.

<Location *document*>

Инструкция Location создает контейнер для инструкций, относящихся к конкретному документу (*document*). Любая инструкция настройки, следующая за инструкцией Location и предшествующая следующему оператору </Location>, применяется только к указанному документу.

<Files *filename*>

Инструкция Files создает контейнер для инструкций, относящихся к конкретному файлу (*filename*). Любая инструкция настройки, следующая за инструкцией Files и предшествующая следующему оператору </Files>, применяется только к указанному файлу. Имя файла (*filename*) может указывать одновременно несколько файлов, если содержит один из специальных символов Unix, * или ?. Кроме того, если за инструкцией Files следует необязательный символ ~ (тильда), имя файла (*filename*) интерпретируется как регулярное выражение.

С каталогами и файлами легко разобраться, поскольку это знакомые каждому администратору компоненты файловой системы Unix. Понятие документа характерно уже не для файловой системы, но для веб-сервера. Страница с информацией, отображаемая в ответ на запрос к веб-серверу, является документом: она может состоять из многих файлов, хранящихся в различных каталогах. Контейнер Location позволяет обращаться со сложными документами как с единичными сущностями. Ниже в тексте главы мы рассмотрим примеры использования контейнеров Location и Files. А пока обратимся к контейнерам Directory.

В настройках Solaris определены контейнеры Directory для корневого каталога сервера и каталога DocumentRoot:

```
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
<Directory "/var/apache/htdocs">
    Options Indexes FollowSymLinks
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

Каждый контейнер Directory начинается с инструкции Directory и заканчивается тегом </Directory>. Каждый из контейнеров содержит операторы настройки, которые относятся только к одному из каталогов. Назначение инструкций в данных контейнерах описано далее в разделе «Безопасность веб-сервера». На данном этапе достаточно понять, что контейнеры в файле *httpd.conf* применяются с целью ограничения области действия различных инструкций настройки.

Инструкции Alias и ScriptAlias выполняют отображение URL-пути в имя каталога на сервере. Так, файл настройки в Solaris содержит следующие три инструкции:

```
Alias /icons/ "/var/apache/icons/"
Alias /manuals/ "/usr/apache/htdocs/manual/"
ScriptAlias /cgi-bin/ "/var/apache/cgi-bin/"
```

Первая строка отображает URL-путь */icons/* в каталог */var/apache/icons/*. Таким образом, запрос *www.wrotethebook.com/icons/* является запросом *www.wrotethebook.com/var/apache/icons/*. Вторая инструкция отображает URL-путь */manuals/* в *www.wrotethebook.com/usr/apache/htdocs/manual/*.

Инструкций Alias, устанавливающих подобные отображения, может быть сколько угодно, но инструкция ScriptAlias может быть только одна. ScriptAlias действует точно так же, как инструкция Alias, с той разницей, что конечный каталог содержит исполняемые CGI-программы. Следовательно, `httpd` разрешает исполнение файлов из данного каталога. Инструкция ScriptAlias особенно важна потому, что позволяет хранить исполняемые веб-сценарии в каталоге вне иерархии DocumentRoot. Сценарии CGI – без преувеличений наиболее уязвимое место вашего сервера; хранение этих сценариев в отдельном каталоге позволяет более жестко разграничивать доступ к ним.

Настройки для Solaris содержат контейнеры каталогов */var/apache/icons* и */var/apache/cgi-bin*, но не каталога */usr/apache/htdocs/manual*. Упоминание каталога в *httpd.conf* вовсе не требует создания контейнера Directory для этого каталога. Контейнеры */var/apache/icons* и */var/apache/cgi-bin* выглядят следующим образом:

```
<Directory "/var/apache/icons">
    Options Indexes MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
<Directory "/var/apache/cgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>
```

Контейнеры содержат операторы AllowOverride, Options, Order и Allow, которые относятся к безопасности. Большинство инструкций в контейнерах файлов, документов и каталогов явно или неявно связаны с вопросами безопасности. Все инструкции из приведенных выше контейнеров рассмотрены далее в этой главе в разделе «Безопасность веб-сервера».

Инструкция UserDir позволяет пользователям создавать домашние страницы и указывает на каталог, в котором они хранятся. UserDir, как правило, указывает на каталог *public_html*, что и можно наблюдать в настройках So-

laris. Такие настройки по умолчанию позволяют пользователю создать подкаталог *public_html* в своем домашнем каталоге и поместить в него домашнюю веб-страницу. Например, если поступает запрос по адресу *www.wrotethebook.com/~sara*, происходит преобразование в адрес *www.wrotethebook.com/export/home/sara/public_html*. Кроме того, инструкция UserDir позволяет указать полное имя – скажем, */export/home/userpages*. Администратор системы создает этот каталог и разрешает пользователям сохранять домашние веб-страницы в подкаталогах этого каталога. В последнем случае запрос адреса *www.wrotethebook.com/~sara* будет преобразован в запрос адреса *www.wrotethebook.com/export/home/userpages/sara*. Такой подход имеет свои преимущества и недостатки: облегчает наблюдение за содержимым страниц пользователей, однако требует создания и защиты отдельного дерева веб-каталогов, в то время как веб-каталог в домашнем каталоге пользователя автоматически наследует существующие права доступа.

Инструкции PidFile и ScoreBoardFile указывают пути к файлам, определяющим состояния процессов. Параметр PidFile обозначает файл, в котором `httpd` хранит свой идентификатор процесса, параметр ScoreBoardFile – файл, в который `httpd` записывает сведения о состоянии процесса.

Параметр DirectoryIndex определяет имя файла, возвращаемого сервером, если запрос клиента не содержит имени файла. В нашей системе Solaris данный параметр определен следующим образом:

```
DirectoryIndex index.html
```

Исходя из значений параметров DocumentRoot и DirectoryIndex сервер, получив запрос документа *http://www.wrotethebook.com*, вернет клиенту файл */var/apache/htdocs/index.html*. Получив запрос документа *http://www.wrotethebook.com/books/*, сервер вернет клиенту файл */var/apache/htdocs/books/index.html*. Значение DocumentRoot предваряет все имена файлов, а DirectoryIndex добавляется к любому запросу, который не заканчивается именем файла.

Ранее в этой главе мы видели из вывода `ls` для каталога */var/apache/htdocs*, что в каталоге существует файл *index.html*. Что же произойдет в отсутствие такого файла? Что Apache передаст клиенту? Если файл *index.html* не существует, `httpd` посылает клиенту список файлов каталога – если это допускается настройками. Передача содержимого каталога разрешена, если инструкция Options в контейнере Directory данного каталога содержит ключевое слово `Indexes`. (Более подробно рассмотрим Options позже.) Если разрешено получение клиентом индекса каталога, форматированием списка содержимого можно управлять с помощью ряда дополнительных инструкций.

Создание красивого индекса

Ключевое слово `FancyIndexing` в строке инструкции IndexOptions включает создание «красивого индекса» каталога для случаев, когда сервер Apache вынужден отправить индекс в ответ на запрос клиента. В случае «красивого»

оформления индекса `httpd` создает список файлов каталога с использованием графики, ссылок и прочих дополнительных элементов. В настройках для Solaris создание красивых индексов включено в инструкции IndexOptions, а файл настройки содержит около 20 вспомогательных строк, выполняющих настройку собственно вида индекса. Следующие инструкции используются в Solaris для определения состава графических и прочих элементов красивого индекса каталога:

IndexIgnore

Указывает файлы, которые не следует включать в индекс. Могут указываться полные имена файлов, неполные имена, расширения, а кроме того, допустимо использование стандартных специальных символов.

HeaderName

Указывает имя файла, содержащего «шапку» индекса каталога.

ReadmeName

Указывает имя файла, содержащего сведения, отображаемые в конце страницы индекса каталога.

AddIconByEncoding

Указывает пиктограмму для обозначения файлов, представленных определенным типом MIME-кодировки.

AddIconByType

Указывает пиктограмму для обозначения файлов, представленных определенным файловым MIME-типом.

AddIcon

Указывает пиктограмму для обозначения файлов, представленных определенным расширением.

DefaultIcon

Указывает пиктограмму для обозначения файлов, которым не были назначены пиктограммы.

Определение файловых типов

Файловые типы MIME и расширения файлов играют важную роль в процессе определения способа обработки файла сервером. Сегмент параметров MIME – важная часть файла *httpd.conf*, созданного для Solaris. Используются следующие инструкции:

DefaultType

Определяет MIME-тип, используемый в случаях, когда сервер не может определить тип файла. В настройках Solaris данный параметр имеет значение text/plain. Таким образом, если файл не имеет расширения, сервер предполагает, что это обычный текстовый файл.

AddEncoding

Связывает кодировку MIME с расширением файла. Настройки Solaris содержат две инструкции AddEncoding:

```
AddEncoding x-compress Z
AddEncoding x-gzip gz tgz
```

Первая инструкция связывает расширение `Z` с типом MIME-кодировки `x-compress`. Вторая строка связывает расширения `gz` и `tgz` с типом MIME-кодировки `x-gzip`.

AddLanguage

Связывает MIME-тип языка с расширением файла. Настройки Solaris содержат такие отображения для шести языков, например `.en` для английского и `.fr` для французского.

LanguagePriority

Устанавливает приоритет кодировки языка при подготовке многоязычных представлений (*multiviews*), а также язык, используемый по умолчанию. В настройках Solaris порядок старшинства следующий: английский (`en`), французский (`fr`), немецкий (`de`). Это означает, что при использовании многоязычных видов будут подготовлены варианты страниц на английском, французском и немецком. Если предпочтительный язык не указан, клиенту передается вариант на английском языке.

AddType

Связывает файловый тип MIME с расширением файла. В файле *httpd.conf* Solaris присутствует лишь одна инструкция AddType – она связывает тип MIME application/x-tar с расширением `.tgz`. Таких инструкций в файле настройки может быть несколько.

Другая инструкция, часто используемая для обработки файлов на основе их расширений, – инструкция AddHandler. AddHandler связывает файловый обработчик с определенным расширением. Файловый обработчик – это программа, которая «знает», как обрабатывать файлы конкретного типа. Например, обработчик `cgi-script` способен выполнять файлы CGI. В файле *httpd.conf* Solaris отсутствуют какие-либо дополнительные обработчики, поэтому все инструкции AddHandler закомментированы.

Инструкции настройки производительности

Инструкция KeepAlive позволяет использовать постоянные соединения. В противном случае клиенту приходится создавать новое соединение с сервером для каждой ссылки, по которой переходит пользователь. Поскольку HTTP работает поверх TCP, каждое соединение требует дополнительного времени на создание, что замедляет получение каждого файла. Постоянные соединения позволяют серверу некоторое время ожидать получения от клиента дополнительных запросов, прежде чем закрыть соединение. Клиенту, таким образом, не требуются новые соединения для обращения к другим документам. Инструкция KeepAliveTimeout определяет интервал ожидания

(в секундах) новых запросов от клиента для уже открытого соединения. В файле *httpd.conf* Solaris режим KeepAlive включен, а KeepAliveTimeout присвоено значение, равное 15 секундам.

MaxKeepAliveRequests определяет максимально допустимое число запросов, полученных через постоянное соединение. В Solaris этот параметр имеет значение 100, то есть значение по умолчанию для Apache. Присвоение MaxKeepAliveRequests значения 0 разрешает обрабатывать неограниченное число запросов. Значение 100 – разумное для данного параметра: очень немногие пользователи запрашивают 100 документов, а значит, такая настройка постоянных соединений охватывает все нормальные ситуации. Если же клиент создал более 100 запросов, не исключено, что причиной тому стали проблемы в системе клиента, и в таком случае создание нового соединения, скорее всего, будет удачной идеей.

Инструкция Timeout определяет интервал ожидания сервером завершения передачи (в секундах). Значение должно быть достаточно большим, чтобы клиенты успевали получать самые большие файлы сайта через низкопроизводительное модемное соединение. Однако слишком большое значение приведет к тому, что сервер будет поддерживать соединения клиентов, которые уже отключились от сети. В файле *httpd.conf* Solaris Timeout имеет весьма распространенное значение – 5 минут (300 секунд).

BrowserMatch – параметр настройки иного рода: он снижает производительность в пользу совместимости. Настройки Solaris содержат следующий набор инструкций BrowserMatch:

```
BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0
BrowserMatch "RealPlayer 4\.0" force-response-1.0
BrowserMatch "Java/1\.0" force-response-1.0
BrowserMatch "JDK/1\.0" force-response-1.0
```

Операторы BrowserMatch используются для представления информации способами, понятными различным веб-броузерам. Например, определенный броузер умеет работать только с протоколом HTTP 1.0, но не HTTP 1.1. В таком случае значение `downgrade-1.0` в строке инструкции BrowserMatch позволяет гарантировать, что сервер, работая с этим броузером, использует только HTTP 1.0.

В приведенных строках из файла *httpd.conf* системы Solaris постоянные соединения отключены для двух из броузеров. Одному из броузеров предлагается для использования только протокол HTTP 1.0, а для четырех из них принудительно передаются ответы в формате, совместимом с HTTP 1.0.

Не играйте с инструкциями BrowserMatch. Приведенные настройки являются стандартными для дистрибутива Apache и позволяют справляться с ограничениями различных броузеров. Это параметры настройки, но используются они преимущественно разработчиками Apache для адаптации под ограничения устаревших броузеров.

Инструкция HostnameLookups определяет, должен ли демон `httpd` регистрировать в журнале обращений имена узлов наряду с адресами IP. Преимущество регистрации имен узлов в том, что журнал получается более удобным для чтения. Недостаток – дополнительная нагрузка на `httpd`, связанная с поиском имен в DNS. Значение `off`, стандартное для настройки Solaris, повышает производительность сервера. Инструкция HostnameLookups влияет на состав информации, заносимой в журнал, однако оказывает существенное влияние на производительность системы – почему мы и рассматриваем ее в разделе, посвященном параметрам настройки производительности, а не параметрам настройки журнала.

Инструкции настройки журнала

Файлы журналов представляют огромное количество информации о веб-сервере. Следующий набор строк из стандартного файла *httpd.conf* системы Solaris 8 определяет конфигурацию журнала:

```
ErrorLog /var/apache/logs/error_log
LogLevel warn
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
CustomLog /var/apache/logs/access_log common
```

ErrorLog определяет полное имя файла журнала ошибок. Обращайтесь к журналу ошибок в целях поиска и исправления сбоев. Этот журнал следует просматривать по меньшей мере раз в сутки на предмет обнаружения проблем. Работая в системе, вы можете непосредственно наблюдать за попадающей в файл журнала информацией посредством команды `tail` с ключом `-f`:

```
$ tail -l 1 -f /var/log/httpd/apache/error_log
```

Команда `tail` выводит набор строк из конца файла; в данном случае – из файла */var/log/httpd/apache/error_log*. Ключ `-l` определяет количество выводимых строк. В данном примере `-l 1` предписывает `tail` отобразить (одну) последнюю строку файла. Ключ `-f` предписывает процессу `tail` продолжать работу, что позволяет наблюдать записи по мере их добавления в файл. Таким образом, у нас есть возможность наблюдать за файлом в реальном времени.

Инструкция LogLevel определяет тип событий, регистрируемых в журнале ошибок. В настройках Solaris LogLevel имеет значение `warn`, то есть в журнал попадают предупреждения и все сообщения о более серьезных ошибках. Это безопасная установка для файла журнала, поскольку позволяет зафиксировать широкий диапазон ошибок работы сервера. LogLevel имеет восемь возможных значений: `debug`, `info`, `notice`, `warn`, `error`, `crit`, `alert` и `emerg`. Уровни являются накопительными. К примеру, `warn` регистрирует предупреждения, ошибки, критически важные сообщения, сигналы тревоги и аварийные сообщения; `debug` регистрирует все сообщения, что приводит к очень быстрому росту объема файла журнала; `emerg` сокращает объем файла, но уведомляет

администратора только о катастрофах. `warn` – неплохой компромисс для уровня диагностики.

Помимо важной информации о возникающих ошибках, журналы содержат сведения о том, кто пользуется сервером, насколько активно и насколько качественно сервер обслуживает клиентов. Веб-серверы используются для распространения информации; если информацией никто не пользуется или она никому не нужна, об этом стоит знать. Инструкции LogFormat и CustomLog настраивают не журнал ошибок, но *способ* регистрации активности сервера.

Определение формата файла журнала

Инструкции LogFormat определяют формат записей файла журнала. Инструкция LogFormat содержит два элемента: формат записи файла и метку, используемую внутри файла *httpd.conf* для указания на такую запись. Формат записи заключен в двойные кавычки и непосредственно следует за ключевым словом LogFormat. Формат состоит из постоянных подстрок (литералов) и переменных.

Чтобы понять, как используются переменные, обратимся к примеру инструкции LogFormat. Базовый файл журнала Apache подчиняется формату CLF (Common Log Format). CLF – это стандарт, используемый всеми разработчиками веб-серверов. Таким образом, журналы, созданные серверами Apache, могут обрабатываться любыми инструментами анализа журналов, понимающими стандарт CLF. Формат стандартной записи CLF четко определен второй инструкцией LogFormat из файла *httpd.conf* Solaris:

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
```

Данная инструкция LogFormat указывает только те сведения, которые требуются для записи журнала CLF. С этой целью используется набор из семи переменных LogFormat:

`%h`

Регистрирует IP-адрес клиента. При включенном параметре HostnameLookups регистрируется абсолютное имя узла клиента. Для нашей системы Solaris это будет IP-адрес клиента, поскольку HostnameLookups заблокированы с целью повышения производительности сервера.

`%l`

Регистрирует имя, под которым зарегистрировался клиент, если таковое доступно. Имя извлекается по протоколу `identd`; однако большинство клиентов не используют `identd` и не предоставляют сведения такого рода. Как следствие, данное поле обычно содержит дефис, обозначающий отсутствие значения. Точно так же, если сервер не способен определить значение для поля, журнал содержит дефис в этом поле.

`%u`

Регистрирует имя пользователя, использованное для доступа к веб-странице, защищенной паролем. Имя должно совпадать с именем, определенным в файле сервера AuthUser или базе данных AuthDBMUser. (Auth-

User и AuthDBMUser рассмотрены в разделе «Безопасность веб-сервера» далее в этой главе.) Парольная защита используется не так уж часто, и данное поле в большинстве записей журнала содержит дефис.

`%t`

Регистрирует дату и время создания записи журнала.

`%r`

Регистрирует первую строку запроса клиента. Часто она содержит URL-идентификатор запрошенного документа. Символы `\"` в инструкции LogFormat показывают, что вывод должен содержать символ двойной кавычки. В результате в файле журнала запрос клиента будет заключен в двойные кавычки.

`%>s`

Регистрирует состояние последнего запроса. Состояние – это код ответа сервера клиенту, состоящий из трех цифр.

`%b`

Фиксирует число байтов в документе, который был отправлен клиенту.

Записи журнала Apache не ограничены форматом CLF. Инструкция LogFormat позволяет указать, какую информацию следует фиксировать в журнале. Диапазон таких сведений достаточно широк.

Файл *httpd.conf* Solaris содержит три дополнительных инструкции LogFormat, демонстрирующих варианты форматов журнальных записей. Вот эти инструкции:

```
LogFormat "%{User-agent}i" agent
LogFormat "%{Referer}i -> %U" referer
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
```

Все эти инструкции регистрируют содержимое HTTP-заголовков. Так, первая инструкция регистрирует значение, полученное от клиента в заголовке `User-agent`. User-agent – это программа пользователя, генерирующая запрос на документ; как правило, речь идет о названии броузера. Строка формата, позволяющая зафиксировать в журнале содержимое такого заголовка:

```
%{User-agent}i
```

Данный формат работает для всех заголовков: достаточно заменить `User-agent` другим названием (заголовка). Буква `i` указывает, что речь идет о полученном заголовке (*i*nput header); отправленные заголовки (*o*utput headers) фиксируются по букве `o`. Сервер Apache способен регистрировать содержимое любых заголовков, полученных или отправленных.

Вторая инструкция LogFormat фиксирует содержимое заголовка `Referer`, полученного от клиента (`%{Referer}i`), символы дефиса и знака «больше» (`->`), а также URL-адрес, на который поступил запрос (`%U`). `Referer` – это имя внешнего сайта, который перенаправил клиента на ваш веб-сайт; `%U` – документ, к которому этот сайт направил клиента.

Последняя инструкция LogFormat начинается с переменных CLF (`%h %l %u %t \"%r\" %>s %b \"`) и дополнительно содержит значения из заголовков `Referer` и `User-agent`. Данному формату присвоена метка `combined`, поскольку он сочетает CLF с прочими сведениями; два предшествующих формата разумно отмечены как `agent` и `referer`. Однако ни один из этих форматов на деле не используется в настройках Solaris. Наличия инструкции LogFormat недостаточно для создания файла журнала: следует добавить соответствующую инструкцию CustomLog, которая свяжет формат с файлом. Как это сделать, будет рассказано ниже.

В инструкции LogFormat спецификация формата записи журнала заключается в двойные кавычки. Метка, следующая за спецификацией, не является частью формата. В инструкции LogFormat, определяющей формат CLF, метка `common` – это произвольная строка, связывающая данную инструкцию LogFormat с определенной инструкцией CustomLog. В файле *httpd.conf* Solaris данная конкретная инструкция LogFormat связывается с файлом */var/apache/logs/access_log* при помощи следующей строки:

```
CustomLog /var/apache/logs/access_log common
```

Две инструкции связываются благодаря общей метке – `common`. В результате записи CLF, определенные данной конкретной инструкцией LogFormat, записываются в файл, определенный в указанной инструкции CustomLog.

В настройках Solaris прочие инструкции CustomLog (создающие файлы журналов для записей `agent`, `referer` и `combined`) закомментированы:

```
#CustomLog /var/apache/logs/referer_log referer
#CustomLog /var/apache/logs/agent_log agent
#CustomLog /var/apache/logs/access_log combined
```

Файл *referer_log* хранит URL-адреса страниц, с которых произошел переход на ваш веб-сервер. Это позволяет определить, какие сайты указывают на ваши веб-страницы. Записи в *referer_ log* имеют следующий формат:

```
LogFormat "%{Referer}i -> %U" referer
```

Чтобы создать журнал, раскомментируйте следующую строку:

```
CustomLog /var/apache/logs/referer_log referer
```

Файл *agent_log* фиксирует, какие броузеры используются для доступа к вашему сайту, и формат его записей определяется следующим оператором LogFormat:

```
LogFormat "%{User-agent}i" agent
```

Чтобы создать журнал, раскомментируйте следующую строку:

```
CustomLog /var/apache/logs/agent_log agent
```

Наконец, формат расширенного CLF-журнала определен строкой:

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
```

Чтобы создать комбинированный журнал, раскомментируйте следующую строку:

```
CustomLog /var/apache/logs/access_log combined
```

и закомментируйте эту:

```
#CustomLog /var/apache/logs/access_log common
```

В результате этих изменений для создания файла журнала */var/apache/logs/access_log* будет использоваться формат `combined`. Именно для этого файла журнала по умолчанию используется формат `common`. Чтобы избежать дублирования записей, выключите регистрацию `common`, если включили регистрацию `combined`. По существу, эти изменения переключают формат файла *access_log* с `common` на `combined`.

Оператор LogFormat и соответствующий ему оператор CustomLog заканчиваются одной и той же меткой. Метка – это произвольное имя, связывающее формат и файл журнала.

Регистрация по условию

Apache также поддерживает занесение сведений в журнал по условию, позволяя указать поля, значения которых фиксируются, только когда сервер возвращает клиенту определенный код состояния. Коды состояний перечислены в табл. 11.2.

Таблица 11.2. Коды состояний сервера Apache

Код состояния	Значение
200: OK	Корректный запрос
302: Found	Документ не найден
304: Not Modified	Указанный документ не был изменен
400: Bad Request	Некорректный запрос
401: Unauthorized	Клиенту или пользователю отказано в доступе
403: Forbidden	Недостаточно полномочий для доступа
404: Not Found	Указанный документ не существует
500 Server Error	Неустановленная ошибка в работе сервера
503: Out of Resources (Service Unavailable)	Недостаточно серверных ресурсов для выполнения запроса
501: Not Implemented	Отсутствует функциональность, необходимая для обработки запроса
502: Bad Gateway	Клиент указал некорректный шлюз

Чтобы поставить поле в зависимость от условия, добавьте один или несколько кодов состояний в спецификацию поля в инструкции LogFormat. Если ис-

пользуется список, разделяйте коды состояний запятыми. Предположим, необходимо записывать название броузера только в том случае, если броузер запрашивает службу, не реализуемую сервером. Используйте код состояния Not Implemented (501) в сочетании с заголовком `User-agent` следующим образом:

```
%501{User-agent}i
```

Если такое значение присутствует в операторе LogFormat, название броузера записывается в журнал только для кода состояния 501.

Восклицательный знак перед кодом (или кодами) состояния предписывает регистрировать поле только в случае, когда действительный код состояния не равен одному из перечисленных значений. К примеру, чтобы фиксировать адрес сайта, с которого пользователь пришел на вашу веб-страницу, только в случаях, когда код состояния сигнализирует об ошибке, добавьте следующий фрагмент в спецификацию LogFormat:

```
%!200,302,304{Referer}i
```

Данная конкретная условная запись журнала весьма полезна, поскольку позволяет определить, что на внешней странице существует некорректная ссылка на ваш веб-сайт.

Используйте эти возможности совместно с форматом `common` для создания более содержательных записей журнала. Вот так мы изменили формат Solaris `combined`, включив в него регистрацию по условию:

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%!200,302,304{Referer}i\" \"%{User-Agent}i\"
" combined
```

Данный формат предписывает фиксировать все данные CLF, так что полученные записи журнала поддаются анализу при помощи стандартных инструментов. При этом дополнительно записи содержат имя броузера и, в случаях обращения по устаревшим ссылкам, позволяют определить, какой из внешних сайтов распространяет неверную информацию.

Файл настройки сервера Apache в Solaris содержит более 160 активных строк, но при этом есть ряд интересных возможностей Apache, которые в настройках Solaris не используются. Прежде чем мы перейдем к важным повседневным вопросам обеспечения безопасности сервера и наблюдения за работой сервера, проведем краткий обзор трех возможностей, не включенных в стандартную настройку Solaris: прокси-серверы и кэширование, настройки многосетевых серверов, виртуальные узлы.

Прокси-серверы и кэширование

Серверы, действующие в качестве промежуточных звеньев между клиентами и веб-серверами, называются *прокси-серверами* (*proxy servers*, посредники). При использовании брандмауэра прямой веб-доступ часто блокируется. В этом случае пользователи подключаются к прокси-серверу по локальной сети, тогда как прокси-сервер обладает полномочиями для подключения к

удаленному веб-серверу. Прокси-серверы могут хранить копии веб-страниц с удаленных веб-серверов в целях повышения производительности, сокращения объема трафика в территориальной сети и уменьшения нагрузки на популярные веб-сайты. Вот параметры, управляющие подсистемой кэширования:

CacheNegotiatedDocs

Разрешает прокси-серверам кэшировать веб-страницы с вашего сервера. По умолчанию Apache сообщает прокси-серверам, что страницы с вашего веб-сервера кэшировать нежелательно. Данная инструкция используется без аргументов.

ProxyRequests

Данный параметр позволяет превратить сервер в прокси-сервер. По умолчанию имеет значение `off` (выключено).

ProxyVia

Разрешает или запрещает использование заголовков `Via:`, которые позволяют определять действительный источник кэшированных страниц.

CacheRoot

Указывает путь к каталогу, в котором хранятся кэшированные веб-страницы, если данный сервер является прокси-сервером. Чтобы избежать возможности записи в этот каталог пользователем *nobody*, создайте специальный идентификатор пользователя, если `httpd` используется в качестве прокси-сервера.

CacheSize

Устанавливает максимальный размер кэша. По умолчанию – 5 Кбайт.

CacheGcInterval

Устанавливает интервал для усечения кэша (в часах). По умолчанию – 4. Если значения по умолчанию не менялись, сервер будет сокращать объем хранимой информации до 5 Кбайт каждые 4 часа.

CacheMaxExpire

Устанавливает максимальное время хранения документа без запроса обновления с удаленного сервера (в часах). По умолчанию документы хранятся 24 часа. Это означает, что кэшированный документ может иметь давность до суток.

CacheLastModifiedFactor

Устанавливает длительность кэширования документа на основе даты последнего изменения. Коэффициент по умолчанию – 0,1. Так, если документ был изменен 10 часов назад, он хранится в кэше только один час, прежде чем копия обновляется. Предполагается, что если документ изменяется часто, последнее изменение было внесено не так давно; поэтому документы, изменяемые часто, кэшируются на непродолжительные периоды времени. В любом случае ни один документ не хранится дольше, чем указано значением CacheMaxExpire.

CacheDefaultExpire

Устанавливает значение по умолчанию устаревания данных кэша для протоколов, не позволяющих получать подобное значение. По умолчанию 1 час.

NoCache

Позволяет указать список серверов, страницы которых не должны кэшироваться. Если известно, что информация на сервере постоянно обновляется, нет смысла кэшировать страницы этого сервера, поскольку в кэше будут постоянно храниться устаревшие данные. Указание имени такого сервера в командной строке NoCache предписывает передавать все запросы напрямую серверу и не кэшировать ответы сервера.

Все эти инструкции закомментированы в настройках Solaris. По умолчанию сервер Solaris Apache не настроен на работу в качестве прокси-сервера. Если необходимо создать прокси-сервер, обратитесь к книге, посвященной настройке Apache, например «Linux Apache Web Server Administration».

Параметры настройки сервера с несколькими IP-адресами

Веб-серверы с несколькими адресами IP известны в качестве *многосетевых* (*multihomed*). Такой сервер должен знать, на какой адрес поступают запросы, подлежащие обработке. Для решения этой задачи Apache предоставляет следующие параметры настройки:

BindAddress

Указывает адрес, используемый в работе сервера. По умолчанию имеет значение *, то есть сервер должен отвечать на запросы к веб-службе, поступившие на любой из его IP-адресов. Если в строке BindAddress указан конкретный адрес, обрабатываются только запросы, поступившие на этот адрес.

Listen

Указывает адреса и порты, через которые принимаются запросы, в дополнение к стандартным. Адрес и порт в каждой паре разделяются двоеточием. Например, для приема запросов через порт 8080 и IP-адрес 172.16.12.5 создайте строку `Listen 172.16.12.5:8080`. Если указан только порт (без адреса), используется адрес сервера. В отсутствие инструкции Listen `httpd` работает только через порт, определенный инструкцией Port.

Инструкции BindAddress и Listen в настройках Solaris закомментированы.

Создание виртуальных узлов

Некоторые из закомментированных параметров файла *httpd.conf* могут использоваться, если на вашем сервере размещаются многие веб-сайты. К примеру, чтобы разместить веб-сайты доменов *fish.edu* и *mammals.com* на сервере *crab.wrotethebook.com*, добавьте следующие строки в *httpd.conf*:

```
<VirtualHost "www.fish.edu">
DocumentRoot /var/apache/fish
ServerName www.fish.edu
</VirtualHost>
<VirtualHost "www.mammals.com">
DocumentRoot /var/apache/mammals
ServerName www.mammals.com
</VirtualHost>
```

Каждый оператор VirtualHost определяет псевдоним узла, на который отзывается ваш сервер. Чтобы такая схема заработала, псевдоним должен иметь CNAME-определение в DNS. Наш пример требует существования CNAME-записей, назначающих узлу *crab.wrotethebook.com* псевдонимы *www.fish.edu* и *www.mammals.com*. Получив запрос, адресованный одному из этих псевдонимов, *crab* использует параметры настройки, определенные в соответствующем операторе, и замещает ими обычные настройки. Следовательно, получив запрос, адресованный *www.fish.edu*, сервер использует *www.fish.edu* в качестве значения ServerName вместо собственного имени сервера, а */var/apache/fish* в качестве значения DocumentRoot.

Безопасность веб-сервера

Веб-серверы уязвимы для всех стандартных проблем безопасности, описанных в главе 12, но привносят и собственные аспекты этой проблематики. Помимо защиты от обычных опасностей, веб-серверы должны способствовать сохранению целостности информации, которую предоставляют, а также информации, полученной от клиентов.

Доступ к информации сервера разграничивается посредством механизмов управления доступом. Файл *httpd.conf* позволяет администратору системы управлять доступом к серверу на уровне узла или на уровне отдельных пользователей. Управление доступом – важный аспект защиты внутренних и частных веб-страниц, однако большая часть информации в среде Web предназначена для массового распространения. В последнем случае нет никакого смысла ограничивать доступ к информации, но есть необходимость сохранять ее целостность.

Одной из опасностей, присущих только веб-серверу, является возможность изменения информации веб-страниц злоумышленником. Нам неоднократно приходилось слышать о вопиющих случаях таких нарушений, когда злоумышленники изменяли домашнюю страницу какого-нибудь правительственного ведомства, включая в нее материалы юмористического или порнографического содержания. И хотя подобные атаки не нацелены на причинение серверу долгосрочного вреда, они, вне всякого сомнения, мешают работе организации, которой принадлежит сайт.

Права доступа Unix защищают файлы и каталоги с документами. Серверу не нужны права записи, однако необходимы права на чтение и исполнение этих файлов. Плохо спроектированные исполняемые файлы всегда представляют потенциальную угрозу безопасности.

Угроза CGI и SSI

Сам по себе сервер Apache надежен и достаточно защищен. Наибольшей угрозой безопасности сервера является пользовательский код, исполняемый сервером – как правило, речь идет о программах CGI (Common Gateway Interface) и SSI (Server Side Includes).

Приложения CGI могут разрабатываться на C, Perl, Python и на любом другом языке программирования. Некачественные программы CGI представляют собой одну из самых серьезных угроз для безопасности сервера: злоумышленники могут использовать код этих программ для атак, основанных на переполнении буфера или передаче команд интерпретатора через программу системе. Защита состоит в предельно внимательном отношении к коду, который размещается в системе. Следует лично проверять все программы, хранимые в каталоге *cgi-bin*. Старайтесь писать программы, не принимающие от пользователя данные, вводимые в свободном формате: пользуйтесь элементами меню, а не клавиатурным вводом, где только возможно. Ограничивайте и проверяйте данные, поступающие от пользователя к системе.

Чтобы облегчить задачу аудита сценариев CGI, храните все сценарии в каталоге ScriptAlias. Не разрешайте исполнение сценариев из любых других каталогов, если нет полной уверенности, что в эти каталоги попадают только сценарии, проверенные вами лично. В следующем разделе представлены сведения о том, как определять каталоги исполняемых сценариев CGI, в ходе рассказа об инструкции Options.

Серверные включения SSI (Server Side Includes) также представляют собой потенциальную проблему – по тем же причинам, что и программы CGI. Серверные включения называют еще HTML серверной обработки (Server Parsed HTML), а файлы SSI часто имеют расширение *.shtml*. Эти файлы обрабатываются сервером до передачи клиенту и могут включать в себя другие файлы либо вызывать код внешних сценариев. Если для динамического изменения файла SSI используется ввод пользователя, файл становится уязвим для тех же видов атак, что и сценарии CGI.

Команды SSI встраиваются в комментарии HTML, а следовательно, начинаются символами `<!--` и завершаются символами `-->`. Команды SSI перечислены в табл. 11.3.

Таблица 11.3. Команды SSI (Server Side Include)

Команда	Назначение
`#config`	Задает формат отображения размера файла и времени
`#echo`	Отображает значения переменных
`#exec`	Исполняет сценарий CGI или команду интерпретатора
`#flastmod`	Отображает дату последнего изменения документа
`#fsize`	Отображает размер документа
`#include`	Включает внешний файл в текущий документ

Наиболее безопасный режим работы требует запрета на обработку инструкций SSI. Такова установка по умолчанию, изменяемая аргументами All или Includes инструкции Options в файле *httpd.conf*. Существует и компромисс – разрешить обработку SSI, но запретить команды `#include` и `#exec`. Они представлют наибольшую опасность, поскольку `#include` записывает в документ данные из внешнего файла, а `#exec` позволяет исполнять сценарии и команды. Воспользуйтесь аргументом IncludesNOEXEC инструкции Options, чтобы реализовать такую схему. Рассмотрим теперь создание инструкций Options для отдельных каталогов.

Управление параметрами сервера

Файл *httpd.conf* позволяет разграничивать доступ ко всем веб-документам либо доступ к документам в отдельных каталогах. Инструкция Options определяет, какие возможности сервера разрешены к применению для тех или иных документов. Инструкция Options внутри контейнера Directory действует только на конкретный каталог. Пример мы можем найти непосредственно в системе Solaris:

```
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
<Directory "/var/apache/htdocs">
    Options Indexes FollowSymLinks
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
<Directory "/var/apache/icons">
    Options Indexes MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
<Directory "/var/apache/cgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>
```

Данные инструкции управляют доступом к возможностям сервера для четырех каталогов: корневого (/), */var/apache/htdocs*, */var/apache/icons* и */var/apache/cgi-bin*. В примере отражены четыре из допустимых значений инструкции Options: FollowSymLinks, Indexes, None и MultiViews. Инструкция Options позволяет использовать следующие аргументы:

All

Разрешает использовать все возможности сервера.

ExecCGI

Разрешает исполнение сценариев CGI из указанного каталога. Параметр ExecCGI разрешает исполнение сценариев CGI из других каталогов, а не только из каталога ScriptAlias. Многие администраторы устанавливают этот параметр для каталога ScriptAlias, но это избыточное действие: инструкция ScriptAlias уже определила */var/apache/cgi-bin* в качестве каталога сценариев. В данном примере Options имеет значение None для каталога */var/apache/cgi-bin*, что не отменяет действия инструкции ScriptAlias.

FollowSymLinks

Разрешает использование символических ссылок. Если включен данный параметр, сервер считает символическую ссылку обычным документом из данного каталога.

Includes

Разрешает обработку инструкций SSI (Server Side Includes).

IncludesNOEXEC

Разрешает обработку файлов SSI, не содержащих команды `#exec` и `#include`.

Indexes

Разрешает передачу списка файлов каталога в отсутствие файла *index.html*.

MultiViews

Разрешает согласование языка документа. См. также описание инструкций AddLanguage и LanguagePriority (раздел «Определение файловых типов» выше по тексту).

None

Запрещает все возможности. Мой любимый аргумент!

SymLinksIfOwnerMatch

Разрешает использование символических ссылок в случаях, когда целевые файлы принадлежат владельцу ссылок.

Используйте серверные параметры с осторожностью. Параметры None и MultiViews, использованные в настройках Solaris, не должны вызывать проблем с безопасностью, хотя MultiViews поглощает ресурсы сервера. Параметр Indexes связан с легкими рисками, поскольку афиширует содержимое каталога в отсутствие файла *index.html*, что может быть нежелательным в отдельных случаях. FollowSymLinks потенциально связан с проблемами, поскольку символические ссылки увеличивают общее число каталогов с документами. Чем больше каталогов, тем сложнее обеспечивать их безопасность, поскольку для каждого необходимо установить определенные права доступа и за каждым необходимо наблюдать на предмет обнаружения поврежденных файлов. (В главе 12 содержится информация о Tripwire, инструменте, облегчающем задачу наблюдения за файлами.)

Контейнеры Directory из последнего примера содержат, помимо прочего, инструкции AllowOverride. Эти инструкции ограничивают возможности настройки отдельных каталогов.

Механизмы управления уровня каталога

Оператор `AccessFileName .htaccess` разрешает управление доступом на уровне отдельных каталогов и указывает имя файла с настройками каталога – *.htaccess*. Если сервер находит файл с таким именем в каталоге, из которого извлекается информация, то применяет содержащиеся в файле настройки, прежде чем передать данные. Инструкция AccessFileName передает управление настройкой тем, кто создает и сопровождает отдельные веб-страницы, предоставляя им файл, в который они могут записывать инструкции настройки. Инструкции настройки в файле *.htaccess* – те же, что встречаются в файле *httpd.conf* и определяют конфигурацию системы в целом. Настройки Solaris содержат строку `AccessFileName .htaccess`, поэтому в системах Solaris настройки уровня отдельных каталогов разрешены по умолчанию.

Инструкция AllowOverride может использоваться в целях ограничения степени свободы в настройках, предоставляемой отдельным каталогом. Она определяет, в каких случаях определения из файла *.htaccess* имеют больший приоритет, чем инструкции *httpd.conf*. Присутствие инструкции AllowOverride внутри контейнера Directory ограничивает область действия AllowOverride данным конкретным каталогом, как мы видели в предшествующем примере.

Инструкция AllowOverride имеет множество различных аргументов. В дополнение к ключевому слову `All`, разрешающему файлу *.htaccess* переопределять любые параметры файлов настройки, а также ключевому слову `None`, запрещающему переопределение, аргументами могут выступать отдельные инструкции, переопределение которых разрешено. К примеру, чтобы разрешить создание связей для файловых расширений в файле *.htaccess*, необходимо использовать инструкцию `AllowOverride AddType`. Когда это значение используется в качестве аргумента инструкции AllowOverride, файл *.htaccess* данного каталога может содержать инструкции AddType. AllowOverride позволяет разрешить переопределение практически любого параметра в файле *.htaccess*.

Инструкции Options и AllowOverride ограничивают доступ к возможностям сервера и механизмам управления, а также позволяют избежать повреждения информации. Но в некоторых случаях речь идет об информации, которая не должна получить широкого распространения. Распространение информации может пресекаться благодаря управлению доступом.

Создание правил доступа

Используйте файл *httpd.conf* для разграничения доступа на уровне отдельных узлов и пользователей. Несколько примеров позволят вам полностью понять возможности этого механизма. Начнем с примера разграничения доступа для узлов:

```
<Directory "/var/apache/htdocs/internal">
Order deny,allow
Deny from all
Allow from wrotethebook.com
</Directory>
```

В данном случае создаются правила доступа для каталога */var/apache/htdocs/internal*. Эти правила написаны таким образом, что доступ имеют только узлы домена *wrotethebook.com*. Контейнер Directory содержит три инструкции управления доступом:

`Order`

Определяет порядок использования правил управления доступом. `deny,allow` предписывает `httpd` использовать сначала запрещающее правило (`deny`), а затем разрешить исключения из него, описанные разрешающим правилом (`allow`). В данном примере мы заблокировали доступ всем узлам при помощи запрещающего правила, а затем создали исключение для систем, состоящих в домене *wrotethebook.com*, при помощи разрешающего правила. Это пример правил доступа, которые могут использоваться для защиты внутреннего веб-сайта.

`Deny from`

Указывает узлы, с которых запрещен доступ к веб-документам, хранимым в каталоге */var/apache/htdocs/internal*. Узлы могут обозначаться полными или частичными именами либо адресами IP. Каждая инструкция `Deny from` может обозначать только один источник; чтобы перечислить несколько источников, следует использовать несколько инструкций `Deny from`. Однако в случае использования доменного имени или адреса сети источник может включать каждый узел целого домена или сети. Ключевое слово `all` блокирует доступ с любых узлов.

`Allow from`

Указывает узлы, с которых разрешен доступ к документам каталога. Узлы могут обозначаться полными или частичными именами либо адресами IP. Каждая инструкция `Allow from` может обозначать только один источник; чтобы перечислить несколько источников, следует использовать несколько инструкций `Allow from`. Однако в случае использования доменного имени или адреса сети источник может включать каждый узел целого домена или сети. Ключевое слово `all` разрешает доступ с любых узлов.

Приведенный пример организует управление доступом на уровне отдельных узлов. Этот механизм обычно применяется для разделения информации, предназначенной для внутренних пользователей и внешних клиентов. Управление доступом можно также осуществлять на уровне пользователей и групп.

Принудительная проверка подлинности пользователей

Аутентификация пользователя может быть сделана необходимым условием для доступа к документу или каталогу. Как правило, эта возможность ис-

пользуется для предоставления доступа небольшой группе пользователей. Вот пример разграничения доступа для пользователей:

```
<Directory "/var/apache/htdocs/internal/accounting">
AuthName "Accounting"
AuthType Basic
AuthUserFile /etc/apache/http.passwords
AuthGroupFile /etc/apache/http.groups
Require hdqtrs rec bill pay
Order deny,allow
Deny from all
Allow from Limit>
</Directory>
```

Первые две инструкции в данном контейнере Directory называются AuthName и AuthType. AuthName определяет значение *области идентификации* (authentication realm), которое передается клиенту в заголовке `WWW-Authenticate`. Область – это группа ресурсов сервера с общими полномочиями доступа. К примеру, каталог */var/apache/htdocs/internal/accounting* является единственным элементом области Accounting. Однако области Accounting могут принадлежать и другие документы и каталоги, защищенные паролем. В таком случае пользователь, прошедший аутентификацию в области Accounting, получает доступ ко всем ресурсам этой области.

Инструкция AuthType определяет тип аутентификации, основанной на паролях. Может принимать значение Basic или Digest. Basic предписывает выполнять аутентификацию на основе паролей, передаваемых открытым текстом. Digest предписывает использовать алгоритм Message Digest 5 (MD5). Digest используется редко, отчасти потому, что этот механизм не полностью реализован в некоторых броузерах, но больше потому, что данные, требующие серьезной защиты, лучше защищать при помощи механизма безопасности SSL (Secure Sockets Layer). SSL мы рассмотрим позже, в разделе «Шифрование».

В приведенном примере доступ разрешается, если пользователь принадлежит к указанной группе и предоставил верный пароль. Эти группы и пароли не имеют ничего общего с теми, что используются программой `login`, но определяются и существуют только в контексте веб-сервера. Файлы, созданные с целью хранения сведений и пользователях и группах, обозначаются при помощи строк AuthUserFile и AuthGroupFile. Добавление паролей в файл паролей веб-сервера выполняется посредством программы `htpasswd`, поставляемой в составе системы Apache; добавление групп в файл групп может выполняться при помощи любого редактора текстов. Запись файла групп состоит из имени группы, за которым следует двоеточие и список пользователей, входящих в группу. К примеру:

```
hdqtrs: amanda pat craig kathy
```

Инструкция Require требует от пользователя ввода веб-имени и веб-пароля. В данном примере доступ разрешается только пользователям, принадлежа-

щим одной из групп `hdqtrs`, `rec`, `bill` или `pay`, причем пользователь должен ввести верный пароль. Ключевое слово `valid-user` в строке Require разрешает доступ всем пользователям с верными паролями, а файл групп при этом не используется.

Даже если вы не используете группы пользователей веб-сервера, создавайте строку AuthGroupFile, когда работаете с парольной аутентификацией. Чтобы не создавать пустой файл группы, просто укажите значение */dev/null*.

Инструкции Order, Deny и Allow в данном примере выполняют те же задачи, что и в предыдущем. Мы предписали выполнять парольную аутентификацию наряду с проверкой узлов. Последнее необязательно. Если удалить из примера инструкции Order, Deny и Allow, доступ к документам будет разрешен с любой системы сети Интернет, при условии, что пользователь указал верное имя и пароль.

Более совершенная аутентификация

Стандартный модуль аутентификации, mod_auth, хранит данные идентификации пользователей в текстовых файлах, поиск в которых осуществляется последовательно. На последовательный поиск может уходить значительное время, даже для нескольких сотен записей. Чтобы повысить производительность на данном этапе работы, воспользуйтесь базой данных с возможностью индексирования.

Поддержка баз данных аутентификации осуществляется двумя модулями: mod_auth_db, который работает с базами данных Berkeley DB, либо mod_auth_dbm, который работает с базами данных Unix DBM. В базовой настройке Solaris динамически загружается модуль mod_auth_dbm, так что мы можем использовать базу данных паролей в системе Solaris, затратив минимум усилий.

База данных паролей в применении мало чем отличается от последовательной базы. Если говорить о приведенном выше примере, мы можем перейти к использованию базы данных паролей, всего лишь заменив инструкцию AuthUserFile на инструкцию AuthDBMUserFile, а инструкцию AuthGroupFile на AuthDBMGroupFile. Вот пример:

```
<Directory "/var/apache/htdocs/internal/accounting">
AuthName "Accounting"
AuthType Basic
AuthDBMUserFile /etc/apache/passwords
AuthDBMGroupFile /etc/apache/groups
Require hdqtrs rec bill pay
Order deny,allow
Deny from all
Allow from Limit>
</Directory>
```

Этими небольшими изменениями все и ограничивается для файла *httpd.conf*. Самые серьезные изменения при переходе на базы данных паролей

связаны с тем, что пароли определяются теперь совсем не по команде htpassword. Вместо нее для создания записей паролей и групп используется команда dbmmanage, синтаксис которой приведен ниже :

```
dbmmanage filename command username password
```

Элементы командной строки dbmmanage по большей части прозрачны. *filename* – это имя файла базы данных. *username* и *password* – те самые данные, что должны храниться в базе данных паролей. *command* – одно из ключевых слов, определяющих назначение данной команды dbmmanage. Существуют следующие ключевые слова:

add

Добавляет имя пользователя и пароль в базу данных. Пароль должен быть предварительно зашифрован, поскольку dbmmanage не шифрует пароль по ключевому слову add. См. описание ключевого слова adduser.

adduser

Добавляет имя пользователя и пароль в базу данных. Пароль записывается открытым текстом и шифруется программой dbmmanage.

check

Проверяет соответствие имени пользователя и пароля той паре, что хранится в базе данных.

delete

Удаляет из базы данных имя пользователя и пароль.

import

Копирует записи *username*:*password* со стандартного потока ввода. Предполагается, что пароли уже зашифрованы.

update

Изменяет пароль для пользователя, запись которого уже существует в базе данных.

view

Отображает содержимое базы данных.

В следующем примере создается файл */etc/apache/passwords*, после чего в базу данных добавляются два новых пользователя:

```
# dbmmanage /etc/apache/passwords adduser sara
New password:
Re-type new password:
User sara added with password encrypted to XsH4aRiQbEzp2
# dbmmanage /etc/apache/passwords adduser alana
New password:
Re-type new password:
User alana added with password encrypted to AslrgF/FPQvF6
# dbmmanage /etc/apache/passwords view
```

```
alana:AslrgF/FPQvF6
sara:XsH4aRiQbEzp2
```

Обратите внимание, что если пароль не указан в командной строке, `dbmmanage` предлагает ввести его.

Все приведенные примеры осуществляют управление доступом для отдельных каталогов. Такое же управление доступом может выполняться для всех каталогов сервера либо отдельных документов. Чтобы задействовать механизмы управления доступом для всех документов сервера, достаточно просто разместить инструкции управления вне контейнера Directory; инструкции управления внутри контейнера Directory действуют на один каталог. Чтобы задействовать механизмы управления для отдельных файлов или документов, достаточно поместить соответствующие инструкции внутри контейнера Files или Document.

Управление доступом на уровне отдельных файлов

В настройках Solaris присутствует пример разграничения доступа к отдельным файлам. Чтобы скрыть файл *.htaccess* от глаз любопытного клиента, файл *httpd.conf* Solaris содержит следующий контейнер Files:

```
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>
```

Инструкции Order и Deny отличаются от тех, что мы видели в предшествующих примерах. В данном случае инструкция Order предписывает серверу Apache обработать сначала инструкцию Allow, а затем инструкцию Deny. Это позволяет инструкции Deny изменять правила доступа, созданные инструкцией Allow. Инструкция Allow отсутствует, а инструкция Deny запрещает любой внешний доступ к файлу *.htaccess*.

В действительности данная инструкция Deny действует не только для файлов *.htaccess* file. Тильда (~) в строке `Files` предписывает серверу Apache интерпретировать имена файлов в качестве регулярных выражений. Регулярное выражение `^\.ht` отбирает все файлы, имена которых начинаются с последовательности символов `.ht`. Причиной существования такого определения является привычка пользователей и администраторов начинать имена файлов настройки `httpd` со строки `.ht`: например, файл паролей пользователей может называться *.htpassword*. Регулярное выражение вместо имени файла в строке `Files` позволяет задействовать правила доступа для широкого диапазона файлов.

Управление доступом на уровне отдельных документов

Управление доступом к отдельным документам выполняется при помощи инструкции-контейнера Location. Вместо имени каталога (как в инструкции Directory) в инструкции Location фигурирует имя документа из URL-адреса. Инструкции внутри контейнера Location действуют только для указанного

документа. В следующем примере правила доступа действуют только для документа *server-status*:

```
<Location /server-status>
SetHandler server-status
Order deny,allow
Deny from all
Allow from wrotethebook.com
</Location>
```

Если сервер Apache получает запрос по адресу *www.wrotethebook.com/server-status*, то проверяет возможность доступа по созданным правилам. */server-status* – это имя документа, а не имя каталога. По сути, это специальный документ, отображающий сведения о состоянии сервера и создаваемый специальным обработчиком. Правила из контейнера Location разрешают просматривать состояние сервера всем пользователям нашего домена, но запрещают всем другим пользователям. Последний раздел этой главы рассказывает о том, как страница *server-status* используется для наблюдения за состоянием сервера. Однако прежде чем перейти к этой теме, рассмотрим последний аспект системы безопасности – защиту информации, передаваемой клиентом серверу.

Шифрование

Описанные в предшествующих разделах механизмы безопасности все до единого предназначены для защиты информации, которую предоставляет сервер. Однако на сервере лежит еще и ответственность за безопасность данных клиента. Чтобы создать систему электронной коммерции, необходимо использовать защищенный сервер, не позволяющий злоумышленникам получить доступ к личным данным покупателя, в частности к номеру его кредитной карты. Для шифрования сеансов работы, в которых передаются конфиденциальные данные, сервер Apache использует механизм SSL (Secure Sockets Layer, уровень защищенных сокетов).

В сравнении с механизмами, описанными ранее, SSL и сложнее и надежнее. Надежнее – потому, что использует шифрование с открытым ключом для реализации серьезной аутентификации и согласования шифрования в отдельных сеансах. SSL позволяет защищать обмен данными между клиентом и сервером – посредством их шифрования.

SSL сложнее, поскольку задействует шифрование с открытым ключом. Шифрование – область в принципе сложная, а шифрование с открытым ключом – в особенности. Работа шифрования с открытым ключом и, в частности, протокола SSL описана в главе 12. Если вам необходимы дополнительные сведения, прочтите указанную главу, прежде чем включать поддержку SSL в своем сервере Apache.

Поддержка SSL в Apache реализована модулем mod_ssl. В свою очередь mod_ssl в работе полагается на библиотеки шифрования, инструменты и реализации протоколов SSL из пакета OpenSSL. Пакет OpenSSL включен в со-

став многих систем Linux и некоторых систем Unix. Прежде чем устанавливать mod_ssl, убедитесь в наличии OpenSSL в системе; если пакет отсутствует, скопируйте исходные тексты с сайта *http://www.openssl.org*. Выполните программу `config`, поставляемую в комплекте, а затем `make`, чтобы скомпилировать OpenSSL. Выполните `make test` и `make install`, чтобы установить пакет.

После установки OpenSSL можно устанавливать и модуль mod_ssl. Во многих системах Linux и некоторых системах Unix mod_ssl является частью базовой системы Apache. Если для вашей системы это неверно, скопируйте пакет mod_ssl с сайта *http://www.modssl.org*. Повторно скомпилируйте Apache с ключом `--with-ssl`, чтобы включить в Apache поддержку расширений SSL.[1]

Установка mod_ssl добавляет в пример файла настройки Apache (обычно он называется *httpd.conf.default*) различные строки настройки SSL. Эти новые строки помещаются внутри контейнеров IfDefine, что позволяет включать поддержку SSL из командной строки `httpd`. Система Red Hat, в которой mod_ssl интегрирован в базовую систему Apache, является хорошим примером такого подхода. Вот контейнеры IfDefine для инструкций mod_ssl LoadModule и AddModule (система Red Hat):

```
<IfDefine HAVE_SSL>
LoadModule ssl_module         modules/libssl.so
</IfDefine>
<IfDefine HAVE_SSL>
AddModule mod_ssl.c
</IfDefine>
```

Инструкции LoadModule и AddModule выполняются только в случае, когда в командной строке `httpd` определен символ HAVE_SSL. Строка «HAVE_SSL» является произвольным выбором – в другой системе может использоваться просто «SSL». Главное, чтобы эта строка совпадала со значением, определенным в командной строке `httpd`. К примеру:

```
# httpd -DHAVE_SSL
```

Назначение команды – запустить сервер SSL Apache в системе Red Hat 7.2.

Помимо контейнеров для инструкций LoadModule и AddModule, существует контейнер IfDefine для специальных настроек сервера SSL. Вот такой контейнер из файла настройки Red Hat:

```
<IfDefine HAVE_SSL>
Listen 80
Listen 443
</IfDefine>
<IfDefine HAVE_SSL>
AddType application/x-x509-ca-cert .crt
AddType application/x-pkcs7-crl    .crl
</IfDefine>
```

[1] Отличным справочником по компиляции сервера Apache является книга «Linux Apache Web Server Administration», издательство Sybex.

```
<IfDefine HAVE_SSL>
<VirtualHost _default_:443>
ErrorLog logs/error_log
TransferLog logs/access_log
SSLEngine on
SSLCertificateFile /etc/httpd/conf/ssl.crt/server.crt
SSLCertificateKeyFile /etc/httpd/conf/ssl.key/server.key
<Files ~ "_(cgi|shtml|phtml|php3?)$">
    SSLOptions +StdEnvVars
</Files>
<Directory "/var/www/cgi-bin">
    SSLOptions +StdEnvVars
</Directory>
SetEnvIf User-Agent ".*MSIE.*" \
         nokeepalive ssl-unclean-shutdown \
         downgrade-1.0 force-response-1.0
CustomLog logs/ssl_request_log \
          "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x '%r' %b"
</VirtualHost>
</IfDefine>
```

Две строки в первом контейнере IfDefine предписывают серверу принимать соединения через порт 443, а не только через стандартный порт 80. Порт 443 используется в работе протокола SSL. Две строки во втором контейнере IfDefine связывают файловые расширения *.crt* и *.crl* с определенными файловыми MIME-типами. Расширения *.crt* и *.crl* относятся к сертификатам SSL. К сертификатам мы еще вернемся в этой главе.

Основная часть настроек сервера SSL содержится в контейнере VirtualHost. Данные настройки виртуального узла используются, когда поступает соединение на порт сервера по умолчанию, SSL-порт 443. Специальный файл журнала позволяет отслеживать SSL-запросы. Инструкции ErrorLog, TransferLog и CustomLog мы уже ранее встречали. Большинство прочих инструкций настройки действует только при работающей подсистеме SSL:

SSLEngine

Включает обработку SSL для данного виртуального узла.

SetEnvIf

Выполняет, по существу, ту же функцию, что и инструкции BrowserMatch, описанные ранее. В данном примере инструкция SetEnvIf проверяет, является ли User-Agent (броузер) агентом Microsoft Internet Explorer. В случае положительного ответа параметр `ssl-unclean-shutdown` уведомляет Apache, что данный броузер не умеет корректно закрывать соединения и что не следует использовать постоянные (keepalive) соединения при работе с броузером Internet Explorer.

SSLOptions

Определяет значения специальных параметров протокола SSL. В данном примере разрешены StdEnvVars для каталога */var/www/cgi-bin*, равно как и для всех файлов CGI и SSI. StdEnvVars – это переменные среды, пе-

редаваемые по соединению клиенту. Извлечение значений этих переменных требует от сервера затрат времени, поэтому они передаются только в случаях, когда клиент способен воспользоваться значениями – как в случаях сценариев CGI или файлов SSI.

SSLCertificateFile

Указывает на файл, содержащий открытый ключ сервера.

SSLCertificateKeyFile

Указывает на файл, содержащий закрытый ключ сервера.

Шифрование с открытым ключом требует существования двух ключей шифрования: открытого – и доступного всем клиентам, а также закрытого, который держится в секрете. Открытый ключ хранится в специальном формате, известном в качестве *сертификата*. Прежде чем запускать SSL на сервере, необходимо создать оба ключа.

OpenSSL предоставляет инструменты для создания открытых и закрытых ключей SSL. Простейшим из этих инструментов является файл сборки *Makefile* из каталога *ssl/certs*[1], позволяющий создавать сертификаты и ключи по команде `make`. Для создания сертификатов или ключей SSL в `make` может использоваться два типа аргументов. В первом типе для определения вида сертификата или ключа используется расширение файла:

`make` *`name`*`.key`

Создает закрытый ключ и записывает его в файл *`name`*`.key`.

`make` *`name`*`.crt`

Создает сертификат с открытым ключом и сохраняет его в файле *`name`*`.crt`.

`make name.pem`

Создает сертификат и ключ в формате PEM (Privacy Enhanced Mail) и сохраняет результат в файле *`name`*`.pem`. В главе 12 эта команда `make` используется в целях создания ключей для программы `stunnel`.

`make` *`name`*`.csr`

Создает запрос на подпись сертификата. Сертификат может получить цифровую подпись от доверенной инстанции, известной как *сертифицирующий орган* (*certificate authority*, CA), которая ручается за подлинность открытого ключа, хранимого в сертификате. Подробнее – через несколько абзацев.

Вторым типом аргументов для файла сборки являются ключевые слова. Ключевые слова позволяют создавать сертификаты и ключи, предназначенные для использования только в Apache:

`make genkey`

Создает закрытый ключ для сервера Apache. Ключ сохраняется в файле, на который указывает переменная KEY из файла *Makefile*.

---

[1] Путь указан относительно каталога, в котором установлен пакет OpenSSL. В нашей системе Red Hat полный путь выглядит так: */usr/share/ssl/certs*.

`make certreq`

Создает запрос на подпись сертификата для сервера Apache. Запрос на подпись сохраняется в файле, на который указывает переменная CSR из файла Makefile.

`make testcert`

Создает сертификат для сервера Apache. Данный сертификат может использоваться для загрузки и проверки сервера SSL. Однако этот сертификат не подписан действующим CA и, следовательно, не приемлем для использования в сети Интернет. Сертификат сохраняется в файле, на который указывает переменная CRT из файла Makefile.

В каталоге */etc/httpd/conf* системы Red Hat существует ссылка на файл сборки *Makefile*, облегчающая создание ключей в том месте файловой системы, где *httpd.conf* ожидает их найти. Если взглянуть в каталог */etc/httpd/conf* системы Red Hat, мы обнаружим, что ключи, обозначенные инструкциями SSLCertificateFile и SSLCertificateKeyFile, уже существуют, хотя мы их не создавали.

Для создания сертификатов и ключей *Makefile* использует команду `openssl`. Синтаксис команды `openssl` сложен и обширен, так что использование файла сборки приносит реальную пользу. Однако по-прежнему остается возможность напрямую пользоваться командой `openssl`, чтобы решать задачи, для которых не приспособлен файл *Makefile*. Например, чтобы просмотреть содержимое сертификата, который Red Hat поместил в каталог */etc/httpd/conf*, выполните следующую команду:

```
# openssl x509 -noout -text -in ssl.crt/server.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 0 (0x0)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=--, ST=SomeState, L=SomeCity, O=SomeOrganization,
                OU=SomeOrganizationalUnit,
                CN=localhost.localdomain/Email=root@localhost.localdomain
        Validity
            Not Before: Jul 27 12:58:42 2001 GMT
            Not After : Jul 27 12:58:42 2002 GMT
        Subject: C=--, ST=SomeState, L=SomeCity, O=SomeOrganization,
                 OU=SomeOrganizationalUnit,
                 CN=localhost.localdomain/Email=root@localhost.localdomain
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:a3:e7:ef:ba:71:2a:52:ff:d9:df:da:94:75:59:
                    07:f9:49:4b:1c:d0:67:b2:da:bd:7b:0b:64:63:93:
                    50:3d:a1:02:e3:05:3b:8e:e6:25:06:a3:d2:0f:75:
                    0a:85:71:66:d0:ce:f9:8b:b0:73:2f:fe:90:75:ad:
```

```
                d6:28:77:b0:27:54:81:ce:3b:88:38:88:e7:eb:d6:
                e9:a0:dd:26:79:aa:43:31:29:08:fe:f8:fa:90:d9:
                90:ed:80:96:91:53:9d:88:a4:24:0a:d0:21:7d:5d:
                53:9f:77:a1:2b:4f:62:26:13:57:7f:de:9b:40:33:
                c3:9c:33:d4:25:1d:a3:e2:47
            Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                55:E9:ED:C1:BF:1A:D4:F8:C2:78:6E:7A:2C:D4:9C:AC:7B:CD:D2
            X509v3 Authority Key Identifier:
                keyid:55:E9:ED:C1:BF:1A:D4:6E:7A:2C:D4:DD:9C:AC:7B:CD:D2
                DirName:/C=-/ST=SomeState/L=SomeCity/O=SomeOrganization/
                       OU=SomeOrganizationalUnit/CN=localhost.localdomain/
                       Email=root@localhost.localdomain
                serial:00
            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: md5WithRSAEncryption
        76:78:77:f0:a2:19:3b:39:5f:2a:bd:d0:42:da:85:6e:c2:0c:
        5e:80:40:9c:a8:65:da:bf:38:2b:f0:d6:aa:30:72:fb:d3:1d:
        ce:cd:19:22:fb:b3:cc:07:ce:cc:9b:b6:38:02:7a:21:72:7c:
        26:07:cc:c9:e0:36:4f:2f:23:c9:08:f7:d4:c1:57:2f:3e:5c:
        d5:74:70:c6:02:df:1a:62:72:97:74:0a:a6:db:e0:9d:c9:3d:
        8e:6b:18:b1:88:93:68:48:c3:a3:27:99:67:6f:f7:89:09:52:
        3a:a3:fb:20:52:b0:03:06:22:dd:2f:d2:46:4e:42:f2:1c:f0:
        f1:1a
```

Как видите, сертификат содержит большой объем информации. Однако лишь некоторые его фрагменты необходимы, чтобы определить, является ли он действительным для нашего сервера:

`Issuer`

Issuer – это *отмеченное имя* организации-издателя, которая выдала и подписала данный сертификат. Отмеченное имя записывается в формате, спроектированном таким образом, что позволяет обозначать организации уникальными именами. Очевидно, название организации в данном сертификате – вымышленное, и является всего лишь примером.

`Subject`

Subject – отмеченное имя организации, получившей сертификат. В нашем случае поле Subject должно содержать название нашей организации. Содержимое поля Subject в данном случае – тоже всего лишь пример.

`Validity`

Поле Validity содержит сведения о сроках действия сертификата. В данном случае сертификат действителен в течение года. Указанные даты позволяют использовать сертификат для тестирования работы SSL.

Чтобы убедиться, что сервер SSL действительно работает, воспользуйтесь броузером для подключения к локальному серверу. При этом начните URL с указания протокола `https://`, а не `http://`. `https` позволяет подключиться че-

рез порт 443, то есть через порт SSL. Броузер отвечает предупреждением, что сервером используется недействительный сертификат (рис. 11.4).

Кнопка View Certificate позволяет просмотреть часть той информации о сертификате, с которой мы только что познакомились. Пользователь может разрешить использование сертификата в данном сеансе и начать работу с «защищенным документом». В данном случае защищенный документ представлен просто тестовой страницей, поскольку мы еще не создали в своей системе ни одного настоящего защищенного документа.

Сервер запущен и функционирует, но внешние клиенты не смогут работать с ним, пока мы не получим действительный подписанный сертификат. Для создания запроса на подпись сертификата вашего сервера воспользуйтесь командой `make certreq`. Пример:

```
# cd /etc/httpd/conf
# make certreq
umask 77 ; \
/usr/bin/openssl req -new -key /etc/httpd/conf/ssl.key/server.key -out /etc/http
d/conf/ssl.csr/server.csr
Using configuration from /usr/share/ssl/openssl.cnf
You are about to be asked to enter information that will be incorporated
into your certificate request.
```

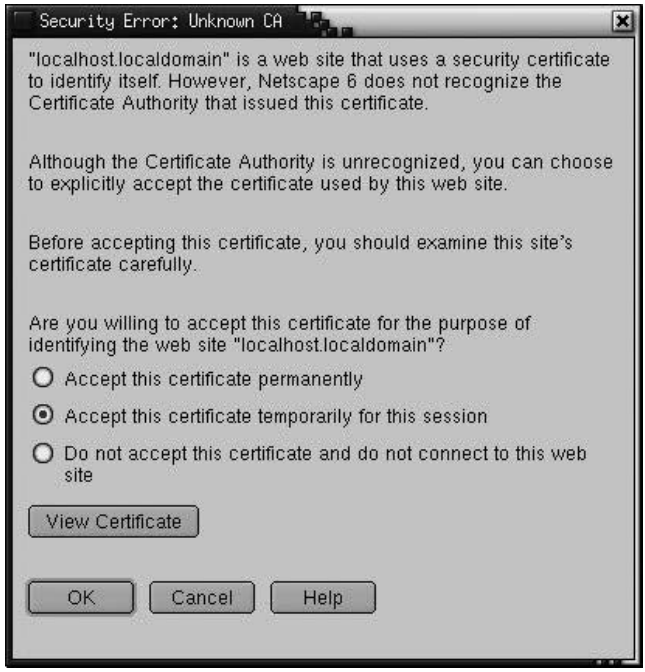


Рис. 11.4. *Предупреждение о недействительном сертификате*

```
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank.
For some fields there will be a default value.
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Maryland
Locality Name (eg, city) []:Gaithersburg
Organization Name (eg, company) [Internet Widgits Ltd]:WroteThebook.com
Organizational Unit Name (eg, section) []:Headquarters
Common Name (eg, your name or hostname)[]:crab.wrotethebook.com
Email Address []:alana@wrotethebook.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Только что созданный запрос можно просмотреть при помощи команды `openssl`**. Обратите внимание, что в данном запросе содержится уже действительное отмеченное имя нашего сервера в поле Subject. Однако отсутствует информация в поле Issuer. Чтобы сертификат стал действительным, он должен быть подписан признанным сертифицирующим органом.**

```
# openssl req -noout -text -in server.csr
Using configuration from /usr/share/ssl/openssl.cnf
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=US, ST=Maryland, L=Gaithersburg, O=WroteThebook.com,
                 OU=Headquarters,
                 CN=crab.wrotethebook.com/Email=alana@wrotethebook.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:a3:e7:ef:ba:71:2a:52:ff:d9:df:da:94:75:59:
                    07:f9:49:4b:1c:d0:67:b2:da:bd:7b:0b:64:63:93:
                    50:3d:a1:02:e3:05:3b:8e:e6:25:06:a3:d2:0f:75:
                    0a:85:71:66:d0:ce:f9:8b:b0:73:2f:fe:90:75:ad:
                    d6:28:77:b0:27:54:81:ce:3b:88:38:88:e7:eb:d6:
                    e9:a0:dd:26:79:aa:43:31:29:08:fe:f8:fa:90:d9:
                    90:ed:80:96:91:53:9d:88:a4:24:0a:d0:21:7d:5d:
                    53:9f:77:a1:2b:4f:62:26:13:57:7f:de:9b:40:33:
                    c3:9c:33:d4:25:1d:a3:e2:47
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: md5WithRSAEncryption
        3f:c2:34:c1:1f:21:d7:93:5b:c0:90:c5:c9:5d:10:cd:68:1c:
        7d:90:7c:6a:6a:99:2f:f8:51:51:69:9b:a4:6c:80:b9:02:91:
```

```
f7:bd:29:5e:a6:4d:a7:fc:c2:e2:39:45:1d:6a:36:1f:91:93:
77:5b:51:ad:59:e1:75:63:4e:84:7b:be:1d:ae:cb:52:1a:7c:
90:e3:76:76:1e:52:fa:b9:86:ab:59:b7:17:08:68:26:e6:d4:
ef:e6:17:30:b6:1c:95:c9:fc:bf:21:ec:63:81:be:47:09:c7:
67:fc:73:66:98:26:5e:53:ed:41:c5:97:a5:55:1d:95:8f:0b:
22:0b
```

Сертифицирующие органы работают на коммерческой основе и имеют своей целью получение прибылей. Чтобы ваш сертификат был подписан, необходимо заполнять формы, платить взносы, а также создать CSR-запрос. Ваш веб-броузер хранит список признанных сертифицирующих органов. В броузере Netscape 6.1 можно просмотреть этот список в разделе Certificate Manager меню Preferences (рис. 11.5). Каждый сертифицирующий орган поддерживает веб-сайт, на котором доступны сведения об оплате и процессе регистрации.

Хотя сертификаты, подписанные признанным CA, используются наиболее широко, возможно создание сертификатов с подписью автора. Впрочем, подобные сертификаты имеют ограниченную ценность. Как мы видели из рис. 11.4, сертификат, который не подписан признанным CA, требует подтверждения от клиента. Таким образом, подписанные автором сертификаты могут использоваться только в случае небольшого числа клиентов. Чтобы

Рис. 11.5. *Список признанных сертифицирующих органов в Netscape 6.1*

подписать сертификат самостоятельно, воспользуйтесь командой `openssl` следующим образом:

```
# openssl req -x509 -key ssl.key/server.key \
> -in ssl.csr/server.csr -out ssl.crt/server.crt
```

При просмотре созданного файла *server.crt* по команде `openssl` можно видеть, что поля Issuer и Subject содержат одно и то же отсеченное имя. И на этот раз оно является действительным именем нашего сервера.

Управление веб-сервером

Несмотря на огромное число параметров, определяемых в файле настройки *httpd.conf*, настройка – не самая сложная задача из тех, что приходится решать в процессе работы с веб-сервером. Настройка обычно требует изменения нескольких параметров при установке сервера, а вот наблюдение за использованием сервера, за его производительностью, обеспечение надежности и безопасности – задачи уже каждодневные. Сервер Apache предоставляет ряд инструментов, упрощающих их решение.

Наблюдение за сервером

Apache предоставляет инструменты для наблюдения за состоянием сервера, а также позволяет отслеживать использование и качество работы системы при помощи журналов. Отчасти этот вопрос был затронут ранее – в разговоре о настройке журналов. Мы даже изучили способ наблюдать за журналами в реальном времени.

Более удобным способом наблюдения за сервером в реальном времени является монитор состояния сервера – server-status. Монитор должен быть встроен в `httpd` либо установлен в виде динамически загружаемого модуля. Следующие две строки из файла Solaris *httpd.conf* выполняют установку модуля:

```
LoadModule status_module      modules/mod_status.so
AddModule mod_status.c
```

Чтобы получить максимум информации, добавьте в файл *httpd.conf* параметр ExtendedStatus:

```
ExtendedStatus on
```

Включите монитор в файле *httpd.conf* – при помощи контейнера `Location /server-status`. В файле *httpd.conf* Solaris такой контейнер существует по умолчанию, но закомментирован и потому не используется. Чтобы включить монитор, раскомментируйте строки и отредактируйте инструкцию Allow, указав узлы, с которых разрешено наблюдение за сервером. Пример:

```
<Location /server-status>
SetHandler server-status
```

```
Order deny,allow
Deny from all
Allow from wrotethebook.com
</Location>
```

Когда монитор установлен и включен, к нему можно обращаться при помощи броузера. Для нашей системы необходимо воспользоваться адресом URL *http://www.wrotethebook.com/server-status/?refresh=20*. Значение refresh не является обязательным, однако оно предписывает автоматическое обновление страницы состояния. В данном примере мы запрашиваем обновление страницы каждые 20 секунд. Страница состояния для нашего сервера показана на рис. 11.6.

Рис. 11.6. *Страница состояния сервера Apache*

Наблюдение позволяет оценивать состояние сервера в реальном времени. Ведение журналов позволяет получать информацию об использовании сервера в тот или иной момент. Сочетание журналов и наблюдения позволит вам поддерживать веб-службу в хорошей форме.

Резюме

Веб-серверы – неотъемлемая часть сети любой организации, и веб-сервер Apache является превосходным выбором. Он реализован в качестве демона HTTP (`httpd`) и настраивается посредством файла *httpd.conf*.

Предварительно настроенный и готовый к применению сервер Apache входит в состав систем Linux и Solaris. Просмотрите настройки и скорректируйте такие параметры, как ServerAdmin, ServerName и DocumentRoot, чтобы они соответствовали нуждам вашего сервера.

Для пристального наблюдения за использованием и производительностью системы задействуйте инструменты наблюдения и файлы журналов. Помните о безопасности сервера – тщательно контролируйте сценарии CGI (Common Gateway Interface) и инструкции SSI (Server Side Includes). Воспользуйтесь механизмом SSL для защиты конфиденциальных данных, поступающих от клиентов.

Данная глава завершает исследования в области настройки сервера TCP/IP – мы выполнили последнее задание. В следующей главе мы обратимся к повседневным задачам, неизбежно возникающим в работающей сети после первичной настройки и установки, и начнем этот разговор с безопасности.

- *Планирование безопасности*
- *Проверка подлинности пользователей*
- *Безопасность приложений*
- *Наблюдение за безопасностью*
- *Управление доступом*
- *Шифрование*
- *Брандмауэры*
- *Последнее напутствие*

12

Сетевая безопасность

Узлы, подключенные к сети – в особенности к всемирной сети Интернет, – сталкиваются с бульшим числом проблем, связанных с безопасностью, нежели прочие. Система безопасности сети сокращает риск несанкционированного подключения. Но по своей природе доступ к сети и компьютерная безопасность имеют прямо противоположные цели. Сеть – это скоростное шоссе, служащее для транспортировки данных и организации доступа к ним компьютерных систем, тогда как смысл безопасности заключен в управлении доступом к этим системам. Обеспечение сетевой безопасности – это процесс поиска равновесия между открытым доступом и безопасным доступом.

Аналогия со скоростным шоссе очень уместна. Подобно шоссе, сеть обеспечивает всех равным доступом – как желанных, так и незваных гостей. Находясь у себя дома, вы обеспечиваете безопасность собственности, запирая дверь, а не баррикадируя улицы. Точно так же сетевая безопасность требует принятия адекватных мер на уровне отдельных узлов. Недостаточно просто закрыть сеть брандмауэром.

В очень маленьких городках, где все люди знают друг друга, двери часто оставляют незапертыми. Однако в больших городах двери оборудуют замками и цепочками. Сеть Интернет из небольшого городка в несколько тысяч пользователей выросла в мегаполис с миллионами жителей. Как анонимность большого города превращает соседей в незнакомцев, так рост сети Интернет сократил уровень доверия между соседями по сети. И постоянно растущая потребность в компьютерной безопасности – неприятный побочный эффект такой ситуации. Рост, тем не менее, имеет и преимущества. Большой город дает больше свобод и предоставляет больше услуг, а растущая сеть – больше служб. Для большинства из нас мысли о безопасности – приемлемая цена за доступ к сети.

Взломы сетей с ходом времени учащаются и становятся обезличенными, но масштабы таких нарушений очень легко преувеличить. Помешательство на угрозе взлома может помешать использованию потенциала сети в полном объеме. Лекарство не должно быть хуже болезни. Лучший совет в разговоре о сетевой безопасности – используйте здравый смысл. Документ RFC 1244, на смену которому пришел RFC 2196, излагает данный принцип очень четко:

> Здравый смысл – наиболее подходящий инструмент для определения политики безопасности. Детально проработанные механизмы и программы обеспечения безопасности производят впечатление и, разумеется, находят применение, однако нет особого смысла вкладывать деньги и тратить время на сложные схемы, если забываются простейшие методы управления.

В данной главе акцент сделан на простые методы, которые позволяют совершенствовать безопасность сети. В смысле реальных финансовых расходов и производительности наиболее эффективен подход к безопасности, основанный на потребностях конкретной системы.

Планирование безопасности

Одной из наиболее важных – и, вероятно, самых неприятных – задач в сетевой безопасности является разработка политики безопасности сети. Компьютерщики чаще всего склоняются к поиску технического решения любой проблемы. Мы хотим найти программу, которая «излечит» проблему безопасности сети. Мало кому хочется писать документ, посвященный правилам и механизмам обеспечения сетевой безопасности. Однако четко проработанный план обеспечения безопасности поможет решить, что именно необходимо защищать, какие средства могут быть потрачены на решение задачи и кто будет отвечать за проведение плана в жизнь.

Оценка угрозы

Первый шаг к разработке эффективного плана обеспечения сетевой безопасности – оценка угрозы, создаваемой для ваших систем подключением к сети. Документ RFC 2196, *Site Security Handbook* (Учебник по безопасности сетевых площадок), описывает три различимых типа угроз, обычно связанных с подключением к сети:

Несанкционированный доступ

Доступ к системам постороннего лица.

Раскрытие информации

Любая проблема, вызывающая раскрытие ценной или чувствительной информации в пользу людей, которые не должны иметь к ней доступа.

Перегрузка служб (Denial of service, DoS)

Любая проблема, осложняющая продуктивную работу системы или препятствующая ее продолжению.

Перечисленные угрозы следует оценивать относительно числа пользователей, которых могут коснуться возникшие проблемы, а также степенью секретности информации, которая может быть раскрыта. Для некоторых организаций взлом – неприятность, подрывающая доверие к этой организации третьих лиц. Нарушители обычно направляют свои усилия на правительственные и университетские организации, у которых могут возникнуть осложнения из-за взлома. Но для большинства организаций несанкционированный доступ – явление вполне терпимое, пока оно не связано с двумя другими угрозами – раскрытием информации и отказом в обслуживании (denial of service).

Оценка угрозы раскрытия информации основывается на качестве информации, которую необходимо защитить. Разумеется, ни одна система, хранящая данные высокой секретности, не должна быть напрямую подключена к сети Интернет, но системы, работающие с другими видами секретной информации, могут иметь подобное подключение, не подвергаясь чрезмерной опасности. В большинстве случаев файлы, хранящие кадровую и медицинскую информацию, корпоративные планы и кредитные отчеты, могут быть адекватно защищены средствами разграничения сетевого доступа и стандартными механизмами разграничения доступа к файлам Unix. Тем не менее, если последствия раскрытия данных могут оказаться серьезными, имеет смысл пересмотреть вопрос о подключении узла к сети Интернет.

Отказ в обслуживании (denial of service, DoS), если он затрагивает многих пользователей или важные задачи организации, может оказаться серьезным осложнением. Некоторые системы могут подключаться к сети без особых опасений. Преимущество подключения отдельных рабочих станций и небольших серверов к сети Интернет обычно перевешивает риск нарушения работы служб, которыми пользуются отдельные пользователи и группы пользователей. Иные системы могут являться жизненно важными для функционирования вашей организации. Прежде чем подключать их к сети, следует всесторонне оценить угрозу нарушения работы служб ответственных систем.

Вероломство DoS-атаки проявляется в тот момент, когда ваша система становится игрушкой в руках злоумышленников. Получив несанкционированный доступ к системе, нарушители могут разместить в ней вредоносный код и использовать в качестве стартовой площадки для атак на другие системы. Чаще всего подобное случается с операционными системами Microsoft, но жертвой может стать компьютерная система любого типа. Не позволить использовать систему во вредоносных целях – важная причина для ее защиты.

На своих занятиях по компьютерной безопасности Брент Чепмен (Brent Chapman) приводит следующую классификацию угроз безопасности: угрозы секретности, угрозы доступности и угрозы целостности данных. Секретность – это необходимость предотвратить раскрытие конфиденциальной информации. Доступность – наличие информации и ресурсов для ее обработки в нужный момент (атака DoS нарушает доступность). Необходимость сохранять целостность информации также очевидна, но ее связь с компьютерной

безопасностью тоньше. Как только доступ к системе получило постороннее лицо, целостность информации, хранимой этой системой, оказывается под сомнением. Некоторые злоумышленники стремятся нарушить именно целостность данных; все мы слышали о том, как вандалы проникают на веб-серверы с единственной целью – изменить данные на сервере и таким образом воспрепятствовать нормальной работе организации. Оценить угрозу будет легче, если вы задумаетесь о последствиях для ваших данных.

Разумеется, сетевыми угрозами дело не ограничивается, существуют другие угрозы компьютерной безопасности, другие причины для недоступности служб. Природные бедствия и внутренние угрозы (исходящие от людей, которые обладают законным доступом к системе) также следует воспринимать всерьез. Сетевая безопасность широко рекламируется в прессе, так что беспокоиться о ней – очень модно, но из-за пожаров и перебоев в подаче электроэнергии потеряно, вероятно, больше машинного времени, чем из-за проблем с сетевой безопасностью. Точно так же больший объем данных подвергся раскрытию привилегированными пользователями, нежели в результате действий злоумышленников. Естественно, в этой книге сделан упор на сетевую безопасность, однако сетевая безопасность – лишь часть общей стратегии безопасности, которая включает аспекты физического уровня и способы восстановления в аварийных ситуациях.

Многие традиционные (не связанные с работой в сети) угрозы безопасности блокируются уровнем физической безопасности. Не забудьте обезопасить свое сетевое оборудование и кабели должным образом. Повторюсь, вложение средств в физическую безопасность должно основываться на реалистичной оценке угрозы.

Распределенное управление

Одним из подходов к обеспечению сетевой безопасности является распределение ответственности за управление различными сегментами крупной сети и делегирование ее отделам организации. Такой подход требует участия большого числа людей и идет вразрез с идеологией обеспечения безопасности путем централизации управления. Однако распределение ответственности и управления по небольшим административным группам позволяет создавать среду небольших, простых в наблюдении сетей с известными подгруппами пользователей. По аналогии с маленькими и большими городами, риск сокращается благодаря взаимовыручке жителей, взаимной ответственности друг за друга и возможности каждого из них управлять своей жизнью.

Кроме того, распределение должностных обязанностей, связанных с безопасностью, лишний раз подтверждает тот факт, что меры по обеспечению безопасности в большинстве случаев принимаются на отдельных системах. Руководители таких систем должны быть в курсе, что отвечают за безопасность и что их вклад в безопасность сети ценится и признан окружающими. Чтобы человек выполнял определенную работу, он должен получить соответствующие полномочия.

Распределение полномочий посредством подсетей

Подсети можно считать инструментом распределения полномочий управления сетью. При образовании подсети следует назначать администратора. Администратор подсети отвечает за ее безопасность и назначение IP-адресов подключаемым сетевым устройствам. Назначение адресов IP позволяет администратору подсети до некоторой степени контролировать состав машин, подключающихся через подсеть, а также помогает администратору определять, какие машины входят в подсеть и кто отвечает за каждую из машин. Когда администратор подсети назначает машине IP-адрес, он также делегирует администратору этой машины определенные обязанности в области безопасности. Точно так же, если администратор создает для пользователя учетную запись, пользователю вменяется в обязанность соблюдение определенных норм безопасности.

Иерархия ответственности выглядит так: администратор сети, администраторы подсети, администраторы систем, пользователи. На каждом уровне иерархии отдельные лица наделяются определенной ответственностью и сопутствующими полномочиями. Чтобы облегчить существование такой структуры, важно уведомить пользователей, за что они отвечают и как должны выполнять свои обязанности. Описанный в следующем разделе руководящий документ по стратегии обеспечения безопасности содержит подобную информацию.

Сеть как средство распространения информации

Если на сетевой площадке начинает использоваться распределенное управление, возникает потребность в системе распространения информации безопасности по группам. На каждом из административных уровней для предупреждений и передачи другой срочной информации могут использоваться списки рассылки. Кроме того, может быть создан веб-сайт внутреннего пользования, предоставляющий доступ к руководящим документам, дополнительной и архивной информации, а также ссылкам на важные сайты, посвященные проблемам безопасности.

Сетевой администратор получает информацию по обеспечению безопасности из компетентных внешних источников, отбрасывает ненужные сведения, и передает актуальные данные администраторам подсетей. Администраторы подсетей передают соответствующие фрагменты информации системным администраторам, а те, в свою очередь, передают сведения, которые считают важными, отдельным пользователям. Фильтрация информации на каждом из уровней гарантирует, что отдельные участники процесса получат информации не больше, чем требуется. Если объемы распространяемых материалов регулярно превышают разумный предел, пользователи начинают игнорировать все поступающие сведения.

На вершине описанной информационной структуры – сведения, полученные сетевым администратором из компетентных внешних источников. Чтобы получать такие сведения, сетевой администратор должен подписаться на соответствующие списки рассылки и форумы, а также регулярно просмат-

ривать тематические веб-сайты. Есть несколько пунктов, с которых имеет смысл начать поиск информации по компьютерной безопасности:

Поставщик/разработчик вашей системы Unix

Многие поставщики систем ведут собственные списки рассылки, посвященные безопасности, а у большинства на веб-сайте присутствует страница аналогичного содержания. Разместите на веб-сайте для внутреннего пользования ссылку на информацию от поставщика системы, которую сочтете важной и полезной.

Архив Bugtraq

Бюллетень Bugtraq сообщает об ошибках в программах, некоторые из которых используются злоумышленниками. Знание о подобных ошибках и способах их исправить – самое важная составляющая процесса повышения безопасности системы. Bugtraq широко доступен в среде Web. Я посещаю сайты *http://www.geek-girl.com/bugtraq* и *http://www.securityfocus.com*, предоставляющие массу информации по безопасности.

Форумы по безопасности

Конференции иерархии *comp.security* – *comp.security.unix*, *comp.security.firewalls*, *comp.security.announce* и *comp.security.misc* – позволяют получать сведения, представляющие интерес. Подобно большинству конференций, они содержат массу неинтересного и малополезного. Однако время от времени здесь случаются находки.

Веб-сайт FIRST

Форум реагирования на происшествия и комитетов безопасности (The Forum of Incident Response and Security Teams, FIRST) – это всемирная организация, объединяющая команды реагирования на происшествия в области компьютерной безопасности. Веб-сайт FIRST содержит информацию, связанную с компьютерной безопасностью.

Предупреждения института NIST

Отделение компьютерной безопасности Национального института стандартов и технологий ведет веб-сайт, предоставляющий указатели на посвященные безопасности веб-страницы по всему миру. Чтобы получить доступ к ссылкам, перейдите в раздел Alerts со страницы *http://csrc.nist.gov*.

Бюллетени CERT

Бюллетени CERT (Computer Emergency Response Team) содержат информацию об известных проблемах безопасности и средствах их решения. Получить бюллетени можно на сайте CERT по адресу *http://www.cert.org*.

Институт SANS

Институт Системного администрирования, сетей и безопасности (System Administration, Networking and Security, SANS) еженедельно рассылает информативные сообщения электронной почты, посвященные безопасности. На сайте SANS (*http://www.sans.org*) существует удобная читальня, а также прочие полезные ресурсы.

Сайты, посвященные уязвимостям систем (Exploit sites)

Большинство злоумышленников использует готовые сценарии для взлома, распространяемые в среде Web. Сайты, предоставляющие подобные сценарии, часто содержат форумы, посвященные обсуждению существующих уязвимостей различных систем. *http://www.insecure.org* – хороший сайт, поскольку хранит описания существующих уязвимостей, а также массу другой полезной информации.

Разработка политики безопасности

Безопасность, по большому счету, – «проблема персонала». За претворение в жизнь механизмов безопасности отвечают люди, а не компьютеры, и при нарушении системы безопасности ответственность ложится именно на людей. Следовательно, сетевая безопасность неэффективна, если персонал не в курсе собственных обязанностей. Очень важно создать руководящий документ, описывающий политику безопасности и четко определяющий права и обязанности каждого. Политика безопасности сети должна определять следующие моменты:

Обязанности по обеспечению безопасности отдельных пользователей сети

Политика безопасности может требовать от пользователей регулярной смены паролей, применения паролей, соответствующих определенным правилам, либо выполнения определенных проверок с целью выяснить, пользовался ли учетной записью посторонний. Какими бы ни были обязанности пользователей, они должны быть четко определены.

Обязанности системного администратора

Политика может требовать применения определенных средств обеспечения безопасности на каждом узле, присутствия уведомляющих приветственных сообщений для пользователей, равно как организации работы механизмов наблюдения и учета. Кроме того, могут быть перечислены приложения, запрещенные к применению на узлах сети.

Надлежащее использование ресурсов сети

Укажите, кому разрешено использовать ресурсы сети, как разрешено использовать ресурсы, что запрещено делать. Если ваша организация считает, что сообщения электронной почты, файлы и история активности пользователей являются предметом изучения для службы безопасности, доведите до сведения пользователей, что таковы правила.

Регламент действий при обнаружении проблемы безопасности

Что следует сделать, если обнаружена проблема безопасности? Кого уведомить? В момент кризиса легко что-то пропустить, поэтому необходимо четко определить точные шаги, которые должен предпринять администратор или пользователь при нарушении системы безопасности. Это может быть просто фраза «ничего не трогать, сообщить сотруднику службы безопасности». Но даже столь простые действия должны быть регламентированы в письменном виде в руководящем документе политики безопасности и доступны для изучения в любой момент.

Подключение к сети Интернет привносит определенные обязанности, связанные с безопасностью. В документе RFC 1281, *A Guideline for the Secure Operation of the Internet* (Указания по безопасной работе сети Интернет), приведены руководящие указания для пользователей и сетевых администраторов относительно безопасной и ответственной работы с сетью Интернет. Прочтение этого документа даст вам понимание того, какая информация должна быть представлена в руководящем документе по политике безопасности.

Чтобы создать полноценную политику безопасности, необходимо многое продумать. Приведенный выше примерный план описывает содержание такого документа, но если вы лично отвечаете за его создание, необходимо более подробное руководство. Я рекомендую прочесть RFC 2196, документ весьма полезный при создании плана обеспечения безопасности.

Планирование безопасности (оценка угрозы, распределение обязанностей и создание политики безопасности) – это фундамент сетевой безопасности, однако план должен быть проведен в жизнь, прежде чем даст результаты. В оставшейся части главы мы сосредоточимся на реализации базовых механизмов обеспечения безопасности.

Проверка подлинности пользователей

Выбор хороших паролей – одно из простейших мероприятий по обеспечению качественной сетевой безопасности. Пароли требуются для входа в системы, работа с которыми разрешается только после аутентификации пароля. Народные мифы гласят, что все нарушения в сетевой безопасности происходят по вине невероятно умных хакеров, находящих уязвимости в программном обеспечении. В действительности некоторые из наиболее известных нарушителей обрели доступ к системам простым угадыванием или кражей паролей либо эксплуатацией хорошо известных уязвимостей устаревшего программного обеспечения. Позже в этой главе приводятся указания по процессу обновления ПО, а также перечисляются способы предотвратить кражу паролей. Но сначала посмотрим, что можно сделать, чтобы предотвратить угадывание паролей.

Угадывание паролей облегчается рядом обстоятельств:

- Пароль совпадает с названием учетной записи. Учетные записи с такими тривиальными паролями называются тупыми (*joe accounts*).
- Существуют гостевые и демонстрационные учетные записи, не требующие пароля либо требующие опубликованного и широко известного пароля.
- Существуют системные учетные записи с паролями по умолчанию.
- Пользователи сообщают свой пароль другим пользователям или посторонним лицам.

Отгадывание подобных паролей не требует вообще никакого умения, только много свободного времени! Частая смена пароля является хорошей контрме-

рой. Однако, выбирая хорошие пароли, не торопитесь менять их настолько часто, чтобы стало трудно запоминать. Многие специалисты по безопасности рекомендуют менять пароль раз в 3–6 месяцев.

Более сложным вариантом угадывания паролей является *угадывание по словарю*. При этом используется программа, которая шифрует каждое слово из словаря (скажем, */usr/dict/words*) и сравнивает результат с зашифрованным паролем, хранящемся в файле */etc/passwd*. Угадывание по словарю не ограничивается словарными статьями. Доступные сведения о пользователе (имя, инициалы, номер телефона и т. д.) также служат исходными данными для программы угадывания. Угадывание по словарю делает очевидной необходимость защиты файла */etc/passwd*.

В некоторых системах для сокрытия шифрованных паролей от потенциальных злоумышленников применяется *файл теневых паролей*. Если ваша система предоставляет возможность работы с теневыми паролями, воспользуйтесь этой возможностью. Сокрытие шифрованных паролей значительно сокращает риск успешного угадывания.

Файл теневых паролей

Файлы теневых паролей имеют ограниченные права доступа, которые предотвращают их прочтение злоумышленниками. Зашифрованный пароль хранится только в файле теневых паролей, */etc/shadow*, но не в файле */etc/passwd*. Файл *passwd* доступен для чтения всем приложениям и пользователям, поскольку содержит информацию, востребованную самыми разнообразными программами. Файл *shadow* может быть прочитан только администратором и не дублирует сведения, хранящиеся в *passwd*. Он содержит только пароли и информацию, необходимую для работы с ними. В системе Solaris запись файла *shadow* имеет следующий формат:

```
username:password:lastchg:min:max:warn:inactive:expire:flag
```

username – регистрационное имя пользователя. *password* – зашифрованный пароль либо, в случае системы Solaris, одно из ключевых слов `NP` и `*LK*`. *lastchg* – дата последней смены пароля, записанная в виде числа дней, истекших с 1 января 1970 года до дня смены пароля. *min* – минимальное число дней, по истечении которого разрешено сменить пароль. *max* – число дней, по истечении которого пользователь обязан сменить пароль. *warn* – число дней до истечения срока действия пароля, за которое пользователь получает предупреждение. *inactive* – число дней бездействия учетной записи, по истечении которых происходит ее блокировка. *expire* – дата истечения действия учетной записи. Поле *flag* не используется.

Зашифрованный пароль хранится только в этом файле. Все поля паролей в файле */etc/passwd* содержат букву `x`, которая сообщает системе, что настоящие пароли следует искать в файле */etc/shadow*, содержащем зашифрованный пароль либо ключевые слова `NP` или `*LK*`. Ключевое слово `NP` означает отсутствие пароля и возможности регистрации в системе посредством данной

учетной записи. Системные учетные записи, такие как *daemon* и *uucp*, не позволяют пользователю работать с системой, поэтому в их полях паролей содержится ключевое слово NP. *LK* в поле пароля означает, что учетная запись блокирована и не может использоваться в принципе. Подобные ситуации в других системах обозначаются по-разному; в частности, некоторые системы Linux используют символы * и !!. Однако во всех системах существует способ отличить активные учетные записи пользователей от учетных записей прочих типов.

Наиболее важной целью применения файла *shadow* является защита пароля, но дополнительные поля этого файла могут использоваться для решения сопутствующих задач системы безопасности. Механизм *устаревания паролей* (*password aging*) позволяет определить время действия для каждого пароля. Когда допустимое время использования пароля подходит к концу, механизм устаревания уведомляет пользователя о том, что пароль необходимо сменить. Если смена пароля не произошла в отведенный период времени, пароль удаляется из системы, а учетная запись пользователя блокируется.

В механизме устаревания паролей задействованы поля *lastchg*, *max* и *warn*. С их помощью система узнает, когда в последний раз менялся пароль, сколько его следует хранить, а также когда следует уведомить пользователя о нависшей над ним опасности. Поле *min* файла теневых паролей предоставляет еще одну удобную возможность и более тонкий аспект устаревания паролей. Оно запрещает пользователю менять свой любимый пароль на любой другой, а потом сразу обратно. После смены пароля он должен использоваться в течение числа дней, определенного полем *min*, прежде чем будет разрешено сменить его снова. Таким образом, можно запретить пользователям проделывать популярный трюк, позволяющий избежать действительной смены пароля.

Поля *inactive* и *expire* способствуют исключению из системы незадействованных учетных записей. В данном случае «бездействие» определяется числом дней, в течение которого учетная запись пользуется устаревшим паролем. После устаревания пароля пользователь получает определенный срок для входа в систему и смены пароля. Если пользователь не сделает этого до истечения указанного числа дней, учетная запись блокируется и пользователь теряет возможность работать с системой.

Поле *expire* позволяет создавать учетные записи пользователей с ограниченным временем «жизни». По наступлении даты, хранимой в поле *expire*, учетная запись пользователя блокируется, даже если все еще является активной. Дата прекращения действия учетной записи хранится в виде числа дней, истекших с 1 января 1970 года.

В системе Solaris файл */etc/shadow* не редактируется напрямую. Его правка выполняется посредством окна Users инструмента admintool либо специальных ключей командной строки passwd. Окно Users представлено на рис. 12.1. Можно наблюдать поля username, password, min, max, warn, inactive и expire.

Ключи -n *min*, -w *warn* и -x *max* команды passwd системы Solaris позволяют указывать значения полей *min*, *max* и *warn* для файла */etc/shadow*. Перечислен-

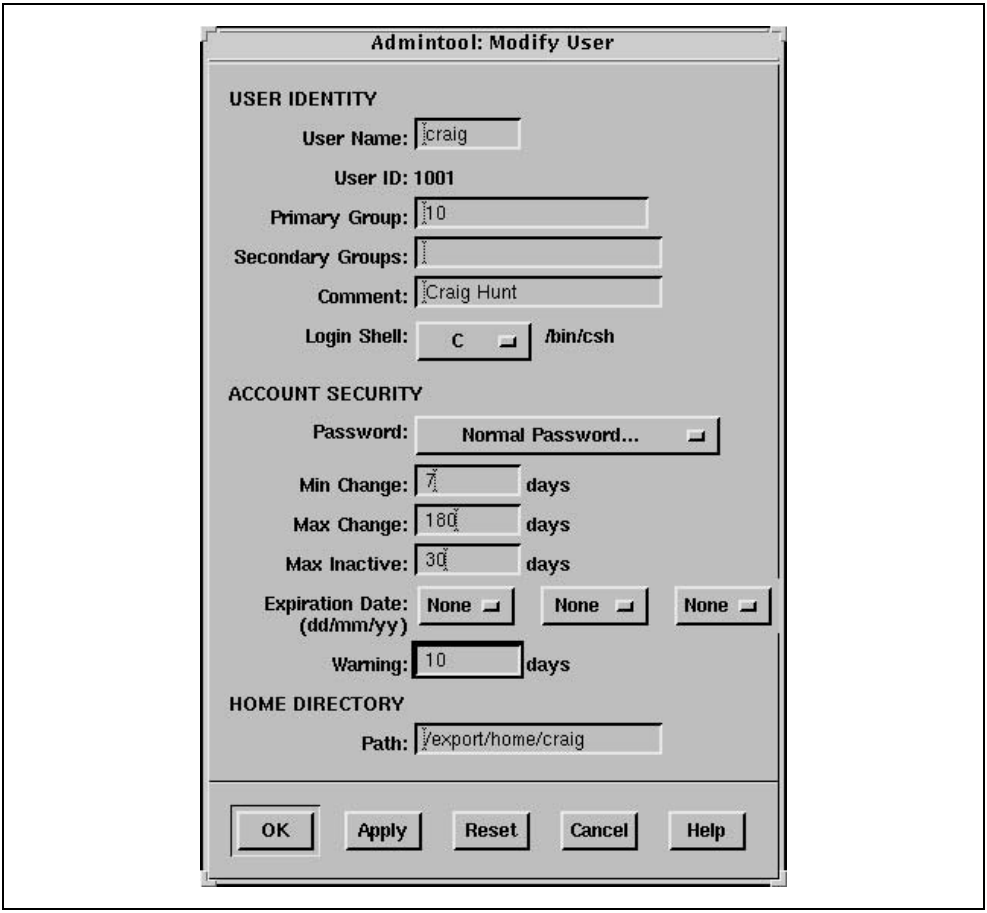


***Рис. 12.1.** Управление паролями посредством admintool*

ные ключи доступны только пользователю root. В данном примере администратор ограничивает время жизни пароля пользователя Tyler сроком в 180 дней:

```
# passwd -x 180 tyler
```

Система Solaris позволяет администратору установить значения по умолчанию для всех подобных параметров, чтобы не указывать их каждый раз при создании пользователя посредством `admintool` или командной строки `passwd`. Значения по умолчанию сохраняются в файле */etc/default/passwd*.

```
% cat /etc/default/passwd
#ident  "@(#)passwd.dfl 1.3     92/07/14 SMI"
MAXWEEKS=
MINWEEKS=
PASSLENGTH=6
```

В файле */etc/default/passwd* могут устанавливаться следующие значения по умолчанию:

`MAXWEEKS`

Максимальная продолжительность жизни пароля. Срок определяется в неделях, а не днях. Период в 180 дней из приведенного выше примера можно определить при помощи этого параметра как `MAXWEEKS=26`.

`MINWEEKS`

Минимальное число недель использования пароля до его смены.

`PASSLENGTH`

Минимальное ограничение на число символов в пароле. В приведенном примере минимальная длина пароля – 6 символов. В системе Solaris значимыми являются лишь 8 первых символов пароля, и установка большего значения не меняет этого факта.

`WARNWEEKS`

Число недель до окончательного устаревания пароля, за которое пользователь получает предупреждение.

В данном разделе приведен пример для системы Solaris. Система теневых паролей входит в состав операционной системы Solaris. Она также включается в дистрибутивы Linux. Описанный здесь файл *shadow* в системе Linux имеет в точности такой же формат и действует точно таким же образом.

Задача расшифровки зашифрованного пароля невероятно сложна, однако шифрованные пароли могут подвергаться сравнению с зашифрованными словами из словаря. Поэтому применение слабых паролей облегчает их угадывание. Позаботьтесь о защите файла */etc/passwd* и пользуйтесь качественными паролями.

Выбор пароля

Хороший пароль – существенный аспект обеспечения безопасности. Обычно мы думаем о паролях в контексте входа в систему, однако пароли, ключевые фразы и ключи используются также в более совершенных системах проверки подлинности. В любом случае следует выбирать сильный пароль. Выбор сильного пароля сводится к тому, чтобы не пользоваться паролями, которые могут быть отгаданы с применением описанных выше методов. Вот некоторые указания по выбору паролей:

- Не используйте регистрационное имя пользователя (login).
- Не используйте названия и вообще какие-либо имена.
- Не используйте слова и сокращения английского или другого языка.
- Не используйте персональные данные, связанные с владельцем учетной записи. Например, забудьте о собственных инициалах, номере телефона, номере социального страхования, названии должности, отделе организации и т. д.

- Не используйте последовательности клавиш, такие как qwerty.
- Не используйте перечисленную информацию, набранную наоборот, только заглавными буквами, искаженную любыми иными способами.
- Не используйте пароли, состоящие только из цифр.
- Не используйте примеры паролей, взятые из книг по компьютерной безопасности, независимо от их качества.
- *Используйте* смесь цифр, специальных символов и букв различного регистра.
- *Используйте* пароли не менее шести символов длиной.
- *Используйте* на первый взгляд случайные сочетания букв и цифр.

Следующие распространенные методы могут применяться для создания паролей из кажущихся случайными сочетаний:

- Используйте первые буквы слов строки текста из книги, песни или стихотворения. Например, «People don't know you and trust is a joke»[1] дает пароль Pd'ky&tiaj.
- Используйте вывод генератора случайных паролей. Выбирайте случайным образом полученные строки, которые легки для произношения и запоминания. Например, случайная строка «adazac» произносится как «а-да-зак», и ее можно запомнить при помощи мнемоники «от а до зет». Разбавьте результат заглавными буквами, чтобы внести собственное ударение: aDAzac.[2]
- Используйте пару коротких слов, связанных определенным символом: wRen%Rug.
- Используйте сочетание цифр и букв для создания воображаемого престижного регистрационного номера автомобиля: 2hot4U?.

Общее во всех перечисленных вариантах – пароль должен легко запоминаться. Избегайте паролей, которые необходимо записывать, чтобы запомнить. Если не заслуживающие доверия люди получат доступ к вашему рабочему месту и найдут записанный пароль, безопасность системы окажется под угрозой.

Не стоит думать, что невозможно запомнить случайный пароль. Его может быть нелегко использовать поначалу, однако любой часто применяемый пароль запоминается очень легко. Если речь идет об учетной записи на системе, с которой вы редко работаете, могут возникнуть сложности с запоминанием случайного пароля. В таком случае лучшим решением будет просто избавиться от учетной записи. Не используемые или мало востребованные учетные записи – первые цели злоумышленников. Им нравится атаковать такие учетные записи, поскольку пользователь не заметит изменений, внесенных в файлы, или подозрительных дат в сообщениях *Last login:*. Удаляйте все редко используемые учетные записи.

---

[1] Группа «Toad the Wet Sprocket», песня «Walk on the Ocean».

[2] Данный пароль создан при помощи генератора паролей.

Каким образом можно гарантировать, что пользователи будут следовать правилам создания новых паролей? Самый важный шаг – донести до пользователей собственно правила и то, насколько важно следование этим правилам. Уделите этой теме внимание при планировании безопасности, периодически обращайте на нее внимание пользователей в рассылках и на веб-доске объявлений.

Кроме того, возможно применение программ, принуждающих пользователей следовать определенным правилам выбора паролей. По адресу *http://csrc.nist.gov/tools/tools.htm* расположена страница, перечисляющая ряд таких программ.

Одноразовые пароли

Иногда хороших паролей оказывается недостаточно. Пароли передаются по сети открытым текстом. Злоумышленники могут использовать приложения анализа протоколов для просмотра сетевого трафика и кражи паролей. Если пароль украден, его качество перестает иметь какое-либо значение.

Вор может находиться в любой сети, через которую проходят ваши пакеты TCP/IP. Если обращение к системе происходит через локальную сеть, необходимо беспокоиться только о локальных шпионах. Но если обращение происходит через сеть Интернет, следует беспокоиться из-за невидимых слушателей в целом ряде неизвестных сетей.

Программы, работающие с шифрованными паролями, неуязвимы для подобных атак. По этой причине на смену `telnet` в широких масштабах приходит защищенный интерпретатор команд `ssh` (secure shell). Однако клиент защищенного интерпретатора команд может отсутствовать на удаленной сетевой площадке. В таком случае следует использовать одноразовые пароли для регистрации в удаленных системах. Поскольку одноразовый пароль может использоваться только раз, человек, укравший пароль, не сможет им воспользоваться.

Естественно, одноразовые пароли существенно осложняют жизнь. Необходимо иметь с собой перечень одноразовых паролей либо какой-то генератор таких паролей при каждой необходимости провести сеанс работы с системой. Не взяв с собой перечень, вы не сможете войти в систему. На первый взгляд проблема может показаться чрезмерно серьезной. Обычно пользователь регистрируется в системе с рабочего места и работает в основном с настольным компьютером либо одной из машин локальной сети. При работе с настольным компьютером пароль не передается по сети, поэтому пароли можно использовать повторно. А в случаях, когда обе стороны канала находятся под контролем, можно использовать `ssh`. Одноразовые пароли нужны только в случаях, когда работа происходит с удаленной системой без применения `ssh`. Таким образом, некоторые механизмы одноразовых паролей позволяют применять многоразовые пароли, когда это допустимо.

Существует несколько систем применения одноразовых паролей. В некоторых из них применяется специализированное аппаратное обеспечение, на-

пример смарт-карты. OPIE – это бесплатная программная система, не требующая специального аппаратного обеспечения.

OPIE

OPIE (*One-time Passwords In Everything, одноразовые пароли всегда и везде*) – бесплатное приложение, созданное Военно-морской исследовательской лабораторией США (NRL, U.S. Naval Research Laboratory), которое модифицирует систему Unix, позволяя использовать одноразовые пароли. OPIE является производной от Skey, системы одноразовых паролей, разработанной компанией Bell Communications Research (Bellcore).

Получить OPIE можно в сети Интернет по адресу *http://inner.net/opie*. Файл с текущей версией OPIE называется *opie-2.4.tar.gz*. Распакуйте файл командой `gunzip`, извлеките содержимое архива при помощи `tar`. В результирующем каталоге содержатся файлы исходных текстов, файлы сборки (Makefiles) и сценарии, необходимые для компиляции и установки OPIE.

В составе OPIE присутствует сценарий автоматической настройки `configure`, определяющий конфигурацию системы и надлежащим образом корректирующий make-файл. Сценарий хорошо выполняет свои задачи, но, тем не менее, следует проверить содержимое файла сборки и убедиться в его корректности. Например, в моей системе Linux установлен FTP-демон Вашингтонского университета – `wu.ftpd`. OPIE заменяет `login`, `su` и `ftpd` собственными вариантами этих программ. В случае моей системы более ранние версии OPIE `configure` не находили `ftpd`, а я не замечал этого, просматривая Makefile. Команда `make` отрабатывала без ошибок, однако в работе команды `make install` возникала ошибка при попытке установить FTP-демон OPIE. После тривиальной правки файла *Makefile* повторное выполнение `make install` было успешным.

Действие OPIE становится очевидным сразу после завершения установки. Выполнив команду `su`, вы получите приглашение `root's response:` вместо `Password:`. `login` отображает приглашение `Response or Password:` вместо просто `Password:`. Программы, выводящие приглашение `Response or Password`, позволяют ввести OPIE-ответ либо обычный пароль из файла */etc/passwd*. Данная возможность позволяет пользователям постепенно переходить от использования традиционных паролей к системе OPIE, а также проходить регистрацию на локальной машине при помощи паролей многократного применения и регистрацию на удаленных машинах при помощи одноразовых паролей. Все прелести жизни – удобная локальная регистрация без необходимости создавать раздельные учетные записи для локальной и удаленной проверки подлинности!

Чтобы использовать OPIE, необходимо, прежде всего, выбрать секретный пароль, на основе которого будет сгенерирован перечень одноразовых паролей, а затем выполнить программу-генератор. Чтобы выбрать пароль, выполните `opiepasswd` следующим образом:

```
$ opiepasswd -c
Updating kristin:
```

```
Reminder  -  Only use this method from the console; NEVER from remote.
 If you are using telnet, xterm, or a dial-in, type ^C now or exit with
 no password. Then run opiepasswd without the -c parameter.
Using MD5 to compute responses.
Enter old secret pass phrase: 3J5Wd6PaWP
Enter new secret pass phrase: 9WA11WSfW95/NT
Again new secret pass phrase: 9WA11WSfW95/NT
```

В данном примере пользователь Кристина (*kristin*) обновляет свой секретный пароль. Она выполняет `opiepasswd` с консоли компьютера (что и отражает ключ командной строки `-c`). Выполнение `opiepasswd` с консоли – вариант наиболее безопасный. Если команда выполняется не с консоли, необходимо иметь с собой копию программы `opiekey`, которая позволяет сгенерировать верные ответы, необходимые при вводе старых и новых секретных паролей, поскольку ввод паролей открытым текстом допускается только с консоли. Кристине предлагается ввести ее старый пароль и выбрать новый. Пароли OPIE должны иметь длину не менее 10 символов. Поскольку новый пароль имеет надлежащую длину, программа `opiepasswd` принимает его и выводит следующие две строки:

```
ID kristin OPIE key is 499 be93564
CITE JAN GORY BELA GET ABED
```

Эти строки дают Кристине информацию, необходимую для создания регистрационных OPIE-ответов и первого ответа для входа в систему. Одноразовый пароль для следующего входа в систему содержится во второй строке: группа из шести коротких строк прописных символов. Первая строка содержит начальный порядковый номер (499) и значение инициализации (be93564), которые в совокупности с секретным паролем позволяют Кристине генерировать регистрационные OPIE-ответы. Такие ответы генерируются программой `opiekey`.

`opiekey` на входе берет порядковый номер, значение инициализации, а также секретный пароль пользователя, а генерирует корректный одноразовый пароль. Если в системе, с которой происходит регистрация, присутствует `opiekey`, одноразовые пароли можно генерировать по одному за раз. В противном случае можно заранее применить ключ `-n` и получить набор паролей. Запишите пароли, положите их в бумажник – и вперед![1] В следующем примере мы запрашиваем пять (`-n 5`) ответов у `opiekey`:

```
$ opiekey -n 5 495 wi01309
Using MD5 algorithm to compute response.
Reminder: Don't use  opiekey  from  telnet  or dial-in sessions.
```

[1] Специалисты по безопасности испытают отвращение, прочтя это предложение. Записывать пароли – что может быть хуже? Честно говоря, я считаю, что люди, крадущие мой бумажник, больше заинтересованы в моих деньгах, чем в паролях к моей системе. Однако в каждом конкретном случае это предложение следует рассматривать в контексте уровня защищенности, необходимого вашей системе.

```
Enter secret pass phrase: UUaX26CPaU
491: HOST VET FOWL SEEK IOWA YAP
492: JOB ARTS WERE FEAT TILE IBIS
493: TRUE BRED JOEL USER HALT EBEN
494: HOOD WED MOLT PAN FED RUBY
495: SUB YAW BILE GLEE OWE NOR
```

Сначала `opiekey` сообщает нам, что для создания ответов используется алгоритм MD5, что для OPIE является умолчанием. В целях совместимости с более старыми реализациями Skey и OPIE можно предписать `opiekey` использовать алгоритм MD4 – при помощи ключа командной строки `-4`. `opiekey` предлагает ввести секретный пароль, то есть пароль, определенный при помощи `opiepasswd`. Затем `opiekey` отображает столько ответов, сколько запрошено, и выводит их в порядке возрастания порядковых номеров. В данном примере используются порядковые номера от 495 до 491. Когда порядковый номер опустится до 10, выполните повторно `opiepasswd` и выберите новый секретный пароль. Выбор нового пароля вновь устанавливает порядковый номер в значение 499.

Регистрационное приглашение OPIE содержит порядковый номер, и пользователь должен набрать ответ, соответствующий этому номеру. Пример:

```
login: tyler
otp-md5 492 wi01309 Response or Password:
JOB ARTS WERE FEAT TILE IBIS
```

По приглашению `login:` пользователь Tyler вводит свое имя. Система отображает строку, уведомляя пользователя, что одноразовые пароли генерируются по алгоритму MD5 (`otp-md5`), порядковый номер регистрации в системе – 492, и что значение инициализации для генератора одноразовых паролей – wi01309. Пользователь находит ответ для порядкового номера регистрации 492 и набирает шесть коротких строк. Затем вычеркивает ответ из своего списка, поскольку его уже нельзя использовать для входа в систему. Ответ из списка должен использоваться всякий раз, когда регистрация в системе происходит не с локальной консоли. Многоразовые пароли могут использоваться только при регистрации с консоли.

Защищенный командный интерпретатор используется для удаленной регистрации, если доступен клиенту. Вследствие этого одноразовые пароли требуются лишь в особых случаях. Как правило, достаточно иметь один небольшой OPIE-сервер в вашей сети. Удаленные пользователи, вынужденные использовать одноразовые пароли, входят на этот сервер, а затем применяют предпочтительный метод (скажем, `ssh`) для работы с настоящими серверами.

Защита r-команд

В некоторых приложениях применяются собственные механизмы безопасности. Убедитесь, что эти механизмы настроены надлежащим образом. В частности, проверьте `r`-команды Unix, а именно набор сетевых Unix-приложений, сравнимых с `ftp` и `telnet`. Необходимо знать точно, что `r`-команды не

нанесут вреда безопасности системы. Неверно настроенные r-команды могут открыть доступ к вашим компьютерам практически всем желающим. Вследствие этого r-команды использовать не рекомендуется.

Вместо парольной аутентификации r-команды действуют по системе доверенных узлов и пользователей. Доверенным пользователям с доверенных узлов разрешается доступ к локальной системе без пароля. Доверенные узлы называют также «равноценными узлами», поскольку система предполагает, что пользователь, имеющий доступ к доверенному узлу, должен иметь равноценный доступ к локальному узлу. Предполагается, что учетные записи с одинаковыми именами на разных узлах «принадлежат» одному пользователю. Например, пользователь *becky* на доверенном узле имеет те же полномочия доступа, что и пользователь *becky* локальной системы.

Такой механизм проверки подлинности требует присутствия баз данных, определяющих доверенные узлы и доверенных пользователей. Такими базами данных для r-команд служат файлы */etc/hosts.equiv* и *.rhosts*.

Файл */etc/hosts.equiv* определяет, какие узлы и пользователи могут иметь доступ к локальной системе посредством r-команд. Файл может также явным образом запрещать доступ определенным узлам и пользователям. Отсутствие доверительного доступа не значит, что доступ пользователю запрещен; пользователь по-прежнему может получить доступ, указав пароль.

Базовый формат записей файла */etc/hosts.equiv*:

```
[+ | -][имя узла] [+ | -][имя пользователя]
```

Здесь *имя узла* – это имя «доверенного» узла, которому может предшествовать знак сложения (+). Знак сложения не имеет особого смысла, за исключением случая, когда фигурирует отдельно, без имени узла. Знак сложения без имени узла – это маска, обозначающая все узлы.

Пользователям равноценного узла разрешено работать с вашей системой, не указывая пароля, при условии, что совпадают имена учетных записей. (Это одна из причин для администраторов соблюдать последовательность в выборе регистрационных имен.) Необязательное *имя пользователя* – это имя пользователя доверенного узла, которому предоставляется доступ ко всем учетным записям. Указание имени пользователя означает, что пользователь не ограничен учетной записью с идентичным именем, но получает беспарольный доступ ко всем пользовательским учетным записям.[1]

Имени узла может предшествовать знак вычитания (-), который говорит о том, что указанный узел *не* является равноценной системой. Пользователи этого узла, обращаясь к локальной системе при помощи r-команд, всегда должны указывать пароль. Знак вычитания может предшествовать также имени пользователя. В таком случае, независимо от других разрешенных для узла действий, указанный пользователь не является доверенным, и всегда обязан указывать пароль.

---

[1] За исключением учетной записи *root*.

Следующие примеры иллюстрируют интерпретацию записей файла *hosts.equiv*:

rodent

Разрешает беспарольный доступ к локальной системе всем пользователям узла *rodent*, для которых существуют учетные записи с идентичными именами.

-rodent

Запрещает беспарольный доступ всем пользователям узла *rodent*.

rodent -david

Запрещает беспарольный доступ пользователю *david* узла *rodent*.

rodent +becky

Разрешает пользователю *becky* узла *rodent* беспарольный доступ ко всем учетным записям (кроме *root*) локальной системы.

+ becky

Разрешает пользователю *becky* беспарольный доступ ко всем учетным записям (кроме *root*) локальной системы независимо от того, с какой системы работает пользователь.

Последняя запись – пример того, чего никогда не следует допускать при настройке. Не используйте просто знак сложения вместо имени узла. Он разрешает доступ с любых узлов и открывает широкие просторы для деятельности злоумышленников. Например, если приведенная выше запись присутствует в файле *hosts.equiv*, злоумышленник может создать на своей системе учетную запись *becky* и получить доступ ко всем учетным записям вашей системы. Проверьте файлы */etc/hosts.equiv*, *~/.rhosts* и */etc/hosts.lpd* на предмет наличия опасных +-записей. Не забудьте проверить файлы *.rhosts* во всех домашних каталогах пользователей.

Простая опечатка может стать причиной появления самостоятельного знака сложения. К примеру, возьмем такую запись:

```
+ rodent becky
```

Системный администратор, вероятно, имел в виду «предоставить *becky* беспарольный доступ ко всем учетным записям, если обращение исходит от узла *rodent*». Однако лишний пробел после символа + изменяет смысл записи на такой: «разрешить пользователям *rodent* и *becky* беспарольный доступ, если обращение исходит от любого узла». Не используйте символ + перед именем узла и всегда проявляйте осторожность в работе с файлом */etc/hosts.equiv*, чтобы избежать неприятных последствий.

Работая с файлом */etc/hosts.equiv*, предоставляйте доверенный доступ лишь системам и пользователям, которым действительно доверяете. Ни в коем случае не следует доверять всем системам локальной сети. Более того, лучше вообще не пользоваться r-командами. Если же без них не обойтись, доверяйте только узлам локальной сети, для которых известны ответственные

лица, которые доступны лишь ограниченному кругу пользователей, и только в случае, если локальная сеть защищена брандмауэром. Не стоит предоставлять доступ по умолчанию – должны быть причины для присвоения статуса доверенного пользователя или узла. Никогда не доверяйте внешним системам. Злоумышленник может с легкостью нарушить маршрутизацию или скорректировать данные DNS, чтобы обмануть вашу систему и получить доступ. Кроме того, никогда не начинайте файл *hosts.equiv* со знака вычитания. В некоторых системах это приводит к некорректному предоставлению доступа. Создавая файл *hosts.equiv*, всегда лучше перестраховаться. Добавлять доверенные узлы по мере их появления гораздо легче, чем восстанавливать систему после атаки злоумышленника.

Файл *.rhosts* разрешает или блокирует беспарольный доступ посредством `r`-команд для учетной записи конкретного пользователя. Он хранится в исходных каталогах пользователей и содержит записи для доверенных узлов и пользователей. Формат записей файла *.rhosts* совпадает с форматом записей *hosts.equiv*, и записи действуют почти таким же образом. Различие заключено в границах доступа, который предоставляют записи. Записи файла *.rhosts* предоставляют или запрещают доступ к учетной записи отдельного пользователя, записи файла *hosts.equiv* управляют доступом к системе в целом.

Данное функциональное различие можно проиллюстрировать на простом примере. Рассмотрим такую запись:

```
horseshoe anthony
```

В файле *hosts.equiv* узла *crab* данная запись разрешает пользователю *anthony* узла *horseshoe* беспарольный доступ к любой учетной записи *crab*. В файле *.rhosts* в исходном каталоге пользователя *resnick* точно такая же запись разрешает пользователю *anthony* обратиться посредством `rlogin` с узла *horseshoe* к учетной записи *resnick*, не предоставляя пароля, но не предоставляет беспарольного доступа к прочим учетным записям узла *crab*.

Пользователь при помощи файла *.rhosts* может определить в качестве равноценных принадлежащие ему различные учетные записи. Приведенный выше пример записи может иметь место, вероятнее всего, лишь в том случае, если пользователи *anthony* и *resnick* – одно лицо. Например, у меня есть учетные записи в ряде различных систем. Иногда мое имя пользователя – *hunt*, а иногда – *craig*. Было бы здорово иметь везде одинаковые имена учетных записей, но это не всегда возможно; в моей локальной сети имена *craig* и *hunt* уже используются другими людьми. Я хочу иметь возможность обратиться к своей рабочей станции с любого узла, на котором у меня есть учетная запись, но запретить доступ другим пользователям с именами *craig* и *hunt*. Файл *.rhosts* позволяет мне решить эту задачу.

Допустим, мое имя на узле *crab* – *craig*, а на узле *filbert* – *hunt*. Имя другого пользователя узла *filbert* – *craig*. Чтобы получить беспарольный доступ к своей учетной записи на *crab* с узла *filbert*, а также запретить такой доступ второму пользователю, я создаю следующий файл *.rhosts* в своем исходном каталоге:

```
filbert hunt
filbert -craig
```

Как правило, прежде всего происходит обращение к файлу *hosts.equiv*, а затем – к файлу пользователя *.rhosts*, если таковой существует. Первое явное соответствие определяет, разрешен ли беспарольный доступ. Следовательно, файл *.rhosts* имеет приоритет меньший, нежели файл *hosts.equiv*. Исключением из этого правила является доступ для пользователя root. Когда администратор системы пытается получить доступ к системе при помощи r-команд, происходит чтение только файла *.rhosts* в исходном каталоге пользователя *root*. Это позволяет более жестко управлять доступом пользователя *root*. Если бы для разграничения root-доступа использовались файлы *hosts.equiv*, записи, предоставляющие доступ доверенным узлам, давали бы пользователям *root* этих узлов полномочия администратора на локальной системе. Можно добавлять доверенные узлы в *hosts.equiv*, не предоставляя удаленным администраторам root-доступ к локальной системе.

Следует помнить, что пользователь способен предоставить доступ посредством файла *.rhosts*, даже если файл *hosts.equiv* не существует. Единственный способ воспрепятствовать этому – периодически проверять наличие файлов *.rhosts* и удалять их. До тех пор пока в системе присутствуют r-команды, пользователи имеют возможность случайно нарушить безопасность системы.

Защищенный интерпретатор команд

Слабая защищенность r-команд представляет угрозу безопасности системы. Этими командами нельзя пользоваться для обеспечения защищенного удаленного доступа, даже учитывая все аспекты, описанные в предшествующем разделе. В лучшем случае доступ посредством r-команд можно предоставлять только доверенным системам защищенной локальной сети. Причина кроется в том, что доверие r-команд основывается на убежденности в четком соответствии IP-адреса совершенно определенному компьютеру. Обычно так и есть. Однако злоумышленник способен исказить данные DNS и подменить адрес IP либо данные маршрутизации с целью доставки информации в другую сеть, таким образом обойдя механизмы проверки подлинности, используемые в r-командах.

Альтернативой такому удаленному доступу является *защищенный интерпретатор команд*. Защищенный интерпретатор заменяет стандартные r-команды более защищенными вариантами, задействующими шифрование и более совершенный механизм проверки подлинности. Последний позволяет гарантировать, что доверенный узел действительно является таковым. Методы шифрования на основе открытых ключей реализуют проверку подлинности пакетов сетевого потока. Защищенный интерпретатор команд не только безопасен, но и легок в применении.

В настоящее время широкое распространение получили два варианта защищенного интерпретатора команд: коммерческий продукт SSH Secure Shell и продукт с открытым исходным кодом – OpenSSH. OpenSSH входит в состав

различных вариантов Unix и Linux, а оба продукта – как коммерческий, так и бесплатный – доступны для загрузки из сети Интернет на случай, если в вашей системе нет никакого. Примеры этого раздела созданы с применением OpenSSH, но базовые функции обоих вариантов интерпретатора по существу одинаковы.

Основные компоненты защищенного интерпретатора команд:

`sshd`

Демон защищенного интерпретатора команд обрабатывает входящие SSH-соединения. `sshd` следует запускать в процессе загрузки системы – из загрузочного сценария, но не при помощи *inetd.conf*. При каждом запуске `sshd` генерирует ключ шифрования, поэтому его запуск может быть слишком долгим для *inetd.conf*. На системе, принимающей SSH-соединения, должен работать демон `sshd`.

`ssh`

Пользовательская команда для защищенного интерпретатора команд. Команда `ssh` заменяет `rsh` и `rlogin`. Она позволяет передать команду удаленной системе либо провести сеанс работы с удаленной системой безопасным способом. Команда создает исходящие соединения, обращенные к удаленному демону защищенного интерпретатора команд. Система-клиент для работы с SSH-соединением должна иметь доступ к команде `ssh`.

`scp`

Команда `scp` (secure copy) является заменой команды `rcp`.

`ssh-keygen`

Генерирует открытый и закрытый ключи шифрования, используемые для защищенной передачи данных.

`sftp`

Вариант клиента FTP, работающий через защищенное соединение.

Когда `ssh`-клиент обращается к серверу `sshd`, они обмениваются открытыми ключами. Системы сравнивают полученные ключи с ключами, хранимыми в файлах */etc/ssh_known_hosts* и *.ssh/known_hosts* (в исходном каталоге пользователя).[1] Если ключ не найден или изменился, пользователю предлагается подтвердить прием ключа:

```
> ssh horseshoe
Host key not found from the list of known hosts.
Are you sure you want to continue connecting (yes/no)? yes
Host 'horseshoe' added to the list of known hosts.
craig's password: Watts.Watt.
Last login: Thu Sep 25 15:01:32 1997 from rodent
```

[1] Администратор системы может инициализировать файл *ssh_known_hosts*, выполнив команду `make-ssh-known-hosts`, которая извлекает ключи всех узлов выбранного домена.

```
Linux 2.0.0.
/usr/X11/bin/xauth:  creating new authority file /home/craig/.Xauthority
```

Если ключ обнаружен в одном из файлов либо принят пользователем, клиент использует его для шифрования случайным образом сгенерированного ключа сеанса. Ключ сеанса передается серверу, и обе системы используют этот ключ для шифрования данных до конца текущего сеанса SSH.

Клиент проходит проверку подлинности, если он упомянут в файле *hosts.equiv*, *shost.equiv*, пользовательском файле *.rhosts* либо файле *.shosts*. Такой механизм проверки подлинности схож с тем, что используется `r`-командами, а формат файлов *shost.equiv* и *.shosts* совпадает с форматом их `r`-эквивалентов. Обратите внимание, что в приведенном выше примере пользователю было предложено набрать пароль. Если клиент не упомянут в одном из файлов, включается механизм парольной аутентификации. Как можно видеть, пароль отображается открытым текстом. Однако не стоит беспокоиться о краже пароля, поскольку SSH шифрует его, прежде чем передать второй стороне.

В целях аутентификации пользователь может задействовать протокол двустороннего обмена открытыми ключами. Прежде всего, необходимо сгенерировать открытый и закрытый ключи шифрования:

```
> ssh-keygen
Initializing random number generator...
Generating p:  .....................................++ (distance 616)
Generating q:  ...................++ (distance 244)
Computing the keys...
Testing the keys...
Key generation complete.
Enter file in which to save the key (/home/craig/.ssh/identity):
Enter passphrase: Pdky&tiaj.
Enter the same passphrase again: Pdky&tiaj.
Your identification has been saved in /home/craig/.ssh/identity.
Your public key is:
1024 35 158564823484025855320901702005057103023948197170850159592181522
craig@horseshoe
Your public key has been saved in /home/craig/.ssh/identity.pub
```

Команда `ssh-keygen` создает ключи. Наберите пароль (или «ключевую фразу») не менее 10 символов длиной. При выборе легко запоминаемого пароля воспользуйтесь приведенными ранее правилами. Если вы забудете ключевую фразу, никто не сможет ее восстановить.

Создав ключи на клиентской системе, скопируйте открытый ключ на сервер. В исходном каталоге клиента ключ хранится в файле *.ssh/identity.pub*. Скопируйте его в подкаталог *.ssh/authorized_keys* своего исходного каталога на сервере. Теперь при обращении к системе посредством `ssh` пользователю предлагается набрать ключевую фразу:

```
> ssh horseshoe
Enter passphrase for RSA key 'craig@horseshoe': Pdky&tiaj.
Last login: Thu Sep 25 17:11:51 2001
```

Из соображений безопасности r-команды следует отключать после установки SSH. Закомментируйте rshd, rlogind, rexcd и rexd в файле *inetd.conf*, чтобы заблокировать входящие обращения к r-командам. Чтобы наверняка использовать SSH для исходящих соединений, замените rlogin и rsh на ssh. С этой целью сохраните копии исходных программ rlogin и rsh в надежном месте, выполните configure повторно – со специальными ключами, приведенными в примере, а затем выполните make install:

```
# whereis rlogin
/usr/bin/rlogin
# whereis rsh
/usr/bin/rsh
# cp /usr/bin/rlogin /usr/lib/rlogin
# cp /usr/bin/rsh /usr/lib/rsh
# ./configure --with-rsh=/usr/bin --program-transform-name='s/ s/r/'
# make install
```

В данном примере предполагается, что исходные программы rlogin и rsh хранятся в каталоге */usr/bin*. Воспользуйтесь соответствующим именем каталога для своей системы.

Заменив rlogin и rsh, вы по-прежнему можете работать с системами, не поддерживающими SSH, однако при этом будете получать уведомление, что соединение не является защищенным:

```
> rlogin cow
Secure connection to cow refused; reverting to insecure method.
Using rsh.  WARNING: Connection will not be encrypted.
Last login: Wed Sep 24 22:15:28 from rodent
```

SSH – прекрасный способ защищенного сообщения систем по сети Интернет. Однако он требует установки пакетов SSH на обеих сообщающихся системах. Разумеется, это не проблема, если вы имеете административный доступ к обеим системам. Но в некоторых случаях необходимо работать с системы, которая управляется не вами. И тогда несомненно важными остаются одноразовые пароли, вроде тех, что предлагает OPIE.

Безопасность приложений

Аутентификация – важный механизм обеспечения безопасности. Но эта мера – лишь одна из многих, позволяющих повысить защищенность компьютера и сети в целом. Большинство успешных атак основано на использовании ошибок в программном обеспечении либо ошибок в настройке программного обеспечения. В этом разделе мы рассмотрим способы улучшения защищенности приложений.

Удаляйте ненужные программы

Любая программа, принимающая внешние соединения, потенциально может стать каналом для атаки злоумышленника. Некоторые специалисты по

безопасности рекомендуют удалять из файла */etc/inetd.conf* настройки всех демонов, которые не являются абсолютно необходимыми. (Работа с файлами *inetd.conf* и */etc/xinetd.conf* описана в главе 5, там же приводятся примеры удаления службы `tftp`.)

Серверным системам требуется присутствие ряда демонов, но большинству рабочих станций демоны не нужны в принципе либо нужны в ограниченных количествах. Удаление демонов в файле *inetd.conf* блокирует только входящие соединения, но не исходящие. Пользователь по-прежнему может обратиться к удаленной системе при помощи `telnet`, даже если демон `telnet` заблокирован в файле *inetd.conf* локальной системы. Простейший подход – удалить все настройки из файла *inetd.conf*, а затем добавлять настройки лишь тех демонов, которые действительно необходимы.

Обновляйте программы

Поставщики систем часто выпускают новые версии сетевых программ с единственной целью – улучшить безопасность сетей. Используйте самые свежие версии сетевых программ от поставщика системы. Отслеживайте предупреждения, связанные с безопасностью и бюллетени CERT, чтобы знать, какие именно программы особенно важно обновить.

Отставание в обновлении программного обеспечения открывает широкие возможности для злоумышленников. Большинство из них пользуются широко известными уязвимостями, а вовсе не занимаются поиском новых. Следите за тем, какие проблемы безопасности обнаружены, чтобы вовремя обновлять свю систему.

Осведомленность о самых свежих «заплатах» для вашей системы не будет лишней – пользуйтесь бюллетенями по безопасности. Свяжитесь с поставщиком/разработчиком системы и выясните, какие услуги по распространению обновлений он предлагает. Позаботьтесь о том, чтобы ваша заинтересованность в безопасности стала для него очевидной.

На рис. 12.2 представлен список программных обновлений, расположенный на веб-сайте Red Hat. Переход по ссылке любого из перечисленных обновлений позволяет получить описание проблемы, а также ссылку на обновление, позволяющее от проблемы избавиться.

Ресурсы, подобные описанному, очень важны для поддержания актуальности программного обеспечения. Чтобы эти ресурсы стали эффективным средством, их следует использовать. Администраторы часто жалуются, что разработчики не спешат устранять проблемы, и, конечно, во многих случаях это справедливо. Однако гораздо более распространенная проблема связана с тем, что системные администраторы не устанавливают доступные обновления. Уделяйте определенное количество времени установке обновлений и делайте это ежемесячно.

Службы обновления ПО, подобные Сети Red Hat (Red Hat Network), потенциально способны облегчить тяжелое бремя обновления программ. Служба

Рис. 12.2. *Обновления от поставщика системы*

обновления ПО позволяет разработчику системы периодически обновлять системные программы по сети. Получит ли такая методика признание – вопрос пока еще открытый. Потенциально службы могут улучшить безопасность и сократить работу по администрированию, но многие администраторы не согласны на утрату полного контроля, связанную с передачей полномочий на обновление сторонней организации.

Наблюдение за безопасностью

Ключевым элементом качественной системы обеспечения безопасности является наблюдение. Высокий уровень защищенности требует регулярного проведения сопутствующих мероприятий, и приведенные выше рекомендации – только начало. Необходимо, помимо прочего, контролировать системы на предмет обнаружения несанкционированной активности, равно как и обнаружения и удаления уязвимостей. Система меняется с течением времени – активные учетные записи становятся пассивными, происходит смена прав доступа для файлов. Необходимо отслеживать и разрешать подобные проблемы по мере их возникновения.

Познайте свою систему

Контроль безопасности в сети – это изучение файлов и журналов отдельных систем сети. Чтобы обнаружить необычную активность, необходимо, прежде всего, знать, какая активность системы нормальна. Какие процессы обычно работают в системе? Какие пользователи? Кто обычно работает с системой после окончания рабочего дня? Следует знать все это и многое другое о системе, чтобы обрести «ощущение» того, как должны обстоять дела. Общеизвестные команды Unix `ps` и `who` помогают узнать, что для вашей системы является нормальной активностью.

Команда `ps` отображает состояние процессов, запущенных в данный момент. Периодическое выполнение `ps` позволяет получить четкую картину процессов, работающих в системе в различное время суток, а также информацию о том, кому принадлежат процессы. Команда Linux `ps -au` и команда Solaris `ps -ef` отображают для каждого процесса сведения о пользователе и команде, создавшей процесс. Этой информации вполне достаточно, чтобы понять, кто с чем работает и в какое время. Заметив нечто необычное, потрудитесь провести расследование. Убедитесь, что понимаете, каким образом используется ваша система.

Команда `who` перечисляет пользователей, работающих в системе в данный момент. Для каждого пользователя отображается имя, устройство терминала, время входа в систему и, если возможно, координаты удаленного узла, с которого инициирован сеанс работы. (Команда `w`, вариант `who`, доступный в некоторых системах, указывает также активный процесс для каждого из пользователей.) Команда `who` позволяет понять, кто обычно работает с системой, а также с каких внешних узлов пользователи проводят сеансы работы. Проводите расследования по любым отклонениям от нормы.

Если любая из этих стандартных проверок дает основания подозревать наличие проблем с безопасностью, изучите систему на предмет присутствия необычных или измененных файлов, файлов, которые должны существовать, но не существуют, а также на предмет необычной активности пользователей. Столь пристальное изучение системы может производиться и при помощи стандартных команд Unix. Команды и файлы, которые мы обсудим, присутствуют в полном составе не во всех системах, однако в каждой операционной системе есть определенные инструменты, помогающие внимательно следить за тем, как используется компьютер.

В поисках неприятностей

Злоумышленники часто оставляют после себя файлы или сценарии, позволяющие им повторно войти в систему либо получить полномочия администратора. Используйте команду `ls -a | grep '^\.'` для поиска файлов с именами, начинающимися с точки (`.`). Злоумышленникам особенно нравятся такие имена, как `.mail`, `.xx`, `...` (три точки), `..` (точка, точка, пробел) или же `..^G` (точка, точка, <Ctrl>+<G>).

Если найден файл с таким или подобным именем, вероятен взлом. (Помните, что в каждом каталоге, за исключением корневого, существует один каталог с именем . и один каталог с именем ..) Изучите содержимое всех подозрительных файлов и следуйте обычной процедуре доклада информации о происшествии.

Следует также изучить ряд ключевых файлов, если возникает подозрение относительно безопасности:

/etc/inetd.conf и /etc/xinetd.conf

Проверьте имена программ, запускаемых настройками файла */etc/inetd.conf*, либо файла */etc/xinetd.conf*, если используется `xinetd`. В частности, убедитесь, что не запускается какой-либо интерпретатор команд (к примеру, */bin/csh*). Убедитесь также, что программы, запускаемые демонами `inetd` и `xinetd`, не были изменены. Файлы */etc/inetd.conf* и */etc/xinetd.conf* не должны быть доступны для записи всем пользователям.

Файлы настройки r-команд

Проверьте файлы */etc/hosts.equiv*, */etc/hosts.lpd*, а также файлы *.rhosts* в домашних каталогах всех пользователей. Убедитесь, что файлы не были изменены неподобающим образом. В частности, обратите особое внимание на записи со знаком сложения (+) и любые записи, в которых фигурируют узлы, не принадлежащие локальной сети. Перечисленные файлы не должны быть доступны для записи всем пользователям. Лучше всего удалите `r`-команды из системы и позаботьтесь о том, чтобы они не устанавливались повторно.

/etc/passwd

Убедитесь, что файл */etc/passwd* не был изменен. Ищите новые имена пользователей и изменения в идентификаторах UID и GID любых учетных записей. Файл */etc/passwd* не должен быть доступен для записи всем пользователям.

Файлы, выполняемые командами cron и at

Проверьте все файлы, выполняемые по командам `cron` и `at`. Ищите новые файлы или необъяснимые изменения в существующих. Временами злоумышленники используют процедуры, выполняемые `cron` и `at`, чтобы повторно получить доступ к системе даже после того, как их доступ заблокирован.

Исполняемые файлы

Проверьте все исполняемые файлы и файлы сценариев на предмет изменений, внесенных злоумышленниками. Исполняемые файлы не должны быть доступны для записи всем пользователям.

Если вы обнаружили или хотя бы заподозрили наличие проблемы, следуйте стандартной процедуре и уведомите о ней пользователей. Это особенно важно, если система подключена к локальной сети. Проблема, возникшая в одной системе сети, может распространиться и на другие системы.

Проверка файлов

Команда `find` является мощным инструментом для обнаружения потенциальных проблем, связанных с безопасностью файловой системы, поскольку позволяет производить всеобъемлющий поиск файлов с определенными правами доступа. Злоумышленники часто оставляют программы, исполняемые с полномочиями администратора, чтобы получить root-доступ к системе. Следующая команда выполняет рекурсивный поиск таких файлов начиная с корневого каталога:

```
# find / -user root -perm -4000 -print
```

Данная команда `find` начинает поиск с корневого каталога (`/`) и находит файлы, принадлежащие пользователю root (`-user root`), с установленным битом setuid (`-perm -4000`). Все найденные соответствия отображаются на терминале (`-print`). Если команда `find` выведет какие-либо имена файлов, внимательно изучите каждый файл и убедитесь, что такие права доступа установлены по необходимости. Как правило, сценарии командного интерпретатора не должны выполняться с полномочиями администратора.

Команду `find` можно использовать для обнаружения других проблем, которые делают систему уязвимой для злоумышленников. В частности, `find` позволяет находить файлы, доступные для записи всем пользователям (`-perm -2`), файлы, исполняемые с полномочиями группы (`-perm -2000`), файлы без владельца (`-nouser -o -nogroup`). Первые две категории файлов необходимо проверять на уместность установленных прав доступа. Обычно файлы, имена которых начинаются с точки (`.`), не должны быть доступны для записи всем пользователям, а групповых полномочий выполнения для сценариев командного интерпретатора рекомендуется избегать.

Процесс сканирования файловой системы можно автоматизировать при помощи программы Tripwire. Версия Tripwire с коммерческой поддержкой доступна по адресу *http://www.tripwiresecurity.com*, а версия для Linux с открытым исходным кодом – по адресу *http://www.tripwire.org*. Данный пакет не только сканирует файловую систему в поисках проблем, но и вычисляет цифровые подписи, позволяющие обнаружить любые изменения файлов.

Проверка активности пользователей

Подозрительная активность пользователей (непривычное время дня или незнакомая удаленная система) может быть признаком того, что злоумышленники пытаются получить доступ к вашей системе. Мы уже познакомились с командой `who`, перечисляющей пользователей, работающих в данный момент в системе. Чтобы выяснить, кто работал в системе ранее, используйте команду `last`.

Команда `last` отображает содержимое файла *wtmp*.[1] Она полезна для создания картины нормальной активности пользователей и вычисления отклонений. Файл *wtmp* хранит историю обращений пользователей к системе –

[1] Данный файл часто хранится в каталоге */usr/adm*, */var/log* или */etc*.

в частности время входа в систему, координаты удаленной системы, а также время завершения сеанса работы.

На рис. 12.3 представлена одна строка вывода команды `last`. Строка содержит поля, определяющие имя пользователя, координаты удаленной системы, с которой подключался пользователь (при необходимости), устройство терминала, день, дату и время начала входа в систему, время завершения сеанса работы (при необходимости) и общую длительность сеанса работы.

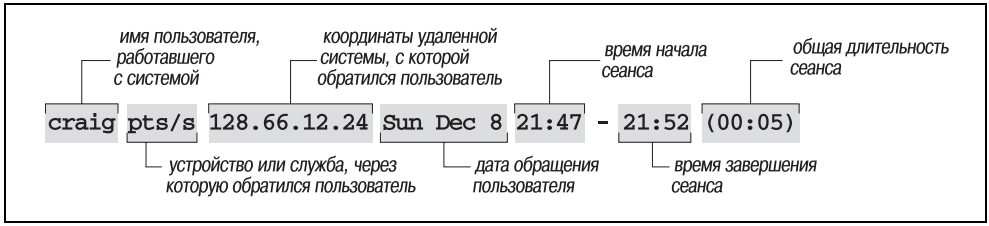


***Рис. 12.3.** Вывод команды last*

Если набрать просто `last`, мы получим весьма объемный вывод, поскольку команда отобразит все записи из файла *wtmp*. Чтобы ограничить вывод, укажите в командной строке имя пользователя или устройство терминала, для которого необходимо получить сведения. Кроме того, можно использовать `grep` для фильтрации вывода `last` по определенным условиям. Например, следующая команда выводит сеансы работы, происходившие в субботу и воскресенье:

```
% last | grep 'S[au]' | more
craig     console     :0            Sun Dec 15 10:33   still logged in
reboot    system boot               Sat Dec 14 18:12
root      console                   Sat Dec 14 18:14
craig     pts/5       jerboas       Sat Dec 14 17:11 - 17:43  (00:32)
craig     pts/2       172.16.12.24  Sun Dec  8 21:47 - 21:52  (00:05)
        .
        .
--More--
```

В следующем примере происходит поиск root-сеансов, проводившихся не с консоли системы. Если неизвестно, кто работал в перечисленных сеансах, у вас должны возникнуть соответствующие подозрения:

```
% last root | grep -v console
root   pts/5   rodent.wrotethebook.com   Tue Oct 29 13:12 - down  (00:03)
root   ftp     crab.wrotethebook.com     Tue Sep 10 16:37 - 16:38  (00:00)
```

Команда `last` является серьезным источником сведений по активности пользователей. Подозрительными являются сеансы работы, происходящие в непривычное время либо с незнакомых удаленных систем. Удаленные root-сеансы должны быть запрещены явным образом. Используйте команду `last` для обнаружения подобных проблем.

Сообщайте о любых проблемах безопасности, которые обнаружите или хотя бы заподозрите. Не стоит стесняться сообщать о проблеме, даже если тревога

может оказаться ложной. Не стоит молчать, потому что «вину» за нарушение безопасности могут возложить на вас. Ваше молчание лишь поможет злоумышленнику.

Автоматизированное наблюдение

Наблюдение за системой вручную требует серьезных затрат времени и связано с ошибками. К счастью, существует ряд инструментов для автоматизации наблюдения. В настоящее время наиболее популярные инструменты перечислены на веб-сайте *http://www.insecure.com*. Одним из таких инструментов является упомянутая выше программа Tripwire. Другие широко применяемые инструменты:

Nessus

Nessus – это сетевой сканер, работающий по модели клиент/сервер. Nessus сканирует целевые системы на предмет обнаружения широкого спектра проблем защищенности.

SATAN

SATAN (Security Auditing Tool for Analyzing Networks) – первый сетевой сканер уязвимостей, получивший широкое распространение. Несмотря на почтенный возраст, по-прежнему популярен и способен обнаружить широкий спектр известных проблем защищенности. SATAN является предком других, также популярных инструментов – SAINT и SARA.

SAINT

Инструмент SAINT (System Administrator's Integrated Network Tool) сканирует системы в поисках широкого спектра уязвимостей. SAINT основан на SATAN.

SARA

SARA (Security Auditor's Research Assistant) – сканер безопасности третьего поколения, основанный на SATAN и SAINT. Выполняет обнаружение широкого спектра известных уязвимостей.

Whisker

Сканер безопасности. Особенно эффективно находит некоторые ошибки в CGI-сценариях, ставящие под угрозу безопасность веб-сайта.

ISS

ISS (Internet Security Scanner) – коммерческий сканер безопасности. Для тех, кто предпочитает коммерческие продукты.

Cybercop

Еще один коммерческий сканер безопасности.

Snort

Snort реализует систему регистрации пакетов, основанную на правилах. Snort пытается обнаружить вторжение и сообщает о проблемах администратору в реальном времени.

PortSentry

PortSentry выявляет сканирование портов и позволяет в реальном времени заблокировать доступ для системы, с которой производится сканирование. Сканирование портов часто предшествует полномасштабной атаке.

Самая большая проблема сканеров и инструментов обнаружения вторжений заключается в том, что они быстро устаревают. Эти инструменты не готовы к отражению новых видов атак. Потому в этой книге нет подробностей работы с подобными сканерами. Список актуален лишь на момент написания книги. На момент прочтения книги появятся новые инструменты обеспечения безопасности и новые версии существующих инструментов. Воспользуйтесь списком лишь в качестве отправной точки для поиска наиболее современных инструментов в среде Web.

Хорошо осведомленные пользователи и администраторы, качественная парольная безопасность, а также скрупулезное наблюдение за системой – вот основы безопасности сети. Но нужно что-то еще. Таким «что-то» является способ управления доступом к системам сети либо управления доступом к данным, передаваемым по сети. В оставшейся части главы мы изучим различные методы разграничения доступа.

Управление доступом

Управление доступом – это механизм ограничения доступа. Маршрутизаторы и узлы, использующие управление доступом, проверяют наличие адреса узла, обратившегося к службе, в *списке управления доступом*. Если список явным образом разрешает удаленному узлу использовать службу, узлу предоставляется доступ к этой службе. Если узел запрещает удаленному узлу доступ к службе, этот узел остается не у дел. Управление доступом существует независимо от стандартных механизмов обеспечения безопасности, добавляя проверку источника запроса к службе и сохраняя все обычные проверки, происходящие на уровне пользователя.

Механизмы управления доступом часто встречаются в терминальных серверах и маршрутизаторах. Например, маршрутизаторы Cisco имеют встроенные возможности для разграничения доступа. Программы управления доступом доступны и для узлов Unix. Мы изучим два таких пакета – `xinetd` и программу TCP wrapper. Начнем с TCP wrapper (`tcpd`), название которой («обертка») происходит от того факта, что она служит оболочкой для сетевой службы. Чтобы обратиться к службе, необходимо сначала проникнуть через эту «обертку».

wrapper

Пакет wrapper выполняет две основные функции: регистрирует обращения к интернет-службам и реализует механизм управления доступом для систем Unix. Регистрация обращений к конкретным сетевым службам – полезная функция наблюдения, особенно при поиске возможных нарушителей. Даже

не имея иной функциональности, пакет wrapper был бы полезен. Однако его настоящая сила заключена в способности управлять доступом к сетевым службам.

Программа wrapper входит в состав многих вариантов систем Linux и Unix. Кроме того, tar-архив wrapper, содержащий исходный текст на языке C и файлы сборки, необходимые для компиляции демона tcpd, доступен на ряде сайтов в сети Интернет.

Если в вашей системе Unix нет программы wrapper, загрузите ее исходный текст, соберите tcpd, а затем установите программу в каталог, где хранятся прочие сетевые демоны. Отредактируйте файл */etc/inetd.conf* и замените путь для всех демонов сетевых служб, доступ к которым необходимо разграничить, путем к демону tcpd. Установка tcpd влияет лишь на одно поле файла */etc/inetd.conf* – шестое, а именно то, которое содержит путь к сетевому демону.

К примеру, запись демона finger в файле */etc/inetd.conf* нашей системы Solaris 8 выглядит так:

```
finger  stream  tcp6  nowait  nobody  /usr/sbin/in.fingerd  in.fingerd
```

Значение шестого поля – */usr/sbin/in.fingerd*. Чтобы контролировать доступ к демону finger, укажите в этом поле значение */usr/sbin/tcpd*, как показано ниже:

```
finger   stream  tcp6  nowait  nobody  /usr/sbin/tcpd   in.fingerd
```

Теперь, получив запрос к демону fingerd, inetd запустит tcpd. tcpd фиксирует обращение к fingerd, сверяется с данными управления доступом, а затем, при наличии разрешения, запускает уже собственно демон finger, который и обрабатывает запрос. Таким образом, tcpd выступает в роли сторожа для других демонов.

Внесите подобные изменения в записи всех служб, для которых необходимо осуществлять управление доступом. Подходящие кандидаты – ftpd, tftpd, telnetd и fingerd. Очевидно, tcpd не может напрямую управлять доступом к службам, которые не находятся под контролем inetd, таким как sendmail и NFS. Однако другие инструменты, например portmapper, используют файлы настройки tcpd для определения собственной политики управления доступом. Таким образом, настройка wrapper может положительно влиять на безопасность даже тех демонов, которые запускаются независимо от inetd.

В большинстве систем Linux применять wrapper еще проще. Не требуется искать и устанавливать программу tcpd – она входит в состав системы. Не требуется даже редактировать файл */etc/inetd.conf*, поскольку шестое поле записей этого файла уже указывает на программу tcpd:

```
finger   stream  tcp  nowait  nobody  /usr/sbin/tcpd   in.fingerd -w
```

Файлы управления доступом tcpd

Информация, которой руководствуется tcpd при управлении доступом, хранится в файлах */etc/hosts.allow* и */etc/hosts.deny*. Назначение файлов одно-

значно определяется их именами. *hosts.allow* содержит список узлов, которым разрешен доступ к службам сети, тогда как *hosts.deny* – список узлов, которым запрещен доступ. Если такие файлы не существуют, `tcpd` разрешает доступ всем узлам и просто регистрирует обращения к службам. Очевидно, если необходимо осуществлять только наблюдение за обращениями, эти два файла можно не создавать.

Если файлы существуют, `tcpd` сначала обращается к файлу *hosts.allow*, а затем к файлу *hosts.deny*. Поиск прекращается, как только найдена запись, соответствующая узлу и службе, о которых идет речь. Таким образом, доступ, разрешенный файлом *hosts.allow*, не может быть заблокирован записью из *hosts.deny*.

Формат записей одинаков для обоих файлов:

```
список_служб: список_узлов [: команда_интерпретатора]
```

Здесь *список_служб* – список сетевых служб, элементы которого разделены запятыми, а точнее говоря – разрешенных (*hosts.allow*) или заблокированных (*hosts.deny*) служб. Каждая служба определяется именем процесса из седьмого поля записи файла */etc/inetd.conf*. Это просто имя, следующее непосредственно за путем к `tcpd` в файле *inetd.conf*. (Описание поля аргументов файла */etc/inetd.conf* дано в главе 5.)

Воспользуемся снова службой `finger` для примера. Мы изменили ее запись в *inetd.conf* на такую:

```
finger   stream  tcp  nowait  nobody  /usr/etc/tcpd   in.fingerd
```

Исходя из этого, в качестве имени службы в файлах *hosts.allow* и *hosts.deny* мы используем `in.fingerd`.

список_узлов, элементы которого разделены запятыми, – это список имен узлов, доменных имен, IP-адресов или номеров сетей. Системам, перечисленным в этом списке, разрешается (*hosts.allow*) или запрещается (*hosts.deny*) доступ к службам, указанным в *списке_служб*. Имя узла или IP-адрес обозначает отдельный узел. Например, *rodent* – имя узла, а 172.16.12.2 – IP-адрес. Оба адреса относятся к совершенно определенному узлу. Доменное имя обозначает все узлы домена; так, *.wrotethebook.com* соответствует *crab.wrotethebook.com*, *rodent.wrotethebook.com*, *horseshoe.wrotethebook.com* и любым другим узлам домена. В списке управления доступом `tcpd` доменные имена всегда начинаются с точки (`.`). Номер сети обозначает все IP-адреса из адресного пространства этой сети. Например, 172.16. соответствует 172.16.12.1, 172.16.12.2, 172.16.5.1, а также любым другим адресам, которые начинаются с 172.16. Адреса сетей в списке управления доступом `tcpd` всегда заканчиваются точкой (`.`).

Ниже приводится полноценная запись *hosts.allow*, разрешающая всем узлам домена *wrotethebook.com* доступ к службам FTP и Telnet:

```
ftpd,telnetd : .wrotethebook.com
```

В записях *hosts.allow* и *hosts.deny* допустимы специальные ключевые слова ALL и LOCAL. Ключевое слово ALL может использоваться в списке служб для указания всех служб сети, а также в списке узлов для указания всех имен узлов и адресов. Второе ключевое слово, LOCAL, может использоваться только в списке узлов. Оно соответствует всем локальным именам узлов. tcpd считает имя узла «локальным», если оно не содержит точек. Таким образом, имя узла *rodent* соответствует указанию LOCAL, а имя *rodent.wrotethebook.com* – нет. Следующая запись относится ко всем службам и всем локальным узлам:

```
ALL : LOCAL
```

Чтобы лучше понять принципы применения tcpd, рассмотрим более сложный пример. Прежде всего, предположим, что необходимо разрешить всем узлам локального домена (*wrotethebook. com*) доступ ко всем службам системы, но запретить доступ к каким-либо службам всем прочим узлам. Создайте в файле */etc/hosts.allow* запись, разрешающую всем узлам локального домена доступ к любым службам:

```
ALL : LOCAL, .wrotethebook.com
```

Ключевое слово ALL в списке служб говорит о том, что правило относится ко всем сетевым службам. Двоеточие (:) отделяет список узлов от списка служб. Ключевое слово LOCAL показывает, что допустимы все локальные имена без доменного расширения, а строка *.wrotethebook.com* – что все имена узлов, дополненные доменным именем *wrotethebook.com*, также допустимы.

Разрешив доступ только тем системам, которые необходимо обслуживать, следует явно запретить доступ всем остальным системам – в файле *hosts.deny*. Чтобы запретить обращения со стороны прочих узлов, внесите такую запись в файл */etc/hosts.deny*:

```
ALL : ALL
```

Все системы, не соответствующие записи из */etc/hosts.allow*, обрабатываются по правилам из */etc/hosts.deny*. В данном случае запись запрещает доступ всем, независимо от службы, о которой идет речь. Помните, даже в присутствии ключевого слова ALL в списке служб правило действует только на службы, работающие под управлением inetd, и только в том случае, если записи этих служб в файле *inetd.conf* соответствующим образом изменены. Автоматическая защита для всех прочих служб не предоставляется.

Синтаксис стандартного файла управления доступом для wrapper может быть чуть более сложным, чем в приведенных примерах. Файл *hosts.allow* может содержать такие записи:

```
imapd, ipopd3 : 172.16.12.
ALL EXCEPT imapd, ipopd3 : ALL
```

Первая запись разрешает доступ к службам IMAP и POP всем узлам, IP-адреса которых начинаются с 172.16.12. Вторая строка разрешает всем узлам доступ ко всем службам, кроме IMAP и POP. Данные записи ограничивают доступ к почтовой службе одной подсетью, в то же время предоставляя все

прочие службы любым узлам. Ключевое слово EXCEPT позволяет исключать элементы из всеохватывающих списков служб. Оно может присутствовать и в списке узлов правила доступа. Рассмотрим такой пример:

```
ALL: .wrotethebook.com EXCEPT public.wrotethebook.com
```

В файле *hosts.allow* такая запись разрешает всем системам домена *wrotethebook.com* доступ ко всем службам. Исключением является узел *public.wrotethebook.com*. Смысл таков, что по какой-то причине узел *public.wrotethebook.com* не является доверенным – возможно, с ним разрешено работать пользователям из других доменов.

И последняя вариация синтаксиса – символ @, сужающий определение служб или узлов. Два примера:

```
in.telnetd@172.16.12.2 : 172.16.12.0/255.255.255.0
in.rshd : KNOWN@robin.wrotethebook.com
```

В списке служб символ @ указывает, что сервер имеет несколько IP-адресов и что правило справедливо лишь для одного из этих адресов. Примерами систем с многими адресами являются многосетевые узлы и маршрутизаторы. Если сервер является маршрутизатором, связывающим локальную сеть с внешними сетями, имеет смысл предоставлять доступ к службам только через интерфейс с локальной сетью, но не через интерфейс с внешним миром. Символ @ позволяет решить именно такую задачу. Если первая строка из примера расположена в файле *hosts.allow*, она разрешает доступ к демону Telnet через сетевой интерфейс 172.16.12.2 для всех клиентов, адреса которых начинаются с 172.16.12.

Назначение @ в списке узлов совсем иное. В списке узлов символ @ указывает, что клиент должен предоставить имя пользователя для прохождения проверки по списку управления доступом. Это означает, что клиент должен работать с демоном identd. Список узлов позволяет проверять конкретное имя пользователя, но чаще всего используется одно из трех ключевых слов:

KNOWN

Результат проверки принимает значение KNOWN, если удаленная система возвращает имя пользователя в ответ на запрос.

UNKNOWN

Результат проверки принимает значение UNKNOWN, если удаленная система не способна ответить на запрос вследствие отсутствия демона identd.

ALL

Данный вариант требует от удаленного узла предоставления имени пользователя. Является эквивалентом KNOWN, но используется реже.

И последнее поле записей – необязательная команда интерпретатора. В случае обнаружения соответствия правилу, с которым связана команда интерпретатора, tcpd регистрирует попытку доступа, разрешает или запрещает доступ к службе, а затем передает команду для выполнения интерпретатору команд.

Создание команды интерпретатора

Команда интерпретатора позволяет осуществлять дополнительную обработку по событию соответствия попытки доступа правилу из списка управления доступом. Во всех практических примерах эта возможность используется в файле *hosts.deny* для сбора дополнительных сведений о злоумышленнике либо для уведомления администратора системы о возможной атаке. Например:

```
ALL : ALL : (safe_finger -l @%h | /usr/sbin/mail -s %d - %h root) &
```

Данное правило из файла *hosts.deny* запрещает доступ ко всем службам всем системам, которым доступ не разрешен явным образом правилами *hosts.allow*. Зарегистрировав и заблокировав попытку доступа, `tcpd` передает команду `safe_finger` командному интерпретатору для выполнения. Все варианты `finger`, включая `safe_finger`, обращаются к удаленному узлу, чтобы определить, кто работает на этом узле. Такие сведения полезны при поиске нарушителя. Результаты выполнения команды `safe_finger` передаются в почтовом сообщении пользователю учетной записи root. Символ `&` в конце строки предписывает интерпретатору выполнять команду в фоновом режиме. Этот символ очень важен. В его отсутствие `tcpd` будет ожидать завершения работы команды, и лишь после вернется к выполнению своих задач.

Программа `safe_finger` входит в состав пакета wrapper. Она особым образом модифицирована и менее уязвима для атак, чем стандартная программа `finger`.

В приведенных примерах использовались некоторые переменные, такие как `%h` и `%d`. Переменные позволяют использовать параметры входящего соединения в работе процесса командного интерпретатора. Существующие переменные перечислены в табл. 12.1.

Таблица 12.1. Переменные, используемые в командах интерпретатора

Переменная	Значение
`%a`	IP-адрес клиента
`%A`	IP-адрес сервера
`%c`	Вся доступная информация по клиенту, включая имя пользователя (когда возможно)
`%d`	Имя процесса демона сетевой службы
`%h`	Имя узла клиента. Если имя узла недоступно, переменная хранит IP-адрес клиента
`%H`	Имя узла сервера
`%n`	Имя узла клиента. Если имя узла недоступно, переменная хранит ключевое слово `UNKNOWN`. Если поиск в DNS возвращает адрес, не совпадающий с IP-адресом клиента, используется ключевое слово `PARANOID`
`%N`	Имя узла сервера
`%p`	Идентификатор процесса демона сетевой службы (PID)

Таблица 12.1 (продолжение)

Переменная	Значение
%s	Вся доступная информация по серверу, включая имя пользователя (когда возможно)
%u	Имя пользователя клиента либо ключевое слово UNKNOWN, когда имя пользователя недоступно
%%	Символ процента (%)

Из табл. 12.1 видно, что %h – это имя удаленного узла, а %d – процесс демона, к которому происходит обращение. Вернемся к примеру команды интерпретатора. Предположим, что попытка обращения к in.rshd исходила от узла *foo.bar.org*. Интерпретатору передается команда:

```
safe_finger -l @foo.bar.org |
   /usr/sbin/mail -s in.rshd-foo.bar.org root
```

Стандартный язык управления доступом wrapper является полноценным языком настройки и позволяет решать любые разумные задачи. Несмотря на это существует расширенная версия этого языка.

Необязательные расширения языка управления доступом

Если пакет wrapper компилировался в присутствии PROCESS_OPTIONS в make-файле, синтаксис языка управления доступом изменяется и расширяется. Включение PROCESS_OPTIONS изменяет число полей в правилах. Новый синтаксис выглядит следующим образом:

```
список_служб : список_узлов : параметр : параметр ...
```

Списки узлов и служб определены так же, как в исходном варианте синтаксиса. Новым элементом являются параметры, равно как и факт, что в каждом правиле их может быть несколько. Существуют следующие параметры:

`allow`

Разрешает доступ к службе. Параметр указывается в конце правила.

`deny`

Запрещает доступ к службе. Параметр указывается в конце правила.

`spawn` *команда*

Выполняет указанную команду интерпретатора в порожденном процессе.

`twist` *команда*

Выполняет указанную команду интерпретатора вместо службы, к которой произошло обращение.

`keepalive`

Посылает сообщения keepalive удаленному узлу. Если узел не отвечает, соединение закрывается.

`linger` *секунд*

Указывает интервал времени, в течение которого следует пытаться доставить данные после того, как сервер закрыл соединение.

`rfc931 [` *интервал* `]`

Использует протокол IDENT для поиска имени пользователя на удаленном узле. *интервал* (*timeout*) определяет число секунд ожидания ответа от удаленного узла.

`banners` *путь*

Передает содержимое файла (сообщения) удаленной системе. *путь* – это имя каталога, содержащего файлы с «шапками» (banners). Отображается тот файл, имя которого совпадает с именем процесса сетевого демона.

`nice [` *число* `]`

Устанавливает значение приоритета выполнения (`nice`) для процесса сетевой службы. Значение по умолчанию – 10.

`umask` *маска*

Устанавливает значение `umask` для файлов, задействованных в работе процесса сетевой службы.

`user` *пользователь*`[.`*группа*`]`

Определяет идентификатор пользователя и идентификатор группы, с правами которых выполняется процесс сетевой службы. Данный параметр имеет более высокий приоритет, чем определения в файле *inetd.conf*.

`setenv` *переменная значение*

Устанавливает значение переменной среды выполнения процесса.

Чтобы проиллюстрировать отличия нового синтаксиса, рассмотрим ряд примеров, основанных на приводившихся ранее. В случае расширенного синтаксиса *hosts.allow* может содержать такие записи:

```
ALL : LOCAL, .wrotethebook.com : ALLOW
in.ftpd,in.telnetd : eds.oreilly.com : ALLOW
ALL : ALL : DENY
```

Новый синтаксис позволяет сократить количество файлов настройки до одного. Параметры `ALLOW` и `DENY` позволяют хранить все правила в одном файле. Первая строка разрешает доступ ко всем службам для всех локальных узлов и всех узлов домена *wrotethebook.com*. Вторая строка разрешает доступ к службам FTP и Telnet удаленному узлу *eds.oreilly.com*. Третья строка идентична правилу `ALL : ALL` в файле *hosts.deny*; она запрещает всем прочим узлам доступ ко всем службам. Параметры `ALLOW` и `DENY` позволяют переписать команду

```
ALL: .wrotethebook.com EXCEPT public.wrotethebook.com
```

следующим образом:

```
ALL: .wrotethebook.com : ALLOW
ALL: public.wrotethebook.com : DENY
```

Пример, включающий команду интерпретатора, в новом варианте выглядит почти так же:

```
in.rshd : ALL: spawn (safe_finger -l @%h | /usr/sbin/mail -s %d - %h root) & : DENY
```

Более интересная вариация на тему команд интерпретатора связана с параметром `twist`. Вместо того чтобы передавать команду интерпретатору для выполнения, команда `twist` выполняет для удаленного пользователя программу, но совсем не искомую. Например:

```
in.ftpd : ALL: twist /bin/echo 421 FTP not allowed from %h : DENY
```

В данном случае, когда удаленный пользователь обращается к демону FTP, вместо FTP запускается `echo`. Программа `echo` передает сообщение удаленной системе и разрывает соединение.

Расширенный синтаксис wrapper применяется редко, поскольку все задачи можно решить посредством традиционного синтаксиса. Следует разбираться в расширенном синтаксисе на случай встречи с ним, однако маловероятно, что вы испытаете необходимость его использовать. Альтернативой wrapper, которая вам встретится, будет `xinetd`. Этот демон заменяет `inetd` и позволяет управлять доступом. Основы работы с `xinetd` рассмотрены в главе 5. Здесь мы сосредоточим внимание на функциях управления доступом этого демона.

Управление доступом в xinetd

Как уже говорилось в главе 5, большая часть информации в файле *xinetd.conf* представляет собой те же значения, что и хранимые в файле *inetd.conf*. Дополнительная функциональность `xinetd` аналогична функциональности пакета wrapper. `xinetd` читает файлы */etc/hosts.allow* и */etc/hosts.deny* и реализует управление доступом на основе информации на основе этих файлов. Кроме того, в `xinetd` существуют собственные механизмы управления доступом и регистрации обращений. Если в вашей системе применяется `xinetd`, имеет смысл создать файлы *hosts.allow* и *hosts.deny*, чтобы повысить защищенность служб, работающих с этими файлами, таких как `portmapper`, а также задействовать существующие в `xinetd` функции безопасности, поскольку они реализуют более совершенное управление доступом.

Регистрация обращений в `xinetd` реализуется параметрами `log_on_success` и `log_on_failure`. Используйте эти параметры для подстройки стандартного формата записей журнала, создаваемых для успешных и неудачных попыток обращения к службам. `log_on_success` и `log_on_failure` имеют следующие ключи:

USERID

Регистрирует идентификатор удаленного пользователя. Параметр USERID может регистрироваться как для успешных, так и для неудачных попыток обращения.

HOST

Регистрирует адрес удаленного узла. Подобно USERID, HOST может регистрироваться как для успешных, так и для неудачных попыток.

PID

Регистрирует идентификатор серверного процесса, созданного для обработки соединения. Параметр PID применим только для `log_on_success`.

DURATION

Регистрирует длительность работы серверного процесса, обработавшего данное соединение. Параметр DURATION применим только для `log_on_success`.

EXIT

Регистрирует состояние сервера на момент завершения соединения. Параметр EXIT применим только для `log_on_success`.

ATTEMPT

Регистрирует неудавшиеся попытки соединения. Параметр ATTEMPT применим только для `log_on_failure`.

RECORD

Регистрирует информацию о соединении, полученную от удаленного сервера. Параметр RECORD применим только для `log_on_failure`.

Помимо регистрации обращений `xinetd` имеет три параметра для управления доступом. Используйте эти параметры для настройки `xinetd` на прием соединений от определенных узлов в соответствии с файлом *hosts.allow*, блокировку соединений от определенных узлов в соответствии с файлом *hosts.deny*, а также на прием соединений только в определенные интервалы времени. Вот эти три параметра:

`only_from`

Обозначает узлы, которым разрешено обращаться к службе. Узлы обозначаются при помощи:

- Числовых адресов. Так, адрес 172.16.12.5 обозначает конкретный узел, а номер 129.6.0.0 – все узлы, адреса которых начинаются с 129.6. Адрес 0.0.0.0 обозначает все адреса.
- Наборов адресов. Например, 172.16.12.{3,6,8,23} обозначает четыре различных узла: 172.16.12.3, 172.16.12.6, 172.16.12.8 и 172.16.12.23.
- Имен сетей. Имени сети должно соответствовать определение в файле */etc/networks*.
- Канонических имен узлов. Представленный удаленной системой IP-адрес должен отображаться в указанное имя узла.
- Доменных имен. Имя узла, полученное обратным отображением, должно принадлежать указанному домену. Например, значение *.wrotethebook.com* требует, чтобы узел входил в домен *wrotethebook.com*. Обратите внимание, что доменные имена должны начинаться с точки.
- IP-адресов в сочетании с адресными масками. Так, записи 172.16.12.128/25 соответствуют все адреса с 172.16.12.128 по 172.16.12.255.

no_access

Обозначает узлы, которым запрещен доступ к службе. Узлы обозначаются способами, описанными выше для атрибута only_from.

access_times

Определяет время суток, когда служба доступна, в формате *час*:*мин*-*час*:*мин*. Используется 24-часовая шкала времени. Часы пронумерованы от 0 до 23, а минуты – от 0 до 59.

В отсутствие атрибутов only_from и no_access разрешается доступ всем узлам. Если существуют оба атрибута, используется наиболее точное соответствие. Рассмотрим пример:

```
no_access           = 172.16.12.250
only_from           = 172.16.12.0
```

Здесь команда only_from разрешает доступ к службе всем системам сети 172.16.12.0. Команда no_access запрещает доступ одной конкретной системе. Порядок следования команд no_access и only_from не имеет значения, результат всегда одинаков, поскольку приоритет имеет наиболее точное соответствие.

Ниже приводится пример записи для POP3 из файла *xinetd.conf*:

```
# default: on
# описание: Служба  POP3 позволяет удаленным пользователям обращаться к своей почте\
#           посредством клиента POP3, такого как  Netscape Communicator, mutt \
#           или fetchmail.

service login
{
        socket_type             = stream
        wait                    = no
        user                    = root
        log_on_success          += USERID
        log_on_failure          += USERID
        only_from               = 172.16.12.0
        no_access               = 172.16.12.231
        server                  = /usr/sbin/ipop3d
}
```

В данном фрагменте команда only_from разрешает доступ всем системам сети 172.16.12.0, то есть локальной сети, и запрещает доступ всем остальным системам. Кроме того, существует одна система подсети 17.16.12.0 (узел 172.16.12.231), которая не является настолько доверенной, чтобы получить POP-доступ. Команда no_access запрещает доступ всем пользователям системы 172.16.12.231.

Помните, что wrapper и xinetd позволяют лишь управлять доступом к службам, но не способны ограничивать доступ к данным системы или данным, передаваемым по сети. С этой целью следует использовать шифрование.

Шифрование

Шифрование – это способ ограничения доступа к данным, передаваемым по сети. Шифрование связано с кодированием данных, переводом их в представление, которое может быть прочитано только системами, обладающими «ключом» к схеме кодирования. Исходный текст, известный как «открытый текст», шифруется при помощи ключа шифрования по схеме, реализованной на программном или аппаратном уровне. Результатом становится закодированный текст (cipher). Чтобы воссоздать открытый текст, закодированный текст необходимо расшифровать по той же схеме шифрования и при помощи соответствующего ключа.

Когда люди задумываются о безопасности, первое, что приходит им на ум, – шифры. Происходит это в основном благодаря шпионским романам и фильмам о второй мировой войне. Однако шифрование далеко не всегда было применимо к сетевой безопасности. Традиционно шифрование данных для передачи по сети требовало применения обеими сторонами одного и того же ключа шифрования, известного в качестве совместно используемого *секретного*, или *закрытого, ключа*. Сквозное шифрование можно относительно эффективно использовать только в том случае, если вы контролируете обе стороны канала и можете гарантировать, что всем участникам обмена доступен один и тот же ключ шифрования. По этой причине шифрование использовалось преимущественно для передачи данных между точками сети, находящимися под управлением одной инстанции: в военных сетях, частных сетях, отдельных системах, либо в случаях, когда стороны обмена могли прийти к соглашению относительно метода шифрования и ключей. Шифрование, требующее предварительного согласования и совместного использования секретного ключа, носит название *симметричного шифрования*.

Шифрование с открытым ключом – это технология, которая делает шифрование важным механизмом обеспечения безопасности в глобальных сетях – таких как Интернет. Например, веб-сервер электронной коммерции и броузер любого заказчика могут обмениваться шифрованными данными, поскольку обе стороны используют шифрование с открытым ключом. Системы открытых ключей кодируют текст при помощи ключа, который широко известен и доступен всем, но раскодировать зашифрованный текст можно, только зная закрытый ключ. То есть Дэн может найти открытый ключ Кристины в специальной базе данных и с его помощью зашифровать адресованное ей сообщение так, что его не сможет прочесть никакой другой получатель. Все пользователи сети Интернет имеют доступ к открытому ключу, но только Кристина может расшифровать сообщение – при помощи своего закрытого ключа. Такая передача зашифрованных данных позволяет Кристине сохранить свой закрытый ключ в тайне.

Кроме того, сообщения, зашифрованные закрытым ключом, могут быть расшифрованы только при помощи открытого ключа. Это означает, что открытый ключ может использоваться для проверки подлинности источника сообщения, поскольку лишь один источник должен иметь доступ к закрытому ключу. Поскольку в шифровании с открытым ключом используются раз-

личные ключи для кодирования и декодирования сообщений, такое шифрование носит называние *асимметричного*.

С асимметричным шифрованием связана одна проблема – оно требовательно к вычислительным мощностям и непроизводительно, если сравнивать с симметричными методами. По этой причине шифрование используется лишь для небольших фрагментов данных. Шифрование с открытым ключом используется как для шифрования, так и для проверки подлинности в момент создания шифруемого соединения. В процессе «рукопожатия» стороны обмениваются общим закрытым ключом, подвергая его шифрованию открытыми ключами. В последующем обмене данными используется симметричное шифрование на основе общего ключа.

Вторая проблема шифрования с открытыми ключами – в глобальной сети требуется общепризнанная, надежная инфраструктура распределения открытых ключей и сохранения их целостности. Чтобы послать сообщение Кристине, Дэн прежде всего обзавелся ее открытым ключом. Откуда он получил этот ключ? Вероятно, источником стал частный обмен открытыми ключами либо сеть, авторизованная специальным органом управления. Когда число участников ограничено, открытые ключи могут распространяться частным образом – точно так же, как это раньше делалось для закрытых ключей. Однако в приложениях глобальной сети, где личности участников обмена не известны заранее, такой подход не срабатывает. В этом случае открытый ключ приходит по сети и сертифицируется третьей стороной, известной как *сертифицирующий орган* (*certificate authority*, CA). CA передает открытый ключ в сообщении, называемом *сертификатом*. Сертификат содержит открытый ключ, название организации, которой он принадлежит, а также даты начала и окончания действия ключа. Сообщение сертификата подписано закрытым ключом органа CA. Таким образом, если подлинность сертификата подтверждается при помощи открытого ключа CA, получатель сообщения может быть уверен, что сертификат поступил от доверенного издателя. Открытые ключи CA широко известны и распространены. Так, разработчики броузеров включают открытые ключи многих издателей сертификатов в комплект поставки.

В примерах следующего раздела используется симметричное шифрование. Оно требует применения одинаковых методов шифрования обеими сторонами, а также использования одного и того же секретного ключа как для кодирования, так и для раскодирования сообщений. Симметричное шифрование не задействует открытые ключи, цифровые подписи и не имеет повсеместно принятой инфраструктуры, но при этом имеет ограниченный потенциал.

В каких случаях полезно симметричное шифрование?

Прежде чем начать использовать шифрование, определитесь, почему вам необходимо шифровать данные, следует ли защищать данные посредством шифрования, а также следует ли вообще хранить данные в компьютерной системе, подключенной к сети.

Вот несколько разумных причин для шифрования данных:

- Предотвращение случайного просмотра файлов с секретными данными
- Предотвращение случайного раскрытия секретных данных
- Предотвращение просмотра файлов секретных данных привилегированными пользователями (например, администраторами систем)
- Усложнение задачи злоумышленникам, которые попытаются выполнить поиск по файлам системы

Существуют различные инструменты, предназначенные для шифрования файлов данных, и многие из таких пакетов являются платным ПО. Две файловые системы с открытым исходным текстом реализуют автоматическое шифрование файлов – Cryptographic File System (CFS) и Practical Privacy Disk Driver (PPDD).[1] Кроме того, в состав Solaris и Linux входит пара инструментов для шифрования файлов.

В Solaris присутствует старая команда Unix – `crypt`, которая легка в применении, но имеет небольшую ценность. Шифрование `crypt` легко «сломать». В лучшем случае `crypt` позволяет защитить файлы от случайного просмотра.

Возраст программы `crypt` и тот факт, что прочие, более совершенные и более современные инструменты симметричного шифрования не включаются в состав операционных систем, позволяют предположить, что спрос на такие инструменты невелик. Шифрование с открытым ключом более универсально и может использоваться в более широком диапазоне приложений. Инструмент шифрования файлов, включенный в состав Linux, является как раз инструментом асимметричного шифрования.

Инструменты шифрования с открытым ключом

Именно к шифрованию с открытым ключом проявляется основное внимание общественности. Наиболее популярные инструменты шифрования в Unix, `ssh` и SSL, работают на основе открытых ключей. Даже в задачах, связанных с шифрованием локально хранимых файлов, системы на основе открытых ключей пользуются большим спросом, поскольку не требуют распространения закрытых ключей.

В систему Linux часто включается GNU Privacy Guard (`gpg`), который, подобно широко известному PGP[2], может использоваться для шифрования файлов или почтовых сообщений. Кроме того, данный инструмент содержит средства для работы с цифровыми подписями, которые позволяют проверить подлинность сообщений электронной почты. В следующем примере

---

[1] Установка, настройка и применение обеих файловых систем описаны в книге «Linux Security» Рамона Хонтанона (Ramon Hontanon), Sybex.

[2] В книге Симсона Гарфинкела (Simson Garfinkel) «PGP: Pretty Good Privacy» (O'Reilly) подробно описывается PGP – программа шифрования для файлов и электронной почты.

при помощи gpg выполняется шифрование и декодирование файла. Прежде всего, мы создаем ключи при помощи параметра --gen-key:

```
$ gpg --gen-key
gpg (GnuPG) 1.0.4; Copyright (C) 2000 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.
gpg: Warning: using insecure memory!
gpg: /home/craig/.gnupg/secring.gpg: keyring created
gpg: /home/craig/.gnupg/pubring.gpg: keyring created
Please select what kind of key you want:
   (1) DSA and ElGamal (default)
   (2) DSA (sign only)
   (4) ElGamal (sign and encrypt)
Your selection? 1
DSA keypair will have 1024 bits.
About to generate a new ELG-E keypair.
              minimum keysize is  768 bits
              default keysize is 1024 bits
    highest suggested keysize is 2048 bits
What keysize do you want? (1024) 1024
Requested keysize is 1024 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct (y/n)? y
A User-ID identifies your key; the software constructs the user id
from Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"
Real name: Craig Hunt
Email address: craig.hunt@wrotethebook.com
Comment:
You selected this USER-ID:
    "Craig Hunt <craig.hunt@wrotethebook.com>"
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
You need a Passphrase to protect your secret key.
Type the passphrase: Fateful lightening
Repeat: Fateful lightening
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
+++++.+++++.+++++.++++++++++++++++++++.+++++.++++++++++++++++++++++++.
++++++++++.++++++++++++++++++++.++++++++++++++++++++++++++++++++++>.+++++.......
.....................+++++^^^
public and secret key created and signed.
```

Использование опции `--gen-key` связано с необходимостью ответить на ряд вопросов. Вопросы весьма просты, а создание исходных ключей следует выполнить всего один раз. Прежде всего, `gpg` спрашивает, какой тип ключа необходимо сгенерировать. А именно – с какой целью будут использоваться ключи: для работы с цифровыми подписями, для шифрования либо для того и другого. Выберите вариант (1), то есть вариант по умолчанию. Программа создаст оба типа ключей, и вы сможете встретить любую задачу шифрования во всеоружии. Затем программа запрашивает длину ключа: чем длиннее ключ, тем труднее его сгенерировать и подобрать. По умолчанию ключ имеет длину 1024 бита, которой будет более чем достаточно для любого реального применения `gpg`. `gpg` запрашивает ваше имя, адрес электронной почты и необязательный комментарий. Данная информация используется для поиска ваших ключей в базах данных. Наконец, программа запрашивает парольную фразу, позволяющую получать доступ к вашему секретному ключу.

Для хранения закрытых и открытых ключей в `gpg` используется пара баз данных. Эти базы данных в `gpg` называются «кольцами для ключей». Файл базы данных закрытых ключей называется *secring.gpg*, а файл базы данных открытых ключей – *pubring.gpg*. Оба ключа используются при шифровании и последующем декодировании файла. Следующий пример иллюстрирует процесс шифрования:

```
$ cat test.txt
This is a test file.
$ gpg --recipient craig.hunt@wrotethebook.com --encrypt test.txt
gpg: Warning: using insecure memory!
$ cat test.txt.gpg
` Y →  °u ï 2J  ë Y;O¬[  o# LHu¨ e `EDO Si P -EU ® ëïO!7 n6 IOei
cEo$2[9oAIi@E¬wY $2'6 $B«'6yk_¬o1NOΦBi giy [CyoU6®&V'g TWn2qO⋆Bx
nOnmT5ª Y[uuYEA2 ,heq¨?°i 'J\Po o/o?"ËTTeBaaUU»°5oNB= a}/O@Nemstu
$ rm test.txt
```

Команда `cat` показывает, что мы создали простой текстовый файл *test.txt* и желаем его зашифровать. Действие ключа командной строки `--encrypt` вполне очевидно, однако не столь очевидно назначение аргумента `--recipient`. База данных *pubring.gpg* может содержать большое число открытых ключей. Аргумент `--recipient` указывает на открытый ключ, который следует использовать для шифрования файла. Слово «recipient» (получатель) использовано потому, что `gpg` часто применяется для шифрования почты, а в таких случаях используется открытый ключ адресата (получателя) сообщения. По этой же причине обычной практикой является идентификация ключа по почтовому адресу, который был указан в момент создания этого ключа.

`gpg` создает файл с зашифрованным текстом и дает ему имя исходного файла, добавляя суффикс *.gpg*. Команда `cat` для зашифрованного файла показывает нам, что файл стал совершенно нечитаемым. Убедившись, что зашифрованный файл создан, мы удаляем файл с открытым текстом. Разумеется, нет особой пользы в создании зашифрованного файла, если исходный файл продолжает существовать и может быть прочитан всеми желающими!

Чтобы прочесть зашифрованный текст, его необходимо раскодировать. В следующем примере для решения задачи используется ключ `--decrypt` командной строки `gpg`. Расшифруем файл *test.txt.gpg*:

```
$ gpg --output test.txt --decrypt test.txt.gpg
gpg: Warning: using insecure memory!
You need a passphrase to unlock the secret key for
user: "Craig Hunt <craig.hunt@wrotethebook.com>"
1024-bit ELG-E key, ID D99991BA, created 2001-09-18 (main key ID 9BE3B5AD)
Enter passphrase: Fateful lightening
$ cat test.txt
This is a test file.
```

Ключ `--output` указывает `gpg`, куда записывать расшифрованный открытый текст. В примере мы записываем результат в файл *test.txt*. Команда `cat` для *test.txt* показывает, что файл читаем, и содержит исходный текст.

Приведенные примеры работы `gpg` схожи с примерами для `ssh`, расположенными выше в данной главе, и примерами для `openssl` в главе 11. Все эти программы предоставляют инструменты для создания открытых и закрытых ключей, используемых для решения определенных задач. `gpg` защищает файлы и сообщения электронной почты, `ssh` – терминальные соединения, `openssl` – веб-трафик. При этом SSL может применяться для защиты передаваемых данных в широком спектре приложений.

stunnel

Программа `stunnel` шифрует – при помощи SSL – трафик демонов, не обладающих способностями к шифрованию. `stunnel` делает преимущества шифрования с открытым ключом доступными широкому спектру сетевых приложений. `stunnel` входит в состав пакета OpenSSL и устанавливается вместе с OpenSSL.[1]

Подобно прочим приложениям, работающим с SSL, `stunnel` для нормальной работы требует наличия сертификата. Простейший способ создать сертификат `stunnel`: перейти в каталог сертификатов SSL и выполнить команду `make`, как показано в следующем примере:

```
# cd /usr/share/ssl/certs
# make stunnel.pem
umask 77 ; \
PEM1=`/bin/mktemp /tmp/openssl.XXXXXX` ; \
PEM2=`/bin/mktemp /tmp/openssl.XXXXXX` ; \
/usr/bin/openssl req -newkey rsa:1024 -keyout $PEM1 -nodes -x509 -days 365 -out $PEM2
; \
cat $PEM1 >  stunnel.pem ; \
echo ""    >> stunnel.pem ; \
cat $PEM2 >> stunnel.pem ; \
rm -f $PEM1 $PEM2
```

[1] Система OpenSSL описана в главе 11.

```
Using configuration from /usr/share/ssl/openssl.cnf
Generating a 1024 bit RSA private key
....++++++
.........++++++
writing new private key to '/tmp/openssl.3VVjex'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request. What you are about to enter is what is
called a Distinguished Name or a DN. There are quite a few fields but you
can leave some blank. If you enter '.', the field will be left blank. For
some fields there will be a default value.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Maryland
Locality Name (eg, city) []:Gaithersburg
Organization Name (eg, company) [Internet Widgits Ltd]:WroteTheBook.com
Organizational Unit Name (eg, section) []:Books
Common Name (eg, your name or your server's hostname) []:Craig Hunt
Email Address []:craig.hunt@wrotethebook.com
```

По умолчанию при установке `openssl` создается каталог */usr/share/ssl/certs*, предназначенный для хранения сертификатов, и `stunnel` по умолчанию ищет в этом каталоге сертификат с именем *stunnel.pem*.[1] Как и для всех других новых сертификатов, `openssl` необходимо предоставить информацию, однозначным образом определяющую сертификат.

После создания сертификата программа `stunnel` готова к применению. Отличными примерами служб, работающих через защищенный канал `stunnel`, могут послужить POP и IMAP. Главной причиной совместного использования POP, IMAP и `stunnel` является защита паролей пользователей, передаваемых в ходе сеансов POP и IMAP. `stunnel` шифрует все: регистрацию на сервере и передаваемые почтовые сообщения. Последнее защищает содержимое сообщения от перехвата в процессе передачи от сервера клиента, хотя с точки зрения системного администратора важно защитить именно пароль.

Чтобы защищенный обмен по POP и IMAP заработал, обе стороны должны уметь передавать данные по каналам SSL. Однако это не всегда так. У некоторых клиентов нет `stunnel`; у других вообще нет SSL. По этой причине серверы обычно предоставляют стандартный доступ к службам POP и IMAP через широко известные порты, а SSL-доступ к тем же службам – через другие порты. При работе через `stunnel` служба POP называется *pops* и использует порт TCP 995; IMAP называется *imaps* и получает порт TCP 993. *pops* и *imaps* – это не специальные протоколы, а просто имена служб из файла */etc/services*, связанные с номерами портов 995 и 993. Следующая команда загрузочного сценария организует доступ к POP через туннель SSL и порт 995:

```
stunnel -d 995 -l /usr/sbin/ipop3d -- ipop3d
```

[1] Путь к каталогу сертификатов можно явно указать при помощи ключа `-p` командной строки `stunnel`.

Как вариант, stunnel может выполняться демоном inetd. Чтобы это происходило, следует создать соответствующую запись в файле *inetd.conf*. Например, следующая запись организует доступ к POP через туннель SSL – по запросам клиентов:

```
pops stream tcp nowait root /usr/sbin/stunnel -l /usr/sbin/ipop3d -- ipop3d
```

Если используется демон xinetd, работу stunnel следует настраивать в файле *xinetd.conf*. Следующая запись xinetd выполняет настройку *imaps*:

```
service imaps
{
        socket_type             = stream
        wait                    = no
        user                    = root
        server                  = /usr/sbin/stunnel
        server_args             = -l /usr/sbin/imapd -- imapd
        log_on_failure          += USERID
}
```

Приложение stunnel никоим образом не связано именно с POP или IMAP, оно может применяться для обеспечения безопасности многих демонов. Если необходимо защитить демон, выполняемый inetd или xinetd, команда stunnel размещается в файле *inetd.conf* или *xinetd.conf*, как и должно быть. Если демон стартует по команде в загрузочном файле, там же должна размещаться и команда stunnel.

Несмотря на серьезные возможности инструментов вроде stunnel и ssh, шифрование не является заменой для качественной компьютерной безопасности. Оно способно защитить чувствительную или частную информацию от просмотра, но никогда не должно быть единственным средством защиты важной информации. Системы шифрования подвержены взломам, а зашифрованные данные – удалению и повреждениям, точно так же, как любые другие данные. Пусть шифрование не смущает вас ложным ощущением защищенности. Некоторая информация настолько чувствительна и важна, что ее не следует хранить на подключенных к сети компьютерах даже в закодированном виде. Шифрование – лишь небольшой фрагмент полноценной системы безопасности.

Брандмауэры

Брандмауэр (firewall, сетевой экран) является неотъемлемой составляющей системы безопасности сети. Термин «firewall»[1] подразумевает защиту от опасности: как противопожарная перегородка в автомобиле защищает пассажиров от двигателя, так сетевой экран защищает сеть от внешнего мира.

[1] Экран или кожух, предотвращающий распространение огня. Термин применяется в строительстве, автомобильной промышленности, а теперь и в компьютерных технологиях. – *Примеч. перев.*

Компьютер-брандмауэр позволяет жестко управлять взаимодействием систем вашей сети и внешнего мира.

Идея брандмауэра довольно проста. Это некая заслонка, фильтрующая весь трафик, которым обмениваются защищенная и внешняя сети. На практике, это обычно заслонка между сетью предприятия и Интернетом. Создание централизованного фильтра упрощает задачу наблюдения за трафиком и позволяет сконцентрировать усилия по обеспечению безопасности в одной точке.

Существуют различные варианты реализации брандмауэров. Более того, различных типов брандмауэров существует так много, что термин практически потерял свою значимость. Если кто-то говорит, что защищен брандмауэром, нельзя с уверенностью сказать, что именно этот человек имеет в виду. Рассмотрение различных типов архитектур брандмауэров требует целой книги – например «Создание защиты в Интернете» (Building Internet Firewalls) от O'Reilly & Associates.[1] Здесь мы рассмотрим архитектуру экранированной подсети (вероятно, наиболее распространенную архитектуру брандмауэров) и архитектуру многосетевого узла – по сути дела, представляющую экран в себе.

Наиболее распространенная архитектура брандмауэра включает по меньшей мере четыре аппаратных элемента: внешний маршрутизатор, защищенный сервер (известный как *узел-бастион*), *сетевой периметр* и внутренний маршрутизатор. Каждый аппаратный элемент обеспечивает определенный аспект функционирования общей схемы защиты. Архитектура представлена на рис. 12.4.

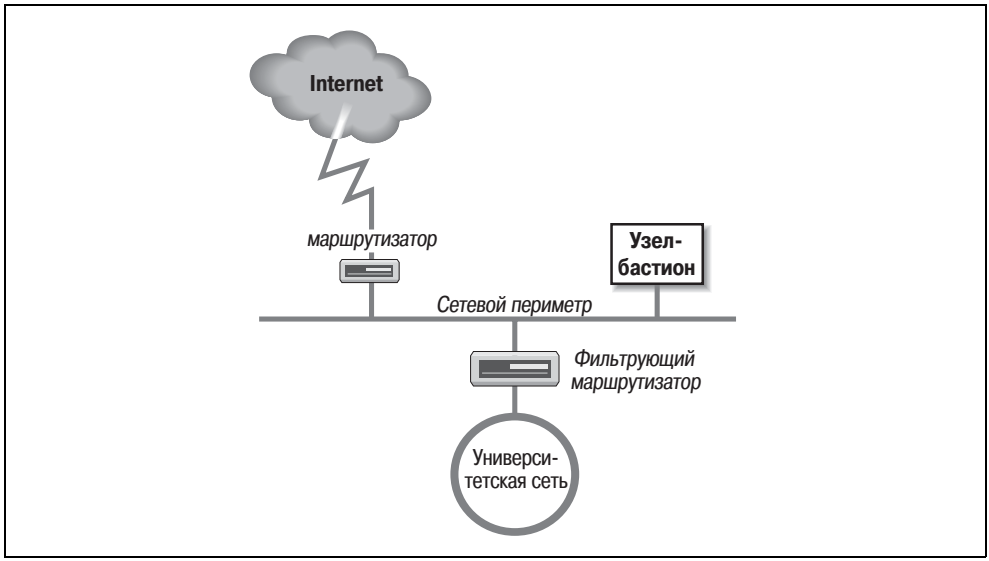


***Рис. 12.4.** Экранирование подсети*

[1] Э. Цвики, С. Купер, Б. Чапмен «Создание защиты в Интернете». – Пер. с англ. – СПб: Символ-Плюс, 2002.

Внешний маршрутизатор – единственный канал между сетью предприятия и внешним миром. Он осуществляет минимальное управление доступом. Маршрутизатор проверяет, что адреса пакетов, приходящих из внешнего мира, не принадлежат внутренней сети. Если номер нашей сети 172.16, внешний маршрутизатор удаляет все пришедшие через внешний интерфейс пакеты, адрес которых начинается с 172.16. Пакеты с такими адресами источников должны приходить на маршрутизатор только через внутренний интерфейс. Специалисты по безопасности называют такой тип управления доступом *фильтрацией пакетов*.

Большую часть работы по управлению доступом выполняет внутренний маршрутизатор. Он фильтрует пакеты не только на основе адреса, но также на основе протокола и номеров портов, что позволяет контролировать службы, доступные как внутри сети, так и вне ее. Какие службы блокирует данный маршрутизатор, решает администратор системы. Если вы планируете использовать брандмауэр, доступ к службам должен быть регламентирован в руководящем документе по безопасности. Практически любая служба может оказаться угрозой безопасности системы. Подобные угрозы следует оценивать в свете общих требований к безопасности. Службы, предназначенные только для внутренних пользователей (NIS, NFS, X-Windows и др.), блокируются практически всегда. Службы, разрешающие запись на внутренние системы (Telnet, FTP, SMTP и т. д.), обычно блокируются. Службы, предоставляющие сведения о внутренних системах (DNS, `fingerd` и т. д.), обычно блокируются. Доступных служб практически не остается! И здесь в игру вступают узел-бастион и сетевой периметр.

Узел-бастион – это защищенный сервер. Он является точкой соприкосновения сети предприятия и внешнего мира – в том, что касается служб с ограниченным доступом. Некоторые из служб, заблокированных внутренним шлюзом, могут оказаться тем, ради чего создавалась сеть. Такие жизненно необходимые службы работают безопасным образом на узле-бастионе. Одни службы узел-бастион предоставляет самостоятельно (в частности, DNS, почтовые службы SMTP, а также анонимный FTP-доступ), другие представлены *посредническими службами* (proxy services). Когда узел-бастион выступает в роли прокси-сервера, внутренние клиенты подключаются ко внешнему миру через узел-бастион, а внешние системы отвечают внутренним клиентам через этот же узел. Узел-бастион, таким образом, имеет возможность контролировать проходящий в обе стороны трафик в желательной степени.

Защищенных серверов может быть (и часто бывает) несколько. Сетевой периметр объединяет такие серверы, а также связывает внешний и внутренний маршрутизаторы. Системы сетевого периметра гораздо более подвержены атакам, чем системы внутренней сети. Так оно и должно быть. В конце концов, защищенные серверы нужны, чтобы предоставлять доступ к службам как для внешних, так и для внутренних клиентов. Изоляция систем, которые должны быть доступны внешнему миру, в отдельной сети сокращает возможность того, что нарушение безопасности одной из систем этой сети приведет к нарушению безопасности системы из внутренней сети.

Архитектура многосетевого узла – это попытка сочетать все описанные функции брандмауэра на одном компьютере. Маршрутизатор IP заменяется многосетевым узлом, который не выполняет пересылку пакетов на уровне IP.[1] Многосетевой узел, по существу, разрывает связь между внутренней и внешней сетями. Чтобы дать внутренней сети определенный уровень подключения, такой узел выполняет функции, сходные с функциями узла бастиона.

На рис. 12.5 представлены различия IP-маршрутизатора и брандмауэра на базе многосетевого узла. Маршрутизатор передает пакеты через уровень IP. Передача для каждого пакета основывается на конечном адресе, а также маршруте к этому конечному адресу, существующем в таблице маршрутизации. Узел же не просто пересылает пакеты. Многосетевой узел способен обрабатывать пакеты на прикладном уровне, что дает ему полный контроль над тем, как именно обрабатываются пакеты.[2]

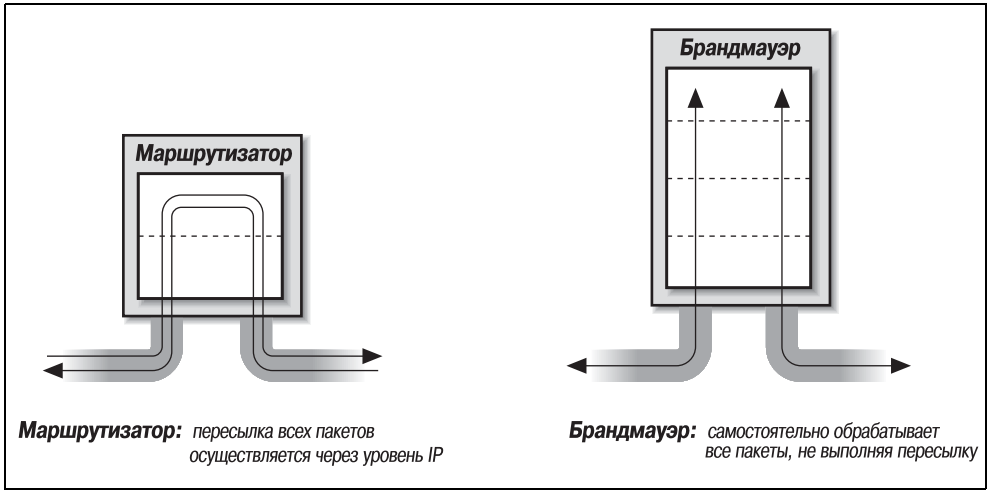


***Рис. 12.5.** Брандмауэры против маршрутизаторов*

Такое определение брандмауэра – устройство, не имеющее ничего общего с маршрутизатором IP, – не является повсеместно принятым. Некоторые склонны называть брандмауэрами маршрутизаторы с функциями экранирования, но по большому счету это все – вопрос семантики. В настоящей книге маршрутизаторы с функциональностью для обеспечения безопасности называются «защищенными маршрутизаторами» или «защищенными шлюзами». Брандмауэры, хоть и могут сочетаться с маршрутизаторами, не просто фильтруют пакеты.

---

[1] Роль IP-маршрутизаторов, называемых также шлюзами, в построении сети Интернет подробно рассмотрена в предшествующих главах.

[2] Информация о том, как запретить многосетевому узлу пересылку пакетов, приведена в главе 5.

Функции брандмауэра

В идеале злоумышленник не имеет возможности напрямую атаковать какую-либо из систем, защищенных брандмауэром. Пакеты, предназначенные защищенным узлам, доставляются на машину-брандмауэр. Так что злоумышленнику приходится атаковать непосредственно эту машину. Поскольку брандмауэр может стать мишенью атак с целью взлома, он должен придерживаться очень жестких мер обеспечения безопасности. Но поскольку всего один брандмауэр обслуживает многие машины локальной сети, проводить в жизнь политику обеспечения безопасности становится существенно проще.

Недостаток системы брандмауэров очевиден. Сетевой экран ограничивает доступ не только со стороны внешнего мира, но и доступ локальной сети к внешнему миру. Чтобы сократить неудобства, вызываемые присутствием брандмауэра, эта система должна выполнять гораздо больше задач, чем маршрутизатор. Некоторые брандмауэры предоставляют службы:

- DNS для внешнего мира
- Ретрансляции сообщений электронной почты
- Посреднические

В системе брандмауэра должен работать лишь минимальный набор служб, действительно необходимых для общения с внешними системами. Прочие распространенные сетевые службы (NIS, NFS, X Windows, `finger` и т. д.), как правило, предоставлять не рекомендуется. Ограничение набора служб связано с необходимостью сократить число потенциальных уязвимостей, посредством которых может получить доступ злоумышленник. Для брандмауэров безопасность превыше всего, и даже служб.

Самые серьезные осложнения для машин брандмауэров связаны со службой `ftp` и службой удаленного терминального доступа. С целью сохранения высокого уровня защищенности на машине брандмауэра не рекомендуется создавать пользовательские учетные записи; при этом данные пользователей должны проходить через брандмауэр, чтобы работали службы `ftp` и удаленного терминального доступа. Проблема решается созданием специальных учетных записей для `ftp` и `telnet`, общих для всех внутренних пользователей. Однако групповые учетные записи обычно считаются проблемой безопасности. Более приемлемое решение – разрешить работу службы `ssh` через брандмауэр. Это поощряет применение `ssh`, а значит – серьезных механизмов проверки подлинности и шифрования обмена данными.

Создание эффективного брандмауэра требует не только особого внимания, но и настройки многочисленных переменных в процессе создания экрана. Поэтому существуют специальные программные комплексы для создания брандмауэров. Продается даже специализированное аппаратное обеспечение, выполняющее функции системы-брандмауэра. Существует ряд дешевых программных пакетов для Linux. Прежде чем приступать к созданию собственного брандмауэра, выясните, какие варианты предлагают разработчики программных и аппаратных решений.

Подробности процесса установки брандмауэра выходят за пределы этой книги. Я рекомендую прочесть книги «Building Internet Firewalls» («Создание защиты в Интернете») и «Firewalls and Internet Security» (Брандмауэры и безопасность в Интернете). Решение о самостоятельной установке брандмауэра будет ошибкой, если у вас нет квалифицированных системных администраторов, способных уделить адекватное время этой задаче. Обратитесь в компанию, которая специализируется на проектировании и установке брандмауэров. Если ваша информация настолько ценна, что требует защиты экраном, она достаточно ценна, чтобы защитить ее брандмауэром, созданным профессионалами.

Разумеется, услуги профессионалов по карману не каждой сетевой площадке – особенно, если речь идет о небольшом офисе или домашней сети. Если у вас нет времени или денег, можно купить дешевый брандмауэр-маршрутизатор, называемый иногда *экранирующим устройством* (*firewall appliance*). Эти устройства разрабатываются специально для защиты небольших офисных и домашних сетей и предоставляют возможности простой фильтрации пакетов, работы с посредническими службами, а также преобразования адресов. Стоимость подобных устройств часто не превышает нескольких сотен долларов. В большинстве случаев можно просто купить устройство и включить его в сеть. Ваша сеть заслуживает как минимум такого уровня защиты. Если у вас есть время и возможность создать брандмауэр, можно воспользоваться специальным пакетом или инструментами, встроенными в операционную систему. Пакет брандмауэра стоит дополнительных денег, но с ним легко работать. Инструменты для фильтрации пакетов, встроенные в операционную систему, бесплатны, но с ними процесс настройки наиболее труден. Хорошим примером встроенного инструмента для Unix-системы является `iptables`, поставляемый в составе Linux.

Фильтрация трафика при помощи iptables

В простейшем случае брандмауэр – это фильтрующий маршрутизатор, который блокирует нежелательный трафик. Используйте возможности маршрутизации многосетевого узла под управлением Linux и функции фильтрации `iptables` для создания фильтрующего маршрутизатора.

Ядро Linux делит трафик маршрутизатора на три категории и применяет для каждой из категорий отдельный набор правил фильтров:

INPUT

Входящий трафик, адресованный процессу локальной системы, должен пройти через правила фильтра INPUT, прежде чем будет принят системой.

OUTPUT

Исходящий трафик, источником которого является локальная система, должен пройти через правила фильтра OUTPUT, прежде чем будет отправлен.

FORWARD

Трафик, исходящий от внешней системы и адресованный другой внешней системе, должен пройти через правила фильтра FORWARD.

Правила INPUT и OUTPUT используются в случае, когда система выступает в роли узла. Правила FORWARD используются, когда система выступает в роли маршрутизатора. Помимо трех стандартных категорий, `iptables` позволяет пользователям создавать собственные категории.

Создание фильтрующих правил iptables

Ядро Linux хранит список правил для каждой из описанных категорий. Работа со списком правил осуществляется при помощи команды `iptables`.[1] Используйте ключи команды `iptables`, описанные в табл. 12.2, для создания или удаления пользовательских цепочек правил, добавления правил в цепочку, удаления правил из цепочки, а также для изменения порядка следования правил в цепочке.

Таблица 12.2. Ключи командной строки iptables

Ключ	Назначение
-A	Добавляет правила в конец набора
-D	Удаляет правила из набора
-E	Изменяет имя набора
-F	Удаляет все правила из набора
-I	Вставляет правило в указанной точке цепочки (набора)
-L	Перечисляет все правила набора
-N	Создает пользовательский набор правил с указанным именем
-P	Устанавливает область применения цепочки
-R	Заменяет правило цепочки
-X	Удаляет указанный пользовательский набор правил
-Z	Обнуляет все счетчики пакетов и байтов

Правило брандмауэра состоит из фильтра, с которым сопоставляются пакеты, и действия, предпринимаемого, если пакет соответствует фильтру. Действие может являться стандартным правилом либо переходом к пользовательскому набору правил, реализующему дополнительную обработку. Ключ командной строки `-j` *цель* указывает пользовательский набор правил или стандартное правило обработки пакета. *цель* может быть представлена

[1] Команда `iptables` появилась в ядре Linux ветви 2.4. В более ранних версиях использовались команды `ipfwadm` и `ipchains`. Информацию об этих командах можно почерпнуть в книге Роберта Зиглера (Robert Ziegler) «Linux Firewalls» («Брандмауэры в Linux», Вильямс, 2000) .

именем набора правил либо ключевым словом стандартного правила. Существуют следующие ключевые слова для стандартных правил:

ACCEPT

Разрешает передачу пакета через брандмауэр.

DROP

Предписывает удалить пакет.

QUEUE

Предписывает передать пакет в область пользовательских процессов для обработки.

RETURN

В наборе правил, определенном пользователем, данное ключевое слово предписывает вернуться в набор правил, из которого произошел вызов. В одном из трех стандартных наборов правил ядра RETURN предписывает прервать обработку цепочки и воспользоваться стандартным правилом для этой цепочки.

Команды `iptables` создают фильтры, применяемые в зависимости от протокола, адресов источника и адресата либо сетевого интерфейса, через который поступил пакет. Этой цели служат многочисленные ключи командной строки. Ниже описаны основные ключи `iptables` для создания фильтров:

`-p` *протокол*

Определяет протокол, для которого справедливо правило. В качестве значения аргумента *протокол* может выступать любой номер из файла */etc/protocols* либо одно из ключевых слов: `tcp`, `udp`, `icmp`.

`-s` *адрес*[*/маска*]

Определяет адрес источника пакетов, для которых справедливо правило. *адрес* может быть представлен именем узла, именем сети либо адресом IP.

`--sport` [*порт*[`:`*порт*]]

Определяет исходный порт пакетов, для которых справедливо правило. *порт* может быть представлен именем или номером из файла */etc/services*. Синтаксис *порт*`:`*порт* позволяет определить диапазон портов. Если значение порта не указано, правило применяется для всех исходных портов.

`-d` *адрес*[*/маска*]

Определяет конечный адрес пакетов, для которых справдливо правило. *адрес* может быть представлен именем узла, именем сети либо адресом IP.

`--dport` [*порт*[`:`*порт*]

Определяет целевой порт пакетов, для которых справедливо правило. Правило фильтрует весь трафик, проходящий через указанный порт. *порт* определяется по правилам, описанным для ключа `--sport`.

`--icmp-type` *тип*

Определяет тип ICMP, для которого справедливо правило. *тип* может быть представлен любым именем или номером типа сообщения ICMP.

`-i` *имя*

Определяет имя входящего сетевого интерфейса, для которого справедливо правило. Правило действует только на пакеты, полученные через этот интерфейс. Символ + в конце имени позволяет задавать интерфейсы по маске (`eth+` соответствует всем Ethernet-интерфейсам, имена которых начинаются с `eth`).

`-o` *имя*

Определяет имя исходящего сетевого интерфейса, для которого справедливо правило. Правило действует только на пакеты, передаваемые через этот интерфейс. Символ + в конце имени позволяет задавать интерфейсы по маске (`eth+` соответствует всем Ethernet-интерфейсам, имена которых начинаются с `eth`).

`-f`

Указывает, что правило относится только ко второму и последующим фрагментам пакета, разбитого на несколько частей.

Примеры команд iptables

Если собрать все описанные средства, мы получим брандмауэр, способный защитить сеть. Предположим, что у нас есть Linux-маршрутизатор, подключенный к сетевому периметру с адресом 172.16.12.254 через интерфейс eth0 и к внешней сети с адресом 192.168.6.5 через интерфейс eth1. Кроме того, предположим, что в сетевом периметре лишь два сервера – сервер `sendmail` и сервер Apache. Вот пример команд `iptables`, которые мы могли бы выполнить на этой Linux-системе с целью защиты сетевого периметра:

```
iptable -F INPUT
iptables -F FORWARD
iptables -A INPUT -i eth1 -j DROP
iptables -A FORWARD -i eth1 -s 172.16.0.0/16 -j DROP
iptables -A FORWARD -o eth1 -d 172.16.0.0/16 -j DROP
iptables -A FORWARD -d 172.16.12.1 25 -j ACCEPT
iptables -A FORWARD -d 172.16.12.6 80 -j ACCEPT
iptables -A FORWARD -j DROP
```

Первые две команды при помощи ключа `-F` очищают наборы правил, с которыми мы намереваемся работать. Третья команда предписывает удалять все пакеты из внешней сети, адресованные процессам Linux-маршрутизатора. Мы не хотим, чтобы к процессам маршрутизатора обращался кто-либо из внешнего мира.

Следующие две команды предписывают удалять пакеты, передаваемые во внешний мир с внутреннего адреса. Если пакеты приходят через внешний интерфейс и имеют внутренний адрес, они удаляются. Точно так же, если пакеты, передаваемые через внешний интерфейс, имеют конечный адрес в локальной сети, они удаляются. Эти правила говорят, что если пакеты из внешней сети (проходящие через интерфейс eth1) некорректно используют адреса внутренней сети (172.16), кто-то пытается организовать атаку, основанную на подделке пакетов, и такие пакеты следует удалять.

Следующие два правила практически идентичны. Они предписывают принимать пакеты, если пункт назначения и номер порта соответствуют конкретному серверу. Например, порт 25 – это порт SMTP, а 172.16.12.1 – адрес почтового сервера, тогда как порт 80 – это порт HTTP, а 172.16. 12.6 – адрес веб-сервера. Мы принимаем такие входящие соединения, поскольку они адресованы нужным системам. Последнее правило запрещает пропускать любой другой трафик.

Эти примеры иллюстрируют мощь встроенных механизмов фильтрации Linux и содержат достаточный объем информации, чтобы вы могли начать самостоятельную работу. Разумеется, при создании боевого брандмауэра можно и нужно сделать гораздо больше. Если вы хотите узнать больше о команде `iptables`, обратитесь к книгам «Создание защиты в Интернете» (Building Internet Firewalls) и «Безопасность Linux» (Linux Security) в поисках более подробных примеров.

Последнее напутствие

Я не специалист по безопасности, я сетевой администратор. С моей точки зрения, хорошая безопасность – это хорошее системное администрирование, и наоборот. По большей части эта глава состоит из практических советов. Этих советов, вероятно, будет достаточно во многих случаях, но, разумеется, не во всех.

Потрудитесь узнать, существует ли уже руководящий документ по безопасности для вашей сети или системы. Если существуют правила, нормы, законы, определяющие курс действий, следуйте им. Ни при каких обстоятельствах не совершайте действий, направленных на нарушение установленной для вашей площадки системы безопасности.

Абсолютно защищенных систем не бывает. Независимо от объема принятых мер, проблемы будут возникать. Примите это как должное и будьте готовы. Разработайте план восстановления после происшествий и сделайте все необходимое, чтобы даже в наихудшем случае была возможность восстановиться с минимальными потерями времени и данных.

Тем, кого интересуют дополнительные подробности по безопасности, я рекомендую прочесть следующие документы и книги:

- RFC 2196, Site Security Handbook (Учебник по безопасности сетевых площадок), Б. Фрейзер (B. Fraser), September 1997.
- RFC 1281, Guidelines for the Secure Operation of the Internet (Указания по безопасной работе сети Интернет), Р. Пефья (R. Pethia), С. Крокер (S. Crocker), Б. Фрейзер (B. Fraser), November 1991.
- «Practical Unix and Internet Security» (Практическая безопасность Unix и Интернет), Симсон Гарфинкел (Simson Garfinkel) и Джин Спеффорд (Gene Spafford), O'Reilly & Associates, 1996.
- «Linux Security» (Безопасность Linux), Рамон Хонтанон (Ramon Hontanon), Sybex, 2001.

- «Building Internet Firewalls» Элизабет Цвики (Elizabeth Zwicky), Саймон Купер (Simon Cooper) и Брент Чапмен (Brent Chapman), O'Reilly & Associates, 2000.[1]
- «Linux Firewalls» Роберт Зиглер (Robert Ziegler), New Riders, 2000.[2]
- «Firewalls and Internet Security» (Брандмауэры и безопасность в Интернете), Уильям Чезвик (William Cheswick) и Стивен Белловин (Steven Bellovin), Addison Wesley, 1994.

Резюме

Доступ к сети и компьютерная безопасность имеют прямо противоположные цели. Подключение компьютера к сети повышает его уязвимость. Чтобы определить, какие типы защиты использовать и насколько ревностно защищать свои системы, необходимо оценить потребность в защищенности. Разработайте письменный план обеспечения безопасности площадки, определяющий процедуры действий и документирующий связанные с безопасностью обязанности сотрудников всех уровней.

Безопасность сети – это, по сути дела, хорошая безопасность отдельных систем. Проверка подлинности пользователей, эффективные способы наблюдения за системой, а также хорошо подготовленные системные администраторы – вот залог хорошей безопасности. Существуют инструменты, облегчающие решение подобных задач. SSH, OPIE, Tripwire, OpenSSL, `iptables`, TCP-оболочки, шифрование и брандмауэры – все они в вашем распоряжении.

[1] Э. Цвики, С. Купер и Б. Чапмен «Создание защиты в Интернете». – Пер. с англ. – СПб: Символ-Плюс, 2002.

[2] Роберт Зиглер «Брандмауэры в Linux». – Пер. с англ. – Вильямс, 2000.

- *Подход к проблеме*
- *Инструменты диагностирования*
- *Проверка наличия подключения*
- *Разрешение проблем доступа к сети*
- *Проверка маршрутизации*
- *Проверка службы имен*
- *Анализ проблем протоколов*
- *Пример исследования для протокола*

13

Разрешение проблем TCP/IP

Задачи сетевого администрирования делятся на две очень непохожих категории: настройка и отладка. Задачи настройки связаны с предсказуемыми явлениями: они требуют обстоятельных познаний в синтаксисе команд, но просты и прямолинейны. Когда система корректно настроена, редко возникает причина изменять настройки. Процесс настройки пов торяется каждый раз после установки новой версии операционной системы, но с минимальными вариациями.

Напротив, диагностирование сетевых проблем связано с непредвиденными явлениями. Разрешение проблем часто требует знаний не столько обстоятельных, сколько концептуальных. Сетевые проблемы обычно неповторимы, и иногда их сложно решать. Разрешение проблем – важная составная часть обеспечения стабильности и надежности работы сети.

В этой главе мы обсудим инструменты администратора, помогающие поддерживать сеть в хорошем рабочем состоянии. Но просто хороших инструментов недостаточно. От хорошего инструмента отладки мало толку, если он используется наобум. Эффективное диагностирование требует методичного подхода к обнаружению проблем, а также базовых познаний в принципах работы сети. Мы начнем с выбора подхода к сетевой проблеме.

Подход к проблеме

Чтобы правильно подойти к проблеме, необходимы базовые познания в TCP/IP. Первые главы этой книги рассказывают об основах TCP/IP и содержат достаточный для разрешения большинства сетевых проблем объем те-

матических сведений. Знание того, как TCP/IP маршрутизирует данные по сети, между отдельными узлами и между уровнями стека протоколов, важно для понимания сетевой проблемы. При этом знание подробностей функционирования каждого из протоколов обычно не требуется. Если вам понадобилась такая информация, ее можно получить из полноценного справочника, а не пытаться восстановить по памяти.

Не все проблемы TCP/IP похожи, и не все проблемы можно решить одним способом. Однако ключом к разрешению любой проблемы является понимание, в чем проблема заключена. Найти ответ на этот вопрос не так просто, как кажется на первый взгляд. «Внешний вид» проблемы временами обманчив, а «настоящая» проблема часто скрыта многочисленными уровнями программного обеспечения. Когда действительная суть проблемы раскрыта, решение часто оказывается очевидным.

Прежде всего, соберите подробнейшую информацию о том, что именно происходит. Если о проблеме сообщает пользователь, поговорите с ним. Узнайте, с каким приложением связан сбой. Определите имя и IP-адрес удаленного узла, имя и адрес узла пользователя. Какое сообщение об ошибке было получено? По возможности воспроизведите проблему в процессе беседы с пользователем. Дайте пользователю повторно выполнить приложение. Если возможно, воспроизведите проблему на своей собственной системе.

Действуя с системы пользователя и других систем, уточните следующие моменты:

- Возникает ли подобная проблема в других приложениях на узле пользователя или же проблема связана с единственным приложением? В последнем случае дело может быть в неверной настройке приложения либо в том, что это приложение не поддерживается удаленным узлом. Из соображений безопасности многие системы блокируют работу отдельных служб.
- Проблема возникает при взаимодействии с единственным удаленным узлом, всеми удаленными узлами либо только определенной «группой» удаленных узлов? Если такой удаленный узел один, проблема может быть связана именно с ним. Если же под подозрение попадают сразу все удаленные узлы – проблема, вероятно, в системе пользователя (в особенности если подобная проблема не возникает у других узлов локальной сети). Если же под вопросом только узлы определенных подсетей или внешних сетей, проблема может быть связана с маршрутизацией.
- Присутствует ли та же проблема на других локальных системах? Не забудьте проверить другие системы той же подсети. Если проблема возникает только на узле пользователя, сосредоточьте усилия на этой машине. Если проблема возникает на всех системах подсети, сосредоточьтесь на маршрутизаторе этой подсети.

Разобравшись с симптомами проблемы, представьте себе все протоколы и устройства, через которые проходят данные. Это убережет от чрезмерных упрощений и предположений, что причина известна уже до начала тестирования. Воспользуйтесь своими познаниями в TCP/IP, чтобы сосредоточить

внимание на наиболее вероятных причинах проблемы, но при этом сохраняйте непредвзятый подход.

Советы по разрешению проблем

Здесь я привожу ряд полезных советов по разрешению проблем. Они не являются частью методологии диагностирования. Это просто идеи, которые могут пригодиться.

- Подходите к проблемам методично. Руководствуйтесь в тестировании информацией, накопленной в процессе всех тестов. Не вздумайте по наитию резко менять курс тестирования, если нет уверенности, что вы потом сможете продолжить тест с того же места.
- Разбирайте проблему постепенно, разделяя ее на удобоваримые части. Выполняйте тестирование для каждой части, прежде чем перейти к следующей. Например, при проверке сетевого подключения проверяйте поочередно все сегменты сети, пока источник проблемы не будет найден.
- Ведите учет проведенных тестов и полученных результатов. Сохраняйте сведения о каждой из возникавших проблем – на случай их повторного появления.
- Подходите к вопросу непредвзято. Не стройте предположений относительно истинной природы проблемы. Некоторые всегда валят проблемы на свою сеть, тогда как другие постоянно считают источником проблем удаленный сервер. Некоторые настолько хорошо «знают», в чем проблема, что игнорируют результаты тестирования. Не делайте подобных ошибок. Проверяйте все возможности и в своих действиях исходите из показателей тестов.
- Помните о барьерах безопасности. Брандмауэры могут блокировать работу программ `ping` и `traceroute` и даже сообщения об ошибках ICMP. Если проблемы сосредоточены вокруг конкретной удаленной сетевой площадки, проверьте, не защищена ли она брандмауэром.
- Внимательно относитесь к сообщениям об ошибках. Формулировки сообщений об ошибках зачастую туманны, но периодически содержат важные намеки, позволяющие решить проблему.
- Воспроизводите проблему, о которой вам сообщили, самостоятельно. Не слишком полагайтесь на отчет пользователя о проблеме. Пользователь, вероятно, рассматривал ее только на прикладном уровне. Если необходимо, используйте файлы данных пользователя для воспроизведения проблемы. Даже если проблему не удается воспроизвести, зафиксируйте подробности на будущее.
- Причиной большинства проблем служат человеческие ошибки. Некоторые из подобных ошибок можно избежать, если предоставить пользователям информацию и учебные материалы по настройке и использованию сети.
- Своевременно информируйте пользователей, поскольку это сокращает число дублирующихся сообщений о проблемах и позволяет избежать слу-

чаев, когда несколько системных администраторов занимаются одной и той же проблемой совершенно независимо друг от друга. Возможно, что кто-то уже встречался ранее с этой проблемой и может поделиться полезными идеями на тему ее решения.

- Не рассуждайте о возможных причинах возникновения проблемы в разговоре с пользователем. Оставьте это для разговоров с коллегами по сети. Эти соображения могут быть восприняты пользователем в качестве откровения, и быстро превратятся в слухи. Эти слухи заставят пользователей избегать использования нормальных сетевых служб и подорвут уверенность в вашей сети. Пользователям нужны решения проблем, но никак не жаргонная болтовня.
- Работайте с небольшим набором простых инструментов диагностирования. В случае большинства программ TCP/IP можно обойтись инструментами, описанными в этой главе. Приобретение навыков работы с новым инструментом часто отнимает больше времени, чем решение проблемы при помощи старого, знакомого инструмента.
- Тщательно изучите проблему в своем участке сети, прежде чем начинать поиск владельцев удаленной системы с целью скоординировать с ними усилия по тестированию. Наибольшая сложность в диагностировании сетевых проблем заключена в том, что не всегда есть доступ к системам по обе стороны канала. Во многих случаях вообще неизвестно, кто управляет удаленной системой. Чем больше сведений вы получите на своей стороне, тем проще будет задача, если придется в итоге связаться с администратором удаленной системы.
- Не пренебрегайте очевидными решениями. Поврежденный или плохо подключенный кабель вполне может оказаться действительной причиной проблемы. Проверьте штекеры, разъемы, кабели и коммутаторы. Мелочи способны вызывать серьезные сложности.

Инструменты диагностирования

Поскольку причины большинства проблем просты, приобретение четкого представления о проблеме зачастую и является решением. К сожалению, это не всегда так, и в этом разделе мы начнем беседу об инструментах, позволяющих подготовить атаку на самую трудноизлечимую проблему. Существует множество средств диагностики – начиная от коммерческих систем со специализированным программным и аппаратным обеспечением стоимостью многие тысячи долларов и заканчивая бесплатными программами из комплекта поставки системы Unix. Кроме того, следует иметь под рукой некоторые инструменты.

Для работы с сетевым оборудованием и кабелями пригодится простейший ручной инструмент. Может оказаться достаточно пары острогубцев и набора отверток, однако не исключено, что возникнет необходимость в специализированных инструментах. Например, для присоединения разъема RJ-45 к не-

экранированной витой паре (UTP) требуется использование специального обжимного инструмента. Как правило, проще всего купить готовый набор инструментов для сетевых работ у вашего поставщика кабельной продукции.

Полноценный кабельный тестер также полезен. Современные кабельные тестеры – это небольшие портативные устройства с клавиатурой и жидкокристаллическим экраном, позволяющие тестировать как коаксиальные, так и UTP-кабели. Выбор тестов производится с клавиатуры, а результаты отображаются на жидкокристаллическом экране. Интерпретировать результаты не нужно – устройство делает все за вас и выводит сообщение об ошибке в виде простого текста. К примеру, тест кабеля может привести к выводу сообщения «Short at 74 feet». Из этого сообщения можно понять, что в кабеле присутствует замыкание в 74 футах[1] от тестера. Что может быть проще? Качественные инструменты тестирования упрощают поиск, а значит, и разрешение проблем с кабелями.

Весьма полезным элементом тестировочного оборудования может стать правильно настроенный портативный компьютер. Установите программное обеспечение TCP/IP. Возьмите компьютер с собой, направляясь к пользователю, сообщившему о сетевой проблеме. Отсоедините кабель Ethernet от пользовательской системы и подключите его к своему портативному компьютеру. Выполните настройку с подходящим адресом из подсети пользователя и перезагрузитесь. Затем прозвоните различные системы сети при помощи программы `ping` и подключитесь к одному из серверов пользователя. Если все работает, ошибка, скорее всего, связана с компьютером пользователя. Пользователи доверяют такому тесту, поскольку он воспроизводит то, что они делают ежедневно. Пользователи больше доверяют портативному компьютеру, чем непонятному прибору для тестирования, отображающему сообщение «No faults found» (Нет ошибок). Если же тестирование дает сбой, проблема, вероятно, связана с сетевым оборудованием или подключением. И вот тогда на сцене появляется тестер для кабелей.

Другим преимуществом использования портативного компьютера в качестве инструмента тестирования является его врожденная гибкость. Он позволяет работать с широким спектром приложений для тестирования, диагностики и управления. Установите Unix на этот портативный компьютер и работайте с программами, описанными в этой главе, как с настольного, так и с портативного компьютера.

В этой книге сделан упор на бесплатные или «встроенные» программные диагностические инструменты систем Unix. Программные инструменты, обсуждаемые в этой главе, а также многие другие описаны в документе RFC 1470, *FYI on a Network Management Tool Catalog: Tools for Monitoring and Debugging TCP/IP Internets and Interconnected Devices* (Информативный документ по каталогу инструментов управления сетями: инструменты для наблюдения и отладки Интернет-сетей TCP/IP и взаимосвязанных устройств). Броский заголовок и полезный документ! Документ в некоторой степени уста-

[1] Около 22 метров. – *Примеч. перев.*

рел, однако действительно обозначает ряд весьма полезных инструментов. Инструменты, перечисленные в каталоге и описанные в этой книге:

ifconfig

Предоставляет сведения о базовых настройках интерфейса. Полезна для поиска неверных IP-адресов, некорректных масок подсетей, а также неверных широковещательных адресов. `ifconfig` подробно описывается в главе 6. Данный инструмент поставляется в составе операционной системы Unix.

arp

Предоставляет информацию о преобразовании адресов Ethernet/IP. Может использоваться для обнаружения систем локальной сети, при настройке которых использовался неверный IP-адрес. `arp` описывается в этой главе и используется в примерах главы 2. `arp` входит в состав Unix.

netstat

Предоставляет самую разнообразную информацию. Повсеместно используется для вывода подробной статистики по каждому из сетевых интерфейсов, по сетевым сокетам, а также таблицам маршрутизации. `netstat` многократно упоминается в этой книге, а подробнее всего в главах 2, 6 и 7. `netstat` входит в состав Unix.

ping

Позволяет определить, доступен ли удаленный узел. `ping` отображает статистику по потерянным пакетам и времени доставки. Программа `ping` описана в главе 1 и используется в главе 7. `ping` также входит в состав Unix.

nslookup

Предоставляет информацию о службе имен DNS. `nslookup` подробно описывается в главе 8 и входит в состав программного пакета BIND.

dig

Также предоставляет информацию о службе имен. По функциональности схожа с `nslookup`.

traceroute

Выводит информацию о каждом из транзитных участков маршрута, по которому проходит пакет от вашей системы до удаленной.

snoop

Анализирует отдельные пакеты, которыми обмениваются узлы сети. `snoop` – это анализатор протоколов TCP/IP, включенный в состав системы Solaris 8. Он позволяет изучать содержимое пакетов, включая заголовки, и наиболее полезен для анализа проблем, связанных с протоколами. `tcpdump` – инструмент с аналогичными функциями, он поставляется в системах Linux.

В этой главе описаны все перечисленные инструменты, даже те, что уже описывались ранее. Мы начнем с программы `ping`, которая при отладке применяется чаще любого другого инструмента диагностирования.

Проверка наличия подключения

Команда ping проверяет, можно ли получить доступ к удаленному узлу с вашего компьютера. Эта простая функциональность крайне полезна при тестировании сетевого подключения, независимо от того, в каком приложении возникла проблема. ping позволяет определить, в каком направлении следует проводить дальнейшее тестирование – сетевого соединения (более низкие уровни) или же приложения (верхние уровни). Если ping сообщает, что пакеты достигают удаленной системы и возвращаются обратно, проблема пользователя, вероятно, связана с верхними уровнями. Если пакеты не могут вернуться, дело, скорее всего, в протоколах более низкого уровня.

Пользователь часто сообщает о сетевой проблеме, заявляя, что не может открыть сеанс telnet (или ftp, или же отправить почту, или что-то еще) на определенном удаленном узле. И тут же подкрепляет свое утверждение словами, что раньше все работало. В таких случаях, если возможность подключения к удаленному узлу оказывается под вопросом, ping – очень полезное средство.

Выполните прозвонку удаленного узла, указанного вам пользователем. Если ping сообщает, что все в порядке, попросите пользователя прозвонить узел. Если и на этот раз пакеты проходят, сосредоточьте дальнейший анализ в области конкретного приложения, с которым возникают проблемы у пользователя. Возможно, пользователь пытается наладить сеанс telnet с узлом, который предоставляет только анонимный ftp-доступ. Возможно, узел не работал именно в тот момент, когда пользователь попытался решить свою задачу. Попросите пользователя попробовать еще раз и очень внимательно наблюдайте за всеми его действиями. Если пользователь все делает правильно, однако ошибка в приложении по-прежнему возникает, может потребоваться тщательный анализ приложения при помощи snoop и даже обращение к администратору удаленной системы.

Если ваша ping-прозвонка проходит успешно, а ping с пользовательской машины дает сбой, проверьте настройки пользовательской системы, а также протестируйте отличия путей к удаленному узлу от вашей машины и от машины пользователя.

Если ваши ping-команды или ping-команды пользователя дают сбой, внимательно отнеситесь к любым сообщениям об ошибках. Сообщения об ошибках ping – полезные проводники в планировании дальнейшего тестирования. Конкретика сообщений может варьироваться от реализации к реализации, однако основных типов ошибок существует не так много:

Unknown host (Неверный узел)

Имя удаленного узла не может быть преобразовано службой имен в IP-адрес. Могут быть виноваты серверы имен (ваш локальный сервер имен или сервер удаленной системы), имя узла может быть неверным, или же возникли неполадки в сети, разделяющей вашу систему и удаленный сервер. Если известен IP-адрес удаленного узла, попробуйте выполнить для него команду ping. Если удалось связаться с узлом по его IP-адресу, в сбоях виновата служба имен. Воспользуйтесь nslookup или dig для тестирования

локальных и удаленных серверов, а также для проверки корректности имени узла, которое сообщил пользователь.

Network unreachable (Сеть недоступна)

В локальной системе нет маршрута к этой удаленной системе. Если в командной строке `ping` использовался численный IP-адрес, повторно наберите команду `ping` с именем узла. Это исключает возможность некорректного ввода адреса IP либо использования изначально неверного адреса. Если используется протокол маршрутизации, убедитесь, что он запущен, и проверьте таблицу маршрутизации при помощи `netstat`. Если используется статический маршрут по умолчанию, установите его повторно. Если создается впечатление, что на локальном узле все работает, проверьте шлюз по умолчанию на наличие проблем с маршрутизацией.

No answer (Нет ответа)

Удаленная система не ответила. Версии этого сообщения есть в большинстве сетевых приложений. Отдельные реализации `ping` выводят сообщение «100% packet loss (100% пакетов утеряно)». `telnet` отображает сообщение «Connection timed out» (Истек интервал ожидания соединения), а `sendmail` возвращает ошибку «cannot connect» (невозможно подключиться). Все эти сообщения означают одно: локальная система знает маршрут к удаленной системе, однако не получила ответы от удаленной системы ни на один из отправленных пакетов.

Существует множество возможных причин этой проблемы. Неработоспособность удаленного узла, некорректные настройки локального или удаленного узла, неработоспособность шлюза или цепи между локальным и удаленным узлами, проблемы маршрутизации на удаленном узле. Выявить истинную причину позволяет только дополнительное тестирование. Внимательно проверьте локальные настройки при помощи `netstat` и `ifconfig`. Отследите маршрут к удаленной системе при помощи `traceroute`. Свяжитесь с администратором удаленной системы и сообщите о проблеме.

Все упомянутые здесь инструменты мы еще обсудим в этой главе. Но прежде чем расстаться с `ping`, взглянем более пристально на команду и сведения статистики, которые она отображает.

Команда ping

Базовый формат команды `ping` в системе Solaris следующий:[1]

```
ping host [packetsize] [count]
```

host

Имя или IP-адрес удаленного узла, для которого выполняется прозвонка. Используйте имя узла или адрес, полученные от пользователя, сообщившего об ошибке.

---

[1] Сверьтесь с документацией по своей системе. Реализации `ping` немного отличаются одна от другой. В Linux приведенный синтаксис выглядит иначе: `ping [-c count] [-s packetsize] host`.

packetsize

Определяет размер тестовых пакетов в байтах. Это поле является обязательным, только если вы собираетесь использовать поле *count*. Используйте размер пакета по умолчанию, который составляет 56 байтов.

count

Число пакетов, передаваемых в ходе прозвонки. Используйте поле *count* и выбирайте небольшие значения. В противном случае команда `ping` будет посылать тестовые пакеты, пока вы не прервете ее работу, к примеру, комбинацией клавиш <Ctrl>+<C> (^C). Передача чрезмерного количества тестовых пакетов – пустая растрата пропускающей способности сетевых каналов и системных ресурсов. Для теста обычно достаточно пяти пакетов.

Чтобы проверить возможность доступа к *ns.uu.net* с узла *crab*, мы посылаем пять 56-байтовых пакетов следующей командой:

```
% ping -s ns.uu.net 56 5
PING ns.uu.net: 56 data bytes
64 bytes from ns.uu.net (137.39.1.3): icmp_seq=0. time=32.8 ms
64 bytes from ns.uu.net (137.39.1.3): icmp_seq=1. time=15.3 ms
64 bytes from ns.uu.net (137.39.1.3): icmp_seq=2. time=13.1 ms
64 bytes from ns.uu.net (137.39.1.3): icmp_seq=3. time=32.4 ms
64 bytes from ns.uu.net (137.39.1.3): icmp_seq=4. time=28.1 ms

----ns.uu.net PING Statistics----
5 packets transmitted, 5 packets received, 0% packet loss
round trip (ms)  min/avg/max = 13.1/24.3/32.8
```

Ключ `-s` нужен потому, что узел *crab* – рабочая станция под управлением Solaris, а нам нужна статистика по отдельным пакетам. В отсутствие ключа `-s` программа `ping` от Sun выводит только окончательный вердикт: «ns.uu.net is alive». Прочие реализации `ping` не требуют использования ключа `-s`; они отображают статистику по умолчанию, как можно видеть из приводимого ниже Linux-примера:

```
$ ping -c5 ns.uu.net
PING ns.uu.net (137.39.1.3) from 172.16.12.3 : 56(84) bytes of data.
64 bytes from ns.UU.NET (137.39.1.3): icmp_seq=0 ttl=244 time=98.283 msec
64 bytes from ns.UU.NET (137.39.1.3): icmp_seq=1 ttl=244 time=94.114 msec
64 bytes from ns.UU.NET (137.39.1.3): icmp_seq=2 ttl=244 time=66.565 msec
64 bytes from ns.UU.NET (137.39.1.3): icmp_seq=3 ttl=244 time=24.301 msec
64 bytes from ns.UU.NET (137.39.1.3): icmp_seq=4 ttl=244 time=37.060 msec

--- ns.uu.net ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round trip min/avg/max/mdev = 24.301/64.064/98.283/29.634 ms
```

Оба теста показывают наличие хорошего канала в территориальной сети к *ns.uu.net*, отсутствие потерь пакетов и быструю реакцию удаленной системы. Среднее время отклика для *almond* и *ns.uu.net* составило всего 24,3 миллисекунды. Потери некоторых пакетов и на порядок большее время отклика

не служат симптомами неисправностей для территориальных каналов. Однако статистика, отображаемая командой `ping`, может свидетельствовать о низкоуровневых сетевых проблемах. Ключевые элементы статистики:

- Последовательность прибытия пакетов, обозначенная порядковыми номерами ICMP (`icmp_seq`), выводимыми для всех пакетов.
- Длительность кругового путешествия пакетов, выраженная в миллисекундах; выводится после строки `time=`.
- Процент потерянных пакетов, отображаемый в последних строках, завершающих вывод `ping`.

Высокий процент потерь для пакетов, очень низкая скорость отклика либо неупорядоченное прибытие пакетов может свидетельствовать о проблемах на аппаратном уровне сетевых устройств. Если подобные показатели встречаются при работе с географически удаленными системами по территориальной сети, не о чем беспокоиться. Архитектура TCP/IP позволяет работать в ненадежных сетях, и некоторые территориальные сети в значительной степени подвержены потере пакетов. Те же проблемы в локальной сети – признак неисправностей.

В кабельном сегменте локальной сети время «кругосветного» путешествия пакета должно быть близко к нулю, потери пакетов должны быть минимальны или полностью отсутствовать и должен сохраняться порядок прибытия пакетов. Если одно из этих утверждений не соответствует действительности, существуют проблемы с сетевыми устройствами. В случае Ethernet речь может идти о неправильной оконечной блокировке кабеля, испорченном кабельном сегменте либо сбойном элементе «активного» оборудования, такого как коммутатор, концентратор или трансивер. Протестируйте кабель при помощи кабельного тестера, как описывалось выше. Качественные концентраторы и коммутаторы обычно имеют встроенное программное обеспечение, позволяющее выполнять диагностирование. Дешевые концентраторы и трансиверы могут потребовать решения проблемы методом «перебора» – то есть отключения отдельных устройств по одному, до тех пор, пока проблема не исчезнет.

Результаты простого `ping`-теста, даже успешного, помогут направить дальнейшие усилия по тестированию в сторону наиболее вероятных причин проблемы. Однако для более пристального изучения проблемы и обнаружения точной причины необходимы и другие инструменты диагностирования.

Разрешение проблем доступа к сети

Ошибки «no answer» и «cannot connect» свидетельствуют о проблеме в нижних уровнях сетевых протоколов. Если предварительные тесты указывают на проблему такого рода, сконцентрируйте тестирование в области маршрутизации и сетевого интерфейса. Воспользуйтесь командами `ifconfig`, `netstat` и `arp` для тестирования уровня доступа к сети (Network Access Layer).

Диагностирование при помощи ifconfig

ifconfig выводит настройки сетевого интерфейса. Используйте эту команду для проверки корректности настроек пользователя, если система пользователя настраивалась недавно или с нее невозможно получить доступ к удаленному узлу, с которым спокойно работают все прочие системы той же сети.

Если выполнить ifconfig с именем интерфейса в качестве единственного аргумента, программа отображает текущие значения параметров интерфейса. Например, проверка интерфейса dnet0 системы Solaris 8 дает такой результат:

```
% ifconfig dnet0
dnet0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
        inet 172.16.55.105 netmask ffffff00 broadcast 172.16.55.255
```

Команда ifconfig выводит две строки. Первая строка содержит имя интерфейса и его характеристики. Обратите внимание на следующие характеристики:

UP

Интерфейс функционален, может использоваться. Если интерфейс отключен, попросите администратора системы включить его командой ifconfig (например, ifconfig dnet0 up). Если интерфейс невозможно инициализировать, замените кабель интерфейса и попробуйте еще раз. Если и это не помогает, проверьте работоспособность устройства сетевого интерфейса.

RUNNING

Интерфейс действует. Если интерфейс не действует, возможно, драйвер устройства не был установлен как подобает. Администратор системы должен проверить, что все шаги установки были безошибочно выполнены.

Вторая строка вывода ifconfig содержит IP-адрес, маску подсети (в шестнадцатеричном формате), а также широковещательный адрес. Проверьте значения этих трех полей, чтобы убедиться, что интерфейс настроен правильно.

Некорректные маски подсетей и некорректные IP-адреса – две распространенные проблемы настройки интерфейсов. Некорректность маски подсети проявляется в том, что узел может связаться с другими узлами своей локальной подсети и удаленными узлами внешних сетей, однако не может связаться с узлами других локальных подсетей. ifconfig позволяет быстро обнаружить, что установлена неверная маска подсети.

Неверный IP-адрес может стать проблемой, с трудом поддающейся обнаружению. Если сетевая часть адреса неверна, все команды ping будут завершаться с ошибкой «no answer». В этом случае использование ifconfig позволяет раскрыть ошибку в адресе. Однако если неверно задан номер узла в адресе, проблему становится труднее обнаружить. Небольшая система – например персональный компьютер, который подключается к другим системам, но никогда не обрабатывает входящие соединения, – может долгое время работать с неверным адресом, и пользователь ничего не заметит. Более того, проблемы может испытывать вовсе не та система, что была некорректно настроена. Вполне возможно случайное использование вашего IP-адреса в дру-

гой системе, которое приведет к периодическим проблемам при обмене данными. Пример такого случая мы обсудим позже. Ошибки настройки такого типа не могут быть обнаружены при помощи `ifconfig`, поскольку ошибка сделана на удаленном узле. Чтобы решать подобные проблемы, необходимо использовать команду `arp`.

Диагностирование при помощи arp

Команда `arp` используется при анализе проблем, связанных с отображениями адресов IP в адреса Ethernet. Команда `arp` предоставляет три полезных для целей отладки ключа:

`-a`

Отобразить все ARP-записи из таблицы.

`-d` *имя-узла*

Удалить запись из таблицы ARP.

`-s` *имя-узла ether-адрес*

Добавить запись в таблицу.

Эти три ключа позволяют просматривать содержимое таблицы ARP, удалять проблемную запись, а также устанавливать корректную запись. Возможность установки корректной записи – это полезная «отсрочка», дающая время на поиск постоянного решения.

Используйте `arp`, если заподозрили наличие некорректных записей в таблице разрешения адресов. Одним из четких признаков проблем с таблицей ARP является сообщение, что на команду – скажем, `ftp` или `telnet` – ответил «не тот» узел. Нерегулярные проблемы, затрагивающие лишь определенные узлы, также могут свидетельствовать об ошибках в таблице ARP. Проблемы в таблице ARP обычно вызываются использованием одного и того же IP-адреса двумя различными системами. Проблемы ощущаются пользователями сети нерегулярно, поскольку в таблицу попадает запись для адреса узла, который быстрее других ответил на последний ARP-запрос. Иногда первым отвечает «правильный» узел, а иногда «не тот».

Если вы заподозрили, что две системы используют один и тот же IP-адрес, выведите таблицу разрешения адресов при помощи команды `arp -a`. Вот пример для системы Solaris.[1]

```
% arp -a
Net to Media Table: IPv4
Device   IP Address               Mask      Flags   Phys Addr
------ -------------------- --------------- ----- ---------------
dnet0  pecan                255.255.255.255       08:00:20:05:21:33
dnet0  horseshoe            255.255.255.255       00:00:0c:e0:80:b1
dnet0  crab                 255.255.255.255  SP   08:00:20:22:fd:51
dnet0  BASE-ADDRESS.MCAST.NET 240.0.0.0      SM   01:00:5e:00:00:00
```

[1] Формат вывода таблицы ARP может варьироваться от системы к системе.

Проще всего проверить корректность пар адресов IP/Ethernet, если существует информация о корректном для каждого отдельного узла адресе Ethernet. По этой причине следует записывать Ethernet- и IP-адреса каждого узла при его добавлении к сети. Обладая такими записями, вы быстро обнаружите несоответствия в таблице.

Если же такой записи нет, первые три байта адреса Ethernet помогут обнаружить проблему. Первые три байта адреса идентифицируют производителя устройства. Перечень таких идентификационных префиксов доступен по адресу *http://www.iana.org/assignments/ethernet-numbers*.

Исходя из префиксов, мы можем видеть, что две записи из таблицы ARP в нашем примере относятся к системам Sun (8:0:20). Предположим, *horseshoe* должен быть второй системой под управлением системы Sun, и тогда префикс 0:0:0c Cisco показывает, что маршрутизатор Cisco по ошибке получил IP-адрес узла *horseshoe*.

Если ни справочник корректных соответствий, ни префиксы производителей устройств не помогли определить источник ошибки в таблице ARP, попытайтесь использовать `telnet` для подключения к системе, IP-адрес которой содержится в записи ARP. Если устройство поддерживает `telnet`, приветственное сообщение может помочь определить, какой из узлов некорректно настроен.

Проблема ARP, случай из жизни

Позвонил пользователь и поинтересовался, доступен ли сервер, а затем сообщил о следующей проблеме. Рабочая станция пользователя, *limulus*, «подвисала» на время до нескольких минут при выполнении определенных команд, тогда как прочие команды работали превосходно. Проблемы с зависанием вызывали все сетевые команды, задействующие сервер имен NIS, но также и некоторые команды другого рода. Пользователь сообщил, что получает такое сообщение об ошибке:

```
NFS getattr failed for server crab: RPC: Timed out
```

Сервер *crab* предоставляет узлу *limulus* службы NIS и NFS. Команды, вызвавшие проблемы на узле *limulus*, требовали доступа к службе NIS, либо к файлам из экспортируемого узлом *crab* каталога */usr/local*. Программы, работавшие нормально, были установлены локально на рабочей станции пользователя. Больше ни у кого не возникали подобные проблемы с сервером, а прозвонка узла *limulus* с узла *crab* дала положительные результаты.

Мы попросили пользователя проверить файл *messages*[1] на предмет наличия сообщений об ошибках, и вот что он обнаружил:

```
Mar  6 13:38:23 limulus vmunix: duplicate IP address!!
      sent from ethernet address: 0:0:c0:4:38:1a
```

[1] Обратитесь к файлу */etc/syslog.conf* за полным именем файла сообщений. Обычно он называется */var/adm/messages* или */var log/messages*.

Это сообщение свидетельствует о том, что рабочая станция обнаружила в Ethernet-сегменте другой узел, отзывающийся на ее IP-адрес. «Самозванец» в своем ARP-ответе использовал адрес Ethernet 0:0:c0:4:38:1a, тогда как верный адрес Ethernet для *limulus* – 8:0:20:e:12:37.

Мы проверили таблицу ARP узла *crab* и обнаружили, что она содержит некорректную запись для узла *limulus*. Мы удалили эту запись при помощи команды `arp -d`, а затем установили корректную запись при помощи ключа `-s` следующим образом:

```
# arp -d limulus
limulus (172.16.180.130) deleted
# arp -s limulus 8:0:20:e:12:37
```

Записи ARP, полученные по протоколу ARP, являются временными. Значения хранятся в таблице определенный период времени и удаляются по его истечении. Затем по протоколу ARP система получает новые значения. Следовательно, если какие-либо удаленные интерфейсы изменяются, локальная таблица подстраивается под изменения, и обмен данными продолжается. Обычно это хорошая идея, но если кто-то пользуется некорректным IP-адресом, этот адрес будет постоянно всплывать в таблице ARP, даже если его удалять. Однако значения, указанные вручную, постоянны: они сохраняются в таблице и могут быть удалены только вручную. Поэтому мы смогли установить корректную запись в таблицу, не беспокоясь о том, что она будет перезаписана некорректным адресом.

Итак, мы быстро достигли результата и разрешили проблему узла *limulus*, однако нужно было отыскать виновника проблемы. Мы обратились к файлу */etc/ethers* в поисках записи для адреса Ethernet 0: 0:c0:4:38:1a, но не нашли ее. По первым трем байтам адреса, 0:0:c0, мы определили, что это устройство является картой Western Digital. Поскольку в нашей сети есть только рабочие станции Unix и персональные компьютеры, мы предположили, что карта Western Digital установлена в одном из последних. Кроме того, мы пришли к выводу, что проблемный адрес недавно установлен, поскольку пользователь раньше не обращался к нам с этой проблемой. Мы отправили всем пользователям срочное уведомление с запросом сведений о том, не устанавливал ли кто-либо в последнее время новый персональный компьютер, TCP/IP на такой машине, не выполнялась ли перенастройка существующих машин. Мы получили один ответ. Проверив систему этого пользователя, мы обнаружили, что он использовал адрес 172.16.180.130, тогда как следовало набирать 172.16.180.138. После исправления адреса проблема больше не возникала.

Вот так, без изысков, мы решили эту проблему. Обратившись к сообщениям об ошибках, мы выяснили, в чем проблема и как ее решать. Вовлечение в процесс всех пользователей сети позволило нам быстро обнаружить виновную систему и обойтись без поиска компьютера по всем комнатам. Нежелание привлекать пользователей для участия в решении проблемы – одна из самых дорогих и самых распространенных ошибок, совершаемых администраторами сетей.

Проверка интерфейса при помощи netstat

Если предварительные тесты дали вам основание заподозрить, что подключение к локальной сети ненадежно, полезную информацию может предоставить команда `netstat -i`. Следующий пример содержит результат выполнения команды `netstat -i` для системы Solaris 8:[1]

```
% netstat -i
Name  Mtu  Net/Dest         Address   Ipkts Ierrs Opkts Oerrs Collis Queue
dnet0 1500 wrotethebook.com crab      442697  2   633424  2   50679  0
lo0   1536 loopback         localhost 53040   0   53040   0   0      0
```

Строку кольцевого интерфейса `lo0` можно игнорировать. Значимой является только строка настоящего сетевого интерфейса, и только последние пять полей этой строки содержат информацию, полезную для диагностирования.

Взглянем сначала на последнее поле. В очереди (`Queue`) не должно быть пакетов, передача которых невозможна. Если интерфейс инициализирован и функционирует, но система не может передать пакеты в сеть, дело, возможно, в испорченном кабеле или интерфейсе. Замените кабель. Если проблемы остались, свяжитесь с производителем устройства на предмет починки.

Число входящих (`Ierrs`) и исходящих (`Oerrs`) ошибок должно быть близко к нулевому. Независимо от объема прошедшего через интерфейс объема данных значение 100 в любом из этих полей является высоким. Большое число исходящих ошибок может указывать на перегрузку локальной сети или же некачественный физический канал между узлом и сетью. Большое число входящих ошибок может указывать на перегрузку сети, узла либо проблемы на физическом уровне сети. Подтвердить наличие проблем на физическом уровне помогут такие инструменты, как `ping` и кабельный тестер. Оценка числа конфликтов поможет определить, произошла ли перегрузка локального сегмента Ethernet.

Высокое значение в поле конфликтов (`Collis`) допустимо, но если процент конфликтов для исходящих пакетов слишком высок, это свидетельствует о перегруженности сети. Уровень конфликтов, превышающий 5%, уже требует внимания. Если высокий уровень конфликтов наблюдается постоянно, причем в самых разных точках сети, может быть необходимо измельчение сегментов сети с целью сокращения загрузки трафиком.

Уровень конфликтов выражается в процентах от количества исходящих пакетов. Не используйте общее число отправленных и полученных пакетов, обращайте внимание на значения в полях Opkts и Collis, когда определяете уровень конфликтов. Например, вывод в приведенном ранее примере сообщает о 50 679 конфликтах из 633 424 исходящих пакетов. Уровень конфликтов в этом случае составляет 8%. Вполне возможно, что в этой сети возникает перегрузка – обратитесь к статистике с других узлов этой сети. Если и

[1] Формат вывода в системе Linux выглядит иначе, но содержит те же статистические данные.

другие системы демонстрируют высокий уровень конфликтов, задумайтесь о разделении сети на сегменты.

Деление сети Ethernet на сегменты

Чтобы снизить уровень конфликтов, необходимо сократить объем передаваемых данных в сегменте сети. Простейший способ решить задачу – разделить один сегмент на несколько сегментов. В каждом из новых сегментов будет меньше узлов и, как следствие, меньший трафик. Как станет ясно, на деле все несколько сложнее.

Наиболее эффективный способ выполнить деление сети Ethernet – установить коммутатор Ethernet. Каждый порт коммутатора по сути дела является отдельным сегментом. Таким образом, 16-портовый коммутатор дает возможность создать до 16 сегментов Ethernets в целях балансировки загрузки. В большинстве коммутаторов порты могут использоваться различными способами (рис. 13.1). Системы с небольшой загрузкой могут подключаться к концентратору, который в свою очередь подключается к одному из портов коммутатора; это позволяет объединять системы в один сегмент. Для серверов и систем с высокой сетевой нагрузкой могут формироваться выделенные сегменты – при помощи прямого подключения к портам коммутатора. В большинстве коммутаторов присутствуют порты Ethernet, работающие на скоростях как 10 Мбит/с, так и 100 Мбит/с (Fast Ethernet). Такие коммутаторы называются *асимметричными*, потому что различные порты работают с различными скоростями. Для подключения загруженных серверов или сетевых сегментов используйте порты Fast Ethernet. В большинстве коммутаторов с формулой 10/100 порты автоматически настраиваются на нужную скорость работы, что позволяет использовать любой порт на скорости 100 или 10 Мбит/с и дает максимальную гибкость при настройке.

Также могут применяться коммутаторы Gigabit Ethernet, но у них особое место в сетевой топологии. Коммутаторы 10/100 соединяют серверы и локальные сети. Гигабитные коммутаторы используются преимущественно для создания «вырожденной опорной сети» (collapsed backbone), объединяющей другие коммутаторы. Гигабитные коммутаторы используются при проектировании новой корпоративной магистральной сети, тогда как коммутаторы 10/100 – для выделения сегментов в сети Ethernet.

На рис. 13.1 представлен 8-портовый коммутатор Ethernet 10/100. Порты 1 и 2 соединены с концентраторами Ethernet. Через каждый концентратор подключен ряд систем. При добавлении систем они распределяются поровну между концентраторами в целях предотвращения перегрузки одного из сегментов. К свободным портам коммутатора в целях дальнейшего расширения могут подключаться дополнительные концентраторы. Порт 4 выделен под канал индивидуального сегмента системы с высокой сетевой загрузкой. Порт 6 работает на скорости 100 Мбит/с и обеспечивает подключение загруженного сервера. Один из портов может быть зарезервирован для подключения в будущем дополнительного коммутатора Ethernet 10/100, который даст возможность дальнейшего расширения.

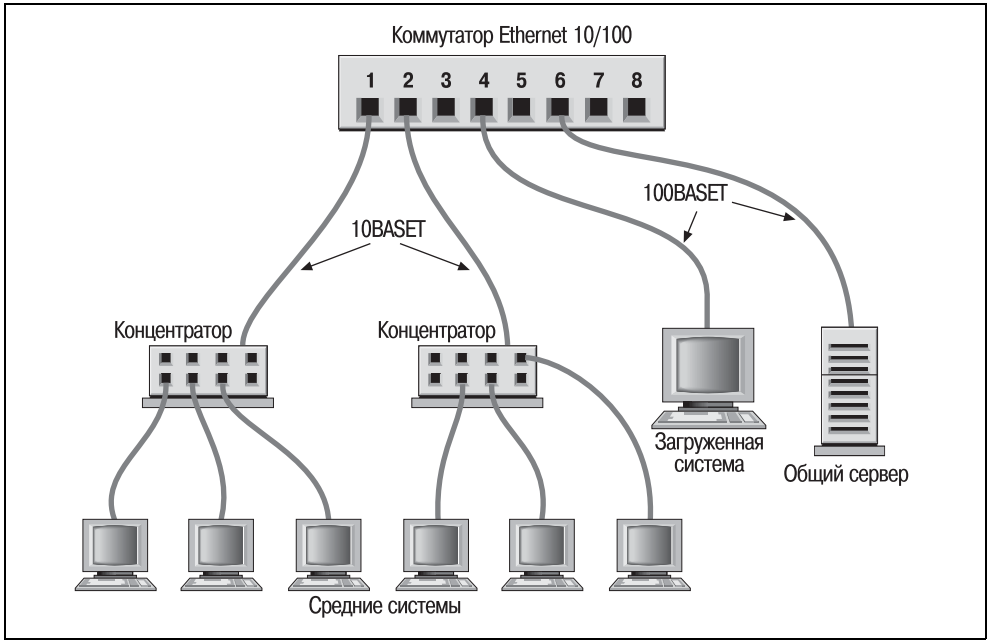


Рис. 13.1. *Подразделение сети Ethernet при помощи коммутаторов*

Прежде чем выделять порты на коммутаторе, выясните, какие службы пользуются спросом и кто с кем работает. Затем разработайте план, сокращающий объем трафика по отдельным сегментам. Например, если загруженная система на порту 4 использует значительную часть полосы пропускания канала только потому, что общается с одной из систем, подключенных через порт 1, все системы, подключенные через порт 1, будут испытывать затруднения вследствие этого. Компьютер, с которым общается загруженная система, необходимо перенести на свободный порт либо в тот же сегмент (порт 4). Максимально используйте возможности коммутатора для балансировки нагрузки.

Следует ли сегментировать старые коаксиальные Ethernet-сети, разрезая кабель и соединяя его при помощи маршрутизатора или ретранслятора? Нет. Если ваша старая сеть близка к насыщению, настало время создавать новую сеть, построенную на более надежной технологии. *Сеть с разделяемой пропускной способностью*, в которой все узлы подключены к одному кабелю (в данном случае – коаксиальному кабелю Ethernet), – это бочка с порохом. Сеть следует проектировать таким образом, чтобы пользователь не смог затруднить ее работу простым отключением собственной системы или даже случайным разрывом кабеля в своем офисе. Используйте неэкранированную витую пару (UTP), в идеале – кабель категории 5, для создания сетей 10BaseT Ethernet или 100BaseT Fast Ethernet, соединяющих оборудование в офисе пользователя с концентратором, надежно защищенным специальным шкафом. Сетевые компоненты на рабочем месте пользователя должны быть

в достаточной степени изолированы от сети, чтобы их повреждение не мешало ее работе. Новая сеть разрешит ваши проблемы с конфликтами пакетов и сократит объем работ по диагностированию аппаратных проблем.

Сетевые аппаратные проблемы

Некоторые из тестов, описанных в данном разделе, могут диагностировать проблему с сетевыми устройствами. При обнаружении такой проблемы свяжитесь с людьми, ответственными за аппаратное обеспечение. Если проблема предположительно связана с выделенной телефонной линией, свяжитесь с телефонной компанией. Если с территориальной сетью – с руководством этой сети. Не ждите, что проблема исчезнет сама по себе. Она может еще и усу губиться.

Если проблема в локальной сети, вам придется решать ее самостоятельно. Могут помочь некоторые инструменты, такие как кабельный тестер. Однако часто единственным подходом к обнаружению проблемы в аппаратной части становится перебор – отключение устройств по одному. Удобнее всего выполнять такую операцию на коммутаторе или концентраторе. Обнаружив устройство, вызывающее проблемы, почините или замените его. Помните, что проблемой может быть не только определенное устройство, но и просто кабель.

Проверка маршрутизации

Сообщение об ошибке «network unreachable» четко указывает на проблему маршрутизации. Если проблема в локальной таблице маршрутизации, ее легко обнаружить и устранить. Во-первых, воспользуйтесь командами `netstat -nr` и `grep`, чтобы определить, присутствует ли в таблице допустимый маршрут к целевой системе.[1] В данном примере проверяется наличие маршрута к сети 128.8.0.0:

```
% netstat -nr | grep '^128\.8\.'
128.8.0.0      26.20.0.16      UG      0    37    dnet0
```

Этот же тест на системе, не имеющей такого маршрута в таблице, завершится вообще без результатов. К примеру, пользователь сообщает, что «сеть легла», потому что он не может провести сеанс `ftp` с *helios.metalab.unc.edu*, а прозвонка возвращает следующие результаты:

```
% ping -s helios.metalab.unc.edu 56 2
PING helios.metalab.unc.edu: 56 data bytes
sendto: Network is unreachable
ping: wrote helios.metalab.unc.edu 64 chars, ret=-1
sendto: Network is unreachable
ping: wrote helios.metalab.unc.edu 64 chars, ret=-1
```

[1] `netstat -nr` работает в большинстве систем, однако администраторы Linux предпочитают команду `route -n`.

```
----helios.metalab.unc.edu PING Statistics----
2 packets transmitted, 0 packets received, 100% packet loss
```

Руководствуясь сообщением «network unreachable», проверьте таблицу маршрутизации пользователя. В данном случае мы ищем маршрут к системе *helios.metalab.unc.edu*. IP-адрес[1] *helios.metalab.unc.edu* – 152.2.210.81. Выполним поиск всех маршрутов, конечный пункт которых начинается с последовательности 152.2:

```
% netstat -nr | grep '^152\.2\.'
%
```

Данная проверка показала, что таких маршрутов нет. Если бы маршрут существовал, команда `grep` отобразила бы его. Поскольку нет конкретного маршрута для данного конечного пункта, вспомним, что есть еще и маршрут по умолчанию. Следующий пример показывает успешную проверку маршрута по умолчанию в системе Solaris:[2]

```
% netstat -nr | grep def
default       172.16.12.1     UG    0   101277   dnet0
```

Если `netstat` свидетельствует о наличии конкретного маршрута или действующего маршрута по умолчанию, проблема не в таблице маршрутизации. В таком случае воспользуйтесь командой `traceroute` (как описано в следующем разделе) для трассировки сегментов маршрута на пути к пункту назначения.

Если в таблице маршрутизации ожидаемый маршрут отсутствует, проблема в локальной маршрутизации. Существует два подхода к таким проблемам, в зависимости от того, статическую или динамическую маршрутизацию утилизует система. В случае статической маршрутизации установите нужный маршрут при помощи команды `route add`. Помните, большинство систем, использующих статическую маршрутизацию, полагаются на маршрут по умолчанию, поэтому нужным маршрутом вполне может быть маршрут по умолчанию. Убедитесь, что загрузочные файлы добавляют необходимый маршрут в таблицу при загрузке системы. Подробно команда `route add` описана в главе 7.

В случае динамической маршрутизации убедитесь, что программа маршрутизации запущена. Например, следующая команда позволяет убедиться, что запущен демон `gated`:

```
% ps 'cat /etc/gated.pid'
  PID TT STAT  TIME COMMAND
27711 ?  S   304:59 gated -tep /etc/log/gated.log
```

Если нужный демон маршрутизации не запущен, перезапустите его и предпишите трассировку. Трассировка позволяет обнаружить проблемы, вынуждающие демон аварийно завершать работу.

---

[1] Воспользуйтесь `nslookup` для определения адреса IP, если не знаете его. Программа `nslookup` описана далее в этой главе.

[2] В системе Linux выполняйте `grep` для сети 0.0.0.0, которая в Linux применяется для обозначения маршрута по умолчанию – вместо слова «default».

Трассировка маршрутов

Если локальная таблица маршрутизации в порядке, проблема может существовать на некотором удалении от локального узла. Удаленные проблемы маршрутизации могут приводить к получению сообщений об ошибке «no answer», а также «network unreachable». При этом сообщение «network unreachable» не всегда указывает на проблему маршрутизации, оно может означать, что удаленная сеть недостижима из-за физического разрыва между локальным узлом и удаленным пунктом назначения. Обнаружить такие проблемы позволяет программа `traceroute`.

`traceroute` отслеживает маршрут пакетов UDP, адресованных локальным узлом удаленному узлу. Программа отображает имя (по возможности) и адрес IP каждого шлюза на пути к удаленному узлу.

`traceroute` при трассировке пакетов использует два механизма: небольшое TTL (время жизни) пакета и некорректные номера портов. `traceroute` посылает пакеты UDP с небольшими значениями TTL, чтобы обнаружить промежуточные шлюзы. Значения TTL начинаются с 1 и увеличиваются на 1 для каждой группы из трех отправленных пакетов UDP. Получив пакет, шлюз уменьшает TTL. Если TTL к этому моменту имеет значение 0, пакет не пересылается дальше, а источнику пакета возвращается ICMP-сообщение «Time Exceeded» (время доставки превышено). `traceroute` выводит одну строку для каждого шлюза, от которого получено сообщение «Time Exceeded». На рис. 13.2 представлена одна строка вывода и описаны значения ее полей.

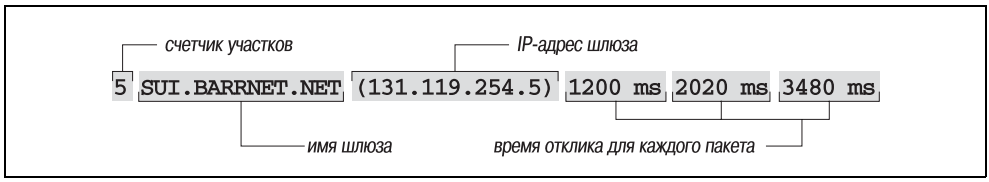


***Рис. 13.2.** Вывод traceroute*

Конечный узел, получив пакет от `traceroute`, возвращает ICMP-сообщение «Unreachable Port» (порт не доступен). Это происходит потому, что `traceroute` преднамеренно использует недействительный номер порта (33434), чтобы вызвать такую ошибку. Получив сообщение «Unreachable Port», `traceroute` делает вывод, что пункт назначения достигнут и прекращает трассировку. Итак, `traceroute` обладает способностью получать список шлюзов, обозначая транзитные участки, вплоть до момента, когда удаленный узел получит адресованный ему пакет. На рис. 13.3 представлена передача пакетов трассировки узлу, расположенному в трех транзитных участках от локального. Ниже приведен вывод `traceroute` для *www.internic.net*, инициированный системой Solaris в сети Comcast. `traceroute` посылает три пакета для каждого значения TTL. Если для пакета не получен ответ, `traceroute` выводит звездочку (*). Если ответ получен, `traceroute` отображает имя и адрес ответившего шлюза, а также время отклика для пакета в миллисекундах.

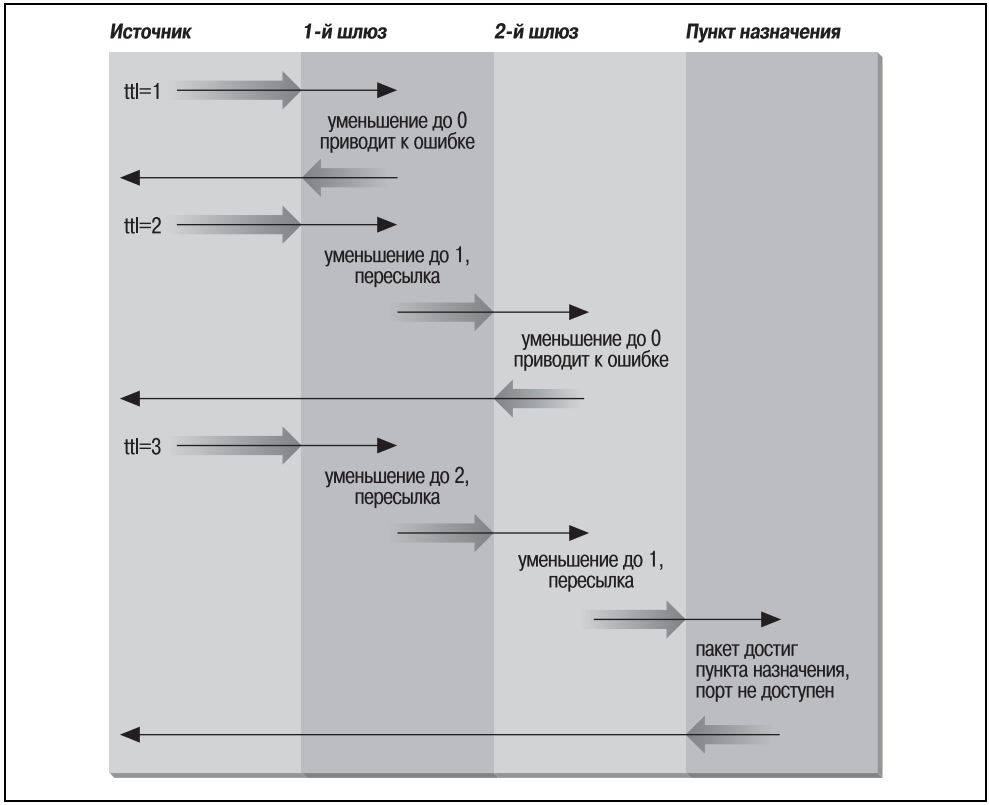


Рис. 13.3. *Передача пакетов traceroute*

```
$ traceroute www.internic.net
traceroute to www.internic.net (207.151.159.3), 30 hops max, 40 byte packets
 1 ani (192.168.0.1)  1.712 ms  1.40 ms  1.34 ms
 2 10.81.130.1 (10.81.130.1)  52.01 ms  34.38 ms  118.97 ms
 3 bb1-fe1-0.mtgmry1.md.home.net (24.11.248.1) 13.30 ms 100.92 ms 31.99 ms
 4 c2-se9-0-10.washdc1.home.net (24.7.73.25) 118.63 ms 94.92 ms 121.10 ms
 5 24.7.71.6 (24.7.71.6) 127.63 ms  26.29 ms  132.07 ms
 6 p4-6-1-0.r00.plalca01.us.bb.verio.net (129.250.2.245) 186.02 ms 164.81 ms 156.44 ms
 7 p16-0-0-0.r06.plalca01.us.bb.verio.net (129.250.2.161) 86.59 ms 130.28 ms 121.09 ms
 8 p16-0-0-0.r04.snjsca03.us.bb.verio.net (129.250.3.162) 84.594 ms 117.42 ms 174.59 ms
 9 p16-3-0-0.r01.snjsca03.us.bb.verio.net (129.250.2.63) 123.87 ms 91.39 ms 119.79 ms
10 p4-2-0-0.r00.lsanca01.us.bb.verio.net (129.250.2.26) 142.38 ms 166.11 ms 95.32 ms
11 ge-0-0-0.a02.lsanca02.us.ra.verio.net (129.250.29.116) 137.59 ms 98.28 ms 256.11 ms
12 uscisi-pl.customer.ni.net (209.189.66.66) 98.64 ms 125.03 ms 231.11 ms
13 207.151.151.2 (207.151.151.2) 192.06 ms 164.52 ms 103.30 ms
14 icann-IWC.interworld.net (206.124.230.170) 113.33 ms 145.72 ms 107.39 ms
15 * host159-3.icann.org (207.151.159.3) 99.67 ms 178.72 ms
```

Приведенная трассировка показывает, что маршрут включает 15 промежуточных шлюзов, пакеты проходят по маршруту и время отклика для паке-

тов, адресованных данным узлом системе *www.internic.net*, составляет около 140 миллисекунд.

Вариации и ошибки в реализациях ICMP на шлюзах различных типов, а также непредсказуемая природа сетевого пути, по которому может путешествовать дейтаграмма, могут приводить к странным результатам. По этой причине не слишком заостряйтесь на выводе `traceroute`. Самые важные вопросы, на которые призваны ответить результаты работы программы:

- Добрался ли пакет до пункта назначения?
- Если нет, где он был остановлен?

Следующий листинг содержит другую трассировку пути к *www.internic.net*. На этот раз пакет до InterNIC не добрался.

```
$ traceroute www.internic.net
traceroute to www.internic.net (207.151.159.3), 30 hops max, 40 byte packets
 1 ani (192.168.0.1)  1.712 ms  1.40 ms  1.34 ms
 2 10.81.130.1 (10.81.130.1)  52.01 ms  34.38 ms  118.97 ms
 3 bb1-fe1-0.mtgmry1.md.home.net (24.11.248.1) 13.30 ms 100.92 ms 31.99 ms
 4 c2-se9-0-10.washdc1.home.net (24.7.73.25) 118.63 ms 94.92 ms 121.10 ms
 5 24.7.71.6 (24.7.71.6) 127.63 ms  26.29 ms  132.07 ms
 6 p4-6-1-0.r00.plalca01.us.bb.verio.net (129.250.2.245) 186.02 ms 164.81 ms 156.44 ms
 7 p16-0-0-0.r06.plalca01.us.bb.verio.net (129.250.2.161) 86.59 ms 130.28 ms 121.09 ms
 8 p16-0-0-0.r04.snjsca03.us.bb.verio.net (129.250.3.162) 84.594 ms 117.42 ms 174.59 ms
 9  * * *
10  * * *
    .
    .
    .
29  * * *
30  * * *
```

Если `traceroute` не удается доставить пакеты удаленной системе, трассировка сходит на нет, и вывод в этом случае содержит тройки звездочек вместо сведениях о шлюзах между транзитными участками, вплоть до тридцатого. Свяжитесь с администратором удаленного узла, а также с администратором последнего шлюза из вывода `traceroute`. Объясните им, в чем проблема, возможно, они смогут оказать помощь. В нашем примере последним шлюзом из ответивших на пакет стал *p16-0-0-0.r04.snjsca03.us.bb.verio.net*. Следовательно, мы свяжемся с администратором этой системы и администратором *www.internic.net*.

Поиск администратора

Чтобы связаться с администратором удаленной системы, необходимо определить, кто он. `whois` помогает в поиске важных лиц. Один из важнейших элементов информации в сети – сведения о том, кто управляет второй стороной. В случае диагностирования сетевых проблем эту информацию позволяет получить `whois`.

whois извлекает нужную информацию из белых страниц сети Интернет. Белые страницы – это база данных сведений по ответственным лицам, ведут которую регистраторы сети Интернет. Чтобы получить официальный номер сети или доменное имя, вы должны предоставить контактную информацию, которая становится вашей личной записью в базе данных белых страниц. По этой причине предполагается, что все лица, ответственные за официальные сети или домены, имеют запись в белых страницах, и эта запись доступна всем, кому необходимо связаться с этими лицами.

Во многих системах Unix существует команда whois, позволяющая запрашивать сведения из белых страниц. Общий синтаксис команды:

```
% whois [-h server] name
```

Поле *name* определяет, какие сведения необходимо извлечь из базы данных белых страниц. Поле *server* содержит имя сервера системы белых страниц.

В следующем примере мы выполняем поиск контактной информации для домена *verio.net*, то есть для домена, в котором располагается удаленный маршрутизатор из примера по traceroute.

```
$ whois verio.net
[whois.crsnic.net]

Whois Server Version 1.3

Domain names in the .com, .net, and .org domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

   Domain Name: VERIO.NET
   Registrar: MELBOURNE IT, LTD. D/B/A INTERNET NAMES WORLDWIDE
   Whois Server: whois.inww.com
   Referral URL: http://www.inww.com
   Name Server: NS0.VERIO.NET
   Name Server: NS1.VERIO.NET
   Name Server: NS2.VERIO.NET
   Updated Date: 13-jun-2001


>>> Last update of whois database: Tue, 17 Jul 2001 02:04:28 EDT <<<

The Registry database contains ONLY .COM, .NET, .ORG, .EDU domains and
Registrars.


[whois.inww.com]

Domain Name.......... verio.net
  Creation Date........ 1996-12-07
  Registration Date.... 2000-05-10
  Expiry Date.......... 2001-12-06
  Organisation Name.... Verio, Inc.
  Organisation Address. 8005 South Chester Street
  Organisation Address. Suite 200
  Organisation Address. Englewood
```

```
   Organisation Address. CO
   Organisation Address. 80112
   Organisation Address. UNITED STATES

Admin Name.......... Hostmaster Verio
   Admin Address........ 8005 South Chester Street
   Admin Address........ Suite 200
   Admin Address........ Englewood
   Admin Address........ 80112
   Admin Address........ CO
   Admin Address........ UNITED STATES
   Admin Email.......... DomainAdmin@verio.net
   Admin Phone.......... 214 290 8620
   Admin Fax............ .

Tech Name........... Hostmaster Verio
   Tech Address......... 8005 South Chester Street
   Tech Address......... Suite 200
   Tech Address......... Englewood
   Tech Address......... CO
   Tech Address......... 80112
   Tech Address......... UNITED STATES
   Tech Email........... hostmaster@verio.net
   Tech Phone........... 214 290 8620
   Tech Fax............. .
   Name Server.......... NSO.VERIO.NET
   Name Server.......... NS1.VERIO.NET
   Name Server.......... NS2.VERIO.NET
```

По запросу отображается имя, адрес и телефонный номер контактных лиц домена, а также перечень компетентных серверов имен домена. Данный пример показывает, как должна работать система, и для серьезных сетей с грамотным управлением, таких как *verio.net*, она обычно так и работает. К сожалению, многие запросы `whois` не возвращают сколько-нибудь полезной информации, так как сопровождение базы данных белых страниц не на высоте. Если `whois` не возвращает нужной информации, попробуйте обратиться к службе имен DNS. Запись DNS SOA должна содержать почтовый адрес контактного лица для домена. Этот человек, вполне возможно, выдаст вам координаты искомого системного администратора.

Проверка службы имен

На проблемы с сервером имен указывает сообщение об ошибке «unknown host», полученное от пользовательского приложения. Проблемы, связанные со службой имен, обычно могут диагностироваться при помощи `nslookup` или `dig`. `nslookup` подробно обсуждался в главе 8; `dig` — это альтернативный инструмент со схожей функциональностью, и речь о нем пойдет в этой главе. Прежде чем обратиться к `dig`, взглянем еще раз на `nslookup` и его применение для отладки службы имен.

Особо важными для отладки проблем удаленного сервера имен являются три возможности nslookup, описанные в главе 8. Эти возможности отражают способность программы:

- Находить компетентные серверы удаленного домена при помощи запросов записей NS
- Извлекать все записи по удаленному узлу при помощи запросов ANY
- Просматривать все записи удаленной зоны при помощи команд nslookup ls и view

При диагностировании проблемы удаленного сервера обращайтесь напрямую к компетентным серверам, координаты которых получены по запросу NS. Не полагайтесь на информацию, предоставляемую некомпетентными серверами. Если проблемы возникают нерегулярно, опросите все компетентные серверы по очереди и сравните их ответы. Причиной нерегулярных проблем с серверами имен иногда становится тот факт, что удаленные серверы возвращают различные ответы на один и тот же запрос.

Запрос ANY возвращает все записи, в которых фигурирует определенный узел, что дает наиболее широкий диапазон сведений для отладки. Простое знание, какая информация доступна (и недоступна), может разрешить множество проблем. Например, если запрос возвращает запись MX, но не возвращает запись A (адресную), становится легко понять, почему пользователь не мог инициировать сеанс telnet с этим узлом! Многие узлы доступны для почтовой системы, но не для других сетевых служб. В данном случае пользователь запутался и пытается использовать удаленный узел неподобающим образом.

Если невозможно получить какую-либо информацию по имени узла, которое сообщил вам пользователь, возможно, имя узла неверно, и поиск правильного имени в этом случае можно сравнить с поиском иголки в стоге сена. Тем не менее программа nslookup способна помочь. Воспользуйтесь командой nslookup ls, чтобы получить копию файла удаленной зоны и записать ее в локальный файл. Затем используйте команду nslookup view для просмотра этого файла в поисках имен, похожих на представленное пользователем. Многие проблемы возникают из-за неверных имен узлов.

Все упомянутые здесь возможности и команды nslookup используются в главе 8. Тем не менее примеры использования этих команд для разрешения реальных проблем с серверами имен будут полезны. Следующие три примера основаны на реальных сообщениях о проблемах.[1]

Одни системы работают, другие – нет

Пользователь сообщил, что на его рабочей станции имя определенного узла разрешается, а на центральной системе – нет. При этом центральная система была способна разрешать имена других узлов. Мы провели ряд тестов и обнаружили, что на одних системах имя этого узла разрешается, а на других –

[1] Имена узлов и серверов вымышлены, хотя проблемы вполне реальны.

нет. Какой-то определенной и предсказуемой схемы у сбоев не было. Поэтому мы воспользовались nslookup для проверки удаленных серверов:

```
% nslookup
Default Server:  crab.wrotethebook.com
Address:  172.16.12.1

> set type=NS
> foo.edu.
Server:  crab.wrotethebook.com
Address:  172.16.12.1

foo.edu         nameserver = gerbil.foo.edu
foo.edu         nameserver = red.big.com
foo.edu         nameserver = shrew.foo.edu
gerbil.foo.edu   inet address = 198.97.99.2
red.big.com   inet address = 184.6.16.2
shrew.foo.edu    inet address = 198.97.99.1
> set type=ANY
> server gerbil.foo.edu
Default Server:  gerbil.foo.edu
Address:  198.97.99.2

> hamster.foo.edu
Server:  gerbil.foo.edu
Address:  198.97.99.2

hamster.foo.edu        inet address = 198.97.99.8
> server red.big.com
Default Server:  red.big.com
Address:  184.6.16.2
> hamster.foo.edu
Server:  red.big.com
Address:  184.6.16.2
 *** red.big.com can't find hamster.foo.edu: Non-existent domain
```

Данный сеанс nslookup содержит несколько шагов. Прежде всего, необходимо найти компетентные серверы для имени узла, о котором идет речь (*hamster.foo.edu*). Мы установили тип запроса в NS, чтобы получить соответствующие записи, и выполнили поиск для домена (*foo.edu*), к которому принадлежит имя узла. В результате мы получили имена трех компетентных серверов: *gerbil.foo.edu*, *red.big.com* и *shrew.foo.edu*.

Затем мы установили тип запроса в ANY с целью поиска произвольных записей, связанных с именем проблемного узла. Затем мы предписали nslookup обращаться к первому серверу из списка компетентных, *gerbil.foo.edu*, и выполнили запрос для *hamster.foo.edu*. Мы получили адресную запись. Итак, сервер *gerbil.foo.edu* работает замечательно. Мы повторили тест для сервера *red.big.com* и получили сбой. Запрос не вернул каких-либо записей.

Следующий шаг – получить записи SOA с каждого из серверов и выяснить, идентичны ли они:

```
> set type=SOA
> foo.edu.
```

```
Server:  red.big.com
Address:  184.6.16.2

foo.edu        origin = gerbil.foo.edu
   mail addr = amanda.gerbil.foo.edu
   serial=10164, refresh=43200, retry=3600, expire=3600000,
   min=2592000
> server gerbil.foo.edu
Default Server:  gerbil.foo.edu
Address:  198.97.99.2

> foo.edu.
Server:  gerbil.foo.edu
Address:  198.97.99.2

foo.edu        origin = gerbil.foo.edu
   mail addr = amanda.gerbil.foo.edu
   serial=10164, refresh=43200, retry=3600, expire=3600000,
   min=2592000
 > exit
```

Если записи SOA имеют различные порядковые номера, вполне возможно, что файл зоны (а следовательно, и данные узла) еще не был скопирован на подчиненный сервер. Если порядковые номера совпадают, а данные различаются, как в этом случае, присутствует конкретная проблема. Уведомите администратора удаленного домена. Почтовый адрес администратора содержится в поле «mail addr» записи SOA. В нашем примере письмо с описанием проблемы отправляется на адрес *amanda@gerbil.foo.edu*.

Данные на месте, но сервер не может их обнаружить!

Об этой проблеме сообщил администратор одного из наших подчиненных серверов имен. Администратор сообщил, что его сервер не может выполнить разрешение имени определенного узла из домена, для которого сервер является подчиненным. Основной сервер при этом совершенно свободно выполнял ту же операцию. Администратор получил образ кэша (более подробно об этом поговорим в следующем разделе), и таким образом убедился, что его сервер обладает корректной записью для проблемного узла. Несмотря на это, сервер не мог выполнить преобразование имени узла в IP-адрес!

Проблема была воспроизведена на ряде других подчиненных серверов. Имя узла поддавалось разрешению только основным сервером, но не подчиненными. Все серверы имели один и тот же порядковый номер в записи SOA, и образ кэша на каждом сервере показывал, что для узла существует корректная адресная запись. Так почему же они не могли преобразовать имя узла в адрес?

Поразмыслив о различиях между способами, которыми загружают данные основные и дополнительные серверы, мы заподозрили, что дело в передаче файла зоны. Основные серверы загружают свои данные непосредственно из локальных дисковых файлов. Подчиненные серверы получают данные от ос-

новного сервера путем передачи зоны. Возможно, происходила порча файлов зоны. Мы вывели файл зоны на одном из подчиненных серверов и увидели такие данные:

```
% cat /usr/etc/events.wrotethebook.com.hosts
PCpma       IN   A       172.16.64.159
            IN   HINFO   "pc" "n3/800eventsnutscom"
PCrkc       IN   A       172.16.64.155
            IN   HINFO   "pc" "n3/800eventsnutscom"
PCafc       IN   A       172.16.64.189
            IN   HINFO   "pc" "n3/800eventsnutscom"
accu        IN   A       172.16.65.27
cmgds1      IN   A       172.16.130.40
cmg         IN   A       172.16.130.30
PCgns       IN   A       172.16.64.167
            IN   HINFO   "pc" "(3/800eventsnutscom"
gw          IN   A       172.16.65.254
zephyr      IN   A       172.16.64.188
            IN   HINFO   "Sun" "sparcstation"
ejw         IN   A       172.16.65.17
PCecp       IN   A       172.16.64.193
            IN   HINFO   "pc" "n Lsparcstationstcom"
```

Обратите внимание на странное значение в последнем поле оператора HINFO каждой из машин.[1] Эти данные могли быть повреждены при передаче либо изначально были некорректными на основном сервере. Мы воспользовались nslookup, чтобы проверить это обстоятельство:

```
% nslookup
Default Server:  crab.wrotethebook.com
Address:  172.16.12.1

> server 24seven.events.wrotethebook.com
Default Server:  24seven.events.wrotethebook.com
Address:  172.16.6.1

> set query=HINFO
> PCwlg.events.wrotethebook.com
Server:  24seven.events.wrotethebook.com
Address:  172.16.6.1

PCwlg.events.wrotethebook.com     CPU=pc  OS=ov
packet size error (0xf7fff590 != 0xf7fff528)
> exit
```

В данном примере мы предписали nslookup обращаться к серверу *24seven.events.wrotethebook.com*, то есть основному серверу домена *events.wrotethebook.com*. Затем мы запросили запись HINFO одного из тех узлов, для которых заподозрили порчу данных. Сообщение «packet size error» четко показывает, что nslookup столкнулась с проблемами, пытаясь получить запись

[1] Подробное описание оператора HINFO приводится в приложении C.

HINFO напрямую с основного сервера. Мы связались с администратором основного сервера и рассказали ему о проблеме, указав на конкретные записи, попавшие под подозрение. Администратор обнаружил, что забыл добавить запись операционной системы в некоторые из записей HINFO. Он исправил ошибку, и проблема исчезла.

Повреждение кэша

Описанная выше проблема была вызвана порчей кэша сервера имен некорректными данными. Повреждение данных кэша может происходить даже в случаях, когда ваша система не является подчиненным сервером. Все серверы кэшируют ответы. Если эти ответы повреждены, записи кэша могут также получать повреждения. Просмотр образа кэша может способствовать диагностированию подобных проблем.

Например, пользователь сообщил о нерегулярных сбоях в работе сервера имен. С именами локального домена и некоторыми именами за его пределами проблем не возникало, однако имена ряда удаленных доменов не поддавались разрешению. Тестирование посредством `nslookup` не привело к результатам, которые можно было бы уверенно интерпретировать, поэтому мы перешли к просмотру образа кэша сервера имен. Записи корневых серверов оказались поврежденными, и демон `named` был перезагружен в целях очистки кэша и повторного чтения файла корневых указателей *named.ca*. Операцию мы выполнили следующим образом.

Команда `ndc dumpdb` или сигнал SIGINT предписывает `named` создать образ кэша сервера имен в файле */var/tmp/named_dump.db*. В следующем примере используется сигнал:

```
# kill -INT 'cat /etc/named.pid'
```

Идентификатор процесса `named` можно получить из файла */etc/named.pid*, что мы и сделали, поскольку `named` записывает идентификатор процесса в этот файл при старте.[1]

Записанный в файл образ можно изучить на предмет проверки корректности имен и адресов серверов. Файл *named_dump.db* состоит из трех разделов: таблицы зон, кэша и данных, а также указателей.

Раздел таблицы зон

Первый раздел файла образа содержит таблицу зон, отражающую зоны, загруженные при запуске сервера. Таблица зон основного сервера для зон *wrotethebook.com* и *16.172.in-addr.arpa* имеет следующий состав:

```
; Dumped at Tue Jul 17 16:08:18 2001
;; ++zone table++
```

[1] В нашей системе Linux идентификатор процесса записывается в */var/run/named.pid*.

```
; . (type 6, class 0, source Nil)
;       time=0, lastupdate=0, serial=0,
;       refresh=0, retry=0, expire=0, minimum=0
;       ftime=0, xaddrcnt=0, state=0000, pid=0
; . (type 3, class 1, source named.ca)
;       time=0, lastupdate=965723221, serial=0,
;       refresh=0, retry=0, expire=0, minimum=4294967295
;       ftime=965723221, xaddrcnt=0, state=0040, pid=0
; 0.0.127.in-addr.arpa (type 1, class 1, source named.local)
;       time=0, lastupdate=0, serial=1997022700,
;       refresh=0, retry=14400, expire=3600000, minimum=86400
;       ftime=965723221, xaddrcnt=0, state=0041, pid=0
; wrotethebook.com (type 1, class 1, source wrotethebook.com.hosts)
;       time=0, lastupdate=0, serial=2001070501,
;       refresh=0, retry=1800, expire=604800, minimum=900
;       ftime=982967703, xaddrcnt=0, state=0041, pid=0
; 16.172.in-addr.arpa (type 1, class 1, source 172.16.rev)
;       time=0, lastupdate=0, serial=2001071602,
;       refresh=0, retry=1800, expire=604800, minimum=900
;       ftime=982968091, xaddrcnt=0, state=0041, pid=0
;; --zone table--
```

Данный раздел начинается с даты и времени создания образа. Метки в начале и конце таблицы определяют ее границы. Как можно определить по двоеточиям, начинающим каждую строку, все эти строки являются комментариями, предназначенными для передачи информации администратору системы. Ни один из комментариев не является действительной записью базы данных DNS. Исходя из полученных данных можно сказать, что в файле *named.conf* этого сервера фигурируют операторы `zone` для следующих доменов:

. (точка)

Корневой домен, загруженный из исходного файла *named.ca*. Это файл корневых указателей, описанный в главе 8.

0.0.127.in-addr.arpa

Кольцевой домен, загруженный из исходного файла *named.local*.

wrotethebook.com

Домен *wrotethebook.com*, загруженный из исходного файла *wrotethebook.com.hosts*.

16.172.in-addr.arpa

Обратный домен *16.172.in-addr.arpa*, загруженный из исходного файла *172.16.rev*.

Кроме того, для каждой зоны отображаются значения из записи SOA. В приведенном примере запись SOA есть у каждой зоны, за исключением корневой (.).

Таблица зон включает все зоны, для которых сервер является компетентным. Она позволяет определить, откуда сервер получил информацию о зоне

и какие стандартные настройки установлены для зоны записью SOA. Если зона отсутствует или загружена из неверного источника, исправьте оператор `zone` в файле *named.conf*.

Раздел кэша и данных

Второй раздел файла образа – самый объемный. Именно этот раздел содержит всю информацию DNS, известную серверу. Из-за длины раздела здесь мы приводим только выдержку из него:

```
; Note: Cr=(auth,answer,addtnl,cache) tag only shown for non-auth RR's
; Note: NT=milliseconds for any A RR which we've used as a nameserver
; --- Cache & Data ---
$ORIGIN .
.     513482    IN    NS     H.ROOT-SERVERS.NET.     ;Cr=auth
      513482    IN    NS     C.ROOT-SERVERS.NET.     ;Cr=auth
      513482    IN    NS     G.ROOT-SERVERS.NET.     ;Cr=auth
      513482    IN    NS     F.ROOT-SERVERS.NET.     ;Cr=auth
      513482    IN    NS     B.ROOT-SERVERS.NET.     ;Cr=auth
      513482    IN    NS     J.ROOT-SERVERS.NET.     ;Cr=auth
      513482    IN    NS     K.ROOT-SERVERS.NET.     ;Cr=auth
      513482    IN    NS     L.ROOT-SERVERS.NET.     ;Cr=auth
      513482    IN    NS     M.ROOT-SERVERS.NET.     ;Cr=auth
      513482    IN    NS     I.ROOT-SERVERS.NET.     ;Cr=auth
      513482    IN    NS     E.ROOT-SERVERS.NET.     ;Cr=auth
      513482    IN    NS     D.ROOT-SERVERS.NET.     ;Cr=auth
      513482    IN    NS     A.ROOT-SERVERS.NET.     ;Cr=auth
... многие строки удалены ...
$ORIGIN ROOT-SERVERS.NET.
K     599882    IN    A      193.0.14.129     ;NT=9 Cr=answer
A     599882    IN    A      198.41.0.4     ;NT=10 Cr=answer
L     599882    IN    A      198.32.64.12     ;NT=5 Cr=answer
M     599882    IN    A      202.12.27.33     ;NT=15 Cr=answer
B     599882    IN    A      128.9.0.107     ;NT=5 Cr=answer
C     599882    IN    A      192.33.4.12     ;NT=165 Cr=answer
D     599882    IN    A      128.8.10.90     ;NT=12 Cr=answer
E     599882    IN    A      192.203.230.10     ;NT=6 Cr=answer
F     599882    IN    A      192.5.5.241     ;NT=1021 Cr=answer
G     599882    IN    A      192.112.36.4 ;NT=1023 Cr=answer
H     599882    IN    A      128.63.2.53     ;NT=6 Cr=answer
I     599882    IN    A      192.36.148.17     ;NT=7 Cr=answer
J     599882    IN    A      198.41.0.10     ;NT=6 Cr=answer
... многие строки удалены ...
$ORIGIN com.
foobirds 86400 IN RP admin.foobirds.org. hotline.foobirds.org. ;Cl=2
      86400    IN    MX     10 wren.foobirds.org.     ;Cl=2
      86400    IN    MX     20 parrot.foobirds.org.     ;Cl=2
      86400    IN    NS     wren.foobirds.org.     ;Cl=2
      86400    IN    NS     parrot.foobirds.org.     ;Cl=2
      86400    IN    SOA     wren.foobirds.org. admin.wren.foobirds.org. (
               2000020501 21600 1800 604800 900 )     ;Cl=2
```

```
$ORIGIN foobirds.org.
ducks     86400     IN     NS     ruddy.ducks.foobirds.org.     ;Cl=2
          86400     IN     NS     wren.foobirds.org.     ;Cl=2
          86400     IN     NS     bear.mammals.org.     ;Cl=2
news      86400     IN     CNAME     parrot.foobirds.org.     ;Cl=2
robin     86400     IN     RP     admin.foobirds.org. hotline.foobirds.org.     ;Cl=2
          86400     IN     MX     5 wren.foobirds.org.     ;Cl=2
          86400     IN     A     172.16.5.2     ;Cl=2
puffin    86400     IN     RP     admin.foobirds.org. hotline.foobirds.org.     ;Cl=2
          86400     IN     MX     5 wren.foobirds.org.     ;Cl=2
          86400     IN     A     172.16.5.17     ;Cl=2
wren      86400     IN     RP     admin.foobirds.org. hotline.foobirds.org.     ;Cl=2
          86400     IN     A     172.16.5.1     ;Cl=2
parrot    86400     IN     RP     logan.parrot.foobirds.org. logan.foobirds.org.     ;Cl=2
          86400     IN     A     172.16.5.3     ;Cl=2
logan     86400     IN     TXT     "Logan Little (301)555-2021"     ;Cl=2
crow      86400     IN     RP     doris.crow.foobirds.org.foobirds.org. crowRP.foobirds.
org.     ;Cl=2
          86400     IN     A     172.16.5.5     ;Cl=2
localhost    86400     IN     A     127.0.0.1     ;Cl=2
terns     86400     IN     NS     sooty.terns.foobirds.org.     ;Cl=2
          86400     IN     NS     arctic.terns.foobirds.org.     ;Cl=2
www       86400     IN     CNAME     wren.foobirds.org.     ;Cl=2
hotline   86400     IN     TXT     "Support hotline (301)555-2000"     ;Cl=2
bob       86400     IN     CNAME     robin.foobirds.org.     ;Cl=2
redbreast     86400     IN     CNAME     robin.foobirds.org.     ;Cl=2
hawkRP    86400     IN     TXT     "Clark Smart (301)555-2099"     ;Cl=2
kestrel   86400    IN    RP    clark.foobirds.org.foobirds.org. hawkRP.foobirds.org.
;Cl=2
          86400     IN     A     172.16.5.20     ;Cl=2
crowRP    86400     IN     TXT     "Doris Nathan (301)555-2078"     ;Cl=2
kestral   86400     IN     CNAME     kestrel.foobirds.org.     ;Cl=2
hawk     86400    IN    RP    clark.foobirds.org.foobirds.org. hawkRP.foobirds.org.
;Cl=2
          86400     IN     A     172.16.5.4     ;Cl=2
foobirds-net     86400     IN     PTR     0.0.16.172.in-addr.arpa.     ;Cl=2
$ORIGIN terns.foobirds.org.
arctic    86400     IN     A     172.16.30.251     ;Cl=2
sooty     86400     IN     A     172.16.30.250     ;Cl=2
$ORIGIN 172.in-addr.arpa.
16        86400     IN     NS     wren.foobirds.org.     ;Cl=4
          86400     IN     SOA     wren.foobirds.org. admin.wren.foobirds.org. (
         2000021602 21600 1800 604800 900 )     ;Cl=4
$ORIGIN 6.16.172.in-addr.arpa.
1     86400     IN     PTR     arctic.terns.foobirds.org.     ;Cl=4
$ORIGIN 12.16.172.in-addr.arpa.
3     86400     IN     PTR     wren.foobirds.org.     ;Cl=4
$ORIGIN 5.16.172.in-addr.arpa.
20    86400     IN     PTR     kestrel.foobirds.org.     ;Cl=4
4     86400     IN     PTR     hawk.foobirds.org.     ;Cl=4
2     86400     IN     PTR     robin.foobirds.org.     ;Cl=4
```

```
17    86400    IN    PTR    puffin.foobirds.org.    ;Cl=4
5     86400    IN    PTR    crow.foobirds.org.    ;Cl=4
3     86400    IN    PTR    parrot.foobirds.org.    ;Cl=4
$ORIGIN 0.127.in-addr.arpa.
0     86400    IN    NS    localhost.    ;Cl=5
      86400    IN    SOA    localhost. root.localhost. (
         1997022700 28800 14400 3600000 86400 )    ;Cl=5
$ORIGIN 0.0.127.in-addr.arpa.
1     86400    IN    PTR    localhost.    ;Cl=5
```

Вот такая длинная выдержка, несмотря на то что образ был создан вскоре после запуска сервера, а из раздела удалены многие строки. Основную часть данных представляют сведения, загруженные из локальных файлов зон, однако файл образа содержит также серьезный объем кэшированной информации. Крупные блоки кэша своим существованием обязаны информации, полученной из разделов компетенции и дополнительных разделов ответов на запросы. Таким путем в кэш попадает, по меньшей мере, столько же данных, сколько благодаря конкретным ответам на запросы. Это ясно видно из большого числа NS-записей и сопутствующих им A-записей.

Раздел кэша и данных разбит на части инструкциями $ORIGIN. Все прочие строки этого раздела однозначно сопоставимы с записями ресурсов DNS. Однако в конце каждой записи добавляется дополнительная информация в виде комментария. В частности, сервер добавляет три следующих комментария к каждой записи:

`Cl`

Указывает число полей для текущей зоны. Для текущей зоны координат *0.0.127.in-addr.arpa* значение Cl равно 5, а для текущей зоны *wrotethebook.com* Cl имеет значение 2. Корень (.) получает значение Cl, равное 0.

`Nt`

Время отклика для запросов, адресованных указанному серверу имен. Этот комментарий добавляется только к адресным записям. Время отклика позволяет выбирать оптимальный сервер для определенного запроса.

`Cr`

Тег «достоверности» определяет уровень авторитетности источника кэшированной информации. Уровней авторитетности в BIND три:

`auth`

Компетентный ответ.

`answer`

Ответ из некомпетентного источника.

`addtnl`

Запись, полученная из раздела компетенции или дополнительного раздела ответа на запрос.

Значение Cr используется `named` при получении записи, которая уже существует в кэше сервера имен. Если полученная запись имеет более высокий по-

казатель достоверности, чем хранимая, новая запись заменяет кэшированную. Если новая запись имеет более низкий показатель достоверности, сохраняется кэшированная запись. В иерархии значений Cr наивысшую достоверность имеет `auth`, а самую низкую – `addtnl`.

Помимо комментариев в конце записи в разделе кэша и данных файла образа могут встречаться и другие комментарии. Отрицательная кэшированная информация также отображается в файле образа в виде комментариев. В приведенном фрагменте нет соответствующих примеров, а в принципе каждый такой комментарий выглядит как обычная RR-запись, предваренная точкой с запятой. Иначе говоря, отрицательная кэшированная информация представлена закомментированными записями ресурсов. Кроме того, ближе к концу такой записи фигурирует тег NXDOMAIN.

Изучите раздел кэша и данных и убедитесь, что сведения из файлов зон загружены точно так, как ожидалось. Кроме того, используйте раздел для проверки информации, загруженной с удаленного сервера. Локальные данные можно скорректировать самостоятельно. Некорректные данные на удаленном сервере могут потребовать обращения к администратору удаленного домена.

Раздел указателей

Последний раздел файла образа – раздел указателей. Он содержит список корневых серверов имен, загруженный из файла указателей. (О создании и применении файла указателей рассказано в главе 8.) Файл указателей используется только при запуске сервера имен. Когда сервер запущен, он обращается к одному из корневых серверов имен за компетентной информацией о корневых серверах. Именно компетентный список, полученный от корневого сервера, отображается в разделе кэша и данных после оператора `$ORIGIN .`.

Раздел указателей для нашей системы приведен ниже. Обратите внимание, что всем серверам имен раздела указателей присвоены значения Nt. `named` опрашивает все серверы с целью определения времени отклика для каждого и выбора наилучшего сервера для работы.

```
; --- Hints ---
$ORIGIN .
.    3600000    IN    NS    A.ROOT-SERVERS.NET.    ;Cl=0
     3600000    IN    NS    B.ROOT-SERVERS.NET.    ;Cl=0
     3600000    IN    NS    C.ROOT-SERVERS.NET.    ;Cl=0
     3600000    IN    NS    D.ROOT-SERVERS.NET.    ;Cl=0
     3600000    IN    NS    E.ROOT-SERVERS.NET.    ;Cl=0
     3600000    IN    NS    F.ROOT-SERVERS.NET.    ;Cl=0
     3600000    IN    NS    G.ROOT-SERVERS.NET.    ;Cl=0
     3600000    IN    NS    H.ROOT-SERVERS.NET.    ;Cl=0
     3600000    IN    NS    I.ROOT-SERVERS.NET.    ;Cl=0
     3600000    IN    NS    J.ROOT-SERVERS.NET.    ;Cl=0
     3600000    IN    NS    K.ROOT-SERVERS.NET.    ;Cl=0
     3600000    IN    NS    L.ROOT-SERVERS.NET.    ;Cl=0
     3600000    IN    NS    M.ROOT-SERVERS.NET.    ;Cl=0
$ORIGIN ROOT-SERVERS.NET.
```

```
K     3600000     IN     A     193.0.14.129     ;NT=2 Cl=0
L     3600000     IN     A     198.32.64.12     ;NT=5 Cl=0
A     3600000     IN     A     198.41.0.4       ;NT=6 Cl=0
M     3600000     IN     A     202.12.27.33     ;NT=10 Cl=0
B     3600000     IN     A     128.9.0.107      ;NT=134 Cl=0
C     3600000     IN     A     192.33.4.12      ;NT=8 Cl=0
D     3600000     IN     A     128.8.10.90      ;NT=24 Cl=0
E     3600000     IN     A     192.203.230.10   ;NT=2 Cl=0
F     3600000     IN     A     192.5.5.241      ;NT=22 Cl=0
G     3600000     IN     A     192.112.36.4     ;NT=2 Cl=0
H     3600000     IN     A     128.63.2.53      ;NT=22 Cl=0
I     3600000     IN     A     192.36.148.17    ;NT=2 Cl=0
J     3600000     IN     A     198.41.0.10      ;Cl=0
```

Смысл создания образа кэша DNS – в изучении данных, хранимых DNS, и их представления. Исследования компетентных сведений, которые вы передаете серверу посредством файлов зон, даст понимание того, как хранятся эти данные. Изучение прочих данных кэша покажет, как пользователи применяют DNS. Знание того, как обычно используется DNS, позволяет определять момент, когда сценарии использования меняются.

Обнаружив проблемы в файле образа, перезагрузите кэш `named` при помощи команды `ndc reload` (в BIND 9 используйте `rndc reload`) либо при помощи сигнала SIGHUP, как показано ниже:

```
# kill -HUP 'cat /etc/named.pid'
```

Эта команда очищает кэш и повторно загружает действительные записи корневых серверов из файла *named.ca*.

Если известно, какая система ответственна за повреждения кэша, предпишите своей системе игнорировать обновления от проблемной системы – при помощи оператора `server` и параметра `bogus` со значением `yes` (в файле */etc/named.conf*). Оператор `server` перечисляет IP-адреса сервера имен. Установка параметра `bogus` в значение `yes` в операторе `server` сообщает `named`, что информации от указанного сервера доверять нельзя. Например, в предшествующем разделе описана ситуация, когда сервер *24seven.events.wrotethebook.com* (172.16.16.1) служил причиной повреждения кэша, предоставляя записи HINFO в некорректном формате. Следующая запись файла *named.conf* приведет к блокировке ответов от *24seven.events.wrotethebook.com* и таким образом предотвратит повреждение кэша:

```
 server 172.16.16.1 {
    bogus yes;
};
```

Установка параметра `bogus` в значение `yes` – мера временная, она позволяет сохранить работоспособность системы до момента, когда администратор удаленного домена займется диагностированием и исправлением проблемы. Когда ошибки в удаленной системе исправлены, удалите оператор `server` из файла *named.conf*.

dig, альтернатива nslookup

Альтернативой nslookup в области запросов к службе имен является dig. Запросы dig обычно представляют собой однострочные команды, тогда как nslookup обычно используется в диалоговом режиме. При этом команды dig выполняют ту же работу, что и nslookup. Выбор того или иного инструмента – в большой степени просто дело вкуса. Оба работают достойно.

Для примера мы используем dig, чтобы запросить у корневого сервера *b.root-servers.net* NS-записи домена *mit.edu*. Выполните следующую команду:

```
% dig @b.root-servers.net mit.edu ns
```

В данном примере *@b.root-servers.net* – это сервер, к которому обращен запрос. Сервер может обозначаться именем или IP-адресом. При отладке проблемы, связанной с удаленным доменом, указывайте компетентный сервер имен этого домена. В данном примере мы запрашиваем имена серверов домена второго уровня (*mit.edu*), поэтому обращаемся к корневому серверу.

Если сервер не указан явным образом, dig использует локальный сервер имен или сервер имен, упомянутый в файле */etc/resolv.conf*. (Файл *resolv.conf* описан в главе 8.) Кроме того, имя альтернативного файла *resolv.conf* может быть указано в качестве значения переменной среды LOCALRES. Этот альтернативный файл будет использоваться вместо */etc/resolv.conf* в запросах dig. Установка значения LOCALRES влияет только на dig, прочие программы, работающие со службами имен, продолжают использовать */etc/resolv.conf*.

Последний элемент в примере команды – ns. Это тип запроса. Тип запроса – это значение, определяющее тип информации DNS, о которой идет речь в запросе. Это аналог значения, устанавливаемого при помощи команды nslookup set type. В табл. 13.1 перечислены возможные типы запросов dig и даны их расшифровки.

Таблица 13.1. Типы запросов dig

Тип запроса	Поиск записей DNS
a	Адресные
any	Записи любых типов
mx	Записи Mail Exchange (MX)
ns	Записи серверов имен (Name Server)
soa	Записи начала компетенции (Start of Authority)
hinfo	Записи Host Info
axfr	Все записи зоны
txt	Текстовые записи

Обратите внимание, что команда nslookup ls реализована типом запросов dig axfr.

В dig существует также ключ, полезный для определения имени узла по его IP-адресу. Имея только IP-адрес узла, вы можете захотеть определить его имя, поскольку численные адреса более подвержены опечаткам. Работа с именем вместо адреса может облегчить жизнь пользователя. Домен *in-addr.arpa* обеспечивает преобразование адресов в имена, а dig предоставляет простой способ создания запросов по именам домена *in-addr.arpa*. Ключ -x позволяет создавать запросы преобразования адрес–имя и избавляет от необходимости вручную выполнять обращение номеров и добавлять «in-addr.arpa». К примеру, чтобы выполнить поиск имени узла по IP-адресу 18.72.0.3, просто введите:

```
% dig -x 18.72.0.3

; <<>> DiG 2.2 <<>> -x
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3
;; QUERY SECTION:
;;      3.0.72.18.in-addr.arpa, type = ANY, class = IN

;; ANSWER SECTION:
3.0.72.18.in-addr.arpa.  6H IN PTR  BITSY.MIT.EDU.

;; AUTHORITY SECTION:
18.in-addr.arpa.         6H IN NS        W20NS.MIT.EDU.
18.in-addr.arpa.         6H IN NS        BITSY.MIT.EDU.
18.in-addr.arpa.         6H IN NS        STRAWB.MIT.EDU.

;; ADDITIONAL SECTION:
W20NS.MIT.EDU.           6H IN A         18.70.0.160
BITSY.MIT.EDU.           6H IN A         18.72.0.3
STRAWB.MIT.EDU.          6H IN A         18.71.0.151

;; Total query time: 367 msec
;; FROM: wren.foobirds.org to SERVER: default -- 0.0.0.0
;; WHEN: Thu Jul 19 16:00:39 2001
;; MSG SIZE  sent: 40  rcvd: 170
```

Ответ на наш запрос – BITSY.MIT.EDU, однако dig отображает много другой информации. Для целей данного конкретного запроса достаточно только строки ответа[1], но дополнительная информация, отображаемая dig, полезна для обретения понимания формата пакетов ответов DNS и того, откуда появляются различные элементы информации DNS.

Формат сообщения DNS определен в документе RFC 1035, Domain Names – Implementation and Specification (Доменные имена, реализация и спецификация). Данный документ гласит, что сообщение стандартного формата может включать до пяти разделов:

---

[1] Чтобы увидеть только строку ответа на данный запрос, передайте вывод dig команде grep; скажем, так: dig -x 18.72.0.3 | grep PTR.

Заголовок (Header)

Содержит административную информацию о сообщении, включая сведения о том, что содержится в последующих разделах сообщения.

Вопрос (Question)

Определяет суть вопроса, поставленного в запросе. Если раздел вопроса содержится в ответном сообщении, он позволяет определить, на какой вопрос отвечает ответное сообщение.

Ответ (Answer)

Часть ответного сообщения, содержащая ответ на конкретный вопрос, полученный в запросе.

Компетенция (Authority)

Содержит указатели на компетентные серверы домена, фигурирующего в запросе.

Дополнительный раздел (Additional)

Содержит прочие записи ресурсов с дополнительной важной информацией, сопутствующей ответу. Это не ответ на запрос, но эта информация способствует интерпретации или использованию ответа.

Основа вывода команды `dig` представлена в различных разделах ответного пакета DNS. Сведения заголовка в вышеприведенном примере представлены следующим образом:

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3
```

`dig` отображает данные заголовка не в том порядке, в каком они хранятся в заголовке, но вывод команды достаточно легко связать с полями заголовка, описанными в RFC 1035. Ниже описаны различные значения из примера:

`opcode: QUERY`

Указывает, что данный запрос является стандартным.

`status: NOERROR`

Указывает, что в поле RCODE отсутствует код ошибки, то есть поле RCODE содержит значение 0.

`id: 6`

Указывает, что для данного сообщения в качестве идентификатора использовалось число 6.

`flags: qr aa rd ra`

`flags` объединяет все однобитовые поля заголовка. В данном случае это значение покрывает четыре различных поля раздела заголовка и предоставляет сведения о трех других полях. Данная группа флагов указывает, что QR имеет значение 1, то есть мы имеем дело с ответным сообщением. AA имеет значение 1, поскольку ответ поступил от компетентного сервера. RD имеет значение 1, поскольку в запросе присутствовал флаг рекур-

сии. RA имеет значение 1 – рекурсия была доступна на сервере. TC отсутствует, а значит, имеет значение 0, и это означает, что ответ не был усечен. AD и CD также имеют значение 0, поскольку механизмы DNSSEC не задействованы.

```
QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3
```

Эти значения представляют поля заголовка QDCOUNT, ANCOUNT, NSCOUNT и ARCOUNT, определяющие число записей ресурсов в оставшихся разделах сообщения. Можно видеть, что в разделе вопроса одна запись, в разделе ответа одна запись, в разделе компетенции три записи и в дополнительном разделе три записи.

Команда `dig` из примера отображает следующие сведения о запросе:

```
;; QUERY SECTION:
;;      3.0.72.18.in-addr.arpa, type = ANY, class = IN
```

Четко прослеживаются три поля данного запроса. Поле класса имеет значение `IN`, поскольку запрос касается записей Internet. Смысл запроса: поиск любой записи (`type = any`), связанной с именем *3.0.72.18.in-addr.arpa*. Отметим, что программа `dig` выполнила обращение адреса и создала для данного запроса соответствующее обратное доменное имя.

Далее `dig` выводит раздел ответа, компетенции и дополнительный:

```
;; ANSWER SECTION:
3.0.72.18.in-addr.arpa.  6H IN PTR  BITSY.MIT.EDU.

;; AUTHORITY SECTION:
18.in-addr.arpa.         6H IN NS         W20NS.MIT.EDU.
18.in-addr.arpa.         6H IN NS         BITSY.MIT.EDU.
18.in-addr.arpa.         6H IN NS         STRAWB.MIT.EDU.

;; ADDITIONAL SECTION:
W20NS.MIT.EDU.           6H IN A          18.70.0.160
BITSY.MIT.EDU.           6H IN A          18.72.0.3
STRAWB.MIT.EDU.          6H IN A          18.71.0.151
```

Ответ такой, как можно было ожидать: запись PTR для имени *3.0.72.18.in-addr.arpa*. Эта запись сообщает нам, что имя узла для адреса 18.72.0.3 – *bitsy.mit.edu*.

В разделе компетенции перечислены серверы, выступающие в роли компетентных для домена *18.in-addr.arpa*. В этом разделе три записи NS, и каждая содержит имя компетентного сервера. Таким образом, мы знаем, что серверы *w20ns.mit.edu*, *bitsy.mit.edu* и *strawb.mit.edu* являются компетентными для обратного домена *18.in-addr.arpa*.

Дополнительный раздел завершает сообщение и содержит адреса всех компетентных серверов имен. Адреса важны, поскольку если локальному серверу понадобится отправить один из последующих запросов напрямую компетентному серверу имен, он должен знать адреса таких серверов. Адреса в данном случае: 18.70.0.160, 18.72.0.3, 18.71.0.151.

Помимо ответа DNS, `dig` предоставляет сведения о состоянии в трех первых и четырех последних строках вывода. Первая строка повторяет параметры командной строки `dig` (`-x` в данном примере). Вторая строка содержит параметры библиотеки DNS-клиента, а третья указывает, был ли найден ответ на запрос. Последние четыре строки отображают время поиска, имя и адрес сервера, ответившего на запрос, дату получения запроса, а также размер пакетов запроса и ответа. Все эти сведения могут быть полезны при отладке проблем с DNS.

Программа `dig` удобна, поскольку формат сообщения DNS четко прослеживается в ее выводе. `dig` включается в состав Linux, однако присутствует не во всех системах Unix. Не беспокойтесь, если в вашей системе отсутствует `dig`. Для атаки на те же проблемы можно смело использовать `nslookup`. `nslookup` и `dig` одинаково эффективны для тестирования DNS.

Анализ проблем протоколов

Причинами проблем существенно чаще становятся неверные настройки TCP/IP, чем некорректные реализации протоколов TCP/IP. Большинство проблем, с которыми вы столкнетесь, вполне поддаются анализу посредством уже описанных ранее простых инструментов. Но временами может возникать необходимость в анализе взаимодействия протоколов двух систем. В худшем случае может дойти и до побитового анализа пакетов в потоке данных. Задачу облегчают анализаторы протоколов.

Мы воспользуемся инструментом под названием `snoop`. `snoop` поставляется в составе операционной системы Solaris.[1] Во всех примерах фигурирует `snoop`, однако концепции этого раздела применимы и к прочим анализаторам, поскольку большинство анализаторов протоколов функционирует по сути дела одинаково. Анализаторы протоколов позволяют выбрать, или отфильтровать, пакеты для анализа и изучать такие пакеты побайтово. Мы обсудим обе функции.

Анализаторы протоколов просматривают все пакеты сети. Следовательно, нужна только одна система с анализатором в том сегменте сети, для которого выполняется отладка. Одна система Solaris при помощи `snoop` может наблюдать за сетевым трафиком и уведомлять вас о том, что делают (или не делают) другие узлы в настоящий момент. Разумеется, речь идет только о сетях с разделяемой пропускной способностью. Если используется коммутатор Ethernet, будет доступен только трафик в пределах отдельного сегмента. Некоторые коммутаторы имеют специальный порт для наблюдения. В остальных случаях необходимо доставить средство наблюдения в зону проблемы.

[1] Пользователи Linux могут использовать `tcpdump`. Эта программа подобна `snoop`.

Фильтры для пакетов

snoop читает все пакеты сегмента Ethernet. С этой целью snoop переводит интерфейс Ethernet в *беспорядочный режим* (promiscuous mode).[1] В нормальной ситуации интерфейс Ethernet передает протоколам более высокого уровня только пакеты, предназначенные локальному узлу. В беспорядочном режиме принимаются и передаются все пакеты, что и позволяет snoop просматривать и выбирать для анализа пакеты, исходя из фильтров, определенных пользователем. Фильтры позволяют отбирать пакеты, адресованные определенному узлу, протоколу, порту или произвольному их сочетанию либо исходящие от определенного узла, протокола, порта или произвольного их сочетания. В качестве примера рассмотрим очень простой фильтр snoop. Следующая команда snoop приводит к отображению всех пакетов из обмена узлов *crab* и *rodent*:

```
# snoop host crab and host rodent
Using device /dev/le (promiscuous mode)
rodent.wrotethebook.com -> crab.wrotethebook.com ICMP Echo request
crab.wrotethebook.com -> rodent.wrotethebook.com ICMP Echo reply
rodent.wrotethebook.com -> crab.wrotethebook.com RLOGIN C port=1023
crab.wrotethebook.com -> rodent.wrotethebook.com RLOGIN R port=1023
^C
```

Фильтр «host crab and host rodent» отбирает только пакеты, исходящие от узла *rodent* и адресованные узлу *crab*, и пакеты, исходящие от *crab* и адресованные *rodent*. Фильтр состоит из набора примитивов, связанных с ними имен узлов, протоколов и номеров портов. Примитивы изменяются и комбинируются при помощи операторов and, or и not. Фильтр может отсутствовать; в этом случае snoop отображает все пакеты сети.

В табл. 13.2 описаны примитивы фильтров snoop. Существует ряд дополнительных примитивов и варианты примитивов, дублирующие функциональность, но в таблицу включены только наиболее востребованные. Дополнительная информация содержится на страницах руководства (man) по snoop.

Совместное использование примитивов и операторов and и or позволяет создавать сложные фильтры. Однако фильтры, как правило, просты. Фильтр для выявления обмена между парой узлов – вероятно, наиболее распространенный. Дополнительно можно ограничить полученные данные определенным протоколом, но часто неизвестно, какой из протоколов приводит к сбоям. Если пользователь встречается с проблемой при работе с ftp или telnet, это еще не повод обвинять соответствующие протоколы. Часто анализ следует начинать с перехвата всех пакетов и последующего сужения области поиска лишь после того, как результаты тестов укажут на более конкретную проблему.

---

[1] Это возможно, только если интерфейс поддерживает работу в беспорядочном режиме, что верно не для всех интерфейсов.

Таблица 13.2. Примитивы выражений

Примитив	Отбирает пакеты
dst host | net | port *destination*	Адресованные целевому узлу (host), сети (net) или на порт (port)
src host | net | port *source*	Исходящие от узла (host), сети (net) или порта (port)
host *destination*	Исходящие или адресованные узлу (host)
net *destination*	Исходящие или адресованные сети
port *destination*	Исходящие или адресованные на порт
ether *address*	Исходящие или адресованные на адрес Ethernet
protocol	С типом протокола (icmp, udp или tcp)

Изменение вывода анализатора

Пример из предшествующего раздела показывает, что `snoop` выводит единственную строку-резюме для каждого из полученных пакетов. В каждой строке содержатся адреса источника и получателя, а также используемый протокол (ICMP и RLOGIN в нашем примере). Строки пакетов ICMP содержат сведения о типах пакетов (Echo request и Echo reply в нашем примере). Строки пакетов прикладных протоколов содержат номер порта источника и первые 20 символов данных пакета.

Этой сводки достаточно, чтобы понять, как обмениваются пакетами узлы и какие потенциальные проблемы могут возникать. Однако диагностирование проблем протоколов требует более подробных сведений о каждом пакете. Управление составом отображаемой информации реализуется ключами `snoop`. Чтобы вывести данные пакета, воспользуйтесь ключом `-x`. Ключ предписывает отображать полное содержимое пакета в шестнадцатеричном формате или ASCII. В большинстве случаев нет необходимости просматривать пакеты целиком – достаточно заголовков, чтобы продиагностировать проблему протокола. Ключ `-v` предписывает отображать заголовки – подробно и в удобном для чтения формате. Для каждого пакета отображается большое число строк, поэтому использовать ключ `-v` следует только по необходимости.

Следующий пример содержит пакет ICMP Echo Request, вывод которого предписан ключом `-v`. Пакет этого типа присутствовал в виде строки-резюме в предшествующем примере.

```
# snoop -v host crab and host minasi
Using device /dev/le (promiscuous mode)
ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 3 arrived at 16:56:57.90
ETHER:  Packet size = 98 bytes
ETHER:  Destination = 8:0:20:22:fd:51, Sun
ETHER:  Source      = 0:0:c0:9a:d0:db, Western Digital
ETHER:  Ethertype = 0800 (IP)
```

```
ETHER:
IP:   ----- IP Header -----
IP:
IP:   Version = 4
IP:   Header length = 20 bytes
IP:   Type of service = 0x00
IP:         xxx. .... = 0 (precedence)
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:   Total length = 84 bytes
IP:   Identification = 3049
IP:   Flags = 0x0
IP:         .0.. .... = may fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 64 seconds/hops
IP:   Protocol = 1 (ICMP)
IP:   Header checksum = fde0
IP:   Source address = 172.16.55.106, minasi.wrotethebook.com
IP:   Destination address = 172.16.12.1, crab.wrotethebook.com
IP:   No options
IP:
ICMP:  ----- ICMP Header -----
ICMP:
ICMP:  Type = 8 (Echo request)
ICMP:  Code = 0
ICMP:  Checksum = ac54 ICMP:
```

Подробное форматирование `snoop` связывает байты, полученные из сети, со структурой заголовка. В поисках более подробной информации о различных полях заголовка обращайтесь к главе 1 и приложению G.

Пример анализа протокола

Данный пример повторяет случай из реальной жизни, когда проблема решилась при помощи анализа протокола. Суть проблемы: время от времени происходил сбой в работе `ftp` со следующим сообщением об ошибке:

```
netout: Option not supported by protocol 421 Service not available, remote server
has closed connection
```

О проблеме заявил только один пользователь, и он сталкивался с ней только при передаче больших файлов с рабочей станции на центральную машину по нашей магистральной сети.

Мы взяли файл данных пользователя и смогли воспроизвести проблему с других рабочих станций. При этом проблема возникала только при передаче файла на ту же центральную систему по магистральной сети. Рисунок 13.4 наглядно отражает тесты, которые мы выполнили для воспроизведения проблемы.

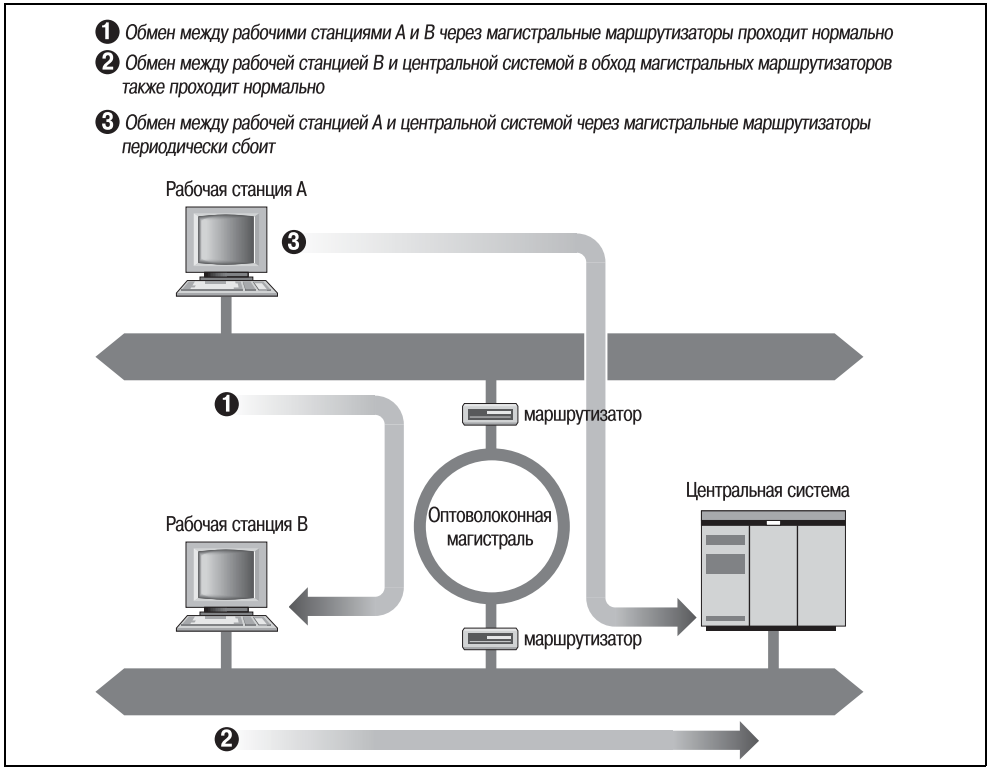


Рис. 13.4. *Резюме теста FTP*

Мы уведомили о проблеме всех пользователей. В ответ мы получили сообщения, что и другие пользователи сталкивались с проблемой, но опять же – только при передаче данных на центральную систему и только при передаче по магистрали. Другие пользователи не сообщали о проблеме, поскольку редко сталкивались с ней. Эти дополнительные сообщения свидетельствовали в пользу того, что проблема не связана с последними изменениями в сети.

Поскольку мы воспроизвели проблему на других системах, вероятнее всего, ее причина не в неверных настройках системы пользователя. Кроме того, сбой `ftp` происходил только в случаях, когда взаимодействовали маршрутизаторы магистральной сети и центральная система. Поэтому мы сосредоточили внимание на этих машинах. Мы проверили таблицы маршрутизации и таблицы ARP, а также выполнили прозвонку центральной системы и маршрутизатораов. И не столкнулись ни с какими проблемами.

На основе предварительных данных можно было предположить, что сбой `ftp` происходил в результате некой проблемы взаимодействия маршрутизаторов определенной фирмы и центрального компьютера. Мы сделали такое предположение потому, что передача данных устойчиво прерывалась только при взаимодействии этих двух видов систем, но никогда – в других ситуациях.

Если неверны настройки маршрутизатора или центральной системы, сбой должен возникать и при передаче данных на другие узлы. Если проблема связана со случайными сбоями физического характера, она должна возникать время от времени и не зависеть от того, какие узлы участвуют в процессе. Проблема же, напротив, поддавалась прогнозированию и возникала только при взаимодействии машин определенных типов. Возможно, дело было в несовместимости реализаций TCP/IP этих систем.

Поэтому мы воспользовались `snoop` и перехватили заголовки TCP/IP в ходе нескольких сеансов тестирования `ftp`. Изучение результатов показало, что во всех сеансах, заканчивающихся сообщением об ошибке «netout», ближе к концу сеанса присутствовал пакет ICMP Parameter Error – обычно за 50 пакетов до окончания передачи. Ни в одном успешном сеансе передачи такого пакета не было. Обратите внимание, что ошибка возникала вовсе *не на последнем* пакете потока данных, как можно было бы предположить. Часто бывает так, что после обнаружения ошибки поток данных остается открытым какое-то время до разрыва соединения. Не следует считать, что ошибка всегда присутствует в конце потока данных.

Вот заголовки из ключевых пакетов. Сначала IP-заголовок пакета от магистрального маршрутизатора, который привел к созданию сообщения об ошибке на центральной системе:

```
ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 1 arrived at 16:56:36.39
ETHER:  Packet size = 60 bytes
ETHER:  Destination = 8:0:25:30:6:51, CDC
ETHER:  Source      = 0:0:93:e0:a0:bf, Proteon
ETHER:  Ethertype = 0800 (IP)
ETHER:
IP:   ----- IP Header -----
IP:
IP:   Version = 4
IP:   Header length = 20 bytes
IP:   Type of service = 0x00
IP:         xxx. .... = 0 (precedence)
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:   Total length = 552 bytes
IP:   Identification = 8a22
IP:   Flags = 0x0
IP:         .0.. .... = may fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 57 seconds/hops
IP:   Protocol = 6 (TCP)
IP:   Header checksum = ffff
IP:   Source address = 172.16.55.106, fs.wrotethebook.com
IP:   Destination address = 172.16.51.252, bnos.wrotethebook.com
IP:   No options IP:
```

А вот пакет ICMP Parameter Error, отправленный центральной системой в ответ:

```
ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 3 arrived at 16:56:57.90
ETHER:  Packet size = 98 bytes
ETHER:  Destination = 0:0:93:e0:a0:bf, Proteon
ETHER:  Source      = 8:0:25:30:6:51, CDC
ETHER:  Ethertype = 0800 (IP)
ETHER:
IP:   ----- IP Header -----
IP:
IP:   Version = 4
IP:   Header length = 20 bytes
IP:   Type of service = 0x00
IP:         xxx. .... = 0 (precedence)
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:   Total length = 56 bytes
IP:   Identification = 000c
IP:   Flags = 0x0
IP:         .0.. .... = may fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 59 seconds/hops
IP:   Protocol = 1 (ICMP)
IP:   Header checksum = 8a0b
IP:   Source address = 172.16.51.252, bnos.wrotethebook.com
IP:   Destination address = 172.16.55.106, fs.wrotethebook.com
IP:   No options
IP:
ICMP:  ----- ICMP Header -----
ICMP:
ICMP:  Type = 12 (Parameter problem)
ICMP:  Code = 0
ICMP:  Checksum = 0d9f ICMP:  Pointer = 10
```

Каждый из заголовков поделен на биты и связан с соответствующими полями заголовка TCP/IP. Из этого подробного анализа каждого пакета мы можем видеть, что маршрутизатор передал параметр IP Header Checksum (контрольная сумма заголовка IP) со значением 0xffff, и центральной системе контрольная сумма не понравилась. Нам известно, что центральная система среагировала именно на этот параметр, поскольку она вернула ошибку ICMP Parameter Error и указатель (Pointer) со значением 10. Parameter Error указывает на некорректность только что полученных данных, а Pointer указывает на данные, которые система полагает ошибочными. Десятый байт в заголовке IP-пакета маршрутизатора – поле IP Header Checksum. Поле данных сообщения об ошибке ICMP содержит заголовок, который считается

ошибочным. При просмотре этих данных мы заметили, что, возвращая заголовок, центральная система «скорректировала» поле контрольной суммы и записала в него значение 0000. Очевидно, центральная система не согласна с вычисленной маршрутизатором контрольной суммой.

Время от времени ошибки с вычислением контрольной суммы происходят. Они могут быть вызваны проблемами передачи и необходимы для выявления подобного рода проблем. В каждом семействе протоколов существует механизм для восстановления после ошибок в контрольной сумме. Как они могут обрабатываться в TCP/IP?

Чтобы определить действие протокола, корректное в данной ситуации, мы обратились к авторитетным источникам – документам RFC. RFC 791, *Internet Protocol*, снабдил нас информацией о вычислении контрольной суммы, но лучшим источником для данной конкретной проблемы стал документ RFC 1122, *Requirements for Internet Hosts – Communication Layers* (Требования к узлам сети Интернет – уровни передачи данных), за авторством Р. Брейдена (R. Braden). Этот документ содержит две отсылки к действиям, которые должны быть предприняты. Следующие выдержки взяты со страницы 29 документа RFC 1122:

> В нижеследующем тексте в определенных случаях в отношении полученной дейтаграммы используется выражение «молча удалить». Это означает, что дейтаграмма удаляется без дополнительной обработки, а узел не посылает какие-либо ICMP-сообщения об ошибках (см. раздел 3.2.2) в результате...
>
> ...Узел ОБЯЗАН проверять правильность контрольной суммы заголовка IP для каждой полученной дейтаграммы и молча удалять все дейтаграммы с неверными контрольными суммами.

Следовательно, получив пакет с неверной контрольной суммой, система не должна ничего с ним делать. Пакет следует удалить, а затем ожидать поступления следующего пакета. Система не должна отвечать сообщением об ошибке. Система не может ответить на неверную контрольную сумму IP-заголовка, поскольку не может определить, откуда в действительности поступил пакет. Если под сомнением контрольная сумма заголовка, как определить, что верны адреса в заголовке? А если нет уверенности относительно источника пакета, как можно на него ответить?

IP полагается на протоколы верхних уровней в восстановлении после таких ошибок. Если используется TCP (как было в данном случае), источник TCP в конечном итоге замечает, что получатель не подтвердил доставку сегмента, и посылает сегмент заново. Если используется UDP, приложение-источник отвечает за восстановление после ошибки. Ни в том ни в другом случае восстановление не зависит от сообщения об ошибке, генерируемого получателем.

Итак, получив неверную контрольную сумму, центральная система должна была просто удалить испорченный пакет. Мы уведомили о проблеме разработчиков, и, следует отметить, они прислали нам поправку к программному обеспечению в пределах двух недель. Более того, поправка замечательно работала!

Не все проблемы разрешаются так же прямолинейно. Однако методика анализа не зависит от того, какова проблема.

Резюме

В каждой сети возникают проблемы. В этой главе описаны инструменты и методы, способствующие преодолению этих проблем, а также вопросы планирования и наблюдения, позволяющие проблем избежать. Решение часто оказывается очевидным, если можно получить достаточный объем информации о проблеме. Unix предоставляет ряд системных программных инструментов, облегчающих сбор информации о настройках системы, адресации, маршрутизации, службе имен, а также прочих жизненно важных сетевых элементах. Держите инструменты под рукой и научитесь пользоваться имя до того, как наступит кризис.

Поиск неисправностей – непрерывный процесс. Эта книга – начало другого непрерывного процесса – учебы. По мере изучения своей системы и сети вы обнаружите, что мир сетей гораздо богаче, чем способна описать отдельно взятая книга. Эта книга была вашей стартовой площадкой – и помогла подключить вашу систему к сети. Теперь, когда ваша система встала в строй, используйте ее для расширения своего кругозора.

A

Инструментарий PPP

Данное приложение представляет собой справочник по программам `dip`, `pppd` и `chat`. Эти инструменты используются для создания удаленных IP-соединений по протоколу точка-точка (Point-to-Point Protocol, PPP). `dip` и `chat` являются языками сценариев. Самая сложная задача в настройке PPP-соединения – это создание сценария для инициализации модема, подключения, регистрации и настройки удаленного сервера. Учебная информация и примеры по всем трем программам содержатся в главе 6. Приложение выполняет функцию справочника.

IP-соединения по коммутируемым линиям

`dip` – это язык сценариев, облегчающий создание SLIP- и PPP-соединений.[1] Синтаксис команды `dip`:

```
dip [ключи] [файл сценария]
```

Команда `dip` выполняется с набором ключей, с указанием имени файла либо с тем и другим. Если указано имя файла сценария, `dip` выполняет сохраненные в этом файле команды для создания соединения точка-точка. Примеры сценариев для `dip` приведены в главе 6. Допустимые для файлов сценариев *ключи*:

`-v`

`dip` работает в режиме подробной диагностики: отображаются каждая строка файла сценария, по мере его выполнения, а также подробные сообщения о состоянии.

---

[1] Протокол Serial Line IP (SLIP) появился раньше PPP. Сегодня большинство последовательных соединений работают по протоколу PPP, на который и сделан упор в данном приложении.

`-m mtu`

Устанавливает значение MTU (maximum transmission unit, максимальная длина передаваемого блока) равным количеству байтов, определенному параметром *mtu*. По умолчанию MTU имеет значение 296 байт.

`-p proto`

Выбирает протокол последовательного соединения. Допустимые значения параметра *proto*: SLIP, CSLIP, PPP и TERM.

Прочие ключи командной строки `dip`:

`-k`

Принудительно завершает работу последнего созданного процесса `dip`. Этот ключ работает только для собственных процессов пользователя. Исключением является пользователь *root*.

`-l device`

Уточняет, что процесс, подлежащий принудительному завершению, блокирует указанное устройство. Используется только совместно с ключом `-k`.

`-i [username]`

При организации PPP-сервера `dip` работает в качестве интерпретатора команд, запускаемого при регистрации. Эквивалентом `dip -i` является команда `diplogin`. Команды являются взаимозаменяемыми, но чаще используется `diplogin`. `diplogin` указывается в поле интерпретатора команд, запускаемого при регистрации, – в записи */etc/passwd* каждого PPP-клиента. Интерпретатор запускается программой `login`. Для извлечения дополнительной информации из файла */etc/diphosts* используется регистрационное имя пользователя, указанное в файле */etc/passwd,* либо необязательное имя пользователя (*username*), если оно указано в ключе команды `diplogin`. Инструкции и примеры, связанные с созданием PPP-сервера, содержатся в главе 6.

`-a`

Запрашивает имя пользователя и пароль. Ключ `-a` может использоваться только совместно с ключом `-i`. Эквивалентом `dip -i -a` является команда `diplogini`. `diplogini` используется как интерпретатор команд регистрации в файле */etc/passwd* и запускается программой `login`.

`-t`

`dip` работает в режиме тестирования, позволяющем вводить отдельные команды языка сценариев прямо с клавиатуры. Ключ `-t` часто используется в сочетании с ключом `-v`, что позволяет более точно оценивать результаты выполнения каждой команды. Как показано в главе 6, этот ключ используется для отладки сценариев `dip`.

Команды `diplogin` и `diplogini` применяются только на серверах, но не в файлах сценариев. Файлы сценариев используются клиентами PPP при настройке `dip` для коммутируемого подключения к удаленному серверу. Файл сценария содержит инструкции, решающие данную задачу.

Файл сценария dip

Файл сценария состоит из комментариев, меток, переменных и команд. Строка, которая начинается символом решетки (#), является комментарием. Метка – это строка, содержащая набор символов, завершаемый двоеточием. Метки используются для разделения сценария на отдельные процедуры. Например, раздел сценария, осуществляющий коммутируемое подключение к удаленному узлу, может начинаться с такой метки:

```
Dial-in:
```

Переменная служит для хранения значений. Имя переменной – это строка, которая начинается символом доллара ($). Можно, например, создать переменную для хранения счетчика цикла и дать ей имя `$loopcntr`. Возможность создания пользовательских переменных применяется редко. В большинстве сценариев используются специальные переменные `dip`. Специальные переменные и их значения приведены в табл. А.1.

Таблица А.1. Специальные переменные dip

Переменная	Хранимое значение
`$errlvl`	Код завершения последней выполненной команды
`$locip`	IP-адрес локального узла
`$local`	Полное доменное имя локального узла
`$rmtip`	IP-адрес удаленного узла
`$remote`	Полное доменное имя удаленного узла
`$mtu`	Значение MTU в байтах
`$modem`	Тип модема; в настоящее время только HAYES
`$port`	Имя последовательного устройства, например `cua0`
`$speed`	Скорость передачи порта

Последний элемент файла сценария – перечень команд. Команд в языке сценариев довольно много, и поскольку настоящее приложение является справочным, мы рассмотрим их все. При этом в большинстве сценариев используется лишь ограниченное подмножество команд. Примеры рабочих сценариев `dip` приведены в главе 6 и в конце данного раздела. Команды, доступные в `dip`:

`beep [`*n*`]`

Указывает системе уведомить пользователя звуковым сигналом. Повторить *n* раз.

`bootp`

Предписывает системе использовать протокол BOOTP для получения локального и удаленного IP-адреса. Команда применима только к протоколу

SLIP. В PPP существует специальный протокол для присвоения адресов. В SLIP дела обстоят иначе. Как правило, адреса SLIP статически назначаются в теле сценария. Однако в некоторых серверах, использующих SLIP, появились механизмы динамического присвоения адресов. Чаще всего сервер отображает адрес в виде открытого текста сразу после того, как установлено соединение. Для получения адреса от SLIP-сервера такого типа воспользуйтесь командой `get $locip remote`. Другие типы SLIP-серверов требуют выполнения определенных команд для отображения адреса. Используйте необходимую команду в сценарии, а затем выполните команду `get`. Наконец, в некоторых SLIP-серверах для распределения адресов применяется протокол BOOTP. Используйте команду `bootp` в сценарии, чтобы включить работу с BOOTP, если того требует SLIP-сервер.

`break`

Посылает символ BREAK, который некоторые из серверов интерпретируют в качестве символа внимания.

`chatkey` *`keyword code`*

Связывает ответ модема (ключевое слово *`keyword`*) с числовым кодом (*`code`*). Предопределенные коды:

```
0     OK
1     CONNECT
2     ERROR
3     BUSY
4     NO CARRIER
5     NO DIAL TONE
```

`config [interface|routing] [pre|up|down|post]` *`arguments...`*

Изменяет свойства интерфейса (`interface`) или таблицу маршрутизации (`routing`) до того, как создано соединение (`pre`), когда создано соединение (`up`), при разрыве соединения (`down`) либо после закрытия соединения (`post`). Следующая команда:

```
config up routing add canary gw ibis
```

добавляет маршрут к *canary* через шлюз *ibis*, если существует соединение. Изменение таблицы маршрутизации или свойств интерфейса обычными пользователями представляет серьезную опасность. Команда `config` отключена в коде `dip`, для ее применения требуется повторная компиляция пакета.

`databits 7|8`

Устанавливает число битов данных в 7 или 8. Для каналов PPP и SLIP рекомендуется значение 8.

`dec` *`$variable`* `[`*`value`*`]`

Уменьшает значение переменной *`$variable`* на число *`value`*. По умолчанию число *`value`* равно 1.

`default`

Предписывает использовать соединение PPP в качестве маршрута по умолчанию.

`dial` *`phonenumber`* [*`timeout`*]

Набирает номер *`phonenumber`*. Если удаленный модем не ответил в течение *`timeout`* секунд, соединение разрывается. `$errlvl` получает численное значение, основанное на ключевом слове, возвращаемом локальным модемом. Отображение ключевых слов в коды устанавливается посредством команды `chatkey`.

`echo on|off`

Включает или отключает отображение команд модема.

`exit` [*`n`*]

Завершает работу сценария; необязательное значение *`n`* является кодом завершения. Очищает буфер ввода.

`get` *`$variable`* [`ask | remote` [*`timeout`*]] *`value`*

Присваивает переменной *`$variable`* значение *`value`*, если не указаны параметры `ask` или `remote`. В случае параметра `ask` значение переменной запрашивается у пользователя. Параметр `remote` предписывает получить значение от удаленного компьютера; необязательный параметр *`timeout`* определяет число секунд ожидания ответа удаленной системы.

`goto` *`label`*

Предписывает выполнить переход к разделу сценария, отмеченному меткой *`label`*.

`help`

Перечисляет команды, доступные в сценариях `dip`.

`if` *`expr`* `goto` *`label`*

Условный оператор, осуществляющий переход к разделу сценария, отмеченному меткой *`label`*, если выражение *`expr`* истинно. Выражение должно содержать сравнение значения переменной с константой посредством одного из операторов: `==` (равенство), `!=` (неравенство), `<` (меньше), `>` (больше), `<=` (меньше либо равно), `>=` (больше либо равно).

`inc` *`$variable`* [*`value`*]

Увеличивает значение переменной *`$variable`* на число *`value`*. По умолчанию число *`value`* равно 1.

`init` *`command`*

Определяет строку команд инициализации модема. По умолчанию — `ATE0 Q0 V1 X1`.

`mode SLIP|CSLIP|PPP|TERM`

Указывает последовательный протокол. По умолчанию установлен режим `SLIP`, поэтому необходимо выполнить команду с аргументом `PPP`.

`modem` *`type`*

Указывает тип модема. Про данную команду можно смело забыть: единственное допустимое значение (`HAYES`) является значением по умолчанию.

`netmask` *`mask`*

Устанавливает адресную маску.

`parity E|O|N`

Устанавливает контроль четности в значения `even` (`E`) – четный, `odd` (`O`) – нечетный или `no` (`N`). Для каналов SLIP и PPP рекомендуется значение `N`, то есть отсутствие контроля четности.

`password`

Запрашивает у пользователя пароль.

`proxyarp`

Создает в таблице ARP локального узла запись для удаленной системы.

`print` *`$variable`*

Отображает значение переменной *`$variable`*.

`psend` *`command`*

Выполняет команду *`command`* в сеансе интерпретатора команд по умолчанию и направляет ее вывод на последовательное устройство. Команда выполняется с действительным идентификатором пользователя (UID).

`port` *`device`*

Указывает последовательное устройство, такое как cua0, с которым связан модем.

`quit`

Завершает работу сценария с ненулевым кодом, разрывая соединение.

`reset`

Выполняет инициализацию модема.

`send` *`string`*

Передает строку *`string`* на последовательное устройство.

`shell` *`command`*

Выполняет команду *`command`* в сеансе интерпретатора команд по умолчанию. Команда выполняется с действительным идентификатором пользователя (UID).

`skey [`*`timeout`*`]`

В течение *`timeout`* секунд ожидает S/Key-запрос от удаленного терминального сервера, предлагает пользователю ввести секретный ключ, генерирует и передает ответ. Если интервал ожидания истек, `$errlvl` получает значение 1; в противном случае – значение 0. Поддержка S/Key должна быть включена при компиляции `dip`.

`sleep` *time*

Задержка выполнения сценария на *time* секунд.

`speed` *bits-per-second*

Устанавливает скорость порта. По умолчанию – 38400.

`stopbits 1|2`

Устанавливает число стоп-битов (1 или 2). Включает режим терминала. В режиме терминала клавиатурный ввод передается прямо на последовательное устройство.

`timeout` *time*

Устанавливает интервал бездействия на линии (в секундах), по истечении которого происходит разрыв связи.

`wait` *text* [*timeout*]

Предписывает ожидать получения от удаленной системы текста *text* в течение *timeout* секунд. Если время не указано, ожидание длится бесконечно.

В следующем разделе мы применим некоторые из описанных команд для создания рабочего сценария.

Пример dip-сценария

Приведенный сценарий основан на примере для PPP из главы 6. Для повышения надежности сценария добавлены метки и команды обнаружения ошибок.

```
# Выполняем настройку параметров
setup:
# Запрашиваем адреса у PPP
get $local 0.0.0.0
# Выбираем порт
port cua1
# Устанавливаем скорость порта
speed 57600
# Создаем счетчик цикла
get $loopcntr 0

# Коммутация с удаленным сервером
dialin:
# Аппаратная инициализация модема и очистка буфера ввода
reset
flush
# Набираем номер PPP-сервера и проверяем ответ модема
dial *70,301-555-1234
# Если занято (BUSY), набрать номер снова
if $errlvl == 3 goto redial
# В случае иной ошибки прервать выполнение
if $errlvl != 1 goto dial-error
# Иначе – инициализировать счетчик цикла
get $loopcntr 0
```

```
# Делаем паузу в две секунды, чтобы сервер успел подготовиться
sleep 2

# Регистрация на удаленном сервере
login:
# Передаем символ возврата каретки, чтобы «разбудить» сервер
send \r
# Ожидаем приглашения Username> и передаем имя пользователя
wait name> 20
if $errlvl != 0 goto try-again
send kristin\r
# Ожидаем приглашения Password> и передаем пароль
wait word> 10
if $errlvl != 0 goto server-failure
password
# Ожидаем приглашения командной строки сервера PPP
wait > 20
if $errlvl != 0 goto server-failure
# Передаем команду, регламентированную сервером PPP
send ppp enabled\r

# Успех! Мы подключились.
connected:
# Переводим интерфейс в режим PPP
mode PPP
# Завершаем работу сценария
exit

# Подпрограммы обработки ошибок

# Набирать номер до трех раз. Между попытками делать паузы по 5 секунд
redial:
inc $loopcntr
if $loopcntr > 3 goto busy-failure
sleep 5
goto dialin

# Повторная попытка «разбудить» сервер
try-again:
inc $loopcntr
if $loopcntr > 1 goto server-failure
goto login


dial-error:
print Не удалось подключиться к $remote.
quit

server-failure:
print Сервер $remote не ответил.
quit

busy-failure:
print Линия сервера $remote занята. Перезвоните позже.
quit
```

Данный сценарий представляет собой реалистичный пример использования наиболее востребованных команд dip. Тем не менее есть шансы столкнуться с особыми проблемами при написании подобных сценариев. В таком случае поможет широкий выбор доступных в dip команд. Если посредством dip задачу решить не удается, обратитесь к инструменту expect. Полное описание языка сценариев expect дается в книге Дона Либиса (Don Libes) «Exploring Expect» (Изучаем Expect), O'Reilly & Associates.

Демон PPP

Демон PPP (pppd) – это свободно доступная реализация протокола точка-точка (Point-to-Point Protocol, PPP), работающая на многих системах Unix. Примеры настройки и применения pppd приводятся в главе 6. Команда pppd подчиняется следующему синтаксису:

```
pppd [устройство] [скорость] [ключи]
```

устройство – это имя последовательного порта, с которым работает протокол PPP, а *скорость* – его скорость передачи данных, указанная в битах в секунду. Сложность команды проистекает не из этих простых параметров, но из большого числа опций, которые она поддерживает. На деле опций так много, что их часто хранят в отдельном файле. Совместно с pppd используются три файла параметров: */etc/ppp/options*, предназначенный для установки общесистемных параметров pppd; файл *~/.ppprc*, в котором определяются параметры pppd для конкретного пользователя; и файл */etc/ppp/options.device*, отвечающий за установку параметров для последовательного устройства (например, параметры настройки устройства cua0 содержатся в файле */etc/ppp/options.cua0)*. Порядок старшинства для параметров следующий: параметры из файла */etc/ppp/options.device* имеют наивысший приоритет, за ними следуют параметры командной строки, параметры из файла *~/.ppprc* и, наконец, параметры из файла */etc/ppp/options*. Некоторые из параметров, а именно связанные с безопасностью системы, будучи установлены в файле */etc/ppp/options*, не могут переопределяться пользователем в командной строке или в файле *~/.ppprc*. Системный администратор может изменять любые установленные пользователем параметры в файле */etc/ppp/options.device*.

Приводимый ниже перечень содержит все параметры настройки pppd, за исключением тех, что не относятся к TCP/IP:

локальный_IP-адрес:удаленный_IP-адрес

Определяет статические IP-адреса для локальной и удаленной машины. Любой из адресов может быть опущен. Например, 172.16.25.3: определяет только локальный адрес, а конструкция :172.16.25.12 определяет только удаленный адрес. По умолчанию в качестве локального IP-адреса используется адрес, связанный с именем узла локальной системы.

active-filter *filter-expression*

Создает пакетный фильтр, определяющий, какие из пакетов представляют активный трафик соединения. Пакеты, соответствующие фильтру

filter-expression, обнуляют таймер бездействия или приводят к инициализации канала, если он находится в режиме соединения по требованию. Компиляцию ядра и pppd следует производить с символом PPP_ FILTER.

allow-ip *address*

Разрешает работу без аутентификации системам, соответствующим указанному адресу, который определяет отдельный узел или целую сеть.

asyncmap *map*

Определяет управляющие ASCII-символы, передаваемые в виде 2-байтовых управляющих escape-последовательностей. Первые 32 символа ASCII являются управляющими. Карта *map* – это 32-битное шестнадцатеричное число, каждый бит которого представляет управляющий символ. Бит 0 (00000001) представляет символ 0x00; бит 31 (80000000) представляет символ 0x1f. Если бит включен в карту, соответствующий биту символ должен передаваться в виде управляющей escape-последовательности. В отсутствие инструкции asyncmap все управляющие символы передаются в виде управляющих escape-последовательностей.

auth

Требует применения протокола аутентификации. Протоколы аутентификации CHAP и PAP описаны в главе 6.

bsdcomp *receive*, *transmit*

Включает механизм сжатия пакетов BSD-Compress. Максимальная длина кодового слова (в битах), применяемого для сжатия принимаемых узлом пакетов, определяется параметром *receive*. Максимальная длина кодового слова (в битах), применяемого для сжатия передаваемых узлом пакетов, определяется параметром *transmit*. Допустимая длина кодового слова лежит в интервале от 9 до 15 бит. Чтобы отключить сжатие принимаемых или передаваемых пакетов, следует установить в значение 0 параметр *receive* или *transmit* соответственно.

call *name*

Считывает настройки из файла */etc/ppp/peers/name*.

cdtrcts

Указывает pppd, что в модеме применяется нестандартный аппаратный контроль передачи, основанный на сигналах DTR и CTS.

chap-interval *n*

Предписывает использовать протокол CHAP (*Challenge Handshake Authentication Protocol*) для повторной идентификации удаленной системы с интервалом *n* секунд.

chap-max-challenge *n*

Предписывает посылать вызовы CHAP до *n* раз, пока удаленная система не ответит. По умолчанию – 10 раз.

`chap-restart` *n*

Предписывает выждать *n* секунд перед повторным вызовом CHAP, если удаленная система не отвечает. По умолчанию – 3 секунды.

`connect` *script*

Вызывает сценарий *script* для создания последовательного соединения. Сценарий может быть написан на любом языке, но, как правило, используется `chat`. В главе 6 приводится пример применения инструкции `connect` для вызова `chat`-сценария.

`connect-delay` *n*

Предписывает ожидать получения корректного PPP-пакета от удаленной системы в течение *n* миллисекунд после завершения работы сценария подключения.

`crtscts`

Включает аппаратный контроль передачи (RTS/CTS).

`debug`

Регистрирует все переданные и полученные пакеты посредством `syslogd` в режиме демона с уровнем `debug`. Инструкция `debug` может записываться в виде `-d`.

`default-asyncmap`

Отключает согласование карт `asyncmap`; все управляющие символы передаются в виде 2-байтных escape-последовательностей.

`default-mru`

Отключает согласование максимальной длины принимаемого блока; используется значение MRU по умолчанию – 1500 байт.

`defaultroute`

Определяет канал PPP в качестве маршрута по умолчанию. Маршрут удаляется при разрыве соединения.

`deflate` *nr*,*nt*

Предписывает `pppd` использовать по возможности сжатие пакетов Deflate. *nr* определяет максимальный размер окна получения данных в виде степени двойки. При значении *nr*, равном 8, окно получения (2 в восьмой степени) имеет размер 256 байт. *nt* определяет максимальный размер окна передачи данных в виде степени двойки. Если значение *nt* не указано, оно принимается равным значению *nr*.

`demand`

Переводит канал в режим подключения при необходимости (dial-on-demand mode). Сетевое соединение осуществляется только при наличии сетевого трафика.

`disconnect` *script*

Вызывает сценарий *script* для мягкого прерывания последовательного соединения. Может применяться любой язык сценариев, но обычно это `chat`.

`domain` *name*

Определяет имя локального домена. Воспользуйтесь данной настройкой, если команда `hostname` не возвращает полное доменное имя локальной машины.

`escape` *x*,*x*,...

Предписывает передавать перечисленные символы в виде двухсимвольных управляющих escape-последовательностей. При перечислении символы разделяются запятыми и записываются в шестнадцатеричной системе. Допускаются любые символы, кроме принадлежащих интервалу 0x20–0x3f и символа 0x5e.

`endpoint` *epdisc*

Определяет конечный ограничитель, передаваемый удаленной системе в процессе согласования многоканального логического соединения (multilink). По умолчанию передается MAC-адрес первого Ethernet-интерфейса либо, в отсутствие Ethernet-интерфейсов, IP-адрес системы. *epdisc* имеет формат *тип*:*значение*, где *тип* может принимать одно из значений: `local`, `IP`, `MAC`, `magic` или `phone`, а *значение* – IP-адрес в десятичном представлении через точку для типа `IP`, имя Ethernet-интерфейса для типа `MAC` либо строка, состоящая из шестнадцатеричных байтов, разделяемых двоеточиями, для всех прочих типов. Многоканальные логические соединения возможны только в системах Linux.

`file` *file*

Определяет дополнительный файл настроек с именем *file*. Обычно настройки читаются из файлов */etc/ppp/options*, *~/.ppprc*, командной строки, а также из */etc/ppp/options.device*. Описание этих файлов дается выше в тексте раздела.

`hide-password`

Скрывает строки паролей при регистрации в журнале содержимого пакетов протокола парольной идентификации PAP (*Password Authentication Protocol*).

`holdoff` *n*

Предписывает выжидать *n* секунд перед восстановлением разорванного соединения.

`idle` *n*

Предписывает разрывать соединение, если в течение *n* секунд не происходила передача или получение пакетов.

`init` *script*

Выполняет сценарий *script* для инициализации последовательной линии.

`ipcp-accept-local`

Предписывает использовать локальный IP-адрес, предоставленный удаленным сервером, даже если адрес определен локально.

`ipcp-accept-remote`

Предписывает использовать удаленный IP-адрес, предоставленный удаленным сервером, даже если адрес определен локально.

`ipcp-max-configure` *n*

Предписывает посылать пакеты IPCP configure-request не более *n* раз. По умолчанию не более 10 раз.

`ipcp-max-failure` *n*

Предписывает принимать до *n* пакетов IPCP configure-NAK перед посылкой пакета configure-reject. По умолчанию до 10 пакетов.

`ipcp-max-terminate` *n*

Предписывает передавать не более *n* пакетов IPCP terminate-request в случае отсутствия подтверждения. По умолчанию не более 3 пакетов.

`ipcp-restart` *n*

Предписывает выждать *n* секунд перед повторной передачей пакета IPCP configure-request. По умолчанию 3 секунды.

`ipparam` *string*

Передает строку *string* сценариям ip-up и ip-down. */etc/ppp/ip-up* – это сценарий командного интерпретатора, который выполняется `pppd`, когда канал начинает работу. Сценарий интерпретатора */etc/ppp/ip-down* выполняется `pppd`, когда работа канала прерывается.

`ipv6` *local_interface_identifier*,*remote_interface_identifier*

Устанавливает 64-битные идентификаторы для локального и удаленного интерфейсов. В качестве формата идентификатора служит стандартное ASCII-представление адреса IPv6. В отсутствие заданных значений система создает случайные идентификаторы. (См. также `ipv6cp-use-ipaddr` и `ipv6cp-use-persistent`.)

`ipv6cp-max-configure` *n*

Предписывает системе посылать пакеты IPv6CP configure-request не более *n* раз. По умолчанию не более 10 раз.

`ipv6cp-max-failure` *n*

Предписывает системе принимать до *n* пакетов IPv6CP configure-NAK. По умолчанию до 10 пакетов.

`ipv6cp-max-terminate` *n*

Предписывает системе передавать не более *n* пакетов IPv6CP terminate-request. По умолчанию не более 3 пакетов.

`ipv6cp-restart` *n*

Предписывает системе ждать *n* секунд перед повторной передачей пакета IPv6CP configure-request. По умолчанию 3 секунды.

`ipv6cp-use-ipaddr`

Предписывает системе использовать IPv4-адрес системы в качестве IPv6-идентификатора локального интерфейса.

`ipv6cp-use-persistent`

Предписывает системе использовать уникальный постоянный идентификатор системы в качестве IPv6-идентификатора локального интерфейса. Большинство систем не поддерживают работу с постоянными идентификаторами.

`kdebug` *n*

Включает диагностику на уровне ядра. При *n* = 1 отображаются общие отладочные сообщения, при *n* = 2 отображаются полученные пакеты, при *n* = 4 – переданные пакеты.

`ktune`

Предписывает системе разрешить `pppd` изменять настройки ядра. Например, в Linux-системе программа `pppd` способна включать IP-ретрансляцию установкой в единицу значения */proc/sys/net/ipv4/ip_forward* – если это разрешено инструкцией `ktune`.

`lcp-echo-failure` *n*

Предписывает разорвать соединение, если нет реакции на *n* LCP-пакетов echo-request. Как правило, сообщения echo-request не используются с такой целью, поскольку разрыв канала определяется аппаратурой модема.

`lcp-echo-interval` *n*

Предписывает выждать *n* секунд до передачи следующего LCP-пакета echo-request, если удаленная система не отвечает.

`lcp-max-configure` *n*

Предписывает посылать LCP-пакеты configure-request не более *n* раз. По умолчанию не более 10 раз.

`lcp-max-failure` *n*

Предписывает принимать до *n* LCP-пакетов configure-NAK перед посылкой configure-reject. По умолчанию до 10 пакетов.

`lcp-max-terminate` *n*

Предписывает передавать не более *n* LCP-пакетов terminate-request в случае отсутствия подтверждения. По умолчанию не более 3 пакетов.

`lcp-restart` *n*

Предписывает выждать *n* секунд перед повторной передачей LCP-пакета configure-request. По умолчанию 3 секунды.

`linkname` *name*

Устанавливает логическое имя канала (*name*). `pppd` записывает идентификатор собственного процесса в файл `ppp-`*name*`.pid`, расположенный в ката-

логе */var/run* либо */etc/ppp*. Файл отражает связь экземпляра `pppd` с определенным каналом.

`local`

Предписывает игнорировать управляющие линии модема DCD (Data Carrier Detect) и DTR (Data Terminal Ready).

`lock`

Предписывает использовать блокировку в стиле UUCP, чтобы обеспечить `pppd` монопольный доступ к последовательному устройству.

`logfd` *n*

Предписывает регистрировать сообщения в файле с дескриптором *n*.

`logfile` *filename*

Предписывает добавлять сообщения к файлу журнала (*filename*).

`login`

Предписывает использовать файл */etc/passwd* для аутентификации пользователей PAP. Факт регистрации в системе отражается в файле *wtmp*.

`maxconnect` *n*

Ограничивает максимальное время соединения значением в *n* секунд. По истечении *n* секунд соединение разрывается, даже если оно активно.

`maxfail` *n*

Устанавливает максимальное число последовательных попыток подключения к удаленной системе. По умолчанию *n* равно 10.

`modem`

Предписывает использовать управляющие линии модема DCD (Data Carrier Detect) и DTR (Data Terminal Ready); перед открытием последовательного устройства ожидать сигнала DCD (наличие несущей частоты); сбрасывать сигнал DTR (готовность к принятию данных) при закрытии соединения.

`mp`

Сокращенный вариант инструкции `multilink`. См. описание `multilink`.

`mpshortseq`

Предписывает использовать короткие, 12-битные порядковые номера в заголовках многоканальных соединений вместо стандартных 24-битных.

`mrru` *n*

Устанавливает размер MRRU (Maximum Reconstructed Receive Unit) в *n* байт. MRRU – это максимальный размер пакета, который может быть получен по многоканальному соединению. Значение интерпретируется аналогично MRU для других видов транспорта.

`mru` *n*

Устанавливает MRU (Maximum Receive Unit) в *n* байт. Значение MRU сообщает удаленной системе максимальный размер пакета, который спо-

собна принять локальная система. Минимальное значение – 128. По умолчанию – 1500.

`ms-dns` *address*

Позволяет указывать адреса DNS для клиентов Microsoft Windows.

`ms-wins` *address*

Позволяет указывать адреса серверов WINS (Windows Internet Name Services) для клиентов Microsoft Windows.

`mtu` *n*

Устанавливает MTU (Maximum Transmission Unit) в *n* байтов. MTU определяет максимальную длину для передаваемых пакетов. Максимальная длина пакета определяется меньшим из значений локального MTU и MRU удаленной системы.

`multilink`

Включает протокол многоканальных соединений, позволяющий использовать ряд физических каналов в качестве одного логического. Протокол применяется для повышения скорости обмена данными с удаленной системой. Например, два модемных подключения к одной удаленной системе могут рассматриваться в качестве одной многоканальной связки, удваивающей полосу пропускания. Данная возможность в настоящее время доступна только для систем Linux.

`name` *name*

Предписывает использовать имя *name* в качестве имени локальной системы при проверке подлинности.

`netmask` *mask*

Задает маску подсети.

`noaccomp`

Запрещает согласование сжатия Address/Control.

`noauth`

Разрешает доступ без идентификации.

`nobsdcomp`

Отключает сжатие BSD-Compress.

`noccp`

Отключает согласование по протоколу CCP (Compression Control Protocol, протокол управления сжатием).

`nocrtscts`

Отключает все виды аппаратного контроля передачи.

`nodtrcts`

Отключает все виды аппаратного контроля передачи.

`nodefaultroute`

Запрещает пользователям создавать маршрут по умолчанию посредством инструкции `defaultroute`.

`nodeflate`

Отключает сжатие Deflate.

`nodetach`

Предотвращает запуск `pppd` в фоновом режиме. См. примеры в главе 6.

`noendpoint`

Запрещает прием и передачу конечного ограничителя для многоканальных соединений.

`noip`

Отключает протоколы IPCP и IP.

`noipv6`

Запрещает согласование по IPv6CP и обмен данными по IPv6.

`noipdefault`

Запрещает использование команды `hostname` для определения локального IP-адреса. Адрес должен быть получен от удаленной системы либо установлен явным образом при помощи параметра.

`noktune`

Запрещает `pppd` изменять значения параметров настройки ядра.

`nolog`

Отключает ведение журнала.

`nomagic`

Запрещает согласование «магического числа».

`nomp`

Отключает протокол многоканальных соединений.

`nompshortseq`

Запрещает применение коротких, 12-битных порядковых номеров для протокола многоканальных соединений.

`nomultilink`

Отключает протокол многоканальных соединений.

`nopcomp`

Запрещает согласование сжатия полей протокола. По умолчанию сжатие полей протокола не используется. Применение данной инструкции означает, что сжатие не будет использовано даже при явном запросе удаленной системы.

`nopersist`

Предписывает завершить работу после того, как установлено соединение. Разорванное соединение не восстанавливается. Таково поведение по умолчанию.

`nopredictor1`

Запрещает использование сжатия Predictor-1.

`noproxyarp`

Запрещает применение `proxyarp`, то есть создание прокси-ARP-записей пользователями `pppd`.

`notty`

Предписывает `pppd` передавать символы на стандартный вывод, а получать их со стандартного ввода. Применение инструкции увеличивает задержки и непроизводительные расходы.

`novj`

Запрещает сжатие заголовков по методу Van Jacobson.

`novjccomp`

Запрещает при сжатии заголовков по методу Van Jacobson сжимать идентификатор соединения.

`papcrypt`

Запрещает принимать пароли, совпадающие с указанными в файле */etc/ppp/pap-secrets*, поскольку хранимые пароли зашифрованы, и полученный в процессе проверки подлинности пароль не должен совпадать с записями файла *pap-secrets* до шифрования.

`pap-max-authreq` *n*

Предписывает передавать не более *n* PAP-запросов на идентификацию, если удаленная система не отвечает. По умолчанию не более 10 раз.

`pap-restart` *n*

Предписывает выждать *n* секунд перед повторной передачей PAP-запроса на идентификацию. По умолчанию 3 секунды.

`pap-timeout` *n*

Устанавливает интервал ожидания идентификации удаленной системы в *n* секунд. При *n* = 0 ограничений по времени нет.

`pass-filter` *filter-expression*

Создает пакетный фильтр, определяющий, какие пакеты разрешено передавать и принимать по каналу PPP. Пакеты, не соответствующие фильтру, удаляются без предупреждения. Выражение *filter-expression* имеет синтаксис, определенный для команды `tcpdump`.

`passive`

Предписывает дождаться пакета LCP (Link Control Protocol, протокол управления каналом) от удаленной системы, даже если она не ответила на первый LCP-пакет, отправленный локальной системой. В отсутствие данной инструкции локальная система разрывает соединение, не получив ответа. Инструкция `passive` может записываться в виде `-p`.

`persist`

Предписывает системе повторно открыть соединение, если оно было разорвано по сигналу SIGHUP.

`plugin` *filename*

Загружает объект динамической библиотеки в качестве подключаемого модуля `pppd`.

`predictor1`

Предписывает запрашивать у удаленной системы применение сжатия Predictor-1.

`privgroup` *group-name*

Разрешает всем членам группы *group-name* пользоваться привилегированными инструкциями.

`proxyarp`

Разрешает использование прокси-ARP. В таблицу ARP локальной системы добавляется прокси-ARP-запись для удаленной системы.

`pty` *script*

Указывает сценарий, выполняемый в порожденном процессе. Сценарий используется в качестве источника данных вместо терминала. При использовании совместно с инструкцией `record` каналы стандартного ввода/вывода порожденного процесса связаны с конвейерами.

`receive-all`

Предписывает принимать все управляющие символы от второй стороны, даже подлежащие удалению согласно стандартным процедурам async-map-обработки, определенным в RFC 1662.

`record` *filename*

Предписывает записывать все переданные и полученные символы в файл *filename*.

`remotename` *name*

Предписывает использовать имя *name* в качестве имени удаленной системы при проверке подлинности.

`refuse-chap`

Запрещает применение CHAP. Это не очень хорошая мысль.

`refuse-pap`

Запрещает применение PAP.

`require-chap`

Предписывает использовать CHAP.

`require-pap`

Предписывает использовать PAP.

`show-password`

Разрешает отображать пароль при регистрации пакетов PAP.

`silent`

Предписывает дожидаться LCP-пакета от удаленной системы, не посылая начальный LCP-пакет.

`sync`

Предписывает использовать синхронные протоколы физического уровня HDLC вместо стандартного асинхронного протокола.

`updetach`

Предписывает отсоединиться от управляющего терминала (переход в фоновый режим работы) после того, как установлено соединение.

`usehostname`

Предписывает использовать локальное имя узла для идентификации. Отменяет действие инструкции `name`.

`usepeerdns`

Предписывает запрашивать у удаленной системы до двух адресов серверов DNS. Полученные адреса передаются сценарию */etc/ppp/ip-up* в переменных среды DNS1 и DNS2. Кроме того, `pppd` использует эти адреса для создания nameserver-записей в файле */etc/ppp/resolv.conf*.

`user` *`username`*

Предписывает использовать имя *`username`* для PAP-идентификации, инициированной удаленной системой.

`vj-max-slots` *`n`*

Предписывает использовать *`n`* каналов соединений для сжатия заголовков Van Jacobson. Число *`n`* должно принадлежать интервалу от 2 до 16.

`welcome` *`script`*

Предписывает выполнить сценарий *`script`* перед началом согласования взаимодействия PPP.

`xonxoff`

Включает программный контроль передачи (XON/XOFF).

Некоторые из перечисленных инструкций касаются вопросов безопасности PPP, которая представляет одну из сильных сторон протокола. Предпочтительным протоколом безопасности PPP является CHAP (*Challenge Handshake Authentication Protocol*). Менее безопасный протокол PAP (*Password Authentication Protocol*) поддерживается только для совместимости с менее функциональными системами. Имена пользователей, IP-адреса и секретные ключи, используемые в работе этих протоколов, определяются в файлах */etc/ppp/chap-secrets* и */etc/ppp/pap-secrets*. Форматы файлов и их применение описаны в главе 6.

Очень важно проследить, чтобы каталог */etc/ppp* и его содержимое не были доступны для записи всем пользователям или определенной группе пользователей. Изменения в файлах *chap-secrets*, *pap-secrets* и *options* могут подвергнуть риску безопасность системы. Кроме того, файлы сценариев */etc/*

ppp/ip-up и */etc/ppp/ip-down* могут выполняться с привилегиями администратора системы. Если в каталоге */etc/ppp* существует файл с именем *ip-up*, он выполняется pppd, как только установлено PPP-соединение. Сценарий *ip-up* используется для внесения изменений в таблицу маршрутизации, обработки очередей sendmail, а также решения других задач, связанных с наличием сетевого соединения. Сценарий *ip-down* выполняется pppd после закрытия PPP-соединения и используется для завершения процессов, работающих с каналом. Очевидно, что эти сценарии, как и весь каталог */etc/ppp*, должны быть надежно защищены.

Обработка сигналов

pppd обрабатывает следующие сигналы:

SIGUSR1

Включает и отключает отладку. Первый сигнал SIGUSR1, полученный pppd, включает отладку и приводит к записи диагностических сообщений посредством демона syslogd (режим *daemon*, приоритет *debug*). Второй сигнал SIGUSR1 отключает отладку и закрывает файл журнала. См. также описание инструкции debug выше по тексту.

SIGUSR2

Приводит к повторному согласованию сжатия. Сфера применения сигнала весьма ограничена, поскольку он нужен только для перезапуска системы сжатия после возникновения критических ошибок. Большинство пользователей при возникновении критических ошибок просто устанавливают PPP-соединение заново.

SIGHUP

Закрывает PPP-соединение, возвращает последовательное устройство в обычный режим работы и завершает работу pppd. Если определена инструкция persist, pppd продолжает работу и открывает новое соединение.

SIGINT

Данный сигнал, как и сигнал SIGTERM, закрывает PPP-соединение, возвращает последовательное устройство в обычный режим работы и завершает работу pppd. Инструкция persist влияния не оказывает.

chat

chat – это язык сценариев общего назначения, применяемый для управления модемом, выполнения звонка и регистрации на удаленном сервере. chat не столь функционален, как dip, но достаточно широко используется. Структура сценария chat – «ожидание/передача» – является общей для большинства подобных языков.

chat-сценарий состоит из пар ожидание/передача. Пара состоит из двух значений, разделенных пробельными символами. Первое значение представляет строку, которая ожидается от удаленной системы, второе – строку, кото-

рую следует передать в ответ. Специальный случай, когда нет необходимости получать строку от удаленной системы, обозначается парой двойных (`""`) или одинарных (`''`) кавычек. В качестве примера рассмотрим простейший сценарий:

```
"" \r name> jane word> TOga!toGA
```

Сценарий не ожидает получения каких-либо строк (`""`), пока удаленной системе не будет передан символ возврата каретки (`\r`). Затем сценарий ожидает получения от удаленной системы строки `name>`, которая является частью системного приглашения `Username>`. В ответ на приглашение сценарий передает регистрационное имя пользователя, `jane`. Наконец, сценарий ожидает получения части приглашения `Password>` и отвечает паролем `TOga!toGA`. Сценарий настолько простой, что его можно создать прямо в командной строке программы `chat`:

```
% chat -v -t30 "" \r name> jane word> TOga!toGA
```

Приведенная команда запускает `chat` в режиме подробной диагностики, устанавливает длительность ожидания строки в 30 секунд, а затем выполняет простой сценарий регистрации в удаленной системе.

Синтаксис команды `chat`:

```
chat [ключи] [сценарий]
```

Ключи команды `chat`:

`-e`

Отображать на экране вывод модема в поток stderr. Идентичный результат можно получить при помощи ключевого слова ECHO в `chat`-сценарии.

`-E`

Разрешает использование переменных среды в тексте сценария.

`-s`

Предписывает направлять все записи журнала и сообщения об ошибках в поток stderr.

`-S`

Запрещает направлять сообщения журнала и сообщения об ошибках службе SYSLOG.

`-T` *phone-number*

Заменяет escape-последовательность \T в тексте `chat`-сценария значением *phone-number*.

`-U` *phone-number-2*

Заменяет escape-последовательность \U в тексте `chat`-сценария значением *phone-number-2*.

`-v`

Режим подробной диагностики. В этом режиме информационные сообщения записываются посредством `syslogd`.

`-V`

Режим подробной диагностики с записью информационных сообщений в поток stderr. В главе 6 содержится пример использования этого ключа с `pppd`.

`-t` *`timeout`*

Устанавливает длительность ожидания строки. Если строка не получена за *`timeout`* секунд, ответ не посылается, а сценарий завершает работу (кроме тех случаев, когда определена альтернативная обработка). В случае альтернативной обработки (о которой мы поговорим ниже) удаленной системе передается определенная строка и наступает следующий интервал ожидания. Если и это не приводит к получению строки, сценарий прекращает работу с ненулевым кодом завершения. По умолчанию интервал *`timeout`* равен 45 секундам.

`-f` *`scriptfile`*

Сценарий `chat` читается из файла *`scriptfile`*, а не из командной строки. Файл может содержать произвольное количество пар ожидание/передача.

`-r` *`reportfile`*

`chat` производит запись информации, сгенерированной строками REPORT, в файл *`reportfile`*. По умолчанию строки REPORT записываются в поток stderr. Ключевое слово REPORT будет описано ниже.

В целях повышения надежности и функциональности сценариев в программе `chat` реализованы специальные ключевые слова, поддержка escape-последовательностей, а также возможность применения альтернативных пар передача/ожидание. Прежде всего, рассмотрим семь самых важных ключевых слов.

Два ключевых слова отвечают за передачу специальных сигналов удаленной системе. Ключевое слово EOT приводит к посылке символа End of Transmission (конец передачи). В системах Unix это, как правило, символ конца файла, <Ctrl>+<D>. Ключевое слово BREAK приводит к посылке удаленной системе символа разрыва строки. Еще пять ключевых слов (TIMEOUT, ABORT, REPORT, CONNECT и SAY) определяют параметры работы собственно сценария.

Ключевое слово TIMEOUT позволяет указать длительность ожидания строки. При определении в тексте сценария данное значение может изменяться для каждой из ожидаемых строк. Предположим, мы даем удаленному серверу 30 секунд, чтобы отобразить приглашение `Username>`, но лишь 5 секунд, чтобы отобразить приглашение `Password>` после передачи имени пользователя. Задача решается при помощи следующей команды:

```
TIMEOUT 30 name> karen TIMEOUT 5 word> beach%PARTY
```

Ключевые слова ABORT и REPORT функционально схожи. Оба слова определяют строки, получение которых приводит к выполнению специальных действий. Ключевое слово ABORT определяет строки, получение которых вместо строки CONNECT от модема приводит к прерыванию работы сценария. Ключевое слово REPORT определяет подстроки, присутствие которых

в сообщениях, полученных от последовательного порта, приводит к записи сообщений в поток stderr или файл отчета. Проиллюстрируем применение этих ключевых слов следующим примером:

```
REPORT CONNECT
ABORT BUSY
ABORT 'NO CARRIER'
ABORT 'RING - NO ANSWER'
SAY "Производится подключение к серверу PPP..."
"" ATDT5551234
CONNECT \r
name> karen
word> beach%PARTY
```

Первая строка предписывает регистрировать в журнале все сообщения, полученные сценарием, которые содержат слово CONNECT. Если при запуске программы `chat` присутствовал ключ командной строки `-r`, сообщение заносится в файл, определенный аргументом ключа. В противном случае сообщение попадает в поток stderr. Назначение данной команды – отображать на терминале пользователя полученное от модема сообщение о подключении. Полное сообщение может выглядеть так: `CONNECT 28,800 LAPM/V`; оно позволяет пользователю определить, какова скорость соединения и какой используется протокол передачи. Сообщение CONNECT означает успешное подключение. Следующие три строки сценария начинаются ключевым словом ABORT и связаны с сообщениями модема, означающими различные ошибки. Если от модема поступают сообщения BUSY, NO CARRIER или RING – NO ANSWER, работа сценария прерывается.

Ключевое слово SAY передает строку-аргумент на терминал пользователя. В данном случае мы говорим пользователю, что сценарий начал звонить.

Последние четыре строки – это пары ожидание/передача, с которыми мы уже познакомились в этом разделе. Не ожидая получения строки (`""`), сценарий передает модему команду набора номера (`ATDT`). Сценарий ожидает получения строки CONNECT от модема и передает символ возврата каретки удаленному серверу; ожидает получения строки `Username>` от удаленного сервера и отвечает именем `karen`. Наконец, сценарий ожидает получения строки `Password>` от сервера и отвечает паролем `beach%PARTY`.

`chat` расширяет функциональность стандартных пар ожидание/передача альтернативными вариантами, повышающими надежность сценариев. Существует возможность задать альтернативную строку для передачи и альтернативное ожидаемое значение, которые используются, если истек временной интервал для основного ожидаемого значения. Альтернативные варианты в тексте сценария предваряются дефисами. Пример:

```
gin:-BREAK-gin: becca
```

В данном случае ожидается получение строки `gin:`, предполагается ответ строкой `becca`. Первая и последняя строки составляют стандартную пару ожидание/передача. Альтернативная пара передача/ожидание использует-

ся, если истек интервал ожидания и строка gin: не была получена. Тогда сценарий передает разрыв строки, обнуляет таймер и снова переходит к ожиданию строки gin:, поскольку именно такое поведение предписывается альтернативной парой передача/ожидание (-BREAK-gin:). Заметим, что для альтернативной пары передача значения выполняется до перехода в режим ожидания строки, то есть передача предшествует ожиданию. Еще один пример в серии наших сценариев:

```
name>--name> karen
```

Сценарий ожидает получения строки name>. Если строка не получена, сценарий передает пустую строку (возврат каретки) и снова переходит к ожиданию строки name>. Такое поведение определено альтернативной парой передача/ожидание, --name>. Пара начинается с дефиса, который отмечает начало строки для передачи, но следующий символ также является дефисом и отмечает начало альтернативного ожидаемого значения. Строка для передачи, таким образом, отсутствует. Именно эта «пустая строка» приводит к передаче одного символа возврата каретки. Подобная конструкция, будучи менее прозрачной, применяется чаще, чем вариант со словом BREAK, описанный выше.

Помимо символа возврата каретки в chat существуют и другие escape-последовательности для приема и передачи специальных символов. Они перечислены в табл. А.2.

Таблица А.2. Escape-последовательности chat

Escape-последовательность	Значение
\b	Забой
\	Передать без завершающего символа возврата каретки
\d	Задержка передачи значения на одну секунду
\K	Передать символ BREAK
\n	Передать символ новой строки
\N	Передать нулевой символ
\p	Задержка передачи значения на одну десятую долю секунды
\xd5	Передать строку, но не фиксировать передачу в журнале
\r	Возврат каретки
\s	Символ пробела
\T	Передать значение, полученное из командной строки chat в виде аргумента ключа -T
\t	Символ табуляции
\U	Передать значение, полученное из командной строки chat в виде аргумента ключа -U
\\	Символ \

Таблица А.2 (продолжение)

Escape-последовательность	Значение
\ddd	ASCII-символ с восьмеричным значением ddd
^C	Управляющий символ

Все escape-последовательности, кроме той, что используется для ввода управляющих символов, начинаются с обратной косой черты (\). Управляющий символ состоит из символа ^ и заглавной буквы. Например, управляющий символ X записывается в виде ^X. Escape-последовательности, описанные в табл. А.2 с упоминанием передачи, могут использоваться только в строках для передачи; все прочие последовательности могут использоваться и для приема, и для передачи. Следующий пример иллюстрирует применение некоторых escape-последовательностей:

```
"" \d\d^G\p^G\p\p^GПросыпайся\sскорее!\nЛежебока\sты!
```

Не ожидая получения значения (""), сценарий делает паузу в две секунды (\d\d). Передает три символа ASCII BELL, которые в клавиатурной записи обозначаются как <Ctrl>+<G>, с интервалом в одну десятую долю секунды (^G\p^G\p\p^G). Передает строку Просыпайся скорее!. Переходит на новую строку (\n) и передает строку Лежебока ты!.

Из соображений безопасности сервер может перезванивать (callback) клиенту для создания соединения. Таким образом сервер проверяет, что звонок клиента поступил с допустимого телефонного номера. Механизм работает следующим образом:

- Клиент звонит на сервер и передает строку-идентификатор.
- Получив идентификатор, сервер разрывает соединение.
- Сервер использует идентификатор, чтобы определить телефонный номер клиента, и звонит по этому номеру.
- Клиент продолжает процесс регистрации на сервере.

Тот факт, что сервер разрывает соединение, может стать источником проблем для chat-сценария. Обычно разрыв коммутируемого канала приводит к безусловному завершению подключения. Для работы с callback-серверами существует команда HANGUP. Команда HANGUP OFF предотвращает завершение сценария регистрации в случае, если сервер разрывает соединение. Команда HANGUP OFF должна следовать непосредственно за командой, передающей серверу строку-идентификатор. Когда сервер перезвонит и будет установлено соединение, воспользуйтесь командой HANGUP ON, чтобы вернуться к нормальной обработке разрыва подключения. По умолчанию (HANGUP ON) работа сценария прерывается, если произошел разрыв подключения.

В момент завершения работы chat-сценария устанавливается значение кода завершения. Код завершения – это числовое значение, обозначающее состо-

яние сценария на момент завершения его работы. Ниже перечислены основные числовые коды и дана интерпретация каждого из них:

0 Сценарий завершился нормально.

1 Не удалось выполнить сценарий из-за некорректного параметра либо из-за переполнения буфера в процессе получения строки.

2 Работа сценария принудительно завершена из-за ошибки ввода/вывода либо по сигналу SIGINT/SIGTERM.

3 Работа программы завершена: истек интервал ожидания для строки.

4 и далее

Возникло условие, определенное командой ABORT. Числовое значение указывает, какое из условий выполнено. Первое условие, определенное командой ABORT, связано со значением 4; второе – со значением 5; третье – со значением 6 и т. д.

Коды завершения от 0 до 3 вполне прозрачны. Остальные значения легче понять на примере.

Ранее в этом разделе приводился пример сценария, содержавшего три команды ABORT: первая определяла поведение для события BUSY, вторая для события NO CARRIER, третья – для события RING – NO ANSWER. Если модем возвращает строку BUSY, сценарий завершает работу с кодом 4. Если модем возвращает строку RING – NO ANSWER, сценарий завершает работу с кодом 6. Значение кода завершения в каждом конкретном случае зависит от порядкового номера команды ABORT. Если бы другой пользователь переписал данный сценарий, расположив команду ABORT RING – NO ANSWER ранее всех прочих команд ABORT, завершение по событию RING – NO ANSWER приводило бы к получению кода 4, а не 6, как в нашем случае.

B

gated, справочник

Данное приложение описывает синтаксис команды gated и языка настройки gated для Gated 3.6 – свободно доступной версии программы. Приложение может использоваться в качестве самостоятельного справочника по языку gated, но чтобы полностью разобраться в работе программы, следует использовать его в сочетании с примерами из главы 7.

gated постоянно развивается. Со сменой версий изменяется и язык команд. Чтобы получить свежую информацию по установленной версии gated, обращайтесь к файлам руководства man.

Команда gated

Синтаксис команды gated:

```
gated [-v] [-c] [-C] [-n] [-N] [-t параметры трассировки] [-f файл_настройки]
[файл_трассировки]
```

Ключи командной строки -c и -n выполняют отладку файла настройки маршрутизации, не затрагивая настройки сети и таблицу маршрутизации ядра. Часто эти ключи используются для тестирования файла настройки, указанного в ключе -f:

-c

Предписывает gated прочитать файл настройки и проверить его на наличие синтаксических ошибок. Завершив чтение файла, gated создает образ своего состояния и завершает работу. Образ записывается в файл */usr/tmp/gated_dump*. Запуск gated с ключом -c не требует полномочий администратора, равно как и обязательного завершения активного процесса gated.

-C

Выполняет проверку файла настройки на наличие ошибок синтаксиса. gated завершает работу с кодом 1, если ошибки обнаружены, и с кодом 0, если ошибок нет. Код завершения может быть полезен при вызове gated из сценариев.

-n

Запрещает gated обновлять таблицу маршрутизации ядра. Используется для проверки настроек маршрутизации на реальных данных маршрутизации без вмешательства в работу операционной системы.

-f *файл настройки*

Предписывает gated читать настройки из указанного файла, а не из файла настройки по умолчанию, */etc/gated.conf*. В сочетании с ключом -c ключ -f выполняет проверку настроек, не изменяя настройки работающего в настоящий момент демона gated.

Ключ -v приводит к отображению номера версии gated. Если он фигурирует в командной строке, все другие ключи не обрабатываются, поскольку gated завершает работу сразу после вывода сведений о версии.

Ключ -N запрещает gated переход в фоновый режим демона. Данный параметр используется при запуске gated из *inittab*. По умолчанию gated работает в режиме демона.

Аргументы командной строки *параметры трассировки* и *файл трассировки* используются для трассировки протоколов. Аргумент *файл трассировки* определяет имя файла, в который записывается вывод трассировки. Если имя такого файла не указано, информация передается в поток стандартного вывода. Трассировка обычно приводит к получению больших объемов вывода.

Параметры командной строки для трассировки:

-t

Включает трассировку. Если ключ -t используется в отсутствие параметров трассировки, gated по умолчанию выполняет трассировку в режиме general, то есть фиксирует нормальное взаимодействие протоколов и изменения в таблице маршрутизации. gated всегда регистрирует в журнале ошибки протоколов, даже если не включена трассировка. Могут быть указаны *параметры трассировки*, описанные ниже в данном приложении. Некоторые из этих параметров (detail, send, recv) не могут фигурировать в командной строке gated. Два других наиболее полезны при указании в командной строке:

symbols

Трассировка имен, полученных от ядра; представляет интерес преимущественно для разработчиков, занятых отладкой взаимодействия ядра и gated.

`iflist`

Трассировка списка интерфейсов, полученного от ядра. Используется для определения интерфейсов, обнаруженных при сканировании ядром системы.

Преимущество указания параметра трассировки в командной строке заключается в возможности отслеживать ход событий, происходящих до обработки файла настройки. Для двух описанных выше параметров это важное преимущество. Для других параметров оно не столь значительно. Большинство параметров трассировки хранится в файле настройки. Более подробная информация содержится в описании команды `traceoptions`, включенном в данное приложение.

Обработка сигналов

`gated` обрабатывает следующие сигналы:

SIGHUP

Предписывает `gated` повторно прочитать файл настройки. Новые настройки заменяют те, с которыми работает `gated`. SIGHUP загружает новый файл настройки, не останавливая работу `gated`. SIGHUP можно использовать для оперативного изменения настроек. Настройки маршрутизации большинства площадок изменяются редко. В тех редких случаях, когда необходимо перейти к новым настройкам, завершайте работу `gated` и повторно запускайте службу с новыми настройками – это позволит точнее определить, как служба будет работать после перезагрузки системы.

SIGINT

Предписывает `gated` записать образ своего состояния в файл */usr/tmp/gated_dump*.

SIGTERM

Предписывает `gated` мягко завершить работу. Все протоколы завершают работу в соответствии с правилами. Например, EGP передает сообщение CEASE и ожидает получения подтверждения. SIGTERM удаляет из таблицы маршрутизации ядра все маршруты, полученные посредством протоколов внешней маршрутизации. Чтобы сохранить эти маршруты, но прервать работу `gated`, воспользуйтесь сигналом SIGKILL.

SIGKILL

Предписывает `gated` немедленно завершить работу и создать образ памяти. Маршруты не удаляются из таблицы маршрутизации, мягкое завершение работы не применяется.

SIGUSR1

Предписывает `gated` переключить трассировку. В отсутствие флагов трассировки SIGUSR1 ни на что не влияет. Но если трассировка включена, первый сигнал SIGUSR1 приводит к отключению трассировки и закрытию файла трассировки. Следующий сигнал SIGUSR1 снова включает

трассировку и открывает файл трассировки. Когда файл трассировки закрыт, его можно перемещать или удалять, не мешая работе gated. Используйте эту возможность для периодического удаления файла трассировки с целью предотвращения его чрезмерного роста.

SIGUSR2

Предписывает gated проверить наличие изменений в состоянии сетевых интерфейсов.

Ниже приводятся примеры обработки сигналов командой gated. Прежде всего, процессу gated передается сигнал SIGUSR1 (идентификатор процесса извлекается из файла *gated.pid* – в данном случае */var/run/gated.pid*).

```
# kill -USR1 'cat /var/run/gated.pid'
```

Затем старый файл трассировки (*/usr/tmp/gated.log* в данном случае) удаляется, а gated передается второй сигнал SIGUSR1.

```
# rm /usr/tmp/gated.log
# kill -USR1 'cat /etc/gated.pid'
```

Получив второй сигнал, gated открывает новый файл трассировки (все с тем же именем */usr/tmp/gated.log*). Команда ls показывает, что создан новый файл.

```
# ls -l /usr/tmp/gated.log
-rw-rw-r--  1 root          105 Jul  6 16:41 /usr/tmp/gated.log
```

Язык настройки gated

Язык настройки gated – жестко структурированный язык, схожий внешне с языком C. Комментарии либо начинаются символом решетки (#), либо начинаются символами /* и заканчиваются символами */. Операторы настройки gated заканчиваются точкой с запятой, а группа связанных операторов заключается в фигурные скобки. Структура языка знакома большинству системных администраторов Unix и облегчает чтение разделов настройки. Это важное свойство для файлов, которые могут содержать операторы настройки многих протоколов одновременно.

Язык настройки образован операторами девяти типов. *Инструктирующие операторы* и *операторы трассировки* могут располагаться в любом месте файла *gated.conf* и напрямую не связаны с настройкой какого-либо протокола. Эти операторы инструктируют интерпретатор файла настройки и управляют трассировкой. Семь других типов операторов: *операторы параметров*, *операторы интерфейсов*, *операторы определений*, *операторы протоколов*, *статические операторы*, *управляющие операторы* и *операторы объединения*. Эти операторы должны присутствовать в файле настройки в строго определенном порядке, начиная с операторов параметров и заканчивая операторами объединения. Нарушение порядка следования операторов приводит к возникновению ошибки в процессе интерпретации файла.

Ниже приводится описание всех команд языка настройки gated, упорядоченное по типу операторов.

Инструктирующие операторы

Инструктирующие операторы передают интерпретатору команд gated указания относительно «включаемых» файлов. Включаемый файл – это внешний файл, содержимое которого подвергается интерпретации в ходе чтения настроек, как если бы являлось частью исходного файла *gated.conf*. Включаемые файлы могут содержать ссылки на другие включаемые файлы, и уровень вложенности ссылок может достигать 10.

Инструктирующих операторов всего два:

`%include` *filename*

Указывает включаемый файл. Содержимое файла «включается» в файл *gated.conf* в точке присутствия инструкции `%include`. Именем файла (*filename*) может быть любое допустимое в Unix имя файла. Если имя файла является относительным, то есть не начинается символом /, оно интерпретируется относительно каталога, указанного в инструкции `%directory`.

`%directory` *pathname*

Определяет каталог, в котором хранятся включаемые файлы. gated ищет в каталоге *pathname* все включаемые файлы с относительными именами.

Используйте включаемые файлы только в случае очень сложных конфигураций маршрутизации. В сложной конфигурации разбиение крупного файла настройки на мелкие, более простые для понимания сегменты может оказаться полезным, однако в большинстве случаев файлы настройки gated весьма невелики. Одним из больших преимуществ gated является возможность сочетать настройки различных протоколов маршрутизации в одном файле. Если этот файл невелик и легко читаем, разбиение лишь усложнит ситуацию.

Операторы трассировки

Операторы трассировки позволяют управлять файлом трассировки и его содержимым. Оператор трассировки выглядит следующим образом:

```
traceoptions
 ["trace file" [replace] [size bytes[k|m] files n]]
 [nostamp]
 параметры трассировки [except параметры трассировки]
;
```

Его составляющие:

trace_file

Указывает файл, в который записывается вывод трассировки. Его назначение полностью идентично назначению аргумента *файл трассировки* командной строки gated.

`replace`

Переписывает существующий файл трассировки. В отсутствие этого ключевого слова вывод трассировки добавляется к существующему содержимому файла.

`size` *`bytes`*`[k|m] [files` *`n`*`]`

Ограничивает файл трассировки размером в определенное число байт (*`bytes`*). Необязательные суффиксы `k` и `m` обозначают тысячи (`k`) и миллионы (`m`) байт. Таким образом, значения `10000000` и `10m` эквивалентны. Размер файла трассировки не может быть меньше, чем 10k байт. *`n`* определяет максимальное число сохраняемых файлов трассировки. Когда файл трассировки достигает максимального размера, он сохраняется под именем *`trace_file`*.0, *`trace_file`*.1, *`trace_file`*.2 и так до *`trace_file`*.*`n`*. Следующее сохранение приводит к перезаписи *`trace_file`*.0. Значение *`n`* должно быть не меньше 2.

`nostamp`

Указывает, что строки файла трассировки не должны начинаться с отметки времени. По умолчанию отметка времени проставляется в каждой строке.

`параметры трассировки`

Указывает события, трассировку которых будет выполнять `gated`. Каждый из параметров трассировки обозначается определенным ключевым словом. Существуют следующие параметры трассировки:

`none`

Отключает трассировку.

`all`

Включает глобальную трассировку всех типов.

`general`

Включает трассировку `normal` и `route`.

`state`

Трассировка переходов автоматов состояний для протоколов OSPF и BGP. В документах RFC эти протоколы описываются при помощи диаграмм или таблиц *конечных автоматов состояний*. Переход протокола из одного состояния в другое основан на возникновении определенных событий. К примеру, состояние *idle* может смениться на состояние *connect* в результате возникновения события открытия соединения. Этот узко специализированный флаг трассировки полезен только тем, кто досконально знает соответствующие протоколы. Используйте этот параметр совместно с оператором протокола для трассировки изменения состояний этого протокола.

`normal`

Трассировка нормальных взаимодействий протоколов. Трассировка ошибок выполняется всегда.

`policy`

Трассировка применения правил маршрутизации. Используется для проверки корректности конфигурации правил маршрутизации.

`task`

Трассировка обработки системного уровня.

`timer`

Трассировка различных таймеров протокола или других таймеров того же уровня.

`route`

Трассировка изменений таблицы маршрутизации. Используйте для проверки корректности маршрутов, установленных протоколом.

`detail`

Трассировка содержимого пакетов маршрутизатора. Должен быть определен до установки параметра `send` или `recv`.

`send`

Ограничивает трассировку `detail` пакетами, исходящими от маршрутизатора.

`recv`

Ограничивает трассировку `detail` пакетами, полученными маршрутизатором. Без этих двух параметров происходит трассировка всех пакетов при указании `detail`.

`symbols`

Трассировка имен, полученных от ядра при запуске. См. описание ключа командной строки `-t`.

`iflist`

Трассировка списка интерфейсов ядра. См. описание ключа командной строки `-t`.

`parse`

Трассировка работы лексического анализатора и интерпретатора.

`adv`

Трассировка выделения и освобождения блоков.

`except` *`параметры трассировки`*

Отключает указанные параметры трассировки. Данный элемент должен использоваться в сочетании с *`параметрами трассировки`*, включающими трассировку различных типов. Например, `traceoptions all except state` включает все виды трассировки, за исключением трассировки состояний конечных автоматов.

`gated` позволяет управлять трассировкой как из командной строки, так и посредством файла настройки. В общем, в командной строке `gated` могут ука-

зываться те же параметры, что и в файле настройки. `detail`, `send` и `recv` могут фигурировать только в файле настройки.

Два других параметра, `symbols` и `iflist`, используются в основном в командной строке. Указание параметров трассировки при помощи ключа командной строки `-t` описано в разделе, посвященном команде `gated`.

Некоторые из параметров трассировки будут полезны только разработчикам протоколов и другим специалистам. Для большинства из нас информацию, необходимую для отладки маршрутизации, обеспечивает параметр `general`, включающий трассировку `normal` и `route`. Время от времени при тестировании правил маршрутизации возникает потребность в параметре `policy`. Однако в большинстве случаев трассировка не нужна.

Операторы параметров

Данные операторы определяют параметры, предписывающие `gated` выполнять специальную обработку. В файле *gated.conf* операторы параметров предшествуют всем другим операторам настройки.

Синтаксис оператора параметров:

```
options
[nosend]
[noresolv]
[gendefault [preference preference] [gateway gateway]]
[syslog [upto] log_level]
[mark time]
;
```

Оператор параметров может содержать следующие элементы:

`nosend`

Запрещает системе передавать любые пакеты. Данный параметр тестирует `gated` без реальной передачи информации маршрутизации. Используйте параметр для протоколов RIP и HELLO. Для протокола BGP он еще не реализован, а для OSPF не имеет смысла.

`noresolv`

Запрещает системе использовать систему доменных имен (DNS, Domain Name System) для разрешения имен узлов и адресов. Сбой DNS может приводить к зависанию `gated` при старте. Используйте данный параметр, чтобы предотвратить зависание.

`gendefault [preference preference] [gateway gateway]`

Генерирует маршрут по умолчанию с приоритетом 20 в случаях, когда соседняя для `gated` точка сети работает по протоколу EGP или BGP. В отсутствие определения `gateway` шлюзом в этом маршруте является сама система, маршрут по умолчанию не добавляется в таблицу маршрутизации ядра, а данный параметр используется только для объявления данной сис-

темы шлюзом по умолчанию. Если определение `gateway` присутствует, маршрут по умолчанию добавляется в таблицу маршрутизации ядра и указанный шлюз в этом случае является следующей точкой маршрута. Данный параметр может переопределяться параметром `nogendefault`.

`syslog [upto] log_level`

Предписывает использовать механизм setlogmask для управления журналами `gated`. Чтобы узнать, доступен ли этот механизм в вашей системе, обратитесь к странице setlogmask(3) справочника man.

`mark time`

Предписывает периодически записывать в файл трассировки временную отметку. Аргумент *time* определяет частоту записи таких сообщений. Используйте данный параметр для определения активности `gated`.

Операторы интерфейсов

Операторы интерфейсов определяют параметры настройки сетевых интерфейсов. Список интерфейсов (*interface_list*) определяет интерфейсы, на которые распространяется действие параметров настройки. Интерфейсы в списке могут обозначаться именами (к примеру, le0), именами узлов, IP-адресами либо ключевым словом `all`. Ключевое слово `all` обозначает все интерфейсы системы. Имя интерфейса может относиться к одному интерфейсу или группе интерфейсов. К примеру, имя интерфейса eth0 относится к интерфейсу eth0, тогда как имя le относится ко всем установленным интерфейсам, имена которых начинаются с le (скажем, le0, le1 и le2). Имя узла может использоваться в том случае, если ему соответствует единственный адрес.

Большинство системных администраторов предпочитают использовать IP-адреса для указания интерфейсов. В конце концов, IP-адреса изначально входят в состав TCP/IP, а описываемый файл выполняет настройку маршрутизации TCP/IP.

Кроме того, удаленным системам этот интерфейс известен по его IP-адресу, а не по имени. Наконец, DNS может предоставлять более одного адреса для имени узла, а ОС Unix в будущем сможет допускать использование нескольких адресов для одного интерфейса. IP-адрес – наиболее удачный выбор.

`gated` поддерживает интерфейсы четырех типов: кольцевой (loopback), широковещательный (broadcast), точка-точка (point-to-point), а также NBMA (nonbroadcast multiple access). Все эти интерфейсы описаны в тексте книги, за исключением NBMA. Последний является интерфейсом параллельного доступа, работающим в сетях, не предоставляющих широковещательного доступа. Примерами таких сетей являются Frame Relay и X.25.

`gated` игнорирует любые интерфейсы в списке с некорректными локальными, удаленными или широковещательными адресами либо некорректной маской подсети. `gated` также игнорирует интерфейсы типа точка-точка, у которых совпадают локальный и удаленный адреса. `gated` предполагает, что интерфейсы, не отмеченные ядром как функционирующие (UP), не существуют.

Синтаксис оператора `interfaces`:

```
interfaces {
   options
     [strictinterfaces]
     [scaninterval time]
     [ aliases-nexthop ( primary | lowestip | keepall ) ];
   interface interface_list
     [preference preference]
     [down preference preference]
     [passive]
     [simplex]
     [reject]
     [blackhole]
     [ AS autonomoussystem ];
   define address
     [broadcast address] | [pointopoint address]
     [netmask mask]
     [multicast] ;
 } ;
```

Операторы настройки, предшествующие списку интерфейсов, определяют глобальные параметры:

`strictinterfaces`

Генерирует критическую ошибку, если интерфейс, упомянутый в файле настройки, не найден `gated` при сканировании ядра в момент запуска и не упомянут в операторе `define`. (Описание `define` приводится ниже в данном разделе.) Стандартное поведение `gated` в такой ситуации: сгенерировать предупреждение и продолжить работу.

`scaninterval` *`time`*

Указывает частоту сканирования списка интерфейсов ядра на предмет изменений. По умолчанию в большинстве систем сканирование производится каждые 15 секунд и каждые 60 – в системах, передающих изменения в состоянии интерфейсов через сокеты маршрутизации (например, BSD 4.4). Обратите внимание, что `gated` также сканирует список интерфейсов по сигналу SIGUSR2.

`aliases-nexthop ( primary | lowestip | keepall )`

Указывает адрес следующей транзитной точки, который `gated` устанавливает для маршрутов интерфейса. `primary`, значение по умолчанию, предписывает использовать адрес основного интерфейса в качестве шлюза для маршрута; `lowestip` – предписывает использовать наименьший IP-адрес в качестве адреса следующей транзитной точки; `keepall` сохраняет все маршруты интерфейсов в ядре.

Команда `interface` определяет список интерфейсов (*`interface_list`*) и все параметры, относящиеся к указанным интерфейсам. Для этого оператора доступны следующие параметры:

`preference` *preference*

Устанавливает приоритет для данного интерфейса. Значение *preference* – число из интервала от 0 до 255. `gated` отдает предпочтение маршрутам, пролегающим через интерфейсы с низкими значениями приоритета. По умолчанию значение приоритета для всех собственных сетевых интерфейсов системы равно 0.

`down preference` *preference*

Устанавливает значение приоритета, используемое в случаях, когда `gated` считает, что интерфейс работает неправильно. По умолчанию – 120.

`passive`

Запрещает `gated` снижать приоритет интерфейса, если он работает неправильно. `gated` предполагает, что интерфейс неработоспособен, если перестает получать информацию маршрутизации через этот интерфейс. `gated` выполняет такую проверку, только если интерфейс активно участвует в работе протокола маршрутизации.

`simplex`

Указывает, что `gated` не должен считать пакеты, сгенерированные системой, подтверждением того, что интерфейс работает нормально. Работоспособность интерфейса определяется только на основании пакетов, поступающих от удаленных систем.

`reject | blackhole`

Любое из этих ключевых слов определяет интерфейс как «черную дыру» и используется для установки в ядре заблокированных маршрутов. (Более подробная информация о заблокированных маршрутах содержится в разделе, посвященном управляющим операторам.) Возможность доступна только в системах BSD с установленным псевдоинтерфейсом reject/blackhole.

`AS` *autonomoussystem*

Указывает номер автономной системы, который должен использоваться `gated` при создании AC-вектора пути для данного маршрута. Вы, вероятно, помните, что некоторые протоколы, в частности BGP, связывают с маршрутом AC-путь.

Команда `define` *address* перечисляет интерфейсы, которые могут отсутствовать на момент сканирования `gated` списка интерфейсов ядра при запуске. Для указанного интерфейса (*address*) эта команда имеет приоритет более высокий, чем параметр `strictinterfaces`. Допустимые параметры команды `define`:

`broadcast` *address*

Определяет широковещательный адрес.

`pointopoint` *address*

Определяет локальный адрес для интерфейса точка-точка. (Описание таких интерфейсов содержится в главе 6.) В присутствии данного параметра адрес, указанный оператором `define`, интерпретируется в качестве адреса удаленного узла, а адрес, следующий за ключевым словом `pointopo-`

int, в качестве локального адреса. Не используйте одновременно broadcast и pointopoint в пределах одной команды define.

netmask *mask*

Определяет маску подсети.

multicast

Указывает, что данный интерфейс поддерживает многоадресную передачу.

Операторы определений

Операторы определений – это общие операторы настройки, действие которых распространяется более чем на один протокол. Операторы определений должны предшествовать всем операторам протоколов в файле *gated.conf*. Операторы определений:

autonomoussystem *asn* [loops *n*] ;

Определяет номер автономной системы (*asn*), используемый BGP и EGP. Число loops определяет, сколько раз данная автономная система может фигурировать в AC-пути для протоколов, использующих векторы путей, таких как BGP. Значение *n* по умолчанию равно 1.

routerid *address* ;

Определяет идентификатор маршрутизатора, используемый BGP и OSPF. Используйте адрес основного интерфейса OSPF или BGP. По умолчанию gated использует адрес первого найденного интерфейса.

martians {host *address* [allow]; *address* [mask *mask* | masklen *number*] [allow] ; default [allow] ; } ;

Изменяет список адресов, для которых игнорируется вся информация маршрутизации. Случается, что некорректно настроенная система распространяет очевидно недопустимый адрес пункта назначения. Такие недопустимые адреса, известные как «*пришельцы*» (*martians*), блокируются программами маршрутизации. Данная команда позволяет изменять список адресов-пришельцев. Адрес-пришелец может обозначаться адресом узла при помощи ключевого слова host перед адресом либо адресом сети (достаточно просто указать адрес).

Адрес сети может сопровождаться адресной маской. Маска адреса определяется ключевым словом mask и маской в десятичной записи через точку либо ключевым словом masklen и численным префиксом длины. Адресные маски mask 255.255.0.0 и masklen 16 эквивалентны. В отсутствие адресной маски используется обычная маска сети. Указание адреса в операторе martians приводит к его добавлению в список адресов-пришельцев. Ключевое слово allow используется для удаления адреса из списка пришельцев. После удаления адреса из списка пришельцев он становится доступным для маршрутизации.

gated содержит стандартный список недопустимых адресов-пришельцев. Это список пришельцев по умолчанию. Параметр default allow удаляет все стан-

дартные записи из списка пришельцев и разрешает неограниченную маршрутизацию. Не используйте данный параметр в сети, подключенной к Интернету.

Вот примеры каждого из операторов определений:

```
autonomoussystem 249 ;
routerid 172.16.12.2 ;
martians {
        host 0.0.0.26 ;
        192.168.0.0 masklen 16 allow ; } ;
```

Действия операторов из примера:

- Оператор `autonomoussystem` предписывает `gated` использовать AC-номер 249 для пакетов BGP и EGP.
- Оператор `routerid` предписывает `gated` использовать адрес 172.16.12.2 в качестве идентификатора маршрутизатора для OSPF и BGP.
- Оператор `martians` предотвращает включение в таблицу всех маршрутов к пункту назначения 0.0.0.26, но допускает маршруты к частным IP-адресам из диапазона 192.168.0.0–192.168.255.255.

Операторы протоколов

Операторы протоколов разрешают или запрещают отдельные протоколы и устанавливают параметры их работы. Операторы протоколов следуют за операторами определений и предшествуют статическим операторам. Существует много операторов протоколов, и в любой момент их может стать еще больше. Существуют операторы для различных внутренних и внешних протоколов маршрутизации, а также для некоторых других, протоколами маршрутизации на деле не являющихся.

В этом разделе мы начнем с внутренних протоколов, затем перейдем ко внешним протоколам и закончим специальными «протоколами».

Оператор ospf

```
ospf yes | no | on | off  [{
  defaults {
    preference preference ;
    cost cost ;
    tag [as] tag ;
    type 1 | 2 ;
    inherit-metric; } ;
  exportlimit routes ;
  exportinterval time ;
  traceoptions trace_options ;
  syslog [first count] [every count];
  monitorauthkey key ;
  backbone | area number {
    authtype 0 | 1 | none | simple ;
```

```
    stub [cost cost]  ;
    networks {
     address [mask mask | masklen number] [restrict]  ;
     host address [restrict]  ; } ;
    stubhosts {
     address cost cost ; } ;
    interface interface_list [nonbroadcast] [cost cost]  {
     pollinterval time ;
     routers {
     address [eligible]  ; } ;
     interface_parameters } ;
    virtuallink neighborid router_id transitarea area {
     interface_parameters } ;
} ; } ] ;
```

Оператор `ospf` включает или отключает протокол OSPF (Open Shortest Path First). По умолчанию OSPF отключен. Он включается аргументами `yes` и `on` (выбирайте любой), а отключается аргументами `no` и `off`.

В целях сокращения объема изложения каждый параметр файла *gated.conf* описывается только один раз, если его использование в последующих командах подчиняется тем же правилам. Различия в командах упоминаются особо. К примеру, конструкция `yes | no | on | off` не описана больше нигде в тексте, поскольку используется одинаково во всех операторах для включения или отключения протокола.

Оператор `ospf` содержит множество параметров настройки:

`defaults`

Определяет соглашения по умолчанию, используемые при импортировании маршрутов OSPF из внешней автономной системы и при передаче этих маршрутов другим маршрутизаторам OSPF. Пакеты LSA (link-state advertisement, сообщение о состоянии канала), используемые для объявления таких маршрутов, известны в качестве пакетов ASE (autonomous system external, внешние для автономной системы), поскольку содержат маршруты из внешней автономной системы. См. описание OSPF в главе 7.

`preference` *preference*

Задает приоритет для маршрутов OSPF ASE. По умолчанию 150.

`cost` *cost*

Задает стоимость, используемую для объявления не-OSPF-маршрута в сообщении ASE. По умолчанию 1.

`tag [as]` *tag*

Задает значение тега (*tag*) для сообщений OSPF ASE. Тег не используется протоколом OSPF, но может применяться в правилах экспортирования для фильтрации маршрутов. (См. описание оператора `export` далее в приложении.) Ключевое слово `as` позволяет указывать информацию AC-пути в поле tag.

`type 1 | 2`

Определяет тип сообщений ASE. По умолчанию `type 1`. Сообщения типа 1 содержат маршруты, полученные от внешнего протокола, имеющего метрику, напрямую сравнимую с метрикой OSPF. Эта метрика добавляется к стоимости маршрута до граничного маршрутизатора при объявлении маршрутов. Сообщения ASE типа 2 содержат маршруты, полученные от протокола внешних шлюзов, не имеющего метрики маршрутизации, сравнимой с метрикой OSPF. Такие маршруты объявляются со стоимостью маршрута до граничного маршрутизатора. См. главу 7.

`inherit-metric`

Предписывает `gated` использовать внешнюю метрику для маршрутов ASE, если в операторе `export` метрика не указана.

`exportlimit` *routes*

Задает максимальное число пакетов ASE LSA, рассылаемых одновременно. По умолчанию – 100.

`exportinterval` *time*

Указывает, с какой частотой в сети выполняется одновременная рассылка пакетов ASE LSA. По умолчанию – один раз в секунду.

`traceoptions` *trace_options*

Задает параметры трассировки для отладки OSPF. В дополнение к стандартным флагам трассировки OSPF поддерживает следующие флаги:

`lsabuild`

Отслеживает создание уведомлений о состоянии каналов (сообщений LSA).

`spf`

Отслеживает выбор кратчайших путей (Shortest Path First, SPF).

`hello`

Отслеживает пакеты OSPF HELLO.

`dd`

Отслеживает пакеты OSPF Database Description.

`request`

Отслеживает пакеты OSPF Link-State Request.

`lsu`

Отслеживает пакеты OSPF Link-State Update.

`ack`

Отслеживает OSPF Link-State Ack.

`syslog [first` *number*`] [every` *count*`]`

Определяет параметры перехвата пакетов. `first` указывает число перехватываемых пакетов для каждого типа пакетов OSPF. `every` указывает, с какой частотой выполняется перехват пакетов после перехвата первой

группы. К примеру, если *count* имеет значение 50, перехватывается каждый пятидесятый пакет каждого типа.

`monitorauthkey` *password*

Определяет пароль для запросов `ospf_monitor`. По умолчанию для этих запросов не выполняется проверка подлинности. В присутствии параметра `monitorauthkey` входящие запросы должны содержать указанный пароль.

`backbone | area` *number*

Указывает область OSPF, в которую входит данный маршрутизатор. Каждый маршрутизатор должен принадлежать к определенной области. Если областей несколько, одна из них должна быть магистральной. Магистральная область создается при помощи ключевого слова `backbone`. Все прочие области определяются при помощи ключевого слова `area` и номера области: к примеру, `area 1`. Области OSPF описаны в главе 7. С каждой из областей связан ряд параметров настройки:

`stub [cost` *cost*`]`

Указывает, что данная область является запасной. То есть такой, в которой отсутствуют ASE-маршруты. Если указана стоимость (cost), она используется для объявления маршрута по умолчанию в запасную область.

`networks`

Определяет диапазон сетей внутри области. Определенные диапазоны объявляются в других сетях в качестве суммарных сетевых LSA, а не в качестве межобластных маршрутов. Ключевое слово `restrict` запрещает объявление суммарных LSA. Элементы списка сетей представляются адресами узлов (при помощи ключевого слова `host`, предваряющего каждый адрес) либо адресами сетей (достаточно просто указать адрес). Адрес сети может сопровождаться адресной маской. Маска адреса определяется ключевым словом `mask` и маской в десятичной записи через точку либо ключевым словом `masklen` и числовым префиксом. Адресные маски `mask 255.255.0.0` и `masklen 16` эквивалентны. В отсутствие адресной маски используется обычная маска сети. Данный параметр может сократить объем информации маршрутизации, передаваемой между областями.

`stubhosts`

Перечисляет непосредственно подключенные узлы, объявляемые в качестве доступных с данного маршрутизатора, а также стоимости их маршрутов. Используйте данный параметр для перечисления интерфейсов точка-точка.

`interface` *interface_list* `[nobroadcast] [cost` *cost*`]`

Определяет интерфейсы, используемые OSPF. Если упомянуто ключевое слово `nobroadcast`, интерфейс подключается к сети NBMA (nonbroadcast multiple access). В отсутствие `nobroadcast` интерфейс подключается к широковещательной сети или сети точка-точка. Стоимость интерфейса ука-

зывается при помощи ключевого слова `cost`, например `cost 5`. Стоимость по умолчанию равна 1. Следующие два параметра используются только для интерфейсов NBMA:

`pollinterval` *time*

Задает интервал передачи пакетов OSPF HELLO соседям.

`routers`

Перечисляет адреса всех соседей. Ключевое слово `eligible` указывает, может ли сосед стать назначенным маршрутизатором.

Любые интерфейсы – NBMA и широковещательные – могут использовать следующие параметры:

`enable | disable ;`

Включает или отключает интерфейс.

`retransmitinterval` *time* ;

Задает интервал для повторной передачи уведомлений о состояниях каналов (LSA).

`transitdelay` *time* ;

Задает ожидаемое время, затрачиваемое на передачу обновления LSA через этот интерфейс, в секундах. Значение должно быть больше 0.

`priority` *priority* ;

Определяет приоритет данной системы в контексте выбора назначенного маршрутизатора. *priority* – число из диапазона от 0 до 255. Маршрутизатор с наивысшим приоритетом становится назначенным. Маршрутизатор с нулевым приоритетом не может стать назначенным маршрутизатором. Назначенные маршрутизаторы описаны в главе 7.

`hellointerval` *time* ;

Задает интервал передачи пакетов HELLO, в секундах.

`routerdeadinterval` *time* ;

Задает интервал ожидания, по истечении которого соседу присваивается статус нефункционирующего. *time* – это максимальное число секунд ожидания данным маршрутизатором пакета HELLO от соседа.

`auth [ none | simple` *password* `| md5` *key* `] ;`

Указывает способ проверки подлинности пакетов OSPF. `none` – проверка подлинности не выполняется. `simple` – парольная проверка подлинности. Пароль (*password*) – от одной до восьми десятичных цифр, разделенных точками, от одного до восьми шестнадцатеричных байтов, предваренных символами 0x, либо от одного до восьми символов в двойных кавычках. `md5` – проверка подлинности по алгоритму MD5. *key* – допустимый криптографический ключ MD5.

`virtuallink neighborid` *router_id* `transitarea` *area*

Определяет виртуальный канал для магистральной области. *router_id* – идентификатор удаленного маршрутизатора на другом конце виртуально-

го канала. Транзитной областью должна являться одна из областей, существующих в данной системе. Все стандартные параметры интерфейсов, описанные выше, могут применяться для виртуальных каналов.

Оператор rip

```
rip yes | no | on | off [ {
   broadcast ;
   nobroadcast ;
   nocheckzero ;
   preference preference ;
   defaultmetric metric ;
   query authentication [none | [simple | md5 password]]  ;
   interface interface_list
     [noripin] | [ripin]
     [noripout] | [ripout]
     [metricin metric]
     [metricout metric]
     [version 1 | 2 [multicast | broadcast]]
     [[secondary] authentication [none | [simple | md5 password]] ;
   trustedgateways gateway_list ;
   sourcegateways gateway_list ;
   traceoptions trace_options ; } ] ;
```

Оператор `rip` включает или отключает RIP. По умолчанию RIP работает. Параметры оператора `rip`:

`broadcast`

Принуждает `gated` выполнять широковещательную передачу пакетов RIP, даже если в системе всего один сетевой интерфейс. По умолчанию обновления RIP не передаются широковещательно, если в системе только один сетевой интерфейс, и передаются, если интерфейсов больше; то есть широковещательная передача не выполняется узлами, но выполняется маршрутизаторами.

`nobroadcast`

Запрещает `gated` широковещательную передачу пакетов обновлений RIP, даже если в системе более одного сетевого интерфейса. В присутствии уточнения `sourcegateways` маршруты по-прежнему передаются напрямую указанному шлюзу. См. описание `sourcegateways` далее в этом разделе.

`notcheckzero`

Указывает, что `gated` не должен отвергать поступающие пакеты RIP версии 1 с нулевыми зарезервированными полями. Блокировка таких пакетов – стандартный подход.

`preference` *`preference`* ;

Устанавливает приоритет `gated` для маршрутов, полученных по протоколу RIP. По умолчанию такие маршруты получают значение приоритета 100.

`defaultmetric` *`metric`* ;

Определяет метрику, используемую при объявлении по RIP маршрутов, полученных по другим протоколам. По умолчанию *`metric`* имеет значение 16, что в случае RIP указывает на непригодный к использованию маршрут. То есть по умолчанию маршруты, полученные по другим протоколам, не объявляются RIP в качестве действующих. Используйте более низкие значения только в случае, если необходимо объявлять с такой метрикой все маршруты, полученные по другим протоколам.

`query authentication [none | [simple | md5` *`key`*`]] ;`

Определяет способ проверки подлинности пакетов запросов, исходящих не от маршрутизатора. По умолчанию `none`. В случае `simple` ключ *`key`* является 16-байтовым паролем. В случае `md5` ключ является 16-байтовым значением, используемым совместно с содержимым пакета для вычисления криптографической контрольной суммы MD5.

`interface` *`interface_list`*

Указывает интерфейсы, через которые работает RIP, а также параметры настройки этих интерфейсов. Список интерфейсов (*`interface_list`*) может содержать имена интерфейсов, имена узлов, IP-адреса либо ключевое слово `all`. Допустимые параметры:

`noripin`

Предписывает системе игнорировать пакеты RIP, поступающие через данный интерфейс. По умолчанию принимаются пакеты RIP через все интерфейсы, за исключением кольцевого (loopback).

`ripin`

Предписывает системе принимать пакеты RIP, поступающие через данный интерфейс. Режим по умолчанию.

`noripout`

Запрещает системе посылать пакеты RIP через данный интерфейс. По умолчанию пакеты RIP посылаются через все широковещательные и нешироковещательные интерфейсы при работе в широковещательном режиме. См. описание параметра `nobroadcast` выше.

`ripout`

Предписывает системе посылать пакеты RIP через данный интерфейс. Режим по умолчанию.

`metricin` *`metric`*

Указывает метрику RIP, используемую для маршрутов, полученных через данный интерфейс. По умолчанию к метрике интерфейса ядра добавляется единица, что дает число транзитных участков в RIP по умолчанию. Если указана метрика, она используется в качестве абсолютного значения и не добавляется к метрике ядра.

`metricout`

Указывает метрику RIP, добавляемую к маршрутам, распространяемым через данный интерфейс. По умолчанию 0. Данный параметр позволяет только увеличивать метрику.

`version 1 | 2 [multicast | broadcast]`

Указывает версию RIP, используемую для передачи обновлений через данный интерфейс. Доступны версии RIP 1 и RIP 2. RIP 1 используется по умолчанию. Если указана версия RIP 2 и присутствует поддержка многоадресной передачи IP-пакетов, полноценные пакеты версии 2 передаются посредством многоадресной передачи. Если многоадресная передача недоступна, осуществляется широковещательная передача пакетов версии 2, совместимых с версией 1. Ключевое слово `multicast`, принимаемое по умолчанию, предписывает именно такое поведение. Ключевое слово `broadcast` указывает, что через данный интерфейс следует осуществлять широковещательную передачу пакетов версии 2, совместимых с версией 1, даже если доступна многоадресная передача IP-пакетов. Ни одно из упомянутых ключевых слов не используется для версии 1.

`[secondary] authentication [none | simple | md5 key]`

Для данного интерфейса определяет способ проверки подлинности для RIP версии 2. По умолчанию проверка подлинности не выполняется (`none`). Указание `simple` сопровождается ключом (*`key`*), представленным 16-байтовым паролем. Для `md5` ключ является 16-байтовым значением, используемым совместно с содержимым пакета для вычисления криптографической контрольной суммы MD5. Ключевое слово `secondary` указывает, что параметр определяет вид вторичной аутентификации. Пакеты всегда передаются с использованием первичного метода аутентификации. Вторичный метод аутентификации определяется только для входящих пакетов. Входящие пакеты проверяются как первичным, так и вторичным механизмом и лишь после этого отвергаются как недопустимые.

`trustedgateways` *`gateway_list`* ;

Определяет список шлюзов, от которых RIP принимает обновления. Список шлюзов (*`gateway_list`*) – это просто список имен узлов или IP-адресов. По умолчанию все шлюзы сети маршрутизатора являются доверенными и могут поставлять информацию маршрутизации. Если использован оператор `trustedgateways`, принимаются обновления только от перечисленных в этом операторе шлюзов.

`sourcegateways` *`gateway_list`* ;

Определяет список шлюзов, которым пакеты RIP передаются напрямую. По умолчанию пакеты RIP передаются ряду систем общей сети посредством широковещательной или многоадресной передачи. Данный оператор предписывает RIP осуществлять индивидуальную передачу пакетов перечисленным шлюзам.

`traceoptions` *`trace_options`*

Определяет параметры трассировки для RIP. RIP поддерживает большинство стандартных параметров трассировки, а также следующие параметры трассировки пакетов:

`packets`

Отслеживает все пакеты RIP.

`request`

Отслеживает пакеты запросов сведений RIP, такие как REQUEST, POLL и POLLENTRY.

`response`

Отслеживает все пакеты RIP RESPONSE.

`other`

Отслеживает пакеты любых других типов RIP.

Оператор isis

```
isis on | off {
     [ area areaid ; ]
     [ area auth simple key ; ]
     [ domain auth simple key ; ]
     [ domain-wide on | off ; ]
     [ export-defaults ; ]
     [ export-defaults level 1 | 2 ; ]
     [ export-defaults metric metric | inherit ; ]
     [ export-defaults metric-type internal | external ; ]
     [ external preference preference ; ]
     [ level 1 | 2 | 1 and 2 ; ]
     [ interface name | address [ {
        [ enable | disable ; ]
        [ auth simple key ; ]
        [ csn-interval interval [ level 1 | 2 | 1 and 2 ] ; ]
        [ dis-hello-interval interval [ level 1 | 2 | 1 and 2 ] ; ]
        [ encap [ iso | ip ] ; ]
        [ hello-interval interval [ level 1 | 2 | 1 and 2 ] ; ]
        [ hello-multiplier number [ level 1 | 2 | 1 and 2 ] ; ]
        [ lsp-interval interval ; ]
        [ level 1 | 2 | 1 and 2 ; ]
        [ max-burst number ; ]
        [ metric metric [ level 1 | 2 | 1 and 2 ] ; ]
        [ passive on | off ; ]
        [ priority priority [ level 1 | 2 | 1 and 2 ] ; ]
        [ retransmit-interval interval ; ]
     } ] ; ]
     [ overload-bit on | off ; ]
     [ preference preference ; ]
     [ psn-interval intervalt ; ]
     [ require-snp-auth on | off ; ]
     [ ribs unicast | unicast multicast ; ]
```

```
        [ spf-interval interval ; ]
        [ inet6 on | off ; ]
        [ summary-originate [ inet | inet6 ] {
            [network (mask mask | masklen n ) metric cost-value ; ]
        } ; ]
        [ summary-filter [ inet | inet6 ] {
            [network mask mask | masklen number ; ]
        } ; ]
        [ systemid systemid ; ]
        [ traceoptions traceoptions ; ]
        [ config-time seconds ; ]
        [ es-config-time seconds ; ]
        [ hold-time seconds ; ]
    };
```

Оператор `isis` включает протокол IS-IS. По умолчанию протокол выключен. Допустимые для оператора `isis` параметры:

`area` *areaid*

Добавляет адреса областей к тем, которые определены автоматически. Адреса областей IS-IS настраиваются автоматически на основе тех физических маршрутов, на которых работает IS-IS. Оператор `area` позволяет добавить до трех областей.

`area auth simple` *key*

Включает проверку подлинности для маршрутизации первого уровня и позволяет указать ключ (*key*). Формат ключа: от одной до восьми десятичных цифр, разделенных точками, от одного до восьми шестнадцатеричных байтов, предваренных символами 0x, либо от одного до восьми символов в двойных кавычках. Такой же формат ключа используется и в других параметрах оператора `isis`.

`domain auth simple` *key*

Включает проверку подлинности для маршрутизации второго уровня и позволяет указать ключ.

`export-defaults level 1 | 2`

Устанавливает уровень протокола для экспортируемых маршрутов. По умолчанию маршрутизатор уровня 1 экспортирует маршруты на уровне 1, а маршрутизатор уровня 2 поддерживает уровни 1 и 2.

`export-defaults metric` *metric* `| inherit`

Определяет метрику, по умолчанию используемую для маршрутов, экспортируемых в формате IS-IS из других протоколов. По умолчанию используется метрика, уже содержащаяся в маршруте (такое поведение предписывает ключевое слово `inherit`).

`export-defaults metric-type internal | external`

Определяет тип метрики, используемой для маршрутов, экспортируемых в формате IS-IS из других протоколов. По умолчанию – `internal`.

`external preference` *preference*

Задает приоритет для внешних маршрутов, полученных по протоколу IS-IS. По умолчанию 151.

`level 1 | 2 | 1 and 2`

Устанавливает уровень протокола для данной промежуточной системы. Система уровня `1` является внутриобластным маршрутизатором. В такой системе не могут существовать интерфейсы уровня 2. Система уровня `2` – межобластной маршрутизатор, и она не может иметь интерфейсов уровня 1. Системы уровня `1 и 2` могут иметь интерфейсы уровня 1, уровня 2, а также уровней 1 и 2. Кроме того, отдельные параметры, сопутствующие настройке протокола, могут указываться как принадлежащие уровню 1, 2 или уровням 1 и 2, в зависимости от уровня установки параметра для системы, поддерживающей уровни 1 и 2. По умолчанию принимается значение `1 и 2`.

`interface` *name* | *address*

Указывает интерфейсы, с которыми работает IS-IS. По умолчанию IS-IS работает через все интерфейсы. Для каждого интерфейса могут использоваться следующие параметры:

`enable | disable`

Включает (`enable`) или отключает (`disable`) интерфейс. По умолчанию `enable`.

`auth simple` *key*

Включает проверку подлинности и позволяет указать ключ для данного интерфейса.

`csn-interval` *interval* `[ level 1 | 2 | 1 and 2 ]`

Устанавливает интервал многоадресной передачи пакетов CSN, если система выбрана в качестве DIS (Designated Intermediate System, назначенная промежуточная система). *interval* может принимать значения от 1 до 100 секунд.

`dis-hello-interval` *interval* `[level 1 | 2 | 1 and 2]`

Устанавливает интервал передачи сообщений hello, если данная система выбрана в качестве DIS. *interval* может принимать значения от 1 до 100 секунд.

`encap [ ip | iso ]`

Позволяет выбрать тип инкапсуляции. По умолчанию – `ip`.

`hello-interval` *interval* `[ level 1 | 2 | 1 and 2 ]`

Задает интервал посылки пакетов hello через данный интерфейс. *interval* может принимать значения от 1 до 300 секунд.

`hello-multiplier` *number* `[ level 1 | 2 | 1 and 2 ]`

Определяет число отсутствующих пакетов hello, позволяющее считать, что сосед не функционирует. Так, если значение *number* равно 3 и

в течение временного интервала, отведенного на поступление трех пакетов hello, не получен ни один пакет, соседу присваивается статус нефункционирующего. Значение *number* лежит в интервале от 1 до 100.

`lsp-interval` *interval*

Задает интервал передачи пакетов LSP через этот интерфейс.

`level 1 | 2 | 1 and 2 ;`

Задает уровень протокола для данного интерфейса.

`max-burst` *number*

Задает максимальное число пакетов, передаваемых за один прием.

`metric` *metric* `[ level 1 | 2 | 1 and 2 ]`

Задает стоимость для интерфейса.

`passive on | off`

Указывает, является интерфейс активным или пассивным.

`priority` *priority* `[ level 1 | 2 | 1 and 2 ]`

Устанавливает значение приоритета, используемое при выборе системы DIS. Значение *priority* лежит в диапазоне от 1 до 127.

`retransmit-interval` *interval*

Задает интервал повторной передачи пакетов через данный интерфейс.

`overload-bit on | off`

Разрешает или запрещает использование бита переполнения.

`preference` *preference*

Устанавливает значение приоритета `gated` для маршрутов IS-IS. По умолчанию 11.

`psn-interval` *interval*

Определяет частоту посылки пакетов PSN данной системой. *interval* может принимать значения от 1 до 20 секунд.

`ribs unicast | unicast multicast`

Задает базовый формат информации маршрутизации для маршрутов IS-IS. По умолчанию `unicast`.

`spf-interval` *interval*

Определяет длительность ожидания изменений, по истечении которого производится пересчет таблицы маршрутизации. *interval* может принимать значения от 1 до 60 секунд.

`inet6 on | off`

Включает поддержку маршрутизации IPv6.

`summary-originate`

Определяет, как маршруты уровня 1 суммируются в данной системе при передаче сведений на второй уровень маршрутизации. `summary-originate` используется только для маршрутизаторов уровня 2. *network* указывает

полученный адрес первого уровня, а маска сети (обозначенная маской или длиной маски) объединяет маршруты.

`summary-filter`

Определяет, как маршруты уровня 1 суммируются при распространении их по каналам маршрутизации второго уровня. `summary-filter` используется только для маршрутизаторов уровня 2.

`systemid` *`systemid`*

Определяет системный идентификатор IS-IS. Если идентификатор не указан, используется раздел системного идентификатора NSAP-адреса первой цепи.

`traceoptions` *`traceoptions`*

Определяет параметры трассировки для IS-IS. По умолчанию трассировка не выполняется.

Оператор bgp

```
bgp yes | no | on | off [{
    preference preference ;
    defaultmetric metric ;
    traceoptions trace_options ;
    group type external peeras as_number
       | internal peeras as_number
       | igp peeras as_number proto proto
       | routing peeras as_number proto proto interface interface_list
      | test peeras as_number {
      allow {
         address mask mask | masklen number
         all
         host address } ;
      peer address
        [metricout metric]
        [localas as_number]
        [nogendefault]
        [gateway address]
        [preference preference]
        [preference2 preference]
        [lcladdr address]
        [holdtime time]
        [version number]
        [passive]
        [sendbuffer number]
        [recvbuffer number]
        [indelay time]
        [outdelay time]
        [keep all | none]
        [showwarnings]
        [noaggregatorid]
        [keepalivesalways]
```

```
        [v3asloopokay]
        [nov4asloop]
        [logupdown]
        [ttl ttl]
        [traceoptions trace_options]  ; }
    ; }] ;
```

Данный оператор включает или отключает BGP. По умолчанию BGP отключен. Значение *preference* по умолчанию равно 170. По умолчанию BGP не объявляет метрику. В отличие от метрики RIP, метрика BGP не играет особой роли в выборе оптимального маршрута. Метрика BGP – это просто произвольное 16-разрядное значение, которое может использоваться в качестве одного из критериев при выборе маршрута. Для указания метрики, объявляемой BGP с маршрутами, можно использовать оператор `defaultmetric`.

Параметры трассировки могут определяться как для всех, так и для отдельных систем, использующих BGP. BGP поддерживает большинство стандартных параметров трассировки, а также следующий:

`packets`

Отслеживает все пакеты BGP. Отслеживает пакеты BGP OPEN, BGP UPDATE, BGP KEEPALIVE.

BGP-системы должны состоять в одной группе. Оператор `group` объявляет группу, определяет, какие системы входят в эту группу, а также указывает тип группы. Операторов группы может быть несколько, однако сочетание типа и номера автономной системы в каждом из них должно быть уникальным. Существует пять типов групп:

`group type external peeras` *as_number*

Указывает, что BGP следует работать в качестве классического протокола внешних шлюзов. Системы, перечисленные в данной группе, образуют внешнюю автономную систему. На все входящие и исходящие маршруты распространяется действие всех правил маршрутизации.

`group type internal peeras` *as_number*

Указывает, что BGP используется для распространения маршрутов в пользу внутренней группы, не работающей с традиционным протоколом внутренних шлюзов. Маршруты, полученные от внешних BGP-систем, повторно объявляются в этой группе с исходной метрикой.

`group type igp peeras` *as_number* `proto` *proto*

Указывает, что BGP используется для распространения атрибутов пути в пользу внутренней группы, работающей с протоколом внутренних шлюзов. BGP объявляет AC-путь, исходный пункт, а также транзитные необязательные атрибуты в случае, если атрибуты пути получены из меток IGP. *proto* – это имя протокола внутренних шлюзов, например `proto ospf`.

`group type routing peeras` *as_number* `proto` *proto* `interface` *interface_list*

Указывает, что BGP используется в пределах системы для доставки внешних маршрутов, тогда как протокол внутренних шлюзов использу-

ется для доставки только внутренних маршрутов. Обычно маршруты, полученные BGP от внешних автономных систем, записываются в таблицу маршрутизации, откуда их читает и распространяет по локальной автономной системе внутренний протокол. Для групп этого типа BGP распространяет внешние маршруты самостоятельно, а внутренний протокол ограничивается распространением только внутренних маршрутов локальной автономной системы. *proto* – имя внутреннего протокола.

`group type test peeras` *as_number*

Указывает, что в группу входят тестовые системы. Вся информация маршрутизации, которой обмениваются тестовые системы, удаляется.

Предложение `group` содержит уточнения `peer`. В группу может входить любое число уточнений `peer`. Системы явным образом обозначаются посредством оператора `peer` либо неявно – посредством оператора `allow`.

`allow`

Любая система с адресом из указанного диапазона входит в группу. Ключевое слово `all` обозначает все возможные адреса. Ключевое слово `host` предваряет адрес отдельного узла. Диапазоны адресов определяются парами адресов и масок. Маска сети определяется ключевым словом `mask` и маской в десятичной записи через точку либо ключевым словом `masklen` и десятичным числом, обозначающим длину префикса. Все параметры для указанных равных систем должны быть определены в предложении `group`.

`peer` *address*

Система с указанным адресом входит в группу.

Уточнение BGP `peer` допускает следующие параметры в предложении `group`, влияющие на все системы группы:

`metricout` *metric*

Определяет основную метрику для маршрутов, передаваемых данной системе. Имеет приоритет больший, чем метрика по умолчанию, метрика группы, а также любая метрика, определенная правилом экспортирования.

`localas` *as_number*

Определяет номер автономной системы для локальной системы (asn). По умолчанию используется номер asn, определенный в операторе `autonomoussystem`.

`nogendefault`

Запрещает `gated` генерировать маршрут по умолчанию, когда BGP общается с данной системой, даже если в операторе инструкции `options` фигурирует параметр `gendefault`.

`gateway` *address*

Обозначает следующий транзитный шлюз, через который маршрутизируются пакеты для данной системы. Используйте только в случае, если сосед находится не в той же сети, что и локальная система. Необходимость в данном параметре возникает редко.

`preference` *preference*

Определяет значение приоритета, используемое для маршрутов, полученных от данной системы, что позволяет `gated` отдавать предпочтение маршрутам, полученным от определенной системы либо группы систем.

`preference2` *preference*

Определяет «второе» значение приоритета. В случае одинаковых значений приоритета для выбора будет использоваться второе значение. По умолчанию – 0.

`lcladdr` *address*

Определяет адрес локального интерфейса, используемого для сообщения с соседней системой.

`holdtime` *time*

Определяет число секунд ожидания системой сообщения keepalive, обновления или уведомления о закрытии соединения. Значение передается этой системе в поле Hold Time сообщения BGP Open. Значение должно равняться 0 (keepalive-сообщения не посылаются) или быть не меньше 3.

`version` *version*

Указывает версию протокола BGP, которую следует использовать с данной системой. По умолчанию согласование версии происходит при создании соединения. В настоящее время поддерживаются версии 2, 3 и 4.

`passive`

Указывает, что `gated` должен ожидать получения сообщения OPEN от данной системы. По умолчанию `gated` периодически посылает сообщения OPEN, пока не получит ответ от системы.

`sendbuffer` *buffer_size*
`recvbuffer` *buffer_size*

Определяют размеры буферов отправки и получения. По умолчанию – 65 535 байт, и это максимальный размер. Данные параметры не используются в нормально работающих системах.

`indelay` *time*
`outdelay` *time*

Осуществляет «демпфирование маршрута». `indelay` предписывает принимать только маршруты, стабильные в течение *time* секунд. `outdelay` определяет, сколько секунд маршрут должен храниться в базе данных маршрутизации `gated`, прежде чем будет экспортирован данной системе. Значение по умолчанию для обоих параметров – 0, то есть демпфирование маршрута не действует. Используйте этот механизм только в случае, если изменения маршрутов происходят настолько быстро, что это приводит к нестабильности таблицы маршрутизации.

`keep all`

Предписывает системе сохранять маршруты, полученные от данной системы, даже если AC-пути маршрутов содержат номер локальной автоном-

ной системы. Обычно маршруты, содержащие номер локальной АС, удаляются, поскольку могут создавать петли маршрутизации.

`showwarnings`

Предписывает системе генерировать предупреждения в случае событий, которые обычно «молча игнорируются» (скажем, получение дубликатов маршрута).

`noaggregatorid`

Устанавливает routerid в атрибуте aggregator в значение 0. По умолчанию это значение является идентификатором маршрутизатора. Используйте данный параметр, чтобы запретить маршрутизатору создание объединенных маршрутов с АС-путями, отличными от созданных другими маршрутизаторами автономной системы.

`keepalivesalways`

Предписывает системе посылать сообщение keepalive, даже если его с успехом можно заменить обновлением. Используется для обеспечения взаимодействия с некоторыми маршрутизаторами.

`v3asloopokay`

Разрешает распространение маршрута с петлей в АС-пути (номер АС фигурирует в пути более одного раза) для внешних систем версии 3.

`nov4asloop`

Запрещает распространение маршрута с петлей в АС-пути для внешних систем версии 4. Позволяет избежать передачи таких маршрутов системам, ошибочно передающим их соседям версии 3.

`logupdown`

Регистрирует вход и выход BGP-системы из состояния ESTABLISHED.

`ttl` *`ttl`*

Определяет IP TTL (время жизни IP-пакета) для локальных соседей. По умолчанию имеет значение 1. Используйте данный параметр, если локальный сосед удаляет пакеты с *`ttl`*, равным 1. Не все ядра Unix позволяют указывать TTL для TCP-соединений.

Параметры трассировки BGP мы уже рассматривали ранее.

Оператор egp

```
egp yes | no | on | off [{
    preference preference ;
    defaultmetric metric ;
    packetsize maxpacketsize ;
    traceoptions trace_options ;
    group [peeras as_number] [localas as_number] [maxup number] {
    neighbor address
    [metricout metric]
    [preference preference]
```

```
        [preference2 preference]
        [ttl ttl]
        [nogendefault]
        [importdefault]
        [exportdefault]
        [gateway address]
        [lcladdr address]
        [sourcenet network]
        [minhello | p1 interval]
        [minpoll | p2 interval]
        [traceoptions trace_options] ; }
    ; }] ;
```

Данный оператор включает или отключает EGP. По умолчанию EGP отключен. Метрика по умолчанию для маршрутов, распространяемых по EGP, равна 255, а значение приоритета для маршрутов, полученных по EGP, составляет 200.

Аргумент `packetsize` определяет максимальный допустимый размер исходящего или входящего пакета EGP. *`maxpacketsize`* определяет размер в байтах. По умолчанию значение равно 8192 байт. Получив пакет, размер которого превышает *`maxpacketsize`*, `gated` удаляет его, однако *`maxpacketsize`* увеличивается до размера большего пакета, чтобы последующие пакеты не пришлось удалять.

Оператор `traceoptions` определяет параметры трассировки для EGP. Трассировка может выполняться для протокола EGP или отдельного EGP-соседа. Параметры трассировки EGP:

`packets`

Отслеживает все пакеты EGP.

`hello`

Отслеживает пакеты EGP HELLO/I-HEARD-U.

`acquire`

Отслеживает пакеты EGP ACQUIRE/CEASE.

`update`

Отслеживает пакеты EGP POLL/UPDATE.

Оператор `egp` содержит два предложения: `group` и `neighbor`. EGP-соседи должны входить в группу, а все соседи одной группы должны принадлежать одной автономной системе. Используйте предложение `group` для определения параметров группы EGP-соседей. Значения, указанные в предложении `group`, действуют для всех предложений `neighbor` в группе. Предложений `group` может быть несколько. Следующие параметры допустимы в предложении `group`:

`peeras`

Указывает номер автономной системы, в которую входят участники группы. Если номер не указан, он будет получен от соседей.

`localas`

Указывает номер автономной системы для локальной системы. По умолчанию используется номер asn из оператора `autonomoussystem`.

`maxup`

Определяет число EGP-соседей, с которыми должен наладить связь `gated`. По умолчанию налаживается связь со всеми перечисленными соседями.

Предложение `neighbor` определяет параметры одного EGP-соседа. Единственной обязательной частью предложения является аргумент `address`, то есть имя или IP-адрес соседа. Все прочие параметры – необязательные. Все необязательные параметры могут фигурировать и в предложении `group` – если требуется распространить их действие на всех соседей. Параметры предложения `neighbor`:

`metricout` *`metric`*

Используется для всех маршрутов, передаваемых данному соседу. Переопределяет значение `defaultmetric` из оператора `egp`, но только для данного конкретного соседа.

`preference` *`preference`*

Определяет значение приоритета, используемое для маршрутов, полученных от этого соседа, что позволяет `gated` отдавать предпочтение маршрутам определенного соседа или группы соседей.

`preference2` *`preference`*

Определяет «второе» значение приоритета, которое используется для разрешения ситуации с одинаковыми значениями приоритета. По умолчанию равно 0.

`ttl` *`ttl`*

Определяет IP TTL для локальных соседей. По умолчанию равно 1. Используйте данный параметр, если локальный сосед удаляет пакеты с TTL, равным 1.

`nogendefault`

Запрещает `gated` генерировать маршрут по умолчанию при общении EGP с данным соседом, даже если в операторе `options` фигурирует параметр `gendefault`.

`importdefault`

Разрешает системе принять маршрут по умолчанию, включенный в EGP-обновление данного соседа. По умолчанию такой маршрут игнорируется.

`exportdefault`

Предписывает системе включать маршрут по умолчанию в EGP-обновления, посылаемые данному соседу. Обычно маршрут по умолчанию не включается в EGP-обновление.

`gateway` *address*

Указывает следующий транзитный шлюз, через который передаются пакеты для данного соседа. Используйте данный параметр, если сосед не входит в ту же сеть, что и локальная система. Потребность в данном параметре возникает редко.

`lcladdr` *address*

Указывает адрес локального интерфейса, через который происходит общение с соседом.

`sourcenet` *network*

Изменяет запрашиваемую сеть в пакетах EGP POLL. По умолчанию это общая сеть. Однако если сосед находится не в той же сети, что локальная система, здесь необходимо указать адрес сети соседа. Обычно данный параметр не требуется. Не используйте его, если локальная система и сосед находятся в одной сети.

`minhello | p1` *time*

Устанавливает интервал передачи пакетов EGP HELLO. По умолчанию – 30 секунд. Если сосед не ответил на три пакета HELLO, система прекращает попытки установить с ним связь. Установка большего интервала дает соседу больше шансов отреагировать. Интервал может указываться в секундах, в формате минуты:секунды или часы:минуты:секунды. Например, трехминутный интервал можно указать как 180 (секунд), 3:00 (минут) или 0:3:00 (ноль часов, три минуты). Вместо ключевого слова `minhello` можно использовать `p1`.

`minpoll | p2` *time*

Устанавливает интервал посылки соседям пакетов POLL. По умолчанию – 120 секунд. Если сосед не отреагировал на три запроса, система приходит к выводу, что он не работает, после чего из таблицы маршрутизации удаляются все маршруты, полученные от данного соседа. Если сосед подвергается высокой нагрузке и не может отвечать на быстрые запросы, это может привести к крайней нестабильности таблицы маршрутизации. Более длительные интервалы опроса позволяют иметь более стабильную таблицу маршрутизации, но с замедленной реакцией. Данный интервал также задается в секундах, в формате минуты:секунды или часы:минуты:секунды.

Оператор smux

```
smux yes | no | on | off [ {
    port port ;
    password string ;
    traceoptions trace_options ; } ] ;
```

Данная команда пришла на смену оператору `snmp`, который использовался в предшествующих версиях `gated`. Команда `smux` определяет, сообщает ли `gated` о своем состоянии управляющему программному модулю SNMP. SNMP – не протокол маршрутизации, и не запускается данной командой. Программное

обеспечение SNMP должно запускаться независимо. Данный оператор лишь определяет, передает ли gated управляющей программе сведения о своем состоянии. По умолчанию принимается значение on, то есть gated уведомляет SNMP о своем состоянии.

Оператор smux поддерживает три параметра:

port *port*

Изменяет порт SNMP, используемый gated. По умолчанию демон SNMP прослушивает порт 199.

password *string*

Включает парольную проверку подлинности и позволяет указать используемый пароль.

traceoptions *trace_options*

Отслеживает взаимодействие gated и демона SNMP. Допустимые параметры трассировки: packets, send, receive.

Оператор redirect

```
redirect yes | no | on | off [{
    preference preference ;
    interface interface_list [noredirects | redirects]  ;
    trustedgateways gateway_list ;
    traceoptions trace_options ; } ] ;
```

Данный оператор определяет, разрешено ли перенаправлениям ICMP изменять таблицу маршрутизации ядра. Он не предотвращает посылку перенаправлений системой, а только их прием. Указание no или off предписывает gated препятствовать воздействию перенаправлений на таблицу маршрутизации ядра. Вспомните, что ICMP являются частью IP, а потому перенаправления могут устанавливаться в таблицу маршрутизации ядра до того, как их обнаружит gated. Если отключить перенаправления, gated будет удалять маршруты с перенаправлениями из таблицы маршрутизации. По умолчанию перенаправления ICMP разрешены на узлах, пассивно принимающих информацию от протоколов внутренней маршрутизации, и запрещены на шлюзах, активно участвующих в работе таких протоколов.

По умолчанию значение приоритета для маршрута, полученного из ICMP-перенаправления, равно 30 и может изменяться посредством параметра preference. Оператор interface управляет обработкой перенаправлений на уровне отдельных интерфейсов. Перенаправления игнорируются по указанию noredirects и разрешаются указанием redirects (поведение по умолчанию). Оператор trustedgateways разрешает перенаправления на уровне отдельных шлюзов. По умолчанию перенаправления принимаются от всех маршрутизаторов локальной сети. В присутствии оператора trustedgateways принимаются перенаправления только от шлюзов, перечисленных в списке *gateway_list*. *gateway_list* – это просто список имен узлов или адресов. Параметры *trace_options* в операторе traceoptions – стандартные параметры трассировки gated.

Оператор icmp

```
icmp {
    traceoptions trace_options ; }
```

В некоторых системах gated принимает все сообщения ICMP, но обрабатывает только пакеты ICMP-перенаправлений. Такое поведение управляется оператором redirect. В будущем может быть реализована и другая функциональность. В настоящее время оператор icmp используется только для трассировки сообщений ICMP. Параметры трассировки, допустимые в операторе icmp:

packets

Отслеживает все пакеты ICMP.

redirect

Отслеживает все пакеты ICMP REDIRECT.

routerdiscovery

Отслеживает все пакеты ICMP ROUTER DISCOVERY.

info

Отслеживает информационные пакеты ICMP.

error

Отслеживает пакеты ICMP-ошибок.

Оператор routerdiscovery

Протокол обнаружения маршрутизаторов (Router Discovery Protocol) уведомляет узлы о маршрутизаторах, доступных в сети. Он является альтернативой статическим маршрутам, протоколам маршрутизации и перенаправлениям ICMP для узлов, которым просто требуется знать адрес маршрутизатора, используемого по умолчанию. Протокол RDP реализован в виде сервера, работающего на маршрутизаторе, и клиента, работающего на узле. Работу серверной (маршрутизатор) и клиентской (узел) программы обеспечивает gated.

Сначала рассмотрим оператор настройки сервера:

```
routerdiscovery server yes | no | on | off [{
    traceoptions trace_options ;
    interface interface_list
      [minadvinterval time]
      [maxadvinterval time]
      [lifetime time] ;
    address interface_list
      [advertise | ignore]
      [broadcast | multicast]
      [ineligible | preference preference]  ;
} ] ;
```

Оператор routerdiscovery поддерживает трассировку – как для клиента, так и для сервера. Флаг трассировки state может быть использован для отсле-

живания переходов состояний конечного автомата. Отслеживание пакетов обнаружения маршрутизаторов включается посредством оператора ICMP.

Предложение `interface` определяет физические интерфейсы и параметры их работы. Только физические интерфейсы могут фигурировать в предложении `interface`. Адреса указываются в адресных предложениях, описанных ниже. Параметры `interface`:

`maxadvinterval` *time*

Указывает максимальный интервал времени для передачи уведомлений о маршрутизаторах. Не должен быть меньше 4 секунд и больше 30:00 минут. По умолчанию равен 10:00 минутам (600 секундам).

`minadvinterval` *time*

Указывает минимальный интервал времени для передачи уведомлений о маршрутизаторах. Не должен быть меньше 3 секунд и больше `maxadvinterval`. По умолчанию – 0,75 значения `maxadvinterval`.

`lifetime` *time*

Указывает срок действия адресов, полученных клиентами в уведомлении о маршрутизаторах. Должен быть больше `maxadvinterval` и не больше, чем 2:30:00 (два часа тридцать минут). По умолчанию 3 интервала `maxadvinterval`.

Предложение `address` определяет используемые IP-адреса и их параметры. Параметры предложения `address`:

`advertise | ignore`

`advertise` указывает, что адрес следует включать в уведомления о маршрутизаторах (поведение по умолчанию). `ignore` указывает, что адрес распространять не следует.

`broadcast | multicast`

`broadcast` указывает, что адрес следует включать в широковещательные уведомления о маршрутизаторах, поскольку некоторые системы сети не поддерживают многоадресную передачу. Таково поведение по умолчанию, если маршрутизатор не поддерживает многоадресную передачу.

`multicast` указывает, что адрес следует включать только в многоадресные уведомления. Если система не поддерживает многоадресную передачу, адрес не распространяется.

`ineligible | preference` *preference*

Задает значение приоритета для адреса, используемое для выбора маршрутизатора по умолчанию. *preference* – 32-разрядное знаковое целое. Чем выше значение, тем более предпочтительным является адрес. Обратите внимание: это не значение приоритета `gated`, это значение, распространяемое протоколом обнаружения маршрутизаторов.

Ключевое слово `ineligible` назначает шестнадцатеричное значение приоритета 80000000, что означает, что адрес не может использоваться в ка-

честве адреса маршрутизатора по умолчанию. Узлы используют такие запрещенные адреса для проверки ICMP-перенаправлений.

Чтобы оператор routerdiscovery заработал, узлы должны иметь клиентскую часть routerdiscovery. Клиентская часть также реализуется gated и настраивается оператором routerdiscovery client.

Оператор routerdiscovery для клиента

```
routerdiscovery client yes | no | on | off [{
   traceoptions trace_options ;
   preference preference ;
   interface interface_list
     [enable | disable | multicast]
     [quiet | solicit]  ;
} ] ;
```

Клиенту доступны те же параметры трассировки, что и серверу, однако прочие параметры отличаются. Вот полный перечень параметров клиента:

`preference` *preference* ;

Задает значение приоритета для маршрутов по умолчанию, полученных от routerdiscovery. По умолчанию – 55. Данное значение является значением приоритета gated.

`interface` *interface_list*

Указывает интерфейсы, используемые routerdiscovery.

`enable | disable | multicast`

Включает или отключает routerdiscovery для интерфейса. По умолчанию enable. multicast принуждает gated использовать многоадресную передачу для обнаружения маршрутизаторов. Если многоадресная передача недоступна, обнаружение маршрутизаторов не производится. Обычно gated использует многоадресную или широковещательную передачу, исходя из возможностей интерфейса.

`broadcast | multicast`

Указывает, как должны выполняться запросы на поиск маршрутизаторов – посредством широковещательной или многоадресной передачи через интерфейс. По умолчанию для таких запросов используется многоадресная передача (если возможно), в противном случае – широковещательная. Если указано ключевое слово multicast, а многоадресная передача недоступна, запросы на поиск маршрутизаторов не посылаются. Обычно, если данный параметр не определен, gated принимает правильные решения.

`quiet | solicit`

Указывает, передаются ли запросы на поиск маршрутизаторов через данный интерфейс. Значение solicit, принимаемое по умолчанию, предписывает посылать такие запросы. quiet предписывает принимать уведомления о маршрутизаторах, но не посылать запросы на их поиск.

Оператор kernel

```
kernel {
   options
     [nochange]
     [noflushatexit]
     [protosync];
   remnantholdtime  ;
   routes number ;
   flash
     [limit number]
     [type interface | interior | all]  ;
   background
     [limit number]
     [priority flash | higher | lower]  ;
   traceoptions trace_options ; } ;
```

Оператор `kernel` регулирует взаимодействие `gated` и ядра системы.

`options`

Определяет три допустимые параметра настройки:

`nochange`

Ограничивает действия `gated` удалениями и добавлениями. Используется с ранними версиями кода сокетов маршрутизации, в которых некорректно функционирует операция изменения.

`noflushatexit`

Запрещает удаление маршрутов при завершении работы. Обычно в момент завершения работы удаляются маршруты без пометки «retain». Используется для ускорения запуска в системах, работающих с тысячами маршрутов.

`protosync`

Заменяет поле протокола ядра текущим значением протокола `gated`.

`remnantholdtime`

Хранит маршруты, прочитанные из таблицы ретрансляции ядра при запуске, до трех минут, за исключением случаев, когда выполняется переопределение маршрутов.

`routes` *`number`*

Определяет максимальное число маршрутов, устанавливаемых `gated` в ядро. По умолчанию число маршрутов в таблице ретрансляции ядра не ограничивается.

`flash`

Выполняет настройку параметров, используемых для мгновенных обновлений (flash update). Когда изменяется маршрут, демон передает ядру такое обновление.

`limit` *number*

Устанавливает максимальное число маршрутов, обрабатываемых за одно мгновенное обновление. По умолчанию – 20. Значение –1 приводит к единовременной обработке всех изменений. Крупные обновления могут замедлять работу для протоколов, чувствительных ко времени обработки. Значение 20 – разумный компромисс.

`type interface | interior | all`

Определяет тип маршрутов, обрабатываемых при мгновенных обновлениях. По умолчанию происходит установка только маршрутов интерфейсов. `interior` указывает, что должны устанавливаться также внутренние маршруты, а `all` – что следует обрабатывать внутренние и внешние маршруты. Параметр `flash limit -1 all` приводит к установке всех маршрутов в мгновенном обновлении, что копирует поведение предыдущих версий `gated`.

`background`

Выполняет настройку параметров фоновой обработки. Поскольку обычно в мгновенных обновлениях устанавливаются только маршруты интерфейсов, большая часть маршрутов обрабатывается в фоновом режиме отдельными заданиями.

`limit` *number*

Устанавливает число маршрутов, обрабатываемых одним заданием. По умолчанию – 120.

`priority flash | higher | lower`

Устанавливает приоритет для обработки пакетных обновлений. По умолчанию `lower`, то есть пакетные обновления обрабатываются с меньшим приоритетом, нежели мгновенные обновления. Чтобы обрабатывать обновления ядра с тем же приоритетом, что и мгновенные, укажите значение `flash`.

Многочисленные параметры трассировки действуют и для интерфейса ядра, поскольку во многих случаях интерфейс интерпретируется как протокол маршрутизации. Параметры трассировки командной строки `symbols` и `iflist` позволяют получать информацию о ядре. Параметры трассировки оператора `kernel`:

`remnants`

Отслеживает маршруты, прочитанные из ядра при запуске `gated`.

`request`

Отслеживает операции Add/Delete/Change, выполняемые `gated` для ядра.

Прочие операторы трассировки действуют только в системах, применяющих сокеты маршрутизации для обмена информацией маршрутизации с ядром.

`info`

Отслеживает информационные сообщения, полученные через сокет маршрутизации.

`routes`

Отслеживает маршруты, полученные от ядра или переданные ядру.

`redirect`

Отслеживает сообщения перенаправления, полученные от ядра.

`interface`

Отслеживает сообщения о состоянии интерфейса, полученные от ядра.

`other`

Отслеживает все прочие сообщения, полученные от ядра.

Статические операторы

Операторы `static` определяют статические маршруты, используемые `gated`. Один оператор `static` может содержать определения нескольких маршрутов. Операторы `static` в файле *gated.conf* следуют за операторами протоколов и предшествуют управляющим операторам. Для `gated` статическими являются маршруты, определенные посредством операторов `static`. При этом, в отличие от маршрутов из таблицы статической маршрутизации, `static`-маршруты могут заменяться другими маршрутами, имеющими более высокие значения приоритета.

Структура оператора `static`:

```
static {
   [default] | [[host] address [mask mask | masklen n]] gateway gateways
     [interface interface_list]
     [preference preference]
     [retain]
     [reject]
     [blackhole]
     [noinstall] ;
   address [mask mask | masklen n] interface interface
     [preference preference]
     [retain]
     [reject]
     [blackhole]
     [noinstall] ;
} ;
```

Оператор `static` состоит из двух различных предложений. Как правило, используется предложение, начинающееся с ключевого слова `gateway`. Данное предложение содержит информацию, схожую с той, что предоставляет команда `route`. Статический маршрут определяется как адрес пункта назначения, доступного через шлюз. Формат этого предложения следующий:

`[default] | [[host] address [mask mask | masklen number]] gateway gateways`

Определяет статический маршрут, пролегающий через один или более шлюзов. Пункт назначения указывает ключевое слово `default` (для марш-

рута по умолчанию) либо его адрес. Адрес пункта назначения может предварять ключевое слово `host` – в случае, если это адрес узла. За адресом может следовать маска адреса. Маска адреса определяется ключевым словом `mask` и маской в десятичной записи через точку либо ключевым словом `masklen` и численной длиной префикса. Перечисленные шлюзы должны располагаться в сети, непосредственно подключенной к узлу. Допустимые параметры настройки:

`interface` *interface_list*

Шлюзы из списка *gateway_list* должны быть прямо доступны через один из перечисленных интерфейсов.

`preference` *preference*

Устанавливает приоритет `gated` для данного статического маршрута. По умолчанию – 60.

`retain`

Предотвращает удаление данного статического маршрута при нормальном завершении работы демона. Обычно в таблице ретрансляции ядра сохраняются только маршруты интерфейсов. Используйте для сохранения маршрутизации в моменты, когда `gated` не работает.

`reject`

Блокирует маршрут. Пакеты, отправленные по заблокированному маршруту, удаляются, а источнику пакетов направляется сообщение о том, что адрес недоступен («unreachable»). Не все ядра поддерживают блокировку маршрутов.

`blackhole`

Определяет маршрут как «черную дыру». Такой маршрут похож на заблокированный. Разница заключается в том, что источнику не посылается сообщение о недостижимости адреса («unreachable»).

`noinstall`

Предписывает системе объявлять данный маршрут через протоколы маршрутизации, но не устанавливать его в таблицу ретрансляции ядра.

Второе предложение оператора `static` начинается с ключевого слова `interface` вместо `gateway`. Используйте данное предложение, только если работаете с одиночной физической сетью, обладающей более чем одним сетевым адресом (что встречается весьма редко). Команда `ifconfig` обычно создает лишь один пункт назначения на каждый интерфейс. Описываемая специальная форма оператора `static` добавляет для интерфейса дополнительные пункты назначения.

address [mask *mask* | masklen *number*] interface *interface*

Параметры `preference`, `retain`, `reject`, `blackhole` и `noinstall` интерпретируются так, как описано выше.

По умолчанию статическим маршрутам назначается приоритет 60, что делает статические маршруты более предпочтительными, чем маршруты из ряда

других источников маршрутизации. Если необходимо сделать прочие типы маршрутов более предпочтительными, воспользуйтесь аргументом preference оператора static для увеличения значения приоритета. (Помните, что более высокий приоритет означает менее предпочтительный маршрут.)

В следующем примере содержится определение статического маршрута через шлюз 172.16.12.1. Установлено значение приоритета 125, чтобы сделать маршруты, полученные от RIP, более предпочтительными, чем данный статический маршрут:

```
static  {
    default gateway 128.66.12.1 preference 125 ;  } ;
```

Управляющие операторы

Управляющие операторы определяют правила маршрутизации. Когда администраторы слышат слова «правила маршрутизации» и «маршрутизация, регулируемая правилами», то часто предполагают, что речь идет о внутренних механизмах протокола маршрутизации.

На деле правила маршрутизации существуют вне протокола маршрутизации – в файле настройки. Правила определяют, какие маршруты принимаются и какие маршруты распространяются. В gated данной цели служат управляющие операторы import и export. Оператор import определяет, какие маршруты принимаются и от каких источников. Оператор export определяет, какие маршруты распространяются, исходя из источника маршрутов и протокола распространения.

Операторы import и export для создания правил маршрутизации задействуют значения приоритета gated, метрики маршрутизации, фильтры маршрутизации, а также AC-пути. Значениями приоритета и метриками управляют следующие ключевые слова:

restrict

Запрещает импортировать (в случае команды import) или экспортировать (в случае команды export) маршруты. Данное ключевое слово позволяет заблокировать определенный маршрут.

preference *preference*

Определяет значение приоритета (*preference*), используемое при сравнении данного маршрута с другими. Значения приоритета используются при установке маршрутов, а не при их распространении.

metric *metric*

Определяет метрику, используемую при распространении маршрута.

Фильтры маршрутов отбирают маршруты по адресу пункта назначения. Среди прочего, фильтры маршрутов используются для «пришельцев», в операторах import и export. Маршрут соответствует наиболее определенному фильтру. Повторение пункта назначения, маски или модификаторов в раз-

личных фильтрах является ошибкой. Фильтры маршрутов обозначаются при помощи конструкции:[1]

address [mask *mask* | masklen *number*] [exact | refines | between n1 and n2]

Определяет диапазон адресов при помощи адреса и маски адреса. Маска адреса определяется ключевым словом mask и маской в десятичной записи через точку либо ключевым словом masklen и численной длиной префикса. В отсутствие определения маски используется обычная маска сети. Доступны следующие параметры:

exact

Отбор сетей, но не подсетей или узлов этих сетей.

refines

Отбор подсетей и/или узлов сетей, но не собственно сетей.

between n1 and n2

Отбор адресов, в которых фильтру соответствуют не менее *n1* и не более *n2* бит.

all

Отбор всех возможных адресов.

default

Отбор только маршрута по умолчанию.

host *address*

Отбор узла с указанным адресом.

Фильтр маршрутов, отбирающий все адреса сети с номером 192.168.12.0, а также отдельный узел 10.104.19.12, содержит:

```
192.168.12.0 masklen 24 ; host 10.104.19.12 ;
```

Если фильтрация маршрутов в операторе import или export не задействована, действие оператора распространяется на все маршруты указанного источника. Если определены фильтры, импортируются или экспортируются только маршруты, этим фильтрам соответствующие.

Архитектура протокола пограничных шлюзов BGP поддерживает маршрутизацию, регулируемую правилами. Главная особенность BGP в том, что это протокол вектора пути. Операторы import и export позволяют использовать вектор AC-пути для проведения в жизнь правил маршрутизации.

AC-путь перечисляет все автономные системы маршрута и указывает полноту пути. Каждая автономная система, через которую проходит маршрут, добавляет свой AC-номер в начало AC-пути.

---

[1] Фильтры маршрутов могут содержать дополнительные параметры. В операторах import это значение приоритета, в операторах export – метрика. Значения приоритета и метрики описаны выше.

«Происхождение» (origin) пути указывает его полноту. Происхождение igp указывает, что маршрут получен от протокола внутренней маршрутизации и является, вероятнее всего, полным. Происхождение egp указывает, что маршрут получен от протокола внешней маршрутизации, не поддерживающего АС-пути (например, EGP), так что путь, вероятнее всего, не является полным.

Когда информация пути определенно не является полной, используется происхождение incomplete. Любое из происхождений может быть указано в операторах import и export, а значит – использовано в правилах маршрутизации. Ключевое слово any указывает, что правило действует для всех вариантов происхождения.

АС-путь может также фигурировать в управляющих операторах в качестве регулярного выражения АС-пути.[1] Такие регулярные выражения позволяют фильтровать маршруты, исходя из номеров автономных систем в АС-путях, связанных с маршрутами.

Регулярное выражение АС-пути состоит из номеров автономных систем и специальных операторов. В табл. B.1 описаны операторы АС-пути. Оператор АС-пути действует на операнд АС-пути, то есть на номер автономной системы, точку (.), обозначающую номера всех автономных систем, либо выражение, заключенное в фигурные скобки.

Таблица B.1. Операторы АС-пути

Символ	Значение
{m,n}	Не менее m и не более n повторений
{m}	Ровно m повторений
{m,}	m или более повторений
*	0 или более повторений
+	1 или более повторений
?	0 или 1 повторение
aspath_term|*aspath_term*	АС-операнд слева или справа от символа |

Простое регулярное выражение АС-пути может выглядеть так:

```
import proto bgp aspath 164+ origin any restrict ;
```

Оно блокирует все маршруты, в векторах путей которых присутствует одно или более вхождений номера автономной системы 164.

Оператор import

Формат оператора import может меняться в зависимости от протокола-источника. Формат операторов import для протоколов внешних шлюзов:

[1] Определение регулярных выражений АС-путей содержится в RFC 1164.

```
import proto bgp | egp autonomoussystem as_number
   [restrict] |
   [[preference preference] {
   route_filter [restrict | (preference preference)]] ; } ;
import proto bgp aspath aspath_regexp
   origin any | igp | egp | incomplete
   [restrict] |
   [[preference preference] {
   route_filter [restrict | (preference preference)]] ; } ;
```

Импортирование BGP и EGP может управляться номером автономной системы. BGP также позволяет управлять импортированием посредством регулярных выражений AC-путей. Маршруты, отвергнутые правилами маршрутизации, сохраняются в таблице маршрутизации с отрицательными значениями приоритета. Отрицательное значение приоритета не позволяет установить маршрут в таблицу ретрансляции или экспортировать его в другие протоколы. Такая обработка отвергнутых маршрутов частично снимает необходимость прерывать и восстанавливать сеанс работы, если при перенастройке изменяются правила маршрутизации.

Формат операторов `import` для протоколов RIP и redirect:

```
import proto rip | redirect
 [interface interface_list | gateway gateway_list]
 [restrict] |
 [[preference preference] {
  route_filter [restrict | (preference preference)]] ; } ;
```

Данный оператор указывает, какие маршруты импортируются, исходя из протокола-источника, интерфейса и шлюза. Порядок старшинства – от общего (protocol) к частному (gateway). В отличие от BGP и EGP, эти протоколы не сохраняют отвергнутые маршруты, поскольку имеют короткие интервалы обновления.

Параметр `preference` не используется для RIP. В RIP отсутствует механизм выбора наиболее предпочтительного из двух маршрутов одного и того же протокола. В RIP используются метрики протоколов.

Формат оператора `import` для протокола OSPF:

```
import proto ospfase [tag ospf_tag] [restrict] |
 [[preference preference] {
  route_filter [restrict | (preference preference)]] ; } ;
```

Вследствие природы OSPF доступно управление импортированием только ASE-маршрутов. Более того, ограничение импортирования маршрутов OSPF ASE возможно, только если система функционирует в качестве пограничного маршрутизатора AC. Такое ограничение требует присутствия `export ospfase` наряду с оператором `import ospfase`. Используйте пустой оператор `export` для управления импортированием маршрутов ASE, когда маршруты ASE не экспортируются. (См. следующий раздел «Оператор export».) Если указан тег (tag), оператор `import` действует только на маршруты, обладающие таким

тегом. Маршруты OSPF ASE, отвергнутые в соответствии с правилами, сохраняются в таблице с отрицательными значениями приоритета.

Маршруты OSPF импортируются в таблицу маршрутизации gated со значением приоритета 10. Это значение не используется для выбора между маршрутами OSPF ASE; данной цели служит стоимость маршрута OSPF.

Оператор export

По синтаксису оператор export схож с оператором import, а значения многих параметров идентичны. Важное различие между этими операторами: импортирование маршрутов подчинено информации, полученной от источника, тогда как экспортирование подчинено источнику и адресату одновременно. Операторы export определяют, куда передаются маршруты, а также откуда они происходят. Пункт назначения информации о маршруте обозначается уточнением proto в начале оператора export. Источник маршрутов указывается в списке экспорта (*export list*).

Синтаксис оператора export зависит от протокола. Чтобы распространять маршруты по протоколам EGP и BGP, воспользуйтесь следующей конструкцией:

```
export proto bgp | egp as as_number
 [restrict] |
 [[metric metric] {
  export_list ; }] ;
```

Маршруты экспортируются посредством EGP и BGP в указанную автономную систему. restrict блокирует экспорт в AC. Могут присутствовать допустимые метрики BGP или EGP. Если отсутствует список экспорта, экспортируются только прямые маршруты локальных интерфейсов. Если используется список экспорта, все экспортируемые параметры должны быть указаны явным образом.

Для распространения маршрутов по RIP используйте синтаксис:

```
export proto rip
  [interface interface_list | gateway gateway_list]
  [restrict] |
  [[metric metric] {
  export_list ; }] ;
```

Маршруты, экспортируемые посредством RIP, могут передаваться через конкретный интерфейс или на определенный шлюз. Укажите значение *metric*, если намереваетесь экспортировать статические маршруты или созданные системой маршруты по умолчанию. Параметр metric используется только при экспортировании не-RIP-маршрутов по протоколу RIP.

Если отсутствует список экспорта, RIP экспортирует прямые маршруты и маршруты RIP. Если используется список экспорта, все экспортируемые параметры должны быть указаны явным образом.

Для распространения маршрутов по OSPF используйте конструкцию:

```
export proto osfpase [type 1 | 2] [tag ospf_tag]
 [restrict] |
 [[metric metric] {
  export_list ; }] ;
```

gated может экспортировать только маршруты OSPF ASE. Существует два типа маршрутов OSPF ASE, 1 и 2. Они описаны в главе 7 и выше в этом приложении. Значение по умолчанию для типа указывается в операторе ospf protocol, но здесь может быть переопределено. Аргумент *ospf_tag* – это произвольное 32-разрядное число, используемое для фильтрации информации маршрутизации. Значение по умолчанию данного параметра указывается в операторе ospf protocol, но здесь может быть переопределено.

Источник маршрутов, распространяемых протоколом, определяется списком экспорта. Каждая из перечисленных выше команд содержит параметр списка экспорта. Синтаксис списка, подобно синтаксису этих команд, изменяется в зависимости от протокола-источника маршрутов. Описанные выше команды определяют протоколы, используемые для распространения маршрутов. Описанные ниже списки экспорта определяют протоколы, от которых исходят маршруты. Наибольшая путаница со списком экспорта происходит по причине того, что его синтаксис практически идентичен описанному выше. И в том и в другом случае присутствуют определения протоколов, автономных систем, интерфейсов, шлюзов и т. д. В первом случае мы определяем протоколы, интерфейсы и прочие адресаты, которым передаются маршруты, а в списке экспорта – протоколы, интерфейсы и прочие источники, от которых исходят маршруты.

Чтобы экспортировать маршруты, полученные по протоколам BGP и EGP, воспользуйтесь таким синтаксисом списка экспорта:

```
export proto bgp | egp autonomoussystem as_number
 [restrict [noagg]] |
 [[metric metric] {
  route_filter [restrict | metric metric] ; }] ;
```

Таким образом обозначаются маршруты, полученные по BGP или EGP от определенной автономной системы. На маршруты могут накладываться ограничения, для них могут указываться метрики – на основе сопоставления номера AC-источника или фильтра маршрутов. Использование noagg совместно с restrict предотвращает отбор фильтром любых объединенных маршрутов.

При настройке BGP gated назначает всем маршрутам AC-путь. Для внутренних маршрутов AC-путь включает происхождение igp и не содержит автономных систем (текущая AC добавляется при экспортировании маршрута). Для маршрутов EGP AC-путь включает происхождение egp и исходную AC в качестве AC-пути. Для маршрутов BGP используется AC-путь, полученный по BGP. Если вы работаете с BGP, экспортированием всех маршрутов можно управлять при помощи AC-пути. Синтаксис:

```
proto proto | all
 aspath aspath_regexp origin any | igp | egp | incomplete
 [restrict] |
 [[metric metric] {
  route_filter [restrict | metric metric] ; }] ;
```

Источником маршрутов может быть любой протокол (*proto*) или все протоколы (all). Импортированием маршрутов можно управлять, сравнивая их AC-пути с регулярным выражением AC-пути (*aspath_regexp*) или их адреса с фильтром маршрутов (*route_filter*). Фильтры маршрутов и регулярные выражения AC-путей описаны выше.

Чтобы экспортировать маршруты, полученные по RIP, используйте следующий синтаксис списка экспорта:

```
proto rip
 [interface interface_list | gateway gateway_list]
 [restrict] |
 [[metric metric] {
  route_filter [restrict | metric metric] ; }] ;
```

Экспортированием маршрутов RIP можно управлять на основе интерфейса-источника, шлюза-источника или же фильтра маршрутов.

Чтобы экспортировать маршруты, полученные по OSPF, используйте следующий синтаксис списка экспорта:

```
proto ospf | ospfase
 [restrict] |
 [[metric metric] {
  route_filter [restrict | metric metric] ; }] ;
```

Экспортированием маршрутов OSPF и OSPF ASE можно управлять на основе протокола и фильтра маршрутов. Кроме того, экспортирование маршрутов OSPF может управляться параметром *tag* при использовании следующей конструкции:

```
proto proto | all tag tag
   [restrict] |
   [[metric metric] {
    route_filter [restrict | metric metric] ; }] ;
```

В OSPF и RIP версии 2 существует поле tag. Для всех прочих протоколов *tag* всегда имеет значение 0. Выбор маршрутов может происходить на основе содержимого поля tag.

Существуют иные источники маршрутов, не являющиеся в действительности протоколами маршрутизации, и для этих источников также могут определяться списки экспорта. Ниже приводятся конструкции для таких источников.

```
proto direct | static | kernel
 [interface interface_list]
```

```
 [restrict] |
 [[metric metric] {
  route_filter [restrict | metric metric] ; }] ;
```

Экспортированием таких маршрутов можно управлять на основе «протокола»-источника и интерфейса-источника. В данном случае «протоколами» являются маршруты к прямым интерфейсам, статические маршруты либо маршруты, полученные от ядра.

```
proto default | aggregate
 [restrict] |
 [[metric metric] {
  route_filter [restrict | metric metric] ; }] ;
```

Экспортированием таких маршрутов можно управлять только на основе «протокола»-источника. `default` относится к маршрутам, созданным параметром `gendefault`. `aggregate` относится к маршрутам, созданным операторами объединения, о которых пойдет речь в следующем разделе.

Операторы объединения

Объединение маршрутов используется в региональных и национальных сетях для сокращения числа распространяемых маршрутов. Тщательное планирование позволяет крупным поставщикам сетевых услуг объявлять для клиентских сетей лишь несколько объединенных маршрутов вместо сотен. Объединение маршрутов – основная причина выделения непрерывных адресных блоков CIDR.

В большинстве случаев не приходится распространять сотни маршрутов. Однако речь может идти о внеклассовом адресе, состоящем из нескольких адресов класса C, и тогда необходимо объяснить `gated`, как работать с таким адресом. Более старые версии `gated` автоматически генерируют объединенный маршрут в естественную сеть, пользуясь прежними классами A, B и C. То есть адрес интерфейса 192.168.16.1 приводил к созданию маршрута в сеть 192.168.16.0. В мире внеклассовой междоменной маршрутизации такое действие может оказаться некорректным. `gated` не выполняет объединение маршрутов, если не получит явного предписания оператором `aggregate`:

```
aggregate default | address [[mask mask | masklen number] [bgp]]
  [preference preference] [brief] {
  proto proto
    [as as_number | tag tag | aspath aspath_regexp]
    [restrict] |
    [[preference preference] {
      route_filter [restrict | (preference preference)]] ; } ;
```

Оператор `aggregate` имеет ряд параметров:

`bgp`

Объединение производится согласно правилам протокола BGP.

`preference` *preference;*

Задает значение приоритета для конечного объединенного маршрута. По умолчанию 130.

`brief`

Указывает, что AC-путь объединенного маршрута должен быть наибольшим общим AC-путем. По умолчанию AC-путь является суммой всех составляющих его AC-путей.

`proto` *proto*

Предписывает выполнять объединение только для маршрутов, полученных по указанному протоколу. Аргумент *proto* может указывать на любой функциональный протокол, включая «протоколы» `direct`, `static` и `kernel`, описанные в предшествующем разделе; `all` – для всех доступных протоколов; либо `aggregate` для объединения всех прочих маршрутов.

`as` *as_number*

Предписывает выполнять объединение только для маршрутов, полученных от указанной автономной системы.

`tag` *tag*

Предписывает выполнять объединение только для маршрутов с указанным тегом.

`aspath` *aspath_regexp*

Предписывает выполнять объединение только для маршрутов, соответствующих указанному AC-пути.

`restrict`

Указывает маршруты, не участвующие в объединении.

Маршруты, соответствующие фильтрам маршрутов, могут входить в объединенный маршрут. Маршрут может входить только в более общее объединение. Каждый маршрут может входить только в один объединяющий маршрут, однако объединяющий маршрут может являться составной частью более общего объединения.

Незначительной вариацией объединения является создание маршрута по определенным условиям. Чаще всего применяется для создания маршрута по умолчанию на основе маршрута от системы к соседней магистральной сети. Задача решается посредством оператора `generate`:

```
generate default | address [mask mask | masklen number]
  [preference preference]
  [brief] {
  proto proto
    [as as_number | tag tag | aspath aspath_regexp]
    [restrict] |
    [[preference preference] {
    route_filter [restrict | preference preference]] ; } ;
} ;
```

Многие параметры оператора `generate` повторяют параметры оператора `aggregate` и описаны ранее в этом приложении.

C

named, справочник

В данном приложении содержится подробное описание синтаксиса named, команд и файлов, используемых для настройки демона. Оно является, прежде всего, справочником и должно использоваться совместно с информацией, представленной в главе 8. Это приложение будет полезно всем администраторам доменов.

Команда named

На стороне сервера работой DNS заведует демон сервера имен named. Синтаксис команды named:[1]

```
named [-d уровень] [-p порт] [[-b|c] файл_настройки] [-q -r -f -v] [-u пользователь]
[-g группа] [-t путь] [-w путь] [файл_настройки]
```

При использовании в командной строке named доступны следующие ключи:

-d *уровень*

Записывает отладочную информацию в файл *named.run*. Аргумент *уровень* – число от 1 до 11. Чем больше значение, тем более подробную информацию named записывает в файл, но даже при уровне, равном 1, файл *named.run* растет очень быстро. Используя отладку, обращайте внимание на размер файла *named.run* и применяйте команду ndc notrace или сигнал SIGUSR2 для закрытия файла, если он станет слишком большим. Обработка сигналов – тема следующего раздела.

Чтобы получать от named сообщения об ошибках, нет необходимости включать отладку посредством ключа -d. named выводит сообщения об

[1] В системах Sun вместо named используется имя in.named.

ошибках на консоль и сохраняет их в файле *messages*, даже при выключенной отладке. Ключ `-d` позволяет получать дополнительную отладочную информацию.

`-p` *порт*

Указывает порт UDP/TCP, используемый в работе `named`. *порт* — номер порта для подключения к удаленному серверу имен. В отсутствие ключа `-p named` работает через стандартный порт (53). Поскольку порт 53 является широко известным, смена номера порта делает сервер недоступным для стандартных программных пакетов. Как следствие, ключ `-p` используется только для отладки.

`-b` *файл_настройки* или `-c` *файл_настройки*

Указывает файл, в котором хранятся настройки `named`. По умолчанию файлом настройки является */etc/named.conf*, однако ключи `-b` и `-c` позволяют администратору указать другой файл. Обратите внимание, что сами ключи `-b` и `-c` необязательные. Если имя файла настройки не начинается с дефиса, флаги `-b` и `-c` можно опустить. Любое имя файла, фигурирующее в строке команды `named`, считается именем файла настройки, как видно из последнего элемента синтаксиса.

`-q`

Регистрирует все поступающие запросы. Использование данного вида регистрации в журнале требует компиляции `named` с параметром QRYLOG.

`-r`

Отключает рекурсию. В результате сервер предоставляет ответ, только если он является компетентным для зоны, и не обращается с запросом к другим серверам или зонам.

`-f`

Запускает `named` как приложение первого плана. Обычно `named` выполняется как демон, в фоновом режиме.

`-v`

Выводит номер версии. Использование ключа `-v` не приводит к старту `named`.

`-u` *пользователь*

Устанавливает идентификатор пользователя, с полномочиями которого сервер работает после инициализации. По умолчанию `named` работает с полномочиями пользователя root.

`-g` *группа*

Устанавливает идентификатор группы, с полномочиями которой `named` работает после инициализации. По умолчанию используется идентификатор master-группы пользователя, с полномочиями которого выполняется `named`.

`-t` *путь*

Определяет путь к каталогу, который `named` использует при запуске chroot.

`-w` *путь*

Определяет рабочий каталог `named`. По умолчанию рабочим является текущий каталог. Данный параметр можно переопределить в файле настройки (параметр `directory`).

Обработка сигналов

`named` обрабатывает следующие сигналы:

SIGHUP

Предписывает `named` повторно прочитать файл *named.conf* и перезагрузить базу данных сервера имен. `named` продолжает работу с новыми настройками. Если компиляция `named` выполнялась с параметром FORCED_RELOAD, данный сигнал предписывает вторичным серверам имен получить зону с основного сервера. Действие сигнала идентично действию команды `ndc reload`.

SIGINT

Предписывает `named` записать образ кэша в файл *named_dump.db*. Файл образа содержит всю доменную информацию, известную локальному серверу. Файл начинается перечнем корневых серверов и отмечает все домены кроме корневого, о которых что-либо известно локальному серверу. Изучив этот файл, вы получите полную информацию о том, что узнал сервер. Действие сигнала идентично действию команды `ndc dumpdb`.

SIGUSR1

Включает отладку; каждый последующий сигнал SIGUSR1 повышает уровень отладки. Отладочная информация записывается в файл *named.run*, так же как в случае ключа командной строки `-d`. Отладку не обязательно нужно включать ключом `-d`, чтобы сигнал SIGUSR1 сработал. SIGUSR1 позволяет включать отладку при появлении подозрений на проблемы, не останавливая `named` для перезапуска сервера с параметром `-d`. Действие данного сигнала идентично действию команды `ndc trace`.

SIGUSR2

Отключает отладку и закрывает файл *named.run*. После передачи сигнала SIGUSR2 можно изучить файл *named.run* или удалить его, если размер стал слишком большим. Действие данного сигнала идентично действию команды `ndc notrace`.

Дополнительно `named` может обрабатывать и некоторые другие сигналы. Поддержка следующих сигналов требует указания соответствующих параметров при компиляции `named`:

SIGILL

Записывает данные статистики в файл *named.stats*. `named` следует компилировать с параметром –DSTATS.

SIGSYS

Записывает данные профилирования в каталог, определенный параметром `directory` в файле *named.conf*. `named` следует компилировать с поддержкой профилирования.

SIGTERM

Производит запись файлов главной и подчиненной баз данных сервера. Используется для сохранения данных, измененных динамическими обновлениями, перед остановкой системы. `named` следует компилировать с поддержкой динамических обновлений.

SIGWINCH

Переключает регистрацию всех поступающих запросов посредством `syslogd`. `named` следует компилировать с параметром QRYLOG. Действие сигнала идентично действию команды `ndc querylog`.

Команды настройки named.conf

Файл *named.conf* определяет настройки сервера имен и указывает `named`, откуда получать информацию базы данных DNS. В BIND 8 доступны команды настройки `key`, `acl`, `options`, `logging`, `zone`, `server`, `controls` и `trusted-keys`. В BIND 9 доступна дополнительно команда `view`.

Помимо этих команд настройки в BIND 8 и BIND 9 существует оператор `include`, используемый для загрузки внешних файлов, содержащих любые допустимые команды настройки. Например:

```
include /var/named/keys
```

копирует содержимое файла */var/named/keys*, который может содержать команды `key` и `trusted-key`, в файл *named.conf*.

Оператор key

Оператор `key` назначает внутренние имена, используемые для указания методов аутентификации. Операторы `key` обычно встречаются ближе к началу файла настройки, поскольку ссылки на еще не определенные значения недопустимы. Синтаксис оператора для BIND версий 8 и 9:

```
key key_id {
  algorithm algorithm_id;
  secret secret_string;
};
```

key_id

Имя, назначенное методу аутентификации.

algorithm_id

Алгоритм аутентификации.

secret_string

Используемый алгоритмом ключ в кодировке base64.

Оператор acl

Команда `acl` определяет список отбора адресов и назначает ему имя для последующего использования в файле настройки. Ссылки на еще не определенные значения недопустимы. Синтаксис команды `acl` для BIND версий 8 и 9:

```
acl name {
   address_match_list
};
```

name

Внутреннее имя списка. По умолчанию существуют следующие имена:

`any`

Все возможные адреса.

`none`

Ни один из адресов.

`localhost`

Все адреса локального узла.

`localnet`

Все адреса, у которых сетевая часть такая же, как у адреса локального узла.

address_match_list

Список IP-адресов в десятичной записи через точку с необязательными префиксами адресных масок. Восклицательный знак (`!`) перед адресом исключает его из списка («не сопоставлять адрес»). Список *address_match_list* может также содержать имена списков управления доступом, определенных ранее, включая и четыре стандартных имени.

Оператор trusted-keys

Оператор `trusted-keys` позволяет вручную указать открытый ключ для удаленного домена, если этот ключ не может быть получен по сети безопасным способом. Синтаксис оператора `trusted-keys` BIND версий 8 и 9:

```
trusted-keys {
   domain_name flags protocol algorithm key; [...]
};
```

domain_name

Имя удаленного домена.

flags, protocol, algorithm

Атрибуты метода аутентификации, используемого удаленным доменом.

key

Строка открытого ключа удаленного домена в кодировке base64.

Оператор server

Оператор `server` определяет характеристики удаленного сервера. Синтаксис BIND 8:

```
server address {
  [ bogus yes|no; ]
  [ support-ixfr yes|no; ]
  [ transfers number; ]
  [ transfer-format one-answer|many-answers; ]
  [ keys { key_id [key_id ... ] }; ]
};
```

Оператор `server` действует для удаленного сервера с адресом *address*.

`transfer-format`

Устанавливает формат передачи зоны, используемый для обмена с этим сервером. Допустимые значения: более эффективный формат `many-answers` или обратно совместимый `one-answer`.

`bogus yes`

Запрещает локальному серверу посылать данному серверу запросы. По умолчанию `no`, то есть запросы, обращенные к данному удаленному серверу, разрешены.

`support-ixfr yes`

Указывает, что удаленный сервер поддерживает пошаговую передачу зоны. `no`, значение по умолчанию, указывает, что удаленный сервер не умеет выполнять пошаговую передачу.

`transfers`

Определяет максимальное число параллельных входящих потоков передачи от данного сервера.

`keys`

Указывает ключ, необходимый удаленному узлу для безопасной передачи данных.

Оператор server в BIND 9

Синтаксис оператора `server` в BIND 9 несколько отличается:

```
server address {
  [ bogus yes|no; ]
  [ provide-ixfr yes|no; ]
  [ request-ixfr yes|no; ]
  [ transfers number; ]
  [ transfer-format one-answer|many-answers; ]
```

```
    [ keys { key_id [key_id ... ] }; ]
  };
```

Поля в основном такие же, как для BIND 8, за исключением поля `support-ixfr`, представленного здесь двумя параметрами:

`provide-ixfr`

Указывает, что локальный сервер готов предоставить удаленному серверу возможность пошагового получения зоны.

`request-ixfr`

Указывает, что локальный сервер будет запрашивать пошаговую передачу зоны при работе с удаленным сервером.

Оператор options

Оператор `options` определяет глобальные параметры для BIND и протокола DNS. Синтаксис BIND 8:

```
  options {
    [ version string; ]
    [ directory pathname; ]
    [ named-xfer pathname; ]
    [ dump-file pathname; ]
    [ memstatistics-file pathname; ]
    [ pid-file pathname; ]
    [ statistics-file pathname; ]
    [ auth-nxdomain yes|no; ]
    [ deallocate-on-exit yes|no; ]
    [ dialup yes|no; ]
    [ fake-iquery yes|no; ]
    [ fetch-glue yes|no; ]
    [ has-old-clients yes|no; ]
    [ host-statistics yes|no; ]
    [ multiple-cnames yes|no; ]
    [ notify yes|no; ]
    [ recursion yes|no; ]
    [ rfc2308-type1 yes|no; ]
    [ use-id-pool yes|no; ]
    [ treat-cr-as-space yes|no; ]
    [ also-notify { address-list; };
    [ forward only|first; ]
    [ forwarders { address-list; }; ]
    [ check-names master|slave|response  warn|fail|ignore; ]
    [ allow-query { address_match_list }; ]
    [ allow-transfer { address_match_list }; ]
    [ allow-recursion { address_match_list }; ]
    [ blackhole { address_match_list }; ]
    [ listen-on [ port ip_port ] { address_match_list }; ]
    [ query-source [address ip_addr|*] [port ip_port|*] ; ]
    [ lame-ttl number; ]
    [ max-transfer-time-in number; ]
```

```
    [ max-ncache-ttl number; ]
    [ min-roots number; ]
    [ serial-queries number; ]
    [ transfer-format one-answer|many-answers; ]
    [ transfers-in  number; ]
    [ transfers-out number; ]
    [ transfers-per-ns number; ]
    [ transfer-source ip_addr; ]
    [ maintain-ixfr-base yes|no; ]
    [ max-ixfr-log-size number; ]
    [ coresize size; ]
    [ datasize size; ]
    [ files size; ]
    [ stacksize size; ]
    [ cleaning-interval number; ]
    [ heartbeat-interval number; ]
    [ interface-interval number; ]
    [ statistics-interval number; ]
    [ topology { address_match_list }; ]
    [ sortlist { address_match_list }; ]
    [ rrset-order { order_spec ; [ order_spec ; ... ] ] };
};
```

Существует около десятка различных типов значений для этих параметров. Параметры `check-names` и `transfer-format` определяются ключевыми словами. Логические параметры принимают значения `yes` и `no`. Все прочие параметры предполагают указание подходящих значений в определенных форматах. Некоторые форматы (*string*, *number*, *pathname*, *domain*, *type*, *class*, *ip_port* и *ip_addr*) очевидны. Прочие требуют небольших пояснений.

address-list

Список IP-адресов, разделенных двоеточиями. Этот список более ограничен, нежели список *address_match_list*.

address_match_list

Список адресов, имен `acl`, а также идентификаторов ключей.

order_spec

Сложное правило, определяющее порядок следования записей ресурсов при передаче нескольких записей в ответ на один запрос. Структура *order_spec*:

```
[class класс ][ type тип ][ name "домен" ] order порядок
```

класс, *тип* и *домен* пояснений не требуют. *порядок* может принимать одно из трех возможных значений:

`fixed`

Сохраняется порядок следования записей в файле зоны.

`random`

Записи ресурсов размещаются в случайном порядке.

`cyclic`

Записи ресурсов подвергаются циклическому сдвигу (порядок по умолчанию).

Параметры BIND 8:

`version`

Строка, возвращаемая в ответ на запрос номера версии сервера.

`directory`

Имя рабочего каталога, из которого сервер читает и в который записывает файлы.

`named-xfer`

Имя программы `named-xfer`.

`dump-file`

Имя файла, в который записывается образ базы данных при поступлении сигнала SIGINT. По умолчанию *named_dump.db*.

`memstatistics-file`

Файл, в который записывается статистика использования памяти. По умолчанию *named.memstats*.

`pid-file`

Файл, в котором хранится идентификатор процесса.

`statistics-file`

Файл, в который записывается статистика при поступлении сигнала SIGILL. По умолчанию *named.stats*.

`auth-nxdomain`

`yes`, значение по умолчанию, предписывает серверу функционировать в качестве компетентного.

`deallocate-on-exit`

`yes` приводит к сохранению статистики использования памяти в файл *named.memstats*. По умолчанию `no`.

`dialup`

`yes` оптимизирует сервер для работы с коммутируемыми соединениями. По умолчанию `no`.

`fake-iquery`

`yes` предписывает серверу возвращать поддельный ответ вместо ошибки на обратные запросы. По умолчанию `no`.

`fetch-glue`

`yes`, значение по умолчанию, приводит к получению всех связующих записей при формировании ответа.

has-old-clients

yes устанавливает auth-nxdomain и maintain-ixfr-base в значение yes, а rfc2308-type1 — в значение no.

host-statistics

yes собирает статистику по всем узлам. По умолчанию no.

multiple-cnames

yes разрешает использовать несколько записей CNAME для одного доменного имени. По умолчанию no.

notify

yes, значение по умолчанию, предписывает посылать сообщения DNS NOTIFY в случае обновления зоны.

recursion

yes, значение по умолчанию, предписывает выполнять рекурсивный поиск ответов на запросы.

rfc2308-type1

yes предписывает возвращать NS-записи наряду с SOA-записью для отрицательных ответов. По умолчанию (no) возвращается только запись SOA в целях совместимости со старыми серверами.

use-id-pool

yes отслеживает идентификаторы запросов с целью увеличения их «случайности». По умолчанию no.

treat-cr-as-space

yes предписывает считать символы возврата каретки пробелами при загрузке файла зоны. По умолчанию no.

also-notify

Указывает неофициальные серверы имен, которым также следует посылать сообщения DNS NOTIFY.

forward

first предписывает обращаться с запросом сначала к серверам forwarders, а затем искать ответ самостоятельно. Значение only предписывает обращаться с запросом только к серверам forwarders.

forwarders

Перечисляет IP-адреса серверов, которым перенаправляются запросы. По умолчанию ретрансляция запросов не используется.

check-names

Предписывает проверять имена узлов на соответствие указаниям RFC. Проверка может выполняться при загрузке зоны основным сервером (master), при получении зоны подчиненным сервером (slave) либо при обработке ответа (response). Если обнаружена ошибка, ее можно проигнори-

ровать (`ignore`), послать соответствующее предупреждение либо отвергнуть некорректное имя (`fail`).

`allow-query`

Принимаются только запросы от узлов из списка. По умолчанию запросы принимаются от всех узлов.

`allow-transfer`

Только узлам из списка разрешено получать зону. По умолчанию получение зоны разрешено для всех узлов.

`allow-recursion`

Выполнение рекурсивных запросов через этот сервер разрешено только узлам из списка. По умолчанию рекурсивные запросы выполняются для всех узлов.

`blackhole`

Перечисляет узлы, от которых данный сервер не принимает запросы.

`listen-on`

Определяет интерфейсы и порты, через которые предоставляет доступ к службе имен данный сервер. По умолчанию сервер принимает запросы через стандартный порт (53) на всех установленных интерфейсах.

`query-source`

Определяет адрес и порт, используемые при обращении к другим серверам.

`lame-ttl`

Длительность кэширования информации о сервере с некорректным делегированием. По умолчанию – 10 минут.

`max-transfer-time-in`

Максимальная длительность ожидания завершения входящей передачи зоны. По умолчанию – 120 минут (2 часа).

`max-ncache-ttl`

Длительность кэширования отрицательных ответов. По умолчанию – 3 часа, максимально допустимое значение – 7 дней.

`min-roots`

Минимальное число доступных корневых серверов, при котором принимаются запросы, требующие использования корневых серверов. По умолчанию 2.

`serial-queries`

Максимальное число ожидающих обработки SOA-запросов для подчиненного сервера. По умолчанию – 4.

`transfer-format`

`one-answer` приводит к передаче одной RR-записи в одном сообщении. `many-answers` – к передаче максимально возможного числа записей в одном сообщении.

`transfers-in`

Устанавливает максимальное число единовременно существующих входящих потоков передачи зоны. По умолчанию – 10.

`transfers-out`

Число единовременно существующих исходящих потоков передачи зоны.

`transfers-per-ns`

Ограничивает число единовременно существующих входящих потоков передачи зоны для отдельного сервера имен. По умолчанию – 2.

`transfer-source`

IP-адрес сетевого интерфейса, используемого данным сервером для получения зоны от удаленных основных серверов.

`maintain-ixfr-base`

`yes` предписывает хранить журнал пошаговой передачи зон. По умолчанию `no`.

`max-ixfr-log-size`

Устанавливает максимальный размер файла журнала пошаговой передачи зон.

`coresize`

Устанавливает максимальный размер файла образа памяти.

`datasize`

Ограничивает объем памяти, используемой сервером.

`files`

Ограничивает число файлов, одновременно открываемых сервером. По умолчанию ограничений нет.

`stacksize`

Ограничивает объем стека, используемого сервером.

`cleaning-interval`

Устанавливает интервал удаления из кэша устаревших записей ресурсов. По умолчанию 60 минут.

`heartbeat-interval`

Устанавливает интервал выполнения служебных операций для зоны, если параметр `dialup` имеет значение `yes`. По умолчанию – 60 минут.

`interface-interval`

Устанавливает интервал сканирования сервером сетевых интерфейсов с целью поиска новых интерфейсов или обнаружения факта удаления существующих. По умолчанию – 60 минут.

`statistics-interval`

Устанавливает интервал записи статистики. По умолчанию – 60 минут.

`topology`

Принуждает сервер отдавать предпочтение определенным удаленным серверам имен. Обычно сервер отдает предпочтение наиболее топологически близким серверам.

`sortlist`

Определяет алгоритм сортировки, применяемый к записям ресурсов перед отправкой их клиенту.

`rrset-order`

Указывает порядок следования записей, возвращаемых в ответ на один запрос.

Оператор options в BIND 9

Синтаксис команды `options` в BIND 9:

```
options {
  [ version string; ]
  [ directory pathname; ]
  [ additional-from-auth yes|no; ]
  [ additional-from-cache yes|no; ]
  [ dump-file pathname; ]
  [ pid-file pathname; ]
  [ statistics-file pathname; ]
  [ auth-nxdomain yes|no; ]
  [ dialup yes|no; ]
  [ notify yes|no|explicit; ]
  [ notify-source [ip_addr|*] [port ip_port] ; ]
  [ notify-source-v6 [ip_addr|*] [port ip_port] ; ]
  [ recursion yes|no; ]
  [ recursive-clients number; ]
  [ tcp-clients number; ]
  [ also-notify { address-list; };
  [ forward only|first; ]
  [ forwarders { address-list; }; ]
  [ allow-notify { address_match_list }; ]
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ allow-recursion { address_match_list }; ]
  [ blackhole { address_match_list }; ]
  [ listen-on [ port ip_port ] { address_match_list }; ]
  [ listen-on-v6 [ port ip_port ] { address_match_list }; ]
  [ port ip_port; ]
  [ query-source [address ip_addr|*] [port ip_port|*] ; ]
  [ query-source-v6 [address ip6_addr|*] [port ip_port|*] ; ]
  [ lame-ttl number; ]
  [ max-transfer-time-in number; ]
  [ max-transfer-time-out number; ]
  [ max-transfer-idle-in number; ]
  [ max-transfer-idle-out number; ]
```

```
    [ max-refresh-time number; ]
    [ max-retry-time number; ]
    [ max-cache-ttl number; ]
    [ max-ncache-ttl number; ]
    [ min-refresh-time number; ]
    [ min-retry-time number; ]
    [ transfer-format one-answer|many-answers; ]
    [ transfers-in  number; ]
    [ transfers-out number; ]
    [ transfers-per-ns number; ]
    [ transfer-source ip_addr|*] [port ip_port|*]; ]
    [ transfer-source-v6 ip6_addr|*] [port ip_port|*]; ]
    [ coresize size; ]
    [ datasize size; ]
    [ files size; ]
    [ stacksize size; ]
    [ cleaning-interval number; ]
    [ heartbeat-interval number; ]
    [ interface-interval number; ]
    [ sortlist { address_match_list }; ]
    [ sig-validity-interval number; ]
    [ tkey-dhkey key_name key_tag; ]
    [ tkey-domain domain; ]
    [ zone-statistics yes|no; ]
};
```

Многие параметры BIND 9 совпадают с параметрами BIND 8 и выполняют те же функции. Некоторые параметры BIND 9 отражают настройки протокола IPv6, который является составной частью BIND 9. Эти параметры – `listen-on-v6`, `notify-source-v6`, `query-source-v6` и `transfer-source-v6` – выполняют такие же функции, что и параметры IPv4 со схожими именами. Многие параметры BIND 8 стали не нужны, поскольку важные функции были интегрированы в новый код BIND 9. Однако список параметров короче не стал, поскольку появилось много новых:

`additional-from-auth`

`yes`, значение по умолчанию, предписывает серверу использовать информацию всех зон, для которых он является компетентным, при составлении дополнительного раздела ответа.

`additional-from-cache`

`yes`, значение по умолчанию, предписывает серверу использовать кэшированную информацию при составлении дополнительного раздела ответа.

`notify-source`

Определяет адрес и порт, используемые для посылки сообщений NOTIFY.

`recursive-clients`

Определяет максимальное число ожидающих обработки рекурсивных запросов от клиентов. По умолчанию – 1000.

`tcp-clients`

Определяет максимальное число единовременно существующих соединений с клиентами. По умолчанию – 1000.

`allow-notify`

Указывает серверы, которым разрешено посылать сообщения NOTIFY подчиненным серверам.

`port`

Определяет номер порта, используемый сервером. По умолчанию используется стандартный порт 53.

`max-transfer-time-out`

Определяет максимальную разрешенную длительность исходящей передачи зоны. По умолчанию – 2 часа.

`max-transfer-idle-in`

Определяет максимальную допустимую длительность бездействия процесса входящей передачи зоны. По умолчанию – 1 час.

`max-transfer-idle-out`

Определяет максимальную допустимую длительность бездействия процесса исходящей передачи зоны. По умолчанию – 1 час.

`max-refresh-time`

Устанавливает максимальное время обновления для случаев, когда данный сервер выступает в роли подчиненного. Данное значение имеет более высокий приоритет, чем время обновления, указанное в SOA-записи зоны, для которой сервер является подчиненным.

`max-retry-time`

Устанавливает максимальный интервал повторения попытки для случаев, когда данный сервер выступает в роли подчиненного. Данное значение имеет более высокий приоритет, чем время повторения попытки, указанное в SOA-записи зоны, для которой сервер является подчиненным.

`max-cache-ttl`

Устанавливает максимальную длительность кэширования данных этим сервером. Данное значение имеет более высокий приоритет, чем значения TTL зоны, из которой были получены данные.

`min-refresh-time`

Устанавливает минимальное время обновления для случаев, когда сервер выступает в роли подчиненного. Данное значение имеет более высокий приоритет, чем время, указанное в SOA-записи зоны, для которой сервер является подчиненным.

`min-retry-time`

Устанавливает максимальный интервал повторения попытки для случаев, когда данный сервер выступает в роли подчиненного. Данное значение

имеет более высокий приоритет, чем время повторения попытки, указанное в SOA-записи зоны, для которой сервер является подчиненным.

`sig-validity-interval`

Устанавливает срок действия цифровых подписей, генерируемых для автоматических обновлений. По умолчанию – 30 дней.

`tkey-dhkey`

Указывает ключ Диффи-Хеллмана, используемый сервером для создания общих ключей.

`tkey-domain`

Определяет доменное имя, добавляемое к именам общих ключей. Обычно это доменное имя сервера.

`zone-statistics`

`yes` предписывает серверу собирать статистику для всех зон. По умолчанию – `no`.

Параметры меняются со временем. Обращайтесь к документации, поставляемой в составе дистрибутива BIND 9, за наиболее актуальным перечнем параметров.

Оператор logging

Оператор `logging` определяет параметры ведения журнала. Он может содержать два различных типа подчиненных предложений: `channel` и `category`. Синтаксис BIND 8:

```
logging {
  [ channel channel_name {
     file pathname
       [ versions number|unlimited ]
       [ size size ]
    |syslog kern|user|mail|daemon|auth|syslog|lpr
               |news|uucp|cron|authpriv|ftp
               |local0|local1|local2|local3
               |local4|local5|local6|local7
    |null;

    [ severity critical|error|warning|notice
                |info|debug [level]|dynamic; ]
    [ print-category yes|no; ]
    [ print-severity yes|no; ]
    [ print-time yes|no; ]
  }; ]

  [ category category_name {
    channel_name; [ channel_name; ... ]
  }; ]
  ...
};
```

Предложение channel определяет способ обработки сообщений журнала. Сообщения могут записываться в файл (file), передаваться демону syslog (syslog) либо удаляться (null). Если используется файл, можно указать число сохраняемых версий файла (version), максимальный допустимый объем файла (size), а также приоритет сообщений, записываемых в файл (severity). Можно предписать включение в журнал времени создания (print-time), категории (print-category) и приоритета (print-severity) сообщения.

Предложение category определяет типы сообщений, посылаемых через канал. Таким образом, предложение category определяет, что именно фиксируется в журнале, а предложение channel – куда производится запись. Категории перечислены в табл. С.1.

Таблица С.1. Категории сообщений журнала BIND 8

Категория	Тип сообщений
cname	Сообщения, фиксирующие обращения к записям CNAME
config	Сообщения, связанные с обработкой файла настройки
db	Сообщения об операциях с базой данных
default	Различные типы сообщений. Категория по умолчанию
eventlib	Сообщения, содержащие отладочные данные системы событий
insist	Сообщения, уведомляющие о сбоях при проверке непротиворечивости данных
lame-servers	Сообщения о серверах с некорректным делегированием
load	Сообщения о загрузке зоны
maintenance	Сообщения о служебных событиях.
ncache	Сообщения, связанные с кэшированием отрицательных ответов
notify	Сообщения трассировки протокола NOTIFY
os	Сообщения, уведомляющие о проблемах уровня операционной системы
packet	Сообщения, содержащие образы всех отправленных и полученных пакетов
panic	Сообщения, вызванные ошибками, приведшими к завершению работы сервера
parser	Сообщения, связанные с обработкой команд настройки
queries	Сообщения обо всех полученных запросах DNS
response-checks	Сообщения, уведомляющие о результатах проверки ответов
security	Сообщения, относящиеся к действию критериев безопасности. Наиболее информативны, если используются параметры allow-update, allow-query и allow-transfer
statistics	Сообщения, содержащие статистику сервера

Таблица C.1 (продолжение)

Категория	Тип сообщений
update	Сообщения, связанные с динамическими обновлениями
xfer-in	Сообщения, фиксирующие входящую передачу зоны
xfer-out	Сообщение, фиксирующие исходящую передачу зоны

Оператор logging в BIND 9

Синтаксис команды logging в BIND 9:

```
logging {
  [ channel channel_name {
     file pathname
       [ versions number|unlimited ]
       [ size size ]
    |syslog kern|user|mail|daemon|auth|syslog|lpr
               |news|uucp|cron|authpriv|ftp
               |local0|local1|local2|local3
               |local4|local5|local6|local7
    |stderr
    |null;

    [ severity critical|error|warning|notice
                |info|debug [level]|dynamic; ]
    [ print-category yes|no; ]
    [ print-severity yes|no; ]
    [ print-time yes|no; ]
  }; ]

  [ category category_name {
    channel_name; [ channel_name; ... ]
  }; ]
  ...
};
```

Предложение channel, по сути, такое же, как в BIND 8, – добавился лишь поток stderr как возможный пункт назначения для сообщений. Предложение category выглядит так же, однако в списке поддерживаемых категорий произошли существенные изменения. Одну категорию переименовали из db в database. Около десяти категорий более не поддерживаются: cname, eventlib, insist, load, maintenance, ncache, os, packet, panic, parser, response-check и statistics. Шесть новых категорий:

general

Широкий спектр сообщений.

resolver

Сообщения, относящиеся к разрешению запросов DNS.

client

Сообщения, связанные с обработкой запросов клиентов.

network

Сообщения, связанные с сетевыми операциями.

dispatch

Сообщения трассировки пакетов, передаваемых различным модулям сервера.

dnssec

Сообщения, отслеживающие обработку протоколов DNSSEC и TSIG.

Оператор zone

Оператор zone определяет зону обслуживания и источник информации базы данных DNS. Существует четыре варианта оператора zone: для основного сервера, для подчиненных серверов, для корневой кэш-зоны и специальный вариант для ретрансляции. Синтаксис BIND 8 для каждого из вариантов:

```
zone domain_name [ in|hs|hesiod|chaos ] {
  type master;
  file pathname;
  [ forward only|first; ]
  [ forwarders { address-list; }; ]
  [ check-names warn|fail|ignore; ]
  [ allow-update { address_match_list }; ]
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ dialup yes|no; ]
  [ notify yes|no; ]
  [ also-notify { address-list };
  [ ixfr-base pathname; ]
  [ pubkey flags protocol algorithm key; ]
};

zone domain_name [ in|hs|hesiod|chaos ] {
  type slave|stub;
  [ file pathname; ]
  [ ixfr-base pathname; ]
  masters [port ip_port] { address-list };
  [ forward only|first; ]
  [ forwarders { address-list; }; ]
  [ check-names warn|fail|ignore; ]
  [ allow-update { address_match_list }; ]
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ transfer-source ip_addr; ]
  [ dialup yes|no; ]
  [ max-transfer-time-in number; ]
  [ notify yes|no; ]
  [ also-notify { address-list };
  [ pubkey flags protocol algorithm key; ]
};
```

```
zone "." [ in|hs|hesiod|chaos ] {
  type hint;
  file pathname;
  [ check-names warn|fail|ignore; ]
};

zone domain_name [in|hs|hesiod|chaos] {
  type forward;
  [ forward only|first; ]
  [ forwarders { address-list; }; ]
  [ check-names warn|fail|ignore; ]
};
```

За ключевым словом `zone` следует имя домена. Для корневой кэш-зоны это имя всегда «.». За доменным именем следует класс данных. Для службы DNS сети Интернет класс всегда IN, принимаемый по умолчанию.

Предложение `type` определяет тип сервера: основой, подчиненный, сервер зоны ретрансляции либо файла корневой кэш-зоны. Сервер-заглушка (stub server) – это сервер, загружающий только NS-записи вместо целого домена.

Предложение `file` для основного сервера указывает на исходный файл, из которого загружается зона. Для дополнительного сервера – на файл, в который записывается зона, а предложение `master` указывает на источник данных, записываемых в файл. Для корневой кэш-зоны `file` указывает на файл скрипта, используемого для инициализации кэш-зоны. Домен ретрансляции не имеет предложения `file`, поскольку локальный сервер не хранит никакие данные такого домена.

За исключением параметра `pubkey`, все параметры, доступные для оператора `zone` BIND 8, уже описаны ранее в этом приложении. В рамках оператора `zone` действие параметра распространяется только на указанную зону. В рамках оператора `options` действие параметра распространяется на все зоны. Частные настройки зоны имеют более высокий приоритет, чем глобальные настройки оператора `options`.

Параметр `pubkey` определяет открытый ключ шифрования DNSSEC для зоны в отсутствие надежного механизма распределения открытых ключей по сети. `pubkey` определяет флаги DNSSEC, протокол и алгоритм, а также содержит версию ключа в кодировке base64. Удаленный сервер, обращающийся к данному домену по DNSSEC, указывает те же настройки при помощи команды `trusted-key`, описанной ранее.

Оператор zone в BIND 9

Синтаксис четырех вариантов оператора `zone` в изложении для BIND 9 :

```
zone domain_name [ in|hs|hesiod|chaos ] {
  type master;
  file pathname;
  [ forward only|first; ]
  [ forwarders { address-list; }; ]
  [ allow-update { address_match_list }; ]
```

```
  [ allow-update-forwarding { address_match_list }; ]
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ allow-notify { address_match_list }; ]
  [ dialup yes|no; ]
  [ notify yes|no|notify|notify-passive|refresh|passive; ]
  [ also-notify { address-list };
  [ database string; [...] ]
  [ update-policy { policy }; ]
  [ sig-validity-interval number; ]
  [ max-refresh-time number; ]
  [ max-retry-time number; ]
  [ max-transfer-idle-out number; ]
  [ max-transfer-time-out number; ]
  [ min-refresh-time number; ]
  [ min-retry-time number; ]
};

zone domain_name [ in|hs|hesiod|chaos ] {
  type slave|stub;
  [ file pathname; ]
  [ ixfr-base pathname; ]
  masters [port ip_port] { address-list };
  [ forward only|first; ]
  [ forwarders { address-list; }; ]
  [ check-names warn|fail|ignore; ]
  [ allow-update { address_match_list }; ]
  [ allow-update-forwarding { address_match_list }; ]
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ transfer-source ip_addr; ]
  [ dialup yes|no|notify|notify-passive|refresh|passive; ]
  [ max-transfer-time-in number; ]
  [ notify yes|no; ]
  [ also-notify { address-list };
  [ max-refresh-time number; ]
  [ max-retry-time number; ]
  [ max-transfer-idle-in number; ]
  [ max-transfer-idle-out number; ]
  [ max-transfer-time-in number; ]
  [ max-transfer-time-out number; ]
  [ min-refresh-time number; ]
  [ min-retry-time number; ]
  [ transfer-source ip_addr|*] [port ip_port|*]; ]
  [ transfer-source-v6 ip6_addr|*] [port ip_port|*]; ]
};

zone "." [ in|hs|hesiod|chaos ] {
  type hint;
  file pathname;
};

zone domain_name [in|hs|hesiod|chaos] {
```

```
    type forward;
    [ forward only|first; ]
    [ forwarders { address-list; }; ]
};
```

В BIND 9 используются те же четыре варианта команды `zone`, что и в BIND 8. Различие заключается в используемых параметрах. Большинство параметров оператора `zone` BIND 9 мы уже встречали при рассмотрении оператора `options` BIND 9. Следующие два параметра встречаются только в операторе `zone` BIND 9:

`allow-update-forwarding`

Перечисляет системы, имеющие право присылать подчиненному серверу зоны динамические обновления, передаваемые затем основному серверу.

`database`

Указывает тип базы данных, используемой для хранения данных зоны. По умолчанию – `rbt`, то есть единственный тип баз данных, поддерживаемый стандартным исполняемым файлом BIND 9.

Оператор controls

Оператор BIND 8 `controls` определяет управляющие каналы, используемые `ndc`. `ndc` может работать через сокеты Unix или сетевые сокеты, используя их в качестве управляющих каналов. Определения таких сокетов и дает оператор `controls`. Синтаксис оператора:

```
controls {
  [ inet ip_addr
    port ip_port
    allow { address_match_list; }; ]
  [ unix pathname
    perm file_permissions
    owner uid
    group gid; ]
};
```

Первые три параметра, `inet`, `port` и `allow`, определяют IP-адрес и номер порта сетевого сокета, а также список управления доступом, перечисляющий системы, которым разрешено осуществлять управление через данный сокет. Поскольку механизмы аутентификации в BIND 8 слабые, создание управляющего канала, доступного по сети, – решение рискованное. Человек, получивший доступ к этому каналу, получает полную власть над процессом сервера имен.

Последние четыре параметра – `unix`, `perm`, `owner` и `group` – определяют управляющий сокет Unix. Сокет Unix выглядит как файл. Его идентификатором служит обычное полное имя файла, например */var/run/ndc*. Как и всякому другому файлу, сокету Unix назначается идентификатор пользователя-владельца (*uid*) и допустимый идентификатор группы (*gid*). Защита сокета осуществляется на основе стандартных файловых прав доступа. Допустимы только численные значения *uid*, *gid* и *file_permissions*. Значение *file_permis-*

sions должно начинаться с 0. Например, чтобы установить полномочия чтения и записи для владельца, чтения для группы и запретить любой доступ всем остальным пользователям, следует использовать значение 0640.

Оператор controls в BIND 9

Оператор `controls` BIND 9 определяет каналы, используемые `rndc`. `rndc` выполняет те же функции, что и более старая программа `ndc`, но более надежна для сетевых взаимодействий. Синтаксис оператора для BIND 9:

```
controls {
  [ inet ip_addr|*
    port ip_port
    allow address_match_list;
    keys key_list; ]
};
```

В BIND 9 оператор `controls` всегда определяет сетевой сокет. При этом используются сильные механизмы аутентификации, требующие наличия ключей шифрования.

Оператор view (BIND 9)

Оператор `view` позволяет создавать различные виды той же самой зоны для различных клиентов. Это позволяет предоставлять одни сведения клиентам внутри организации и другие, более ограниченные сведения, внешним клиентам. Синтаксис команды `view`:

```
view view-name {
  match-clients { address_match_list };
  [ view-option; ... ]
  [ zone-statement; ... ]
};
```

`view-name`

Произвольное имя, используемое в настройках для идентификации вида. Чтобы избежать конфликтов с ключевыми словами, *view-name* следует заключать в кавычки (например, «internal»).

`match-clients`

Определяет список клиентов, имеющих доступ к данному виду зоны.

`view-option`

Стандартный параметр BIND 9. Любой параметр в рамках оператора `view` действует только для данного вида, что позволяет применять различные параметры для зоны в зависимости от вида.

`zone-statement`

Стандартный оператор `zone` BIND 9. Полный оператор `zone` внедряется в оператор `view` с целью определения зоны, доступной через этот вид.

Оператор `view` существует только в BIND 9. BIND 8 не поддерживает виды.

Записи файлов зон

Для создания файла зоны используются записи двух типов: *инструкции*, упрощающие создание файла, а также *стандартные записи ресурсов*, определяющие данные домена в файле зоны. Типов стандартных записей ресурсов существует довольно много, а инструкций всего четыре:

`$INCLUDE` *filename*

Указывает файл, содержащий данные, которые следует включить в файл зоны. Данные включаемого файла должны быть представлены допустимыми инструкциями или стандартными записями ресурсов. $INCLUDE позволяет разделить крупный файл зоны на несколько мелких, более управляемых фрагментов.

Имя файла (*filename*) интерпретируется относительно каталога, указанного в параметре `directory` в файле *named.conf*. К примеру, если файл *named.conf* узла *crab* указывает на каталог */etc* посредством параметра `directory`, а файл зоны на узле *crab* содержит оператор `$INCLUDE events.hosts`, содержимое файла */etc/events.hosts* включается в этот файл зоны. Если вы не желаете, чтобы имя файла интерпретировалось относительно этого каталога, укажите абсолютное имя, такое как */usr/dns/events.hosts*.

`$ORIGIN` *domainname*

Изменяет значение по умолчанию доменного имени, используемое в последующих записях файла зоны. Используйте эту команду для сохранения в файле зоны данных нескольких доменов. К примеру, оператор `$ORIGIN events` в файле зоны *wrotethebook.com* устанавливает доменное имя *events.wrotethebook.com*. Все последующие записи ресурсов интерпретируются относительно этого нового домена.

В `named` операторы $ORIGIN используются для организации информации сервера. Образ базы данных `named`, созданный по команде `ndc dumpdb`, – это один файл, в котором представлена вся информация, известная серверу. Этот файл, *named_dump.db*, содержит много записей $ORIGIN, позволяющих разместить информацию обо всех доменах в одном файле.

`$TTL` *time-to-live*

Определяет значение по умолчанию для TTL, используемое в записях ресурсов. Каждый файл зоны должен начинаться с инструкции $TTL, чтобы все записи ресурсов наверняка имели корректные значения TTL. Численное поле *time-to-live* определяет значение TTL в секундах. Может использоваться также смешанный формат значений. К примеру, `1w` устанавливает TTL в одну неделю. Возможные значения для смешанного формата:

- `w` – недели
- `d` – дни
- `h` – часы
- `m` – минуты
- `s` – секунды

`$GENERATE` *range template*

Генерирует записи ресурсов для диапазона значений – на основе шаблона *template*. *range* – численный диапазон значений в формате минимум-максимум. $GENERATE создает RR-запись для каждого значения диапазона. Диапазон `1-9` приведет к созданию девяти различных записей. Тип создаваемых записей определяется шаблоном. Шаблон состоит из литеральных значений, в неизменном виде переходящих в окончательные записи, и символа $, который заменяется текущим значением из диапазона. Следовательно, если текущим значеним из диапазона является `7`, а шаблон представлен строкой `$ CNAME $.first64`, генерируется RR-запись `7 CNAME 7.first64`.

Эти инструкции полезны для организации и управления данными в файлах зон, однако собственно информация базы основана на стандартных записях ресурсов. Все файлы, на которые ссылается *named.conf*, вносят вклад в создание базы данных DNS, так что все эти файлы состоят из стандартных записей ресурсов.

Стандартные записи ресурсов

Формат стандартных записей ресурсов, называемых иногда RR-записями, определен в RFC 1033, *Domain Administrators Operations Guide* (Руководство администраторов домена). Формат следующий:

```
[name] [ttl] class type data
```

Отдельные поля стандартной записи:

name

Имя объекта, с которым связана запись. Именованным объектом может быть как отдельный узел, так и целый домен. Строка в поле *name* интерпретируется относительно текущего домена, за исключением случая, когда представлена абсолютным доменным именем.[1] Отдельные имена имеют специальное значение. Вот эти имена:

Пустое поле *name* означает текущий объект. Текущее имя остается в силе, пока не будет обнаружено новое значение в поле *name*. Это позволяет связывать наборы RR-записей с объектом, не повторяя имя объекта в каждой записи.

`..`

Две точки в поле *name* обозначают корневой домен. Однако одна точка (действительное имя корня) также обозначает корневой домен и используется чаще.

[1] Абсолютное доменное имя следует указывать вплоть до корня, то есть оно должно заканчиваться точкой.

`@`

Символ `@` в поле *name* обозначает текущую зону – доменное имя, производное от текущего доменного имени, либо имя, явным образом установленное администратором при помощи команды $ORIGIN.

`*`

Звездочка в поле *name* – символ маски. Она обозначает имя, представленное любой строкой. Звездочку можно комбинировать с доменными именами или использовать отдельно. Отдельно стоящая звездочка в поле *name* означает, что запись ресурсов относится к объектам, имена которых состоят из любых сочетаний символов и имени текущего домена. В сочетании с доменным именем звездочка интерпретируется относительно этого домена. Например, **.имя.* в поле *name* означает «любая строка плюс *.имя*».

ttl

Время жизни (time-to-live) определяет длительность кэширования информации данной RR-записи. Если *ttl* представлено численным значением, оно определяет длительность хранения в секундах. Для представления *ttl* может использоваться и смешанный формат, описанный выше для инструкции $TTL. Если значение *ttl* не установлено, используется значение по умолчанию, определенное для всей зоны при помощи инструкции $TTL.

class

Определяет класс адреса для записи. Класс адресов Интернета – IN. В данном поле для всех RR-записей DNS Интернета фигурирует значение IN, однако файл зоны может хранить и другую информацию, не связанную с Интернетом. Например, информация сервера имен Hesiod, разработанного в МТИ (MIT), обозначается значением HS в поле *class*, а информация chaosnet – значением CH. Для всех записей ресурсов в этой книге используется класс адресов IN.

type

Данное поле указывает тип данных, предоставляемых записью. Например, RR-запись типа A содержит адрес узла, обозначенного в поле *name*. Наиболее востребованные типы стандартных записей ресурсов описаны в последующих разделах.

data

Содержит информацию, специфичную для записи. Формат и содержимое поля *data* зависят от типа записи. Поле *data* – суть RR-записи. К примеру, в записи типа A поле *data* содержит IP-адрес.

В дополнение к специальным символам поля *name* в записях файлов зон используются также следующие:

`;`

Символ комментария. Все символы после точки с запятой и до конца текущей строки являются комментарием.

`( )`

Скобки позволяют разделять данные на несколько строк. После открывающей скобки все данные в последующих строках считаются частью текущей строки – вплоть до закрывающей скобки.

`\x`

Обратный слэш – символ маскировки. Нечисловые символы, следующие за обратным слэшем (\), принимаются буквально, а их специальные значения игнорируются. Например, `\;` означает точку с запятой, а не комментарий.

`\ddd`

Обратный слэш может также предварять три десятичные цифры. В этом случае десятичные цифры интерпретируются в качестве абсолютного значения байта. Например, `\255` означает байт 11111111.

Тот же общий формат записи используется для всех записей файла зоны. Ниже описаны наиболее востребованные записи ресурсов.[1]

Запись начала компетенции (Start of Authority)

Запись Start of Authority (SOA) отмечает начало зоны и обычно является первой записью в файле зоны. Все последующие записи являются частью зоны, объявленной в SOA. В каждой зоне только одна запись SOA; следующая SOA отмечает начало другой зоны. Поскольку файл зоны обычно связан только с одной зоной, он, как правило, содержит только одну запись SOA.

Формат записи SOA:

```
[zone] [ttl] IN SOA origin contact  (
                    serial
                    refresh
                    retry
                    expire
                    negative_cache_ttl  )
```

Составляющие записи SOA:

zone

Имя зоны. Обычно поле имени SOA содержит символ `@`. Внутри записи SOA символ `@` является ссылкой на доменное имя, объявленное в соответствующем операторе zone файла *named.conf*.

ttl

Время жизни в записи SOA не указывается.

`IN`

Для всех RR-записей сети Интернет класс адресов – IN.

---

[1] Существует более 40 типов RR-записей, большинство из которых не используются. Все типы описаны в книге Крэйга Ханта «Linux DNS Administration» (Sybex).

SOA

SOA – тип записи. Вся последующая информация составляет поле данных и имеет смысл только в контексте записи SOA.

origin

Имя узла основного сервера данного домена. Обычно записывается в виде абсолютного доменного имени. Если *crab* является основным сервером *wrotethebook.com*, данное поле в записи SOA зоны *wrotethebook.com* содержит значение *wrotethebook.com*.

contact

Адрес электронной почты лица, ответственного за домен. Адрес немного видоизменяется. Символ @, присущий адресам электронной почты сети Интернет, заменяется точкой. Следовательно, для адреса электронной почты *david@crab.wrotethebook.com* администратора домена *wrotethebook.com* запись SOA зоны *wrotethebook.com* содержит значение *david.crab.wrotethebook.com*. в поле *contact*.

serial

Номер версии файла зоны. Данное десятиразрядное численное поле обычно содержит простые числа, например 117. Однако создание чисел возлагается на администратора. Некоторые используют формат, отражающий дату обновления зоны (к примеру, 2001061800). Независимо от выбранного формата важно помнить, что порядковый номер должен увеличиваться при каждом изменении данных в файле зоны.

Поле *serial* имеет особое значение. Оно используется подчиненными серверами для определения того, что файл зоны обновлен. Чтобы выявить это обстоятельство, подчиненный сервер запрашивает запись SOA у основного сервера и сравнивает порядковый номер хранимых данных с полученным от сервера. Если порядковый номер увеличился, подчиненный сервер запрашивает полную передачу зоны. В противном случае он предполагает, что хранимые данные отражают действительность. Увеличивайте порядковый номер при каждом обновлении данных зоны. Если вы этого не сделаете, новые данные могут не попасть на подчиненные серверы.

refresh

Длительность интервала ожидания, по истечении которого подчиненный сервер должен обратиться к основному и выяснить, обновилась ли зона. Каждые *refresh* секунд подчиненный сервер запрашивает запись SOA, получает порядковый номер и определяет, необходима ли перезагрузка файла зоны. Подчиненные серверы проверяют порядковые номера своих зон при каждом перезапуске. При этом важно сохранять синхронизацию баз данных подчиненных серверов с базой данных основного сервера, чтобы named в работе не полагался на нерегулярное событие перезагрузки. Интервал обновления позволяет организовать предсказуемый цикл для перезагрузки зоны, находящейся под контролем администратора домена.

Значение в поле *refresh* – это число длиной до восьми разрядов, указывающее максимальную длительность рассинхронизации баз данных основ-

ного и подчиненных серверов в секундах. Низкое значение обновления обеспечивает разумную степень синхронизации данных на серверах, а очень низкое значение обычно не требуется. Неоправданно короткий интервал обновления создает ненужную нагрузку на сеть и подчиненные серверы. Значение поля *refresh* должно отражать частоту обновлений вашей базы данных DNS.

Базы данных DNS большинства сетевых площадок очень стабильны. Системы периодически добавляются, но обычно вовсе не раз в час. Добавляя новую систему, вы можете назначить ей имя узла и адрес до непосредственного подключения к сети. Затем можно добавить соответствующую информацию в базу данных сервера имен – раньше, чем она потребуется, так что информация распространится по подчиненным серверам задолго до входа системы в строй.

Если планируются серьезные изменения, интервал обновления можно временно сократить – только на период работ. Следовательно, в обычной ситуации интервал обновления может быть длительным, что позволяет сократить нагрузку на сеть и серверы. От двух (43 200 секунд) до четырех (21 600 секунд) раз в день – подходящее значение поля *refresh* для многих сетевых площадок.

Процесс получения записи SOA, анализа порядкового номера и при необходимости загрузки файла зоны носит название *обновления зоны* (*zone refresh*). Отсюда и название поля *refresh*.

retry

Длительность паузы, возникающей перед повторной попыткой обновления зоны, если основной сервер не ответил на предыдущий запрос обновления. Значение *retry* указывается в секундах и может быть до восьми десятичных разрядов длиной.

Не следует использовать слишком маленькие значения. Если основной сервер не ответил, дело может быть в сбое сервера или сети. Быстрый повтор попытки обращения к неработоспособной системе ничего не дает, только расходует ресурсы сети. Подчиненный сервер, обслуживающий большое число зон, может испытывать проблемы в случае низких значений *retry*. Если подчиненный сервер не может связаться с основным сервером нескольких из своих зон, он зацикливается на повторных попытках.[1] Избегайте осложнений; используйте час (3600) или полчаса (1800) в качестве значения *retry*.

expire

Длительность хранения данных зоны подчиненными серверами (без учета обновлений). Значение указывается в секундах и может быть до вось-

[1] Сервер может время от времени переставать реагировать на запросы либо отображать сообщение об ошибке «too many open files» (слишком много открытых файлов).

ми цифр длиной. Если по истечении *expire* секунд подчиненный сервер так и не смог обновить зону, он должен удалить все данные зоны.

Значение *expire* обычно очень велико. Широко используется значение в 604 800 секунд (около недели). Если основной сервер не отвечал на запросы обновлений, повторяемые каждые *retry* секунд, семь дней, данные удаляются. Семь дней – разумное значение, однако и гораздо более длинные интервалы устаревания данных не являются чем-то из ряда вон выходящим.

negative_cache_ttl

Поле *negative_cache_ttl* записи SOA содержит значение TTL по умолчанию для отрицательной информации о данном домене, кэшируемой удаленными серверами. Все серверы кэшируют ответы и используют их для реагирования на последующие запросы. Большинство кэшируемых ответов представлено стандартными записями ресурсов. Но, кроме того, сервер имен может узнать от компетентного сервера, что определенная RR-запись не существует. Такая информация также является ценной и кэшируется.

Сервер хранит кэшированные записи, пока они не устареют, а время старения определяется значением TTL. Каждой записи соответствует TTL – будь то время жизни, указанное непосредственно для данной записи, или же значение по умолчанию TTL, определенное инструкцией $TTL. Однако для отрицательных «ответов» нет записей ресурсов, а значит – нет и явно установленных значений TTL. Именно значение *negative_cache_ttl* определяет для удаленных серверов длительность кэширования отрицательной информации.

Значение *negative_cache_ttl* обычно лежит в интервале от 5 до 15 минут. Достаточно большое значение, предохраняющее ваш сервер от повторных запросов несуществующей информации, но достаточно короткое время с точки зрения повторных запросов от удаленного пользователя, который в курсе, что система с определенным именем скоро вновь проявится в сети.

Большинство полей записи SOA предназначено для синхронизации подчиненных серверов с основным. Эти значения позволяют гарантировать, что подчиненный сервер будет периодически получать копию зоны от основного сервера. В дополнение к этому (и совершенно безотносительно значений записи SOA) основной сервер уведомляет подчиненные серверы, что зона обновилась, с целью скорейшего распространения новой копии данных. Сочетание обновлений, инициируемых подчиненными серверами, и обновлений, инициируемых основным сервером, гарантирует, что файлы серверов надежно синхронизируются.

Пример записи SOA для домена *wrotethebook.com*:

```
@    IN  SOA  crab.wrotethebook.com. david.crab.wrotethebook.com. (
              2001061801              ; порядковый номер
              21600                   ; обновление четыре раза в день
              1800                    ; повтор попытки каждые полчаса
```

```
                604800          ; устаревание через  неделю
                900             ; время жизни отрицательных ответов - 15 минут
    )
```

Обратите внимание на порядковый номер в данной записи SOA. Он имеет формат *yyyymmddvv*, где *yyyy* – год, *mm* – месяц, *dd* – день, а *vv* – номер версии, созданной в этот день. Такой вид порядкового номера позволяет администратору определить, в какой день была обновлена зона. Добавление номера версии позволяет многократно вносить изменения в один день. Данный файл зоны был создан 18 июня 2001 года и для этой даты был первой версией.

Данная запись сообщает, что основным сервером для зоны является *crab*, а связь с лицом, ответственным за зону, можно осуществить по электронной почте: *david@crab.wrotethebook.com*. Запись предписывает подчиненным серверам проверять зону на предмет изменений четыре раза в день, повторять попытку обновления каждые полчаса, если нет ответа от основного сервера. Если в течение недели повторных попыток не получен ответ, следует удалить данные этой зоны. Наконец, если RR-запись не существует в данной зоне и удаленный сервер кэширует эту информацию, время кэширования составляет 15 минут.

Запись сервера имен (Name Server)

Записи Name Server (NS) указывают компетентные серверы зоны. Эти записи являются указателями, связующими всю иерархию доменов. NS-записи в доменах высшего уровня указывают на серверы доменов второго уровня, а домены второго уровня в свою очередь содержат NS-записи, указывающие на серверы поддоменов. Записи NS, указывающие на подчиненные домены, обязательны – без них поддомены просто не будут доступны. В отсутствие NS-записей серверы домена остаются неизвестными.

Формат записи NS:

```
[domain] [ttl] IN NS server
```

domain

Имя домена, для которого узел, обозначенный в поле *server*, является компетентным сервером имен.

ttl

Время жизни обычно не указывается.

IN

Класс адресов – IN.

NS

Тип записи сервера имен – NS.

server

Имя узла, предоставляющего компетентную информацию по именам данного домена.

Обычно по меньшей мере один сервер имен находится за пределами домена. Имя сервера не может указываться относительно локального домена, следует использовать абсолютное доменное имя. Соблюдая последовательность, многие администраторы используют абсолютные имена для всех серверов, хотя для серверов локального домена это не обязательно.

Адресная запись

Большую часть записей ресурсов в файле зоны прямого отображения[1] составляют адресные записи. Адресные записи используются для преобразования имен узлов в IP-адреса, то есть для реализации самой востребованной функции базы данных DNS.

Адресная запись имеет следующий формат:

```
[host] [ttl] IN A address
```

host

Имя узла, адрес которого содержится в поле данных этой записи. Чаще всего имя узла записывается относительно текущего домена.

ttl

Время жизни обычно не указывается.

IN

Класс адресов – IN.

A

Тип адресной записи – A.

address

IP-адрес узла в десятичной записи через точку (например, 172.16.12.2).

Связующая запись (*glue record*) – это особый тип адресной записи. Большинство адресных записей относится к узлам одной зоны, но иногда необходимо ссылаться в адресной записи на узел другой зоны. Это необходимо для указания адресов серверов имен подчиненных доменов. Вспомните, что запись NS для сервера поддомена указывает сервер по имени. Чтобы обращаться к серверу, необходим еще и адрес, так что нужна еще и A-запись. Адресная запись в сочетании с записью сервера имен связывает домены, отсюда и название «связующая запись».

Запись почтового ретранслятора (Mail Exchanger)

Запись Mail Exchanger (MX) перенаправляет почту на почтовый сервер. Она может выполнять перенаправление для отдельного компьютера или для целого домена. Записи MX крайне полезны в доменах, включающих системы, не использующие серверное программное обеспечение SMTP. Почта, адресованная таким системам, может быть переправлена компьютерам, которые

[1] Различные файлы настройки named описаны в главе 8.

работают с серверным ПО. Записи MX используются также для упрощения адресации почты путем перенаправления почты на серверы, понимающие упрощенные адреса.

Формат записи MX:

```
[name] [ttl] IN MX preference host
```

name

Имя узла или домена, которому адресована почта. Считайте это поле значением, фигурирующем в адресе электронной почты после символа @. Почта, адресованная на это имя, передается почтовому серверу, обозначенному в поле *host*.

ttl

Время жизни обычно не указывается.

IN

Класс адресов – IN.

MX

Тип записи почтового ретранслятора – MX.

preference

С узлом или доменом может быть связано несколько записей. Поле *preference* определяет порядок опроса почтовых серверов. Обращение к серверам с более низкими значениями приоритета происходит раньше, так что наиболее предпочтительному серверу соответствует значение приоритета 0. Значения приоритета обычно назначаются с шагом 5 или 10, чтобы оставалась возможность добавлять новые серверы с промежуточными значениями без необходимости редактировать старые записи MX.

host

Имя почтового сервера, которому передается почта, если она адресована узлу или домену, обозначенному в поле *name*.

Записи MX работают следующим образом. Если удаленная система намеревается передать почту узлу, она запрашивает MX-записи для этого узла. DNS возвращает все найденные записи MX для искомого узла. Удаленный сервер выбирает запись MX с наименьшим значением приоритета и пытается доставить почту на этот сервер. Если к серверу подключиться невозможно, система опрашивает каждый из оставшихся серверов в порядке возрастания значений приоритета, пока не будет доставлена почта. Если DNS не возвращает MX-записи, удаленный сервер доставляет почту непосредственно узлу, которому она адресована. Записи MX определяют только перенаправления почты. Удаленная система и почтовый сервер выполняют всю работу по собственно доставке почты.

Поскольку удаленная система прежде всего пытается использовать запись MX, многие администраторы доменов создают записи MX для всех узлов зоны. Многие из таких записей указывают на узлы, которым адресована поч-

та, то есть запись MX для узла *crab* может содержать в поле `host` значение *crab.wrotethebook.com*. Такие записи гарантируют, что удаленный компьютер сначала попытается доставить почту непосредственно узлу, а сервером MX воспользуется только в случае, когда прямая доставка невозможна.

Важное применение записей MX: доставка почты в не-Интернет-сети с использованием адресации в стиле Интернет. Записи MX позволяют перенаправлять почту на компьютеры, которые умеют выполнять доставку в не-Интернет-сети. Например, сети, использующие `uucp`, могут зарегистрировать доменное имя сети Интернет в UUNET. UUNET использует записи MX для перенаправления почты, адресованной неподключенным сетям, в *uunet.uu.net*, откуда доставка выполняется уже посредством `uucp`.

Вот некоторые примеры использования MX. Все примеры относятся к воображаемому домену *wrotethebook.com*. В первом примере почта, адресованная *clock.wrotethebook.com*, перенаправляется на узел *crab.wrotethebook.com* посредством такой записи MX:

```
clock IN MX 10 crab
```

Второй пример: запись MX, упрощающая адресацию почты. Люди могут посылать сообщения любому пользователю данного домена, не зная, на каком компьютере пользователь читает свою почту. Почта, адресованная на *user@wrotethebook.com*, следующей записью MX перенаправляется на узел *crab*, почтовый сервер, знающий, как выполнять доставку почты отдельным пользователям домена.

```
wrotethebook.com. IN MX 10 crab.wrotethebook.com.
```

Последний пример – запись MX, реализующая перенаправление почты, адресованной любому узлу домена, на центральный почтовый сервер. Почта, адресованная любому узлу – *horseshoe.wrotethebook.com*, *24seven.wrotethebook.com* или *anything.wrotethebook.com*, – передается узлу *crab*. Это самое распространенное применение символа маски (`*`).

```
*.wrotethebook.com. IN MX 10 crab.wrotethebook.com.
```

В приведенных примерах использовано значение приоритета 10, так что мы имеем возможность добавить запись для нового сервера с меньшим значением приоритета, не редактируя существующую MX-запись. Обратите внимание также, что имена узлов в первом примере указываются относительно домена *wrotethebook.com*, однако все прочие имена относительными не являются, поскольку заканчиваются точкой. Все эти имена *могли* бы быть относительными, поскольку относятся к узлам домена *wrotethebook.com*; абсолютные имена использовались здесь только для того, чтобы внести некоторое разнообразие в примеры. Наконец, запись MX, использующая маску, действует только для узлов, не имеющих конкретных записей MX. Если существует конкретная запись для *clock* в области видимости записи с маской, действие записи с маской не распространяется на систему *clock*.

Запись псевдонима (Canonical Name)

Запись Canonical Name (CNAME) определяет псевдоним для официального имени узла. Запись CNAME работает подобно псевдонимам в таблице узлов. Механизм позволяет пользователям иметь дело с удобными альтернативными именами узлов, а приложениям – с общеупотребительными именами (такими как *loghost*, используемое `syslogd`).

Запись CNAME часто используется в целях облегчения перехода от старого имени узла к новому. И хотя лучше всего избегать смены имен узлов, проявляя изначально прозорливость при их выборе, не всяких изменений можно избежать. Когда приходится изменить имя, может пройти долгое время, прежде чем оно полностью встанет в строй, особенно если имя узла используется в списке рассылки, функционирующем на удаленной площадке. Чтобы сократить проблемы для удаленных систем, используйте записи CNAME, пока изменения не вступят в силу везде.

Формат записи CNAME:

```
nickname [ttl] IN CNAME host
```

nickname

Данное имя узла является псевдонимом официального имени узла, указанного в поле *host*. Поле *nickname* может содержать любое допустимое имя узла.

ttl

Время жизни обычно не указывается.

IN

Класс адреса – IN.

CNAME

Тип RR-записи – CNAME.

host

Здесь указывается каноническое имя узла. Данное имя должно быть официальным, оно не может быть псевдонимом.

Есть одно важное обстоятельство, связанное с записью CNAME: все прочие записи ресурсов должны быть связаны с официальным именем узла, а не с псевдонимом. Это означает, что запись CNAME не следует размещать между определением узла и списком RR-записей, связанных с узлом. Данный пример демонстрирует корректное размещение записи CNAME:

```
rodent      IN    A       172.16.12.2
            IN    MX      5 rodent.wrotethebook.com.
            IN    RP      alana.wrotethebook.com. alana
            IN    TXT     "Linux workstation in room A15"
mouse       IN    CNAME   rodent.wrotethebook.com.
```

В следующем примере имя узла *rodent* остается в силе для записей MX, RP и TXT, поскольку в этих записях поле *name* пусто. Запись CNAME изменяет значение поля *name* на *mouse*, то есть на псевдоним *rodent*. Любые RR-записи с пустыми полями *name*, следующие за данной CNAME-записью, будут связаны с псевдонимом *mouse*, что недопустимо. Неверное размещение CNAME:

```
rodent      IN    A       172.16.12.2
mouse       IN    CNAME   rodent.wrotethebook.com.
            IN    MX      5 rodent.wrotethebook.com.
            IN    RP      alana.wrotethebook.com. alana
            IN    TXT     "Linux workstation in room A15"
```

В результате такого неверного размещения записи CNAME `named` выводит сообщение об ошибке: «mouse.wrotethebook.com has CNAME and other data (illegal)». Проверьте файл */var/adm/messages* на предмет наличия сообщений об ошибках и убедитесь, что правильно разместили записи CNAME.

Указатель доменного имени (Domain Name Pointer)

Записи Domain Name Pointer (PTR) используются для преобразования численных IP-адресов в имена узлов. Задача обратна той, что решается адресными записями, позволяющими преобразовывать имена узлов в адреса. Записи PTR используются для создания файлов обратных доменов *in-addr.arpa*.

Многие администраторы игнорируют обратные домены, поскольку внешне все прекрасно работает и без них. Не повторяйте эту ошибку. Своевременно обновляйте обратные зоны. Некоторые программы используют обратные домены для преобразования IP-адресов в имена узлов при подготовке сведений состояния. Хорошим примером является программа `netstat`. Некоторые операторы служб используют обратные домены, чтобы выяснить, кто пользуется их службами. Если они не могут отобразить IP-адрес в имя узла, то запрещают устанавливать соединение.

Форма записи PTR:

```
name [ttl] IN PTR host
```

name

Указанное здесь имя является на самом деле числом. Число определяется относительно текущего домена *in-addr.arpa*. Имена домена *in-addr.arpa* – это IP-адреса, записанные в обратном порядке. Если за текущий домен принять *16.172.in-addr.arpa*, тогда в поле *name* для узла *rodent* (172.16.12.2) должно содержаться значение 2.12. Эти цифры (2.12) добавляются к текущему домену (*16.172.in-addr.arpa*), что в результате дает имя *2.12.16.172.in-addr.arpa*. Уникальная структура доменных имен *in-addr.arpa* описана в главе 4.

ttl

Время жизни обычно не указывается.

IN

Класс адресов – IN.

PTR

Тип RR-записи – PTR.

host

Абсолютное доменное имя компьютера, адрес которого указан в поле *name*. Имя должно быть абсолютным, потому что имя узла не может определяться относительно текущего домена *in-addr.arpa*.

В примере файла зоны обратного отображения (*172.16.rev*), представленном в главе 8, содержится большое число записей PTR.

Ответственное лицо (Responsible Person)

Запись Responsible Person (RP) указывает контактные координаты для узла или домена. Формат записи RP:

```
[name] [ttl] IN RP mail_address text_pointer
```

name

Имя доменного объекта, для которого определяется ответственное лицо.

ttl

Время жизни обычно не указывается.

IN

Класс адресов – IN.

RP

Тип RR-записи – RP.

mail_address

Адрес электронной почты ответственного лица. Символ @, обычно присутствующий в адресе электронной почты, заменяется точкой. Так, *craig@wrotethebook.com* становится *craig.wrotethebook.com*.

text_pointer

Доменное имя записи TXT, содержащей дополнительную информацию об ответственном лице.

Вот пример использования записи RP совместно с записью TXT:

```
crab.wrotethebook.com.    IN RP  craig.wrotethebook.com. crabRP
crabRP.wrotethebook.com. IN TXT "Craig Hunt (301)555-1234 X237"
```

Запись RP утверждает, что с лицом, ответственным за узел *crab.wrotethebook.com*, можно связаться по адресу электронной почты *craig@wrotethebook.com* и что дополнительные сведения об этом человеке могут быть получены из записей TXT для *crabRP.wrotethebook.com*. В данном случае текстовая запись содержит имя и телефонный номер контактного лица.

Текстовая запись

Запись Text (TXT) хранит строковые данные. Текстовые данные могут иметь произвольный формат. Некоторые системы определяют локальный формат для информации. Например, запись TXT может хранить адрес Ethernet-узла на одной сетевой площадке и номер комнаты, в которой расположен узел, на другой площадке.

Формат записи TXT:

```
[name] [ttl] IN TXT string
```

name

Имя доменного объекта, с которым связаны строковые данные.

ttl

Время жизни обычно не указывается.

IN

Класс адресов – IN.

TXT

Тип RR-записи – TXT.

string

Строка текстовых данных, заключенных в кавычки.

Информация об узле (Host Information)

Запись Host Information (HINFO) содержит краткое описание аппаратной и программной части конкретного узла. Аппаратные и программные средства описываются посредством стандартной терминологии, определенной в документе RFS *Assigned Numbers* (разделы *Machine Names* – аппаратное обеспечение – и *System Names* – программное). Существует большое число обозначений для аппаратного и программного обеспечения, описанных в RFC. Имена в основном следуют одному базовому формату. Имена с внедренными пробелами должны заключаться в кавычки, поэтому в некоторых именах вместо пробелов используются дефисы (-). Имя компьютера – обычно просто название фирмы-производителя прописными буквами, отделенное дефисом от номера модели. Имя системы – название фирмы-производителя операционной системы прописными буквами. Естественно, благодаря скорости изменения компьютерного рынка данные в RFC *Assigned Numbers* быстро устаревают. По этой причине многие администраторы придумывают собственные значения для названий компьютеров и операционных систем.

Формат записи HINFO:

```
[host] [ttl] IN HINFO hardware software
```

host

Имя узла, аппаратное и программное обеспечение которого описано в разделе данных этой записи.

ttl

Время жизни обычно не указывается.

IN

Класс адресов – IN.

HINFO

HINFO – тип RR-записи. Вся последующая информация составляет раздел данных записи.

hardware

Данное поле указывает аппаратное обеспечение, используемое узлом. Оно содержит название компьютера из документа RFC *Assigned Numbers*. Если поле содержит пробелы, его необходимо заключить в кавычки. Один пробел отделяет поле *hardware* от следующего за ним поля *software*.

software

Данное поле указывает операционную систему, под управлением которой работает узел. Оно содержит название для операционной системы, определенное в документе RFC *Assigned Numbers*. Используйте кавычки, если название системы содержит хотя бы один пробел.

Приложения, использующие записи HINFO, не получили широкого распространения. Данная запись просто предоставляет информацию. Из соображений безопасности в некоторых системах использование HINFO не приветствуется. Администраторы опасаются, что такая дополнительная информация поможет злоумышленникам сориентироваться и направить свои усилия на взаимодействие с конкретным оборудованием и конкретной операционной системой. Для предоставления сведений о системе чаще используется универсальная запись TXT.

Запись широко известных служб (Well-Known Services)

Запись Well-Known Services (WKS) перечисляет сетевые службы, доступные на указанном узле. Официальные названия протоколов и служб, используемые в записях WKS, определены документом RFC *Assigned Numbers*. Простейший способ получить список имен широко известных служб – просмотреть файл */etc/services* своей системы. Каждый узел может иметь не более двух записей WKS; одну для TCP и одну для UDP. Поскольку в записи WKS обычно перечислен набор служб, каждая такая запись может занимать несколько строк.

Формат записи WKS:

```
[host] [ttl] IN WKS address protocol services
```

host

Имя узла, предоставляющего указанные службы.

ttl

Время жизни обычно не указывается.

`IN`

Класс адресов – IN.

`WKS`

Тип RR-записи – WKS. Вся последующая информация является переменной информацией записи WKS.

`address`

IP-адрес узла в десятичной записи через точку (к примеру, 172.16.12.2).

`protocol`

Протокол транспортного уровня, по которому работает служба, – TCP или UDP.

`services`

Список служб, предоставляемых данным узлом. Число служб, сведения о которых распространяются в записи, может быть любым, однако их имена должны совпадать с именами из файла */etc/services*. Элементы списка служб разделяются пробелами. Для продолжения списка на последующих строках используются круглые скобки.

Приложения, использующие записи WKS, не получили широкого распространения. Записи WKS используются только для предоставления общих сведений о системе. Повторимся, из соображений безопасности на некоторых площадках предпочитают скрывать подобные сведения. Отдельные протоколы, такие как `tftp` и `finger`, являются притягательными мишенями для злоумышленников. Записи SRV более полезны в плане предоставлении информации о службах, реализованных тем или иным сервером.

Запись выбора сервера (Server Selection)

Запись Server Selection (SRV) стандартизирует способ поиска сетевых серверов. Она предоставляет стандартные правила создания обобщенных имен серверов и добавляет возможности выбора серверов и распределения нагрузки. Формат записи SRV:

```
name [ttl] IN SRV preference weight port server
```

`name`

Запись SRV имеет уникальный формат: *`_service._protocol.name`*. Точки используются для разделения составляющих в поле имени точно так же, как в любом доменном имени. Подчеркивания (_) используются для предотвращения конфликтов имен служб и имен протоколов с реальными доменными именами. *`service`* – название службы в том виде, в каком оно приводится в файле */etc/services*. *`protocol`* – название протокола, связанного со службой в файле */etc/services*. *`name`* – стандартное имя узла или доменное имя, встречающееся во всех прочих полях *`name`*. Исходя из этих критериев для поиска серверов FTP домена *wrotethebook.com* можно использовать сочетание *_ftp._tcp.wrotethebook.com*.

ttl

Время жизни обычно не используется.

IN

Класс адресов – IN.

SRV

Тип записи – SRV.

preference

Число, используемое для выбора наиболее предпочтительного сервера в случае, когда запрос для службы возвращает несколько записей SRV. Сервер с наименьшим числом *preference* является наиболее предпочтительным. Весь трафик направлен в сторону наиболее предпочтительных серверов; серверы с более высокими значениями приоритета используются только в случаях, когда предпочтительные серверы недоступны.

weight

Число, определяющее долю трафика, приходящуюся на сервер; отчет ведется с 1. Если сервер A обладает весом 1, а сервер B – весом 2, сервер B получает в два раза больше трафика, чем сервер A. Параметр *weight* используется только для распределения нагрузки между серверами с равными значениями приоритета.

port

Номер порта указанной службы. Обычно это номер порта, определенный для службы в файле */etc/services*. Но можно указывать и нестандартные номера портов для служб, способных использовать нестандартные порты.

server

Каноническое имя машины, на которой функционирует служба.

D

dhcpd, справочник

В данном приложении описывается синтаксис команды dhcpd и файла настройки *dhcpd.conf*. Оно является справочником по серверу протокола динамической настройки (Internet Software Consortium Dynamic Host Configuration Protocol) – dhcpd. Чтобы полностью разобраться с настройкой и применением dhcpd в реально существующих сетях, обратитесь к руководству и примерам файлов настройки, приведенным в главе 9.

Информация в этом приложении относится к версии dhcpd, доступной на момент написания книги. Являясь бета-версией, программа, несомненно, будет дорабатываться и изменяться. Информация о наиболее актуальной версии dhcpd доступна по адресу *http://www.isc.org/dhcp.html*. И помните, другая реализация DHCP, вероятнее всего, настраивается совершенно иным образом.

Сборка dhcpd

Исходный текст dhcpd можно получить на веб-сайте ISC, расположенном по адресу *http://www.isc.org*, либо с анонимного FTP-сервера по адресу *ftp://ftp.isc.org/isc/dhcp*. Название сжатого tar-файла изменяется с выпуском новых версий. Однако самый последний вариант пакета должен храниться в файле с именем *dhcp-latest.tar.gz*. Скопируйте этот файл, распакуйте посредством gunzip и untar:

```
> ftp ftp.isc.org
Connected to pub1.bryant.vix.com.
220 pub1.bryant.vix.com FTP server ready.
Name (ftp.isc.org:craig): anonymous
331 Guest login ok, send your complete email address as password.
Password:
230 Guest login ok, access restrictions apply.
```

```
ftp> cd isc/dhcp
250 CWD command successful.
ftp> binary
200 Type set to I.
ftp> get dhcp-latest.tar.gz
200 PORT command successful.
150 Opening BINARY mode data connection for dhcp-latest.tar.gz
226 Transfer complete.
181892 bytes received in 17 secs (10 Kbytes/sec)
ftp> quit
221 Goodbye.
> gunzip dhcp-latest.tar.gz
> tar -xvf dhcp-latest.tar
drwxrwxr-x mellon/engsrc     0 2001-10-05 00:22:41 dhcp-3.0/
drwxrwxr-x mellon/engsrc     0 2001-10-05 00:22:32 dhcp-3.0/doc/
...
-rw-rw-r-- mellon/engsrc 150274 2001-08-23 12:25:51 dhcp-3.0/server/failover.c
-rw-rw-r-- mellon/engsrc  67711 2001-08-23 12:30:58 dhcp-3.0/server/mdb.c
-rw-rw-r-- mellon/engsrc  62087 2001-06-21 22:28:51 dhcp-3.0/server/omapi.c
-rw-rw-r-- mellon/engsrc   7612 2001-06-21 22:31:39 dhcp-3.0/server/salloc.c
-rw-rw-r-- mellon/engsrc  34248 2001-06-21 22:35:08 dhcp-3.0/server/stables.c
drwxrwxr-x mellon/engsrc      0 2001-10-05 00:22:42 dhcp-3.0/tests/
drwxrwxr-x mellon/engsrc      0 2001-10-05 00:22:42 dhcp-3.0/tests/failover/
-rw-rw-r-- mellon/engsrc   3585 2001-05-31 16:16:05 dhcp-3.0/tests/failover/dhcp-1.cf
-rw-rw-r-- mellon/engsrc   3463 2001-05-31 16:16:06 dhcp-3.0/tests/failover/dhcp-2.cf
-rwxrwxr-x mellon/engsrc    537 2001-05-31 16:16:07 dhcp-3.0/tests/failover/new-
failover
```

Перейдите во вновь созданный каталог и выполните `configure`. `configure` определяет тип используемой системы Unix и создает корректные для этой системы файлы сборки. Если `configure` не может определить вариант Unix, необходимо вручную создать файл сборки *Makefile*. Наберите `make`, чтобы выполнить компиляцию демона. Наконец, скопируйте полученный исполняемый файл и страницы руководства в подходящие каталоги:

```
# cd dhcp-3.0
# ./configure
System Type: linux
# make
cc -g      -c dhcpd.c -o dhcpd.o
cc -g      -c dhcp.c -o dhcp.o
cc -g      -c bootp.c -o bootp.o
...
nroff -man dhcpd.conf.5 >dhcpd.conf.cat5
# make install
```

Демон DHCP должен компилироваться без ошибок. Если вы получаете сообщения об ошибках компиляции либо если `configure` не может определить конфигурацию системы, следует прервать компиляцию и уведомить группу поддержки. Чтобы подписаться на список рассылки группы поддержки, посетите сайт *http://www.fugue.com/dhcp*. После регистрации отправьте сообщение на адрес списка рассылки *dhcp-server@fugue.com*: опишите конфигу-

рацию системы и подробно возникшую проблему. Подписчиками этого списка является большинство пользователей dhcpd. Возможно, вашу проблему уже кто-то решил.

Не исключено, что просто установкой dhcpd дело не кончится. Прочтите очень внимательно файл *README*. dhcpd работает с широким диапазоном систем, включая OSF/1, большинство современных производных от BSD, Solaris и Linux. Лучше всего демон чувствует себя в OSF/1 и BSD; в прочих системах могут присутствовать определенные ограничения. Например, в Solaris и Linux демон поддерживает работу лишь с одним сетевым интерфейсом. Кроме того, dhcpd может потребовать определенной настройки, специфичной для данной системы. Отличным примером такой ситуации являются старые системы с ядром Linux 2.0.0. Чтобы успешно работать с dhcpd на одной из таких систем, добавьте следующую запись в таблицу */etc/hosts*:

```
255.255.255.255 all-ones
```

Затем добавьте специальный маршрут для ограниченного широковещательного адреса, 255.255.255.255:

```
# route add -host all-ones dev eth0
```

Ограниченный широковещательный адрес следует добавлять в таблицу маршрутизации ядра после каждой загрузки системы, поэтому внесите следующий фрагмент кода в загрузочный сценарий:

```
# Установить ограниченный широковещательный адрес и запустить DHCP
  if [ -f /etc/dhcpd.conf ]; then
    echo -n " dhcpd"
    route add -host all-ones dev eth0
    /usr/sbin/dhcpd   fi
```

Информация, необходимая для выполнения этих действий, явным образом содержится в файле *README*. Прочтите его, прежде чем приступать к работе с dhcpd. Разумеется, такая настройка не требуется в современных версиях Linux, однако следует иметь в виду, что иногда приходится прибегать к дополнительным мерам.

Команда dhcpd

Синтаксис команды dhcpd:

```
dhcpd [-p порт] [-f] [-d] [-cf файл-настройки] [-lf файл-аренды] [if0 [ifn]]
```

dhcpd обычно запускается без каких-либо аргументов командной строки. Аргументы используются в основном для тестирования и отладки. Два аргумента связаны с необходимостью специальной настройки:

`-f`

Предписывает dhcpd работать в качестве приложения первого плана. По умолчанию dhcpd выполняется в качестве фонового процесса демона. Используйте -f при запуске dhcpd из файла *inittab* в системе Unix System V.

if0[*...ifn*]

Указывает интерфейсы, через которые dhcpd следует принимать пакеты BOOTREQUEST. Это перечень имен интерфейсов, разделенных пробелами. Например, команда dhcpd ec0 ec1 wd0 предписывает dhcpd прослушивать интерфейсы ec0, ec1 и wd0. В обычной ситуации этот аргумент не требуется. В большинстве случаев dhcpd самостоятельно находит все существующие интерфейсы и исключает интерфейсы, не допускающие широковещательной передачи. Используйте аргументы только в том случае, если окажется, что dhcpd неспособен самостоятельно выявить нужные интерфейсы.

Все прочие ключи командной строки используются для отладки или тестирования:

-p *порт*

Предписывает dhcpd принимать пакеты через нестандартный порт. Для DHCP широко известный порт имеет номер 67. Если порт изменить, клиенты не смогут обращаться к серверу. В редких случаях изменение порта выполняется в ходе тестирования.

-d

Передает сообщения об ошибках в поток ошибок stderr. Обычно сообщения об ошибках записываются в журнал посредством syslog в режиме DAEMON.

-cf *файл-настройки*

Предписывает dhcpd использовать указанный файл настройки вместо *dhcpd.conf*. Данный ключ применяйте только для тестирования новых настроек перед их записью в *dhcpd.conf*. В повседневной работе пользуйтесь стандартным файлом.

-lf *файл-аренды*

Предписывает dhcpd записывать сведения об аренде адресов в указанный файл (вместо *dhcpd.leases*). Данный ключ используйте только для тестирования. Изменение имени файла аренды может привести к некорректному динамическому выделению адресов. Применяйте данный ключ с осторожностью.

Работу демона dhcpd можно принудительно завершить сигналом SIGTERM. Идентификатор процесса (PID) демона dhcpd хранится в файле */var/run/dhcpd.pid* file. Пример:

```
# kill -TERM 'cat /var/run/dhcpd.pid'
```

dhcpd использует в работе три файла. Идентификатор процесса хранится в файле */var/run/dhcpd.pid*, записи об аренде адресов в файле */var/db/dhcpd.leases*, а настройки в файле */etc/dhcpd.conf*. Два последних файла создаются администратором. Создайте пустой файл аренды, прежде чем запустить dhcpd в первый раз (скажем, командой *touch /var/db/dhcpd.leases*). Создайте файл настройки с именем *dhcpd.conf*.

Файл настройки – dhcpd.conf

В момент запуска `dhcpd` читает свои настройки из файла */etc/dhcpd.conf*. В *dhcpd.conf* содержится определение сети, обслуживаемой сервером DHCP, и информация настройки, передаваемая сервером клиентам.

dhcpd.conf – это текстовый ASCII-файл. Символ решетки (#) обозначает начало комментария. Ключевые слова не чувствительны к регистру символов. Для форматирования файла могут использоваться пробелы. Связанные операторы группируются фигурными скобками. IP-адреса могут записываться в десятичном формате либо в виде имен узлов, преобразуемых в соответствующие адреса.

Операторы файла настройки определяют топологию сети, обслуживаемой сервером. В документации такие операторы называются «объявлениями», поскольку декларируют определенные факты, относящиеся к топологии сети. Следующие операторы определяют топологию сети: `shared-network`, `subnet`, `group` и `host`. Каждый из них может многократно встречаться в файле настройки. Операторы определяют иерархическую структуру. `shared-network` содержит подсети, а подсети могут содержать узлы.

С каждым из этих операторов могут быть связаны параметры и опции. Параметры – это определения, относящиеся к серверу и протоколу, такие как длительность аренды адреса либо расположение файла загрузки. Опции позволяют передавать клиентам стандартные значения DHCP, определенные различными документами RFC, такие, в частности, как необходимость включать пересылку IP-пакетов. Параметры и опции, расположенные вне конкретных операторов топологии, действуют для всех сетей, обслуживаемых данным сервером. Параметры и опции, указанные в группирующем операторе, действуют для всех общих сетей, подсетей либо узлов, сгруппированных в операторе. Параметры и опции оператора `shared-network` действуют для всех подсетей объединяющей их сети. Параметры и опции `subnet` действуют для всех объектов подсети. Параметры и опции оператора `host` – только для отдельного узла. Опции, действующие в глобальных масштабах, могут переопределяться теми же настройками на более низких уровнях. Опции `subnet` имеют приоритет более высокий, чем глобальные, а опции `host` – приоритет более высокий, чем опции `subnet`. Такая структура позволяет администратору сети задавать конфигурацию всей сети и отдельных ее частей.

В последующих разделах мы рассмотрим синтаксис всех операторов топологии, а также всех сопутствующих параметров и опций. Справочник содержит гораздо больше параметров и опций, чем вам когда-либо придется использовать, но нет необходимости изучать их все. Используйте настоящий справочник для поиска сведений по конкретным опциям и параметрам – по мере необходимости. Примеры использования операторов, параметров и опций в рабочих системах приведены в главе 9.

Операторы топологии

group {[*parameters*] [*options*]}

Оператор group группирует операторы shared-network, subnet, host и другие операторы group и позволяет применять наборы параметров и опций ко всем элементам группы.

shared-network *name* {[*parameters*] [*options*] }

Оператор shared-network используется только в случае, когда несколько IP-подсетей находятся в одной физической сети. В большинстве случаев различные подсети находятся в различных физических сетях. В качестве обязательного имени (*name*) может использоваться любое описательное имя. Оно используется только в отладочных сообщениях. Параметры и опции, связанные с общей сетью (*shared network*), объявляются внутри фигурных скобок и действуют на все подсети общей сети. Подсети общей сети должны быть определены внутри фигурных скобок оператора shared-network. Предполагается, что каждый оператор shared-network содержит, по меньшей мере, два оператора subnet; в противном случае нет необходимости использовать оператор shared-subnet. dhcpd не может определить, в какой из подсетей общей сети должен находиться клиент. Следовательно, динамически выделяемые адреса происходят из диапазонов адресов всех подсетей общей сети и назначаются по мере необходимости.

subnet *address* mask *netmask* { [*parameters*] [*options*] }

Определяет IP-адрес и адресную маску каждой подсети, обслуживаемой демоном. Адреса и маски используются для идентификации клиентов, принадлежащих подсети. Параметры и опции внутри фигурных скобок действуют для всех клиентов подсети. Каждой подсети, физически связанной с сервером, должен соответствовать оператор subnet, даже если в подсети вообще нет клиентов.

host *hostname* {[*parameters*] [*options*] }

Определяет параметры и опции для отдельных клиентов. Каждому клиенту BOOTP должен соответствовать оператор host в файле *dhcpd.conf*. Для клиентов DHCP оператор host является необязательным. Оператор host сопоставляется с клиентом DHCP или BOOTP на основе идентификатора dhcp-client-identifier, предоставленного клиентом, либо на основе параметра hardware, который сравнивается с аппаратным адресом клиента. Клиенты BOOTP не предоставляют идентификаторы dhcp-client-identifier, так что для них следует использовать аппаратные адреса. Идентификация клиентов DHCP может выполняться при помощи значения dhcp-client-identifier либо аппаратного адреса.

Параметры конфигурации

Операторы параметров, описанные в данном разделе, управляют работой сервера DHCP и протокола DHCP. Стандартные значения настройки DHCP, передаваемые клиентам, определяются посредством операторов опций и

описаны в следующем разделе. Некоторые операторы параметров могут быть связаны с любым из описанных выше операторов топологии, другие могут использоваться только с конкретными операторами. Последние отмечены соответствующим образом в описаниях.

`range [dynamic-bootp] low-address [high-address] ;`

Параметр `range` определяет диапазон (верхнюю и нижнюю границу) IP-адресов, доступных для динамического назначения. Параметр `range` должен быть связан с оператором `subnet`. Все адреса диапазона, определенного параметром `range`, должны принадлежать подсети, с которой связан оператор. Флаг `dynamic-bootp` указывает, что адреса могут автоматически назначаться клиентам BOOTP наравне с клиентами DHCP. Параметр `range` должен присутствовать, если вы намереваетесь задействовать динамическое назначение адресов. Если оператор `subnet` не содержит параметра `range`, для клиентов подсети динамическое назначение адресов не действует.

`default-lease-time seconds;`

Значение длительности аренды адреса в секундах, используемое, если клиент явным образом не запрашивает определенную длительность аренды.

`max-lease-time seconds;`

Максимально разрешенная длительность аренды адреса в секундах, независимо от длительности аренды, запрошенной клиентом.

`hardware type address;`

Определяет аппаратный адрес клиента. В настоящее время тип (*type*) должен иметь значение `ethernet` или `token-ring`. Адрес (*address*) должен быть представлен соответствующим данному типу устройства физическим адресом. Параметр `hardware` должен быть связан с оператором `host`. Он необходим для распознавания клиента BOOTP, а для клиентов DHCP является необязательным, хотя может выступать в роли альтернативы идентификатору `dhcp-client-identifier`.

`filename file;`

Указывает файл загрузки для бездисковых клиентов. Имя файла (*file*) – это ASCII-строка, заключенная в кавычки.

`server-name name;`

Имя узла сервера DHCP, передаваемое клиенту. Имя узла (*name*) – это ASCII-строка, заключенная в кавычки.

`next-server name;`

Имя узла или адрес сервера, с которого следует получать загрузочный файл.

`fixed-address address[, address... ] ;`

Назначает узлу один или несколько фиксированных IP-адресов. Параметр `fixed-address` действителен только в сочетании с оператором `host`. Если указано несколько адресов, клиенту назначается адрес, корректный

для сети, из которой выполняет загрузку клиент. Если такого адреса в списке нет, клиенту не передаются никакие данные настройки.

`dynamic-bootp-lease-cutoff` *`date`*;

Устанавливает дату завершения действия адресов, назначенных клиентам BOOTP. Клиенты BOOTP не обладают способностью обновлять аренду и не знают, что срок действия аренды может истечь. По умолчанию `dhcpd` назначает клиентам BOOTP постоянные адреса. Данный параметр изменяет поведение `dhcpd`. Он используется только в особых ситуациях, когда длительность жизни всех систем известна заранее – например, в студенческом городке может быть заранее известно, что системы всех студентов будут удалены из сети к июню.

`dynamic-bootp-lease-length` *`seconds`*;

Определяет длительность аренды в секундах для адресов, автоматически назначаемых клиентам BOOTP. Как говорилось выше, клиенты BOOTP не способны воспринимать аренду адресов. Данный параметр используется только в особых ситуациях, когда клиенты используют образ загрузки BOOTP PROM и работают с операционной системой, поддерживающей DHCP. В ходе загрузки клиент действует в качестве клиента BOOTP, а после загрузки начинает работать с протоколом DHCP и умеет обновлять аренду. Данный параметр, как и предыдущий, следует использовать с осторожностью.

`get-lease-hostnames` *`flag`*;

Указывает `dhcpd`, следует ли передавать клиенту имя узла DNS при динамическом назначении IP-адреса. Если флаг имеет значение `true`, `dhcpd` использует DNS для поиска имен, соответствующих всем динамически назначаемым адресам, что существенно снижает производительность DHCP. По умолчанию флаг (*`flag`*) имеет значение `false`, и поиск не производится.

`use-host-decl-names` *`flag`*;

Предписывает передавать имя, указанное в операторе `host`, клиенту в качестве его имени узла.

`server-identifier` *`hostname`*;

Определяет значение, передаваемое в качестве идентификатора сервера. По умолчанию передается первый IP-адрес сетевого интерфейса.

`authoritative`;
`not authoritative`;

Указывает, является ли сервер DHCP компетентным. По умолчанию принимается значение `authoritative`. `not authoritative` может использоваться, если в компетенцию сервера DHCP не входит назначение адресов клиентам. Сервер DHCP может обслуживать несколько сетей, иметь полномочия назначения адресов в одних сетях и не иметь в других.

`use-lease-addr-for-default-route` *`flag`*;

Предписывает передавать клиенту арендованный IP-адрес в качестве маршрута по умолчанию, что принуждает клиентов Windows 95 исполь-

зовать ARP для всех IP-адресов. Параметр используется только в случае, когда локальный маршрутизатор является сервером-посредником ARP. Оператор настройки `routers` имеет более высокий приоритет.

`always-reply-rfc1048` *`flag`*;

Предписывает посылать клиенту BOOTP ответы в соответствии с RFC 1048, даже если от клиента исходят запросы, не подчиняющиеся этому документу. Этот параметр используется, если сервер записывает в журнал сообщение «(non-rfc1048)» в ответ на запрос BOOTREQUEST клиента BOOTP. Данный параметр используется преимущественно для отдельных клиентов. Предпочтительно обновление клиентов, дающее им способность работать с протоколом DHCP.

`allow` *`keyword`*;
`deny` *`keyword`*;

Определяет необходимость отвечать на запросы различных типов. Ключевое слово (*`keyword`*) указывает тип разрешенных (`allow`) или запрещенных (`deny`) запросов. Существуют следующие ключевые слова:

`unknown-clients`

Определяет возможность динамического назначения адресов неизвестным клиентам. По умолчанию динамические адреса неизвестным клиентам назначаются.

`bootp`

Определяет необходимость отвечать на запросы BOOTP. По умолчанию запросы BOOTP обслуживаются.

`booting`

Используется внутри объявления `host` для указания необходимости отвечать тому или иному клиенту. По умолчанию сервер DHCP отвечает всем клиентам.

Опции DHCP

Операторы опций `dhcpd` отражают все стандартные параметры настройки DHCP, определенные в существующих RFC. Более того, синтаксис оператора *dhcpd.conf* `option` является расширяемым. Новая опция может быть обозначена своим десятичным кодом. Всем опциям присваивается такой код – в документе RFC, описывающем опцию, либо в документации разработчика системы. Назначенное новой опции значение может быть выражено в виде заключенной в кавычки строки либо в виде списка шестнадцатеричных значений, разделенных двоеточиями. Представим, что появилась новая опция DHCP и получила код 133. Предположим также, что значение этой опции является 16-разрядной двоичной маской, и клиенты должны иметь возможность «включать» 4 старших разряда и «отключать» все прочие разряды маски. Тогда в конфигурацию можно добавить следующую строку:

```
option option-133 F0:00
```

Все операторы опций начинаются с ключевого слова `option`. За ключевым словом следует имя и значение, строго в этом порядке. В приведенном примере имя опции имеет формат `option-`*`nnn`*, где *`nnn`* – десятичный код, назначенный опции. Таким образом, в файле *dhcpd.conf* может фигурировать любая новая опция. Значение, назначенное этой воображаемой опции, – F000.

При взгляде на огромный список стандартных опций может возникнуть закономерный вопрос: возникнет ли необходимость расширять его. Стандартные опции перечислены в следующих разделах и могут иметь следующие типы значений:

Адрес (Address)

IP-адрес в десятичной записи через точку либо имя узла, соответствующее этому адресу.

Строка (String)

Последовательность символов, заключенная в кавычки.

Число (Number)

Численное значение.

Флаг (Flag)

Переключатель, принимающий значения `true` или `false`, либо `1` или `0`, либо `yes` или `no`.

В данной книге перечень опций поделен на «Общеупотребительные опции» и «Прочие опции».

Общеупотребительные опции

`option subnet-mask` *`mask`*;

Определяет маску подсети в формате десятичной записи через точку. Если `subnet-mask` отсутствует в файле *dhcpd.conf*, `dhcpd` использует маску подсети из оператора `subnet`.

`option time-offset` *`seconds`*;

Указывает разницу данного часового пояса со временем UTC (Coordinated Universal Time) в секундах.

`option routers` *`address`*`[, address...] ;`

Перечисляет доступные клиентам маршрутизаторы в порядке предпочтения.

`option domain-name-servers` *`address`*`[, address...] ;`

Перечисляет доступные клиентам серверы имен DNS (Domain Name System) в порядке предпочтения.

`option lpr-servers` *`address`* `[, address...] ;`

Перечисляет доступные клиентам серверы печати LPR в порядке предпочтения.

`option host-name host;`

Указывает имя узла для клиента.

`option domain-name domain;`

Определяет имя домена.

`option interface-mtu bytes;`

Указывает значение MTU для клиента. Минимальное допустимое значение для MTU – 68.

`option broadcast-address address;`

Определяет широковещательный адрес для подсети клиента.

`option static-routes destination gateway[, destination gateway... ] ;`

Перечисляет доступные клиенту статические маршруты. Маршрут по умолчанию не может быть указан таким способом. Воспользуйтесь с целью его указания опцией `routers`.

`option trailer-encapsulation 0 | 1;`

Определяет, следует ли клиенту выполнять инкапсуляцию завершителей (trailer encapsulation; оптимизация, основанная на изменении порядка данных). 0 означает, что инкапсуляцию выполнять не следует, тогда как 1 предписывает включить оптимизацию.

`option nis-domain string;`

Строка символов, определяющая имя домена NIS (Network Information Services).

`option nis-servers address[, address...] ;`

Перечисляет IP-адреса доступных клиентам серверов NIS в порядке предпочтения.

`option dhcp-client-identifier string;`

Используется в операторе `host` для определения идентификатора клиента DHCP. `dhcpd` может использовать данное значение для идентификации клиентов DHCP вместо аппаратного адреса.

Прочие опции

`option time-servers address[, address...] ;`

Перечисляет доступные клиентам серверы синхронизации времени в порядке предпочтения.

`option ien116-name-servers address[, address...];`

Перечисляет доступные клиентам серверы IEN 116 в порядке предпочтения. IEN 116 – это вышедшая из употребления служба имен. Используйте DNS.

`option log-servers address[, address...] ;`

Перечисляет доступные клиентам серверы журналов MIT-LCS UDP в порядке предпочтения.

`option cookie-servers` *address*`[, ` *address*`...] ;`

Перечисляет доступные клиентам серверы cookie-квитанций в порядке предпочтения.

`option impress-servers` *address*`[, ` *address*`...] ;`

Перечисляет доступные клиентам серверы Image Impress в порядке предпочтения.

`option resource-location-servers` *address*`[, ` *address*`...] ;`

Перечисляет доступные клиентам серверы Resource Location в порядке предпочтения.

`option boot-size` *blocks*`;`

Указывает число блоков из 512 байт в загрузочном файле.

`option merit-dump` *path*`;`

path – это строка символов, определяющая расположение файла, в который записывается образ памяти клиента в случае сбоя.

`option swap-server` *address*`;`

Указывает IP-адрес сервера подкачки клиента.

`option root-path` *path*`;`

path – это строка символов, определяющая расположение корневой файловой системы.

`option ip-forwarding 0 | 1;`

Указывает, следует ли клиенту выполнять пересылку IP-пакетов (IP forwarding). Значение 0 отключает пересылку, значение 1 – включает.

`option non-local-source-routing 0 | 1;`

Разрешает или запрещает клиенту работать с нелокальными маршрутами, исходящими от источника передачи. Такие маршруты представляют собой потенциальную угрозу безопасности, поскольку могут использоваться злоумышленниками для маршрутизации данных из локальной сети способами, не предусмотренными администратором локальной сети. Значение 0 отключает пересылку дейтаграмм по таким маршрутам, 1 разрешает пересылку. 0 – более безопасный вариант.

`option policy-filter` *address mask*`[, ` *address mask*`...] ;`

Перечисляет пары IP-адресов и масок, допустимые для использования в качестве координат пункта назначения для входящих маршрутов. Любая входящая дейтаграмма, маршрутизируемая источником, адрес следующей транзитной точки которой не совпадает с одним из перечисленных, удаляется клиентом.

`option max-dgram-reassembly` *bytes*`;`

Определяет максимальный размер дейтаграммы (в байтах), которую должен быть готов собрать клиент. Значение *bytes* не может быть меньшим 576.

`option default-ip-ttl` *ttl* ;

Определяет время жизни по умолчанию (time-to-live, TTL) для исходящих дейтаграмм.

`option path-mtu-aging-timeout` *seconds*;

Определяет длительность хранения значений Path MTU, обнаруженных механизмом, описанным в RFC 1191.

`option path-mtu-plateau-table` *bytes*[, *bytes*...] ;

Определяет таблицу размеров MTU, используемую в механизме Path MTU Discovery, описанном в RFC 1191. Значение MTU не может быть меньшим 68.

`option all-subnets-local 0 | 1;`

Сообщает клиенту, все ли подсети локальной сети используют одно и то же значение MTU. 1 означает, что все подсети используют одинаковое значение MTU. 0 означает, что в некоторых подсетях значения MTU меньше.

`option perform-mask-discovery 0 | 1;`

Указывает, следует ли клиенту использовать ICMP для обнаружения маски подсети. Значение 0 разрешает ICMP-обнаружение масок, значение 1 запрещает. Поскольку сервер DHCP способен предоставить верную маску подсети, данный механизм редко используется в сетях с серверами DHCP.

`option mask-supplier 0 | 1;`

Указывает, следует ли клиенту отвечать на ICMP-запросы маски подсети. 0 означает, что отвечать не следует, тогда как 1 означает, что следует отвечать.

`option router-discovery 0 | 1;`

Указывает, следует ли клиенту для поиска маршрутизаторов использовать механизм Router Discovery, определенный в документе RFC 1256. 0 означает, что не следует использовать механизм, тогда как значение 1 означает, что следует выполнять обнаружение маршрутизаторов. Поскольку сервер DHCP предоставляет перечень доступных маршрутизаторов, механизм обнаружения редко используется в сетях с серверами DHCP.

`option router-solicitation-address` *address*;

Определяет адрес, на который клиенту следует передавать запрос поиска маршрутизатора, если доступно обнаружение маршрутизаторов.

`option arp-cache-timeout` *seconds*;

Определяет длительность хранения записей в кэше ARP в секундах.

`option ieee802-3-encapsulation 0 | 1;`

Указывает тип Ethernet-инкапсуляции, который следует использовать клиенту в этой сети: Ethernet II (DIX) или IEEE 802.3 Ethernet. 0 предписывает клиенту использовать Ethernet II, а 1 – использовать инкапсуляцию IEEE 802.3.

`option default-tcp-ttl` *`ttl`*;

Определяет значение TTL по умолчанию для сегментов TCP. Допустимы значения из интервала от 1 до 255.

`option tcp-keepalive-interval` *`seconds`*;

Число секунд ожидания перед посылкой протоколом TCP сообщения keepalive. 0 запрещает TCP генерировать такие сообщения. Использование keepalive-сообщений обычно не приветствуется.

`option tcp-keepalive-garbage 0 | 1;`

Определяет, должен ли клиент посылать сообщения TCP keepalive с байтом «мусора» в целях совместимости с более старыми реализациями. 0 означает, что нет необходимости посылать такой байт, 1 – что такая необходимость есть. Использование keepalive-сообщений обычно не приветствуется.

`option ntp-servers` *`address`*`[, address...] ;`

Перечисляет IP-адреса доступных клиенту серверов NTP (Network Time Protocol) в порядке предпочтения.

`option netbios-name-servers` *`address`*`[, address...] ;`

Перечисляет доступные клиенту серверы имен NetBIOS (NBNS, NetBIOS name servers) в порядке предпочтения.

`option netbios-dd-server` *`address`*`[, address...] ;`

Перечисляет доступные клиенту серверы распределения дейтаграмм NetBIOS (NBDD, NetBIOS datagram distribution servers) в порядке предпочтения.

`option netbios-node-type` *`type`*;

Определяет тип узла NetBIOS клиента. Значение 1 определяет B-узел NetBIOS; значение 2 определяет P-узел; 4 – M-узел; 8 – H-узел.

`option netbios-scope` *`string`*;

Символьная строка, определяющая параметр диапазона NetBIOS over TCP/IP в соответствии с RFC 1001/1002.

`option font-servers` *`address`*`[, address...] ;`

Перечисляет доступные клиенту серверы шрифтов X Window System Font в порядке предпочтения.

`option x-display-manager` *`address`*`[, address...] ;`

Перечисляет доступные клиенту системы, на которых функционирует менеджер X Window System Display Manager в порядке предпочтения.

`option nisplus-domain` *`string`*;

Определяет имя домена NIS+.

`option nisplus-servers` *`ip-address`* `[,` *`ip-address...`* `];`

Перечисляет IP-адреса серверов NIS+ в порядке предпочтения.

`option tftp-server-name` *`string`*;

Указывает сервер загрузки TFTP.

`option bootfile-name` *`string`*;

Указывает имя загрузочного файла, доступного на сервере загрузки TFTP.

`option mobile-ip-home-agent` *`ip-address`* [, *`ip-address`*...];

Перечисляет IP-адреса домашних агентов Mobile IP, доступных клиенту.

`option smtp-server` *`ip-address`* [, *`ip-address`*...];

Перечисляет IP-адреса серверов SMTP в порядке предпочтения.

`option pop-server` *`ip-address`* [, *`ip-address`*...];

Перечисляет IP-адреса серверов POP3 в порядке предпочтения.

`option nntp-server` *`ip-address`* [, *`ip-address`*...];

Перечисляет IP-адреса серверов Network News Transport Protocol (NNTP) в порядке предпочтения.

`option www-server` *`ip-address`* [, *`ip-address`*...];

Перечисляет IP-адреса веб-серверов в порядке предпочтения.

`option finger-server` *`ip-address`* [, *`ip-address`*...];

Перечисляет IP-адреса серверов finger в порядке предпочтения.

`option irc-server` *`ip-address`* [, *`ip-address`*...];

Перечисляет IP-адреса серверов IRC в порядке предпочтения.

`option streettalk-server` *`ip-address`* [, *`ip-address`*...];

Перечисляет IP-адреса серверов StreetTalk в порядке предпочтения.

`option streettalk-directory-assistance-server` *`ip-address`* [, *`ip-address`*...];

Перечисляет IP-адреса серверов StreetTalk Directory Assistance (STDA) в порядке предпочтения.

E

sendmail, справочник

В данном приложении содержатся подробные сведения о синтаксисе команды `sendmail`, файла *sendmail.cf*, а также макроопределений `m4`, которые могут использоваться для создания этого файла. Вы узнаете, как получить последнюю версию исходного текста sendmail и скомпилировать его. Это справочник, а не руководство. Руководство по настройке sendmail содержится в главе 10.

Приложение начинается с информации о поиске, загрузке и компиляции новейшей версии sendmail.

Компиляция sendmail

Исходный текст sendmail доступен для анонимного FTP-копирования с сервера *ftp.sendmail.org*, где он хранится в каталоге *pub/sendmail*. sendmail постоянно обновляется. В последующих примерах использован дистрибутив sendmail 8.11.3. Помните, что в новых версиях происходят изменения. Всегда обращайтесь к файлам README и установочной документации, прежде чем начинать обновление.

Чтобы получить возможность скомпилировать программу sendmail, загрузите сжатый `tar`-файл в качестве двоичного, а затем распакуйте и извлеките архив при помощи команды `tar`, как показано ниже:

```
$ ftp ftp.sendmail.org
Connected to ftp.sendmail.org.
220 pub2.pa.vix.com FTP server ready.
Name (ftp.sendmail.org:craig): anonymous
331 Guest login ok, send your e-mail address as password.
Password:
230 Guest login ok, access restrictions apply.
```

```
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd pub/sendmail
ftp> get sendmail.8.11.3.tar.gz
local: sendmail.8.11.3.tar.gz remote: sendmail.8.11.3.tar.gz
200 PORT command successful.
150 Opening BINARY mode data connection for sendmail.8.11.3.tar.gz
   (1347756 bytes).
226 Transfer complete.
1347756 bytes received in 18.68 Seconds (72.42 Kbytes/sec)
ftp> quit
221-You have transferred 1347756 bytes in 1 files.
221-Thank you for using the FTP service on pub2.pa.vix.com.
221 Goodbye.
$ cd /usr/local/src
$ tar -zxvf /home/craig/sendmail.8.11.3.tar.gz
```

Затем перейдите в каталог *sendmail-8.11.3*, созданный при распаковке архива, и воспользуйтесь сценарием `Build` для компиляции новой программы sendmail:

```
$ cd sendmail-8.11.3
$ ./Build
Making all in:
/usr/local/src/sendmail-8.11.3/libsmutil
Configuration: pfx=, os=Linux, rel=2.2.10, rbase=2, rroot=2.2, arch=i586, sfx=,
variant=optimized
Using M4=/usr/bin/m4
Creating ../obj.Linux.2.2.10.i586/libsmutil using ../devtools/OS/Linux
Making dependencies in ../obj.Linux.2.2.10.i586/libsmutil
make[1]: Entering directory
     `/usr/local/src/sendmail-8.11.3/obj.Linux.2.2.10.i586/libsmutil'
cc -M -I. -I../../sendmail -I../../include -DNEWDB
      -DNOT_SENDMAIL debug.c
errstring.c lockfile.c safefile.c snprintf.c strl.c    >> Makefile
make[1]: Leaving directory
      `/usr/local/src/sendmail-8.11.3/obj.Linux.2.2.10.i586/libsmutil'
Making in ../obj.Linux.2.2.10.i586/libsmutil
make[1]: Entering directory
      `/usr/local/src/sendmail-8.11.3/obj.Linux.2.2.10.i586/libsmutil'
cc -O -I. -I../../sendmail -I../../include -DNEWDB
      -DNOT_SENDMAIL -c debug.c -o debug.o
cc -O -I. -I../../sendmail -I../../include -DNEWDB
      -DNOT_SENDMAIL -c errstring.c -o errstring.o

... Удалено много, много, много строк...

cc -O -I. -I../../sendmail -I../../include -DNEWDB
      -DNOT_SENDMAIL -c vacation.c -o vacation.o
cc -o vacation   vacation.o ../libsmdb/libsmdb.a
      ../libsmutil/libsmutil.a -ldb -lresolv -lcrypt -lnsl -ldl
groff -Tascii -man vacation.1 > vacation.0 ||
```

```
        cp vacation.0.dist vacation.0
make[1]: Leaving directory
        `/usr/local/src/sendmail-8.11.3/obj.Linux.2.2.10.i586/vacation'
```

Сценарий `Build` самостоятельно определяет архитектуру системы и создает корректный для данной системы файл сборки Makefile. Затем выполняется компиляция sendmail при помощи созданного Makefile.

Если верить документации, выполнение `Build` – все, что нужно для компиляции sendmail в большинстве систем. Такой способ работает в Red Hat Linux и Solaris 8. Однако гарантий никаких нет. В вашей системе могут использоваться нестандартные каталоги или отсутствовать определенные библиотеки. Может потребоваться уточнить флаги компилятора для вашей системы.

Если у вас есть опыт компиляции sendmail, вы можете испытать соблазн заглянуть в поисках параметров компиляции в файл Makefile, расположенный в каталоге исходных текстов дистрибутива. Возможно, вы даже вспомните, что некогда указывали параметры компиляции именно в этом файле. Но теперь все иначе. Параметры компилятора устанавливаются в файлах, расположенных в каталоге *devtools* дистрибутива исходных текстов sendmail.

Параметры компилятора, принимаемые по умолчанию, обычно устанавливаются в специальных для каждой ОС файлах из каталога *devtools/OS* и изменяются в файлах, создаваемых непосредственно для вашего сервера в каталоге *devtools/Site*. Имена файлов devtools/OS основаны на названиях операционных систем; к примеру, файл настройки Solaris 8 называется SunOS5.8. Если ваша система Solaris 8 отличается от стандартной, создайте собственный файл в каталоге *devtools/Site*, дайте ему имя *site.SunOS5.8.m4*. В этом файле сохраните скорректированные настройки. Кроме того, вы можете создать файл *site.config.m4* в каталоге *devtools/Site*, если устанавливаемые параметры компилятора относятся более к особенностям сервера, чем к поправкам на операционную систему. `Build` находит и использует файлы с такими именами.

Как следует из расширений *.m4* имен данных файлов, команды, используемые для установки параметров компилятора, являются макроопределениями `m4`, а не простыми параметрами компилятора. В табл. E.1 перечислены команды `m4`, используемые в sendmail 8.11.3 для управления процессом компиляции.

Таблица E.1. m4, параметры компилятора

Команда	Назначение
`confBEFORE`	Указывает файлы, которые необходимо создать перед компиляцией
`confBLDVARIANT`	Запрашивает варианты сборки OPTIMIZED, DEBUG или PURIFY
`confBUILDBIN`	Путь к вспомогательным исполняемым файлам сборки
`confCC`	Имя компилятора C

Таблица E.1 (продолжение)

Команда	Назначение
confCCOPTS	Параметры, передаваемые компилятору
confCOPY	Имя программы, копирующей файлы
confDEPEND_TYPE	Имя файла из *devtools/M4/depend*, определяющего способ построения зависимостей
confDEPLIBS	Зависимые библиотеки для разделяемых объектов
confEBINDIR	Путь к программе, исполняемой другими программами
confENVDEF	Флаги -D, передаваемые компилятору
confFORCE_RMAIL	Принудительная установка rmail
confHFDIR	Путь к файлу справки sendmail
confHFFILE	Имя файла справки
confINCDIRS	Флаги -I, передаваемые компилятору
confINCGRP	Идентификатор группы для включаемых файлов
confINCMODE	Файловые права доступа для включаемых файлов
confINCOWN	Идентификатор пользователя для включаемых файлов
confINCLUDEDIR	Путь установки включаемых файлов
confINSTALL	Программа установки
confINSTALL_RAWMAN	Устанавливать неформатированные страницы руководства
confLDOPTS	Параметры для редактора связей
confLIBDIR	Путь к файлам библиотеки установки
confLIBDIRS	Флаги -L для редактора связей
confLIBGRP	Идентификатор группы для библиотек
confLIBMODE	Файловые права доступа для библиотек
confLIBOWN	Идентификатор пользователя для библиотек
confLIBS	Флаги -I, передаваемые редактору связей
confLIBSEARCH	Имена библиотек, используемых при связывании
confLIBSEARCHPATH	Путь к библиотекам, используемым при связывании
confLINKS	Имена логических ссылок на sendmail, к примеру, newaliases
confLN	Команда, используемая для создания логических ссылок

Команда	Назначение
confMAN1	Путь к файлам man1
confMAN1EXT	Расширение имен файлов man1
confMAN1SRC	Источник страниц man1
confMAN3	Путь к файлам man3
confMAN3EXT	Расширение имен файлов man3
confMAN3SRC	Источник страниц man3
confMAN4	Путь к файлам man4
confMAN4EXT	Расширение имен файлов man4
confMAN4SRC	Источник страниц man4
confMAN5	Путь к файлам man5
confMAN5EXT	Расширение имен файлов man5
confMAN5SRC	Источник страниц man5
confMAN8	Путь к файлам man8
confMAN8EXT	Расширение имен файлов man8
confMAN8SRC	Источник страниц man8
confMANDOC	Макроопределения, используемые для форматирования страниц руководства
confMANGRP	Идентификатор группы для файлов страниц руководства
confMANMODE	Файловые права доступа для страниц руководства
confMANOWN	Идентификатор пользователя для файлов страниц руководства
confMANROOT	Корневой каталог для различных каталогов, содержащих отформатированные страницы руководства
confMANROOTMAN	Корневой каталог для различных каталогов, содержащих неформатированные страницы руководства
confMAPDEF	Указывает типы баз данных, поддержку которых следует встроить в sendmail
confMBINDIR	Путь установки для программы sendmail
confNO_HELPFILE_INSTALL	Не устанавливать файл справки
confNO_MAN_BUILD	Не создавать страницы руководства
confNO_MAN_INSTALL	Не устанавливать страницы руководства
confNO_STATISTICS_INSTALL	Не устанавливать файл статистики
confNROFF	Команда, используемая для форматирования страниц руководства

Таблица E.1 (продолжение)

Команда	Назначение
confOBJADD	Указывает объекты, которые следует встроить в sendmail
confOPTIMIZE	Флаги, передаваемые компилятору в виде ${O}
confRANLIB	Путь к программе ranlib
confRANLIBOPTS	Параметры, передаваемые ranlib
confSBINDIR	Каталог, в котором сохраняются команды вроде makemap
confSBINGRP	Идентификатор группы для исполняемых файлов setuid
confSBINMODE	Файловые права доступа для исполняемых файлов setuid
confSBINOWN	Идентификатор пользователя для исполняемых файлов setuid
confSHAREDLIB_EXT	Расширения имен файлов разделяемых библиотек
confSHAREDLIB_SUFFIX	Суффикс для разделяемых объектов
confSHAREDLIBDIR	Каталог, в который устанавливаются разделяемые библиотеки
confSHELL	Полное имя интерпретатора команд, используемого make
confSMOBJADD	Объекты, которые следует встраивать в sendmail
confSMSRCADD	Исходные файлы на языке C для объектов, перечисленных в confSMOBJADD
confSMSRCDIR	Каталог, содержащий исходные тексты sendmail
confSRCADD	Исходные файлы на языке C для объектов, перечисленных в confOBJADD
confSRCDIR	Путь к иерархии каталогов исходных текстов
confSONAME	Флаг редактора связей для записи имени разделяемого объекта
confSTDIO_TYPE	Указывает тип буферной зоны, portable или torek
confSTDIR	Каталог, в котором сохраняется файл статистики
confSTFILE	Имя файла статистики
confSTRIP	Указывает программу, используемую для удаления из исполняемых файлов отладочной информации
confSTRIPOPTS	Параметры, передаваемые программе confSTRIP

Команда	Назначение
confUBINDIR	Путь к пользовательским программам
confUBINGRP	Идентификатор группы для пользовательских исполняемых программ
confUBINMODE	Файловые права доступа для пользовательских исполняемых программ
confUBINOWN	Идентификатор пользователя для пользовательских исполняемых программ

После компиляции sendmail устанавливается посредством команды Build с аргументом install, как показано ниже:

```
# ./Build install
Making all in:
/usr/local/src/sendmail-8.11.3/libsmutil
Configuration: pfx=, os=Linux, rel=2.2.10, rbase=2, rroot=2.2,
      arch=i586, sfx=, variant=optimized
Making in ../obj.Linux.2.2.10.i586/libsmutil
make[1]: Entering directory
      `/usr/local/src/sendmail-8.11.3/obj.Linux.2.2.10.i586/libsmutil'

... Удалено много, много, много строк...

Making in ../obj.Linux.2.2.10.i586/vacation
make[1]: Entering directory
      `/usr/local/src/sendmail-8.11.3/obj.Linux.2.2.10.i586/vacation'
install -c -o bin -g bin -m 555 vacation /usr/bin
install -c -o bin -g bin -m 444 vacation.0 /usr/man/man1/vacation.1
make[1]: Leaving directory
      `/usr/local/src/sendmail-8.11.3/obj.Linux.2.2.10.i586/vacation'
```

Команда Build устанавливает страницы руководства, исполняемые файлы, файл справки, а также файл состояния в соответствующие каталоги вашей системы.

sendmail готова к запуску. В следующем разделе описывается синтаксис команды sendmail.

Команда sendmail

Синтаксис команды sendmail обманчиво прост:

```
sendmail [аргументы] [адрес ...]
```

Синтаксис обманчив, поскольку не отражает того факта, что число аргументов командной строки очень велико. Все эти аргументы перечислены в табл. E.2.

Таблица E.2. Аргументы командной строки sendmail

Аргумент	Назначение
-U	Сообщение передается пользователем
-V*envid*	Установить идентификатор конверта в *envid*
-N*dsn*	Установить значение уведомления о состоянии доставки в *dsn*
-M*xvalue*	Назначить макроопределению *x* значение *value*
-R*return*	Указать часть сообщения, возвращаемую наряду с ошибкой
-B*type*	Установить MIME-тип для тела сообщения
-p*protocol*	Указать протокол приема и имя узла
-X*logfile*	Записывать весь трафик в файл журнала *logfile*
-f*addr*	Адрес машины отправителя – *addr*
-r *addr*	Устаревшая форма ключа -f
-h *cnt*	Удалить сообщение, подвергшееся пересылке *cnt* раз
-F*name*	Указать полное имя данного пользователя (*name*)
-n	Не выполнять обработку псевдонимов и пересылку
-T*value*	Установить значение *value* для параметра QueueTimeout
-t	Послать копии сообщения по всем адресам из полей To:, Cc: и Bcc:
-bm	Доставка почты (режим по умолчанию)
-bD	Работа в качестве демона в приоритетном режиме
-ba	Режим ARPAnet
-bs	Указывать SMTP на входе
-bd	Режим демона
-bH	Очистить каталог состояния узла; эквивалент purgestat
-bh	Вывести отчет о состоянии узла; эквивалент hoststat
-bt	Работа в режиме тестирования
-bv	Проверить адреса; не выполнять сбор и доставку сообщений
-bi	Инициализировать базу данных псевдонимов
-bp	Вывести почтовую очередь
-bz	Создать разобранную копию файла *sendmail.cf*
-q[*time*]	Обработать сообщения в почтовой очереди. Повторять операцию с указанным интервалом времени (*time*)
-C*file*	Использовать указанный файл в качестве файла настройки
-c	Установить параметр HoldExpensive в значение true
-d*level*	Установить уровень отладки

Аргумент	Назначение
-e	Установить значение параметра ErrorMode
-O*option=value*	Установить параметр *option* в значение *value*
-o*xvalue*	Установить значение параметра при помощи его старого, односимвольного имени
-I	Альтернативный способ указать ключ -bi
-i	Игнорировать точки в поступающих сообщениях
-m	Послать копию отправителю
-v	Режим подробной диагностики
-s*addr*	Альтернативная форма ключа -f

В табл. E.2 перечислено более 30 аргументов командной строки. Эта таблица является кратким справочником по всем существующим аргументам, многие из которых в последней версии sendmail уже не используются. Вероятно, наиболее известным из таких аргументов является ключ -bz. В свое время он использовался для предварительной обработки файла *sendmail.cf*. Идея заключалась в том, что хранение обработанных настроек повышает скорость работы. Этот устаревший ключ не работает в новейших версиях sendmail. Если вы пользовались этим аргументом в более старых версиях sendmail, то можете ошибочно предполагать, что он все еще необходим. Попытка использовать данный ключ с современной версией sendmail приведет к ошибке.

Отдельные аргументы являются избыточными формами других ключей. Например, ключи -c, -e, -I, -m, -r, -T и -s устарели, а на смену им пришли другие аргументы. Все аргументы, выполняющие установку параметров *sendmail.cf*, даже те, что еще в ходу, такие как -i и -o, могут эмулироваться ключом -O. Так, команду:

```
sendmail -m -s < mail.file
```

можно заменить на:

```
sendmail -OMeToo=true -OSaveFromLine=true < mail.file
```

Ключ -O обладает тем явным преимуществом, что позволяет устанавливать любой параметр файла *sendmail.cf*. Аргументы вроде -m и -s устанавливают каждый только по одному параметру. Конструкцию -O также проще читать и понимать, особенно когда команда sendmail фигурирует в сценарии.

Некоторые аргументы командной строки из табл. E.2 описаны в главе 10. Вот эти ключи:

-f

Разрешает доверенным пользователям переопределять адрес отправителя в исходящих сообщениях. По соображениям безопасности в некоторых системах данная возможность блокируется. Ключи -r и -s являются устаревшими альтернативными формами данного аргумента.

`-t`

Читает поля заголовка To:, Cc: и Bcc: со стандартного ввода. Используется для отправки файла, содержащего такие поля, либо при наборе тестового сообщения, как показано в главе 10.

`-bd`

Запускает sendmail в фоновом режиме для сбора поступающей почты. Используйте данный ключ в строке команды `sendmail` в загрузочном сценарии.

`-bt`

Используется для тестирования правил подстановки sendmail.

`-bi`

Инициализирует базу данных псевдонимов. Действие ключа идентично действию команды `newaliases`, описанной в главе 10.

`-q`

Устанавливает временной интервал обработки очереди почтовых сообщений. Используйте данный ключ в строке команды `sendmail` в загрузочном сценарии.

`-C`

Загружает альтернативный файл настройки sendmail. Используйте данный ключ для тестирования настроек перед копированием их в файл *sendmail.cf*.

`-v`

Позволяет наблюдать за выполнением команд SMTP в реальном времени.

`-bv`

Выполняет проверку обработки адресов, но не отправляет почтовые сообщения.

Если не принимать во внимание два аргумента (`-bd` и `-q`), используемые в строке команды `sendmail` в загрузочном сценарии с целью обработки входящей почты, наиболее распространенным применением аргументов `sendmail` является отладка. Перечисленные в приведенном выше списке ключи `-bt`, `-C`, `-bv`, `-v` и `-t` использовались в примерах отладки в главе 10. Прочие ключи отладки:

`-bp`

Выводит список сообщений, находящихся в очереди доставки. Действие ключа идентично действию команды `mailq`. Почтовые сообщения попадают в очередь, если не могут быть доставлены немедленно по причине того, что удаленный узел временно не способен принимать почту. sendmail периодически обрабатывает очередь исходя из временного интервала, определенного аргументом ключа `-q`, и пытается доставить сообщения из очереди. Рост очереди сообщений может значительно снижать производительность sendmail в случае неработоспособности удаленного узла. `mailq` сообщает, сколько сообщений находится в очереди, а также источник и пункт назначения для каждого сообщения.

Если очередь требует немедленной обработки, выполните `sendmail` с ключом `-q` без аргумента. Эта команда обрабатывает всю очередь. Некоторые вариации аргумента ключа `-q` позволяют избирательно обрабатывать очередь. Используйте `-qI`*queue-id* для обработки только тех сообщений, с которыми связан указанный идентификатор очереди; ключ `-qR`*recipient* для обработки сообщений, адресованных указанному получателю; или же ключ `-qS`*sender* для обработки писем, исходящих от указанного отправителя. Команда `mailq` отображает идентификатор очереди, адрес отправителя и адрес получателя для всех сообщений очереди.

`-o`

Устанавливает значение параметра для данного экземпляра sendmail, например `-oA/tmp/test-aliases`. Используйте данный аргумент, чтобы тестировать альтернативные значения параметров, не внося изменения в файл *sendmail.cf*. Ключ `-o` требует использования старого синтаксиса установки параметров sendmail. Альтернативой является аргумент `-O`, работающий с новым синтаксисом. Например, `-OAilasFile=/tmp/test-aliases`. См. раздел «Параметры sendmail» далее в этом приложении.

`-d`

Устанавливает уровень диагностики, выводимой при отладке кода sendmail. Может использоваться для отладки правил подстановки и для проверки параметров настройки (например, `sendmail -bt -d0.4`). Большая часть параметров отладки применима для отладки исходных текстов sendmail.

`-h`

Устанавливает значение счетчика, используемого для обнаружения зацикливания почты. По умолчанию имеет значение 30. Это хорошее рабочее значение. При диагностировании проблемы, связанной с зацикливанием почты, уменьшайте значение счетчика транзитных участков (например, `-h10`), чтобы сократить число повторений обработки сообщения системой. В ином случае не изменяйте данное значение.

`-bh`

Отображает состояние узла, если настройки sendmail предписывают наблюдать за этим состоянием. Вывод содержит имя каждого из удаленных узлов, которым передавалась почта, время последнего обновления состояния узла, а также результат последней попытки доставить почту этому узлу. Каталог файлов состояний узлов может стать очень большим. Используйте ключ `-bH` для очистки этого каталога.

Оставшиеся аргументы редко встречаются в командной строке:

`-B`

Указывает MIME-тип тела сообщения. Допустимые значения: 7BIT и 8BITMIME.

`-N`

Запрашивает передачу отправителю уведомления о состоянии доставки почты. Значение по умолчанию FAILURE, DELAY; отправитель получает

уведомление, если почта задерживается в очереди или не может быть доставлена. Прочие допустимые значения: NEVER (никакие уведомления не должны передаваться отправителю) и SUCCESS (запрос передачи уведомления об успешной доставке почты).

`-M`

Устанавливает значение макроопределения для данного экземпляра sendmail. Например, аргумент `-MMwrotethebook.com` устанавливает значение макроопределения M в *wrotethebook.com*.

`-p`

Определяет протокол отправки и передающий узел. Действие ключа эквивалентно установке значений внутренних макроопределений `s` и `r`. Если в системе несколько внешних почтовых протоколов (например, UUCP и SMTP), данный ключ предписывает системе использовать определенный протокол для данного сообщения.

`-R`

Устанавливает объем информации, возвращаемый отправителю в случае, когда сообщение не может быть доставлено. Допустимые значения: HDRS (только заголовки) и FULL (заголовки и тело сообщения).

`-U`

Указывает, что данное сообщение поступило напрямую через интерфейс пользователя, а не получено от удаленного обработчика почты.

`-V`

Вставляет идентификатор конверта в исходящие сообщения, посылаемые в случае невозможности доставить сообщение.

`-X`

Записывает все почтовые сообщения в указанный файл журнала. Это быстро приводит к непомерному росту файла.

`-n`

Отключает обработку псевдонимов и пересылку почты.

`-bm`

Предписывает sendmail доставить почту, что является поведением по умолчанию.

`-ba`

Читает строку заголовка From: в целях обнаружения адреса отправителя. Используются коды ответов из трех цифр, а строки ошибок завершаются символами <CRLF>. Аргумент вышел из употребления.

`-bs`

Предписывает sendmail использовать SMTP для входящей почты. Когда возможно, sendmail делает это и без аргумента `-bs`.

`-i`

Обычно признаком завершения сообщения SMTP является строка, содержащая только точку. Данный аргумент предписывает sendmail игнорировать точки в поступающих сообщениях.

`-m`

Посылает копию сообщения автору письма. Обычно этой цели служат заголовки сообщения CC: или BCC:, а не ключ `-m`.

`-bD`

Выполняет sendmail в приоритетном режиме. Демон остается прикреплен к управляющему терминалу.

`-F`

Указывает полное имя отправителя.

Таков полный список аргументов командной строки sendmail на момент написания этой книги. Некоторые из аргументов появились недавно, другие устарели в последних версиях sendmail. Чтобы определить, какие аргументы доступны в вашей системе, обращайтесь к страницам руководства по sendmail.

При выполнении команда `sendmail` читает свои настройки из файла *sendmail.cf*. Начальный файл *sendmail.cf* может быть создан при помощи макроопределений `m4`, поставляемых в дистрибутиве исходных текстов sendmail. Примеры решения этой задачи приводятся в главе 10. Полный перечень макроопределений `m4` из дистрибутива sendmail приводится в следующем разделе.

m4, макроопределения sendmail

Дистрибутив sendmail включает несколько примеров файла настройки. В главе 10 содержится пример создания файла настройки для системы Linux на основе файла *tcpproto.mc*. Файл прототипа – это файл макроопределений `m4`, результатом обработки которого является пригодный к использованию файл *sendmail.cf*. Файлы прототипов хранятся в каталоге *sendmail/cf/cf* дистрибутива sendmail. Используйте эти прототипы в качестве примеров разумных настроек sendmail.

Все файлы настройки sendmail состоят из следующих макроопределений `m4`:[1]

`divert`

Перенаправляет вывод процесса `m4`.

`dnl`

Удаляет все символы до следующей новой строки.

[1] Макроопределения `m4` записаны прописными буквами, а встроенные команды `m4` строчными.

VERSIONID

Определяет номер версии исходного файла *.mc*. Обычно для этой цели используются номера версий RCS или SCCS. Данная команда является необязательной.

OSTYPE

Указывает исходный файл m4, содержащий информацию, специфичную для используемой операционной системы. Данное макроопределение является обязательным.

DOMAIN

Указывает исходный файл m4, содержащий информацию настройки, специфичную для данного домена. Данное макроопределение является необязательным.

LOCAL_DOMAIN

Определяет псевдонимы данного сервера.

CANONIFY_DOMAIN

Указывает домены, которые должны подвергаться преобразованию в канонический формат, даже если канонизация отключена.

CANONIFY_DOMAIN_FILE

Указывает файл, содержащий список доменов, которые должны подвергаться преобразованию в канонический формат, даже если канонизация отключена.

GENERICS_DOMAIN

Определяет доменные имена, которые должны обрабатываться с использованием базы данных genericstable.

GENERICS_DOMAIN_FILE

Указывает файл, содержащий список доменных имен, которые должны обрабатываться с использованием базы данных genericstable.

LDAPROUTE_DOMAIN

Определяет домены, маршрутизация для которых должна осуществляться в соответствии с указаниями из каталога LDAP.

LDAPROUTE_DOMAIN_FILE

Указывает файл, содержащий список доменов, маршрутизация для которых должна осуществляться в соответствии с указаниями из каталога LDAP.

RELAY_DOMAIN

Определяет домены, для которых данный сервер должен осуществлять пересылку почты.

RELAY_DOMAIN_FILE

Указывает файл, содержащий список доменов, для которых данный сервер должен осуществлять пересылку почты.

VIRTUSER_DOMAIN

Определяет виртуальные домены, которые должны обрабатываться с использованием virtusertable.

VIRTUSER_DOMAIN_FILE

Указывает файл, содержащий список доменов, которые должны обрабатываться с использованием virtusertable.

FEATURE

Указывает исходный файл m4, содержащий определение необязательных возможностей sendmail. Для обработки исходного файла *.mc* данное макроопределение не требуется, однако во многих файлах настройки присутствует большое число записей FEATURE.

MASQUERADE_AS

Определяет доменное имя, используемое для сокрытия имени сервера в исходящей почте.

MASQUERADE_DOMAIN

Определяет домены, для которых следует выполнять сокрытие.

MASQUERADE_DOMAIN_FILE

Указывает файл, содержащий список доменов, для которых должно выполняться сокрытие.

MASQUERADE_EXCEPTION

Определяет узел, для которого не должно выполняться сокрытие, даже если оно выполняется для всего домена.

EXPOSED_USER

Определяет имена пользователей, исключающие сокрытие. Если адрес содержит одно из перечисленных имен, сокрытие узла в адресе не выполняется.

HACK

Указывает исходный файл m4, содержащий специфичную для сервера информацию настройки. Это временные настройки, исправляющие временные проблемы. Использование HACK не приветствуется.

SITE

Указывает узел UUCP с локальным подключением.

SITECONFIG

Указывает исходный файл, содержащий команды m4 SITE, определяющие площадки UUCP, подключенные к данному узлу. Формат команды: SITECONFIG (*файл*, *локальное имя узла*, *класс*); имена узлов UUCP добавляются из *файла* в *класс*.

UUCPSMTP

Отображает имя узла UUCP в имя узла Интернета.

`define`

Определяет локальное значение. Большинство «определений» содержится в исходных файлах `m4`, включаемых файлом *.mc*, а не в самом файле *.mc*. Определять можно значения для макроопределений и параметров *sendmail.cf* либо другие команды.

`undefine`

Снимает значение, установленное для параметра настройки.

`MAILER`

Указывает исходный файл `m4`, содержащий команды настройки, определяющие почтовую программу sendmail. В файле настройки должна присутствовать, по меньшей мере, одна команда MAILER. Обычно таких команд несколько.

`MAILER_DEFINITIONS`

Открывает раздел команд *sendmail.cf*, определяющих пользовательскую почтовую программу.

`MODIFY_MAILER_FLAGS`

Переопределяет флаги, определенные для почтовой программы.

`MAIL_FILTER`

Определяет почтовый фильтр.

`INPUT_MAIL_FILTER`

Определяет почтовый фильтр и переменные, необходимые для вызова фильтра.

`DAEMON_OPTIONS`

Определяет параметры работы демона sendmail.

`TRUST_AUTH_MECH`

Определяет список доверенных механизмов авторизации.

`LOCAL_RULE_`*n*

Открывает раздел кода, добавляемого к набору правил *n*, где *n* – число 0, 1, 2 или 3. Код, следующий за командой LOCAL_RULE_*n*, состоит из стандартных правил подстановки *sendmail.cf*.[1] Команда LOCAL_RULE_*n* используется редко.

`LOCAL_RULESETS`

Открывает раздел кода, определяющий пользовательский набор правил.

`LOCAL_USER`

Определяет имена пользователей, для которых не должна выполняться пересылка, даже когда она выполняется для локальной почты.

---

[1] Единственным исключением является макроопределение UUCPSMTP, которое может фигурировать в локальном правиле.

`LOCAL_NET_CONFIG`

Открывает раздел кода *sendmail.cf*, определяющего, как обрабатывается почта, предназначенная локальному узлу.

`LOCAL_CONFIG`

Открывает раздел кода, добавляемого в файл *sendmail.cf* после раздела локальной информации и перед правилами подстановки. Данный раздел кода содержит стандартные команды настройки *sendmail.cf*. Данное макроопределение редко используется.

Описанные выше встроенные команды `m4` – а именно те, что обозначены строчными буквами – делятся на два вида: управляющие выводом и устанавливающие значения макроопределений. Выводом управляют команды `dnl` и `divert`. Текст, следующий за командой `dnl`, не попадает в файл вывода. Эта команда используется в начале строки комментария. Команда `divert(-1)` направляет вывод в «никуда» и отмечает начало блока комментариев. Команда `divert(0)` направляет вывод в стандартный поток результатов `m4`. В дополнение к `-1` и `0` команда `divert` принимает девять других численных аргументов: значения от 1 до 9. Эти другие значения используются в исходном тексте макроопределений `m4` для записи данных в различные части файла *sendmail.cf*. В собственных настройках вы не будете использовать эти значения. Вместо этого вы будете использовать другие команды для записи данных в определенные части файла *sendmail.cf*.

Команды LOCAL_CONFIG, LOCAL_USER, LOCAL_RULESETS, MAILER_DEFINITION, LOCAL_NET_CONFIG и LOCAL_RULE позволяют записывать данные в различные части файла *sendmail.cf*, не задействуя напрямую различные значения `divert`. Команды вроде LOCAL_CONFIG и MAILER_DEFINITION отмечают начало фрагментов кода *sendmail.cf*, который должен включаться в ту или иную часть конечного файла. Эти команды позволяют изменять файл *sendmail.cf* всеми возможными способами.

Встроенные команды `m4`, `define` и `undefine`, устанавливают значения макроопределений. `define` инициализирует переменную значением, `undefine` сбрасывает переменную в значение по умолчанию. Команда `define` позволяет контролировать больше параметров настройки, чем любая другая, и, как следствие, большая часть этого приложения отведена под описание параметров `define`.

Около половины макроопределений `m4` действуют подобно команде `define` – просто устанавливают значения параметров. Примеры таких команд: MASQUERADE_AS, MASQUERADE_DOMAIN, RELAY_DOMAIN и VIRTUSER_DOMAIN_FILE.

TRUST_AUTH_MECH – хороший пример макроопределения, дополняющего `define`. Как вы увидите в разделе «define» данного приложения, параметр confAUTH_MECHANISMS может использоваться для определения доверенных механизмов аутентификации, о которых ваш сервер сообщает другим серверам. Макроопределение TRUST_AUTH_MECH имеет обратный смысл, оно указывает механизмы других серверов, с которыми готов работать ваш сервер. Для настройки TRUST_AUTH_MECHANISMS используется тот же

список ключевых слов, что и для настройки confAUTH_MECHANISMS (см. раздел «define»).

Имена макроопределений OSTYPE, DOMAIN, FEATURE, MAILER, HACK и SITECONFIG соответствуют именам подкаталогов каталога *cf*. Значения этих макроопределений соответствуют именам файлов из каталогов. К примеру, команда FEATURE(nouucp) предписывает `m4` загрузить файл *nouucp.m4* из каталога *ostype* и обработать его содержимое (код `m4`). Исходные файлы *.m4* для команд OSTYPE, DOMAIN, FEATURE и MAILER состоят преимущественно из команд `define` и FEATURE.

Еще два макроопределения, SITECONFIG и HACK, используются редко. SITECONFIG указывает исходный файл, содержащий макроопределения SITE, обозначающий системы UUCP, подключенные к локальному узлу. Создав самостоятельно файл, содержащий макроопределения SITE, вы должны подключить его при помощи команды SITECONFIG. Эти команды, наряду с UUCPSMTP, вышли из употребления и сохраняются только из соображений обратной совместимости.

Макроопределение HACK указывает исходный файл `m4`, содержащий временные, специфичные для данной системы поправки на проблемы sendmail. Администратор создает файл в каталоге *hack* и подключает его к настройкам при помощи команды HACK. Использование этого метода не приветствуется и обычно не требуется.

В следующем разделе содержится дополнительная информация о макроопределениях OSTYPE, DOMAIN, FEATURE и MAILER, а также сведения о различных командах, используемых для создания соответствующих этим макроопределениям исходных файлов `m4`. Пример создания частного варианта исходного файла макроопределений DOMAIN содержится в главе 10. Исходные файлы могут содержать любые макроопределения из уже упомянутых, а также дополнительные, речь о которых пойдет далее. Файл макроопределений настройки (*.mc*) также может содержать любую из описанных ниже команд. По сути дела, практически любое макроопределение может фигурировать в любом из файлов.

Чтобы некоторым образом упорядочить хаос, команды сгруппированы по файлам, в которых наиболее вероятно их присутствие. Подобная организация описаний существует и в документации из дистрибутива sendmail. Запомните, что на практике файлы могут иметь иную структуру. Мы начнем с изучения макроопределений `define` и FEATURE, базовых строительных блоков всех прочих файлов.

define

Синтаксис макроопределения `define`:

```
define(`parameter', `value')
```

Здесь *parameter* – ключевое слово, имя параметра настройки sendmail, а *value* – значение, назначаемое указанному параметру настройки. Параметр и значе-

ние обычно заключаются в одинарные кавычки, что позволяет избежать некорректного раскрытия макроопределения. Кавычки не одинаковы: открывающая кавычка представлена акцентом (`), а закрывающая кавычка апострофом (').

Многие из параметров настройки, установка которых может выполняться командой `define`, описаны ниже. Большая их часть соответствует параметрам, макроопределениям и классам sendmail. Имя параметра, макроопределения или класса sendmail, значение которого устанавливается параметром `m4`, содержится в описании параметра и заключено в квадратные скобки ([]). Имена макроопределений начинаются символом доллара ($j), имена классов начинаются символом доллара и знаком равенства ($=w), а параметры настройки обозначаются длинными именами (SingleThreadDelivery). Чтобы узнать больше об этих параметрах, обращайтесь к описаниям макроопределений, параметров и классов sendmail, представленным далее в данном приложении.

Поскольку многие параметры `define` эквивалентны параметрам настройки, макроопределениям и классам, команда:

```
define(`confDOMAIN_NAME', `rodent.wrotethebook.com')
```

в исходном файле `m4` имеет то же действие, что строка:

```
Djrodent.wrotethebook.com
```

расположенная непосредственно в файле *sendmail.cf*. Если вы компилируете и устанавливаете новую версию sendmail, создайте настройки при помощи `m4` – используйте макроопределение `m4 define` для указания значений макроопределений, классов и параметров настройки.

Перечень параметров `define` имеет приличную длину. При этом большинство параметров имеют разумные значения по умолчанию, и нет необходимости явным образом устанавливать их значения в исходном файле `m4`. Значение по умолчанию каждого из параметров содержится в перечне (если оно есть).

`confMAILER_NAME`

По умолчанию – MAILER-DAEMON. Имя отправителя, используемое в сообщениях об ошибках. [$n]

`confDOMAIN_NAME`

Полное имя узла. [$j]

`confCF_VERSION`

Номер версии файла настройки. [$Z]

`confFROM_HEADER`

По умолчанию – `$?x$x <$g>$|$g$.`. Формат заголовка From:.

`confRECEIVED_HEADER`

По умолчанию – `$?sfrom $s $.$?_($?s$|from $.$_) $.by $j ($v/$Z)$?r with $r$. id $i$?u for $u$.; $b`. Формат заголовка Received:.

confCW_FILE

По умолчанию – */etc/sendmail.cw*. Файл псевдонимов локального узла. [$=w]

confCT_FILE

По умолчанию – */etc/sendmail.ct*. Файл имен доверенных пользователей. [$=t]

confTRUSTED_USERS

Имена доверенных пользователей в дополнение к *root*, *uucp* и *daemon*.

confSMTP_MAILER

По умолчанию – esmtp. Почтовая программа, используемая для соединений SMTP; одно из значений smtp, smtp8 или esmtp.

confUUCP_MAILER

По умолчанию – uucp-old. Почтовая программа по умолчанию для UUCP.

confLOCAL_MAILER

По умолчанию – local. Почтовая программа, используемая для локальных соединений.

confRELAY_MAILER

По умолчанию – relay. Имя почтовой программы, по умолчанию используемой для пересылки.

confSEVEN_BIT_INPUT

По умолчанию – False. Семибитный ввод. [SevenBitInput]

confEIGHT_BIT_HANDLING

По умолчанию – pass8. Указывает способ обработки 8-битных данных. [EightBitMode]

confALIAS_WAIT

По умолчанию – 10m. Период ожидания завершения пересборки файла псевдонимов. [AliasWait]

confMIN_FREE_BLOCKS

По умолчанию – 100. Минимальное число свободных блоков в файловой системе почтовой очереди, позволяющее принимать почту SMTP. [MinFreeBlocks]

confMAX_MESSAGE_SIZE

По умолчанию без ограничений. Максимальный размер сообщения. [MaxMessageSize]

confBLANK_SUB

Символ, заменяющий немаскированные пробелы в адресах электронной почты. [BlankSub]

confCON_EXPENSIVE

По умолчанию – False. Предписывает системе задерживать почту, адресованную почтовым программам с установленным флагом e, до следующей обработки очереди. [HoldExpensive]

confCHECKPOINT_INTERVAL

По умолчанию – 10. Предписывает системе создавать контрольную точку для файлов очереди после обработки указанного числа элементов очереди. [CheckpointInterval]

confDELIVERY_MODE

По умолчанию – фоновый (режим). Режим доставки по умолчанию. [DeliveryMode]

confAUTO_REBUILD

По умолчанию – False. Автоматически выполнять пересборку файла псевдонимов. [AutoRebuildAliases]

confERROR_MODE

По умолчанию – печать. Определяет способ обработки ошибок. [ErrorMode]

confERROR_MESSAGE

Указывает файл, содержащий сообщение, предваряющее сообщения об ошибках. [ErrorHeader]

confSAVE_FROM_LINES

Запрещает системе удалять строки Unix From:. Если параметр не установлен, строки удаляются. [SaveFromLine]

confTEMP_FILE_MODE

По умолчанию – 0600. Файловые права доступа для временных файлов. [TempFileMode]

confMATCH_GECOS

Предписывает системе сопоставлять имя пользователя из адреса электронной почты с полем GECOS. Сопоставление не выполняется, если параметр не установлен. [MatchGECOS]

confMAX_HOP

По умолчанию – 25. Счетчик, используемый для обнаружения почтовых петель. [MaxHopCount]

confIGNORE_DOTS

По умолчанию – False. Предписывает системе игнорировать точки в поступающих сообщениях. [IgnoreDots]

confBIND_OPTS

По умолчанию не определен. Устанавливает параметры работы клиента DNS. [ResolverOptions]

confMIME_FORMAT_ERRORS

По умолчанию – True. Предписывает системе посылать MIME-кодированные сообщения об ошибках. [SendMimeErrors]

confFORWARD_PATH

По умолчанию – *$z/.forward.$w:$z/.forward*. Каталоги для поиска файлов *.forward*. [ForwardPath]

confMCI_CACHE_SIZE

По умолчанию – 2. Число кэшируемых открытых соединений. [ConnectionCacheSize]

confMCI_CACHE_TIMEOUT

По умолчанию – 5m. Длительность кэширования пассивного открытого соединения. [ConnectionCacheTimeout]

confHOST_STATUS_DIRECTORY

Каталог, в котором сохраняется состояние узла. [HostStatusDirectory]

confUSE_ERRORS_TO

По умолчанию – False. Выполнять доставку ошибок с использованием заголовка Errors-To:. [UseErrorsTo]

confLOG_LEVEL

По умолчанию – 9. Уровень подробностей для файла журнала. [LogLevel]

confME_TOO

По умолчанию – False. Посылать копию сообщения отправителю. [MeToo]

confCHECK_ALIASES

По умолчанию – False. Искать все псевдонимы в процессе сборки файла псевдонимов. [CheckAliases]

confOLD_STYLE_HEADERS

По умолчанию – True. Считать, что заголовки без специальных символов записаны в старом стиле. [OldStyleHeaders]

confDAEMON_OPTIONS

Параметры демона SMTP. [DaemonPortOptions]

confPRIVACY_FLAGS

По умолчанию – authwarnings. Данные флаги ограничивают использование некоторых почтовых команд. [PrivacyOptions]

confCOPY_ERRORS_TO

Адрес приема сообщений об ошибках. [PostmasterCopy]

confQUEUE_FACTOR

По умолчанию – 600000. Используется для определения момента, когда загруженная система должна помещать сообщения в очередь, а не пытаться их доставить. [QueueFactor]

confDONT_PRUNE_ROUTES

По умолчанию – False. Не усекать маршруты до минимально возможной длины. [DontPruneRoutes]

confSAFE_QUEUE

Создавать файл в очереди, затем выполнять доставку, что не выполняется, если не установлен данный параметр. [SuperSafe]

confTO_INITIAL

По умолчанию – 5m. Максимальное время ожидания ответа на запрос соединения. [Timeout. initial]

confTO_CONNECT

По умолчанию – 0. Максимальное время ожидания завершения операции создания соединения. [Timeout.connect]

confTO_ICONNECT

Максимальное время ожидания завершения самой первой попытки подключения к узлу. [Timeout.iconnect]

confTO_HELO

По умолчанию – 5m. Максимальное время ожидания ответа HELO или EHLO. [Timeout. helo]

confTO_MAIL

По умолчанию – 10m. Максимальное время ожидания ответа на команду MAIL. [Timeout. mail]

confTO_RCPT

По умолчанию – 1h. Максимальное время ожидания ответа на команду RCPT. [Timeout. rcpt]

confTO_DATAINIT

По умолчанию – 5m. Максимальное время ожидания ответа на команду DATA. [Timeout. datainit]

confTO_DATABLOCK

По умолчанию – 1h. Максимальное время ожидания поступления блока на этапе DATA. [Timeout. datablock]

confTO_DATAFINAL

По умолчанию – 1h. Максимальное время ожидания ответа на завершающую точку («.»). [Timeout.datafinal]

confTO_RSET

По умолчанию – 5m. Максимальное время ожидания ответа на команду RSET. [Timeout. rset]

confTO_QUIT

По умолчанию – 2m. Максимальное время ожидания ответа на команду QUIT. [Timeout. quit]

confTO_MISC

По умолчанию – 2m. Максимальное время ожидания ответов на прочие команды SMTP. [Timeout.misc]

confTO_COMMAND

По умолчанию – 1h. Максимальное время ожидания команды. [Timeout.command]

confTO_IDENT

По умолчанию – 30s. Максимальное время ожидания ответа на запрос IDENT. [Timeout. ident]

confTO_FILEOPEN

По умолчанию – 60s. Максимальное время ожидания открытия файла. [Timeout.fileopen]

confTO_QUEUERETURN

По умолчанию – 5d. Время удаления сообщения из очереди из-за невозможности доставки. [Timeout.queuereturn]

confTO_QUEUERETURN_NORMAL

Интервал ожидания «undeliverable» для сообщений с нормальным приоритетом. [Timeout.queuereturn.normal]

confTO_QUEUERETURN_URGENT

Интервал ожидания «undeliverable» для сообщений срочного приоритета. [Timeout.queuereturn. urgent]

confTO_QUEUERETURN_NONURGENT

Интервал ожидания «undeliverable» для сообщений с низким приоритетом. [Timeout.queuereturn.non- urgent]

confTO_QUEUEWARN

По умолчанию – 4h. Время создания предупреждения «still queued» (все еще находится в очереди) для сообщения. [Timeout.queuewarn]

confTO_QUEUEWARN_NORMAL

Время отправки предупреждения «still queued» для сообщений с нормальным приоритетом. [Timeout. queuewarn.normal]

confTO_QUEUEWARN_URGENT

Время отправки предупреждения «still queued» для сообщений срочного приоритета. [Timeout. queuewarn.urgent]

confTO_QUEUEWARN_NONURGENT

Время отправки предупреждения «still queued» для сообщений с низким приоритетом. [Timeout. queuewarn.non-urgent]

confTO_HOSTSTATUS

По умолчанию – 30m. Время устаревания информации о состоянии узла. [Timeout.hoststatus]

confTIME_ZONE

По умолчанию – USE_SYSTEM. Устанавливает часовой пояс из системной переменной (USE_SYSTEM) или переменной TZ (USE_TZ). [TimeZoneSpec]

confDEF_USER_ID

По умолчанию – 1:1. Значения по умолчанию идентификатора пользователя и группы. [DefaultUser]

confUSERDB_SPEC

Путь к базе данных пользователей. [UserDatabaseSpec]

confFALLBACK_MX

Резервный узел MX. [FallbackMXhost]

confTRY_NULL_MX_LIST

По умолчанию – False. Предписывает системе подключаться напрямую к удаленному узлу, если MX указывает на локальный узел. [TryNullMXList]

confQUEUE_LA

По умолчанию – 8. Почта записывается сразу в очередь по достижении данного значения средней загрузки системы. [QueueLA]

confREFUSE_LA

По умолчанию – 12. Входящие SMTP-соединения перестают приниматься по достижении данного значения средней загрузки системы. [RefuseLA]

confMAX_DAEMON_CHILDREN

Если параметр установлен, соединения перестают устанавливаться по достижении указанного числа порожденных процессов. [MaxDaemonChildren]

confCONNECTION_RATE_THROTTLE

Если параметр установлен, указывает максимальное допустимое число соединений за одну секунду. [ConnectionRateThrottle]

confWORK_RECIPIENT_FACTOR

По умолчанию – 30000. Коэффициент, используемый для снижения приоритетов заданий каждого из дополнительных адресатов сообщения. [RecipientFactor]

confSEPARATE_PROC

По умолчанию – False. Выполнять доставку сообщений в самостоятельных процессах. [ForkEachJob]

confWORK_CLASS_FACTOR

По умолчанию – 1800. Коэффициент, используемый для выбора высокоприоритетных заданий. [ClassFactor]

confWORK_TIME_FACTOR

По умолчанию – 90000. Коэффициент, используемый для снижений приоритета задания в каждой из попыток доставки. [RetryFactor]

`confQUEUE_SORT_ORDER`

По умолчанию – Priority. Сортирует очередь по приоритетам (Priority) или узлам (Host). [QueueSortOrder]

`confMIN_QUEUE_AGE`

По умолчанию – 0. Минимальное время хранения задания в очереди. [MinQueueAge]

`confDEF_CHAR_SET`

По умолчанию – unknown-8bit. Умолчание набора символов для непомеченных 8-битных данных MIME. [DefaultCharSet]

`confSERVICE_SWITCH_FILE`

По умолчанию – */etc/service.switch*. Полное имя файла коммутации служб. [ServiceSwitchFile]

`confHOSTS_FILE`

По умолчанию – */etc/hosts*. Полное имя файла таблицы узлов. [HostsFile]

`confDIAL_DELAY`

По умолчанию – 0s. Задержка повторных попыток установления соединений «по необходимости». 0s означает «не повторять попытки» [DialDelay]

`confNO_RCPT_ACTION`

По умолчанию отсутствует (none). Обработка почты без заголовков получателей: ничего не делать (none); добавить заголовок To: (add-to); добавить заголовок Apparently-To: (add-apparently-to); добавить заголовок Bcc: (add-bcc); добавить заголовок «To: undisclosed-recipients» header (add-to-undisclosed). [NoRecipientAction]

`confSAFE_FILE_ENV`

По умолчанию не определен. Выполнять `chroot()` в этот каталог перед записью файлов. [SafeFileEnvironment]

`confCOLON_OK_IN_ADDR`

По умолчанию – True. Считать двоеточия в адресе обычными символами. [ColonOkInAddr]

`confMAX_QUEUE_RUN_SIZE`

По умолчанию – 0. Ограничивает число сообщений очереди, обрабатываемых за один проход. Значение 0 снимает ограничения. [MaxQueueRunSize]

`confDONT_EXPAND_CNAMES`

По умолчанию – False. Запрещает преобразования псевдонимов в канонические имена. Значение False предписывает преобразование. [DontExpandCnames]

`confFROM_LINE`

По умолчанию – `From $g $d`. Формат строки Unix From:. [UnixFromLine]

confOPERATORS

По умолчанию – `. :%@! ^/[ ]+`. Символы-операторы адресов. [OperatorChars]

confSMTP_LOGIN_MSG

По умолчанию – `$j sendmail $v/$Z; $b`. Сообщение приветствия SMTP. [SmtpGreetingMessage]

confDONT_INIT_GROUPS

По умолчанию – False. Значение True отключает подпрограмму initgroups(3). False предписывает использовать подпрограмму initgroups(3). [DontInitGroups]

confUNSAFE_GROUP_WRITES

По умолчанию – False. Если True – ссылаться на программы или файлы из доступных для записи группой файлов *:include:* и *.forward* запрещено. [UnsafeGroupWrites]

confDOUBLE_BOUNCE_ADDRESS

По умолчанию – postmaster. Если возникает ошибка в процессе отправки сообщения об ошибке, второе сообщение об ошибке посылается на этот адрес. [DoubleBounceAddress]

confRUN_AS_USER

По умолчанию не определен. Работать с полномочиями указанного пользователя при чтении и доставке почты. [RunAsUser]

confSINGLE_THREAD_DELIVERY

По умолчанию – False. Принудительная доставка почты в одном потоке при установленном параметре HostStatusDirectory. [SingleThreadDelivery]

confALLOW_BOGUS_HELO

Указывает обычно недопустимые специальные символы, которые будут разрешены в именах узлов DNS – в командах HELO и EHLO. [AllowBogusHELO]

confAUTH_MECHANISMS

Содержит список механизмов аутентификации, предоставляемых данным сервером. Элементы списка разделяются пробелами. Допустимые значения: GSSAPI, KERBEROS_V4, DIGEST-MD5 и CRAM-MD5. [AuthMechanisms]

confAUTH_OPTIONS

Если параметр имеет значение A, аргумент AUTH= добавляется к заголовку MAIL FROM, только если аутенификация успешно пройдена. [AuthOptions]

confCACERT

Указывает файл, содержащий криптографический сертификат, полученный от владельца сертификатов. [CACERTFile]

confCACERT_PATH

Определяет каталог, в котором хранятся криптографические сертификаты. [CACERTPath]

confCLIENT_CERT

Указывает файл криптографического сертификата, используемый sendmail при работе в качестве клиента. [ClientCertFile]

confCLIENT_KEY

Указывает файл, содержащий закрытый ключ сертификата, используемого sendmail при работе в качестве клиента. [ClientKeyFile]

confCLIENT_OPTIONS

Определяет параметры порта для исходящих клиентских SMTP-соединений. [ClientPortOptions]

confCONNECT_ONLY_TO

Ограничивает подключения. Используется разработчиками sendmail в целях тестирования. Не используется в рабочих системах. [ConnectOnlyTo]

confCONTROL_SOCKET_NAME

Определяет сокет управления демоном sendmail. [ControlSocketName]

confCR_FILE

Указывает файл, содержащий список узлов, для которых данный сервер выполняет пересылку почты. По умолчанию – */etc/mail/relay-domains*. [$=R]

confDEAD_LETTER_DROP

Указывает файл, в котором сохраняются сообщения, не подлежащие возврату отправителю или не доставленные пользователю postmaster. [DeadLetterDrop]

confDEF_AUTH_INFO

Указывает файл, содержащий информацию идентификации для исходящих соединений. [DefaultAuthInfo]

confDF_BUFFER_SIZE

Определяет максимальный объем используемой буферной памяти, после превышения которого начинает использоваться дисковое пространство. [DataFileBufferSize]

confDH_PARAMETERS

Указывает файл, содержащий параметры DH для алгоритма цифровых подписей DSA/DH. [DHParameters]

confDONT_BLAME_SENDMAIL

Предписывает sendmail не выполнять определенные проверки безопасности для файлов. По умолчанию выполняются все проверки. Данный параметр снижает защищенность вашего сервера. Полный перечень зна-

чений для данного параметра приводится ниже, в описании параметра DontBlameSendmail. [DontBlameSendmail]

confDONT_PROBE_INTERFACES

По умолчанию False. Запрещает sendmail автоматически принимать адреса сетевых интерфейсов сервера в качестве корректных. [DontProbeInterface]

confEBINDIR

Определяет каталог, где хранятся исполняемые файлы для FEATURE(`local_lmtp') и FEATURE(`smrsh'). По умолчанию используется каталог */usr/libexec*.

confLDAP_DEFAULT_SPEC

Определяет значения по умолчанию для баз данных LDAP, используемые всегда, за исключением случая, когда параметры переопределяются командой K для отдельной карты. [LDAPDefaultSpec]

confMAX_ALIAS_RECURSION

Псевдонимы могут ссылаться на другие псевдонимы. Данный параметр устанавливает максимальный уровень вложенности для таких ссылок. По умолчанию – 10. [MaxAliasRecursion]

confMAX_HEADERS_LENGTH

Определяет максимальную длину всех заголовков в байтах. [MaxHeadersLength]

confMAX_MIME_HEADER_LENGTH

Определяет максимальную длину заголовков MIME. [MaxMimeHeaderLength]

confMAX_RCPTS_PER_MESSAGE

Определяет максимально допустимое число получателей сообщения. [MaxRecipientsPerMessage]

confMUST_QUOTE_CHARS

Добавляет символы в список символов, подлежащих маскировке при использовании в полном имени пользователя ($x). Символы @,;:\()[] маскируются всегда. По умолчанию в список добавляются символы . и ‘. [MustQuoteChars]

confPID_FILE

Указывает полное имя файла PID. [PidFile]

confPROCESS_TITLE_PREFIX

Содержит строку, используемую в данной системе в качестве префикса имени процесса в выводе команды ps. [ProcessTitlePrefix]

confRAND_FILE

Указывает файл, содержащий случайные данные, необходимые для работы STARTTLS, если при компиляции sendmail не был установлен флаг HASURANDOM. [RandFile]

confREJECT_MSG

Определяет сообщение, отображаемое при возврате письма по причинам, связанным с базой данных управления доступом. По умолчанию – «550 Access denied».

confRRT_IMPLIES_DSN

Значение True предписывает sendmail считать заголовок Return-Receipt-To: запросом на уведомление о состоянии доставки (DSN). По умолчанию – False. [RrtImpliesDsn]

confSERVER_CERT

Указывает файл, содержащий криптографический сертификат, используемый системой при работе в качестве сервера. [ServerCertFile]

confSERVER_KEY

Указывает файл, содержащий закрытый ключ криптографического сертификата, используемого системой при работе в качестве сервера. [ServerKeyFile]

confSINGLE_LINE_FROM_HEADER

Значение True сворачивает многострочный заголовок From: в одну строку. По умолчанию – False. [SingleLineFromHeader]

confTO_RESOLVER_RETRANS

Определяет интервал ожидания для повторной передачи всех запросов клиента DNS в секундах. [Timeout. resolver.retrans]

confTO_RESOLVER_RETRANS_FIRST

Определяет интервал ожидания для повторной передачи запросов клиента DNS. Действие параметра распространяется на первую попытку доставки сообщения. [Timeout.resolver.retrans.first]

confTO_RESOLVER_RETRANS_NORMAL

Определяет интервал ожидания для повторной передачи запросов клиента DNS. Действие параметра распространяется на все попытки доставки сообщения, кроме первой. [Timeout.resolver.retrans.normal]

confTO_RESOLVER_RETRY

Определяет число повторных попыток для запросов клиента DNS. [Timeout.resolver.retry]

confTO_RESOLVER_RETRY_FIRST

Определяет число повторных попыток для запросов клиента DNS. Действие параметра распространяется на первую попытку доставки сообщения. [Timeout.resolver.retry.first]

confTO_RESOLVER_RETRY_NORMAL

Определяет число повторных попыток для запросов клиента DNS. Действие параметра распространяется на все попытки доставки сообщения, кроме первой. [Timeout.resolver.retry.normal]

confTRUSTED_USER

Определяет пользователя, управляющего демоном sendmail и владеющего файлами, созданными sendmail. Не путайте данный параметр с confTRUSTED_USERS. [TrustedUser]

confXF_BUFFER_SIZE

Определяет максимальный объем буферной памяти, используемой под файл расшифровки ошибок доставки, после превышения которого файл должен быть записан на диск. По умолчанию – 4096 байт. [XScriptFileBufferSize]

Макроопределения `define` – наиболее распространенные макроопределения в исходных файлах `m4`. Следующим по популярности является макроопределение FEATURE.

FEATURE

Макроопределение FEATURE обрабатывает исходный код `m4` из каталога *cf/feature*. Исходные файлы в этом каталоге определяют дополнительные возможности sendmail, которые администратор может пожелать включить в настройки. Синтаксис макроопределения FEATURE:

```
FEATURE(name, [argument])
```

Исходный файл FEATURE может вызываться как с необязательным аргументом, так и без него. Если аргумент передается указанному исходному файлу, он используется в создании кода для файла *sendmail.cf*. Пример:

```
FEATURE(`mailertable', `hash /etc/mail/mailertable')
```

Генерирует код для доступа к mailertable и указывает, что таблица является базой данных типа hash и хранится в файле */etc/mail/mailertable*.

Все возможности, доступные в sendmail V8, перечислены в табл. Е.3. Для каждой возможности приводится имя и назначение.

Таблица Е.3. Возможности sendmail

Имя	Назначение
use_cw_file	Загружать $=w из файла */etc/mail/local-host-names*
use_ct_file	Загружать $=t из файла */etc/mail/trusted-users*
relay_based_on_MX	Осуществлять пересылку почты для любой системы, MX-запись которой указывает на этот сервер
relay_entire_domain	Осуществлять пересылку почты для любого узла домена
relay_hosts_only	Осуществлять пересылку почты только для узлов, упомянутых в базе данных управления доступом
relay_local_from	Осуществлять пересылку почты, если источником является локальный узел

Таблица E.3 (продолжение)

Имя	Назначение
relay_mail_from	Осуществлять пересылку почты, если отправитель упомянут как RELAY в базе данных управления доступом
promiscuous_relay	Осуществлять пересылку почты от любой системы в любую систему
rbl	Вышедшая из употребления возможность Realtime Blackhole List, заменена dnsbl
dnsbl	Отвергать почту от узлов, перечисленных в DNS-списке блокировки. Заменяет rbl
blacklist_recipients	Фильтровать поступающие сообщения, исходя из значений, установленных в базе данных управления доступом
delay_checks	Откладывать вызов наборов правил check_mail и check_relay до вызова check_rcpt
loose_relay_check	Отключить проверку допустимости для адресов, использующих символ % для разделения имени пользователя и имени компьютера
redirect	Поддержка псевдодомена .REDIRECT
no_default_msa	Разрешить переопределение настроек по умолчанию агента передачи сообщений (Message Submission Agent) посредством макроопределения DAEMON_OPTIONS
nouucp	Не включать код обработки адресов UUCP
nocanonify	Запретить преобразование имен синтаксиса $[*name*$]
stickyhost	Интерпретировать формат «user» иначе, чем «user@local.host»[1]
mailertable	Для маршрутизации почты используется таблица почтовых программ
domaintable	Для отображения доменных имен используется таблица доменов
access_db	Для управления пересылкой используется база данных управления доступом
bitdomain	Использовать таблицу для отображения узлов bitnet в адреса Интернета
uucpdomain	Использовать таблицу для отображения узлов UUCP в адреса Интернета
accept_unqualified_senders	Разрешить получение почты с адресов, не содержащих корректного имени узла
accept_unresolvable_domains	Принимать почту от узлов, не известных DNS

[1] «stickyhost» описывается в разделе «DOMAIN» далее в приложении.

Имя	Назначение
always_add_domain	Добавлять имя локального узла во все сообщения, для которых выполняется локальная доставка
allmasquerade	Скрывать также адрес получателя
limited_masquerade	Скрывать только узлы из $=M
masquerade_entire_domain	Выполнять сокрытие для всех узлов из доменов
masquerade_envelope	Скрывать также адреса на конверте. По умолчанию сокрытие выполняется только для адресов заголовка
genericstable	Использовать таблицу для переписывания локальных адресов
generics_entire_domain	Выполнять отображение доменных имен из класса G посредством genericstable
virtusertable	Отображать имена виртуальных доменов в реальные адреса электронной почты
virtuser_entire_domain	Выполнять отображение доменных имен посредством virtusertable
ldap_routing	Включить маршрутизацию почты на основе каталогов LDAP
nodns	Не включать поддержку DNS
nullclient	Пересылать всю почту на центральный сервер
local_lmtp	Использовать mail.local с поддержкой LMTP
local_procmail	Использовать procmail для локальной доставки
bestmx_is_local	Принимать в качестве локальной почту, исходящую от узла, который считает данную систему своим MX-сервером
smrsh	Использовать smrsh в качестве почтовой программы prog

Возможности use_cw_file и use_ct_file эквивалентны командам `Fw/etc/sendmail.cw` и `Fw/etc/sendmail.ct` в файле *sendmail.cf*. Описание псевдонимов узла (`$=w`) и доверенных пользователей (`$=t`) содержится в главе 10.

Код псевдодомена .REDIRECT возвращает отправителю сообщение об ошибке с рекомендацией воспользоваться новым адресом получателя. Данный механизм используется для обработки почты пользователей, которые уже не читают почту на вашем сервере, но по-прежнему получают почту, отправленную на очень старый адрес. Включите данный механизм при помощи команды FEATURE(redirect), а затем добавьте псевдонимы для всех устаревших адресов в формате:

```
старый-адрес новый-адрес.REDIRECT
```

Предположим, что Эдвард Уинслоу более не является пользователем *crab.wrotethebook.com*. Следовательно, его старое имя пользователя, *ed*, уже не может

использоваться для приема почты. Его новый почтовый адрес – *WinslowE @industry.com*. Мы добавляем следующий псевдоним в файл */etc/aliases*:

```
ed    WinslowE@industry.com.REDIRECT
```

Теперь, если почта поступает на учетную запись *ed* узла *crab*, отправителю посылается следующее сообщение об ошибке:

```
551 User not local; please try <WinslowE@industry.com>
```

Некоторые макроопределения FEATURE в действительности исключают определенные возможности из файла *sendmail.cf*, а не добавляют их. Так, nouucp удаляет код обработки адресов UUCP для систем, которые не имеют доступа в сети UUCP, а nodns удаляет код поиска в DNS для систем, не работающих с DNS. nocanonify блокирует конструкцию $[*name*]$, позволяющую преобразовывать псевдонимы и IP-адреса в канонические имена. Наконец, возможность nullclient удаляет из настроек все возможности, за исключением возможности пересылать почту единственному почтовому серверу по локальному каналу SMTP. Имя почтового сервера фигурирует в качестве аргумента строки команды nullclient. Например, `FEATURE(nullclient, ms.big.com)` пересылает всю почту *ms.big.com*, не выполняя локальной обработки.

Отдельные возможности связаны с пересылкой почты и сокрытием элементов адресов. В частности, stickyhost, relay_based_on_MX, allmasquerade, limited_masquerade и masquerade_ entire_domain. Все эти возможности рассмотрены в разделе «DOMAIN» далее по тексту.

Отдельные возможности связаны с базами данных, позволяющими выполнять специальную обработку адресов. Все эти возможности принимают необязательный аргумент, определяющий базу данных. (См. команду `mailertable` в начале данного раздела – пример определения базы данных с необязательным аргументом.) Если необязательный аргумент отсутствует, умолчанием описания базы данных всегда является `hash -o /etc/mail/`*filename*, где *filename* идентично названию возможности. К примеру, умолчанием для `mailertable` будет определение `hash -o /etc/mail/mailertable`. Возможности баз данных:

`mailertable`

Отображает имена узлов и доменов в конкретные пары почтовая_программа:узел. Если имя узла или домена в адресе получателя совпадает с одним из ключей базы данных `mailertable`, она возвращает имя почтовой программы и узла для этого адреса. Формат записей `mailertable`:

доменное-имя почтовая программа:*узел*

где *доменное-имя* – полное имя узла (узел и домен) либо имя домена. Если используется доменное имя, оно должно начинаться с точки (.) и будет соответствовать всем узлам этого домена.

`domaintable`

Преобразует старое доменное имя в новое доменное имя. Старое имя является ключом, а новое имя – значением, возвращаемым по ключу.

`bitdomain`

Преобразует имя узла Bitnet в имя узла Интернета. Имя Bitnet является ключом, а Интернет-имя – значением, возвращаемым по ключу. Программа bitdomain, поставляемая в составе sendmail, может использоваться для создания этой базы данных.

`uucpdomain`

Преобразует имя UUCP в имя узла Интернета. Имя узла UUCP является ключом, а имя узла Интернета – значением, возвращаемым по ключу.

`genericstable`

Преобразует адрес электронной почты отправителя. Ключом базы данных является имя пользоваля либо полный адрес электронной почты (имя пользователя и имя узла). Найденное в базе данных значение представляет собой новый адрес электронной почты. `genericstable` часто используется для обработки тех же адресов, что подвергаются сокрытию, и поэтому возможности sendmail, затрагивающие сокрытие, и возможности, затрагивающие преобразования `genericstable`, получают одинаковые значения. Если вы используете `genericstable` и сокрытие, установите GENERICS_DOMAIN и GENERICS_DOMAIN_FILE в те же значения, что MASQUERADE_DOMAIN и MASQUERADE_DOMAIN_FILE.

`virtusertable`

Реализует обработку псевдонимов для адресов электронной почты поступающих сообщений. По сути дела, это расширенная база данных псевдонимов для обработки адресов, не являющихся локальными для данного узла. Ключом базы данных является полный адрес электронной почты или доменное имя. Значение, найденное в базе данных, – адрес получателя, на который выполняется доставка почты. Если в качестве ключа выступает доменное имя, оно должно начинаться с символа @. Почта, адресованная любому пользователю из указанного домена, отправляется получателю, определенному базой данных virtusertable. Любое имя узла, используемое в качестве ключа базы данных virtusertable, должно быть также определено в классе w или классе {VirtHost}. Имя узла можно добавить в класс w при помощи макроопределения LOCAL_DOMAIN. Имена узлов можно добавлять в класс {VirtHost} при помощи макроопределения VIRTUSER_DOMAIN. Чтобы загрузить класс {VirtHost} из файла, воспользуйтесь макроопределением VIRTUSER_DOMAIN_FILE.

Некоторые возможности важны для борьбы со спамом, поскольку помогают контролировать, какую почту сервер доставляет или передает другим серверам для доставки; а именно accept_unqualified_senders, accept_unresolvable_domains, access_db, blacklist_recipients и dnsbl. В базе данных управления доступом перечислены источники электронной почты и способы обработки почты из этих источников. dnsbl использует специальную базу данных DNS для блокировки почты из определенных источников. blacklist_recipients расширяет механизмы access_db и dnsbl, затрагивая не только источники, но и пункты назначения электронной почты. Две из этих воз-

можностей, accept_unqualified_senders и accept_unresolvable_domains, ослабляют управление пересылкой, разрешая пересылку для узлов или доменов, которые не могут быть найдены в базе данных DNS. Будьте осторожны в применении этих возможностей.

Из оставшихся команд FEATURE две связаны с доменами. Макроопределение always_add_domain предписывает sendmail добавлять имя локального узла во все сообщения, для которых осуществляется локальная доставка, даже в сообщения, содержащие – в нормальной ситуации – только имя пользователя в поле адреса. Возможность bestmx_is_local позволяет принимать почту, адресованную узлу, считающему локальную систему предпочтительным MX-сервером, в качестве локальной. Если эта возможность не используется, почта, предназначенная удаленному узлу, передается напрямую удаленному узлу, даже если его MX-запись указывает локальную систему в качестве предпочтительного MX-сервера. Возможность bestmx_is_local не следует использовать, если вы применяете маску в записи MX для своего домена.

Последние две возможности используются для выбора необязательных программ для доставки почты local и prog. local_procmail выбирает `procmail` в качестве программы локальной доставки. Укажите полное имя procmail в качестве аргумента команды FEATURE. Возможность `smrsh` выбирает ограниченный интерпретатор sendmail (sendmail Restricted SHell, `smrsh`) в качестве почтовой программы prog. В плане безопасности `smrsh` совершеннее, чем */bin/sh*, используемый обычно в качестве почтовой программы prog. Укажите полное имя `smrsh` в качестве аргумента команды FEATURE.

Команда FEATURE, описанная в данном разделе, и макроопределения `define`, о которых мы говорили выше, используются для создания исходных файлов `m4`. В последующих разделах описано назначение и структура файлов OSTYPE, DOMAIN и MAILER.

OSTYPE

Исходный файл макроопределения OSTYPE содержит параметры, специфичные для операционной системы. Для многих операционных систем существуют готовые определения. Полный список таких систем доступен в каталоге *sendmail/cf/ostype*.

Исходные файлы OSTYPE состоят в основном из макроопределений `define`. В табл. E.4 перечислены и описаны параметры `define`, чаще всего фигурирующие в исходных файлах OSTYPE. Если параметру назначается значение по умолчанию, оно приводится в квадратных скобках после описания параметра.

Таблица E.4. Параметры define для файлов OSTYPE

Параметр	Назначение
ALIAS_FILE	Имя файла псевдонимов. [*/etc/mail/aliases*]
HELP_FILE	Имя файла справки. [*/etc/mail/helpfile*]
QUEUE_DIR	Каталог файлов очереди. [*/var/spool/mqueue*]

Параметр	Назначение
STATUS_FILE	Имя файла состояния. [*/etc/mail/statistics*]
LOCAL_MAILER_PATH	Программа локальной доставки почты. [*/bin/mail*]
LOCAL_MAILER_FLAGS	Флаги почтовой программы local, добавляемые к «lsDFMAW5:/|@q». [Prmn9]
LOCAL_MAILER_ARGS	Аргументы для локальной доставки почты. [mail -d $u]
LOCAL_MAILER_MAX	Максимальный размер локальной почты
LOCAL_MAILER_CHARSET	Набор символов для локальной 8-разрядной почты MIME
LOCAL_MAILER_DSN_ DIAGNOSTIC_CODE	Код уведомления о состоянии доставки, используемый для локальной почты. [X- Unix]
LOCAL_MAILER_EOL	Символ конца строки для локальной почты
LOCAL_MAILER_ MAXMSG	Максимальное число сообщений, доставляемых в одном соединении
LOCAL_SHELL_PATH	Интерпретатор, используемый для доставки почты программам. [*/bin/sh*]
LOCAL_SHELL_FLAGS	Флаги почтовой программы prog, добавляемые к lsDFM. [eu9]
LOCAL_SHELL_ARGS	Аргументы для почты «prog». [sh -c $u]
LOCAL_SHELL_DIR	Каталог, в котором выполняется интерпретатор. [*$z:/*]
USENET_MAILER_PATH	Программа для работы с конференциями. [*/usr/lib/news/inews*]
USENET_MAILER_FLAGS	Флаги почтовой программы для Usenet. [rDFMmn]
USENET_MAILER_ARGS	Аргументы почтовой программы usenet. [-m -h -n]
USENET_MAILER_MAX	Максимальный размер почтовых сообщений для usenet. [100000]
SMTP_MAILER_FLAGS	Флаги, добавляемые к «mDFMuX», для всех почтовых программ SMTP
SMTP_MAILER_MAX	Максимальный размер сообщения для всех почтовых программ SMTP
SMTP_MAILER_ARGS	Аргументы почтовой программы smtp. [IPC $h]
ESMTP_MAILER_ARGS	Аргументы почтовой программы esmtp. [IPC $h]
DSMTP_MAILER_ARGS	Аргументы почтовой программы dsmtp. [IPC $h]
SMTP8_MAILER_ARGS	Аргументы почтовой программы smtp8. [IPC $h]
RELAY_MAILER_ARGS	Аргументы почтовой программы relay. [IPC $h]
RELAY_MAILER_FLAGS	Флаги, добавляемые к «mDFMuX», для почтовой программы relay

Таблица E.4 (продолжение)

Параметр	Назначение
RELAY_MAIL_MAXMSG	Максимальное число сообщений для почтовой программы relay, доставляемых в одном соединении
SMTP_MAILER_CHARSET	Набор символов для 8-разрядной SMTP-почты MIME
SMTP_MAIL_MAXMSG	Максимальное число сообщений SMTP, доставляемых в одном соединении
UUCP_MAILER_PATH	Путь к почтовой программе UUCP. [*/usr/bin/uux*]
UUCP_MAILER_FLAGS	Флаги, добавляемые к «DFMhuU», для почтовой программы UUCP
UUCP_MAILER_ARGS	Аргументы почтовой программы UUCP. [uux - -r -z -a$g -gC $h!rmail ($u)]
UUCP_MAILER_MAX	Максимальный размер сообщения UUCP. [100000]
UUCP_MAILER_CHARSET	Набор символов для 8-разрядной UUCP-почты MIME
FAX_MAILER_PATH	Путь к программе FAX. [*/usr/local/lib/fax/mailfax*]
FAX_MAILER_ARGS	Аргументы почтовой программы FAX. [mailfax $u $h $f]
FAX_MAILER_MAX	Максимальный размер сообщения для FAX. [100000]
POP_MAILER_PATH	Путь к почтовой программе POP. [*/usr/lib/mh/spop*]
POP_MAILER_FLAGS	Флаги, добавляемые к «lsDFMq», для почтовой программы POP. [Penu]
POP_MAILER_ARGS	Аргументы почтовой программы POP. [pop $u]
PROCMAIL_MAILER_ PATH	Имя программы procmail. [*/usr/local/bin/procmail*]
PROCMAIL_MAILER_ FLAGS	Флаги, добавляемые к «DFM», для почтовой программы Procmail. [SPhnu9]
PROCMAIL_MAILER_ ARGS	Аргументы почтовой программы Procmail. [procmail -Y -m $h $f $u]
PROCMAIL_MAILER_ MAX	Максимальный размер сообщения для почтовой программы Procmail
MAIL11_MAILER_PATH	Имя почтовой программы mail11. [*/usr/etc/mail11*]
MAIL11_MAILER_FLAGS	Флаги для почтовой программы mail11. [nsFx]
MAIL11_MAILER_ARGS	Аргументы почтовой программы mail11. [mail11 $g $x $h $u]
PH_MAILER_PATH	Имя программы phquery. [*/usr/local/etc/phquery*]
PH_MAILER_FLAGS	Флаги для почтовой программы phquery. [ehmu]
PH_MAILER_ARGS	Аргументы почтовой программы phquery. [phquery -- $u]

Параметр	Назначение
QPAGE_MAILER_ARGS	Аргументы почтовой программы qpage. [qpage -10 -m -P$u]
QPAGE_MAILER_FLAGS	Флаги для почтовой программы qpage. [mDFMs]
QPAGE_MAILER_MAX	Максимальный размер сообщения для почтовой программы qpage. [4096]
QPAGE_MAILER_PATH	Имя почтовой программы qpage. [*/usr/local/bin/qpage*]
CYRUS_MAILER_FLAGS	Флаги, добавляемые к «lsDFMnPq», для почтовой программы cyrus. [A5@/:|]
CYRUS_MAILER_PATH	Имя почтовой программы cyrus. [*/usr/cyrus/bin/deliver*]
CYRUS_MAILER_ARGS	Аргументы почтовой программы cyrus. [deliver -e -m $h -- $u]
CYRUS_MAILER_MAX	Максимальный размер сообщения для почтовой программы cyrus
CYRUS_MAILER_USER	Пользователь и группа для почтовой программы cyrus. [cyrus:mail]
CYRUS_BB_MAILER_ FLAGS	Флаги, добавляемые к «lsDFMnP», для почтовой программы cyrusbb
CYRUS_BB_MAILER_ ARGS	Аргументы почтовой программы cyrusbb. [deliver -e -m $u]

Несмотря на длину списка параметров в табл. E.4, большинство макроопределений OSTYPE очень коротки. На то есть несколько причин. Во-первых, многие из параметров, перечисленных в таблице, избыточны. Они определяют одни и те же вещи для различных почтовых программ, а все почтовые программы одновременно не используются ни в одной операционной системе. Во-вторых, значения по умолчанию часто оказываются верными. Определение необходимо только в случае, если операционной системе требуется значение, отличное от принимаемого по умолчанию.

DOMAIN

Исходный файл DOMAIN определяет параметры настройки, относящиеся к локальному домену. Пример файла DOMAIN для воображаемого домена *wrotethebook.com* содержится в главе 10.

В табл. E.5 перечислены некоторые макроопределения `define`, которые обычно фигурируют в файлах DOMAIN. (Синтаксис `define` описан выше.) В таблице описано назначение каждого параметра. Все эти параметры используются для определения узлов пересылки почты. Значением параметра является либо имя узла (то есть имя сервера пересылки почты), либо пара *почтовая_программа*:*имя_узла*, где *почтовая_программа* – внутреннее имя локаль-

ной почтовой программы sendmail, а *имя_узла* – имя удаленного сервера пересылки почты. Если указано только имя узла, по умолчанию используется почтовая программа *relay*, осуществляющая пересылку почты SMTP. Если отсутствуют оба значения, псевдодомены BITNET, DECNET и FAX не используются, а локальный узел должен обладать способностью обработать собственную почту UUCP и «локальную» почту.

Таблица E.5. Макроопределения define, пересылка почты

Параметр	Назначение
UUCP_RELAY	Сервер для почты с UUCP-адресацией
BITNET_RELAY	Сервер для почты с BITNET-адресацией
DECNET_RELAY	Сервер для почты с DECNET-адресацией
FAX_RELAY	Сервер для почты, адресованной в псевдодомен .FAX[1]
LOCAL_RELAY	Сервер для неполных имен
LUSER_RELAY	Сервер для локальных – по внешнему виду – имен, не являющихся на самом деле локальными
MAIL_HUB	Сервер для всей входящей почты
SMART_HOST	Сервер для всей исходящей почты

[1] Почтовая программа «fax» имеет более высокий приоритет, чем данное указание.

Приоритет для серверов, определенных данными параметрами, убывает от частных к общим. Если определены UUCP_RELAY и SMART_HOST, для исходящей почты UUCP используется UUCP_RELAY, несмотря на то, что ретранслятор SMART_HOST обрабатывает «всю» исходящую почту. Если определены LOCAL_RELAY и MAIL_HUB, обязательно также использовать команду FEATURE(stickyhost), чтобы получить ожидаемое поведение системы.

В присутствии возможности stickyhost LOCAL_RELAY обрабатывает все локальные адреса, не содержащие раздел узла, а MAIL_HUB обрабатывает все адреса, содержащие раздел узла. Если не предписано использование stickyhost, но определены оба ретранслятора, LOCAL_RELAY игнорируется, а все локальные адреса обрабатываются MAIL_HUB.

Помимо определений, перечисленных в табл. E.5, существует группа макроопределений, относящихся к сокрытию и пересылке; эти макроопределения также хранятся в исходном файле DOMAIN. Некоторые из них используются в примерах главы 10.

`LOCAL_USER(`*usernames*`)`

Определяет имена локальных пользователей, для которых не должна выполняться пересылка, даже если существует определение LOCAL_RELAY или MAIL_HUB. Действие команды идентично добавлению имен пользователей в класс L в файле *sendmail.cf*.

`MASQUERADE_AS(host.domain)`

Преобразует раздел узла адреса отправителя для исходящих сообщений, заменяя его доменным именем *host.domain*. Адреса отправителей, не содержащие имен узлов или использующие имя узла из класса w, подвергаются преобразованию. Действие команды идентично указанию host.domain для макроопределения M в файле *sendmail.cf*. См. примеры использования MASQUERADE_AS и макроопределения M в главе 10.

`MASQUERADE_DOMAIN(otherhost.domain)`

Преобразует раздел узла адреса отправителя для исходящих сообщений, заменяя его доменным именем, определенным в команде MASQUERADE_AS, если раздел узла представлен значением otherhost.domain. Данная команда должна использоваться в сочетании с MASQUERADE_AS. Ее действие идентично добавлению имен узлов в класс M в файле *sendmail.cf*. См. главу 10.

`MASQUERADE_DOMAIN_FILE(filename)`

Загружает имена *otherhost.domain* из файла *filename*. Может использоваться вместо набора команд MASQUERADE_DOMAIN. Действие идентично загрузке класса M из файла при помощи команды `FMfilename` в файле *sendmail.cf*.

`MASQUERADE_EXCEPTION(host.domain)`

Определяет узел, на который не распространяется сокрытие, даже если он принадлежит к домену, который должен быть скрыт. Это позволяет осуществлять сокрытие всего домена при помощи макроопределения MASQUERADE_DOMAIN, а затем вводить исключения для ряда узлов, которые должны быть открыты внешнему миру.

`EXPOSED_USER(username)`

Отключает сокрытие, если имя пользователя в адресе отправителя совпадает с указанным (*username*). Некоторые имена пользователей, такие как root, существуют во многих системах, а следовательно, не уникальны в рамках домена. Для таких имен преобразование раздела узла адреса делает невозможным определение действительного источника сообщения, а значит – и ответы на это сообщение. Данная команда исключает действие команды MASQUERADE_AS на адреса отправителей, принадлежащие определенным пользователям. Действие команды идентично указанию значений в классе E в файле *sendmail.cf*.

`RELAY_DOMAIN(otherhost.domain)`

Указывает узел, для которого осуществляется пересылка почты. Узел, указанный таким способом, добавляется в класс R.

`RELAY_DOMAIN_FILE(filename)`

Указывает файл, содержащий список узлов, для которых осуществляется пересылка почты. Данное макроопределение загружает класс R из указанного файла.

Существует ряд возможностей, влияющих на пересылку и сокрытие. Мы уже обсуждали FEATURE(stickyhost). Вот и другие:

`FEATURE(masquerade_envelope)`

Приводит к сокрытию адресов на конверте, аналогично тому, как выполняется сокрытие для адреса отправителя. Пример использования этой команды приводится в главе 10.

`FEATURE(allmasquerade)`

Приводит к сокрытию адресов получателей, аналогично тому, как выполняется сокрытие для адреса отправителя. Так, если раздел узла адреса получателя соответствует требованиям команды MASQUERADE_AS, он подвергается преобразованию. Не используйте данную возможность, если нет уверенности, что каждый псевдоним, известный локальной системе, известен также и почтовому серверу, обрабатывающему почту для скрываемого домена.

`FEATURE(limited_masquerade)`

Ограничивает сокрытие только узлами, входящими в класс M. Для узлов из класса w сокрытие не выполняется.

`FEATURE(masquerade_entire_domain)`

Предписывает интерпретировать MASQUERADE_DOMAIN как указание выполнять сокрытие для всех узлов домена. В отсутствие этой возможности преобразованию подвергаются только адреса, в точности соответствующие значению MASQUERADE_DOMAIN. Если эта возможность активирована, преобразованию подвергаются все адреса, заканчивающиеся значением MASQUERADE_DOMAIN. Предположим, что определены параметры MASQUERADE_AS(wrotethebook.com) и MASQUERADE_DOMAIN(sales.wrotethebook.com). Если установлена возможность FEATURE(masquerade_entire_domain), каждое имя узла в домене *sales.wrotethebook.com* подвергается преобразованию в *wrotethebook.com* для исходящих сообщений. В противном случае преобразованию подвергается только имя узла *sales.wrotethebook.com*.

Некоторые возможности определяют, как сервер обрабатывает почту, являясь сервером пересылки почты (ретранслятором). Вот эти возможности, упоминавшиеся уже в разделе «FEATURE»:

`FEATURE(access_db)`

Добавляет код, необходимый для использования базы данных управления доступом. База данных связывает пользователя, доменное имя либо IP-адрес с ключевым словом, на основе которого sendmail принимает решение по пересылке для узла, домена или сети.

`FEATURE(blacklist_recipient)`

Использует базу данных управления доступом для управления доставкой почты на основе адреса получателя. Основное назначение access_db – управлять пересылкой и доставкой на основе источника сообщения. Дан-

ная же возможность добавляет механизмы управления пересылкой и доставкой почты на основе пункта назначения.

`FEATURE(dnsbl)`

Управляет доставкой почты на основе черного списка DNS. Для адресов отправителей и получателей, перечисленных в черном списке DNS, можно запретить доставку или пересылку почты.

`FEATURE(promiscuous_relay)`

Выполняет пересылку почты из любой системы в любую. Обычно sendmail не выполняет пересылку почты. Почтовые ретрансляторы могут использоваться для рассылки спама и подделки IP-адресов. Используйте пересылку с осторожностью.

`FEATURE(relay_entire_domain)`

Выполняет пересылку почты из любого домена, упомянутого в классе M, в любую систему.

`FEATURE(relay_hosts_only)`

Выполняет пересылку почты от любого узла, упомянутого в базе данных управления доступом или в классе R.

`FEATURE(relay_based_on_MX)`

Выполняет пересылку почты от любой системы, для которой данная система является MX-сервером.

`FEATURE(relay_local_from)`

Выполняет пересылку сообщений, адрес отправителя в которых содержит локальное доменное имя.

Входящая почта также может фильтроваться с целью сокращения эффектов спам-рассылок. Два макроопределения, доступные для этих целей:

`MAIL_FILTER(`*`name`*`, `*`equates`*`)`

Определяет почтовый фильтр с использованием синтаксиса Sendmail Mail Filter API.

`INPUT_MAIL_FILTER(`*`name`*`, `*`equates`*`)`

Определяет почтовый фильтр и подготавливает вызов для этого фильтра.

Исходный файл DOMAIN используется также для указания возможностей и макроопределений, непосредственно относящихся к DNSv:

`FEATURE(accept_unqualified_senders)`

Принимает почту, даже если адрес отправителя не содержит имени узла. Обычно такие письма принимаются только от пользователей, работающих в системе. Это опасная возможность, которую следует использовать только в изолированных сетях.

`FEATURE(accept_unresolvable_domains)`

Принимает почту от узлов, имена которых не поддаются разрешению посредством DNS. Это опасная возможность, она используется только в системах, не имеющих постоянного доступа к службе DNS.

`FEATURE(always_add_domain)`

Добавляет имя узла системы во все локальные сообщения. Если эта возможность активирована на сервере *crab@wrotethebook.com*, почта от локального пользователя *craig*, адресованная локальному пользователю *kathy*, будет доставлена как сообщение от *craig@crab@wrotethebook.com*, адресованное *kathy@crab@wrotethebook.com*.

`FEATURE(bestmx_is_local)`

Почту, адресованную любому узлу, считающему локальный сервер своим MX-сервером, этот сервер считает локальной почтой.

Макроопределения DNS :

`CANONIFY_DOMAIN(`*`domain`*`)`

Определяет имя домена, которое будет передаваться DNS для преобразования в каноническую форму, даже если действует возможность nocanonify. Компьютеры могут быть известны по своим псевдонимам. Официальное доменное имя узла, хранимое в DNS, называется каноническим именем узла. Данное макроопределение обычно используется для включения канонизации локального домена, если действует указание nocanonify.

`CANONIFY_DOMAIN_FILE(`*`filename`*`)`

Указывает файл, содержащий список доменных имен, для которых должна выполняться канонизация, даже если действует указание nocanonify.

`LOCAL_DOMAIN(`*`alias-hostname`*`)`

Определяет псевдоним для локального узла. Почта, адресованная по этому псевдониму, будет принята так, как если бы была адресована напрямую локальному узлу.

Макроопределения и возможности, описанные в данном разделе, не ограничены исходным файлом DOMAIN, они могут фигурировать в любом исходном файле `m4`, и часто встречаются в управляющем файле макроопределений. Однако они наиболее естественно ассоциируются с файлом DOMAIN, как и указано в документации, в файле *cf/cf/README*.

MAILER

Вполне возможно, вам понадобится изменить имя файла в файле OSTYPE или определения специфичной для домена информации в файле DOMAIN, но никогда не возникнет необходимость создавать исходный файл `MAILER`, если только вы не занимаетесь разработкой собственной программы доставки почты. Достаточно включить один или несколько существующих файлов определений в файл макроопределений настройки.

Доступные файлы MAILER и назначение каждого из них перечислены в табл. E.6. Включение файлов производится командой `MAILER(`*`value`*`)` в файле *.mc*, где *value* – имя почтовой программы из следующей таблицы.

Таблица Е.6. Значения MAILER

Имя	Функциональность
Local	Почтовые программы local и prog
Smtp	Все почтовые программы SMTP: smtp, esmtp, smtp8 и relay
Uucp	Все почтовые программы UUCP: uucp-old (uucp) и uucp-new (suucp)
Usenet	Поддержка конференций Usenet
fax	Поддержка факса посредством программы FlexFAX
pop	Поддержка протокола POP (Post Office Protocol)
procmail	Интерфейс для procmail
mail11	Почтовая программа DECnet mail11
phquery	Программа phquery для телефонной книги CSO
qpage	Почтовая программа QuickPage, используемая для передачи сообщений электронной почты на пейджер
cyrus	Почтовые программы cyrus и cyrusbb

Ваш конфигурационный файл макроопределений должен содержать строки MAILER(local) и MAILER(smtp). Это обеспечивает доступ к почтовым программам local и prog, необходимым для работы sendmail, программе smtp для стандартной почты SMTP, программе esmtp для Extended SMTP, программе smtp8 для 8-разрядной почты MIME, а также программе relay для различных серверов пересылки почты, упомянутых в разделе «DOMAIN» данного приложения. Выбор local и smtp обеспечивает все необходимое для стандартной системы TCP/IP.

Из всех оставшихся почтовых программ широко используется только uucp. uucp обеспечивает поддержку почты UUCP для систем, напрямую подключенных к сетям UUCP. Почтовая программа uucp-old реализует поддержку стандартной почты UUCP, а uucp-new используется для работы с удаленными системами, не способными выполнять обработку для нескольких получателей за один прием. Системе нужна почтовая программа, соответствующая способностям удаленной почтовой площадки. Используйте класс U для указания имен систем, работающих со старой почтовой программой, и класс Y для указания имен удаленных систем, способных работать с новой программой. Добавьте MAILER(uucp) после строки MAILER(smtp), если система работает одновременно с соединениями TCP/IP и UUCP. Такой порядок операторов MAILER добавляет к стандартным почтовым программам UUCP еще две: программу uucp-dom для поддержки стандартных доменных имен и uucp-uudom для поддержки стандартных доменных имен в контексте стандартного конверта UUCP.

Прочие почтовые программы используются редко:

`usenet`

Изменяет правила подстановки sendmail таким образом, чтобы локальная почта, содержащая «.usenet» в имени пользователя, передавалась программе inews. Вместо этой почтовой программы используйте пользовательскую почтовую программу, поддерживающую работу с конференциями Usenet. Не пытайтесь приспособить для этой цели sendmail.

`fax`

Экспериментальная почтовая программа, поддерживающая программу HylaFax.

`pop`

В большинстве систем поддержка POP реализуется отдельно – демоном popd, так что команда MAILER(pop) не используется.

`procmail`

Реализует интерфейс с procmail, используемый в mailertable. Дистрибутив sendmail V8 не содержит procmail. Даже если procmail используется в качестве локальной почтовой программы, как в Slackware Linux, команда MAILER(procmail) не требуется.

`mail11`

Применяется только в почтовых сетях DECNET.

`phquery`

Предоставляет программу поиска имен для службы каталогов телефонной книги CSO. Пользовательские службы каталогов обычно настраиваются в пользовательских почтовых программах, а не в sendmail.

`qpage`

Обеспечивает интерфейс с пейджинговой службой посредством программы QuickPage.

`cyrus`

Программа локальной доставки почты с архитектурой почтовых ящиков. Почтовые программы cyrus и cyrusbb не получили широкого распространения.

На этом завершается наш разговор о макроопределениях `m4`. Результатом обработки всех файлов и команд, поступающих на вход макропроцессора `m4`, является файл *sendmail.cf*. Оставшаяся часть этого приложения содержит дополнительные подробности о настройках *sendmail.cf*. Основной массив информации о файле *sendmail.cf* содержится в главе 10.

И снова sendmail.cf

Для создания файла *sendmail.cf* могут использоваться многочисленные параметры и флаги. Все важные параметры настройки описаны в главе 10. Но если вам не повезло столкнуться с настройками, требующими изменения од-

ного или нескольких второстепенных параметров, приводимые далее таблицы будут полезны.

Макроопределения sendmail

Файл *sendmail.cf* содержит большое число макропеременных. Макроопределения полезны, поскольку позволяют хранить значения, специфичные для вашей системы, и при этом работать с именами, которые не зависят от конкретной системы или машины. Это делает возможным использование во многих системах практически одного и того же файла настройки – достаточно изменять лишь значения макроопределений. В данном приложении в двух таблицах перечислены все внутренние макроопределения sendmail. В табл. E.7 перечислены все макроопределения с односимвольными именами.

Таблица E.7. Макроопределения с односимвольными именами

Макроопределение	Значение
a	Дата и время, когда было отправлено письмо
b	Текущая дата в формате RFC 822
B	Имя ретранслятора Bitnet
c	Число пересылок сообщения
C	Имя ретранслятора DECnet
d	Текущая дата и время в формате ctime
E	Зарезервировано для ретранслятора X.400
f	Адрес отправителя
F	Имя ретранслятора FAX
g	Адрес отправителя в формате полного обратного адреса
h	Узел получателя
H	Имя почтового концентратора
i	Идентификатор почтовой очереди
j	Абсолютное доменное имя локального компьютера
k	Имя узла UUCP для локальной системы
L	Имя ретранслятора LUSER_RELAY
m	Имя локального домена
M	Имя, используемое для сокрытия имен в исходящей почте
n	Имя отправителя, используемое в сообщениях об ошибках
p	Идентификатор процесса sendmail, функционирующего в качестве агента доставки почты
r	Протокол, использовавшийся для первоначального получения сообщения

Таблица E.7 (продолжение)

Макроопределение	Значение
R	Имя ретранслятора LOCAL_RELAY
s	Имя узла машины отправителя
S	Имя ретранслятора SMART_HOST
t	Численное представление текущей даты и времени
u	Имя пользователя-адресата
U	Локальное имя UUCP, переопределяющее значение $k
v	Номер версии работающего демона sendmail
V	Имя ретранслятора UUCP для узлов класса V
w	Имя узла для локальной системы
W	Имя ретранслятора UUCP для узлов класса W
x	Полное имя отправителя
X	Имя ретранслятора UUCP для узлов класса X
Y	Имя ретранслятора UUCP для всех прочих узлов
z	Исходный каталог адресата
Z	Номер версии
_	Адрес отправителя, проверенный identd

Текущая версия sendmail допускает многосимвольные имена макроопределений. В табл. E.8 перечислены макроопределения с длинными именами.

Таблица E.8. Зарезервированные макроопределения с длинными именами

Макроопределение	Значение
{auth_authen}	Личность пользователя, прошедшего аутентификацию
{auth_author}	Источник данных аутентификации
{auth_ssf}	Количество битов в ключе шифрования, используемом AUTH
{auth_type}	Тип используемого механизма аутентификации
{bodytype}	Значения из параметра ESMTP BODY
{cert_issuer}	Официальное имя издателя сертификата
{cert_subject}	Официальное имя предмета сертификации
{cipher_bits}	Длина ключа шифрования, используемого в соединениях
{cipher}	Механизм шифрования соединений
{client_addr}	IP-адрес удаленного клиента, подключившегося через порт TCP 25

Макроопределение	Значение
{client_name}	Каноническое имя клиента, подключившегося через порт TCP 25
{client_port}	Исходный номер порта удаленного клиента
{client_resolve}	Ключевое слово OK, FAIL, Forged или TEMP; указывает результат обратного запроса DNS для IP-адреса клиента
{currHeader}	Содержимое текущего заголовка (в процессе обработки заголовка)
{daemon_addr}	IP-адрес сетевого интерфейса, через который принимает почту демон. Обычно имеет значение 0.0.0.0 (все интерфейсы)
{daemon_family}	Используемое семейство протоколов. Обычно имеет значение inet (означает TCP/IP). Прочие значения: inet6, iso и ns
{daemon_flags}	Флаги, установленные командой DaemonPortOption, если таковые имеются
{daemon_info}	Общая информация о демоне
{daemon_name}	Имя демона. Обычно Daemon1, если не было переопределено командой DaemonPortOptions
{daemon_port}	Порт, на котором ведет прием демон, обычно 25
{deliveryMode}	Текущий режим доставки
{envid}	Значение DSN ENVID из заголовка Mail From:
{hdrlen}	Длина строки, хранимой в {currHeader}
{hdr_name}	Название текущего заголовка (в процессе обработки заголовка)
{if_addr}	IP-адрес сетевого интерфейса, используемого в текущем входящем соединении
{if_name}	Имя узла, назначенное сетевому интерфейсу, используемому в текущем входящем соединении
{mail_addr}	Почтовый адрес пользователя из тройки значений доставки почты, созданной на основе заголовка конверта MAIL From:
{mail_host}	Имя узла из тройки значений доставки почты, созданной на основе заголовка конверта MAIL From:
{mail_mailer}	Имя почтовой программы из тройки значений доставки почты, созданной на основе заголовка конверта MAIL From:
{MessageIdCheck}	Значение из входящего заголовка Message-Id:
{ntries}	Число попыток доставки
{opMode}	Режим работы, указанный в командной строке sendmail
{queue_interval}	Интервал обработки очереди, предписанный аргументом ключа -q командной строки
{rcpt_addr}	Почтовый адрес пользователя из тройки значений доставки почты, созданной на основе заголовка конверта RCPT To:

Таблица E.8 (продолжение)

Макроопределение	Значение
{rcpt_host}	Имя узла из тройки значений доставки почты, созданной на основе заголовка конверта RCPT To:
{rcpt_mailer}	Имя почтовой программы из тройки значений доставки почты, созданной на основе заголовка конверта RCPT To:
{server_addr}	IP-адрес удаленного сервера для исходящего соединения
{server_name}	Имя удаленного сервера для исходящего соединения
{tls_version}	Версия TLS/SSL, используемая в данном соединении
{verify}	Результат процесса проверки

Классы sendmail

Как видно из предшествующих таблиц, число внутренних макроопределений sendmail велико. Кроме того, в sendmail существуют внутренние классы. Большинство из них по-прежнему имеют односимвольные имена. Есть несколько классов и с новыми длинными именами. Полный перечень внутренних классов содержится в табл. E.9.

Таблица E.9. Внутренние классы sendmail

Имя	Содержимое
B	Доменные имена, вовлеченные в процесс bestmx-is-local
E	Имена пользователей, которые не должны подвергаться сокрытию
G	Домены, для которых следует осуществлять поиск в genericstable
L	Локальные пользователи, для которых не выполняется пересылка на MAIL_HUB и LOCAL_RELAY
e	Список поддерживаемых кодировок передачи содержимого MIME (Content-Transfer-Encodings). Инициализируется значениями 7bit, 8bit и binary
k	Имена узла UUCP для данной системы
M	Домены, для которых следует осуществлять сокрытие
m	Все локальные домены для данного узла
n	Типы тела сообщения MIME, которые не должны подвергаться 8- или 7-битному кодированию. Исходное значение: multipart/signed
q	Типы содержимого MIME (Content-Types), которые не должны подвергаться кодированию Base64. Исходное значение: text/plain
N	Узлы и домены, для которых не должно выполняться сокрытие
O	Символы, запрещенные к использованию в именах локальных пользователей
P	Имена псевдодоменов, такие как REDIRECT

Имя	Содержимое
R	Домены, для которых эта система осуществляет пересылку почты
s	Подтипы сообщений MIME, подверженные рекурсивной обработке. Исходное значение: rfc822
t	Список доверенных пользователей
U	Список узлов UUCP с локальным подключением
V	Узлы UUCP, доступные через ретранслятор $V
W	Узлы UUCP, доступные через ретранслятор $W
X	Узлы UUCP, доступные через рестранслятор $X
Y	Напрямую подключенные «умные» узлы UUCP
Z	Напрямую подключенные узлы UUCP, использующие доменные имена
.	Литерал точки (.)
[	Литерал левой квадратной скобки ([)
{LDAPRoute}	Список доменов, для которых могут изменяться маршруты в результате поиска в каталоге LDAP
{VirtHost}	Список имен узлов и доменов, которые являются допустимыми виртуальными именами узлов
w	Все имена, которые данная система считает своими

Параметры sendmail

Файл *sendmail.cf* позволяет устанавливать значения большого числа параметров настройки sendmail. Синтаксис команды установки параметра описан в главе 10 (см., в частности, табл. 10.1). Ниже приводится полный перечень параметров:

`AliasFile=[`*class*`:]`*file*`, [`*class*`:]`*file*`...`

Указывает файл(ы) псевдонимов. Аргумент *class* – необязательный, по умолчанию принимает значение `implicit`. Допустимые классы: `implicit`, `hash`, `dbm`, `stab` (внутренняя таблица символов) или `nis`. Поддержка выбранного типа базы данных должна быть встроена в sendmail на этапе компиляции. *file* – полное имя файла псевдонимов.

`AliasWait=`*timeout*

Выждать *timeout* минут появления записи «@:@» в базе данных псевдонимов, прежде чем начинать работу. Когда заканчивается интервал ожидания, база данных автоматически пересобирается, если установлен параметр `AutoRebuildAliases`; в противном случае генерируется предупреждающее сообщение.

`AuthMechanisms=`*list*

Объявить поддержку перечисленных механизмов проверки подлинности.

`AuthOptions=`*list*

Список параметров, поддерживаемых для аргумента SMTP AUTH.

`AllowBogusHELO`

Принимать недопустимые команды HELO SMTP, не содержащие имени узла.

`AutoRebuildAliases`

Автоматически выполнять пересборку базы данных псевдонимов, когда это необходимо. Предпочтительный вариант: осуществлять пересборку базы данных псевдонимов явной командой `newaliases`.

`BlankSub=`*c*

Использовать *c* для замены немаскированных символов пробелов в адресах. По умолчанию пробелы сохраняются без изменений.

`CACERTFile=`*filename*

Указывает файл, содержащий сертификат издателя сертификатов.

`CACERTPath=`*path*

Указывает каталог, содержащий сертификаты различных издателей сертификатов.

`CheckAliases`

При пересборке базы данных псевдонимов проверять, что адреса доставки для каждого псевдонима являются допустимыми. Обычно такая проверка не выполняется. Выполнение проверки существенно замедляет сборку базы данных. Параметр принимает логические значения.

`CheckpointInterval=`*n*

Создавать контрольную точку для очереди после обработки каждых *n* элементов в целях упрощения восстановления после сбоя системы, произошедшего во время обработки. По умолчанию – 10.

`ClassFactor=`*fact*

Множитель, используемый для выбора сообщений с более высокими значениями в заголовке Priority:. По умолчанию – 1800.

`ClientCertFile=`*file*

Указывает файл, содержащий сертификат, используемый данной системой при работе в качестве клиента.

`ClientKeyFile=`*file*

Указывает файл, содержащий закрытый ключ, используемый данной системой при работе в качестве клиента.

`ClientPortOptions=`*options*

Определяет нестандартные настройки для случаев, когда данная система выступает в роли клиента SMTP. *options* – это список пар *ключевое_слово=значение*, разделяемых запятыми. Допустимые пары:

`Port=`*port*

Определяет порт-источник, используемый клиентом для исходящих соединений. Порт может обозначаться именем или номером. Если используется имя, ему должно соответствовать определение в файле */etc/services*. По умолчанию порт-источник для исходящих соединений генерируется системой.

`Addr=`*address*

Определяет адрес сетевого интерфейса, используемого клиентом для исходящих соединений. Адрес может быть представлен именем или в десятичной записи через точку. По умолчанию используется любой доступный интерфейс.

`Family=`*protocol*

Определяет семейство протоколов, используемое для соединения. По умолчанию принимается значение `inet`, то есть семейство протоколов TCP/IP.

`SndBufSize=`*bytes*

Определяет размер буфера передачи.

`RcvBufSize=`*bytes*

Определяет размер буфера приема.

`Modifier=`*flags*

Определяет флаги демона для клиента. Доступен только один флаг, `h`. Флаг `h` предписывает клиенту использовать имя, назначенное интерфейсу, в командах SMTP HELO и EHLO.

`ColonOkInAddr`

Разрешает использование двоеточий в адресах электронной почты (к примеру, *host:user*). Двоеточия всегда допустимы для маршрутизации почты (*nodename::user*) или в групповых конструкциях RFC 822 (groupname: member1, member2, ...;). По умолчанию данный параметр «включен», если уровень версии настроек меньше 6.

`ConnectionCacheSize=`*n*

Число единовременно открытых (кэшированных) соединений для данного экземпляра sendmail. По умолчанию – 1. Максимум – 4. Значение 0 приводит к тому, что соединения закрываются сразу же после передачи данных, что является для sendmail традиционным режимом работы.

`ConnectionCacheTimeout=`*timeout*

Длительность поддержания пассивного открытого соединения. По истечении *timeout* минут бездействия соединение закрывается. По умолчанию – 5 минут.

`ConnectionRateThrottle=`*n*

Ограничивает число входящих соединений, поступающих за одну секунду, числом *n*. По умолчанию – 0, то есть ограничений нет.

`ConnectOnlyTo=address`

Ограничивает все соединения SMTP единственным адресом получателя. Используется только для тестирования.

`ControlSocketName=path`

Определяет путь управляющего сокета Unix, используемого для управления соединениями демона. По умолчанию значение не определено.

`DaemonPortOptions=options`

Устанавливает серверные параметры SMTP. Значение *options* представлено парами ключ=значение. Параметры:

`Port=portnumber`

где *portnumber* – любой допустимый номер порта. Можно использовать номер или имя, упомянутое в файле */etc/services*. По умолчанию – порт 25, SMTP.

`Addr=mask`

где mask – адресная маска в десятичной записи через точку или имя сети. По умолчанию – INADDR-ANY, то есть принимаются все адреса.

`Family=addressfamily`

где *addressfamily* – допустимое семейство адресов (см. описание команды `ifconfig`). По умолчанию – INET, что позволяет использовать IP-адреса.

`Listen=n`

где *n* – допустимое число соединений, ожидающих обработки. По умолчанию – 10.

`SndBufSize=n`

где *n* – размер буфера передачи.

`RcvBufSize=n`

где *n* – размер буфера приема.

`DataFileBufferSize=bytes`

Определяет максимальный объем памяти, который может использоваться для буферизации файла данных.

`DeadLetterDrop=file`

Указывает файл, в котором сохраняются сообщения, не подлежащие возврату отправителю или не доставленные пользователю postmaster.

`DefaultAuthInfo=file`

Указывает файл, содержащий информацию идентификации для исходящих соединений.

`DefaultCharSet=charset`

Набор символов, указываемый в заголовке Content-Type: при преобразовании 8-разрядных данных в формат MIME. По умолчанию – `unknown-8bit`. Данный параметр может переопределяться полем `Charset=` в описании почтовой программы.

`DefaultUser=`*`user`*`[:`*`group`*`]`

Значения по умолчанию идентификаторов группы и пользователя для почтовых программ, определения которых не содержат флага S. Если не указана группа (*group*), используется группа пользователя, указанная в файле */etc/passwd*. По умолчанию – 1:1.

`DeliveryMode=`*`x`*

Доставка в режиме *x*, где *x* принимает значения i (interactive delivery, диалоговая доставка), b (background delivery, фоновая доставка), q (queue the message, поместить сообщение в очередь) либо d (defer until the queue run, отложить сообщение до обработки очереди). По умолчанию– b.

`DHParameters=`*`parameters`*

Определяет параметры DH для шифрования DSA/DH.

`DialDelay=`*`delaytime`*

Пауза в *delaytime* секунд перед восстановлением разорванного соединения в сетях с подключением по необходимости. По умолчанию – 0 (восстановление не выполняется).

`DontBlameSendmail=`*`options`*

Отключает проверки безопасности, выполняемые sendmail на уровне файлов. *options* – список ключевых слов, разделяемых запятыми. Значения данного параметра устанавливаются командой `define` confDONT_BLAME_SENDMAIL в исходном файле `m4`. Допустимые ключевые слова:

`AssumeSafeChown`

Разрешить команду `chown`, поскольку она доступна только пользователю root.

`ClassFileInUnsafeDirPath`

Использовать любой каталог из команды F.

`DontWarnForwardFileInUnsafeDirPath`

Не предупреждать о небезопасном каталоге файла *.forward*.

`ErrorHeaderInUnsafeDirPath`

Использовать файл заголовка ошибочных сообщений независимо от его расположения.

`FileDeliveryToHardLink`

Разрешить доставку в файл, являющийся жесткой ссылкой.

`FileDeliveryToSymLink`

Разрешить доставку в файл, являющийся символической ссылкой.

`ForwardFileInUnsafeDirPath`

Использовать файл *.forward*, даже если он расположен в небезопасном каталоге.

`ForwardFileInUnsafeDirPathSafe`

Использовать ссылки на программы и файлы из файла *.forward*, даже если он расположен в небезопасном каталоге.

`ForwardFileIngroupWritableDirPath`

Использовать файл *.forward*, даже если он расположен в каталоге, доступном для записи группе.

`GroupWritableAliasFile`

Использовать файл псевдонимов, даже если он доступен для записи группе.

`GroupWritableDirPathSafe`

Считать все каталоги, доступные для записи группе, «безопасными».

`GroupWritableForwardFileSafe`

Использовать файл *.forward*, даже если он доступен для записи группе.

`GroupWritableIncludeFileSafe`

Использовать файлы *:include:*, даже если они доступны для записи группе.

`HelpFileinUnsafeDirPath`

Использовать файл справки, даже если он расположен в небезопасном каталоге.

`IncludeFileInUnsafeDirPath`

Использовать файлы *:include:*, даже если они расположены в небезопасных каталогах.

`IncludeFileInUnsafeDirPathSafe`

Использовать ссылки на программы и файлы из файлов *:include:*, даже если они расположены в небезопасном каталоге.

`IncludeFileIngroupWritableDirPath`

Использовать файлы *:include:*, даже если они расположены в каталоге, доступном для записи группе.

`InsufficientEntropy`

Использовать STARTTLS, даже если начальное значение генератора случайных чисел для SSL является неадекватным.

`LinkedAliasFileInWritableDir`

Использовать файл псевдонимов, являющийся ссылкой в каталоге, доступном для записи.

`LinkedClassFileInWritableDir`

Загружать значения классов из файлов, являющихся ссылками в каталогах, доступных для записи.

`LinkedForwardFileInWritableDir`

Использовать файлы *.forward*, являющиеся ссылками в каталогах, доступных для записи.

`LinkedIncludeFileInWritableDir`

Использовать файлы *:include:*, являющиеся ссылками в каталогах, доступных для записи.

`LinkedMapInWritableDir`

Использовать файлы баз данных, являющиеся ссылками в каталогах, доступных для записи.

`LinkedServiceSwitchFileInWritableDir`

Использовать файл коммутации служб имен, являющийся ссылкой в каталоге, доступном для записи.

`MapInUnsafeDirPath`

Использовать файлы баз данных, расположенные в небезопасных каталогах.

`NonRootSafeAddr`

Не помечать операции в качестве небезопасных, если sendmail не работает с полномочиями пользователя root.

`RunProgramInUnsafeDirPath`

Выполнять программы, расположенные в каталогах, доступных для записи.

`RunWritableProgram`

Выполнять программы, доступные для записи группе или всем пользователям.

`Safe`

Оставить все проверки включенными. Настройка по умолчанию.

`TrustStickyBit`

Доверять каталогам, доступным для записи группе и всем пользователям, если установлен «липкий» бит (sticky bit).

`WorldWritableAliasFile`

Использовать файл псевдонимов, даже если он доступен для записи всем пользователям.

`WriteMapToHardLink`

Осуществлять запись в файл базы данных, даже если он является жесткой ссылкой.

`WriteMapToSymLink`

Осуществлять запись в файл базы данных, даже если он является символической ссылкой.

`WriteStatsToHardLink`

Осуществлять запись в файл состояния, даже если он является жесткой ссылкой.

`WriteStatsToSymLink`

Осуществлять запись в файл состояния, даже если он является символической ссылкой.

`DontExpandCnames`

Заблокировать конструкцию `$[`*name*`$]`, используемую для преобразования псевдонимов в канонические имена.

`DontInitGroups`

Не использовать вызовы initgroups(3). Это сокращает нагрузку на сервер NIS, однако ограничивает пользователя группой, указанной для него в файле */etc/passwd*.

`DontProbeInterfaces`

Установка значения true для данного параметра предотвращает добавление имен и адресов сетевых интерфейсов в класс w. По умолчанию значение false, поэтому имена и адреса интерфейсов сохраняются в классе w.

`DontPruneRoutes`

Не оптимизировать явные почтовые маршруты. Обычно sendmail старается максимально «выпрямить» маршрут. Однако оптимизация маршрута может быть неуместной для систем, защищенных брандмауэрами.

`DoubleBounceAddress=`*error-address*

Отправить сообщение об ошибке, произошедшей во время отправки сообщения об ошибке, на адрес *error-address*. По умолчанию `postmaster`.

`EightBitMode=`*action*

Обрабатывать незадекларированные 8-разрядные данные указанным образом. Возможные действия: `s` (strict), отвергнуть незадекларированные 8-разрядные данные; `m` (mime), преобразовать в формат MIME; либо `p` (pass), передать дальше без изменений.

`ErrorHeader=`*file-or-message*

Предварять исходящие сообщения об ошибках текстом *file-or-message*. Если *file-or-message* является именем текстового файла, строка должна начинаться с символа /. Если параметр не определен, никакой текст не добавляется в сообщения об ошибках.

`ErrorMode=`*x*

Обрабатывать сообщения об ошибках указанным образом. *x* может принимать значения: `p` (печатать сообщения), `q` (сообщать код завершения, но не отображать сообщения), `m` (возвращать сообщения почтой), `w` (записывать сообщения на терминал пользователя) либо `e` (возвращать сообщения почтой и всегда завершать работу с нулевым кодом). Если параметр не определен, сообщения об ошибках выводятся на печать.

`FallbackMXhost=`*fallbackhost*

Использовать *fallbackhost* в качестве резервного сервера MX для всех узлов.

`ForkEachJob`

Порождать отдельный процесс для доставки каждого элемента почтовой очереди. Данный параметр сокращает объем памяти, необходимый для обработки очереди.

`ForwardPath=`*path*

Путь для поиска файлов *.forward*. Чтобы указать несколько каталогов, разделите их двоеточиями. По умолчанию – *$z/.forward*.

`HelpFile=`*file*

Путь к файлу справки.

`HoldExpensive`

Задерживать почту, адресованную почтовым программам с установленным флагом `e` (expensive), до следующей обработки очереди. Обычно доставка почты выполняется немедленно.

`HostsFile=`*path*

Имя файла таблицы узлов. По умолчанию – */etc/hosts*.

`HostStatusDirectory=`*path*

Каталог, в котором хранится информация состояния узлов, доступная различным процессам sendmail. Обычно состояние узла или соединения известно только процессу, определившему это состояние. Чтобы данный параметр подействовал, значение ConnectionCacheSize должно быть не меньше 1.

`IgnoreDots`

Игнорировать точки в поступающих сообщениях. Точки не могут игнорироваться в почте SMTP, поскольку используются для обозначения конца почтового сообщения.

`LDAPDefaultSpec=`*specification*

Значение по умолчанию для спецификации баз данных LDAP.

`LogLevel=`*n*

n указывает уровень подробности информации, заносимой в файл журнала. По умолчанию *n* равно 9, что обычно соответствует достаточно большому количеству информации.

`MatchGECOS`

Сопоставлять имя пользователя из адреса электронной почты с полем GECOS файла *passwd*, если оно не было найдено в базе данных псевдонимов или в поле имени пользователя файла *passwd*. Использовать данный параметр не рекомендуется.

`MaxAliasRecursion=`*n*

Псевдонимы могут ссылаться на другие псевдонимы, и получение окончательного почтового адреса может потребовать нескольких итераций. Данный параметр устанавливает максимальный уровень вложенности для псевдонимов. По умолчанию *n* равно 10.

`MaxDaemonChildren=`*n*

Не принимать соединения, если входящую почту обрабатывают *n* процессов. Обычно sendmail не ограничивает число порожденных процессов.

`MaxHeadersLength=bytes`

Максимальная суммарная длина всех заголовков сообщения.

`MaxHopCount=n`

Считать, что сообщение зациклилось, если оно подверглось обработке более *n* раз. По умолчанию – 25.

`MaxHostStatAge=n`

Сохранять информацию о состоянии узла *n* минут.

`MaxMessageSize=n`

Максимальный размер сообщения, передаваемый в ответ на ESMTP EHLO. Сообщения, превышающие указанный размер, не принимаются.

`MaxMimeHeaderLength=size`

Максимальная длина полей заголовков MIME.

`MaxQueueRunSize=n`

Максимальное число элементов почтовой очереди, обрабатываемых за один проход. По умолчанию ограничений нет.

`MaxRecipientsPerMessage=n`

n ограничивает максимальное число получателей одного сообщения. Если число не указано, ограничений нет.

`MeToo`

Посылать копию письма отправителю.

`MinFreeBlocks=n`

n определяет минимальное число свободных блоков в файловой системе почтовой очереди, позволяющее принимать почту.

`MinQueueAge=n`

Не обрабатывать задания, находящиеся в очереди менее *n* минут.

`MustQuoteChars=s`

Список символов, добавляемых к набору «@,;:\()[]»; такие символы должны маскироваться при использовании в составе имен пользователей в адресах электронной почты. Если значение `s` для MustQuoteChars не указано, к списку добавляется символ «.».

`NoRecipientAction=action`

Действие, выполняемое в случае отсутствия в сообщении корректных заголовков получателей. *action* может принимать значения: `add-to` (добавляется заголовок To: с адресами получателя, взятыми с конверта), `add-apparently-to` (добавляется заголовок Apparently-To:), `add-to-undisclosed` (добавляется заголовок «To: undisclosed-recipients:;») либо `add-bcc` (добавляется пустой заголовок Bcc:). Если значение не определено, сообщение передается далее без изменений.

`OldStyleHeaders`

Разрешает разделять имена пробелами. Обычно имена разделяются запятыми.

`OperatorChars=`*`charlist`*

Список символов-операторов, обычно перечисляемых в макроопределении `o`. По умолчанию используется стандартный набор операторов. В главе 10 описаны лексемы правил подстановки и применение операторов в выделении лексем.

`ProcessTitlePrefix=`*`prefix`*

Строка, используемая в заголовках отчетов о состоянии процессов.

`PostmasterCopy=`*`username`*

Посылать копии сообщений об ошибках пользователю *`username`*. По умолчанию копии сообщений об ошибках не посылаются на адрес `postmaster`.

`PrivacyOptions=`*`options`*

Установить параметры протокола SMTP, где *`options`* – список ключевых слов, разделяемых запятыми. Ключевые слова:

`public`

Разрешить все команды.

`needmailhelo`

Требовать получения HELO или EHLO перед MAIL.

`needexpnhelo`

Требовать получения HELO или EHLO перед EXPN.

`noexpn`

Запретить EXPN.

`needvrfyhelo`

Требовать получения HELO или EHLO перед VRFY.

`novrfy`

Запретить VRFY.

`restrictmailq`

Ограничить доступ к `mailq` пользователями, имеющими групповой доступ к каталогу очереди.

`restrictqrun`

Обрабатывать очередь разрешается только пользователю root и владельцу каталога очереди.

`noreceipts`

Не возвращать сообщения об успешной доставке.

`goaway`

Запретить все запросы состояний SMTP.

`authwarnings`

Добавлять заголовок X-Authentication-Warning: в сообщения.

`QueueDirectory=`*`directory`*

Имя каталога почтовой очереди.

`QueueFactor=`*`factor`*

Коэффициент, используемый совместно с разницей между текущей загрузкой и пределом средней загрузки, а также приоритетом сообщения – для определения того, должно ли сообщение быть помещено в очередь или отправлено немедленно. Смысл заключается в том, чтобы помещать низкоприоритетные сообщения в очередь, если система серьезно загружена. По умолчанию – 600000.

`QueueLA=`*`n`*

Помещать сообщения в очередь, если средняя загрузка системы превышает *n*. По умолчанию – 8.

`QueueSortOrder=`*`sequence`*

Сортировать очередь в указанной последовательности, где *sequence*: `h` (сортировка по узлам), `t` (сортировка по времени создания) либо `p` (сортировка по приоритету сообщений). По умолчанию выполняется сортировка по приоритету.

`RandFile=`*`file`*

Указывает файл, содержащий псевдослучайные данные для определенных механизмов шифрования. Используется, только если опция компиляции HASURANDOM выключена.

`ResolverOptions=`*`options`*

Устанавливает параметры клиента DNS. Доступные значения параметров: `debug`, `aaonly`, `usevc`, `primary`, `igntc`, `recurse`, `defnames`, `stayopen` и `dnsrch`. Чтобы включить параметр, следует предварить его символом сложения (+), чтобы выключить – символом вычитания (–). Еще один параметр, `HasWildcardMX`, фигурирует без символов + и –. Наличие `HasWildcardMX` включает действие соответствующего параметра.

`RrtImpliesDsn`

Значение true предписывает sendmail считать заголовок Return-Receipt-To: запросом на уведомление о состоянии доставки (DSN). По умолчанию – false.

`RunAsUser=`*`userid`*`[:`*`groupid`*`]`

Выполнять sendmail с полномочиями пользователя *userid* из группы *groupid*, а не с полномочиями root. Это может повысить защищенность, когда sendmail работает на качественном брандмауэре. В системах общего назначения безопасность, напротив, может снижаться, поскольку такой режим работы требует полномочий записи и чтения многих файлов для указанного пользователя.

`RecipientFactor=`*`factor`*

Приоритет задания снижается на указанный коэффициент для каждого получателя сообщения. Смысл в том, чтобы задания с большим числом получателей имели более низкий приоритет. По умолчанию – 30000.

`RefuseLA=`*`n`*

Не принимать входящие соединения SMTP, если средняя загрузка системы превышает *n*. По умолчанию – 12.

`RetryFactor=`*`factor`*

Данный коэффициент снижает приоритет задания при каждой его обработке, чтобы почта, которую невозможно доставить, не попадала в начало очереди. По умолчанию – 90000.

`SafeFileEnvironment=`*`directory`*

Выполнять `chroot(2)` в указанный каталог перед записью файла, а также запретить доставку по символическим ссылкам.

`SaveFromLine`

Сохранять строки From: в стиле Unix в начале заголовка. Обычно такие строки удаляются.

`SendMIMEErrors`

Посылать сообщения об ошибках в формате MIME.

`ServerCertFile=`*`file`*

Указывает файл, содержащий сертификат, используемый системой при работе в качестве почтового сервера.

`ServerKeyFile=`*`file`*

Указывает файл, содержащий закрытый ключ, используемый системой при работе в качестве почтового сервера.

`ServiceSwitchFile=`*`path`*

Указывает файл, в котором перечислены методы разрешения имен для различных служб. Запись файла ServiceSwitchFile содержит имя службы и метод. sendmail проверяет наличие служб «aliases» и «hosts», а также поддерживает методы доступа «dns», «nis», «nisplus» и «files» (поддержка методов должна быть встроена в исполняемый файл sendmail при компиляции). По умолчанию ServiceSwitchFile имеет значение */etc/service.switch*. Если файл не существует, sendmail использует следующие методы: поиск псевдонимов осуществляется в файлах псевдонимов, поиск узлов осуществляется сначала в dns, затем в nis и наконец в файле *hosts*. Если операционная система имеет встроенный механизм коммутации служб, используется этот механизм, а значение данного параметра игнорируется. Описание файла коммутации служб *nsswitch.conf* приведено в главе 9.

`SevenBitInput`

Удалять восьмой бит принимаемых данных в целях совместимости со старыми системами. В использовании данного параметра не должно быть необходимости.

`SingleLineFromHeader`

В целях совместимости с некоторыми версиями Lotus Notes преобразовать строки From:, имеющие внедренные символы новой строки, в одну длинную строку.

`SingleThreadDelivery`

Не открывать единовременно более одного соединения SMTP с удаленным узлом. Данный параметр требует наличия параметра HostStatusDirectory.

`SmtpGreetingMessage=message`

Приветствие, передаваемое удаленному узлу, когда он подключается к порту сервера SMTP. Это значение содержится в макроопределении e.

`StatusFile=`*`file`*

Записывать статистическую сводку в указанный файл. По умолчанию сводка не фиксируется.

`SuperSafe`

Создавать файл очереди, даже при попытке немедленно доставить сообщение.

`TempFileMode=`*`mode`*

Использовать режим *mode* для назначения прав доступа для файлов очереди. *mode* – восьмеричное значение. По умолчанию равно 0600.

`Timeout.`*`type`*`=`*`timeout`*

Установить значения интервалов ожидания. Здесь *type* – событие, для которого происходит ожидание, а *timeout* – длительность ожидания. В табл. E.10 перечислены допустимые значения типа, события и значения по умолчанию интервалов ожидания.

Таблица E.10. Типы интервалов ожидания

Тип	Ожидание	По умолчанию
connect	Завершения соединения	1m
control	Завершения передачи через управляющий сокет	2m
iconnect	Подключения к первому узлу для сообщения	5m
initial	Начального приветственного сообщения	5m
helo	Ответа на команду HELO или EHLO	5m
mail	Ответа на команду MAIL	10m
rcpt	Ответа на команду RCPT	1h
datainit	Ответа на команду DATA	5m
datablock	Чтения блока данных	1h
datafinal	Ответа на завершающий символ «.»	1h
rset	Ответа на команду RSET	5m

Тип	Ожидание	По умолчанию
quit	Ответа на команду QUIT	2m
misc	Ответа на команды NOOP и VERB	2m
ident	Ответа протокола IDENT	30s
fileopen	Открытия файла *.forward* или *:include:*	60s
command	Чтения команды	1h
queuereturn	Возврата сообщения из очереди из-за невозможности доставки	5d
queuereturn.normal	Возврата нормального сообщения из очереди из-за невозможности доставки	5d
queuereturn.non-urgent	Возврата несрочного сообщения из очереди из-за невозможности доставки	7d
queuereturn.urgent	Возврата срочного сообщения из очереди из-за невозможности доставки	2d
queuewarn	Предупреждения, что сообщение все еще находится в очереди	4h
queuewarn.normal	Предупреждения, что нормальное сообщение все еще находится в очереди	4h
queuewarn.non-urgent	Предупреждения, что несрочное сообщение все еще находится в очереди	12h
queuewarn.urgent	Предупреждения, что срочное сообщение все еще находится в очереди	1h
resolver.retrans	Ответа на запрос клиента DNS	5s
resolver.retrans.first	Ответа на первый запрос клиента DNS	5s
resolver.retrans.normal	Ответа на обычный запрос клиента DNS	5s
resolver.retry	Устанавливает число повторов запроса клиента DNS	4
resolver.retry.first	Устанавливает число повторов первого запроса клиента DNS	4
resolver.retry.normal	Устанавливает число повторов обычного запроса клиента DNS	4
hoststatus	Удаления устаревших сведений о состоянии узла	30m

`TimeZoneSpec=`*`tzinfo`*

Устанавливает часовой пояс *tzinfo*. Если параметр TimeZoneSpec не установлен, используется системное умолчание; если параметр инициализирован пустым значением, используется переменная пользователя `TZ`.

`TrustedUser=`*`users`*

Список пользователей, которым разрешено отправлять почту, используя имена других пользователей.

`TryNullMXList`

Подключаться напрямую к любому удаленному узлу, который считает локальную систему наиболее предпочтительным сервером MX, как если бы удаленный узел не имел записей MX. Использование этого параметра не приветствуется.

`UnixFromLine=`*`fromline`*

Определяет формат строк From: в стиле Unix. Именно это значение хранится в макроопределении l.

`UnsafeGroupWrites`

Доступные для записи группе файлы *:include:* и *.forward* не могут ссылаться на программы или осуществлять запись напрямую в файлы. Для файлов, доступных для записи всем пользователям, всегда действуют такие ограничения.

`UseErrorsTo`

Отправлять сообщения об ошибках на адрес, указанный в заголовке Errors-To:. Обычно сообщения об ошибках отправляются на адрес отправителя исходного сообщения, указанный на конверте.

`UserDatabaseSpec=`*`udbspec`*

Указание базы данных пользователей.

`UserSubmission`

Указывает, что сообщение не является ретранслированным, оно получено напрямую от пользовательской почтовой программы (Mail User Agent).

`Verbose`

Работа в режиме подробной диагностики.

Примеры установки значений параметров содержатся в главе 10.

Флаги почтовых программ sendmail

Флаги почтовых программ указываются в поле F определений почтовых программ. Каждый флаг включается одним символом, представляющим этот флаг. К примеру, `F=lsDFMe` устанавливает шесть различных флагов. Имена и назначения флагов перечислены в табл. E.11.

Таблица E.11. Флаги почтовых программ sendmail

Имя	Назначение
`C`	Добавлять `@`*`domain`* к адресам, не содержащим символа `@`
`D`	Почтовая программа требует наличия строки заголовка Date:
`E`	Добавлять символ `>` к строкам сообщения, начинающимся с From:
`e`	«Дорогая» почтовая программа. См. описание параметра sendmail `c`
`F`	Почтовая программа требует наличия строки заголовка From:

Имя	Назначение
f	Почтовая программа принимает флаг -f от доверенных пользователей
h	Сохранять прописные буквы в именах узлов
I	Почтовая программа использует SMTP для общения с другой программой sendmail
L	Ограничить длину строки согласно RFC 821
l	Почтовая программа local
M	Почтовая программа требует наличия строки заголовка Message-Id:
m	Почтовая программа способна посылать сообщения нескольким пользователям в одной транзакции
n	Не использовать в сообщении строку From: в стиле Unix
P	Почтовая программа требует наличия строки Return-Path:
R	Использовать путь возврата MAIL FROM: вместо обратного адреса
r	Почтовая программа принимает флаг -r от доверенных пользователей
S	Не сбрасывать идентификатор пользователя перед вызовом почтовой программы
s	Удалять кавычки из адреса перед вызовом почтовой программы
U	Почтовая программа требует, чтобы строки From: записывались в стиле Unix
u	Сохранять прописные буквы в именах пользователей
X	Предварять точкой строки, начинающиеся с точки
x	Почтовая программа требует наличия строки заголовка Full-Name:

Примеры объявлений флагов в определениях почтовых программ приводятся в главе 10.

Команда K

Команда K используется для создания определения базы данных в файле *sendmail.cf*. Синтаксис команды K:

```
Кимя тип [аргументы]
```

В главе 10 приводятся примеры создания определения и использования базы данных sendmail, а также описан синтаксис команды K. В данном приложении перечислены допустимые значения типов и аргументы, которые могут использоваться в команде K.

Поле *тип* команды K указывает, какого рода база данных требует определения. Существует несколько внутренних типов баз данных, присущих только sendmail, и несколько внешних типов, работа с которыми осуществляется посредством внешних библиотек. Поддержка баз данных внешних типов должна быть встроена в sendmail на этапе компиляции. Необходимо явным

образом перечислить типы при помощи команды confMAPDEF в файле *devtools/OS* или *devtools/Site*, используемом сценарием `Build` при компиляции программы. Пример компиляции sendmail приводится в начале данного приложения.

Допустимые значения типа:

`dbm`

База данных формата «новый dbm». Доступ осуществляется посредством библиотеки ndbm(3). Поддерживается только в случае компиляции sendmail с определением NDBM.

`btree`

База данных формата btree. Доступ осуществляется посредством библиотеки Berkeley db(3). Поддерживается только в случае компиляции sendmail с определением NEWDB.

`hash`

База данных формата hash (ассоциативный массив). Доступ осуществляется посредством библиотеки Berkeley db(3). Поддерживается только в случае компиляции sendmail с определением NEWDB.

`nis`

Обращение к серверу NIS. sendmail следует компилировать с определением NIS.

`nisplus`

Обращение к серверу NIS+. sendmail следует компилировать с определением NISPLUS.

`hesiod`

Обращение к серверу MIT hesiod. sendmail следует компилировать с определением HESIOD.

`ldap`

Поиск в LDAP. sendmail следует компилировать с определением LDAPMAP. sendmail поддерживает большинство стандартных аргументов командной строки программы `ldapsearch`.

`netinfo`

Поиск NeXT NetInfo. sendmail следует компилировать с определением NETINFO.

`text`

Поиск в текстовом файле. Не требует внешних библиотек или параметров компиляции. Формат текстовой базы данных: поле ключа, поле значения плюс флаги разделителей. Флаги команды `K` описаны в следующем разделе.

`ph`

Обращение к серверу имен CCSO.

`program`

Запросы передаются для разрешения внешней программе.

`stab`

База данных внутренней таблицы символов.

`implicit`

Внутренний формат sendmail для файла псевдонимов, используемый по умолчанию, если для этого файла не указан тип.

`user`

Специальный тип sendmail, используемый для проверки существования пользователя посредством getpwnam(3).

`host`

Специальный тип sendmail, используемый для преобразования псевдонимов и IP-адресов в канонические имена при помощи сервера доменных имен. Это альтернатива конструкции `$[`*name*`]$`.

`sequence`

Специальный тип sendmail, используемый для указания порядка поиска в определенных ранее базах данных. Предположим, что посредством команды `K` мы определили три базы данных (file1, file2 и file3). Можно добавить четвертую команду `K` вида `Kallfiles sequence file3 file1 file2`, «сочетающую» все три базы данных и предписывающую осуществлять поиск сначала в `file3`, затем в `file1` и наконец в `file2`.

`switch`

Специальный тип sendmail, использующий файл коммутации служб имен для определения порядка поиска в файлах баз данных. В качестве *аргумента* команды `K` с *типом* «switch» должно выступать имя службы в файле коммутации. Значения, связанные с именем службы в файле коммутации, используются для создания имен баз данных, поиск в которых выполняется в порядке следования их определений. Например, команда `Kali switch aliases` ищет запись `aliases`. Если такая запись содержит значение `nis files`, sendmail выполняет поиск в базах данных *ali.nis* и *ali.files* в таком порядке.

`dequote`

Специальный тип sendmail, используемый для удаления нежелательных двойных кавычек (") из адресов электронной почты.

`arith`

Внутренняя подпрограмма, предназначенная для выполнения специальных арифметических вычислений.

`bestmx`

Внутренняя подпрограмма, позволяющая получить MX-запись для узла.

`dns`

Внутренняя подпрограмма, позволяющая получить адрес по имени узла.

`null`

Внутренняя подпрограмма, возвращающая сообщение «Not found» для всех запросов.

`regex`

Внутренняя подпрограмма для работы с регулярными выражениями.

Многие из допустимых типов не относятся к реальным базам данных. Некоторые из них – специальные значения, используемые только в пределах sendmail. Другие ссылаются на внутренние подпрограммы sendmail, доступные из правил подстановки посредством тех же конструкций, что используются для доступа к базам данных.

В качестве аргумента для большинства типов выступает имя файла. Имя файла указывает на внешний файл, в котором хранится база данных. Указывается только собственно имя файла, расширение, соответствующее типу базы данных, sendmail добавляет автоматически. Например, `Krealname dbm /usr/etc/names` становится */usr/etc/names.db*, поскольку верным расширением для баз данных dbm является *.db*.

В дополнение к имени файла поле аргументов может содержать необязательные флаги:

`-o`

Необязательная база данных. sendmail продолжает работу, если файл не найден.

`-N`

Допустимые ключи базы данных завершаются символом NULL.

`-O`

Допустимые ключи базы данных никогда не завершаются символом NULL. Никогда не используйте одновременно `-N` и `-O`, это указание на отсутствие допустимых ключей! Безопаснее всего избегать использования и того и другого флага и дать sendmail возможность самостоятельно определять верную структуру ключа, за исключением случаев, когда вы наверняка знаете, что нужно использовать тот или иной флаг.

`-a`*x*

Добавлять строку *x* к значению, возвращаемому при успешном поиске.

`-f`

Не преобразовывать прописные буквы в строчные перед поиском по ключу.

`-m`

Проверять существование ключа в базе данных, но не заменять ключ значением, найденным при поиске.

`-k`*keycol*

Координаты ключа в записи базы данных. В большинстве баз данных ключ хранится в первом поле записи, так что в данном флаге нет необхо-

димости. При поиске в текстовых файлах этот флаг является обязательным; аргумент *keycol* указывает номер колонки, в которой начинается поле ключа.

`-v`*valcol*

Координаты значения в записи базы данных. В большинстве баз данных значение следует за ключом, так что флаг `-v` не используется. При поиске в текстовых файлах данный флаг является обязательным и указывает номер колонки, в которой начинается поле значения.

`-z`*delim*

Символ-разделитель полей записи базы данных. По умолчанию – пробел.

`-t`

При неудачном поиске в базе данных, связанном с обращением к удаленным серверам, не помещать сообщения в очередь для последующей обработки. В основном используется, если возникают проблемы с сервером DNS. Обычно, если удаленный сервер не ответил, почта сохраняется в очереди для последующей доставки. Установка этого флага приводит к немедленному возврату сообщения отправителю.

`-s`*spacesub*

Использовать символ *spacesub* для замены пробелов после обработки адреса по информации базы данных `dequote`.

`-A`

Разрешить дубликаты ключей. В большинстве баз данных дубликаты ключей запрещены.

`-q`

Сохранять все кавычки внутри ключа. Обычно кавычки удаляются.

Полный список типов баз данных и флагов, приведенный в настоящем приложении, поможет вам разобраться с командами `K`, помещаемыми в файл *sendmail.cf* процессором `m4`. Ваши собственные команды `K` будут гораздо проще. Вы будете придерживаться типов баз данных, которые поддерживает ваша версия `sendmail` и команда `makemap`, и будете создавать простые базы, предназначенные для решения конкретных задач. Примеры таких баз данных приводятся в главе 10, а в следующем разделе приведены простые сценарии, используемые для создания этих баз данных.

Пример сценария

В главе 10 база данных *realnames* используется для переписывания регистрационных имен пользователей в формат «имя точка фамилия» для исходящих сообщений. Приводимый ниже сценарий создает базу данных realnames на основе файла */etc/passwd*.

```
#! /bin/sh
#
# Исключить «непользовательские» учетные записи
```

```
grep -v ':*:' /etc/passwd | \
# Исключить «открытые» имена, т.е. имена пользователей,
#  определенные в классе E как не подлежащие подстановке
grep -v ' root:' | \
# Заменить двоеточия пробелами
sed 's/:/ /g' | \
# Вывод регистрационного имени и имени в формате имя.фамилия
awk '{ print $1, $5"."$6 }' > realnames
# Выполнить сборку базы данных realnames
makemap dbm realnames < realnames
```

Сборка базы данных *realnames* на основе файла *passwd* полностью зависима от формата этого файла. Файл *passwd* должен иметь последовательный формат поля GECOS и четкие признаки для выявления «непользовательских» учетных записей. Такие учетные записи не нужны пользователям для входа в систему или для работы с почтой. Обычно они предназначены для системных и прикладных программ. Классическим примером является учетная запись *uucp*. В каждой системе существует определенный способ указать, что с этими учетными записями пользователи не работают. В некоторых системах это звездочка в поле пароля, в других – восклицательный знак, буквы NP, буква x либо что-то еще. Приведенный сценарий предполагает, что используется звездочка – как в моей Linux-системе. (В моей системе Solaris используется x.) Распечатайте файл *passwd*, определите, какой способ используется, и измените соответствующим образом сценарий.

Кроме того, сценарий предполагает, что первые два значения в поле GECOS – это имя и фамилия пользователя, разделенные пробелом. Если начало поля GECOS имеет любой другой формат, результатом работы сценария станет непригодный к использованию мусор. Действия, выполняемые при регистрации в системе нового пользователя, должны приводить к созданию полей GECOS одного формата. Непоследовательность – враг автоматизации. Приведенный ниже пример демонстрирует файл с непоследовательной структурой и произведенные им не пригодные к использованию данные:

```
% cat /etc/passwd
root:oRd1L/vMzzxno:0:1:System Administrator:/:/bin/csh
nobody:*:65534:65534::/:
daemon:*:1:1::/:
sys:*:2:2::/:/bin/csh
bin:*:3:3::/bin:
uucp:*:4:8::/var/spool/uucppublic:
news:*:6:6::/var/spool/news:/bin/csh
ingres:*:7:7::/usr/ingres:/bin/csh
audit:*:9:9::/etc/security/audit:/bin/csh
craig:1LrpKlz8sYjw:198:102:Craig Hunt:/home/craig:/bin/csh
dan:RSU.NYlKuFqzh2:214:885:Dan Scribner:/home/dan:/bin/csh
becca:monfTHdnjj:101:102:"Becky_Hunt":/home/becca:/bin/csh
dave:lniuhugfds:121:885:David H. Craig:/home/dave:/bin/csh
kathy:TUVigddehh:101:802:Kathleen S McCafferty:/home/kathy:/bin/csh
% build.realnames
```

```
% cat realnames
craig Craig.Hunt
dan Dan.Scribner
becca "Becky_Hunt"./home/becca
dave David.H.  kathy Kathleen.S
```

Ваш файл *passwd* мог расти под надзором различных системных администраторов, он может быть полон непоследовательных структур. Если это так, исправьте ситуацию, прежде чем выполнять сценарий создания псевдонимов, а затем постоянно его контролируйте.

F

Файл httpd.conf в Solaris

Настройка веб-сервера, описанная в главе 11, основывается на стандартном для поставки системы Solaris 8 файле *httpd.conf*. В данном приложении файл приводится во всей полноте – многим читателям будет удобно иметь перед глазами целостный файл настройки, иллюстрирующий отдельные инструкции, описанные в главе 11.[1]

Строки, которые начинаются с символа #, являются комментариями. Многие комментарии описывают функциональность и синтаксис конкретных инструкций файла настройки, так что их можно использовать в качестве дополнительного источника информации по инструкциям, описанным в главе 11.

Ниже полностью приведено содержимое файла *httpd.conf* в Solaris 8.

```
#
# Основан на конфигурационных файлах сервера NSCA,
# созданных Робом Мак-Кулом (Rob McCool)
#
# Это основной конфигурационный файл сервера Apache. Он содержит директивы,
# управляющие работой сервера.
# За подробной информацией об этих директивах обращайтесь по адресу:
# <URL:http://www.apache.org/docs/>
#
# Не стоит просто читать инструкции, не задумываясь о том, что они делают.
# Они выступают здесь только как подсказки. Если не уверены – обращайтесь
# к сопроводительной документации. И считайте, что мы вас предупредили.
#
```

[1] Поскольку этот файл поставляется вместе с Apache, мы сочли нецелесообразным публиковать его в исходном, «непереведенном» виде и взяли на себя смелость перевести на русский язык комментарии для удобства наших читателей. – *Примеч. ред.*

```
# Обработав этот файл, сервер попытается найти и обработать файл
# /etc/apache/srm.conf, а затем /etc/apache/access.conf,
# если только эти имена не будут переопределены директивами
# ResourceConfig и AccessConfig
#
# Конфигурационные директивы сгруппированы в три основных раздела:
#  1. Директивы, управляющие процессом сервера Apache как единым целым
#     (глобальное окружение).
#  2. Директивы, устанавливающие параметры работы "главного" сервера
#     (или "сервера по умолчанию"), который отвечает на те запросы, которые
#     не обрабатываются виртуальными узлами. Также эти директивы определяют
#     параметры по умолчанию для всех виртуальных узлов.
#  3. Параметры виртуальных узлов. Виртуальные узлы позволяют посылать
#     веб-запросы на разные IP-адреса или имена узлов и обрабатывать их
#     одним и тем же процессом сервера Apache.
#
# Имена конфигурационных файлов и файлов журналов: Если заданные вами имена
# файлов управления сервером начинаются с символа "/" (или "диск:/"
# для Win32), сервер будет использовать этот явно указанный путь. Если
# имена файлов не начинаются с "/", то перед указанным путем будет
# добавлено значение параметра ServerRoot. Так, при ServerRoot, установленном
# в "/usr/local/apache", значение "logs/foo.log" будет интерпретировано
# сервером как "/usr/local/apache/logs/foo.log"
#

### Раздел 1: Общее окружение
#
# Директивы этого раздела определяют общие параметры работы Apache,
# такие, как количество запросов, которые он может обработать одновременно,
# или место расположения конфигурационных файлов.
#

#
# Директива ServerType может принимать значение inetd или standalone.
# Режим inetd поддерживается только на Unix-платформах.
#
ServerType standalone

#
# ServerRoot: Корень дерева каталогов, в котором хранятся файлы
# конфигурации, ошибок и журналов.
#
# ВНИМАНИЕ! Если вы хотите разместить эти файлы на файловой системе NFS
# (или на любой другой сетевой файловой системе), прочтите документацию
# на LockFile.
# (по адресу <URL:http://www.apache.org/docs/mod/core.html#lockfile>)
# Этим вы убережете себя от большого количества проблем!
#
# В конце строки НЕ НУЖНО добавлять завершающий слэш.
#
ServerRoot "/var/apache"

#
```

```
# Директива LockFile устанавливает путь к файлу блокировок, используемому,
# когда Apache скомпилирован с параметром USE_FCNTL_SERIALIZED_ACCEPT или
# USE_FLOCK_SERIALIZED_ACCEPT. Обычно нет необходимости изменять ее
# значение по умолчанию. Однако если каталоги журналов смонтированы на NFS,
# то файл блокировок СЛЕДУЕТ РАЗМЕСТИТЬ НА ЛОКАЛЬНОМ ДИСКЕ. PID главного
# процесса сервера будет автоматически добавлен к указанному имени файла.
#
#LockFile /var/apache/logs/accept.lock

#
# PidFile: Файл, в который сервер при запуске будет записывать свой
# идентификатор процесса.
#
PidFile /var/run/httpd.pid

#
# ScoreBoardFile: Файл, в котором процесс сервера хранит свою внутреннюю
# информацию. Он используется не во всех архитектурах. Но если у вас он
# используется (чтобы об этом узнать – просто посмотрите, создается ли этот
# файл при запуске Apache), тогда следует убедиться, что любые два
# экземпляра Apache используют разные файлы.
#
ScoreBoardFile /var/run/httpd.scoreboard

#
# В стандартной конфигурации сервер сначала обрабатывает данный файл, затем
# файл srm.conf, а за ним файл access.conf. Мы рекомендуем вам для простоты
# держать все директивы сервера в одном файле, поэтому последние два файла
# сейчас поставляются пустыми. Следующие закомментированные значения – это
# встроенные значения по умолчанию. Можно просто заставить сервер
# игнорировать эти файлы, указав в этих директивах путь "/dev/null"
# (для Unix) или "nul" (для Win32)
#
#ResourceConfig /etc/apache/srm.conf
#AccessConfig /etc/apache/access.conf

#
# Timeout: Время ожидания (в секундах), прежде чем сервер отправит или
# получит сообщение о тайм-ауте.
#
Timeout 300

#
# KeepAlive: Указывает, разрешать или нет постоянные соединения (т.е. больше
# одного запроса на соединение). Для запрета установите аргумент в "Off".
#
KeepAlive On

#
# MaxKeepAliveRequests: Максимальное количество запросов, разрешенных в течение
# одного постоянного соединения. Значение 0 соответствует неограниченному
# количеству запросов. Чтобы добиться максимальной производительности, лучше
# оставить это значение достаточно большим.
#
```

```
MaxKeepAliveRequests 100

#
# KeepAliveTimeout: Время ожидания (в секундах) следующего запроса от того же
# клиента по тому же соединению.
#
KeepAliveTimeout 15

#
# Настройка величины серверного пула. Вместо того чтобы заставлять вас
# угадывать, сколько серверных процессов вам понадобится, Apache динамически
# подстраивается под текущую загрузку. То есть он пытается поддерживать
# достаточное количество процессов для обработки текущих запросов плюс еще
# несколько свободных для обработки возможных кратковременных пиковых нагрузок
# (например, несколько одновременных запросов от одного броузера Netscape).
#
# Он осуществляет эту подстройку, отслеживая количество серверных процессов,
# ожидающих запросов (т.е. свободных). Если их меньше, чем MinSpareServers, -
# будут созданы новые. Если их больше, чем MaxSpareServers, - часть из них будет
# остановлена. Значения по умолчанию довольно неплохо подходят для большинства
# сайтов.
#
MinSpareServers 5
MaxSpareServers 10

#
# Количество серверных процессов, которые будут запущены изначально. Значение
# должно быть разумным и соразмерным с двумя предыдущими параметрами.
#
StartServers 5

#
# Максимальное количество одновременно работающих серверов, т. е. максимально
# возможное количество одновременно подключенных клиентов. Если эта величина будет
# достигнута, клиенты будут ЗАБЛОКИРОВАНЫ, поэтому она НЕ ДОЛЖНА БЫТЬ СЛИШКОМ МАЛА.
# Это значение в основном призвано удержать на плаву систему в ситуации, когда
# сервер захлебывается под шквалом запросов.
#
MaxClients 150

#
# MaxRequestsPerChild: Максимальное количество запросов, которое разрешено
# обработать каждому порожденному процессу. После достижения этого числа
# порожденный процесс будет уничтожен. Таким образом решается проблема утечки памяти
# и других ресурсов, возникающая при длительной эксплуатации Apache (и, возможно,
# некоторых его библиотек). Для большинства систем этой проблемы нет, но в некоторых
# существуют заметные утечки в библиотеках. Для подобных систем установите это
# значение примерно в 10 000. Значение 0 соответствует неограниченному количеству
# запросов.
# ПРИМЕЧАНИЕ: При создании постоянного соединения учитывается только первый
#     запрос клиента. То есть если порожденный процесс обработал один запрос,
#     создавший постоянное соединение, а за ним еще 10 - в рамках того же
#     соединения - все это будет засчитано за один запрос.
```

```
#
MaxRequestsPerChild 0

#
# Listen: Позволяет привязать Apache к определенному IP-адресу и/или порту в
# дополнение к заданным по умолчанию. См. также директиву <VirtualHost>.
#
#Listen 3000
#Listen 12.34.56.78:80

#
# BindAddress: Этим параметром обеспечивается поддержка виртуальных узлов. Здесь
# указывается IP-адрес, который серверу необходимо отслеживать. Адрес может быть
# задан символом "*", IP-адресом в десятичной нотации или полным доменным
# именем. См. также директивы <VirtualHost> и Listen.
#
#BindAddress *

#
# Поддержка динамических распределенных объектов (DSO, Dynamic Shared Object)
#
# Чтобы воспользоваться функциональностью модуля, который был собран как DSO,
# здесь необходимо разместить соответствующую директиву LoadModule так, чтобы
# она была обработана до любых директив, содержащихся в модуле. Дополнительную
# информацию о механизме DSO можно прочесть в файле README.DSO из поставки
# Apache 1.3. Список встроенных в httpd модулей (скомпонованных статически и,
# таким образом, всегда доступных) можно получить, выполнив команду 'httpd -l'.
#
# ПРИМЕЧАНИЕ: Порядок, в котором загружаются модули, весьма важен. Не изменяйте его,
# не посоветовавшись со специалистом.
#
# Пример:
# LoadModule foo_module libexec/mod_foo.so
LoadModule vhost_alias_module /usr/apache/libexec/mod_vhost_alias.so
LoadModule env_module         /usr/apache/libexec/mod_env.so
LoadModule config_log_module  /usr/apache/libexec/mod_log_config.so
LoadModule mime_magic_module  /usr/apache/libexec/mod_mime_magic.so
LoadModule mime_module        /usr/apache/libexec/mod_mime.so
LoadModule negotiation_module /usr/apache/libexec/mod_negotiation.so
LoadModule status_module      /usr/apache/libexec/mod_status.so
LoadModule info_module        /usr/apache/libexec/mod_info.so
LoadModule includes_module    /usr/apache/libexec/mod_include.so
LoadModule autoindex_module   /usr/apache/libexec/mod_autoindex.so
LoadModule dir_module         /usr/apache/libexec/mod_dir.so
LoadModule cgi_module         /usr/apache/libexec/mod_cgi.so
LoadModule asis_module        /usr/apache/libexec/mod_asis.so
LoadModule imap_module        /usr/apache/libexec/mod_imap.so
LoadModule action_module      /usr/apache/libexec/mod_actions.so
LoadModule speling_module     /usr/apache/libexec/mod_speling.so
LoadModule userdir_module     /usr/apache/libexec/mod_userdir.so
LoadModule alias_module       /usr/apache/libexec/mod_alias.so
LoadModule rewrite_module     /usr/apache/libexec/mod_rewrite.so
```

```
LoadModule access_module      /usr/apache/libexec/mod_access.so
LoadModule auth_module        /usr/apache/libexec/mod_auth.so
LoadModule anon_auth_module   /usr/apache/libexec/mod_auth_anon.so
LoadModule dbm_auth_module    /usr/apache/libexec/mod_auth_dbm.so
LoadModule digest_module      /usr/apache/libexec/mod_digest.so
LoadModule proxy_module       /usr/apache/libexec/libproxy.so
LoadModule cern_meta_module   /usr/apache/libexec/mod_cern_meta.so
LoadModule expires_module     /usr/apache/libexec/mod_expires.so
LoadModule headers_module     /usr/apache/libexec/mod_headers.so
LoadModule usertrack_module   /usr/apache/libexec/mod_usertrack.so
LoadModule unique_id_module   /usr/apache/libexec/mod_unique_id.so
LoadModule setenvif_module    /usr/apache/libexec/mod_setenvif.so
LoadModule perl_module        /usr/apache/libexec/libperl.so

#  Реконструкция полного списка из всех доступных модулей (как статических, так
#  и распределенных), нужная для исполнения их в корректном порядке.
#  [ЕСЛИ ВЫ ИЗМЕНЯЛИ РАЗДЕЛ LOADMODULE, ВНЕСИТЕ ЗДЕСЬ СООТВЕТСТВУЮЩИЕ ИЗМЕНЕНИЯ]
ClearModuleList
AddModule mod_vhost_alias.c
AddModule mod_env.c
AddModule mod_log_config.c
AddModule mod_mime_magic.c
AddModule mod_mime.c
AddModule mod_negotiation.c
AddModule mod_status.c
AddModule mod_info.c
AddModule mod_include.c
AddModule mod_autoindex.c
AddModule mod_dir.c
AddModule mod_cgi.c
AddModule mod_asis.c
AddModule mod_imap.c
AddModule mod_actions.c
AddModule mod_speling.c
AddModule mod_userdir.c
AddModule mod_alias.c
AddModule mod_rewrite.c
AddModule mod_access.c
AddModule mod_auth.c
AddModule mod_auth_anon.c
AddModule mod_auth_dbm.c
AddModule mod_digest.c
AddModule mod_proxy.c
AddModule mod_cern_meta.c
AddModule mod_expires.c
AddModule mod_headers.c
AddModule mod_usertrack.c
AddModule mod_unique_id.c
AddModule mod_so.c
AddModule mod_setenvif.c
AddModule mod_perl.c
```

```
#
# Директива ExtendedStatus определяет, будет ли Apache при вызове обработчика
# "server-status" выдавать "полную" (ExtendedStatus On) или только основную
# информацию о состоянии(ExtendedStatus Off). Значение по умолчанию - Off.
#
#ExtendedStatus On

### Раздел 2: Конфигурация сервера по умолчанию
#
# Директивы этого раздела устанавливают значения, используемые 'главным'
# сервером, который отвечает на все запросы, не обработанные виртуальными узлами
# (заданными директивами <VirtualHost>). Также эти значения выступают в
# качестве значений по умолчанию для всех контейнеров <VirtualHost>,
# определенных ниже в этом файле.
#
# Все эти директивы могут использоваться внутри контейнеров <VirtualHost>;
# в этом случае для определяемого виртуального узла значения по умолчанию будут
# заменены новыми.
#

#
# Если директива ServerType (расположенная выше в разделе 'Общее окружение')
# была выставлена в значение "inetd", то несколько следующих директив не будут
# использоваться, поскольку их значения уже определены конфигурацией демона
# inetd. В этом случае переходите сразу к директиве ServerAdmin
#

#
# Port: Порт, который слушает автономный сервер. Чтобы использовать порт с
# номером < 1023, необходимо, чтобы httpd изначально был запущен от имени
# пользователя root.
#
Port 80

#
# Если вам нужно, чтобы httpd запускался от имени другого пользователя или
# группы, все равно необходимо запустить его от имени пользователя root. Позже
# он сам переключится на другую учетную запись.
#
# User/Group: Имя (или #номер) пользователя/группы, от имени которого
# запускается httpd.
#   . На SCO (ODT 3) используйте "User nouser" и "Group nogroup".
#   . На HPUX нельзя использовать распределенную память от имени пользователя
#     nobody. Это можно обойти, воспользовавшись специально созданным
#     пользователем www.
#
# ПРИМЕЧАНИЕ: Некоторые ядра не выполняют setgid(Group) или semctl(IPC_SET),
# когда значение (unsigned)Group больше 60 000.
# Не используйте директиву Group #-1 на таких системах!
#
User nobody
Group nobody
```

```
#
# ServerAdmin: Ваш адрес электронной почты, на который будут отправляться
# сообщения о проблемах с сервером. Этот адрес появится на некоторых созданных
# сервером страницах, таких как страницы с сообщениями об ошибках.
#
ServerAdmin you@your.address

#
# Директива ServerName позволяет установить имя узла, возвращаемое клиентам,
# если оно отличается от того, которое получила программа (например, используйте
# "www" вместо настоящего имени узла).
#
# ВНИМАНИЕ: Нельзя просто придумать имя узла и использовать его здесь с
# надеждой, что все заработает. Определенное здесь имя должно быть правильным
# DNS-именем узла. Если вы этого не понимаете, обратитесь к администратору сети.
# Если у вашего узла нет зарегистрированного DNS-имени, укажите здесь его
# IP-адрес. При этом для доступа к узлу придется использовать именно его
# (например, http://123.45.67.89), и это может усложнить переадресацию ресурсов.
#
#ServerName new.host.name

#
# DocumentRoot: Каталог, в котором будут находиться ваши документы. По умолчанию
# все запросы обращаются к документам из этого каталога. Для доступа к другим
# местам могут использоваться символьные ссылки и псевдонимы.
#
DocumentRoot "/var/apache/htdocs"

#
# Свойства и сервисы могут быть разрешены и/или запрещены для каждого каталога
# (и его подкаталогов), к которому у Apache есть доступ.
#
# Сначала зададим очень ограниченный набор разрешений в качестве значений по
# умолчанию.
#
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>

#
# Обратите внимание, что начиная отсюда вам необходимо явно разрешать нужные
# свойства. Поэтому если что-то не работает так, как вам нужно, убедитесь, что
# где-то ниже это “что-то” разрешено.
#

#
# Здесь нужно указать каталог, который был указан в директиве DocumentRoot.
#
<Directory "/var/apache/htdocs">

#
# Здесь могут использоваться значения "None" (нет разрешенных опций), "All" (все
# опции разрешены) или любые комбинации из "Indexes", "Includes",
```

```
# "FollowSymLinks", "ExecCGI" и "MultiViews".
#
# Обратите внимание, что опция "MultiViews" должна быть указана явно –
# директива "Options All" не включает эту опцию.
#
    Options Indexes FollowSymLinks

#
# Следующая директива указывает, какие опции могут быть переназначены из файла
# .htaccess текущего каталога. Также может принимать значение "All" или любые
# комбинации из "Options", "FileInfo", "AuthConfig" и "Limit".
#
    AllowOverride None

#
# Указывает, кто может получать что-либо с этого сервера.
#
    Order allow,deny
    Allow from all
</Directory>

#
# UserDir: Имя каталога, которое добавляется к исходному каталогу пользователя
# при получении запроса вида "~имя_пользователя".
#
UserDir public_html

#
# Управляет доступом к каталогам UserDir. Ниже приведен пример сайта, где эти
# каталоги разрешены только для чтения.
#
#<Directory /home/*/public_html>
#    AllowOverride FileInfo AuthConfig Limit
#    Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
#    <Limit GET POST OPTIONS PROPFIND>
#        Order allow,deny
#        Allow from all
#    </Limit>
#    <Limit PUT DELETE PATCH PROPPATCH MKCOL COPY MOVE LOCK UNLOCK>
#        Order deny,allow
#        Deny from all
#    </Limit>
#</Directory>

#
# DirectoryIndex: Имя файла или файлов, используемых в качестве страницы-индекса
# каталога (страницы по умолчанию). Если нужно указать несколько значений –
# перечисляйте их через пробел.
#
DirectoryIndex index.html

#
# AccessFileName: Имя файла, используемого для хранения информации управления
# доступом к текущему каталогу.
```

```
#
AccessFileName .htaccess

#
# Следующие строки скрывают файлы .htaccess от просмотра веб-клиентами. Доступ к ним
# запрещается из соображений безопасности, поскольку файлы .htaccess часто содержат
# авторизационную информацию. Закомментируйте эти строки, если нужно, чтобы
# веб-посетители могли увидеть содержимое этих файлов. Если значение предыдущей
# директивы AccessFileName было изменено, внесите соответствующие изменения
# и здесь.
#
# Файлы с именами вроде .htpasswd часто содержат пароли. Поэтому следующие
# строки скрывают и подобные файлы тоже.
#
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>

#
# CacheNegotiatedDocs: По умолчанию Apache посылает заголовок "Pragma: no-cache"
# с каждым документом, использующим механизм динамического определения содержимого
# (content negotiation). Этот заголовок предписывает прокси-серверам не кэшировать
# этот документ. Если раскомментировать следующую строку, заголовок не будет
# посылаться с подобными документами и прокси-серверы получат возможность
# их кэшировать.
#
#CacheNegotiatedDocs

#
# UseCanonicalNames: (новая для версии 1.3) Если эта директива включена, каждый
# раз при построении URL, ссылающегося на самого себя (URL, указывающий обратно
# на сервер, от которого пришел ответ), Apache будет использовать значения
# ServerName и Port для формирования "канонического" имени. Когда эта директива
# выключена, Apache по возможности будет использовать предоставленное клиентом
# имя сервера и номер порта. Значение этой директивы также влияет на значения
# серверных переменных SERVER_NAME и SERVER_PORT в CGI-сценариях.
#
UseCanonicalName On

#
# TypesConfig указывает каталог, в котором хранятся файлы mime.types (или их
# эквиваленты).
#
TypesConfig /etc/apache/mime.types

#
# Директива DefaultType устанавливает MIME-тип по умолчанию, который будет
# использоваться сервером при невозможности определить тип по таким признакам,
# как, например, расширение имени файла. Если сервер содержит в основном тексты
# или HTML-документы, значение "text/plain" будет неплохим компромиссом. Если
# большая часть содержимого – исполнимые файлы, стоит установить значение в
# "application/octet-stream", чтобы броузеры не пытались отображать исполнимые
```

```
# файлы в виде текста.
#
DefaultType text/plain

#
# Модуль mod_mime_magic позволяет серверу воспользоваться подсказками из
# содержимого самого файла для определения его MIME-типа. Директива
# MIMEMagicFile указывает месторасположение файла определений подсказок.
# mod_mime_magic не входит в стандартную конфигурацию (его необходимо добавить
# вручную с помощью LoadModule [см. абзац, посвященный DSO, в разделе "Глобальное
# окружение"] или заново собрать сервер, включив в конфигурацию mod_mime_magic),
# поэтому эта директива заключена в контейнер <IfModule>. Это означает, что
# она будет обработана, только если модуль входит в состав сервера.
#
<IfModule mod_mime_magic.c>
    MIMEMagicFile /etc/apache/magic
</IfModule>

#
# HostnameLookup: Определяет, записывать в журнал имена узлов клиентов или
# только их IP-адреса, например www.apache.org (при включенной, On) или
# 204.62.129.132 (при выключенной, Off). По умолчанию эта директива выключена,
# поскольку для сети в целом будет лучше, если эта возможность будет включаться
# только при необходимости, ведь во включенном состоянии каждый запрос клиента
# будет порождать КАК МИНИМУМ один запрос к серверу имен.
#
HostnameLookups Off

#
# ErrorLog: Месторасположение файла журнала ошибок.
# Если в контейнере <VirtualHost> нет директивы ErrorLog, все сообщения об
# ошибках для этого виртуального узла будут записываться здесь. Если файл
# журнала ошибок определен в контейнере <VirtualHost>, сообщения об ошибках для
# виртуального узла будут записываться в его файле журнала, а не здесь.
#
ErrorLog /var/apache/logs/error_log

#
# LogLevel: Управляет числом сообщений об ошибках, записываемых в журнал ошибок.
# Возможные значения: debug, info, notice, warn, error, crit, alert, emerg.
#
LogLevel warn

#
# Следующие директивы определяют несколько именованных форматирующих строк для
# использования в директиве CustomLog (см. ниже).
#
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent

#
# Месторасположение и формат файла журнала запросов (обычный формат
```

```
# журнала). Если файл журнала запросов не определен в контейнере <VirtualHost>,
# то соответствующие записи будут делаться здесь. И наоборот, если файл журнала
# запросов определен в контейнере <VirtualHost>, транзакции будут записываться
# в нем, а не здесь.
#
CustomLog /var/apache/logs/access_log common

#
# Если вы хотите вести журналы значений referer и agent, раскомментируйте
# следующие директивы.
#
#CustomLog /var/apache/logs/referer_log referer
#CustomLog /var/apache/logs/agent_log agent

#
# Если вы предпочитаете вести единый журнал, в который будет записываться
# и информация о запросах, и значения agent и referer (комбинированный формат
# журнала), – воспользуйтесь следующей директивой:
#
#CustomLog /var/apache/logs/access_log combined

#
# Ко всем документам, созданным сервером (документы с сообщениями об ошибках,
# списки файлов в FTP-каталогах, вывод модулей mod_status и mod_info и т. п.,
# но не страницы, созданные CGI-сценариями), можно добавить строку, содержащую
# версию сервера и имя виртуального узла. Значение "EMail" также добавит ссылку
# mailto: на почтовый адрес, указанный в ServerAdmin.
# Возможные значения директивы: On | Off | EMail
#
ServerSignature On

#
# Aliases: Здесь можно добавить столько псевдонимов, сколько вам нужно
# (ограничений нет). Формат: Alias псевдоним настоящее_имя
#
# Обратите внимание, что если вы добавите к псевдониму завершающий /,
# сервер будет проверять его наличие в URL. Поэтому "/icons" не будет являться
# псевдонимом в следующем примере, а только "/icons/".
#
Alias /icons/ "/var/apache/icons/"

<Directory "/var/apache/icons">
    Options Indexes MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>

Alias /manual/ "/usr/apache/htdocs/manual/"

#
# ScriptAlias: Указывает на каталог, содержащий серверные сценарии. По сути, эта
# директива – то же самое, что и Alias, но документы, расположенные в папке, скрытой
# псевдонимом, обрабатываются сервером как приложения. При поступлении к ним запроса
```

```
# сервер запускает их на исполнение, вместо того чтобы передать клиенту как обычный
# документ. Для ScriptAlias, как и для Alias, действует правило завершающего "/".
#
ScriptAlias /cgi-bin/ "/var/apache/cgi-bin/"

#
# "/var/apache/cgi-bin" нужно заменить на путь, который вы указали для
# ScriptAlias. Если, конечно, вы его указали.
#
<Directory "/var/apache/cgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>

#
# Перенаправления позволяют уведомить клиентов о документах, которые раньше
# существовали в пространстве имен вашего сервера, но теперь отсутствуют.
# Это позволяет сообщить клиенту новый адрес перемещенного документа.
# Формат: Redirect старый_URI новый_URL
#

#
# Директивы, управляющие внешним видом создаваемых сервером листингов каталогов.
#

#
# FancyIndexing указывает, следует ли создавать "красивый" индекс каталогов или
# стандартный.
#
IndexOptions FancyIndexing

#
# Директивы AddIcon* сообщают серверу, какие значки следует использовать для
# отображения разных файлов или расширений их имен. Значки используются только
# для отображения "красивых" каталогов (при включенной директиве FancyIndexing).
#
AddIconByEncoding (CMP,/icons/compressed.gif) x-compress x-gzip

AddIconByType (TXT,/icons/text.gif) text/*
AddIconByType (IMG,/icons/image2.gif) image/*
AddIconByType (SND,/icons/sound2.gif) audio/*
AddIconByType (VID,/icons/movie.gif) video/*

AddIcon /icons/binary.gif .bin .exe
AddIcon /icons/binhex.gif .hqx
AddIcon /icons/tar.gif .tar
AddIcon /icons/world2.gif .wrl .wrl.gz .vrml .vrm .iv
AddIcon /icons/compressed.gif .Z .z .tgz .gz .zip
AddIcon /icons/a.gif .ps .ai .eps
AddIcon /icons/layout.gif .html .shtml .htm .pdf
AddIcon /icons/text.gif .txt
AddIcon /icons/c.gif .c
```

```
AddIcon /icons/p.gif .pl .py
AddIcon /icons/f.gif .for
AddIcon /icons/dvi.gif .dvi
AddIcon /icons/uuencoded.gif .uu
AddIcon /icons/script.gif .conf .sh .shar .csh .ksh .tcl
AddIcon /icons/tex.gif .tex
AddIcon /icons/bomb.gif core

AddIcon /icons/back.gif ..
AddIcon /icons/hand.right.gif README
AddIcon /icons/folder.gif ^^DIRECTORY^^
AddIcon /icons/blank.gif ^^BLANKICON^^

#
# DefaultIcon указывает значок, применяемый для тех файлов, значок для которых
# не указан.
#
DefaultIcon /icons/unknown.gif

#
# Директива AddDescription позволяет разместить короткое описание после имени
# файла в созданном сервером индексе каталога. Описание отображается только для
# "красивых" каталогов.
# Формат: AddDescription "описание" имя_файла
#
#AddDescription "GZIP compressed document" .gz
#AddDescription "tar archive" .tar
#AddDescription "GZIP compressed tar archive" .tgz

#
# Директива ReadmeName указывает имя файла, содержимое которого сервер будет
# добавлять к концу индекса каталога.
#
# Директива HeaderName указывает имя файла, содержимое которого сервер будет
# выводить перед индексом каталога.
#
# Сервер будет сначала искать файл с именем имя.html и использовать его
# содержимое, если он найдется. Если файл имя.html не существует, тогда файл
# будет искать файл имя.txt и, если найдет, вставит его содержимое как обычный
# текст.
#
ReadmeName README
HeaderName HEADER

#
# Директива IndexIgnore определяет список имен файлов, которые не должны
# включаться в индекс каталога. При перечислении имен можно использовать
# подстановочные символы в стиле оболочки.
#
IndexIgnore .??* *~ *# HEADER* README* RCS CVS *,v *,t

#
# Директива AddEncoding позволяет заставить некоторые броузеры (Mosaic/X 2.1+)
# распаковывать информацию "на лету". Внимание: не все броузеры поддерживают
```

```
# это. Несмотря на схожесть имен нижеследующие директивы Add* не имеют никакого
# отношения к настройкам отображения индекса каталогов.
#
AddEncoding x-compress Z
AddEncoding x-gzip gz tgz

#
# Директива AddLanguage позволяет задать язык документа. После этого можно
# использовать механизм динамического определения содержимого для передачи
# броузеру файла на том языке, который он понимает. Обратите внимание, что
# расширения совсем не обязательно должны соответствовать ключевым словам,
# определяющим язык. Например, документ на польском языке (которому по сетевым
# стандартам соответствует код pl) может определяться директивой
# "AddLanguage pl .po", избегая, тем самым, конфликта с распространенным
# расширением для сценариев на языке Perl.
#
AddLanguage en .en
AddLanguage fr .fr
AddLanguage de .de
AddLanguage da .da
AddLanguage el .el
AddLanguage it .it

#
# Директива LanguagePriority позволяет задать приоритеты языков относительно их
# выбора при динамическом определении содержимого. Просто перечислите языки в
# порядке убывания их приоритета.
#
LanguagePriority en fr de

#
# Директива AddType позволяет подкорректировать файл mime.types без его
# непосредственного редактирования или задать определенный тип некоторым файлам.
#
# Например, модуль PHP3 (не являющийся частью поставки Apache,
# см. http://www.php.net) обычно использует следующие директивы:
#
#AddType application/x-httpd-php3 .php3
#AddType application/x-httpd-php3-source .phps

AddType application/x-tar .tgz

#
# AddHandler позволяет назначить соответствие между расширениями имен файлов и
# "обработчиками", вне зависимости от mime-типа файла. Обработчики могут быть
# как встроенными в сервер, так и добавленными командой Action (см. ниже).
#
# Если нужно использовать серверные включения (SSI) или CGI-сценарии за
# пределами каталогов ScriptAlias, раскомментируйте следующие строки.
#
# Для использования CGI-сценариев:
#
#AddHandler cgi-script .cgi
```

```
#
# Для использования серверных включений (SSI):
#
#AddType text/html .shtml
#AddHandler server-parsed .shtml

#
# Раскомментируйте следующие строки, чтобы разрешить отправку HTTP-файлов
# (файлов, содержащих свои HTTP-заголовки) "как есть".
#
#AddHandler send-as-is asis

#
# Если вы хотите использовать изображения-карты на стороне сервера,
# воспользуйтесь следующей строкой:
#
#AddHandler imap-file map

#
# Для разрешения карты типов можно воспользоваться следующей директивой:
#
#AddHandler type-map var

#
# Директива Action позволяет определить MIME-типы, которые будут исполнять
# сценарий при вызове соответствующего файла. Она позволяет избавиться от
# необходимости постоянно указывать пути к часто используемым CGI-обработчикам.
# Формат: Action mime-тип /путь/к/cgi-обработчику
# Формат: Action обработчик /путь/к/cgi-обработчику
#

#
# MetaDir: задает имя каталога, в котором Apache может найти файлы с мета-
# информацией. Эти файлы содержат дополнительные HTTP-заголовки, добавляемые
# при отправке документа.
#
#MetaDir .web

#
# MetaSuffix: указывает расширение для имен файлов, содержащих мета-информацию.
#
#MetaSuffix .meta

#
# Настраиваемые сообщения об ошибках (в стиле Apache) бывают трех типов.
#
#    1) обычный текст
#ErrorDocument 500 "Сервер сделал что-то страшное."
#  Важно:  Кавычки используются для выделения строки текста и не попадают в вывод.
#
#    2) локальные перенаправления
#ErrorDocument 404 /missing.html
#  перенаправит к локальному URL /missing.html
#ErrorDocument 404 /cgi-bin/missing_handler.pl
```

```
#  Важно: Можно перенаправить к сценарию или к документу, использующему
#         серверные включения.
#
#    3) внешние перенаправления
#ErrorDocument 402 http://другой.сервер.com/subscription_info.html
#  Важно: Многие переменные окружения, связанные с исходным запросом, будут
#         не доступны при внешнем перенаправлении.

#
# Следующие директивы изменяют обычное поведение сервера при формировании HTTP-
# ответа.Первая директива запрещает запросы keepalive для Netscape 2.x и других
# броузеров, которые не понимают их. Это известная проблема с этими броузерами.
# Вторая директива предназначена для Microsoft Internet Explorer 4.0b2,
# в котором присутствует ошибка в реализации HTTP/1.1 и отсутствует поддержка
# запросов keepalive при ответах сервера 301 или 302 (перенаправления).
#
BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0

#
# Следующая директива отключает ответы HTTP/1.1 для броузеров, которые нарушают
# спецификацию HTTP/1.0 тем, что не могут разобрать основные ответы 1.1.
#
BrowserMatch "RealPlayer 4\.0" force-response-1.0
BrowserMatch "Java/1\.0" force-response-1.0
BrowserMatch "JDK/1\.0" force-response-1.0

#
# Разрешает вывод отчетов о состоянии при запросе к URL
# http://servername/server-status
# Замените ".your_domain.com" на ваш домен, чтобы разрешить с него доступ.
#
#<Location /server-status>
#    SetHandler server-status
#    Order deny,allow
#    Deny from all
#    Allow from .your_domain.com
#</Location>

#
# Разрешает вывод отчетов о настройках при запросах к URL
# http://servername/server-info (необходимо, чтобы модуль mod_info.c был
# загружен).
# Замените ".your_domain.com" на ваш домен, чтобы разрешить с него доступ.
#
#<Location /server-info>
#    SetHandler server-info
#    Order deny,allow
#    Deny from all
#    Allow from .your_domain.com
#</Location>

#
```

```
# Ходят слухи о людях, пытающихся эксплуатировать старую ошибку, которая была
# исправлена еще в Apache 1.1. Эта ошибка использовала CGI-сценарий,
# поставляемый в составе Apache. Раскомментировав следующие строки, можно
# перенаправить такие атаки прямо в журналирующий сценарий на phf.apache.org.
# Также вы можете записать подобные атаки сами, воспользовавшись сценарием
# support/phf_abuse_log.cgi.
#
#<Location /cgi-bin/phf*>
#    Deny from all
#    ErrorDocument 403 http://phf.apache.org/phf_abuse_log.cgi
#</Location>

#
# Директивы прокси-сервера. Раскомментируйте следующие строки, чтобы включить
# прокси-сервер.
#
#<IfModule mod_proxy.c>
#ProxyRequests On
#
#<Directory proxy:*>
#    Order deny,allow
#    Deny from all
#    Allow from .your_domain.com
#</Directory>

#
# Включить/выключить обработку заголовков HTTP/1.1 "Via:".
# Значение "Full" добавляет версию сервера, "Block" удаляет исходящие заголовки
# "Via:".
# Установите одно из следующих значений: Off | On | Full | Block
#
#ProxyVia On

#
# Отредактируйте и раскомментируйте следующие строки, чтобы разрешить
# кэширование:
# (с неустановленной директивой CacheRoot кэширование отключено)
#
#CacheRoot "/var/apache/proxy"
#CacheSize 5
#CacheGcInterval 4
#CacheMaxExpire 24
#CacheLastModifiedFactor 0.1
#CacheDefaultExpire 1
#NoCache a_domain.com another_domain.edu joes.garage_sale.com

#</IfModule>
# Конец директив прокси-сервера.

### Раздел 3: Виртуальные узлы
#
# VirtualHost: Если вы хотите поддерживать несколько доменов/имен узлов на одной
# машине, необходимо создать для них контейнеры VirtualHost.
```

```
# Перед установкой виртуальных узлов прочтите документацию, расположенную по
# адресу <URL:http://www.apache.org/docs/vhosts/>.
# Для проверки конфигурации виртуальных узлов можно воспользоваться ключом
# командной строки `-S`.

#
# Чтобы использовать виртуальные узлы на основе имен, необходимо определить хотя
# бы один IP-адрес (и номер порта) для них.
#
#NameVirtualHost 12.34.56.78:80
#NameVirtualHost 12.34.56.78

#
# Пример использования VirtualHost:
# Почти все директивы Apache могут использоваться в контейнерах VirtualHost.
#
#<VirtualHost ip.address.of.host.some_domain.com>
#    ServerAdmin webmaster@host.some_domain.com
#    DocumentRoot /www/docs/host.some_domain.com
#    ServerName host.some_domain.com
#    ErrorLog logs/host.some_domain.com-error_log
#    CustomLog logs/host.some_domain.com-access_log common
#</VirtualHost>

#<VirtualHost _default_:*>
#</VirtualHost>

#<IfModule mod_perl.c>
#
#<Location /perl-status>
#    SetHandler perl-script
#    PerlHandler Apache::Status
#    order deny,allow
#    deny from all
#    allow from yourhost
#</Location>
#
#</IfModule>
```

G

Выдержки из RFC

В главе 13 упоминаются определенные заголовки TCP/IP, которые представлены здесь. Перечень заголовков не полон, он включает лишь заголовки, затронутые в примерах главы 13:

- Заголовок дейтаграммы IP, согласно определению RFC 791, *Internet Protocol*
- Заголовок сегмента TCP, согласно определению RFC 793, *Transmission Control Protocol*
- ICMP-заголовок сообщения об ошибочном параметре, согласно определению RFC 792, *Internet Control Message Protocol*

Каждый заголовок представлен выдержкой из RFC, содержащей определение. Цитаты не являются точными, выдержки слегка отредактированы и приведены в соответствие с текстом книги. Однако важность использования первичных источников при отладке проблем, связанных с протоколами, остается очевидной. Заголовки в настоящем приложении призваны проиллюстрировать примеры из главы 13. В реальной ситуации используйте настоящие документы RFC. Инструкции, поясняющие, как получить документы RFC, даны в конце этого приложения.

Заголовок дейтаграммы IP

Следующее описание приводится на страницах 11–15 документа RFC 791, *Internet Protocol*.

```
Формат заголовка Internet

    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

Версия: 4 бит

Поле версии определяет формат internet-заголовка.
В настоящем документе описана версия 4.

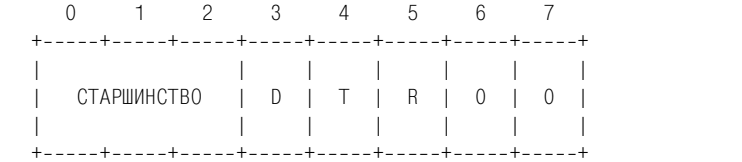
IHL: 4 бит

Длина заголовка Internet (Internet Header Length) – это длина заголовка в 32-битных словах. Минимальное значение для корректного заголовка – 5.

Тип службы: 8 бит

Указание желаемого качества службы.
Значение битов приводится далее.

Биты 0-2: Старшинство.
Бит 3: 0 = Нормальная задержка, 1 = Небольшая задержка.
Бит 4: 0 = Нормальная производительность, 1 = Высокая производительность.
Бит 5: 0 = Нормальная надежность, 1 = Высокая надежность.
Биты 6-7: Зарезервированы для использования в будущем.



Старшинство

111 – Сетевой контроль
110 – Межсетевой контроль
101 – CRITIC/ECP
100 – Скоростная молния
011 – Молния
010 – Немедленная
001 – Приоритетная
000 – Обычная

Общая длина: 16 бит

Общая длина – это длина дейтаграммы, измеренная в октетах (байтах), включающая длину заголовка и данных.

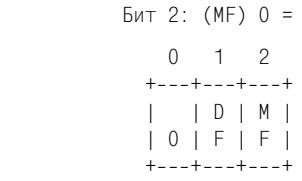
Идентификация: 16 бит

Значение-идентификатор, назначенное отправителем, чтобы упростить сборку фрагментов дейтаграммы.

Флаги: 3 бит

Различные управляющие флаги. Интерпретируются следующим образом:

Бит 0: зарезервирован, обязательно нулевое значение
Бит 1: (DF) 0 = Можно фрагментировать, 1 = Нельзя фрагментировать.
Бит 2: (MF) 0 = Последний фрагмент, 1 = Есть еще фрагменты.



Смещение фрагмента: 13 бит

Данное поле показывает, какому участку дейтаграммы принадлежит фрагмент. Смещение фрагмента измеряется в единицах по 8 октетов (64 бит). Первый фрагмент имеет нулевое смещение.

Время жизни: 8 бит

Данное поле указывает максимальное допустимое время пребывания дейтаграммы в internet-системе.

Протокол: 8 бит

Поле указывает протокол транспортного уровня, которому передаются данные дейтаграммы. Значения для различных протоколов перечислены в документе RFC "Assigned Numbers".

Контрольная сумма заголовка: 16 бит

Контрольная сумма только заголовка. Поскольку некоторые из полей заголовка изменяются (к примеру, время жизни), поле повторно вычисляется и проверяется в каждой точке обработки internet-заголовка. Алгоритм вычисления контрольной суммы:

Поле контрольной суммы является 16-битным дополнением суммы дополнений всех 16-битных слов заголовка. При вычислении контрольной суммы значение в поле контрольной суммы принимается равным нулю.

Адрес источника: 32 бит

Исходный IP-адрес. Описание IP-адресов содержится в главе 2.

Адрес получателя: 32 бит

Конечный IP-адрес. Описание IP-адресов содержится в главе 2.

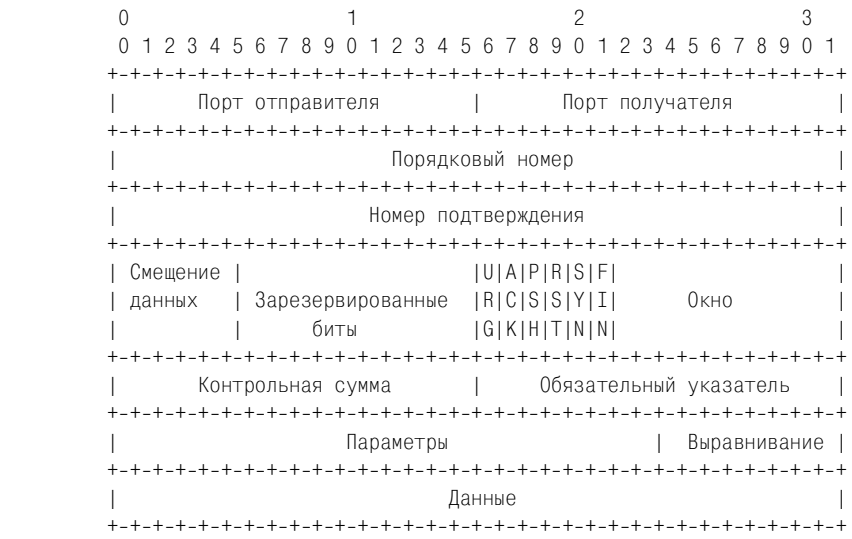
Параметры: переменная длина

Параметры могут присутствовать или отсутствовать в дейтаграмме, но обязательно должны быть реализованы всеми модулями IP (как узлами, так и шлюзами). Ни в одной из дейтаграмм, рассмотренных в главе 13, параметры не использовались.

Заголовок сегмента TCP

Следующее описание приводится на страницах 15–17 документа RFC 793, *Transmission Control Protocol*.

Формат заголовка TCP



Порт источника: 16 бит

Номер порта источника.

Порт получателя: 16 бит

Номер порта получателя.

Порядковый номер: 32 бит

Порядковый номер первого октета данных (байта) данного сегмента (за исключением случаев, когда присутствует SYN). В присутствии SYN номер последовательности является исходным порядковым номером (ISN), а первый октет данных – ISN+1.

Номер подтверждения: 32 бит

Если установлен управляющий бит ACK, это поле содержит значение следующего порядкового номера, ожидаемого отправителем этого сегмента. После установления соединения это значение передается всегда.

Смещение данных: 4 бит

Число 32-битных слов в заголовке TCP. Указывает координату начала данных. Заголовок TCP (даже включающий параметры) имеет целочисленный размер, измеряемый в 32-битных словах.

Зарезервированные биты: 6 бит

```
  Зарезервированы для использования в будущем. Должны быть сброшены.

Управляющие биты:  6 однобитных значений (слева направо):

  URG:  Поле указателя срочных данных актуально
  ACK:  Поле подтверждения актуально
  PSH:  Функция Push
  RST:  Сброс соединения
  SYN:  Синхронизация порядковых номеров
  FIN:  Все данные переданы отправителем

Окно:  16 бит

  Число октетов данных (байтов), которое может принять отправитель этого сегмента.

Контрольная сумма:  16 бит

  Поле контрольной суммы является 16-битным дополнением суммы дополнений
  всех 16-битных слов заголовка и текста.

Обязательный указатель:  16 бит

  Данное поле содержит текущее значение указателя важных данных в виде
  положительного смещения относительно порядкового номера этого сегмента.
  Указатель важных данных указывает на порядковый номер октета, следующего
  за областью важных данных. Поле интерпретируется только в сегментах с
  установленным управляющим битом URG.

Параметры:  переменная длина

  Параметры могут присутствовать в конце заголовка TCP;
  длина блока параметров должна быть кратна 8 битам.
```

ICMP–заголовок сообщения об ошибочном параметре

Следующее описание приводится на страницах 8 и 9 документа RFC 792, *Internet Control Message Protocol*.

```
Сообщение об ошибочном параметре

   0                   1                   2                   3
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |     Тип       |     Код       |     Контрольная сумма         |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |   Указатель   |                не используется                |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |  Заголовок Internet + 64 бита исходной дейтаграммы с данными  |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Тип

   12
```

```
Код

   0 = Указатель определяет ошибку.

Контрольная сумма

   Контрольная сумма – 16-битное дополнение суммы дополнений ICMP-сообщения,
   начиная с поля Тип. При вычислении контрольной суммы
   значение в поле должно быть нулевым.

Указатель

   При нулевом коде указывает на октет, в котором была обнаружена ошибка.

Заголовок Internet + 64 бита исходной дейтаграммы с данными

   internet-заголовок плюс первые 64 бита дейтаграммы, послужившей
   причиной отправки данного сообщения об ошибке.
```

Доступ к документам RFC

Текст книги содержит многочисленные ссылки на документы RFC. Эти документы посвящены сети Интернет и содержат как общую информацию, так и определения стандартов протоколов TCP/IP. Сетевым администраторам обязательно следует обратить внимание на ряд важных документов RFC. В этом разделе рассказано, как получить эти документы.

Документы RFC доступны по адресу *http://www.ietf.org*. С главной страницы перейдите по ссылке RFC Pages. Здесь можно получить RFC по номеру документа. Кроме того, на странице присутствуют ссылки на указатель RFC и сайт RFC Editor Web Pages. Указатель полезен в общем случае. Он позволяет связать названия документов с их номерами, а также содержит сведения, отмечающие обновления и замены документов. Рисунок G.1 иллюстрирует поиск документа RFC 1122 в указателе, выполняемый сетевым администратором.

Еще больший интерес представляют веб-страницы RFC Editor. Переход по этой ссылке приводит по адресу *http://www.rfc-editor.org*, где возможен поиск документов RFC. Страница поиска предоставляет доступ к указателю RFC, содержащему перекрестные ссылки, а также к инструменту поиска, позволяющему находить документы по названиям, номерам, авторам либо ключевым словам.

Предположим, необходимо получить информацию о расширениях SMTP, предложенных для механизма Extended SMTP. На рис. G.2 приведена первая страница с результатами поиска по этому запросу.

Среда Web предоставляет наиболее распространенный, лучший способ поиска документов RFC. В случае, когда известен предмет поиска, гораздо быстрее можно получить документ анонимным доступом на FTP-сервер. Документы RFC хранятся на сервере *ftp.ietf.org* в каталоге *rfc*. Имена файлов RFC имеют формат *rfcnnnn.txt* или *rfcnnnn.ps*, где *nnnn* – номер RFC, а расширение *txt* или *ps* указывает на формат файла – ASCII-текст или PostScript. Чтобы получить документ RFC 1122, отправляйтесь на сервер *ftp.ietf.org* и набе-

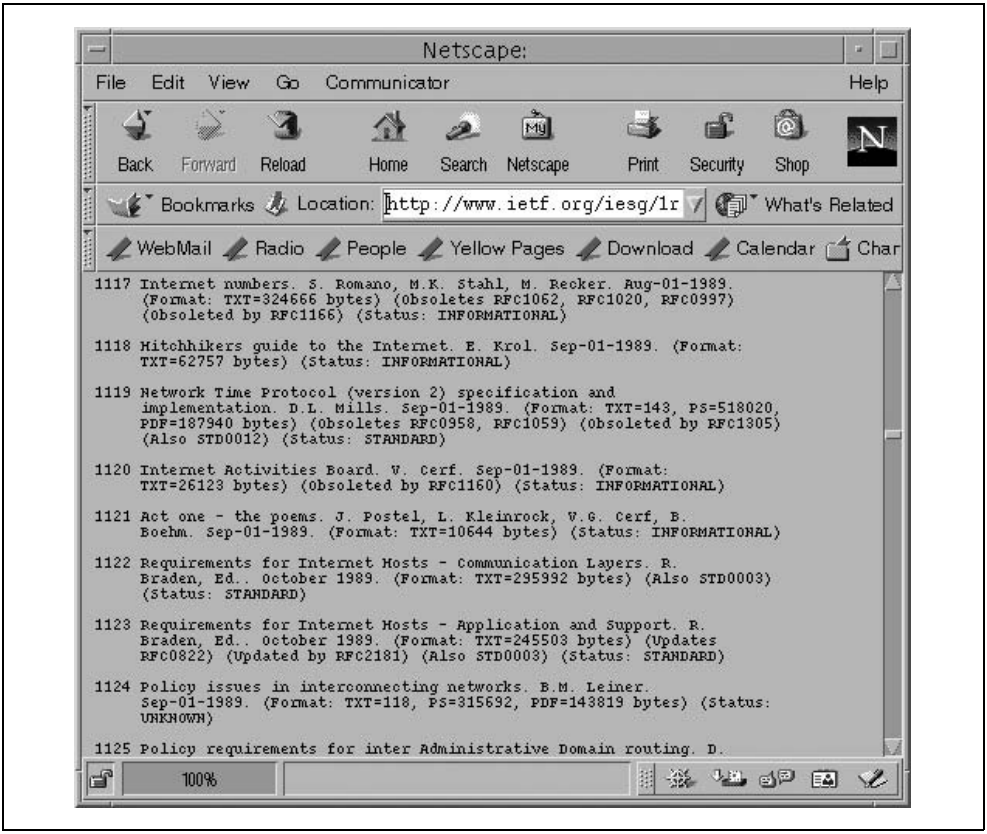


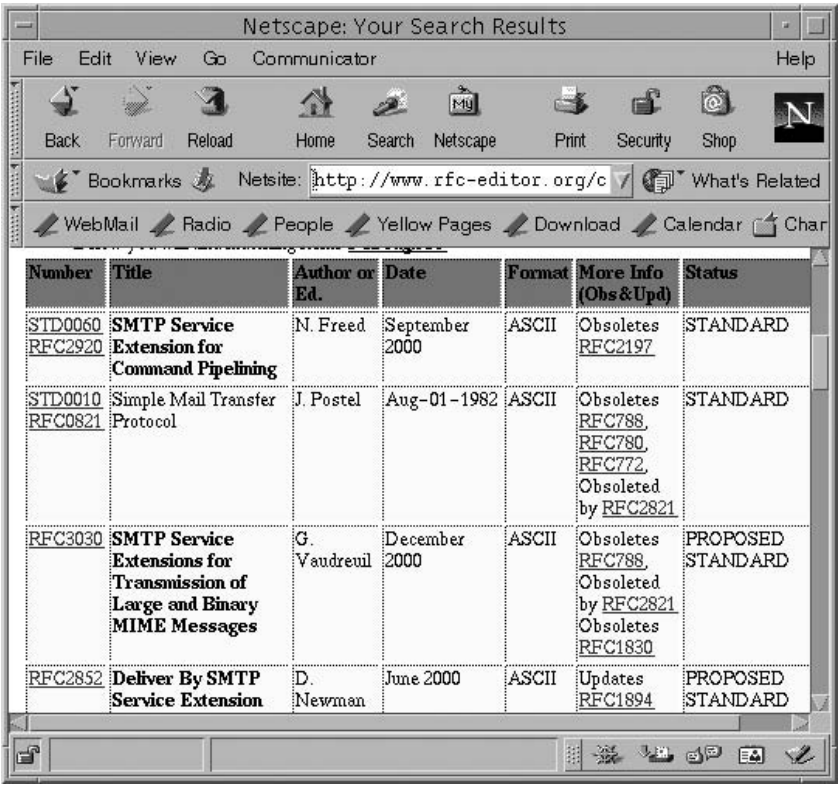
Рис. G.1. *Указатель RFC*

рите `get rfc/rfc1122.txt` в ответ на приглашение `ftp>`. Обычно этот метод позволяет очень быстро получить искомый документ, если известен его номер.

Доступ к документам RFC посредством электронной почты

Обращение к FTP-серверу – самый быстрый способ получить документ, а среда Web – наиболее комфортный вариант, но существуют и другие способы. Документы RFC могут быть получены посредством электронной почты. Электронная почта есть у многих пользователей, не имеющих прямого доступа к службам Интернета (поскольку они находятся в несвязанной сети либо ограничены настройками брандмауэра). Кроме того, в некоторых ситуациях электронная почта является приемлемым вариантом, поскольку документ требуется не срочно.

Чтобы получить документы по электронной почте, отправьте сообщение на адрес *mailserv@ietf.org*. Оставьте поле темы письма пустым. Запрос содер-



Number	Title	Author or Ed.	Date	Format	More Info (Obs&Upd)	Status
STD0060 RFC2920	**SMTP Service Extension for Command Pipelining**	N. Freed	September 2000	ASCII	Obsoletes RFC2197	STANDARD
STD0010 RFC0821	Simple Mail Transfer Protocol	J. Postel	Aug-01-1982	ASCII	Obsoletes RFC788, RFC780, RFC772, Obsoleted by RFC2821	STANDARD
RFC3030	**SMTP Service Extensions for Transmission of Large and Binary MIME Messages**	G. Vaudreuil	December 2000	ASCII	Obsoletes RFC788, Obsoleted by RFC2821 Obsoletes RFC1830	PROPOSED STANDARD
RFC2852	**Deliver By SMTP Service Extension**	D. Newman	June 2000	ASCII	Updates RFC1894	PROPOSED STANDARD

Рис. G.2. *Веб-поиск RFC*

жится в теле сообщения. Путь к документу следует предварить ключевым словом `FILE`. В данном примере мы запрашиваем документ RFC 1258.

```
% mail mailserv@ietf.org
Subject:
FILE /rfc/rfc1258.txt
^D
```

Этот механизм замечательно работает. Пока я набирал эти абзацы, запрошенный документ оказался в моем почтовом ящике.

Алфавитный указатель

Символы

Числа

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

A

Б

В

Г

Д

Ж

З

И

К

Л

М

Н

О

П

P

C

Т

У

Ф

Ц

Ч

Ш

Э

Я