В помощь сетевым администраторам

Solaris 8

Руководство администратора

СИМВОЛ®

O'REILLY®

Пол Уоттерс

Solaris 8

Administrator's Guide

Paul Watters

O'REILLY®

Solaris 8

Руководство администратора

Пол Уоттерс

Санкт-Петербург—Москва
2003

Пол Уоттерс

Solaris 8. Руководство администратора

Перевод М. Зислиса

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*С. Клименков*
Редактор	*А. Лосев*
Корректура	*С. Беляева*
Верстка	*Н. Гриценко*

Уоттерс П.

Solaris 8. Руководство администратора. – Пер. с англ. – СПб: Символ-Плюс, 2003. – 336 с., ил.

ISBN 5-93286-052-9

В книге «Solaris 8. Руководство администратора» подробно освещаются аспекты применения Solaris в качестве сервера, обеспечивающего работу сетевых служб всех уровней. Учитывая новую волну интереса к Solaris как к корпоративной сетевой ОС, автор уделяет основное внимание службам уровня предприятий и предлагает концептуальный, сложный материал, отсутствующий в прочих руководствах. Издание ориентировано на опытных сетевых администраторов, нуждающихся в предметном руководстве по сетевым функциям Solaris; рассмотрен процесс установки системы на платформах Intel и Sparc.

Материал книги, содержащий описание современных сетевых технологий, применяемых на практике, поможет освоить использование Solaris в качестве файлового сервера, сервера приложений и сервера баз данных и приобрести навыки, необходимые для понимания влияния новых служб и новых программных продуктов на существующие серверные системы.

ISBN 5-93286-052-9
ISBN 0-596-00073-1 (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 16.04.2003. Формат 70x100 1/16 . Печать офсетная.
Объем 21 печ. л. Тираж 2000 экз. Заказ N
Отпечатано с диапозитивов в Академической типографии «Наука» РАН
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие

Электронная коммерция навсегда изменила мир компьютеров. Приземленные задачи, связанные с созданием сетей, установкой систем, обеспечением удаленного доступа, в мировом масштабе вызывают гораздо больший интерес, чем проявляемый отдельными пользователями на местах. Клиент может получать доступ к приложению электронной коммерции, находясь на другом конце города и даже на другом конце света.

Пришло время, когда привычные витрины заменяются виртуальными, так что действия системных администраторов и архитекторов теперь подвергаются более тщательному анализу. Представьте себе, что реальный магазин вынужден закрыть двери для покупателей по причине аппаратного сбоя! Чтобы предоставить тот уровень обслуживания, который ожидается и даже требуется в эпоху сети Интернет, виртуальные магазины должны работать двадцать четыре часа в сутки, семь дней в неделю, по пятьдесят две недели в году. Меняются потребности покупателей, а значит, меняются потребности разработчиков и обслуживающего персонала систем: архитектурные решения ставятся под сомнение, а любое изменение бюджета должно быть оправдано в контексте существующих доходов. Вдумчивые системные администраторы системы Solaris™ теперь считают обязанностью вникать в деловую активность своих фирм, поскольку их действия при построении систем высокой доступности могут коренным образом повлиять на успешность любого предприятия.

Для кого эта книга

Эта книга является руководством для администраторов, перед которыми стоят задачи настройки сетевых служб Solaris, доступных далеко за пределами локальных сетей. Многие из принципов и примеров, приводимых в книге, применимы как в глобальных, так и в локальных сетях, хотя требования к безопасности и уровню обслуживания в приложениях электронной коммерции, как правило, оставляют в тени вопросы, связанные с локальными службами. Тем не менее в электронной коммерции особенно важна грамотная поддержка и сопровож-

дение локальных сетевых служб, от потенциала масштабирования которых может зависеть решение глобальных задач.

Методы, процедуры и техники, описанные в книге, не привязаны жестко к конкретному изданию системы Solaris; при этом отдельные команды, возможности и параметры могут варьироваться. Помимо прочего, некоторые из недавно появившихся служб доступны только в системе Solaris версии 8 и более поздних.

Чтение этой книги принесет пользу любому, кто имеет дело с сетевыми службами Solaris, будь то системный администратор, архитектор, инженер или разработчик.

Обзор

Настоящая книга рассказывает о построении сетей на основе служб системы Solaris. Сначала мы изучим основы сетевого взаимодействия и планирования подсетей и, в частности, рассмотрим процесс установки и настройки служб для отдельной вычислительной системы. От практических вопросов межсетевой маршрутизации мы перейдем к распространенным службам имен, реализующим управление различными ресурсами (узлами, пользователями и т. д.) в масштабах целых сетей. Речь пойдет об управлении этими ресурсами на уровне отдельных вычислительных систем, после чего будут подробно освещены две широко применяемые системы доступа к файлам (и связанные с ними протоколы). Дальше нас ждет обсуждение методик управления данными и вопросов, связанных с оценкой и измерением емкостных потребностей. Я представлю анализ способов обеспечения безопасности вычислительных систем и сетей посредством инструментов операционной системы Solaris и решений независимых разработчиков. И наконец, я расскажу о разработке системы электронной коммерции с применением уровней приложений и служб архитектуры Sun ONE™ (Open Network Environment, открытое сетевое окружение).

В главе 1 «Сеть – это компьютер» представлено будущее сетевых информационных систем глазами компании Sun – будущее, основанное на среде ONE и применении систем SunOS™ и Solaris в качестве базовых технологий, способных обеспечить солидный фундамент для деловой активности. На конкретном примере рассмотрен анализ требований и спецификаций для гипотетической системы электронной коммерции, построенной на базе Solaris, и того, каким образом различные уровни среды ONE могут настраиваться в целях эффективного решения поставленных задач.

В основе существования современных сетевых служб лежит семейство протоколов TCP/IP и сеть Интернет. Глава 2 «Создание сетей Solaris» посвящена краеугольным принципам работы сетей (в частности формату IP-адресов, разрешению имен узлов и адресов), а также вопросам построения сетей Solaris. Мы обсудим поддержку сетевых устройств, реа-

лизованную в системе Solaris, рассмотрим установку и настройку основных сетевых служб TCP/IP (включая демон интернет-служб *inetd*). Чтобы продемонстрировать основные принципы работы протокола TCP (Transmission Control Protocol) и родственных ему, мы разберем содержимое реальных пакетов, которыми обмениваются узлы под управлением Solaris.

В главе 3 «Установка системы Solaris» даны пошаговые инструкции по установке и выбору значений основных параметров сетевых настроек для платформ Solaris Intel и Solaris SPARC. Существует три метода установки системы Solaris: из командной строки (текстовый интерфейс), диалоговый (на основе меню) и веб-старт (на основе Java). Мы изучим последний. Кроме того, затронем задачи планирования, выполнение которых необходимо для выбора корректных параметров настройки сети. Вы узнаете о распространенных ошибках при установке системы Solaris, а также о методах разрешения возникающих проблем.

Учитывая сложность задач, связанных с настройкой единственной вычислительной системы, планирование и настройка целой сети могут выглядеть устрашающе. В главе 4 «Настройка сетей» рассказывается о настройке сети из машин под управлением Solaris, описываются настройка сетевых интерфейсов в различных конфигурациях, статическая и динамическая маршрутизация, а также способы диагностирования и разрешения сетевых проблем. В дополнение к этому материалу приводятся примеры инициализации устройств и проверки интерфейсов.

Когда сеть начинает расти, и ответственность за решение различных задач делится между системами, особую важность приобретает установка по меньшей мере одной системы каталогов, которая позволит находить узлы сети. Для доменов сети Интернет могут потребоваться дополнительные службы имен. В главе 5 «Службы имен» рассказывается о важности распределенных служб имен, позволяющих находить машины, пользователей и различные виды сетевых ресурсов, и читатели знакомятся с процессом настройки службы доменных имен (DNS) системы Solaris. DNS позволяет преобразовывать IP-адреса в имена, удобные для использования людьми, и обратно. Сетевая информационная служба (Network Information System, NIS/NIS+), разработанная Sun, позволяет сделать шаг вперед от традиционной системы DNS в плане поддержки иерархических пространств имен. Система NIS/NIS+, которую мы рассмотрим в подробностях, является универсальным решением по управлению сетевыми ресурсами, позволяющим проводить как авторизацию доступа к ресурсам, так и идентификацию, то есть проверку подлинности, во взаимодействиях клиентов и серверов. И наконец, речь пойдет о более современном облегченном протоколе доступа к каталогам (Lightweight Directory Access Protocol, LDAP) и его реализации на платформе Solaris.

В главе 6 «Системное администрирование» рассказывается об управлении пользователями и ресурсами в масштабе одной вычислительной

системы, описывается применение графического интерфейса *admintool* для удаления, добавления и изменения учетных записей пользователей, пакетов программного обеспечения, последовательных портов и принтеров. Кроме того, упоминаются инструменты командной строки, позволяющие выполнять те же действия. По мере появления в системе новых пользователей и ресурсов задача их эффективного сопровождения и обеспечения непрерывности действия служб становится все более важной. Читатели узнают о том, как управлять доступом пользователей в диалоговом режиме с помощью инструментов пользовательских квот, входящих в состав Solaris. Наконец, будет рассмотрена установка и настройка таких сетевых служб, как почтовый агент *sendmail*.

В сетях довольно широко применяется клиент-серверная архитектура, в которой сервер обеспечивает клиентам надежный доступ к файлам. В главе 7 «Файловые серверы» система Solaris представлена в пользующейся спросом роли сетевого файлового сервера. Solaris в качестве кроссплатформенного сервера обладает рядом привлекательных особенностей, таких как встроенная поддержка очень длинных имен файлов и впечатляющая производительность системы ввода-вывода. Читатели научатся применять Solaris для организации файловых серверов с помощью протокола сетевой файловой системы NFS (Network File System) и программного пакета Samba. Несмотря на великолепные показатели производительности NFS, в гетерогенных сетях предпочтительно применение Samba, поскольку этот пакет позволяет организовать совместный доступ к дисковым разделам и принтерам всех клиентов, понимающих протокол SMB, в частности операционных систем Microsoft Windows и Mac OS. Применение Samba позволяет решениям на базе системы Solaris полностью заменить существующие серверы Windows NT: сервер под управлением Solaris может выступать в роли первичного контроллера домена (Primary Domain Controller, PDC) для существующих сетей NT.

Перечисленные главы посвящены разнообразным способам хранения данных и работы с ними на системах под управлением Solaris в случае пользовательских приложений и системных служб. Следующее утверждение может показаться вполне очевидным, но основной причиной создания многих компьютерных систем является необходимость *управления* данными. Так, главной задачей систем управления базами данных является запись изначально достоверных данных, их безопасное хранение и обеспечение надежного доступа. В конечном итоге хранилище достоверных данных бесполезно, если нет возможности получить к нему доступ, а безопасное хранилище данных, не гарантирующее корректность выполняемых для таблиц транзакций, и вовсе полная глупость. В главе 8 «Управление данными» рассказывается об основах грамотного управления данными и об инструментах, предоставляемых системой Solaris для воплощения этих идей на практике. Мы изучим реальные примеры, связанные с хранением данных, и в

частности создание резервных копий и восстановление из них. Потерю или повреждение информации можно предотвратить, если придерживаться некоторых решений (например, зеркалирование и расслоение), которые реализуются с помощью специальных аппаратных средств, программного обеспечения сторонних разработчиков либо набора простых сценариев и ленточного накопителя. Вы узнаете о распространенных вариантах хранения данных, таких как жесткие диски, приводы CD-ROM и Zip/Jaz.

Глава 9 «Сетевая безопасность» – это курс практической безопасности, который начинается с изучения средств анализа сетевой безопасности, пакетов SAINT и WebSAINT. Читатели научатся строить брандмауэры с фильтрацией пакетов на базе маршрутизаторов и применять проверку подлинности клиентов по протоколу Kerberos 5, реализация которого входит в состав системы Solaris 8. Глава завершается примерами управления безопасностью транспортных уровней сети с помощью протокола SSL и удаленным доступом с помощью интерпретатора SSH.

В главе 10 «Сетевые информационные системы» рассказывается о создании информационных систем с веб-интерфейсом на базе сети интранет с трехзвенной архитектурой. Отметим, что хотя задача обучить читателей программированию на языке Java™ и не ставилась, но в этой главе приводится полностью рабочий пример создания сервера приложений на базе Solaris (Apache/JServ), установки сервера баз данных (Postgres), а также установки и тестирования конкретного приложения. Для понимания данного примера обязательно понадобятся навыки, приобретенные при чтении предшествующих глав. В качестве архитектуры для сетевого распределения приложений мы рассмотрим посредник объектных запросов VisiBroker.

Соглашения, принятые в книге

В этой книге используются следующие типографские обозначения:

Курсивом

оформлены команды, имена файлов настройки, каталогов и демонов; кроме того, курсив применяется для смыслового выделения и при первом использовании термина.

`Моноширинным шрифтом`

оформлены MAC- и IP-адреса.

Комментарии и вопросы

Пожалуйста, направляйте вопросы и комментарии, связанные с книгой, издательству:

O'Reilly & Associates, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472
(800) 998-9938 (в США или Канаде)
(707) 829-0515 (международный/местный телефон)
(707) 829-0104 (факс)

У данной книги существует веб-сайт, где представлены примеры, замеченные ошибки и другая дополнительная информация. Страница доступна по адресу:

http://www.oreilly.com/catalog/solaris8

Чтобы задать технический вопрос или прокомментировать содержание книги, воспользуйтесь адресом:

bookquestions@oreilly.com

Информация о других книгах, конференциях, центрах ресурсов (Resource Centers) и сети сайтов O'Reilly (O'Reilly Network) доступна на вебсайте O'Reilly по адресу:

http://www.oreilly.com

Благодарности

Мне хотелось бы упомянуть техническую поддержку, которую обеспечили замечательные ребята из Sun Microsystems, в частности Кейт Уотсон (Keith Watson), Равиндра Айер (Ravindra Iyer) и Дрю Ренн (Drew Wrenn). Многочисленные ценные предложения технических рецензентов также заслуживают благодарности. Спасибо, Экинвенд Зигмунд Уолтер-Джонсон третий (Akinwande Seigmund Walter-Johnson III), Эльза Ланкфорд (Elsa Lankford), Уильям Оссер (William Osser) и доктор Тим Гиббс (Tim Gibbs)! Хочется думать, что ваши усилия сделали книгу технически более выдержанной и упростили восприятие текста.

Что касается издательства O'Reilly, я хочу поблагодарить Джима Самзера (Jim Sumser), полного энтузиазма, терпеливого редактора, который помогал создавать эту книгу всеми доступными ему средствами. Спасибо, Джим, ты лучше всех! Кроме того, большое спасибо Робу Романо (Rob Romano) и Ханне Дайер (Hanna Dyer) за великолепные иллюстрации и Тиму О'Рейлли, парню, который является стержнем всего процесса.

В агентстве «Studio B» я говорю спасибо Нилу Залкинду (Neil Salkind) и Вики Хардинг (Vicki Harding), двум лучшим агентам в нашей части Техаса, а также Стэйси Бароун (Stacey Barone), Кристен Пикенс (Kristen Pickens), Крэйгу Уайли (Craig Wiley), Шерри и Дэвиду Рогельбергам (Sherry, David Rogelberg) за их отличную работу и терпение.

И, конечно же, спасибо моей супруге Майе и всей моей семье за то, что прощали мне ранние подъемы, работу допоздна и непрерывный стук клавиатуры.

1

Сеть – это компьютер

Рекламные формулы компании Sun уже давно сосредоточены вокруг единственного базового принципа: «Сеть – это компьютер». Не имея отношения к типичной рекламной шумихе, которая обычно окружает выпуск операционных систем, этот девиз четко отражает политику компании Sun и сущность ее флагманской операционной системы SunOS. Хотя SunOS является операционной системой Unix, она входит в состав полноценной рабочей среды, которая известна под названием Solaris. Таким образом, операционная система является ядром среды, но между системой и средой существует граница. Если вам нужна только операционная система Unix, SunOS является наиболее популярной в ряду таких систем, поскольку реализует преимущества существующих стандартов System V (несмотря на то, что изначально основывалась на системе BSD).

При этом, если требуется целостная рабочая среда, разработанная специально для развертывания сетевых служб, к вашим услугам многочисленные приложения и службы, входящие в состав Solaris. И действительно, среда Solaris разрабатывалась не для применения на независимых рабочих станциях, а для совместного функционирования большого количества вычислительных систем, предоставляющих пользователям высокопроизводительные и обладающие высокой готовностью распределенные службы. Такая перспектива не является новой: инженеры компании Sun стояли у истоков разработки служб и протоколов, которые сегодня считаются стандартами для сетей, начиная с RPC и NFS на уровне системы и заканчивая языком Java на уровне разработки приложений.

Система SunOS изменяется, обновляется и проектируется с учетом специфики аппаратного обеспечения, производимого компанией Sun,

но никак не в качестве базы, под которую прочие производители пытаются адаптировать свое аппаратное обеспечение. Этим достигается слаженность работы, которой трудно ожидать от операционных систем, не спроектированных под конкретные устройства.

В то время как прочие разработчики операционных систем колебались между пользовательскими и корпоративными вариантами, компания Sun сосредоточила усилия на разработке системы, которая выглядела бы совершенно одинаково на настольных компьютерах и корпоративных системах: масштабируемость зависит от аппаратного обеспечения, используемого с системой Solaris, а не является свойством самой системы. Любой администратор, разработчик или пользователь может спокойно поменять Sun Ray за 1 000 долларов на E10000 за 1 000 000 долларов и получить в свое распоряжение точно такой же пользовательский интерфейс и набор инструментов.

Sun ONE, открытое сетевое окружение

В качестве обобщенного комплексного плана развертывания служб Solaris компания Sun разработала спецификацию Sun ONE (Open Network Environment) – открытого сетевого окружения. Материалы доступны по адресу *http://www.sun.com/software/sunone/*. Спецификация описывает структуру из уровней, которые чаще всего используются для обеспечения работы сетевых служб корпоративного класса. Уровни приведены на рис. 1.1. Базовой топологией является модель клиент–сервер: клиент запрашивает услугу или данные, запрос передается через интерфейсную (frontend) систему, затем проходит через уровень, состоящий из серверов приложений и контейнеров, интегрированных с недоступными напрямую (backend) системами (в том числе и унаследованными) посредством связующего ПО. Услуга или информация предоставляется клиенту, пройдя тем же маршрутом, что и запрос, но в противоположном направлении. В основе такого взаимодействия лежит распределенное сопровождение пользователей и управление ресурсами, реализованное в SunOS. Еще один уровень, не связанный с транзакциями в реальном времени, но в конечном итоге дополняющий существующие службы, – это уровень разработки. Таким образом, службы Solaris можно условно разделить на службы разработки и службы развертывания.

Хотя среда ONE, описанная компанией Sun, может быть полностью реализована продуктами Sun (например, спектром продуктов iPlanet), одним из преимуществ открытой архитектуры является возможность применения продуктов других разработчиков. Скажем, сервер приложений iPlanet и веб-сервер iPlanet – замечательные продукты, но фирма, которая разрабатывает приложения электронной коммерции на Borland JBuilder, вполне может воспользоваться возможностями интеграции среды JBuilder с сервером приложений Borland и Java-вари-

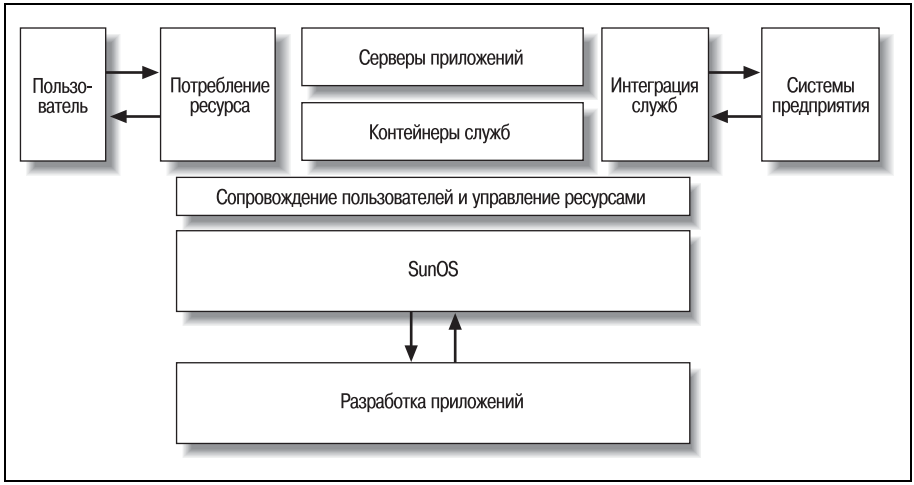

Рис. 1.1. *Сетевая архитектура Sun ONE*

антом CORBA-сервера VisiBroker. Таким же образом единая система сопровождения пользователей iPlanet может быть замещена продуктом, реализующим распределенную проверку подлинности по протоколу Kerberos. Выбор продуктов для конкретного уровня (например, уровня служб приложения) не должен сводиться к применению только продуктов, поставляемых разработчиком операционной системы, поскольку секретные API-вызовы и фирменные технологии, доступные только сотрудникам конторы-разработчика, делают невозможной интеграцию с продуктами третьих фирм. Открытая архитектура Sun делает продукты Sun и разработки третьих фирм взаимозаменяемыми.

Сейчас мы рассмотрим некоторые из инструментов среды Sun ONE и сравним их с отдельными альтернативными решениями третьих фирм.

Инструменты разработки

Семейство инструментов Forte от Sun содержит графические среды разработки для языков Java, C, C++ и Fortran. Эти инструменты поддерживают разработку, редактирование и просмотр кода, построение классов, отладку и оптимизацию широкого спектра приложений. Из существующих коммерческих альтернатив Forte можно упомянуть Borland JBuilder и C++ Builder, которые могут использоваться для разработки приложений на языках Java и C/C++. Другим источником качественных бесплатных компиляторов для языков C, C++ и Fortran является проект GNU (*http://www.gnu.org/*); в настоящее время эти компиляторы входят в состав дистрибутива Solaris. Бесплатный компилятор языка Java входит в состав инструментария Java Development Kit (*http://java.sun.com/*) от Sun, а более производительный и ка-

чественный компилятор Jikes так же бесплатно распространяется компанией IBM (*http://alphaworks.ibm.com/*).

Интерфейсные службы

Сервер порталов iPlanet реализует набор служб, ориентированных на работу с пользователями. Данный набор может применяться для поддержки самостоятельных приложений электронной коммерции. В числе его возможностей – сопровождение пользователей и групп, управление знаниями, персонализация информационного наполнения и организация совместной работы. Кроме того, сервер порталов iPlanet поддерживает разнообразные типы клиентов, начиная с клиентов WAP и заканчивая стандартными и безопасными HTTP-клиентами. Среди альтернатив можно назвать пакет Lotus Domino (*http://www.lotus.com/*), работающий на платформе Solaris, а также Reef Internetware, написанный на языке Java (*http://www.reef.com/*).

Контейнеры служб

Сервер приложений iPlanet реализует спецификацию Java 2 Enterprise Edition (J2EE), открывая дорогу в жизнь для объектно-ориентированных серверов приложений. J2EE поддерживает следующие службы: серверные страницы Java (Java Server Pages, JSP), сервлеты (servlets) и Enterprise JavaBeans – компоненты с сохранением состояния (stateful), компоненты без сохранения состояния (stateless), компоненты хранения данных (entity beans) и компоненты сеансов (session beans), основанные на стандартных паттернах проектирования Model-View-Controller (MVC). Альтернативным решением является сервер приложений Borland (*http://www.borland.com/*), который полноценно интегрируется с архитектурой CORBA и инструментами разработки от Borland. Бесплатный вариант – модуль Apache JServ (*http://java.apache.org/*), реализующий поддержку сервлетов на базе стандартного вебсервера Apache (*http://www.apache.org/*).

Интеграция служб

Сервер интеграции iPlanet является межплатформенным решением и позволяет связывать серверы приложений, источники данных и существующие системы. Этот продукт входит в число тех немногих систем, с помощью которых возможна успешная интеграция процессов планирования ресурсов предприятия (Enterprise Resource Planning, ERP) и управления взаимодействием с клиентами (Customer Relationship Management, CRM) и более современных систем обработки данных, в том числе и основанных на распределенных объектных архитектурах. Новые транзакционные стандарты, такие как служба сообщений Java (Java Message Service, JMS), применяются совместно с описаниями данных в формате XML и традиционных форматах (вроде DDL –

Data Description Language, язык описания данных) с целью координации и интеграции разнообразных источников данных и приложений.

Сопровождение пользователей и управление ресурсами

Sun предлагает целый ряд продуктов, предназначенных для реализации распределенного сопровождения пользователей и управления ресурсами. Вот некоторые из них:

- Система управления сертификатами iPlanet
- Администратор каталогов iPlanet
- Маршрутизатор доступа к каталогам iPlanet
- Сервер каталогов iPlanet
- Метакаталог iPlanet
- Proxy-сервер iPlanet

Перечисленные инструменты не обязательно должны использоваться, если предпочтение отдается продуктам сторонних разработчиков. Некоторые из таких продуктов входят в число стандартных инструментов SunOS. Так, в сетевой информационной службе NIS+ существует LDAP-шлюз, позволяющий передавать данные серверу каталогов. Этот подход может оказаться полезным для организаций, переходящих с NIS+ на LDAP; как правило, никто не хочет рисковать существующей информацией, а значит, и совершать резкий переход на новую систему. Другой пример: для управления авторизацией доступа к ресурсам может применяться система аутентификации (или подтверждения подлинности), основанная на протоколе Kerberos. В настоящее время доступно большое число коммерческих и бесплатных реализаций альтернатив.

SunOS

Компания Sun поставляет два варианта операционной системы SunOS: для платформы SPARC, предназначенный для работы с системами на базе масштабируемой процессорной архитектуры (Scalable Processor Architecture, SPARC), и для платформы Intel, совместимый с архитектурой x86. В SunOS входят все базовые службы, наличие которых необходимо для реализации всех остальных уровней архитектуры ONE, в частности:

- Ядро, разработанное для поддержки многопользовательских, многопоточных приложений, соответствующих стандартам POSIX и Unix.
- Набор драйверов устройств, охватывающий большую часть распространенного на платформах SPARC и Intel аппаратного обеспечения.

- Ряд командных интерпретаторов (shell), позволяющих одновременно запускать и управлять работой нескольких приложений посредством простого языка программирования.
- Графический интерфейс, удовлетворяющий промышленным стандартам и основанный на системе X11, – Common Desktop Environment (CDE).
- Полный набор сетевых служб на базе TCP/IP, включая RPC и NFS.

Помимо перечисленных компонентов, доступных на обеих платформах, существуют и возможности, специфичные для каждой из платформ и ориентированные на различные категории пользователей:

- На платформе SPARC система поддерживает до 128 процессоров. На платформе Intel – максимум 8. На платформе SPARC поддерживаются 64-битные процессоры, на платформе Intel только 32-битные.
- На платформе Intel возможен запуск приложений для Linux с помощью пакета *lxrun*. На платформе SPARC выполнение Linux-приложений невозможно.

Очевидно, выбор архитектуры SPARC или Intel зависит от потребностей конкретной системы. Необходимо отметить, что на платформе SPARC невозможно выполнение двоичного кода для платформы Intel и наоборот.

Версии Solaris

Если не считать крупных изменений, которым платформа SunOS подверглась при переходе между версиями 4.x и 5.x (изменения были связаны с переводом многих служб со стандарта BSD на стандарт System V), собственно платформа оставалась и остается последовательной. Так, сценарии командных интерпретаторов, написанные десять лет назад, по-прежнему работают корректно, а приложения, написанные для более ранних версий операционной системы, зачастую продолжают работать и в новых версиях, не требуя изменений. Авторам систем с непомерно раздутыми возможностями стабильность может казаться скучным довеском, но в долгосрочных проектах, длящихся временами более десяти лет, многократная смена операционной системы может быть связана с высокой степенью риска. Гарантия того, что основа системы электронной коммерции останется стабильной и через десять лет, положительно влияет на душевный покой руководства компаний, а также, не в последнюю очередь, разработчиков.

Платформа Solaris с течением лет приобрела много новых возможностей, но, как правило, их появление отражалось в качественном улучшении работы, а не в существенных изменениях в платформе каждые две недели. Один из недостатков эволюционирующих операционных систем (таких как Linux) – это возможность существования многочисленных несовместимостей между различными промежуточными вер-

сиями ядра, которая может приводить к неожиданным сбоям. Обновление ядра, добавляющее новые возможности, может хорошо взаимодействовать с существующими фукциональными модулями, а может сделать их вовсе неработоспособными. Стабильность же, как можно видеть из рис. 1.1, является очень важным фактором, когда стратегия электронной коммерции построена на фундаменте операционной системы.

Вот некоторые из возможностей, появившихся в Solaris 8:

- Автоматическое восстановление после сбоев
- Соответствие стандарту безопасности C2
- Снимки файловой системы
- Построение кластеров на основе пакета «Full Moon»
- Набор приложений iPlanet
- IPSec и IPv6
- Kerberos 5 (обновление с Kerberos 4)
- Фильтрация сетевых пакетов на уровне ядра
- Загрузка и установка «заплаток» без остановки вычислительной системы
- Поддержка Oracle 8i
- Управление доступом на основе ролей (Role-based access control, RBAC)
- Смарт-карты
- Соответствие стандарту Unix 98

Существует один момент, который может вызывать сложности, когда речь заходит о Solaris. Он связан с принятыми для рабочей среды и операционной системы соглашениями по именованию. В табл. 1.1 приведено соответствие между номерами наиболее широко применяемых версий операционной системы и рабочей среды. Отметим, что имя Solaris стало использоваться лишь после появления первых версий Solaris 2.

Таблица 1.1. Нумерация версий Solaris и SunOS

Версия SunOS	**Версия Solaris**
4.1.3	1.1.3
4.1.4	1.1.4
5.0	2.0
5.1	2.1
5.5.1	2.5.1
5.6	2.6
5.7	7
5.8	8

Ресурсы, посвященные Solaris

Одной из наиболее примечательных черт операционной системы SunOS является исчерпывающий набор высококачественной системной и пользовательской документации, поставляемой в комплекте с системой. Помимо системной справки, доступной для чтения из командного интерпретатора в любой момент, существуют следующие документы, доступные в форматах HTML и PDF через дополнительно поставляемый AnswerBook-сервер:[1]

Руководство по совместимости исполняемых файлов	Управление питанием, руководство пользователя
CDE, руководство пользователя	Руководство разработчика 64-битных приложений
Руководство по переходу на CDE	Руководство по совместимости исходных текстов
Руководство по драйверам устройств	Руководство по ассемблеру для SPARC
Руководство по интернационализации	Руководство по разработке STREAMS-модулей ядра
Руководство по системе автоматической инсталляции JumpStart	Руководство по модулю безопасности (SunShield)
Руководство по организации почтового сервера	Набор руководств по системному администрированию
Руководство по службам имен	Руководство по TCP/IP
NFS, руководство администратора	Руководства, посвященные диагностированию и разрешению проблем
Руководство по NIS+	WebNFS, руководство разработчика
OpenWindows, руководство пользователя	

Постоянно обновляемый перечень ресурсов по Solaris расположен по адресу *http://www.cassowary.net/solaris/*.

[1] Все эти руководства доступны по адресу *http://docs.sun.com*. – *Примеч. науч. ред.*

2

Создание сетей Solaris

В основе существования современных сетевых служб лежит семейство протоколов TCP/IP и сеть Интернет. В этой главе мы изучим краеугольные принципы работы сетей, в частности формат IP-адресов, разрешение имен узлов и адресов, а также рассмотрим архитектурные решения, подходящие для создания сетей Solaris. Я расскажу о поддержке сетевых устройств, реализованной в системе Solaris, рассмотрю установку и настройку основных сетевых служб TCP/IP (включая демон интернет-служб *inetd*). Чтобы продемонстрировать основные принципы работы протокола Transmission Control Protocol (TCP) и родственных ему, мы разберем содержимое реальных пакетов, которыми обмениваются вычислительные системы под управлением Solaris.

Системные понятия

Прежде чем начать проектировать сети, следует рассмотреть отдельные вычислительные системы и их характеристики. Причиной существования плохо спроектированной сети зачастую является ее незапланированный, хаотичный рост, приводящий к поглощению подсетей класса C или B. Понимание принципов наименования вычислительных систем, доменов, служб имен, а также принципов, связанных с IP-адресами, позволяет более эффективно управлять долгосрочным ростом сети.

Узел (host) – это самостоятельная вычислительная система, работающая под управлением Solaris. Узел должен иметь уникальный идентификатор в сети, в которую входит, что гарантирует уникальность его имени в сети Интернет, поскольку там все доменные имена различны. Таким образом, в мировом масштабе может существовать произволь-

ное число принадлежащих различным сетям узлов с именем *foxy*, но в каждом домене может существовать лишь один узел с таким именем. Как следствие невозможно перепутать узлы *foxy.savetheanimals.com* и *foxy.huntingholidays.com*. Кроме того, разными узлами являются *foxy.admin.huntingholidays.com, foxy.com*, *foxy.huntingholidays.com* и *foxy.fashion.huntingholidays.com*. Комбинация имени узла и доменного имени носит название полного доменного имени узла и однозначно определяет адрес узла в сети Интернет, ничего не предполагая относительно его географических координат.

Ethernet-адреса (или MAC-адреса) присваиваются элементам аппаратного обеспечения системы и являются уникальными идентификаторами сетевых интерфейсов узла. Адреса состоят из шестнадцатеричных чисел, разделенных двоеточиями (например, `0:50:ba:13:8:18`). MAC-адреса часто используются приложениями для выявления лицензированных интерфейсов, с которыми разрешено работать. В таких случаях MAC-адрес передается поставщику приложения, и на его основе создается ключ лицензии. К примеру, лицензия веб-сервера может быть связана только с одним интерфейсом, тогда как второй будет взаимодействовать с внутренней сетью, работая шлюзом маршрутизации. Таким образом, необходимо передать поставщику приложения MAC-адрес лишь одного из внешних интерфейсов.[1]

В дополнение к MAC-адресу каждому сетевому интерфейсу должен быть присвоен IP-адрес, который позволит обращаться к узлу в сети. Может возникнуть вопрос, зачем нужны IP-адреса, если у любого интерфейса всегда есть уникальный MAC-адрес. Ответ таков: для адресации в подсетях используется общая система, в которой различаются локальные и нелокальные адреса. Ethernet-адреса зашиты в аппаратное обеспечение, и нет способа определить по MAC-адресу интерфейса, является ли интерфейс локальным или удаленным для подсети.

IP-адрес – это 32-битное число, обычно представляемое в формате *w.x.y.z*. В сети класса C составляющая *z* содержит числа, присвоенные локальным узлам, поэтому широковещательное сообщение, адресованное сети *w.x.y.0*, достигнет всех узлов локальной подсети, но не будет получено узлами, которые лежат за ее пределами. Как мы увидим позже в этой главе, концепция IP-адресов является основой протокола Интернета (Internet Protocol, IP).

Для достижения логической завершенности подсетям требуются маски подсетей (или просто маски сетей), входящие в состав IP-адресов и определяющие принадлежность той или иной сети. Сети класса C опре-

[1] В Solaris лицензирование обычно производится по hostid (идентификатору узла), который может частично совпадать, а может и вовсе быть отличным от MAC-адреса. Кроме того, в Solaris по умолчанию все установленные сетевые интерфейсы одинакового типа имеют одинаковый MAC-адрес. – *Примеч. науч. ред.*

деляются маской 255.255.255.0, то есть широковещательные сообщения достигают узлов из диапазона *w.x.y.0–w.x.y.255*.[1]

IP-адреса необходимы для реализации протокола IP, но людям трудно их запоминать. Именно поэтому так важна служба доменных имен (DNS, domain name service) и другие, сходные с ней службы. Для работы протокола IP не требуется наличие DNS, но это, вне всякого сомнения, упрощает адресацию и позволяет более четко понять, как структурируются сети.

Архитектура сетей

Теперь изучим проектирование и конфигурации сетей Solaris. Начнем с простейшего варианта – узла, не входящего ни в какую *локальную* сеть. Хотя существует не так много случаев, когда узлы под управлением Solaris настроены таким образом, некоторым из рабочих станций не требуется доступ в локальную сеть, особенно если речь идет о домашней машине. Подобные узлы могут подключаться к сети Интернет посредством традиционного модемного (коммутируемого) соединения.

Если существует необходимость объединить в сеть два узла, прежде всего следует связать их кабелем типа «карта-карта» (crossover). Простая сеть в данном случае создается без необходимости применять дополнительное аппаратное обеспечение. Данная конфигурация часто используется для варианта единственного сервера с единственным удаленным терминалом.

Включение в сеть большего числа узлов требует наличия концентратора (hub) или (по меньшей мере) коммутатора (switch). Концентратор заботится обо всех соединениях и подключениях, необходимых для формирования простой сети с топологией «звезда»: каждый узел подключен к концентратору и способен общаться с другими узлами. Отметим, что концентраторы и коммутаторы могут «комбинироваться» – например, коммутатор может быть подключен к другому коммутатору через вспомогательный порт, а тот, в свою очередь, к следующему коммутатору или концентратору. Таким методом можно создать целую подсеть. Но следует помнить, что между маршрутизатором и наиболее удаленным из концентраторов должно быть не более трех транзитных участков (hops); в противном случае может происходить потеря пакетов в результате конфликтов (collisions).

Набора взаимосвязанных концентраторов и коммутаторов вполне достаточно для создания одной подсети, члены которой могут посылать и получать широковещательные сообщения. Но когда конфигурация сети

[1] Автор книги немного неправ. Диапазон будет *w.x.y.1–w.x.y.254*. Адреса *w.x.y.0* и *w.x.y.255* зарезервированы и не могут принадлежать конкретному узлу. – *Примеч. науч. ред.*

начинает распространение за пределы локальной зоны, которой изначально ограничивалась, зачастую приходится создавать дополнительные подсети. Такие подсети могут отражать существующие физические ограничения или структуру организаций. Так, для каждого из зданий на территории университета может быть создана отдельная подсеть. Как вариант в одном здании может существовать несколько подсетей, связанных с различными отделами (администрации, бухгалтерии, кадров и т. д.). Более веская причина создания подсетей – безопасность. Самые разнообразные данные свободно циркулируют по отдельным подсетям, и их можно скрыть от посторонних глаз, соответствующим образом разграничив подсети. Кроме того, для активной фильтрации определенных типов данных могут применяться брандмауэры.

Мы вплотную приблизились к прадедушке всех сетевых средств: маршрутизатору. Маршрутизаторы отвечают за связь между различными подсетями. Маршрутизатор передает данные, полученные через сетевой интерфейс первой подсети, через другой сетевой интерфейс, принадлежащий второй подсети. Поэтому если узел из подсети A желает обменяться данными с узлом из подсети B, все данные должны передаваться через маршрутизатор, связующий подсети A и B. Узлы различных подсетей не могут общаться напрямую. Мощь подсетей можно полностью оценить, если принять во внимание еще и такие технологии, как кэширующие серверы и прокси-серверы, а также пакетные фильтры, которые позволяют контролировать виды данных, циркулирующих между сетями.

Разумеется, в большую часть сетей входит более чем две подсети. Возьмем за основу предыдущий пример и предположим, что у нас есть сеть, состоящая из четырех подсетей: A, B, C и D. Пусть подсеть A связана с подсетью B, подсеть B связана с подсетью C, а подсеть C – с подсетью D. Как узлу из подсети A связаться с узлом из подсети D? Пакеты могут свободно циркулировать между подсетями A и B благодаря общему маршрутизатору, но необходимо еще определить маршрут из подсети A в подсеть D. Это достигается с помощью демона маршрутизации. В обнаружении маршрута нет никакого волшебства, хотя путешествие пакетов из одного полушария планеты в другое иногда кажется магией – ведь пакеты проходят через весьма ограниченное число маршрутизаторов.

Изложенное, в свою очередь, подводит нас к сети Интернет: большинство сетей, состоящих из одной или нескольких подсетей, связаны с сетью Интернет через поставщиков услуг Интернета (Internet Service Provider, ISP). Маршрутизатор локальной сети часто подключается (посредством ISDN, волнового передатчика или кабеля) с маршрутизатором, расположенным на стороне провайдера. Из этой точки пакеты, адресаты которых расположены за пределами локальной подсети, передаются далее, в сеть Интернет. Как только все организации подключатся к Сети через своих провайдеров, Интернет станет еще и широкомасштабной системой взаимосвязей.

Мы кратко перечислили основные конфигурации различных типов сетевых архитектур и теперь рассмотрим их более подробно. В следующем разделе приводятся примеры конфигураций для каждого из вариантов.

Одиночный узел

Одиночный узел не связан с другими узлами посредством традиционных сетевых технологий, что значительно ограничивает его способность обмениваться данными с другими узлами, локальными или удаленными. Подобные системы применяются в случаях, когда большая часть рабочих операций выполняется на одном узле. Скажем, художник-дизайнер может работать на машине Ultra 10, оборудованной графической картой с 3D-ускорителем, чтобы выполнить заказ, записать его на Zip-диск и отправить клиенту. В такой обособленности от сетей есть свои преимущества, в частности – повышенная безопасность (отсутствует физический уровень, через который злоумышленники могут атаковать систему) и экономия средств (нет необходимости приобретать сетевые карты и специальное сетевое оборудование). Разумеется, есть и целый ряд минусов: нет возможности общаться с коллегами по электронной почте и использовать вычислительные мощности других машин, разделяя нагрузку между несколькими серверами.

В данном варианте постоянное подключение к сети не требуется, но существует возможность установить модем в систему под управлением Solaris, а значит, и получить спорадический доступ к сети Интернет. На рис. 2.1 приведена простая конфигурация одиночного узла с моде-

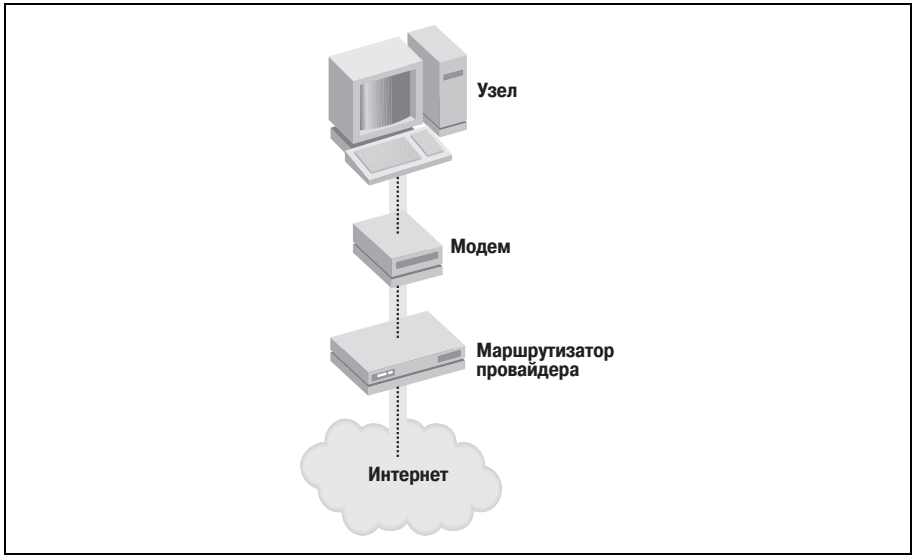

Рис. 2.1. *Сетевая конфигурация: самостоятельный узел*

мом, который позволяет устанавливать связь с сетью Интернет через маршрутизатор провайдера интернет-услуг.

Два узла (соединение «карта–карта»)

Простейшим решением для объединения в сеть пары узлов является связь их сетевых интерфейсов кабелем «карта-карта». Данный подход является простым и дешевым, поскольку необходимо купить только собственно кабель.

Подобные конфигурации известны под названием «одноранговых»: между узлами не существует закрепленных отношений вида клиент–сервер, каждый из узлов может пользоваться службами, предоставляемыми вторым. Одноранговые конфигурации полезны в условиях небольших офисов, когда рабочая станция подключается к более мощному серверу и получает доступ к приложениям, на выполнение которых у нее самой не хватает ресурсов. К примеру, PC-машина может быть подключена к E450 с целью выполнения симуляций MATLAB; визуализация данных будет производиться на PC по мере их получения от сервера.

Три узла или более (концентратор)

Как только сеть вырастает за пределы двух узлов, кабеля «карта-карта» становится недостаточно: для управления сетевым трафиком потребуется специальное устройство – концентратор. Концентратор отвечает за передачу пакетов между узлами – только в пределах локальной подсети. Можно провести аналогию с аэропортами: рейс из международного аэропорта Лос-Анджелеса в Чикаго никогда не бывает прямым, обязательно производится посадка в узловом аэропорту (к примеру, в DEN, если речь идет о компании United Airlines). Таким же образом концентратор с подключенными к нему узлами A, B и C отвечает за обмен данными между этими узлами. Отметим, что концентратор не является маршрутизатором, он не позволяет производить обмен данными между подсетями, хотя концентратор, связующий локальную подсеть, может быть подключен к маршрутизатору в целях обеспечения доступа к другим сетям для локальных узлов.

На рис. 2.2 отображена типичная конфигурация для концентратора с тремя узлами. Пакеты данных, адресуемых узлом A узлу B, путешествуют по маршруту узел A → концентратор → узел B.

Три узла или более (коммутатор)

До сих пор речь шла о стандартных одноранговых системах, в которых нет выраженных отношений вида клиент–сервер. Для подсетей, состоящих только из рабочих станций, это вполне нормально. В компьютерной лаборатории университета может существовать площадка, полная станций Blade 100, связанных в сеть цепочками концентрато-

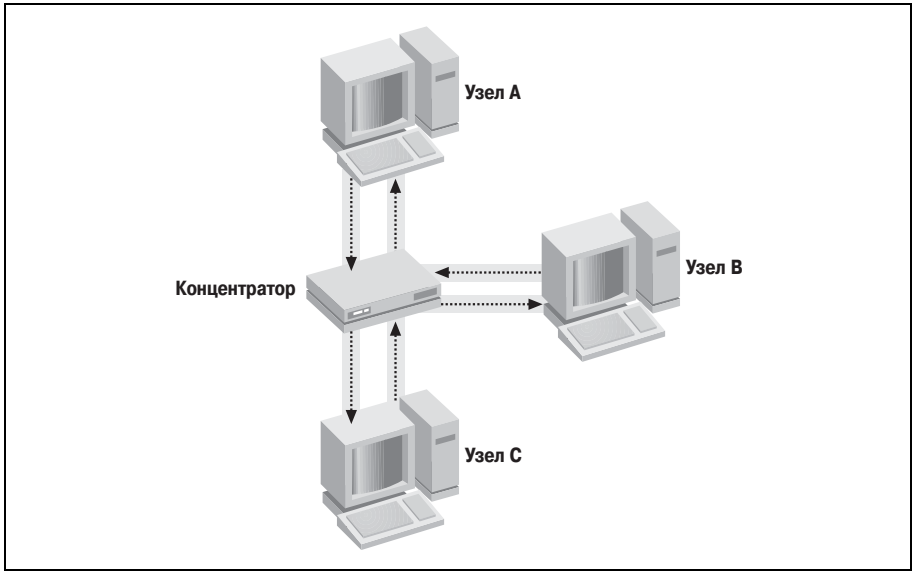

Рис. 2.2. *Сетевая конфигурация: несколько узлов (концентратор)*

ров и работающих без выделенного сервера (поскольку каждая из систем способна самостоятельно обеспечить работу всех необходимых служб). Но рассмотрим традиционный вариант клиент-серверной архитектуры: «тонкий» клиент большую часть задач выполняет, обращаясь к службам «толстого» сервера. В более старых Unix-системах выделенные терминалы VT-100 не имели собственных процессоров, были в буквальном смысле «немыми» (dumb) и в плане вычислительных мощностей полностью полагались на серверы. В наши дни на смену терминалу VT-100 пришло сетевое устройство Sun Ray, сочетающее графические возможности X11 с дизайном в стиле рабочих станций. Устройства Sun Ray зависимы от выделенного сервера (такого, как E-450), службами которого пользуются для выполнения своих задач. Этот подход имеет ряд преимуществ перед прочими:

- Установка ПО производится один раз для всей сети. Для семидесяти пяти машин Sun Blade 100 установку каждого программного пакета потребуется производить 75 раз. В случае аналогичного числа устройств Sun Ray ПО устанавливается один раз – на сервере.

- Пользователи могут регистрироваться на любом устройстве Sun Ray с помощью смарт-карт и продолжать прерванные сеансы работы, поскольку сервер позволяет сохранять состояние. Это означает, что сеанс работы можно приостановить в любой момент и продолжить позже, воспользовавшись другим устройством.

- Процессорные мощности могут контролироваться более эффективно, поскольку все клиенты пользуются одним и тем же диапазоном

процессорных циклов. В одноранговой среде процессы одной системы не могут простым путем использовать процессорные мощности других систем.

- Покупка 75 машин Sun Ray и выделенного сервера для них обойдется гораздо дешевле, чем покупка 75 машин Sun Blade!

Разумеется, централизация мощностей имеет и минусы: если произойдет сбой на сервере, клиенты Sun Ray не смогут работать сами по себе. Так что если произойдет перебой в подаче питания либо возникнут проблемы с нужным для работы диском, использование клиентов будет невозможно. Компания Sun решает эту проблему, разрабатывая системы высокой доступности и избыточности. Если говорить об избыточности в смысле аппаратного обеспечения, большинство серверов от Sun могут комплектоваться избыточными источниками питания и другими жизненноважными компонентами, которые работают в режиме автоматической защиты от сбоев с целью обеспечения непрерывного обслуживания. Высокая доступность связана с программными пакетами, такими как DiskSuite, который позволяет объединять обычные диски в RAID-массивы, поддерживающие как зеркалирование, так и расслоение (striping). Сбой одного из зеркалированных дисков не должен влиять на работу системы, хотя его следует заменить при первой возможности. (И это можно сделать, не останавливая работу сервера.)

На рис. 2.3 приведена типичная конфигурация архитектуры клиент–сервер для трех устройств Sun Ray и выделенного сервера E-450. Отметим физические параметры подключения: клиенты связаны с высоко-

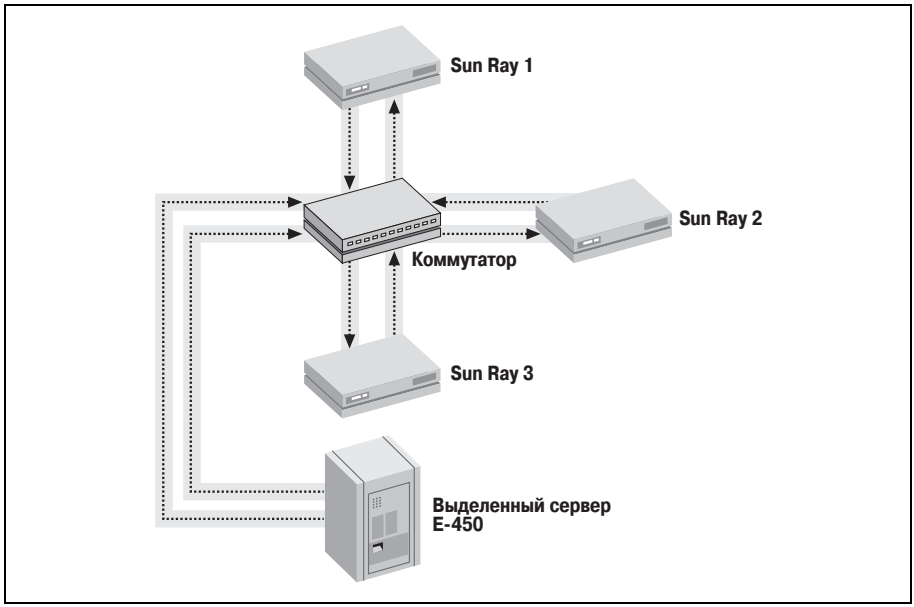

Рис. 2.3. *Сетевая конфигурация: множественные узлы (коммутатор)*

скоростным коммутатором (100 Мбит/сек), а коммутатор связан непосредственно с сервером.

Две подсети

Когда сеть вырастает за пределы одной подсети, возникает потребность в маршрутизаторе, который выступает в качестве посредника между подсетями. Маршрутизатор в буквальном смысле ищет маршруты, которыми должны следовать пакеты, и направляет пакеты. Процесс установки и настройки маршрутизатора подробно рассмотрен в главе 4. Маршрутизатор может быть представлен специальным аппаратным решением, но в сети Solaris это будет, скорее всего, Sun-система с по меньшей мере двумя сетевыми интерфейсами. В случае простой сети, состоящей из пары подсетей A и B, один из интерфейсов ассоциируется с подсетью A, а второй – с подсетью B. Интерфейс подсети A должен быть связан непосредственно с концентратором подсети A; интерфейс подсети B должен быть, точно так же напрямую, связан с концентратором подсети B. При таком раскладе передача данных от узла X подсети A узлу Y подсети B приводит к поступлению пакетов через сетевой интерфейс узла X на концентратор, который передает их интерфейсу подсети A на маршрутизаторе. С настроенного соответствующим образом маршрутизатора пакеты передаются через интерфейс подсети B на концентратор сети B и, наконец, через сетевой интерфейс узла Y доставляются адресату. Процесс, несомненно, более сложный, чем при объединении двух узлов X и Y сетевым кабелем «карта-карта», но если обобщить процедуру для тысяч сетей, мы получим основу, на которой строится доставка пакетов в сети Интернет.

На рис. 2.4 показан маршрут пакетов, адресованных узлом X подсети A узлу Y подсети B; используется маршрутизатор.

Четыре подсети

Для управления множественными подсетями требуется большее число маршрутизаторов. К счастью, для объединения N^s подсетей одной сети требуется лишь N^{s-1} маршрутизаторов. При этом с ростом числа подсетей маршрутизатору необязательно пропускать через себя все данные из всех подсетей. Возьмем для примера сеть, в которой четыре подсети (A, B, C и D). Для такой сети требуется наличие всего трех маршрутизаторов: A↔B, B↔C и C↔D. Дело в том, что маршрутизаторы умеют находить маршруты и самостоятельно определять, что пакет, адресованный из подсети A в подсеть D, должен пройти через маршрутизаторы A↔B, B↔C и C↔D. Если бы автоматический поиск маршрута не существовал, каждую из подсетей пришлось бы напрямую соединять с каждой из оставшихся, и число маршрутизаторов росло бы не линейно, но экспоненциально. (Конечно, подсети можно связывать и таким методом, но подобные связи будут избыточными.)

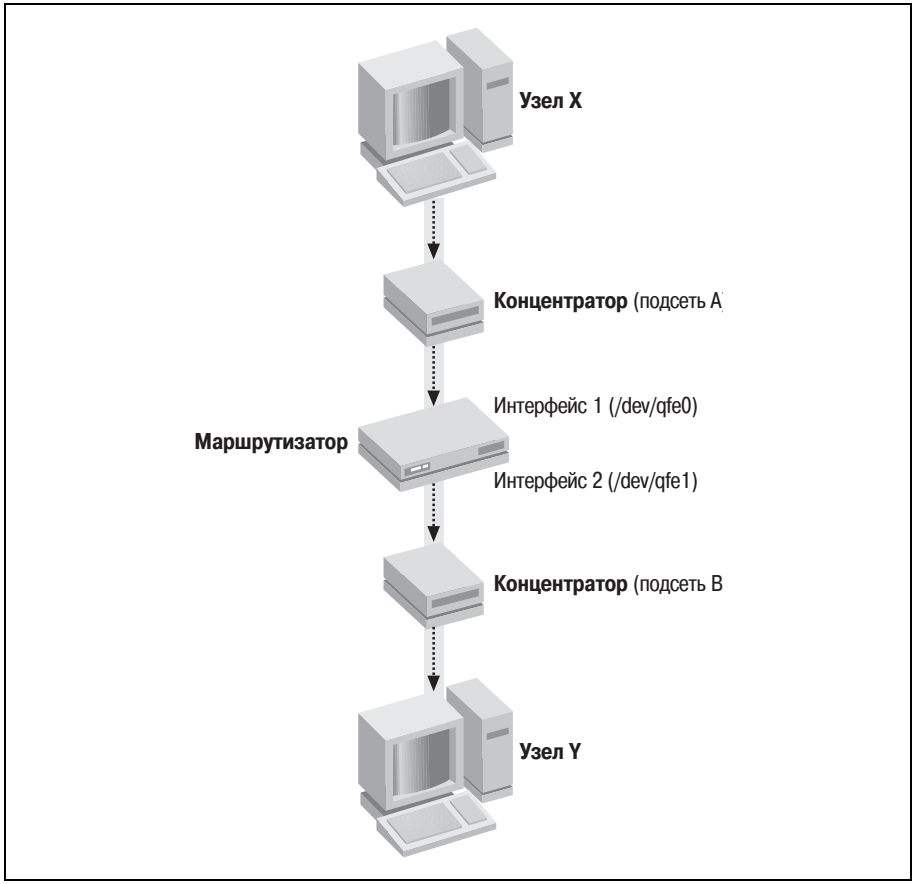

Рис. 2.4. *Сетевая конфигурация: две подсети*

На рис. 2.5 показан маршрут пакетов, адресованных узлом X подсети A узлу Y подсети D. Подсети A, B, C и D сообщаются посредством трех маршрутизаторов.

Протоколы Интернета

Многие сети связаны с сетью Интернет либо используют протокол Интернета (Internet Protocol, IP) и протоколы более высокого уровня, такие как TCP/IP, для организации передачи данных. В этом разделе я расскажу, для чего применяется каждый из протоколов с учетом модели взаимодействия открытых систем (Open Systems Interconnect Network Model, OSI), определяющей семь уровней сетевого взаимодействия. Модель OSI, находящаяся в ведении Международной организации по стандартизации (International Standards Organization, ISO), яв-

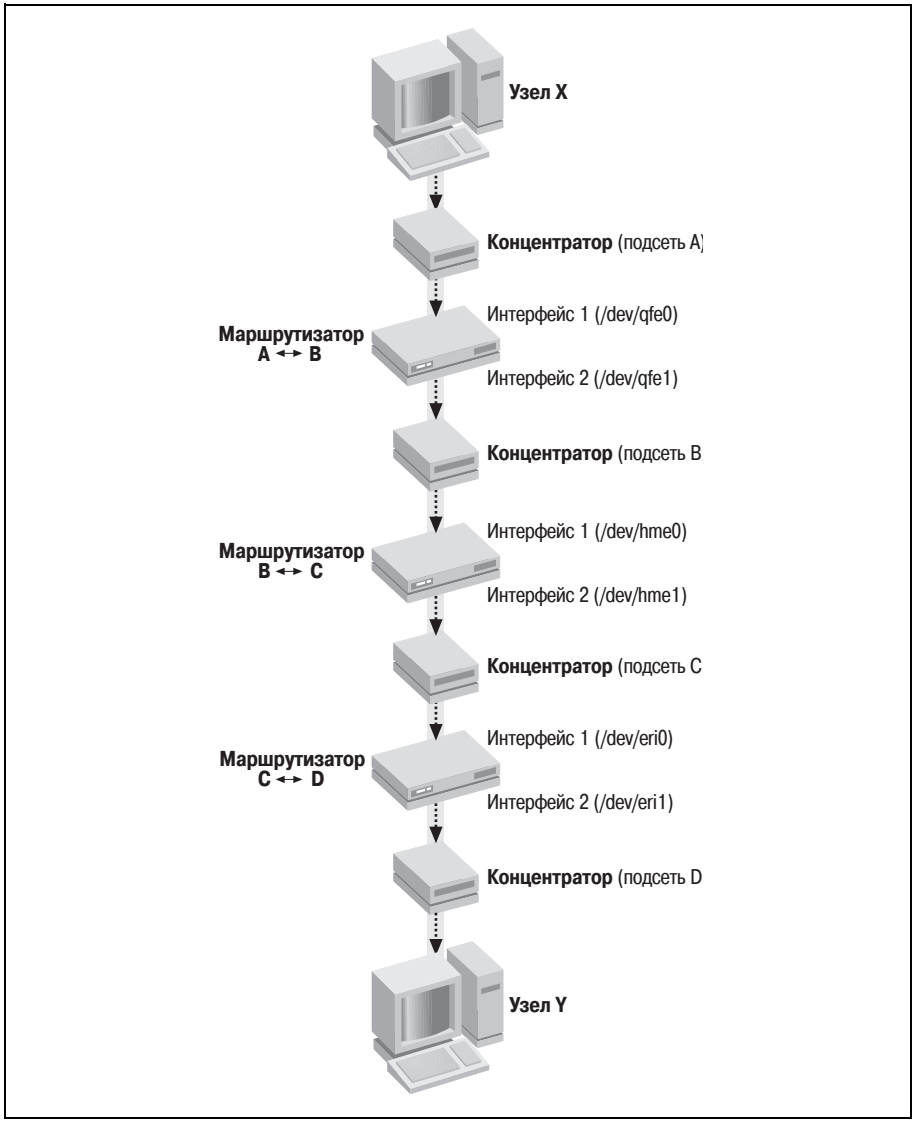

Рис. 2.5. *Сетевая конфигурация: четыре подсети*

ляется фундаментом семейства протоколов TCP/IP. Разобравшись в концепциях OSI, вы с легкостью будете ориентироваться в отношениях различных составляющих TCP/IP.[1]

[1] В общем случае модель OSI и модель TCP/IP – это две разных модели. С достаточной степенью достоверности можно утверждать, что четыре нижних уровня в модели OSI и TCP/IP совпадают. – *Примеч. науч. ред.*

Модель OSI показана на рис. 2.6. Верхним уровнем в модели является прикладной уровень. Разумеется, сетевые приложения входят в данный уровень. Далее следует уровень представления, общий для всех приложений; он гарантирует, что передаваемые по сети данные представлены в согласованных форматах. В основе представления – сеансовый уровень, призванный логически разделять приложения по сеансам (представьте, что перепутаются данные, адресованные различным приложениям!). Транспортный уровень отвечает за реализацию протоколов передачи данных, проверку подлинности данных и гарантии доставки пакетов. На сетевом уровне происходит управление соединениями;[1] по существу, речь идет о связи приложений посредством конкретного транспорта с канальным уровнем, ответственным за передачу данных на самом нижнем уровне. Наконец, физический уровень связан с непосредственной физической реализацией передачи битов и байтов данных по сети.

Уровни TCP/IP отражают уровни модели OSI. Все пользовательские и клиентские приложения, использующие сеть, принадлежат прикладному уровню, и для реализации сетевых функций применяются потоки и сообщения. Уровень представления реализуется в виде стандартизованных форматов данных, передаваемых по сети, таких как протокол XDR (eXternal Data Representations, представления внешних данных). Сеансовый уровень находит отражение в портах и сокетах, в то время как протокол контроля передачи (Transmission Control Protocol, TCP), гарантирующий доставку пакетов, и протокол пользовательских дейтаграмм (User Datagram Protocol, UDP), не дающий подобных гарантий, составляют транспортный уровень. Сетевой уровень реализован в виде протокола IP. Канальный и физический уровни тес-

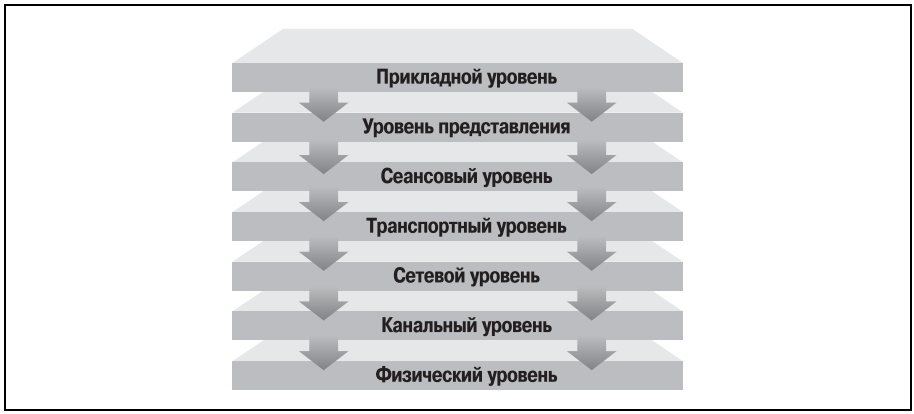

Рис. 2.6. *Сетевая модель OSI*

[1] В TCP/IP сетевой уровень ответственен еще и за маршрутизацию данных между сетями. – *Примеч. науч. ред.*

но связаны с аппаратным обеспечением, применяемым для организации сетей (интерфейсными картами, кабелями и концентраторами).

Применение inetd

Большинство служб Solaris работают в качестве демонов, то есть самодостаточных агентов, принимающих запросы клиентов. Демоны могут существовать в автономном режиме либо в сотрудничестве с демоном служб Интернета (*inetd*). Использование *inetd* в качестве службы запуска для демонов имеет ряд преимуществ: для управления всеми *inetd*-демонами требуется лишь два файла настройки (*/etc/inetd.conf* и */etc/services*), а важная функциональность, такая как ведение журнала сообщений, может быть централизована в одной службе. В результате мы получаем единообразие использования для всех демонов, работающих через *inetd*, тогда как автономные демоны имеют гораздо меньше ограничений на совершаемые действия и параметры настройки. В некоторых ситуациях это может быть даже удобно. Например, сервер Apache может работать автономно или через *inetd*; автономный вариант дает возможность более точной настройки в таких вопросах, как многопоточность и применение облегченных процессов. Демоны, выполняемые через *inetd*, могут работать с использованием транспортных протоколов TCP и UDP.

inetd имеет и свои минусы: поскольку он является основой работы всех прочих служб, их работа будет прервана одновременно, если *inetd* подвергнется внешней атаке либо воздействию (еще не обнаруженной) ошибки программного кода.

Многие стандартные службы Unix-систем работают на базе *inetd*, в частности *ftp*, *telnet*, *rsh*, *rlogin*, *rexec*, *finger*, *talk* и UUCP. Новые демоны, разработанные в той же философии, вероятнее всего будут оптимально работать через *inetd*, и новые демоны Kerberos 5 – не исключение.

Как уже говорилось, *inetd* использует два файла настройки: */etc/inetd.conf* является файлом настройки собственно демона *inetd*, в нем перечисляются все поддерживаемые службы и параметры их работы; в */etc/services* все существующие номера служб привязываются к именам, что облегчает работу с *inetd*. Последний файл используется приложениями, вызывающими функцию *getprotobyname( )* для получения имени (а также номера и синонимов) протокола. После добавления или удаления служб (файл */etc/inetd.conf*) изменения можно передать в работающую копию *inetd*, послав сигнал HUP процессу *inetd*. Результатом будет холодный перезапуск демона *inetd* и повторная обработка файла *inetd. conf*.

```
bash-2.03# kill -HUP `pgrep inetd`
```

Разумным решением является удаление определения службы с помощью символа комментария либо добавление нового определения в

/etc/services при внесении изменений в файл */etc/inetd.conf*. Это позволяет сохранять корректную взаимосвязь файлов настройки и минимизирует возможную путаницу относительного того, какие из служб разрешены для использования внешними узлами, а для каких существует лишь соответствие между именем службы, используемым протоколом и номером порта.

Типичный файл */etc/services* содержит следующие определения:

```
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
smtp            25/tcp          mail
time            37/tcp          timserver
time            37/udp          timserver
name            42/udp          nameserver
whois           43/tcp          nicname         # usually to sri-nic
domain          53/udp
domain          53/tcp
bootps          67/udp                          # BOOTP/DHCP server
bootpc          68/udp                          # BOOTP/DHCP client
hostnames       101/tcp         hostname        # usually to sri-nic
pop2            109/tcp         pop-2           # Post Office Protocol - V2
pop3            110/tcp                         # Post Office Protocol - Version 3
sunrpc          111/udp         rpcbind
sunrpc          111/tcp         rpcbind
imap            143/tcp         imap2           # Internet Mail Access Protocol v2
ldap            389/tcp                      # Lightweight Directory Access Protocol
ldap            389/udp                      # Lightweight Directory Access Protocol
ldaps           636/tcp                     # LDAP protocol over TLS/SSL (was sldap)
ldaps           636/udp                     # LDAP protocol over TLS/SSL (was sldap)
tftp            69/udp
rje             77/tcp
finger          79/tcp
link            87/tcp          ttylink
supdup          95/tcp
iso-tsap        102/tcp
x400            103/tcp                         # ISO Mail
x400-snd        104/tcp
csnet-ns        105/tcp
```

```
pop-2          109/tcp                    # Post Office
uucp-path      117/tcp
nntp           119/tcp      usenet        # Network News Transfer
ntp            123/tcp                    # Network Time Protocol
ntp            123/udp                    # Network Time Protocol
netbios-ns     137/tcp                    # NETBIOS Name Service
netbios-ns     137/udp                    # NETBIOS Name Service
netbios-dgm    138/tcp                    # NETBIOS Datagram Service
netbios-dgm    138/udp                    # NETBIOS Datagram Service
netbios-ssn    139/tcp                    # NETBIOS Session Service
netbios-ssn    139/udp                    # NETBIOS Session Service
NeWS           144/tcp      news          # Window System
slp            427/tcp      slp           # Service Location Protocol, V2
slp            427/udp      slp           # Service Location Protocol, V2
mobile-ip      434/udp      mobile-ip     # Mobile-IP
cvc_hostd      442/tcp                    # Network Console
exec           512/tcp
login          513/tcp
shell          514/tcp      cmd           # no passwords used
printer        515/tcp      spooler       # line printer spooler
courier        530/tcp      rpc           # experimental
uucp           540/tcp      uucpd         # uucp daemon
biff           512/udp      comsat
who            513/udp      whod
syslog         514/udp
talk           517/udp
route          520/udp      router routed
ripng          521/udp
klogin         543/tcp                    # Kerberos authenticated rlogin
kshell         544/tcp      cmd           # Kerberos authenticated remote shell
new-rwho       550/udp      new-who       # experimental
rmonitor       560/udp      rmonitord     # experimental
monitor        561/udp                    # experimental
pcserver       600/tcp                    # ECD Integrated PC board srvr
kerberos-adm   749/tcp                    # Kerberos V5 Administration
kerberos-adm   749/udp                    # Kerberos V5 Administration
kerberos       750/udp      kdc           # Kerberos key server
kerberos       750/tcp      kdc           # Kerberos key server
krb5_prop      754/tcp                    # Kerberos V5 KDC propogation
ufsd           1008/tcp     ufsd          # UFS-aware server
ufsd           1008/udp     ufsd
cvc            1495/tcp                   # Network Console
ingreslock     1524/tcp
www-ldap-gw    1760/tcp                   # HTTP to LDAP gateway
www-ldap-gw    1760/udp                   # HTTP to LDAP gateway
listen         2766/tcp                   # System V listener port
nfsd           2049/udp     nfs           # NFS server daemon (clts)
nfsd           2049/tcp     nfs           # NFS server daemon (cots)
eklogin        2105/tcp                   # Kerberos encrypted rlogin
lockd          4045/udp                   # NFS lock daemon/manager
lockd          4045/tcp
```

```
dtspc          6112/tcp                    # CDE subprocess control
fs             7100/tcp                    # Font server
wnn6           22273/tcp                   # Wnn6 jserver
wnn6           22273/udp                   # Wnn6 jserver
wnn6-ds        26208/tcp    wnn6_DS        # Wnn6 wnnds
wnn6-ds        26208/udp    wnn6_DS        # Wnn6 wnnds
```

Рассмотрим такое соответствие служб более подробно на примере службы *ldap*. Служба *ldap* использует протоколы TCP и UDP, принимая запросы и отправляя ответы через порт 389. Это означает, что никакая другая служба не может использовать порт 389 совместно с протоколом UDP или TCP для каких-либо целей. Также следует отметить, что только службы, выполняемые пользователем root, могут работать на портах с номерами до 1024, поскольку эти порты являются привилегированными и зарезервированы за администратором.

Некоторые из служб в работе используют лишь один из протоколов (только TCP или только UDP). Так, демон *talk* всегда работает по протоколу UDP через порт 517, а демон *telnet* по протоколу TCP через порт 23. Зачастую служба использует более одного порта для работы по TCP или UDP. К примеру, демон *ftp* работает через порты 20 и 21 по протоколу TCP, а сервер ключей *kerberos* работает через порт 750 (UDP и TCP), производя распространение ключей по протоколу TCP через порт 754. Демон *inetd* и система служб являются достаточно гибкими, чтобы охватить многочисленные варианты и комбинации, соответствующие существующим службам.

Формат файла */etc/inetd.conf* схож с форматом */etc/services*: каждому определению службы отведена отдельная строка. Типичный файл */etc/inetd.conf* может выглядеть так:

```
ftp     stream   tcp6   nowait   root   /usr/sbin/in.ftpd    in.ftpd
telnet  stream   tcp6   nowait   root   /usr/sbin/in.telnetd    in.telnetd
name    dgram    udp    wait     root   /usr/sbin/in.tnamed    in.tnamed
shell   stream   tcp    nowait   root   /usr/sbin/in.rshd    in.rshd
shell   stream   tcp6   nowait   root   /usr/sbin/in.rshd    in.rshd
login   stream   tcp6   nowait   root   /usr/sbin/in.rlogind    in.rlogind
exec    stream   tcp    nowait   root   /usr/sbin/in.rexecd    in.rexecd
exec    stream   tcp6   nowait   root   /usr/sbin/in.rexecd    in.rexecd
comsat  dgram    udp    wait     root   /usr/sbin/in.comsat    in.comsat
talk    dgram    udp    wait     root   /usr/sbin/in.talkd    in.talkd
uucp    stream   tcp    nowait   root   /usr/sbin/in.uucpd    in.uucpd
finger  stream   tcp6   nowait   nobody   /usr/sbin/in.fingerd    in.fingerd
time    stream   tcp6   nowait   root   internal
time    dgram    udp6   wait     root   internal
echo    stream   tcp6   nowait   root   internal
echo    dgram    udp6   wait     root   internal
discard stream   tcp6   nowait   root   internal
discard dgram    udp6   wait     root   internal
daytime stream   tcp6   nowait   root   internal
daytime dgram    udp6   wait     root   internal
```

```
chargen stream   tcp6   nowait   root   internal
chargen dgram    udp6   wait     root   internal
100232/10   tli   rpc/udp   wait root /usr/sbin/sadmind   sadmind
rquotad/1   tli   rpc/datagram_v   wait root /usr/lib/nfs/rquotad   rquotad
rusersd/2-3  tli   rpc/datagram_v,circuit_v   wait root /usr/lib/netsvc/rusers/rpc.
    rusersd   rpc.rusersd
sprayd/1   tli   rpc/datagram_v   wait root /usr/lib/netsvc/spray/rpc.sprayd   rpc.
    sprayd
walld/1    tli   rpc/datagram_v   wait root /usr/lib/netsvc/rwall/rpc.rwalld   rpc.
    rwalld
rstatd/2-4   tli   rpc/datagram_v wait root /usr/lib/netsvc/rstat/rpc.rstatd rpc.
    rstatd
100083/1   tli   rpc/tcp wait root /usr/dt/bin/rpc.ttdbserverd rpc.ttdbserverd
100221/1   tli   rpc/tcp   wait root /usr/openwin/bin/kcms_server   kcms_server
    fs      stream   tcp   wait nobody /usr/openwin/lib/fs.auto   fs
100235/1 tli rpc/tcp wait root /usr/lib/fs/cachefs/cachefsd cachefsd
100134/1   tli   rpc/ticotsord   wait   root   /usr/lib/krb5/ktkt_warnd ktkt_warnd
    printer      stream   tcp6   nowait   root   /usr/lib/print/in.lpd
in.lpd
100234/1   tli   rpc/ticotsord   wait   root   /usr/lib/gss/gssd gssd
100146/1   tli   rpc/ticotsord   wait   root   /usr/lib/security/amiserv   amiserv
100147/1   tli   rpc/ticotsord   wait   root   /usr/lib/security/amiserv   amiserv
100150/1   tli   rpc/ticotsord   wait   root   /usr/sbin/ocfserv   ocfserv
    dtspc stream tcp nowait root /usr/dt/bin/dtspcd /usr/dt/bin/dtspcd
100068/2-5 dgram rpc/udp wait root /usr/dt/bin/rpc.cmsd rpc.cmsd
100153/1    dgram   rpc/udp wait root /opt/SUNWvts/bin/sunvts /opt/SUNWvts/bin/
    sunvts -g
100229/1   tli   rpc/tcp     wait   root   /usr/sbin/rpc.metad   rpc.metad100230/1
    tli   rpc/tcp      wait   root   /usr/sbin/rpc.metamhd   rpc.metamhd
```

Записи имеют следующий стандартный формат:

```
имя   тип   протокол   флаги   пользователь   параметры сервера
```

где *имя* – имя службы; *тип* определяет тип службы – потоковая, на основе дейтаграмм либо TLI; *протокол* определяет использование протоколов TCP (*tcp*) или UDP (*udp*) из стандарта IPv4 либо из стандарта IPv6. Тип STREAMS изначально разрабатывался в составе спецификации System V с целью создания модульного, многоуровнего стандарта для сетевых приложений. Напротив, тип TLI связан с абстрактным механизмом, позволяющим обеспечить гарантированный сквозной доступ к службам на основе соединений либо без таковых.

Вслед за именем пользователя (владельца процесса) следует указать флаги (по необходимости), имя сервера и параметры его выполнения. Разберем для примера определение службы демона *finger*. *finger* позволяет определить, какие пользователи работают с локальной или удаленной системой. Демон позволяет получать более подробную информацию по конкретным пользователям, в частности дату и время последней регистрации в системе, имя домашнего каталога, планы и проекты для конкретного пользователя. Разумеется, *finger* имеет преимущест-

венно историческое значение, хотя и может успешно использоваться в интранет-сетях, которые защищены брандмауэром, фильтрующим запросы к данной службе. Тем не менее *finger* выдает довольно много информации, которая может использоваться для подготовки атаки. Имя службы демона (*in.fingerd*) – *finger*, тип службы – *stream*, протокол – TCP (IPv6), а владельцем службы является пользователь *nobody*.

Если выполнить клиентское приложение *finger* на локальной системе, можно получить вывод следующего вида:

```
bash-2.03$ finger
Login       Name               TTY         Idle    When    Where
root     Super-User            console       11 Mon 11:38  :0
pwatters Paul Watters          pts/6            Fri 20:21  hurley
```

Если по соображениям безопасности или экономии ресурсов необходимо остановить работу демона *finger*, закомментируйте соответствующую запись в файлах */etc/inetd.conf* и */etc/services*, а затем перезапустите *inetd*, применив описанный выше способ.

Службы Telnet и FTP входят в число наиболее востребованных и являются наглядным примером демонов, которые обычно сосуществуют с *inetd*.

Telnet

Служба Telnet обеспечивает терминальный удаленный доступ ко всем системам, работающим под управлением Unix. Это означает, что Solaris-клиент Telnet может соединиться с Solaris-демоном *telnet*, но также и с Linux-демоном *telnet* или любой другой системой, которая подчиняется требованиям спецификации по Telnet. Несмотря на способность службы Telnet соединять клиента из одной страны с сервером из другой, сетевые администраторы обычно довольно косо смотрят на применение этой службы за пределами защищенных брандмауэрами подсетей; их в основном беспокоит, что имена пользователей и пароли могут быть перехвачены взломщиками, поскольку вся информация, связанная с проверкой подлинности (да и вообще все данные сеансов Telnet), передается в открытом виде (то есть не шифруется). В главе 9 мы рассмотрим безопасные альтернативы Telnet. Тем не менее Telnet все еще широко используется в локальных сетях для доступа с клиентских систем на сервер. Поскольку Telnet обеспечивает терминальный доступ, полностью совместимый с VT-100, служба может использоваться для подключения к клиентам баз данных, доступа к веб-серверам или другим терминальным приложениям удаленных систем.

Telnet может работать с протоколом X11, разумеется, если применение протокола возможно на локальном узле и если передача с удаленного сервера на локальный дисплей разрешена посредством команды *xhost*. Кроме того, чтобы приложения могли работать корректно, сле-

дует установить значение переменной среды DISPLAY, отразив адрес консоли локальной системы. Клиенты CDE и X11 часто пользуются этим методом запуска приложений на удаленных системах и взаимодействия их с локальными серверами X11. Это одно из крупных преимуществ работы в клиент-серверном окружении, основанном на Solaris, поскольку в прочих операционных системах работа с удаленными сеансами не всегда доступна столь элегантным путем. Кроме того, один пользователь может работать с несколькими приложениями одного или нескольких удаленных серверов. Таким образом, при необходимости возможно создание узкоспециализированных серверов; клиенты будут подключаться к определенному серверу через Telnet, в зависимости от того, какой тип задачи требуется решить – поработать с базами данных, веб-серверами или финансовыми приложениями.

Вот пример стандартного терминального Telnet-сеанса. Работая на клиентской системе (*hurley*), с помощью Telnet мы подключаемся к удаленной системе *austin*:

```
hurley% telnet austin
Trying 10.64.18.3...
Connected to austin.
Escape character is '^]'.


SunOS 5.8

login: pwatters
Password:
Last login: Fri Jun 15 20:46:56 from austin
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000

© 2000 Sun Microsystems, Inc.  901 San Antonio Road, Palo Alto,
California, 94303, U.S.A.  All rights reserved.

This product includes software subject to the license and legal
notice contained herein and which may be found at
/opt/legal/License and /opt/legal/Legal_Notice.

austin%
```

Выполнив команду *telnet*, пользователь должен ввести свое регистрационное имя и пароль, прежде чем будет запущен командный интерпретатор. В нашем случае сеанс Telnet был открыт с рабочего стола CDE, и следующие команды приведут к запуску приложения калькулятора на удаленном узле с отображением окна приложения на локальном дисплее:

```
austin% DISPLAY=hurley:0.0; export DISPLAY
austin% calculator&
```

На рис. 2.7 отражен результат.

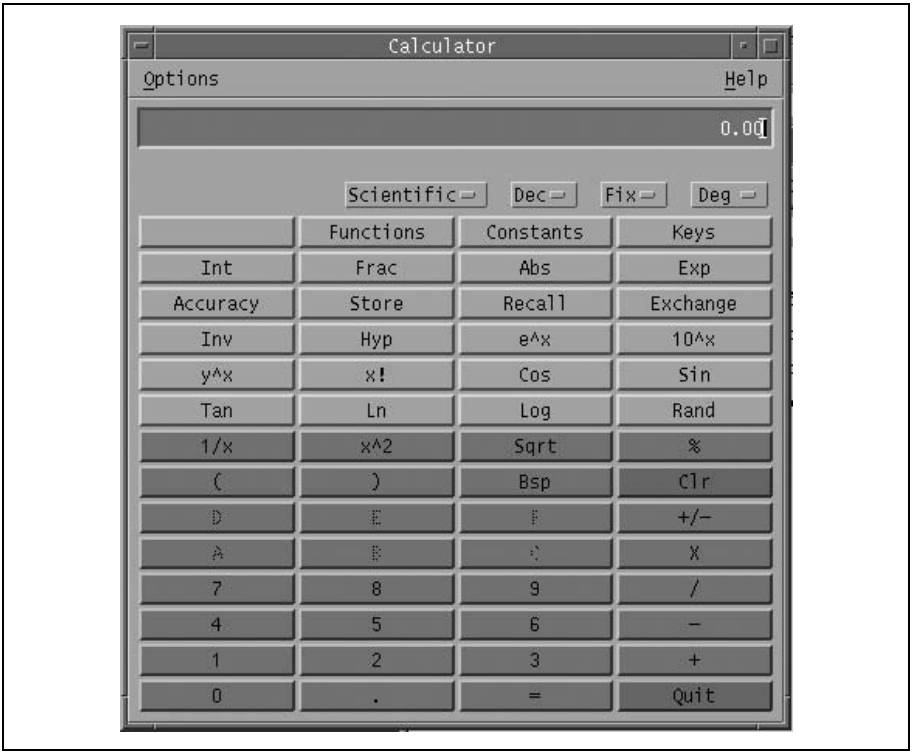

Рис. 2.7. *Приложение «Калькулятор», запущенное на удаленном сервере из-под CDE через сеанс Telnet*

В клиентах Telnet доступен ряд пользовательских команд (табл. 2.1). Эти команды доступны из меню клиента Telnet: управляющая последовательность (CTRL+]) или запуск клиента без имени узла позволяет получить доступ к этому меню.

Таблица 2.1. Команды клиента Telnet

Команда	Описание
!	Запуск командного интерпретатора на локальном узле
?	Вывод справки
Close	Завершение сеанса Telnet
Display	Отображение характеристик соединения
ENTER	Восстановление сеанса
Environ	Модификация переменных окружения
Logout	Завершение сеанса работы текущего пользователя, а затем завершение сеанса Telnet

Команда	Описание
Mode	Изменение режима работы (строчный или символьный)
Open	Создание соединения
Quit	Завершение сеанса Telnet
Send	Передача специальных символов
Set	Настройка характеристик соединения
Slc	Изменение режима для специальных символов
Status	Отображение состояния соединения
Toggle	Изменение характеристик соединения
Unset	Сброс установленных характеристик соединения
Z	Перевод сеанса в фоновый режим

Если, например, воспользоваться командой *display* для отображения характеристик текущего терминала, можно увидеть примерно такой вывод:

```
telnet> display
will flush output when sending interrupt characters.
won't send interrupt characters in urgent mode.
won't skip reading of ~/.telnetrc file.
won't map carriage return on output.
will recognize certain control characters.
won't turn on socket level debugging.
won't print hexadecimal representation of network traffic.
won't print user readable output for "netdata".
won't show option processing.
won't print hexadecimal representation of terminal traffic.
echo            [^E]
escape          [^]]
rlogin          [off]
tracefile       "(standard output)"
flushoutput     [^O]
interrupt       [^C]
quit            [^\]
eof             [^D]
erase           [^?]
kill            [^U]
lnext           [^V]
susp            [^Z]
reprint         [^R]
worderase       [^W]
start           [^Q]
stop            [^S]
forw1           [off]
forw2           [off]
ayt             [^T]
```

Протокол передачи файлов (FTP)

Еще одна популярная служба, работающая через *inetd*, – это протокол FTP (File Transfer Protocol, протокол передачи файлов), который широко используется для передачи файлов между узлами. Несмотря на способность службы FTP соединять клиента из одной страны с сервером из другой, сетевые администраторы обычно довольно косо смотрят на применение этой службы за пределами защищенных брандмауэрами подсетей; их в основном беспокоит, что имена пользователей и пароли могут быть перехвачены взломщиками, поскольку вся информация, связанная с проверкой подлинности (да и вообще все данные сеансов FTP), передается в открытом виде (то есть не шифруется). В главе 9 мы рассмотрим безопасные альтернативы FTP. Тем не менее FTP все еще часто применяется в локальных сетях для передачи файлов между системами.

Одним из применений службы является обеспечение анонимного FTP-доступа к архивам. И действительно, большинство архивов сети Интернет не требуют идентификации пользователя для подключения к анонимному FTP-серверу и получения файлов из общедоступного архива файлов. Учитывая вопросы безопасности, связанные с идентификацией и стандартным FTP, анонимный FTP – одно из разумных применений службы в современных сетях. Тем не менее при установке анонимного FTP-сервера следует проявить осторожность и изменить корневой каталог для анонимных пользователей с помощью команды *chroot*. Дело в том, что подключившимся пользователям доступны для чтения копии файла */etc/passwd*.

chroot ограничивает доступ к файлам из любого приложения некоторым подмножеством файловой системы, таким образом защищая важные данные от ненадежных клиентов. Более подробная информация по *chroot* доступна по адресу *http://hoohoo.ncsa.uiuc.edu/docs/tutorials/chroot.html*.

Из файла */etc/passwd* должны быть удалены все пользовательские записи, кроме самых необходимых, таких как запись пользователя, от которого обычно работает процесс демона FTP (например, пользователя *nobody*).

Вот так выглядит сеанс FTP:

```
hurley$ ftp austin
Connected to austin.
220 austin FTP server (SunOS 5.8) ready.
Name (austin:pwatters):
331 Password required for pwatters.
Password:
230 User pwatters logged in.
ftp>
```

Мы видим момент идентификации пользователя: клиент подключается с узла *hurley* к узлу *austin*, вводит регистрационное имя и пароль.

После проверки подлинности с положительным результатом запускается специальный командный интерпретатор FTP, в котором пользователь может вводить команды, перечисленные в табл. 2.2.

Таблица 2.2. Команды FTP

Команда	Описание
!	Запуск командного интерпретатора на локальном узле
$	Выполнение макрокоманды
?	Вывод справки
append	Добавление данных в конец файла
ascii	Передача файлов в формате ASCII
bell	Звуковой сигнал по завершению передачи
bye	Завершение сеанса работы по FTP
case	Разрешение преобразования регистра (символов в именах файлов)
cd	Смена каталога
cdup	Переход в родительский каталог
close	Завершение сеанса работы по FTP
cr	Разрешение удаления символа RETURN
delete	Удаление файла
dir	Вывод списка файлов в каталоге
disconnect	Завершение сеанса работы по FTP
form	Установка формата передачи данных ASCII/binary
get	Получение файла
glob	Разрешение интерпретации специальных символов в именах файлов
hash	Переключение отображения символа # в процессе передачи
help	Вывод справки
lcd	Смена каталога на локальной машине
ls	Вывод краткого списка файлов в каталоге
macdef	Начало записи макрокоманды
mdelete	Удаление нескольких файлов
mdir	Вывод списка файлов из нескольких каталогов
mkdir	Создание каталога
mls	Вывод краткого списка файлов из нескольких каталогов
mput	Передача нескольких файлов
nmap	Включение шаблонов
ntrans	Задание таблицы отображений символов
open	Создание соединения

Таблица 2.2 (продолжение)

Команда	Описание
prompt	Переключение диалогового режима работы с наборами файлов
proxy	Разрешение работы через посредника (proxying)
put	Передача файла
quit	Завершение сеанса работы по FTP
quote	Выполнение команды протокола FTP
remotehelp	Вывод справки
rename	Переименование файла
reset	Повторное соединение
rmdir	Удаление каталога
runique	Управление созданием уникальных имен файлов
send	Передача файла
sendport	Включение команды PORT
status	Отображение состояния
struct	Указание структуры передаваемых файлов
tenex	Включение режима передачи файлов TENEX
trace	Трассировка IP-пакетов
user	Регистрация пользователя в удаленной системе
verbose	Отображение подробных сообщений

При вводе неправильного пароля на стадии идентификации будет получено следующее сообщение об ошибке:

```
530 Login incorrect.
Login failed.
ftp>
```

Как могли заметить читатели, любой ответ FTP-сервера связан с конкретным кодом ответа (текст может быть различным в зависимости от реализации). Коды, приведенные в табл. 2.3, являются стандартными для всех реализаций FTP.

Таблица 2.3. Коды ответов FTP, Solaris

Код	Описание
110	Маркер ответа на запрос повторного соединения
120	Готовность службы через *nnn* минут
125	Канал данных уже открыт; начинается передача
150	Статус файла в порядке; открывается канал данных
200	Команда выполнена

Код	Описание
202	Команда не реализована, является излишней для данного сайта
211	Состояние системы или ответ на запрос справки
212	Состояние каталога
213	Состояние файла
214	Справочное сообщение
215	<ИМЯ>, тип системы
220	Служба готова к взаимодействию с новым пользователем
221	Служба закрывает управляющий канал
225	Канал данных открыт; данные не передаются
226	Закрывается канал данных
227	Переход в пассивный режим (h1,h2,h3,h4,p1,p2)
230	Пользователь вошел в систему, продолжение работы
250	Запрошенное действие успешно произведено для файла
257	Создан «ПУТЬ»
331	Имя пользователя получено, требуется пароль
332	Для входа в систему требуется учетная запись
350	Запрошенное для файла действие отложено, ожидается получение дополнительной информации
421	Служба недоступна, закрывается управляющий канал
425	Невозможно открыть канал данных
426	Соединение закрыто, передача прервана
450	Запрошенное действие не выполнено для файла
451	Выполнение действия прервано: локальная ошибка при обработке
452	Запрошенное действие не выполнено
500	Ошибка синтаксиса, команда не опознана
501	Ошибка синтаксиса в параметрах или аргументах
502	Команда не реализована
503	Некорректная последовательность команд
504	Команда не реализована для указанного параметра
530	Требуется регистрация в системе
532	Для хранения файлов требуется учетная запись
550	Запрошенное действие не выполнено
551	Выполнение запрошенного действия прервано: неизвестен тип страницы
552	Выполнение запрошенного для файла действия прервано
553	Запрошенное действие не выполнено

Для получения дополнительной информации о реализации конкретной службы *inetd* следует обратиться к документам RFC, касающимся искомой службы. Документы RFC (Request For Comments, запрос комментариев) обычно являются скорее документированными обсуждениями, нежели спецификациями, хотя поступающие комментарии, как правило, принимаются во внимание организацией IETF и органами стандартизации. Итак, чтобы подробнее узнать о проектировании FTP и о реализации ряда нововведений, обратитесь к следующим документам RFC:[1]

RFC 114

File Transfer Protocol.

RFC 2640

Internationalization of the File Transfer Protocol (Интернационализация FTP).

RFC 2389

Feature negotiation mechanism for the File Transfer Protocol (Механизм согласования возможностей для протокола передачи файлов).

RFC 1986

Experiments with a Simple File Transfer Protocol for Radio Links using Enhanced Trivial File Transfer Protocol (ETFTP) (Эксперименты с радиоканалами, передача файлов по упрощенным протоколам SFTP и ETFTP).

RFC 1440

SIFT/UFT: Sender-Initiated/Unsolicited File Transfer (Передача файлов по инициативе отправителя).

RFC 1068

Background File Transfer Program (BFTP) (Программа фоновой передачи файлов).

RFC 2585

Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP (Рабочие протоколы для X.509, интернет-инфраструктуры открытых ключей: FTP и HTTP).

RFC 2428

FTP Extensions for IPv6 and NATs (Расширения FTP для IPv6 и NAT).

RFC 2228

FTP Security Extensions (Расширения безопасности FTP).

---

[1] Все они доступны по адресу *http://www.rfc-editor.org*. – *Примеч. перев.*

RFC 1639

FTP Operation over Big Address Records (FOOBAR) (FTP, экспериментальный протокол для работы с длинными адресами).

Применение snoop

Если вам все еще не ясно, как передаются данные, воспользуйтесь командой *snoop*. *Snoop* позволяет «подслушивать» пакеты, передаваемые по локальной подсети, включая широковещательные пакеты, а также пакеты, передаваемые через соединение «точка-точка», по протоколам TCP, UDP, RPC и многим другим. Таким образом, *snoop* является весьма удобным инструментом для отладки, поскольку позволяет определить сам факт передачи пакетов между узлами и использование ожидаемых протоколов. Если сетевой трафик выглядит так, как ожидалось, любые ошибки в приложениях, связанные с сетевым взаимодействием, могут быть исправлены. При тестировании брандмауэра, фильтрующего пакеты, можно применять *snoop* для обнаружения ошибок в настройках, в результате которых через брандмауэр может проходить нежелательный трафик.

Большая часть сетевых инструментов Solaris может применяться и злоумышленниками; *snoop* не является исключением. Данный инструмент переводит сетевой интерфейс в режим приема всех поступающих пакетов (promiscuous mode), и есть возможность перехватить и отобразить содержимое пакетов либо записать в файл для последующего анализа. Соответственно, пользователь, работающий со *snoop*, может легко перехватывать имена пользователей и пароли, передаваемые по сети. Хорошая новость – *snoop* может выполняться только администратором системы (пользователем root); но есть и плохая – если взломщик получит права администратора на единственной локальной системе, он сможет перехватывать все пакеты от узлов, которые еще не подверглись атаке. Подобное наблюдение за пакетами рано или поздно может привести к перехвату паролей администраторов с других узлов. Более того, могут быть написаны сценарии, реагирующие на ключевые слова в пакетах, скажем, «su root», и автоматически извлекающие передаваемый следом пароль.

В стандартном режиме *snoop* отображает сводную информацию по трафику локальной сети.

```
# snoop
Using device /dev/eri (promiscuous mode)
      hurley -> austin          TELNET C port=1025
      austin -> hurley          TELNET R port=1025 Using device /dev/eri
      hurley -> austin          TELNET C port=1025
      austin -> hurley          TELNET R port=1025 hurley -> austin
      hurley -> austin          TELNET C port=1025
      austin -> hurley          TELNET R port=1025 austin -> hurley
      hurley -> austin          TELNET C port=1025
```

Можно видеть, что единственный сеанс, создающий трафик, связан с Telnet-соединением между узлами *hurley* и *austin*. Если включить средний уровень детализации, можно увидеть сводные характеристики каждого из пакетов:

```
# snoop -V
Using device /dev/eri (promiscuous mode)
_________________________________
hurley -> austin       ETHER Type=0800 (IP), size = 60 bytes
hurley -> austin       IP  D=10.64.18.3 S=10.64.18.1 LEN=40, ID=30981
hurley -> austin       TCP D=23 S=1025   Ack=653255051 Seq=4308772 Len=0
Win=8116
hurley -> austin       TELNET C port=1025
_________________________________
austin -> hurley       ETHER Type=0800 (IP), size = 83 bytes
austin -> hurley       IP  D=10.64.18.1 S=10.64.18.3 LEN=69, ID=60922
austin -> hurley       TCP D=1025 S=23   Ack=4308772 Seq=653255051 Len=2 9
Win=24820
austin -> hurley       TELNET R port=1025 /dev/eri (promiscuou
_________________________________
hurley -> austin       ETHER Type=0800 (IP), size = 60 bytes
hurley -> austin       IP  D=10.64.18.3 S=10.64.18.1 LEN=40, ID=31237
hurley -> austin       TCP D=23 S=1025   Ack=653255080 Seq=4308772 Len=0
Win=8087
hurley -> austin       TELNET C port=1025
_________________________________
```

Мы видим те же пакеты, что и раньше, но более детально; в частности, можно наблюдать тип протокола (IP), длину пакетов (60 байт), IP-адреса отправителя и получателя (`10.64.18.1` и `10.64.18.3`, соответственно), а также порт TCP (23). Но чтобы увидеть собственно содержимое пакетов, необходимо воспользоваться режимом полной детализации следующим образом:

```
# snoop -v
Using device /dev/eri (promiscuous mode)
ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 1 arrived at 20:53:38.80
ETHER:  Packet size = 60 bytes
ETHER:  Destination = 0:3:ba:4:a4:e8,
ETHER:  Source      = 0:50:ba:13:8:18,
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 throughurleyut
IP:         .... .0.. = normal reliability
IP:   Total length = 40 bytes
IP:   Identification = 22277
IP:   Flags = 0x4
IP:         .1.. .... = do not fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 32 seconds/hops
IP:   Protocol = 6 (TCP)
IP:   Header checksum = cb47
IP:   Source address = 10.64.18.1, hurley
IP:   Destination address = 10.64.18.3, austin
IP:   No options
IP:
TCP:  ----- TCP Header -----
TCP:
TCP:  Source port = 1025
TCP:  Destination port = 23 (TELNET)
TCP:  Sequence number = 4308741
TCP:  Acknowledgement number = 653252965
TCP:  Data offset = 20 bytes
TCP:  Flags = 0x10
TCP:        ..0. .... = No urgent pointer
TCP:        ...1 .... = Acknowledgement
TCP:        .... 0... = No push
TCP:        .... .0.. = No reset
TCP:        .... ..0. = No Syn
TCP:        .... ...0 = No Fin
TCP:  Window = 8739
TCP:  Checksum = 0x917a
TCP:  Urgent pointer = 0
TCP:  No options
TCP:
TELNET:  ----- TELNET:   -----
TELNET:
TELNET:  "s"
TELNET:
```

Здесь можно видеть четыре уровня модели OSI в реализации TCP/IP: *ETHER* на физическом уровне, *IP* на сетевом уровне, *TCP* на транспортном уровне и *TELNET* на прикладном уровне. Для каждого из уровней отражается детализированная информация. Так, на уровне *ETHER* мы видим время прибытия пакета, размер пакета, исходный и конечный MAC-адреса. На уровне *IP* – информация о том, что используется IPv4, длина заголовка пакета – 20 байт. Соответственно, длина сегмента содержимого пакета составляет 40 байт. Помимо значений TTL и параметров протокола можно отметить, что на уровне *IP* используются IP-адреса отправителя и получателя, а не MAC-адреса, как на уровне *ETHER*.

На уровне *TCP* отражены исходный порт (1025) и конечный порт (23), а также имя службы (*TELNET*). Помимо обмена sequence/acknowledgment (последовательность/подтверждение) доступны контрольная сумма и сдвиг для сегмента данных. И наконец, на уровне *TELNET* отражены символы содержимого пакета («*s*»). Очевидно, если последовательность пакетов, которыми обмениваются узлы, может быть отслежена, содержимое пакетов может быть извлечено и отображено на экране.

3

Установка Solaris

В этой главе приводятся пошаговые инструкции по установке и выбору значений основных параметров сетевых настроек для платформ Solaris Intel и Solaris SPARC. Существует три метода установки системы Solaris: из командной строки (текстовый интерфейс), диалоговый (на основе меню) и с помощью мастера Web Start (на основе Java). Мы изучим последний. Кроме того, мы затронем задачи планирования, выполнение которых необходимо для выбора корректных параметров настройки сети. Вы узнаете о распространенных ошибках при установке системы Solaris, а также о методах разрешения возникающих проблем.

Планируем установку

Установка системы Solaris – дело несколько более сложное, чем звонок в компьютерную фирму с целью заказать настольный компьютер с системой Windows Me (или аналогичной). Система под управлением Solaris, как правило, не применяется в качестве настольного компьютера, и установку Solaris скорее можно сравнить с выбором PC-сервера, работающего под управлением Windows 2000: серверу предстоит работать с файлами сотен пользователей или десятками разнообразных служб. В последнем случае специалист по PC-машинам, вне всякого сомнения, потратит значительное время на изучение архитектур различных материнских плат и типов процессоров, а также периферии, жизненно важной для эффективной работы, например карт SCSI, управляющих контроллерами RAID, или других сложных устройств.

Независимо от платформы (Solaris SPARC или Intel) выбор подходящего аппаратного обеспечения – это важный вопрос, который следует

решить до начала установки системы. В главе 8 мы обсудим методы измерения емкостных требований к серверам Solaris. Но объемы хранимой информации – лишь одно из неизвестных в уравнении: если система будет применяться в основном для проведения вычислений, возрастают требования к процессорной мощности, но не к дисковому пространству. И напротив, выбор самых быстрых дисковых накопителей в ущерб скорости работы процессоров может положительно повлиять на работу СУБД, использующей для хранения данных неразмеченные разделы (raw partitions). В идеальном варианте подобные жертвы не нужны, но если речь заходит о стоимости, а так обычно и бывает в случае серверных систем, способность убедить бюджетный комитет в оптимальном соотношении цена–качество для определенной конфигурации является весьма полезным умением!

Позже в этой главе мы рассмотрим некоторые из конкретных конфигураций аппаратной части для систем SPARC и Intel, которые поддерживаются текущей версией Solaris. Системы Sun, в отличие от большинства PC-устройств, обычно живут очень долго, и некоторые из них, уже вплотную подошедшие к десяти годам работы, по-прежнему поддерживаются существующей версией рабочей среды.

При планировании установки я рекомендую пользоваться тремя таблицами, сочетание которых является спецификацией создаваемой системы:

- Таблица настройки узла, в которую сведены все детали настройки узла и сети, в частности – имя узла, IP-адрес, имя домена, адрес сервера DNS, а также маска подсети.
- Таблица настройки аппаратного обеспечения, в которой перечислены устройства, установленные в машине, плюс дополнительные параметры, которые к этому моменту уже известны. Например, Ethernet-адрес сетевой карты мог быть установлен производителем, тогда этот адрес следует записать.
- Таблица настройки программного обеспечения, в которой перечислены все крупные пакеты, устанавливаемые в систему. Здесь может упоминаться пакет Standard Operating Environment (SOE), устанавливаемый на всех системах, а также отдельные пакеты, наборы которых варьируются для различных систем.

Сейчас мы более подробно рассмотрим каждую из этих таблиц на примере фирмы «Казуар компьютинг»[1] (Cassowary Computing).

---

[1] Казуары – отряд бескилевых птиц. Крупные, нелетающие, с трехпалыми ногами. Обитают в Австралии, Новой Гвинее и на прилежащих островах. Казуар шлемоносный (Casuarius casuarius) – самая тяжелая из ныне живущих птиц на Земле после африканского страуса (вес достигает 85 кг, высота 1м 80 см). – *Примеч. ред.*

Таблица настройки узла

Таблица настройки узла должна содержать все системные параметры, необходимые для настройки узла после его установки. Как правило, таблица содержит следующую информацию:

- Частное (unqualified) имя узла, которое должно быть уникальным в пределах локального домена, будь то домен NIS, NIS+ или DNS. Каждому узлу, не входящему в сеть, также следует присвоить имя.
- IP-адрес, соответствующий маске сети, – уникальный номер, позволяющий клиентам адресовать пакеты данному узлу.
- Имя домена DNS, связывающее узел с конкретным подразделением организации в пределах домена первого, второго или третьего уровня.
- Имя домена NIS+, используемое службой имен NIS+ для разрешения узлов и другими службами. Может совпадать с именем домена DNS.
- Полное доменное имя узла, которое состоит из имени узла и имени домена.
- IP-адрес сервера DNS, который отвечает на запросы к службе доменных имен, позволяя определять IP-адреса и имена узлов.
- IP-адрес сервера NIS+, который отвечает на запросы по именам узлов и за управление всеми службами NIS+.
- Маска подсети, определяющая масштабы локальной сети.
- Язык локализации системы. В настоящее время через стандартную установку доступны следующие языки:

  Английский
  Французский
  Немецкий
  Итальянский
  Японский
  Корейский
  Упрощенный китайский
  Испанский
  Шведский
  Традиционный китайский

Пример таблицы настройки узла приведен в табл. 3.1.

Таблица 3.1. Пример таблицы настройки узла

Таблица настройки узла	**Казуар компьютинг**
Узел	austin
IP-адрес	192.68.14.21

Таблица 3.1 (продолжение)

Таблица настройки узла	Казуар компьютинг
Полное доменное имя узла (FQDN)	austin.cassowary.net
Маска сети	255.255.255.0
Язык	Японский
Домен DNS	cassowary.net
Домен NIS+	Cassowary.Net.
Сервер DNS	192.68.14.1
Сервер NIS+	192.68.14.2
Сервер DHCP	192.68.14.3

На данном этапе в некоторых случаях указываются параметры подстройки, применимые к различным системам. Так, для сервера баз данных вроде Oracle практически наверняка потребуется существенное (если сравнивать со стандартными значениями Solaris) увеличение объема разделяемой памяти, прежде чем он сможет нормально работать. Необходимо будет вручную указать на данное обстоятельство в файле */etc/system*. Вот перечень наиболее часто применяемых параметров:

bufhwm

Максимальный объем памяти, занимаемый буферами кэширования операций ввода-вывода.

v.v_proc

Максимальное число процессов.

MAXCLSYSPRI

Максимальный глобальный приоритет для класса диспетчеризации процессов sys.

v.v_maxup

Максимальное число процессов для одного пользователя.

NAUTOUP

Интервал автоматического обновления в секундах.[1]

GPGSLO

Нижний порог, за которым начинается «кража» памяти у процессов.[2]

[1] В последних версиях Solaris этот параметр называется *autoup* и обозначает интервал между записями буферов открытых файлов на диск. – *Примеч. науч. ред.*

[2] В настоящее время этот параметр называется *tune_t_gpgslo* и является устаревшим. – *Примеч. науч. ред.*

FSFLUSHR

Число секунд между вызовами fsflush.

MINARMEM

Минимальный объем резидентной памяти, резервируемой для предотвращения взаимных блокировок.

MINASMEM

Минимальный объем памяти, резервируемой в разделе подкачки для предотвращения взаимных блокировок.

NSTRPUSH

Максимальное число STREAMS-модулей, помещаемых в поток.

STRMSGSZ

Максимальный размер сообщения в потоке.

STRCTLSZ

Максимальный размер управляющей части сообщения в потоке.

MSGMAP

Число записей в карте межпроцессных сообщений (IPC).

MSGMAX

Максимальный размер сообщения (IPC).

MSGMNB

Максимальное число байт в очереди (IPC).

MSGMNI

Максимальное число очередей сообщений (IPC).

MSGSSZ

Размер сегмента сообщения (IPC).

MSGTQL

Максимальное число сообщений (IPC).

MSGSEG

Число сегментов памяти для сообщения (IPC).

SEMMAP

Число записей в карте семафоров (IPC).

SEMMNI

Максимальное число идентификаторов семафоров (IPC).

SEMMNS

Максимальное число семафоров в системе (IPC).

SEMMNU

Общее число структур отката, поддерживаемых системой (IPC).

SEMMSL

Максимальное число семафоров на один идентификатор (IPC).

SEMOPM

Максимальное число операций на один вызов функции semop (IPC).

SEMUME

Максимальное число структур отката на один процесс (IPC).

SEMVMX

Максимальное значение для семафора (IPC).

SEMAEM

Максимальное значение для семафора в структуре отката (IPC).

SHMMAX

Максимальный размер сегмента разделяемой памяти.

SHMMIN

Минимальный размер сегмента разделяемой памяти.

SHMMNI

Максимальное число сегментов разделяемой памяти в системе.

SHMSEG

Максимальное число сегментов разделяемой памяти на один процесс.

TSMAXUPRI

Максимальный приоритет процессов пользователя в классе диспетчеризации с разделением времени (time share).

SYS_NAME

Название класса диспетчеризации для системных процессов и ядра.

Пример раздела, посвященного значениям описанных параметров, приведен в табл. 3.2.

Таблица 3.2. Пример набора значений параметров ядра

Параметр	**Значение**
Bufhwm	5120000
v.v_proc	3914
MAXCLSYSPRI	99
v.v_maxup	3909
NAUTOUP	30
GPGSLO	25
FSFLUSHR	5
MINARMEM	25

Параметр	Значение
MINASMEM	25
NSTRPUSH	9
STRMSGSZ	65536
STRCTLSZ	1024
MSGMAP	0
MSGMAX	0
MSGMNB	0
MSGMNI	0
MSGSSZ	0
MSGTQL	0
MSGSEG	0
SEMMAP	10
SEMMNI	100
SEMMNS	1000
SEMMNU	30
SEMMSL	50
SEMOPM	10
SEMUME	10
SEMVMX	32767
SEMAEM	16384
SHMMAX	50000000
SHMMIN	1
SHMMNI	100
SHMSEG	10
TSMAXUPRI	60
SYS_NAME	SYS

Таблица настройки аппаратного обеспечения

В таблицу настройки аппаратного обеспечения сведены все установленные в системе устройства и сопутствующая информация по их настройке, включающая параметры, установленные производителем, и сведения о назначении. Подобная таблица для сервера SPARC, как правило, включает следующую информацию:

- Тип системы
- Архитектура

- Идентификатор узла
- Тип процессора
- Тип графической подсистемы
- Объем оперативной памяти
- Тип шины
- Жесткие диски
- Приводы CD-ROM/DVD-ROM
- Сетевые интерфейсы

Таблица 3.3 содержит пример такого описания.

Таблица 3.3. Пример таблицы настройки аппаратного обеспечения

Таблица настройки аппаратного обеспечения	**Казуар компьютинг**
Тип системы	Ultra 5
Архитектура	sun4u
Идентификатор узла	91b0231a
Тип процессора	UltraSPARC-IIi 300 МГц
Графическая подсистема	CG3
Объем оперативной памяти	256 Мбайт
Тип шины	PCI
Жесткие диски	2x9,1 Гбайт, 10000 RPM
CD-ROM	32x
Сетевой интерфейс	/dev/hme0

Таблица настройки программного обеспечения

Основной причиной установки системы является необходимость обеспечения работы определенного рода служб на основе программного обеспечения. Таким образом, важно определить требования к ПО заранее, поскольку они во многом определяют выбор аппаратного обеспечения.

В число дистрибутивных носителей информации Solaris входит набор компакт-дисков с программами установки и приложениями сторонних разработчиков. Разумеется, лишь часть поставляемого ПО подлежит установке. В действительности собственно среда Solaris может устанавливаться в одном из четырех стандартных вариантов:

- Полный дистрибутив плюс OEM-поддержка (2,4 Гбайт дискового пространства)
- Полный дистрибутив без OEM-поддержки (2,3 Гбайт)
- Система для разработки (1,9 Гбайт)
- Пользовательская система (1,6 Гбайт)

В дистрибутив Solaris входят следующие диски:

- Интегрированная среда разработки Forte для Java
- Бесплатное ПО проекта GNU
- Программные пакеты семейства iPlanet
- Диск локализации и интернационализации
- Система управления базами данных Oracle 8i
- Сервер документации Answer Book
- Пакет офисных приложений Star Office 5.2
- Диски установки SunOS (CD 1 и CD 2)
- Диск мастера установки Web Start

Пример приведен в табл. 3.4.

Таблица 3.4. Пример таблицы настройки программного обеспечения

Таблица настройки ПО	Версия	Путь
Операционная система	SunOS 5.8	/
Тип установки	Полный дистрибутив с OEM-поддержкой	/
Сервер DNS (BIND)	9.0	*/opt/sfw*
Почтовый сервер (sendmail)	8.1	*/opt/sfw*
Брандмауэр (SunScreen)	2.0	*/opt*
Сервер POP (qpopper)	3.1	*/usr/local*
База данных (Oracle)	8.1.5	*/usr/local/ora*
Star Office	5.2	*/opt*
Сетевой интерфейс	*/dev/hme0*	

После установки часто бывает полезно добавить к таблице ПО вывод команды *pkginfo*, чтобы подтвердить установку нужных пакетов:

```
application GNUbash        bash
application SMCgcc         gcc
application SMClsof        lsof
application SMCmake        make
application SMCsamba       samba
system      SUNWab2m       Solaris Documentation Server Lookup
system      SUNWadmap      System administration applications
system      SUNWadmc       System administration core libraries
system      SUNWadmfw      System & Network Administration Framework
system      SUNWadmr       System & Network Administration Root
system      SUNWarc        Archive Libraries
system      SUNWatfsr      AutoFS, (Root)
system      SUNWatfsu      AutoFS, (Usr)
system      SUNWaudio      Audio applications
```

```
system      SUNWbcp         SunOS 4.x Binary Compatibility
system      SUNWbtool       CCS tools bundled with SunOS
system      SUNWcar         Core Architecture, (Root)
system      SUNWcg6         GX (cg6) Device Driver
system      SUNWcg6h        GX (cg6) Header Files
ALE         SUNWciu8        Simplified Chinese iconv modules for UTF-8
system      SUNWcpr         Suspend, Resume package
system      SUNWcsd         Core Solaris Devices
system      SUNWcsl         Core Solaris, (Shared Libs)
system      SUNWcsr         Core Solaris, (Root)
system      SUNWcsu         Core Solaris, (Usr)
```

Установка на платформе SPARC

Система SunOS изначально разрабатывалась для работы на процессорах SPARC (Scalable Processor Architecture, масштабируемая процессорная архитектура), поставляемых компанией Sun и другими ведущими производителями электроники, такими как Fujitsu (*http://www.fujitsu.com/*) и T.Sqware (*http://www.tsqware.com/*). Целый ряд других компаний занимается разработкой периферийных плат и прочих SPARC-совместимых устройств, например устройств для шины SBus (а не PCI). В частности, речь идет о Seagate (*http://www.seagate.com/*), Hitachi (*http://www.hitachi.com/*) и Kingston Technology (*http://www.kingston.com/*). Компания Sun не контролирует стандарты, которым соответствуют процессоры этих производителей, но отвечает за организацию работы участников рынка SPARC (*http://www.sparc.org/*). Стандарты, которым соответствуют SPARC-системы, доступны по адресу *http://www.sparc.org/standards.html*.

SunOS не похожа на прочие операционные системы в том плане, что одна и та же версия этой системы будет одинаково хорошо работать на системах различного уровня. Такое положение гарантируется для широкого спектра систем, которые могут различаться типами шины, скоростями процессоров, объемами памяти, типами дисковых накопителей и другими родственными аспектами. Даже различные версии SunOS в большинстве случаев совместимы между собой, несмотря на крупные изменения, произошедшие при переходе от SunOS 4 к SunOS 5.

В прошлом системы архитектуры SPARC ставили во главу угла скорость шины и объемы пропускаемых данных, а не скорость процессора. Например, в системе SPARC 10/51 установлен процессор, работающий на частоте 51 МГц, и это, по сегодняшним стандартам мира PC, может показаться ничтожной скоростью. Однако сочетание быстрой шины с интегрированным набором быстрых SCSI-дисков и оптимизацией системы ввода-вывода ставит SPARC-серверы даже предшествующих поколений выше менее надежных PC-машин в плане способности выдерживать длительные периоды непрерывной работы.

Диапазон платформ SPARC

Текущая версия SunOS 5.8 поддерживает перечисленные ниже системы SPARC:

Blade 100
Blade 1000
Sun Ray 100
Sun Ray 150
Enterprise 1
Enterprise 10000
Enterprise 150
Enterprise 2
Enterprise 250
Enterprise 3000
Enterprise 3500
Enterprise 4000
Enterprise 450
Enterprise 4500
Enterprise 5000
Enterprise 5500
Enterprise 6000
SPARCcenter 2000
SPARCclassic
SPARCserver 1000
SPARCstation 10
SPARCstation 20
SPARCstation 4
SPARCstation 5
SPARCstation LX
Ultra 1 (включая модели Creator и Creator 3D)
Ultra 10
Ultra 2 (включая модели Creator и Creator 3D)
Ultra 30
Ultra 450
Ultra 5
Ultra 60

К сожалению, некоторые из систем, с которыми работала SunOS 5.7, в SunOS 5.8 уже не поддерживаются. Вот эти системы:

SPARCstation 1

SPARCstation 2

Следует свериться с актуальным руководством по установке аппаратного обеспечения для конкретной платформы и убедиться в том, что аппаратная конфигурация поддерживается наиболее свежей версий операционной системы и рабочей среды.

Установка на платформе Intel

Изначально система Solaris была разработана для систем SPARC, но невероятная популярность систем на базе процессоров Intel, вне всякого сомнения, повлияла на решение Sun по расширению спектра целевого аппаратного обеспечения и включению в этот спектр PC-систем. Платформа Solaris стала доступна для совершенно другой аудитории – бывшим администраторам Microsoft Windows и Linux, которые модернизировали свои системы для работы с Solaris.

Если говорить об установке ПО, то подготовка к ней протекает по-разному для двух аппаратных платформ. Но применяемый метод установки (скажем, мастер Web Start) остается тем же. Таким образом, администраторы, имеющие опыт в установке системы на платформе SPARC, смогут использовать его при установке Solaris на платформе Intel, пусть даже им придется овладеть инструментом Device Configuration Assistant.

Выбор аппаратного обеспечения Intel

Система Solaris поддерживает широкий спектр устройств PC. Прежде всего, Sun и производители устройств поставляют драйверы и модули поддержки этих устройств, хотя есть возможность создать свои собственные драйверы. (Последний вариант значительно упрощается доступностью исходных текстов Solaris.) Кроме того, драйверы устройств для Linux, как правило, могут быть перенесены на платформу Solaris Intel с минимальными изменениями.

При создании системы с нуля нелишне будет обратиться к списку совместимых устройств (Hardware Compatibility List, HCL), доступному по адресу *http://soldc.sun.com/support/drivers/hcl/index.html*. В списке HCL приводятся многочисленные группы устройств, поддерживаемых на платформе Solaris Intel, и отдельные устройства в пределах каждой группы.

При этом в списке HCL упомянуто далеко не каждое устройство, с которым будет работать Solaris, поскольку многие классы изделий и их

вариантов взаимозаменяемые с перечисленными устройствами. Так, многие сетевые карты являются «NE2000-совместимыми», хотя не являются в действительности картами NE2000. Могут поддерживаться, пусть и несколько ограниченным образом, и другие устройства. Например, если ваша графическая карта при работе под управлением Microsoft Windows поддерживает разрешение 1024×768, при работе в Solaris Intel могут быть доступны лишь варианты более низкого разрешения (скажем, 800×600). Кроме того, если графическая карта не поддерживается системой, существует возможность выбора X-сервера XFree86 (*http://www.xfree86.org/*) из предлагаемых для использования с Solaris. Дело в том, что сервер XFree86 поддерживает большее число графических карт, чем стандартный сервер X11 Solaris. Рабочие среды Solaris, такие как CDE и Gnome, совместимы с XFree86, плюс оба сервера X11 (стандартный Solaris и XFree86) соответствуют стандарту X11R6.

В данном разделе мы рассмотрим некоторые из основных групп устройств, поддерживаемых на платформе Solaris Intel, а также примеры наиболее распространенных системных конфигураций.

Системы под ключ

Перечисленные ниже системы были протестированы и сертифицированы компанией Sun для работы под управлением Solaris 8:

- Acer AcerAcros T7000 MT
- Bull Information Systems Express5800-HX4500
- Compaq Deskpro EN 6400
- Dell OptiPlex G1
- Fujitsu FMV-5166D9K
- Hitachi HA8000/150
- HP NetServer LHII
- IBM IntelliStation E Pro 6893
- IBM Netfinity 3500 8644-21U
- Intel SKA4
- Motorola CPV5000 Single-Board Computer
- NCR 3271
- NEC Express5800-HX4500
- Siemens AG PRIMERGY 170
- Toshiba Magnia 3000
- Zenith Data Systems Express5800-ES1200

Материнские платы

Помимо систем под ключ поддерживаются следующие классы материнских плат:

- Acer M9N MP
- ASUS A7V
- EPoX EP-MVP3G
- Intel JN440BX

В дополнение к этим однопроцессорным материнским платам Solaris поддерживает многопроцессорные варианты. На материнской плате класса SMP (Symmetric Multiprocessing, симметричная многопроцессорная система) может работать до восьми процессоров. Ни одна операционная система не дает линейного масштабирования при увеличении числа процессоров, но производительность Solaris SMP приближается к подобным показателям. Вот некоторые из наиболее широко применяемых материнских плат SMP:

- Acer AcerAltos 21000
- Bull Information Systems Express5800-HX4500
- Compaq Professional Workstation 8000
- Dell PowerEdge 6300
- Fujitsu TeamSERVER-T890i
- Gateway 8400
- Hitachi VisionBase8880R UWRAID (8 процессоров)
- HP Kayak XU 6-300 PC Workstation
- Intel SPM8

Графические карты и мониторы

Поддерживаются перечисленные ниже карты с интерфейсами ISA, PCI и AGP:

- ATI 3D RAGE
- Boca Voyager 64
- Chips & Technology 65540
- Cirrus Logic 5420
- Creative Labs 3D Blaster RIVA TNT2
- Diamond Fire GL 1000 Pro
- ELSA Victory 3D
- Everex FIC 864P
- Hercules Dynamite 128/Video
- Intergraph G95

- Matrox Millennium
- NVIDIA TNT2

Многие из этих карт дают возможность работать в различных разрешениях, с глубиной цвета 8 и 24 бита:

- 800×600
- 1024×768
- 1152×900
- 1280×1024
- 1600×1200

Максимально достижимое разрешение для конкретной графической карты зависит еще от размера и частотных характеристик монитора. Solaris поддерживает как мониторы многократного сканирования (multisync), так и мониторы с фиксированными частотами.

Манипулятор «мышь»

Система Solaris поддерживает все распространенные варианты манипулятора «мышь» с интерфейсами PS/2, serial и bus. Кроме того, поддерживаются специальные виды устройств, в частности:

- InterLink PortAPoint
- Kraft Systems MicroTrack
- LogiTech MouseMan (беспроводная)
- MicroSpeed MicroTRAC (трекбол)

Контроллеры SCSI

Система Solaris Intel работает с широким диапазоном контроллеров SCSI; то же верно и для систем SPARC, что позволяет повторно использовать и разделять компоненты между системами. Вот наиболее популярные контроллеры SCSI:

- Adaptec AHA-2940/2940W
- AMD Pcscsi
- Compaq 32-bit Fast-Wide SCSI-2
- DPT PM2024
- DTC DTC-3130
- Hitachi PC-CS7210
- Intel PCISCSI
- LSI Logic NCR 53C810
- QLogic QLA510

Устройства Zip/Jaz

Приводы Zip и Jaz представляют собой устройства, работающие со съемными дисками, объем хранимой информации для которых варьируется от 100 Мбайт до 2 Гбайт; существуют приводы с интерфейсами IDE, ATAPI и SCSI:

- ATAPI Z100A Zip (100 Мбайт)
- IDE Z100A Zip (100 Мбайт)
- SCSI 2250S Zip (250 Мбайт)
- SCSI V2008I Jaz (2 Гбайт)

Сетевые карты

Будучи сетевой операционной системой, Solaris во многом полагается на карты сетевых интерфейсов, позволяющие объединять узлы в сеть Ethernet, а узлам – обмениваться данными со скоростями до 100 Мбит/с. Полностью поддерживаются следующие карты:

- 3Com EtherLink III PCI Bus Master
- Adaptec ANA-6901
- AMD PCnet-PCI
- CNet CN970EBT
- Cogent EM110 T4
- Compaq Deskpro 4000 NetFlex-3
- DEC EtherWORKS
- D-Link DE-530CT
- HP PC LAN NC/16 TP
- Intel EtherExpress PRO/10+
- Kingston KNE40BT
- Linksys LNE100TX
- Samsung SEB-3000C
- SMC EtherPower SMC8432BT

Карты PCMCIA

Хотя применение Solaris на портативных компьютерах сопряжено с некоторыми особенностями, связанными в основном с типами дисплеев и управлением питанием, многие из применяемых в таких компьютерах PCMCIA-карт полностью совместимы с Solaris. В виде карт PCMCIA часто реализуются такие устройства, как модемы и сетевые карты. Вот некоторые из устройств, поддерживаемых Solaris:

- ATI Technologies 14400 ETC-EXPRESS
- Compaq SpeedPaq 192

- Hayes 5361US
- Intel 110-US
- Kingston DataRex 87G9851
- US Robotics Sun/USR WorldPort

Подготовка к установке (SPARC)

Чтобы начать установку Solaris SPARC, включите систему и нажмите STOP+a после появления загрузочных сообщений. Как вариант, если существующая система уже работает в определенном режиме, воспользуйтесь следующей командой, чтобы перейти на уровень загрузочных тестов:

```
# sync; init 0
```

Вставьте установочный диск Web Start в привод CD-ROM и выполните команду:

```
ok boot cdrom
```

Ядро загрузится в режиме обновления настроек:

```
Boot device: /sbus/espdma@e,8400000/esp@e,8800000/sd@6,0:f File and args:
SunOS Release 5.8 Version Generic 32-bit
Copyright 1983-2000 Sun Microsystems, Inc. All rights reserved.
Configuring /dev and /devices
Using RPC Bootparams for network configuration information.
Solaris Web Start 3.0 installer
English has been selected as the language in which to perform the install.
Starting the Web Start 3.0 Solaris installer
Solaris installer is searching the system's hard disks for a
location to place the Solaris installer software.
Your system appears to be upgradeable.
Do you want to do a Initial Install or Upgrade?
1) Initial Install
2) Upgrade
Please Enter 1 or 2 >
```

Если планируется обновление и необходимо сохранить данные и существующие файловые системы, выберите вариант 2. Чтобы начать установку с начала, воспользуйтесь вариантом начальной установки (initial install). В последнем случае следует произвести резервное копирование всех данных, которые требуется перенести на свежеустановленную систему, поскольку все существующие данные будут уничтожены. Отметим, что вариант обновления системы доступен только в том случае, если система может обновляться.

Чтобы выбрать начальную установку, введите цифру 1. После нажатия клавиши <Enter> отображаются следующие сообщения:

```
The default root disk is /dev/dsk/c1t0d0.
The Solaris installer needs to format
```

```
/dev/dsk/c1t0d0 to install Solaris.
WARNING: ALL INFORMATION ON THE DISK WILL BE ERASED!
Do you want to format /dev/dsk/c1t0d0? [y,n,?,q]
```

На этом этапе следует подтвердить установку новой версии Solaris, разрешив полную перезапись содержимого загрузочного диска (в данном случае */dev/dsk/c1t0d0*). Подтвердив операцию форматирования, вы должны выбрать размер раздела подкачки из установленного интервала:

```
NOTE: The swap size cannot be changed during filesystem layout.
Enter a swap slice size between 384MB and 2027MB, default = 512MB [?]
```

Если в системе установлен 1 Гбайт физической оперативной памяти, 2048 Мбайт – вполне адекватное значение для объема раздела подкачки. В общем случае объем виртуальной памяти должен в два раза превышать объем физической памяти. Для системы, в которой установлено 512 Мбайт физической памяти, следует создать раздел подкачки объемом в 1024 Мбайт. Если объем физической оперативной памяти изменится после установки, можно воспользоваться командой *swap* для выделения дополнительных сегментов под виртуальную память.

Поскольку размер раздела подкачки не может быть изменен в процессе формирования файловых систем, разместите его в начале дискового пространства, как рекомендуют приведенные строки:

```
The Installer prefers that the swap slice is at the beginning of the
disk. This will allow the most flexible filesystem partitioning later in the
installation.
Can the swap slice start at the beginning of the disk [y,n,?,q]
```

После размещения раздела подкачки следует завершить операцию сохранением настроек:

```
You have selected the following to be used by the Solaris installer:
Disk Slice : /dev/dsk/c1t0d0
Size : 2048 MB
Start Cyl. : 0
WARNING: ALL INFORMATION ON THE DISK WILL BE ERASED!
Is this OK [y,n,?,q]
```

Если вас устраивают настройки, введите «y». На этом этапе происходит форматирование диска, создание корневой файловой системы, копирование платформо-зависимых файлов и, наконец, перезагрузка системы. После перезагрузки начинается установка системы с помощью мастера Web Start.

```
The Solaris installer will use disk slice, /dev/dsk/c0t0d0s1.
After files are copied, the system will automatically reboot, and
installation will continue.
Please Wait...
```

```
Copying mini-root to local disk....done.
Copying platform specific files....done.
Preparing to reboot and continue installation.
Rebooting to continue the installation.
Syncing file systems... 41 done
rebooting...
Resetting ...
SPARCstation 10 (2 X 390Z50), Keyboard Present
ROM Rev. 2.4, 128 MB memory installed, Serial #34341
Ethernet address 3:a:1b:a:33:1a HostID 32433
Rebooting with command: boot /sbus@1f,0/espdma@e,8400000/esp@e,8800000/sd@0,1:b
Boot device: /sbus@1f,0/espdma@e,8400000/esp@e,8800000/sd@0,1:b File and
args:
SunOS Release 5.8 Version Generic 32-bit
Copyright 1983-2000 Sun Microsystems, Inc. All rights reserved.
Configuring /dev and /devices
Using RPC Bootparams for network configuration information.
```

Далее установка системы следует независимым от платформы процедурам, описанным в разделе «Установка с помощью мастера Web Start».

Подготовка к установке (Intel)

В этом разделе мы изучим применение инструмента Device Configuration Assistant и родственных приложений в целях подготовки системы Intel к установке и использованию Solaris. После сборки машины, на которую производится установка, ее следует включить и поместить диск установки в дисковод CD-ROM. Возможно, потребуется изменить настройки BIOS, разрешив загрузку с CD до загрузки с жесткого диска, особенно если на последний уже установлена другая операционная система. Очевидно, после завершения установки следует вернуть настройки в прежнее состояние, чтобы происходила нормальная загрузка с жесткого диска.

После включения системы и начала загрузки с компакт-диска должны появиться следующие сообщения:

```
SunOS Secondary Boot version 3.00
Solaris Intel Platform Edition Booting System
Running Configuration Assistant...
```

Происходит инициализация вспомогательного инструмента настройки аппаратуры (Device Configuration Assistant, или DCA). DCA сканирует системную шину, определяет ее тип и конфигурацию установленных устройств. После завершения сканирования перечень обнаруженных устройств отображается на экране:

```
ISA: Floppy disk controller
ISA: IDE controller
ISA: IDE controller
```

```
ISA: Motherboard
ISA: PS/2 Mouse
ISA: PnP bios: 16550-compatible serial controller
ISA: PnP bios: 8514-compatible display controller
ISA: PnP bios: Audio device
ISA: System keyboard (US-English)
```

Если существующее в системе устройство не было включено в список, тому может быть две причины: устройство не поддерживается Solaris либо поддерживается, но для продолжения установки параметры его настройки необходимо изменить. Для изменения характеристик устройства воспользуйтесь разделом «Device Task» (Работа с устройством). Помимо изменения настроек в этом разделе можно устанавливать различные параметры устройств ввода (например, клавиатуры) и консоль, используемую по умолчанию. Как вариант информация о настройках может быть считана с диска, а текущие настройки могут быть записаны на диск. Эта возможность весьма полезна в случае параллельной установки и настройки нескольких совместимых систем.

После нажатия клавиши <F2> необходимо произвести настройку жесткого диска для работы с Solaris, особенно в том случае, если до этого на диске не существовало файловых систем UFS. Отметим, что поскольку загрузочные разделы Solaris и разделы подкачки Linux имеют одинаковые сигнатуры типа, вы должны быть в состоянии различать такие разделы, если на машине установлены обе системы. Во-первых, можно изучить размер разделов, которые описываются, как принадлежащие Unix-системам; если размер раздела 127 Мбайт, это практически наверняка раздел подкачки Linux, поскольку это максимально возможный размер такого раздела. Перед запуском программы *fdisk* (служащей для разбиения диска на разделы) отображается состояние дисковой системы:

```
<<< Current Boot Parameters >>>
Boot path: /pci@1,0/pci-ide@6,1/ide@2/sd@1,0:a
Boot args: kernel/unix
<<< Starting Installation >>>
SunOS Release 5.8 Version Generic 32-bit
Copyright 1983-2000 Sun Microsystems, Inc. All rights reserved.
Configuring /dev and /devices
Using RPC Bootparams for network configuration information.
Solaris Web Start 3.0 installer
English has been selected as the language in which to perform the install.
Starting the Web Start 3.0 Solaris installer
Solaris installer is searching the system's hard disks for a
location to place the Solaris installer software.
No suitable Solaris fdisk partition was found.
Solaris Installer needs to create a Solaris fdisk partition
on your root disk, c0d0, that is at least 395 MB.
WARNING: All information on the disk will be lost.
May the Solaris Installer create a Solaris fdisk [y,n,?]
```

Если речь идет о начальной установке (initial install), воспользуйтесь программой *fdisk*, чтобы соответствующим образом распределить дисковое пространство. Если в системе уже существуют разделы с данными либо другая операционная система, следует убедиться, что эти разделы не будут перезаписаны.

После инициализации *fdisk* отображает информацию следующего рода:

```
Total disk size is 4096 cylinders
Cylinder size is 8064 (512 byte) blocks
Cylinders
Partition  Status  Type  Start  End  Length  %
=========  ======  ====  =====  ==== ======  ===
1                  DOS    0     2047  2048   50
2                  DOS    2048  4095  2048   50
SELECT ONE OF THE FOLLOWING:
1. Create a partition
2. Specify the active partition
3. Delete a partition
4. Exit (update disk configuration and exit)
5. Cancel (exit without updating disk configuration)
Enter Selection:
```

В системе из примера один диск; число цилиндров – 4096, каждый состоит из 8064 блоков по 512 байт. На диске существует пара разделов DOS одинакового размера. Если не предполагается наличие второй системы, использующей существующие DOS-разделы, их можно смело удалить с помощью пункта меню 3 (Delete a partition). Удалив существующие разделы, следует создать один или несколько новых разделов UFS с помощью пункта меню 1 (Create a partition). По меньшей мере один раздел должен иметь тип «x86 Boot», а дополнительным разделам UFS может быть присвоен тип «SOLARIS», что отражено в соответствующем меню:

```
Select the partition type to create:
1=SOLARIS 2=UNIX 3=PCIXOS 4=Other
5=DOS12 6=DOS16 7=DOSEXT 8=DOSBIG
A=x86 Boot B=Diagnostic 0=Exit?
```

Чтобы создать загрузочный раздел из 1024 цилиндров (то есть занимающий 25% объема диска), следует ввести 25 или 1024c в ответ на следующее приглашение:

```
Specify the percentage of disk to use for this partition
(or type "c" to specify the size in cylinders).
```

Рекомендуется логически разбить диск на несколько разделов, особенно если размеры разделов были оценены посредством описанных в главе 8 процедур. В частности, помимо загрузочного раздела рекомендуется создать самостоятельный раздел */var*.

После создания разделов укажите активный (загрузочный) раздел с помощью пункта меню 2 (Specify the active partition). С этого раздела будет производиться загрузка после включения системы. По умолчанию система Solaris записывает код менеджера загрузки Solaris в загрузочный сектор, но можно воспользоваться любым другим менеджером, таким как Linux LILO.

Затем следует указать, будет ли создаваемый раздел активным. Это означает, что система попытается загрузить операционную систему по умолчанию с этого раздела. Если вы намереваетесь пользоваться менеджером загрузки Solaris, можно активировать этот раздел. Но если планируется применение Boot Magic или LILO для работы с существующими разделами Microsoft Windows или Linux, следует дать отрицательный ответ.

После определения загрузочного раздела в обновленном меню появятся некоторые из перечисленных разделов:

```
2 Active x86 Boot 8 16 9 1
Total disk size is 4096 cylinders
Cylinder size is 8064 (512 byte) blocks
Cylinders
Partition  Status  Type       Start  End  Length  %
=========  ======  =========  =====  ==== ======  ===
1          Active   x86 Boot   0     1023 1024    25
2                   SOLARIS    1024  2047 1024   25
3                   SOLARIS    2048  4095 2048   50
SELECT ONE OF THE FOLLOWING:
1. Create a partition
2. Specify the active partition
3. Delete a partition
4. Exit (update disk configuration and exit)
5. Cancel (exit without updating disk configuration)
Enter Selection:
```

Когда созданы все разделы и выбран активный, можно сохранить внесенные изменения с помощью пункта меню 4 (Exit). Если вы не уверены в сделанных изменениях, отмените их с помощью пункта меню 5 (Cancel). При выборе пункта 4 происходит перезапуск программы установки и появляется следующий вывод:

```
<<< Current Boot Parameters >>>
Boot path: /pci@1,0/pci-ide@6,1/ide@2/sd@1,0:a
Boot args: kernel/unix
<<< Starting Installation >>>
SunOS Release 5.8 Version Generic 32-bit
Copyright 1983-2000 Sun Microsystems, Inc. All rights reserved.
Configuring /dev and /devices
Using RPC Bootparams for network configuration information.
Solaris Web Start 3.0 installer
English has been selected as the language in which to perform the install.
```

```
Starting the Web Start 3.0 Solaris installer
Solaris installer is searching the system's hard disks for a
location to place the Solaris installer software.
The default root disk is /dev/dsk/c0d0.
The Solaris installer needs to format
/dev/dsk/c0d0 to install Solaris.
WARNING: ALL INFORMATION ON THE DISK WILL BE ERASED!
Do you want to format /dev/dsk/c0d0? [y,n,?,q]
```

Поскольку диск должен быть отформатирован, прежде чем система сможет им воспользоваться, выберите «y». На данном этапе необходимо подтвердить начальную установку Solaris и перезапись содержимого загрузочного диска (в нашем примере это */dev/dsk/c0d0*). Теперь необходимо выбрать размер раздела подкачки из определенного интервала:

```
NOTE: The swap size cannot be changed during filesystem layout.
Enter a swap slice size between 384MB and 2027MB, default = 512MB [?]
```

Если в системе установлен один гигабайт физической оперативной памяти, 2048 Мбайт – вполне адекватное значение для объема раздела подкачки. В общем случае объем виртуальной памяти должен в два раза превышать объем физической оперативной памяти. Для системы, в которой установлено 512 Мбайт физической памяти, следует создать раздел подкачки объемом в 1024 Мбайт. Если объем физической оперативной памяти изменится после установки, можно воспользоваться командой *swap* для выделения дополнительных сегментов под виртуальную память.

Поскольку размер раздела подкачки не может быть изменен в процессе формирования файловых систем, разместите его в начале дискового пространства, как рекомендуют приведенные строки:

```
The Installer prefers that the swap slice is at the beginning of the
disk. This will allow the most flexible filesystem partitioning later in the
installation.
Can the swap slice start at the beginning of the disk [y,n,?,q]
```

После размещения раздела подкачки следует завершить операцию сохранением настроек:

```
You have selected the following to be used by the Solaris installer:
Disk Slice : /dev/dsk/c0d0
Size : 2048 MB
Start Cyl. : 0
WARNING: ALL INFORMATION ON THE DISK WILL BE ERASED!
Is this OK [y,n,?,q]
```

Если вас устраивают настройки, введите «y». На этом этапе происходит форматирование диска, создание корневой файловой системы, копирование платформо-зависимых файлов и, наконец, перезагрузка системы. После перезагрузки начинается установка с помощью мастера Web Start.

```
The Solaris installer will use disk slice, /dev/dsk/c0d0s1.
After files are copied, the system will automatically reboot, and
installation will continue.
Please Wait...
Copying mini-root to local disk....done.
Copying platform specific files....done.
Preparing to reboot and continue installation.
Need to reboot to continue the installation
Please remove the boot media (floppy or cdrom) and press Enter
Note: If the boot media is cdrom, you must wait for the system
to reset in order to eject.
```

Нажав клавишу <Enter>, вы увидите стандартные сообщения, выдаваемые Solaris при остановке системы, включая и это:

```
Syncing file systems... 49 done
rebooting...
```

После перезагрузки должен запуститься менеджер загрузки, позволяющий загрузиться с активного раздела, как показано ниже:

```
SunOS - Intel Platform Edition Primary Boot Subsystem, vsn 2.0
Current Disk Partition Information
Part#   Status  Type      Start  Length
=======================================
1       Active  X86 BOOT  0      1024
Please select the partition you wish to boot:
```

Выбор пункта 1 загружает операционную систему с активного раздела:

```
SunOS Secondary Boot version 3.00
Solaris Intel Platform Edition Booting System
Running Configuration Assistant...
Autobooting from boot path: /pci@1,0/pci-ide@6,1/ide@2/sd@1,0:a
If the system hardware has changed, or to boot from a different
device, interrupt the autoboot process by pressing ESC.
```

Система инициализируется:

```
Initializing system
Please wait...
<<< Current Boot Parameters >>>
Boot path: /pci@0,0/pci-ide@7,1/ata@1/cmdk@0,0:b
Boot args:
Type b [file-name] [boot-flags] <ENTER> to boot with options
or i <ENTER> to enter boot interpreter
or <ENTER> to boot with defaults
<<< timeout in 5 seconds >>>
Select (b)oot or (i)nterpreter:
SunOS Release 5.8 Version Generic 32-bit
Copyright 1983-2000 Sun Microsystems, Inc. All rights reserved.
Configuring /dev and /devices
Using RPC Bootparams for network configuration information.
```

Последний шаг подготовки связан с приложением *kdmconfig*, которое позволяет настроить графическую подсистему для работы с мастером Web Start. Если видеокарта поддерживается системой, вы сможете протестировать ее настройки до того, как будет запущен мастер.

Установка с помощью мастера Web Start

Существует два способа выполнить установку Solaris: в диалоговом режиме или с помощью мастера Web Start. В предшествующих версиях Solaris применялась программа диалоговой установки, работа с которой сводилась к ответам на вопросы в текстовом режиме. Мастер Web Start предоставляет графический интерфейс пользователя на базе языка Java: с его помощью установка упрощается, и, кроме того, становятся доступны более удобные операции редактирования, в частности «копирование и вставка». Мастер позволяет устанавливать целый дистрибутив либо выбирать отдельные пакеты.

При работе с мастером установка делится на несколько самостоятельных этапов:

- Сеть
- Служба имен
- Дата и время
- Пароль администратора
- Управление питанием
- Proxy-сервер
- Киоск

Для выполнения установки необходимо будет воспользоваться данными из таблиц настройки узла, программного и аппаратного обеспечения. Во многих случаях значения из этих таблиц могут без изменений переноситься в настройки мастера. Например, значения IP-адреса, имени узла и маски сети для узла следует ввести в разделе Network (Сеть). Если для получения IP-адреса узел обращается к серверу DHCP, следует ввести IP-адрес этого сервера. Если в дополнение к IPv4 в локальной сети используется также IPv6, следует указать на это обстоятельство.

Следующий этап связан с заполнением параметров серверов служб имен DNS и NIS+ в разделе Name Service (Служба имен). Можно ввести IP-адрес основного сервера DNS и не более двух адресов дополнительных серверов. Кроме того, следует указать IP-адрес локального сервера NIS+.

Когда с настройками сети покончено, необходимо указать локальный часовой пояс, чтобы система установила подходящую дату и время. Если дата и время не соответствуют действительности, исправьте значения на данном этапе.

Далее предстоит указать пароль для пользователя root. Поскольку этот пользователь обладает полномочиями администратора, следует проявить изобретательность при выборе пароля. Пароль не должен быть словом из словаря (язык не имеет значения) и в идеальном случае представляет собой строку случайных символов, которую легко запомнить.

Если планируется использовать управление питанием для узла, необходимо явно указать на это в разделе Power Management (Управление питанием). Но для всех серверов следует отключать управление питанием, поскольку доступ к службам прекратится, если сервер перейдет в режим энергосбережения.

Если в сети используется proxy-сервер, через который организован доступ в сеть Интернет, следует указать его IP-адрес; в противном случае узел не сможет подключиться к Интернету. Некоторые узлы подключаются к сети Интернет напрямую, но в большинстве сетей из соображений безопасности доступ ограничивается посредством proxy-сервера.

И наконец, Киоск отвечает за копирование дистрибутивов и выбранных пакетов на указанный диск. Чтобы установить пакеты, поместите в привод компакт-диск с программным обеспечением. После выбора дистрибутивов и/или пакетов диски будут организованы в соответствии со сделанными настройками. Осталось лишь подтвердить выбор программного обеспечения: на диск будут скопированы соответствующие файлы, и система перезагрузится. Если установка прошла корректно, появится рабочий стол CDE.

4

Настройка сетей

Учитывая сложность задач, связанных с настройкой отдельного узла, планирование и настройка целой сети может выглядеть устрашающе. В этой главе вы узнаете о настройке сети из машин под управлением Solaris, о настройке сетевых интерфейсов в различных конфигурациях, о статической и динамической маршрутизации, а также о способах диагностирования и разрешения сетевых проблем. В дополнение к этому материалу приводятся примеры инициализации устройств и проверки интерфейсов.

Создание сетей и подсетей

Системы под управлением Solaris способны работать в изоляции от сетевой среды, но Solaris все-таки является операционной системой, ориентированной на работу в сети, и включает поддержку сетевого взаимодействия узлов локальной сети между собой и Интернетом, а именно:

- Поддержка устройств ethernet с одним, двумя или четырьмя интерфейсами
- Стандартизированная система именования сетевых устройств
- Поддержка широкого спектра сетевых устройств
- Настройка интерфейсов для работы по протоколам IPv4 и IPv6
- Маршрутизация по статическим и динамическим протоколам
- Диагностика и разрешение проблем, измерение производительности
- Фильтрация пакетов по всем TCP- и UDP-портам
- Передача данных по Ethernet и FDDI

- Поддержка сетей ATM (Asynchronous Transfer Mode), работающих в режиме асинхронной передачи данных

Произвольные сочетания этих возможностей упрощают создание сетей на основе Solaris и в особенности сетей, в которых системы под управлением Solaris играют ведущую роль в маршрутизации и фильтрации пакетов.

Типичная локальная сеть Solaris включает один или несколько серверов, которые предоставляют локальным клиентам доступ к сетевым службам. Клиентами могут быть другие системы Solaris, но с равной вероятностью – системы под управлением Linux, Microsoft Windows либо различных вариантов Unix. В некоторых сетях каждая из служб работает на специально отведенной под нее машине, что позволяет обеспечивать доступ к большинству служб в случаях, когда один из серверов прекращает работу. Такой вид разделения ролей серверов стал еще более эффективным с появлением системы E10000, которая позволяет создавать до 64 виртуальных серверов, работающих на одной машине. Если работа одного из доменов приостанавливается с целью выполнения задач техобслуживания, остальные домены продолжают работать.

Число и виды служб, которые могут работать в локальной сети Solaris, можно перечислять практически бесконечно, но в типичном варианте используются следующие:

Почтовый сервер	Сервер идентификации
Сервер USENET	Сервер управления ресурсами
UNIX-совместимый файловый сервер	Сервер удаленного доступа
PC-совместимый файловый сервер	Сервер RPC
Сервер резервного копирования	Сервер WWW
Сервер печати	Сервер каталогов

Эти виды служб в Solaris представлены следующими пакетами:

Sendmail	Kerberos
Inn	NIS+
NFS	Telnet и FTP
Samba	Демон RPC
Netbackup	Apache
Системы печати System V и BSD	LDAP

На рис. 4.1 приведен пример конфигурации сервера для сети класса C.

Когда функции сервера для локальной сети определены, необходимо решить ряд других вопросов, связанных, в частности, с выбором диапазонов IP-адресов для отдельных подсетей и IP-адресов отдельных узлов (это описано в главе 2). Современные сети, как правило, создают-

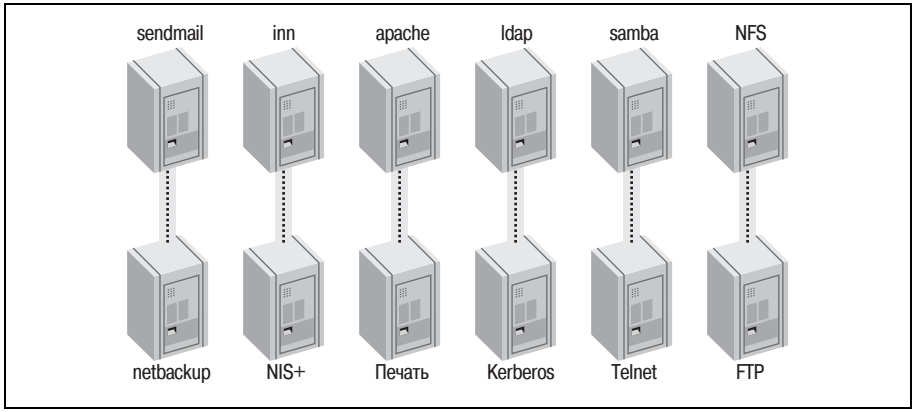

Рис. 4.1. *Пример конфигурации сервера Solaris для сети класса C*

ся на основе 10- и 100-мегабитных Ethernet-соединений; при этом узлы одной подсети связаны с единственным маршрутизатором посредством коммутатора или концентратора. На рис. 4.2 показана самостоятельная локальная сеть: узлы *chardon*, *blanc*, *riesling* и *semillon* соединены центральным коммутатором. Если все узлы подключены с помощью 100-мегабитного кабеля ethernet, весь обмен трафиком в этой сети происходит со скоростью 100 Мбит. Применение различных кабелей и скоростей передач может приводить к проблемам, и, поскольку в настоящее время большинство сетевых интерфейсов Solaris способны работать на скорости 100 Мбит, рекомендуется использовать эту скорость в качестве стандартной. В этой простой сети нет шлюзов, она не связана с другими сетями и с Интернетом и не нуждается в маршрутизаторе.

Если требуется подключение к другой сети, коммутатор может быть подключен к маршрутизатору, как показано на рис. 4.3.

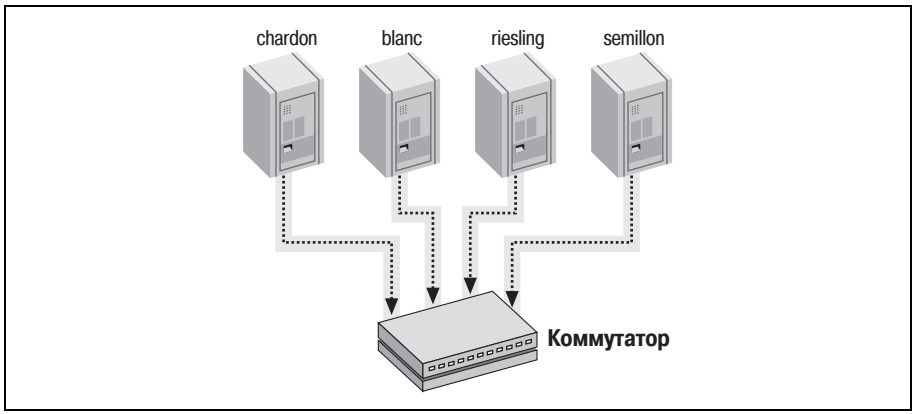

Рис. 4.2. *Простая сеть Solaris класса C, не связанная с сетью Интернет*

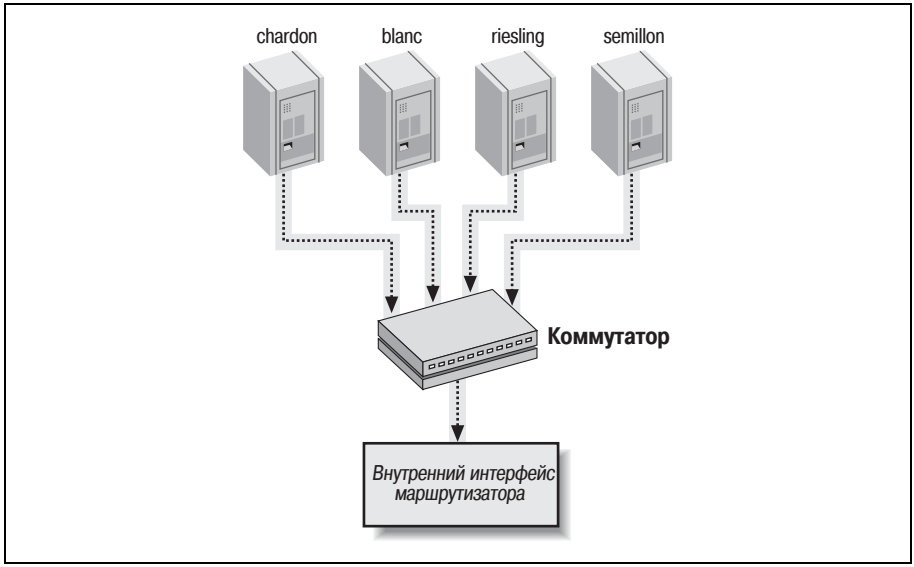

Рис. 4.3. *Простая сеть Solaris класса C, подключенная к другой сети*

В этом случае все пакеты от узлов *chardon*, *blanc*, *riesling* и *semillon* могут передаваться коммутатору и далее «внутреннему» интерфейсу маршрутизатора. Как вариант к одному из узлов, скажем, *chardon*, может быть подключен модем, с помощью которого устанавливается соединение с Интернетом. Узлы *blanc*, *riesling* и *semillon* могут получить прямой доступ к Интернету, не пользуясь доступом по протоколу Telnet на узел *chardon*. Для этого требуется зарегистрировать его в качестве шлюза. Коммутатор гарантирует доставку пакетов на нужный шлюз.

Кроме того, концентраторы и коммутаторы могут объединяться в цепочки с целью подключения отдаленных комнат, этажей и даже зданий к одной сети. Между маршрутизатором и наиболее удаленным клиентом должно быть не более трех транзитных участков, в противном случае количество конфликтов при передаче пакетов вырастет до неприемлемой величины.

Большинство сетевых площадок начинают существование с одной сети класса C. Постепенно число сетей класса C растет, и сети объединяются с помощью маршрутизаторов. Но прежде чем мы рассмотрим установку и настройку маршрутизатора, следует более пристально изучить настройку отдельных сетевых интерфейсов.

Настройка сетевых интерфейсов

Различные программы установки Solaris с радостью займутся настройкой существующих сетевых интерфейсов в процессе установки,

но существуют ситуации, когда требуется добавить новый интерфейс или изменить настройки существующих. А именно:

- Настройка существующего узла для работы в качестве маршрутизатора
- Перенос узла в другую подсеть
- Настройка распределения нагрузки между интерфейсами

Чтобы включить сетевой интерфейс в Solaris, может потребоваться выполнить несколько шагов:

- Установка драйверов устройств
- Перезагрузка системы с целью обновления настроек
- Назначение интерфейсу IP-адреса
- Выбор роли интерфейса (элемент маршрутизатора или элемент многосетевого узла)
- Создание для узла записи, отображающей IP-адрес в имя узла
- Настройка интерфейса на работу с проходящими через него данными

Драйверы устройств, как правило, хранятся в каталоге */kernel/drv* (или в каталоге, определяемым файлом */etc/system*) и перечислены в файле */etc/device_aliases*. Например, стандартному четырехпортовому интерфейсу от Sun соответствует драйвер */kernel/drv/qfe*, его псевдоним присутствует в файле */etc/device_aliases* (*qfe SUNW,qfe*). Следующая команда позволяет выполнить перезагрузку с целью обновления настроек:

```
bash-2.03# touch /reconfigure; init 6
```

Как вариант можно воспользоваться командой монитора OpenBoot PROM:

```
OK boot -r
```

IP-адрес назначается интерфейсом посредством создания hostname-файла, расположенного в каталоге */etc*. Для системы с единственным интерфейсом (скажем, */dev/eri0*), такой как Blade 100, файл носит имя *hostname.eri0*, где *eri0* – имя устройства, а цифра *0* определяет номер интерфейса.

Четырехпортовой Ethernet-карте (устройства */dev/qfe0*, */dev/qfe1*, */dev/qfe2* и */dev/qfe3*) соответствуют четыре файла с различными IP-адресами: *hostname.qfe0*, *hostname.qfe1*, *hostname.qfe2* и *hostname.qfe3*. Адреса могут идти по порядку и принадлежать одной сети (`192.64.18.1`, `192.64.18.2`, `192.64.18.3` и `192.64.18.4`), если узел является многоканальным (multi-homed), либо вразнобой (принадлежать разным сетям), если система является не многоканальным узлом, а маршрутизатором.

Многоканальный узел позволяет производить обмен данными только с локальной сетью (считая маршрутизатор этой сети), в то время как

маршрутизатор отвечает за передачу пакетов между сетями. Чтобы предотвратить маршрутизацию, следует создать на многоканальном узле пустой файл */etc/notrouter*. Кроме того, следует поместить IP-адрес маршрутизатора по умолчанию для локальной сети в файл */etc/defaultrouter*.

Можно создать записи для интерфейсов в файле */etc/hosts* либо записи в службе имен, использование которой определяется файлом */etc/nsswitch.conf*. Например, если необходимо создать отображение IP-адресов из файлов *hostname.qfe0*, *hostname.qfe1*, *hostname.qfe2* и *hostname.qfe3* в имена узлов *www1*, *www2*, *www3* и *www4*, файл */etc/hosts* будет содержать такой фрагмент:

```
bash-2.03# cat /etc/hosts
www1        192.64.18.1
www2        192.64.18.2
www3        192.64.18.3
www4        192.64.18.4
```

На рис. 4.4 показаны логические настройки четырехпортовой Ethernet-карты узла, на котором работают четыре независимых веб-сервера.

Если используется DNS (см. главу 5), в соответствующий файл зоны потребуется добавить такие записи:

```
www1    IN    A    192.64.18.1    ;webserver
www2    IN    A    192.64.18.2    ;webserver
www3    IN    A    192.64.18.3    ;webserver
www4    IN    A    192.64.18.4    ;webserver
```

Прежде чем интерфейс сможет передавать или получать IP-трафик, он должен быть настроен с помощью команды *ifconfig*. Когда интерфейс задействован, можно воспользоваться следующей командой *ifconfig* для перечисления активных интерфейсов:

```
bash-2.03# /usr/sbin/ifconfig -a
lo0: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
```

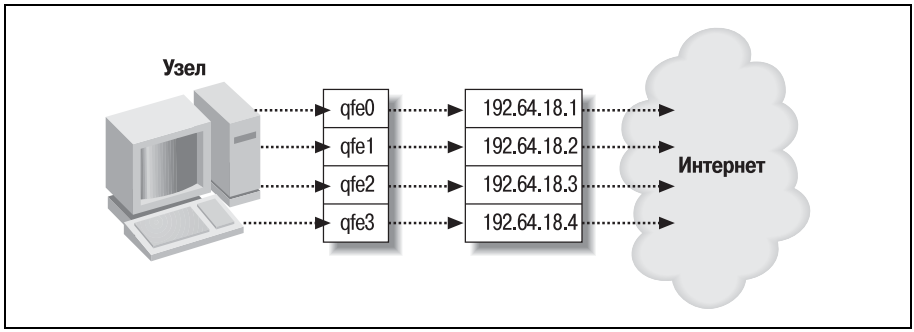

***Рис. 4.4.** Логические настройки четырехпортовой карты Ethernet*

```
eri0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
        inet 10.64.18.3 netmask ffffff00 broadcast 10.64.18.255
lo0: flags=2000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv6> mtu 8252 index 1
        inet6 ::1/128
eri0: flags=2000841<UP,RUNNING,MULTICAST,IPv6> mtu 1500 index 2
        inet6 fe80::203:baff:fe04:a4e8/10
```

Если интерфейс настроен некорректно, при выполнении следующей команды для него будет отображено соответствующее сообщение:

```
bash-2.03# ifconfig eri0
ifconfig: status: SIOCGLIFFLAGS: eri0: no such interface
```

Исходя из предположения, что устройство *eri0* установлено корректно и для него загружены нужные драйверы, мы можем выполнить приведенную ниже команду и произвести настройку на аппаратном уровне.

```
bash-2.03# /usr/sbin/ifconfig eri0 plumb
```

После такой инициализации можно установить рабочие параметры для устройства, такие как IP-адрес (по-прежнему с помощью команды *ifconfig*):

```
bash-2.03# /usr/sbin/ifconfig eri0 10.64.18.3 broadcast 10.64.18.255 netmask
255.255.255.0
```

Чтобы включить интерфейс, следует воспользоваться ключевым словом *up*:

```
bash-2.03# /usr/sbin/ifconfig eri0 up
```

Приведенные команды можно скомбинировать в одну, единовременно выполнив низкоуровневую инициализацию, настройку параметров и включение интерфейса:

```
bash-2.03# /usr/sbin/ifconfig eri0 10.64.18.3 broadcast 10.64.18.255 netmask
255.255.255.0 plumb up
```

В зависимости от конфигурации локальной сети может потребоваться создание соединения вида «точка-точка», а не универсального, как в приведенном примере. Так, если необходимо ограничить доступ к безопасной системе баз данных, можно создать соединение «точка-точка», которое ограничит доступ к базе данных узлом, напрямую подключенным к машине (рис. 4.5). В таком варианте база данных связана с сервером, который, в свою очередь, связан с маршрутизатором; данные не могут передаваться от маршрутизатора к системе БД напрямую, не пройдя через сервер. Поэтому если взломщик собирается проникнуть в систему базы данных, ему придется преодолеть защиту сначала маршрутизатора, а затем и сервера.

Чтобы определить, корректно ли соответствие между IP- и ethernet-адресами с другими узлами локальной сети, воспользуйтесь командой

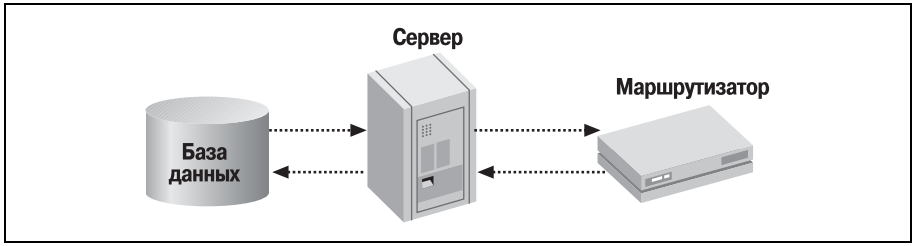

Рис. 4.5. *Повышение уровня безопасности системы баз данных с помощью соединения «точка-точка»*

arp, которая отображает все активные соединения между локальным узлом и другими узлами:

```
bash-2.03# /usr/sbin/arp -a

Net to Media Table: IPv4
Device   IP Address               Mask      Flags   Phys Addr
------ -------------------- --------------- ----- ---------------
eri0   hp                   255.255.255.255       00:50:ba:13:08:18
eri0   austin               255.255.255.255 SP    00:03:ba:04:a4:e8
eri0   224.0.0.0            240.0.0.0       SM    01:00:5e:00:00:00
```

Для локального узла выводится информация по отображению ethernet-адресов в IP-адреса. Флаги имеют следующие значения:

P	Опубликованный адрес
S	Статический адрес
U	Неопределенный адрес
M	Адрес для широковещательных (multicast) сообщений

И наконец, для достижения оптимальной производительности можно вручную установить некоторые из параметров протокола. Воспользуйтесь командой *ndd* для изменения параметров протоколов TCP, UDP, ARP и IP. *ndd* также может применяться для отображения существующих значений параметров, связанных с конкретным протоколом. Чтобы отобразить параметры, связанные с протоколом TCP, воспользуйтесь такой командой:

```
bash-2.03# ndd /dev/tcp \?
?                               (read only)
tcp_close_wait_interval         (read and write)
tcp_conn_req_max_q              (read and write)
tcp_conn_req_max_q0             (read and write)
tcp_conn_req_min                (read and write)
tcp_conn_grace_period           (read and write)
tcp_cwnd_max                    (read and write)
tcp_debug                       (read and write)
tcp_smallest_nonpriv_port       (read and write)
tcp_ip_abort_cinterval          (read and write)
```

```
tcp_ip_abort_linterval        (read and write)
tcp_ip_abort_interval         (read and write)
tcp_ip_notify_cinterval       (read and write)
tcp_ip_notify_interval        (read and write)
tcp_ip_ttl                    (read and write)
tcp_keepalive_interval        (read and write)
tcp_maxpsz_multiplier         (read and write)
tcp_mss_def                   (read and write)
tcp_mss_max                   (read and write)
tcp_mss_min                   (read and write)
tcp_naglim_def                (read and write)
tcp_rexmit_interval_initial   (read and write)
tcp_rexmit_interval_max       (read and write)
tcp_rexmit_interval_min       (read and write)
tcp_wroff_xtra                (read and write)
tcp_deferred_ack_interval     (read and write)
tcp_snd_lowat_fraction        (read and write)
tcp_sth_rcv_hiwat             (read and write)
tcp_sth_rcv_lowat             (read and write)
tcp_dupack_fast_retransmit    (read and write)
tcp_ignore_path_mtu           (read and write)
tcp_rcv_push_wait             (read and write)
tcp_smallest_anon_port        (read and write)
tcp_largest_anon_port         (read and write)
tcp_xmit_hiwat                (read and write)
tcp_xmit_lowat                (read and write)
tcp_recv_hiwat                (read and write)
tcp_recv_hiwat_minmss         (read and write)
tcp_fin_wait_2_flush_interval (read and write)
tcp_co_min                    (read and write)
tcp_max_buf                   (read and write)
tcp_zero_win_probesize        (read and write)
tcp_strong_iss                (read and write)
tcp_rtt_updates               (read and write)
tcp_wscale_always             (read and write)
tcp_tstamp_always             (read and write)
tcp_tstamp_if_wscale          (read and write)
tcp_rexmit_interval_extra     (read and write)
tcp_deferred_acks_max         (read and write)
tcp_slow_start_after_idle     (read and write)
tcp_slow_start_initial        (read and write)
tcp_co_timer_interval         (read and write)
tcp_extra_priv_ports          (read only)
tcp_extra_priv_ports_add      (write only)
tcp_extra_priv_ports_del      (write only)
tcp_status                    (read only)
tcp_bind_hash                 (read only)
tcp_listen_hash               (read only)
tcp_conn_hash                 (read only)
tcp_queue_hash                (read only)
tcp_host_param                (read and write)
tcp_1948_phrase               (write only)
```

Сбор сетевой статистики

Когда все сетевые интерфейсы настроены соответствующим образом, можно применить команду *netstat*, отвечающую за сбор сетевой статистики различного рода в целях проверки работоспособности интерфейсов. Сбор данных происходит для интерфейсов локального узла.

netstat умеет собирать статистику по следующим типам данных:

- Данные, сгруппированные по типу протокола
- Статистика по устройству, сгруппированная по типу адреса (IPv4, IPv6, адреса Unix)
- Данные DHCP
- Данные по интерфейсу, сгруппированные по адресам широковещательной передачи
- Информация из таблицы маршрутизации (включая широковещательные адреса)
- Данные по модулям STREAMS
- Состояние всех доступных IP-интерфейсов
- Состояние всех активных сокетов, маршрутов, физических и логических интерфейсов

В следующих разделах мы рассмотрим операции по сбору каждого из типов данных и обсудим их применение для диагностирования и разрешения проблем, снижающих производительность.

Статистика по протоколам

Статистику по протоколам можно разбить на несколько категорий:

Пакеты RAWIP (IP в чистом виде)	Пакеты TCP
Пакеты IPv4	Пакеты ICMPv4
Пакеты IPv6	Пакеты ICMPv6
Пакеты UDP	Пакеты IGMP

С каждым типом пакетов связан определенный набор средств. Скажем, для пакетов RAWIP существуют счетчики полученных (*rawipInDatagrams*) и отправленных (*rawipOutDatagrams*) после загрузки системы дейтаграмм. В UDP существует аналогичный набор счетчиков, измеряющих число полученных (*udpInDatagrams*) и отправленных (*udpOutDatagrams*) после загрузки системы дейтаграмм. Кроме счетчиков нормальных событий *netstat* отображает также информацию по ошибкам, такую как число UDP-ошибок получения (*udpInErrors*) и отправки (*udpOutErrors*). Эти значения следует проверять регулярно, убеждаясь, что отношение числа ошибок к числу нормальных событий со временем не растет. Например, в приведенном ниже фрагменте имеют место 293 события *tcpActiveOpens* против лишь одного *tcpAttempt-*

Fails. Если со временем отношение *tcpAttemptFails* к *tcpActiveOpens* увеличивается для трафика TCP, может потребоваться изменить определенные параметры TCP с помощью команды *ndd* либо заняться диагностированием сетевых ошибок. Вот типичный набор примеров, позволяющих разобраться с ошибками в работе протоколов для IPv6.

```
bash-2.03$ netstat -s

IPv6    ipv6Forwarding      =     2    ipv6DefaultHopLimit =   255
        ipv6InReceives      =     0    ipv6InHdrErrors     =     0
        ipv6InTooBigErrors  =     0    ipv6InNoRoutes      =     0
        ipv6InAddrErrors    =     0    ipv6InUnknownProtos =     0
        ipv6InTruncatedPkts =     0    ipv6InDiscards      =     0
        ipv6InDelivers      =    25    ipv6OutForwDatagrams=     0
        ipv6OutRequests     =    42    ipv6OutDiscards     =     2
        ipv6OutNoRoutes     =     0    ipv6OutFragOKs      =     0
        ipv6OutFragFails    =     0    ipv6OutFragCreates  =     0
        ipv6ReasmReqds      =     0    ipv6ReasmOKs        =     0
        ipv6ReasmFails      =     0    ipv6InMcastPkts     =     0
        ipv6OutMcastPkts    =    14    ipv6ReasmDuplicates =     0
        ipv6ReasmPartDups   =     0    ipv6ForwProhibits   =     0
        udpInCksumErrs      =     0    udpInOverflows      =     0
        rawipInOverflows    =     0    ipv6InIPv4          =     0
        ipv6OutIPv4         =     0    ipv6OutSwitchIPv4   =     0

ICMPv6  icmp6InMsgs         =     0    icmp6InErrors       =     0
        icmp6InDestUnreachs =     0    icmp6InAdminProhibs =     0
        icmp6InTimeExcds    =     0    icmp6InParmProblems =     0
        icmp6InPktTooBigs   =     0    icmp6InEchos        =     0
        icmp6InEchoReplies  =     0    icmp6InRouterSols   =     0
        icmp6InRouterAds    =     0    icmp6InNeighborSols =     0
        icmp6InNeighborAds  =     0    icmp6InRedirects    =     0
        icmp6InBadRedirects =     0    icmp6InGroupQueries =     0
        icmp6InGroupResps   =     0    icmp6InGroupReds    =     0
        icmp6InOverflows    =     0
        icmp6OutMsgs        =     8    icmp6OutErrors      =     0
        icmp6OutDestUnreachs=     0    icmp6OutAdminProhibs=     0
        icmp6OutTimeExcds   =     0    icmp6OutParmProblems=     0
        icmp6OutPktTooBigs  =     0    icmp6OutEchos       =     0
        icmp6OutEchoReplies =     0    icmp6OutRouterSols  =     3
        icmp6OutRouterAds   =     0    icmp6OutNeighborSols=     1
        icmp6OutNeighborAds =     0    icmp6OutRedirects   =     0
        icmp6OutGroupQueries=     0    icmp6OutGroupResps  =     4
        icmp6OutGroupReds   =     0
```

Статистика по типу адреса

Статистические данные для типов адресов делятся на три категории:

- inet (AF_INET)
- inet6 (AF_INET6)
- unix (AF_UNIX)

Взглянем на пример вывода для сокетов AF_UNIX:

```
bash-2.03$ netstat -f unix

Active UNIX domain sockets
Address     Type       Vnode       Conn     Local Addr      Remote Addr
30000d03738 stream-ord 30000d1eb78 00000000 /tmp/.X11-unix/X0
30000d038e0 stream-ord 00000000    00000000
30000d03a88 stream-ord 30000ce4a30 00000000 /tmp/jd_sockV6
30000d03c30 stream-ord 30000a62d78 00000000 /dev/kkcv
30000d03dd8 stream-ord 30000a62f50 00000000 /dev/ccv
```

Мы видим набор различных активных сокетов с Unix-адресацией, в частности сокет сервера X11 с адресом 30000d03738.

Статистика по широковещательным адресам

Статистика по широковещательным адресам позволяет выявить интерфейсы, которые принимают широковещательные сообщения по адресу 224.0.0.1 (ALL_HOSTS). Это нужно, чтобы пакеты соответствующим образом направлялись с помощью демона обнаружения маршрутизаторов (*in.rdisc*), речь о котором пойдет далее в разделе «Маршрутизация». В следующем примере отображается информация для протоколов IPv4 и IPv6 :

```
bash-2.03$ netstat -g

Group Memberships: IPv4
Interface Group               RefCnt
--------- ------------------- ------
lo0       224.0.0.1                1
eri0      224.0.0.1                1

Group Memberships: IPv6
If    Group                   RefCnt
----- ----------------------- ------
lo0   ff02::1:ff00:1               1
lo0   ff02::1                     1
eri0  ff02::202                    1
eri0  ff02::1:ff04:a4e8            1
eri0  ff02::1                      2
```

Статистика по маршрутизации

Ядро системы ведет таблицу маршрутизации, которая создается демоном маршрутизации *in.routed*. Маршруты, которые были предварительно созданы, всегда доступны для отображения с помощью команды:

```
bash-2.03$ netstat -r

Routing Table: IPv4
Destination          Gateway              Flags Ref   Use    Interface
-------------------- -------------------- ----- ----- ------ ---------
10.64.18.0           austin                U        1      5  eri0
```

```
224.0.0.0          austin              U        1      0  eri0
localhost          localhost           UH      25 215051  lo0
```

Мы видим, что основной Ethernet-интерфейс *eri0* предоставляет два сетевых маршрута для пакетов: в сеть 10.64.18.0 и в сеть широковещательной передачи 224.0.0.0. Кроме того, интерфейс loopback-вызовов (*lo0*) связан с интерфейсом локального узла (*localhost*), который часто используется для отладки и тестирования. Все маршруты относятся к стандарту IPv4, однако информация по маршрутизации доступна и для версии IPv6:

```
Routing Table: IPv6
Destination/Mask           Gateway                   Flags Ref Use    If
-------------------------- ------------------------- ----- --- ------ -----
fe80::/10                  fe80::203:baff:fe04:a4e8    U       1      0 eri0
ff00::/8                   fe80::203:baff:fe04:a4e8    U       1      0 eri0
default                    fe80::203:baff:fe04:a4e8    U       1      0 eri0
localhost                  localhost                   UH      5     28 lo0
```

Статистика по модулям STREAMS

STREAMS представляет собой пакет System V, обеспечивающий доступ к системным вызовам, стандартным библиотекам и ядру с целью разработки сетевых приложений. Любое приложение, использующее STREAMS, обладает определенным набором свойств, по которым можно собрать статистику, поскольку операции ввода–вывода отличаются от аналогичных операций в других сетевых API (скажем, в интерфейсе прикладного программирования для BSD-сокетов). *netstat* позволяет получить эту статистику, включая и информацию по очередям, реализующим характерные для потока операции чтения–записи:

```
bash-2.03$ netstat -m
streams allocation:
                                          cumulative  allocation
                     current   maximum       total    failures
streams                326       340        7634           0
queues                 938       962       18662           0
mblk                  1144      1651        7773           0
dblk                  1140      1729     2349590           0
linkblk                 11       169          18           0
strevent                 9       169      121739           0
syncq                   25        50         101           0
qband                    0         0           0           0

1646 Kbytes allocated for streams data
```

Более подробная информация доступна в системной справке по *streamio*.

Статистика по IP-интерфейсам

netstat позволяет получать статистику, собранную на уровне протокола IP. В частности доступно число полученных и отправленных паке-

тов, число ошибок при отправке и получении, а также число конфликтов пакетов. Как и ранее, информация для версий IPv4 и IPv6 отображается в двух разделах:

```
bash-2.03$ netstat -i
Name  Mtu  Net/Dest      Address        Ipkts  Ierrs Opkts  Oerrs Collis Queue
lo0   8232 loopback      localhost      227695 0     227695 0     0      0
eri0  1500 austin        austin         2573   0     2130   0     0      0

Name  Mtu  Net/Dest      Address          Ipkts  Ierrs Opkts  Oerrs Collis
lo0   8252 localhost     localhost        227705 0     227705 0     0
eri0  1500 fe80::203:baff:fe04:a4e8/10 fe80::203:baff:fe04:a4e8    2573   0
2130   0     0
```

Комбинированная статистика по сокетам, маршрутам и интерфейсам

Большинство администраторов предпочитают получать с помощью команды *netstat* информацию, скомбинированную в один отчет. Пример 4.1 содержит отчет с комбинированной статистикой по маршрутам, сокетам и интерфейсам.

Пример 4.1. Вывод команды netstat -a

```
bash-2.03$ netstat --a
UDP: IPv4
   Local Address         Remote Address     State
-------------------- -------------------- -------
      *.route                             Idle
      *.*                                 Unbound
      *.*                                 Unbound
      *.sunrpc                            Idle
      *.*                                 Unbound
      *.32771                             Idle
      *.sunrpc                            Idle
      *.*                                 Unbound
      *.32775                             Idle
      *.32779                             Idle
      *.32780                             Idle
Routing
*.*                                  Unbound
      *.32821                             Idle
      *.32822                             Idle
      *.32823                             Idle
      *.name                              Idle
      *.biff                              Idle
      *.talk                              Idle
      *.time                              Idle
      *.echo                              Idle

UDP: IPv6
   Local Address                      Remote Address                  State
```

```
If
---------------------------- ---------------------------- ------- ------
      *.*                                                         Unbound
      *.sunrpc                                                    Idle
      *.*                                                         Unbound
      *.32771                                                     Idle
      *.32779                                                     Idle
      *.*                                                         Unbound
      *.32821                                                     Idle
      *.time                                                      Idle

TCP: IPv4
   Local Address        Remote Address    Swind Send-Q Rwind Recv-Q  State
-------------------- -------------------- ----- ------ ----- ------ -------
      *.*                  *.*                0      0 24576      0 IDLE
      *.sunrpc             *.*                0      0 24576      0 LISTEN
      *.*                  *.*                0      0 24576      0 IDLE
      *.sunrpc             *.*                0      0 24576      0 LISTEN
      *.*                  *.*                0      0 24576      0 IDLE
      *.32775              *.*                0      0 24576      0 LISTEN
      *.32776              *.*                0      0 24576      0 LISTEN
      *.32782              *.*                0      0 24576      0 LISTEN
      *.32783              *.*                0      0 24576      0 LISTEN

TCP: IPv6
Local Address   Remote Address  Swind  Send-Q  Rwind  Recv-Q  State  If
*.*             *.*             0              24576  0       IDLE
*.sunrpc        *.*             0      0       24576  0       LISTEN
*.*             *.*             0      0       24576  0       IDLE
*.32775         *.*             0      0       24576  0       LISTEN
localhost.32780 localhost.32775 32768  0       32768  0       CLOSE_WAIT
*.32782         *.*             0      0       24576  0       LISTEN
*.32791         *.*             0      0       24576  0       LISTEN
*.ftp           *.*             0      0       24576  0       LISTEN
*.telnet        *.*             0      0       24576  0       LISTEN

Active UNIX domain sockets
Address  Type          Vnode     Conn  Local Addr      Remote Addr
30000d03738 stream-ord 30000d1eb78 00000000 /tmp/.X11-unix/X0
30000d038e0 stream-ord 00000000 00000000
30000d03a88 stream-ord 30000ce4a30 00000000 /tmp/jd_sockV6
30000d03c30 stream-ord 30000a62d78 00000000 /dev/kkcv
30000d03dd8 stream-ord 30000a62f50 00000000 /dev/ccv
```

Некоторые из типов сообщений TCP, содержащихся в отчете, могут быть читателю незнакомы; их интерпретация приведена в табл. 4.1.

Таблица 4.1. Константы TCP, используемые командой netstat

Сообщение	Описание
BOUND	Сокет связан
CLOSED	Сокет закрыт
CLOSING	Сокет закрывается

Таблица 4.1 (продолжение)

Сообщение	Описание
CLOSE_WAIT	Сокет в ожидании закрытия
ESTABLISHED	Сокет успешно подключен
FIN_WAIT_1	Сокет закрывается (локальный)
FIN_WAIT_2	Сокет закрывается (удаленный)
IDLE	Сокет бездействует
LAST_ACK	Сокет закроется, получив последнее подтверждение
LISTEN	Сокет активен, ведется прием сообщений
SYN_RECEIVED	Сокет синхронизируется
SYN_SENT	Сокет создает соединение
TIME_WAIT	Сокет в ожидании закрытия

Маршрутизация

Представьте себя курьером, путь которого всегда начинается у локального склада. Вам выдают перечень адресов, связанных с набором пакетов. Задача – доставить пакеты за минимальное время, учитывая следующие ограничения:

- Число дорог, пройденных для доставки конкретного пакета, должно быть минимальным.
- Следует избегать тупиков и несчастных случаев.
- Определить, какой дорогой следовать, можно только с помощью каталога улиц и сопоставления найденных имен с теми, что в списке.

Если вам кажется, что для курьера задачка тривиальна, прикиньте, насколько сложной может стать его работа, если учесть следующие факторы:

- Число дорог и их комбинаций растет экспоненциально с каждым годом. Курьера могут попросить проследовать дорогами, о которых он раньше даже не слышал!
- Заранее невозможно предугадать, где произойдет несчастный случай или окажется тупик.
- Каталог улиц, имеющийся в распоряжении, давно устарел, потому что число дорог растет экспоненциально с каждым годом.

Таково описание сложностей, с которыми было связано рождение Интернета и объединение узлов и сетей в глобальных масштабах. Чтобы передать пакет от узла A узлу B, необходимо определить физический путь, которым проследует пакет.

Не существует централизованной службы поиска, позволяющей определить путь маршрута пакета для всех возможных пар узлов Интерне-

та (то есть путь между отправителем и получателем). Это означает, что маршруты должны создаваться динамически. (Единственным исключением из этого правила являются определенные случаи, когда стабильно предсказуемые маршруты определяются статически.)

При передаче данных через Интернет или между подсетями необходимо наличие промежуточных узлов, ответственных за такую передачу. Такие узлы называются маршрутизаторами и отвечают за маршрутизацию пакетов между узлами, которые могут разделять отдельные подсети или целые континенты. Чтобы оценить, через какое число маршрутизаторов может пройти пакет, воспользуемся командой *traceroute* и взглянем на транзитные участки, по которым проходят пакеты при подключении узла в Сиднее (Австралия) к веб-серверу компании Sun Microsystems:

```
bash-2.03$ traceroute wwwwseast.usec.sun.com/
Tracing route to wwwseast.usec.sun.com [192.9.49.30]
over a maximum of 30 hops:
  1   184 ms   142 ms   138 ms  202.10.4.131
  2   147 ms   144 ms   138 ms  202.10.4.129
  3   150 ms   142 ms   144 ms  202.10.1.73
  4   150 ms   144 ms   141 ms  atm11-0-0-11.ia4.optus.net.au [202.139.32.17]
  5   148 ms   143 ms   139 ms  202.139.1.197
  6   490 ms   489 ms   474 ms  hssi9-0-0.sf1.optus.net.au [192.65.89.246]
  7   526 ms   480 ms   485 ms  g-sfd-br-02-f12-0.gn.cwix.net [207.124.109.57]
  8   494 ms   482 ms   485 ms  core7-hssi6-0-0.SanFrancisco.cw.net [204.70.10.9]
  9   483 ms   489 ms   484 ms  corerouter2.SanFrancisco.cw.net [204.70.9.132]
 10   557 ms   552 ms   561 ms  xcore3.Boston.cw.net [204.70.150.81]
 11   566 ms   572 ms   554 ms  sun-micro-system.Boston.cw.net [204.70.179.102]
 12   577 ms   574 ms   558 ms  wwwwseast.usec.sun.com [192.9.49.30]
Trace complete.
```

Можно видеть, что для передачи пакета получателю потребовалось двенадцать узлов. Кроме того, отображаемые интервалы времени, затрачиваемые на получение ответов, довольно велики – до полусекунды. Возможно, что соединение не будет установлено по причине окончания интервала ожидания. Такая диагностика может быть полезна при попытке определить, какой из промежуточных узлов (или какая из промежуточных сетей) является причиной того, что ваше соединение с узлом на другом конце света внезапно оборвалось.

В этом разделе мы рассмотрим способы, которыми в Solaris решаются классические проблемы маршрутизации.

Статическая маршрутизация, как правило, идет рука об руку с прямым физическим соединением пары узлов, то есть со случаем, когда реализация динамической маршрутизации является неэффективной либо связана с понижением уровня безопасности. Например, если в локальной сети есть три подсети, между которыми необходимо организовать обмен данными, для каждого маршрутизатора можно создать статические маршруты к двум другим маршрутизаторам. Число

конкретных маршрутов, необходимых для беспрепятственной циркуляции данных между сетями, прямо пропорционально квадрату числа маршрутизаторов в сети. После каждой модификации топологии сети все эти маршруты придется редактировать вручную. Может показаться, что речь идет о чрезмерных объемах трудной работы, но могут возникать ситуации, когда эта работа оправдана: представьте безопасный сервер баз данных, к которому можно получить доступ, только воспользовавшись конкретным маршрутом, информация о котором широко не распространяется. В этом случае именно статическая маршрутизация (а не динамическое обнаружение маршрута) является подходящим решением. Такую конфигурацию можно реализовать, создав соединение «точка-точка» на дополнительном интерфейсе с помощью команды *ifconfig* (как это сделать, рассказано в разделе, посвященном настройке сетевых интерфейсов).

Альтернативой статической маршрутизации является маршрутизация динамическая, работа которой связана с парой демонов: собственно демона маршрутизации (*in.routed*) и демона обнаружения маршрутизаторов (*in.rdisc*). Демон *in.routed* реализует протокол информации маршрутизации (Routing Information Protocol) и отвечает за обновление и сопровождение записей в таблицах маршрутизации ядра. Для выполнения операций маршрутизации демон использует протокол UDP (порт 520) и работает на всех сетевых интерфейсах, которые были инициализированы на аппаратном уровне (*plumb*) и являются активными (*up*).

В случае отсутствия файла */etc/notrouter* и при наличии двух или более рабочих интерфейсов узел начинает действовать в качестве маршрутизатора. Таким образом, данные могут поступать через один интерфейс и тут же передаваться дальше через другой. В локальной сети интерфейс, связанный со всеми локальными узлами, обычно называется *внутренним* (*internal*) интерфейсом, тогда как интерфейс, доступный другой подсети или интернет-провайдеру, называется *внешним* (*external*). Применив фильтрацию сетевых пакетов, можно создать набор правил, определяющих, какого рода TCP- или UDP-пакеты разрешено передавать через конкретные интерфейсы и порты. Такого рода фильтры, несомненно, важны для защиты локальных сетей и сокрытия служб, предназначенных только локальным узлам, от внешнего доступа.

Демон обнаружения маршрутизаторов, *in.rdisc*, использует протокол управляющих сообщений Интернета (Internet Control Message Protocol, ICMP). Что касается обнаружения маршрутов, демон *in.rdisc*, работающий на обычной машине, принимает широковещательные сообщения по адресу 224.0.0.1 (ALL_HOSTS). Сообщениям назначаются приоритеты, после чего маршрутизатор по умолчанию выбирается на основе его близости к узлу. *in.rdisc* на маршрутизаторе оповещает о своем существовании с помощью групповых сообщений на адрес 224.0.0.1 и принимает запросы по адресу 224.0.0.2 (ALL_ROUTERS). Узлы могут напрямую получить адрес маршрутизатора по адресу 224.0.0.2.

5

Службы имен

Когда сеть начинает расти и ответственность за решение различных задач делится между узлами, особую важность приобретает установка, по меньшей мере, одной системы каталогов, которая позволит находить узлы сети. Для доменов Интернета могут потребоваться дополнительные службы имен. В этой главе рассказывается о важности распределенных служб имен, реализующих поиск узлов, пользователей и различных видов сетевых ресурсов, и читатели знакомятся с процессом настройки службы доменных имен (DNS) системы Solaris. DNS позволяет преобразовывать IP-адреса в имена, удобные для использования людьми, и обратно. Сетевая информационная служба (Network Information System, NIS/NIS+), разработанная Sun, позволяет сделать шаг вперед от традиционной системы DNS в плане поддержки иерархических пространств имен. Система NIS/NIS+ является универсальным решением по управлению сетевыми ресурсами и предоставляет возможность проводить как авторизацию доступа к ресурсам, так и идентификацию, то есть проверку подлинности, во взаимодействиях клиентов и серверов. И наконец, речь пойдет о более современном облегченном протоколе доступа к каталогам (Lightweight Directory Access Protocol, LDAP) и его реализации на платформе Solaris.

Домены и службы имен

Прежде чем мы перейдем к детальному рассмотрению реализаций различных служб имен, я хочу сконцентрироваться на значении термина *домен* (*domain*). Грубо говоря, можно провести аналогию между доменом и государством, у которого есть четкие границы, механизмы безопасности, предотвращающие нарушение этих границ, а также набор

общих предписаний (зафиксированных законами) по поводу того, что является дружелюбным поведением. Между гражданами государства и доменом существуют экономические отношения: услуга предоставляется, если ее потребление можно отследить и учесть.

Домен Solaris любого типа во многом схож с государством: группы узлов находятся под защитой центрального органа власти, который обеспечивает безопасность сети, пользователи имеют доступ к службам ряда узлов локального домена, а средства учета и проверки способны запоминать, кто и как использовал эти службы. Неудивительно, что компания Sun называет службы имен Solaris федеративной службой имен!

Чтобы домен Solaris любого типа успешно работал, необходимо наличие хранилища информации (централизованного или распределенного), которое содержало бы достоверные данные по всем пользователям и узлам, находящимся под его контролем. В автономных Solaris-системах обычно используются файлы для отслеживания информации по отдельным пользователям и группам, которым разрешено работать с системой. В небольшой локальной сети служба доменных имен (Domain Name Service, DNS) используется, как правило, для преобразования IP-адресов в имена узлов, облегчая задачу обращения к узлам при работе во всех IP-ориентированных приложениях.

Однако DNS не реализует механизмы управления чем-либо еще, кроме имен узлов; все прочие системные ресурсы – пользователи, группы, принтеры – должны администрироваться локально, в отдельности для каждого из узлов домена, либо с помощью доменной службы иного рода. Одной из наиболее популярных (и простых) доменных служб является сетевая информационная служба (Network Information Service, NIS). Используя единственный выделенный сервер, NIS позволяет управлять следующими ресурсами всей локальной сети:

- Бездисковыми клиентами
- Адресами Ethernet
- Группами
- Узлами
- Протоколами IP
- Почтовыми псевдонимами
- Сетевыми масками и подсетями
- Паролями
- Службами RPC
- Службами TCP и UDP
- Пользователями

NIS, как правило, хорошо интегрируется с другими системами Unix и подобными им (например, Linux).

Для управления более сложными иерархическими доменными структурами в Solaris существует NIS+, серьезно доработанная версия NIS. NIS+ также позволяет более тонко разграничивать доступ в защищенных рабочих средах, в которых может применяться шифрование передаваемых в процессе проверки подлинности данных. Кроме того, в NIS+ задействована реляционная модель хранения данных, в то время

как NIS основывается на простых картах. К счастью, серверы NIS+ могут работать в режиме совместимости с NIS, что упрощает переход для существующих NIS-клиентов.

Альтернативой NIS/NIS+ является применение открытого стандарта, такого как облегченный протокол доступа к каталогам (Lightweight Directory Access Protocol, LDAP), для организации доступа к службе X.500. В комплект поставки Solaris 8 входит сервер каталогов iPlanet, система более открытая, чем традиционные доменные службы. Сервер каталогов отвечает за информацию о пользователях локального домена и сопровождение их учетных записей, которые могут модифицироваться пользователями после идентификации. Кроме того, пользователи, желающие получить доступ к доменным службам извне корпоративной сети, могут сделать это, пройдя через процесс проверки подлинности. Одной из приятных возможностей сервера каталогов является его способность делегировать ответственность за сопровождение пользовательских учетных записей подходящим лицам или организациям. Например, сотруднику поддержки местного отдела рекламы могут быть выданы полномочия для управления и обновления пользовательских записей этого отдела, но он не будет иметь доступа к данным других отделов. Такая функциональность возможна благодаря интеграции сервера каталогов с маршрутизатором доступа к каталогам (Directory Access Router) iPlanet, LDAP-системой высокой готовности для критически важных сред и приложений. Сервер каталогов идеально интегрируется и с существующими NIS-системами.

Коммутатор служб имен

Поскольку Solaris поддерживает так много разнообразных служб имен (многие из которых могут параллельно сосуществовать на одном узле или в пределах одной локальной сети), сетевым демонам и пользовательским приложениям требуется способ определять, какая из служб соответствует тем или иным задачам. Различные службы имен могут выполнять различные задачи. Скажем, служба DNS может быть выбрана для поиска имен узлов, а разрешение имен протоколов возложено на NIS+. Помимо этого следует установить порядок опроса служб на случай сбоя одной из них. Например, основной, предпочтительной службой имен может быть DNS, но в случае когда у локального узла нет возможности связаться с DNS-сервером из-за сбоя сети, использование NIS+, NIS либо записей узлов из файла на локальном узле может быть вполне приемлемым временным решением. Как только служба продолжит работу, клиенты смогут возобновить обращения к этой службе.

Solaris позволяет производить такое переключение с помощью коммутатора служб имен (*/etc/nsswitch.conf*). Коммутатор служб имен предоставляет первичные, вторичные и третичные указатели на службы имен, которые могут использоваться для решения конкретных задач.

Чаще всего в файле */etc/nsswitch.conf* указываются службы разрешения для:

Службы автоматического монтирования	Паролей
Загрузочных параметров бездисковых клиентов	Протоколов
Адреса ethernet	Служб RPC
Групп	Служб
Имен узлов	Масок подсетей
Сетей	

В составе Solaris поставляются три версии файла */etc/nsswitch.conf*, которыми можно воспользоваться для настройки базового поиска в NIS+, NIS либо файлах (версии называются */etc/nsswitch.nisplus*, */etc/nsswitch.nis* и */etc/nsswitch.files*, соответственно). В небольших сетях в качестве службы имен чаще всего применяется DNS, и для этого варианта следует создать собственную версию файла */etc/nsswitch.conf*. В следующих разделах мы рассмотрим настройку коммутатора служб имен.

Коммутатор служб имен: работа с файлами

Приведенная здесь конфигурация */etc/nsswitch.conf* предполагает отсутствие централизованной службы имен и хранение данных по узлам и сети в обычных файлах. Применение неструктурированных файлов обусловлено историей – в прошлом все известные отображения имен узлов в IP-адреса хранились в единственном файле (*HOSTS.TXT*). А загрузочный файл, содержащий информацию о корневых серверах имен DNS, до сих пор распространяется в неструктурированном формате:

```
>>> Last update of whois database: Mon, 9 Oct 2000 09:43:11 EDT <<<

The Registry database contains ONLY .COM, .NET, .ORG, .EDU domains and
Registrars.

ftp://ftp.rs.internic.net/domain/named.root
;       This file holds the information on root name servers needed to
;       initialize cache of Internet domain name servers
;       (e.g. reference this file in the "cache  .  <file>"
;       configuration file of BIND domain name servers).
;
;       This file is made available by InterNIC registration services
;       under anonymous FTP as
;           file                /domain/named.root
;           on server           FTP.RS.INTERNIC.NET
;       -OR- under Gopher at    RS.INTERNIC.NET
;           under menu          InterNIC Registration Services (NSI)
;              submenu          InterNIC Registration Archives
```

```
;            file                named.root
;
;       last update:    Aug 22, 1997
;       related version of root zone:   1997082200
;
;
; formerly NS.INTERNIC.NET
;
.                        3600000  IN  NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.      3600000      A     198.41.0.4
;
; formerly NS1.ISI.EDU
;
.                        3600000      NS    B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.      3600000      A     128.9.0.107
;
; formerly C.PSI.NET
;
.                        3600000      NS    C.ROOT-SERVERS.NET.
C.ROOT-SERVERS.NET.      3600000      A     192.33.4.12
;
; formerly TERP.UMD.EDU
;
.                        3600000      NS    D.ROOT-SERVERS.NET.
D.ROOT-SERVERS.NET.      3600000      A     128.8.10.90
;
; formerly NS.NASA.GOV
;
.                        3600000      NS    E.ROOT-SERVERS.NET.
E.ROOT-SERVERS.NET.      3600000      A     192.203.230.10
;
; formerly NS.ISC.ORG
;
.                        3600000      NS    F.ROOT-SERVERS.NET.
F.ROOT-SERVERS.NET.      3600000      A     192.5.5.241
;
; formerly NS.NIC.DDN.MIL
;
.                        3600000      NS    G.ROOT-SERVERS.NET.
G.ROOT-SERVERS.NET.      3600000      A     192.112.36.4
;
; formerly AOS.ARL.ARMY.MIL
;
.                        3600000      NS    H.ROOT-SERVERS.NET.
H.ROOT-SERVERS.NET.      3600000      A     128.63.2.53
;
; formerly NIC.NORDU.NET
;
.                        3600000      NS    I.ROOT-SERVERS.NET.
I.ROOT-SERVERS.NET.      3600000      A     192.36.148.17
;
```

```
; temporarily housed at NSI (InterNIC)
;
.                        3600000      NS    J.ROOT-SERVERS.NET.
J.ROOT-SERVERS.NET.      3600000      A     198.41.0.10
;
; housed in LINX, operated by RIPE NCC
;
.                        3600000      NS    K.ROOT-SERVERS.NET.
K.ROOT-SERVERS.NET.      3600000      A     193.0.14.129
;
; temporarily housed at ISI (IANA)
;
.                        3600000      NS    L.ROOT-SERVERS.NET.
L.ROOT-SERVERS.NET.      3600000      A     198.32.64.12
;
; housed in Japan, operated by WIDE
;
.                        3600000      NS    M.ROOT-SERVERS.NET.
M.ROOT-SERVERS.NET.      3600000      A     202.12.27.33
; End of File
```

Формат файла */etc/nsswitch.conf* еще более простой. В левой колонке указан тип поиска, в правой – метод поиска. Для поиска по файлам следует указать метод *files* для всех типов поиска:

```
passwd:         files
group:          files
hosts:          files
networks:       files
protocols:      files
rpc:            files
ethers:         files
netmasks:       files
bootparams:     files
publickey:      files
netgroup:       files
automount:      files
aliases:        files
services:       files
sendmailvars:   files
```

Каждый из перечисленных типов поиска соответствует файлу из каталога */etc*; так что псевдонимы адресов электронной почты хранятся в файле */etc/aliases*, а адреса Ethernet – в файле */etc/ethers*.

Коммутатор служб имен: работа с DNS

Файл базовой настройки для работы с DNS без NIS и NIS+ весьма схож с вариантом работы только с файлами. Единственное исключение сделано для алгоритма поиска узлов: основным методом поиска здесь является DNS (dns). Если поиск узла завершается с отрицательным ре-

зультатом (узел не найден, NOTFOUND), происходит обращение к локальному файлу узлов:[1]

```
passwd:         files
group:          files
hosts:          dns [NOTFOUND=return] files
networks:       files
protocols:      files
rpc:            files
ethers:         files
netmasks:       files
bootparams:     files
publickey:      files
netgroup:       files
automount:      files
aliases:        files
services:       files
sendmailvars:   files
```

В правой колонке файла настройки можно указать более двух служб: например, можно сделать DNS первичной службой для поиска узлов, NIS+ – вторичной, а обращение к файлам – последним вариантом.

Коммутатор служб имен: работа с NIS

Файл */etc/nsswitch.conf* для NIS-варианта более сложный, чем для варианта только файлов или DNS. Доступен целый ряд возможностей в зависимости от того, какая служба опрашивается первой – NIS или файлы. В следующем примере обращение к файлу */etc/passwd* происходит раньше, чем обращение к соответствующей карте NIS, но для разрешения имен узлов NIS используется прежде файлов:

```
passwd:       files nis
group:        files nis
hosts:        nis [NOTFOUND=return] files
networks:     nis [NOTFOUND=return] files
protocols:    nis [NOTFOUND=return] files
rpc:          nis [NOTFOUND=return] files
ethers:       nis [NOTFOUND=return] files
netmasks:     nis [NOTFOUND=return] files
bootparams:   nis [NOTFOUND=return] files
publickey:    nis [NOTFOUND=return] files
netgroup:     nis
```

[1] Здесь автор ошибается: если узел не найден, то обращение к локальному файлу узлов происходить не будет (`NOTFOUND=return`). Обращение к файлу узлов будет производиться, только если сервер DNS недоступен. Данный комментарий справедлив и для остальных примеров nsswitch.conf: параметр `NOTFOUND=return` определяет окончание поиска информации при доступности сервера данного ресурса. Для поиска с помощью dns, а затем files, необходимо испольовать следующую строку: `hosts: dns files`. – *Примеч. науч. ред.*

```
automount:    files nis
aliases:      files nis
```

Единственным исключением из правила первичной/вторичной службы является служба *netgroup*, к которой можно обращаться только через NIS.

Коммутатор сетевых служб: работа с NIS+

Самой сложной из рассмотренных конфигураций является вариант файла */etc/nsswitch.conf* для NIS+. Здесь также существует возможность выбора порядка опроса служб. В нашем случае обращение к файлу */etc/group* происходит раньше, чем обращение к соответствующей таблице NIS+, но для поиска служб RPC применяется сначала NIS+, а затем файлы:

```
passwd:       files nisplus
group:        files nisplus
hosts:        nisplus [NOTFOUND=return] files
services:     nisplus [NOTFOUND=return] files
networks:     nisplus [NOTFOUND=return] files
protocols:    nisplus [NOTFOUND=return] files
rpc:          nisplus [NOTFOUND=return] files
ethers:       nisplus [NOTFOUND=return] files
netmasks:     nisplus [NOTFOUND=return] files
bootparams:   nisplus [NOTFOUND=return] files
publickey:    nisplus
netgroup:     nisplus
automount:    files nisplus
aliases:      files nisplus
sendmailvars: files nisplus
```

Здесь только два исключения из правила первичной/вторичной службы: службы *netgroup* и *publickey*, к которым можно обращаться только через NIS+.

Служба доменных имен (DNS)

Большинство подключенных к Интернету систем пользуются службой доменных имен (Domain Name Service, DNS) для преобразования имен узлов в IP-адреса. Изначальная спецификация DNS содержится в документах RFC 882 и 883. В контексте Solaris имя узла – это значение, возвращаемое по команде *hostname*: одно уникальное имя узла, отличающее этот узел от всех других узлов локальной сети. Скажем, узел по имени *puccini* может иметь полное доменное имя (fully qualified domain name, FQDN) *puccini.musica.it* и IP-адрес 10.64.18.1. Имя домена, к которому принадлежит *puccini*, – *musica.it*. Таким образом:

Имя узла: *puccini*

Домен: *musica.it*

FQDN: *puccini.musica.it*

IP-адрес: 10.64.18.1

Суффикс *.it* доменного имени указывает на *домен высшего уровня* Италии. Разумеется, доменов высшего уровня существует много, и большинство из них связаны с названиями стран. Так, Великобритании присвоено доменное имя *.uk*[1], а Франции – *.fr*. Изначально – видимо, чтобы еще более осложнить положение, – для доменов высшего уровня использовались суффиксы функционального деления: университетам был отведен домен *.edu*, а сетям – домен *.net*. Например, колледжу Амхерст соответствовали домен высшего уровня *.edu* и домен второго уровня *.amherst*, и в результате получалось *amherst.edu*. Но подобные вариации в доменах высшего уровня по большей части используются только организациями, расположенными в США, и во всем остальном мире являются доменами второго уровня. Например, университету Тасмании принадлежит домен *utas.edu.au*, где *.au* – австралийский домен высшего уровня, *.edu* – домен второго уровня, указывающий на принадлежность к образовательным учреждениям, а *.utas* – имя домена университета. Соглашения по доменным именам второго уровня варьируются в зависимости от страны; так, в Великобритании компании попадают в домен второго уровня *.co*, в то время как научным организациям достался домен *.ac*. Книжному магазину Heffers принадлежит домен *heffers.co.uk*, а Кембриджскому университету домен *cam.ac.uk*. Среди прочих доменов высшего уровня США можно упомянуть универсальный *.com* для коммерческих и *.org* для некоммерческих организаций.

Доменом второго уровня для *puccini.musica.it* является *.musica*. DNS позволяет использовать более двух уровней, и общее их количество зависит только от структуры конкретной организации. В домене *musica.it* могут существовать поддомены *concerto.musica.it* и *opera.musica.it*. Естественно, *puccini* в таком случае будет принадлежать *opera.musica.it* (*puccini.opera.musica.it*), а узел с именем *vivaldi* может располагаться в домене *concerto.musica.it*. Связь домена высшего уровня, второго уровня и местной организацией домена показана на рис. 5.1.

Чтение имен доменов от высших уровней вниз приобретает важное значение при преобразовании имен узлов и доменов в IP-адреса. При работе с IP это необходимое преобразование; оно происходит каждый раз, когда полное доменное имя (FQDN) передается IP-приложению, например Telnet: выполняется поиск IP-адреса, в который и преобразуется доменное имя. При необходимости может также выполняться и обратная операция – получение доменного имени из IP-адреса.

---

[1] Стоит заметить, что в соответствии со стандартом ISO 3166 Великобритании присвоено доменное имя *.gb*, однако исторически сложилось так, что большинство сайтов Великобритании и Северной Ирландии (то есть Соединенного Королевства, United Kingdom) зарегистрировано в домене *.uk*. – *Примеч. ред.*

Рис. 5.1. *Связь доменов различных уровней и имен узлов*

Отображение полных доменных имен и IP-адресов работает по формуле «один к одному»; то есть для каждого полного имени должен существовать уникальный IP-адрес, а каждый IP-адрес должен указывать на единственное полное доменное имя. Существуют исключения из этого правила; так, в частных сетях может использоваться адресный диапазон `10.0.0.0`, поскольку в открытых сетях адресами из этого диапазона пользоваться запрещено. В мире может существовать сколько угодно частных сетей с этим специальным адресным диапазоном. Формула «один к одному» действует для всех открытых сетей, подключенных к Интернету.

Отличительной чертой службы DNS в сравнении с другими службами имен является ее распределенная (а не централизованная) архитектура. Это означает, что конкретные организации, подключенные к Интернету, в конечном итоге сами отвечают за сопровождение записей своих узлов и сетей. Организации, таким образом, получают максимальную свободу в вопросах структурирования собственных сетей, но от них требуется глубокое понимание принципов DNS при создании сети, связанной с Интернетом.

Действие DNS основывается на распространении информации об узлах от так называемых компетентных, или авторитативных (authoritative), источников к локальным серверам доменных имен. Каждая сеть, подключенная к Интернету, должна иметь, по меньшей мере, два сервера DNS, компетентных для всех узлов домена. Как правило, эти два сервера указываются при регистрации доменного имени в специальной организации-регистраторе. Рекомендуется физически разграничивать эти серверы, чтобы повысить доступность домена в случае сбоя одного из них. Например, серверы в двух подсетях, расположенных в разных зданиях, будут вполне приемлемым вариантом. В коммерческих системах может быть более разумно указать один из серверов своего интернет-провайдера.

В большинстве доменов в качестве резервного варианта (на случай, когда службы DNS недоступны) применяется файл */etc/hosts* на всех узлах. Этот файл обычно содержит записи для основных серверов, таких как сервер доменной службы, маршрутизатора или шлюза. Файл узлов может выглядеть так:

```
127.0.0.1             localhost
10.64.18.1 gw         gw.musica.it    loghost
10.64.18.2 vivaldi    vivaldi.musica.it
10.64.18.3 puccini    puccini.musica.it
```

Здесь приведена информация для трех узлов:

- Узел localhost (*gw*), к которому можно обращаться по стандартному loopback-адресу (`127.0.0.1`) либо по IP-адресу (`10.64.18.1`). Узел является шлюзом сети.
- На узле *vivaldi* (`10.64.18.2`) работает BIND (демон доменных имен), который и обеспечивает работу служб DNS. BIND используется на многих платформах, включая Unix и Unix-подобные системы, такие как Linux.
- На узле *puccini* (`10.64.18.3`) работает сервер NFS и автоматический монтировщик, что позволяет разделять дисковые системы и организовывать централизованное хранение домашних каталогов, соответственно.

Большинство узлов должны определять отображения адресов в доменные имена в файле */etc/hosts*, равно как и записи для серверов DNS и NFS по необходимости. На рис. 5.2 показана описанная конфигурация.

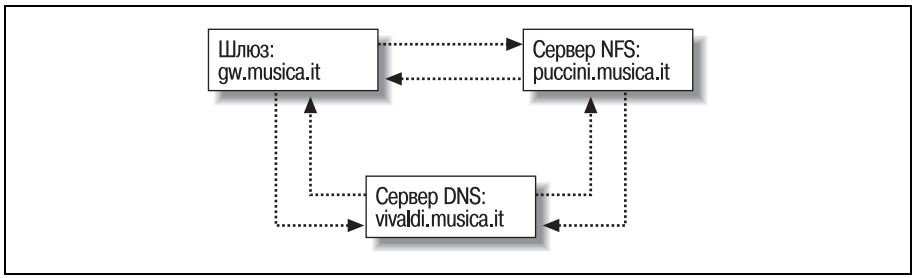

***Рис. 5.2.** Узлы, типично определяемые в /etc/hosts, включая локальный*

Серверы DNS

Solaris поддерживает три вида серверов DNS: первичные, вторичные и кэширующие. Первичный сервер DNS хранит все достоверные данные конкретного домена (то есть является компетентным). Таким образом, любой запрос к первичному серверу DNS должен всегда возвращать наиболее актуальную информацию по отображению имен в адреса и адресов в имена узлов. С другой стороны, вторичный сервер DNS явля-

ется дублирующей системой для первичного сервера; если первичный сервер по какой-либо причине недоступен, наиболее достоверными становятся данные вторичного сервера имен. И наконец, кэширующий сервер имен не хранит достоверные данных локального домена, он просто передает все запросы на разрешение имен и адресов подходящему первичному или вторичному серверу.

Настройка сервера DNS (и запуск службы BIND) требует решения ряда вопросов. Эти вопросы могут влиять на содержание файла */etc/named.conf*, который является файлом настройки BIND. Для работы BIND требуется следующая информация:

- Списки управления доступом
- Дополнительные файлы настройки
- Объем и состав информации, попадающей в журнал
- Параметры работы для локального и удаленных серверов
- Определения локальных зон DNS

Файл */etc/named.conf* состоит из нескольких разделов, которые примерно соответствуют перечисленным требованиям. Возьмем пример файла */etc/named.conf* и рассмотрим параметры и функциональность каждого из его разделов:

```
acl local_network
{
   10.64.18/24
};

include "/usr/named/zones.conf"

options
{
   directory "/usr/named";
   pid-file "/usr/named/pid";
}

server 10.64.18.32
{
   bogus yes;
   transfer-format many-answers;
}

zone "musica.it"
{
   type master;
   file "musica.it.db";
}

zone "32.18.64.in-addr.arpa"
{
   type master;
   file "musica.it.rev";
}
```

Раздел *acl* содержит набор списков управления доступа, которые явно разрешают (или запрещают) клиентам DNS доступ к серверу DNS. Применение списков управления доступом позволяет защитить сервер DNS от DOS-атак (denial-of-service, отказ в обслуживании), поскольку запросы от заблокированных клиентов игнорируются автоматически. В данном примере раздел *acl* разрешает доступ к серверу всем клиентам из сети `10.64.18.0`.

Директива *include* позволяет хранить определенные разделы */var/named.conf* во внешних файлах. Это упрощает сопровождение серверов с многочисленными файлами различных зон (примером может служить DNS-сервер интернет-провайдера). Такие файлы зон являются первыми кандидатами на отдельные файлы настройки. В разделе *include* нашего примера указан внешний файл *zones.conf*.

В разделе, посвященном ведению журнала, указывается, следует ли регистрировать запросы DNS. Такая возможность полезна в процессе отладки, когда не на все получаемые запросы выдаются корректные ответы. В обычном режиме ведение журнала создает дополнительную нагрузку на процессор и оперативную память, а потому, как правило, не используется.

Параметры сервера DNS позволяют указать полный путь к каталогу сервера, а также имя файла, в котором хранится идентификатор процесса *in.named*. Кроме того, можно включить в этот раздел параметры, влияющие на обработку запросов от удаленных серверов, например задать формат передачи зон.

Последний раздел перечисляет зоны, которые находятся в ведении локального сервера имен. Для каждого домена или поддомена, за который отвечает сервер имен (включая виртуальные узлы и виртуальные домены), следует создать отдельную зону и добавить соответствующую запись в файл *named.conf*.

Как вариант, если зональные данные хранятся во внешних файлах, следует поместить данные по каждому домену в отдельный файл зоны. Если вернуться к предыдущим примерам, то мы должны будем создать файлы *musica.it.db* и *musica.it.rev*. Файл *musica.it.db* для сети из четырех узлов должен содержать, по крайней мере, следующие записи:

```
@   IN     SOA    musica.it.    root.musica.it.
(
   2001030101     ;порядковый номер
   10800          ;трехчасовой интервал обновления
   1800           ;повторение попытки через тридцать минут
   1209600        ;устаревание через две недели
   604800         ;отрицательное TTL в одну неделю
)
   IN       NS     ns.musica.it.
   IN       MX     10   firewall.musica.it.
   firewall        IN     A      10.64.18.1     ;брандмауэр
```

```
vivaldi         IN      A       10.64.18.2      ;веб-сервер
puccini         IN      A       10.64.18.3      ;веб-сервер
schicchi        IN      A       10.64.18.4      ;kerberos
security        IN      CNAME   schicchi
```

Порядковый номер следует увеличивать каждый раз, когда изменяется файл зоны, чтобы внесенные изменения распространялись. В нашем примере порядковый номер имеет формат *ГГГГММДД*, плюс к нему добавлена метка номера изменения (01). Это позволяет однозначно идентифицировать каждое изменение файла зоны по дате и номеру изменения, что особенно полезно для предоставления ISP- и ASP-услуг, когда обновление зон может происходить часто. В данном файле зоны существуют адресные записи для четырех узлов: *firewall* (`10.64.18.1`), *vivaldi* (`10.64.18.2`), *puccini* (`10.64.18.3`) и *schicchi* (`10.64.18.4`). Кроме того, для узла *firewall* присутствует MX-запись (mail exchange, почтовая ретрансляция), говорящая о том, что SMTP-соединения для домена *musica.it* должны быть по умолчанию адресованы узлу *firewall*. Для узла *schicchi* создан CNAME-псевдоним *security*, и таким образом, имена *schicchi.musica.it* и *security.musica.it* относятся к одному и тому же физическому узлу.

Файл *musica.it.rev* для сети из четырех узлов должен содержать, по крайней мере, следующие записи:

```
@   IN      SOA   32.18.64.in-addr.arpa.root.musica.it.
(
    2001030101      ;порядковый номер
    10800           ;трехчасовой интервал обновления
    1800            ;повторение попытки через тридцать минут
    1209600         ;устаревание через две недели
    604800          ;отрицательное TTL в одну неделю
)
            IN      NS      ns.musica.it.
1           IN      PTR     firewall.musica.it.
2           IN      PTR     vivaldi.musica.it.
3           IN      PTR     puccini.musica.it.
4           IN      PTR     schicchi.musica.it.
```

Здесь мы видим определения для адресов `10.24.58.1` (*firewall*), `10.24.58.2` (*vivaldi*), `10.24.58.3` (*puccini*) и `10.24.58.4` (*schicchi*) плюс некоторые параметры настройки, которые мы уже встречали в файле *musica.it.db*.

DNSTool

Если процесс ручной настройки DNS кажется чрезмерно сложным или долгим, можно воспользоваться рядом инструментов автоматизации управления доменами DNS. Одной из популярных программ является DNSTool (*http://www.gormand.com.au/tools/dnstool/guide.html*), в которой применяется система шаблонов и сценарии на языке Perl для ав-

томатического создания всех файлов, необходимых для управления одним или несколькими доменами. Эффективное применение DNSTool так же требует понимания всех рассмотренных уже концепций, таких как зоны и включаемые файлы. Однако DNSTool освобождает системного администратора от необходимости ручного сопровождения всех записей и предотвращает возможные ошибки в написании доменных имен или IP-адресов. Для файлов зон, задающих отображение доменных имен в адреса, будут автоматически созданы файлы отображения адресов в имена. DNSTool может использоваться и для создания более сложных конфигураций сетей и узлов, в которых, например, применяются многоканальные узлы и маршрутизаторы.

Каждому из доменов высшего уровня, находящихся в ведении DNSTool, соответствует отдельный файл данных, в котором содержится информация по каждому из узлов домена. Запись узла должна содержать, по меньшей мере, его имя и IP-адрес. Так, узлу *firewall.musica.it* будет соответствовать такая запись:

```
{
   host = firewall
   ip   = 10.64.18.1
}
```

Полная запись для узла *firewall.musica.it*:

```
{
    host     = firewall
    ip       = 10.64.18.1
    aliases  = ns
    location = "Music Hall 1"
    owner    = "Italia Music Productions"
    hostid   = 0x5c8dc65a223b
    ether    = 5C:8D:C6:5A:22:3B
    os       = "Solaris 8"
    hardware = "Sun Blade 100"
    comment  = "Firewall server for Music Hall 1"
}
```

После того как созданы файлы доменов, администратор просто выполняет команду *make* для генерации собственно файлов зон. Изменения становятся доступны после перезапуска процесса сервера DNS.

Клиенты DNS

В Solaris существует большое число клиентских инструментов, которые можно использовать для преобразования IP-адресов в имена и наоборот. Запросы могут выполняться как для локальных сетей, так и для доменов высшего уровня. Чтобы применить клиентское DNS-приложение, следует убедиться, что поисковый анализатор (resolver) системы настроен правильно. Прежде всего, необходимо указать имя ло-

кального домена и перечень IP-адресов доступных серверов имен в файле */etc/resolv.conf*. Для домена *cassowary.net* файл */etc/resolv.conf* может содержать такие записи:

```
domain cassowary.net
nameserver 10.64.18.2
nameserver 198.41.0.4
```

В этом примере локальный сервер DNS указан первым в порядке опроса. Если по каким-то причинам этот сервер недоступен (сбой системы или сети), клиент DNS по-прежнему сможет запрашивать разрешение для внешних IP-адресов и имен узлов, обращаясь к корневому серверу `198.41.0.4`.

nslookup

Команда *nslookup* используется для выполнения прямого и обратного поиска для отдельных узлов и целых доменов. Это первый инструмент, которым следует пользоваться при проверке подлинности IP-адресов и имен доменов. Например, чтобы найти IP-адрес узла *www.cassowary.net*, выполните команду:

```
bash-2.03# nslookup www.cassowary.net
```

Если узел существует, будет отображен его IP-адрес:

```
Server:  ns.cassowary.net
Address:  207.150.192.1
Name:    www.cassowary.net
Address:  207.150.192.12
```

В противном случае будет получено сообщение об ошибке, сводящееся к тому, что для узла не существуют адресные записи. Если же недоступен собственно сервер имен, сообщение об ошибке будет выглядеть приблизительно так:

```
*** Can't find server name for address 207.150.192.1: Server failed
*** Default servers are not available
```

nslookup может работать и в диалоговом режиме, для перехода в который следует просто выполнить программу без параметров:

```
bash-2.03# nslookup
Server:  ns.cassowary.net
Address:  207.150.192.1
>
```

В ответ на приглашение > можно вводить произвольное число адресов или доменов, для которых необходимо произвести поиск, без повторного набора команды *nslookup*. Одной из особенностей диалогового режима *nslookup* является возможность искать по различным критериям, в частности запрашивать для домена запись начала компетенции

(Start of Authority, SOA). Следующие команды выполняют поиск SOA-записи для домена *cassowary.net*:

```
> set q=soa
> www.cassowary.net
Server:  ns.paulwatters.com
Address:  10.64.18.1

cassowary.net
        origin = nsah1.ahnet.net
        mail addr = contact.ahnet.net
        serial = 2001031413
        refresh = 21600 (6 hours)
        retry   = 3600 (1 hour)
        expire  = 691200 (8 days)
        minimum ttl = 86400 (1 day)
cassowary.net   nameserver = nsah1.ahnet.net
cassowary.net   nameserver = ns3.pbi.net
nsah1.ahnet.net internet address = 207.213.224.16
ns3.pbi.net     internet address = 206.13.28.165
```

Отображение SOA-записи в том виде, в каком она существует на сервере, полезно для проверки записей из соответствующих файлов зон. К примеру, если порядковый номер, полученный при помощи *nslookup*, отличается от порядкового номера в файле зоны, это может означать, что новые данные еще не распространились либо вовсе не будут распространены.

Диалоговый режим *nslookup* может также применяться для получения списка серверов, являющихся компетентными для конкретного домена. Если мы желаем определить, какие серверы являются компетентными для домена высшего уровня *.ph* (Филиппины), то воспользуемся такой командой:

```
bash-2.03# nslookup
> set type=ns
> ph.
Server:  ns.cassowary.net
Address:  207.150.192.1

Non-authoritative answer:
ph      nameserver = NS.RIPE.NET
ph      nameserver = NS.UU.NET
ph      nameserver = AUTH00.NS.UU.NET
ph      nameserver = AUTH50.NS.UU.NET
ph      nameserver = MAKISIG.IPHIL.NET
ph      nameserver = NS.EU.NET

Authoritative answers can be found from:
NS.RIPE.NET             IPv6 address = ::ffff:193.0.0.193
NS.RIPE.NET             internet address = 193.0.0.193
NS.UU.NET               internet address = 137.39.1.3
AUTH00.NS.UU.NET        internet address = 198.6.1.65
```

```
AUTH50.NS.UU.NET        internet address = 198.6.1.161
MAKISIG.IPHIL.NET       internet address = 203.176.28.135
NS.EU.NET               internet address = 192.16.202.11
```

Итак, доступен целый ряд альтернативных серверов, от которых может быть получена достоверная информация по доменам, принадлежащим домену высшего уровня *.ph*.

Когда найден компетентный сервер, часто возможно получение списка доменов и сетей, расположенных в иерархии данного домена высшего уровня:

```
bash-2.03# nslookup
> server ns.eu.net
Default Server:  ns.eu.net
Address:  192.16.202.11
> set type=ns
> ls ph.

[ns.eu.net]
 ph.                            server = ns.UU.NET
 ph.                            server = auth00.ns.uu.net
 ph.                            server = auth50.ns.uu.net
 ph.                            server = ns.ripe.net
 ph.                            server = ns.eu.net
 ph.                            210.23.230.195
 sexy                           server = dns1.bell.com.ph
 dns1.bell.com                  203.148.66.100
 sexy                           server = dns2.bell.com.ph
 dns2.bell.com                  203.148.66.101
 chupita                        server = dns1u.emc.com.ph
 dns1u.emc.com                  203.167.64.2
 chupita                        server = dns2u.emc.com.ph
 dns2u.emc.com                  203.167.64.3
 newspaper                      server = ns.gpserver.com
 newspaper                      server = ns2.gpserver.com
```

whois

Командой *whois* можно воспользоваться для получения административной информации по конкретному домену – из централизованной базы данных регистраторов. Эту информацию можно получить через веб-интерфейс по адресу *http://www.nsi.com/*, но многие администраторы находят, что эквивалент командной строки удобнее:

```
bash-2.03# whois cassowary.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: CASSOWARY.NET
```

```
   Registrar: INTERNET DOMAIN REGISTRARS
   Whois Server: whois.registrars.com
   Referral URL: www.registrars.com
   Name Server: NS1.HOSTSAVE.COM
   Name Server: NS3.PBI.NET
   Updated Date: 29-oct-2000

>>> Last update of whois database: Sun, 18 Mar 2001 13:18:40 EST <<<

The Registry database contains ONLY .COM, .NET, .ORG, .EDU domains and
Registrars.
```

dig

Команда *dig* позволяет извлекать записи DNS для конкретной зоны. Могут выполняться запросы адресных записей, записей указателей, почтовых ретрансляторов, службы имен, начала компетенции, информации об узле, текстовые и другие. В следующих разделах мы рассмотрим вывод *dig* для каждого из случаев.

Адресные (A). Ниже приведен вывод *dig* для адресных записей домена *cassowary.net*:

```
; <<>> DiG 2.1 <<>> @nsah1.ahnet.net cassowary.net A
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10
;; flags: qr aa rd ra; Ques: 1, Ans: 1, Auth: 2, Addit: 2
;; QUESTIONS:
;; cassowary.net, type = A, class = IN
;; ANSWERS:
cassowary.net. 691200 A 207.150.192.12
;; AUTHORITY RECORDS:
cassowary.net. 691200 NS nsah1.ahnet.net.
cassowary.net. 691200 NS ns3.pbi.net.
;; ADDITIONAL RECORDS:
nsah1.ahnet.net. 300 A 207.213.224.16
ns3.pbi.net. 128374 A 206.13.28.165
;; Total query time: 174 msec
;; FROM: ns.paulwatters.com to SERVER: nsah1.ahnet.net 207.213.224.16
;; WHEN: Mon Mar 19 02:48:36 2001
;; MSG SIZE sent: 31 rcvd: 127
```

Указатели (PTR). Ниже приведен вывод *dig* для записей указателей домена *cassowary.net*:

```
; <<>> DiG 2.1 <<>> @nsah1.ahnet.net cassowary.net PTR
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10
;; flags: qr aa rd ra; Ques: 1, Ans: 0, Auth: 1, Addit: 0
```

```
;; QUESTIONS:
;; cassowary.net, type = PTR, class = IN
;; AUTHORITY RECORDS: cassowary.net. 86400 SOA nsah1.ahnet.net.
contact.ahnet.net. ( 2001031413 serial 21600 refresh (6 hours) 3600 retry (1
hour) 691200 expire (8 days) 86400 ) minimum (1 day)
;; Total query time: 114 msec
;; FROM: ns.paulwatters.com to SERVER: nsah1.ahnet.net 207.213.224.16 ;;
WHEN: Mon Mar 19 02:51: 17 2001
;; MSG SIZE sent: 31 rcvd: 87
```

Почтовые ретрансляторы (MX). Ниже приведен вывод *dig* для MX-записей домена *cassowary.net*:

```
; <<>> DiG 2.1 <<>> @nsah1.ahnet.net cassowary.net MX
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10
;; flags: qr aa rd ra; Ques: 1, Ans: 4, Auth: 2, Addit: 7
;; QUESTIONS:
;; cassowary.net, type = MX, class = IN
;; ANSWERS:
cassowary.net.
691200
MX
100 smtp-relay.pbi.net.
cassowary.net.
691200
MX
0 mail-incoming.ahnet.net.
cassowary.net.
691200
MX
10 mail-incoming2.ahnet.net.
cassowary.net.
691200
MX
20 mail.ahnet.net.
;; AUTHORITY RECORDS:
cassowary.net.
691200
NS
nsah1.ahnet.net.
cassowary.net.
691200
NS
ns3.pbi.net.
;; ADDITIONAL RECORDS:
smtp-relay.pbi.net.
184
A
```

```
206.13.28.8
smtp-relay.pbi.net.
184
A
206.13.28.30
mail-incoming.ahnet.net.
300
A
207.150.192.13
mail-incoming2.ahnet.net.
300
A
207.150.192.80
mail.ahnet.net.
300
A
207.150.192.80
nsah1.ahnet.net.
300
A
207.213.224.16
ns3.pbi.net.
128195
A
206.13.28.165
;; Total query time: 118 msec
;; FROM: ns.paulwatters.com to SERVER: nsah1.ahnet.net 207.213.224.16
;; WHEN: Mon Mar 19 02:51:35 2001
;; MSG SIZE sent: 31 rcvd: 300
```

Серверы имен (NS). Ниже приведен вывод *dig* для NS-записей домена *cassowary.net*:

```
; <<>> DiG 2.1 <<>> @nsah1.ahnet.net cassowary.net NS
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10
;; flags: qr aa rd ra; Ques: 1, Ans: 2, Auth: 0, Addit: 2

;; QUESTIONS:
;; cassowary.net, type = NS, class = IN

;; ANSWERS:
cassowary.net.    691200  NS     nsah1.ahnet.net.
cassowary.net.    691200  NS     ns3.pbi.net.

;; ADDITIONAL RECORDS:
nsah1.ahnet.net.   300    A      207.213.224.16
ns3.pbi.net.       128132 A      206.13.28.165

;; Total query time: 115 msec
;; FROM: ns.paulwatters.com to SERVER: nsah1.ahnet.net 207.213.224.16
```

```
;; WHEN: Mon Mar 19 02:52:38 2001
;; MSG SIZE sent: 31 rcvd: 111
```

Начало компетенции (SOA). Ниже приведен вывод *dig* для записей начала компетенции домена *cassowary.net*:

```
; <<>> DiG 2.1 <<>> @nsah1.ahnet.net cassowary.net SOA
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10
;; flags: qr aa rd ra; Ques: 1, Ans: 1, Auth: 2, Addit: 2
;; QUESTIONS:
;; cassowary.net, type = SOA, class = IN

;; ANSWERS:
cassowary.net. 691200  SOA nsah1.ahnet.net. contact.ahnet.net. (
                              2001031413
                              serial 21600
                              refresh (6 hours) 3600
                              retry (1 hour) 691200
                              expire (8 days) 6400
                    ) minimum (1 day)

;; AUTHORITY RECORDS:
cassowary.net.                691200          NS
                              nsah1.ahnet.net.
cassowary.net.                691200          NS
                              ns3.pbi.net.

;; ADDITIONAL RECORDS:
nsah1.ahnet.net.              300             A
                              207.213.224.16
ns3.pbi.net.                  128120          A
                              206.13.28.165

;; Total query time: 114 msec
;; FROM: ns.paulwatters.com to SERVER: nsah1.ahnet.net 207.213.224.16
;; WHEN: Mon Mar 19 02:52:50 2001
;; MSG SIZE sent: 31 rcvd: 155
```

Информация об узлах (HINFO). Ниже приведен вывод *dig* для HINFO-записей домена *cassowary.net*:

```
; <<>> DiG 2.1 <<>> @nsah1.ahnet.net cassowary.net HINFO
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10
;; flags: qr aa rd ra; Ques: 1, Ans: 0, Auth: 1, Addit: 0
;; QUESTIONS:
;; cassowary.net, type = HINFO, class = IN

;; AUTHORITY RECORDS:
```

```
cassowary.net. 86400 SOA nsah1.ahnet.net. contact.ahnet.net. (
  2001031413  serial
  21600  refresh (6 hours)
  3600  retry (1 hour)
  691200  expire (8 days)
  86400 )  minimum (1 day)

;; Total query time: 114 msec
;; FROM: ns.paulwatters.com to SERVER: nsah1.ahnet.net 207.213.224.16
;; WHEN: Mon Mar 19 02:53:06 2001
;; MSG SIZE sent: 31 rcvd: 87
```

Текстовые (TXT). Ниже приведен вывод *dig* для текстовых записей домена *cassowary.net*:

```
; <<>> DiG 2.1 <<>> @nsah1.ahnet.net cassowary.net TXT
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10
;; flags: qr aa rd ra; Ques: 1, Ans: 0, Auth: 1, Addit: 0
;; QUESTIONS:
;; cassowary.net, type = TXT, class = IN

;; AUTHORITY RECORDS:
cassowary.net. 86400 SOA nsah1.ahnet.net. contact.ahnet.net. (
  2001031413  serial
  21600  refresh (6 hours)
  3600  retry (1 hour)
  691200  expire (8 days)
  86400 )  minimum (1 day)

;; Total query time: 113 msec
;; FROM: ns.paulwatters.com to SERVER: nsah1.ahnet.net 207.213.224.16
;; WHEN: Mon Mar 19 02:53:22 2001
;; MSG SIZE sent: 31 rcvd: 87
```

Любые записи (ANY). Ниже приведен вывод *dig* для любых записей домена *cassowary.net*:

```
; <<>> DiG 2.1 <<>> @nsah1.ahnet.net cassowary.net ANY
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10
;; flags: qr aa rd ra; Ques: 1, Ans: 8, Auth: 2, Addit: 7
;; QUESTIONS:
;; cassowary.net, type = ANY, class = IN

;; ANSWERS:
cassowary.net. 691200 NS nsah1.ahnet.net.
cassowary.net. 691200 NS ns3.pbi.net.
cassowary.net. 691200 SOA nsah1.ahnet.net. contact.ahnet.net. (
  2001031413  serial
```

```
    21600  refresh (6 hours)
    3600  retry (1 hour)
    691200  expire (8 days)
    86400 )  minimum (1 day)
cassowary.net. 691200 A 207.150.192.12
cassowary.net. 691200 MX 0 mail-incoming.ahnet.net.
cassowary.net. 691200 MX 10 mail-incoming2.ahnet.net.
cassowary.net. 691200 MX 20 mail.ahnet.net.
cassowary.net. 691200 MX 100 smtp-relay.pbi.net.

;; AUTHORITY RECORDS:
cassowary.net. 691200 NS nsah1.ahnet.net.
cassowary.net. 691200 NS ns3.pbi.net.

;; ADDITIONAL RECORDS:
nsah1.ahnet.net. 300 A 207.213.224.16
ns3.pbi.net. 128069 A 206.13.28.165
mail-incoming.ahnet.net. 300 A 207.150.192.13
mail-incoming2.ahnet.net. 300 A 207.150.192.80
mail.ahnet.net. 300 A 207.150.192.80
smtp-relay.pbi.net. 58 A 206.13.28.8
smtp-relay.pbi.net. 58 A 206.13.28.30

;; Total query time: 115 msec
;; FROM: ns.paulwatters.com to SERVER: nsah1.ahnet.net 207.213.224.16
;; WHEN: Mon Mar 19 02:53:41 2001
;; MSG SIZE sent: 31 rcvd: 388
```

Сетевая информационная служба (NIS)

Грубо говоря, сетевая информационная служба (Network Information Service, NIS) – это сетевая версия некоторых файлов, прописанных в каталоге */etc* отдельного узла. Известная также под названием «желтые страницы», служба NIS является централизованным хранилищем данных, связанных с локальной сетью и доступных всем узлам этой сети. Сопровождение версий файлов сетевых настроек в каталоге */etc* вполне работает в случае одного узла, но по мере роста числа узлов в сети становится все труднее сохранять согласованность их настроек. Например, если пользователям необходимо работать с различными узлами локальной сети, то их пароли должны храниться на различных машинах. Если речь идет о кластере из двадцати систем, будет затруднительно регистрироваться на каждой машине и выполнять команду *passwd*. Некоторые организации пошли путем разработки собственных решений распространения паролей на базе CORBA и сходных технологий, а NIS позволяет хранить единственную копию файла паролей на сервере NIS, к которому имеют доступ все клиенты. Таким образом, если пользователь изменит свой пароль на одном из узлов, изменение отправляется прямо на сервер NIS. После этого любой узел, к которому обращается пользователь, будет производить проверку под-

линности по новому паролю. Такая система значительно упрощает системное администрирование в крупных сетях.

Управление данными сетевых настроек организовано в NIS с помощью наборов карт, причем на каждую службу NIS приходится, по меньшей мере, одна карта. Существующие карты:

bootparams

Перечисление параметров загрузки существующих бездисковых клиентов.

ethers.byaddr

Отображение Ethernet-адресов в имена узлов.

ethers.byname

Отображение имен узлов в Etherneta-адреса.

group.bygid

Отображение идентификаторов групп (GID) в имена групп.

group.byname

Отображение имен групп в идентификаторы групп (GID).

hosts.byaddr

Отображение IP-адресов в имена узлов.

hosts.byname

Отображение имен узлов в IP-адреса.

mail.aliases

Отображение почтовых псевдонимов в имена пользователей.

mail.byaddr

Отображение имен пользователей в почтовые псевдонимы.

netgroup

Перечисление локальных сетевых групп (netgroups).

netgroup.byhost

Перечисление локальных сетевых групп, упорядоченных по именам узлов.

netgroup.byuser

Перечисление локальных сетевых групп, упорядоченных по именам пользователей.

netmasks.byaddr

Перечисление допустимых локальных сетевых масок.

networks.byaddr

Перечисление локальных сетей по адресам.

networks.byname

Перечисление локальных сетей по именам.

passwd.byname

База паролей, упорядоченная по именам пользователей.

passwd.byuid

База паролей, упорядоченная по идентификаторам пользователей (UID).

protocols.byname

Отображение имен протоколов в номера протоколов.

protocols.bynumber

Отображение номеров протоколов в имена протоколов.

publickey.byname

Перечень доступных локальных ключей RPC (открытых).

rpc.bynumber

Перечень доступных локальных служб RPC.

services.byname

Перечень доступных локальных сетевых служб.

ypservers

Перечень доступных локальных серверов NIS.

Сетевая информационная служба (NIS+)

Сетевая информационная служба NIS+ представляет собой значительно более совершенную версию NIS. В NIS+ появились две важных особенности: поддержка иерархических пространств имен, позволяющая создавать домены и поддомены, в пределах которых производится управление конкретными службами, а также более надежные процедуры проверки подлинности с применением шифрования DES. Появление таблиц NIS+ привело к увеличению числа и диапазона карт NIS, а поддержка автоматического монтирования существует в явном виде.

Каково главное преимущество иерархического пространства имен? Вообразите масштабы хаоса, который царил бы в Интернете, если бы система доменных имен не была иерархической! В сетях из более чем ста узлов применение стандартного пространства имен NIS часто приводило к путанице. Структура иерархического пространства имен позволяет организовывать доступ к внутренней информации сети с использованием тех сетевых структур, что определяются для DNS: домену DNS *sales.cassowary.net* соответствовал бы домен NIS+ *Sales.Cassowary.Net*; домену DNS *java.cassowary.net* – домен NIS+ *Java.Cassowary.Net* и т. д. NIS+ является более совершенной службой, нежели NIS.

Новые возможности обеспечения безопасности в NIS+ являются большим шагом вперед по сравнению с относительно небезопасной NIS: весь доступ к данным таблиц NIS+ управляется продуманной систе-

мой разграничения доступа и обращение к сетевым данным должно быть явным образом разрешено пользователям, вместо того чтобы быть доступным всем и каждому. Обмен информацией при проверке подлинности в NIS+ уже не происходит в открытую, напротив, данные шифруются по алгоритму DES. Права доступа пользователя могут быть изучены с помощью команды *niscat*.

Перечислим основные таблицы NIS+:

hosts

Сопоставляет имена узлов с IP-адресами в пределах локального домена, реализует поддержку множественных псевдонимов узлов.

bootparams

Определяет параметры бездисковых клиентов, которым для загрузки необходимо обращение к серверу NIS+.

passwd

Содержит параметры идентификации и пользовательские данные для всех пользователей домена.

group

Определяет группы пользователей в пределах локального домена.

netgroups

Содержит списки доступа к ресурсам, определяющие, каким из пользователей разрешен доступ к конкретным службам и выполнение определенных действий в сети.

mail-aliases

Позволяет создавать логические псевдонимы для пользовательских учетных записей в пределах домена, псевдонимы могут ассоциироваться с отдельными пользователями или группами пользователей.

timezone

Гарантирует, что все системы пользуются часовым поясом, соответствующим географическому расположению.

networks

Содержит список сетей, видимых для узлов локального домена.

netmasks

Содержит список сетевых масок, необходимых для поддержания доступа в локальной сети (как правило, речь идет о сетях классов A, B и C).

ethers

Связывает каждый узел с уникальным Ethernet-адресом в пределах локального домена.

services

Определяет все TCP- и UDP-службы, доступные в пределах домена.

protocols

Определяет все сетевые протоколы, необходимые для функционирования служб локальной сети.

rpc

Определяет серверы RPC, доступные локальному домену.

auto_home

Поддержка автоматического монтирования домашних каталогов, существующих на выделенном NFS-сервере.

auto_master

Информация по точкам монтирования, необходимая для экспорта файлов по NFS.

Инициализация домена NIS+ – задача вполне тривиальная, если воспользоваться стандартными сценариями Solaris. Прежде всего, следует создать корневой мастер-сервер для домена при помощи команды *nisserver*:

```
# /usr/lib/nis/nisserver -r -d Cassowary.Net.
This script sets up this machine "austin" as an NIS+
root master server for domain Cassowary.Net..

Domain name             : Cassowary.Net.
NIS+ group              : admin.Cassowary.Net.
NIS (YP) compatibility  : OFF
Security level          : 2=DES

Is this information correct? (type 'y' to accept, 'n' to change) y
```

Стандартный вариант установки приводит к принудительному шифрованию данных идентификации по алгоритму DES и созданию группы *admin.Cassowary.Net* для домена *Cassowary.Net*. Ввод **y** приводит к следующему обмену:

```
This script will set up your machine as a root master server for
domain Cassowary.Net. without NIS compatibility at security level 2.

Use "nisclient -r" to restore your current network service environment.

Do you want to continue? (type 'y' to continue, 'n' to exit this script)

setting up domain information "Cassowary.Net." ...

setting up switch information ...

running nisinit ...
This machine is in the "Cassowary.Net." NIS+ domain.
Setting up root server .

starting root server at security level 0 to create credentials...

running nissetup to create standard directories and tables ...
```

```
running nissetup to create standard directories and tables ...
org_dir.Cassowary.Net. created
groups_dir.Cassowary.Net. created
passwd.org_dir.Cassowary.Net. created
group.org_dir.Cassowary.Net. created
auto_master.org_dir.Cassowary.Net. created
auto_home.org_dir.Cassowary.Net. created
bootparams.org_dir.Cassowary.Net. created
cred.org_dir.Cassowary.Net. created
ethers.org_dir.Cassowary.Net. created
hosts.org_dir.Cassowary.Net. created
ipnodes.org_dir.Cassowary.Net. created
mail_aliases.org_dir.Cassowary.Net. created
sendmailvars.org_dir.Cassowary.Net. created
netmasks.org_dir.Cassowary.Net. created
netgroup.org_dir.Cassowary.Net. created
networks.org_dir.Cassowary.Net. created
protocols.org_dir.Cassowary.Net. created
rpc.org_dir.Cassowary.Net. created
services.org_dir.Cassowary.Net. created
timezone.org_dir.Cassowary.Net. created
client_info.org_dir.Cassowary.Net. created
auth_attr.org_dir.Cassowary.Net. created
exec_attr.org_dir.Cassowary.Net. created
prof_attr.org_dir.Cassowary.Net. created
user_attr.org_dir.Cassowary.Net. created
audit_user.org_dir.Cassowary.Net. created

adding credential for austin.Cassowary.Net...
Enter login password:
creating NIS+ administration group: admin.Cassowary.Net. ...
adding principal austin.Cassowary.Net. to admin.Cassowary.Net. ...

restarting NIS+ root master server at security level 2 ...
starting NIS+ password daemon ...
starting NIS+ cache manager ...

This system is now configured as a root server for domain Cassowary.Net.
You can now populate the standard NIS+ tables by using the
nispopulate script or /usr/lib/nis/nisaddent command.
```

К этому моменту инициализированы все перечисленные выше таблицы. Следует, как и предлагается, воспользоваться сценарием *nispopulate* для предоставления доступа клиентам:

```
# /usr/lib/nis/nispopulate -F -p /etc -d Cassowary.Net.

NIS+ domain name                : Cassowary.Net.
Directory Path                  : /etc

Is this information correct? (type 'y' to accept, 'n' to change) y

This script will populate the standard NIS+ tables for domain
Cassowary.Net. from the files in /etc:
```

```
auto_master auto_home ethers group hosts ipnodes networks passwd protocols servi
ces rpc netmasks bootparams netgroup aliases timezone auth_attr exec_attr prof_a
ttr user_attr audit_user shadow

**WARNING:  Interrupting this script after choosing to continue
may leave the tables only partially populated.  This script does
not do any automatic recovery or cleanup.

Do you want to continue? (type 'y' to continue, 'n' to exit this script)
populating auto_master table from file /etc/auto_master...
auto_master table done.

populating auto_home table from file /etc/auto_home...
auto_home table done.

**WARNING: file /etc/ethers does not exist!
        ethers table will not be loaded.

populating group table from file /etc/group...
group table done.

populating hosts table from file /etc/hosts...
hosts table done.

Populating the NIS+ credential table for domain Cassowary.Net.
from hosts table.

dumping hosts table...
loading credential table...

The credential table for domain Cassowary.Net. has been populated.

The password used will be nisplus.

**WARNING: file /etc/ipnodes does not exist!
        ipnodes table will not be loaded.

populating networks table from file /etc/networks...
networks table done.
populating passwd table from file /etc/passwd...
passwd table done.

Populating the NIS+ credential table for domain Cassowary.Net.
from passwd table.

dumping passwd table...
loading credential table...
nisaddcred: need not add LOCAL entry for root
nisaddcred: unable to create credential.

The credential table for domain Cassowary.Net. has been populated.

The password used will be nisplus.

populating protocols table from file /etc/protocols...
protocols table done.

populating services table from file /etc/services...
services table done.
```

```
populating rpc table from file /etc/rpc...
rpc table done.

populating netmasks table from file /etc/netmasks...
netmasks table done.

**WARNING: file /etc/bootparams does not exist!
        bootparams table will not be loaded.
**WARNING: file /etc/netgroup does not exist!
        netgroup table will not be loaded.

populating mail_aliases table from file /etc/aliases...
mail_aliases table done.

**WARNING: file /etc/timezone does not exist!
        timezone table will not be loaded.

**WARNING: file /etc/auth_attr does not exist!
        auth_attr table will not be loaded.

**WARNING: file /etc/exec_attr does not exist!
        exec_attr table will not be loaded.

**WARNING: file /etc/prof_attr does not exist!
        prof_attr table will not be loaded.

populating user_attr table from file /etc/user_attr...
user_attr table done.

**WARNING: file /etc/audit_user does not exist!
        audit_user table will not be loaded.

populating passwd table from file /etc/shadow...
passwd table done.
Credentials have been added for the entries in the
hosts and passwd table(s).  Each entry was given a default
network password (also known as a Secure-RPC password).
This password is:

                  nisplus

Use this password when the nisclient script requests the
network password.

nispopulate failed to populate the following tables:
 ethers ipnodes bootparams netgroup timezone auth_attr exec_attr prof_attr audit
_user
```

И хотя некоторые из таблиц не были заполнены по причине отсутствия нужных файлов в каталоге */etc* (скажем, файла */etc/ethers*, который обычно используется для заполнения таблицы *ethers*), наиболее важные таблицы были успешно созданы. Чтобы убедиться, что служба NIS+ активизирована, воспользуйтесь командой *nisping*:

```
# /usr/lib/nis/nisping -C Cassowary.Net.
Checkpointing replicas serving directory "Cassowary.Net." :
Master server is "austin.Cassowary.Net."
        Last update occurred at Thu Apr  5 19:31:58 2001
```

```
Master server is "austin.Cassowary.Net."
checkpoint scheduled on "austin.Cassowary.Net.".
```

Облегченный протокол доступа к каталогам (LDAP)

Облегченный протокол доступа к каталогам (Lightweight Directory Access Protocol, LDAP) является независимым от платформы открытым стандартом протокола доступа, основанным на информационной модели X.500. Протокол работает на базе TCP/IP и использует простые строковые кодировки. Приложения LDAP реализуются в архитектуре клиент–сервер; клиентская библиотека в составе Solaris позволяет разработчикам создавать приложения LDAP, а пользователям выполнять LDAP-ориентированные программы.

LDAP реализует четыре связанных службы управления информацией: структура каталога, именование записей, пользовательский доступ и безопасность. Структура каталога зависит от информационных характеристик записей, хранимых в каталоге. Эта структура инкапсулируется в схеме каталога, которой соответствуют все хранимые в каталоге записи. И хотя существует возможность определять собственные схемы, клиенты и серверы LDAP полагаются на стандартный набор схем для совместного доступа к данным. Для пользователей Solaris может быть создана разумная схема для представления карт NIS.

Структура каталога LDAP представлена иерархией различимых компонентов (Distinguished Components, DC), которые могут применяться для поиска конкретного объекта в каталоге. Существует возможность считывать и изменять атрибуты объектов, для которых известны DC. Пространства имен в LDAP аналогичны пространствам имен Java: объекты идентифицируются уточненными именами. Например, пакеты Java, разработанные *cassowary.net*, как правило, принадлежат пространству имен *Net.Cassowary*, а объекты LDAP для *cassowary.net* можно найти с помощью доменных компонентов *dc=cassowary, dc=net*.

LDAP реализует полную систему идентификации, проверки подлинности и авторизации, которая определяет, какие локальные и внешние пользователи могут читать и/или изменять объекты каталога. После прохождения клиентом проверки подлинности его способность читать, добавлять, обновлять или удалять записи определяется правами доступа. Эти права могут устанавливаться для отдельных пользователей или групп, а действовать для отдельных записей или целых каталогов. Это делает LDAP весьма гибкой системой имен, поскольку для чувствительных данных может быть реализовано тонкое управление доступом к объектам.

В составе Solaris 8 поставляется сервер каталогов iPlanet версии 4.11 (СК), который является реализацией LDAP v3, предназначенной для

применения в сетях Solaris. Сервер требует, по меньшей мере, 200 Мбайт дискового пространства для установки; при установке для производственных нужд рекомендуется более 1 Гбайт дискового пространства.

СК 4.11 в стандартной комплектации предоставляет следующие возможности:

- Веб-служба для HTTP-доступа к каталогу LDAP
- Полная поддержка SNMP для управления сетями
- Службы репликации
- Безопасную передачу данных с использованием SSL

Кроме того, в состав Solaris 8 входят дополнительные расширения, позволяющие с легкостью интегрировать LDAP с существующими службами имен:

- Клиент управления каталогами (Deja)
- Удаленная служба доступа для проверки подлинности внешних пользователей (RADIUS)
- Взаимодействие NIS-LDAP

Сервер СК LDAP называется *ns-slapd*. Перед установкой *ns-slapd* необходимо собрать следующие данные:

- Номер порта *ns-slapd* (например, *389*)
- Корень сервера LDAP (*/opt/iPlanet/ds*)
- Пользовательские и групповые права доступа для выполнения СК (например, *nobody/nobody*)
- Суффикс каталога LDAP (например, *dc=cassowary, dc=net*)
- Существующее доменное имя (например, *cassowary.net*)

Чтобы установить сервер каталогов, выполните следующую программу с компакт-диска установки iPlanet:

```
bash-2.03# /cdrom/iplanet_advantage_v1/DS/DS411/setup
```

Программа установки отображает краткую информацию о продукте СК:

```
Netscape Communications Corp.
              Netscape Server Products Installation/Uninstallation
--------------------------------------------------------------------------------

Welcome to the Netscape Server Products installation program
This program will install Netscape Server products and the
Netscape Console on your computer.

It is recommended that you have "root" privilege to install the software.

Tips for using the installation program:
  - Press "Enter" to choose the default and go to the next screen
  - Type "Control-B" to go back to the previous screen
  - Type "Control-C" to cancel the installation program
```

```
  - You can enter multiple items using commas to separate them.
    For example: 1, 2, 3

Would you like to continue with installation? [Yes]:
```

Набрав **Yes** и приняв условия лицензии, нужно выбрать, какие именно продукты будут установлены:

```
Select the items you would like to install:

   1. Netscape Servers
      Installs Netscape Servers with the integrated Netscape Console
      onto your computer.

   2. Netscape Console
      Installs Netscape Console
      as a stand-alone Java application on your computer.
```

Выбрав Netscape Servers, определите тип выполняемой установки:

```
Choose an installation type:

   1. Express installation
      Allows you to quickly install the servers using the most
      common options and pre-defined defaults. Useful for quick
      evaluation of the products.

   2. Typical installation
      Allows you to specify common defaults and options.

   3. Custom installation
      Allows you to specify more advanced options. This is
      recommended for experienced server administrators only.

To accept the default shown in brackets, press the Enter key.

Choose an installation type [1]:
```

Экспресс-установка – самый быстрый и самый простой вариант. Указав полный путь для установки, выберите компоненты и составляющие, которые следует установить:

```
Netscape Server Products components:

Components with a number in () contain additional subcomponents
which you can select using subsequent screens.

    1. Netscape Server Products Core Components (3)
    2. Netscape Directory Suite (2)
    3. Administration Services (2)

Specify the components you wish to install [All]:
```

В большинстве случаев можно воспользоваться вариантом *All* (устанавливаются все компоненты). Затем введите имя пользователя, с правами которого будет выполняться *ns-slapd* (обычно это *nobody*), и имя администратора. После этого происходит собственно установка и запуск демона.

6

Системное администрирование

Эта глава посвящена управлению пользователями, ресурсами и демонами в масштабе одного узла. Читатели узнают, как применять графический интерфейс *admintool* для добавления, изменения и удаления пользовательских записей, программных пакетов, последовательных портов и принтеров. Будут рассмотрены также инструменты командной строки, позволяющие выполнять те же действия. По мере появления в системе новых пользователей и ресурсов все важнее становится задача их эффективного сопровождения и обеспечения непрерывности действия служб. Кроме того, будет описано диалоговое управление доступом пользователей с помощью механизма квот Solaris. И наконец, будет рассмотрена установка и настройка такой сетевой службы, как почтовый агент *sendmail*.

Управление пользователями

В многопользовательской, многозадачной системе (каковой является Solaris) необходимо иметь способ предписывать и навязывать политику владения данными, расположенными в памяти или на дисках. Именно поэтому каждый процесс и каждый файл в системе Solaris «принадлежит» учетной записи, которая связана с конкретным идентификатором и именем пользователя. В однопользовательских системах (вроде MS-DOS) нет необходимости связывать файлы и данные в памяти с конкретными пользователями, поскольку в любой момент времени лишь один пользователь может работать с системой. Безопасность файлов и процессов, основанная на идентификации пользователей, является одним из признаков системы Unix.

В процессе установки Solaris создает ряд стандартных учетных записей, каждая из которых служит определенной цели. Например, учетная запись *root* всегда имеет идентификатор пользователя (UID), равный 0, и это означает, что запись получает полномочия суперпользователя. Однако можно заменить имя учетной записи *root* любым другим (например, *goon*), если существует необходимость сбить с толку приложения, которые знают суперпользователя по имени, но не идентификатору: любой пользователь с нулевым идентификатором получает полномочия суперпользователя.

Во многих организациях учетным записям дают названия на основе имен и/или инициалов пользователей: это позволяет легко ассоциировать действия учетных записей с конкретными пользователями. Например, мне в разное время выделялись учетные записи *pwatters*, *paul* и *paw*, которые являются производными от моего имени. Кроме того, можно создавать чисто ролевые учетные записи, скажем, *apache* для веб-сервера Apache.

Для коллективной обработки в определенных обстоятельствах пользователи системы Solaris могут объединяться в различные группы. В следующих разделах мы рассмотрим управление группами и входящими в них пользователями.

Файл паролей

Файл паролей является централизованным хранилищем всей информации пользовательских учетных записей в системе Solaris. Учетная запись пользователя обладает рядом свойств, некоторые из которых являются уникальными, а другие – общими для многих записей. Наборы пользователей, разделяющих определенные свойства, связанные правами чтения, записи и выполнения файлов, называются группами и определяются в базе данных групп (*/etc/group*). Одним из свойств записи, которое является общим для всех пользователей, является пароль, на основе которого производится проверка подлинности при доступе к системе. База паролей хранится в файле */etc/passwd* и содержит следующие поля:

- Имя пользователя (например, *pwatters*)
- Указатель теневого пароля (например, *x*)
- Идентификатор пользователя, или UID (например, 1001)
- Идентификатор группы, или GID (например, 100)
- Полное имя (например, Paul A. Watters)
- Домашний каталог (например, */home/pwatters*)
- Командный интерпретатор по умолчанию (например, */bin/bash*)

Типичный файл паролей выглядит следующим образом:

```
root:x:0:1:Super-User:/:/bin/ksh
daemon:x:1:1::/:
```

```
bin:x:2:2::/usr/bin:
sys:x:3:3::/:
adm:x:4:4:Administrator:/var/adm:
lp:x:71:8:Line Printer Administrator:/usr/spool/lp:
uucp:x:5:5:uucp Administrator:/usr/lib/uucp:
nuucp:x:9:9:uucp Administrator:/var/spool/uucppublic:/usr/lib/uucp/uucico
listen:x:37:4:Network Administrator:/usr/net/nls:
nobody:x:60001:60001:Nobody:/:
pwatters:x:1001:100:Paul Watters:/home/pwatters:/bin/bash
```

В отличие от ранних версий Solaris, более современные не хранят строки зашифрованных паролей в файле паролей. Для предотвращения несанкционированного доступа к зашифрованным паролям (которые традиционно хранились в файле */etc/passwd*, доступном для чтения всем) теперь применяется система теневых паролей. В системе теневых паролей существуют и дополнительные поля, которые могут использоваться для управления доступом учетной записи к системе – посредством организации устаревания, истечения срока действия и временной блокировки. Вследствие эффективной работы программ подбора паролей, таких как Crack, зашифрованные пароли теперь хранятся в теневой базе данных паролей (*/etc/shadow*), право на чтение которой есть только у администратора системы. Буква *x* во втором поле указывает на использование теневого пароля. Ниже показан типичный теневой файл паролей, некоторые из записей которого соответствуют приведенному выше файлу паролей:

```
root:g8h90fghf989:11241::::::
daemon:NP:6445::::::
bin:NP:6445::::::
sys:NP:6445::::::
adm:NP:6445::::::
lp:NP:6445::::::
uucp:NP:6445::::::
nuucp:NP:6445::::::
listen:*LK*:::::::
nobody:NP:6445::::::
pwatters:f7g9664k43off:11115:0::7:::0
```

Здесь хранятся первые два поля (имя пользователя и зашифрованный пароль), в то время как прочие поля, как правило, остаются пустыми. Вот интерпретация этих полей:

- Дата последнего изменения пароля (число дней, прошедших с первого января 1970 года)
- Число дней, через которое будет разрешено сменить пароль
- Число дней, до истечения которого пароль должен быть изменен
- Число дней, в течение которых пользователь получает предупреждения об окончании действия учетной записи
- Число дней после окончания действия учетной записи, по истечении которого учетная запись окончательно блокируется

- Число дней действующей блокировки учетной записи (с первого января 1970 года)

Зашифрованные пароли создаются автоматически с помощью функции *crypt( )*. Это односторонний DES-подобный шифр, а значит, пароли не проверяются вызовом функции, обратной *crypt( )*; это попросту невозможно. Именно эта особенность функции *crypt( )* делает пароли Solaris более стойкими ко взлому, чем пароли других операционных систем. При регистрации пользователя в системе введенный пароль шифруется с помощью функции *crypt( )*. Если полученная строка совпадает с соответствующим полем в файле */etc/passwd* (либо */etc/shadow*, если используются теневые пароли), подлинность пользователя считается установленной.

Максимальная длина пароля в Solaris – восемь символов, и для его создания могут применяться все алфавитно-цифровые символы (плюс определенные непечатаемые сочетания клавиш). Подобная система гарантирует, что взлом пароля Solaris прямым перебором займет слишком много процессорных лет.

Записи в файлы */etc/passwd* и */etc/shadow* могут добавляться вручную, но наиболее легкий способ создавать и изменять записи пользователей – при помощи инструмента *admintool*, показанного на рис. 6.1. Это приложение позволяет системному администратору создавать, удалять и изменять записи для пользователей и групп. Для выполнения тех же операций из командной строки можно воспользоваться командой *useradd*.

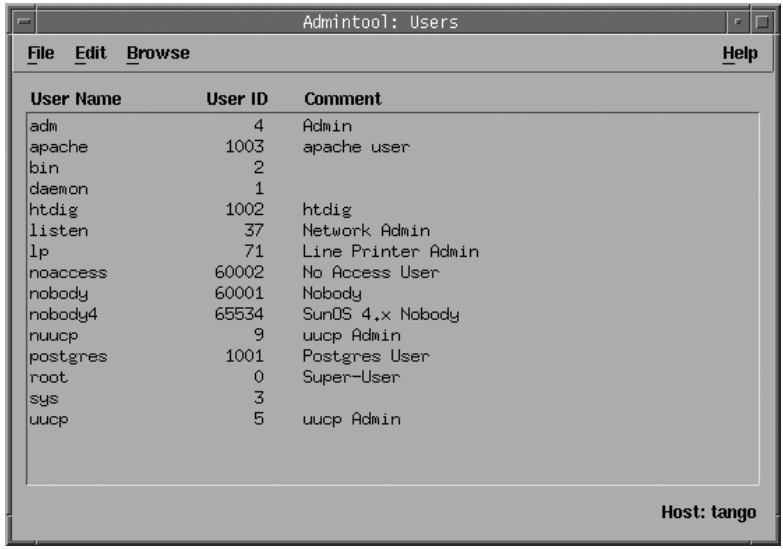

Рис. 6.1. *admintool позволяет добавлять, удалять и изменять записи для пользователей и групп*

Системные учетные записи

Существуют, по меньшей мере, две учетные записи, создаваемые при установке системы, которые в общем случае необходимы для работы Solaris: *root* и *nobody*. Они соответствуют наиболее и наименее привилегированным пользователям системы. Учетная запись *nobody* часто используется в работе для указания пользователя с минимальными правами на запись и выполнение программ: использование *nobody* означает, что нет необходимости создавать дополнительную учетную запись для каждой из низкопривилегированных операций, регулярно выполняемых в системе. Так, *nobody* может использоваться для ежедневного создания резервных копий на внешнем стримере – это единственное из устройств, на которое *nobody* имеет право записывать данные. Однако у *nobody* могут быть права чтения всех файлов системы, чтобы он имел возможность копировать данные, но не изменять их. Существуют и другие учетные записи, создаваемые по умолчанию, в частности *adm*, *sys*, *bin*, *lp* и *uucp*. Каждый из них служит одной цели: если вы никогда не используете UUCP, учетная запись *uucp* не нужна; если в системе нет принтера, можно также удалить *lp*.

Вторым специальным пользователем Solaris является пользователь *root*. *root* известен еще как суперпользователь или администратор и обладает полным контролем над всеми файлами, процессами, пользователями и устройствами системы. Пользователь *root* существует в каждой системе Solaris, и именно он отвечает за создание пользователей, определение групп, управление процессами и изменение настроек аппаратного обеспечения. Поскольку учетной записи *root* доступны многие действия в системе (в том числе и «прослушивание» содержимого пакетов, передаваемых по локальной сети), доступ к этой записи чаще всего и является целью взлома. Для ее защиты принимаются разнообразные меры. Возможно, например, разрешить доступ пользователя *root* только с консоли системы, соответствующим образом изменив файл */etc/default/login*. Это очень удобная мера безопасности, но если консоль по каким-либо причинам перестанет откликаться, невозможно будет получить удаленный доступ к системе с целью мягкой перезагрузки.

Учетная запись *root* не связана с определенным пользователем: она полностью самостоятельна и служит исключительно административным целям. Поскольку *root* имеет достаточные полномочия для перезаписи образа ядра и всех исполняемых файлов приложений, разумно пользоваться этой учетной записью только для добавления пользователей, определения групп и выполнения различных задач администрирования. В противном случае недружелюбное приложение типа «троянского коня» может перезаписать важные системные файлы таким же образом, как это делает распространенный троянский конь «ILOVEYOU», атакующий системы Microsoft Windows. В конечном итоге единственным средством защиты от троянских коней является

осторожное использование учетной записи root – и только в случаях, когда выполняемые операции гарантированно не могут нанести системе вреда.

Выбор паролей

Даже доступ к системе от лица обычного пользователя может дать злоумышленнику большие возможности. Если кто-либо взломает вашу учетную запись, он сможет изучить процессы системы, даже процессы, принадлежащие привилегированному пользователю вроде *root*. Он также сможет принудительно завершать процессы, созданные теми пользовательскими учетными записями, которые были взломаны.

Поэтому очень важно обезопасить систему Solaris, установив строгие правила для доступа и объяснив пользователям, какие пароли являются наиболее устойчивыми к попыткам взлома, предпринимаемым чужаками (а иногда и коллегами). В прежние времена записи паролей в Solaris хранились в зашифрованном виде в файле (*/etc/passwd*). Поскольку чтение файла было разрешено всем пользователям (в целях проверки подлинности и извлечения информации учетных записей), взлом единственной учетной записи даже непривилегированного пользователя давал злоумышленнику доступ к зашифрованным паролям для всей системы. Одно слабое звено в цепочке безопасности может привести к получению злоумышленниками информации из системы, а в худшем варианте – к удалению всех файлов.

Применение теневых паролей, хранимых в отдельной базе (*/etc/shadow*) в зашифрованном виде и доступных только привилегированным пользователям, позволило улучшить ситуацию с безопасностью. Теперь, чтобы получить привилегированный доступ, взломщик обязательно должен подобрать пароль пользователя *root* (в целом для получения root-доступа существует меньше возможностей). Тем не менее взлом обычных учетных записей по-прежнему позволяет злоумышленникам разрушить систему. Списки процессов часто содержат такую информацию, как имена пользователей и пароли из учетных записей баз данных; представьте, что конкурентам удалось получить доступ к конфиденциальным данным по всем вашим клиентам. Хуже того, они могут анонимно опубликовать факт взлома вашей системы во Всемирной паутине. Сколько клиентов у вас останется к концу дня?

В сетевой среде очень важно выбирать пароли, которые не могут быть легко подобраны с помощью программ взлома (таких как Crack). Метод прямого перебора паролей – не для слабонервных, но развитие технологий кластеризации (например, Full Moon) и распределенных вычислений (CORBA) означает, что взлом прямым перебором перестал быть неразумным вариантом, даже в случае очень больших словарей. Пароли, которые легко угадать – «solaris», «sun», «usa» или «newyork», – могут быть так же легко взломаны. Никогда не следует использовать для паролей личную информацию о себе, которая может

быть доступна злоумышленникам: день рождения, номер водительских прав, номер социального страхования и т. д.

Суперпользователь

Один из наиболее безопасных способов доступа к учетной записи root связан с командой *su*. Применение *su* избавляет вас от необходимости завершать сеанс работы от лица обычного пользователя и регистрироваться снова в качестве пользователя *root*, чтобы выполнить определенные действия по сопровождению системы либо установить или удалить программные пакеты. После ввода команды:

```
bash-2.03$ su root
```

будет предложено ввести пароль пользователя root. Если пароль принят, запускается новый командный интерпретатор, а приглашение меняется на символ #:

```
#
```

Чтобы интерпретатор команд унаследовал свойства из файла стартового файла суперпользователя (например, *.bashrc*), а не интерпретатора текущего пользователя, следует добавить ключ «-» в командной строке *su*:

```
su - root
```

Выполнив необходимые действия в качестве администратора, завершите работу с командным интерпретатором:

```
# exit
bash-2.03$
```

Происходит возврат к пользовательскому командному интерпретатору, из которого ранее и был запущен командный интерпретатор администратора. Командой *su* можно пользоваться для выполнения действий других пользователей, а не только root. Если, например, Майя работает с учетной записью *maya* и желает перезапустить веб-сервер Apache, она может воспользоваться *su* и временно стать пользователем *apache* без необходимости завершать текущий сеанс:

```
bash-2.03$ su apache
```

Набрав пароль, соответствующий пользователю apache, Майя должна попасть в новый командный интерпретатор и увидеть приглашение для пользователя *apache*. К примеру, если для учетной записи *apache* используется интерпретатор C shell (*csh*), приглашение командной строки может включать символ %:

```
apache%
```

Опять же, если Майя хочет, чтобы новый интерпретатор унаследовал свойства интерпретатора пользователя *apache*, а не ее собственного, она должна добавить ключ «-» в командную строку *su*:

```
su - apache
```

Другая полезная возможность команды *su* связана с ключом *-c*, который позволяет выполнить от имени указанного пользователя одну команду. Этот ключ часто применяется для выполнения команд с привилегиями конкретных пользователей при запуске системы, а не в целях администрирования. Допустим, необходимо создать сценарий, запускающий веб-сервер Apache при загрузке системы; мы должны выполнить команду */usr/local/bin/apache-1.3.6/apachectl* от имени пользователя apache. Следовательно, к файлу запуска Apache в каталоге */etc* нужно добавить такую строку:

```
su - apache -c "/usr/local/bin/apache-1.3.6/apachectl start"
```

Для выполнения такой команды (если она выполняется не суперпользователем) потребуется пароль.

Кроме того, существует очень полезный пакет для чувствительных к безопасности, он называется *sudo* и позволяет контролировать использование команды su: действия любого пользователя, применяющего su, отслеживаются и регистрируются в специальном журнале. Существует возможность ограничить действия, выполняемые по команде *su* для отдельных пользователей. Допустим, необходимо дать пользователю maya права на создание записей в файле паролей (*/etc/passwd*) посредством *su*, но запретить делать то же пользователю *pwatters*; для этой цели замечательно подойдет *sudo*. В отличие от команды *su*, *sudo* защищает пароль пользователя root; выполнив команду *sudo*, пользователь должен набрать свой собственный пароль, а не пароль администратора. Таким образом, системный администратор может делегировать часть полномочий другим пользователям, занимаясь сопровождением системы в целом.

Удаление пользователей

Часто бывает необходимо удалить пользовательскую учетную запись из системы. Причины могут быть разными, включая изменения в организации, уход с должности, возросшие потребности в ресурсах. Существует ряд шагов, позволяющих убедиться, что в системе не осталось следов присутствия пользователей, чьи учетные записи были удалены:

- Удаление домашнего каталога пользователя (например, *rm -fr /home/pwatters*)
- Удаление почтового ящика пользователя (*rm -fr /var/mail/pwatters*)
- Удаление записей из файлов */etc/passwd* и */etc/shadow*

- Удаление почтовых псевдонимов, определенных для пользователя в файле */etc/aliases*
- Проверка того, что в системе не осталось фоновых процессов пользователя (*ps – eaf | grep имя_пользователя*)
- Проверка того, что пользователем не запланировано автоматическое выполнение задач (таких как отправка пароля по почте ровно в полночь!) с помощью *cron* или *at*

Если предполагается, что файлы пользователя могут понадобиться в будущем (допустим, пользователь участвовал в разработке продукта), следует сделать резервную копию перед удалением. Чтобы временно заблокировать доступ к определенной учетной записи, можно просто заменить зашифрованный пароль пользователя (в */etc/passwd* или */etc/shadow*) одним символом звездочки «*». Пользователь не сможет войти в систему, но его файлы останутся на месте, а все запланированные задачи будут выполнены.

Если вы предпочитаете выполнять сопровождение пользователей с помощью графического интерфейса, удалить учетные записи пользователей можно инструментом *admintool*.

Файл групп

Часто бывает полезно классифицировать или разбивать пользователей на категории на основе каких-то общих характеристик. В Solaris существует деление на группы. Группа – это набор пользователей с общими правами доступа к файлам системы. Так, в системе Solaris, обслуживающей университет, преподаватели могут входить в группу *staff*, а студенты – в группу *student*. Как мы позже увидим, члены группы *staff* могут делиться правами на чтение, изменение или выполнение своих файлов с другими членами этой группы, запрещая при этом доступ членам группы *student*. Каждый пользователь системы принадлежит к первичной группе, которая отражает определенную категорию лиц в организации, такую как *staff* (персонал) и *student* (студенты). При этом каждый пользователь может входить в одну или несколько дополнительных групп. Так, члены группы *staff* могут входить в группы *arts*, *math* и *science*. В результате, члены группы *arts* могут делиться правами на чтение, изменение или выполнение своих файлов с другими членами группы, запрещая доступ членам групп *math* и *science* (несмотря на то что пользователи *math*, *science* и *arts* также являются членами *staff*). Следующий уровень гибкости: компоновать группу *arts* из пользователей, которые принадлежат к первичным группам *staff* и *student*. Число групп, в которые может входить пользователь, не ограничено.

Вот пример файла групп:

```
bash-2.03$ cat /etc/group
root::0:root
```

```
wheel::1:
bin::2:root,bin,daemon
sys::3:root,bin,sys,adm
adm::4:root,adm,daemon
uucp::5:root,uucp
mail::6:root
tty::7:root,tty,adm
lp::8:root,lp,adm
users::10:paul,maya
daemon::12:root,daemon
nobody::60001:
noaccess::60002:
nogroup::65534:
```

На рис. 6.2 показаны группы, создаваемые при установке системы (*admintool*). Каждая группа имеет имя и необязательный пароль для группового доступа к файлам. Каждой группе присваивается уникальный идентификатор (*gid*), члены группы перечислены в списке, элементы которого разделены запятыми. Отдельные группы, такие как *bin*, используются для общегруппового выполнения системных служб любым пользователем из числа привилегированных или полупривилегированных (в частности, *bin*, *daemon* и *root*).

Группы могут удаляться из системы вручную – удалением соответствующей записи из файла */etc/group*.

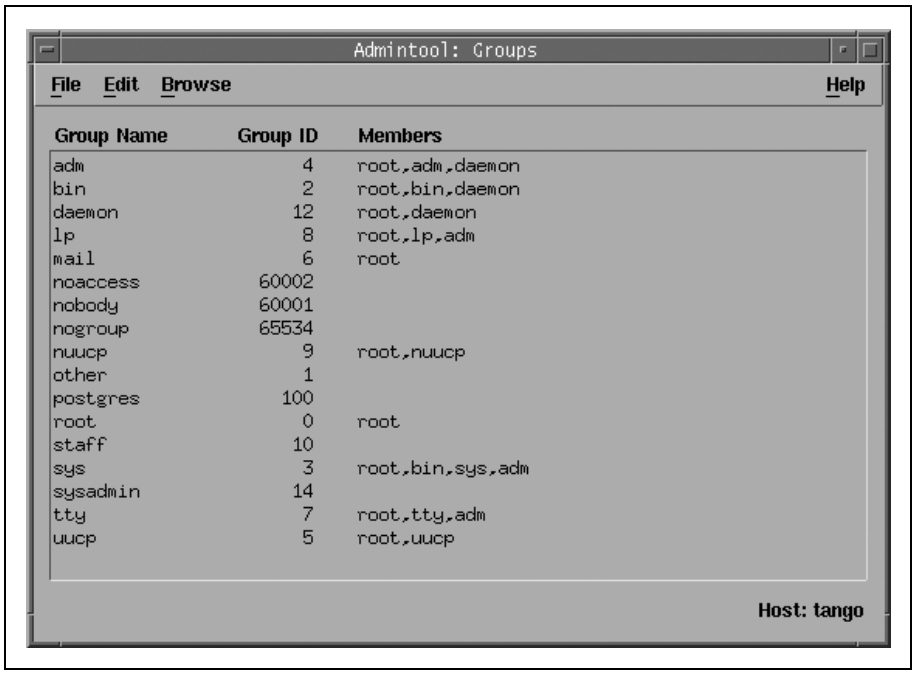

Рис. 6.2. *База данных групп (/etc/group) в изложении инструмента admintool*

Управление программными пакетами

Читатели, у которых уже есть некоторый опыт администрирования Unix- или Linux-систем и которым приходилось устанавливать многочисленные программные пакеты, знают, что через какое-то время становится довольно сложно уследить за всеми установленными файлами. Кроме того, после нескольких обновлений пакета на диске может образовываться несколько каталогов, содержащих, по сути, одну и ту же программу, но занимающих ценное дисковое пространство.

Например, в случае активно работающего веб-сервера довольно часто можно видеть несколько версий Apache, установленных в одном сегменте файловой системы: */usr/local/apache-1.3.1*, */usr/local/apache-1.3.2*, */usr/local/apache-1.3.6* и т. д. Можно очень легко перепутать зависимости между этим экземплярами, если (допустим) каталог *htdocs* символическими ссылками отображается из */usr/local/apache-1.3.1* в */usr/local/apache-1.3.2* и */usr/local/apache-1.3.6*. Удаление содержимого каталога */usr/local/apache-1.3.1* приведет к исчезновению каталога *htdocs* из */usr/local/apache-1.3.2* и */usr/ local/apache-1.3.6*. Разумеется, я не хочу сказать, что символические ссылки вредны, но управление программными пакетами в производственной среде требует дисциплинированного подхода.

Одним из способов достижения цели является управление программным обеспечением с помощью инструментов администрирования пакетов из состава Solaris. Эти программы могут использоваться для создания, изменения и удаления содержимого отдельных архивов программных пакетов. Текстовый файл пакета может содержать произвольное число исходных файлов и готовых приложений и может быть использован для установки ПО. После завершения установки и настройки производственной системы можно создать новый пакет, содержащий все установленные файлы, и таким образом сохранить их расположение в рабочем каталоге. В результате испорченное программное обеспечение может быть с легкостью восстановлено единственной командой, а распространение и установка всех пакетов сводится к созданию одного-единственного и последующего распространения его на всех системах. Без средств администрирования пакетов установка ПО в различные каталоги (*/usr/local/bin*, */usr/local/man*, */usr/local/lib* и т. д.) может быстро привести к тому, что невозможно или очень затруднительно будет понять, какие из файлов принадлежат конкретным пакетам.

Все программное обеспечение, распространяемое Sun в качестве части Solaris, выпускается в формате пакетов (сюда входят все стандартные командные интерпретаторы и наборы команд). Определить принадлежность файла к определенному пакету можно с помощью команды *pkgchk*:

```
bash-2.03$ /usr/sbin/pkgchk -l -p /usr/bin/ls
Pathname: /usr/bin/ls
```

```
Type: regular file
Expected mode: 0555
Expected owner: bin
Expected group: bin
Expected file size (bytes): 18120
Expected sum(1) of contents: 53113
Expected last modification: Oct 06 05:43:05 PM 1998
Referenced by the following packages:
        SUNWcsu
Current status: installed
```

Можно видеть, что команда */usr/bin/ls* принадлежит пакету *SUNWcsu*, ее владельцем является пользователь *bin* (из группы *bin*), права доступа для файла установлены в 0555, а его размер составляет 18 120 байт. Не забывайте, что это *ожидаемые* значения; если действительные атрибуты установленного файла отличаются, это может означать, что были внесены определенные изменения, отражающие потребности системы (например, изменился владелец), либо (если изменился размер файла) возможно заражение троянским вирусом. Для базовых пакетов, таких как *SUNWcsu*, сравнение действительных и ожидаемых значений этих атрибутов должно производиться регулярно в процессе проверки безопасности.

Чаще всего применяются следующие команды управления пакетами:

pkgadd

Добавляет пакет к файлам целевой системы.

pkgchk

Проверяет корректность пакета, хранимого в архиве или установленного в системе.

pkginfo

Отображает информацию о содержимом пакета.

pkgmk

Создает новый пакет на основе спецификаций, содержащихся в файле прототипа.

pkgproto

Позволяет создавать новые спецификации пакетов, которые будут размещены в файле прототипа.

pkgrm

Удаляет установленный пакет из целевой системы.

pkgtrans

Упаковывает все файлы пакета в один архив.

Все эти команды предназначены для работы с отдельными пакетами. Кроме того, в качестве интерфейса к этим командам можно воспользоваться инструментом *admintool*. Установка пакетов с его помощью да-

ет ряд преимуществ, включая простоту применения и возможность выполнять различные операции. С помощью *admintool* можно:

- Определять текущий уровень работы системы (*runlevel*)
- Определять, установлен ли определенный пакет
- Обнаруживать частично установленные либо испорченные программы
- Определять, приведет ли установка нового пакета к перезаписи существующих файлов
- Выполнять установку пакетов
- Разрешать выполнение сценариев *setuid*/*setgid*
- Разрешать установку файлов *setuid*/*setgid*
- Читать тексты, связанные с правообладанием
- Проверять факт выполнения подготовительных задач установки
- Проверять наличие достаточного для успешной установки пакета объема свободного дискового пространства

Файлы пакетов являются текстовыми, и это означает, что их содержимое можно изучать в редакторе или аналогичной программе просмотра текста. Например, загрузив пакет веб-сервера Apache, мы можем проверить его происхождение, изучив заголовки:

```
bash-2.03$ head apache-1.3.12-sol8-sparc-local
# PaCkAgE DaTaStReAm
SMCapache 1 5392
# end of header
NAME=apache
ARCH=sparc
VERSION=1.3.12
CATEGORY=application
VENDOR=Apache Group
EMAIL=steve@smc.vnet.net
```

На рис. 6.3 показан интерфейс *admintool*, применяемый для добавления пакетов.

Мы видим, что данный архив Apache 1.3.12 для платформы Solaris был создан *steve@smc.vnet.net* (Стивом Кристиансеном, заведующим сайтом *www.sunfreeware.com*). За заголовком следуют собственно файлы архива, упакованные в последовательный поток командой *pkgmk*.

В следующих разделах я расскажу, как изучать установленные в системе пакеты, а также добавлять и удалять пакеты.

Изучение установленных пакетов

Для перечисления всех установленных в системе пакетов применяется команда *pkginfo*:

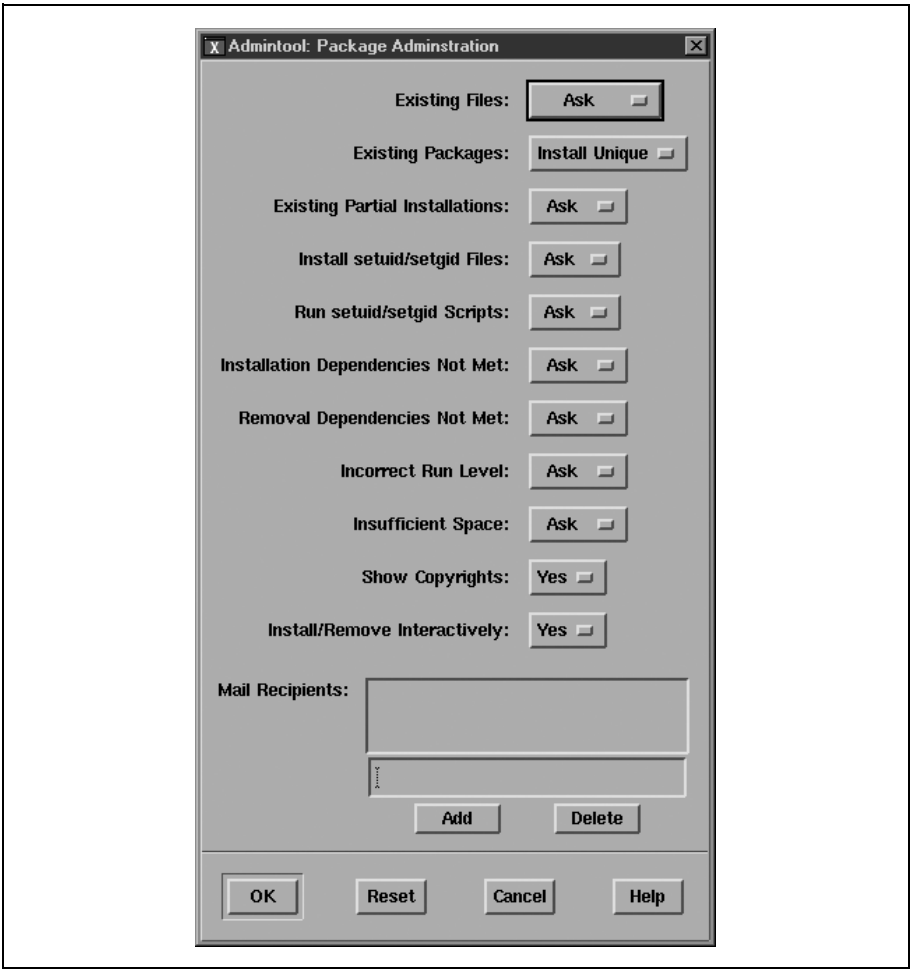

Рис. 6.3. *Интерфейс добавления пакетов инструмента admintool*

```
bash-2.03# pkginfo
application GNUbash        bash
application SMCgcc         gcc
application SMCmake        make
application SMCsamba       samba
ALE         SUNW5xmft      Chinese/Taiwan BIG5 X Windows Platform minimum
                           required Fonts Package
system      SUNWab2m       Solaris Documentation Server Lookup
system      SUNWadmap      System administration applications
system      SUNWadmc       System administration core libraries
system      SUNWadmfw      System & Network Administration Framework
system      SUNWadmr       System & Network Administration Root
system      SUNWarc        Archive Libraries
```

```
system      SUNWarrf       X11 Arabic required fonts
system      SUNWatfsr      AutoFS, (Root)
system      SUNWatfsu      AutoFS, (Usr)
system      SUNWaudio      Audio applications
system      SUNWbcp        SunOS 4.x Binary Compatibility
system      SUNWbtool      CCS tools bundled with SunOS
system      SUNWcar        Core Architecture, (Root)
system      SUNWcg6        GX (cg6) Device Driver
system      SUNWcg6h       GX (cg6) Header Files
ALE         SUNWciu8       Simplified Chinese iconv modules for UTF-8
```

Здесь уже установлен ряд системных и прикладных пакетов, включая *SUNWadmap* (приложения системного администрирования) и *bash*, Bourne-подобный командный интерпретатор (*GNUbash*).

Установка пакетов

Добавление пакетов производится командой *pkgadd*. Продолжая пример с файлом Apache, мы воспользовались бы следующей командой для установки пакета:

```
bash-2.03# /usr/sbin/pkgadd -d apache-1.3.6-sol7-sparc-local
```

После обработки заголовка отображается сообщение:

```
The following packages are available:
(Доступны следующие пакеты:)
  1  SMCapache     apache
                   (sparc) 1.3.12

Select package(s) you wish to process (or 'all' to process
all packages). (default: all) [?,??,q]:
(Выберите пакеты для установки (либо 'all', чтобы установить все
пакеты). (по умолчанию: all)
```

Ввод цифры **1** приведет к установке пакета *SMCapache*:

```
Processing package instance <SMCapache> from </tmp/apache-1.3.12-sol8-sparc-
local>

apache
(sparc) 1.3.12
Apache Group
Using </usr/local> as the package base directory.
## Processing package information.
## Processing system information.
   1 package pathname is already properly installed.
## Verifying disk space requirements.
## Checking for conflicts with packages already installed.
## Checking for setuid/setgid programs.

Installing apache as <SMCapache>
```

```
## Installing part 1 of 1.
/usr/local/apache/bin/ab
/usr/local/apache/bin/apachectl
...
/usr/local/doc/apache/README.NT
/usr/local/doc/apache/README.configure
/usr/local/doc/apache/WARNING-NT.TXT
[ verifying class <none> ]

Installation of <SMCapache> was successful.
```

Разумеется, это идеализированный пример. На практике *pkgadd* часто запрашивает подтверждения различных параметров и/или изменений, которые должны быть внесены для корректной установки пакета. Например, если каталог уже существут и необходимо изменить атрибут, будет отображено такое сообщение:

```
## Checking for conflicts with packages already installed.
(Выявление конфликтов с уже установленными пакетами.)
The following files are already installed on the system and are being
used by another package:
(Следующие файлы уже существуют в системе и используются другим
пакетом.)
* /usr/local/apache <attribute change only>
                        (Только изменение атрибута.)
* - conflict with a file which does not belong to any package.
(Конфликт с файлом, который не принадлежит какому-либо пакету.)

Do you want to install these conflicting files [y,n,?,q]
(Установить конфликтующие файлы?)
```

Ввод **y** приведет к запрошенному изменению атрибута, и установка будет продолжена.

Обработав системную информацию и данные пакета, проверив наличие достаточного для установки объема дискового пространства, команда *pkgadd* копирует все файлы из архива в локальную файловую систему.

Удаление пакетов

Пакеты могут удаляться из системы в любой момент с помощью команды *pkgrm*. Перед удалением пакета следует убедиться, используя команды *ps* и *lsof*, что никакой из работающих процессов не использует файлы пакета. Чтобы удалить пакет *SMCapache*, установленный ранее, воспользуйтесь командой:

```
bash-2.03# pkgrm SMCapache

The following package is currently installed:
   SMCapache       apache
                   (sparc) 1.3.12

Do you want to remove this package? y
```

```
## Removing installed package instance <SMCapache>
## Verifying package dependencies.
## Removing pathnames in class <none>
/usr/local/doc/apache/WARNING-NT.TXT
/usr/local/doc/apache/README.configure
/usr/local/doc/apache/README.NT
/usr/local/doc/apache/README
/usr/local/doc/apache/Makefile.tmpl
...
/usr/local/apache/bin/apachectl
/usr/local/apache/bin/ab
/usr/local/apache/bin
/usr/local/apache <non-empty directory not removed>
## Updating system information.

Removal of <SMCapache> was successful.
```

Создание пакетов

Создание пакетов – задача не столь тривиальная, как их установка или удаление. Необходимо создать спецификацию для архива (файл прототипа) с помощью команды *pkgproto*, а затем собрать собственно пакет с помощью команды *pkgmk*. Изучая заголовок файла пакета Apache, мы уже встречались с некоторыми данными, которые должны быть указаны для создания пакета, в частности с названием разработчика и адресом электронной почты автора пакета. Эти данные должны быть включены в файл *pkginfo*. Вот перечень параметров, наиболее часто встречающихся в файле *pkginfo*:

ARCH

Архитектура (*sparc* или *intel*), для которой предназначен пакет.

BASEDIR

Целевой каталог, в который происходит распаковка файлов.

CATEGORY

Тип архива (системный или приложение).

EMAIL

Контактная информация автора пакета.

NAME

Имя пакета.

PSTAMP

Имя автора пакета.

VENDOR

Имя или название разработчика программы.

VERSION

Версия программы, подлежащей упаковке.

После создания файлов *pkginfo* и прототипа сборка пакета элементарно производится с помощью команды *pkgmk*. В следующем примере мы создадим архив сервера приложений Borland версии 4.5 (*http://www.borland.com*). Как правило, этот шаг выполняется в производственной среде после того, как созданы и локализованы все XML-файлы настройки, а работа службы протестирована. Создание пакета на этом этапе позволяет производить восстановление на определенном уровне в случае, если (например) разрушается жесткий диск либо необходима установка резервного сервера, чтобы справиться с наплывом пользователей.

Файл *pkginfo* должен содержать все необходимые параметры (такие как ARCH, BASEDIR и CATEGORY), описанные выше. Для сервера приложений Borland эти элементы будут выглядеть следующим образом:

```
PKG="BAS_LOCAL"
NAME="Borland App Server"
ARCH="sparc"
VERSION="4.5"
CATEGORY="application"
VENDOR="Borland"
EMAIL="support@borland.com"
PSTAMP="Paul Watters"
BASEDIR="/usr/local/inprise/bas45"
CLASSES="none"
```

Мы можем интерпретировать их так:

- Имя пакета: BAS_LOCAL (указывает на локальную установку BAS)
- Название пакета: Borland Application Server
- Целевая архитектура: SPARC
- Номер версии: 4.5
- Категория BAS: приложение
- Разработчик BAS: Borland
- Электронный адрес разработчика: *support@borland.com*
- Имя автора пакета совпадает с именем автора этой книги
- Каталог установки: /usr/local/inprise/bas45

После создания *pkginfo* необходимо воспользоваться следующей командой для создания файла прототипа:

```
bash-2.03# touch prototype
```

Теперь мы должны указать расположение файла *pkginfo*, добавив в файл прототипа строку:

```
i pkginfo=/usr/local/borland/bas45/pkginfo
```

В каталоге установки (указанном в файле *pkginfo*) выполним следующую команду для создания списка относительных путей и имен файлов и передачи его команде *pkgproto*. Вывод добавляется к файлу прототипа:

```
bash-2.03# find . -print | pkgproto >> prototype
```

Взглянем на содержимое только что созданного файла прототипа:

```
bash-2.03# head /usr/local/borland/bas45/prototype
i pkginfo=/usr/local/borland/bas45/pkginfo
f none install.idb 0644 bas borland
d none bin 0775 bas borland
f none bin/JdsExplorer 0775 bas borland
f none bin/JdsExplorer.config 0775 bas borland
f none bin/JdsServer 0775 bas borland
f none bin/JdsServer.config 0775 bas borland
f none bin/jdbce 0775 bas borland
f none bin/jdbce.config 0775 bas borland
f none bin/jsql 0775 bas borland
f none bin/jsql.config 0775 bas borland
```

Каждый файл (f) и каталог (d) в пределах базового представлен уникальной записью в файле прототипа. Каждая запись включает информацию о пользователе и группе, владеющих файлом, а также установленные для файла права доступа (в восьмеричном формате).

Чтобы изменить некоторые из свойств, определенных в файле прототипа, можно воспользоваться функциями поиска и замены редактора vi перед выполнением команды *pkgmk*. Предположим, необходимо поменять владеющую группу для файлов с *borland* на *inprise*; воспользуемся командой подстановки:

```
:%s/borland/inprise/g
```

Если файл прототипа соответствует вашим потребностям, создайте пакет посредством следующей команды:

```
bash-2.03# pkgmk -o -r /usr/local/borland/bas45
## Building pkgmap from package prototype file.
## Processing pkginfo file.
## Attempting to volumize 986 entries in pkgmap.
part  1 -- 8997 blocks, 986 entries
## Packaging one part.
/var/spool/pkg/BAS_LOCAL/install.idb
/var/spool/pkg/BAS_LOCAL/bin/JdsExplorer
/var/spool/pkg/BAS_LOCAL/bin/JdsExplorer.config
/var/spool/pkg/BAS_LOCAL/bin/JdsServer
/var/spool/pkg/BAS_LOCAL/bin/JdsServer.config
/var/spool/pkg/BAS_LOCAL/bin/jdbce
/var/spool/pkg/BAS_LOCAL/bin/jdbce.config
/var/spool/pkg/BAS_LOCAL/bin/jsql
```

```
/var/spool/pkg/BAS_LOCAL/bin/jsql.config
...
```

На этом этапе файлы для упаковки переносятся в каталог */var/spool/pkg/BAS_LOCAL*. Теперь нужно просто воспользоваться командой *pkgtrans*, чтобы упаковать файлы в единственный архивный файл:

```
bash-2.03# pkgtrans -s /var/spool/pkg /backup/BAS_LOCAL.pkg

The following packages are available:
  1  BAS_LOCAL    Borland Application Server
                  (sparc) 4.5

Select package(s) you wish to process (or 'all' to process
all packages). (default: all) [?,??,q]:
```

После выбора пункта **1** будет создан пакет */backup/BAS_LOCAL.pkg*.

Управление принтерами

В Solaris существует ряд инструментов командной строки и графических интерфейсов строчных принтеров (*lp*, line printer), предназначенных для поддержи служб печати и вывода как ASCII-текста, так и данных в формате Adobe PostScript. Принтеры представлены в системе файлами устройств, точно так же, как другие аппаратные составляющие. Технически принтеры представляются как терминалы, а следовательно, данные могут перенаправляться или передаваться на терминалы печати через конвейеры стандартными средствами командных интерпретаторов. Например, мы можем запланировать на полночь печать объема свободного пространства файловых систем с помощью демона *cron*, воспользовавшись такой командой:

```
df -k | lp
```

Подобная гибкость позволяет с легкостью включать вывод на печать в список регулярных задач планировщика (такого как *cron*).

Печать файлов

Печать файлов в Solaris очень проста: достаточно использовать команду *lp* (в стиле систем System V). Чтобы вывести на печать текстовый файл */home/paul/report.txt*, выполним команду:

```
bash-2.03$ lp -d hp1 /home/paul/report.txt
```

Ключ *-d* позволяет указать имя принтера, которому передается задание. Процесс печати начинается открытием и чтением файла до его записи в буфер (spool) печати, в которой задания накапливаются в FILO-очереди печати (First In, Last Out; задания, поступившие раньше, от-

правляются на печать позже). Демон печати Solaris (*lpd*) отвечает за обработку очереди печати и передачу данных на терминал принтера.

Для вывода на печать файла в формате PostScript применяется аналогичная команда:

```
bash-2.03$ lp -d hp1 /home/paul/report.ps
```

Перечень наиболее часто применяемых ключей *lp* приведен в табл. 6.1.

Таблица 6.1. Ключи команды lp

Ключ	Описание
-c	Немедленно помещает содержимое файла в буферную зону; изменения, вносимые после выполнения операции, не будут отражены в выводе
-d	Позволяет указать имя принтера, которому предназначается задание
-m	Создает почтовое сообщение, информирующее автора задания об успешном завершении печати
-p	Печать страниц по списку
-q	Печать задания с определенным приоритетом
-t	Строка, отображаемая на титульной странице
-w	Предписывает информировать автора задания по завершении печати (с помощью команды *write*)

Если принтер *hp1* был подключен к другой системе, можно уточнить имя системы, чтобы избежать случайной печати задания на локальном принтере с таким же именем:

```
bash-2.03$ lpr -P emu:hp1 report.ps
```

Эта команда приведет к печати PostScript-файла *report.ps* на принтере *hp1* системы *emu*.

Настройка принтеров

Откуда демон печати знает, какому принтеру передавать задание? Ответ на этот вопрос связан с комбинацией файлов */etc/printers.conf* и */etc/nsswitch.conf*. Первый отвечает за определение свойств каждой из очередей печати, доступных на локальной системе, а второй определяет метод разрешения имен принтеров.

Если имена узлов с подключенными принтерами определены в локальном файле */etc/hosts*, в файл */etc/nsswitch.conf* следует добавить такую запись:

```
printers: files
```

Если в качестве локальной службы имен используется NIS, запись должна выглядеть так:

```
printers: nis files
```

Для пользователей DNS запись будет следующей:

```
printers: dns files
```

И наконец, для пользователей NIS+:

```
printers: nisplus nis files
```

После выбора службы имен следует присвоить переменной среды LPDEST имя принтера по умолчанию (для каждого пользователя):

```
bash-2.03$ LPDEST=hp1; export LPDEST
```

Файл */etc/printers.conf* определяет непосредственно свойства каждого принтера, к которому есть доступ с локальной системы. Файл */etc/printers.conf* содержит записи следующего вида:

```
bash-2.03$ cat /etc/printers.conf
hp1:\
        :bsdaddr=server1,hp1,Solaris:\
        :description=HP Primary:
hp2:\
        :bsdaddr=server2,hp2,Solaris:\
        :description=HP Secondary:
_default:\
        :use=hp1:
```

В данном случае файл содержит информацию о двух серверах печати, с каждым из которых связан один принтер Hewlett Packard (принтер *hp1* подключен к серверу *server1*, *hp2* – к серверу *server2*).

Состояние принтера

Чтобы определить состояние принтера, можно воспользоваться командой *lpstat*:

```
bash-2.03$ lpstat -D -p hp1
printer hp1 is idle. enabled since Dec 21 13:45 2000. available.
```

Если бы принтер *hp1* был занят выполнением заданий, мы увидели бы номер текущего задания, а также дату и время его поступления:

```
bash-2.03$ lpstat -D -p hp1
hp1-1232  pwatters  1232 Dec 21 15:45 2000 on hp1
```

Иногда отдельные задания могут препятствовать обработке всей очереди печати, поэтому рекомендуется время от времени проверять состояние очереди и отменять задания, которые не обрабатываются с должной скоростью. (Это может происходить, к примеру, из-за некоррект-

ных PostScript-данных.) Команда *cancel* позволяет удалить задание. Чтобы удалить задание 1232 принтера *hp1*, воспользуйтесь командой:

```
bash-2.03# cancel hp1-1232
request "hp1-1232" canceled
```

Команда *lpstat* должна подтвердить отсутствие заданий в очереди:

```
bash-2.03$ lpstat -D -p hp1
printer hp1 is idle. enabled since Dec 21 13:45 2000. available.
```

Поддержка устройств

Чтобы убедиться, что конкретный принтер будет работать с Solaris, обратитесь к перечню совместимого аппаратного обеспечения (Hardware Compatibility List, HCL) по адресу *http://docs.sun.com/*. Как вариант можно изучить списки терминалов в базе данных *terminfo*. Например, чтобы выяснить, какие из лазерных принтеров Hewlett Packard поддерживаются, мы воспользуемся такой командой:

```
bash-2.03# ls -l /usr/share/lib/terminfo/hplaser*
total 418
-rw-r--r--   2 bin      bin         1560 Sep  1  1998 hplaser
-rw-r--r--   1 bin      bin         1387 Sep  1  1998 hplaser+basic
-rw-r--r--   1 bin      bin         1065 Sep  1  1998 hplaser+high
-rw-r--r--   2 bin      bin         1560 Sep  1  1998 hplaserjet
```

Очевидно, Solaris поддерживает многие модели принтеров HP, включая принтеры Laser Jet (*hplaser*).

Установка, графический интерфейс

Наиболее простой способ установки и управления принтерами в Solaris связан с использованием интерфейса печати инструмента *admintool*. Интерфейс позволяет устанавливать различные параметры работы принтеров, выбирая значения из корректных диапазонов. На рис. 6.4 представлен пример интерфейса для добавления локального принтера (для удаленных принтеров используется другой интерфейс, с доступным полем Print Server (сервер печати).

Администратор может изменять следующие параметры:

Имя принтера

Описание принтера

Порт принтера

Тип принтера

Поддерживаемые типы данных

Метод уведомления об ошибках

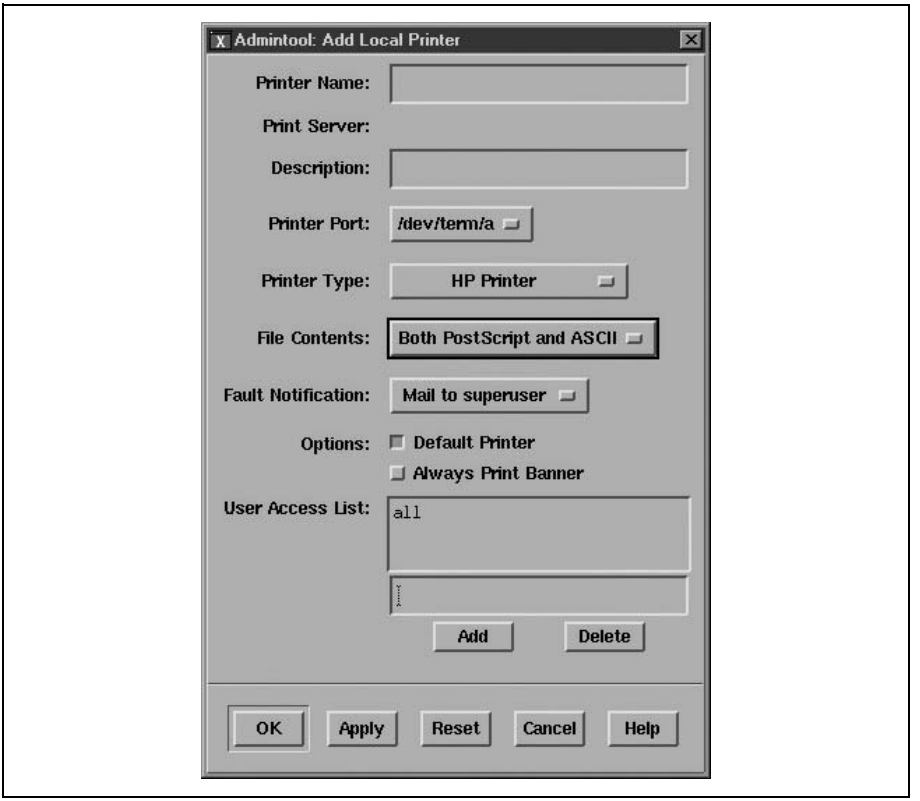

Рис. 6.4. *Установка принтера в admintool*

Стандартные параметры

Список доступа пользователей

Помимо установки новых принтеров, *admintool* может использоваться для изменения настроек принтеров, уже существующих в системе.

Квоты

Система Solaris может быть настолько открытой или закрытой, насколько необходимо. Это касается не только разграничения прав и управления внешним доступом, но также ограничений на объем дискового пространства, доступного отдельным пользователям. Подобные ограничения часто диктуются необходимостью, но могут определяться и политикой безопасности. Например, программисту на языке C может потребоваться больше дискового пространства, чем обычному пользователю почты; квоты для каждой учетной записи должны отражать это обстоятельство. При этом возможные DOS-атаки, связанные

с переполнением почтового ящика пользователя до степени, когда запись данных в разделе */var* становится невозможной, могут быть предотвращены установкой разумных, не связанных с конкретными пользователями, квот для файловой системы */var*.

Квоты могут устанавливаться как для пользователей, так и для файловых систем. Таким образом, пользователь может иметь неограниченные права записи в пределах одной файловой системы (например, зоны подкачки), но ограниченные права на создание файлов в другой. Квоты также могут быть «мягкими» или «жесткими»: мягкие квоты приводят к выдаче пользователям предупреждений до того, как ограничения будут превышены, жесткие квоты определяют абсолютные значения ограничений в байтах[1] и не позволяют записывать на диск данные сверх заданного ограничения.

Включение квот для файловых систем производится на этапе монтирования. Необходимо указать соответствующий параметр в файле */etc/vfstab* либо в командной строке, если используется команда *mount*. Предположим, необходимо установить квоту на файловую систему */var*, ограничив таким образом размер почтовых ящиков. Следует добавить опцию *rq* к записи для системы */var* в файле */etc/vfstab*. Кроме того, следует размонтировать и заново смонтировать файловую систему, чтобы изменения вступили в силу. Затем следует создать файл *quotas* в каталоге */var*; правами на чтение и изменение файла должен обладать только пользователь *root*. Наконец, надо воспользоваться командой *edquota* и ввести для каждого пользователя следующую информацию:

- Число выделяемых inode-узлов, то есть файлов (мягкое ограничение)
- Число выделяемых блоков дискового пространства (мягкое ограничение)
- Число выделяемых inode-узлов (жесткое ограничение)
- Число выделяемых блоков дискового пространства (жесткое ограничение)

После определения квот для пользователей можно воспользоваться командой *quotaon* для включения квот файловой системы.

Установка Sendmail

Одной из самых распространенных служб сети Интернет является электронная почта (известная под названием *email*). Жизнь электронной почты началась с примитивной, ограниченной одним узлом службы, позволявшей пользователям посылать друг другу сообщения. Эта служба существовала на первых мэйнфреймах и компьютерных системах среднего класса, работавших под управлением Unix и VAX/VMS.

[1] На самом деле в килобайтах. – *Примеч. науч. ред.*

В наши дни низкоуровневый сервер Solaris на базе процессора Pentium может управляться с электронной почтой для целой организации, заведуя обменом почтой между локальными узлами и со всей сетью Интернет. Фактически сейчас на базе электронной почты строятся целые организационные и коммуникационные сети; существуют списки рассылки, позволяющие группам единомышленников получать сообщения, отправленные одним человеком, причем автор сообщения посылает его всего один раз. Электронная почта выросла из ограничений текстового формата: текстовое кодирование мультимедиа-сообщений стало возможным с развитием протокола MIME, предложенного документом RFC 2045. Таким образом, документы текстовых процессоров, изображения, электронные таблицы и электронные презентации могут «прикрепляться» к текстовым сообщениям, а затем извлекаться получателем.

Архитектурные решения электронной почты базируются на настройке трех видов взаимодействующих агентов: агента передачи сообщений (Mail Transfer Agent, MTA), агента доставки сообщений (Mail Delivery Agent, MDA) и пользовательского почтового агента (Mail User Agent, MUA). Пользователь отправляет сообщение электронной почты из агента MUA, через локальный MTA, который связывается со внешним MTA. Внешний MDA получает сообщение по запросу удаленного MUA-агента. Типичным MTA является *sendmail*, который применяется на всех Unix-системах и реализует простой протокол передачи почты (Simple Mail Transfer Protocol, SMTP). *sendmail*, как правило, работает в паре с локальным MDA, таким как *procmail*, а пользователи локальной системы читают почту в приложениях *elm*, *pine* («*pine* is not *elm*») либо *mailx*. В этом случае агент MDA просто извлекает сообщения из каталога хранения почты файловой системы */var*. Если за получение почты отвечает выделенный сервер, пользователи могут применять MUA-агенты, реализующие поддержку протокола почтовой службы (Post Office Protocol, POP) либо интернет-протокола доступа к сообщениям (Internet Message Access Protocol, IMAP). Что касается Solaris, наиболее популярным MUA с функциями доступа ко внешним серверам является почтовый клиент Netscape. Solaris-системы могут применяться (и применяются) для поддержки работы MUA-агентов, функционирующих на других платформах, таких как Microsoft Windows и Macintosh, либо на платформах Unix (в том числе Solaris).

В этом разделе мы рассмотрим настройки и установку агентов MTA, MDA и MUA, а также рассмотрим принципы работы SMTP. Будет рассказано об установке последней версии *sendmail* (*sendmail* V8.11.2), поскольку заплаты безопасности и обновления *sendmail* выходят довольно часто, и любой дистрибутив из комплекта поставки системы может оказаться устаревшим. Например, переполнение буферов в демонах Solaris (таких как *sendmail*) является популярным среди злоумышленников методом и часто используется для получения несанкционированного доступа к системе. В *sendmail* за многие годы было

найдено довольно много подобных «дыр», поэтому имеет смысл всегда устанавливать самую последнюю версию системы.

Простой протокол передачи почты

Давным-давно электронная почта не выходила за пределы одного узла: пользователи, у которых были учетные записи на локальных многопользовательских системах или системах с разделением времени, могли обмениваться сообщениями. Как только сети вторглись в рабочее пространство, почтовые системы, основанные на инструментах вроде UUCP (Unix-to-Unix Copy Program) и закрытых протоколах вроде X.400, лишили сетевых пользователей, не говоривших на конкретных почтовых «диалектах», возможности посылать сообщения друг другу. Помимо этого, с диагностированием проблем передачи почты часто возникали сложности, поскольку для этой цели не существовало стандартных вспомогательных инструментов или протоколов.

Эта неприемлемая ситуация привела к созданию документа RFC 821, в котором был описан новый простой способ обмена почтой для агентов MTA. Как и многие TCP/IP Unix-службы, простой протокол передачи почты был спроектирован таким образом, чтобы облегчить реализацию и ручное тестирование работоспособности служб. Одним из минусов такой прозрачности стало злоупотребление различными возможностями, такими как ретрансляция (relaying) почты, которой пользовались для массовой рассылки сообщений и просто для создания анонимных писем. Авторы писем счастья и финансовых «пирамид» часто пользуются чужими почтовыми серверами, чтобы скрыть адрес исходного MTA. Это может приводить к потокам ругани в адрес администратора сервера, не отключившего ретрансляцию.

Существует два варианта SMTP, которым должны соответствовать SMTP-совместимые решения: исходный протокол SMTP и расширенный простой протокол передачи почты (Extended Simple Mail Transfer Protocol, ESMTP). Оба протокола поддерживают базовые команды (такие как указание автора сообщения MAIL FROM: и адресата сообщения RCPT TO:). При этом ESMTP поддерживает дополнительные возможности, вроде уведомления о состоянии доставки (delivery status notification, DSN:) и отображения заявленного размера сообщения (SIZE:).

Агенты MTA, MDA и MUA взаимодействуют в процессе обмена почтой, однако только локальный и удаленный MTA-агенты говорят на языке SMTP между собой. Локальные MDA и MUA не могут общаться с удаленными MDA и MUA. Когда пользователь отправляет сообщение с помощью агента MUA, через локальный MTA, устанавливается SMTP-соединение с удаленным MTA. Удаленный MTA связывается с удаленным MDA, который отвечает за доставку сообщения удаленному MUA.

Обмен по протоколам SMTP и ESMTP между агентами MTA происходит посредством последовательностей простых команд. Сейчас мы узнаем, что делают некоторые из этих команд:

DATA

Идентифицирует передаваемые данные как сообщение электронной почты.

EHLO

Сообщает, что в дополнение к протоколу SMTP поддерживается также ESMTP.

EXPN

Возвращает перечень пользователей, принадлежащих определенному почтовому списку.

HELO

Сообщает о поддержке SMTP.

MAIL

Указывает почтовый адрес отправителя.

QUIT

Завершает сеанс передачи почты.

RCPT

Указывает почтовый адрес получателя.

VRFY

Определяет, имеет ли указанный пользователь корректную учетную запись в системе.

В дополнение к этим стандартным командам SMTP доступен ряд команд ESMTP:

8BITMIME

Предупреждает, что передаются восьмибитные данные.

DSN

Доставка уведомления о состоянии для отправленных сообщений.

ETRN

Старт удаленной очереди сообщений.

ONEX

Передача одного сообщения.

SIZE

Указывает размер сообщения.

VERB

Отображает сообщения о состоянии.

XUSR

Передает данные удаленного пользователя на сервер.

Теперь посмотрим, как эти команды могут использоваться непосредственно для отправки почты. Самый простой локальный MTA – это клиент Telnet; TCP-соединение устанавливается через стандартный порт SMTP (порт с номером 25), а команды, о которых сказано выше, могут выполняться в диалоговом режиме. Таким образом, у нас есть возможность увидеть ответные сообщения удаленного MTA, что очень полезно при отладке системы *sendmail*, на которую пало подозрение в некорректной работе.

Последовательность команд для передачи почты вполне прозрачна:

1. Команда *HELO* или *EHLO* поступает от локального MTA удаленному MTA. Ответ указывает, может ли быть установлено соединение.
2. Команда *MAIL* указывает почтовый адрес отправителя.
3. Команда *RCPT* указывает почтовый адрес получателя.
4. Команда *DATA* инициализирует процесс передачи сообщения.
5. Точка в отдельной строке (.) завершает передачу сообщения.
6. Команда *QUIT* завершает сеанс.

Посмотрим, как эта схема работает для реального сообщения; воспользуемся командой *telnet* для доступа к удаленному MTA и отправки сообщения:

```
recruit:10:01:natashia> telnet develop 25
Trying 204.183.64.22 ...
Connected to develop.cassowary.net.
Escape character is '^]'.
220 develop.cassowary.net ESMTP Sendmail 8.8.8/8.8.8; Wed, 17 Jan 2001
11:15:05 +1000
    (EST)
HELO recruit
250-develop.cassowary.net Hello recruit.cassowary.net [204.183.64.25],
pleased to meet you
MAIL FROM: <natashia@recruit.cassowary.net>
250 <natashia@recruit.cassowary.net>... Recruit ok
RCPT TO: <paul@develop.cassowary.net>
250 <paul@develop.cassowary.net>... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
Testing...
Dear Paul,
We've found a suitable applicant for Position #578568 (Solaris Network
Administrator).
Best,
Natashia
.
250 MAA44556 Message accepted for delivery
```

```
QUIT
221 develop.cassowary.net closing connection
Connection closed by foreign host.
```

В этом примере пользователь *natashia@recruit.cassowary.net* отправил сообщение пользователю *paul@develop.cassowary.net* с помощью базовых команд *HELO*, *MAIL*, *RCPT*, *DATA*, . и *QUIT*. Обратите внимание, что на каждый запрос локального MTA удаленный MTA отвечает кодом состояния (220, 250 или 354). Эти коды определены протоколом SMTP и интерпретируются локальными MTA. Они обозначают успех либо отрицательный результат выполнения запроса. Более подробная информация содержится в упомянутом документе RFC.

Агент передачи сообщений

Изучив протокол SMTP и обмен данными почтовых сообщений между узлами, мы сконцентрируем свое внимание на агенте передачи сообщений (Mail Transfer Agent, MTA) *sendmail*, который является стандартным демоном передачи сообщений в Solaris. Одной из причин популярности *sendmail* на системах Solaris является его широкое признание в качестве стандартного почтового транспорта Unix-систем. *sendmail* обеспечивает непревзойденную гибкость в передаче почтовый сообщений, и если длина файла настройки *sendmail* (*sendmail.cf*) составляет несколько тысяч строк, в этом нет ничего удивительного, поскольку процесс определения способа доставки сообщения одному из многих миллионов узлов невероятно сложен. И неудивительно, что руководства по *sendmail* занимают многие тысячи страниц; большинство системных администраторов знакомы со службой *sendmail*, но очень немногие считают себя специалистами по этой службе.

К счастью, существует возможность задать определенные параметры в процессе установки системы Solaris (и/или воспользоваться командой *sys-unconfig*) и получить стартовые настройки пакета *sendmail* для большинства стандартных сетевых конфигураций. Это означает, что от вас не требуется понимание тонкостей настройки *sendmail*, чтобы эта служба заработала в Solaris. Поэтому в данном разделе мы сконцентрируемся непосредственно на работе *sendmail* в среде систем Solaris. И самое главное, мы рассмотрим процесс получения и установки обновлений для *sendmail*, доступных на сайте пакета (*http://www.sendmail.org/*).

Читатели, возможно, задаются вопросом, зачем производить сборку *sendmail* из исходных текстов, если это настолько замечательная программа и входит в состав Solaris в качестве одного из пакетов. Дело в том, что история существования этого пакета связана с многочисленными уязвимостями различных платформ, которые стали прямым следствием ошибок в коде *sendmail*. И хотя разработчики операционных систем, поддерживающих *sendmail* (включая и Solaris), как пра-

вило, выпускают заплаты по мере обнаружения ошибок, критические ошибки не позволяют ждать так долго. Многие администраторы предпочитают загружать, настраивать и устанавливать *sendmail* из исходных текстов при обнаружении уязвимости и не рисковать, оставляя злоумышленникам возможность ею воспользоваться. Примером такой уязвимости может служить подверженность переполнению буфера демона, которая позволяет внешнему пользователю выполнять в системе команды с полномочиями администратора.

sendmail можно бесплатно получить по адресу *ftp://ftp.sendmail.org/*. Последняя версия *sendmail* – V8.11.2, хотя во многих работающих системах до сих пор используются более ранние версии ветви V8. Лучший способ получить самые свежие новости о *sendmail* и обнаруженных уязвимостях – соединиться с FTP-сайтом и прочесть вступительный текст:

```
bash-2.03# ftp ftp.sendmail.org
Connected to ftp.sendmail.org.
220 vorlon.sendmail.com FTP server (Version 6.00) ready.
Name (ftp.sendmail.org:pwatters): ftp
331 Guest login ok, send your email address as password.
Password:
230- This directory contains sendmail 8.x source distributions.  Those
230- interested in mirroring the sendmail distribution tree should read
230- the MIRROR file in this directory.
230-
230- The latest version is available in sendmail.8.11.2.tar.{Z,gz,sig} --
230- the .Z file is compressed, the .gz file is the same bits gzipped, and
230- the .sig file is a PGP signature for the uncompressed bits in either
230- of the first two files.  Please take ONLY ONE of the .Z or .gz files.
230-
230- A commercial version of sendmail 8.11 including precompiled ``push
230- button'' install and a GUI configuration and administration interface
230- is available from Sendmail, Inc. (see http://www.sendmail.com/
230- for details).
230-
230- Older versions are in sendmail.${VER}.tar.{Z,gz,sig}.  Except for the
230- latest, these are unsupported by the Sendmail.ORG crew.  The status of
230- various interesting ${VER}s is:
230-
230- 8.11.2 Many mostly minor fixes -- see RELEASE_NOTES for details.
230- 8.11.1 Many mostly minor fixes -- see RELEASE_NOTES for details.
230- 8.11.0 Add support for STARTTLS and SASL ecryption.  Some minor fixes.
230- 8.10.2 Detect and avoid a serious Solaris capabilities security bug.
230- 8.10.1 Bug fix release: avoids dangerous AIX 4.X linker behavior
230- 8.10.0 Major new release: multiple queues, SMTP authentication, LDAP
230-        integration, IPv6, enhanced SMTP status codes, and more.
230- 8.9.3  header denial of service fixed.  Minor fixes.
230- 8.9.2  accept() denial of service attack on Solaris systems fixed.
230-        Berkeley DB 2.X integration fixed.  Many minor fixes.
```

```
230- 8.9.1  Many mostly minor fixes -- see RELEASE_NOTES for details. Clarify
230-        LICENSE terms.
230- 8.9.0  New major release with focus on spam control with many other
230-        new features -- see RELEASE_NOTES for details.
230- 8.8.8  Many mostly minor fixes -- see RELEASE_NOTES for details.
230-
230- Since sendmail 8.11 and later includes hooks to cryptography, the
230- following information from OpenSSL applies to sendmail as well.
230-
230- PLEASE REMEMBER THAT EXPORT/IMPORT AND/OR USE OF STRONG CRYPTOGRAPHY
230- SOFTWARE, PROVIDING CRYPTOGRAPHY HOOKS OR EVEN JUST COMMUNICATING
230- TECHNICAL DETAILS ABOUT CRYPTOGRAPHY SOFTWARE IS ILLEGAL IN SOME
230- PARTS OF THE WORLD.  SO, WHEN YOU IMPORT THIS PACKAGE TO YOUR
230- COUNTRY, RE-DISTRIBUTE IT FROM THERE OR EVEN JUST EMAIL TECHNICAL
230- SUGGESTIONS OR EVEN SOURCE PATCHES TO THE AUTHOR OR OTHER PEOPLE
230- YOU ARE STRONGLY ADVISED TO PAY CLOSE ATTENTION TO ANY EXPORT/IMPORT
230- AND/OR USE LAWS WHICH APPLY TO YOU. THE AUTHORS ARE NOT LIABLE FOR
230- ANY VIOLATIONS YOU MAKE HERE. SO BE CAREFUL, IT IS YOUR RESPONSIBILITY.
230-
230- $Revision: 1.25 $, Last updated $Date: 2002/01/10 17:53:15 $
230 Guest login ok, access restrictions apply.
```

Большинство дистрибутивов *sendmail* не требуют внесения изменений в систему. Тем не менее потребуется изменить файл *sendmail.cf* на предмет его соответствия существующей версии. Некоторые режимы и модификаторы могут меняться от версии к версии, но основные правила подстановки, локальная информация и списки надежных пользователей не должны меняться с изменением версии.

sendmail.cf

Главным файлом настройки для *sendmail* является *sendmail.cf*. Каждая строка *sendmail.cf* предваряется буквой, которая соответствует определенному разделу. Строки определяют макроподстановки, правила, заголовки и различные параметры, определяющие стиль работы *sendmail*. Из определяющих функциональность строки букв наиболее часто встречаются следующие:

C Сложная макроподстановка

D Простая макроподстановка

E Переменная окружения

H Почтовый заголовок

M Имя агента MDA

O Параметр

P Приоритет сообщения

R Правило обработки адреса

S Определение набора правил

Сам файл *sendmail.cf* разделен на несколько разделов, которые связаны с настройкой различных аспектов работы агента. Раздел Local Info содержит информацию по настройке и отправке почты для локального узла. Типичный раздел Local Info определяет следующие свойства:

Cwlocalhost

Имя локального узла.

Fw-o /etc/mail/sendmail.cw %[^\#]

Имя файла, в котором перечислены узлы, пользующиеся почтовым сервером для получения почты.[1]

Dj$w.paulwatters.com

Официальное имя домена.

CO @ %

Символы, которые запрещено использовать в именах локальных узлов.

FR-o /etc/mail/relay-domains %[^\#]

Перечень узлов, разрешающих ретрансляцию почты.

Dnroot

Имя локального пользователя, получающего сообщения об ошибках.

Kmailertable hash -o /etc/mail/mailertable.db

Имя файла таблицы пересылки почты.

Kgenerics hash -o /etc/mail/genericstable.db

Имя файла общей таблицы.

Kvirtuser hash -o /etc/mail/virtusertable.db

Имя файла таблицы виртуальных пользователей.

Kaccess hash -0 /etc/mail/access.db

Имя файла базы данных списков доступа для использования ретрансляции.

DZ8.11.2/Solaris Solaris 8DZ8.11.2/Solaris Solaris 8

Номер версии и номер сборки локальной системы *sendmail*.

---

[1] В реальности макропеременная *w* определяет, какие имена узлов будут трактоваться как локальная почта, то есть записываться непосредственно в почтовый ящик адресата. Команды *C* и *F* определяют, откуда читать эти данные: из определения в *sendmail.cf* или из файла, указанного в команде *F*. – *Примеч. науч. ред.*

Раздел Options относится уже к действиям *sendmail* в качестве агента передачи почтовых сообщений. Этот раздел содержит, как минимум, следующие параметры:

O SevenBitInput=False

Определяет необходимость преобразовывать полученные данные в семибитный формат.

O EightBitMode=pass8

Определяет необходимость поддержки передачи восьмибитных данных.

O AliasWait=10

Интервал обновления файла почтовых синонимов (*aliases*) (в минутах).

O AliasFile=/etc/aliases

Полное имя файла почтовых синонимов.

O MinFreeBlocks=100

Минимальное число свободных блоков локальной файловой системы.

O MaxMessageSize=1000000

Максимальный размер для передаваемого сообщения (в байтах).

O BlankSub=.

Символ, используемый вместо пробелов.

O DeliveryMode=background

Режим работы при доставке почты (по умолчанию).

Раздел Message Precedence определяет приоритеты для различных видов сообщений. Обычно используются приоритеты первого класса (first class), срочной доставки (special delivery), списка рассылки (mailing list), групповой отправки (bulk mail) и мусорной почты (junk mail):

Pfirst-class=0

Первый класс.

Pspecial-delivery=100

Срочная доставка.

Plist=–30

Список рассылки.

Pbulk=–60

Групповая отправка.

Pjunk=–100

Мусорная почта.

В разделе Trusted Users перечислены пользователи, которым разрешено осуществлять управление локальной почтовой системой:

Troot

Пользователь *root*.

Tdaemon

Пользователь *daemon*.

Tpwatters

Пользователь *pwatters*.

В разделе Header Format с помощью набора переменных, динамически изменяемых в шаблоне для каждого отдельного сообщения и пользователя, определяется формат заголовков сообщений:

```
H?P?Return-Path: <$g>
HReceived: $?sfrom $s $.$?_($?s$|from $.$_)
   $.$?{auth_type}(authenticated)
   $.by $j ($v/$Z)$?r with $r$. id $i$?u
   for $u; $|;
   $.$b
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>
```

И наконец, в разделе Rewriting Rules определяются правила маршрутизации и доставки сообщений (адресованных определенным адресатам) через соответствующий удаленный MTA:

```
Scanonify=3
R$@                 $@ <@>
R$*                 $: $1 <@>
R$* < $* > $* <@>   $: $1 < $2 > $3
R@ $* <@>           $: @ $1
R$* :: $* <@>       $: $1 :: $2
R:include: $* <@>           $: :include: $1
R$* [ IPv6 $- ] <@>         $: $1 [ IPv6 $2 ]
R$* : $* [ $* ]             $: $1 : $2 [ $3 ] <@>
R$* : $* <@>                $: $2
R$* <@>                     $: $1
R$* ;                       $1
R$* < $* ; >                $1 < $2 >
R$@                         $@ :; <@>
R$*                         $: < $1 >
R$+ < $* >                  < $2 >
R< $* > $+                  < $1 >
R<>                         $@ < @ >
R< $+ >                     $: $1
```

```
R@ $+ , $+                $2
R@ $+ : $+                $2
R $+ : $* ; @ $+          $@ $>Canonify2 $1 : $2 ; < @ $3 >
R $+ : $* ;               $@ $1 : $2;
R$+ @ $+                  $: $1 < @ $2 >
R$+ < $+ @ $+ >           $1 $2 < @ $3 >
R$+ < @ $+ >              $@ $>Canonify2 $1 < @ $2 >
R$* < @ $* : $* > $*      $1 < @ $2 $3 > $4
R$- ! $+          $@ $>Canonify2 $2 < @ $1 .UUCP >
R$+ . $- ! $+             $@ $>Canonify2 $3 < @ $1 . $2 >
R$+ ! $+                  $@ $>Canonify2 $2 < @ $1 .UUCP >
R$* % $*          $1 @ $2
R$* @ $* @ $*     $1 % $2 @ $3
R$* @ $*          $@ $>Canonify2 $1 < @ $2 >
R$*               $@ $>Canonify2 $1
```

Передача почты

Итак, мы рассмотрели файл настройки *sendmail.cf* и переходим к собственно работе *sendmail*. Мы можем предписать *sendmail* работу в режиме подробной диагностики (*sendmail -v*) и передать программе файл, содержащий сообщение удаленному адресату:

```
gladiator:maya> cat data.txt | /usr/lib/sendmail -v paul@paulwatters.com
paul@paulwatters.com... Connecting to mail-incoming.hostsave.com (TCP)...
220 mail.paulwatters.com ZMailer Server 2.99.38 #1 ESMTP ready at Sat, 11 Nov
2000
    08:22:01 -0700
>>> HELO gladiator.cassowary.net
250 mail.paulwatters.com Hello gladiator.cassowary.net
>>> MAIL From:<maya@gladiator.cassowary.net>
250 (verified non-local) Ok
>>> RCPT To:<paul@paulwatters.com>
250 (verified local) Ok
>>> DATA
354 Start mail input; end with <CRLF>.<CRLF>
Paul,
Sales have increased 1000% this month thanks to your innovative marketing
campaign!
Sincerely,
Manager
>>> .
250 2.6.0 Roger
>>> QUIT
221 2.0.0 mail.paulwatters.com Out
paul@paulwatters.com... Sent
```

Успешная передача этого сообщения не только сделает получателя счастливым[1], но будет иметь и ряд других следствий. Мы интерпретируем каждую из строк, чтобы процесс обмена стал более понятным. Команда *cat* передает содержимое файла *data.txt* в поток стандартного

вывода, который связан конвейером (|) с программой *sendmail* (работающей в режиме подробной диагностики); в качестве адресата указан пользователь *paul@paulwatters.com*. Первая строка вывода говорит о том, что устанавливается TCP-соединение с удаленным MTA по инициативе локального; код ответа 220 подтверждает, что соединение успешно установлено. Этот ответ также содержит некоторую информацию об удаленном MTA, в частности номер версии (ZMailer Server 2.99.38) и местное время (Sat, 11 Nov 2000 08:22:01 –0700). Затем локальный MTA пытается инициировать сеанс SMTP (не ESMTP), отправив сообщение *HELO*. Удаленный MTA возвращает код 250, указывающий на поддержку SMTP. Затем локальный MTA выполняет команду MAIL, указывая почтовый адрес отправителя. Удаленный MTA подтверждает получение команды кодом 250 и проверяет, что отправитель является внешним (то есть не принадлежит тому же узлу или домену, что и адресат). Локальный MTA выполняет команду *RCPT*, указывая адресата сообщения. Если пользователь существует на удаленном узле (как в этом случае), MTA-собеседник должен вернуть код 250. Затем локальный MTA выполняет команду *DATA*, за которой передается собственно тело сообщения, завершаемое точкой в отдельной строке. Удаленный MTA подтверждает получение сообщения кодом 250. Наконец, локальный MTA выполняет команду *QUIT*, которую удаленный MTA подтверждает кодом 221.

Этот пример предполагает, что у адресата существует рабочая учетная запись на системе удаленного MTA либо адресат является именем списка рассылки (определенного в */etc/aliases*), существующего для удаленного MTA. При этом некоторые агенты поддерживают возможность ретрансляции, которая позволяет передавать через них почту, адресованную нелокальным пользователям. И хотя в последние годы ретрансляция активно используется взломщиками и спамерами в целях сокрытия исходного MTA, в глобальных и межкорпоративных сетях ретрансляция может быть необходима. Рассмотрим пример, аналогичный предыдущему; в данном случае удаленный MTA является ретранслятором:

```
gladiator:maya> cat data.txt | /usr/lib/sendmail -v paul@paulwatters.com
paul@paulwatters.com... Connecting to relay.paulwatters.com. via relay...
220 relay.paulwatters.com ESMTP Sendmail 8.11.2/8.11.2; Wed, 23 Jan 2001
23:44:02
    +1100 (EST)
>>> EHLO gladiator.paulwatters.com
250- relay.paulwatters.com Hello maya@gladiator.paulwatters.com
[204.34.32.12],
    pleased to meet you
```

[1] Текст сообщения гласит: «Пол, продажи в этом месяце увеличились на 1000% благодаря твоей передовой маркетинговой политике! С уважением, менеджер.» – *Примеч. перев.*

```
250-EXPN
250-VERB
250-8BITMIME
250-SIZE 10000000
250-DSN
250-ONEX
250-ETRN
250-XUSR
250 HELP
>>> MAIL From:<maya@gladiator.paulwatters.com> SIZE=2344
250 <maya@gladiator.paulwatters.com >... Sender ok
>>> RCPT To:<paul@paulwatters.com>
250 <paul@paulwatters.com>... Recipient ok
>>> DATA
354 Enter mail, end with "." on a line by itself
Paul,
Sales have increased 1000% this month thanks to your innovative marketing
campaign!
Sincerely,
Manager
>>> .
250 NAA32434 Message accepted for delivery
paul@paulwatters.com... Sent (NAA32434 Message accepted for delivery)
Closing connection to relay.paulwatters.com.
>>> QUIT
221 relay.paulwatters.com closing connection
```

Заголовки сообщений

Одной из замечательных черт SMTP является возможность в большинстве случаев отследить путь почтового сообщения до исходного MTA по его заголовку. Эти строки кода, формат которых определяется файлом *sendmail.cf* (примеры см. выше), содержат многочисленные характеристики сообщения, включая информацию по отправителю и получателю. Некоторые из полей заголовка являются обязательными, другие – факультативными. Рассмотрим некоторые из наиболее распространенных полей:

Content-Length

Размер сообщения.

Content-Type

Кодировка мультимедиа-вложений (MIME-тип).

Date

Время и дата получения сообщения агентом MTA.

From

Почтовый адрес отправителя.

Message-Id

Идентификатор сообщения, сгенерированный агентом MTA.

Received

Имя агента-получателя или ретранслятора.

Subject

Тема сообщения.

To

Почтовый адрес получателя.

При диагностировании проблем, связанных со службами MTA, эти заголовки могут стать источником ценнейшей информации о том, как сообщение было доставлено исходным MTA и как было получено конечным MTA. Вот пример заголовка сообщения:[1]

```
From paul@paulwatters.com Fri Jan 05 12:56:48 2001
Received: by mail (maya@cassowary.net)
   (with Cubic Circle's cucipop (v1.31 1998/05/13) Thu Jan  4 17:45:18 2001)
X-From_: paul@paulwatters.com  Thu Jan  4 17:43:13 2001
Return-Path: <paul@paulwatters.com>
Delivered-To: maya@mail.cassowary.net
Received: from relay.cassowary.net (unknown [207.213.224.109])
   by mail.cassowary.net (Postfix) with ESMTP id 027D8A9EC
   for <maya@cassowary.net>; Thu,  4 Jan 2001 17:43:01 -0800 (PST)
Received: from relay.paulwatters.com ([203.45.43.12]) by relay.cassowary.net
with ESMTP id <686182-13482>; Thu, 4 Jan 2001 17:38:43 -0800
Received: from mail.paulwatters.com (paul@mail.paulwatters.com [137.111.240.12])
   by relay.paulwatters.com (8.8.8/8.8.8) with ESMTP id MAA26872
   for <maya@cassowary.net>; Fri, 5 Jan 2001 12:38:04 +1100 (EST)
Received: (from paul@localhost)
   by mail.paulwatters.com (8.9.1a/8.9.1) id MAA18951
   for maya@cassowary.net; Fri, 5 Jan 2001 12:38:02 +1100 (EST)
From: WATTERS Paul Andrew <paul@paulwatters.com>
Message-Id: <200101050138.MAA18951@mail.paulwatters.com>
Subject: Re: Neil
In-Reply-To: <5.0.0.25.1.20010105124552.009d4180@mail.cassowary.net> from Maya
   Watters at "Jan 5, 2001 12:46:33 pm"
To: Maya Watters <maya@cassowary.net>
Date: Fri, 5 Jan 2001 12:38:02 +1100 (EST)
X-Mailer: ELM [version 2.4ME+ PL71 (25)]
MIME-Version: 1.0
Content-Type: text/plain; charset=US-ASCII
Content-Transfer-Encoding: 7bit
```

[1] Одновременно с заголовками существует понятие конверта сообщения, которое подробно описано в RFC-821. Конверт отличается от заголовков отсутствием символа «:» в конце наименования поля. Конверт имеет преимущество перед заголовками при обработке почтового сообщения. В приведенном ниже примере первая строчка является конвертом. – *Примеч. науч. ред.*

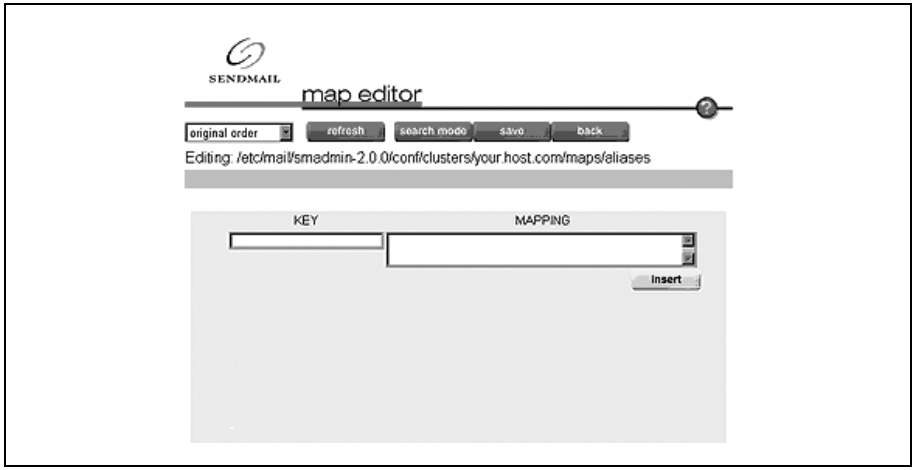

Рис. 6.5. *Графический интерфейс коммерческой версии sendmail*

```
Content-Transfer-Encoding: 7bit
Sender: paul@paulwatters.com
```

Коммерческая поддержка

sendmail является бесплатным пакетом, но для него доступна коммерческая поддержка (*http://www.sendmail.com*). Коммерческую версию *sendmail* несколько легче настраивать; для выполнения всех операций настройки и администрирования применяется графический веб-интерфейс. На рис. 6.5 приведен снимок экрана из этого приложения. Кроме того, в коммерческой версии *sendmail* доступны более совершенные возможности управления и мастеры настройки для тех, кто не знаком с макроподстановками и правилами.

Хотя Solaris предоставляет многочисленные варианты и возможности организации различных служб, рассудительное применение средств и принудительное следование явно определенным правилам (в противовес принятию решений в каждом отдельном случае) позволяют надеяться, что система будет работать согласно ожиданиям. Кроме того, пользователям будет гораздо легче пользоваться службами, если вы сможете четко объяснить, для чего те или иные правила (такие, например, как квоты) нужны.

7

Файловые серверы

Многие службы Solaris основаны на архитектуре клиент–сервер, в которой основные операции возложены на выделенные высокопроизводительные системы. Одной из стандартных задач для серверов Solaris является организация распределенного хранения файлов либо безопасного и надежного доступа к файлам одного или нескольких серверов для внешних клиентов. В большинстве современных компьютерных систем присутствуют собственные средства хранения данных (за исключением «тонких» клиентов, вроде терминалов Sun Ray), так зачем же нужны файловые серверы? Из-за их надежности, емкости и защищенности.

В локальных клиентских системах, как правило, наблюдается довольно низкий уровень безопасности и проверки допустимости операций, связанных с файловыми системами, в особенности, если установленная операционная система не требует идентификации пользователя при доступе к файлам (так обстоит дело с некоторыми версиями Microsoft Windows). Компании, которым необходимо защищать свои файлы, обычно вкладывают средства в безопасность физического уровня (запираемые серверы, доступ к которым имеют только сотрудники техподдержки), а также в безопасные операционные системы (такие как Solaris) в целях обеспечения доступа с проверкой подлинности на уровне узлов, пользователей и паролей.

Кроме того, серверы способны хранить большие объемы данных, но не потому, что их диски имеют физически больший объем, а потому что имеют возможность пользоваться преимуществами RAID-технологии (Redundant Array of Inexpensive Disks, избыточный массив недорогих дисков) в целях увеличения размеров создаваемых разделов. Так что дисковое пространство двух дисков емкостью по 9,1 Гбайт может адре-

соваться как один раздел размером 18,2 Гбайт. Это очень важно, например, для приложений баз данных, поскольку файлы данных и журналы отката со временем могут достигать очень больших размеров. Для реализации программных RAID-решений может использоваться пакет Sun DiskSuite, хотя в большинстве серверных систем применяются специальные аппаратные RAID-решения (такие как RAID-массивы StorEdge).

Надежность является, вероятно, наиболее важным качеством централизованного хранения файлов. Удаленные пользователи доверяют свои файлы серверу, считая его более надежным и безопасным, чем локальный жесткий диск, и у них не должно возникать сомнений в правдивости такого предположения.

Хотя емкость дисков лимитируется аппаратными ограничениями, обеспечение безопасности и надежности является преимущественно задачей базовых протоколов доступа к файлам и операционной системы. В Solaris для создания распределенных файловых систем существует два основных протокола. Первым является сетевая файловая система (Network File System, NFS), разработанная компанией Sun, в которой применена технология вызова удаленных процедур (Remote Procedure Call, RPC) для обработки запросов клиентов и серверов. Реализации клиентов NFS существуют для многих других операционных систем (в том числе для систем Linux и Microsoft Windows), хотя эти клиенты не всегда гладко интегрируются с существующими службами. С Solaris 8 в настоящее время поставляется NFS версии 3, а недавно опубликованная спецификация RFC для NFS версии 4 будет реализована в следующих версиях инструментария NFS. Как правило, NFS применяется для организации совместного доступа к файлам Unix-систем и систем, подобных Unix.

Второй протокол – протокол блоков серверных сообщений (Server Message Block, SMB). Он широко применяется в системах Microsoft Windows для организации совместного доступа к файлам и службам печати. Система совместного доступа к файлам Samba (*http://www.samba.org/*) существует в бесплатном варианте для Solaris. Одним из больших преимуществ Samba в гетерогенной сетевой среде, где сервер Solaris предоставляет доступ к файлам клиентам Microsoft Windows, заключается в том, что Samba позволяет осуществлять доступ и к другим службам (а не только к файлам и принтерам), таким как первичное управление доменом (Primary Domain Control, PDC) для доменов NT и других рабочих групп Microsoft Windows. Реализация Samba для Microsoft Windows позволяет монтировать файловые системы Solaris с помощью сетевого окружения (Network Neighborhood) либо любых других метафор графического интерфейса, облегчая надежную и безопасную интеграцию систем Solaris с PC-клиентами.

В этой главе мы рассмотрим создание вариантов распределенных файловых систем Solaris на базе SMB и NFS.

Samba

Samba – это эффективный, межплатформенный инструмент, позволяющий организовывать централизованное хранение данных на сервере Solaris, к которому обращаются клиенты, работающие с разнообразными операционными системами. Совместное использование файлов и принтеров Solaris становится доступным для не-Unix-клиентов, работающих под управлением Microsoft Windows, Mac OS, а также с другими реализациями Unix и Unix-подобными системами (вроде Linux). Кроме того, Samba позволяет осуществлять управление и сопровождение доменов Windows NT и доменных служб (связанных, например, с проверкой подлинности) для многих видов клиентов Windows (Windows 95, 98, NT и даже Windows 2000). По функциональности Samba превосходит NFS, но ее применение означает применение неподдерживаемой программы, которая дублирует существующие возможности Solaris (в частности, проверку подлинности) собственными реализациями. Применение Samba исключительно оправдано при необходимости работы с большим количеством клиентов Windows, когда использование Windows-сервера невозможно из-за чрезмерно высокой стоимости или вопросов безопасности.

Текущая стабильная версия Samba имеет номер 2.0.x. Существует четыре основных ветви разработки Samba: 2.0, 2.2, 3.0 и TNG. Версия Samba 2.0, в том числе ее различные вариации (2.0.5, 2.0.7), применяется для работы во многих компаниях. Версия 2.2 находится в альфа-состоянии и будет, в конечном итоге, содержать многие возможности, которых не хватает версии 2.0, в частности улучшенную поддержку Windows 2000. Версия 3.0 будет содержать такие нововведения, как модуль *winbind*. И наконец, ветвь TNG является альтернативой программному комплексу Samba и развивается отдельной командой разработчиков.

Хотя пользователи могут применять текущий код версии 2.0, доступный на сайте *ftp://ftp.samba.org/*, некоторые из администраторов могут склоняться к использованию версии 2.2 – в целях тестирования. В последнем случае необходимо извлечь текущую версию из CVS-репозитория по адресу *http://pserver.samba.org/* (доступ только для чтения). Следует выполнить такую команду CVS:

```
cvs -d :pserver:cvs@pserver.samba.org:/cvsroot login
```

Пароль службы (разумеется) *cvs*. Как вариант исполняемые версии программ Samba в формате пакетов Solaris доступны по адресу *http://www.sunfreeware.com*. Samba не входит в дистрибутив Solaris.

Сервер Samba

В Solaris поддержкой работы служб Samba занимаются два демона. Сервер имен NetBIOS (*nmbd*), работающий через порт с номером 137,

поддерживает поиск NetBIOS-имен в пределах локального домена Windows и является обязательным барьером для клиентов, подключающихся к серверу Samba. Все сетевые клиенты Windows умеют работать с системой имен NetBIOS. В дополнение к *nmbd* демон Samba (*smbd*), работающий через порт 139, обеспечивает работу доступных клиентам служб совместного доступа к файлам и принтерам. Каждый клиент, соединяющийся с демоном Samba, обслуживается отдельным экземпляром *smbd*; поэтому в системе с большим числом клиентов будет большое число процессов *smbd*.

Для инициализации упомянутых демонов Samba следует выполнить приведенные ниже команды; предполагается, что пакет Samba установлен в каталог */usr/local/samba-2.2.0*:

```
bash-2.03# /usr/local/samba-2.2.0/bin/nmbd -D
bash-2.03# /usr/local/samba-2.2.0/bin/smbd -D
```

После тщательного тестирования и отладки установки Samba в каталоге */etc/init.d* можно создать сценарий запуска/останова и связать его с каталогом *rc* конкретного уровня функционирования системы, чтобы служба автоматически запускалась после загрузки. Создайте файл */etc/init.d/.samba* следующего содержания и создайте символьную ссылку на файл в выбранном каталоге *rc* (например, */etc/rc2.d/S99samba*):

```
#!/bin/sh
# Samba startup/shutdown
case "$1" in 'start')
if [ -f /usr/local/samba-2.2.0/bin/smbd ] then
/usr/local/samba-2.2.0/bin/smbd -D
echo "Samba daemon starting"
(Запускается демон Samba)
/usr/local/samba-2.2.0/bin/nmbd -D
echo "NetBIOS name daemon starting"
(Запускается демон имен NetBIOS)
fi
;;
'stop')
kill -9 `pgrep smbd`
echo "Samba daemon killed"
(Демон Samba остановлен)
kill -9 `pgrep nmbd`
echo "NetBIOS daemon killed"
(Демон NetBIOS остановлен)
;;
*)
echo "usage: samba {start|stop}"
;;
esac
```

Настройка Samba производится добавлением и/или удалением записей в файле */usr/local/samba-2.2.0/lib/smb.conf*. Типичный файл настройки содержит записи примерно следующего вида:

```
[global]
workgroup = TAHITI
netbios name = VAHINE
server string = Solaris Samba Server 2.2.0
interfaces = 203.65.54.22
security = SHARE
log file = /var/log/samba-2.2.0/log.%m
max log size = 100
socket options = TCP_NODELAY SO_RCVBUF=16384 SO_SNDBUF=16384
dns proxy = yes
guest account = guest
domain logins = yes
hosts allow = localhost, 203.65.54.22/255.255.255.0

[homes]
comment = VAHINE User Directories
read only = No
browseable = No
guest ok = No

[intranet]
comment = VAHINE Intranet System
path = /usr/local/htdocs/intranet
guest ok = Yes

[netlogon]
comment = Windows 95 Net Logons
path = /usr/local/samba-2.2.0/lib/netlogon
guest ok = yes
writable = no
share modes = no

[printers]
comment = VAHINE Printers
path = /var/spool/lp
print ok = Yes
browseable = Yes
guest ok = Yes
```

В этом файле настройки пять разделов: *global*, *homes*, *intranet*, *netlogon* и *printers*. Раздел настроек *global* содержит значения базовых параметров работы Samba, которые являются общими для всех ресурсов совместного доступа. Среди этих параметров имя гостевой учетной записи (*guest*), IP-адреса и/или доменные имена клиентов, которым разрешено обращаться к серверу, имя файла журнала, имя рабочей группы, имя NetBIOS и уровень безопасности (в данном случае применяется безопасность уровня разделяемого ресурса).

Остальные четыре раздела посвящены конкретным ресурсам. Раздел *homes* позволяет производить экспорт отдельных домашних каталогов пользователей и разрешает запись в каталоге для клиентов, подлинность которых установлена. Таким образом, пользователь *tane* не может получить доступ к файлам пользователя *tangaroa* на *vahine*, если только *tangaroa* не сделает свои файлы доступными для чтения всем. Ресурс *intranet* разрешает гостевой доступ и запись файлов в каталог *vahine: /usr/local/htdocs/intranet* для всех пользователей. И хотя на практике глобальное разрешение на запись – идея не очень хорошая, этот пример демонстрирует гибкость и возможности настроек системы Samba.

Кроме того, поддерживается служба Net Logon (настройки приведены в разделе *netlogon*) для клиентов Windows 95; разрешен блокирующий гостевой доступ только для чтения. И наконец, раздел *printers* разрешает гостевой доступ к службам печати на машине *vahine*. Если нет необходимости учитывать доступ пользователей к принтерам и службам печати, можно разрешить гостевой доступ ко всем принтерам системы.

В табл. 7.1 перечислены наиболее распространенные параметры раздела общих настроек файла *smb.conf*.

Таблица 7.1. Основные параметры настройки smb.conf

Параметр	Описание	Пример
Workgroup	Имя домена или рабочей группы Microsoft Windows	*workgroup=eire*
server string	Описание серверной части SMB	*server string=Solaris 2.2.0 Samba Alpha Build*
hosts allow	Список управления доступом для узлов, которым разрешается монтирование удаленных ресурсов	*hosts allow=localhost 143.234.212.23*
load printers	Определяет необходимость загрузки списка принтеров из файла *printcap*	*load printers=yes*
printcap name	Полное имя файла *printcap* или *lpstat*	*printcap name=/etc/printcap*
guest account	Имя пользователя непривилегированной учетной записи	*guest account=nobody*
log file	Полное имя файла журнала Samba	*log file=/var/log/samba.log*
max log size	Максимальный размер файла журнала (в килобайтах)	*max log size=100*
Security	Уровень безопасности для доступа к разделяемым ресурсам	*security=user*
encrypt passwords	Определяет необходимость использования зашифрованных паролей	*encrypt passwords=no*

Параметр	Описание	Пример
socket options	Определяет поведение сокетов TCP/IP	*socket options= TCP_NODELAY*
Interfaces	Указывает сетевые интерфейсы, которые прослушивает демон Samba	*interfaces= 143.234.212.23/24 143.234.211.23/24*
local master	Определяет, является ли Samba мастер-службой локальной подсети	*local master=no*
domain master	Определяет, является ли Samba мастером локального домена	*domain master=yes*
domain controller	Определяет имя локального PDC	*domain controller=maximus*
logon script	Определяет имя сценария регистрации, который может различаться для разных систем и пользователей	*logon script=logon.bat*
domain logins	Определяет поддержку регистрации в домене клиентов Windows 95	*domain logins=no*

Инструменты Samba

Существует ряд команд, которые могут применяться для изучения производительности сервера Samba. Самой важной командой является *smbstatus*, которая отображает состояние сервера. В следующем примере состояние демона Samba отображается с дополнительными подробностями (ключ *-d*):

```
vahine:/usr/local/samba-2.2.0/bin > ./smbstatus -d

Processing section "[homes]"
Processing section "[intranet]"
Processing section "[netlogon]"
Processing section "[printers]"
using configfile = /usr/local/samba-2.2.0/lib/smb.conf
lockdir = /usr/local/samba/var/locks
Opened status file /usr/local/samba/var/locks/STATUS..LCK

Samba version 2.2.0
Service      uid      gid      pid     machine
----------------------------------------------
homes        paul     staff    1845   tane     (203.65.54.23) Mon Jan 21 07:2
1:11 2001
intranet     tim      staff    2343   tiare    (203.65.54.24) Mon Jan 21 10:1
0:12 2001
intranet     sukhdev  staff    9986   moorea   (203.65.54.25) Mon Jan 21 12:1
2:01 2001
homes        brad     staff   15554   borabora (203.65.54.26) Mon Jan 21 14:0
2:56 2001
```

```
Locked files:
Pid    DenyMode   R/W        Oplock           Name
--------------------------------------------------
1845   DENY_NONE  RDWR       NONE             /home/paul/shares.xls
Mon Jan 21 08:21:11 2001
1845   DENY_NONE  RDWR       NONE             /home/paul/securities.doc
Mon Jan 21 09:21:11 2001
2343   DENY_NONE  RDONLY EXCLUSIVE+BATCH      /home/tim/dates.txt
Mon Jan 21 11:07:12 2001
2343   DENY_NONE  RDONLY     NONE             /home/tim/tasks.xls
Mon Jan 21 12:01:10 2001
9986   DENY_NONE  RDWR       NONE             /etc/passwd
Mon Jan 21 13:01:21 2001
9986   DENY_NONE  RDONLY EXCLUSIVE+BATCH      /etc/shadow
Mon Jan 21 14:16:31 2001

Share mode memory usage (bytes):
    2096928 (98%) free + 1500 (0.1%) used + 1500 (0.1%) overhead =  2099928
(100%) total
```

Вывод команды *smbstatus* состоит из трех разделов: отчет по пользователям, отчет по заблокированным файлам и отчет по использованию памяти. В пользовательском отчете содержится информация по каждому из клиентских процессов, работающих с сервером. В отчете упоминается имя ресурса совместного доступа, имя учетной записи, для которой установлено соединение, имя группы, в которую входит пользователь, идентификатор клиентского процесса, а также имя узла и IP-адрес подключившейся машины. Кроме того, отображается дата и время, когда было установлено соединение.

В отчете по заблокированным файлам перечисляются все файлы, открытые клиентскими процессами. Для каждого файла отображается идентификатор процесса, полное имя файла, а также прочая информация (в частности, режим доступа, в котором был открыт файл). Отчет по использованию памяти отражает потребление памяти сервером Samba.

Для службы имен NetBIOS существует также команда состояния *nmblookup*. Команда отображает имена узлов локального домена, откликающихся на широковещательные сообщения:

```
bash-2.03# nmblookup tahiti
Added interface ip=203.65.54.22 bcast=203.65.54.255 nmask=255.255.255.0
Sending queries to 203.65.54.255
Got a positive name query response from 203.65.54.22 (203.65.54.22)
Got a positive name query response from 203.65.54.23 (203.65.54.23)
Got a positive name query response from 203.65.54.24 (203.65.54.24)
Got a positive name query response from 203.65.54.25 (203.65.54.25)
Got a positive name query response from 203.65.54.26 (203.65.54.26)
Got a positive name query response from 203.65.54.27 (203.65.54.27)
Got a positive name query response from 203.65.54.28 (203.65.54.28)
203.65.54.22 TAHITI<00>
203.65.54.23 TAHITI<00>
203.65.54.24 TAHITI<00>
```

```
203.65.54.25 TAHITI<00>
203.65.54.26 TAHITI<00>
203.65.54.27 TAHITI<00>
203.65.54.28 TAHITI<00>
```

Клиент Samba

В Samba существует большое количество типов клиентов, которые могут получать доступ к экспортируемым ресурсам. Например, пользователи Microsoft Windows могут просматривать доступные ресурсы сервера Samba, обращаясь к сетевому окружению (Network Neighborhood) и интересующему их домену. Пользователям Linux доступно прямое монтирование ресурсов Samba – поддержка файловой системы Samba встроена в ядро Linux.

Однако пользователи Solaris более ограничены в доступе к ресурсам Samba. Наиболее распространенным Solaris-клиентом Samba является *smbclient*. Он поддерживает разнообразные файловые операции (перечисленные в табл. 7.2) и предоставляет интерфейс, схожий с интерфейсом FTP-клиента.

Чтобы начать сеанс работы в *smbclient*, достаточно просто передать программе имя сервера Samba, с которым необходимо связаться, с помощью ключа *-L*:

```
tane# smbclient -L vahine
```

Когда соединение узлов *tane* и *vahine* успешно установлено, отображается следующее приглашение в стиле FTP:

```
smb:>
```

В табл. 7.2 перечислены наиболее популярные команды *smbclient*, включая команды смены каталога на удаленной системе, отображения перечня файлов в каталоге, а также копирования файлов в обе стороны.

Таблица 7.2. Удаленная работа с файлами в smbclient

Команда	Описание
cd	Сменить рабочий каталог на удаленной системе
dir	Отобразить подробную информацию по всем файлам в текущем рабочем каталоге
get	Получить указанный файл с сервера
ls	Отобразить краткий список файлов в текущем рабочем каталоге
mget	Получить указанные файлы с сервера; не запрашивать подтверждение передачи
mput	Скопировать указанные файлы на сервер, в текущий рабочий каталог
put	Скопировать указанный файл на сервер, в текущий рабочий каталог

В дополнение *smbclient* является весьма удобным инструментом диагностирования проблем и тестирования, поскольку может использоваться для проверки того, что локальный сервер Samba действительно делает ресурсы доступными, как и запланировано:

```
vahine# smbclient -U% -L vahine
Added interface ip=203.65.54.22 bcast=203.65.54.255 nmask=255.255.255.0
Domain=[TAHITI] OS=[Unix] Server=[Samba 2.2.0]

        Sharename      Type      Comment
        ---------      ----      -------
        homes          Disk      VAHINE User Directories
        intranet       Disk      VAHINE Intranet System
        netlogon       Disk      Windows 95 Net Logons
        IPC$           IPC       IPC Service (Samba 2.2.0)

        Server               Comment
        ---------            -------
        VAHINE               Samba 2.2.0

        Workgroup            Master
        ---------            -------
        TAHITI               TANE
```

Приведенная диагностика *smbclient* показывает, что для всех ресурсов, перечисленных в файле *smb.conf* (включая *homes*, *intranet* и *netlogon*), организован корректный совместный доступ.

Первичное управление доменом (PDC)

Samba PDC (Primary Domain Control) позволяет пользователям различных клиентских систем регистрироваться в домене NT, которым управляет система Solaris. (Этот домен не является доменом Windows 2000, хотя клиенты Windows 2000 могут к нему присоединиться.) Samba PDC обладает многими возможностями Windows NT и 2000; например, проверка подлинности для клиентов домена организована с применением централизованной базы паролей. Кроме того, любому из пользователей домена могут быть назначены административные полномочия для работы с любой из клиентских систем домена; определенные в специальных файлах правила могут применяться к любой из систем домена, а при подключении пользователя к клиенту могут выполняться разнообразные сценарии системы PDC.

При этом сервер Samba PDC обладает и рядом недостатков: в настоящее время не реализована поддержка резервных контроллеров домена (Backup Domain Controllers, BDCs), которые активно применяются в системах непрерывного действия (рис. 7.1). Если необходима поддержка только локальных клиентов и существует дополнительная система с установленным PDC, который может быть запущен в любое время, отсутствие BDC может оказаться не столь критичным для управле-

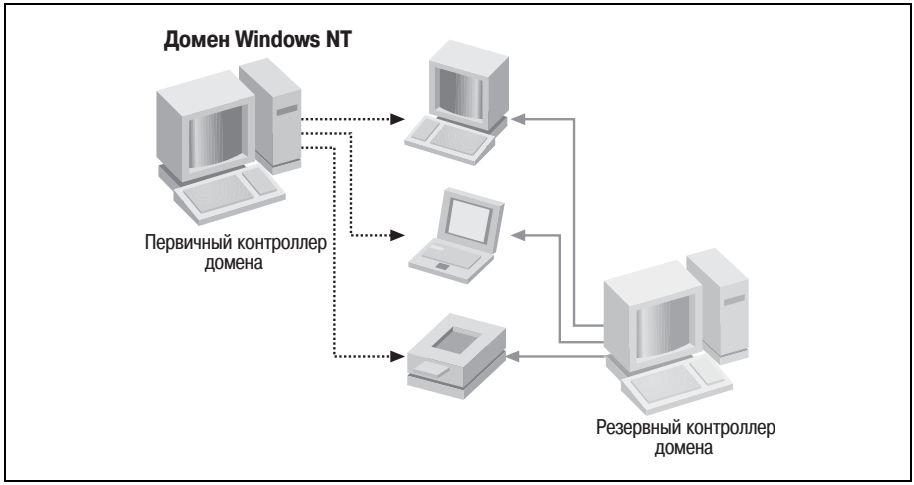

Рис. 7.1. *Управление доменом Microsoft Windows посредством PDC и BDC*

ния доменом NT. Еще одно ограничение связано с системой Windows 2000, доверительные отношения которой не могут быть распространены на ресурсы Samba; в настоящее время Samba поддерживает только клиентов Windows 2000, присоединившихся к домену NT, управляемому PDC. Тем не менее сервер Samba, работающий на системе Solaris, может присоединиться к домену Windows 2000 и быть его полноправным участником; при этом предполагается, что для PDC включена система имен NetBIOS.

Вот пример файла настройки для PDC:

```
[global]
security = share
status = no
workgroup = TAHITI
wins server = 203.65.54.22
encrypt passwords = no
domain logons = yes
logon script = scripts\logon.bat
guest account = nobody
share modes=yes
os level=88
[homes]
guest ok = no
read only = no
create mask = 0666
directory mask = 0666
oplocks = true
locking = yes
[netlogon]
path = /usr/local/samba-2.2.0/netlogon
```

```
writeable = yes
guest ok = yes
```

За более подробной информацией администраторам следует обращаться к файлам *DOMAIN_CONTROL.txt*, *DOMAIN.txt* и *NTDOMAIN.txt* из дистрибутива Samba в целях более тщательного изучения организации управления доменом в Samba.

Совместный доступ к ресурсам Windows

До сих пор мы изучали клиентскую сторону сетевого взаимодействия SMB, не концентрируясь на серверной, которая обычно представлена системой Microsoft Windows NT. Для тех, кто никогда не настраивал совместный доступ к ресурсам Windows, я кратко опишу основные шаги. Во-первых, используя Проводник (Explorer), найдите папку, для которой необходимо организовать совместный доступ. Щелкните правой кнопкой мыши по пиктограмме папки и выберите пункт Sharing (Доступ) из контекстного меню. Появится окно, аналогичное показанному на рис. 7.2, в котором находятся настройки совместного доступа для выбранного ресурса файловой системы. Если совместный доступ к ресурсу уже включен, будет выбрана позиция Shared As (Совместный доступ), а поля Share Name (Сетевое имя) и Comment (Комментарий) будут активными.

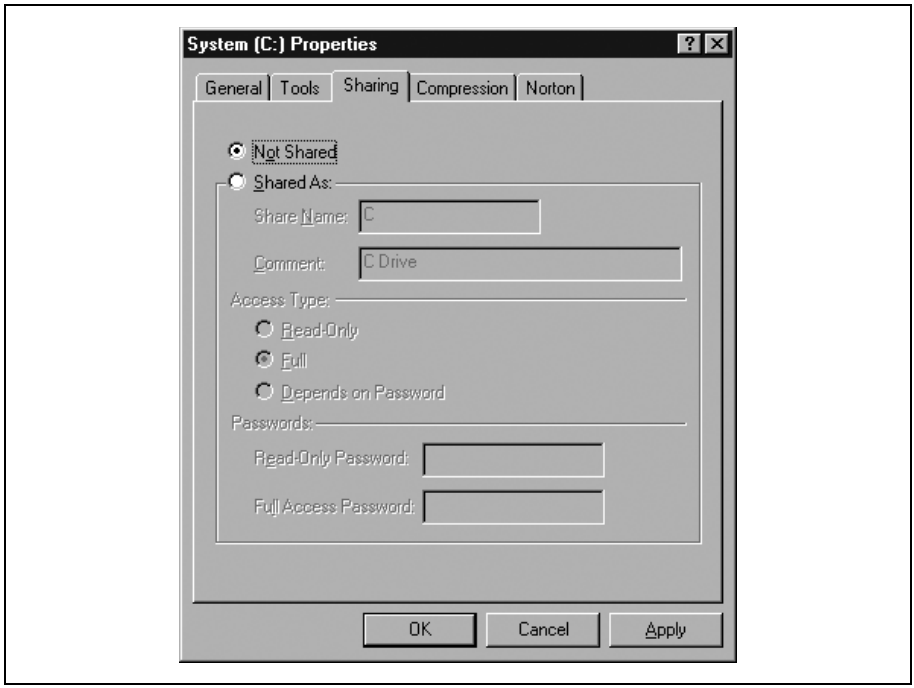

Рис. 7.2. *Просмотр настроек совместного доступа для папки*

Предоставляется возможность задать один из трех уровней доступа: только для чтения (Read-Only), когда клиентам разрешено только читать файлы, но не записывать новые или обновлять существующие; полный доступ (Full), позволяющий запись новых и обновление существующих файлов, а также доступ на основе пароля (Depends on Password), при котором права доступа (только для чтения или полный доступ) определяются на основе пароля, предоставленного подключившимся клиентом. Для режимов Read-Only и Full следует ввести различные пароли в случае использования Depends on Password.

На рис. 7.3 показаны настройки для совместного доступа к диску C: с именем ресурса *C* и комментарием «C Drive». Полный доступ к диску разрешен клиентам, предоставившим соответствующий пароль.

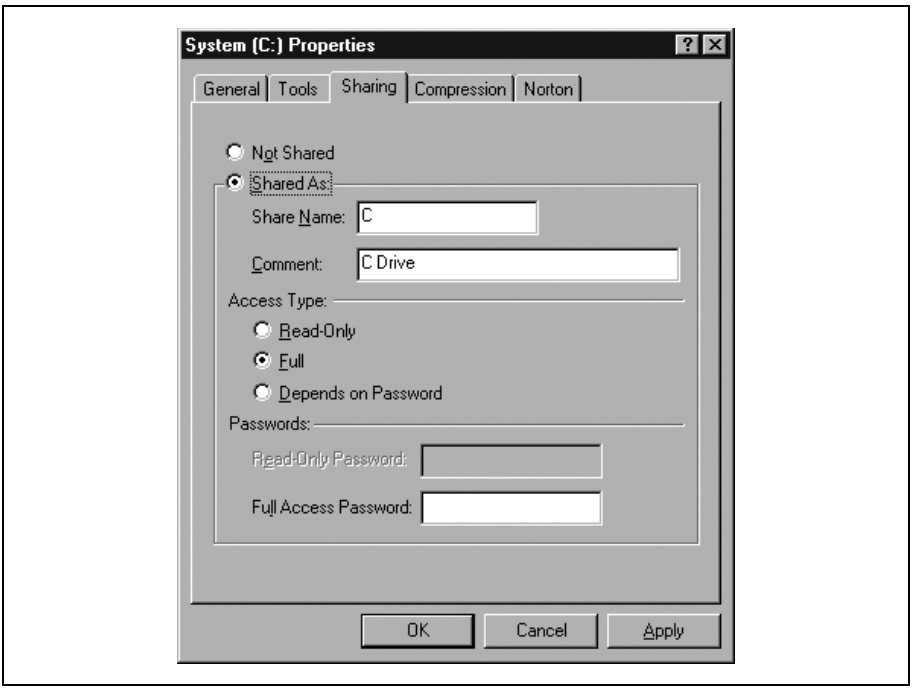

Рис. 7.3. *Совместный доступ к диску C*

Диагностирование проблем

Если есть сложности с запуском Samba, а *smbclient* утверждает, что ресурсы не экспортируются, придется воспользоваться командой *testparm* и точно определить, какие параметры связаны с Samba. Многие из этих параметров, не будучи явно заданными в файле *smb.conf*, возвращаются к значениям по умолчанию, что может привести к неожиданным результатам. Кроме того, существует возможность конфликта

параметров. Скажем, поведение системы не определено в случае одновременного указания следующих настроек:

```
writeable = yes
read only = yes
```

Ресурс доступен одновременно только для чтения и для записи! Команда *testparm* отображает все явно заданные настройки либо неявно предполагаемые демоном Samba. Вывод команды *testparm* довольно объемен, поэтому для изучения параметров настройки разумно будет передать его команде *more*:

```
vahine# testparm | more
Load smb config files from /usr/local/samba-2.2.0/lib/smb.conf
Processing section "[homes]"
Processing section "[netlogon]"
Processing section "[intranet]"
Loaded services file OK.
WARNING: You have some share names that are longer than 8 chars
These may give errors while browsing or may not be accessible
to some older clients
Press enter to see a dump of your service definitions
# Global parameters
[global]
        workgroup = TAHITI
        netbios name =
        netbios aliases =
        server string = Samba 2.2.0
        interfaces =
        bind interfaces only = No
        security = USER
        encrypt passwords = Yes
        update encrypted = No
        allow trusted domains = Yes
        hosts equiv =
        min passwd length = 5
        map to guest = Never
        null passwords = No
        password server =
        smb passwd file = /usr/local/samba-2.2.0/private/smbpasswd
        root directory = /
        passwd program = /bin/passwd
        passwd chat debug = No
        username map =
        password level = 0
        username level = 0
        unix password sync = No
        restrict anonymous = No
        use rhosts = No
        log level = 2
        syslog = 1
```

```
syslog only = No
log file = /var/log/Samba.log.%m
max log size = 50
```

Здесь мы изучили только параметры общей настройки Samba; вывод для всех разделяемых ресурсов будет значительно более объемным.

Серверы NFS

Сетевая файловая система (Network File System, NFS) стала одним из первых протоколов распределенного доступа к файлам благодаря активному продвижению компанией Sun. В Solaris эта система реализована на базе технологии вызовов удаленных процедур (Remote Procedure Call, RPC), которая позволяет клиентам вызывать методы с удаленных серверов, пользуясь в качестве транспорта протоколами UDP и TCP.

Текущая версия NFS имеет номер 3, именно она входит в состав Solaris 8. Серверы NFS, как правило, должны поддерживать клиентов NFS 2 и NFS 3. Если говорить о будущем, недавно был опубликован документ RFC для NFS 4, и предполагается, что эта версия серверов и клиентов NFS будет реализована Sun в последующих версиях Solaris.

В этом разделе будут рассмотрены основные рабочие характеристики различных версий NFS, а также практические примеры реализации архитектур распределенных файловых систем на базе NFS. Централизованное и более рациональное хранение файлов на базе NFS позволяет более эффективно управлять ресурсами и, как правило, сокращать общее число систем, для которых необходимо резервное копирование.

Версии NFS

Система NFS версии 2 была описана в документе RFC 1094, NFS: Network File System Protocol Specification (Спецификация протокола сетевой файловой системы). Главной задачей NFS 2 было создание межплатформенной распределенной файловой системы, которая позволяла бы пользователям одной клиентской системы получать доступ к разделяемым ресурсам любых других систем. Кроме того, система, выступающая в роли клиента, должна была получить возможность независимо работать в качестве сервера и также предоставлять доступ к NFS-ресурсам другим клиентам. Второй задачей NFS 2 была независимость от конкретного транспортного протокола: в одних реализациях применяется UDP, в других TCP. Выбор транспорта оставлен на усмотрение разработчика конкретной реализации NFS 2.

В основу системы NFS 2 легла разработанная Sun технология RPC, позволяющая клиентам вызывать функции и/или методы с сервера. Работа RPC основана на обмене внешними представлениями элементов данных по протоколу eXternal Data Representation (XDR) (описанному в документе RFC 1040, XDR: External Data Representation Stan-

dard). Можно с легкостью определить, какие из служб (с транспортами TCP и UDP) используют RPC, выполнив команду *rpcinfo*:

```
bash-2.03$ rpcinfo -p
   program vers proto   port  service
    100000    4   tcp    111  rpcbind
    100000    3   tcp    111  rpcbind
    100000    2   tcp    111  rpcbind
    100000    4   udp    111  rpcbind
    100000    3   udp    111  rpcbind
    100000    2   udp    111  rpcbind
    100024    1   udp  32778  status
    100024    1   tcp  32771  status
    100133    1   udp  32778
    100133    1   tcp  32771
    100021    1   udp   4045  nlockmgr
    100021    2   udp   4045  nlockmgr
    100021    3   udp   4045  nlockmgr
    100021    4   udp   4045  nlockmgr
    100232   10   udp  32781
    100011    1   udp  32782  rquotad
    100002    2   udp  32783  rusersd
    100002    3   udp  32783  rusersd
    100002    2   tcp  32772  rusersd
    100002    3   tcp  32772  rusersd
    100012    1   udp  32784  sprayd
    100008    1   udp  32785  walld
    100001    2   udp  32786  rstatd
    100001    3   udp  32786  rstatd
    100001    4   udp  32786  rstatd
```

Можно видеть, что демон *walld* использует в качестве транспорта UDP, а сервер статистики ядра (*rstatd*) – TCP. Приложение *rpcbind*, связывающее программы с конкретными портами RPC, прослушивает как TCP-, так и UDP-порты.

Предполагается, что серверы NFS 2 не сохраняют информацию о состоянии клиента; отслеживание активности клиентов возложено на них самих. Это делает системы NFS 2 невероятно устойчивыми к ошибкам: если сервер NFS перезагружен либо перезапущена служба NFS 2, клиенты не теряют изменений, внесенных в открытые файлы. Достаточно снова подключиться к серверу, когда он продолжит работу, и нет необходимости изменять какие-либо настройки или повторно инициализировать процессы; попытки соединения продолжаются, пока соединение не будет установлено. NFS 2 поддерживает 18 самостоятельных методов, описанных в табл. 7.3.

В NFS 2 не реализован механизм проверки подлинности; предполагается, что клиенты и серверы соответствуют базовым стандартам идентификации в стиле Unix-систем. Пользователь, подключающийся с системы-клиента под определенным регистрационным именем, пройдет стандартную проверку подлинности, которая используется в дру-

гих службах удаленного доступа (например, в FTP). Таким образом, NFS 2 лучше всего подходит для систем, поддерживающих идентификацию в стиле Unix (это означает, что для не-Unix-систем, вроде Microsoft Windows, NFS 2 по большей части непригодна). Чтобы не-Unix-клиент мог подключиться к Unix-серверу NFS, должно быть реализовано взаимодействие механизмов проверки подлинности клиента (если таковой присутствует) и сервера. В NFS 2 поддерживается (факультативно) проверка подлинности с применением DES.

Таблица 7.3. Методы, реализованные в NFS 2

Метод	Описание
NFSPROC_NULL	Используется для измерения скорости работы и проверки состояния
NFSPROC_GETATTR	Возвращает все атрибуты для указанного файла
NFSPROC_SETATTR	Устанавливает все атрибуты для указанного файла
NFSPROC_ROOT	Возвращает ссылку на корень файловой системы
NFSPROC_LOOKUP	Разрешает имя файла в указанном каталоге
NFSPROC_READLINK	Возвращает данные из файла, на который указывает данная символическая ссылка
NFSPROC_READ	Читает конкретный объем данных из указанного файла
NFSPROC_WRITECACHE	Не реализована
NFSPROC_WRITE	Записывает конкретный объем данных в указанный файл
NFSPROC_CREATE	Создает указанный файл
NFSPROC_REMOVE	Удаляет указанный файл
NFSPROC_RENAME	Переименовывает указанный файл
NFSPROC_LINK	Создает жесткую ссылку для указанного файла
NFSPROC_SYMLINK	Создает символическую ссылку для указанного файла
NFSPROC_MKDIR	Создает указанный каталог
NFSPROC_RMDIR	Удаляет указанный каталог
NFSPROC_READDIR	Читает содержимое указанного каталога
NFSPROC_STATFS	Возвращает сведения по файловой системе, включая число свободных блоков

В RPC существует ряд параметров, которые могут использоваться для подстройки клиент-серверных соединений NFS, и некоторые из этих параметров могут коренным образом влиять на качество соединений. Например, клиент может смонтировать ресурс NFS 2 одним из двух способов: «жестко», когда попытки смонтировать запрошенный ресурс продолжаются до бесконечности, либо «мягко», когда попытки

прекращаются по достижении определенного числа повторов. Конкретный способ монтирования должен определяться на основе надежности локальной сети, в которую входят клиент и сервер. В дополнение к различным способам монтирования могут задаваться различные размеры для блоков передачи данных (по умолчанию 8192 байт). Последний параметр может меняться в зависимости от природы физического транспорта, применяемого в сети.

После каждой транзакции NFS 2 интерпретируется код завершения, известный как *stat*. Успешному завершению транзакции соответствует код NFS_OK. При этом существует ряд ситуаций, приводящих к получению кода ошибки. Описания кодов приведены в табл. 7.4.

Таблица 7.4. Коды завершения транзакций NFS 2

Код	Описание
NFS_OK	Транзакция завершена успешно
NFSERR_PERM	У клиента недостаточно полномочий для выполнения запроса
NFSERR_NOENT	Запрошенный файл или каталог не существует
NFSERR_IO	Произошла ошибка ввода-вывода
NFSERR_NXIO	Указанное устройство не существует
NFSERR_ACCES	У клиента недостаточно полномочий для выполнения запроса
NFSERR_EXIST	Файл, указанный в запросе, уже существует в целевой файловой системе
NFSERR_NODEV	Указанное устройство не существует
NFSERR_NOTDIR	Указанный каталог является обычным файлом
NFSERR_ISDIR	Указанный файл является каталогом
NFSERR_FBIG	Запрошенный файл имеет слишком большой размер
NFSERR_NOSPC	Переполнение файловой системы сервера
NFSERR_ROFS	Запрос на запись в файловую систему, доступную только для чтения
NFSERR_NAMETOOLONG	Длина имени для запрошенного файла превышает допустимую для файловой системы
NFSERR_NOTEMPTY	Запрос на удаление каталога отвергнут; в каталоге существуют файлы
NFSERR_DQUOT	Квота пользователя на сервере исчерпана
NFSERR_STALE	Файл уже не существует на сервере, либо права доступа для него изменились, и он перестал быть доступным для чтения текущему пользователю
NFSERR_WFLUSH	Кэш сервера был очищен

В NFS 3 появился ряд нововведений, описанных в документе RFC 1813, (NFS Version 3 Protocol Specification). Так, общее число транзакций для клиентов было понижено в целях повышения эффективности управления файловыми атрибутами, а все ограничения на размеры передаваемых файлов были сняты. NFS 3 реализует обратную совместимость с клиентами NFS 2, что позволяет производить обновление сервера до NFS 3 с минимальными неудобствами для клиентов. Существующие механизмы Unix- и DES-идентификации были дополнены поддержкой идентификации Kerberos. Что касается аппаратного обеспечения, самым важным нововведением стала поддержка 64-битных архитектур. Методы, реализованные в NFS 3, описаны в табл. 7.5.

Таблица 7.5. Методы, реализованные в NFS 3

Метод	Описание
NFSPROC3_NULL	Используется для измерения скорости работы и проверки состояния
NFSPROC3_GETATTR	Возвращает все атрибуты для указанного файла
NFSPROC3_SETATTR	Устанавливает все атрибуты для указанного файла
NFSPROC3_LOOKUP	Разрешает имя файла в указанном каталоге
NFSPROC3_ACCESS	Проверяет, что у пользователя достаточно полномочий для выполнения операций над файлами
NFSPROC3_READLINK	Возвращает данные из файла, на который указывает названная символическая ссылка
NFSPROC3_READ	Читает конкретный объем данных из указанного файла
NFSPROC3_WRITE	Записывает конкретный объем данных в указанный файл
NFSPROC3_CREATE	Создает указанный файл
NFSPROC3_MKDIR	Создает указанный каталог
NFSPROC3_SYMLINK	Создает указанную символическую ссылку
NFSPROC3_MKNOD	Создает специальный файл устройства
NFSPROC3_REMOVE	Удаляет указанный файл
NFSPROC3_RMDIR	Удаляет указанный каталог
NFSPROC3_RENAME	Переименовывает указанный файл или каталог
NFSPROC3_LINK	Создает жесткую ссылку для указанного файла
NFSPROC3_READDIR	Читает содержимое указанного каталога
NFSPROC3_READDIRPLUS	Читает содержимое указанного каталога по несколько записей
NFSPROC3_FSSTAT	Возвращает сведения по файловой системе, включая число свободных блоков

Таблица 7.5 (продолжение)

Метод	Описание
NFSPROC3_FSINFO	Возвращает информацию о состоянии файловой системы, включая рекомендованные размеры для операций NFSPROC3_READ и NFSPROC3_WRITE
NFSPROC3_PATHCONF	Возвращает POSIX-данные по указанному файлу или каталогу
NFSPROC3_COMMIT	Очищает кэш записи

Значительным образом изменились коды завершения транзакций NFS (табл. 7.6).

Таблица 7.6. Коды завершения транзакций NFS 3

Код	Описание
NFS3_OK	Транзакция завершена успешно
NFS3ERR_PERM	У клиента недостаточно полномочий для выполнения запроса
NFS3ERR_NOENT	Запрошенный файл или каталог не существует
NFS3ERR_IO	Произошла ошибка ввода-вывода
NFS3ERR_NXIO	Устройство не существует
NFS3ERR_ACCES	У клиента недостаточно полномочий для выполнения запроса
NFS3ERR_EXIST	Файл, указанный в запросе, уже существует в целевой файловой системе
NFS3ERR_XDEV	Сбой при выполнении операции создания жесткой ссылки для устройства
NFS3ERR_NODEV	Указанное устройство не существует
NFS3ERR_NOTDIR	Указанный каталог является обычным файлом
NFS3ERR_ISDIR	Указанный файл является каталогом
NFS3ERR_INVAL	Некорректный аргумент в запросе
NFS3ERR_FBIG	Запрошенный файл имеет слишком большой размер
NFS3ERR_NOSPC	Переполнение файловой системы сервера
NFS3ERR_ROFS	Запрос на запись в файловую систему, доступную только для чтения
NFS3ERR_MLINK	Превышено максимальное число жестких ссылок
NFS3ERR_NAMETOOLONG	Длина имени для запрошенного файла превышает допустимую для файловой системы
NFS3ERR_NOTEMPTY	Запрос на удаление каталога отвергнут; в каталоге существуют файлы

Код	Описание
NFS3ERR_DQUOT	Дисковая квота пользователя на сервере исчерпана
NFS3ERR_STALE	Файл уже не существует на сервере, либо права доступа для него изменились, и он перестал быть доступным для чтения текущему пользователю
NFS3ERR_REMOTE	Превышено максимальное число уровней пути для файла удаленной файловой системы
NFS3ERR_BADHANDLE	В запросе указан неверный дескриптор файла
NFS3ERR_NOT_SYNC	Произошла ошибка синхронизации
NFS3ERR_BAD_COOKIE	Жетон идентификации недействителен
NFS3ERR_NOTSUPP	Запрошенная операция не поддерживается в текущей версии системы
NFS3ERR_TOOSMALL	Размер запроса не соответствует требованиям по минимальному размеру
NFS3ERR_SERVERFAULT	Сервер недоступен
NFS3ERR_BADTYPE	Тип запрошенного объекта не поддерживается текущей версией системы
NFS3ERR_JUKEBOX	Истек интервал ожидания. Следует повторить запрос

Последняя версия стандарта NFS имеет номер 4 и описана в документе RFC 3010 (NFS Version 4 Protocol). Хотя в NFS 3 и появились некоторые новшества, но многие коды завершения транзакций и методы остались неизменными. Однако в NFS 4 планируются отдельные крупномасштабные изменения в работе NFS. В частности, предлагается реализовать собственный механизм проверки подлинности (RPCSEC_GSS), а не полагаться на механизмы Unix, DES и Kerberos. Впервые станет доступной прозрачная работа через брандмауэры, что облегчит экспорт ресурсов NFS в Интернет. Такой подход связан с определенным риском в плане безопасности, но интеграция шифрования по открытому ключу в механизм идентификации в определенной степени компенсирует этот риск.

Широкое распространение и реализация NFS 4 приведет к повышению гибкости NFS-систем, сняв ответственность за проверку подлинности с операционной системы и возложив ее на внешние механизмы.

Работа с NFS

Мы рассмотрели вызов методов в NFS версий 2, 3 и 4, а также основные коды завершения транзакций, определенные протоколом, и переходим теперь к вопросам установки и настройки серверов и клиентов NFS в Solaris.

Практически любой из разделов, смонтированных локально на сервере NFS 3, может экспортироваться для всех потенциальных клиентов либо только для клиентов, которые получают доступ после идентификации на уровне пользователя или узла. Например, если каталог *htdocs* веб-сервера Apache был экспортирован для всех клиентов локальной сети, все пользователи сети потенциально могут получить доступ к ресурсу *htdocs*, если смонтируют его на своих системах. Для имени узла сервера *kiribati* и пути к каталогу */usr/local/apache-1.3.6/htdocs* ссылка на экспортированный ресурс для клиентов NFS 3 выглядела бы так: *kiribati:/usr/local/apache-1.3.6/htdocs*. Локальные клиенты, монтирующие файловую систему *kiribati:/usr/local/apache-1.3.6/htdocs* вольны выбирать точку монтирования самостоятельно. Поэтому клиентские системы *vanuatu*, *samoa* и *aitutaki* могут смонтировать *kiribati:/usr/local/apache-1.3.6/htdocs* как *vanuatu:/intranet/docs*, *samoa:/exports/htdocs* и *aitutaki:/home/paul/intranet/staging*, соответственно.

Такая же возможность есть и у клиентов NFS 2, обслуживаемых сервером NFS 3 *kiribati*. Так, клиенты NFS 2 *pitcairn* и *mururoa* могут смонтировать *kiribati:/usr/local/apache-1.3.6/htdocs* как *pitcairn:/intranet* и *mururoa:/htdocs*, соответственно.

Совместный доступ к ресурсам

Серверы NFS обычно запускаются во время загрузки системы с помощью сценария, хранимого в каталоге */etc/init.d* и символически связанного с соответствующим каталогом *rc* (например, для файла */etc/init.d/nfs.server* может быть создана ссылка */etc/rc2.d/S99nfs.server*). Этот сценарий, как правило, запускает два самостоятельных процесса: процесс демона сервера NFS (*nfsd*) и процесс демона монтирования NFS (*mountd*). *nfsd* реализует все серверные операции, описанные в предшествующих разделах, а *mountd* непосредственно предоставляет клиентам механизмы, позволяющие запрашивать монтирование локальных для сервера файловых ресурсов. Кроме того, должна функционировать служба RPC.

Перечень экспортируемых ресурсов содержится в файлах */etc/dfs/dfstab* и */etc/dfs/sharetab*. К этим файлам при загрузке или перезапуске обращается демон *nsfd*. Например, бездисковые клиенты могут хранить свои файлы на сервере NFS и с его же помощью производить первичную загрузку. Демон протокола обратного разрешения адресов (Reverse Address Resolution Protocol), *in.rarpd*, работает совместно с сервером параметров загрузки (*rpc.bootparamd*), обеспечивая подобную возможность. Для клиентов x86 может дополнительно потребоваться сервер RPL (Network Booting Remote Program Load).

В процессе останова системы, когда она переходит на уровень работы, в котором были запущены службы NFS, следует прекратить работу всех процессов *nfsd*, *mountd*, *rpc.boot*, *in.rarpd* и *rpld*. Поскольку сер-

вер NFS не сохраняет состояния, элегантное завершение работы этих служб не является приоритетным.

Когда сервер NFS запущен, организацию совместного доступа к разделяемому ресурсу можно элементарно выполнить с помощью стандартной команды *share* с ключом *-F* (тип файловой системы) и аргументом ключа *nfs*. Допустим, необходимо организовать доступ к ресурсу */usr/local/apache-1.3.6/htdocs* сервера *kiribati* для клиента *vanuatu*:

```
kiribati# share -F nfs -o rw=vanuatu /usr/local/apache-1.3.6/htdocs
```

Пользователи системы *vanuatu* получают доступ для чтения и записи к разделу */usr/local/apache-1.3.6/htdocs*. Однако для случая веб-сервера имеет смысл ограничить доступ пользователей *vanuatu* только чтением:

```
kiribati# share -F nfs -o ro=vanuatu /usr/local/apache-1.3.6/htdocs
```

Права доступа в NFS должны определяться явным образом. Обе предыдущих команды не дают доступа к *kiribati:/usr/local/apache-1.3.6/htdocs* пользователям *samoa* и *aitutaki*. Для обеспечения такого доступа выполните команду:

```
kiribati# share -F nfs -o rw=vanuatu,samoa,aitutaki /usr/local/apache-1.3.6/
htdocs
```

Серверы NFS могут одновременно являться клиентами тех систем, для которых обеспечивают доступ. Таким образом, *kiribati* может при необходимости монтировать файловые ресурсы с *vanuatu*, *samoa* или *aitutaki*. Например, система *samoa* может обеспечивать совместный доступ к разделу */data*, содержащему общую для лабораторной группы научную информацию, для систем *kiribati*, *vanuatu* и *aitutaki*. Воспользуйтесь командой:

```
samoa# share -F nfs -o rw=vanuatu,kiribati,aitutaki /data
```

Доступ к экспортированным ресурсам может быть закрыт в любой момент; достаточно просто выполнить команду *unshare*. К примеру, если в целях техобслуживания сервер *samoa* переводится в однопользовательский режим работы, будет уместно выполнить *unshare* для ресурса */data*:

```
samoa# unshare -F nfs /data
```

Нет необходимости выполнять *unshare* для каждого из экспортированных ресурсов; большинство администраторов используют команду *unshareall*, которая автоматически перекрывает доступ ко всем ресурсам NFS:

```
samoa# unshareall -F nfs
```

Если вы точно не помните, какие ресурсы экспортирует сервер, воспользуйтесь командой *dfmounts* для получения соответствующего от-

чета. В следующем примере создается отчет по ресурсам, экспортированным NFS-сервером *kiribati*:

```
kiribati# dfmounts
RESOURCE  SERVER PATHNAME                              CLIENTS
  -       kiribati /usr/local/apache-1.3.6/htdocs   samoa,vanuatu,aitutaki
  -       kiribati /cdrom                           samoa,vanuatu
  -       kiribati /data                            samoa,vanuatu,aitutaki
```

dfmounts отображает информацию по всем экспортированным ресурсам, не сообщая, какие из них в действительности востребованы и какие установлены режимы доступа для пользователей. Команда *share* позволяет получить определенную информацию по ресурсам, которые смонтированы:

```
kiribati# share
-         /usr/local/apache-1.3.6/htdocs rw=vanuatu,aitutaki,kiribati "htdocs"
-          /data  ro=vanuatu,aitutaki,kiribati "data"
```

Монтирование удаленных ресурсов

Чтобы система могла выступать в роли клиента NFS, необходимо наличие двух демонов: демона сетевого состояния (*statd*) и демона сетевых блокировок (*lockd*). Клиенты NFS обычно запускаются во время загрузки системы с помощью сценария, хранимого в каталоге */etc/init.d* и символически связанного с соответствующим каталогом *rc* (например, для файла */etc/init.d/nfs.client* может быть создана ссылка */etc/rc2.d/S99nfs*). Этот сценарий, как правило, запускает процессы демонов *statd* и *lockd*. При запуске эти демоны обращаются к файлу */etc/vfstab* и монтируют перечисленные в нем NFS-ресурсы.

После инициализации демонов клиента NFS монтирование ресурсов производится командой *mount*. Таким образом, NFS соответствует стандартным способам монтирования в Solaris: ключ *-F nfs* команды *mount* указывает тип файловой системы (NFS). Чтобы смонтировать раздел *kiribati:/data* для чтения и записи на *samoa*, воспользуйтесь командой:

```
samoa# mount -F nfs -o rw /data /kiribati/data
```

Мы смонтировали раздел *kiribati:/data* в каталог локальной файловой системы */kiribati/data*. Если точка монтирования не существует, ее следует предварительно создать:

```
samoa# mkdir /kiribati/data
```

Чтобы монтировать раздел *kiribati:/data* во время загрузки системы, следует добавить запись в файл */etc/vfstab* системы *samoa*:

```
kiribati:/data       /kiribati/data    nfs   -   yes   rw
```

Эта строка предписывает монтирование раздела *kiribati:/data* в режиме чтения и записи в каталог */kiribati/data*. После добавления этой записи в файл */etc/vfstab* можно смонтировать сегмент вручную (без необходимости перезагружать систему!):

```
samoa# mount /kiribati/data
```

Чтобы смонтировать все еще не смонтированные системы NFS, воспользуйтесь командой:

```
samoa# mountall
```

Размонтирование файловой системы NFS ничуть не сложнее блокировки доступа: этой цели служит команда *umount*, аргументом которой является имя подлежащей размонтированию файловой системы:

```
samoa# umount /kiribati/data
```

Размонтировать все ресурсы NFS можно с помощью команды:

```
samoa# umountall -F nfs
```

Смонтированный раздел NFS ничем не отличается от файловых систем других типов, применяемых в Solaris: выполнение команды *df* с целью определения объема свободного дискового пространства приводит к получению информации и для всех смонтированных ресурсов NFS. Чтобы получить информацию для всех файловых систем, смонтированных с *kiribati*, воспользуйтесь такой командой:

```
samoa# df -k | grep "kiribati"
kiribati:/data               5567876 4567876 1000000     82%    /kiribati/data
kiribati:/cdrom                345654  345654       0     100%    /cdrom
kiribati:/usr/local/apache-1.3.6/htdocs8098765 7654563  444202     94%    /exports/
    htdocs
```

В данном примере NFS-клиент *samoa* смонтировал три ресурса с сервера NFS *kiribati* (*kiribati:/data*, *kiribati:/cdrom* и *kiribati:/usr/local/apache-1.3.6/htdocs*), которые заняты, соответственно, на 82%, 100% и 94%.

Измерение производительности NFS

Совместный доступ к данным по сети связан с повышенными требованиями к пропускной способности каналов, систем ввода-вывода и производительности процессоров. Требования возрастают линейно с ростом числа клиентов на каждый сервер и экспоненциально с ростом числа серверов. Архитектура SPARC, разработанная с упором на высокоскоростную обработку данных, идеально подходит для целей NFS, даже несмотря на то, что скорость работы процессоров более старых систем является низкой по сегодняшним стандартам. Применение быстрых SCSI-дисков также способствует повышению производитель-

ности NFS, как и реализация физической передачи с помощью 100-мегабитных или гигабитных Ethernet-подключений.

Неудивительно, что в Solaris присутствует целый ряд инструментов, предназначенных для наблюдения за работой NFS, а также приложения общего назначения для измерения производительности; эти инструменты можно использовать для ежедневного анализа функционирования систем NFS и для диагностирования возникающих проблем. В частности, команда *nfsstat* предназначена для оценки производительности клиентов и серверов NFS.

Производительность клиента

Команда *nfsstat* измеряет производительность клиентов NFS для смонтированных ресурсов, основываясь на статистике RPC. Чтобы определить базовые характеристики смонтированных ресурсов, воспользуйтесь такой командой:

```
aitutaki# nfsstat -m
/cdorom from kiribati:/cdrom
 Flags:
vers=3,proto=tcp,sec=sys,hard,intr,link,symlink,acl,rsize=32768,wsize=32768,
retrans=5

/data from kiribati:/data
 Flags:
vers=3,proto=tcp,sec=sys,hard,intr,link,symlink,acl,rsize=32768,wsize=32768,
retrans=5
```

Эти два раздела доступны с машины *kiribati*, на которой используется NFS версии 3 с транспортом TCP; размер блоков записи и чтения составляет 32 Кбайт. При жестком монтировании попытки подключения повторяются каждые пять секунд, пока подключение не будет установлено. Чтобы получить дополнительные данные на стороне клиента, выполните команду *nfsstat* с ключом *-c*:

```
aitutaki# nfsstat -c
Client rpc:
Connection oriented:
calls       badcalls    badxids     timeouts    newcreds    badverfs
33111373    31          7           1           0           0
timers      cantconn    nomem       interrupts
0           23          0           7
Connectionless:
calls       badcalls    retrans     badxids     timeouts    newcreds
5402        35          37          36          70          0
badverfs    timers      nomem       cantsend
0           34          0           0

Client nfs:
calls       badcalls    clgets      cltoomany
33062797    9           33062810    0
```

```
Version 2: (2912 calls)
null         getattr      setattr      root         lookup       readlink
0 0%         2621 90%     0 0%         0 0%         210 7%       7 0%
read         wrcache      write        create       remove       rename
0 0%         0 0%         0 0%         0 0%         0 0%         0 0%
link         symlink      mkdir        rmdir        readdir      statfs
0 0%         0 0%         0 0%         0 0%         72 2%        2 0%
Version 3: (32892344 calls)
null         getattr      setattr      lookup       access       readlink
0 0%         25375035 77% 698529 2%    3021499 9%   2323417 7%   16028 0%
read         write        create       mkdir        symlink      mknod
215951 0%    398993 1%    158604 0%    1023 0%      219 0%       0 0%
remove       rmdir        rename       link         readdir      readdirplus
217302 0%    1024 0%      92651 0%     59782 0%     44682 0%     36361 0%
fsstat       fsinfo       pathconf     commit
178477 0%    231 0%       3577 0%      48959 0%

Client nfs_acl:
Version 2: (2 calls)
null         getacl       setacl       getattr      access
0 0%         0 0%         0 0%         2 100%       0 0%
Version 3: (167578 calls)
null         getacl       setacl
0 0%         167578 100%  0 0%
```

Эти цифры говорят о том, что клиент выполнил 33 111 373 вызова, связанных с соединениями, из которых лишь 31 вызов был некорректным. Это очень хорошее соотношение. Высокое соотношение (0,01 и выше) может свидетельствовать о проблемах, связанных с сетевыми настройками. Число вызовов без явного соединения – 5402, из них 35 некорректных. Порог отношения 0,01 (корректных вызовов к некорректным) по-прежнему не превышен.

За информацией по вызовам следует статистика по числу вызовов каждого из методов (речь о которых шла выше), для NFS 2 и NFS 3. Интересно отметить большое число вызовов метода *getattr* (90% для NFS 2 и 77% для NFS 3), возвращающего атрибуты указанного файла. Напротив, операции записи составляют всего 1% клиентских вызовов NFS 3.

Производительность сервера

Командой *nfsstat* можно воспользоваться и для получения данных, связанных с производительностью сервера (этой цели служит ключ *-s*):

```
kiribati# nfsstat -s

Server rpc:
Connection oriented:
calls        badcalls     nullrecv     badlen       xdrcall      dupchecks
63397311     0            0            0            0            849231
dupreqs
0
```

```
Connectionless:
calls       badcalls    nullrecv    badlen      xdrcall     dupchecks
1370793     0           0           0           0           0
dupreqs
0

Server nfs:
calls       badcalls
64767240    7
Version 2: (1268020 calls)
null        getattr     setattr     root        lookup      readlink
49 0%       132735 10%  0 0%        0 0%        406786 32%  331 0%
read        wrcache     write       create      remove      rename
610174 48%  0 0%        0 0%        0 0%        0 0%        0 0%
link        symlink     mkdir       rmdir       readdir     statfs
0 0%        0 0%        0 0%        0 0%        117941 9%   4 0%
Version 3: (63474247 calls)
null        getattr     setattr     lookup      access      readlink
102783 0%   39822981 62% 39486 0%    9535375 15% 11505221 18% 95345 0%
read        write       create      mkdir       symlink     mknod
1284655 2%  80971 0%    51956 0%    1446 0%     95 0%       0 0%
remove      rmdir       rename      link        readdir     readdirplus
43298 0%    1374 0%     2541 0%     30 0%       264790 0%   628033 0%
fsstat      fsinfo      pathconf    commit
380 0%      1483 0%     8778 0%     3226 0%

Server nfs_acl:
Version 2: (0 calls)
null        getacl      setacl      getattr     access
0 0%        0 0%        0 0%        0 0%        0 0%
Version 3: (24726 calls)
null        getacl      setacl
0 0%        24725 99%   1 0%
```

Статистика по вызовам, связанным с соединением, и вызовам, не требующим соединений, показывает, что отсутствовали: неприемлемые вызовы RPC (*badcalls*), не подлежащие дешифровке вызовы RPC (*xdrcall*) и вызовы RPC с недопустимой длиной (*badlen*). Единственный сомнительный момент связан с высоким числом повторных проверок кэширования RPC (*dupchecks*) в отсутствие дублирующихся запросов RPC (*dupreqs*).

Как и в случае статистики для клиента, наиболее востребованным методом для NFS 3 является *getattr* (62% процента всех вызовов), а число операций записи составляет менее 1%, хотя число операций чтения превышает 2%.

vmstat

Производительность подсистемы виртуальной памяти невероятно важна для успешной работы многих серверов NFS. И хотя желательно

иметь как можно больше физической оперативной памяти, многие серверные системы непрерывно подвергаются высоким нагрузкам, и это означает, что использования виртуальной оперативной памяти не избежать. Для получения статистики по подсистеме виртуальной памяти и связанной с ней активностью (процессорной, дисковой, процессов) может использоваться команда *vmstat*.

Команда *vmstat* без аргументов отображает сведения по потреблению виртуальной памяти с интервалом в одну секунду:

```
kiribati# vmstat 1
 r b w   swap  free  re  mf pi po fr de sr dd f0 s3 s3   in   sy   cs us sy id
 0 0 6   3640  5288   0   7  4  0  2  0  0  1  0  1  0  130  410  246  3  0 97
 0 0 51 254296 4432   0   6  0  0  0  0  0  0  0  0  0  121  162   99  0  0 100
 0 0 51 254296 4432   0   0  0  0  0  0  0  0  0  0  0  116  155  110  0  0 100
 0 0 51 254296 4432   0   0  0  0  0  0  0  0  0  0  0  120  209  103  0  0 100
 0 0 51 254296 4432   0   0  0  0  0  0  0  0  0  0  0  118  144   90  0  0 100
 0 0 51 254296 4432   0   0  0  0  0  0  0  0  0  0  0  122  152  106  0  0 100
 0 0 51 254296 4432   0   0  0  0  0  0  0  0  0  0  0  119  171  101  0  0 100
 0 0 51 254296 4432   0   0  0  0  0  0  0  0  0  1  0  126  146   94  0  0 100
 0 0 51 254296 4432   0   0  0  0  0  0  0  0  0  0  0  127  190  113  0  0 100
 0 0 51 254296 4432   0   0  0  0  0  0  0  0  0  0  0  122  167  100  0  0 100
```

Слева направо три колонки процессов (*r*, *b*, *w*) отражают число работающих, заблокированных и выгруженных в область подкачки процессов (в данном случае нет заблокированных или работающих процессов, все шесть были выгружены в первую секунду). Следующие две колонки посвящены памяти; первая содержит объем свободной виртуальной памяти, а вторая – свободной физической (в первую секунду эти объемы составляли 3184 Кбайт и 5280 Кбайт, соответственно). Следующие семь колонок отражают страничную активность системы. Отображается число восстановленных страниц (*re*), число незначительных ошибок доступа (minor faults) (*mf*), число загруженных данных (в килобайтах), число выгруженных данных, объем освобожденного пространства памяти (*fr*), объем памяти, который должен быть образован под текущие потребности, а также число страниц, сканированных подсистемой памяти (*sr*). Следующие четыре колонки содержат статистику slice-разделов – в числе зарегистрированных на каждый раздел операций (в секунду). И еще три колонки содержат число зарегистрированных ошибок прерываний (*in*), системных вызовов (*sy*) и переключений контекстов процессора (*cs*). Наконец, потребление процессорного времени выражается в показателях для пользователя (*us*), системы (*sy*) и времени бездействия (*id*).

Применение ключа *-c* позволяет получить статистику по сбросу кэша:

```
kiribati# vmstat -c
flush statistics: (totals)
    usr     ctx     rgn     seg     pag     par
      0       0       0       0       0       0
```

Если нас интересует статистика прерываний устройств, ее можно получить с помощью ключа *-i*. Для каждого устройства отображается число зарегистрированных прерываний и их частота:

```
kiribati# vmstat -i
interrupt         total     rate
--------------------------------
clock         265196867      100
hmec0          16295315        6
fdc0                  6        0
--------------------------------
Total         281492188      106
```

Более понятная информация по активности, связанной с виртуальной памятью, доступна по ключу *-s*; отображается общее число страничных операций и данные по операциям подкачки, выполненным с момента запуска системы:

```
kiribati# vmstat -s
       53 swap ins
       38 swap outs
       53 pages swapped in
      211 pages swapped out
 18999597 total address trans. faults taken
   370834 page ins
   115006 page outs
  1390178 pages paged in
   329298 pages paged out
    78773 total reclaims
    73967 reclaims from free list
        0 micro (hat) faults
 18999597 minor (as) faults
   336841 major faults
  5669588 copy-on-write faults
   619789 zero fill page faults
  1533124 pages examined by the clock daemon
      100 revolutions of the clock hand
   916449 pages freed by the clock daemon
   423237 forks
      165 vforks
    12408 execs
655221457 cpu context switches
611511611 device interrupts
 34907914 traps
1090863546 system calls
 20285839 total name lookups (cache hits 93%)
  7530771 user   cpu
  1323161 system cpu
254592511 idle   cpu
  1751860 wait   cpu
```

8

Управление данными

До сих пор мы рассматривали разнообразные способы хранения и работы с данными в системах Solaris как для пользовательских приложений, так и для системных служб. Утверждение может показаться очевидным, но главной причиной создания многих компьютерных систем является *управление* данными. Например, смыслом создания базы данных является надежное хранение изначально достоверных данных, а также организация надежного доступа к ним. В конечном итоге, хранилище достоверных данных бесполезно, если нет возможности получить доступ к данным, а безопасное хранилище, не гарантирующее корректности выполняемых для таблиц транзакций, является такой же глупостью.

В этой главе я расскажу об основах грамотного управления данными и об инструментах, предоставляемых системой Solaris для воплощения этих идей на практике. Мы изучим реальные примеры, связанные с хранением, созданием резервных копий и восстановлением данных. Потерю или повреждение информации можно предотвратить, если придерживаться некоторых решений (таких как зеркалирование и расслоение (striping)), которые реализуются с помощью специальных аппаратных средств, программного обеспечения сторонних разработчиков либо набора простых сценариев и ленточного накопителя. Вы узнаете о распространенных вариантах хранения данных, таких как жесткие диски, приводы CD-ROM и Zip/Jaz.

С потерей данных можно бороться с помощью традиционных стратегий резервного копирования и восстановления, а системы контроля версий, такие как Source Code Control System (SCCS), позволяют сократить общий объем информации, подвергаемой резервному копированию. SCCS позволяет избежать избыточного ежедневного полномасш-

табного копирования путем создания централизованного репозитория данных, в котором хранится история изменения файлов, и существенного сокращения объемов данных, подлежащих копированию. В контексте системного администрирования SCCS также является надежным средством хранения и архивирования файлов настройки системы и приложений, позволяющим отслеживать все вносимые изменения.

Принципы управления данными

Грамотное управление данными сводится к трем основным принципам: целостность, надежность и адекватная оценка емкостных требований. Мы рассмотрим эти принципы по очереди, наряду со средствами Solaris, позволяющими реализовать их.

Целостность

Целостность данных связана с необходимостью гарантировать, что записанные на устройство данные сохранены корректно: если приложение или служба производит запись в файл, используя внутренние значения, данные должны быть записаны надежно. Например, если в файл добавляются последовательности символов (в редакторе *vi*), операционная система должна гарантировать, что измененные данные будут корректно сохранены.

Одним из способов определения целостности данных является вычисление контрольной суммы содержимого файла с помощью команды *sum*. Команда производит анализ указанного файла, вычисляет 16-битную контрольную сумму и отображает число блоков в файле. Например, если записать строку *Paul Watters* в файл *name.txt*, команда *sum* представит такой вывод:

```
bash-2.03$ sum name.txt
1190 1 name.txt
```

Можно видеть, что контрольная сумма равна 1190, а файл занимает единственный блок. Если целостность данных соблюдается, можно (например) скопировать содержимое файла в другой файл и вычислить для копии контрольную сумму:

```
bash-2.03$ cp name.txt name1.txt
bash-2.03$ sum name1.txt
1190 1 name1.txt
```

Если мы отредактируем содержимое *name1.txt*, включив в него указание степени (например), контрольная сумма изменится:

```
bash-2.03$ echo "PhD" >> name1.txt
bash-2.03$ sum name1.txt
1452 1 name.txt
```

Надежность

Надежность характеризует постоянство данных: если файл записан в определенный раздел файловой системы, все последующие операции чтения данных из этого раздела должны приводить к одинаковому результату. Если средство хранения данных ненадежно, операции чтения одного и того же файла могут возвращать различные результаты. И снова можно использовать контрольные суммы для определения эквивалентности файлов, прочитанных несколько раз. Следующий простой сценарий командного интерпретатора может использоваться в целях проверки надежности хранения данных:

```
#!/bin/csh -f
foreach pass (1 2 3 4 5 6 7 8 9 10)
    sum $1
end
```

Сценарий выполняет операцию *sum* для указанного файла (*$1*) десять раз. Если вывод для всех операций содержит одинаковые результаты, хранение данных надежно:

```
bash-2.03# ./checksum.sh name1.txt
1452 1 name.txt
1452 1 name.txt
1452 1 name.txt
1452 1 name.txt
1452 1 name.txt
1452 1 name.txt
1452 1 name.txt
1452 1 name.txt
1452 1 name.txt
1452 1 name.txt
```

В этом примере все полученные контрольные суммы одинаковы. Однако если произошла порча данных раздела, можно увидеть такой результат:

```
bash-2.03# ./checksum.sh name1.txt
1452 1 name.txt
3224 2 name.txt
1452 1 name.txt
1452 1 name.txt
1452 1 name.txt
1452 1 name.txt
1452 1 name.txt
1452 1 name.txt
1452 1 name.txt
1452 1 name.txt
```

В этом примере не все контрольные суммы одинаковы, а значит, хранение данных ненадежно. Опробуйте этот пример для раздела с поврежденными секторами – результаты вас поразят!

Оценка емкостных требований

При проектировании системы одним из самых сложных вопросов становится следующий: «Сколько нужно пространства для хранения данных?» Если речь идет о применении конкретного приложения, стандартные характеристики работы которого известны заранее, достаточно легко предсказать потребность в дисковом пространстве на ближайшее будущее. Но как можно делать предсказания на год вперед? А если система проектируется с нуля, какие принципы можно применять для оценки требований к хранению данных на будущее?

Одним из подходов является разработка математической модели требований к дисковому пространству на основе существующей информации о работе приложений. (Как вариант оценки для исходных показателей могут быть получены по данным аналогичных приложений, если данные нужных приложений недоступны.) Поначалу предсказания, сделанные на основе модели, будут содержать значительную долю ошибок. Однако ряд исторических моментов значительно облегчает предсказательный процесс. Например, существует очень немного операционных систем и приложений, потребности которых с течением лет сокращаются (Solaris 8 поставляется на нескольких компакт-дисках, в то время как SunOS 4 умещалась на одной QIC-ленте).

Когда получены собственно данные и для модели вычислены показатели адекватности, модель может подвергаться подстройке. С каждой подстройкой надежность предсказаний возрастает.

Типичная экспоненциальная модель может иметь формулу *y=_ x bg*. Допустим, исходные потребности в дисковом пространстве составляют 50 Мбайт для первой недели, тогда через двенадцать месяцев экспоненциального роста ожидаемая потребность будет составлять 355 Мбайт, то есть мы получим прирост более 700%! Эта модель показана на рис. 8.1. Экспоненциальный рост является обычным явлением для недавно созданных систем, в которых число пользователей растет в геометрической прогрессии.

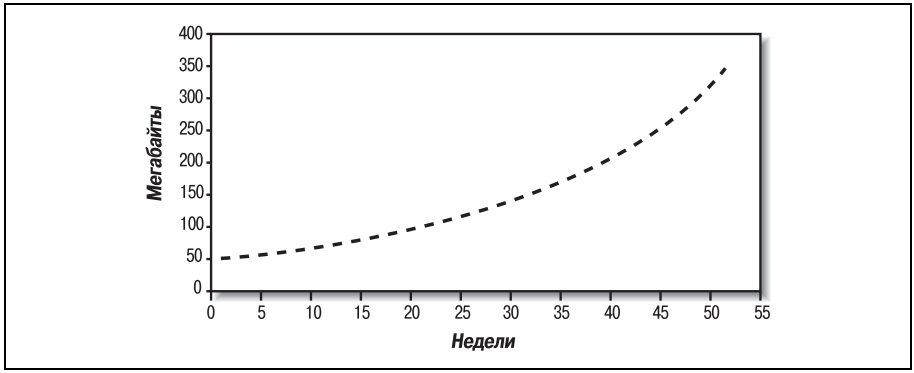

Рис. 8.1. *Математическая модель дисковых потребностей*

Solaris предоставляет инструменты для реализации надежного хранения и целостности данных, но оценка емкостных требований обычно остается задачей несчастного администратора. В отсутствие модели, позволяющей определить разумные размеры дисков и разделов, большинство администраторов выбирают размеры дисков для установки систем фактически наугад. Программа для работы с электронными таблицами, входящая в состав Star Office, является достаточно удобным инструментом моделирования требований к дисковому пространству, поскольку предсказанные значения могут заменяться действительными по мере получения реальных результатов, что позволяет повысить точность предсказаний, сделанных на основе модели.

Решения

В этот момент читатели, вероятно, задаются вопросом, какие же технологии Solaris могут использоваться для обеспечения надежности хранения, целостности и адекватности оценок емкостных требований. Соблюдение целостности данных может быть реализовано с помощью возможностей журналирования UFS, которые появились в Solaris 7, либо с помощью программы проверки файловой системы *fsck*. Традиционно команда *fsck* использовалась для восстановления файловых систем UFS после системных сбоев. Но при включенном журналировании UFS все данные, записываемые в файловую систему, на самом деле попадают сначала в специальный журнал. Таким образом, если операция записи отмечена в журнале, но не выполнена, операции записи игнорируются после перезагрузки системы. Журналирование в UFS консервативно; однако если в журнал попала только часть информации об операции записи, эта информация игнорируется. Это может приводить к потере данных, но позволяет сохранять целостность файловой системы в любых ситуациях. Чтобы включить журналирование для файловой системы UFS, воспользуйтесь ключом *-o logging* команды *mount* (либо в соответствующей записи файла */etc/vfstab*).

Надежность данных и адекватная оценка емкостных потребностей может поддерживаться системой RAID (Redundant Array of Inexpensive Disks, избыточный массив недорогих дисков). RAID-массив состоит из набора самостоятельных или общих дисков (как правило, одинаковой емкости и размера), которые совместно обеспечивают надежный доступ к данным, а также позволяют расширить емкость логического раздела за пределы физических ограничений диска массива.

На некоторых из уровней RAID (RAID уровня 1) реализована поддержка избыточности: отдельные диски проверяются на наличие ошибок и в случае обнаружения таковых могут удаляться и заменяться другими без необходимости останавливать систему (серверы рабочих групп, такие как E450, поддерживают «горячую» замену дисков).[1]

[1] «Горячая» замена позволяет заменять диски без останова системы.

В такой конфигурации (получившей название *зеркалирования*) потеря данных не происходит, поскольку каждый бит данных существует по меньшей мере на двух дисках. Такого рода избыточность гарантирует, что данные всегда могут быть надежно прочитаны из дискового массива, даже если отдельные диски подвержены сбоям.

Зеркалирование сокращает время, затрачиваемое на восстановление утерянных или испорченных данных, поскольку этот процесс протекает совершенно безболезненно и прозрачно. Если сервер не поддерживает зеркалирование, восстановление утерянных или испорченных файлов будет связано со считыванием с медленных ленточных накопителей и, возможно, с необходимостью останавливать работу сервера. Кроме того, единственный испорченный файл может оказывать пагубное влияние на работу многих приложений. Например, существование сбойного файла драйвера устройства будет иметь серьезные последствия для приложений и служб, которые попытаются воспользоваться этим устройством. Разумеется, цикл резервного копирования/восстановления от этого не становится менее полезным; на деле при серьезных сбоях аппаратного обеспечения полное восстановление из образа системы, сделанного днем ранее, может оказаться последней надеждой на приведение системы в нормальное состояние.

Кроме того, RAID-массивы поддерживают логическое распространение дисков за пределы физических объемов; эта возможность получила название *расслоения* логических дисков (RAID уровня 0). Объемы отдельных физических дисков могут объединяться с целью создания одного виртуального диска повышенной емкости. Такая возможность очень важна для приложений баз данных, поскольку таблицы и журналы отката могут вырастать до очень больших размеров: эти файлы нелегко хранить на нескольких разделах, продолжая при этом обработку запросов в режиме реального времени. Однако с появлением дисков большого объема (от 60 Гбайт) расслоение может утратить свою важность для утилизации максимально-возможных объемов дискового пространства.

Существует шесть различных уровней RAID, поддерживающих расслоение и зеркалирование в многочисленных вариантах. Так, RAID уровня 1 поддерживает только зеркалирование, а RAID уровня 4 нацелены на поддержку зеркалирования и сокращения общего объема потребляемого дискового пространства. Различия между уровнями RAID в Solaris приведены в табл. 8.1.

В Solaris реализована поддержка как программных, так и аппаратных RAID-решений. Например, RAID-устройство StorEdge A1000 реализует полноценную избыточность посредством нескольких уровней RAID (0, 1, 1+0, 5), а также обеспечивает аппаратную избыточность с помощью независимых шин SCSI, раздельного питания и охлаждения. Эти дополнительные компоненты делают систему максимально защищенной от отказов при работе в режиме реального времени. Базовые кон-

фигурации A1000 содержат 16 Гбайт дискового пространства; максимально-допустимый объем – 109 Гбайт. Кроме того, A1000 может являться частью кластера, работающего под управлением пакета Sun Cluster 2.2, и управляться с SNMP-консоли.

Таблица 8.1. Уровни RAID, поддерживаемые в Solaris; зеркалирование и расслоение

Уровень RAID	Описание
0	Записывает данные на несколько устройств (расслоение) с целью повышения емкости логических дисков и общей производительности
1	Все данные записываются по меньшей мере на два диска (зеркалирование), что значительно снижает скорость записи, но значительно повышает надежность
2	Реализует коррекцию ошибок зеркалирования с помощью кодов Хэмминга
3	Записывает данные на несколько устройств, а информацию о четности – на единственный диск
4	Записывает блоки данных на несколько устройств, а информацию о четности – на единственный диск
5	Записывает блоки данных на несколько устройств, а информацию о четности – на несколько дисков

Для сборки нескольких обычных жестких дисков в массив RAID может применяться программа Solstice DiskSuite. DiskSuite реализует расслоение и избыточность, а кроме того, позволяет производить «горячую» замену дисков в случаях, когда это допускается контроллером (SCSI для SPARC, PCI для Intel). DiskSuite использует стандартные UFS-диски и не требует наличия специального аппаратного контроллера, могут параллельно применяться и другие методы повышения отказоустойчивости (например, журналирование UFS). Журналы UFS тоже могут распределяться по различным файловым системам в целях защиты операций записи UFS. DiskSuite также совместим с SNMP.

Дополнением к технологии RAID являются возможности «горячей» замены, реализуемые системой Solaris. В Solaris «горячая» замена поддерживается с помощью динамической перенастройки архитектур SCSI и Intel PCI. Серверы, поддерживающие «горячую» замену, могут работать без перезагрузки: сбоящее устройство просто удаляется из системы и заменяется другим. Для плавного перехода применяется команда *cfgadm*. Воспользуйтесь *cfgadm*, чтобы определить гнездо, в котором расположено устройство:

```
# cfgadm
Ap_Id Receptacle Occupant Condition
pci0:hpc1_slot0 connected configured ok
```

```
pci0:hpc1_slot1 empty unconfigured unknown
pci0:hpc1_slot2 empty unconfigured unknown
pci0:hpc1_slot3 empty unconfigured unknown
pci0:hpc1_slot4 empty unconfigured unknown
```

Мы можем видеть, что в гнезде 0 расположено единственное устройство PCI. Чтобы отключить устройство, выполните команду:

```
# cfgadm -c unconfigure pci0:hpc1_slot1
```

Убедитесь, что устройство отключено:

```
# cfgadm
Ap_Id Receptacle Occupant Condition
pci0:hpc1_slot0 disconnected unconfigured unknown
pci0:hpc1_slot1 empty unconfigured unknown
pci0:hpc1_slot2 empty unconfigured unknown
pci0:hpc1_slot3 empty unconfigured unknown
pci0:hpc1_slot4 empty unconfigured unknown
```

Устройство в гнезде 0 теперь диагностируется как отключенное. После физического удаления карты PCI можно вставить заменяющую карту. Для перенастройки используется *cfgadm*. После замены карты вывод команды *cfgadm* будет таким:

```
# cfgadm
Ap_Id Receptacle Occupant Condition
pci0:hpc1_slot0 disconnected unconfigured unknown
pci0:hpc1_slot1 empty unconfigured unknown
pci0:hpc1_slot2 empty unconfigured unknown
pci0:hpc1_slot3 empty unconfigured unknown
pci0:hpc1_slot4 empty unconfigured unknown
```

Чтобы подключить устройство, выполните команду:

```
# cfgadm -c configure pci0:hpc1_slot0
Ap_Id Receptacle Occupant Condition
pci0:hpc1_slot0 connected configured ok
```

Если перенастройка произошла успешно, мы должны увидеть, что карта стала частью системы PCI:

```
# cfgadm
Ap_Id Receptacle Occupant Condition
pci0:hpc1_slot0 connected configured ok
pci0:hpc1_slot1 empty unconfigured unknown
pci0:hpc1_slot2 empty unconfigured unknown
pci0:hpc1_slot3 empty unconfigured unknown
pci0:hpc1_slot4 empty unconfigured unknown
```

Аналогичный подход применяется для перенастройки устройств SCSI, таких как жесткие диски, с той разницей, что вместо адресов устройств PCI появляются адреса и имена устройств SCSI.

Как только управление хранением данных выходит за пределы одной системы, более подходящим решением может стать Storage Area Network (SAN). Основанная на открытом API Jiro для технологии распределенных агентов, система Sun SAN включает специализированные аппаратные решения вроде T3, которые способны работать с системами Solaris, HP-UX, Linux и некоторыми версиями Microsoft Windows. T3 обеспечивает возможность хранения гигантских объемов данных, от 327 Гбайт до 169 Тбайт, а также поддерживает различные уровни RAID. Управление хранением организовано на базе семейства инструментов StorEdge, включая Sun StorEdge Network Data Replicator и Sun StorEdge Instant Image.

Инструменты контроля версий

Стандартным инструментом контроля версий для Solaris является SCCS. SCCS обычно применяется разработчиками, которые желают хранить историю всех изменений, вносимых в исходные файлы приложения. Помимо этого, многие системные администраторы применяют SCCS для создания истории изменений ключевых файлов системы, таких как */etc/passwd*, */etc/shadow* и файлов из каталога *init.d*. Начинающие разработчики отслеживают изменения в исходных текстах, создавая каталоги с указанием даты (например, *source_2/2/01*) или tar-файлы (например, *source_2_2_01.tar*). И хотя в этом случае разработчикам в любое время доступны предшествующие версии исходных текстов, издержки могут быть довольно значительными, когда за несколько дней изменяется лишь один из многих файлов. Кроме того, применение такой системы управления версиями оставляет возможность того, что выбранная версия не будет рабочей, тогда как более строгий подход SCCS рекомендует помещение файлов в архив только в том случае, если они на самом деле работают. Кроме того, каждая версия (или «дельта») может быть отмечена комментарием, отражающим задачи разработки (например, «Исправлена ошибка 136»). Очевидно, одной системы SCCS недостаточно для навязывания строгого контроля версий, поскольку она всего лишь является средством поддержки для правил обработки найденных ошибок и циклов разработки.

Системные администраторы применяют подобный подход при внесении изменений в ключевые файлы, такие как файл паролей, когда могут понадобиться предыдущие версии. Существует две ситуации, когда это может быть полезным. Во-первых, если файл паролей будет запорчен, наиболее поздняя его версия (или любая из сохраненных в репозитории) может быть с легкостью восстановлена. Это избавляет от необходимости выполнять резервное копирование, которое отнимает время и силы (особенно, если вы не способны по памяти использовать команды *ufsrestore*!). Во-вторых, применение контроля версий для файлов настройки позволяет производить ревизии, что весьма полезно для диагностирования проблем неработающих настроек, поскольку их

предыдущая версия всегда доступна для анализа. Например, если один из администраторов системы изменил файл таким образом, что это привело к проблемам (скажем, к сбою сети), ответственного за эти действия человека можно легко выявить, обратившись к истории версий SCCS. Базовые возможности SCCS – поддержка доступа к репозиторию многих пользователей и отслеживание всех изменений, вносимых в исходные файлы.

Команда, предоставляющая доступ к репозиторию SCCS, называется (что неудивительно) *sccs*. Основные команды работы с SCCS приведены в табл. 8.2.

Таблица 8.2. Основные команды SCCS

Команда	Описание
clean	Очищает текущий рабочий каталог
create	Инициализирует файлы истории проекта
deledit	Вносит изменения и извлекает их из репозитория (чтение/запись)
delget	Вносит изменения и извлекает их из репозитория (чтение/запись)
delta	Только вносит изменения
diffs	Сравнивает рабочий файл с версией из архива
edit	Извлекает файл (чтение/запись)
enter	Инициализирует файлы истории проекта
get	Извлекает файл (только чтение)
info	Отображает полную информацию по извлеченным файлам
tell	Перечисляет извлеченные файлы
unedit	Производит откат для последнего изменения файла, помещенного в репозиторий

SCCS может применяться для архивирования ключевых файлов настройки системы, таких как */etc/passwd*, с целью ведения истории изменений. Однако при создании архивов чувствительных файлов (таких как */etc/shadow*) следует проявлять осторожность, чтобы предотвратить доступ к архиву непривилегированных пользователей.

Первым шагом в инициализации репозитория SCCS является создание подкаталога SCCS в текущем рабочем каталоге, где расположены исходные файлы:

```
bash-2.03# mkdir SCCS
```

Затем необходимо создать файл истории проекта с помощью ключевого слова *create* (скажем, для файла */etc/passwd*):

```
bash-2.03# sccs create passwd
```

Выполнение этой команды приводит к получению следующей информации:

```
passwd:
No id keywords (cm7)
1.1
6 lines
No id keywords (cm7)
```

Текущей версии шестистрочного файла */etc/passwd* присвоен номер 1.1. Чтобы изменить содержимое файла */etc/passwd*, следует извлечь его из архива с помощью команды *edit*:

```
bash-2.03# sccs edit passwd
```

Мы получим такой результат:

```
1.1
new delta 1.2
6 lines
```

Программа SCCS присвоила новый номер версии (1.2) данному файлу. После извлечения файл можно редактировать. Внести изменения в архив можно при помощи команды *deledit*:

```
bash-2.03# sccs deledit passwd
```

Результат:

```
comments? Меня изменили
No id keywords (cm7)
1.2
1 inserted
0 deleted
6 unchanged
1.2
new delta 1.3
```

Здесь можно ввести комментарий, описывающий внесенные в файл изменения (в данном случае я добавил одну строку в файл */etc/passwd*, а в качестве комментария использовал строку «Меня изменили»).

Файл истории проекта для файла */etc/passwd* сохраняется под именем */etc/SCCS/s.passwd*. Мы можем изучить, как SCCS хранит историю, взглянув на заголовок файла */etc/SCCS/s.passwd*:

```
bash-2.03# head SCCS/s.passwd
```

Вывод команды:

```
_s 00001/00000/00006
_d D 1.2 01/02/21 14:05:37 root 2 1
_c Меня изменили
```

```
_e
_s 00006/00000/00000
_d D 1.1 01/02/21 14:03:58 root 1 0
_c date and time created 01/02/21 14:03:58 by root
_e
_u
_U
_f e 0
_t
_T
_I 1
```

Можно наблюдать время и дату внесенных изменений, а также число измененных строк для каждой версии. Кроме того, регистрируется автор изменений (*root*). Чтобы отобразить полную историю версий, воспользуйтесь командой *prs*:

```
bash-2.03# prs SCCS/s.passwd
SCCS/s.passwd:

D 1.2 01/02/21 14:23:07 root 2 1        00000/00001/00006
MRs:
COMMENTS:
no comment at this time

D 1.1 01/02/21 14:20:36 root 1 0        00007/00000/00000
MRs:
COMMENTS:
date and time created 01/02/21 14:20:36 by paul
```

Применение *prs* (вместо прямого просмотра файла истории) облегчает интерпретацию истории дельт.

SCCS не ограничивается обработкой отдельных файлов; с помощью масок можно создавать файлы историй версий для наборов файлов, существующих в одном каталоге:

```
bash-2.03# sccs create passwd nsswitch.conf group

SCCS:
ERROR: directory `SCCS' specified as `i' keyletter value (ad29)

passwd:
No id keywords (cm7)

nsswitch.conf:
No id keywords (cm7)

group:
No id keywords (cm7)

SCCS/s.passwd:
1.1
7 lines
No id keywords (cm7)
```

```
SCCS/s.nsswitch.conf:
1.1
10 lines
No id keywords (cm7)

SCCS/s.group:
1.1
2 lines
No id keywords (cm7)
```

Повторимся, применение *prs* облегчает интерпретацию истории дельт для всех хранимых файлов:

```
bash-2.03# prs *
s.passwd:

D 1.1 01/02/21 14:20:36 root 1 0        00007/00000/00000
MRs:
COMMENTS:
date and time created 01/02/21 14:20:36 by root

s.nsswitch.conf:

D 1.1 01/02/21 14:20:37 root 1 0        00010/00000/00000
MRs:
COMMENTS:
date and time created 01/02/21 14:20:37 by root

s.group:

D 1.1 01/02/21 14:20:37 root 1 0        00002/00000/00000
MRs:
COMMENTS:
date and time created 01/02/21 14:20:37 by root
```

SCCS умеет создавать файл истории для любого текстового (не бинарного) файла. После создания истории проекта большинство администраторов производят резервное копирование файлов SCCS на устройства большой емкости, вроде ленточных накопителей, пользуясь одним из методов, рассмотренных в следующем разделе.

Резервное копирование

Резервные копии являются краеугольным камнем стратегий быстрого восстановления систем в случае серьезных аппаратных сбоев или ошибок администраторов. Создание резервных копий – не единственная методика обеспечения надежного обслуживания; зеркалирование RAID позволяет выполнять резервное копирование данных в реальном времени, а контроль версий с помощью SCCS позволяет быстро восстанавливать предшествующие версии файлов настройки. Резервные копии обычно являются последним средством, к которому прибегают в крайнем случае. Именно поэтому администраторы ревностно берегут ре-

зервные копии и хранят их подальше от технологической площадки – это сокращает риск использования носителя с резервной копией для каких-либо посторонних целей.

Как правило, резервное копирование выполняется в определенное время с помощью планировщика задач, что позволяет четко привязывать резервные данные к конкретному моменту во времени. Очень важно знать точно, в какой момент была сделана резервная копия, поскольку дефект или иного рода порча может появиться незаметно. По тем же причинам важно сохранять резервные копии, связанные с ключевыми моментами жизни системы. Например, полное ежемесячное копирование полного образа весьма полезно для ревизий и архивного хранения с целью последующего быстрого восстановления.

Резервное копирование может быть долгим и утомительным процессом, если применяется устаревшая технология, в особенности QIC-ленты. В частности, если используется специальный сервер резервного копирования с ленточным накопителем, копирование файлов по сети (по протоколам NFS или Samba) может значительным образом снизить скорость обмена данными для всех остальных пользователей этой сети. Поэтому резервное копирование часто выполняется в полночь или позже, в моменты минимальной загрузки сети. Выбор для резервного копирования носителя с небольшим временем доступа позволит минимизировать время выполнения операций.

Другой момент связан с действительным состоянием файловой системы в момент создания резервной копии. В идеальном варианте не должно быть открытых файлов и активных пользователей (кроме администратора). Это гарантирует, что файлы будут записаны в том виде, в каком были сохранены пользователями в течение дня. Если пользователь внес изменения в один из своих документов, но не сохранил их, а кроме того не завершил сеанс работы на момент резервного копирования, эти изменения будут отражены только в резервной копии следующего дня. Если сбой в системе произойдет раньше, чем следующее резервное копирование, все изменения будут потеряны.

Здесь следует сделать предупреждение относительно файлов баз данных. Большинство СУБД (таких как Oracle) позволяют выбирать между «холодным» и «горячим» образами при останове сервера баз данных. Если речь идет о высокой доступности, может потребоваться создание «горячего» образа. Для целей анализа следует остановить сервер баз данных, выполнить создание «холодного» образа, а затем снова запустить сервер БД до начала резервного копирования. Это позволяет с минимальным временем ежедневных простоев всегда быть в курсе, откуда взялись данные и каким числом датированы.

Существует две базовые стратегии создания резервных копий. Первая и самая простая – создание полного образа, которое связано с копированием всех выбранных файлов на устройство резервного копирования. Преимущество полного образа состоит в том, что одна или не-

сколько лент содержат все данные, которые может потребоваться восстановить. Минусом является необходимость использовать новый ленточный носитель (или набор носителей) для каждого полного образа. Некоторые администраторы считают, что ежедневное создание копий ценных файлов стоит потраченного времени и пространства ленты; в конце концов, корпоративные данные стоят гораздо дороже.

Более эффективная (и более рискованная) стратегия – использовать инкрементальные резервные копии. Цикл начинается с полного образа, а затем резервное копирование производится только для файлов, которые за определенный период подверглись изменению. Таким образом, если в один из дней изменения коснулись лишь нескольких файлов, резервное копирование будет выполнено практически моментально. Если в системе есть данные, которые изменяются постоянно (образ базы данных), применение инкрементальных копий может оказаться не самой подходящей стратегией. Одна из существующих проблем связана с тем, что необходимо хранить все ленты последовательности инкрементальных образов. Если цикл для инкрементальных образов занимает одну неделю, то понадобится семь лент – одна для полного образа, сделанного в начале недели, и по одной для инкрементальных образов на каждый из оставшихся дней недели. Утеря одной из лент ставит под угрозу существование всей серии, потому что нет возможности определить, какие именно данные пропали. Поэтому некоторые организации предпочитают создавать полные или инкрементальные образы по меньшей мере на двух системах с применением различных видов носителей, чтобы достичь отказоустойчивой избыточности. Администратор также может создать дополнительную систему резервного копирования, расположенную вне основной технологической площадки и доступную только в контексте частной виртуальной сети или аналогичного безопасного соединения.

Обратной стороной цикла резервного копирования является восстановление. Оно связано с переносом файлов с резервного носителя в ту точку, из которой они были скопированы. Все права владения, групповые ассоциации и права доступа к файлам должны воссоздаваться в том виде, в каком существовали на момент создания образа. Для качественной системы резервного копирования это означает, что процесс восстановления незаметен для пользователя; данные восстановленной файловой систем должны выглядеть точно так же, как в момент создания резервной копии.

Восстановление также бывает двух видов: полное и частичное. Полное восстановление означает восстановление всех файлов с определенного набора носителей и запись их на жесткий диск, с которого они были прочитаны. Полное восстановление чаще всего применяется в случае серьезных аппаратных сбоев в системе; после замены диска-нарушителя данные могут быть восстановлены с наиболее позднего набора резервных носителей.

Частичное восстановление часто применяется для восстановления набора файлов, случайно удаленных пользователями (или суперпользователями!). Эта проблема часто возникает у тех, кто игнорирует системы контроля версий (такие как SCCS, описанная ранее) и пользуется командами вроде *rm* * в своих домашних каталогах либо каталогах с рабочими файлами (например, *~/src*). В таких случаях в зависимости от используемого ПО резервного копирования выбор файлов и/или каталогов, подлежащих восстановлению, не должен вызывать сложностей.

Выбор носителей для резервного копирования

Выбор подходящих носителей для конкретной системы должен основываться на трех критериях: скорости, емкости и совместимости с ПО резервного копирования. Выбор носителей с высокой скоростью записи означает, что процесс копирования и восстановления будет занимать минимально возможное время. Это минимизирует неудобства для пользователей, поскольку загрузка системы возрастает при передаче данных с исходного жесткого диска на целевое устройство. Емкость – также очень важный момент: дисковый накопитель работает очень быстро, но его емкость может быть меньше, чем у одной цифровой ленты. Сокращение числа лент, необходимых для выполнения резервного копирования, облегчает администрирование. Для площадок с большими требованиями к объемам информации может быть уместно применение автоматизированных библиотек лент. Какое бы вы ни выбрали устройство, оно должно быть совместимо с используемым программным обеспечением. Как правило, в Solaris проблем подобной совместимости не возникает; если устройство поддерживается системой, все основные инструменты, такие как *tar*, *dd* и *cpio*, смогут производить запись.

В следующих разделах я рассмотрю некоторые из наиболее распространенных устройств и носителей для резервного копирования, а также подготовку носителей к использованию.

Ленточные накопители

Ленты остаются наиболее популярным видом носителей для резервного копирования в Solaris. Solaris поддерживает ленты разнообразных форматов (1/4", 1/2", 4 мм и 8 мм) и протоколов (DAT и DLT), а также различные накопители (аналоговые и цифровые). В большинстве систем Solaris сегодня применяются цифровые ленточные накопители DAT и DLT, поскольку их емкость варьируется от 4 Гбайт (лента DDS-2 и накопитель DAT) до 70 Гбайт (DLT).

Жесткие диски

Хотя жесткие диски обычно считаются основным типом носителей для хранения данных, снижение стоимости гигабайта в последние годы сделало резервное копирование на жесткие диски реальностью. В частности, система резервного копирования может быть построена на основе технологии RAID: файлы из основной RAID-системы могут копироваться в логическую файловую систему второго массива RAID. Это означает, что для резервных данных может выполняться как расслоение, так и зеркалирование. Дополнительное преимущество дисковых копий: некоторые приложения (например, серверы баз данных) способны производить зеркалирование дисков (с основного на дополнительный) в реальном времени. Если произойдет сбой в работе основного диска, сервер баз данных сможет использовать в качестве основного дополнительный диск. Для организации подобной отказоустойчивости жесткие диски выглядят явными фаворитами. Диски не столь мобильны, как ленты, а ленты по-прежнему стоят гораздо дешевле.

Приводы Zip/Jaz

Приводы Zip и Jaz также поддерживаются системой Solaris. Приводы SCSI работают с Solaris Intel и SPARC; приводы IDE поддерживаются только платформой Solaris Intel. Одним из главных преимуществ носителей Zip и Jaz является возможность прочитать и/или изменить данные на машине, работающей под управлением Microsoft Windows или Mac OS. И хотя для этих целей могут применяться диски CD-RW, носители Zip и Jaz являются более надежными в плане операций чтения/записи на различных платформах. Носители Jaz в настоящее время достигают емкости в 2 Гбайт, что делает их идеальным выбором для резервного копирования небольших объемов данных.

Приводы CD-R

В качестве носителей для резервного копирования ленты находятся под давлением со стороны технологии CD-R в области защиты данных от записей. Компакт-диск имеет емкость всего 650 Мбайт – этого недостаточно, чтобы уместить хотя бы один раздел любой из современных систем. При этом носители CD-R обладают преимуществом: после записи данные могут быть только прочитаны. Это предотвращает случайное или преднамеренное изменение сохраненных данных. Из соображений безопасности при хранении данных очень важно позаботиться о том, чтобы они не были случайно или преднамеренно удалены или перезаписаны.

Методы резервного копирования и восстановления

В Solaris существует несколько различных методов резервного копирования и восстановления данных: от стандартного формата «ленточных архивов» (*tar*-файлы) до инструментов низкоуровневого копирования (*dd* и *cpio*). Кроме того, Solaris поддерживает формат образов UFS, что значительно облегчает выполнение частичного и полного восстановления. В этом разделе мы рассмотрим выполнение частичного и полного резервного копирования и восстановления посредством существующих методов.

tar

Файлы ленточных архивов чаще всего применяются для создания резервных копий данных. Их можно записывать напрямую на устройства без файловых систем (накопители на магнитных лентах) либо в файлы для последующего копирования и/или распространения. В отличие от дисков, которые необходимо форматировать перед использованием, ленты, созданные с помощью *tar*, не требуют предварительного форматирования. Это сокращает затраты времени на создание резервных копий. Самым большим недостатком *tar* является ограниченность архива одной лентой; если общий объем данных системы превышает максимальную емкость носителя, придется использовать другой инструмент (или купить другой накопитель). Тем не менее *tar* является хорошим межплатформенным инструментом, который вполне можно применять, если помнить о проверке соответствия объемов.

Первым шагом при создании ленточного архива является определение каталогов высшего уровня (или точек монтирования), соответствующих файловым системам, которые следует архивировать. Чтобы создать ленточный архив установленной версии системы WebNFS и записать его прямо на магнитную ленту, воспользуйтесь командой:

```
bash-2.03# tar cvf /dev/rmt0 *
a api.html 15K
a classes.zip 80K
a faq.html 14K
a index.html symbolic link to overview.html
a javadoc/ 0K
a javadoc/images/ 0K
a javadoc/images/GridBagEx.gif 3K
a javadoc/images/OpenBookIcon.gif 3K
a javadoc/images/blue-ball-small.gif 1K
a javadoc/images/blue-ball.gif 1K
a javadoc/images/class-index.gif 2K
a javadoc/images/constrct.gif 2K
a javadoc/images/constructor-index.gif 2K
```

```
a javadoc/images/constructors.gif 2K
a javadoc/images/cyan-ball-small.gif 1K
a javadoc/images/cyan-ball.gif 1K
a javadoc/images/error-index.gif 2K
a javadoc/images/exception-index.gif 2K
a javadoc/images/field_ix.gif 2K
a javadoc/images/fields.gif 2K
a javadoc/images/green-ball-small.gif 1K
a javadoc/images/green-ball.gif 1K
a javadoc/images/interface-index.gif 2K
a javadoc/images/magenta-ball-small.gif 1K
a javadoc/images/magenta-ball.gif 1K
a javadoc/images/method-index.gif 2K
a javadoc/images/methods.gif 2K
a javadoc/images/package-index.gif 2K
a javadoc/images/red-ball-small.gif 1K
a javadoc/images/red-ball.gif 1K
a javadoc/images/variable-index.gif 2K
a javadoc/images/variables.gif 2K
a javadoc/images/yellow-ball-small.gif 1K
a javadoc/images/yellow-ball.gif 1K
a javadoc/AllNames.html 26K
a javadoc/Package-com.sun.nfs.html 1K
a javadoc/Package-com.sun.xfile.html 2K
a javadoc/com.sun.nfs.nfsXFileExtensionAccessor.html 4K
a javadoc/com.sun.xfile.XFile.html 19K
a javadoc/com.sun.xfile.XFileExtensionAccessor.html 3K
a javadoc/com.sun.xfile.XFileInputStream.html 9K
a javadoc/com.sun.xfile.XFileOutputStream.html 9K
a javadoc/com.sun.xfile.XFileReader.html 3K
a javadoc/com.sun.xfile.XFileWriter.html 3K
a javadoc/com.sun.xfile.XFilenameFilter.html 3K
a javadoc/index.html 1K
a javadoc/com.sun.xfile.XRandomAccessFile.html 34K
a javadoc/packages.html 1K
a javadoc/tree.html 2K
a nfsclasses.html 27K
a overview.html 5K
a probrep.html 1K
a README 1K
a release.html 3K
a sample/ 0K
a sample/rcopy.java 2K
a sample/xcopy.java 1K
```

Аргументы, переданные команде *tar*, включают строку букв *cvf*, которые предписывают создать архив из одного файла с подробной диагностикой по файлам и записать архив на устройство */dev/rmt0*. Подробная диагностика содержит параметры каждого из файлов, добавляемых в ленточный архив, включая относительный путь к файлу и его приблизительный размер в килобайтах.

Чтобы восстановить данные с носителя резервной копии, просто воспользуйтесь такой командой:

```
bash-2.03# tar xvf /dev/rmt0 *
x api.html, 14833 bytes, 29 tape blocks
x classes.zip, 81458 bytes, 160 tape blocks
x faq.html, 14307 bytes, 28 tape blocks
x index.html symbolic link to overview.html
x javadoc, 0 bytes, 0 tape blocks
x javadoc/images, 0 bytes, 0 tape blocks
x javadoc/images/GridBagEx.gif, 2453 bytes, 5 tape blocks
x javadoc/images/OpenBookIcon.gif, 2241 bytes, 5 tape blocks
x javadoc/images/blue-ball-small.gif, 255 bytes, 1 tape blocks
x javadoc/images/blue-ball.gif, 925 bytes, 2 tape blocks
x javadoc/images/class-index.gif, 1497 bytes, 3 tape blocks
x javadoc/images/constrct.gif, 1565 bytes, 4 tape blocks
x javadoc/images/constructor-index.gif, 1711 bytes, 4 tape blocks
x javadoc/images/constructors.gif, 1565 bytes, 4 tape blocks
x javadoc/images/cyan-ball-small.gif, 255 bytes, 1 tape blocks
x javadoc/images/cyan-ball.gif, 925 bytes, 2 tape blocks
x javadoc/images/error-index.gif, 1438 bytes, 3 tape blocks
x javadoc/images/exception-index.gif, 1707 bytes, 4 tape blocks
x javadoc/images/field_ix.gif, 1443 bytes, 3 tape blocks
x javadoc/images/fields.gif, 1241 bytes, 3 tape blocks
x javadoc/images/green-ball-small.gif, 102 bytes, 1 tape blocks
x javadoc/images/green-ball.gif, 886 bytes, 2 tape blocks
x javadoc/images/interface-index.gif, 1648 bytes, 4 tape blocks
x javadoc/images/magenta-ball-small.gif, 104 bytes, 1 tape blocks
x javadoc/images/magenta-ball.gif, 896 bytes, 2 tape blocks
x javadoc/images/method-index.gif, 1588 bytes, 4 tape blocks
x javadoc/images/methods.gif, 1403 bytes, 3 tape blocks
x javadoc/images/package-index.gif, 1607 bytes, 4 tape blocks
x javadoc/images/red-ball-small.gif, 255 bytes, 1 tape blocks
x javadoc/images/red-ball.gif, 527 bytes, 2 tape blocks
x javadoc/images/variable-index.gif, 1576 bytes, 4 tape blocks
x javadoc/images/variables.gif, 1380 bytes, 3 tape blocks
x javadoc/images/yellow-ball-small.gif, 255 bytes, 1 tape blocks
x javadoc/images/yellow-ball.gif, 925 bytes, 2 tape blocks
x javadoc/AllNames.html, 25974 bytes, 51 tape blocks
x javadoc/Package-com.sun.nfs.html, 595 bytes, 2 tape blocks
x javadoc/Package-com.sun.xfile.html, 1224 bytes, 3 tape blocks
x javadoc/com.sun.nfs.nfsXFileExtensionAccessor.html, 3500 bytes, 7 tape blocks
x javadoc/com.sun.xfile.XFile.html, 19399 bytes, 38 tape blocks
x javadoc/com.sun.xfile.XFileExtensionAccessor.html, 2071 bytes, 5 tape blocks
x javadoc/com.sun.xfile.XFileInputStream.html, 8331 bytes, 17 tape blocks
x javadoc/com.sun.xfile.XFileOutputStream.html, 8420 bytes, 17 tape blocks
x javadoc/com.sun.xfile.XFileReader.html, 2608 bytes, 6 tape blocks
x javadoc/com.sun.xfile.XFileWriter.html, 2983 bytes, 6 tape blocks
x javadoc/com.sun.xfile.XFilenameFilter.html, 2150 bytes, 5 tape blocks
x javadoc/index.html, 630 bytes, 2 tape blocks
x javadoc/com.sun.xfile.XRandomAccessFile.html, 34525 bytes, 68 tape blocks
```

```
x javadoc/packages.html, 630 bytes, 2 tape blocks
x javadoc/tree.html, 1685 bytes, 4 tape blocks
x nfsclasses.html, 26977 bytes, 53 tape blocks
x overview.html, 4109 bytes, 9 tape blocks
x probrep.html, 690 bytes, 2 tape blocks
x README, 570 bytes, 2 tape blocks
x release.html, 2608 bytes, 6 tape blocks
x sample, 0 bytes, 0 tape blocks
x sample/rcopy.java, 1290 bytes, 3 tape blocks
x sample/xcopy.java, 737 bytes, 2 tape blocks
```

И снова отображаются параметры каждого обработанного файла, включая относительный путь, число байтов, а также число блоков ленты, отведенных для хранения файла. Основные ключи команды *tar* описаны в табл. 8.3.

Таблица 8.3. Основные ключи команды tar

Ключ	Описание
b	Определить размера блока
c	Создать новый архив из указанных файлов
e	Прервать создание архива, если произошла ошибка
E	Запись имен файлов в расширенном формате
i	Продолжать работу при обнаружении ошибок в контрольных суммах
k	Указание размера архива, позволяет преодолеть ограничение «один архив на одну ленту»
p	Восстановить права доступа и список управления доступом к файлам
r	Добавить указанные файлы в архив
t	Перечислить файлы, хранимые в указанном архиве
u	Обновить содержимое архива для указанных файлов
x	Извлечь указанные файлы из архива

Ленточные архивы не ограничены прямой записью на устройства без файловых систем; команда *tar* может использоваться для создания переносимых архивов в локальной файловой системе. Это облегчает задачу создания архивов разнообразных файлов с целью обмена данными между системами. *tar*-файлы часто подвергаются сжатию стандартным инструментом Solaris *compress* либо одним из более эффективных инструментов проекта GNU (скажем, *gzip*). Чтобы создать архив, содержащий все файлы из каталога WebNFS, воспользуйтесь командой:

```
bash-2.03# tar cvf /home/paul/webnfs.tar *
```

Результатом будет следующий файл:

```
bash-2.03# ls -l
-rw-r--r--   1 paul     other     339968 Mar  2 14:30 webnfs.tar
```

Чтобы извлечь содержимое архива на диск, воспользуйтесь командой:

```
bash-2.03# tar xvf /home/paul/webnfs.tar
```

Чтобы сэкономить место в локальной файловой системе, сожмите архив командой *compress*:

```
bash-2.03# compress webnfs.tar; ls -l
-rw-r--r--   1 paul     other     259387 Mar  2 14:30 webnfs.tar.Z
```

Часто лучших результатов сжатия можно добиться с помощью программы *gzip*:

```
bash-2.03# gzip webnfs.tar; ls -l
-rw-r--r--   1 paul     other     131869 Mar  2 14:30 webnfs.tar.gz
```

dd

Инструмент *dd* позволяет копировать блоки неструктурированных данных произвольного размера с одного устройства на другое (в противоположность созданию файловых архивов командой *tar*). Работа с неструктурированными данными позволяет копировать отдельные разделы в файлы других файловых систем либо на ленту. Такие возможности весьма полезны при зеркалировании целых файловых систем, особенно в случаях, когда важно сохранить порядок следования данных на разделе. Например, если запись на дублирующий диск базы данных производится каждую ночь, сохранение целостности данных – вплоть до уровня отдельных байтов – имеет особую важность. *dd* является великолепным инструментом для решения подобных задач. Кроме того, если существует вторая лента с создаваемым еженедельно полным образом, *dd* позволяет администратору сделать первую копию прямо с файловой системы, затем скопировать содержимое основной ленты, блок за блоком, на дополнительную. Это освобождает дисковое пространство для нужд других приложений, по-прежнему обеспечивая избыточность. Чтобы скопировать содержимое носителя с первого накопителя (*/dev/rmt0*) на второй (*/dev/rmt2*), воспользуйтесь командой:

```
bash-2.03# dd if=/dev/rmt/0h  of=/dev/rmt/2h
```

Чтобы скопировать содержимое ленты в файлы на диске, можно использовать комбинацию *dd* и операторов перенаправления:

```
bash-2.03# dd < /dev/rdsk/c0t1d2s1 > /home/paul/c0t1d2s1.dd
```

Файловая система */dev/rdsk/c0t1d2s1* поблочно копируется в файл */home/paul/c0t1d2s1.dd*. Чтобы скопировать файловую систему на ленту, выполните такую команду:

```
bash-2.03# dd if=/dev/rdsk/c0t1d2s1 of=/dev/rmt/0h
```

Содержимое */dev/rdsk/c0t1d2s1* копируется на основной накопитель (*/dev/rmt/0h*). Подобно *tar*, команда *dd* имеет ряд ключей, изменяющих стандартное поведение. Эти параметры перечислены в табл. 8.4.

Таблица 8.4. Основные ключи команды dd

Ключ	Описание
bs	Определяет размер блока конечного файла
cbs	Определяет размер буфера преобразования (операция преобразования определяется ключом *conv*)
conv	Предписывает выполнение преобразования данных в процессе копирования; включает такие варианты преобразования, как ASCII в EBCDIC (*ibm*), нижнего регистра в верхний (*ucase*), а также верхнего регистра в нижний (*lcase*)
count	Предписывает копирование указанного числа блоков из исходного файла в конечный; размер исходного файла при этом не учитывается
if	Определяет имя исходного файла
of	Определяет имя конечного файла
skip	Указывает число блоков, которые следует пропустить в исходном файле перед началом копирования

cpio

Инструмент *cpio* схож с программой *tar* и может использоваться для резервного копирования определенных наборов файлов на указанное устройство. Но *cpio* обладает рядом преимуществ перед *tar*, включая способность создавать архивы на нескольких устройствах. Программа работает в различных режимах, в частности входящего копирования, исходящего копирования и пассивного копирования. Режим входящего копирования используется для копирования файлов с ленточного накопителя в локальную файловую систему. Файлы, которые следует копировать, могут определяться маской или регулярным выражением, с которым сопоставляются имена файлов. Чтобы произвести входящее копирование файлов из архива */backups/paul.tar* в каталог */home/paul*, воспользуйтесь командой:

```
bash-2.03# cd /home/paul; cat /backups/paul | cpio -i *
```

Режим исходящего копирования используется для копирования файлов в другую файловую систему либо на внешний накопитель, вроде ленточного. Файлы для копирования могут определяться маской либо списком, поступающим со стандартного ввода. Например, чтобы создать архив всех файлов в каталоге */home/paul*, можно воспользоваться такой командой:

```
bash-2.03# ls | cpio -oc > /backups/paul.backup
```

Чтобы создать архив для каждого пользователя, достаточно применить подобный сценарий командного интерпретатора:

```
#!/bin/csh -f
foreach dir (`ls /home`)
{
    echo "Архивируется каталог $dir"
    ls /home/$dir | cpio -oc > /backups/$dir.backup
}
```

Чтобы скопировать все содержимое каждого из домашних каталогов пользователей, включая подкаталоги, воспользуйтесь таким сценарием:

```
#!/bin/csh -f
foreach dir (`ls /home`)
{
    echo "Архивируется каталог $dir"
    find . -name '*' -print | cpio -oc > /backups/$dir.backup
}
```

Режим пассивного копирования применяется в случаях, когда необходимо скопировать файлы из одной файловой системы в другую, не создавая архива. Это особенно полезно при переносе файлов между файловыми системами, когда не подходит команда *mv*. Например, при модернизации жесткого диска домашние каталоги пользователей часто должны переноситься с одного раздела на другой. Следующие команды производят итеративный перенос файлов из */home/paul* в */staff/paul*:

```
bash-2.03# cd /home/paul; find . -depth -print | cpio -pPd /staff/paul
```

Подобно программам *tar* и *dd*, *cpio* имеет ряд ключей, изменяющих стандартное поведение. Эти ключи перечислены в табл. 8.5.

Таблица 8.5. Основные ключи команды cpio

Ключ	Описание
a	Изменяет метку даты для файла на текущее время и дату
A	Добавляет новые файлы в существующий архив *cpio*
B	Размер блока по умолчанию
c	Определяет формат заголовка ASCII-файлов
d	Предписывает создавать каталоги по необходимости
H	Предписывает использовать конкретный формат заголовков (*tar* для tar-файлов, *crc* для ASCII, *ustar* для стандартных заголовков обмена данными IEEE)
r	Предписывает рекурсивно изменять имена файлов
t	Перечисляет файлы архива

ufsdump/ufsrestore

Мы рассмотрели команды Solaris, которые могут применяться для копирования файлов или блоков данных прямо на ленту; исходным материалом служат имена файлов или slice-разделов. Такой способ оптимален для небольших заданий резервного копирования: скажем, tar-файлы чаще применяются в качестве архивов отдельных пакетов, а не целых файловых систем. В качестве решения, охватывающего резервное копирование для многих файловых систем, узлов и сетей, Solaris предоставляет инструменты *ufsdump* и *ufsrestore*, позволяющие соответственно создавать резервные копии и восстанавливать файловые системы UFS. Эти системные программы зависят от платформы, как и формат пакетов Solaris, но полностью поддерживаются всеми системами Solaris. В этом разделе мы рассмотрим эффективное применение *ufsdump* и *ufsrestore* совместно с другими инструментами (такими как *cron* и *at*) для регулярного (ежедневного и еженедельного) создания полных и инкрементальных копий файловых системы.

Использование *ufsdump* в качестве инструмента резервного копирования сосредоточено вокруг понятия *уровня образа* (*dump level*): это число от 0 до 9, определяющее относительную шкалу между полными и инкрементальными образами копий. Полному образу всегда соответствует уровень 0, а первый из набора инкрементальных уровней может соответствовать одному из уровней с 1 по 9. Последующие инкрементальные образы должны принадлежать диапазону уровней со 2 по 9. Подряд может идти только восемь образов инкрементальных уровней; девятый должен быть связан с более низким уровнем, указывая на завершение последовательности образом. В табл. 8.6 показана простая последовательность уровней для случая, когда за полной еженедельной копией следует набор инкрементальных образов.

Таблица 8.6. Расписание уровней для еженедельного полного образа и инкрементальных образов

День	Уровень образа
Понедельник	0
Вторник	2
Среда	3
Четверг	4
Пятница	5
Суббота	6
Воскресенье	1

А вот вывод для полного образа:

```
bash-2.03# ufsdump 0cu /dev/rmt/2 /dev/rdsk/c1d1t0s1
  DUMP: Writing 63 Kilobyte records
```

```
DUMP: Date of this level 0 dump: Mon Mar 05 00:00:01 2001
DUMP: Date of last level 0 dump: Mon Mar 23 00:00:01 2001
DUMP: Dumping /dev/rdsk/c1d1t0s1 (system:/database) to /dev/rmt/2.
DUMP: Mapping (Pass I) [regular files]
DUMP: Mapping (Pass II) [directories]
DUMP: Estimated 563976 blocks (275.4MB).
DUMP: Dumping (Pass III) [directories]
DUMP: Dumping (Pass IV) [regular files]
DUMP: 563976 blocks (275.4MB) on 1 volume at 2342 KB/sec
DUMP: DUMP IS DONE
DUMP: Level 0 dump on Mon Mar 05 00:02:53 2001
```

Здесь создана резервная копия раздела */database* (*/dev/rdsk/c1d1t0s1*) на ленточном накопителе */dev/rmt2*. Чтобы восстановить данные, воспользуемся командой *ufsrestore*:

```
bash-2.03# ufsrestore xvf /dev/rmt/2
You have not read any volumes yet.
Unless you know which volume your file(s) are on you should start
with the last volume and work towards the first.
Specify next volume #: 1
set owner/mode for '.'? [yn] n
1024       ./data/flatfile1.txt
1024       ./data/flatfile2.txt
1024       ./data/flatfile3.txt
1024       ./data/flatfile4.txt
1024       ./data/backup1.txt
1024       ./data/backup2.txt
1024       ./data/backup3.txt
1024       ./data/backup4.txt
...
```

ufsrestore может также работать в диалоговом режиме. Это особенно удобно, когда существует необходимость найти определенный файл или различные его версии на нескольких ленточных носителях:

```
bash-2.03# ufsrestore i
ufsrestore > help
Available commands are:
        ls [arg] - list directory
        cd arg - change directory
        pwd - print current directory
        add [arg] - add `arg' to list of files to be extracted
        delete [arg] - delete `arg' from list of files to be extracted
        extract - extract requested files
        setmodes - set modes of requested directories
        quit - immediately exit program
        what - list dump header information
        verbose - toggle verbose flag (useful with ``ls'')
        help or `?' - print this list
If no `arg' is supplied, the current directory is used
ufsrestore >
```

Выполнение задач резервного копирования может планироваться с помощью команд *at* и *cron*. Команда *at* больше подходит для одноразовых задач, выполняемых нерегулярно. Например, в конце цикла разработки граница может быть проведена в полночь: многопользовательские учетные записи блокируются, происходит объединение дерева исходных текстов, и выполняется резервное копирование. Эти действия можно описать в сценарии и запланировать его выполнение с помощью *at*. И напротив, если вечером каждого воскресенья создается полный образ, за которым следуют ежедневные инкрементальные образы – по ночам будних дней, с понедельника по субботу, более подходящим решением станет *cron*. *cron* позволяет администраторам (и пользователям) задавать моменты регулярного выполнения рабочих заданий; задания выполняются до тех пор, пока не будут отменены.

Задания, выполнение которых запланировано с помощью *at*, перечислены в файле */var/spool/cron/atjobs*. В отличие от *cron*, задания которого должны указываться редактированием пользовательского файла *crontab*, *at* принимает информацию через интерфейс командной строки. Чтобы выполнить полное копирование файловой системы */dev/rdsk/c0t0d0s2* на ленточный накопитель */dev/rmt2*, воспользуйтесь командой:

```
bash-2.03# at 0000
at> ufsdump 0cu /dev/rmt/2 /dev/rdsk/c0t0d0s2
at> <EOT>
commands will be executed using /bin/bash
job 768574498.a at Tue Mar  6 15:22:02 2001
```

Когда задание определено таким образом, очередь заданий *at* может быть изучена перечислением файлов из каталога *atjobs*:

```
bash-2.03# ls -l /var/spool/cron/atjobs | grep root
total 16
-r-Sr--r--   1 root     wheel       1273 Mar  6 15:22:05 768574498.a
```

Этот файл соответствует заданию *at*, которое будет выполнено. Он содержит команды и переменные среды, определенные в качестве параметров задания:

```
bash-2.03# more 768574498.a
: at job
: jobname: stdin
: notify by mail: yes
export PWD; PWD='/root'
export _; _='/usr/bin/at'
cd /root
umask 22
ulimit unlimited
ufsdump 0cu /dev/rmt/2 /dev/rdsk/c0t0d0s2
```

Когда задание резервного копирования успешно выполнено, пользователю посылается подтверждающее сообщение электронной почты:

```
From paul Tue Mar 6 01:32:01 2001
Date: Tue Mar 6 2001 01:32:01 +1000 (EST)
From: root <root>
To: root
Subject: Output from "at" job
Your "at" job on longreach
"/var/spool/cron/atjobs/768574498.a"
produced the following output:
/bin/bash[1]: ufsdump
```

cron позволяет планировать задания резервного копирования (и другие) на определенную минуту, час, день, месяц и/или день недели. Например, можно запланировать создание полного образа раз в неделю (скажем, в 00:30 утра понедельника), а также инкрементальных образов каждое второе утро недели в то же время. Задания *cron*, созданные с помощью команды *crontab -e*, остаются в силе до тех пор, пока не будут явным образом отменены, в отличие от системы *at*, в которой принятое задание выполняется только один раз. Состояние заданий *cron* можно выяснить с помощью команды *crontab -l*.

Стандартная запись файла *crontab* содержит шесть полей, разделяемых пробелами. Слева направо для каждого задания: минута (0–59); час (0–23); день (1–31); месяц (1–12); день недели (0–6); а также команда, которая может являться именем сценария командного интерпретатора или приложения (в нашем случае *ufsdump*). Запись *crontab* для того же задания выглядит так:

```
0 0 6 3 * ufsdump 0cu /dev/rmt/2 /dev/rdsk/c0t0d0s2
```

Чтобы эта команда выполнялась каждый день в полночь, воспользуйтесь такими параметрами:

```
0 0 * * * ufsdump 0cu /dev/rmt/2 /dev/rdsk/c0t0d0s2
```

Следующая запись предписывает выполнение команды только по воскресеньям:

```
0 0 * * 1 ufsdump 0cu /dev/rmt/2 /dev/rdsk/c0t0d0s2
```

Записи для заданий по резервному копированию на всю неделю:

```
0 0 * * 1 ufsdump 0cu /dev/rmt/2 /dev/rdsk/c0t0d0s2
0 0 * * 2 ufsdump 5cu /dev/rmt/2 /dev/rdsk/c0t0d0s2
0 0 * * 3 ufsdump 6cu /dev/rmt/2 /dev/rdsk/c0t0d0s2
0 0 * * 4 ufsdump 7cu /dev/rmt/2 /dev/rdsk/c0t0d0s2
0 0 * * 5 ufsdump 8cu /dev/rmt/2 /dev/rdsk/c0t0d0s2
0 0 * * 6 ufsdump 1cu /dev/rmt/2 /dev/rdsk/c0t0d0s2
```

Пакеты для резервного копирования и восстановления

Стандартные инструменты Solaris для резервного копирования и восстановления, как правило, вполне справляются с возложенными на них задачами в случае отдельных узлов. Но в случаях, когда выделенный сервер является сервером резервного копирования для всего отдела или целой сети, имеет смысл использовать специальные системы управления хранением. Из популярных коммерческих пакетов можно упомянуть Veritas NetBackup (*http://www.veritas.com/*) и Legato Networker (*http://www.legato.com/*), которые поддерживают распределенную модель клиент–сервер; запросы на резервное копирование, поступающие от клиентов (работающих с системами Solaris, Microsoft Windows или другими, для которых реализована поддержка), передаются серверу резервного копирования, который выполняет непосредственное копирование данных без участия клиента. Такой подход позволяет пользователям активно работать с крупными хранилищами данных, доступными через выделенный сервер резервного копирования, но имеет и недостаток – излишние затраты пространства могут достигать значительных величин, если запросы клиентов не будут подвергаться контрольным проверкам. Скажем, запрос на резервное копирование документов и других данных пользователя всегда должен выполняться, но ежедневное полное копирование операционной системы каждого клиента неэффективно, поскольку система может быть в любой момент восстановлена с помощью JumpStart или другого сходного ПО.

Существуют также бесплатные инструменты, ставшие стандартом дефакто для задач резервного копирования, связанных с отдельными узлами. В частности, можно упомянуть систему AMANDA. AMANDA не предназначена для замены стандартных инструментов Solaris, но предоставляет высокоуровневый интерфейс управления, позволяющий создавать сценарии и наборы команд для выполнения резервного копирования. Это избавляет администраторов от необходимости помнить синтаксис редко применяемых команд создания и восстановления образов, но при этом позволяет им полноценно использовать инструменты, которые существуют в Solaris. Более подробная информация о системе AMANDA доступна по адресу *http://www.amanda.org/*.

9

Сетевая безопасность

После создания сети на основе Solaris многими системными администраторами овладевает страх при мысли о том, что сеть может подвергнуться внешней или внутренней атаке. Многие администраторы серьезных сайтов считают, что непрерывно находятся в боевых условиях, и часто достигают крайней степени паранойи, изо дня в день пытаясь предотвратить худшие из вариантов развития событий. Непосредственно после установки в стандартном варианте Solaris (как и любая другая сетевая операционная система) предоставляет потенциальным нарушителям много точек доступа к сети. Однако модульная природа пакетов Solaris и простота настройки большинства служб позволяют довести безопасность системы до нужного уровня. Например, автономная система Solaris, на которой работает финансовое ПО и установлена локальная реляционная СУБД, может не задумываться о сетевых атаках. Сайт электронной коммерции, напротив, предоставляет избирательный доступ внешних пользователей к серверным приложениям, скрывая конфиденциальные и другие чувствительные данные, такие как номера кредитных карт. Кроме того, получение доступа к единственному узлу сети Solaris позволяет воспользоваться распределенными сетевыми информационным службами (скажем, NIS+) и получить доступ к данным многих узлов локальной сети. Это не значит, что следует воздерживаться от применения NIS+ и родственных систем – NIS+ обеспечивает достаточный уровень безопасности, если закрыт несанкционированный доступ к узлам сети. Также важно понимать, что источником атак не обязательно является внешний мир – один из пользователей (например, недовольный сотрудник) вполне может, пользуясь своим терминалом, получить уровень доступа более высокий, чем ему положено для выполнения рабочих задач. Необходимо проводить внешние и внутренние ревизии с целью оценки возможных угроз и рисков.

В этой главе мы изучим стратегию безопасности Solaris, основанную на отключении всех сетевых служб и предоставлении доступа к ним только в том случае, когда существует возможность производить идентификацию, проверку подлинности и управление доступом. Этот процесс состоит из нескольких этапов:

- Следует принять меры для обеспечения безопасности учетных записей, в частности выбрать надежные пароли
- Отключить службы, наличие которых не требуется или связано с определенной уязвимостью, в частности службы удаленного доступа, в которых не реализовано шифрование
- Определить наличие потенциальных уязвимостей активных демонов
- Проверить содержимое файлов с целью обнаружения «троянских коней»
- Установить службы удаленного доступа, обеспечивающие достойный уровень безопасности
- Организовать фильтрацию IP-пакетов и ограничить межсетевую передачу данных на уровне маршрутизатора и отдельных портов
- Производить проверку подлинности узлов с помощью распределенных механизмов
- Реализовать шифрование на уровне IP с целью защиты всех передаваемых данных
- Убедиться, что узлы сети не имеют прямого выхода в сеть Интернет (а используют proxy-сервер)

В этой главе я расскажу о настройке различных служб Solaris и о применении административных приложений для достижения вышеперечисленных целей, в частности:

- О применении программы Ван Влека (Van Vleck) *gpw* для создания случайных паролей, которые легко запоминать
- Об удалении лишних демонов, порождаемых супердемоном интернет-служб (*inetd*)
- О применении алгоритма MD5 для вычисления контрольных сумм ключевых исполняемых файлов системы
- О замене Telnet и FTP защищенным командным интерпретатором (SSH) и безопасным ftp (*sftp*), доступными на сайте *http://www.ssh.com*
- Об установке брандмауэра SunScreen с целью осуществления фильтрации пакетов
- О настройке системы распределенной проверки подлинности Kerberos 5
- О применении модулей IPsec для шифрования IP-пакетов
- Об установке Socks 5 в целях обеспечения прозрачного доступа ко внешним службам (для таких программ, как Netscape Navigator)

Приняв перечисленные меры, администраторы Solaris смогут проявлять меньше активности, реагируя на инциденты, связанные с безопасностью, и тратить больше времени на изучение новых рисков и реализацию противодействия обнаруженным уязвимостям. Повторюсь, степень реализации перечисленных решений зависит от требований, предъявляемых к безопасности конкретных систем.

Парольная безопасность

Базовой формой обеспечения безопасности пользовательской учетной записи Solaris является пароль. Следуя традициям и стандартам Unix, Solaris ограничивает длину имени пользователя и пароля восемью символами. Только для алфавитно-цифровых символов это дает 36^8 различных паролей, что невероятно затрудняет угадывание пароля полным перебором. Но если злоумышленнику достался пароль пользовательской учетной записи Solaris – неважно, каким образом, – он получает возможность создавать, изменять, удалять и читать файлы на смонтированных файловых системах; единственное, что его ограничивает, – это существующие для файлов права доступа. Таким образом, имя пользователя Solaris является идентификатором и может использоваться для разграничения доступа к различным службам системы даже после того, как была произведена проверка подлинности.

В процессе установки Solaris по умолчанию создает ряд «системных» учетных записей, в частности *root*, *daemon*, *sys*, *adm*, *lp*, *uucp* и *nobody*. Причины существования всех этих записей в основном исторические, исключением является только запись суперпользователя *root*, которой присваивается нулевой идентификатор и предоставляется неограниченный доступ к системе. Многие из стандартных учетных записей имеют небольшие (<100) идентификаторы, но абсолютной властью над системой обладают только пользователи с нулевым идентификатором. В действительности во многих случаях запись *root* заменяется другой записью с произвольным именем, что позволяет автоматически отражать атаки, основанные на использовании имени *root*. Система доступна для администрирования, если есть хотя бы одна учетная запись с нулевым идентификатором.

В Solaris идентификация пользователей и парольная проверка подлинности зависит от двух файлов: файла паролей (*/etc/passwd*) и файла теневых паролей (*/etc/shadow*). Изначально файл паролей Solaris содержал строки зашифрованных паролей, которые сравнивались с результатами выполнения функции *crypt( )* для паролей, вводимых пользователями. В случае соответствия проверка подлинности завершалась успешно. При этом для прохождения проверки все пользователи должны были иметь права на чтение файла паролей, и, как следствие, каждый пользователь имел доступ к зашифрованным строкам паролей всех пользователей системы (включая пользователя *root*). Это

привело к появлению многочисленных программ взлома паролей, в которых для быстрого (и правильного) дешифрования паролей применялись словари. Такая система позволяла даже низкопривилегированному пользователю получить зашифрованную строку пароля пользователя *root*. Процесс получения зашифрованных паролей в случае, когда теневые пароли не используются, показан на рис. 9.1.

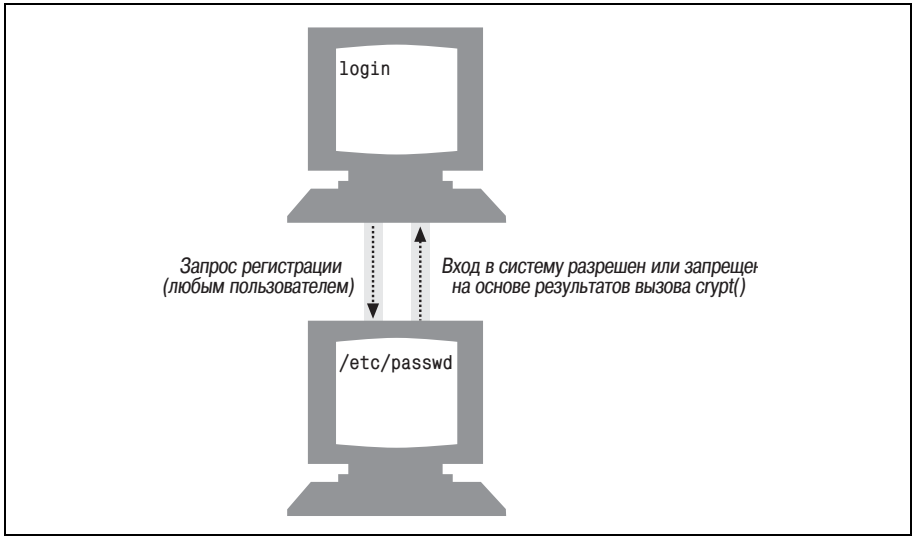

Рис. 9.1. *Регистрация в системе, зашифрованные пароли хранятся в /etc/passwd*

Сокрытие паролей (*shadowing*) является простым, но очень эффективным дополнением к традиционному файлу паролей. Зашифрованная строка пароля хранится в отдельном файле, права на чтение которого есть только у пользователя *root*. В файле паролей соответствующее поле заменяется символом *x*, что не позволяет пользователям системы получать доступ к зашифрованным паролям. Процесс извлечения зашифрованного пароля в случае применения теневых паролей показан на рис. 9.2.

Пример файла */etc/passwd*:

```
bash-2.03# cat /etc/passwd
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1::/:
bin:x:2:2::/usr/bin:
sys:x:3:3::/:
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
uucp:x:5:5:uucp Admin:/usr/lib/uucp:
nuucp:x:9:9:uucp Admin:/var/spool/uucppublic:/usr/lib/uucp/uucico
listen:x:37:4:Network Admin:/usr/net/nls:
nobody:x:60001:60001:Nobody:/:
```

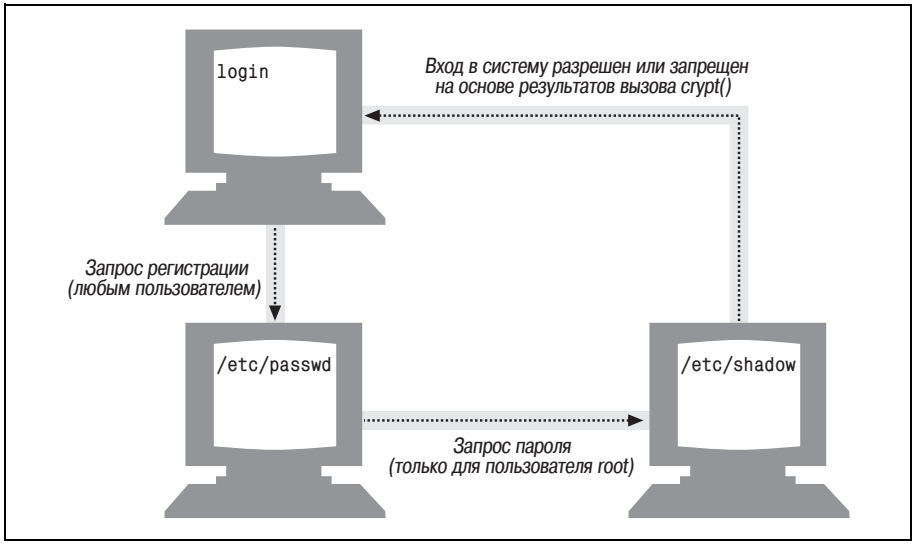

Рис. 9.2. *Регистрация в системе с использованием теневых паролей*

```
noaccess:x:60002:60002:No Access User:/:
nobody4:x:65534:65534:SunOS 4.x Nobody:/:
postgres:x:1001:100:Postgres User:/usr/local/postgres:/bin/sh
htdig:x:1002:10:htdig:/opt/www:/usr/local/bin/bash
apache:x:1003:10:apache user:/usr/local/apache:/bin/sh
www:x:1005:10:Web User:/opt/www:/bin/sh
```

Рассмотрим поля записей подробнее. Первое поле (имя пользователя) является уникальным для каждой пользовательской учетной записи – имена не могут быть «общими» для разных записей. Могут использоваться такие имена, как *www* и *apache*, но не *richmondva*, поскольку длина этого имени превышает восемь символов. Одной учетной записью могут одновременно пользоваться несколько человек, и процессы, принадлежащие каждому из них, будут выполняться с привилегиями данной учетной записи. Если известна верная комбинация имени и пароля, в общем случае можно получить доступ к системе, если только не были отключены службы удаленного доступа (такие как Telnet).

Следующее поле предназначено для паролей, которые теперь хранятся в файле теневых паролей (а соответствующие поля в обычном файле паролей заполняются символом *x*). Минимальная допустимая длина пароля может определяться переменной *PASSLENGTH*, установленной в файле */etc/default/passwd* (по умолчанию шесть символов, но лучше будет использовать максимальное значение, восемь символов). Файл */etc/passwd* должен быть доступен для чтения всем пользователям, а файл */etc/shadow* – только пользователю *root*:

```
bash-2.03# ls -l /etc/passwd
-rw-r--r--   1 root     root         605 Jul 24 11:04 /etc/passwd
```

```
bash-2.03# ls -l /etc/shadow
-r--------   1 root     root           333 Oct 11 15:54 /etc/shadow
```

В прежних версиях Solaris, еще до реализации сокрытия паролей, все пользователи системы имели доступ к зашифрованным паролям из файла */etc/passwd*; теперь зашифрованные пароли хранятся в файле */etc/shadow*. Разумеется, если права доступа для файла */etc/shadow* установлены корректно, только пользователь *root* сможет прочитать строки паролей в целях проверки подлинности. Это предотвращает применение программ взлома паролей, таких как Crack, относительно непривилегированными пользователями.

Поле идентификатора пользователя (UID) содержит число, большее или равное нулю и уникальное для каждого пользователя. Пользователю *adm* обычно присваивается идентификатор 4, и это означает, что пользователь *postgres* не может иметь UID, равный 4. Поле идентификатора группы (GID) содержит число, большее или равное единице и соответствующее базовой группе, в которую входит пользователь. В отличие от пользовательских идентификаторов, групповые идентификаторы могут быть общими для различных пользователей. Кроме того, пользователь может входить в более чем одну группу, и действующий идентификатор пользователя может изменяться в командном интерпретаторе с помощью команды *newgrp*. Информация о группах хранится в файле */etc/group*.

Следующее поле содержит полное имя пользователя для данной учетной записи. Затем идет путь к домашнему каталогу пользователя. Чтобы пользователь мог войти в систему, каталог должен существовать, в противном случае домашний каталог будет установлен в корневой каталог системы «/». Последнее поле определяет полное имя командного интерпретатора пользователя, используемого по умолчанию. Этот интерпретатор должен быть упомянут в файле */etc/shells*; в противном случае пользователь не сможет войти в систему. Новички в администрировании часто совершают ошибку, изменяя командный интерпретатор пользователя *root* на тот, в котором доступны более совершенные возможности работы с терминалом или сценариями (скажем, Bourne-подобный командный интерпретатор, *bash*). Однако если имя */usr/local/bin/bash* отсутствует в файле */etc/shells*, пользователь *root* не сможет войти в систему.[1] Типичный файл */etc/shells* может выглядеть так:

```
/sbin/sh
/bin/sh
```

[1] Данное утверждение автора некорректно. Пользователь не сможет зарегистрироваться в системе только при неправильном указании интерпретатора (типичная ошибка: вместо */sbin/sh* для пользователя *root* указывают */sbin/ksh*, который не существует). Сказанное автором о файле */etc/shells* справедливо только для регистрации при помощи *ftp*. – *Примеч. науч. ред.*

```
/bin/csh
/bin/ksh
/bin/tcsh
/bin/bash
/usr/bin/ksh
/usr/local/bin/bash
```

Как уже говорилось, существует файл */etc/default/passwd*, определяющий ряд общесистемных параметров для паролей. Как правило, файл */etc/default/passwd* содержит такие записи:

```
MAXWEEKS=4
MINWEEKS=
PASSLENGTH=6
WARNWEEKS=1
```

В данном случае определяется число недель, в течение которых пароль может оставаться неизменным (*MAXWEEKS=4*, то есть четыре недели). Наложение такого ограничения на пароль является удобным способом предотвращения несанкционированного доступа к учетной записи пользователя, поскольку списки устаревших паролей, распространяемые в конференциях Usenet и среди криминальных организаций, будут полезны лишь в течение ограниченного времени. Дополнительно можно задать минимальный интервал между сменами пароля (также в неделях, *MINWEEKS*). Установка значения этой переменной приведет к тому, что пользователи не смогут изменять свои пароли в течение указанного периода времени, что является не очень хорошей идеей: если пароль пользователя будет раскрыт на следующий день после его изменения, взломщик сможет как минимум шесть дней работать с системой, прежде чем пользователь получит возможность опять сменить пароль. И наконец, можно указать число недель до наступления момента изменения пароля, когда следует предупредить пользователя. Это очень полезно, поскольку конкретный день изменения пароля может оказаться не самым удобным для придумывания нового пароля (скажем, субботой или воскресеньем).

Администратор системы может напрямую добавлять информацию в файлы */etc/passwd* и */etc/shadow*. Однако ввод неверной информации в единственном поле может привести к раскрытию отдельных записей паролей. Например, если добавить лишнее поле в конец записи пользователя *root*, неправильно проставив двоеточие, записи других пользователей будут прочитаны некорректно. Одним из решений является применение команды *vipw*, которая использует редактор *vi* для безопасного изменения файла */etc/passwd*. В случае обнаружения ошибок будет отображено предупреждение, а изменения не будут сохранены в файле. Например, если командный интерпретатор */sbin/sh* не существует или не упомянут в файле */etc/shells*, попытка указать его в поле стандартного интерпретатора пользователя *root* приведет к ошибке:

```
Invalid root shell:
```

```
root:x:0:1:Super-User:/:/sbin/sh
vipw: you mangled the /etc/ptmp file, /etc/passwd unchanged
Invalid root shell:
root:x:0:1:Super-User:/:/sbin/sh
vipw: you mangled the /etc/ptmp file, /etc/passwd unchanged
```

Проблему можно элементарно решить добавлением */sbin/sh* в файл */etc/shells*. Как уже говорилось, в Solaris для предотвращения несанкционированного доступа к зашифрованным строкам паролей применяется система теневых паролей. Файл теневых паролей, соответствующий файлу паролей, показанному ранее, будет выглядеть так:

```
bash-2.03# cat /etc/shadow
root:Y25pGbjAh9a11:11033::::::
daemon:NP:6445::::::
bin:NP:6445::::::
sys:NP:6445::::::
adm:NP:6445::::::
lp:NP:6445::::::
uucp:NP:6445::::::
nuucp:NP:6445::::::
listen:*LK*:::::::
nobody:NP:6445::::::
noaccess:NP:6445::::::
nobody4:NP:6445::::::
postgres:hBi3b21Z34qHa:::::::
htdig:Y25pGbjAh9a11:::::::
apache:Y25pGbjAh9a11:::::::
www:Y25pGbjAh9a11:11241::::::
```

Файл */etc/shadow* во многом схож с */etc/passwd*: каждому пользователю соответствует одна строка, состоящая из полей, разделенных двоеточиями. Первое поле содержит имя пользователя, второе – зашифрованную строку пароля, которая пропускается в файле */etc/passwd* в случае использования теневых паролей. Многие из полей файла */etc/shadow* являются необязательными и могут пропускаться по желанию. Вот эти поля:

- Дата изменения пароля (число дней, прошедших с первого января 1970 года)
- Число дней, по истечении которых будет разрешено сменить пароль
- Число дней, по истечении которых пароль должен быть изменен
- Число дней, в течение которых пользователь получает предупреждения об окончании действия учетной записи
- Число дней отсутствия активности в отношении учетной записи, по истечении которого учетная запись блокируется
- Дата блокировки доступа к учетной записи

Теневые пароли с точки зрения безопасности гораздо более эффективны, чем доступные всем зашифрованные пароли, но методы прямого

перебора все так же остаются эффективными для случаев, когда пароль легко угадать или когда для учетных записей используются пароли по умолчанию. Проблемы такого рода связаны не только с механизмами проверки подлинности операционной системы, но и с механизмами отдельных приложений. Так, некоторые из серверов баз данных, работающих в Solaris, при установке создают стандартные пользовательские учетные записи с широко известными стандартными паролями. Незащищенный демон базы данных, предоставляющий доступ к чувствительным корпоративным данным, может быть уязвим для внешних атак как раз по причине использования паролей, легко поддающихся угадыванию.

ОС Solaris накладывает некоторые ограничения на пользовательские пароли. Так, пароль не должен быть перестановкой символов имени пользователя: пользователь *pwatters* не сможет использовать в качестве пароля строки *wattersp* и *srettawp*. Кроме того, новый пароль должен отличаться от существующих по меньшей мере тремя символами. Пароль *xyz12345* можно сменить на *xyz67890*, но не на *xyz12367*. И наконец, любой пароль должен содержать по меньшей мере две буквы (допускается произвольная комбинация регистров) и по меньшей мере один специальный символ или цифру. В результате пользователям недоступны пароли, которые представляют собой английские слова и могут быть легко отгаданы (гораздо сложнее взломать пароль вроде *p0ss1ble*, чем визуально очень похожее слово *possible*, в котором нет цифр).

Если пользователи испытывают затруднения, изобретая легкие для запоминания и неудобные для взлома пароли, они могут обратиться к ряду инструментов от сторонних разработчиков, предназначенных для автоматического создания паролей. Превосходным вариантом является программа-генератор паролей *gpw*, автором которой является Ван Влек. Программа доступна в виде Java-апплета по адресу *http://www.multicians.org/thvv/gpw.html*. Исполняемые файлы для системы Solaris доступны в виде пакета по адресу *ftp://nce.sun.ca/pub/freeware/intel/7/gpw-6.94-sol7-intel-local.gz*. Приложение *gpw* обычно применяется для создания паролей, которые не являются словами, но легки в произношении и запоминании. Например, любой пароль из следующего списка более надежен, чем обычные слова:

```
bash-2.03# gpw
oomeresc
tonylvey
vescrenv
ismethms
wistongr
stenumon
mbodware
iblettic
strawore
venstian
```

Кроме того, вывод *gpw* может изменяться с целью удовлетворения ограничений Solaris (необходимо наличие по меньшей мере одного числа). Полученный ранее список паролей может быть преобразован в такой:

```
0Omeresc
tOnylvey
6escrenv
1smethms
w1st0ngr
stenum0n
mb0dware
1blett1c
straw0re
venst1an
```

Самый большой риск, связанный с паролями, – их перехват при передаче в незашифрованном виде в процессе инициализации сеанса удаленного доступа (*telnet* или *ftp*). Злоумышленник должен иметь доступ к сетевому интерфейсу, через который проходят пакеты, путешествуя от клиента к серверу. Чтобы перехват мог быть осуществлен, сетевой интерфейс должен находиться в режиме прослушивания (*promiscuous mode*). Предположим, пользователь из Исландии устанавливает соединение с системой в США; пакеты, передаваемые в обе стороны, должны пройти через ряд промежуточных узлов. Чтобы определить число промежуточных узлов между клиентом и сервером, можно воспользоваться инструментом *traceroute*:

```
traceroute to www.cassowary.net (207.150.192.12), 30 hops max, 40 byte packets
 1  Reykjavik00 (193.4.58.1)  1.059 ms  1.145 ms  0.931 ms
 2  is-gw (193.4.61.193)  0.779 ms  0.649 ms  0.715 ms
 3 if-0-0-1.bb2.NewYork.Teleglobe.net (207.45.202.13)  67.228 ms  67.318 ms
   67.106 ms
 4  sl-gw16-pen-2-0.sprintlink.net (144.228.164.61)  75.623 ms  69.010 ms
   70.072 ms
 5  sl-bb12-pen-1-1.sprintlink.net (144.232.5.93)  69.122 ms  68.886 ms
   68.893 ms
 6  sl-bb22-pen-12-0.sprintlink.net (144.232.16.49)  69.088 ms  68.860 ms
   68.992 ms
 7  sl-bb20-fw-12-0.sprintlink.net (144.232.18.77)  100.232 ms  100.348 ms
   100.010 ms
 8  sl-bb21-ana-9-0.sprintlink.net (144.232.18.62)  123.969 ms  124.328 ms
   124.184 ms
 9  sl-gw24-ana-10-0.sprintlink.net (144.232.1.198)  124.550 ms  124.070 ms
   124.110 ms
10  sl-affinity-3-0-0.sprintlink.net (144.232.254.2)  137.443 ms  137.588 ms
   137.885
   ms
11  207.150.192.12 (207.150.192.12)  139.521 ms  139.371 ms  139.196 ms
```

В данном случае любой из расположенных в различных странах промежуточных узлов может перехватывать данные. Поскольку в целях

установки сеансов FTP и Telnet имена пользователей и пароли передаются в открытом виде, злоумышленник может с легкостью перехватить данные, которыми обмениваются узлы, если маршрут известен заранее. Поскольку информация маршрутизации находится в свободном доступе, взломщик может заранее определить узел, идеально подходящий для перехвата паролей, адресованных определенному узлу (рис. 9.3).

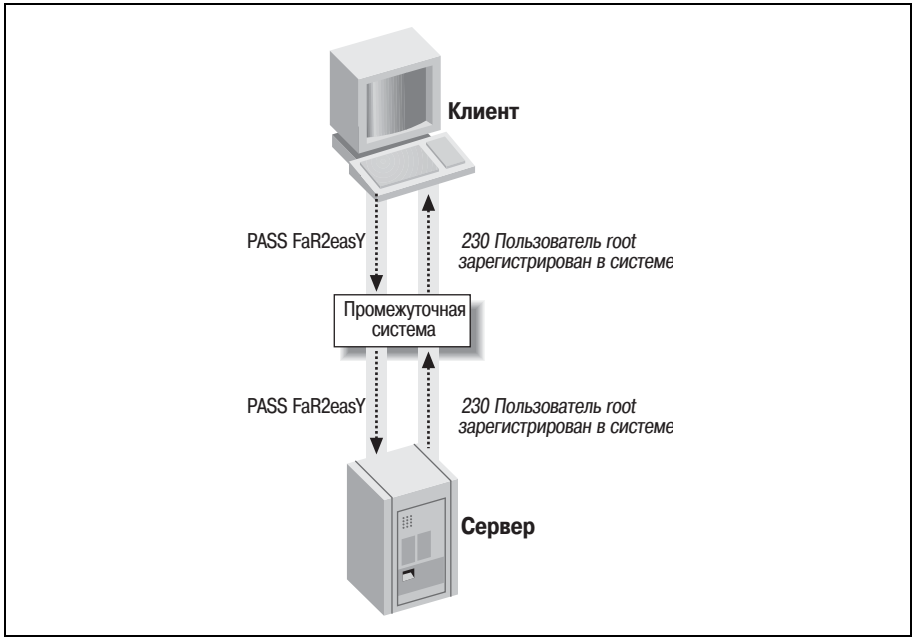

Рис. 9.3. *Перехват паролей промежуточным узлом*

Чтобы проверить эти тезисы на практике, можно воспользоваться программой *snoop* для перехвата и выборочного отображения содержимого передаваемых между узлами пакетов. Возьмем для примера службу FTP, работающую через порт с номером 21. Допустим, пользователь системы *client.paulwatters.com* установил FTP-соединение с сервером *server.cassowary.net* в качестве пользователя *root*. С помощью следующей команды можно просмотреть данные, которыми обмениваются клиент и сервер:

```
bash-2.03# snoop port 21 tcp
Using device /dev/elx (promiscuous mode)
client.paulwatters.com -> server.cassowary.net FTP C port=4854
server.cassowary.net -> client.paulwatters.com FTP R port=4854
client.paulwatters.com -> server.cassowary.net FTP C port=4854
server.cassowary.net -> client.paulwatters.com FTP R port=4854 220 server FTP
server
client.paulwatters.com -> server.cassowary.net FTP C port=4854
client.paulwatters.com -> server.cassowary.net FTP C port=4854 USER root\r\n
```

```
server.cassowary.net -> client.paulwatters.com FTP R port=4854
server.cassowary.net -> client.paulwatters.com FTP R port=4854 331 Password
require
client.paulwatters.com -> server.cassowary.net FTP C port=4854
client.paulwatters.com -> server.cassowary.net FTP C port=4854 PASS
FaR2easY\r\n
server.cassowary.net -> client.paulwatters.com FTP R port=4854 230 User root
logged
client.paulwatters.com -> server.cassowary.net FTP C port=4854
client.paulwatters.com -> server.cassowary.net FTP C port=4854 PORT
204,64,12,33,
server.cassowary.net -> client.paulwatters.com FTP R port=4854 200 PORT
command suc
client.paulwatters.com -> server.cassowary.net FTP C port=4854 LIST\r\n
server.cassowary.net -> client.paulwatters.com FTP R port=4854 150 ASCII data
conne
client.paulwatters.com -> server.cassowary.net FTP C port=4854
server.cassowary.net -> client.paulwatters.com FTP R port=4854 226 ASCII
Transfer c
client.paulwatters.com -> server.cassowary.net FTP C port=4854
client.paulwatters.com -> server.cassowary.net FTP C port=4854 QUIT\r\n
server.cassowary.net -> client.paulwatters.com FTP R port=4854 221
Goodbye.\r\n
client.paulwatters.com -> server.cassowary.net FTP C port=4854
server.cassowary.net -> client.paulwatters.com FTP R port=4854
server.cassowary.net -> client.paulwatters.com FTP R port=4854
client.paulwatters.com -> server.cassowary.net FTP C port=4854
```

Мы можем наблюдать стандартные команды FTP, отображаемые на экране клиента (к примеру, *200 PORT*), а также ответы сервера (например, соединение *150 ASCII*). А кроме того, в глаза бросается имя пользователя (команда *USER*) и пароль администратора – команда *PASS* (для тех, кто не заметил, пароль – *FaR2easY*). Дополнительную информацию о передаваемых пакетах можно получить, включив режим подробного вывода (ключ *-v*).

```
bash-2.03# snoop -v port 21 tcp
Using device /dev/elx (promiscuous mode)
ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 1 arrived at 14:05:34.04
ETHER:  Packet size = 60 bytes
ETHER:  Destination = 1:52:4:22:5a:33,
ETHER:  Source      = 2:48:ef:33:b2:d3,
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 = 44 bytes
IP:   Identification = 21386
IP:   Flags = 0x4
IP:         .1.. .... = do not fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 128 seconds/hops
IP:   Protocol = 6 (TCP)
IP:   Header checksum = 878b
IP:   Source address = 204.64.12.33, client.paulwatters.com
IP:   Destination address = 203.64.12.34, server.cassowary.net
IP:   No options
IP:
TCP:  ----- TCP Header -----
TCP:
TCP:  Source port = 4859
TCP:  Destination port = 21 (FTP)
TCP:  Sequence number = 2430713936
TCP:  Acknowledgement number = 0
TCP:  Data offset = 24 bytes
TCP:  Flags = 0x02
TCP:        ..0. .... = No urgent pointer
TCP:        ...0 .... = No acknowledgement
TCP:        .... 0... = No push
TCP:        .... .0.. = No reset
TCP:        .... ..1. = Syn
TCP:        .... ...0 = No Fin
TCP:  Window = 8192
TCP:  Checksum = 0xf829
TCP:  Urgent pointer = 0
TCP:  Options: (4 bytes)
TCP:    - Maximum segment size = 1460 bytes
TCP:
FTP:  ----- FTP:   -----
FTP:
FTP:  ""
FTP:
```

Оптимальный способ сокращения риска подобного перехвата – не пользоваться приложениями, передающими незашифрованные пароли. Это касается всех *r*-команд (*rsh*, *rlogin* и т. д.), *telnet* и *ftp*. К счастью, многие их этих приложений имеют более защищенные альтернативы, передача имен пользователей и паролей в которых осуществляется только после шифрования по схеме, согласованной клиентом и сервером. Одним из популярных инструментов обеспечения безопасного удаленного доступа является SSH.

Защищенный интерпретатор SSH

В предыдущем разделе мы могли видеть, насколько прост перехват имени пользователя и пароля с помощью специальных инструментов. Одним из способов избежать перехвата открытых имен и паролей является шифрование до передачи в процессе проверки подлинности. Желаемого результата можно добиться с помощью защищенного командного интерпретатора (secure shell, SSH), доступного в ряде коммерческих (*http://www.ssh.com/*) и бесплатных (*http://www.openssh.org/*) реализаций.

Клиент-серверный протокол SSH реализует схему шифрования уровня транспорта, позволяющую приложениям и службам, применяющим шифрование, использовать общий протокол для работы по TCP/IP, не изобретая каждый раз велосипед заново. Пакет программ SSH обычно содержит приложения клиентов и серверов служб, призванных заменять стандартные ненадежные приложения удаленного доступа, о которых мы уже говорили, такие как *telnet*, *ftp*, *rlogin* и *rsh*.

SSH поддерживает симметричное и асимметричное шифрование сеансов в целом, а также обмен лексемами (credentials) идентификации и проверки подлинности. В большинстве вариантов поддерживаются алгоритмы шифрования RSA, Blowfish и 3-DES. Клиенты и серверы SSH используют один из двух протоколов: старый SSH1 и новый SSH2. Переход от SSH1 к SSH2 был связан с рядом важных изменений, включая улучшение поддержки хранения открытых ключей (в различных файлах) и появление возможности использовать несколько закрытых ключей на одного пользователя.

SSH не решает проблему перехвата пакетов; третья сторона все так же сможет перехватывать все пакеты, которыми обмениваются клиент и сервер, если получит доступ к работающему в смешанном режиме сетевому интерфейсу промежуточного сервера. Тем не менее данные пакетов невозможно будет расшифровать без соответствующего ключа. И хотя в теории возможна расшифровка пакетов полным перебором, SSH обеспечивает дополнительный уровень защиты, который связан с понятием сеанса. Можно провести аналогию с одноразовым ключом шифрования (one-time cipher pad), действительным лишь в течение одного часа, поскольку данные отдельных сеансов каждый раз шифруются различными ключами. Согласование протоколов и обмен ключами сеансов (а также их смена каждый час) делают SSH одним из наиболее безопасных способов обмена данными.

Сеанс SSH1 инициируется клиентом SSH, который посылает запрос серверу SSH через порт 22. Если сервер SSH активен на удаленной системе, клиент получает в ответ два RSA-ключа: 1024-битный ключ узла и 768-битный ключ сеанса. Ключ узла выглядит так:

```
---- BEGIN SSH1 PUBLIC KEY ----
Subject: root
Comment: "host key for emu, accepted by root Mon Dec 18 2000 05:\
```

```
58:32"
DfPa3pf33Mt2zHgZe4Ec5+Nkl3IIOOC1IBpA5poJuEJBGPoCedO2+XxXF/q1vB6XT9QHN9
AAAAB3NzaC1kc3MAAACBAJ42ie37em9uaemrkBS8DsI+NzHHplGVyHKZwICFp7o8YlehZQ
q9HE8RleH4+vXVtjI3CUHlLH1gClIgoTWXxhIJDpn52/KmklCnvzAAAAFQD9tzWrgOe9G9
K8xjwB7Asw86OixQAAAIEAmUymx/xwab8JDieDkpTUNppW3FB9wItt/o8ORNa4IPLttyX7
giTnOSYraIfOeQyJ6uYLSjcsc1ZmJnBYSqF8+H6XSDXoksJpHsqTFKJhRbsXYDnwzVjeIb
KC2EyQ+/jq4MN7ODZ/OfQlxxEXCNIHH/y2bv1Ab38LF9tY5T7/8wcAAACAYWdZ5CMxXDZv
xc+OMRr438aSDbEKo6orHOLmfWdDr4/jrLZbo9Um66JPNMGVJabnCpRpTlTUWJxg7Fg86N
8o9Znfqknx7Ywmuz7njl4vtxcwzuGD6NrCnnWNu4+P8nCPlMOI+DGuQmwDZT+OajM2NLfg
O74QjtoCR+7truDRxDQ=
---- END SSH1 PUBLIC KEY ----
```

Получив эти ключи, клиент передает 256-битный ключ сеанса серверу и выбирает алгоритм шифрования, который будет применяться для трафика сеанса. Затем, когда ключ сеанса клиента получен сервером, он шифруется с применением ключей узла и сеанса и в таком зашифрованном виде возвращается клиенту. Начиная с этого момента может происходить обмен зашифрованными данными, который начинается с проверки имени пользователя и пароля. Многие пользователи предпочитают набирать имя и пароль в каждом сеансе (это позволяет достичь максимального уровня безопасности), но при этом существует возможность хранения на клиенте и сервере сертификатов, служащих для проверки подлинности пользователей. Разумеется, применение сертификатов связано с возможностью их перехвата, который позволит злоумышленнику, проникшему на одну из систем, получить доступ ко второй без идентификации.

SSH2 позволяет изменять значения многих параметров, которые были жестко зафиксированы в SSH1; этой цели служит файл */etc/ssh2/sshd2_config*. Кроме того, можно включить поддержку клиентов SSH1 серверами SSH2, добавив следующие строки в файл */etc/ssh2/sshd2_config*:

```
Ssh1Compatibility  yes
Ssh1Path           /usr/local/bin/ssh1
```

Файл *sshd2_config* определяет и многие другие параметры работы демона SSH, а также содержит разделы, посвященные общим и сетевым настройкам, шифрованию, идентификации и открытым ключам пользователей, туннелированию и настройкам отдельных узлов. Вот пример файла *sshd2_config*, определяющего ряд таких свойств:

```
VerboseMode                    no
PasswordPrompt                 "%U's password: "
AuthenticationSuccessMsg       yes
Port                           22
NoDelay                        no
KeepAlive                      yes
Ciphers                        AnyStdCipher
MACs                           AnyMAC
StrictHostKeyChecking          ask
```

```
IdentityFile                    identification
AuthorizationFile               authorization
RandomSeedFile                  random_seed
Ssh1Compatibility               yes
Ssh1AgentCompatibility          none
AllowedAuthentications          publickey,password
```

Смысл настроек вполне очевиден: приглашение ввода пароля имеет формат *%U's password:*, где вместо параметра *%U* подставляется регистрационное имя удаленного пользователя. Кроме того, отключен режим *VerboseMode* и определены имена некоторых файлов настройки (в частности, *IdentityFile*, *AuthorizationFile* и *RandomSeedFile*). Изменения этих параметров будут восприняты демоном SSH только при получении сигнала HUP или перезапуске.

Клиентские приложения также хранят настройки в файле */etc/ssh2/ssh2_config*, который состоит из записей следующего вида:

```
VerboseMode                     no
PasswordPrompt                  "%U's password: "
AuthenticationSuccessMsg        yes
Port                            22
NoDelay                         no
KeepAlive                       yes
Ciphers                         AnyStdCipher
MACs                            AnyMAC
StrictHostKeyChecking           ask
IdentityFile                    identification
AuthorizationFile               authorization
RandomSeedFile                  random_seed
Ssh1Compatibility               yes
Ssh1AgentCompatibility          none
AllowedAuthentications          publickey,password
```

Параметры аналогичны доступным для сервера. В данном примере включены все стандартные шифры (*AnyStdCipher*), а также строгая проверка ключей узлов (*StrictHostKeyChecking*).

Каждый пользователь инструментов SSH должен выполнить ряд действий, прежде чем сможет воспользоваться службой. В частности, необходимо сгенерировать открытый и закрытый ключи, а также выбрать пароль для защиты закрытого ключа. Эту задачу можно возложить на программу *ssh-keygen*:

```
bash-2.03# ssh-keygen
Generating 1024-bit dsa key pair
   7 o.oOo.oOoo.o
Key generated.
1024-bit dsa, root@dingo, Mon Dec 18 2000 05:45:59
Passphrase : ********
Again      : ********
Private key saved to /.ssh2/id_dsa_1024_a
Public key saved to /.ssh2/id_dsa_1024_a.pub
```

В данном примере создается пара 1024-битных ключей, причем закрытый ключ защищен паролем. После ввода пароля закрытый и открытый ключи сохраняются в соответствующих файлах (*id_dsa_1024_a* и *id_dsa_1024_a.pub*). Открытый ключ выглядит так:

```
bash-2.03# cat id_dsa_1024_a.pub
---- BEGIN SSH2 PUBLIC KEY ----
Subject: root
Comment: "1024-bit dsa, root@emu, Mon Dec 18 2000 05:45:59"
AAAAB3NzaC1kc3MAAACBAOrlnlwn/3aNK2BXiFsbvh1xFRt1uaoGtDzrxOwIBry4CDTq9s
dtk+eD9qkoX6gfdnC1bW2VNcaPVRVv9WCLfJbmKlqn8vOcOhnVArAAAAFQDCpCltL+keQl
+jGHKN/oYg+EPFn4qTxmibzr9+WfgMLN2xAyQbT6ItHcoRsdobsDn/9tsfubQIvz2SVllR
ApGWRw/K8WSzL5Asv3YBVrBh2NCjv3Wimo8fiyoB8qjwHEmj/9H1ewg3Do+hFPPNOv+yzH
8HCkj3oENfs2DPC+Rf6rSvDPS94Y/AwGczbut6uOTYCdJBGJgXhOUAAACADww9VC6jerFt
QYa+iKwwwfz9G2WwAAAIEAuZDEb9LsZv6U9z3qU8il8BaILUuP2fr3dsok5y5VE2sw+plo
KkMhAFTZ/M7q/VWT33MgxDidHOqMpYORfkUrrfgIh1y/xAJ+yNa+exGo2nB6kPWnYN26ym
AIM2Rlcy1jYmMAAAHI83pIDDyQ2EOYuU8rEGrO9Z2TmyS1+gBqT8QDyvvOooX7kMZlSS8v
FOzqGGtdlk/pwbd9rQI=
---- END SSH2 PUBLIC KEY ----
```

Закрытый ключ имеет схожий формат:

```
bash-2.03# cat id_dsa_1024_a
---- BEGIN SSH2 ENCRYPTED PRIVATE KEY ----
Subject: root
Comment: "1024-bit dsa, root@emu, Mon Dec 18 2000 05:45:59"
7TGVa+Pmf9ywld+1wYeC7gWkwgrfgdvTyCnJ7Gqw3LQlIxzKVBlXmozAoR1xMVBOxnXsrJ
L1LDKJJEBaHhCNZwUhVOuiB3RyFpWvetfDdsxhH1fwtLafR4M+yHolo+uuWOHgreiZTLWd
+RW+DJGl+bxzZefY7WS7Uk1cPMo73VA1jISP8U6s3h9H9buYSk5+Iofcf28zauotM/cAX3
P2/56wAAAgoAAAAmZGwtbW9kcHtzaWdue2RzYS1uaXNOLXNoYTF9LGRoe3BsYWlufXOAAA
up9ntsfKDgseb5ClCxR539Oozt5NOQZxclpXZTaq9f+fik/lNFysXdxKQk1WLOdc8ROdCL
2YE1lWYzqe1cicsUYRjnDE1zqz/qeY7SeA/Yt5AySvcFSM4TCPkp6ppmlLyjYb7SQSAIwe
APTVnVBywNDNIcLhKJ6oOAG6qX2ho0TuGvSNPyeUymAGjhFKldogPzZXOBwBxLnE4aBc3E
D3k8Ni5QBRli4YE7Y5qa248fFFh+A5jy3cEwusn2/BwJtLIQTEcl1jKCB6z2mS13I9EO2n
wORa7TK1P/XCfg5WHrZd1yNa7Y1ztVCQVrcByarv4AhUX+j5A881QeCYfXWQbhELx8QQa/
cubMkv3bpS8bP7KMIO7GGqLzqld6hfgo+Z9OTzXEUXCJOAOLh7fxSbf36bD23FIewU
---- END SSH2 ENCRYPTED PRIVATE KEY ----
```

Теперь можно создать файл идентификации, имеющий для SSH2 тот же смысл, что пароль, добавив строку *«IdKey id_dsa_1024_a»* в файл *IdentityFile* (как правило, этот файл имеет имя *identification*).

После создания закрытого и открытого ключей для конкретного пользователя попытка подключения к новому удаленному узлу выглядит примерно так:

```
bash-2.03# ssh tasman
Host key not found from database.
Key fingerprint:
xebof-rumik-cydef-bosuf-lomoz-putat-rykec-kihyb-geher-tyneb-nyxux
You can get a public key's fingerprint by running
% ssh-keygen -F publickey.pub
on the keyfile.
```

```
Are you sure you want to continue connecting (yes/no)? yes
Host key saved to /.ssh2/hostkeys/key_22_tasman.pub
host key for localhost, accepted by root Mon Dec 18 2000 05:58:32
root's password: ********
Authentication successful.
Last login: Mon Dec 18 2000 05:56:41 from maria
Sun Microsystems Inc.   SunOS 5.8       Generic October 1999
You have new mail.
tasman:/ #
```

Контрольная сумма ключа (*key fingerprint*) может сравниваться с соответствующим открытым ключом. Для каждого нового узла, к которому производится подключение, открытый ключ узла сохраняется в отдельном файле (в случае SSH1 все открытые ключи сохраняются в одном файле).

В большинстве организаций следует вводить в действие строгие правила относительно применения инструментов удаленного доступа из семейства SSH при передаче данных в сеть Интернет. Одним из способов принудить пользователей следовать таким правилам является предотвращение применения стандартных инструментов удаленного доступа. Соответствующие определения служб (таких как Telnet и FTP) могут быть удалены из файла */etc/services* и списка служб, с которыми работает *inetd*.

Блокировка IP-портов

Основной стратегией предотвращения несанкционированного доступа к таким службам, как Telnet, FTP и *r*-команды, является удаление отображений служб (имен в номера портов) для конкретных TCP- и UDP-служб, а также блокировка их поддержки супердемоном интернет-служб (*inetd*). С этой целью могут удаляться записи из файлов */etc/services* и */etc/inetd.conf*. По умолчанию Solaris устанавливает много служб, которые не используются в ежедневной работе большинством серверов Solaris. И хотя многие администраторы считают, что об этих службах можно спокойно забыть, пока они не понадобятся, число уязвимостей во многих из этих демонов значительно увеличивает затраты на администрирование каждой системы. Например, до недавнего времени многие демоны Solaris были подвержены переполнению буфера. Это было следствием слабого кода приложений, написанных на C, поскольку проверка граничных значений, как правило, не выполнялась. Буфер, в который записываются параметры командной строки, может иметь фиксированный размер (например, 512 байт). Строка в 513 байт должна быть отвергнута, но многие демоны явным образом не обрабатывали такую ситуацию, что приводило к непредсказуемым результатам. Вот пример атаки против демона NFS *statd*, основанной на переполнении буфера (запись в системном журнале */var/adm/messages*):

```
Jan 15 12:00:01 emu statd[1024]: gethostbyname error for
^X=F7=FF=BF^X=F7=FF=BF^Y=F7=FF=BF^Y=F7=FF=BF^Z=F7=FF=BF^Z=F7=FF=BF^[=F7=FF
=BF^[=F7=FF=BF%8x%8x%8x%8x%8x%8x%8x%8x%8x%236x%n%137x%n%10x%n%192x%n\220\
220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\2
20\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\22
0\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220
\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\
220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\2
20\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\22
0\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220
\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\
220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\2
20\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\22
0\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220
\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220
```

Одним из способов решения подобных проблем является установка следующих параметров в файле */etc/system*:

```
set noexec_user_stack=1
set noexec_user_stack_log=1
```

Последствия возможных уязвимостей неиспользуемых демонов можно полностью исключить, прекратив их работу (*/etc/inetd.conf*) и удалив определения соответствующих служб (*/etc/services*). Вот фрагмент типичного файла */etc/inetd.conf*:

```
ftp     stream  tcp     nowait  root    /usr/sbin/in.ftpd       in.ftpd -l
telnet  stream  tcp     nowait  root    /usr/sbin/in.telnetd    in.telnetd
shell   stream  tcp     nowait  root    /usr/sbin/in.rshd       in.rshd
login   stream  tcp     nowait  root    /usr/sbin/in.rlogind    in.rlogind
exec    stream  tcp     nowait  root    /usr/sbin/in.rexecd     in.rexecd
comsat  dgram   udp     wait    root    /usr/sbin/in.comsat     in.comsat
talk    dgram   udp     wait    root    /usr/sbin/in.talkd      in.talkd
uucp    stream  tcp     nowait  root    /usr/sbin/in.uucpd      in.uucpd
finger  stream  tcp     nowait  nobody  /usr/sbin/in.fingerd    in.fingerd
```

Чтобы явным образом запретить работу демонов *ftp*, *telnet*, *shell*, *login*, *exec*, *comsat*, *talk*, *uucp* и *finger*, можно удалить определения, закомментировав соответствующие строки файла */etc/inetd.conf*:

```
#ftp     stream  tcp     nowait  root    /usr/sbin/in.ftpd       in.ftpd -l
#telnet  stream  tcp     nowait  root    /usr/sbin/in.telnetd    in.telnetd
#shell   stream  tcp     nowait  root    /usr/sbin/in.rshd       in.rshd
#login   stream  tcp     nowait  root    /usr/sbin/in.rlogind    in.rlogind
#exec    stream  tcp     nowait  root    /usr/sbin/in.rexecd     in.rexecd
#comsat  dgram   udp     wait    root    /usr/sbin/in.comsat     in.comsat
#talk    dgram   udp     wait    root    /usr/sbin/in.talkd      in.talkd
#uucp    stream  tcp     nowait  root    /usr/sbin/in.uucpd      in.uucpd
#finger  stream  tcp     nowait  nobody  /usr/sbin/in.fingerd    in.fingerd
```

С определениями служб в файле */etc/services* можно поступить точно так же. Вот те же службы, определенные по умолчанию в файле */etc/services*:

```
ftp             21/tcp
telnet          23/tcp
shell           514/tcp         cmd
login           513/tcp
exec            512/tcp
biff            512/udp         comsat
talk            517/udp
uucp            540/tcp         uucpd
finger          79/tcp
```

Пользуемся уже знакомым приемом и удаляем их определения */etc/inetd.conf*:

```
#ftp            21/tcp
#telnet         23/tcp
#shell          514/tcp          cmd
#login          513/tcp
#exec           512/tcp
#biff           512/udp          comsat
#talk           517/udp
#uucp           540/tcp          uucpd
#finger         79/tcp
```

Фильтрация пакетов

Всегда предпочтительно вручную отключать все службы, которые не используются, и удалять определения для этих служб. Но для пакетов, проходящих через маршрутизатор в направлении локальной сети, нужна более строгая система правил. В этом случае следует контролировать порты служб, чтобы предупредить получение несанкционированного доступа к службам локальной сети злоумышленником. Это требование позволяют выполнять системы, реализующие фильтрацию пакетов (*брандмауэры*); в таких системах маршрутизаторы могут избирательно пропускать пакеты трафика, исходя из набора определенных правил.

Наличие настроенного пакетного фильтра сделает невозможным случайное (или преднамеренное) включение службы, доступ к которой запрещен правилами, в особенности, если служба является автономным демоном, не зависящим от *inetd*. Брандмауэры запрещают запуск демонов IRC и подобных им непривилегированными пользователями на портах с номерами, большими 1024, а также предотвращают и более опасные действия. Например, троянский конь может начать вторжение в качестве диспетчера баз данных, работающего через порт с большим номером, и в будущем может обеспечить доступ внешних пользо-

вателей к таким операциям, как выборка, вставка, обновление и удаление записей.[1]

Дополнительные риски связаны с широковещательными UDP-сообщениями локальных подсетей, которые видимы для всех узлов и пользователей, способных получить минимальный уровень доступа к маршрутизатору подсети. Эти широковещательные сообщения могут содержать подробную информацию о структуре локальных сетей, которую следует скрывать от любопытных глаз. Даже если злоумышленник из внешнего мира способен всего лишь получить доступ к маршрутизатору, он сможет в большой степени изучить структуру внутренней сети, даже не обращаясь к другим ее узлам. Брандмауэр может защитить от перехвата широковещательных сообщений на маршрутизаторе.

Лучший способ защитить сеть от такого рода случайных и преднамеренных вторжений – использовать брандмауэр с фильтрацией пакетов, явным образом блокировать и разрешать доступ к определенным портам на уровне IP. Настройка большинства брандмауэров начинается с полного запрета доступа через все порты; постепенно создаются правила, позволяющие пропускать через брандмауэр определенные виды IP-трафика. Правила проверяются по очереди, и если найдено соответствие, правило применяется для пакета, а процесс поиска завершается. Например, порт *sendmail* (порт TCP 25) обычно открыт в обоих направлениях, что позволяет получать и отправлять почту из сети. В типичном наборе правил сначала будет заблокирован доступ через все порты, а затем разрешены посылка и прием пакетов через порт TCP 25.

Будь FTP запрещенной службой, доступ через порт 21 автоматически блокировался бы глобальным правилом для всех служб, если бы для конкретного целевого порта не было создано соответствующее правило. В результате исходящие и входящие FTP-сеансы были бы полностью исключены, поскольку любой порт, для которого нет явного разрешающего правила, автоматически блокируется. Возможно организовать избирательный доступ к определенным службам для отдельных узлов или целых сетей. Таким образом, веб-сервер, распространяющий финансовые данные, может быть защищен брандмауэром, который предоставляет доступ только клиентам со статическими IP-адресами и действующими подписками.

Корректно установленный брандмауэр должен предотвращать подделку IP-пакетов: ситуации, когда внешние сетевые интерфейсы утверждают, что имеют внутренние IP-адреса. Эти «поддельные» адреса используются для получения доступа к внутренним службам сети. Брандмауэры часто способны предотвращать подделку, поскольку по

[1] На самом деле запуск непривелигированными пользователями программ на портах с номерами, большими 1024, запрещается самой операционной системой Solaris, а не брандмауэрами. – *Примеч. науч. ред.*

определению каждый брандмауэр имеет два сетевых интерфейса, а правила применяются (и создаются) для каждого интерфейса в отдельности. За исключением случая перепутанных сетевых кабелей, это обеспечивает дополнительный уровень защиты от внешних атак.

Для Solaris существует два популярных брандмауэра, ни один из которых не входит в состав стандартного дистрибутива. IPFilter – бесплатная программа (*http://cheops.anu.edu.au/~avalon/ip-filter.html*), доступная для Solaris, Linux и некоторых других операционных систем. Она популярна среди любителей повозиться с исходным кодом. Sun Microsystems предоставляет альтернативу – систему SunScreen (*http://www.sun.com/software/securenet/lite/download.html*). Система SunScreen для небольших сетей распространяется бесплатно; коммерческая лицензия для крупных сетей доступна за приемлемую плату. Обе системы предлагают сходные возможности в плане фильтрации пакетов и преобразования сетевых адресов; IPFilter ориентирован на управление брандмауэром из командной строки, тогда как SunScreen включает удобный графический интерфейс для администрирования.

IPFilter

IPFilter загружается ядром в качестве модуля, и не может подвергаться атакам злоумышленников, однако может служить причиной нарушения нормальной работы ядра. IPFilter является высокопроизводительной системой с полноценной поддержкой Solaris 8 (включая 64-разрядное ядро). Среди последних новшеств IPFilter – поддержка IPv6 и MBone, распределения нагрузки «round-robin» и функциональность перенаправления, хранения и извлечения информации о трансляции адресов (NAT) плюс улучшенное отображение статистики брандмауэра. Последняя версия IPFilter – 3.4.15.

IPFilter отличается от многих пакетных фильтров тем, что проверяет все правила из файла настройки, а не только соответствующие определенному условию. Это дает возможность заблокировать весь трафик в обоих направлениях первым правилом, а затем добавлять правила, которые будут последовательно проверяться для входящих и исходящих пакетов. Важно помнить, что более общие правила должны предшествовать более частным, в противном случае действие более частных правил может быть изменено следующими за ними более общими. Например, правило

```
block in all
```

предписывает блокировку всего входящего трафика. Соответственно, ключевое слово *pass* предписывает пропускание входящего трафика:

```
pass in all
```

Ключевое слово *from* определяет происхождение пакета, а ключевое слово *to* – конечный адрес пакета. Глобальное правило блокировки пор-

тов должно предшествовать частному правилу, разрешающему прием пакетов, скажем, адресованных веб-серверу определенной сетью:

```
pass in on hme1 proto tcp from any to 192.55.204.0/24 port = 80
```

Это правило позволяет прием всех пакетов, направленных на порт 80 (веб-сервера) сетью класса C 192.68.204.0 через интерфейс */dev/hme1* (но не через интерфейс */dev/hme0*, если это явно не указано).

В обычных условиях применяются все правила IPFilter. Использование ключевого слова *quick* позволяет завершить процесс перебора правил после нахождения первого соответствия. Например, чтобы обеспечить прием пакета, если для него выполнено приведенное выше правило, следует заменить запись в файле настройки на такую:

```
pass in quick on hme1 proto tcp from any to 192.55.204.0/24 port = 80
```

Таким образом, если за предыдущей записью в файле настройки будет следовать приведенная ниже, она не будет рассмотрена, поскольку процесс перебора завершится на предыдущей записи:

```
block in quick on hme1 proto tcp from any to 192.55.204.0/24 port = 80
```

Каким образом IPFilter применяет все правила и «запоминает», какие из портов были блокированы, а какие открыты? IPFilter ведет внутренний перечень открытых и блокированных портов. Последнее из правил для конкретного порта и является тем правилом, которое будет использовано в работе.

IPFilter позволяет использовать формат CIDR для масок сетей: 192.55.21.0/24 представляет сеть класса C, а 192.55.0.0/24 сеть класса B. Кроме того, можно ссылаться на все сети с помощью ключевого слова *all*, если правило справедливо в равной степени для всех сетей (скажем, речь идет о блокировке всех портов определенного интерфейса):

```
block in quick on hme0 all
```

В состав IPFilter входит полезный инструмент *ipfstat*, который отображает статистику по числу входящих и исходящих пакетов, которые были заблокированы или пропущены, а также по числу сбоев и/или ошибок:

```
bash-2.03# ipfstat
input packets:          blocked 104855 passed 67586665 nomatch 12343 counted 0
output packets:         blocked 4334 passed 31434333 nomatch 34343 counted 0
input packets logged:   blocked 0 passed 0
output packets logged:  blocked 0 passed 0
packets logged:         input 0 output 0
log failures:           input 3898 output 0
fragment state(in):     kept 0  lost 0
fragment state(out):    kept 0  lost 0
packet state(in):       kept 3253252     lost 0
packet state(out):      kept 3243343     lost 0
ICMP replies:   0       TCP RSTs sent:  0
```

```
Result cache hits(in):  3438763 (out):  232345
IN Pullups succeeded:   1       failed: 2
OUT Pullups succeeded:  5       failed: 4
Fastroute successes:    0       failures:       0
TCP cksum fails(in):    0       (out):  0
Packet log flags seat: (0)
```

SunScreen

IPFilter имеет два серьезных недостатка – отсутствие коммерческой поддержки и отсутствие графического интерфейса администрирования, который облегчал бы добавление, удаление и обновление правил пакетного фильтра. По этим причинам многие сетевые администраторы предпочитают программный комплекс Sun *SunScreen*, который недавно был выпущен в бесплатном «облегченном» варианте. Для небольших сетей с ограниченным набором узлов и единственным маршрутизатором SunScreen Lite является легким вариантом обеспечения безопасного подключения к Интернету.

SunScreen Lite версии 3.1 существует в версиях для Solaris SPARC и Solaris Intel. Эта версия поддерживает простой протокол сетевого управления (Simple Network Management Protocol, SNMP), MBone по IPv6, а также высокоскоростные Ethernet-соединения.

В более крупных сетях могут быть уместны дополнительные возможности полного продукта SunScreen, в частности работа в специальном режиме «невидимости», который предназначен для сокрытия факта работы системы SunScreen. Кроме того, доступна поддержка кластеров высокой доступности, а также систем с четырехпортовыми картами, за пределами стандартной конфигурации маршрутизаторов с двумя сетевыми интерфейсами. В SunScreen присутствует встроенная функциональность proxy-сервера, хотя в паре с SunScreen может работать система-посредник от сторонних разработчиков (такая как Socks, речь о которой пойдет позже).

Как уже говорилось, управление SunScreen осуществляется в основном с помощью графической среды, доступ к которой организован через веб-броузер Netscape Navigator, поставляемый в составе Solaris 8. Доступ к интерфейсу может быть ограничен локальным узлом, и в этом случае работа с ним должна осуществляться с системной консоли; при этом существует возможность удаленного доступа через защищенное соединение. Шифрование лексем идентификации гарантирует безопасность удаленного доступа.

После установки (которая производится с помощью сценария */opt/SUNWicg/SunScreen/bin/ss_install*) систему SunScreen необходимо настроить, выбрав режим маршрутизации или невидимости, методы настройки – локальные или удаленные, а также связанные с ними уровни защищенности. Кроме того, можно настроить поддержку службы имен (скажем, DNS).

В процессе установки создаются базы данных *vars*, *authuser*, *proxyuser* и *logmacro*. После этого инициализируется массив ключей простого управления ключами для интернет-протоколов (Simple Key-management for Internet Protocols, SKIP), и создается новая база сертификатов (как правило, в файле */etc/skip/certdb*). В SunScreen применяется SKIP для поддержки шифрования сеансов по открытым каналам с целью администрирования. На смену этой системе в большинстве случаев пришел более общий протокол IP-безопасности (IP Security Protocol, IPsec). Этот протокол впервые был реализован в рабочей среде Solaris 8 и наиболее полезен для создания частных виртуальных сетей (VPN) в Интернете. К IPsec мы еще вернемся позже в этой главе. Менеджер ключей SKIP реализован в виде демона и в первый раз запускается после установки.

Для активизации брандмауэра следует перезагрузить систему, что позволит ядру загрузить необходимые модули. В случае корректной установки программного пакета демон менеджера ключей SKIP запускается автоматически, а веб-сервер начинает принимать соединения через порт 3852. Если были установлены инструменты удаленного администрирования, этот порт будет доступен для удаленных обращений. Альтернативой является запуск Netscape Navigator на системной консоли и обращение к интерфейсу администрирования по адресу *http://localhost:3852/*.

Имя пользователя по умолчанию *admin*, пароль по умолчанию – *admin*. Первое, что следует сделать после запуска системы, – сменить пароль, обратившись к веб-интерфейсу администрирования. После входа в систему отображается основной информационный интерфейс. Он позволяет администратору просмотреть журнал брандмауэра, в который попадают записи о принятых и отвергнутых пакетах, а также статистика по соединениям иного рода.

Чтобы брандмауэр приносил пользу, необходимо создать правила, аналогичные применяемым в IPFilter, – с помощью ключевых слов *ALLOW* и *DENY*, явным образом разрешающих или запрещающих прохождение пакетов через конкретные интерфейсы. Исходный и конечный адреса могут задаваться с помощью масок (которые являются заменой ключевому слову *any* пакета IPFilter). Правила *ALLOW* могут включать дополнительные настройки, в частности *LOG_DETAIL*, *LOG_NONE*, *LOG_SUMMARY*, *SNMP* и *SNMP_NONE*. То же верно и для правил *DENY*: *ICMP_HOST_FORBIDDEN*, *ICMP_HOST_UNREACHABLE*, *ICMP_NET_FORBIDDEN*, *ICMP_NET_UNREACHABLE*, *ICMP_NONE*, *ICMP_PORT_UNREACHABLE*, *LOG_DETAIL*, *LOG_NONE*, *LOG_SUMMARY*, *SNMP* и *SNMP_NONE*.

Все активные правила отображаются в разделе интерфейса Policy Rules. После создания записи можно удалять и изменять, а также в любой момент добавлять новые.

Аналогично IPFilter, SunScreen может применяться для преобразования сетевых адресов (network address translation, NAT). Система NAT действует на маршрутизаторе/брандмауэре и преобразует частные IP-адреса сети, защищенной брандмауэром, во внешние IP-адреса. К примеру, сеть класса C может иметь видимую IP-маску сети 204.12.32.0, но работать с частной IP-маской 10.12.32.0. Соответственно веб-сервер, защищенный брандмауэром, с частным адресом 10.12.32.16 будет иметь внешний адрес 204.12.32.16. Более подробные сведения по NAT содержатся в документе RFC 1631 (*http://info.internet.isi.edu/in-notes/rfc/files/rfc1631.txt*).

Kerberos 5

Многие из описанных в этой главе стратегий предотвращения несанкционированного доступа к сетевым службам концентрируют внимание на обнаружении и блокировке подобных попыток в сетях с минимальными архитектурными ограничениями удаленного доступа. Однако возможно еще более усилить требования к проверке подлинности с помощью распределенной системы проверки подлинности – NIS+, разработанной Sun, либо Kerberos, разработанной университетом MIT (*http://web.mit.edu/network/kerberos-form.html*). Kerberos 5 реализует среду проверки подлинности повышенной защищенности для Solaris (и многих других операционных систем) и всех приложений обслуживания, работающих с другими серверами проверки подлинности, физически отделенных от узлов-клиентов и узлов, предоставляющих службы. Система паспортов (*tickets*), основанная на сильных криптографических алгоритмах, гарантирует, что только прошедшие проверку подлинности клиенты получат доступ к определенным службам определенных узлов на ограниченный период времени, по истечении которого паспорт необходимо будет продлить.

Как и в случае ключей сеансов SSH, это означает, что права доступа, предоставляемые сервером Kerberos, действуют в течение ограниченного времени, предупреждая возможный взлом прямым перебором. Как в SSH, все лексемы проверки подлинности шифруются при передаче по сети. Главным документом является паспорт, идентифицирующий и подтверждающий подлинность пользователя или службы (обобщенно – участников), а также содержащий данные авторизации. Перечень действующих документов хранится в базе данных на основном сервере паспортов определенного владения (владения и серверы паспортов описаны ниже).

Sun предоставляет доступ к проверке подлинности по Kerberos с помощью продукта SEAM (Sun Enterprise Authentication Mechanism). SEAM версии 1.0 содержит полноценные реализации клиентской и серверной частей Kerberos 5; однако в стандартный дистрибутив Solaris 8 включена только клиентская часть Kerberos 5. Чтобы воспользоваться службами Kerberos 5, не приобретая SEAM 1.0, следует загрузить

и установить MIT-реализацию Kerberos 5 (*http://web.mit.edu/network/kerberos-form.html*).

Одним из преимуществ системы Kerberos перед специальными системами, такими как NIS+, является поддержка широкого спектра систем. Клиенты, работающие с одной операционной системой (скажем, Windows 2000), могут запрашивать паспорта у сервера проверки подлинности, работающего на другой операционной системе (скажем, Solaris), с целью обращения к службе на сервере, работающем под управлением третьей операционной системы (скажем, Linux). Такого рода тесная интеграция операционных систем является очень удобной в гетерогенной среде: если хотя бы одна из применяемых операционных сред не поддерживается распределенной системой проверки подлинности, клиенты и серверы вновь станут использовать небезопасные способы обмена документами идентификации и проверки подлинности. Дополнительным преимуществом применения Kerberos является прозрачность для пользователя в большинстве случаев.

Если вы уже работаете с SSH, то можете заинтересоваться причинами установки еще и Kerberos. Одна из причин – SSH требует хранения копий всех хотя бы раз использованных открытых ключей в локальной файловой системе, причем для каждого пользователя. Представьте, что на основной системе 1 000 пользователей регулярно применяют SSH для подключения к серверу E10000 с целью работы с базой данных. В конечном итоге, каждый пользователь должен хранить копию открытого ключа сервера баз данных, что связано с крупными проблемами администрирования, в особенности, если имя узла или IP-адрес сервера базы данных в какой-то момент изменится. Более эффективной будет защищенная централизация проверки подлинности для всех служб.

Kerberos обеспечивает проверку подлинности как для служб, так и для отдельных пользователей. Кроме того, диапазон приложений, с которыми работает Kerberos, гораздо больше, чем у SSH: приложения могут изменяться для работы со службами Kerberos. Это позволяет централизовать управление всеми службами проверки подлинности, сократить нагрузку, связанную с сопровождением многочисленных пользователей и систем администрирования служб. У данного подхода есть и недостатки: если служба Kerberos будет выведена из строя (например, DOS-атакой), проверка подлинности станет невозможной, и службы, работающие с Kerberos, перестанут обрабатывать запросы клиентов. Риск подобных атак может быть минимизирован разумным применением технологии брандмауэров.

В Kerberos существует понятие дискретных «владений», аналогичных доменам NIS+; проверка подлинности узлов и служб в пределах владения производится двумя серверами ключей-паспортов: основным сервером и дополнительным. Оба сервера способны выдавать паспорта и проверять подлинность участников по запросу, но только основной

сервер должен обладать полномочиями на обновление основной базы данных документов.

Определившись с сетевой архитектурой для Kerberos (в частности, речь идет о службах основных и дополнительных серверов, а также об отображении владений в домены NIS+ или DNS), мы можем с легкостью создать файл настройки сервера (он, как правило, имеет имя */etc/krb5/krb5.conf*):

```
[libdefaults]
        default_realm = CASSOWARY.NET
[realms]
          CASSOWARY.NET = {
                kdc = master.cassowary.net
                kdc = slave.cassowary.net
                admin_server = master.cassowary.net
        }
```

Файл настройки определяет подразумеваемое владение *CASSOWARY.NET*, в котором существует два паспортных сервера (основной и дополнительный, *master* и *slave*). За обновление базы данных на узле *master.cassowary.net* отвечает *admin_server*.

IPsec

Одним из наиболее долгожданных нововведений Solaris 8 стало включение технологии частных виртуальных сетей (Virtual Private Network, VPN) в виде протокола IPsec, совместимого со стандартными IPv4 и IPv6. Такая совместимость протоколов облегчает планирование перехода на IPv6. В сетях VPN используется подход к безопасности, отличный от тех, что применяются в других технологиях удаленного доступа: шифрование содержимого передаваемых пакетов. Защита трафика в IPsec начинается на уровне пакетов, для дейтаграмм которых выполняется ряд проверок и балансировок. Такая технология IP-туннелирования позволяет разделенным сетям производить безопасный обмен данными, не опасаясь перехвата.

IPsec реализует поддержку туннелирования и шифрования данных IP. Для согласования протоколов между маршрутизаторами различных сетей и выполнения надежной проверки подлинности на основе ключей в IPsec применяется система обмена ключами Internet Key Exchange (IKE). Сочетание туннелирования с шифрованием делает доступ к данным без авторизации и изменение содержимого дейтаграмм при их передаче между узлами практически невозможным. Но за такую функциональность приходится платить – текущая реализация IPsec в Solaris далека от идеала; файлы настроек и правил редактируются вручную, графический интерфейс администрирования отсутствует.

IPsec основывается на трех ключевых понятиях: связях безопасности (security association, SA), заголовках идентификации (authentication

header, AH), а также надежной передаче данных (encapsulated security payload, ESP). Чтобы два узла (например, маршрутизаторы двух сетей) могли обмениваться данными по надежному каналу, для них необходимо создать две SA-связи. Например, если необходимо использовать маршрутизаторы *router.paulwatters.com* и *router.cassowary.net* для создания VPN-сети на основе сетей *paulwatters.com* и *cassowary.net*, следует создать SA-связь между узлами *router.paulwatters.com* и *router.cassowary.net*, а также между узлами *router.cassowary.net* и *router.paulwatters.com*. SA-связи обладают различными характеристиками безопасности, которые должны быть согласованы между (по меньшей мере) двумя узлами, посредством которых формируется частная виртуальная сеть. Управление SA-связями осуществляется с помощью команды *ipseckey*.

Механизм защиты AH-заголовков работает только между узлами, для которых существуют SA-связи. Заголовок передается с указателем параметров безопасности (security parameters index, SPI) и позволяет считать данные дейтаграммы подлинными, а их порядок доставки – корректным. Это важный момент, поскольку многие IP-атаки основаны на подмешивании данных в незащищенные дейтаграммы. Не все поля дейтаграмм могут быть защищены, поскольку их содержимое время от времени может меняться.

Заголовки AH позволяют сохранять структурную целостность дейтаграмм и предотвращают их изменение, но не защищают данные, которые передаются в дейтаграммах. И здесь на арене появляется ESP. Механизм защиты ESP также работает только для узлов с SA-связями. Аналогично AH-заголовкам, ESP-данные передаются с параметром SPI. При этом для шифрования дейтаграмм используются алгоритмы Data Encryption Standard (DES) и Triple-DES (3-DES). Таким образом, передаваемые по IP данные оказываются надежно защищены от несанкционированного доступа. Разумеется, шифрование и дешифрование всех данных с применением ESP связано со снижением производительности, которое следует принимать во внимание при планировании VPN-сети на базе IPsec.

AH и ESP – два абсолютно независимых аспекта IPsec, эти технологии могут использоваться как по отдельности, так и в паре. Например, в сети, распространяющей общедоступную информацию из компетентного источника, скорее всего, будет актуальным сохранение структурной целостности дейтаграмм с помощью заголовков AH. С другой стороны, банку требуется не только целостность дейтаграмм, но и сокрытие информации о транзакциях клиентов, и в таком случае следует использовать AH в паре с ESP.

Идентификация в IPsec основана на контрольных суммах MD5. Этот алгоритм вычисляет 128-битную цифровую подпись (известную как дайджест сообщения) на основе данных любого вида (в частности, на

основе дейтаграмм). В процессе проверки подлинности IPsec применяет MD5 как для данных, так и для ключей.

Правила безопасности в IPsec определяются при помощи команды *ipsecconf*. Настройки *ipsecconf* хранятся в файлах */etc/inet/ipsecinit.conf* и */etc/inet/ipsecpolicy.conf*. Примеры записей из файла */etc/inet/ipsecinit.conf* приведены ниже. Весь FTP-трафик через порт 21 защищен заголовками AH, при этом используется любой из доступных алгоритмов шифрования (DES или 3-DES), а также проверка подлинности по MD5:

```
{dport 21} apply {encr_algs any encr_auth_algs md5 sa shared}
{sport 21} permit {encr_algs any encr_auth_algs md5}
```

Эти правила действуют для всех клиентов и узлов сети. Возможно также создать правила для конкретных сетей при помощи только заголовков AH. В следующем примере заголовки AH используются для защиты только данных сети класса C (203.54.58.0), в качестве алгоритма идентификации используется MD5:

```
{daddr 203.54.68.0/24} apply {auth_algs md5 sa shared}
{saddr 203.54.68.0/24} permit {auth_algs md5}
```

SOCKS Internet Proxy

Мы рассмотрели применение брандмауэров для эффективной блокировки доступа к уязвимым службам, которые не должны предоставляться внешним клиентам (в частности, *r*-команды). Я также упоминал, что некоторые из брандмауэров умеют выступать в роли посредников (*proxies*) для защищенных узлов. В общем случае посредником является человек, который выполняет определенную задачу за других. Скажем, владельцы акций часто назначают своими представителями брокеров, которые ходят на совещания и имеют право голоса. Таким образом, отдельные акционеры освобождаются от необходимости участвовать в совещаниях, чтении бумаг и т. д., вместо этого они могут сосредоточиться на своих главных задачах (покупке и продаже фондов). Таким же образом сервер-посредник (proxy-сервер), расположенный на брандмауэре, может взять на себя заботу о многих аспектах защиты отдельных узлов сети, с которыми невозможно справиться только фильтрацией пакетов.

Чаще всего применяется конфигурация, в которой proxy-сервер принимает соединения от клиентов, защищенных брандмауэром, через единственный порт (обычно 1080), а затем устанавливает соединения с внешним миром для каждого допустимого порта и протокола. Клиент может воспользоваться proxy-сервером для подключения к удаленному веб-серверу. В таком случае клиент устанавливает TCP-соединение с сервером-посредником через порт 1080, а сервер-посредник устанавливает TCP-соединение с целевым веб-сервером через порт 80. Передача данных в обратном направлении происходит по тому же маршруту.

Как вариант клиент может подключаться к серверу Telnet удаленной системы. В таком случае он устанавливает TCP-соединение с proxy-сервером через порт 1080, а тот, в свою очередь, подключается к удаленному серверу через порт 23. В обратном направлении данные точно так же передаются по установленному маршруту. Заметим, в обоих случаях целевой порт proxy-сервера имеет номер 1080, тогда как взаимодействие с целевыми службами удаленных систем происходит через стандартные порты (порт 80 для веб-сервера, порт 23 для сервера Telnet).

Таким образом, число «открытых» на брандмауэре портов, необходимых для организации доступа клиентов к сети Интернет, может быть сокращено до одного: 1080, порта сервера-посредника. Таким образом производится централизация доступа к службам удаленных систем Интернета, упрощается сопровождение и администрирование клиентских систем и предотвращается ряд известных атак на систему безопасности, связанных с использованием стандартных портов брандмауэра. Сервер-посредник позволяет избежать работы с клиентами через стандартные порты брандмауэра.

После установки и настройки брандмауэра и запуска proxy-сервера пользователи получают прозрачный доступ к внешним службам; то есть пользователь клиентской системы, как правило, может и не знать, что соединения с внешним миром проходят через сервер-посредник. В случае снижения производительности может потребоваться подстройка сетевых параметров.

В этом разделе мы рассмотрим настройку proxy-сервера SOCKS, который описан в следующих документах:

```
RFC 1928: SOCKS Protocol Version 5
RFC 1929: Username/Password Authentication for SOCKS V5
RFC 1961: GSS-API Authentication Method for SOCKS Version 5
```

Внутренний сервер SOCKS

Внутренний SOCKS-сервер является наиболее подходящим решением в локальных сетях, маршрутизаторы которых находятся в ведении сторонних организаций (например, в ведении интернет-провайдера). В таком варианте клиенты указывают SOCKS-сервер в качестве сервера-посредника, а SOCKS-сервер передает все запросы маршрутизатору. Чтобы такая архитектура была безопасной, маршрутизатор должен блокировать все исходящие соединения, обработкой которых должен заниматься сервер SOCKS (в частности, Telnet, FTP, HTTP и т. д.), и принимать соединения только от самого сервера SOCKS.

В следующем примере сеть 204.154.68.0/24 обслуживается SOCKS-сервером 204.154.68. 32. В файле */etc/libsocks5.conf* необходимо создать такие записи для клиента:

```
noproxy - 204.154.68. - -
socks5 - - - - 204.154.68.32
```

Соединения с узлами локальной сети (204.154.68.0/24) не должны проходить через сервер-посредник, тогда как все остальные запросы должны передаваться посреднику на адрес 204.154.68.32.

В файле */etc/socks5.conf* необходимо создать следующие записи для сервера:

```
auth - - -
permit - - 204.154.68. - - -
```

Все узлы локальной сети могут работать с сервером, но любые внешние входящие соединения будут игнорироваться.

Сервер SOCKS на базе маршрутизатора

Более популярный подход связан с установкой сервера SOCKS непосредственно на маршрутизаторе. (Это уместно, когда маршрутизатор находится в ведении локального администратора.) В этом случае клиенты просто указывают маршрутизатор в качестве proxy-сервера, а сервер SOCKS, работающий на маршрутизаторе, напрямую обрабатывает запросы на внешние соединения.

Разумеется, поскольку маршрутизатор имеет минимум два интерфейса, соединения клиентов SOCKS должны проходить через «внутренний» интерфейс, а все внешние соединения – через «внешний».

Чтобы подобная архитектура была защищенной, маршрутизатор должен блокировать все исходящие через внутренний интерфейс соединения, обработка которых возложена на сервер SOCKS (Telnet, FTP, HTTP и т. д.), а также разрешать соединения с внутренним интерфейсом через порт 1080. Прямой доступ внутренних клиентов к внешнему интерфейсу также должен блокироваться.

В следующем примере внутренняя сеть 10.154.68.0/24 обслуживается сервером SOCKS 10.154.68.32. В файле */etc/libsocks5.conf* следует создать такие записи для клиента:

```
noproxy - 10.154.68. - -
socks5 - - - - 10.154.68.32
```

Они оговаривают, что подключения к узлам локальной сети (10.154.68.0/24) должны устанавливаться в обход proxy-сервера, а все прочие запросы должны передаваться этому серверу на адрес 10.154.68.32.

В файле */etc/socks5.conf* следует создать такие записи для сервера:

```
auth - - -
interface 10.154.68. - 10.154.68.32
interface - - 204.154.68.32
permit - - 10.154.68. - - -
```

Весь трафик из внутренней сети, направленный во внешний мир, будет проходить через внутренний интерфейс маршрутизатора (10.154.

68.32). Внешние запросы будут поступать через proxy-сервер на внешний интерфейс (204.154.68.32).

MD5

Большая часть этой главы была посвящена защите данных, передаваемых по сети, а также защите узлов сети от внешних атак. Однако большинству администраторов в какой-то момент придется столкнуться с перспективой того, что злоумышленник сумел проникнуть в систему. Как обнаружить активность такого злоумышленника в системе? Очевидно, это зависит в большой степени от установленных приложений и средств защиты данных сети. Например, если пользователь получит доступ к непривилегированной учетной записи на маршрутизаторе, корректно настроенный брандмауэр не позволит ему проникнуть дальше в сеть.

Одним из популярных способов проникновения в систему является «троянский конь»: злоумышленник получает доступ к системе с полномочиями администратора и заменяет ключевые приложения и файлы своими вариантами. Измененные файлы часто используются для атаки других сетевых площадок (например, для выполнения распределенной DOS-атаки посредством модифицированного демона *inetd*). Одним из способов противодействия такого рода атакам является применение алгоритма генерации цифровых подписей, например дайджестов сообщений MD5. Алгоритм MD5 может использоваться для создания контрольных сумм для всех приложений и файлов настройки системы. Список контрольных сумм следует хранить в надежном месте, оптимально – на другом компьютере. Периодически можно повторять прогон вычислений MD5 для всех приложений, для которых были записаны контрольные суммы. Если MD5-подпись неожиданно изменилось, это может свидетельствовать о проникновении злоумышленника. Возможно также изменение контрольной суммы в результате обновления приложения. Вот контрольные суммы MD5 для файлов */bin/ls* и */bin/cat*:

```
bash-2.03# ./md5 /bin/ls
MD5 (/bin/ls) = 050a9443cf408401b9ce83f3e3f1d91f
bash-2.03# ./md5 /bin/cat
MD5 (/bin/cat) = f68cdcd7f2c0423428e0e6a9d22e413a
```

Великолепным источником информации с многочисленными примерами, посвященными настройке пакетных фильтров, proxy-служб и служб проверки подлинности, является книга E. Zwicky, S. Cooper и B. Chapman «Building Internet Firewalls»[1] (O'Reilly, 2000).

[1] Цвики Э., Купер С. и Чапмен Б., «Создание защиты в Интернете», 2-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2002.

10

Сетевые информационные системы

В этой главе рассмотрены этапы создания информационных систем с веб-интерфейсом на основе трехзвенной интранет-системы. Целью главы является не обучение программированию на Java, но представление одного полностью работоспособного примера создания на базе Solaris сервера приложений (Apache/JServ), сервера баз данных (Postgres), а также установки и тестирования приложения. Для изучения примера понадобятся навыки и знания, полученные при чтении предшествующих глав. Применение распределенной сетевой инфраструктуры приложений демонстрируется на примере посредника запросов к объектам (Object Request Broker, ORB) VisiBroker CORBA от VisiBroker.

Для сетевого администратора Solaris одной из сложнейших задач является поддержка приложений информационных систем в рабочем окружении. Управление комбинацией веб-серверов, серверов приложений, серверов баз данных и серверов связующего ПО сопряжено со сложностями, которые нелегко одолеть. Сетевые администраторы не умеют писать программы на языке Java, не являются специалистами в CORBA или администраторами баз данных, но при этом успешное завершение проектов упомянутых профессионалов зависит от надежности операционной системы и локальной сети, которые как раз находятся в ведении сетевых администраторов. К счастью, в том, что касается поддержки современных распределенных информационных систем, Solaris является идеальным выбором, и значительная часть программного обеспечения, производимого сторонними разработчиками и предназначенного для применения в корпоративной среде, доступна в версиях для Solaris.

Например, язык Java, разработанный Sun Microsystems, комплектуется отличной виртуальной машиной для Solaris SPARC и Solaris In-

tel. Java обычно применяется для объединения клиентских интерфейсов, связующего ПО и серверов объектно-ориентированных хранилищ данных. Именно Java лежит в основе успеха многих информационных систем с веб-интерфейсом. В реализации Java активно применяются потоки (threads) и легковесные (lightweight) процессы. Эти абстрактные операции, выполняемые виртуальной машиной Java, находят отражение в методах многопоточных приложений и облегченных процессов Solaris. Такая однородность выглядит особенно привлекательно на фоне относительно низкой производительности приложений, использующих интерфейс CGI (Common Gateway Interface, общий интерфейс шлюза) для выполнения на сервере однопоточных приложений по каждому из клиентских запросов, полученных веб-сервером.

В этой главе мы шагнем за пределы простого изучения установки и настройки служб Solaris, изучив типичный сценарий применения этой ОС в рабочей среде. И хотя конкретные параметры разработки вашей рабочей среды могут отличаться от рассмотренных здесь, подход в целом остается неизменным. Мы рассмотрим ключевые возможности, которые должны быть реализованы системами Solaris в целях поддержки работы распределенных информационных систем с веб-интерфейсом:

- Высокоскоростной веб-сервер
- Виртуальная машина Java
- Сервер приложений Java с поддержкой сервлетов и серверных страниц Java (JSP)
- Распределенный сервер объектов с поддержкой удаленного вызова методов (RMI, Remote Method Invocation), архитектуры CORBA (Common Object Request Broker Architecture) либо Enterprise JavaBeans (EJB)
- Сервер баз данных, соответствующий стандарту SQL
- Клиентская библиотека и диспетчер соединений JDBC, обеспечивающие возможность установления связи между базой данных и клиентом через слои связующего ПО

В этой главе мы подробно рассмотрим каждую из перечисленных служб, а затем объединим их в цельное приложение с веб-интерфейсом.

Информационные веб-системы

Я начну рассказ с детального рассмотрения распространенных архитектур и компонентов, на которых строятся современные информационные веб-системы. Эти информационные системы могут использоваться как для целей распространения содержательной или справочной информации, например для организации библиотечного каталога, так и для создания полноценных приложений электронной коммер-

ции, реализующих транзакционную обработку фондовых операций или заказов товаров со склада. Стиль и наполнение веб-сайтов, поддерживающих такие приложения, варьируются в широких пределах: от информационных бюллетеней до сложных банковских комплексов. В этой книге мы не станем касаться тонкостей процесса разработки подобных приложений, но сфокусируемся на создании инфраструктуры, без которой эти приложения не смогут существовать, и начнем с веб-сайтов.

Веб-сайты

Среда World Wide Web в различных формах существует уже более десяти лет. Сегодня она отвечает за огромную долю трафика в сети Интернет, а большинство крупных организаций по всему миру обзавелись собственными сайтами и домашними страницами. Отдельные компании пошли дальше и предоставляют доступ в реальном времени к серверным приложениям, позволяющим оплачивать счета и производить расчеты по кредитным картам, что сокращает потребность в центрах телефонной поддержки и обслуживания клиентов. Такую экономию следует сравнить с совокупной стоимостью владения (Total Cost of Ownership, TCO) для процесса создания и сопровождения информационной веб-службы. Вот факторы, влияющие на показатель TCO для веб-сайта:

- Найм персонала для сопровождения информационных систем
- Приобретение программного обеспечения либо найм консультантов и разработка заказного ПО
- Приобретение аппаратного обеспечения: компьютеров, сетевых устройств, систем хранения данных
- Обучение пользованию системой существующего персонала и клиентов

Показатель TCO сильнее всего зависит от типа веб-сайта, который необходимо создать. Простая система, в которой нет динамических составляющих, стоит недорого, а приложение с доступом к базе данных и круглосуточной технической поддержкой, очевидно, обойдется гораздо дороже.

С целью оценки TCO веб-сайты можно разбить на четыре категории, речь о которых пойдет далее.

Статическое наполнение

Стандартный формат для веб-сайта – это набор статических страниц на языке HTML (HyperText Markup Language, язык гипертекстовой разметки), которые хранятся в файловой системе сервера. На большинстве веб-сайтов данные представлены в формате статического HTML. Формат HTML построен на основе конкретных меток (tags, теги), кото-

рые анализируются броузером клиента и соответствующим образом визуализируются. Например, текст между открывающим тегом <TITLE> и закрывающим тегом </TITLE> сохраняется в качестве объекта TITLE (заголовок) и обычно отображается в верхней части окна броузера. Кроме того, в HTML-страницах могут храниться данные, не предназначенные для отображения, но играющие роль при описании документов и их содержимого. Тег <META> содержит подобные метаданные. HTML-страницы строятся по простой объектной модели, которая состоит из тегов <HEAD> и <BODY>.

Динамическое наполнение

Динамический HTML – это вариант стандартного формата веб-сайтов, который дополнен языком выполняемых на стороне клиента сценариев JavaScript. В рамках объектной модели документов HTML сценарии оформляются тегом <SCRIPT>. Они могут применяться для улучшения интерфейса веб-сайта и очень полезны при проверке корректности заполнения форм HTML. В случае «толстого» клиента могут применяться апплеты Java, выполняемые на машине клиента, но в настоящее время поддержка апплетов в броузерах ограничена неэффективными графическими библиотеками конкретных платформ.

Common Gateway Interface (CGI)

Приложения CGI выполняются на сервере и могут использоваться совместно как со статическими, так и с динамическими страницами HTML. Данные от клиента могут передаваться посредством метода POST, а данные от сервера могут получаться посредством метода GET. CGI-приложения обычно создаются на языках Perl, C или Java и часто используются для построения поисковых систем и интерпретации данных из форм HTML. CGI-приложения способны работать с базами данных.

Многозвенные приложения

Многозвенные (multitier) приложения – это по меньшей мере один шаг вперед по сравнению с CGI, поскольку они поддерживают распределенные базы данных, управление объектами, службы каталогов и полный спектр служб, которые могут потребоваться предприятию от стандартной информационной системы. Многие корпоративные компоненты сосредоточены на серверных уровнях Java, в частности корпоративные компоненты (EJB), инкапсулирующие функции логики приложений, такие как вставка и обновление таблиц баз данных. Для поддержки существующих клиентских приложений могут использоваться «местные» протоколы, такие как CORBA.

В данной главе мы сосредоточимся на изучении многозвенных приложений и ресурсов, необходимых для их поддержки.

Многозвенные архитектуры

Простейшее многозвенное веб-приложение состоит из трех звеньев (рис. 10.1). Эта система содержит клиентский HTML-интерфейс, который динамически генерируется Java-сервлетом, работающим на сервере. В сервлетах Java существует функциональность для интерпретации параметров GET и POST, передаваемых клиентом серверу. С недавнего времени сервлеты заменили неэффективные, медленные CGI-приложения, что позволило существенно повысить производительность приложений и скорость обмена запросами и ответами.

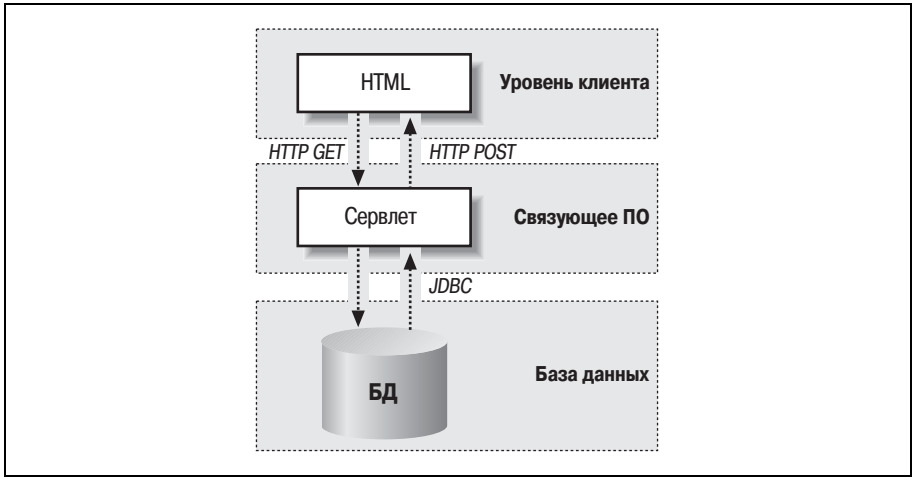

Рис. 10.1. *Типичное трехзвенное приложение электронной коммерции*

CGI является неэффективной методологией в основном потому, что веб-серверы, поддерживающие CGI, представляют собой системы, в которых единовременно может существовать много последовательных процессов. В таких серверах велики непроизводительные издержки – каждый запрос пользователя, достигающий цели, порождает новый процесс Apache и процесс, в котором работает собственно CGI-приложение. Соответственно, для тысячи пользователей, одновременно обратившихся к веб-сайту, потребуется две тысячи процессов, что для многих систем будет серьезной нагрузкой (хотя ОС Solaris вполне справляется с таким числом процессов).

Предпочтительнее использовать многопоточный сервер приложений, исполняющий Java-сервлеты, поскольку для выполнения сервлета требуется всего один процесс. Таким образом, для тысячи пользователей требуется лишь один процесс, работающий в 1000 потоков (нитей), что в итоге и позволяет снизить загрузку системы. Другим плюсом сервлетов является легкость доступа к серверным СУБД с помощью классов JDBC (Java Database Connectivity classes). Эти классы позволяют применять стандартные операторы SQL в коде сервлетов. В таком

простейшем случае практически нет разделения представления от логики работы, но сервлеты, разработанные в таком стиле, часто используются для выполнения запросов к базе данных, не требующих промежуточной обработки (например, поиска по каталогу).

Архитектура типичного четырехзвенного веб-приложения (рис. 10.2) содержит сервлет, генерирующий HTML-интерфейсы и интерпретирующий параметры методов GET/POST. Любой запрос на выполнение функций логики приложения передается инкапсулированным компонентам (Enterprise JavaBeans), которые отвечают за связь с сервером БД с помощью классов JDBC. В данном варианте представление и логика приложения полностью разделены, а собственно подход оптимален с точки зрения архитектуры. Однако каждый новый слой абстракции снижает производительность и повышает риск возникновения сбоя на каком-либо из уровней приложения из-за дефекта кода или исключения.

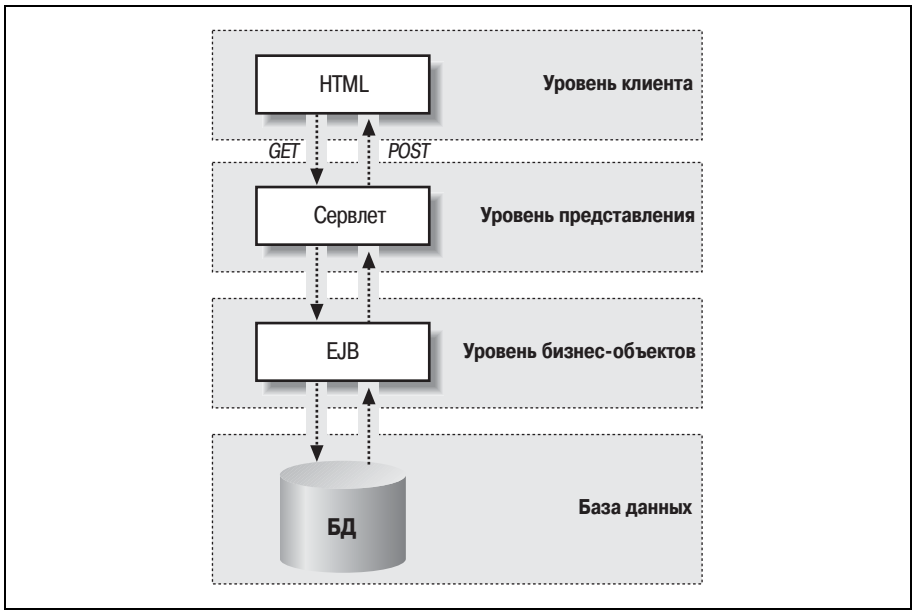

Рис. 10.2. *Типичное четырехзвенное приложение электронной коммерции*

Во многих веб-приложениях ситуация осложняется необходимостью интегрировать данные из HTML-форм, генерируемых сервлетом, с данными, поступающими от существующих клиентов (рис. 10.3). Процесс можно упростить объединением информации в одну базу данных, которая служит хранилищем для всех клиентов, а также централизацией управления ресурсами и доступом клиентов к БД с помощью технологии запросов к объектам CORBA. Применение открытых стандартов, таких как Java и CORBA, означает, что конкретные программные модули, выбранные для выполнения задачи на каждом из концептуаль-

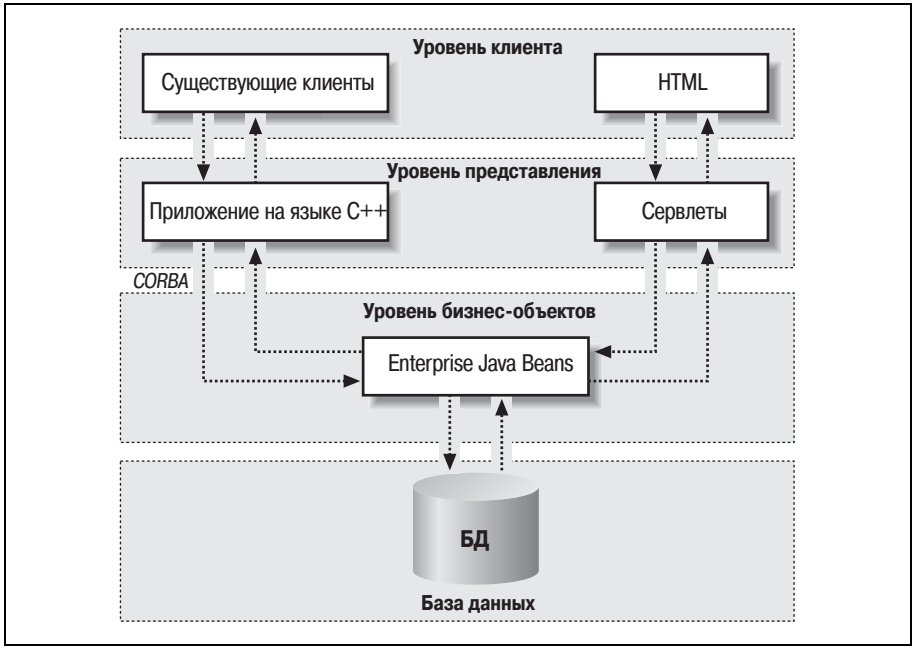

Рис. 10.3. *Интеграция Java с существующими приложениями посредством CORBA*

ных уровней архитектуры, подвержены замене. Например, сервер приложений Inprise и сервер BEA WebLogic оба соответствуют требованиям для сертификации J2EE (Java 2 Enterprise Edition). Таким образом, целые системы и отдельные компоненты, разработанные с применением любого из этих серверов, могут переноситься на другой с минимальными подстройками и без необходимости перекомпилировать код.

В следующих разделах мы изучим технологии, которые будем применять для реализации каждого из звеньев.

Технология клиентов

В настоящее время существует четыре метода разработки веб-клиентов: стандартный HTML, динамический HTML, компилируемые серверные страницы, апплеты. Стандартный HTML обеспечивает максимальную межплатформенную совместимость для броузеров Netscape Navigator и Microsoft Internet Explorer. Но в языке HTML отсутствуют многие возможности, к которым уже привыкли пользователи веб-сайтов (например, динамическое отображение элементов меню, основанное на действиях пользователя).

В динамическом HTML применяется язык сценариев, выполняемых на стороне клиента; с его помощью можно создавать меню, выпадаю-

щие списки и другие элементы страниц без необходимости обращаться к серверу. Это увеличивает объем вычислительных ресурсов, потребляемых клиентом, но сокращает сетевой трафик в сравнении с более привязанными к серверу технологиями, такими как компилируемые серверные страницы или апплеты. Наиболее широко применяемым языком сценариев, выполняемых на стороне клиента, на сегодня является JavaScript.

Компилируемые серверные страницы позволяют HTML-разработчикам создавать статические HTML-документы, содержащие специальные блоки с операторами языков программирования, которые компилируются до выполнения либо во время вызова. Это очень удобный способ доступа к серверным данным, но он создает высокую нагрузку на серверы, которым приходится заниматься компиляцией страниц. Кроме того, здесь не существует межплатформенного стандарта: серверные страницы Java (Java Server Pages, JSP) соревнуются с активными серверными страницами Microsoft (Active Server Pages, ASP), в которых применяется язык Visual Basic.

Апплет – это производительный клиент, он существует в пределах виртуальной машины, работающей в броузере. В результате апплет является наиболее богатым из всех веб-клиентов в плане функциональности, но ограничен в отдельных важных моментах. Например, броузеры пока не поддерживают платформу Java 2, а значит, и удобные компоненты для создания графических интерфейсов (Swing). Кроме того, апплет потенциально требует многих соединений с удаленным сервером, поскольку операции ввода-вывода не могут выполняться на удаленной машине.

Большинству организаций требуются быстрые, эффективные и производительные клиенты, имеющие удобный пользовательский интерфейс, поэтому в качестве технологии для уровня клиентов часто выбирается динамический HTML, а не более требовательные к ресурсам варианты вроде апплетов. По мере роста поддержки Java-клиентов в броузерах и усложнения требований к представлению данных приложений электронной коммерции технология апплетов, вероятно, заменит динамический HTML во многих сетевых приложениях.

Технология уровня представления

Существует два основных вида технологии уровня представления: приложения CGI и сервлеты Java. Большинство пользователей Всемирной паутины сталкиваются с CGI, применяя поисковые машины для поиска телефонных номеров, адресов сайтов и карт расположения. При этом спецификация позволяет осуществлять обмен произвольными данными между клиентом и сервером. Этот интерфейс особенно хорошо сочетается с программами, написанными на языке C (когда для отправки запроса из HTML-документа используется метод POST), поскольку данные принимаются сервером в форме, удобной

для чтения в качестве стандартного потока ввода. Для извлечения информации, вводимой пользователями на HTML-страницах, применяются подпрограммы разбора потоков. Длину потока ввода в байтах в программе на C можно прочитать из переменной среды с помощью функции *getenv( )*:

```
getenv("CONTENT_LENGTH")
```

После разбора ввода, установки значений переменных и выполнения поставленной задачи программа может передать результаты через поток вывода в HTML-форму с помощью стандартной функции *printf( )*:

```
printf("Content-type: text/html\n\n");
```

После этого на терминал клиента передается размеченный текст и URL-адреса.

В качестве альтернативы разработке специальных приложений для CGI-чтения и записи на языке C можно воспользоваться одной из многочисленных бесплатных C-библиотек, предназначенных для преобразования данных из потока в готовые к использованию переменные. Например, пакет CGIHTML Евгения Кима (Eugene Kim) очень прост в применении, успешно компилируется и работает в среде Solaris. Пакет доступен по адресу *http://www.eekim.com/software/cgihtml/*. Страница, посвященная разнообразным ресурсам по CGI, расположена по адресу *http://www.cgi-resources.com/*. Мы упомянули C и CGI, но в зависимости от потребностей приложения может использоваться любой язык программирования, работающий в ОС Solaris и предоставляющий средства для обработки потоков (в частности, это сценарии командных интерпретаторов и язык Perl, разработанный Ларри Уоллом).

Если приложения CGI легко разрабатывать и они широко применяются, какова же может быть причина использования сервлетов Java на уровне представления? Как мы уже говорили, одним из наиболее очевидных преимуществ является возможность обслуживать большое число запросов с помощью многопоточного сервера приложений, существующего в пределах одного процесса. Кроме того, приложения CGI часто не позволяют сохранять состояние, и это значит, что переменные сеанса работы не могут передаваться с одной страницы на следующую. Это очень важно для задач идентификации и авторизации, имя пользователя и пароль могут требоваться для доступа к любой странице, но нельзя требовать от пользователя постоянно вводить их. Сервлеты Java привносят на уровень представления долгоживущие (persistence) объекты и, кроме того, способны взаимодействовать со службами уровня предприятия, такими как компоненты EJB.

Java часто применяется для создания симпатичных пользовательских интерфейсов (GUI), которые не зависят от платформы и позволяют взаимодействовать с пользователем способами более сложными, чем те, что доступны для статического HTML. Но эти «апплеты» – лишь

одна из сторон медали Java; в этом разделе мы взглянем еще и на другую сторону, *серверную*. Приложения Java, выполняемые на сервере, называются *сервлетами*. Сервлеты подчиняются собственному стандартизированному API, который к настоящему моменту получил широкое распространение в продуктах расширения веб-серверов, средах исполнения сервлетов (например, Allaire JRun). Давайте определим степень полезности сервлетов при разработке корпоративных приложений с веб-интерфейсами в среде Solaris.

Для реализации корпоративных приложений все чаще используются веб-интерфейсы. Отчасти это обстоятельство является реакцией на стойкую гетерогенность компьютерных платформ организаций масштаба города, государства или даже нескольких стран. Разнообразие платформ не должно быть связано с беспорядком в стандартах; в качестве доказательства компания Sun Microsystems первой разработала язык программирования, не зависящий от платформы. Приложения Java выполняются в рамках логической «виртуальной машины» (JVM), которая предоставляет разработчикам последовательный прикладной интерфейс. Виртуальная машина реализована для большинства аппаратных платформ и операционных систем, включая (разумеется) и Solaris. По сути дела, виртуальная машина Solaris JVM, созданная Sun, подверглась значительной оптимизации в серии версий, предназначенных для коммерческих применений. Кроме того, виртуальные машины Java интегрируются в популярные броузеры, поэтому Java-программы могут загружаться с сервера и выполняться в рамках этих броузеров (в HTML теперь существует специальный тег <APPLET>). Апплеты повысили сложность структуры пользовательских веб-интерфейсов и позволили от наборов кнопок и форм перейти к динамическому взаимодействию с пользователями, которое стало нормой в обычных приложениях.

И хотя Java успешно применяется в целях улучшения клиентской части веб-интерфейсов, распространение технологии на стороне сервера оказалось не столь быстрым (во многом это результат шумихи вокруг апплетов и дефицит интерфейсов прикладного программирования для сервлетов). Но многие люди считают, что именно на стороне сервера может полностью раскрыться великий потенциал языка Java. Концепция независимых от платформы корпоративных приложений со стандартными веб-интерфейсами может полностью изменить способы взаимодействия пользователей, разработчиков и программного обеспечения. Философия кода, созданного единожды и работающего повсюду, позволяет заменять существующее оборудование и операционные среды совершенно другими, более новыми системами, не беспокоясь при этом за стабильность и перенос приложений. Часто применяемые классы Java могут распространяться в виде bean-компонентов, обеспечивающих быструю реализацию логики приложения для клиентов. Сервлеты Java имеют полный доступ к Java API и серверам баз данных посредством классов JDBC (Java Database Connectivity

classes), поставляемых компанией Oracle и другими крупными фирмами. Такие возможности гарантируют, что сегодняшние серверные Java-программы завтра не станут компьютерными динозаврами, с трудом поддающимися интеграции.

Одним из крупных преимуществ технологий Java для Solaris является возможность разработки и развертывания приложений на одной машине корпоративной среды благодаря таким разработчикам, как компания Inprise, которая в настоящее время поставляет свою платформу Java-разработки (JBuilder) и платформу веб-развертывания (IAS Application Server) в версиях для Solaris в дополнение к вариантам для платформ Linux и Microsoft Windows. Таким образом, разработчики, желающие воспользоваться инструментами с графическим интерфейсом, могут наслаждаться стабильностью, к которой в контексте электронной коммерции уже успели привыкнуть администраторы.

Компилируемые языки, такие как C, в среднем дают более высокую скорость работы, чем интерпретируемые языки (Java); при использовании Java можно добиться повышения производительности, сочетая оптимизирующие JIT-компиляторы для конкретных платформ и сокращение избыточного потребления памяти и процессов (в сравнении с CGI). После загрузки в память Java-сервлет занимает единственный процесс, который является долгоживущим и существует между сеансами, что более эффективно в плане потребления памяти и процессов. Соответственно, сервлеты являются более подходящим решением для приложений, к которым постоянно обращается большое число пользователей, поскольку позволяют воспользоваться преимуществами многопоточности Java и возможностями синхронизации. С другой стороны, для однопользовательских приложений, приложений, которые запускаются не очень часто либо выполняют большие объемы вычислений (например, раз в час), более оптимальным решением может оказаться CGI. Кроме того, CGI-программы, написанные на языке C, логически изолированы одна от другой в пространстве памяти сервера; сервлеты Java выполняются в пределах единственного сеанса менеджера служб (например, Allaire JRun), и необработанное исключение, возникшее из-за некорректных входных данных, потенциально может повлиять на все сервлеты, исполняемые менеджером, в особенности если произойдет аварийный отказ виртуальной машины Java (JVM).

Технология обращения к объектам

Зачастую разработчик проектирует простые веб-приложения, имея в виду единственный сервер. Такая модель, как правило, поначалу функционирует замечательно. Но если веб-сайт получает признание, нагрузка на сервер может возрастать экспоненциально, замедляя время отклика или даже приводя к сбоям. Одним из решений является увеличение мощности конкретной системы, обновление до многопроцессорной конфигурации, повышение объемов памяти, добавление

карт сетевых интерфейсов и т. д. Но даже в случае системы E10000 вычислительные мощности могут неожиданно кончиться. В качестве одного из решений во многих приложениях электронной коммерции используется интеграция данных с серверов, физически расположенных в различных точках географического региона. Например, банковскому приложению, работающему в режиме реального времени, может потребоваться информация из отделения, в котором был открыт счет, а также информация из хранилища данных, которое физически наиболее близко расположено к точке, из которой производится доступ к счету. И наконец, если случится перебой питания сервера в Нью-Йорке, в работу включится идентичный сервер в Лос-Анджелесе, и действие службы возобновится после немногочисленных изменений в сетевых настройках.

Каждый из таких вариантов требует простого способа интеграции данных с различных серверов, расположенных в различных местах, в целях повышения емкости, функциональности и надежности. И хотя эта задача до какой-то степени может быть решена при помощи только операционной системы (например, с применением технологии репликации), разработка распределенного приложения может являться и основной задачей проекта. Так, если приложение должно обеспечивать сохранность сеансов пользователя на конкретном сервере, сеансы должны сохраняться и между запросами, поступающими от клиентов.

Одним из вариантов решения этих задач является распределенная архитектура приложений, такая как стандартизированная Common Object Request Broker Architecture (CORBA). В языке Java существуют классы удаленного обращения к методам (Remote Method Invocation, RMI), но в RMI отсутствует поддержка интеграции с существующими клиентами и системами, которые все еще активно используются сегодня. К счастью, архитектура CORBA спроектирована таким образом, что позволяет осуществлять интеграцию практически всех существующих систем. Плюс ко всему CORBA не зависит от конкретного языка, а значит, клиент, написанный на C++ (и даже на COBOL), может пользоваться общим интерфейсом для обращения к объектам сервера, написанного на Java. Архитектура CORBA в чем-то похожа на вызовы удаленных процедур (Remote Procedure Call, RPC).

Главной службой CORBA является посредник запросов к объектам (Object Request Broker, ORB), который отвечает за передачу упакованных объектов по сети в соответствии с запросами клиентов. Клиент и сервер могут сосуществовать в одной системе, хотя чаще всего самостоятельный ORB-посредник обеспечивает объектами всю гетерогенную сеть. В ORB входит, в частности, служба имен, которая позволяет удаленным клиентам идентифицировать и запрашивать объекты. ORB-посредники могут общаться между собой по протоколу InterORB (IIOP).

Интерфейсы CORBA могут легко создаваться с помощью языка определения интерфейсов (Interface Definition Language, IDL). Код IDL мо-

жет применяться для генерации кода для таких языков, как Java и C++, с целью последующей реализации определяемых интерфейсом методов. Подобные заглушки и каркасы облегчают управление межплатформенной разработкой и развертыванием.

Технология бизнес-объектов

В составе платформы Java 2 Enterprise Edition (корпоративное издание) от Sun появился новый игрок рынка серверных компонентов: компоненты Enterprise JavaBeans (или EJB). Технология EJB применяется для инкапсуляции логики приложений и бизнес-правил в самостоятельных компонентах, которые могут многократно использоваться в приложениях различного рода. Аналогично клиентским JavaBeans, компоненты EJB могут подстраиваться под нужды конкретного приложения. Каждая отдельная составляющая, реализующая определенный аспект логики приложения, может быть реализована в виде компонента EJB; именно так зачастую и поступают разработчики на Java в целях решения стандартных бизнес-задач.

Так, во многих приложениях требуется функциональность создания и хранения учетных записей клиентов в базе данных, а также выполнения определенных операций над этими записями, таких как добавление новой информации, обновление существующих записей и удаление устаревших. Для выделения общей логики для подобных приложений в самостоятельные сущности может быть создан набор компонентов EJB, предоставляющих приложениям интерфейс для работы с объектами конкретного абстрактного типа.

Для чего нужно такое разделение? Хорошей практикой, особенно в крупных проектах разработки, является четкое разделение элементов, специфичных для приложения, и элементов, которые являются общими для многих приложений. Это способствует повторному использованию программных модулей, особенно в пределах одной организации, и сокращает путь до прилавка для многих приложений. Кроме того, сейчас происходит зарождение рынка готовых к многократному применению программных компонентов, и выход на этот рынок потенциально способен принести дополнительную прибыль при работе над внутренними проектами. Это позволит несколько возместить затраты на обучение разработчиков использованию технологии EJB.

В качестве примера применения компонентов EJB в бизнес-контексте вообразим ситуацию, когда заказчик предлагает выполнить разработку системы заказов на базе веб-каталогов. HTML-клиент генерируется динамически Java-сервлетом, который позже обрабатывает данные, полученные в параметрах методов GET и POST, а также, исходя из этих параметров, организует работу с базой данных. Например, покупателю может понадобиться выполнить поиск определенного продукта: HTML-страница поиска генерируется сервлетом, пользователь указывает параметры поиска и нажимает на кнопку поиска.

Передача данных методом POST приводит к получению параметров сервлетом, который конструирует SQL-оператор SELECT и передает запрос базе данных посредством JDBC. Процессы СУБД обрабатывают запрос и возвращают ответ сервлету, который генерирует страницу результатов поиска (которая, будем надеяться, содержит ссылки на товары, интересующие клиента).

Но этому приложению не хватает разделения двух наиболее важных серверных функций: то есть компонента представления, который отвечал бы за генерацию HTML и/или сценариев JavaScript на стороне клиента, плюс компонента логики приложения, реализующего классы, необходимые для создания, чтения, записи и обновления пользовательских данных в БД. Это означает, что при каждом изменении внешнего вида или функциональности интерфейса может появляться необходимость в изменении классов серверной функциональности, если отсутствует должная инкапсуляция. Поскольку инкапсуляция является одной из целей объектно-ориентированного программирования, и тому есть много веских причин, мы можем отделить логику представления от логики приложения с помощью EJB. В таком случае подходящим выбором для реализации уровня представления будет сервлет, а для логики приложения – компоненты EJB. И если теперь внешний вид и функциональность интерфейса сайта подвергнутся значительным изменениям, функциональность базового приложения затронута не будет. Для проектов, в которых заняты многие разработчики, подобная политика облегчает разделение задач разработки на более мелкие, конкретные задачи.

Кроме того, распределенные приложения намного более сложные, чем работающие на одном сервере, и поэтому еще более важным становится логическое разбиение функциональности. Например, приложение может работать с многочисленными базами данных или серверами приложений, в которых используется распределение нагрузки DNS («round-robin») либо похожая технология, в целях эффективной обработки больших объемов веб-транзакций. В таком случае уровень представления может обслуживаться одной либо несколькими машинами. Во многих приложениях электронной коммерции (и это вдвойне верно для крупных команд разработчиков) сегодня применяются компоненты EJB в качестве механизма разделения логики приложения и уровня представления.

Технологии баз данных

Фундаментом любой системы электронной коммерции является сервер баз данных. Базы данных – это хранилища информации, к которым можно обращаться с помощью языка структурированных запросов (Structured Query Language, SQL), если речь идет о реляционных СУБД, таких как Oracle, либо с помощью языка объектных запросов (Object Query Language, OQL) в случае объектно-ориентированных

СУБД, таких как Versant. Реляционные базы данных идеально справляются с такими задачами, как объединение таблиц и создание отчетов в виде, подходящем для публикации в Сети. Напротив, объектно-ориентированные базы данных позволяют простейшим способом делать объекты приложений долгоживущими, не сталкиваясь при этом с ограничениями реляционных БД. Это в особенности важно при работе с большими объемами транзакций, не требующих обращения ко многим таблицам. Многие считают базы данных основанием всех приложений, ориентированных на обработку данных, а потому выбор подходящей для проекта СУБД является важным решением.

Для платформы Solaris доступны как коммерческие, так и бесплатные СУБД. Очень популярной реляционной СУБД, широко применяемой на платформе Solaris в контексте электронной коммерции и хранения данных, является Oracle. Существуют другие, низкобюджетные варианты, и даже замечательные бесплатные продукты, такие как PostgreSQL. PostgreSQL и Oracle поддерживают связь по JDBC (Java Database Connectivity) и, следовательно, могут использоваться в качестве основания любой информационной веб-системы. В большинстве случаев код Java-сервлета может переноситься с системы на систему и работать с Oracle или PostgreSQL без повторной компиляции. В Oracle реализована собственная поддержка JDBC в виде SQL*Net, для PostgreSQL требуется отдельная система серверного менеджера JDBC. В обоих случаях необходимы клиентские Java-библиотеки.

В настоящее время существует четыре различных вида драйверов JDBC:

Вид I: Мост JDBC-ODBC

Данный вид драйвера преобразует вызовы JDBC в эквивалентные вызовы ODBC для систем Microsoft Windows, на которых применяются ODBC-совместимые СУБД. Примером такого драйвера является драйвер, поставляемый в составе Sun JDK.

Вид II: Собственный интерфейс

В этом виде драйвера применяется интерфейс, определяемый разработчиком, для доступа к базе данных на конкретной машине. По производительности этот вид сравним с программами на языке C (или PL/SQL).

Вид III: Открытый интерфейс

Данный вид драйвера, как правило, поддерживает различные типы баз данных. Компоненты клиента и сервера JDBC являются абсолютно самостоятельными и могут располагаться в различных системах. Например, продукт JSVR от Caribou Lake JSVR поддерживает Ingres, Oracle, а также большинство реляционных СУБД.

Вид IV: Фирменный интерфейс

Драйверы этого вида разрабатываются производителем конкретной системы, но при его использовании компоненты сервера и клиента JDBC все так же могут располагаться в различных системах.

Очевидно, если в архитектуре системы присутствует одна машина, на которой работает как сервер приложений, так и сервер баз данных, адекватным выбором будет драйвер первого или второго вида. Однако с увеличением обрабатываемых объемов может потребоваться увеличение числа систем с вероятным физическим разделением сервера приложений и сервера баз данных. Здесь более подходящими будут драйверы третьего и четвертого видов.

Настройка веб-сервера

Вопреки традициям, в составе Solaris 8 теперь присутствует популярное бесплатное ПО, а также программы с открытым кодом. Одна из этих программ является отраслевым стандартом. Речь идет о веб-сервере Apache, наиболее популярном веб-сервере сети Интернет. Для общения с веб-броузерами (клиентами) Apache использует протокол передачи гипертекста (Hypertext Transfer Protocol, HTTP): клиенты передают запросы с помощью инструкций GET или POST, а на сервер возложена ответственность за передачу данных клиенту. Стандартным портом приема соединений является порт 80, но это значение можно изменить в файле настройки Apache (*httpd.conf*).

Настройка Apache состоит из трех главных разделов: глобальные инструкции настройки, настройки конкретного сервера, поддержка виртуальных серверов. Дополнительные настройки могут быть связаны с необходимостью использования системы SSL Apache с целью шифрования информации о кредитных картах, паролей и других чувствительных данных.

Глобальные инструкции настройки определяют значения важных параметров установки веб-службы Apache.

```
ServerType standalone
ServerRoot "/usr/local/apache-1.3.6"
LockFile /usr/local/apache-1.3.6/httpd.lock
PidFile /usr/local/apache-1.3.6/httpd.pid
ScoreBoardFile /usr/local/apache-1.3.6/httpd.scoreboard
ExtendedStatus On
Timeout 600
KeepAlive On
MaxKeepAliveRequests 1000
KeepAliveTimeout 30
StartServers 15
MinSpareServers 15
MaxSpareServers 45
MaxClients 300
MaxRequestsPerChild 30
```

Приведенный фрагмент файла настройки определяет следующие параметры работы сервера:

- Сервер Apache должен работать в качестве самостоятельного демона, а не под управлением демона интернет-служб. Это сокращает общую загрузку системы, поскольку запросы клиентов обрабатываются процессами демона, тогда как в случае использования *inetd* порождается новый процесс для каждого запроса, полученного сервером.
- Корневым каталогом установки Apache является */usr/local/apache-1.3.6*. Файл блокировки, файл идентификатора процесса и файл счета (в котором хранится информация о состоянии) хранятся в этом каталоге.
- Интервал ожидания для сервера установлен в 600 секунд. Он определяет лимит времени, по истечении которого запрос или ответ может считаться устаревшим. В быстрых сетях это значение можно уменьшить, а в медленных его следует увеличивать. Запросы поддерживаются для максимум одной тысячи клиентов, что обеспечивает постоянство соединений для большого числа запросов.
- После запуска сервер порождает пятнадцать процессов. Это значение должно основываться на прогнозируемом минимальном количестве единовременных запросов к серверу. В данном варианте по меньшей мере 15 свободных процессов будут существовать постоянно, максимум таких процессов может быть 45.
- Максимальное число единовременно обслуживаемых клиентов – 300. По достижении предельного значения запросы от новых клиентов не будут обслуживаться. Ограничение является абсолютным и должно устанавливаться с учетом предполагаемого числа клиентов, работающих с сервером в моменты пиковой загрузки. Максимальное число запросов, обслуживаемых каждым порожденным процессом, равно 30. Это предотвращает снижение производительности из-за возможных ошибок и утечки памяти на средних сроках работы.

Значения параметров могут изменяться в зависимости от нужд производственной среды, а уровень обслуживания, достижимый на системе веб-сервера и удовлетворяющий клиентов, может определяться посредством тестирования загрузкой.

Настройки конкретного сервера определяют параметры работы веб-службы Apache:

```
Port 80
User apache
Group apache
ServerName www.paulwatters.com
UseCanonicalName On
ServerAdmin paul@paulwatters.com
```

Веб-служба будет работать на порту с номером 80, а процессы сервера выполняться с полномочиями пользователя *apache* из группы apache. Только привилегированным пользователям разрешено выполнять

службы, работающие на портах с номерами, меньшими 1024, и значит, Apache должен запускаться администратором системы. Здесь также указано полное доменное имя сервера и записан адрес электронной почты системного администратора, который будет отображаться на страницах с сообщениями об ошибках. Применение канонического имени предписывает заменять URL на значение *ServerName*, не пользуясь именем узла, указанным в исходном URL-адресе.

Следующий шаг связан с определением структуры каталогов информационного наполнения веб-сайта:

```
# Инструкции для базового каталога веб-сайта
DocumentRoot "/usr/local/apache-1.3.6/htdocs"
<Directory />
    Options Indexes FollowSymLinks
    AllowOverride None
    Order allow,deny
    allow from all
</Directory>
# Определяем имя файла индекса по умолчанию и имя файла управления доступом
DirectoryIndex index.htm
AccessFileName .htaccess
<Files .htaccess>
    Order allow,deny
    Deny from all
</Files>
TypesConfig /usr/local/apache-1.3.6/conf/mime.types
DefaultType text/plain
# Инструкции, позволяющие задействовать пользовательские веб-каталоги
UserDir public_html
<Directory /*/public_html>
    Options Indexes
    AllowOverride None
    <Limit PUT DELETE>
        Order deny,allow
        deny from all
    </Limit>
</Directory>
# Инструкции, позволяющие исполнять программы из cgi-bin
ScriptAlias /cgi-bin/ "/usr/local/apache-1.3.6/cgi-bin/"
<Directory "/usr/local/apache-1.3.6/cgi-bin">
    AllowOverride None
    Options None
</Directory>
```

По умолчанию все документы HTML хранятся в подкаталоге *htdocs* корневого каталога Apache. Существует возможность ограничить доступ к каталогам, расположенным в каталоге *htdocs*, и в этом случае будет применяться механизм идентификации пользователей, основанный на именах пользователей и паролях, указанных в файле *.htpasswd*. Данный механизм является совершенно самостоятельным и ни-

как не связан с идентификацией пользователей системы Solaris. По умолчанию пользовательские веб-каталоги задействованы для всех пользователей Solaris, создавших подкаталоги *public_html* в своих домашних каталогах. Эти каталоги доступны по адресам вида:

http://hostname/~user/index.html

Для этих каталогов явным образом запрещено выполнение операций PUT и DELETE. И наконец, сценарии CGI хранятся в подкаталоге cgi-bin корневого каталога Apache.

При необходимости можно задать связи для некоторых известных файловых типов MIME, которые используются броузерами для отображения страниц индексов. Кроме того, можно настраивать ведение журнала:

```
# Инструкции автоматического создания индексов
IndexOptions FancyIndexing
AddIcon /icons/binary.gif .bin .exe
AddIcon /icons/binhex.gif .hqx
AddIcon /icons/tar.gif .tar
AddIcon /icons/compressed.gif .Z .z .tgz .gz .zip
AddIcon /icons/layout.gif .html .shtml .htm pdf
AddIcon /icons/text.gif .txt
AddIcon /icons/uuencoded.gif .uu
AddIcon /icons/back.gif ..
AddIcon /icons/hand.right.gif README
AddIcon /icons/folder.gif ^^DIRECTORY^^
AddIcon /icons/blank.gif ^^BLANKICON^^
DefaultIcon /icons/unknown.gif
AddDescription "GZIP compressed document" .gz
AddDescription "tar archive" .tar
AddDescription "GZIP compressed tar archive" .tgz
IndexIgnore .??* *~ *# HEADER* README* RCS CVS *,v *,t
# Инструкции, связанные с ведением журналов
HostnameLookups Off
ErrorLog /usr/local/apache-1.3.6/logs/error_log
LogLevel warn
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combine
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
CustomLog /usr/local/apache-1.3.6/logs/access_log common
CustomLog /usr/local/apache-1.3.6/logs/referer_log referer
CustomLog /usr/local/apache-1.3.6/logs/agent_log agent
CustomLog /usr/local/apache-1.3.6/logs/access_log combined
```

Последний шаг в настройке работы сервера связан с указанием виртуальных серверов, которые могут размещаться на сервере:

```
<VirtualHost www.ethos-development.com>
    ServerAdmin paul@paulwatters.com
```

```
    DocumentRoot /usr/local/apache-1.3.6/virtual_hosts/ethos-development.com
    ServerName www.ethos-development.com
    ErrorLog /usr/local/apache-1.3.6/virtual_hosts/logs/ethos-development-error_log
    TransferLog /usr/local/apache-1.3.6/virtual_hosts/logs/ethos-development-access_
        log
</VirtualHost>
<VirtualHost www.java-support.com>
    ServerAdmin paul@paulwatters.com
    DocumentRoot /usr/local/apache-1.3.6/virtual_hosts/java-support.com
    ServerName www.java-support.com
    ErrorLog /usr/local/apache-1.3.6/virtual_hosts/logs/java-support-error_log
    TransferLog /usr/local/apache-1.3.6/virtual_hosts/logs/java-support-access_log
</VirtualHost>
```

В данном случае мы определили два виртуальных узла: *www.java-support.com* и *www.ethos-development.com*. Каждая из служб связана с собственной структурой каталогов, которые расположены в подкаталоге *virtual_hosts* корневого каталога Apache. Для каждого из серверов могут вестись отдельные журналы обращений и ошибок. Виртуальные узлы обычно создаются на площадках провайдеров интернет-услуг и в крупномасштабных производственных средах.

Управление сервером Apache организовано с помощью сценария *apachectl*, который расположен в подкаталоге *bin* корневого каталога Apache. Сценарий может применяться для:

- Запуска сервера (*apachectl start*)
- Останова сервера (*apachectl stop*)
- Отображения состояния сервера (*apachectl status*)

После запуска сервера клиенты могут обращаться к нему через порт 80 и запрашивать HTML-документы, расположенные в соответствующем каталоге *htdocs*. Кроме того, сценарии CGI могут копироваться в подкаталог *cgi-bin* корневого каталога Apache, а затем выполняться запросами адресов вида:

http://hostname/cgi-bin/application

Сценарий CGI может быть программой на языке C, Perl либо сценарием интерпретатора, как в следующем простом примере. Здесь отображается объем дискового пространства, доступного на системе веб-сервера:

```
#!/bin/sh
echo "Content-type: text/html\n\n"
echo "<HTML>"
echo "<HEAD><BGCOLOR=#FFFFFF></HEAD>"
echo "<h1>Свободное дисковое пространство</h1>"
df -k
echo "</BODY>"
echo "</HTML>"
```

Результатом выполнения будет поток HTML (определяемый MIME-типом text/html), который можно просмотреть в броузере в качестве обычной веб-страницы.

Одной из действительно удобных черт Apache является возможность наращивать функциональность с помощью системы модулей. Разработчики вне проекта Apache имеют возможность создавать дополнительные модули с помощью стандартного интерфейса прикладного программирования. В состав Apache изначально входит ряд модулей:

mod_auth

Поддержка идентификации пользователей с шифрованием паролей стандартной функцией *crypt( )*.

mod_cgi

Поддержка выполнения сценариев CGI.

mod_digest

Поддержка идентификации пользователей с шифрованием MD5.

mod_include

Расширение функциональности HTML поддержкой серверных включений (SSI).

mod_log_agent

Настройка файлов журналов.

mod_mime

Поддержка ассоциирования приложений с MIME-типами файлов (Multipurpose Internet Mail Extension, многоцелевые расширения почтовой службы в Интернете).

mod_proxy

Позволяет использовать Apache в качестве proxy/кэширующей системы, хранящей часто запрашиваемые файлы.

mod_status

Отображает состояние работающего сервера Apache.

Модули могут компилироваться при сборке сервера Apache либо загружаться динамически с помощью пары записей (*LoadModule* и *AddModule*) в файле *httpd.conf* для каждого модуля. Так, чтобы активизировать модуль *mod_auth*, добавим следующие строки в файл *httpd.conf*:

```
LoadModule auth_module    modules/mod_auth.so
AddModule mod_auth.c
```

Исполнение сервлетов

Недавно в Apache появилась возможность исполнения сервлетов Java с помощью модуля JServ (*mod_jserv*). JServ – это среда исполнения сервлетов, сравнимая по функциональности с той, что поставляется Sun в составе инструментария разработки Java-сервлетов (Java Servlet Development Kit, JSDK). Среда исполнения сервлетов из JSDK, разумеется, поддерживает наиболее популярные методы работы служб (в частности, GET и POST), но не предназначена для применения в реальных приложениях. Чаще всего она применяется разработчиками, и доступна для загрузки по адресу *http://java.sun.com/products/servlet/*. Напротив, JServ является модулем Apache, полностью интегрированным с функциями веб-сервера. Более подробная информация по Apache JServ доступна на сайте *http://java.apache.org/*. В настоящее время существует альтернативный JServ вариант, разработанный в рамках проекта Apache Java, его кодовое название – Jakarta. Со временем Jakarta заменит JServ, поскольку предлагает пользователям больше возможностей, включая настройку в формате XML, но пока что в большинстве реальных приложений используется JServ.

В большинстве систем используется одна из многочисленных коммерческих сред исполнения сервлетов. Как правило, такая среда является частью целого набора служебных приложений. В нижней части ценового спектра находятся такие продукты, как JRun от Allaire, в котором для поддержки сервлетов используется модуль Apache, в дополнение к инструментам уровням представления и отображения, таким как Cold Fusion. Более подробная информация о JRun доступна по адресу *http://www.allaire.com*. Верхняя часть ценового спектра занята такими продуктами, как сервер приложений Inprise (IAS), который предоставляет все необходимые для построения полной системы электронной коммерции компоненты, включая веб-сервер, среду исполнения сервлетов, контейнер EJB и сервер CORBA. Кроме того, в состав IAS интегрировано средство администрирования с веб-интерфейсом. Более подробная информация об IAS доступна по адресу *http://www.inprise.com/*. Оба этих пакета поддерживают также серверные страницы Java (JSP), то есть HTML-страницы со встроенным кодом на Java, который компилируется во время выполнения и позволяет с легкостью интегрировать логику приложения с визуализацией данных.

Настройка JServ

Apache JServ с легкостью устанавливается в качестве модуля компилируемого при сборке сервера Apache либо в качестве внешнего DSO-модуля. Поскольку в случае работающей системы потери времени на постоянную перекомпиляцию Apache неприемлемы, рекомендуется использовать модули DSO.

Чтобы настроить JServ на поддержку DSO, необходимо изначально собрать систему с приблизительно такими настройками:

```
bash-2.03$ ./configure \
     --prefix=/usr/local/apache-1.3.6 \
     --enable-rule=SHARED_CORE \
     --enable-module=so
```

Непосредственно сборка сервера осуществляется командой:

```
bash-2.03$ make install
```

Включив поддержку DSO в Apache, можно переходить к настройке Apache JServ. Для нашего случая команда настройки выглядит так:

```
bash-2.03$ ./configure --with-apxs=/usr/local/apache-1.3.6/bin/apxs
--prefix=/usr/local/apache-1.3.6/jserv
--with-jdk-home=/usr/local/java/jdk-1.2.2
--with-JSDK=/usr/local/java/jsdk-2.0/jsdk.jar
```

После задания настроек разделяемый объект может быть создан командой:

```
bash-2.03$ make install
```

Кроме того, следует добавить следующую строку в файл настройки Apache, *httpd.conf*:

```
Include "/usr/local/apache-1.3.6/jserv/jserv.conf"
```

После перезапуска сервера Apache к сервлетам можно обращаться по адресам вида:

http://localhost/servlet/servletname

Так, чтобы выполнить сервлет Simple Servlet, следует воспользоваться адресом:

http://localhost/servlet/SimpleServlet

Заметим, что для выполнения сервлетов не требуется отдельный процесс. Сервер Apache берет на себя начальную обработку всех запросов, а затем для выполнения кода на сервере используются разделяемый объект.

Существует два важных файла настройки Apache JServ: *jserv.conf*, в котором указываются параметры работы службы JServ, и *jserv.properties*, который является файлом свойств в стиле Java и может хранить свойства сервлетов. Свойства могут хранить такие параметры, как URL-адреса JDBC, имена и пароли пользователей баз данных и даже регулярно изменяемые текстовые строки. Хранение подобных строк в файле свойств, а не в исходном коде, сокращает число перекомпиляций, необходимых для внесения изменений в существующие приложения.

Вот основные параметры файла *jserv.conf*:

- *ApJServProperties* содержит полное имя файла свойств JServ (*/usr/local/apache-1.3.6/jserv/conf/jserv.properties*).
- *ApJServLogFile* содержит полное имя файла журнала JServ (*/usr/local/apache-1.3.6/jserv/logs/mod_jserv.log*).
- *ApJServDefaultHost* содержит полное доменное имя веб-сервера.
- *ApJServDefaultPort* содержит номер порта, через который Apache JServ будет принимать запросы (по умолчанию 8007).

Возможно также определить зоны сервлетов в файле *jserv.conf*, то есть логически независимые наборы сервлетов, доступных с одного сервера. Например, в среде сервера приложений клиенты могут частично контролировать свои зоны, не затрагивая при этом кода чужих приложений. Однако рекомендуется также использовать иерархическое именование классов и применяемых пакетов.

Файл *jserv.properties* может содержать как системные, так и пользовательские параметры. Вот некоторые из важных системных параметров:

- *wrapper.bin* содержит путь к исполняемому файлу интерпретатора (*/usr/local/java/jdk-1.2.2/bin*).
- *wrapper.bin.parameters* может содержать перечень стандартных параметров, передаваемых виртуальной машине Java. Например, с помощью ключей *-ms* и *-mx* можно указать, соответственно, минимальный и максимальный объемы используемой памяти.
- *wrapper.classpath* содержит путь ко всем классам, необходимым для работы вызываемых сервлетов.
- *bindaddress* указывает узлы, на которых работает Apache. В целях безопасности следует ограничивать список только локальным узлом, на котором запущен сервер. Но в пределах одной сети, защищенной брандмауэром, вполне допустима привязка к серверам, работающим на других системах.
- *security.authentication* позволяет производить проверку подлинности для входящих соединений.

К значениям этих свойств можно с легкостью обращаться из приложений на языке Java. Чтобы получить значение параметра *wrapper.bin* в приложении, можно воспользоваться следующим кодом:

```
String wrapperBin = getInitParameter("wrapper.bin");
```

К сожалению, не существует простого способа *изменять* параметры из приложений на Java. Тем не менее файл *jserv.properties* можно редактировать напрямую из командной строки. Когда изменения в файле восприняты Apache JServ, любой из активных сервлетов будет повторно инициализирован при следующем обращении.

Создание сервлетов

Итак, изучив процесс установки и настройки веб-сервера и среды исполнения сервлетов Java, мы переходим к вопросам разработки сервлетов и создания простых, но эффективных веб-приложений при помощи методов объектно-ориентированного проектирования и реализации.

Сервлеты обычно используются для создания динамических вариантов страниц HTML. Это означает, что содержимое HTML-страницы, генерируемой сервлетом, может изменяться в зависимости от решений, принятых в коде сервлета. Помимо этого, для модификации содержимого генерируемых страниц могут использоваться параметры, передаваемые сервлету методами GET и POST. Такая гибкость существенно облегчает проектирование динамических приложений, в особенности предназначенных для применения в корпоративных сетях.

Рассмотрим очень простое интранет-приложение, которое предоставляет на выбор три действия:

- Чтение доски объявлений (*http://localhost/servlet/Intranet?menu_option=bulletin_board*).
- Создание сообщения электронной почты (*http://localhost/servlet/Intranet?menu_option=email_client*).
- Отображение запланированных встреч (*http://localhost/servlet/Intranet?menu_ option=daily_calendar*).

Подобные приложения часто применяются в корпоративных сетях существующих организаций, и многие из этих приложений написаны на Java. Их базовая структура выглядит примерно так:

```
//Intranet.java
package com.paulwatters.applications;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class Intranet extends HttpServlet
{
  Connection databaseCon; // Дескриптор соединения с БД

  public void init(ServletConfig config) throws ServletException
  {
    super.init(config);
    String dbUsername = getInitParameter("db_username");
    String dbPassword = getInitParameter("db_password");
    String dbUrl = getInitParameter("db_url");
    String jdbcDriver = getInitParameter("jdbcDriver");
    try
    {
      Class.forName(jdbcDriver);
```

```
      databaseCon = DriverManager.getConnection(dbUrl, dbUsername, dbPassword);
      databaseCon.setAutoCommit(false);
    }
    catch (Exception e) // (ClassNotFoundException and SQLException)
    {
      e.printStackTrace();
    throw(new UnavailableException(this, "E001 - База данных недоступна"));
    }
  }

protected void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException
  {
    res.setContentType("text/html");
    HttpSession session = req.getSession(true);
    PrintWriter out = res.getWriter();
    String menuOption=req.getParameter("menu_option");
    if (menuOption.equalsIgnoreCase("bulletin_board"))
    {
      BulletinBoard board = new BulletinBoard();
      // Реализация приложения доски объявлений
    }
    else if (menuOption.equalsIgnoreCase("email_client"))
    {
      EmailClient email = new EmailClient();
      // Реализация приложения клиента электронной почты
    }
    else if (menuOption.equalsIgnoreCase("daily_calendar"))
    {
      DailyCalendar calendar = new DailyCalendar();
      // Реализация приложения ежедневника
    }
    else // Вывод главной страницы корпоративного приложения
    {
      out.print("<HTML>");
      out.print("<HEAD>");
      out.print("<TITLE>Приложение корпоративной сети");
      out.print("</HEAD>");
      out.print("<BODY>");
      out.print("<H1>Приложение корпоративной сети</H1>");
      out.print("<P>Пожалуйста, выберите один из пунктов:");
      out.print("<P><A HREF=/servlet/Intranet?command=bulletin_board>Доска
                объявлений");
      out.print("<P><A HREF=/servlet/Intranet?command=email_client>Почтовый
                клиент");
      out.print("<P><A HREF=/servlet/
                Intranet?command=daily_calendar>Ежедневник");
      out.print("</BODY>");
      out.print("</HTML>");
    }
  }
```

```
public void destroy()
{
  try
  {
    if (databaseCon != null)
    {
      databaseCon.close();
    }
  }
  catch (SQLException ignored)
  {
  }
}
```

В первой строке исходного текста содержится декларация пакета. Поскольку пространства имен Java могут быстро наполняться методами и классами, имена которых звучат похоже, а назначение почти совпадает, хорошей практикой считается иерархическое построение методов и классов. Разработчики часто идентифицируют свои классы именами, основанными на доменных (например, *paulwatters.com*).

Следующий раздел кода содержит операторы импорта, которые позволяют обращаться к методам Java API из класса *Intranet* по сокращенным именам. Нам требуется импортировать классы, определяющие класс *HttpServlet*, являющийся базовым для всех сервлетов. Расширение класса *HttpServlet* классом *Intranet* формализовано декларацией класса, за которой следуют методы, объекты и переменные, входящие в класс.

В классе *Intranet* существует единственный объект *databaseCon*, который используется в качестве глобального дескриптора для создания соединений JDBC. Кроме того, в классе *Intranet* объявлены три метода. Метод *init* определен в API сервлетов и применяется для получения значений параметров, определенных в файле свойств сервера, таких как имена и пароли пользователей базы данных, URL-адреса баз данных и имена драйверов JDBC. В нашем примере сделана попытка установить соединение с базой данных в пределах блока *try/catch*, который обеспечивает механизм отказоустойчивой обработки ошибок времени выполнения, связанных с обращением к БД. Если возникает ошибка, отображается распечатка стека, а код ошибки записывается в журнал.

Метод *doGet* реализует требования, предъявляемые к обработке запросов и ответов HTTP GET. Прежде всего выполняются следующие действия:

- MIME-тип содержимого устанавливается в *text/html*.
- Создается сеанс HTTP, который может использоваться для хранения информации идентификации, такой как имена и пароли пользователей.

- С целью записи HTML-вывода создается экземпляр *printwriter*.
- Для хранения значения параметра *menu_option*, передаваемого броузером в строке URL, создается строковая переменная.

Метод *doGet* способен интерпретировать три различных значения параметра *menu_option*: *bulletin_board*, *email_client* и *daily_calendar*. В каждом случае создается экземпляр объекта соответствующего типа – доска объявлений (Bulletin Board), почтовый клиент (Email Client) или ежедневник (Daily Calendar). Если получено иное значение *menu_option*, отображается главная страница приложения. Это означает, что ввод следующих адресов приведет к получению главной страницы:

http://localhost/servlet/Intranet
http://localhost/servlet/Intranet?menu_option=not_supported

После того как страница отображена, пользователю остается выбрать желаемую ссылку в броузере, чтобы получить доступ к одному из корпоративных приложений.

Метод *destroy( )* всегда вызывается последним при уничтожении сервлета: в нем обычно выполняются такие функции, как уничтожение ссылок на соединения с базой данных, а также другие необходимые действия.

Такой подход позволяет постепенно наращивать функциональность главной страницы по мере появления соответствующих классов. Но в корпоративных сетях, как правило, требуются механизмы идентификации и проверки подлинности. В безопасной среде этим целям может служить самостоятельный сервер Kerberos. Как вариант может быть разработана специальная система проверки подлинности с применением распределенной объектной архитектуры, такой как CORBA, о которой мы сейчас и поговорим.

CORBA

CORBA – это распределенная объектная архитектура, облегчающая объединение служб, работающих на различных серверах. CORBA имеет ряд преимуществ по сравнению с традиционными технологиями распределенной обработки, такими как протокол вызова удаленных процедур от Sun (RPC):

- Посредники запросов к объектам CORBA (Object Request Brokers, ORBs) доступны в исполнении многочисленных разработчиков; число участников консорциума по технологии манипулирования объектами (Object Management Group, OMG), контролирующего развитие спецификации CORBA, достигает почти 1 000 членов.
- CORBA-ORB-посредники не зависят от платформы в том, что касается обмена данными по протоколу Internet Inter-ORB Protocol

(IIOP). Благодаря этому клиентские приложения, написанные на языке C++, могут общаться с сервером, написанным на Java.

- В CORBA применяется объектно-ориентированный язык определения интерфейсов (Interface Definition Language, IDL), с помощью которого описывается обмен данными между клиентами и серверами ORB и реализация логики приложений.
- Клиент и ORB-посредник могут сосуществовать на одном сервере.
- CORBA реализует откат транзакций и устойчивость к ошибкам в распределенной архитектуре.

CORBA применяется в распределенных транзакционных системах с потенциально высоким количеством веб-серверов, серверов баз данных и серверов объектов, для которых необходимо организовать совместный доступ к данным. Так, во многих корпоративных сетях требуется реализация распределенной проверки подлинности и разграничение доступа на основе предоставленной пользователем информации (например, имени и пароля). И хотя эта роль может быть возложена на существующие продукты (такие как Kerberos), специализированные решения на базе CORBA представляют собой привлекательную, нетребовательную к ресурсам альтернативу. В примере, представленном в этом разделе, мы создадим сервер, принимающий соединения от клиента, который предъявляет имя и пароль. Сервер создает JDBC-соединение с опорной базой данных и с помощью простого оператора SQL проверяет соответствие имени паролю по главной таблице. Наше приложение будет называться PassCheck.

Для развертывания сервера мы воспользуемся посредником VisiBroker ORB (разработанным компанией Inprise). VisiBroker обычно применяется как на стороне клиента, так и на стороне сервера, в таких веб-приложениях, как Netscape Navigator и веб-сервер Netscape. В настоящее время на рынке существует большое число реализаций ORB (среди которых есть и бесплатные продукты очень высокого качества), но VisiBroker обладает преимуществом прямой интеграции со службами EJB, о которых мы поговорим в следующем разделе. Кроме того, VisiBroker доступен для языков C++ и Java, что делает его привлекательным для организаций, использующих в своих разработках оба языка. Более подробная информация о VisiBroker доступна по адресу *http://www.inprise.com/*.

Создание приложения CORBA следует стандартной схеме, оно начинается с определения клиент-серверного интерфейса, а заканчивается выполнением сервера и клиента. Мы поочередно рассмотрим все шаги этого процесса.

Создание интерфейса

Интерфейс между клиентом и сервером создается с помощью языка определения интерфейсов (Interface Definition Language, IDL). Для

нашего приложения (PassCheck) мы можем создать IDL-файл (*passcheck.idl*) следующего содержания:

```
module paulwatters
{
        interface PassCheck
        {
                string verifyUser
                (
                        in string username,
                        in string password
                );

        };
};
```

В данном случае мы объявили модуль с именем *paulwatters*, с которым связано одно определение интерфейса (*passcheck*) и один метод (*verifyUser*), который передает два «входных» (*in*) строковых параметра: имя и пароль пользователя. Метод *verifyUser* возвращает строку, которая может принимать значение *USER_OK* либо *BAD_USER*. Параметры могут передаваться не только от клиента серверу (*in*), но и от сервера клиенту (*out*), а также в обе стороны (*inout*). Это означает, что сервер CORBA способен передавать параметры как по ссылке, так и по значению, что делает его более гибким, чем стандартный код на Java.

Когда спецификации IDL готовы, каркасы для сервера и клиентские заглушки могут быть сгенерированы для C++ или Java. Применение IDL исключает проблемы взаимодействия интерфейсов Java и C++, поскольку базовый код создается на основе спецификаций, не зависящих от конкретного языка.

Чтобы создать каркас сервера и клиентские заглушки на языке Java, выполните команду:

```
bash-2.03$ idl2java passcheck.idl
```

На основе IDL-спецификации в подкаталоге *paulwatters* будет создан ряд файлов, включая перечисленные ниже:

```
_ PassCheckStub.java          PassCheckOperations.java
PassCheck.java                PassCheckPOA.java
PassCheckHelper.java          PassCheckPOATie.java
PassCheckHolder.java
```

Имя каждого файла предваряется именем интерфейса и определяет предназначение кода. Файл *PassCheckPOA* определяет для сервера характеристики переносимого адаптера объектов (Portable Object Adapter, POA). Как правило, разработчику не требуется редактировать файлы, созданные на основе IDL.

Реализация методов

Итак, имея в руках заглушки и каркас, мы должны реализовать класс с единственным определенным в IDL-файле методом:

```
bash-2.03$ cat AuthenticateImpl.java
public class AuthenticateImpl extends security.AuthenticatePOA
{
  public String verifyUser(String username, String password)
  {
    if (authenticatePassword(username, password).equalsIgnoreCase("PASSWD_OK"))
    {
      return "USER_OK";
    }
    else
    {
      return "BAD_USER";
    }
  }

  public static String authenticatePassword(Connection dbCon, String
      username, String
      password)
  {
    String queryString="select password from users where username="+username;
    String result=null;
    try
    {
      Statement statement = dbCon.createStatement();
      ResultSet rs = statement.executeQuery(queryString);
      ResultSetMetaData rsMeta = rs.getMetaData();
      if (!rs.next())
      {
        result="PASSWORD_NOT_FOUND";
      }
      else
      {
        result="PASSWORD_OK";
      }
      rs.close();
      statement.close();
      dbCon.commit();
    }
    catch (SQLException ex)
    {
        ex.printStackTrace();
    }
    return result;
  }
}
```

Это определение должно храниться в файле реализации (то есть в файле *PassCheckImpl.java*). Метод *verifyUser* вызывает другой метод, *authenticatePassword*, который непосредственно соединяется с базой данных по JDBC. Этот метод может быть реализован в виде компонента EJB, как мы увидим в следующем разделе, а не просто встроен в реализацию методов CORBA в целях достижения максимальной гибкости. Таблица базы данных *users* будет включать колонки *username* и *password*. Если соответствующий пароль найден в таблице для переданного клиентом имени, вызывающему методу возвращается сообщение *PASSWORD_OK*, а тот, в свою очередь, передает код завершения операции клиенту.

Реализация клиента и сервера

Определив методы, необходимые для реализации интерфейса, мы должны создать сервер, предоставляющий эти вызовы клиентам, а также создать клиента, выполняющего подключение к серверу. Для инициализации POA (*paulwatters_agent_poa*) можно воспользоваться стандартным сервером, чтобы создать экземпляры серверных объектов. Эта ссылка используется и клиентом для обнаружения нужного сервера:

```
import org.omg.PortableServer.*;

public class Server
{
  public static void main(String[] args)
  {
    try
    {
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
      POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      org.omg.CORBA.Policy[] policies = { rootPOA.create_lifespan_
          policy(LifespanPolicyValue.PERSISTENT)
      };
      POA myPOA = rootPOA.create_POA( "paulwatters_agent_poa", rootPOA.the_
          POAManager(), policies );
      PassCheckImpl paulwattersServant = new PassCheckImpl();
      byte[] paulwattersId = "PassCheck".getBytes();
      myPOA.activate_object_with_id(paulwattersId, paulwattersServant);
      rootPOA.the_POAManager().activate();
      orb.run();
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }
}
```

После установки ряда правил, в частности предписывающих создание долгоживущих объектов, инициализируется рабочая часть ORB. Анало-

гичным образом клиент (*Client.java*) должен инициализировать ORB-посредник, подключиться к серверному POA (*paulwatters_agent_poa*), а затем передать необходимые параметры методу *verifyUser*, как показано в следующем примере:

```
public class Client
{
  public static void main(String[] args)
  {
    try
    {
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
      byte[] paulwattersId = "PassCheck".getBytes();
      paulwatters.PassCheck p = paulwatters.PassCheckHelper.bind(orb, "/
        paulwatters_
          agent_poa", paulwattersId);
       if (p.checkUsername("pwatters","sydney").equalsIgnoreCase("PASSWD_OK"))
       {
         System.out.println("Ваш пароль принят.");
       }
       else
       {
         System.out.println("Пароль отвергнут.");
       }
    }
    catch (Exception ex)
    {
      System.err.println(ex);
    }
  }
}
```

Сборка приложения

Сборку приложения можно произвести с помощью компилятора VisiBroker для Java (*vbjc*). Команда выглядит так:

```
bash-2.03$ vbjc *.java paulwatters/*.java
```

На данном этапе должны присутствовать следующие исходные файлы и файлы классов:

```
bash-2.03$ ls
passcheck.idl*          Client.class            Server.class
Client.java             Server.java             PassCheckImpl.class
PassCheckImpl.java      paulwatters/

bash-2.03$ ls paulwatters
_PassCheckStub.class        PassCheckHolder.java
_PassCheckStub.java         PassCheckOperations.class
PassCheck.class             PassCheckOperations.java
PassCheck.java              PassCheckPOA.class
```

```
PassCheckHelper.class       PassCheckPOA.java
PassCheckHelper.java        PassCheckPOATie.class
PassCheckHolder.class       PassCheckPOATie.java
```

Прогон кода

После сборки приложения и отладки возможных ошибок сервер можно запустить командой *vbj*:

```
bash-2.03$ vbj Server
Stub[repository_id=IDL:paulwatters/PassCheck:1.0,key=ServiceId[service=/
paulwatters_agent_poa,id={9 bytes: [P][a][s][s][C][h][e][c][k] }]] is ready.
```

Вывод сообщения означает, что все серверные методы, определенные IDL-спецификациями, стали доступны клиентам. Так что осталось лишь запустить клиентское приложение командой *vbj*:

```
bash-2.03$ vbj Client
Ваш пароль принят.
```

Такое сообщение свидетельствует о том, что в таблице пользователей существует запись для имени *pwatters* и пароля *sydney*. Однако если соответствие не было найдено (отсутствует имя пользователя *pwatters* либо был указан пароль *melbourne*), мы увидим следующее сообщение:

```
bash-2.03$ vbj Client
Пароль отвергнут.
```

В этом примере мы рассмотрели простейшее приложение, однако в нем задействованы элементы, обычно присутствующие в большинстве распределенных приложений; в частности проверка подлинности пользователя, удаленный вызов методов, доступ к базе данных. И хотя число методов может расти либо для конкретной службы могут определяться различные интерфейсы, схема разработки, связанная с написанием интерфейсов IDL, реализацией объявленных методов, размещением клиента и сервера, остается неизменной.

Enterprise JavaBeans

Хотя технология CORBA (и RMI) может использоваться для связывания клиентов и служб, работающих на различных системах, эти протоколы не определяют компонентной архитектуры для инкапсуляции методов, реализующих логику приложений. Разумеется, можно реализовать логику приложения в пределах сервлетов и/или методов CORBA, а затем определить интерфейсы клиент–серверного доступа. Однако в крупных проектах, когда за создание разнообразных классов отвечают различные разработчики, применение компонентной архитектуры позволяет максимально увеличить независимость конкретных объектных структур и их реализаций. Кроме того, для выполне-

ния отдельных функций могут приобретаться компоненты обработки данных, созданные сторонними разработчиками. Опять же, использование интерфейсных методов, встроенных в компонентную архитектуру, позволяет логически разделять все компоненты от сторонних разработчиков и фирменные элементы.

Основной компонентной архитектурой для Java-разработки является архитектура Enterprise JavaBean (корпоративные bean-компоненты, EJB). EJB-компоненты схожи с компонентами «JavaBeans», спроектированными для применения с апплетами. Схожи в том смысле, что являются инкапсулирующими самостоятельными сущностями, которые посредством интерфейсов могут работать совместно с существующим кодом. Но на этом сходство и заканчивается: создание компонентов EJB, как правило, нацелено на обработку данных, а не на дизайн интерфейса. Компоненты EJB могут работать со службой транзакций Java (Java Transaction Service, JTS) и технологией JDBC в целях создания экземпляров объектов «на лету» – записей по сотрудникам и объектов всевозможных финансовых транзакций, – обеспечивая при этом существование долгоживущих объектов без необходимости использовать SQL. Это достигается с помощью объектно-реляционного отображения в случае стандартных реляционных СУБД либо применением объектно-ориентированных СУБД.

Одним из наиболее сложных аспектов EJB-разработки является успешное развертывание на сервере, которое и позволяет обеспечить доступ к службе различных клиентов. Для хранения компонентов, размещаемых на сервере в составе интегрированных приложений обработки данных, могут применяться архивы Java (*jar*). Как правило, развертывание связано с разработкой XML-файла дескриптора, который используется контейнером EJB для организации доступа к методам компонента с помощью его собственного интерфейса. Сервер EJB делает эти компоненты доступными клиентам.

Существуют два базовых вида компонентов EJB: компоненты сеансов (session beans) и компоненты хранения данных (entity beans). Короткоживущие компоненты сеансов (stateless session beans) схожи с протоколом HTTP в том смысле, что информация о состоянии не сохраняется между обращениями клиента. Это уместно в условиях, когда приходится экономить память, а отдельные транзакции составляют основную часть обращений. С другой стороны, долгоживущие компоненты сеансов (stateful session beans) хранят информацию о данных клиента, пока существует соединение между клиентом и сервером. Оба вида компонентов сеансов предоставляют полную поддержку транзакций.[1]

[1] В литературе по EJB иногда вышеприведенные термины называют session-компонентами и entity-компонентами. Также в EJB общеупотребимыми относительно «stateful» и «stateless» являются термины «с сохранением состояния» и «без сохранения состояния». – *Примеч. науч. ред.*

Компоненты хранения данных являются более сложными сущностями, поскольку обеспечивают долговременное существование объектов, схожее с хранением информации в базе данных. Это означает, что при перезагрузке системы либо внезапном завершении процесса сервера EJB-данные не будут потеряны. Для компонентов хранения данных доступно два варианта долговременного существования: реализуемое контейнером и реализуемое компонентом. Первый вариант означает сокращение трудозатрат на разработку, поскольку весь необходимый код уже существует. Но долговременное существование, реализуемое компонентом, является более гибким вариантом, поскольку для работы с данными каждого компонента должны быть написаны специализированные методы.

В этом разделе мы изучим развертывание простого короткоживущего EJB-компонента сеанса при помощи сервера приложений Inprise (IAS), который мы уже использовали для реализации транзакций CORBA в предшествующем разделе. Мы создадим компонент, реализующий простейшую службу, которая по запросу клиента возвращает текущую дату. Это полезное приложение, поскольку многие сетевые службы требуют синхронизации даты и времени локальных систем. Первым шагом в разработке нашего приложения (DateServer) является проектирование домашнего интерфейса (хранимого в файле *DateServerHome.java*):

```
import javax.ejb.*;
import java.rmi.RemoteException;
public interface DateServerHome extends EJBHome
{
  DateServer create() throws RemoteException, CreateException;
}
```

Этот код отвечает непосредственно за создание объекта *DateServer*. Второй шаг – спроектировать сервер, в котором объявлены методы, доступные внешним клиентам (код хранится в файле *DateServer.java*):

```
import javax.ejb.*;
import java.rmi.RemoteException;
import java.rmi.Remote;

public interface DateServer extends EJBObject
{
  public String dateserver() throws RemoteException;
}
```

Здесь просто декларируется интерфейс *DateServer* как расширение класса EJBObject, единственный метод которого, *dateserver( )*, возвращает строку. Эта строка содержит дату; и таким образом внешние клиенты получают интерфейс для вызова метода *dateserver( )*. Теперь рассмотрим реализацию этого метода в bean-компоненте (код хранится в файле *DateServerHome.java*):

```
import javax.ejb.*;
import java.util.*;
import java.text.*;
public class DateServerBean implements SessionBean {

  public void ejbCreate() {
      System.out.println("ejbCreate()");
  }

  public void ejbRemove() {
      System.out.println("ejbRemove()");
  }

  public void ejbActivate() {
      System.out.println("ejbActivate()");
  }

  public void ejbPassivate() {
      System.out.println("ejbPassivate()");
  }

  public void setSessionContext(SessionContext ctx) {
      System.out.println("setSessionContext()");
  }

  public String dateserver() {
     java.text.SimpleDateFormat f = new java.text.SimpleDateFormat("dd/MM/yyyy")
;
      java.util.Date d = new java.util.Date();
      String dateString = f.format(d);
      return dateString;
  }
}
```

Можно видеть, что здесь реализовано несколько стандартных методов EJB, в частности *ejbCreate( )*, *ejbRemove( )* и *ejbActivate( )*. Единственный метод, содержащий собственно логику приложения, уже объявлен в сервере: это метод *dateserver( )*, возвращающий строку с датой, полученной из объекта SimpleDateFormat. И наконец, мы должны создать XML-дескриптор развертывания для компонента DateServer:

```
<?xml version="1.0"?>

<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.
1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">

<ejb-jar>
    <description>
        Это пример для короткоживущего компонента сеанса
    </description>
    <enterprise-beans>
        <session>
            <description>
              Этот короткоживущий компонент сеанса реализует выдачу текущей даты
```

```
            </description>
            <ejb-name>dateserver</ejb-name>
            <home>DateServerHome</home>
            <remote>DateServer</remote>
            <ejb-class>DateServerBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>dateserver</ejb-name>
                <method-name>*</method-name>
            </method>
            <trans-attribute>NotSupported</trans-attribute>
        </container-transaction>
    </assembly-descriptor>
</ejb-jar>
```

Достаточно просто реализовать классы приложения DateServer. Теперь следует создать метод клиента, который будет обращаться к методу *dateserver( )* (код хранится в файле *DateServerClient.java*):

```
import javax.ejb.*;
import javax.naming.*;
import java.rmi.*;
import java.util.Properties;

public class DateServerClient
{
       public static void main(String[] args)
       {
              try
              {
                     System.out.println("Создается контекст");
                     Context ctx = new InitialContext();
                     System.out.println("Поиск контекста");
                     Object ref = ctx.lookup("dateserver");
                     System.out.println("Поиск домашнего интерфейса");
                     DateServerHome home =    (DateServerHome) javax.rmi.
                        PortableRemoteObject.narrow(ref, DateServerHome.class);
                     System.out.println("Вызов create()");
                     DateServer dateserver = home.create();
                     System.out.println("Сегодняшняя дата: "+dateserver.
                         dateserver());
                     dateserver.remove();
              } catch (Exception e) {
                      e.printStackTrace();
              }
       }
}
```

Клиент создает новый начальный контекст, прежде чем производить поиск контекста объекта DateServer. После получения ссылки на объект создается экземпляр домашнего интерфейса и объект DateServer. Наконец, вызывается удаленный метод и отображается текущая дата. Если во время удаленного вызова метода возникает исключение, отображается цепочка вызовов в стеке.

Для сборки приложения можно воспользоваться простым файлом *Makefile*:

```
# Makefile

default: all

.SUFFIXES: .java .class .idl .module

JAVA2IIOP = java2iiop -compile

IDL2JAVA = idl2java -compile -boa

GENERATE_JAR = vbj com.inprise.ejb.util.GenerateJar

JAR = jar

VERIFY = vbj com.inprise.ejb.util.Verify

DD2XML = vbj com.inprise.ejb.util.dd2xml

.java.class:
        vbjc $<

.idl.module:
        $(IDL2JAVA) $<
        touch $@

clean:
       -rm -f *Helper.java *Holder.java *Stub.java *Operations.java *POA*.java
*DefaultFactory.java
       -rm -f *.ser *.class *.jar *.module *.jds *~ *.xml *_LOGA_* *_STATUS_*

SRCS = \
        DateServerClient.java \
        DateServer.java \
        DateServerHome.java \
        DateServerBean.java

CLASSES = $(SRCS:.java=.class)

dateserver_beans.jar: $(CLASSES) META-INF/ejb-jar.xml META-INF/ejb-inprise.xml
        $(JAVA2IIOP) DateServerHome
        $(JAR) cMf dateserver_beans.jar META-INF *.class
        $(VERIFY) dateserver_beans.jar

all: $(CLASSES) dateserver_beans.jar
```

Этот код отвечает за компиляцию кода сервера и клиента, трансляцию Java в IIOP, а также создание Java-архива для развертывания компонента. Сборка приложения происходит по команде *make*:

```
bash-2.03$ make
vbjc DateServerClient.java
vbjc DateServerBean.java
java2iiop -compile DateServerHome
jar cMf dateserver_beans.jar META-INF *.class
vbj com.inprise.ejb.util.Verify dateserver_beans.jar
```

Теперь можно выполнить приложения сервера и клиента. Рассмотрим типичное взаимодействие клиента и сервера в архитектуре EJB:

```
bash-2.03$ vbj com.inprise.ejb.Container ejbcontainer dateserver_beans.jar -jts

Inprise EJB Container
====================
     server version : 4.1
  server build date : July 26, 2000
       java version : 1.2.2
        java vendor : Sun Microsystems Inc.
          heap size : 8 Mb
      javax.* flags : javax.rmi.PortableRemoteObjectClass:
com.inprise.vbroker.
                        rmi.PortableRemoteObjectImpl
                    : javax.rmi.CORBA.UtilClass:
com.inprise.vbroker.rmi.CORBA.
                        UtilImpl
                    : javax.rmi.CORBA.StubClass:
com.inprise.vbroker.rmi.CORBA.
                        StubImpl
    vbroker.* flags : vbroker.orb.procId: 17827
    org.omg.* flags : org.omg.CORBA.ORBClass: com.inprise.vbroker.orb.ORB
                    : org.omg.CORBA.ORBSingletonClass:
com.inprise.vbroker.orb.
                        ORBSingleton
    java class path : /usr/local/jdk-1.2.2/lib/tools.jar
                    : /usr/local/jdk-1.2.2/lib/dt.jar
: .
                    : /usr/local/inprise/ias41/lib
                    : /usr/local/inprise/ias41/lib/vbjorb.jar
                    : /usr/local/inprise/ias41/lib/vbsec.jar
                    : /usr/local/inprise/ias41/lib/vbejb.jar
                    : .
                    : /usr/local/jdk-1.2.2/bin/../jre/lib/rt.jar
                    : /usr/local/jdk-1.2.2/bin/../lib/tools.jar

====================
Initializing ORB...... done
Initializing JTS..... done
Initializing EJBs...... done
Container [ejbcontainer] is ready
                EJB Container Statistics
                ========================
                Time                    Tue Oct 03 15:13:59 GMT+10:00 2000
```

```
Memory (used)           1875 Kb (max 1875 Kb)
Memory (total)          8192 Kb (max 8192 Kb)
Memory (free)           77.0%
------------------------
Home                    dateserver
Total in memory         0
Total in use            0
```

Когда сервер загружен в память, можно выполнить клиентское приложение и получить от сервера текущую дату:

```
bash-2.03$ vbj DateServerClient
Создается контекст
Поиск контекста
Поиск домашнего интерфейса
Вызов create()
setSessionContext()
ejbCreate()
Сегодняшняя дата: 03/10/2000
```

Итак, дата успешно получена клиентом и отображена. Очевидно, по мере развития приложения могут появляться новые методы и классы. Но подход, который мы использовали для проектирования, разработки, развертывания и тестирования приложения, состоящего из одного метода, является основательным фундаментом для продолжения разработки и/или добавления классов, созданных сторонними разработчиками.

Установка сервера баз данных

Как мы могли видеть в двух предшествующих разделах, распределенным архитектурам обычно требуется базовое хранилище информации для хранения долгоживущих объектов и реляционных данных. У большинства крупных организаций есть хранилища данных или серверы баз данных, которые могут использоваться в сочетании с архитектурами CORBA и EJB в целях разработки информационных веб-систем, поскольку все наиболее популярные СУБД сегодня поддерживают JDBC. И хотя объектно-ориентированные базы данных являются более удобным решением для EJD-компонентов хранения данных, компонентам сеансов и метода CORBA требуется SQL-доступ по JDBC. Таким образом, любая база данных, соответствующая стандарту ANSI SQL и имеющая JDBC-драйвер, может использоваться для организации доступа к данным интерфейсов CORBA и EJB-компонентов сеансов.

В реляционных СУБД для задания отношений переменных используются таблицы. Формат таблицы удобен, поскольку позволяет создавать записи, соотносимые с данными конкретных сущностей. Например таблица адресной книги может содержать фамилию, имя, адрес, город, почтовый индекс и номер телефона. Очевидно, что для хране-

ния этих переменных требуются различные типы данных: телефонный номер содержит только цифры, в то время как фамилия – только буквы. Стандарт SQL определяет различного рода переменные, которые могут использоваться в таблицах для описания данных и определения видов операций, которые могут выполняться над этими данными.

Мы уже рассматривали JDBC-вариант доступа к данным в примере CORBA PassCheck, созданном ранее. Вот один оператор SQL, подбирающий пароль для определенного имени пользователя:

```
select password from users where username=...
```

Очевидно, таблица users, состоящая по меньшей мере из двух колонок (username, password), должна быть предварительно создана на JDBC-совместимом сервере баз данных, чтобы приложение корректно работало.

Выбор СУБД – непростая задача. С одной стороны, перечень возможностей растет с каждым годом; соответствие стандартам ANSI становится все более эфемерным по мере появления «надмножеств» SQL, разрабатываемых поставщиками СУБД, а требования к памяти и процессорам возрастают с каждой версией. Очевидно, для высокодоступной системы с кластеризацией и поддержкой сложного управления хранением данных может потребоваться полномасштабный коммерческий СУРБД-сервер. Но если нужна просто надежная основа, с которой можно работать по ODBC, изучение альтернативных вариантов может оказаться нелишним.

Одним из ведущих бесплатных продуктов является СУБД PostgreSQL (*http://www.postgresql.org/*), которая поддерживается на платформах Solaris SPARC и Solaris Intel. PostgreSQL во многом отличается от прочих бесплатных СУБД, поскольку имеет общие корни с некоторыми из коммерческих СУРБД, доступными на рынке. PostgreSQL является транзакционной СУБД, что делает систему идеальным выбором для обработки веб-транзакций. Кроме того, исходные тексты PostgreSQL также распространяются свободно, что облегчает настройку, оптимизацию и отладку кода в любых целях. Последняя версия PostgreSQL – 7.0.

В этом разделе мы рассмотрим создание таблиц, необходимых для работы приложения CORBA PassCheck, а также процесс добавления и удаления данных из таблицы. Для работы с данными мы будем использовать диалоговый инструмент *pgsql*, но могут быть созданы методы JDBC, выполняющие многие из этих задач в приложении Java.

Прежде чем переходить к созданию таблицы *users*, мы должны создать учетную запись администратора (*postgres*), а также установить и настроить сервер PostgreSQL, после чего создать раздел базы данных *users*, в котором и будет создана таблица *users*. Администратор *postgres* обычно связывается с учетной записью *postgres* системы Solaris; и, вообще говоря, сервер баз данных *postgres* должен работать с полномочиями пользователя *postgres*. Следует создать учетную запись поль-

зователя *postgres*, которому принадлежат все исходные и исполняемые файлы. Помимо этого, следует создать группу *postgres*, единственным членом которой будет пользователь *postgres*. Опять же, все исходные и исполняемые файлы должны принадлежать группе *postgres*.

Сервер баз данных PostgreSQL использует диспетчер соединений, во многом схожий с диспетчером соединений JDBC, который обеспечивает подключение клиентских приложений к базе данных. Этот диспетчер известен под именем postmaster, он отвечает за первичную обработку всех клиентских сеансов, в частности за интерпретацию блокировок и проверку подлинности. Удобной чертой PostgreSQL является разграничение доступа к табличным данным на основе пользовательских и групповых данных Solaris, а не иной схемы идентификации. Это сокращает затраты на администрирование сервера баз данных, поскольку исчезает необходимость дополнительно администрировать таблицы идентификации пользователей (в отдельных коммерческих системах ситуация выглядит иначе).

Получив доступ к системе PostgreSQL, пользователи могут создавать и уничтожать базы данных, которые им принадлежат. Вспомните, с какими трудностями может быть сопряжено создание дополнительных баз данных в некоторых коммерческих СУРБД – иногда требуется повторное выполнение программы установки. Как мы увидим, в PostgreSQL создание базы данных производится одной командой, для выполнения которой не надо даже быть привилегированным пользователем. Это в особенности полезно в среде разработки при выполнении регрессионных тестов, когда происходит создание многочисленных БД со схожей структурой с целью тестирования загрузкой и вероятными сбоями и/или отладки кода, вызывающего проблемы.

Как уже говорилось, после создания базы данных для передачи команд серверу баз данных используется клиент *pgsql*. Некоторые из команд PostgreSQL выполняются вне *pgsql*, в частности *createdb* (создание базы данных) и *destroydb* (удаление базы данных).

Для всех серверов баз данных, работающих под управлением Solaris, следует установить разумный объем разделяемой памяти в файле */etc/system*, иначе могут возникать ошибки *IpcMemoryCreate* при подключении к серверу большого числа клиентов. После добавления следующей строки в файл */etc/system* систему следует перезагрузить, а диспетчер соединений запустить повторно:

```
set shmsys:shminfo_shmmax=0x80000000
```

Когда система готова к работе с PostgreSQL, можно запустить диспетчер следующей командой (пользователя *postgres*); предполагается, что исполняемые файлы PostgreSQL расположены в */opt/postgres/bin*:

```
bash-2.03$ /opt/postgres/bin/postmaster -S -D /opt/postgres/data
```

Чтобы обращаться к удаленной базе данных PostgreSQL с помощью клиента *pgsql*, следует присвоить переменной среды *PGHOST* полное доменное имя или IP-адрес целевого узла.

Когда сервер запущен, можно создать базу данных командой *createdb*. Таким образом, чтобы создать базу данных *users*, мы выполним команду:

```
bash-2.03$ createdb users
```

База данных существует, и теперь можно передавать серверу команды в диалоговом режиме клиента *pgsql*:

```
bash-2.03$ pgsql users
Welcome to the POSTGRESQL interactive sql monitor:
  Please read the file COPYRIGHT for copyright terms of POSTGRESQL
[PostgreSQL 7 on i386-pc-solaris2.7, compiled by gcc 2.95.2]
   type \? for help on slash commands
   type \q to quit
   type \g or terminate with semicolon to execute query
 You are currently connected to the database: users
users=>
```

Как можно видеть, *pgsql* работает в системе Solaris Intel абсолютно без проблем. Чтобы теперь создать таблицу *users*, воспользуемся такой командой SQL:

```
users=> create table users(username varchar(8), password varchar(8), auth_level
    integer (1));
```

В этом операторе указано, что таблица *users* состоит из трех колонок: имя пользователя (*username*) до восьми символов длиной, пароль (*password*) до восьми символов длиной и уровень доступа (*auth_level*), который может использоваться для разграничения доступа к данным в приложении. Такая возможность очень полезна во внутрикорпоративных сетях, поскольку существует большое число различных классов пользователей (гости, обычные пользователи, администраторы). Когда таблица создана, сервер возвращает подтверждение, которое отображается на экране:

```
CREATE
```

Теперь мы можем добавлять строки данных, соответствующие пользователям нашей системы. В следующем примере мы создадим записи для трех пользователей с различными уровнями доступа: 1 (администратор), 2 (обычный пользователь) и 3 (гость):

```
users=> insert into users values ('guest,guest',3);
users=> insert into users values ('pwatters','L1nuX123',1);
users=> insert into users values ('mwatters','f1nance1',2);
```

После добавления каждой из строк сервер точно так же возвращает подтверждение:

```
INSERT 398754 1
```

Перечислить строки таблицы *users* в порядке их добавления можно так:

```
solaris=> select * from users;

username|password|auth_level
--------+--------+----------
guest   |guest   |3
pwatters|L1nuX123|1
mwatters|f1nance1|2
 (3 rows)
```

Однако возможно вывести лишь определенные колонки таблицы *users*:

```
users=> select username, password from users;

username|password
--------+--------
guest   |guest
pwatters|L1nuX123
mwatters|f1nance1
 (3 rows)
```

Кроме того, мы можем упорядочить список пользователей по уровню доступа, скажем, в целях создания отчета:

```
users=> select * from users order by auth_level;

username|password|auth_level
--------+--------+----------
pwatters|L1nuX123|1
mwatters|f1nance1|2
guest   |guest   |3
 (3 rows)
```

Выбрать всех пользователей с высшим уровнем доступа можно такой командой:

```
users=> select username from users where auth_level=1;

username
--------
pwatters
(1 row)
```

Когда таблица больше не нужна приложению, ее можно с легкостью удалить командой *drop*:

```
users=> drop table users;
```

Это лишь некоторые из команд, используемых обычно для помещения и извлечения данных из реляционных СУБД, понимающих SQL. Большинство этих команд могут выполняться из приложений Java через JDBC, что позволяет работать с таблицами баз данных напрямую, без необходимости применять инструмент *pgsql*. PostgreSQL поддерживает стандарт ANSI SQL, работает на платформах Solaris SPARC и Solaris Intel и потому является идеальным выбором для поддержки приложений Java на платформах Solaris.

Алфавитный указатель

Числа

A

B

C

D

E

F

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

А

Б

В

Г

Д

Ж

З

И

К

Л

М

Н

О

П

P

C

Т

У

Ф

Х

Ц

Ч

Ш

Э

Я