2-е издание

ВНУТРЕННЕЕ УСТРОЙСТВО LINUX

Дмитрий Кетов

bhv®

УДК 004.451
ББК 32.973.26-018.2
К37

Кетов Д. В.

К37 Внутреннее устройство Linux. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2021. — 400 с.: ил.

ISBN 978-5-9775-6630-8

Книга представляет собой введение во внутреннее устройство операционной системы Linux. Все положения наглядно проиллюстрированы примерами, разработанными автором и проверенными им на практике. Рассмотрены основные подсистемы ядра и их сущности — файлы и файловые системы, виртуальная память и отображаемые файлы, процессы, нити и средства межпроцессного взаимодействия, каналы, сокеты и разделяемая память. Раскрыты дискреционный и мандатный (принудительный) механизмы контроля доступа, а также привилегии процессов. Подробно описано пользовательское окружение и интерфейс командной строки CLI, оконные системы X Window и графический интерфейс GUI, а также сетевая подсистема и служба SSH. Особое внимание уделено языку командного интерпретатора и его использованию для автоматизации задач эксплуатации операционной системы.

Во втором издании добавлены новые главы, описывающие графическую систему Wayland, контейнеры, виртуализацию и функционирование Linux как единой системы всех своих компонент, учтены изменения в последних версиях ОС, а также пожелания и отзывы читателей.

Для студентов, пользователей, программистов и системных администраторов Linux

УДК 004.451
ББК 32.973.26-018.2

Группа подготовки издания:

Руководитель проекта	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Сависте*
Компьютерная верстка	*Ольги Сергиенко*
Дизайн обложки	*Марины Дамбиевой*
Оформление обложки	*Карины Соловьевой*

Подписано в печать 02.11.20.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 32,25.
Тираж 1200 экз. Заказ № 12456.
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Отпечатано с готового оригинал-макета
ООО "Принт-М", 142300, М.О., г. Чехов, ул. Полиграфистов, д. 1

ISBN 978-5-9775-6630-8

Оглавление

Введение

О чем эта книга?

Книга, которую вы держите в руках, адресована начинающим пользователям[1] операционной системы Linux и представляет собой ***иллюстрированное*** введение в ее ***внутреннее устройство*** — от ядра до сетевых служб и от утилит командной строки до графического интерфейса.

По моему скромному мнению, никакие книги не могут превратить читателя из новичка в эрудированного специалиста — ни обзорные, рассказывающие «ничего обо всем», ни узкоспециализированные, повествующие «все об одном». Без самостоятельного опыта, получаемого в результате исследования происходящих процессов и явлений, осмысление любого книжного знания практически невозможно.

Именно поэтому данная книга преследует всего одну важную цель — привить читателю ***навыки*** самостоятельного ***исследования*** постоянно ***эволюционирующей*** операционной системы Linux через ***умения*** пользоваться соответствующим инструментарием и на основе теоретических ***знаний*** о ее устройстве и философии[2].

Порядок и стиль изложения материала, иллюстративные изображения и листинги, приведенные в книге, являются результатом обобщения моего достаточно долгого опыта преподавания технических дисциплин, посвященных операционным системам в целом и операционной системе Linux в частности. Именно такой мне представляется картина «правильного» минимального набора знаний, умений и навыков, необходимых для построения качественной ***ментальной модели*** современной ОС Linux.

---

[1] Хотя термин «пользователь» зачастую ассоциируется с этаким домашним любителем или офисным работником, но никак не с IT-специалистом, книга адресована абсолютно всем, кому интересно понимать то, как устроена операционная система изнутри. Именно для ее компетентного применения в своей профессиональной деятельности такое понимание в первую очередь нужно программистам и системным администраторам — разработчикам и эксплуатационникам информационных систем на базе Linux.

[2] **W:[Философия UNIX]**.

Основная задача книги — иллюстративно изложить внутреннее устройство операционной системы Linux, связав ***«теорию»*** и ***«практику»***. В отличие от многих изданий, скупо и сухо излагающих теоретические аспекты построения операционной системы или излишне концентрирующихся на деталях конкретной реализации того или иного ее программного обеспечения, в книге рассматриваются ***абстрактные*** концепты внутреннего устройства ОС, иллюстративно подкрепляемые примерами анализа (а иногда и синтеза) ее ***конкретных*** компонентов и связей между ними.

Все части операционной системы рассматриваются в контексте типичных задач, решаемых при их помощи на практике, а сама иллюстрация проводится ***посредством*** соответствующего инструментария пользователя, администратора и разработчика.

Многие концепты и компоненты, инструменты и задачи будут иметь ***общность***, характерную для всех **W:[дистрибутивов Linux]**[1], но их реализации могут существенно ***различаться***. Поэтому, преследуя цель сохранить практическую значимость и жертвуя некоторой потерей общности излагаемого материала, повествование ведется в контексте отдельно взятой операционной системы, в качестве которой выступает дистрибутив **W:[Ubuntu Linux]** (версии 19.10, выпущенной в октябре 2019 года). Читатель может ***свободно*** скачать этот дистрибутив из Интернета и использовать в абсолютно любых целях, например для проработки примеров, приведенных в книге.

Выбор в пользу дистрибутива **W:[Ubuntu Linux]**, в частности, сделан еще и по совокупности присущих ему свойств, как то: большое сообщество его пользователей (включая русскоговорящих), громадная пакетная база, регулярность обновлений и их стабильность, дружественность к начинающим и, наконец, широкая распространенность технических решений на его основе — от бизнес-решений и до сертифицированных разработок в оборонной промышленности.

Кому адресована книга

Понимание внутреннего устройства важно как в системном администрировании для направленной диагностики проблем эксплуатации информационных систем, построенных на платформе Linux, так и для разработки программного обеспечения, учитывающего особенности ее внутреннего устройства.

Поэтому книга адресуется абсолютно всем, кто не любит использовать непрозрачный «черный ящик», кому интересна механика всех этих шестеренок, что вращаются внутри Linux, ежедневно решая задачи миллионов пользователей по всему

[1] А зачастую и для всех операционных систем семейства **W:[UNIX]**, к которому Linux принадлежит.

миру, а также всем, кто в детстве разобрал не один игрушечный луноход в попытке узнать, как там чувствуют себя человечки в скафандрах.

Принятые соглашения и обозначения

Наиболее полезным навыком при исследовании любой незнакомой информационной системы является способность оперативно получать справочную информацию непосредственно изнутри ее, не прибегая к внешним источникам (вроде поиска по Google). Именно поэтому в данной книге *нигде* не встречаются даже частичные цитирования содержания справочных систем.

Классическим электронным справочником UNIX являются страницы руководства (manual pages, подробнее *см. главу 2*), которые сгруппированы в тематические секции, а ссылки на страницу *page* в секции *N* принято записывать как **page(N)**, в том числе и в этой книге. Другая полезная информация, в том числе историческая, доступна из статей всемирной онлайн-энциклопедии Wikipedia, поэтому полезные ссылки на статью *useful article* изображаются как W:**[useful article]**.

Разнообразные значки [☛♁?☻☠⊶❶...❾①...⑨☼♠⇆⇅✂] на полях и в теле листингов позволяют акцентировать внимание на *нужном* и сопровождать *важное* дополнительными комментариями в тексте повествования.

Полужирный шрифт **command** в листингах идентифицирует вводимые пользователем команды или управляющие символы и последовательности, а нажатия на клавиши или их комбинации изображаются как Ctrl Alt F4.

Курсив используется для выделения *понятий* и *терминов*, а полужирный курсив выделяет ***ключевые слова***, определяющие основное существо текста абзаца.

Значок указательного пальца ☛ и ☟ и вот такое «выделение маркером» укажет на те места в листингах, куда нужно обратить внимание. Номера ❶...❾①...⑨ и звездочка ★ позволят сослаться на конкретные места листингов в основном тексте.

Вопрос ? отметит те места листингов, которые должны вызывать недоумение вида «А это еще что?», а знак восклицания ! укажет на места, которые должны сопровождаться возгласами «Ох ты ж! ничего себе!». На особенно эмоциональные места листингов указывает значок ☠, а поднятый вверх большой палец ♁, как и обычно, одобряет определенные команды листингов, которые приводят к долгожданному результату.

Аналогично, закрытый замок 🔒 символизирует закрытость (заблокированность) или отсутствие доступа, а открытый замок 🔓, наоборот, — открытость (разблокированность), наличие доступа. Ключ ⊶ указывает на операцию открытия/закрытия чего-либо (например, ввод пароля при аутентификации).

Кинжал 🗡 предупреждает о выполнении некоторой деструктивной операции, а значки часов 🕐🕑🕒🕓 символизируют ожидание завершения достаточно длительной операции.

Для удобства чтения листингов значки ⇆⇵ указывают на «замещающий» вывод (т. е. без скроллинга), круговые стрелки 🔄 обозначают очередную итерацию некоторого цикла, а значок 🚫 указывает на его окончание. Ну и, наконец, разнообразные стрелки ↴↩▸→←↑ связывают отдельные части листинга друг с другом.

Методические рекомендации

Многие иллюстративные листинги, приведенные в книге, нужно воспринимать не как «дополнение» к основному тексту, а как сам основной текст, просто (пока) на незнакомом и непонятном языке, а саму книгу — как учебник нового иностранного языка.

Получение читателем собственного отклика от взаимодействия с операционной системой представляется мне наиболее значимым результатом освоения содержимого этой книги. Поэтому чтение стоит непременно сопровождать самостоятельным выполнением команд листингов, т. к. успешное изучение нового иностранного языка никак невозможно без его непосредственного практического использования в процессе изучения.

Не менее важным для построения целостных взаимоотношений с системой является самостоятельное изучение первоисточников знаний[1] — ее внутренней документации, страниц руководства **man(1)**.

Для выполнения команд листингов и доступа к внутренним справочникам потребуется сама операционная система, что несложно получить, например инсталлировав настольную версию Ubuntu Linux в виртуальную машину **W:[VirtualBox]**.

Дополнительную пользу принесет ознакомление и с дополнительными источниками информации в Интернете, например со статьями на **http://help.ubuntu.ru**, а также поиск ответов на возникающие вопросы на **http://forum.ubunu.ru**, **http://askubuntu.com** и даже на **http://UNIX.stackexchange.com** и **http://stackoverflow.com**.

Кроме текстовых источников информации в Интернете можно воспользоваться услугами таких очаровательных онлайн-сервисов, как **http://explainshell.com** для объяснений конструкций и **http://www.shellcheck.net** для проверки проблем в конструкциях командного интерпретатора. Для проверки примеров из книги и разработки собственных сценариев командного интерпретатора незаменимую помощь может оказать сервис **http://www.tutorialspoint.com/execute_bash_online.php**.

---

[1] Это обязательно потребует знания технического английского языка, но без него вообще трудно рассчитывать на сколько-нибудь значительные успехи в отрасли информационных технологий.

Что должен знать читатель

В операционной системе Linux программное обеспечение поставляется в виде так называемых *пакетов (package)* — специальным образом подготовленных архивов, содержащих само *программное* обеспечение, его *конфигурационные* файлы, его данные и *управляющую* информацию. Управляющая информация пакета включает контрольные суммы устанавливаемых файлов, зависимости устанавливаемого пакета от других пакетов, краткое описание пакета, сценарии установки, сценарии удаления пакета и прочие данные, необходимые *менеджеру* пакетов.

В примерах, приведенных в листингах, часто встречаются программы, которые, возможно, не будут установлены «по умолчанию», поэтому важно знать и понимать, как устроены подсистема управления установкой и удалением программного обеспечения — так называемые менеджеры пакетов и зависимостей[1].

Менеджер пакетов производит непосредственную установку и удаление пакетов программного обеспечения, а также ведет их учет в системе. Вспомогательная «грязная» работа по подбору зависящих друг от друга пакетов, получению их из *репозиториев*[2] (например, скачивание с FTP/HTTP-серверов), выбору правильных версий пакетов, определению правильного их порядка установки достается *менеджеру зависимостей.*

При помощи менеджера пакетов (листинг B1) всегда можно узнать имя пакета ❶, в который входит та или иная *установленная* компонента операционной системы (например, **/bin/date**), или, наоборот, узнать список компонент ❷, *установленных* из указанного пакета (например, **coreutils**).

Листинг B1. Пакетный менеджер dpkg

```
    bart@ubuntu:~$ which -a date
    /usr/bin/date
    /bin/date
❶   bart@ubuntu:~$ dpkg -S /bin/date
    coreutils: /bin/date
❷   bart@ubuntu:~$ dpkg -L coreutils
                    ...
```

[1] Условно, можно выделить две ветви операционной системы Linux — ветвь debian, к которой относятся дистрибутивы **W:[Debian]** и **W:[Ubuntu]**, и ветвь redhat, куда нужно отнести **W:[RHEL]**, **W:[CentOS]** и **W:[Fedora]**. В debian-ветви используется пакетный менеджер **dpkg** и построенные над ним менеджеры зависимостей **apt**, **aptitude**, **synaptic** и **software-center**, а в ветви redhat — пакетный менеджер **rpm** и основной менеджер зависимостей **yum**.

[2] Хранилище пакетов у разработчиков дистрибутива Linux.

```
/bin/chmod
/bin/chown
/bin/cp
/bin/date
/bin/dd
/bin/df
...
bart@ubuntu:~$ dpkg -s coreutils
Package: coreutils
Essential: yes
Status: install ok installed
Priority: required
Section: utils
Installed-Size: 7196
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Architecture: amd64
Multi-Arch: foreign
Version: 8.30-3ubuntu2
Pre-Depends: libacl1 (>= 2.2.23), libattr1 (>= 1:2.4.44), libc6 (>= 2.28), libselinux1 (>=
2.1.13)
Description: GNU core utilities
 This package contains the basic file, shell and text manipulation
 utilities which are expected to exist on every operating system.
 .
 Specifically, this package includes:
 arch base64 basename cat chcon chgrp chmod chown chroot cksum comm cp
 csplit cut date dd df dir dircolors dirname du echo env expand expr
 ...                ...                ...
Homepage: http://gnu.org/software/coreutils
Original-Maintainer: Michael Stone <mstone@debian.org>
```

Если при попытке выполнить ту или иную команду операционной системы Ubuntu Linux (это одна из причин, по которой иллюстрация в книге ведется именно с ее помощью) обнаружится, что нужный пакет с программным обеспечением не установлен, то при наличии доступа в Интернет можно тривиальным способом доустановить недостающие компоненты (листинг B2).

Листинг B2. Установка недостающего программного обеспечения

```
bart@ubuntu:~$ finger
Команда «finger» не найдена, но может быть установлена с помощью:
sudo apt install finger
```

```
bart@ubuntu:~$ sudo apt install finger
[sudo] password for bart:
Чтение списков пакетов... Готово
Построение дерева зависимостей
Чтение информации о состоянии... Готово
Следующие НОВЫЕ пакеты будут установлены:
  finger
Обновлено 0 пакетов, установлено 1 новых пакетов, для удаления отмечено 0 пакетов,
и 11 пакетов не обновлено.
Необходимо скачать 16,9 kB архивов.
После данной операции объём занятого дискового пространства возрастёт на 51,2 kB.
Пол:1 http://ru.archive.ubuntu.com/ubuntu eoan/universe amd64 finger amd64 0.17-17 [16,9 kB]
Получено 16,9 kB за 0c (200 kB/s)
                ...                ...                ...
Подготовка к распаковке .../finger_0.17-17_amd64.deb ...
Распаковывается finger (0.17-17) ...
Настраивается пакет finger (0.17-17) ...
Обрабатываются триггеры для man-db (2.8.7-3) ...
```

В редких случаях, когда нужно узнать, с каким, даже еще неустановленным, пакетом программного обеспечения поставляется тот или иной файл, может выручить утилита **apt-file(1)** (❶, листинг В3). В обратную сторону посмотреть список файлов, входящих в еще неустановленный пакет, можно этой же утилитой ❷.

Листинг В3. В каком пакете файл (файлы)?

```
❶ bart@ubuntu:~$ apt-file search bin/7z
  Finding relevant cache files to search ...E: The cache is empty. You need to run "apt-file
  update" first.
  bart@ubuntu:~$ sudo apt-file update
  Сущ:1 http://ru.archive.ubuntu.com/ubuntu eoan InRelease
  Пол:2 http://ru.archive.ubuntu.com/ubuntu eoan-updates InRelease [97,5 kB]
  Пол:3 http://ru.archive.ubuntu.com/ubuntu eoan-backports InRelease [88,8 kB]
                ...                ...                ...
  Получено 91,9 MB за 37c (2 515 kB/s)
  Чтение списков пакетов... Готово
  Построение дерева зависимостей
  Чтение информации о состоянии... Готово
  bart@ubuntu:~$ apt-file search bin/7z
  p7zip: /usr/bin/7zr
  p7zip-full: /usr/bin/7z
  p7zip-full: /usr/bin/7za
```

```
❷ bart@ubuntu:~$ apt-file show p7zip
   p7zip: /usr/bin/7zr
   p7zip: /usr/bin/p7zip
   p7zip: /usr/lib/p7zip/7zr
   p7zip: /usr/share/doc/p7zip/NEWS.Debian.gz
   p7zip: /usr/share/doc/p7zip/README.Debian
   p7zip: /usr/share/doc/p7zip/changelog.Debian.gz
   p7zip: /usr/share/doc/p7zip/copyright
   p7zip: /usr/share/man/man1/7zr.1.gz
   p7zip: /usr/share/man/man1/p7zip.1.gz
```

Совет для начинающих

И напоследок, самый важный совет для начинающих — начните!

Глава 1

Архитектура ОС Linux

1.1. Обзор внутреннего устройства

Операционная система (ОС) в общем и **W**:[Linux] в частности (рис. 1.1) представляет собой набор специализированных *программных* средств, обеспечивающих *доступ* потребителей (пользователей) к ресурсам (устройствам) и *распределение* последних между потребителями наиболее *удобным* и *эффективным* способом. Под ресурсами в первую очередь понимают аппаратные средства, такие как центральные процессоры, память, устройства ввода-вывода, дисковые и прочие накопители и другие периферийные устройства. Во вторую очередь к ресурсам относят такие «виртуальные» сущности (на рис. 1.1 не показаны), как процессы, нити, файлы, каналы и сокеты, семафоры и мьютексы, окна и прочие артефакты самой операционной системы, которые не существуют вне ее границ.

Как и во многих других операционных системах, в Linux выделяют два главных режима работы ее программных средств — ядерный режим (*kernel mode*), он же пространство ядра (*kernel space*), и пользовательский, внеядерный режим (*user mode*), или же пользовательское пространство (*user space*). Основное различие этих двух режимов состоит в привилегиях доступа к аппаратным средствам — памяти и устройствам ввода-вывода, к которым разрешен полный доступ из режима ядра и ограниченный доступ из режима пользователя.

Совокупность работающих в ядерном режиме программ называют *ядром*, которое в Linux состоит из основы (или же остова) и присоединяемых к ней объектов — динамически загружаемых модулей (подробнее *см. разд. 4.1.1*). Несмотря на компонентность, **W**:[Ядро Linux] относят к классу *монолитных* в силу того, что все его компоненты выполняются с одинаковыми (ядерными) привилегиями. В классе *микроядерных* систем, наоборот, компоненты ядра работают с разными привилегиями: основная компонента — микроядро (планировщик процессов/нитей и менеджер памяти) — с наибольшими привилегиями, менеджер ввода-вывода, драйверы устройств, файловые системы, сетевые протоколы и пр. — с другими, обычно меньшими привилегиями (возможно даже, с привилегиями пользовательского режима).

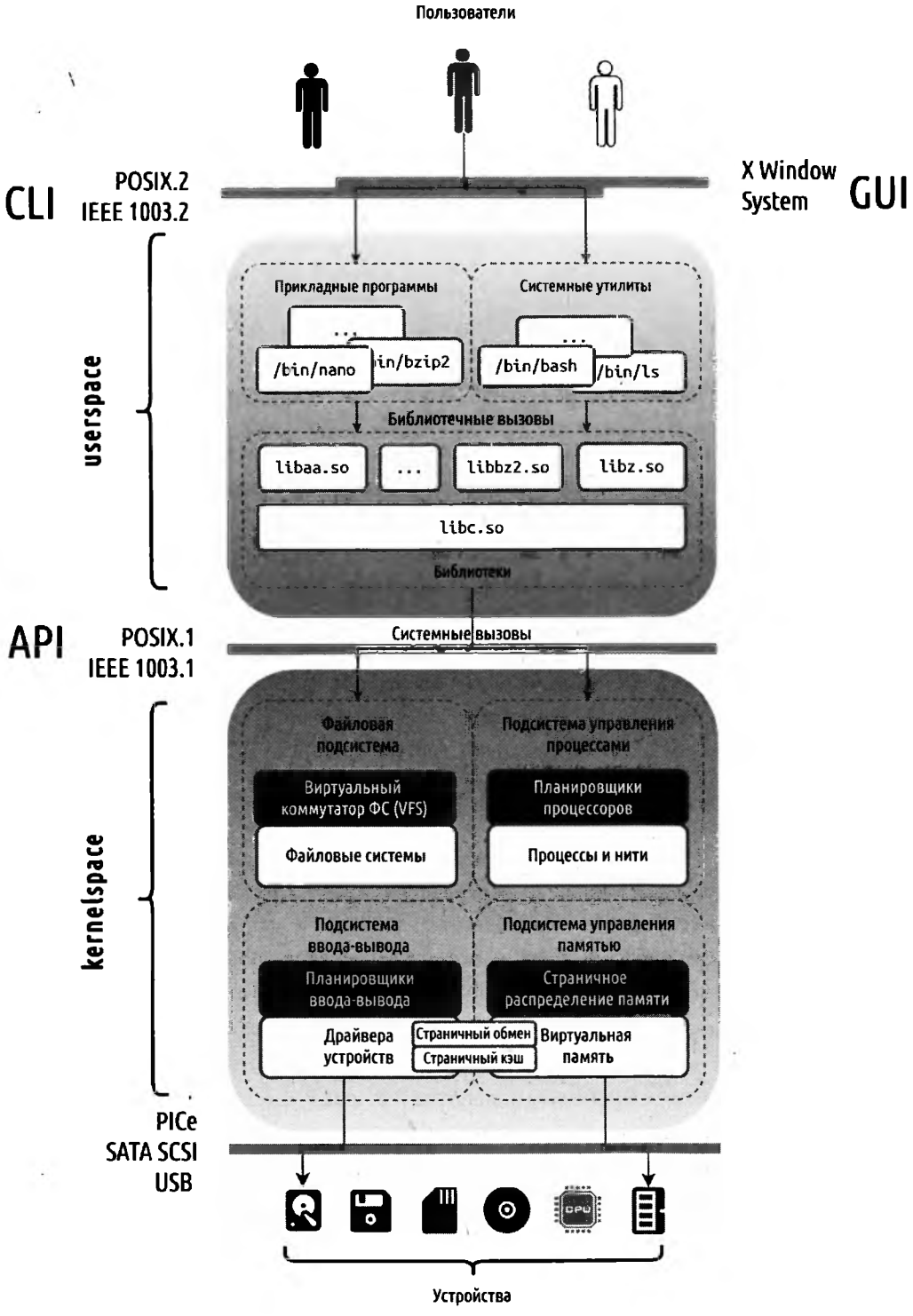


Рис. 1.1. Компоненты операционной системы Linux

1.2. Внеядерные компоненты: программы и библиотеки

Ядерные компоненты в основном обеспечивают решение задачи диспетчеризации (распределения) ресурсов между потребителями и предоставляют им базовый интерфейс доступа к ресурсам. Решение задачи обеспечения удобства доступа реализуется компонентами внеядерного режима — библиотеками динамической и статической компоновки (подробнее *см. разд. 4.1*).

Функции операционной системы, реализуемые ядерными компонентами, доступны внеядерным компонентам посредством *системных вызовов* — специализированных наборов обращений для получения услуг ядра. Системные вызовы выполняются в ядре, а вызываются при помощи основной внеядерной компоненты — библиотеки **libc.so** языка программирования W:[Си] (так исторически сложилось в силу того, что большая часть системных программных средств написана на этом языке).

Функции, реализуемые внеядерными компонентами, доступны посредством *библиотечных вызовов* и вызываются (выполняются) в самих библиотеках (например, алгоритмы сжатия информации в библиотеках **libz.so** и **libbz2.so**).

1.3. Ядерные компоненты: подсистемы управления процессами, памятью, вводом-выводом, файлами

Одна из основных задач ядра — распределение ресурсов между потребителями — вполне естественным образом приводит к тому, что среди ядерных компонент выделяют соответствующие (аппаратным ресурсам) менеджеры, так называемые подсистемы управления *процессами*, *памятью*, *вводом-выводом* и *файловую* подсистему.

Подсистема управления *процессами* распределяет время центральных процессоров (ЦП) между выполняющимися задачами (т. е. реализует W:[многозадачность]). Она создает и уничтожает такие сущности, как *процессы* и *нити (см. разд. 4.2)*, и организует одновременное (параллельное или псевдопараллельное) их выполнение при помощи планировщиков *(scheduler)*, реализующих алгоритмы распределения процессорного времени.

Подсистема *ввода-вывода* распределяет доступ к устройствам ввода-вывода (УВВ) между процессами и предоставляет им унифицированные интерфейсы *блочного, символьного (см. разд. 3.2.5)* и *пакетного* (сетевого) устройств *(см. разд. 6.1)*. Для устройств *внешней* памяти (дисковых или твердотельных накопителей, более медленных по сравнению с оперативной памятью) подсистема ввода-вывода организует W:[кэширование] при помощи подсистемы управления памятью.

Подсистема управления *памятью* (подробнее *см. разд. 4.7*) распределяет пространство оперативного запоминающего устройства (ОЗУ) между процессами при помо-

щи механизма страничного отображения — *выделяет* (и высвобождает) процессам страничные кадры физической памяти и *отображает* на страницы их адресного пространства. Кроме того, эта подсистема организует W:[**виртуальную память**] за счет механизма *страничного обмена* (W:[**подкачка страниц**]) — *вытесняет* неиспользуемые страницы процессов во внешнюю память и *загружает* их обратно по требованию при помощи подсистемы ввода-вывода.

Стоит отметить, что распределение ресурсов процессоров и устройств ввода-вывода происходит «во времени», т. е. в отдельные промежутки времени эти устройства выполняют операции только *одного* процесса или нити. Наоборот, распределение ресурсов запоминающих устройств происходит «в пространстве», т. к. информация *нескольких* процессов одновременно размещается в разных их *областях*.

Файловая подсистема ядра (подробнее *см. главу 3*) предоставляет процессам унифицированный интерфейс *файлового* доступа к внешней памяти (внешним запоминающим устройствам, ВЗУ — магнитным дисковым, твердотельным накопителям и т. п.) и распределяет между ними пространство ВЗУ при помощи *файлов* и *файловых систем*. Особенное назначение файловой подсистемы состоит еще и в том, что при помощи ее *файлового интерфейса* процессам предоставляется доступ и к другим подсистемам. Так, доступ к устройствам ввода-вывода организуется посредством специальных файлов устройств (блочных и символьных), например к CD/DVD-накопителю — через файл **/dev/sr0**, а к манипулятору мыши — через **/dev/input/mouse0**. Доступ к физической памяти и памяти ядра ОС организуется через файлы виртуальных устройств **/dev/mem** и **/dev/kmem**, а доступ процессов к страницам памяти друг друга — через файлы **/proc/PID/mem** псевдофайловой системы **procfs**. Даже доступ к внутренним параметрам и статистике различных компонент ядра операционной системы возможен через файлы разнообразных псевдофайловых систем, например **procfs**, **securityfs**, **debugfs** и прочих, а к списку обнаруженных ядром устройств ввода-вывода — через файлы псевдофайловой системы **sysfs**.

Кроме всех вышеперечисленных задач, файловая подсистема, подсистема ввода-вывода и подсистема управления процессами в совокупности предоставляют процессам средства межпроцессного взаимодействия, такие как сигналы *(см. разд. 4.8)*, каналы, сокеты и разделяемая память *(см. разд. 4.9)*.

1.4. Трассировка системных и библиотечных вызовов

Для наблюдения за обращениями программ к услугам ядерных компонент операционной системы, т. е. за системными вызовами, служит утилита **strace(1)**, предназначенная для трассировки — построения трасс выполнения той или иной про-

граммы. В листинге 1.1 представлена трассировка программ **whoami(1)**, **hostname(1)** и **pwd(1)** относительно системных вызовов **geteuid(2)**, **uname(2)**, **getcwd(2)** и **sethostname(2)**, где оказывается, что для получения и установки значения (сетевого) имени системы программа **hostname(1)** использует разные системные вызовы **uname(2)** и **sethostname(2)**, при этом один из системных вызовов является привилегированным и доступен только суперпользователю **root** *(см. разд. 2.7)*.

Листинг 1.1. Трассировщик системных вызовов strace

```
bart@ubuntu:~$ whoami
bart

bart@ubuntu:~$ hostname
ubuntu

bart@ubuntu:~$ pwd
/home/bart

bart@ubuntu:~$ strace -fe uname,getcwd,geteuid,sethostname whoami
geteuid()                               = 1000
bart
+++ exited with 0 +++

bart@ubuntu:~$ strace -fe uname,getcwd,geteuid,sethostname hostname
uname({sysname="Linux", nodename="ubuntu", ...}) = 0
ubuntu
+++ exited with 0 +++

bart@ubuntu:~$ strace -fe uname,getcwd,geteuid,sethostname pwd
getcwd("/home/bart", 4096)              = 11
/home/bart
+++ exited with 0 +++

bart@ubuntu:~$ strace -fe uname,getcwd,geteuid,sethostname hostname springfield
sethostname("springfield", 11)          = -1 EPERM (Операция не позволена)
hostname: you must be root to change the host name
+++ exited with 1 +++
```

В листинге 1.2 показана трассировка программы **date(1)** относительно библиотечных вызовов **fwrite(2)** или таких, в имени которых встречается строка **time**. Оказывает-

ся, что эти библиотечные функции находятся в библиотеке языка Си, что весьма ожидаемо в силу внутреннего устройства операционной системы (см. рис. 1.1).

Листинг 1.2. Трассировщик системных вызовов ltrace

```
bart@ubuntu:~$ ltrace -x *time*+fwrite date
clock_gettime@libc.so.6(0, 0x7ffc8601a300, 1, 0x56056f3b04c0) = 0
localtime_r@libc.so.6(0x7ffc8601a230, 0x7ffc8601a240, 4, 269) = 0x7ffc8601a240
strftime@libc.so.6( <unfinished ...>
strftime_l@libc.so.6(0x7ffc86019db0, 1024, 0x7ffc86019dab, 0x7ffc8601a240) = 5
<... strftime resumed> " \320\241\320\261", 1024, " %a", 0x7ffc8601a240) = 5
fwrite@libc.so.6("\320\241\320\261", 4, 1, 0x7f3d7e8d56a0)   = 1
strftime@libc.so.6( <unfinished ...>
strftime_l@libc.so.6(0x7ffc86019db0, 1024, 0x7ffc86019dab, 0x7ffc8601a240) = 7
<... strftime resumed> " \320\275\320\276\321\217", 1024, " %b", 0x7ffc8601a240) = 7
fwrite@libc.so.6("\320\275\320\276\321\217", 6, 1, 0x7f3d7e8d56a0) = 1
fwrite@libc.so.6("16", 2, 1, 0x7f3d7e8d56a0)                  = 1
fwrite@libc.so.6("20", 2, 1, 0x7f3d7e8d56a0)                  = 1
fwrite@libc.so.6("5", 1, 1, 0x7f3d7e8d56a0)                   = 1
fwrite@libc.so.6("9", 1, 1, 0x7f3d7e8d56a0)                   = 1
fwrite@libc.so.6("MSK", 3, 1, 0x7f3d7e8d56a0)                 = 1
fwrite@libc.so.6("2019", 4, 1, 0x7f3d7e8d56a0)                = 1
Сб ноя 16 20:05:09 MSK 2019
+++ exited (status 0) +++
```

1.5. Интерфейсы прикладного программирования

Системные и библиотечные вызовы Linux, формирующие интерфейс прикладного программирования (API, application programming interface), соответствуют определенным промышленным спецификациям, в частности (практически идентичным друг другу) стандартам **W:[POSIX]**[1], **Portable Operating System Interface** и **SUS**, **W:[Single UNIX Specification]**, доставшимся «в наследство» от семейства операционных систем UNIX, членом которого Linux и является.

Стандарт POSIX условно делится на две части: **POSIX.1**, программный интерфейс (API) операционной системы, и **POSIX.2**, интерфейс командной строки (CLI) пользователя *(см. главу 2)* и командный интерпретатор *(см. главу 5)*.

[1] Стандарт POSIX выпускается комитетом 1003 организации **W:[IEEE]**, поэтому имеет формальное обозначение IEEE 1003, а части стандарта POSIX.1 и POSIX.2 формально обозначаются IEEE 1003.1 и IEEE 1003.2 соответственно.

1.6. В заключение

Совершенно очевидно, что в современной медицине без понимания анатомии живых организмов невозможно представить ни их терапию, ни хирургию. В информационных технологиях абсолютно аналогичным образом разработка и эксплуатация программного обеспечения будут успешны только на основе понимания внутреннего устройства операционной системы.

Рисунок 1.1 изображает эдакий «рентгеновский снимок» внутренностей операционной системы Linux, подробная анатомия которой шаг за шагом и раскрывается в последующих главах.

Глава 2

Пользовательское окружение ОС Linux

2.1. Командный интерфейс

Основным интерфейсом взаимодействия между ЭВМ и человеком в классической операционной системе W:[UNIX] был единственно возможный, диктуемый аппаратными устройствами ее времени[1] *командный интерфейс*. Называемый сегодня интерфейсом командной строки (Command Line Interface, W:[CLI]), он в неизменном виде сохранил все свои элементы — понятие *терминала*, двусторонний попеременный диалог при помощи клавиши ↵ENTER, управляющие символы и клавишу CTRL для их набора.

Терминал является оконечным (*англ. terminal*) оборудованием, предназначенным для организации человеко-машинного интерфейса. Обычно он состоит из устройств вывода — принтера или дисплея, и устройств ввода — клавиатуры, манипулятора «мышь» и пр.

Алфавитно-цифровой терминал позволяет вводить и выводить символы из некоторого заданного набора (например, семибитной кодировки ascii(7)) или другого набора символов charsets(7), состоящего из букв алфавита, цифр, знаков препинания, некоторых других значков, и символов специального назначения для управления самим терминалом — *управляющих символов*.

Ранние, *печатающие терминалы*, представляли собой телетайпы W:[телетайп] (телепринтеры W:[teleprinter]), которые печатали символы из фиксированного набора на ленте или рулоне бумаги — слева направо, сверху вниз. Управляющие символы использовались для управления перемещением печатающей головки справа налево (символ BS), возврата головки к началу строки (символ CR), прокрутки рулона бумаги (символ LF) и пр.

Дисплейный терминал (видеотерминал на основе электронно-лучевой трубки) в упрощенном своем режиме эмулирует поведение печатающего терминала: пово-

---

[1] ЭВМ W:[PDP-11] и терминалы W:[Teletype Model ASR-33].

рачивающийся рулон бумаги — при помощи *скроллинга* изображаемых строк снизу вверх, а перемещающуюся вдоль строки печатающую головку — при помощи *курсора*. В расширенном режиме видеотерминал используется как матрица символов, например в 24 строки по 80 символов в строке, и позволяет выводить символы в произвольное место матрицы, задавать символам стиль изображения, как то: мерцание, жирность, инвертирование, подчеркивание и цвет, — и даже менять шрифты символов терминала. Для управления курсором, его позиционирования, смены стиля изображения символов и прочих возможностей видеотерминала применяются управляющие символы *(см. разд. 2.3)* и управляющие последовательности *(см. разд. 2.4)*.

Двусторонний попеременный диалог (рис. 2.1) командного интерфейса между пользователем и операционной системой представляет собой процесс ввода команд ❶ пользователем посредством клавиатуры и получения результата их выполнения ❷ на бумаге или дисплее алфавитно-цифрового терминала.

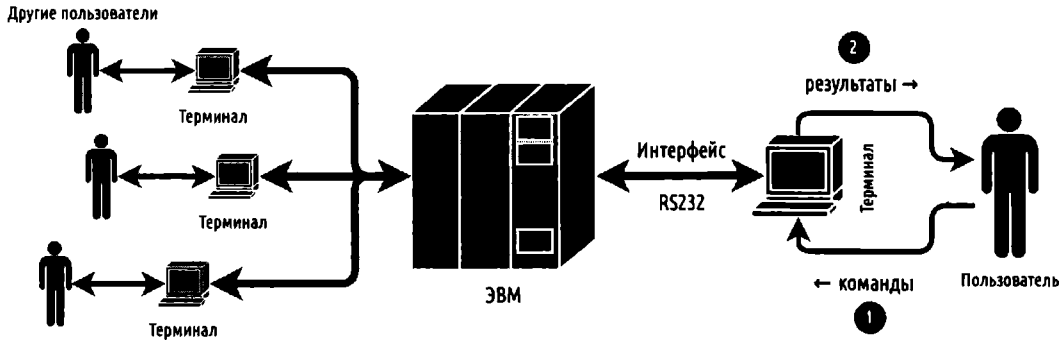


Рис. 2.1. Терминалы и командный интерфейс

В начале сеанса работы в многопользовательской среде операционной системы пользователь должен произвести регистрацию (**logging in**) себя в системе (обычно говорят «произвести вход» в систему) при помощи предъявления имени своей учетной записи (login) и соответствующего ему пароля (password, буквально — пропускное **pass** слово **word**) (рис. 2.2).

Процедура регистрации начинается с заставки операционной системы ❶ и приглашения к вводу имени учетной записи ❷, в ответ на которое пользователь вводит имя своей учетной записи ①. Затем, в ответ на приглашение к вводу пароля ❸, пользователь вводит пароль ②, и при этом на алфавитно-цифровых терминалах никакие символы не изображаются. При положительном исходе регистрации пользователь получает сообщение ❹ о последней (**last**) успешной регистрации, сообщение дня и приглашение командного интерпретатора ❺.

Передача управления от пользователя к операционной системе на каждом шаге диалога происходит при помощи нажатия клавиши ↵ ENTER, а передача управления

в обратную сторону — при помощи *приглашений* к вводу регистрационного имени, пароля, командного интерпретатора и пр. Приглашение командного интерпретатора исторически состояло из символа **$** или символа **%**, а при регистрации под учетной записью администратора — из символа **#**. Позднее приглашение развилось в **finn@ubuntu:~$** и состоит теперь из имени зарегистрировавшегося пользователя **finn**, собственного имени компьютера **ubuntu**, условного имени домашнего каталога пользователя, обозначенного символом **~**, и «классического» символа приглашения **$**.

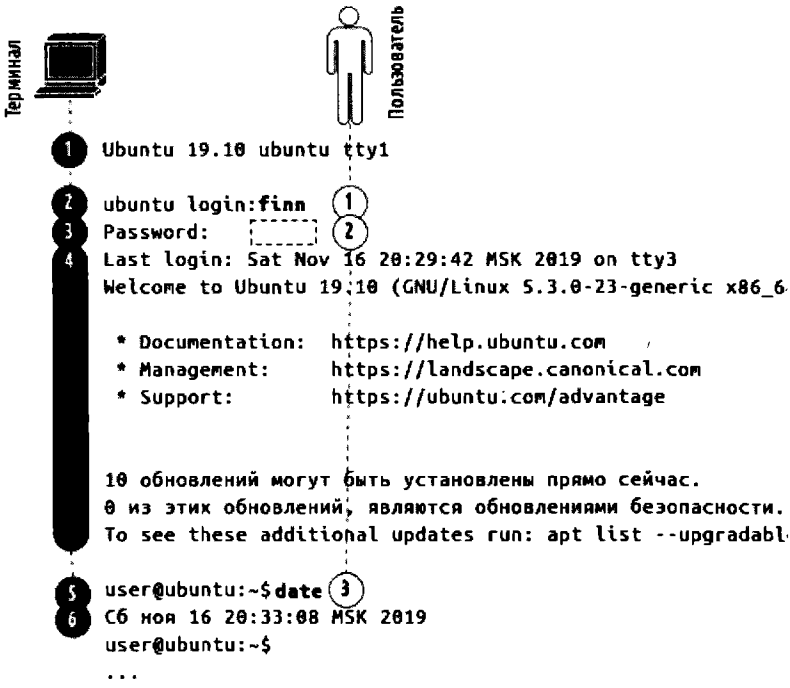


Рис. 2.2. Регистрация пользователя в системе

Сеанс командного интерфейса пользователя продолжается двусторонним попеременным диалогом с командным интерпретатором, где пользователь вводит команды ❸ и получает результаты их выполнения ❻.

2.2. Виртуальные терминалы

На текущий момент времени многопользовательские системы с настоящими физическими терминалами, подключенными посредством интерфейса RS232 и его драйвера ttyS(4) к *большой* ЭВМ, — экзотическая редкость. На *персональных* ЭВМ для взаимодействия с пользователем используются стандартные клавиатура, видеоадаптер и монитор, формирующие так называемую **W:[консоль]**, которая используется драйвером *виртуальных терминалов* для эмуляции нескольких физических терминалов (рис. 2.3).

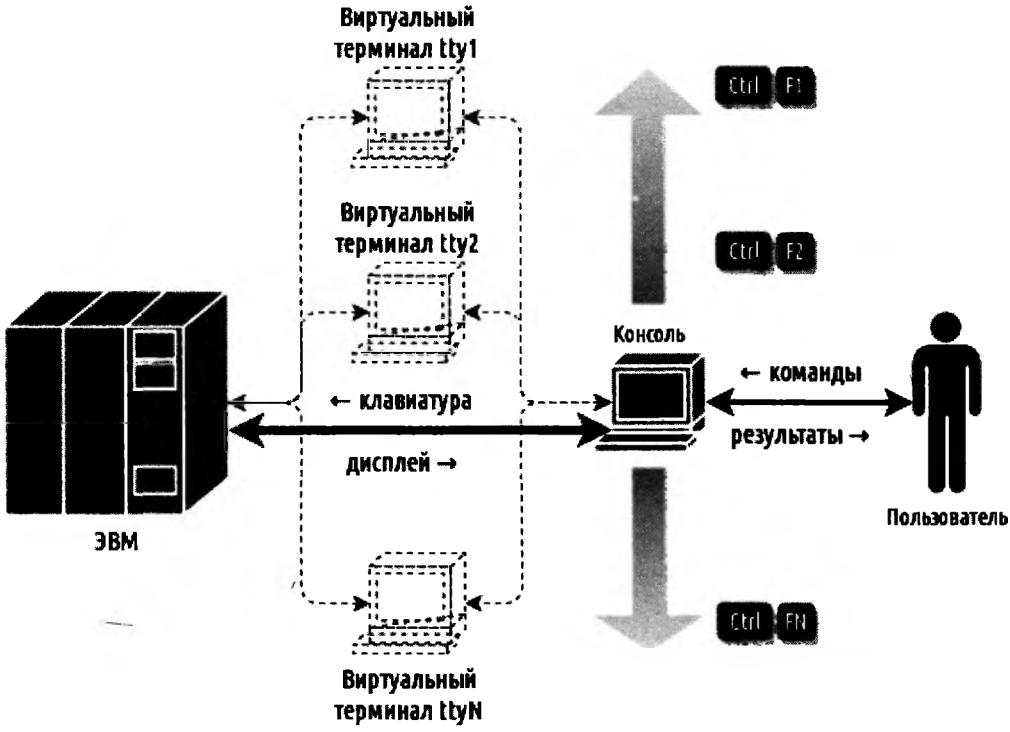


Рис. 2.3. Виртуальные терминалы

Узнать имя текущего терминала (а точнее — имя специального файла устройства[1] терминального драйвера, см. листинг 2.1), на котором выполнен вход в систему, позволяет команда **tty(1)**, а список всех терминальных входов пользователей — команды **users(1)**, **who(1)** и **w(1)**.

Листинг 2.1. Утилиты tty, users, who и w

```
finn@ubuntu:~$ tty ↵
/dev/tty1
finn@ubuntu:~$ users ↵
bubblegum finn iceking jake jake marceline
finn@ubuntu:~$ who ↵
iceking   pts/0        2019-11-16 10:46 (176.10.35.129)
bubblegum tty5         2019-11-16 10:46
marceline tty3         2019-11-16 10:47
finn      tty1         2019-11-16 10:46
```

[1] Подробнее о специальных файлах устройств *см. в разд. 3.2.5.*

```
   jake      :0           2019-11-16 10:47 (:0)
   jake      pts/4        2019-11-16 22:09 (:0)
   finn@ubuntu:~$ w↵
    21:08:15 up  3:54,  6 users,  load average: 0,02, 0,05, 0,01
   USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
❸  iceking  pts/0    176.10.35.129    10:46    1:20   0.05s  0.00s -bash
   bubblegu tty5     -                10:46    1:04   0.04s  0.00s -bash
   marcelin tty3     -                10:47   32.00s  0.06s  0.00s -bash
❶  finn     tty1     -                10:46    1:04   0.50s  0.44s w
   jake     :0       :0.0             10:47   ?xdm?   0.21s  0.00s /usr/lib/gdm3/gdm-x-s...
❷  jake     pts/4    :0               22:09   15.00s  0.01s  0.01s bash
```

Драйвер виртуального терминала позволяет переключаться между эмулируемыми терминалами при помощи сочетания клавиш Ctrl F1, ..., Ctrl F12 (первые двенадцать терминалов из 63 возможных), Alt ← для переключения на предыдущий, Alt → для переключения на следующий виртуальный терминал. При переключении из *графического* виртуального терминала на другой виртуальный терминал необходимо добавлять к сочетанию еще и клавишу Ctrl, т. к. сочетания с клавишей Alt востребованы самим графическим интерфейсом, например Alt F4 закрывает активное окно. Таким образом, для переключения из графического на третий виртуальный терминал используется сочетание Alt Ctrl F3. Также драйвер виртуальных терминалов позволяет листать буфер вывода виртуального терминала при помощи сочетаний Ctrl PgUp и Ctrl PgDn (к сожалению, после переключения терминалов буфер пропадает).

Как и любым другим, драйвером виртуальных терминалов можно управлять при помощи специально предназначенных команд, например, программа **chvt(1)** позволяет переключаться на заданный терминал по его номеру, а команда программа **setfont(1)** — загружать шрифты, формирующие начертания алфавитно-цифровых знаков (см. листинг 2.12).

2.2.1. Псевдотерминалы

При работе в оконной системе X Window System *(см. главу 7)* используются графические терминалы, тогда как для командного интерфейса требуется алфавитно-цифровой терминал. В этом случае (рис. 2.4) он эмулируется при помощи драйвера *псевдотерминала* **pty(4)** (**p**seudo **t**t**y**) и приложения-посредника — эмулятора терминала (например, **xterm** или **gnome-terminal**), который связывает действительный обмен ① в графическом окне с мультиплексором псевдотерминалов ② **ptmx(4)** (**p**seudoterminal **m**ultiplexer), а тот, в свою очередь, присоединен драйвером к подчиненному псевдотерминалу **pts(4)** (**p**seudoterminal **s**lave) командного интерфейса.

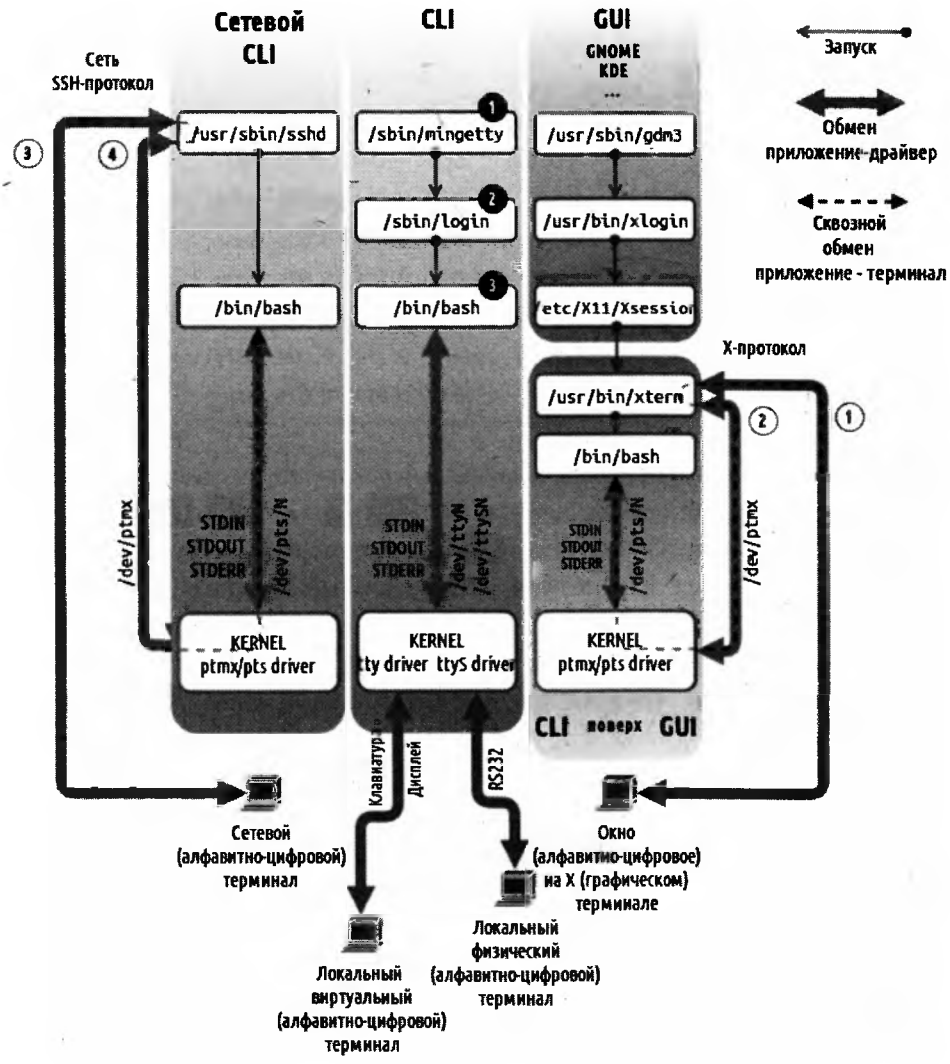


Рис. 2.4. Псевдотерминалы

Аналогично, при подключении удаленного алфавитно-цифрового терминала посредством протоколов удаленного доступ (например, SSH) приложение-посредник (например, демон сетевой службы **sshd**, *см. разд. 6.4.1*) связывает действительный обмен ③ в сетевом соединении с мультиплексором псевдотерминалов ④.

Таким образом, командный интерпретатор и запускаемые им программы работают с воображаемым псевдотерминалом так, как будто окно графического приложения или средства удаленного доступа являются настоящим физическим дисплеем и настоящей физической клавиатурой настоящего терминала. В примере из листинга 2.1 пользователь **finn** зарегистрирован в системе ❶ на первом виртуальном терминале **tty1**, пользователь **jake** зарегистрирован ❷ на псевдотерминале **pts/4**

в эмуляторе терминала оконной системы, а пользователь **iceking** зарегистрирован ❸ при помощи удаленного доступа на псевдотерминале **pts/0**.

Нужно заметить, что на этапе входа пользователя в систему посредством алфавитно-цифрового терминала (см. средний фрагмент под заголовком **CLI** на рис. 2.4) последовательно запускаются: обработчик терминалов ❶, обработчик аутентификации и авторизации пользователей ❷, а затем командный интерпретатор ❸, например **bash(1)**. Именно **getty(1)** предъявляет пользователю (см. рис. 2.2) заставку операционной системы и приглашение к вводу имени пользователя, а **login(1)** — приглашение к вводу пароля, сообщение о последнем успешном входе и сообщение дня.

Аналогичные процессы происходят при любом входе пользователя в систему, например через псевдотерминалы «графического» или «сетевого» доступа (см. левый и правый фрагменты на рис. 2.4). В любом случае после аутентификации и авторизации основной программой (и первой в сеансе пользователя), интерпретирующей вводимые пользователем команды, является командный интерпретатор.

2.3. Управляющие символы

При *вводе* с терминала управляющие символы служат командами драйверу терминала и в большинстве своем генерируются при помощи сочетания клавши Ctrl (отсюда ее название control — управление) с одной из алфавитно-цифровых клавиш. В отдельных случаях управляющие символы генерируются специально предназначенными для этого клавишами, например ↵ Enter, ↹ Tab или ⇤ Backspace.

Таблица 2.1. Управляющие символы терминала

Управляющий символ			*Клавиши*	*Код символа*	
Нотация	*Стандартное действие драйвера*				
	ввод символа	*вывод символа*			
^C	intr	—	Ctrl C	0x03	ETX
^\	quit	—	Ctrl \ или Ctrl 4	0x1C	FS
^Z	susp	—	Ctrl Z	0x1A	SUB
^D	eof	—	Ctrl D	0x04	EOT
^?	erase	—	⇤ Backspace или Ctrl ? или Ctrl 8	0x7F	DEL
^H или \b	—	backspace	Ctrl H	0x08	BS
^W	werase	—	Ctrl W	0x17	ETB

Таблица 2.1. *(окончание)*

Управляющий символ			*Клавиши*	*Код символа*	
Нотация	*Стандартное действие драйвера*				
	ввод символа	*вывод символа*			
^U	kill	—	Ctrl U	0x15	NAK
^I или \t	—	tab	⇆ Tab или Ctrl I	0x09	HT
^M или \r	eol	cr	↵ Enter или Ctrl M	0x0D	CR
^J или \n	eol	nl	Ctrl J	0x0A	LF
^S	stop	—	Ctrl S	0x13	DC3
^Q	start	—	Ctrl Q	0x11	DC1
^R	rprnt	—	Ctrl R	0x12	DC2
^V	lnext	—	Ctrl V	0x16	SYN
^N	—	so	Ctrl N	0x0E	SO
^O	—	si	Ctrl O	0x0F	SI
^[или \e	esc	esc	Esc или Ctrl [или Ctrl 3	0x1B	ESC

Так, например, нажатие клавиши ↵ Enter или эквивалентное сочетание Ctrl J, записывающееся как **^J**, генерирует управляющий символ **LF** (таким же действием обладает символ **CR**, **^M**), который сигнализирует драйверу терминала о завершении ввода строки (eol, end of line) и необходимости «отдать команду на выполнение» (листинг 2.2).

Листинг 2.2. Управляющие символы ^J и ^M

```
finn@ubuntu:~$ date ↵
Вс. февр.  1 22:39:00 MSK 2015
finn@ubuntu:~$ hostname ^M
ubuntu
finn@ubuntu:~$ whoami ^J
finn
```

Нажатие клавиши ← Backspace или сочетания клавиш Ctrl ? приводит к генерации управляющего символа **DEL**, что заставляет драйвер выполнить управляющее дейст-

вие **erase** (**^?**) — удалить последний набранный символ. Аналогично, **werase** (**^W**) и **kill** (**^U**) удаляют последнее набранное слово и всю набранную строку соответственно.

Управляющие символы **intr** (**^C**) и **quit** (**^**) — соответственно штатно и аварийно завершают запущенную ранее и выполняющуюся сейчас программу, а символ **susp** (**^Z**) временно приостанавливает выполняющуюся программу, что проиллюстрировано в листинге 2.3.

Листинг 2.3. Управляющие символы ^C и ^

```
finn@ubuntu:~$ dd if=/dev/dvd of=dvd.iso
^C6227352+0 записей получено
6227351+0 записей отправлено
скопировано 3188403712 байт (3,2 GB), 2,72618 с, 1,2 GB/с

finn@ubuntu:~$ dd if=/dev/cdrom of=cd.iso
^\Выход (сделан дамп памяти)
```

Символы **stop** (**^S**) и **start** (**^Q**) управляют потоком вывода (и, как следствие, скроллингом терминала), что можно использовать для временной приостановки вывода команд с многострочным выводом. Однако случайное нажатие **^S** может привести начинающего пользователя в замешательство — будет казаться, что терминал «завис», т. е. отсутствует реакция со стороны операционной системы на какие-либо нажимаемые клавиши и посылаемые символы, тогда как на самом деле отсутствует (приостановлен) лишь ее вывод — до нажатия **^Q**, **^C** или **^**.

Управляющий символ **eof** (**^D**) используется для оповещения драйвера о завершении ввода при работе с интерактивными (ведущими с пользователем двусторонний попеременный диалог) программами (листинг 2.4).

Листинг 2.4. Управляющий символ ^D

```
finn@ubuntu:~$ mail dketov@gmail.com
Cc:
Subject: Не забыть про Ctrl+D
Символ ^D полезен для mail, at... где еще?
^D
finn@ubuntu:~$ at 21:30
warning: commands will be executed using /bin/sh
at> mplayer ~/sounds/alarm.mp3
at> notify-send -i info 'Хватит работать'
```

```
at> <EOT>
job 1 at Sun Nov 17 21:30:00 2019
finn@ubuntu:~$ lftp ftp.ubuntu.com
lftp ftp.ubuntu.com:~> get /ubuntu/pool/main/m/manpages/manpages_5.03.orig.tar.xz
1677908 байтов перемещено за 1 секунду (1.28 Миб/с)
lftp ftp.ubuntu.com:/> exit
```

Нужно заметить, что при работе с диалоговыми программами **^C** или **^** завершит выполняющуюся программу (**at**), не дав ей выполнить свое основное действие, или вообще будет проигнорирован (**ftp**, **mail**). Именно символ завершения ввода (**eof**, end of file) сообщит драйверу о нежелании больше вести диалог с программой (который в свою очередь сообщит программе об отсутствии для нее вводимых данных).

В очень редких случаях, возможно, потребуется ввести сам управляющий символ, например **^C**, **^** или **^D**, *непосредственно* в выполняющуюся на терминале программу, что невозможно сделать соответствующими клавиатурными комбинациями, потому как управляющие символы будут поглощены драйвером терминала, что приведет к завершению программы, в которую вводятся символы. Для отмены (экранирования) специального назначения управляющих символов в пользу его *непосредственного* (литерального) значения служит управляющий символ (literal next) **lnext** (**^V**), сигнализирующий драйверу терминала об отмене специального назначения следующего за ним символа (листинг 2.5).

Листинг 2.5. Управляющий символ ^V

```
   finn@ubuntu:~$ tee cc.bin
^  Ctrl+C:^C
   finn@ubuntu:~$ od -ca cc.bin
?  0000000
   finn@ubuntu:~$ tee cc.bin
   Ctrl+C:^V^C ^
   Ctrl+C:
   ^
   finn@ubuntu:~$ od -ca cc.bin
   0000000   C   t   r   l   +   C   003 ^ \n
             C   t   r   l   +   C   etx   nl
   finn@ubuntu:~$ hd cc.bin
   00000000  43 74 72 6c 2b 43 3a 03 0a                     |Ctrl+C:..|
   00000009
```

Реакция драйвера терминала на получаемые управляющие символы и предпринимаемые им управляющие действия (а точнее, наоборот — управляющие символы, закрепленные за управляющими действиями) стандартно предопределена, но почти все эти соответствия могут быть просмотрены и изменены командой **stty(1)**, что иллюстрируется в листинге 2.6.

Листинг 2.6. Утилита stty

```
finn@ubuntu:~$ stty -a
speed 38400 baud; rows 38; columns 136; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; eol2 = <undef>;
swtch = <undef>; start = ^Q; stop = ^S; susp = ^Z;
rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 hupcl -cstopb cread -clocal -crtscts
-ignbrk brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff -iuclc -ixany
imaxbel iutf8
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt echoctl echoke
```

Кроме того, команда stty(1) позволяет получить (а также задать) и другие настройки драйвера терминала:

- скорость приемопередатчика последовательного интерфейса терминала (**speed 38400 baud**);
- количество изображаемых терминалом строк и столбцов (**rows 33; columns 119**);
- флаги режимов работы приемопередатчика интерфейса (**-parenb...hupcl -cstopb...-inpck**);
- флаги режимов обработки вводимых из терминала символов (**-istrip...-igncr icrnl...iutf8**);
- флаги режимов обработки выводимых на терминал символов (**opost...-ofdel**) и пр.

Так, например, флаг **icanon** включает или выключает (**-icanon**) «канонический» (**canon**ical) режим обработки вводимых (input) символов, т. е. возможности редактирования вводимой строки при помощи управляющих символов **^?** и **^U**, а также сигнализацию завершения ввода при помощи **^D**.

Флаг **iexten** включает «расширения» канонического режима стандарта POSIX, т. е. удаление последнего введенного слова при помощи **^W**, перерисовку введенной строки при помощи **^R** и ввод литеральных значений управляющих символов при помощи **^V**.

Управляющие символы **^C**, **^** и **^Z** штатного или аварийного завершения и приостановки выполняющейся программы активируются флагом **isig** (листинг 2.7), разрешающим или запрещающим (**-isig**) посылку сигналов (**sig**nal, *см. разд. 4.8*).

Флаг **icrnl** включает трансляцию вводимого символа **CR** в символ **LF**, что позволяет запускать команды клавишей ↵ ENTER (неожиданно, правда?), потому как основной символ для этого действия все же — **LF** (так уж сложилось в UNIX со времен телетайпа ASR-33).

К счастью, большинство из этих параметров и флагов применимы только при работе с настоящими аппаратными терминалами и интерфейсами. Поэтому пояснение их назначения[1] можно опустить хотя бы просто потому, что оно требует дополнительных знаний специфики соответствующей аппаратуры, что не имеет ни особой актуальности, ни является предметом нашего рассмотрения.

Нужно также отметить, что некоторые[2] диалоговые программы «игнорируют» некоторые настройки терминала, например флаг «канонического» режима. Точнее, они всегда работают в «неканоническом» режиме и сами обрабатывают управляющие символы, зачастую переопределяя некоторые из них или добавляя обработку дополнительных, например для командного интерпретатора **bash(1)** это **^R** (reverse-search-history), **^S** (forward-search-history), **^D** (delete-char) **^L** (clear-screen), **^A** (beginning-of-line), **^E** (end-of-line), **^F** (forward-char) **^B** (backward-char), **^P** (previous-history), **^N** (next-history) и др.

Листинг 2.7. Настройки драйвера терминала

```
finn@ubuntu:~$ stty
speed 38400 baud; line = 0;
iutf8
finn@ubuntu:~$ stty -isig
finn@ubuntu:~$ stty
speed 38400 baud; line = 0;
iutf8
-isig
finn@ubuntu:~$ dd if=/dev/dvd of=dvd.iso
!☛ ^C^\^Z
```

[1] Исчерпывающе описанные в **termios(3)**, вместе с управляющими действиями и предопределенными им управляющими символами.

[2] В большинстве своем те, которые используют библиотеку **readline(3)** для расширенного редактирования вводимой строки или библиотеку **ncurses(3)** для работы с расширенным режимом терминала. К ним относятся командный интерпретатор **bash(1)**, постраничный листатель **less(1)**, текстовые редакторы **vi(1)** и **nano(1)**, Web-браузеры **links(1)** и **lynx(1)**, ftp-клиенты **lftp(1)** и **ncftp(1)**, файловый менеджер **mc(1)** и пр.

При *выводе* информации на терминал управляющие символы[1] (в отличие от алфавитно-цифровых символов, подлежащих изображению каким-либо значком) служат для управления терминалом (см. **ascii(7)**, **console_codes(7)** и табл. 2.1). Например, символ **CR** (carriage return) перемещает печатающую головку или курсор терминала в начало строки, символ **LF** (line feed) — в начало новой строки, символ **NL** (nl, new line) — на следующую строку, символ **HT** (horizontal tab) — на несколько символов вправо, а символ **BS** (back space) стирает один символ слева от курсора (или просто перемещает печатающую головку на один символ влево) и т. д.

Так, символ **LF**, используемый в «текстовых» файлах при выводе их на экран, позволяет разделять «логические» строки файла и изображать их на разных «физических» строках терминала, а символы **SO** (shift out) и **SI** (shift in) соответственно включают и выключают альтернативный шрифт терминала, содержащий другие знаки вместо маленьких букв латинского алфавита. Именно поэтому при ошибочном *выводе* на терминал «бинарного» файла, в содержимом которого весьма вероятно встретить символ **SO** ❶, активирующий альтернативный шрифт, возникает ощущение «испорченности» терминала, которую легко починить *выводом* символа **SI**, возвращающего терминал к стандартному шрифту. Для этого достаточно литерально *ввести* символ **SI** ❶, который будет расценен командным интерпретатором как (несуществующая) команда и *выведен* на экран в сообщении об ошибке ❷ (листинг 2.8).

Листинг 2.8. Управляющие символы SO (^N) и SI (^O)

```
finn@ubuntu:~$ cat /etc/localtime
... ❶...
#<#(1#X?
        *B@FP *B@*! 8@8@LMTMMTMSTMDSTMSKMSDMSMEETEEST
MSK↑4
°i┼┼@┤b├┼┤:-$ ^O ←❶
❷ : команда не найдена
finn@ubuntu:~$ tee si-and-so.txt
hahaha^V^N+hahaha^V^O
hahaha→
finn@ubuntu:~$
```

[1] См. W:[управляющие символы].

2.4. Управляющие последовательности

В расширенном режиме видеотерминалов W:[VT52], W:[VT100], W:[VT220] появилась возможность *вывода* символов в произвольное место экрана и использования полужирного, затемненного, негативного, подчеркнутого и других начертаний символов. Возможности *ввода* дополнились функциональными клавишами, клавишами перемещения курсора, дополнительной клавиатуры и пр.

Для этого потребовались дополнительные управляющие символы, которые не поместились в кодировку **ascii(7)**, потому терминалы стали использовать управляющие последовательности символов[1] **console_codes(7)**, предваряемые управляющим символом **ESC** с кодом **0x1B**.

Так, например, последовательность **ESC # 8** вызовет визуальный тест выравнивания краев терминала, заполнением буквой E всех строк и столбцов, **ESC c** сбросит терминал в исходное состояние, **ESC [1 m** включит полужирное начертание, **ESC [2 m** — затемненное начертание, **ESC [4 m** — подчеркивание, а **ESC [0 m** вернет стандартное начертание символов.

Управляющие символы и их последовательности являются обычными байтами и при литеральном вводе ❶ с терминала могут быть сохранены в файл (листинг 2.9), например, при помощи команды **tee(1)**. При последующем выводе ❷ на терминал, например при помощи команды **cat(1)**, будут задействованы соответствующие расширенные возможности. Побайтное содержимое файла можно при этом увидеть ❸ на терминале посредством восьмеричного **od(1)** или шестнадцатеричного дампа **hexdump(1)**, **hd(1)**.

Листинг 2.9. Управляющие последовательности терминала

```
finn@ubuntu:~$ tee esc.txt
❶ ^[[1mbold^[[0m,^[[2mdim^[[0m,^[[4munderscore^[[0m,^[[7mreverse^[[0m
bold.dim.underscore.reverse
^D
❷ finn@ubuntu:~$ cat esc.txt
bold.dim.underscore.reverse
finn@ubuntu:~$ od -c esc.txt
❸ 0000000 033   [   1   m   b   o   l   d 033   [   0   m   , 033   [   2
0000020   m   d   i   m 033   [   0   m   , 033   [   4   m   u   n   d
0000040   e   r   s   c   o   r   e 033   [   0   m   , 033   [   7   m
0000060   r   e   v   e   r   s   e 033   [   0   m  \n
0000074
```

[1] См. W:[управляющие последовательности ANSI].

Многие «дополнительные» клавиши современных терминалов, такие как функциональные F1...F12, клавиши управления курсором ↓ ↑ → ←, скроллингом PgUp PgDn Home End и пр., *генерируют* (листинг 2.10) управляющие последовательности, которые обрабатываются, например, библиотекой **readline(3)** и используются для редактирования командной строки.

Листинг 2.10. Управляющие последовательности клавиатуры

```
finn@ubuntu:~$ od -a
F1 ^[OP
0000000 esc   O   P  nl
0000004
finn@ubuntu:~$ od -a
End ^[OF
0000000 esc   O   F  nl
0000004
```

Несмотря на стандартизацию управляющих последовательностей, разные терминалы все же имеют различия, поэтому в операционной системе появились базы данных с описанием свойств и управляющих последовательностей терминалов **termcap(5)** и **terminfo(5)** (листинг 2.11). Узнать ESC-последовательности можно при помощи команды **infocmp(1)**, а вывести их на терминал — включить соответствующий режим — при помощи команды **tput(1)**.

Листинг 2.11. База данных управляющих последовательностей termcap(5)

```
finn@ubuntu:~$ infocmp
#     Reconstructed via infocmp from file: /lib/terminfo/l/linux
linux|linux console,
      am, bce, ccc, eo, mir, msgr, xenl, xon,
      colors#8, it#8, ncv#18, pairs#64,
                  ...       ...       ...       ...
      op=\E[39;49m, rc=\E8, rev=\E[7m , ri=\EM, rmacs=\E[10m,
                  ...       ...       ...       ...
      sgr0=\E[0;10m , smacs=\E[11m, smam=\E[?7h, smir=\E[4h,
                  ...       ...       ...       ...
      smpch=\E[11m, smso=\E[7m, smul=\E[4m , tbc=\E[3g,
                  ...       ...       ...       ...
finn@ubuntu:~$ tput smul
finn@ubuntu:~$ tput rev
finn@ubuntu:~$ tput sgr0
finn@ubuntu:~$ tput smul | od -ac
0000000 esc   [   4   m
       033   [   4   m
0000004
```

Примером простейшей программы, использующей управляющие последовательности для форматирования символов при выводе на экран, является утилита просмотра справочных страниц **man(1)**. В качестве более изощренных программам, использующих управление расширенными возможностями терминала, можно привести **less(1)**, **nano(1)**, **mc(1)**, многие из которых используют для этого библиотеку **ncruses(3)**. А самым экстремальным примером использования управляющих последовательностей терминалов является W:[ASCII-графика] и W:[ANSI-графика], реализующаяся библиотеками **aalib** и **caca**, при помощи которых на алфавитно-цифровом (!) терминале можно просматривать видеофильмы (листинг 2.12), например, при помощи видеоплееров **mplayer(1)** или **mpv(1)**, поддерживающего эти библиотеки.

Листинг 2.12. Просмотр видео на алфавитно-цифровом терминале

```
finn@ubuntu:~$ setfont Uni1-VGA8
finn@ubuntu:~$ mpv --quiet --vo=caca https://www.youtube.com/watch?v=Zo7_OOW3GzA
finn@ubuntu:~$ youtube-dl --exec 'mplayer -quiet -vo aa:dim:bold:reverse'
https://www.youtube.com/watch?v=Zo7_OOW3GzA
```

2.5. Основной синтаксис командной строки

Основу интерфейса командной строки UNIX составляет командный интерпретатор (КИ), являющийся первой и главной программой, запускаемой в сеансе пользователя. Двусторонний попеременный диалог с командным интерпретатором начинается с приглашения ❶, в ответ на которое пользователь вводит команду, отправляя ее на выполнение управляющим символом **LF 0x0A**, получает результат ее исполнения на терминале и новое приглашение, сигнализирующее о готовности КИ к исполнению очередной команды (рис. 2.5). Многие другие *диалоговые* программы, например **lftp**, так же будут придерживаться синтаксиса и соглашений, принятых в языке командного интерпретатора.

Базовый синтаксис (подчиняющийся второй части стандарта W:[POSIX]) языка любого[1] командного интерпретатора на самом деле достаточно прост и напоминает

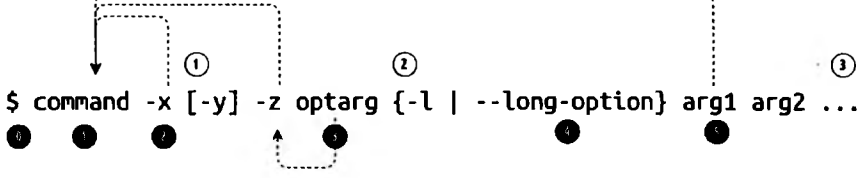


Рис. 2.5. Основной синтаксис командной строки

[1] Диалектов языка командного интерпретатора достаточно много: Bourne shell **sh**, Korn shell **ksh**, C shell **csh** и т. д.

язык, близкий к естественному. Например, **rm -f -R Изображения Музыка** переводится на человеческий как «удалить (**rm**) без шума и пыли (**-f**) и со всеми потрохами (**-R**) каталоги **Изображения** и **Музыка**». Принято говорить, что команда состоит (см. рис. 2.5) из лексем (лексических элементов), разделенных пробельными символами — пробелами **SP 0x20** и табуляциями **HT 0x09** — в любом количестве и сочетании. Первая лексема ❶ — это название команды, за которой следуют ее параметры: сначала опции (они же — ключи, они же — модификаторы) без аргументов ❷ и ❹ или с аргументами ❸, а в конце — аргументы ❺ самой команды. Название команды — это глагол, указывающий, ЧТО делать; опции — это наречия и прочие части речи, объясняющие, КАК это делать; и наконец, аргументы задают то, с ЧЕМ это делать. В разнообразной документации при описании синтаксиса команды, ее опций и аргументов принято использовать квадратные скобки ① для указания необязательности опции или аргумента, фигурные скобки ② и вертикальную черту для указания выбора из вариантов и многоточие ③ для указания повторяемости.

Получив команду, интерпретатор определяет, является ли она *псевдонимом*, *встроенной* в интерпретатор, или реализуется *внешней* программой, подлежащей запуску (листинг 2.13).

Листинг 2.13. Утилиты which и type

```
finn@ubuntu:~$ which date
/usr/bin/date
finn@ubuntu:~$ type date
date является /usr/bin/date
finn@ubuntu:~$ type -a ls
ls – это псевдонимом для «ls --color=auto»
ls является /usr/bin/ls
ls является /bin/ls
finn@ubuntu:~$ type -a pwd
pwd – это встроенная команда bash
pwd является /usr/bin/pwd
pwd является /bin/pwd
finn@ubuntu:~$ type which
для which вычислен хэш (/usr/bin/which)
finn@ubuntu:~$ type type
type – это встроенная команда bash
```

При наличии нескольких вариантов команды наивысший приоритет имеют псевдонимы, наименьший — внешние команды. Подстановку псевдонимов можно увидеть, включив трассировку выполнения команд интерпретатора при помощи команды **set** (листинг 2.14).

Листинг 2.14. Подстановка псевдонимов

```
finn@ubuntu:~$ type ls
ls – это псевдонимом для «ls --color=auto»
finn@ubuntu:~$ set -x
finn@ubuntu:~$ ls -ASs
☛ + ls --color=auto -ASs
итого 36
12 examples.desktop   4 .bashrc         4 .profile      4 .lesshst
 4 .cache             4 .bash_history   4 .bash_logout
```

2.5.1. Опции командной строки

В истории развития операционной системы UNIX программы использовали разные способы задания своих опций:

- односимвольные, например **ls -a -l** (что эквивалентно **ls -l -a** или **ls -al** или **ls -la**);
- многосимвольные, например **find /var -xdev**;
- длинные, например **ps --help**;
- с аргументами, например **kill -n 15 1**, или **kill -n15 1**, или **du -B M**, или **du -BM**, или **find /etc -type d**, или даже **ls --sort=size**;
- «нестандартные», например **set +x**, **tar czf tar.tgz ~** или **dd if=/dev/dvd of=dvd.iso**.

Знак «минус», предваряющий опции, естественно, используется для того, чтобы отличать[1] их от аргументов. Среди прочих он был выбран потому, что редко встречается как первый символ в *аргументах* команд (в качестве которых зачастую выступают имена файлов), и еще потому, что на терминале классической[2] UNIX набор более логичного знака «плюс» (что могло бы означать «включить», «активировать») требовал достаточных усилий по нажатию клавиши ⇧ Shift. В результате получилось, что, например, в команде **set** опция **x** (execution trace) в форме **-x включает**, а **+x выключает** трассировку выполнения команд.

В тех редких случаях, когда аргумент команды все же начинается с символа «минус» и тем самым похож на опцию (представим, что нужно выполнить действие над файлом с именем **-=filename=-**), специальная опция **--** сигнализирует о конце списка опций (листинг 2.15), за которым следуют лексемы, обязанные расцениваться как *аргументы* вне зависимости от их написания.

---

[1] В примере с командой **rm** лексема **-R** означает *опцию*, требующую выполнить рекурсию (*англ.* recursion), но не *аргумент* — объект, подлежащий удалению.

[2] Очередной привет из прошлого от Teletype ASR-33.

Листинг 2.15. Конец списка опций

```
finn@ubuntu:~$ stat -=filename=-
stat: неверный ключ — «=»
По команде «stat --help» можно получить дополнительную информацию.
finn@ubuntu:~$ stat -- -=filename=-
  Файл: -=filename=-
  Размер: 0             Блоков: 0          Блок В/В: 4096   пустой обычный файл
Устройство: fc00h/64512d        Inode: 26870044    Ссылки: 1
Доступ: (0600/-rw-------)  Uid: ( 1000/    finn)   Gid: ( 1000/    finn)
Доступ: 2019-11-17 10:43:36.520984570 +0300
Модифицирован: 2019-11-17 10:43:36.520984570 +0300
Изменён: 2019-11-17 10:43:36.520984570 +0300
 Создан: -
```

Короткие, односимвольные опции (например, **-l -a**) без своих аргументов издревле можно было объединять в группы (**-la** или **-al**), однако их в этом случае сложно отличать от многосимвольных (**-xdev**) или односимвольных, склеенных со своими аргументами (**-BM**). Поэтому позже появились длинные (в так называемом, GNU-стиле) опции, обозначаемые двумя знаками «минус», позволяющие навести некоторый порядок в виде **--block-size=M** вместо **-BM** или, предположим, **--dont-descent** вместо **-xdev**.

2.6. Справочные системы

Используемые в Linux электронные справочные системы (online help) являются логичным следствием как его родства с семейством операционных систем UNIX — страницы руководства **man(1)** (**man**ual pages), так и принадлежностью к свободному программному обеспечению под эгидой движения GNU — справочная система **info(1)**. Следует отметить, что понятие online в контексте справочных систем вовсе не означает их доступность через Интернет, как это часто, но ошибочно воспринимается сегодня. В рассматриваемом контексте online означает немедленную доступность справочной информации непосредственно из программного обеспечения по сравнению со справочной информацией, доступной в печатной, offline, форме.

2.6.1. Система страниц руководства

Самой известной справочной системой, сопровождающей UNIX практически с момента ее рождения, является справочная система страниц руководства, информация из которой доступна при помощи команд **man(1)**, **apropos(1)** и **whatis(1)**. Справочная

система **man-pages(7)** состоит из отдельных страниц, посвященных отдельным командам, специальным файлам устройств, конфигурационным файлам, системным и библиотечным вызовам и другим понятиям, которые сгруппированы по восьми (обычно, но есть исключения из правил) секциям. Каждая секция имеет заголовочную страницу **intro**, описывающую назначение самой секции (листинг 2.16).

Листинг 2.16. Секции справочной системы man(1)

```
finn@ubuntu:~$ whatis intro
intro (8)            - introduction to administration and privileged commands
intro (7)            - introduction to overview and miscellany section
intro (3)            - introduction to library functions
intro (4)            - introduction to special files
intro (1)            - introduction to user commands
intro (5)            - introduction to file formats and filesystems
intro (6)            - introduction to games
intro (2)            - introduction to system calls
finn@ubuntu:~$ whatis whatis
whatis (1)           - показывает однострочные описания справочных страниц
finn@ubuntu:~$ whatis apropos
apropos (1)          - поиск в именах справочных страниц и кратких описаниях
finn@ubuntu:~$ whatis man
man (1)              - доступ к справочным страницам
man (7)              - macros to format man pages
```

Естественным образом справочная система описывает сама себя, поэтому известнейшей «мантрой» в операционной системе является **man man**, т. е. запрос страницы руководства, посвященной самой команде **man(1)**. Сами страницы руководства написаны на языке разметки текста **roff**[1] и размещаются в сжатых gz-файлах «секционных» подкаталогов **man1**, **man2**, ..., **man8** каталога **/usr/share/man** (листинг 2.17). Страницы руководства частично поставляются переведенными на различные национальные языки, отличные от английского.

Листинг 2.17. Формат страниц справочной системы man

```
finn@ubuntu:~$ man -w man
/usr/share/man/ru/man1/man.1.gz
finn@ubuntu:~$ file /usr/share/man/ru/man1/man.1.gz
```

[1] Система подготовки текстов, доставшаяся в наследство от классической UNIX.

```
/usr/share/man/ru/man1/man.1.gz: gzip compressed data, max compression, from Unix,
                                                    original size modulo 2^32 65164
finn@ubuntu:~$ file -z /usr/share/man/ru/man1/man.1.gz
/usr/share/man/ru/man1/man.1.gz: troff or preprocessor input, UTF-8 Unicode text
                                    (gzip compressed data, max compression, from Unix)
finn@ubuntu:~$ whatis file
file (1)             - determine file type
```

Команда **man(1)**, таким образом, ответственна за поиск указанной пользователем страницы, распаковку ее сжатого файла при помощи распаковщика **gzip(1)**, форматирования при помощи процессора **troff(1)** и (по умолчанию) вывод результата на терминал при помощи постраничного «листателя» **less(1)**. Именно процессор **troff** умеет посредством управляющих последовательностей нужного терминала раскрашивать вывод страниц руководства правильным образом, а «листатель» **less** прокручивать подготовленную справку вперед и назад.

Использование языка разметки позволяет форматировать страницу одинаково удобно для просмотра на разных терминалах с различным количеством столбцов, которые учитывает **troff**, и разным количеством строк, учитываемым **less**. Так, например, результат одинаково хорош и на псевдотерминале в окне эмулятора терминала **xterm** или **gnome-terminal** развернутого в любой размер, и на виртуальном терминале консоли с загруженным шрифтом любого размера. Более того, использование универсального языка разметки и соответствующий «драйвер» **troff** позволяет преобразовывать страницы руководства в самые разные виды. Например, в «принтерный» PostScript или PCL, пригодный для печати на принтере с высокой разрешающей способностью, или в HTML[1] для просмотра в html-браузере (листинг 2.18).

Листинг 2.18. Форматирование справочных страниц руководства для печати и для html-браузера

```
finn@ubuntu:~$ man -t man > man.print.1
finn@ubuntu:~$ file man.print.1
man.print.1: PostScript document text conforming DSC level 3.0
finn@ubuntu:~$ man -Tlj4 man > man.print.2
finn@ubuntu:~$ file man.print.2
man.print.2: HP PCL printer data
finn@ubuntu:~$ man -Thtml man > man.html
finn@ubuntu:~$ file man.html
man.html: HTML document, ASCII text, with very long lines
```

[1] При наличии установленного пакета **groff**.

При просмотре страниц руководства на терминале навигация по изображаемой справочной странице производится так, как предусмотрено используемым «листателем», которым по умолчанию в большинстве случаев будет **less(1)**, приемник классического **more(1)**. Одним из самых полезных навигационных действий в справочных системах (табл. 2.2) является поиск регулярных выражений[1]. Собственно, страницы руководства разбиты на разделы, заголовки которых размещаются в начале строк и записываются в верхнем регистре. Например, **SYNOPSIS** — краткий обзор, **EXAMPLES** — примеры, **FILES** — используемые программой (конфигурационные) файлы, **ENVIRONMENT** — переменные окружения и пр. Поэтому для перемещения к разделу **TOPIC** очень удобно использовать символ поиска **/** и выражение **^TOPIC**, что буквально означает: найти в начале строки — **^** слово — **TOPIC**.

Таблица 2.2. Клавиши навигации «листателей» страниц

Навигационное действие	*Управляющий символ или клавиша less*	*Управляющий символ или клавиша more*	*Управляющий символ или клавиша info*
Выход	Q или q	Q или q или ^C	Q или q или ^C
Справка содержимое этой таблицы	h или H	h или ?	h или ? или ^H
Вниз одну строку	CR или j или ↓	CR	↓ или ^N
Вверх на одну строку	^P или k или ↑		↑ или ^P
Вниз один экран	f или SPACE или PGDN	f или SPACE	SPACE или PGDN
Вверх один экран	b или PGUP	b	DEL или PGUP
Поиск регулярных выражений вперед	/	/	/ или s
Поиск назад	?		
Повторить поиск	n	n	}
Повторить поиск в обратном направлении	N		{
В конец страницы (ноды)	G или > или END		e или END или ESC >
В начало страницы (ноды)	g или < или Home		b или Home или ESC <
Следующая нода	Неприменимо	Неприменимо	]
Предыдущая нода	Неприменимо	Неприменимо	[

[1] Подробнее о регулярных выражениях *см. разд. 5.8*.

2.6.2. Справочная система GNU

Еще одной системой документации является система W:[GNU texinfo]. В отличие от справочника man, выступающего, по сути, *кратким* руководством по командам, их опциям и аргументам, справочник info представляет собой *развернутое* руководство с примерами и объяснениями.

Справочная система состоит из предварительно подготовленных (гипер-)текстовых страниц, размещенных в сжатых файлах каталога **/usr/share/info**, оглавлением которым служит так называемый «каталог» документации. Каждая страница структурирована при помощи иерархически упорядоченных, так называемых «нод», аналогов книжных разделов/подразделов/глав/секций.

Язык разметки texinfo, как и язык roff в системе страниц руководства, позволяет подготавливать разные представления справочной информации из единого источника при помощи специальных[1] инструментов, но в отличие от страниц руководства man только при наличии исходных файлов документации.

2.6.3. Встроенная справка командного интерпретатора

Как было указано ранее, команды интерпретатору могут приводить к вызову *внешних* программ операционной системы или исполняться непосредственно интерпретатором, будучи *встроенными* в него. Внешние программы зачастую документируются отдельными страницами руководства man или отдельными справочными страницами info, тогда как встроенные команды являются частью интерпретатора и естественным образом описываются все вместе на справочной странице, посвященной интерпретатору (листинг 2.19).

Листинг 2.19. Справка по встроенным командам интерпретатора

```
finn@ubuntu:~$ type cd
cd — это встроенная команда bash
finn@ubuntu:~$ man cd
Нет справочной страницы для cd
finn@ubuntu:~$ whatis cd
cd: ничего подходящего не найдено.
finn@ubuntu:~$ man bash
                ...             ...             ...             ...
SHELL BUILTIN COMMANDS
       Unless otherwise noted, each builtin command documented in this section
       as accepting options preceded by - accepts -- to signify the end of the
```

[1] При наличии установленного пакета texinfo и исходных файлов документации `.texi`.

```
        options.  The  :, true, false, and test builtins do not accept options
        and do not treat -- specially.  The exit, logout, break, continue, let,
                    ...          ...          ...          ...
• cd [-L|[-P [-e]] [-@]] [dir]
              Change the current directory to dir.  if dir is not supplied, the
              value of the HOME shell variable is the default. Any additional ar
                    ...          ...          ...          ...
```

Однако обращаться каждый раз к весьма внушительной справке по командному интерпретатору не совсем удобно, поэтому встроенная в командный интерпретатор команда **help** позволяет получить краткую справку по встроенным командам интерпретатора (листинг 2.20).

Листинг 2.20. Встроенная справка командного интерпретатора

```
finn@ubuntu:~$ help -d help
help - Display information about builtin commands.
finn@ubuntu:~$ help -d cd
cd - Change the shell working directory.
finn@ubuntu:~$ help cd
cd: cd [-L|[-P [-e]] [-@]] [каталог]
    Change the shell working directory.

    Change the current directory to DIR.  The default DIR is the value of the
    HOME shell variable.
                    ...          ...          ...          ...
```

2.7. Пользователи и группы

Как было указано ранее, для начала работы в многопользовательской операционной системе пользователю необходимо «зарегистрироваться», предъявляя имя своей пользовательской учетной записи и пароль, подтверждающий право на ее использование. В результате регистрации в системе запускается командный интерпретатор — первая программа пользовательского сеанса.

Учетные записи (УЗ) служат для *авторизации*, т. е. для разграничения прав доступа субъектов (процессов пользователей или процессов системных служб) к объектам (файлам, другим процессам, системным вызовам пр.).

Различают пользовательские и групповые учетные записи, при этом каждая пользовательская учетная запись *идентифицируется* уникальным числовым «пользовательским идентификатором» — UID (**U**ser **Id**entifier), а каждая групповая — таким же уникальным числовым «групповым идентификатором» GID (**G**roup **Id**entifier). Именно эти числовые идентификаторы и используются ядром операционной систе-

мы при определении и проверке прав доступа субъектов относительно объектов (листинг 2.21).

Учетные записи пользователей используются для *аутентификации* (проверки подлинности) их при регистрации в системе по *имени* и *паролю*, а учетные записи групп — для классификации пользовательских УЗ (по функциям, задачам, ролям, проектам или другими способами) и последующей раздачи прав доступа этим «классам» пользователей.

Листинг 2.21. Пользовательские идентификаторы UID и GID первичной и дополнительных групп

```
finn@ubuntu:~$ id
uid=1000(finn) gid=1000(finn) группы=1000(finn),4(adm),..., 130(lxd),131(sambashare)
```

Каждая учетная запись пользователя обязательно связана с одной групповой учетной записью, так называемой «первичной» группы пользователя. Исторически сложилось, что в ранней UNIX членство пользователей в группах определялось динамически, т. е. после регистрации пользователя в системе ему (точнее — его процессу, *см. разд. 4.5.1*) выдавался идентификатор (и, как следствие, права доступа) только одной группы. Для выполнения действий в другой роли (получения других групповых идентификаторов) нужно было «заново зарегистрироваться в группе» при помощи команды **newgrp**, предъявляя имя и пароль (!) группы. Позднее (и до сих пор) членство в группе стало статическим, т. е. при регистрации в системе пользователю выдают идентификаторы всех групп, в которых он состоит, при этом первая группа носит название первичной (primary), а остальные — дополнительных (supplementary).

Кроме этих основных свойств, каждая учетная запись характеризуется именем *домашнего каталога* и именем *командного интерпретатора* (запускаемого при регистрации в системе). Дополнительно, учетная запись пользователя может содержать полное имя пользователя, рабочий и .домашний телефоны, рабочий адрес и прочую информацию, которую можно посмотреть при помощи **pinky(1)** и **finger(1)** (листинг 2.22), а изменить при помощи **chfn(1)**.

Листинг 2.22. Свойства учетной записи пользователя

```
finn@ubuntu:~$ finger dvk
Login: dvk                              Name: Dmitry V. Ketov
Directory: /home/dvk                    Shell: /bin/bash
Office Phone: +7(812)703-02-02          Home Phone: ---
On since Sun Nov 17 01:51 (MSK) on :0 from :0 (messages off)
On since Sun Nov 17 10:38 (MSK) on pts/1 from 10.0.2.2
   5 seconds idle
```

```
On since Sat Nov 16 20:59 (MSK) on tty3    34 minutes 19 seconds idle
    (messages off)
On since Sat Nov 16 21:12 (MSK) on tty4    10 hours 3 minutes idle
    (messages off)
New mail received Sun Nov 17 11:14 2019 (MSK)
    Unread since Sun Nov 16 22:31 2019 (MSK)
No Plan.
```

Учетная запись системного администратора с привилегированными (а точнее — неограниченными в буквальном смысле) правами доступа обычно называется **root** и всегда имеет идентификатор **UID=0**. Учетные записи, «от лица которых» выполняются процессы системных служб, называются псевдопользовательскими и идентифицируются в диапазоне **UID=1—499** или **UID=1—999**, а начиная с **UID=500** (redhat) или **UID=1000** (debian) и далее идентифицируются учетные записи обычных пользователей.

2.7.1. Передача полномочий

Для выполнения определенных административных (привилегированных) действий, например для установки системного времени при помощи команды **date(1)**, нужны права доступа к определенным системным вызовам. В классическом UNIX были предусмотрены простые правила[1] разграничения «все или ничего», т. е. *все* привилегированные действия были разрешены суперпользователю **root** с **UID=0**, и *никакие* привилегированные — всем остальным пользователям. В этих и подобных ситуациях для администрирования операционной системы непривилегированным пользователям необходимо временно воспользоваться правами суперпользователя, что реализуется посредством классической команды *явной* передачи полномочий **su(1)** — switch user, или более поздней команды **sudo(1)** — switch user **do** *контролируемой* передачи полномочий.

Основное различие между **su(1)** и **sudo(1)** заключается в том, что команда **su** реализует «повторную регистрацию в системе», требуя указать имя и ввести пароль того пользователя, чьи полномочия нужно получить. Напротив, команда **sudo** реализует явные правила **sudoers(5)** передачи полномочий, указанные в файле **/etc/sudoers**, и требует подтвердить передачу полномочий паролем того пользователя, который получает передаваемые полномочия (листинг 2.23).

---

[1] На текущий момент времени в Linux реализована система POSIX.1e привилегий **capabilities(7)**, подробнее *см. разд. 4.5.2*.

Листинг 2.23. Передача полномочий

```
iceking@ubuntu:~$ su -l finn
Пароль: <пароль finn'a>
finn@ubuntu:~$ id
uid=1001(finn) gid=1001(finn) группы=1001(finn)
finn@ubuntu:~$ su -l jake
Пароль: <пароль jake'a>
jake@ubuntu:~$ id
uid=1002(jake) gid=1002(jake) группы=1002(jake)
jake@ubuntu:~$ ^D
finn@ubuntu:~$ ^D
iceking@ubuntu:~$ sudo -i -u finn
[sudo] пароль для iceking: <пароль iceking'a>
finn@ubuntu:~$ id
uid=1001(finn) gid=1001(finn) группы=1001(finn)
finn@ubuntu:~$ sudo -i -u jake
[sudo] пароль для finn: <пароль finn'a>
finn отсутствует в файле sudoers. Данное действие будет занесено в журнал.
```

Нужно заметить, что в Ubuntu Linux пароль суперпользователя **root** заблокирован, что не позволяет использовать учетную запись как «обычную» для регистрации в системе и превращает ее в «ролевую». Как следствие, привилегиями «роли» суперпользователя можно пользоваться лишь при помощи **sudo** и только непривилегированным пользователям, явно указанным в правилах передачи **sudoers(5)**.

2.7.2. Хранилища учетных записей

Информация об идентификаторах UID и GID, именах пользователей и групп, их паролях и прочих свойства учетных записей размещаются (в простейшем случае) в файловых «хранилищах» — обычных текстовых файлах каталога **/etc**, формируя базы данных пользовательских **/etc/passwd**, **/etc/shadow** и групповых **/etc/group**, **/etc/gshadow** учетных записей. Формат и структура этих файлов хорошо документированы в руководстве **passwd(5)**, **shadow(5)** и **group(5)**, **gshadow(5)** и представляют собой простейшие текстовые таблицы, где свойства каждой учетной записи представлены набором столбцов одной строки, разделенных символом двоеточия **:** (листинг 2.24).

Листинг 2.24. Базы данных пользовательских учетных записей

```
finn@ubuntu:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
                ...          ...          ...          ...
```

```
dvk:x:1000:1000:Dmitry V. Ketov,,+7(812)703-02-02,,:/home/dvk:/bin/bash
                    ...          ...          ...          ...
finn@ubuntu:~$ cat /etc/group
                    ...          ...          ...          ...
dvk:x:1000:
lpadmin:x:119:dvk,finn
sambashare:x:131:dvk,finn
                    ...          ...          ...          ...
```

При использовании «коммутатора службы имен» (NSS, W:[Name Service Switch]) имеется возможность хранить базы данных учетных записей в любых хранилищах, включая сетевые службы каталогов NIS, NIS+, LDAP, активный каталог Microsoft Windows и даже реляционные сетевые базы данных SQL — при помощи соответствующих модулей NSS (листинг 2.25) и согласно настройкам коммутатора nsswitch.conf(5).

Листинг 2.25. Хранилища пользовательских учетных записей и модули NSS

```
finn@ubuntu:~$ cat /etc/nsswitch.conf
                    ...          ...          ...          ...
passwd:         files systemd
group:          files systemd
shadow:         files
                    ...          ...          ...          ...
finn@ubuntu:~$ find /lib/ -name 'libnss_*'
                    ...          ...          ...          ...
/lib/x86_64-linux-gnu/libnss_systemd.so.2
/lib/x86_64-linux-gnu/libnss_files.so
                    ...          ...          ...          ...
/lib/x86_64-linux-gnu/libnss_winbind.so.2
                    ...          ...          ...          ...
finn@ubuntu:~$ dpkg -S /lib/x86_64-linux-gnu/libnss_winbind.so.2
libnss-winbind:amd64: /lib/x86_64-linux-gnu/libnss_winbind.so.2
finn@ubuntu:~$ dpkg -s libnss-winbind
Package: libnss-winbind
                    ...          ...          ...          -
 This package provides nss_winbind, a plugin that integrates
 with a local winbindd server to provide user/group name lookups to the
 system; and nss_wins, which provides hostname lookups via both the NBNS and
 NetBIOS broadcast protocols.
```

2.8. Переменные окружения и конфигурационные dot-файлы

Для *одноразовой* параметризации выполнения команд служат их *индивидуальные* ключи, указываемые каждый раз при запуске команды, но иногда требуется установить некий параметр, который бы действовал в течение всего *сеанса* работы пользователя с системой, или *общий* параметр, который действовал бы для всех

команд, запускаемых в сеансе. Таким механизмом являются окружение **environ(7)** и переменные окружения, значения которых можно увидеть при помощи команды **env(1)**. Переменные окружения обычно документируются на странице руководства к тем программам, на которые воздействуют, как правило, в разделе **ENVIRONMENT**.

Например, переменная окружения **PATH** содержит перечисление разделенных символом : имен каталогов, где любой командный интерпретатор ищет одноименные запускаемым командам программы (листинг 2.26).

Листинг 2.26. Переменная окружения PATH

```
finn@ubuntu:~$ date
Вс ноя 17 11:31:24 MSK 2019
finn@ubuntu:~$ help -d unset
unset - Unset values and attributes of shell variables and functions.
finn@ubuntu:~$ unset PATH
finn@ubuntu:~$ date
bash: date: Нет такого файла или каталога
```

Переменные окружения **LANGUAGE** и **LANG** содержат идентификаторы языка, на котором программы стараются общаться с пользователем (листинг 2.27); например, **man(1)** ищет перевод страницы руководства. Если точнее, то переменная **LANGUAGE** на самом деле определяет список языков, в порядке которого определяется язык общения, т. к. далеко не все программы имеют переводы, а если и имеют, то не на все языки. Переменная **LANG** определяет язык по умолчанию, если в порядке просмотра **LANGUAGE** ничего подходящего не найдено. Набор переменных окружения **LC_*** определяет другие языковые особенности, отличные от языка сообщений; например, переменная **LC_TIME** определяет формат выводы даты и времени. Кроме того, переводы устанавливаются в систему из специальных языковых пакетов, а список доступных можно увидеть при помощи команды **locale(1)**.

Листинг 2.27. Переменные окружения LANGUAGE LANG и LC_*

```
finn@ubuntu:~$ date
Вс ноя 17 12:55:36 MSK 2019
finn@ubuntu:~$ locale
...             ...             ...             ...
LC_TIME=ru_RU.UTF-8
...             ...             ...             ...
finn@ubuntu:~$ locale -a
C
C.UTF-8
POSIX
```

```
en_GB.utf8
                ...          ...          ...          ...
en_US.utf8
                ...          ...          ...          ...
ko_KR.utf8 ☚
ru_RU.utf8
ru_UA.utf8
finn@ubuntu:~$ export LC_TIME=ko_KR.UTF-8
finn@ubuntu:~$ date
2019. 11. 17. (일) 13:33:12 MSK
finn@ubuntu:~$ cal
     11월 2019
일 월 화 수 목 금 토
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
finn@ubuntu:~$ tar
tar: Необходимо указать один из параметров «-Acdtrux», «--delete» или «--test-label»
Попробуйте «tar --help» или «tar --usage» для
получения более подробного описания.
finn@ubuntu:~$ locale
                ...          ...          ...          ...
LANGUAGE=ru:en ☚
                ...          ...          ...          ...
finn@ubuntu:~$ export LANGUAGE=ko:ru:en
finn@ubuntu:~$ tar
tar: '-Acdtrux', '--delete', '--test-label' 옵션 중 하나를 지정해야합니다
자세한 정보는 'tar --help' 또는 'tar --usage'를 입력하십시오.
finn@ubuntu:~$ $ whatis passwd
passwd (5)          - 패스워드 파일 ☚
passwd (1)          - изменяет пароль пользователя ☚
passwd (1ssl)       - compute password hashes
```

Переменная окружения **PAGER** указывает имя программы «листателя», использующегося другими программами, вывод которых не умещается на один экран. Так, например, поступает **man(1)** при отображении отформатированной страницы руководства, **mail(1)** — при просмотре длинного письма, **mysql(1)** или **psql(1)** — при выводе большого количества результирующих строк ответа за запрос к базе данных. Наиболее распространенным «листателем» является **less(1)**, используемый как замена менее удобного «классического» **more(1)**.

Переменные окружения **EDITOR** и **VISUAL** указывают имя текстового редактора, который будет вызван другими программами при необходимости редактировать текст. Например, **crontab(1)** использует указанный таким образом редактор для изменения списка периодических заданий, **mail(1)** — для редактирования отправляемого сообщения, **mysql(1)** или **psql(1)** — для редактирования длинных запросов к базе данных, а **lftp(1)** — для редактирования списка «закладок». Очень часто в качестве редактора используется «классический» и достаточно непривычный **vi(1)**, который можно таким образом заменить более удобным **nano(1)**.

Переменная **BROWSER** указывает имя просмотрщика HTML (листинг 2.28), который будет использован другими программами при необходимости показать HTML-страницу, например отформатированную таким образом страницу руководства **man(1)**.

Листинг 2.28. Переменная окружения BROWSER

```
finn@ubuntu:~$ man -H ls
sh: 1: exec: www-browser: not found
man: couldn't execute any browser from exec www-browser
finn@ubuntu:~$ export BROWSER=chromium-browser
finn@ubuntu:~$ man -H ls
В текущем сеансе браузера создано новое окно.
```

Некоторые программы имеют специальную переменную окружения, которая содержит ключи, применяемые каждый раз при вызове программы, например **MANOPT** для **man(1)**. Такие переменные для программ **XXX** чаще всего имеют имя **XXXOPT** или **XXX_OPTIONS** или даже **XXX**, например **ZIPOPT** для **zip(1)**, **TAR_OPTIONS** для **tar(1)**, **GZIP** для **gzip(1)** и **LESS/MORE** для **less(1)** и **more(1)** соответственно.

Установив, например, **MANOPT=-H** (**BROWSER=chromium-browser**, или **BROWSER=firefox**, или **BROWSER=links**, или **BROWSER=lynx**), можно просматривать страницы руководства в (одном из указанных графическом и/или текстовом) Web-браузере, а установив **GZIP=-9**, можно заставить упаковщик всегда использовать девятую (самую сильную, но самую медленную) степень сжатия.

Переменная окружения **PS1** изменяет приглашение командного интерпретатора и может содержать подстановки (документированные в **bash(1)**, см. раздел **PROMPTING**), например, **\u** — имя зарегистрированного в системе пользователя, **\h** — короткое собственное имя компьютера, **\w** — имя текущего каталога и **\$** символ приглашения **$** для обычного пользователя и **#** для суперпользователя **root**, **\t** — время в 24-часовом формате, **\e** — управляющий символ **ESC** и пр.

Используя подстановки PS1 и управляющие **ESC**-последовательности **console_codes(7)** или воспользовавшись базой данных управляющих **ESC**-последовательностей **terminfo(5)**

и командой **tput(1)**, можно модифицировать приглашение по своему предпочтению (листинг 2.29).

Листинг 2.29. Переменная окружения PS1

```
finn@ubuntu:~$ man 5 terminfo
                ...             ...             ...             ...
enter_reverse_mode      rev     mr      turn on reverse
                                        video mode
                ...             ...             ...             ...
enter_dim_mode          dim     mh      turn on half-bright
                                        mode
                ...             ...             ...             ...
exit_attribute_mode     sgr0    me      turn off all
                                        attributes
                ...             ...             ...             ...
finn@ubuntu:~$ tput rev | od -a
0000000 esc   [   7   m
0000004
finn@ubuntu:~$ tput dim | od -a
0000000 esc   [   2   m
0000004
finn@ubuntu:~$ tput sgr0 | od -a
0000000 esc   (   B esc   [   m
0000006
finn@ubuntu:~$ PS1='\e[2m\t\n\e(B\e[m\u@\h \e[7m\w\e(B\e[m \$ '
13:17:25
finn@ubuntu ~ $ cd /etc
13:17:25
finn@ubuntu /etc $
```

Или даже можно воспользоваться поддержкой символов **unicode(7)** в **UTF-8(7)** представлении на терминале[1] (листинг 2.30).

Листинг 2.30. Переменная окружения PS1

```
finn@ubuntu:~$ stty
speed 38400 baud; line = 0;
eol = M-^?; eol2 = M-^?; swtch = M-^?;
ixany iutf8
```

[1] В этом примере эмулятор терминала в графическом интерфейсе по умолчанию имеет нужные шрифты, а на консоли необходимо загрузить подходящий Unicode-шрифт, например командой **setfont Uni2-Terminus16**.

```
finn@ubuntu:~$ PS1='\u\342\230\273 \h:\w\$ '
finn☻ ubuntu:~$
```

Переменная окружения **TERM** устанавливает имя терминала, по которому программы, использующие управляющие ESC-последовательности (например, файловый менеджер **mc(1)**), берут их значения из базы данных **terminfo(5)**. Для эмуляторов терминала в графическом интерфейсе ее значение обычно **TERM=xterm** или **TERM=xterm-color**, для консоли Linux **TERM=linux**, а для настоящего аппаратного терминала W:[VT100] **TERM=vt100**. При неправильном значении переменной (листинг 2.31), например, могут «перестать работать» функциональные клавиши просто потому, что программа будет ожидать поступление определенной **ESC**-последовательности ❶, соответствующей функциональной клавише, а в реальности будет поступать другая ❷.

Листинг 2.31. Переменная окружения TERM

```
finn@ubuntu:~$ env
                    ...          ...          ...          ...
TERM=xterm
                    ...          ...          ...          ...
finn@ubuntu:~$ TERM=linux
finn@ubuntu:~$ infocmp
                    ...          ...          ...          ...
❶       kf18=\E[32~, kf19=\E[33~, kf2=\E[[B ☚, kf20=\E[34~,
                    ...          ...          ...          ...
finn@ubuntu:~$ od -a
E↵
   ^D
❷ 0000000 esc   O   Q  ☚ nl
0000004
```

Стоит отметить, что переменные окружения сохраняют свои установленные значения в оперативной памяти командного интерпретатора и теряют их при завершении *сеанса*. Для установки *постоянных* значений параметров программ логично поместить их в какое-либо долгосрочное хранилище, например в файлы на диске. Специальные файлы и каталоги, сохраняющие конфигурационные параметры, по соглашению имеют имена, начинающиеся с точки (англ. *dot* — точка), располагаются в домашнем каталоге пользователя и носят название dot-файлов (листинг 2.32).

Листинг 2.32. Конфигурационные dot-файлы

```
finn@ubuntu:~$ ls -A
                    ...          ...          ...          ...
☚ .bash_history
```

```
.bash_logout
.bashrc
                    ...          ...          ...          ...
.profile
.lftp
.ssh
finn@ubuntu:~$ file .profile .bashrc .lftp .ssh
.profile: ASCII English text
.bashrc:  ASCII English text
.lftp:    directory
.ssh:     directory
```

Некоторые программы, например **lftp(1)**, **ssh(1)** или **ssh(1)**, имеют собственные конфигурационные файлы и/или каталоги, тогда как простейшие **less(1)**, **tar(1)** и **zip(1)** предполагают, что *«постоянные»* параметры по-прежнему задаются при помощи переменных окружения **LESS**, **TAR_OPTIONS** и **ZIPOPT**.

В таких случаях «постоянные» значения переменных сохраняются в каком-либо конфигурационном файле командного интерпретатора **bash**, например считываемом в начале сеанса — **.profile** или при каждом запуске — **.bashrc** (листинг 2.33).

Листинг 2.33. Конфигурационные dot-файлы интерпретатора bash

```
finn@ubuntu:~$ less ~/.bashrc
                    ...          ...          ...          ...
GZIP=-9v
PS1='\e[2m\t\e(B\e[m \u@\h \e[7m\w\e(B\e[m \$ '
                    ...          ...          ...          ...
```

Конфигурационные dot-файлы представляют собой текст на некотором языке, понятном конфигурируемой программе, например язык командного интерпретатора **bash(1)** используется в **.profile** и **.bashrc**.

В большинстве случаев имена dot-файлов и их язык документируются в страницах руководства к «их» программам, обычно в разделе **FILES**. Нередко конфигурационному файлу посвящается отдельная страница в пятой секции, например **nanorc(5)** для dot-файла **.nanorc** текстового редактора **nano(1)** или **netrc(5)** для файла **.netrc** FTP-клиентов **ftp(1)** и **lftp(1)**, или **ssh_config(5)** для **.ssh/config** SSH-клиента **ssh(1)**.

В отдельных случаях, когда конфигурируется не конкретная программа, а общая для многих программ библиотека, например **readline(3)**[1], название переменных окру-

[1] **readline(3)** — библиотека расширенного редактирования вводимой командной строки, которая используются **bash(1)**, **lftp(1)**, **mysql(1)** и пр.

жения, имя и язык конфигурационного файла можно получить (**INPUTRC** и **.inputrc** для **readline(3)**, соответственно) из страницы руководства самой библиотеки.

2.9. В заключение

Командный интерфейс Linux, каким бы «страшным» ни казался, в реальности удивительно функционален для решения массы разнообразных задач, хотя он и не решает абсолютно все задачи одинаково эффективно. Например, его невероятно сложно и неудобно использовать для обработки графической информации, когда при взаимодействии с пользователем требуется ввести колоссальное количество «графических» данных, например указать обтравочный[1] контур. В этом случае графический интерфейс с «непосредственным» манипулированием подойдет гораздо лучше.

Для начинающего пользователя интерфейс командной строки действительно является *непривычным*, что зачастую путают с *неудобством*, так толком и не разобравшись со всеми его возможностями. Вся сила языка командного интерпретатора в полной мере раскрывается в *главе 5*, до освоения которой читателю предлагается не делать скоропалительных выводов.

Естественные языки, которые используют люди для взаимодействия между собой, на порядок сложнее формального командного языка операционной системы. Однако использование глаголов (команд), существительных (аргументов) и наречий (опций) родного языка мало у кого вызывает чувство неудобства. Наоборот, странным покажется тот человеческий индивидуум, который попытается в обществе использовать непосредственное манипулирование, например указывая (щелкая) пальцем в магазине на товары (значки) и мыча что-то нечленораздельное. Скорее всего, мы примем его за иностранца (или это будет ребенок), еще не в полной мере владеющего языком.

Именно *командный интерфейс* в современном виде — аудиоформе — больше не является уделом художественных фантастических произведений, где капитаны межгалактических кораблей командуют кораблям «включить защитное поле». Теперь мы все можем при помощи командного аудиоинтерфейса смартфона найти ближайшую пиццерию или маршрут к нужному месту. Надеюсь, что и алфавитно-цифровая форма командного интерфейса, доставшаяся «в наследство» от UNIX из 70-х годов прошлого века, вас тоже не особо испугает. Эй, Алиса?

---

[1] Выделить пушистого зверька — песца, сфотографированного сидящим в траве.

Глава 3

Подсистема управления файлами и вводом-выводом

3.1. Файлы и дерево каталогов

Все операционные системы семейства W:[UNIX], включая Linux, базируются на одной универсальной идее, заложенной в их общем предке, определившем основные черты семейства — операционной системе UNICS. В аббревиатуре UNICS[1], или же UNiplexed Information & Computing Service, центральное место занимает идея «uniplex»ирования, или же односоставности (односложности) — идея решать разные задачи единым способом.

Одним из выражений этой идеи является утверждение о том, что *информация есть файл*, откуда бы эта информация в систему ни поступала. При помощи файлов обеспечивается доступ к информации на устройствах хранения (записанной ранее), информации с устройств связи (принимаемой из каналов связи в реальном времени), информации из любых других источников. *Файл*, таким образом, является единицей обеспечения *доступа* к информации, а не единицей ее хранения, как в других операционных системах.

Одни файлы обеспечивают доступ к информации, хранимой на разнообразных носителях: магнитных дисках и дискетах, оптических CD/DVD/BD, твердотельных «дисках» и пр. Другие файлы обеспечивают доступ к информации, поступающей из/в устройств ввода-вывода — клавиатур, манипуляторов «мышь», тачпадов, сенсорных экранов, последовательных и параллельных портов, видеокамер, звуковых карт и пр. Особенные файлы обеспечивают доступ к информации о сущностях ядра операционной системы (процессы, нити, модули, драйвера и пр.).

Так или иначе, все файлы одинаково идентифицируются своими *именами*, упорядоченными в форме единой и единственной иерархической структуры, называемой *деревом каталогов* (рис. 3.1).

---

[1] Более поздняя аббревиатура UNIX произносится идентично, но на одну букву короче.

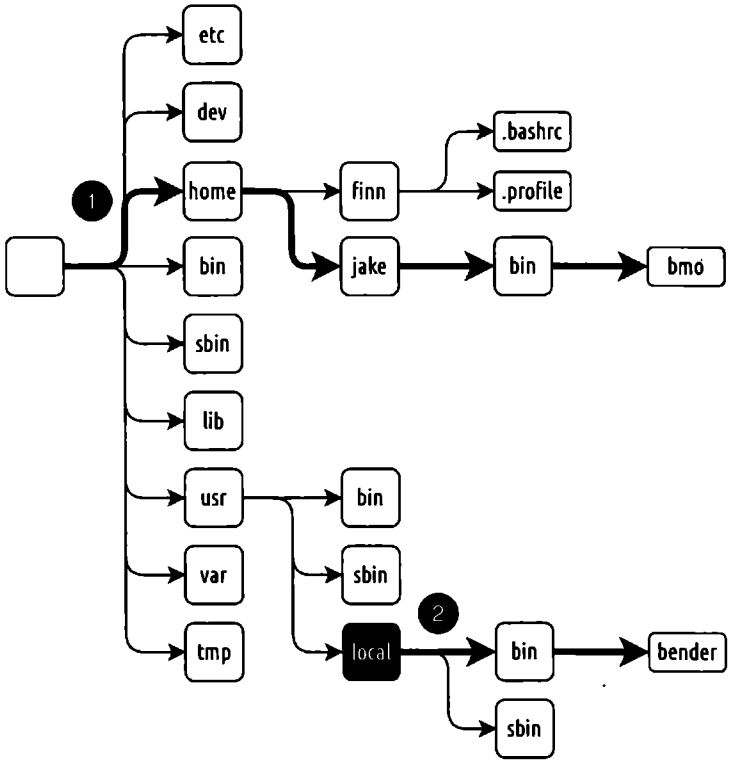


Рис. 3.1. Абсолютное ❶ и относительное ❷ путевые имена

3.1.1. Путевые имена файлов

Глобально уникальным идентификатором файла в пределах операционной системы является его *абсолютное путевое имя*, определямое как путь от корня дерева каталогов до целевого файла, включая начало и конец пути.

Необходимо акцентировать внимание на том, что имя у корневого каталога отсутствует, т. е. является пустой строкой . Таким образом, абсолютное ❶ путевое имя файла bmo (см. рис. 3.1) записывается как разделенные символом / имена всех каталогов пути и имя самого файла, включая концы — ABS: /home/jake/bin/bmo.

Относительное путевое имя вычисляется как остаток пути от некоторого заранее заданного (называемого *рабочим*, WD — working directory) каталога, до целевого файла, включая конец пути. Для рабочего каталога WD: **/usr/local** относительное ❷ путевое имя файла bender записывается как разделенные символом / имена всех каталогов остатка пути и имя самого файла — REL: bin/bender.

Для проверки правильности записи путевых имен всегда можно воспользоваться проверкой ABS = WD ⁄ REL, например **/usr/local/bin/bender = /usr/local ⁄ bin/bender**.

Некоторые каталоги дерева (например, каталоги первого уровня) носят устоявшиеся в семействе операционных систем UNIX имена (см. **hier(7)**) и дифференцируют содержание по смысловому признаку.

Например, каталог **/bin** (**bin**ary) предназначен для *системных* программ общего назначения, каталог **/usr**[1]**/bin** — для *прикладных* (условно) программ общего назначения, каталог **/usr/local/bin** — для *«локально»*[2] установленных прикладных программ общего назначения, а каталоги **bin** внутри домашних каталогов пользователей — для программ *персонального* назначения.

Аналогично определяется назначение каталогов **/sbin**, **/usr/sbin**, **/usr/local/sbin** с той лишь разницей, что каталоги **sbin** расшифровываются как superuser's **bin**aries и предназначаются для программ системного администрирования: системных, прикладных и «локально установленных» соответственно. Каталоги **/lib**, **/usr/lib** и **/usr/local/lib** аналогично содержат *системные* и *прикладные* библиотеки.

Каталог **/etc** содержит общесистемные конфигурационные файлы и с полным правом может[3] расшифровываться как **e**ditable text **c**onfiguration.

Каталог **/home** является контейнером домашних каталогов пользователей (кроме суперпользователя **root**). Каталог **/var** служит хранилищем динамических данных (журнальные файлы **/var/log**, почтовые ящики **/var/mail**, разнообразные очереди **/var/spool** и подобное), а каталог **/tmp** выступает хранилищем временных данных.

Каталоги **/dev**, **/proc** и **/sys** содержат специальные файлы устройств[4] и файлы псевдофайловых систем **proc** и **sysfs**.

3.2. Типы файлов

Файлы как единицы обеспечения доступа к данным различаются операционной системой по *типам*, указывающим источник информации. *Обычные (regular)* файлы и *каталоги (directory)* обеспечивают сохранение информации на тех или иных носителях. *Специальные файлы устройств (special device file)* позволяют обмениваться информацией с тем или иным аппаратным устройством ввода-вывода, а *именованные каналы* и *файловые сокеты* предназначены для обмена информацией между процессом одной программы и процессами других программ.

---

[1] **usr** — в ранней UNIX сокращение от **u**nix **s**ource **r**epositrary, современное сокращение от **u**nix **s**ystem **r**esources.

[2] Установленных системным администратором из сторонних источников, т. е. не из дистрибутива системы.

[3] **etc** — в ранней UNIX сокращение от лат. *et cetera* (и тому подобное) — содержал файлы, которым не нашлось место в других каталогах.

[4] Подробнее *см. разд. 3.2.5* и *3.4.4*.

В примере из листинга 3.1 в полном (**-l**, long) выводе команды **ls(1)** проиллюстрирован признак типа файла. Символом **-** обозначается обычный файл, символом **b** или **c** — специальные файлы блочного (**b**lock) или символьного (**c**haracter) устройства, символом **p** — именованный канал (**p**ipe), символом **s** — сокет (**s**ocket), а символом **l** — символическая ссылка (**l**ink).

Листинг 3.1. Признак типа файлов ls

```
finn@ubuntu:~$ ls -l /bin/ls /dev/sda /dev/tty /sbin/halt
-rwxr-xr-x 1 root root 142144 сен  5 13:38 /bin/ls
brw-rw---- 1 root disk   8, 0 ноя 17 03:31 /dev/sda
crw-rw-rw- 1 root tty    5, 0 ноя 17 12:18 /dev/tty
lrwxrwxrwx 1 root root     14 ноя 13 00:20 /sbin/halt -> /bin/systemctl
finn@ubuntu:~$ ls -l /run/initctl /run/udev/control
prw------- 1 root root 0 ноя 17 03:30 /run/initctl
srw------- 1 root root 0 ноя 17 03:30 /run/udev/control
```

3.2.1. Обычные файлы

Обычные файлы содержат пользовательскую информацию: текст, изображения, звук, видео и прочие данные в виде *набора байтов* (см. ② на рис. 3.2, листинг 3.2). За структуру содержания и имена обычных файлов ответственны прикладные программы, а операционная система не накладывает никаких ограничений.

Листинг 3.2. Содержание обычных файлов

```
finn@ubuntu:~$ file /usr/share/man/man1/file.1.gz
/usr/share/man/man1/file.1.gz: gzip compressed data, max compression, from Unix,
original size modulo 2^32 21484
finn@ubuntu:~$ file /etc/passwd
/etc/passwd: ASCII text
finn@ubuntu:~$ file /bin/ls
/bin/ls: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=2f15ad836be3339dec0e2e6a3c637e08e48aacbd, for GNU/Linux 3.2.0, stripped
finn@ubuntu:~$ file /usr/share/sounds/alsa/Noise.wav
/usr/share/sounds/alsa/Noise.wav: RIFF (little-endian) data, WAVE audio, Microsoft PCM,
16 bit, mono 48000 Hz
finn@ubuntu:~$ file /usr/share/backgrounds/Sky_Sparkles_by_Joe_Thompson.jpg
/usr/share/backgrounds/Sky_Sparkles_by_Joe_Thompson.jpg: JPEG image data, JFIF standard 1.01,
aspect ratio, density 1x1, segment length 16, progressive, precision 8, 3840x2160,
components 3
```

Создать обычный файл можно при помощи любой программы, сохраняющей информацию в файл, например посредством текстовых редакторов **vi(1)**, **nano(1)** или **mcedit(1)**. Для создания пустого обычного файла можно воспользоваться командой **touch(1)** — см. листинг 3.5. Для удаления обычного файла предназначается команда **rm(1)** — см. листинг 3.6.

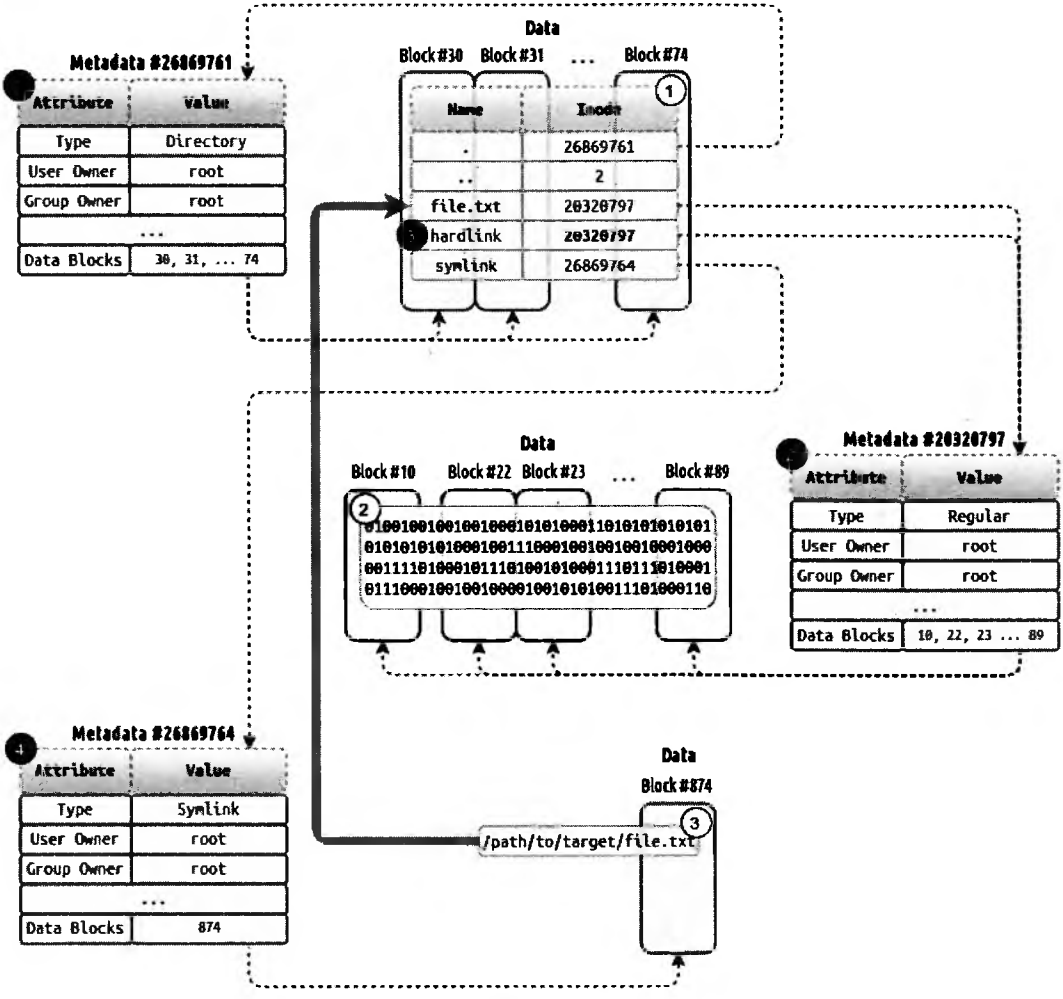


Рис. 3.2. Имена, метаданные и данные файлов

3.2.2. Каталоги

Файлы-каталоги, в отличие от обычных файлов, имеют служебное для операционной системы содержимое — *таблицу имен* (см. ① на рис. 3.2) файлов и соответствующих им номеров индексных дескрипторов (*inode*, index **node**), проиллюстрированных в листинге 3.3.

Листинг 3.3. Имена и номера индексных дескрипторов файлов

```
finn@ubuntu:~$ ls -ai
20332580 .              20318930 .bash_logout  20320866 examples.desktop
20316161 ..             20320868 .bashrc       20320867 .profile
20320797 .bash_history  20332712 .cache
```

Каждый индексный дескриптор содержит *метаданные* (см. ❶❷❹ на рис. 3.2) — список стандартных свойств файла, в том числе указывающих местоположение *данных* файла (набора блоков) на файловой системе. Полный набор метаданных (листинг 3.4) позволяет получить команда **stat(1)**, включая размер файла ❶, количество занимаемых блоков на диске ❷, тип файла ❸, номер индексного дескриптора ❹, права доступа ❺, владельцев ❻ и пр.

Листинг 3.4. Метаданные файла

```
finn@ubuntu:~$ stat .profile
  Файл: .profile
  Размер: 807 ❶       Блоков: 8 ❷        Блок В/В: 4096   ❸ обычный файл
Устройство: 802h/2050d          Inode: ❹ 393433    Ссылки: 1
❺ Доступ: (0644/-rw-r--r--) ❻ Uid: ( 1001/    finn)   Gid: ( 1001/    finn)
Доступ: 2019-11-17 15:14:31.673047212 +0300
Модифицирован: 2019-11-13 00:25:26.723714502 +0300
Изменён: 2019-11-13 00:25:26.723714502 +0300
 Создан: -
```

Для создания каталогов предназначена команда **mkdir(1)**, а для удаления — **rmdir(1)**, при этом удалению подлежат только пустые каталоги (см. листинг 3.8).

3.2.3. Имена, данные, метаданные и индексные дескрипторы

Каждый раз, когда используется путевое (абсолютное или относительное) имя файла, производится итеративный поиск файла в дереве путем последовательного разбора на имя (по которому можно найти метаданные и данные) первого каталога пути, содержащего, в свою очередь, имя второго каталога пути, который содержит имя третьего и т. д., пока в конце поиска не будут найдены имя, метаданные и данные указанного файла. Такая ссылочность позволяет сформировать удобную древовидную структуру для каталогизации файлов (называемую деревом каталогов), однако сам поиск является относительно длительной операцией.

3.2.4. Ссылки

Каталог как файл-список имен других файлов, которым сопоставлены номера индексных дескрипторов, не запрещает иметь два разных имени файла, указывающих на одни и те же *метаданные* (см. ❸ на рис. 3.2). Такой эффект носит название *жесткой ссылки*, создать которую можно при помощи команды **ln(1)** (листинг 3.5).

Листинг 3.5. Жесткая ссылка

```
finn@ubuntu:~$ touch readme
finn@ubuntu:~$ ls -li readme
20318653 -rw-r--r-- 1 finn finn 0 апр.   1 01:22 readme
finn@ubuntu:~$ ln readme readme.txt
finn@ubuntu:~$ touch README
finn@ubuntu:~$ ls -li readme readme.txt README
20318653 -rw-r--r-- 2 finn finn 0 апр.   1 01:22 readme
20319121 -rw-r--r-- 1 finn finn 0 апр.   1 01:23 README
20318653 -rw-r--r-- 2 finn finn 0 апр.   1 01:22 readme.txt
```

Более того, оба имени являются равнозначными, и нет возможности узнать, какое из них создано первым, из чего нужно заключить, что первое и единственное имя файла уже является его жесткой ссылкой (на номер индексного дескриптора). При добавлении файлу нового имени (жесткой ссылки) в его метаданных увеличивается счетчик количества имен (см. ❶ в листинге 3.6), а при удалении файла сначала удаляется имя и уменьшается счетчик количества имен ❷, и только при удалении последнего имени высвобождаются метаданные и данные файла.

Листинг 3.6. Счетчик имен файла

```
finn@ubuntu:~$ ln readme read.me
finn@ubuntu:~$ ls -li read*
20318653 -rw-r--r-- ❶ 3 finn finn 0 апр.   1 01:22 readme
20318653 -rw-r--r--   3 finn finn 0 апр.   1 01:22 read.me
20318653 -rw-r--r--   3 finn finn 0 апр.   1 01:22 readme.txt
finn@ubuntu:~$ rm readme
finn@ubuntu:~$ ls -li read*
20318653 -rw-r--r-- ❷ 2 finn finn 0 апр.   1 01:22 read.me
20318653 -rw-r--r--   2 finn finn 0 апр.   1 01:22 readme.txt
finn@ubuntu:~$ rm readme.txt
finn@ubuntu:~$ ls -li read*
20318653 -rw-r--r-- ❷ 1 finn finn 0 апр.   1 01:22 read.me
```

Нужно заметить, что удаление файла — двухшаговая операция, состоящая из удаления имени файла, а затем — удаления метаданных (и высвобождения блоков, занимавшихся этим файлом). Удаление метаданных файла не выполняется *вообще*, если у файла еще остались имена (жесткие ссылки), и не происходит *сразу*, если файл открыт *(см. разд. 3.3)* каким-либо процессом. Метаданные и блоки, занимаемые файлом, высвобождаются только при закрытии этого файла всеми открывшими его процессами, что проиллюстрировано в примере из листинга 3.7.

Команда **df(1)** измеряет доступное (свободное, **d**isk **f**ree) место на файловой системе указанного файла, тогда как команда **du(1)**, наоборот, измеряет занимаемое (**d**isk **u**sage) указанным файлом место на его файловой системе.

Листинг 3.7. Удаление открытого файла

```
   finn@ubuntu:~$ df -h .
   Файл.система            Размер Использовано  Дост Использовано% Смонтировано в
   /dev/mapper/ubuntu-root   455G        400G   32G           93% /
   finn@ubuntu:~$ du -sh astra-linux-1.3-special-edition-smolensk-disk3-devel.iso
☛  2,8G astra-linux-1.3-special-edition-smolensk-disk3-devel.iso
\  finn@ubuntu:~$ rm astra-linux-1.3-special-edition-smolensk-disk3-devel.iso
   finn@ubuntu:~$ df -h .
   Файл.система            Размер Использовано  Дост Использовано% Смонтировано в
?  /dev/mapper/ubuntu-root   455G        400G   32G           93% /
   finn@ubuntu:~$ lsof astra-linux-1.3-special-edition-smolensk-disk3-devel.iso
   COMMAND   PID   USER   FD   TYPE DEVICE   SIZE/OFF     NODE NAME
   fuseiso 16925   finn    3r   REG  252,0 2947385344 20316584 astra-linux-1.3-special-edition-
   smolensk-disk3-devel.iso
\  finn@ubuntu:~$ kill 16925
   finn@ubuntu:~$ df -h .
   Файл.система            Размер Использовано  Дост Использовано% Смонтировано в
!  /dev/mapper/ubuntu-root   455G        397G   35G           92% /
```

Специальные имена текущего `.` и родительского `..` каталогов на поверку тоже оказываются жесткими ссылками, поэтому у любого каталога по крайней мере два имени — свое «собственное» в родительском и специальное `.` в самом себе, а у каталогов с подкаталогами еще и имена `..` в каждом из дочерних (листинг 3.8).

Листинг 3.8. Имена каталогов

```
finn@ubuntu:~$ mkdir folder
finn@ubuntu:~$ ls -ldi folder
```

```
20357139 drwxr-xr-x  2 finn finn 4096 апр.  1 01:59 folder
finn@ubuntu:~$ cd folder
finn@ubuntu:~/folder$ ls -lai
итого 8
20357139 drwxr-xr-x 2 finn finn 4096 апр.  1 01:59 .
20332580 drwxr-xr-x  4 finn finn 4096 апр.  1 01:59 ..
finn@ubuntu:~/folder$ mkdir child
finn@ubuntu:~/folder$ cd child
finn@ubuntu:~/folder/child$ ls -lai
итого 8
20357140 drwxr-xr-x  2 finn finn 4096 апр.  1 02:02 .
20357139 drwxr-xr-x 3 finn finn 4096 апр.  1 02:02 ..
finn@ubuntu:~/folder/child$ cd ../..
finn@ubuntu:~$ stat folder/
  Файл: «folder/»
  Размер: 4096          Блоков: 8          Блок В/В: 4096   каталог
Устройство: fc00h/64512d        Inode: 20357139    Ссылки: 3
...           ...               ...
finn@ubuntu:~$ rmdir folder
rmdir: не удалось удалить «folder»: Каталог не пуст
```

Существенным ограничением жесткой ссылки в дереве каталогов, куда смонтирована более чем одна файловая система, является локальность жесткой ссылки в пределах своей файловой системы в силу локальной значимости номеров индексных дескрипторов. Так как на каждой новой файловой системе номера индексных дескрипторов начинают нумероваться с нуля, то жесткая ссылка всегда указывает на метаданные файла в «своей» файловой системе и не может указывать на метаданные файла в «чужой» файловой системе общего дерева каталогов. Для преодоления этого ограничения служит *символическая ссылка* **symlink(7)**, являющаяся самостоятельным служебным типом (см. ❹ на рис. 3.2) и содержащая путевое имя (см. ③ на рис. 3.2 и ★ в листинге 3.9) к целевому файлу.

Листинг 3.9. Символическая ссылка

```
finn@ubuntu:~$ ln -s read.me readme.1st
finn@ubuntu:~$ ls -li read*
20318653 -rw-r--r-- 1 finn finn 0 апр.  1 01:22 read.me
20319944 lrwxrwxrwx 1 finn finn 7 апр.  2 00:03 readme.1st -> read.me ★
```

В случае с символической ссылкой при удалении целевого файла сама ссылка будет указывать в никуда и называться «сиротой» (orhpan). Попытка прочитать (лис-

тинг 3.10) такую ссылку приводит к странным на первый взгляд результатам: файл «существует» ❶ для команды **ls(1)**, но команда просмотра содержимого **cat(1)** говорит об обратном ❷. Ничего удивительного, если помнить, что **ls(1)** работает с именами файлов, а **cat(1)** — с их данными (которые действительно не существуют).

Листинг 3.10. Сиротская ссылка

```
finn@ubuntu:~$ rm read.me
finn@ubuntu:~$ ls read*
❶ readme.1st
finn@ubuntu:~$ cat readme.1st
❷ cat: readme.1st: Нет такого файла или каталога
finn@ubuntu:~$ ls -l read*
lrwxrwxrwx 1 finn finn 7 апр.   2 00:03 readme.1st -> read.me
finn@ubuntu:~$ cat read.me
cat: read.me: Нет такого файла или каталога
```

Символические ссылки могут ссылаться на *имена* друг друга неограниченное количество раз, а при попытке использовать одну из таких ссылок будут прочитаны *данные* файла, последнего в цепочке ссылок. По ошибке можно даже закольцевать (листинг 3.11) две или более ссылок, с чем разберется операционная система при чтении одной из ссылок.

Листинг 3.11. Кольцевые ссылки

```
finn@ubuntu:~$ ln -s readme.1st read.me
finn@ubuntu:~$ ls -l read*
lrwxrwxrwx 1 finn finn 10 апр.   2 00:41 read.me -> readme.1st
lrwxrwxrwx 1 finn finn  7 апр.   2 00:03 readme.1st -> read.me
finn@ubuntu:~$ cat read.me
cat: read.me: Слишком много уровней символьных ссылок
```

Основным назначением символических (и изначально, жестких) ссылок является «множественная каталогизация» файлов, т. е. различные наборы разных имен одних и тех же данных. Типичным примером использования ссылок является организация **boot(7)** сценариев запуска системных служб при старте операционной системы. Сами сценарии располагаются в каталоге **/etc/init.d**, а в каталогах **/etc/rcS.d**, **/etc/rc0.d**, ..., **/etc/rc6.d** размещены символические ссылки на эти сценарии, которые должны быть запущены с определенными параметрами при переключении состояния системы между так называемыми «уровнями исполнения» (листинг 3.12).

Листинг 3.12. Сценарии запуска служб и их каталогизация по уровням исполнения

```
finn@ubuntu:~$ ls -l /etc/rc?[1].d
                  ...
/etc/rc2.d:
                  ...                  ...                  ...
lrwxrwxrwx 1 root root 20 ноя 13 00:16 /etc/rc2.d/S19postgresql -> ../init.d/postgresql
lrwxrwxrwx 1 root root 17 ноя 13 00:16 /etc/rc2.d/S20postfix -> ../init.d/postfix
                  ...                  ...                  ...
/etc/rc6.d:
                  ...                  ...                  ...
lrwxrwxrwx 1 root root 17 ноя 13 00:16 /etc/rc6.d/K20postfix -> ../init.d/postfix
lrwxrwxrwx 1 root root 17 ноя 13 00:16 /etc/rc6.d/K21postgresql -> ../init.d/postgresql
                  ...                  ...                  ...
```

В этом примере сценарии **postfix** и **postgresql** имеют вторичные имена, начинающиеся с **K** в каталоге **rc6.d**, и другие вторичные имена, начинающиеся с **S** в каталоге **rc2.d**. Это символизирует необходимость запускать (start) службы **postfix** и **postgresql** при переключении системы на уровень исполнения № 2 и уничтожать (kill) процессы этих служб при переключении системы на уровень исполнения № 6.

3.2.5. Специальные файлы устройств

Специальные файлы устройств предназначены для ввода данных с аппаратных устройств и вывода данных на них. Настоящую работу по вводу и выводу данных проделывает драйвер устройства, а специальные файлы (листинг 3.13) играют роль своеобразных «порталов» связи с драйверами. Различают *символьные* ❶ и *блочные* ❷ специальные файлы устройств, у которых минимальной единицей обмена информацией с драйверами является блок (обычно размером 512 байт) или символ (1 байт), соответственно.

Листинг 3.13. Специальные файлы устройств

```
   finn@ubuntu:~$ ls -l /dev/sd* /dev/input/mouse* /dev/video* /dev/snd/pcm*
❶  crw-rw----  1 root input  13 ①, 32 ② ноя 17 03:31  /dev/input/mouse0
   crw-rw----  1 root input  13,    33    ноя 17 03:31  /dev/input/mouse1
   brw-rw----  1 root disk    8,     0    ноя 17 03:31  /dev/sda
   brw-rw----  1 root disk    8,     1    ноя 17 03:31  /dev/sda1
❷  brw-rw----  1 root disk    8,     5    ноя 17 03:31  /dev/sda5
   crw-rw----+ 1 root audio 116,     3    ноя 17 15:03  /dev/snd/pcmC0D0c
```

[1] Подробнее о шаблонных символах **?**, ***** и пр. *см. разд. 5.3.*

```
crw-rw----+ 1 root audio 116,   2  ноя 17 16:01  /dev/snd/pcmC0D0p
crw-rw----+ 1 root audio 116,   4  ноя 17 03:31  /dev/snd/pcmC0D1c
crw-rw----+ 1 root video  81,   0  ноя 17 03:31  /dev/video0
```

Все драйверы ядра пронумерованы главными (мажорными, **major**) числами ①, а аппаратные устройства, находящиеся под их управлением, — дополнительными (минорными, **minor**) числами ②.

Например, все IDE-диски работают под управлением драйвера **hd(4)**, имеющего 3^{major} (первичный контроллер) и 22^{major} (вторичный контроллер), и нумеруются как 0_{minor} (мастер-диск) и 64_{minor} (слэйв-диск). Аналогично, SCSI-диски работают под управлением драйвера **sd(4)**, имеющего 8^{major}, и нумеруются 0_{minor} (первый диск), 16_{minor} (второй диск) и т. д.

Основной характеристикой специальных файлов устройств является пара чисел **major**, **minor** (иногда называемых характеристическими числами), привязывающая их к конкретному драйверу и управляемому им устройству. Имена специальных файлов и их местоположение в дереве каталогов не имеют никакого значения, но по соглашению **MAKEDEV(8)** их принято располагать в каталоге **/dev** и именовать созвучно именам драйверов.

Специальными файлами дисковых устройств пользуются программы, управляющие структурами самого носителя, например таблицами разделов **fdisk(8)** и **parted(8)** (листинг 3.14), или механикой накопителя **eject(1)** (листинг 3.15), или файловыми системами разделов носителя **mount(8)**, **fsck(8)**, **mkfs(8)** и пр.

Листинг 3.14. Чтение таблицы разделов диска

```
finn@ubuntu:~$ sudo fdisk -l  /dev/sda
Диск /dev/sda: 500.1 Гб, 500107862016 байт
255 головок, 63 секторов/треков, 60801 цилиндров, всего 976773168 секторов
Units = секторы of 1 * 512 = 512 bytes
Размер сектора (логического/физического): 512 байт / 4096 байт
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Идентификатор диска: 0x000c8d62

Устр-во Загр     Начало       Конец       Блоки   Id  Система
/dev/sda1   *       2048      499711      248832   83  Linux
/dev/sda2         501758   976771071   488134657    5  Расширенный
Partition 2 does not start on physical sector boundary.
/dev/sda5         501760   976771071   488134656   8e  Linux LVM
```

Листинг 3.15. Открытие лотка DVD

```
finn@ubuntu:~$ ls -l /dev/dvd
lrwxrwxrwx 1 root root 3 марта 27 14:54 /dev/dvd -> sr0
finn@ubuntu:~$ eject /dev/dvd
```

Особенное место занимают драйверы терминалов[1] *(см. главу 2)* — оконечных устройств для взаимодействия с пользователями. Аппаратные терминалы (например, VT100), подключающиеся посредством последовательного интерфейса RS232, доступны при помощи специальных файлов **/dev/ttyS*N*** драйвера приемопередатчика последовательного порта **ttyS(4)**. Виртуальные терминалы, реализуемые консолью (стандартной клавиатурой и дисплеем в алфавитно-цифровом режиме — см. рис. 2.3), доступны при помощи специальных файлов **/dev/tty*N*** (листинг 3.16) драйверов консоли **console_ioctl(4)** и **tty_ioctl(4)**.

Листинг 3.16. Специальные файлы терминалов

```
finn@ubuntu:~$ ls -la /dev/tty?
crw--w---- 1 root      tty 4, 0 ноя 17 03:31 /dev/tty0
crw------- 1 finn      tty 4, 1 ноя 17 15:03 /dev/tty1
crw------- 1 jake      tty 4, 2 ноя 17 13:27 /dev/tty2
crw------- 1 marceline tty 4, 3 ноя 17 15:14 /dev/tty3
crw------- 1 iceking   tty 4, 4 ноя 17 03:31 /dev/tty4
crw------- 1 bubblegum tty 4, 5 ноя 17 03:31 /dev/tty5
               ...             ...             ...
```

Именно эти специальные файлы терминалов используют команды **write(1)** и **wall(1)** для отсылки сообщений пользователям, зарегистрировавшимся в системе, **setfont(1)** — для смены шрифтов, **chvt(1)** — для переключения между терминалами, а **setleds(1)** — для управления светодиодами клавиатуры (листинг 3.17).

Листинг 3.17. Включение светодиодов CAPS LOCK и SCROLL LOCK

```
finn@ubuntu:~$ tty
/dev/tty1
finn@ubuntu:~$ setleds -L +num +scroll
```

Кроме специальных файлов настоящих аппаратных устройств, в арсенале Linux имеются особые файлы псевдоустройств, такие как **/dev/null**, **/dev/full** и **/dev/zero**,

[1] См. W:[TTY абстракция].

симулирующих всегда пустое, всегда полное и бесконечно нулевое устройство, см. null(4).

Всегда пустое устройство **/dev/null** широко применяется на практике в качестве «подавляющего» стока информации при перенаправлениях[1], а бесконечно нулевое устройство **/dev/zero** зачастую используют в качестве источника для получения нулевых файлов нужной величины.

Псевдоустройства **/dev/random** и **/dev/urandom** организуют доступ к ядерному генератору случайных и псевдослучайных чисел random(4), основанных на внешних событиях периферийных устройств. Потоки случайных чисел используются в качестве источников энтропии для криптоалгоритмов, в частности библиотекой W:[OpenSSL], или служат команде shred(1) источником случайных байтов для надежного стирания данных.

3.2.6. Именованные каналы и файловые сокеты

Именованные каналы и файловые сокеты являются простейшими средствами межпроцессного взаимодействия (IPC, InterProcess Communication) и служат программам для обмена информацией между собой. Разные программы выполняются в рамках различных процессов[2] (изолированных друг от друга), поэтому для общения нуждаются в специальных средствах взаимодействия. Таким средством могли бы стать обычные файлы, но их основное назначение состоит в *сохранении* информации на каком-либо накопителе, что будет при *обмене* информацией сопряжено с накладными расходами, например задержками записи/чтения дискового (т. е. механического) носителя. Предоставить процессам возможность использовать файловые операции для эффективного *взаимодействия* между собой призваны *именованные каналы* (named pipe) pipe(7), они же *FIFO-файлы* (first in first out) fifo(7) и *файловые сокеты* (socket) unix(7). Каналы и сокеты используют для передачи данных от процесса к процессу оперативную память ядра операционной системы, а не память накопителя, как обычные файлы.

Основное отличие (подробнее *см. разд. 4.9*) именованного канала от сокета состоит в способе передачи данных. Через именованный канал организуется однонаправленная (симплексная) передача без мультиплексирования, а через сокет — двунаправленная (дуплексная) мультиплексированная передача.

Именованный канал обычно используют при взаимодействии процессов по схеме «поставщик — потребитель» (producer-consumer), когда один потребитель принимает информацию от одного поставщика (на самом деле от разных, но в различ-

[1] О перенаправлениях *см. разд. 5.3*.

[2] О процессах *см. главу 4*.

ные моменты времени). Например, программы **halt(8)**, **shutdown(8)**, **reboot(8)**, **poweroff(8)** и **telinit(8)** передавали ранее посредством именованного канала **/dev/initctl** команды перезагрузки, выключения питания и другие диспетчеру **init(8)**[1], который и выполнял соответствующие действия.

Сокет используют при взаимодействии по схеме «клиент — сервер» (client-server), т. е. один сервер принимает и отправляет информацию от многих и ко многим (одновременно) клиентам. Например, в целях централизованного сбора событийной информации разнообразные службы операционной системы (в частности, служба периодического выполнения заданий **cron(8)**, служба печати **cupsd(8)** и даже команда **logger(1)**) передают посредством файлового сокета **/dev/log** (в современный момент является символической ссылкой на актуальный **/run/systemd/journal/dev-log**) сообщения о происходящих событиях службе журнализации, представленной в современных системах **systemd-journald(8)**.

3.3. Файловые дескрипторы

Основными операциями, предоставляемыми ядром операционной системы программам (а точнее — процессам[2]) для работы с файлами, являются системные вызовы **open(2)** **read(2)**, **write(2)** и **close(2)**. В соответствии со своими именами эти системные вызовы предназначены для открытия и закрытия файла, для чтения из файла и записи в файл. Дополнительный системный вызов **ioctl(2)** (**i**nput **o**utput **c**ontrol) используется для управления драйверами устройств и, как следствие, применяется в основном для специальных файлов устройств.

При запросе процесса на открытие файла системным вызовом **open(2)** производится его однократный (относительно медленный, *см. разд. 3.2.2*) поиск имени файла в дереве каталогов и для запросившего процесса создается так называемый *файловый дескриптор* (описатель, от англ. *descriptor*). Файловый дескриптор «содержит» информацию, описывающую файл, например индексный дескриптор **inode** файла на файловой системе, номера **major** и **minor** устройства, на котором располагается файловая система файла, режим открытия файла и прочую служебную информацию. При последующих операциях **read(2)** и **write(2)** доступ к самим данным файла происходит с использованием файлового дескриптора (что исключает медленный поиск файла в дереве каталогов).

Файловые дескрипторы пронумерованы и содержатся в таблице открытых процессом файлов, которую можно получить (листинг 3.18) при помощи диагностической программы **lsof(1)**. Наоборот, получить список процессов, открывших тот или иной

---

[1] В современных системах его роль выполняет **systemd(1)**, *см. главу 10*.

[2] Подробнее о процессах *см. главу 4*.

файл, можно при помощи программ **lsof(1)** и **fuser(1)**, что бывает полезно для идентификации программ, «занявших» файловую систему, подлежащую отмонтированию (см. ★ в листинге 3.21).

Листинг 3.18. Таблица файловых дескрипторов

```
① finn@ubuntu:~$ lsof -p $$[1]
  COMMAND   PID USER   FD ☚ TYPE DEVICE SIZE/OFF     NODE NAME
    ...              ...                ...
  bash    17975 finn   1u ❶ CHR  136,2      0t0        5 /dev/pts/2
    ...              ...                ...

② root@ubuntu:~# ls -la /dev/log
  lrwxrwxrwx 1 root root 28 ноя 17 03:30 /dev/log -> /run/systemd/journal/dev-log
  root@ubuntu:~# lsof /run/systemd/journal/dev-log
  COMMAND   PID USER   FD     TYPE             DEVICE SIZE/OFF  NODE NAME
  systemd     1 root   35u    unix 0xffff964892232400      0t0 13321 /run/.../dev-log ...
  systemd-j 285 root    6u ❷ unix 0xffff964892232400      0t0 13321 /run/.../dev-log ...

  root@ubuntu:~# fuser /run/systemd/journal/dev-log
  /run/systemd/journal/dev-log:     1   285 ☚

  root@ubuntu:~# ps p 285
    PID TTY      STAT   TIME COMMAND
    285 ?        S<s    0:04 /lib/systemd/systemd-journald

  root@ubuntu:~# lsof /var/log/syslog
  COMMAND  PID   USER   FD     TYPE DEVICE SIZE/OFF   NODE NAME
  rsyslogd 606 syslog   8w ❸  REG    8,2  1920291 131082 /var/log/syslog

  root@ubuntu:~# ps up 606
  USER       PID %CPU %MEM    VSZ   RSS TTY    STAT START   TIME COMMAND
  syslog     606  0.0  0.1 224360  4584 ?      Ssl  ноя17   0:00 /usr/sbin/rsyslogd -n ...
```

В первом ① примере из листинга 3.18 показано получение списка файловых дескрипторов (столбец **FD**) процесса командного интерпретатора **bash** пользователя **finn**, на котором файловый дескриптор ❶ номер **1** описывает открытый на чтение и запись **u** специальный символьный **CHR** файл устройства **/dev/pts/2**. Во втором ② примере показано получение информации о процессах, открывших файловый сокет

[1] О подстановках командного интерпретатора *см. разд. 5.4.2*.

unix с именем **/run/systemd/journal/dev-log** (файловый дескриптор ❷ номер **6** на чтение и запись **u**) и обычный файл **REG** с именем **/var/log/sysog** (файловый дескриптор ❸ номер **8** на запись **w**).

Пронаблюдать за использованием системных вызовов файлового программного интерфейса в момент выполнения программам позволяет системный трассировщик **strace(1)** (листинг 3.19).

Листинг 3.19. Трассировка файлового программного интерфейса

```
finn@ubuntu:~$ date
Пн ноя 18 00:13:53 MSK ☚ 2019
finn@ubuntu:~$ strace -fe open,openat,close,read,write,ioctl date
                ...                ...                ...
☛ openat(AT_FDCWD, "/etc/localtime", O_RDONLY|O_CLOEXEC) = 3
                ...                ...                ...
finn@ubuntu:~$ file /etc/localtime
/etc/localtime: symbolic link to /usr/share/zoneinfo/Europe/Moscow
finn@ubuntu:~$ file /usr/share/zoneinfo/Europe/Moscow
/usr/share/zoneinfo/Europe/Moscow: timezone data, version 2, 17 gmt time flags, 17 std time
flags, no leap seconds, 78 transition times, 17 abbreviation chars

finn@ubuntu:~$ ls -la /dev/dvd
lrwxrwxrwx 1 root root 3 ноя 17 22:35 /dev/dvd -> sr0
finn@ubuntu:~$ strace -fe open,openat,close,read,write,ioctl eject
                ...                ...                ...
openat(AT_FDCWD, "/dev/sr0", O_RDWR|O_NONBLOCK) = 3 ☚
☛ ioctl(3, CDROMEJECT, 0x01)            = 0
close(3)                                = 0

finn@ubuntu:~$ strace -fe open,openat,read,write,close,ioctl setleds -L +num +scroll
                ...                ...                ...
ioctl(0, KDGKBLED, 0x7ffc1ced3d27)      = 0
ioctl(0, KDGETLED, 0x7ffc1ced3d26)      = 0
ioctl(0, KDSETLED, 0x3)                 = 0
```

Предположив, что программа **date(1)** показывает правильное московское время, потому что узнаёт заданную временную зону MSK из некоего конфигурационного файла операционной системы, при трассировке ее работы можно установить его точное имя — **/etc/localtime** (оказавшимся символической ссылкой на **/usr/share/zoneinfo/Europe/Moscow**). Аналогично предположив, что программа **eject(1)** открывает лоток привода CD/DVD при помощи специального файла устройства, при трассировке можно узнать имя файла — **/dev/sr0**, номер файлового дескриптора при работе с файлом — **3** и команду **CDROMEJECT** соответствующего устройству

драйвера ioctl_list(2). Трассировка команды setleds(1) показывает, что она вообще не открывает никаких файлов, но пользуется файловым дескриптором 0 так называемого стандартного потока ввода (прикрепленного к текущему терминалу[1]) и командами KDGETLED и KDSETLED драйвера консоли console_ioctl(4).

3.4. Файловые системы

3.4.1. Файловые системы и процедура монтирования

Доступ к информации организуется при помощи файлов, упорядоченных в единое «воображаемое» дерево каталогов, тогда как «настоящими» *источниками* данных являются *файловые системы* — структуры, решающие задачи *хранения*[2] информации. Отображение *множества* файловых систем в *единое* дерево каталогов реализуется посредством процедуры *монтирования*. Таким образом, все, что наблюдается в дереве каталогов, в реальности размещается на файловых системах.

Состав дерева каталогов показывает команда mount(8), равно как и присоединяет к нему — монтирует очередную файловую систему. Отсоединяет (отмонтирует) файловую систему от дерева каталогов команда umount(8), но только при условии, что ни один файл на этой файловой системе не используется никакой программой (а правильнее — никаким процессом) операционной системы.

В примере на рис. 3.3 и в листинге 3.20 иллюстрируются: файловая система W:[ext4], располагающаяся на дисковом накопителе, идентифицирующемся файлом устройства **/dev/sda2**, и смонтированная непосредственно в корень дерева каталогов ❶; файловая система **vfat** flash-накопителя ❹ на устройстве **/dev/sdb1**, смонтированная в **/media/flash**; файловая система W:[ISO 9660] CD-диска ❺ на устройстве **/dev/sr0**, смонтированная в **/media/cdrom**.

Кроме этого, в дерево каталогов смонтированы две псевдофайловые системы **proc** ❷ и **sysfs** ❸, считывающие из оперативной памяти ядра операционной системы информацию о процессах, обнаруженных устройствах, загруженных драйверах и предоставляющих «файловый» доступ к ней.

Листинг 3.20. Состав дерева каталогов

```
finn@ubuntu:~$ mount
❶ /dev/sda2 on / type ext4 (rw,relatime,errors=remount-ro)
❷ proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
```

---

[1] О стандартных потоках ввода-вывода *см. разд. 5.3.*

[2] Или извлечения информации откуда-либо *(см. разд. 3.4.4 и 3.4.5)*.

❸ sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
... ...
❹ /dev/sdb1 on /media/flash type vfat (rw,...)
❺ /dev/sr0 on /media/cdrom type iso9660 (ro,...)

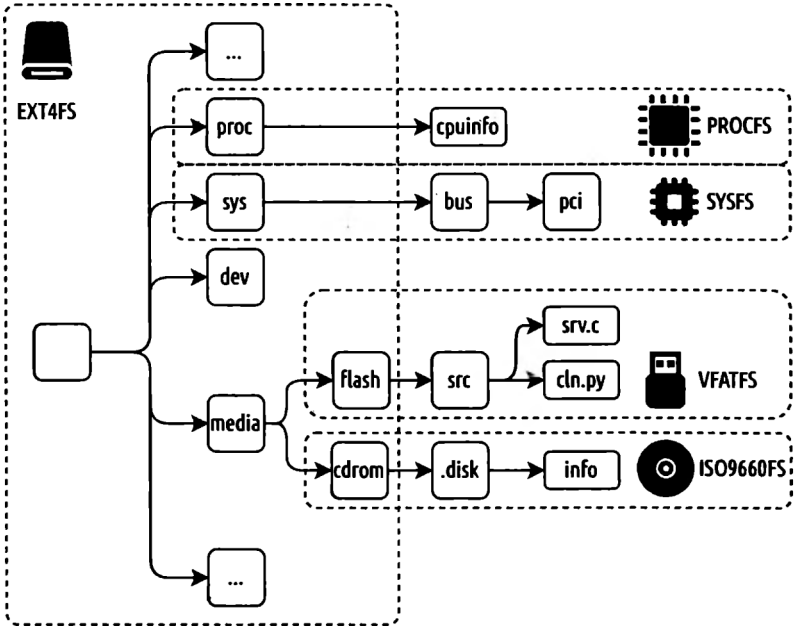


Рис. 3.3. Монтирование файловых систем

Несмотря на то, что в современных дистрибутивах Linux обнаружение и процедуры монтирования файловых систем автоматизированы, операции монтирования/размонтирования могут быть произведены вручную (листинг 3.31).

Листинг 3.21. Процедуры монтирования/размонтирования файловых систем

```
finn@ubuntu:~$ mount /dev/dvd /media/cdrom
mount: только root может сделать это
finn@ubuntu:~$ sudo mount /dev/dvd /media/cdrom
mount: /media/cdrom: ВНИМАНИЕ: устройство защищено от записи, смонтировано только для чтения.
finn@ubuntu:~$ mount
                ...          ...          ...          ...
/dev/dvd on /media/cdrom type iso9660 (ro,...)
                ...          ...          ...          ...
finn@ubuntu:~$ cat /media/cdrom/.disk/info
Ubuntu 19.10 "Eoan Ermine" - Release amd64 (20191017)
finn@ubuntu:~$ umount /media/cdrom
umount: /media/cdrom: umount failed: Операция не позволена.
```

```
finn@ubuntu:~$ sudo umount /media/cdrom
finn@ubuntu:~$ cat /media/cdrom/.disk/info
cat: /media/cdrom/.disk/info: Нет такого файла или каталога
finn@ubuntu:~$ sudo umount /proc
umount: /proc: target is busy.

finn@ubuntu:~$ mount /dev/sdc1 /media/flash
finn@ubuntu:~$ mount
                ...          ...          ...          ...
/dev/sdc1 on /media/flash type vfat (rw,...)
                ...          ...          ...          ...
```

3.4.2. Дисковые файловые системы

Разные файловые системы fs(5), как упоминалось ранее, предназначены для *хранения* информации на внешних носителях и преследуют различные цели, например обеспечивают надежное хранение при помощи журнала транзакций или быстрый поиск метаданных файла (среди множества каталогов, подкаталогов и других файлов) по его имени либо учитывают специфику свойств самого носителя и т. д. В большинстве случаев до сих пор носителями информации являются магнитные или оптические диски; благодаря чему файловые системы, размещаемые на них, зачастую называются «дисковыми» файловыми системами, даже если используются на твердотельных (flash) носителях.

Для магнитных дисков, характеризуемых возможностью чтения и записи блоков информации в произвольное место носителя (random access), в Linux на текущий момент времени используются «родные» файловые системы **W:[Ext2]**, **W:[Ext3]** и **W:[Ext4]**, специально разработанные **W:[ReiserFS]** и **W:[Reiser4]**, а также заимствованные **W:[XFS]** и **W:[JFS]**.

Для оптических CD/DVD-дисков, имеющих специфику записи в виде спиральной дорожки, применяются файловые системы **W:[ISO 9660]** и **W:[udf]**. Для USB-flash-накопителей в большинстве случаев используются заимствованные файловые системы **W:[FAT]** и **W:[NTFS]** в силу использования этих накопителей как мобильных средств переноса данных между разными компьютерами с различными операционными системами.

3.4.3. Сетевые файловые системы

Сетевые файловые системы, равно как и дисковые, обеспечивают *хранение* информации на внешнем носителе, которым в этом случае выступает файловый сервер (например, домашний NAS, Network Attached Storage), доступный по протоколу **NFS** (Network File System, **W:[Network File System]**), **CIFS/SMB** (Common Internet File

System или Server Message Block, W:[Server_Message_Block]) или им подобным. Одноименные файловые системы **nfs** и **cifs/smb** используются для монтирования файлов сервера в дерево каталогов клиента. Таким образом, обычные (ничего не знающие ни про какие сетевые протоколы) программы, запускаемые в операционной системе клиента, используют файлы сетевого сервера точно так, как если бы они были размещены на локальных дисках, под управлением дисковых файловых систем.

В примере из листинга 3.22 программы **avconv(1)** и **avprobe(1)**, предназначенные для работы с «обычными» видеофайлами, используются для обработки записей сетевого видеорегистратора, видеофайлы которого доступны по протоколу **NFS**. Смонтированные при помощи сетевой файловой системы **nfs** в дерево каталогов файлы *сетевого* регистратора становятся никак неотличимы от файлов *локальных* дисковых файловых систем.

Листинг 3.22. Сетевая файловая система NFS

```
finn@ubuntu:~$ mount -t nfs 182.168.1.10:/share/video /mnt/nas/video
finn@ubuntu:~$ mount
                    ...          ...          ...          ...
182.168.1.10:/share/video on /mnt/nas/video type nfs (rw,...)
                    ...          ...          ...          ...
finn@ubuntu:~$ cd /mnt/nas/video/screencasts
finn@ubuntu:~$ ls
                    ...          ...          ...          ...
20140523142626.mp4
                    ...          ...          ...          ...

finn@ubuntu:~$ file 20140523142626.mp4
20140523142626.mp4: ISO Media, MPEG v4 system, version 2

finn@ubuntu:~$ avprobe 20140523142626.mp4
                    ...          ...          ...          ...
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from '20140523142626.mp4':
                    ...          ...          ...          ...
  Duration: 00:00:07.94, start: 0.000000, bitrate: 11020 kb/s
    Stream #0.0(eng): Video: h264 (High), yuv420p, 1920x1080 [PAR 1:1 DAR 16:9], 10843 kb/s,
50 fps, 50 tbr, 50k tbn, 100 tbc
                    ...          ...          ...          ...

finn@ubuntu:~$ avconv -i 20140523142626.mp4 20140523142626.mkv
                    ...          ...          ...          ...
Output #0, matroska, to '20140523142626.mkv':
                    ...          ...          ...          ...
    Stream #0.0(eng): Video: mpeg4, yuv420p, 1920x1080 [PAR 1:1 DAR 16:9], q=2-31, 200 kb/s,
1k tbn, 50 tbc
                                 ...          ...          ...
Stream mapping:
  Stream #0:0 -> #0:0 (h264 -> mpeg4)
  Stream #0:1 -> #0:1 (aac -> libvorbis)
```

```
Press ctrl-c to stop encoding
frame=  395 fps= 72 q=31.0 Lsize=    2481kB time=7.94 bitrate=2561.5kbits/s dup=0 drop=1
video:2339kB audio:128kB global headers:4kB muxing overhead 0.457686%
```

Аналогично, в примере из листинга 3.23 геотеги файлов изображений сетевого видеорегистратора анализируются утилитой **exiv2(1)**, предназначенной для работы с «обычными» изображениями. За счет файловой системы **cifs** и доступности видеорегистратора по протоколу **CIFS** его содержимое смонтировано в дерево каталогов так, словно *сетевой* регистратор является *локальным* дисковым накопителем.

Листинг 3.23. Сетевая файловая система CIFS/SMB

```
finn@ubuntu:~$ mount -t cifs -o username=guest //182.168.1.10/share/photos /mnt/nas/photos
Password:
finn@ubuntu:~$ mount
                  ...          ...              ...              ...
//182.168.1.10/share/photos on /mnt/nas/video type cifs (rw,...)
                  ...          ...              ...              ...
finn@ubuntu:~$ cd /mnt/nas/photos
finn@ubuntu:~$ ls
                  ...          ...              ...              ...
DSC_0034.JPG  DSC_0043.JPG  DSC_0062.JPG  DSC_0074.JPG  DSC_0100.JPG  DSC_0189.JPG
                  ...          ...              ...              ...

finn@ubuntu:~$ exiv2 -p a DSC_0043.JPG
                  ...          ...              ...              ...
Exif.GPSInfo.GPSLatitudeRef                 Ascii      2  North
Exif.GPSInfo.GPSLatitude                    Rational   3  60deg 10' 3.479"
Exif.GPSInfo.GPSLongitudeRef                Ascii      2  East
Exif.GPSInfo.GPSLongitude                   Rational   3  24deg 57' 23.294"
                  ...          ...              ...              ...
```

3.4.4. Специальные файловые системы

Развитие идеи файла как единицы обеспечения доступа к информации привело к тому, что абстракцию файловой системы перенесли и на другие сущности, доступ к которым стал организовываться в виде иерархии файлов. Например, информацию о процессах, нитях и прочих сущностях ядра операционной системы и используемых ими ресурсах предоставляет программам виде файлов (!) *псевдофайловая* система **proc(5)**. Таким же образом, информацию об аппаратных устройствах, обнаруженных ядром операционной системы на шинах PCI, USB, SCSI и пр., предоставляет *псевдофайловая* система **sysfs**.

Различные утилиты, пользующиеся ядерной информацией, например показывающие нагрузку на операционную систему **uptime(1)** или списки процессов и загруженных модулей (драйверов) ядра операционной системы — **ps(1)** и **lsmod(8)**, пользуются

псевдофайловой системой **proc**, в чем позволяет убедиться трассировка системных вызовов **open(2)** (листинг 3.24).

Листинг 3.24. Псевдофайловая система proc

```
finn@ubuntu:~$ strace -fe open,openat uptime
                    ...             ...             ...             ...
openat(AT_FDCWD, "/proc/sys/kernel/osrelease", O_RDONLY) = 3
                    ...             ...             ...             ...
openat(AT_FDCWD, "/proc/uptime", O_RDONLY) = 3
                    ...             ...             ...             ...
openat(AT_FDCWD, "/proc/loadavg", O_RDONLY) = 4
                    ...             ...             ...             ...
15:42:32 up 12 days,  5:27,  7 users,  load average: 0.48 ↵, 0.33, 0.32
finn@ubuntu:~$ cat /proc/uptime
1056774.23 1667210.55
finn@ubuntu:~$ cat /proc/loadavg
0.48 ↵ 0.33 0.32 1/623 14277
```

Аналогично, утилиты, показывающие список устройств на шинах PCI, USB и SCSI — **lspci(8)**, **lsusb(8)** и **lsscsi(8)**, пользуются псевдофайловой системой **sysfs** (листинг 3.25).

Листинг 3.25. Псевдофайловая система sysfs

```
finn@ubuntu:~$ strace -fe open,openat lspci -nn
                    ...             ...             ...             ...
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/resource", O_RDONLY) = 4
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/irq", O_RDONLY) = 4
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/vendor", O_RDONLY) = 4
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/device", O_RDONLY) = 4
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/class", O_RDONLY) = 4
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/config", O_RDONLY) = 3
                    ...             ...             ...             ...
00:02.0 VGA compatible controller [0300 ↵]: Intel Corporation 2nd Generation Core Processor
Family Integrated Graphics Controller [8086 ↵:0116 ↵] (rev 09)
finn@ubuntu:~$ cat /sys/bus/pci/devices/0000:00:02.0/vendor
0x8086 ↵
finn@ubuntu:~$ cat /sys/bus/pci/devices/0000:00:02.0/device
0x0116 ↵
finn@ubuntu:~$ cat /sys/bus/pci/devices/0000:00:02.0/class
0x030000 ↵
```

3.4.5. Внеядерные файловые системы

Сосуществование разных по своей сути файловых систем в едином дереве файлов и каталогов позволяет использовать для работы с его файлами «обычные» программы, ничего не знающие ни о местоположении, ни о свойствах самих источников информации этих файлов.

Таким же «файловым» образом можно организовать доступ к файлам внутри GZ/ZIP/...-архивов, файлам внутри ISO-образов CD/DVD-дисков, файлам зашифрованных каталогов, файлам других узлов сети, медиафайлам на медиаустройствах (плееры, смартфоны) и т. п. Файловая абстракция в определенных случаях оказывается удобной даже для осуществления доступа к информации, организованной совершенно «нефайловым» образом — к данным таблиц и представлений внутри реляционных баз данных, к элементам и атрибутам внутри XML-файлов и пр.

Для реализации такого «файлового» доступа к произвольным источниками информации используются так называемые *«внеядерные»* файловые системы **W:[FUSE]** (Filesystem in **USE**rspace), реализуемые не ядерными модулями файловых систем (как **ext4**, **nfs**, **proc** и пр.), а обычными программами[1], запущенными в обычных процессах и работающими вне ядра.

В примере из листинга 3.26 показано, как можно без распаковки смонтировать сжатый архив исходных текстов ядра **linux-4.2.3.tar.xz** и прочитать отдельный файл **fuse.txt**.

Листинг 3.26. Внеядерная файловая система «fuse.archivemount»

```
finn@ubuntu:~$ wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.3.9.tar.xz
                    ...          ...          ...          ...
finn@ubuntu:~$ file linux-5.3.9.tar.xz
linux-5.3.9.tar.xz: XZ compressed data
finn@ubuntu:~$ archivemount linux-5.3.9.tar.xz ~/mnt/archive
                              ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ⓵
finn@ubuntu:~$ mount
                   ...          ...          ...          ...
archivemount on /home/finn/mnt/archive type fuse.archivemount (rw,...)
finn@ubuntu:~$ cd ~/mnt/archive
finn@ubuntu:~/mnt/archive$ ls -l
итого 0
drwxrwxr-x 0  root root 0  ноя  6 15:09 linux-5.3.9
finn@ubuntu:~/mnt/archive$ cd linux-5.3.9/Documentation/filesystems
```

[1] Для чего в ядре все-таки требуется один универсальный модуль **fuse**.

```
finn@ubuntu:~/mnt/archive/linux-5.3.9/Documentation/filesystems$ less fuse.txt
finn@ubuntu:~/mnt/archive/linux-5.3.9/Documentation/filesystems$ cd ~
finn@ubuntu:~ $ fusermount -u ~/mnt/archive
```

В примере из листинга 3.27 показано, как при помощи сетевого протокола SSH *(см. разд. 6.4.1)* можно смонтировать часть дерева каталогов (домашний каталог пользователя **jake**) с удаленного узла **jake@grex.org** в каталог **~/mnt/net** локального дерева каталогов.

Листинг 3.27. Внеядерная файловая система «fuse.sshfs»

```
finn@ubuntu:~$ sshfs jake@grex.org: ~/mnt/net
The authenticity of host 'grex.org (75.61.90.157)' can't be established.
ECDSA key fingerprint is SHA256:pMO3fe6UTyqtqzUMq5SmTmH5tqUuN9WdvlwdpcEJhSU.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
jake@grex.org's password: P@s$w0rd
                          ⓟ ⓐ ⓣ ⓘ ⓝ ⓘ ⓐ ⓝ ⓣ ⓢ ⓘ
finn@ubuntu:~$ mount
                ...          ...          ...          ...
jake@grex.org: on /home/finn/mnt/net type fuse.sshfs (rw,...)
finn@ubuntu:~$ cd ~/mnt/net
finn@ubuntu:~/mnt/net$ ls -l
итого 488
-rw-rw-r-- 1 156620 131077 495604 окт.  10 20:28 OPENME.txz
-rw-r--r-- 1 156620 131077    106 окт.  10 20:30 README.gz
finn@ubuntu:~/mnt/net$ zless README.gz
finn@ubuntu:~/mnt/net$ cd ~
finn@ubuntu:~ $ fusermount -u ~/mnt/net
```

В примере из листинга 3.28 показано, как можно смонтировать зашифровываемый каталог **/media/flash/secret** USB-flash-накопителя (уже смонтированного в каталог **/meida/flash** при помощи дисковой файловой системы **vfat**) в каталог **~/mnt/exposed** и скопировать туда файлы, подлежащие зашифровыванию. Теперь при утере накопителя можно не беспокоиться об информации, попавшей третьим лицам.

Листинг 3.28. Внеядерная файловая система «fuse.encfs»

```
finn@ubuntu:~ $ mount
                ...          ...          ...          ...
/dev/sdc1 on /media/flash type vfat (rw,...)
                ...          ...          ...          ...
finn@ubuntu:~ $ encfs /media/flash/secret ~/mnt/exposed
Директория "/media/flash/secret" не существует. Создать ее? (y,n) y
Директория "/home/finn/mnt/exposed" не существует. Создать ее? (y,n) y
```

```
Создание нового зашифрованого раздела.
Выберите одну из следующих букв:
 введите "x" для режима эксперта,
 введите "p" для режима максимальной секретности,
 любой другая буква для выбора стандартного режима.
?> ↵
Выбрана стандартная конфигурация.

Конфигурация завершена. Создана файловая система
со следующими свойствами:
Шифр файловой системы: "ssl/aes", версия 3:0:2
Шифр файла: "nameio/block", версия 4:0:1
Размер ключа: 192 бит
Размер блока: 1024 байт
Каждый файл содержит 8-байтный заголовок с уникальными IV данными.
Файловые имена зашифрованы с использованием режима сцепления вектора инициализации.
File holes passed through to ciphertext.

Введите пароль для доступа к файловой системе.
Запомните пароль, так как в случае утери его
будет невозможно восстановить данные. Тем не менее
этот пароль можно изменить с помощью утилиты encfsctl.

Новый пароль EncFS: $ecr6t ↵
Повторите пароль EncFS: $ecr6t ↵
finn@ubuntu:~$ mount
encfs on /home/finn/mnt/exposed type fuse.encfs (rw,...)
finn@ubuntu:~$ cp ~/Изображения/IMG_20150801*.jpg ~/mnt/exposed
finn@ubuntu:~$ ls -l ~/mnt/exposed/
                  ...        ...          ...          ...
-rw-r--r-- 1 finn finn 1014408 окт.  10 19:06 IMG_20140801_123522.jpg
                  ...        ...          ...          ...
-rw-r--r-- 1 finn finn 1728838 окт.  10 19:06 IMG_20140801_124215.jpg
finn@ubuntu:~$ ls -l /media/flash/secret
                  ...        ...          ...          ...
-rw-rw-r-- 1 finn finn  0 ноя 18 00:57 JEG0j8acowSgiyCUeyqqAbohACIKSEM4JShALYNc6PF1l0
                  ...        ...          ...          ...
-rw-rw-r-- 1 finn finn  0 ноя 18 00:57 znchYIQd4VP1q7u5UkH2wNqzeYhFeLzTpKDN455rf1o8T1
finn@ubuntu:~$ file ~/mnt/exposed/IMG_20140801_123522.jpg
IMG_20140801_123522.jpg: JPEG image data, EXIF standard
finn@ubuntu:~$ file /media/flash/secret/9WgX2m6hiNNz8,ii7UI1AIPetU3qGuLbRNyww9eHTiNNi-
9WgX2m6hiNNz8,ii7UI1AIPetU3qGuLbRNyww9eHTiNNi-: data
finn@ubuntu:~$ fusermount -u /media/flash/secret
```

В примере из листинга 3.29 показано, как можно монтировать файловые системы **fuse** друг поверх друга в стек — смонтировать содержимое дерева каталогов FTP-сервера **mirror.yandex.ru**, далее смонтировать содержимое ISO-образа **FreeBSD-12.1-RELEASE-amd64-dvd1.iso**, а затем смонтировать архив исходных текстов **src.txz**. При чтении файла страницы руководства **ls.1** файловые системы будут прозрачно и на лету (!) извлекать файл из архива, архив из образа и образ с сервера без предварительных скачиваний и распаковываний.

Листинг 3.29. Внеядерные файловые системы «fuse.curlftpfs», «fuse.fuseiso» и стекирование файловых систем

```
finn@ubuntu:~$ curlftpfs mirror.yandex.ru ~/mnt/net
                              ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ①
finn@ubuntu:~$ fuseiso ~/mnt/net/freebsd/releases/ISO-IMAGES/12.1/FreeBSD-12.1-RELEASE-amd64-dvd1.iso ~/mnt/cd
                              ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ①
finn@ubuntu:~$ archivemount ~/mnt/cd/usr/freebsd-dist/src.txz ~/mnt/archive
                              ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ①
finn@ubuntu:~$ mount
curlftpfs#ftp://mirror.yandex.ru/ on /home/finn/mnt/net type fuse (rw,...)
fuseiso on /home/finn/mnt/cd type fuse.fuseiso (rw,...)
archivemount on /home/finn/mnt/archive type fuse.archivemount (rw,...)

finn@ubuntu:~$ man ~/mnt/archive/usr/src/bin/ls/ls.1
                  ...          ...          ...          ...
LS(1)                      BSD General Commands Manual

NAME
     ls – list directory contents
                  ...          ...          ...          ...

finn@ubuntu:~$ fusermount -u ~/mnt/archive
finn@ubuntu:~$ fusermount -u ~/mnt/cd
finn@ubuntu:~$ fusermount -u ~/mnt/net
```

3.5. Дискреционное разграничение доступа

В Linux, как и в любой многопользовательской системе, абсолютно естественным образом возникает задача разграничения доступа *субъектов* — пользователей к *объектам* — файлам дерева каталогов. Один из подходов к разграничению доступа — так называемый *дискреционный* (от англ. *discretion* — чье-либо усмотрение) — предполагает назначение *владельцев* объектов, которые по собственному усмотрению определяют *права доступа* субъектов (других пользователей) к объектам (файлам), которыми владеют.

Дискреционные механизмы разграничения доступа используются для разграничения прав доступа процессов *(см. разд. 4.5.1)* как обычных пользователей ❶, так и для ограничения прав системных программ ❷ (например, служб операционной системы), которые работают от лица псевдопользовательских учетных записей *(см. разд. 2.7)*.

В примере из листинга 3.30 при помощи команды **ps(1)** проиллюстрированы процессы операционной системы, выполняющиеся от лица разных учетных записей.

Листинг 3.30. Субъекты разграничения прав доступа: пользователи и псевдопользователи

```
finn@ubuntu:~$ ps axfu
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           2  0.0  0.0      0     0 ?        S    ноя17   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   ноя17   0:00  \_ [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   ноя17   0:00  \_ [rcu_par_gp]
                ...                 ...           ...           ...
root           1  0.0  0.2 167164 10936 ?        Ss   ноя17   0:06 /sbin/init splash
systemd+     577  0.0  0.2  20784  9836 ?        Ss   ноя17   0:01 /lib/systemd/systemd-resolved
syslog       606  0.0  0.1 224360  4584 ?        Ssl  ноя17   0:01 /usr/sbin/rsyslogd -n -iNONE
message+     617  0.0  0.1   9808  6448 ?        Ss   ноя17   0:18 /usr/bin/dbus-daemon --system ...
                ...                 ...           ...           ...
avahi        628  0.0  0.0   8532  3648 ?        Ss   ноя17   0:00 avahi-daemon: running ...
                ...                 ...           ...           ...
daemon       675  0.0  0.0   3736  2232 ?        Ss   ноя17   0:00 /usr/sbin/atd -f
                ...                 ...           ...           ...
whoopsie     812  0.0  0.3 330936 15844 ?        Ssl  ноя17   0:00 /usr/bin/whoopsie -f
root        1145  0.0  0.1  37168  4040 ?        Ss   ноя17   0:00 /usr/lib/postfix/sbin/master -w
postfix     1150  0.0  0.1  37556  5520 ?        S    ноя17   0:00  \_ qmgr -l -t unix -u
postfix    21924  0.0  0.1  37504  5400 ?        S    00:39   0:00  \_ pickup -l -t unix -u -c
                ...                 ...           ...           ...
root        2539  0.0  0.2 251392  9792 ?        Ssl  ноя17   0:00 /usr/sbin/gdm3
root       17750  0.0  0.2 316608  9932 ?        Sl   ноя17   0:00  \_ gdm-session-worker [pam/gdm-...
finn       17764  0.0  0.1 166540  6908 tty3     Ssl+ ноя17   0:00      \_ /usr/lib/gdm3/gdm-x- ...
finn       17766  0.1  1.8 237008 75900 tty3     Sl+  ноя17   0:23          \_ /usr/lib/xorg/Xorg ...
finn       17774  0.0  0.3 194680 15940 tty3     Sl+  ноя17   0:00          \_ /usr/lib/gnome-sessi...
                ...                 ...           ...           ...
finn        2987  0.0  0.3  21112 12356 ?        Ss   ноя17   0:03 /lib/systemd/systemd --user
finn        2992  0.0  0.0 168572  3124 ?        S    ноя17   0:00  \_ (sd-pam)
                ...                 ...           ...           ...
finn       17921  0.7  8.2 2641464 331536 ?      Ssl  ноя17   2:01  \_ /usr/bin/gnome-shell
```

3.5.1. Владельцы и режим доступа к файлам

В рамках дискреционного разграничения доступа каждому файлу назначены *пользователь-владелец* ❶ и *группа-владелец* ❷ файла (листинг 3.31).

Листинг 3.31. Владельцы файлов

```
finn@ubuntu:~$ ls -la /etc/profile .profile
-rw-r--r-- 1 ❶ root ❷ root 581 авг 27 21:31 /etc/profile
-rw-r--r-- 1 finn       finn 807 ноя 13 00:25 .profile
finn@ubuntu:~$ stat .profile
                  ...          ...          ...          ...
Доступ: (0644/-rw-r--r--)  ❶ Uid: ( 1000/    finn)   ❷ Gid: ( 1000/    finn)
                  ...          ...          ...          ...
```

Назначаются владельцы файлов при их создании. По умолчанию пользователем-владельцем файла становится пользователь, создавший файл, а группой-владельцем файла становится его первичная группа (листинг 3.32).

Листинг 3.32. Назначение владельцев файлов при создании

```
finn@ubuntu:~$ id
uid=1000(finn) gid=1000(finn) группы=1000(finn),4(adm),24(cdrom),...,131(sambashare)
finn@ubuntu:~$ nano README
finn@ubuntu:~$ ls -la README
-rw-rw-r-- 1 finn❶ finn❷ 3 ноя 18 01:20 README
```

Изменить (листинг 3.33) пользователя-владельца ❶ файлов может только суперпользователь ① **root** при помощи команды **chown(1)**, а группу-владельца — владелец файла ❷ при помощи команды **chgrp(1)**, но только на ту ❸ к которой он сам принадлежит.

Листинг 3.33. Смена владельцев файлов

```
finn@ubuntu:~$ ls -la README
-rw-rw-r-- 1 finn finn 3 ноя 18 01:20 README
finn@ubuntu:~$ chown jake README
❶ chown: изменение владельца 'README': Операция не позволена
finn@ubuntu:~$ id
uid=1000(finn) gid=1000(finn) группы=1000(finn),4(adm),24(cdrom),...,131(sambashare)
❷ finn@ubuntu:~$ chgrp adm README
finn@ubuntu:~$ ls -la README
-rw-rw-r-- 1 finn adm 3 ноя 18 01:20 README
finn@ubuntu:~$ chgrp daemon README
❸ chgrp: изменение группы для 'README': Операция не позволена
① finn@ubuntu:~$ sudo chown jake README
```

```
[sudo] password for user: <пароль finn'а>
finn@ubuntu:~$ ls -la README
-rw-rw-r-- 1 jake admin 3 ноя 18 01:20 README
```

В примере из листинга 3.16 стоит обратить внимание на владельцев специальных файлов устройств виртуальных терминалов — ими назначены пользователи, совершившие вход в систему. Специальный файл устройства терминалов, на которых вход не осуществлен, принадлежит суперпользователю **root**, например как в случае с терминалом **/dev/tty9**. Такие назначения естественным образом делаются при входе пользователей в систему и для того, чтобы они могли распоряжаться правами устройства терминала, через который осуществили вход (см. листинг 3.41).

3.5.2. Базовые права доступа и дополнительные атрибуты

Для разграничения действий над файлами определены три базовых права доступа (базовые разрешения): чтение **r** — «read», запись **w** — «write» и выполнение **x** — «execute», соответствующие разрешению выполнять системные вызовы **read(2)**[1], **write(2)** и **execve(2)**[2]. Каждое базовое право назначается на файл тому или иному пользователю или группе, разрешая соответствующую операцию.

В наследии классической UNIX определены только три субъекта (листинг 3.34), которым назначаются базовые права — пользователь-владелец (owner) ❶, группа-владелец (group owner) ❷ и все остальные (others) ❸. Совокупность их базовых прав называется *режимом доступа* (access mode) к файлу.

Базовое право может быть назначено **r**, **w** или **x** или отозвано **-**, поэтому в метаданных файла представляется одним битом, а для режима доступа требуется девять бит: по три бита прав на каждый из трех субъектов доступа. Компактно режим доступа может быть записан соответствующим числом в восьмеричной системе счисления **rw-r--r--** ≡ 110100100_2 ≡ 644_8.

Листинг 3.34. Режим доступа к файлу

```
finn@ubuntu:~$ stat .profile
  Файл: «.profile»
                ...          ...          ...          ...
            ❶↓    ↓❸
Доступ: (0644/-rw- r-- r--)  Uid: ( 1000/    finn)   Gid: ( 1000/    finn)
                 ❷↑
                ...          ...          ...          ...
```

[1] Точнее, системному вызову **open(2)** с флагами **O_RDONLY** и **O_WRONLY**, но для простоты можно считать **r** — **read(2)**, а **w** — **write(2)**.

[2] Подробнее о системном вызове **execve(2)** *см. разд. 4.3.*

```
finn@ubuntu:~$ ls -l .profile
-rw-r--r-- 1 finn finn 677 ноя 13 00:25 .profile
```

Проверка режима доступа (листинг 3.35) при операциях с файлами проверяется «слева направо» до первого совпадения. Если пользователь, осуществляющий операцию с файлом, является его владельцем, тогда используются *только* ❶ права владельца. В противном случае проверяется членство пользователя, осуществляющего операцию с файлом, в группе-владельцев файла, и тогда используются *только* ❷ права группы-владельцев. В других случаях используются права для всех остальных ❸, а для суперпользователя **root** вообще никакие проверки не осуществляются.

Листинг 3.35. Использование режима доступа к файлу

```
finn@ubuntu:~$ id finn
uid=1000(finn) gid=1000(finn) группы=1000(finn),4(adm),24(cdrom),...,131(sambashare)
finn@ubuntu:~$ ls -l README.*
❶ -rw-r--r-- 1 finn        adm       2471 окт.  11 01:12 README.finn
  --w-r--r-- 1 finn        adm       2471 окт.  11 01:12 README.finn.locked
❷ -rw-r--r-- 1 jake        adm        776 окт.  11 01:12 README.jake
  -rw----r-- 1 bubblegum   adm        171 окт.  11 01:12 README.bubblegum
❸ -rw-r--r-- 1 marceline   marceline   16 окт.  11 01:12 README.marceline
  -rw-r----- 1 iceking     iceking     31 окт.  11 01:12 README.iceking
finn@ubuntu:~$ wc README.*
wc: README.bubblegum: Отказано в доступе
  24  177 2471 README.finn
wc: README.finn.locked: Отказано в доступе
wc: README.iceking: Отказано в доступе
  12   70  776 README.jake
   1    1   16 README.marceline
  37  248 3263 итого
```

Режим доступа новых файлов

Назначается режим доступа файлов при их создании программой, создавшей файл, исходя из назначения файла, но с учетом пожеланий (точнее, нежеланий) пользователя. Так, например, текстовые редакторы назначают создаваемым (текстовым) файлам права **rw** для всех субъектов, а компиляторы назначают создаваемым (программным) файлам права **rwx** для всех субъектов. Пользователь может выразить свое нежелание назначать вновь создаваемым файлам те или иные права доступа для тех или иных субъектов, установив так называемую реверсивную (т. е. обрат-

ную, символизирующую НЕжелание) маску доступа при помощи встроенной команды интерпретатора umask (листинг 3.36).

Листинг 3.36. Реверсивная маска доступа

```
finn@ubuntu:~$ umask
0002
finn@ubuntu:~$ umask -S
u=rwx,g=rwx,o=rx
finn@ubuntu:~$ touch common.jnl
finn@ubuntu:~$ ls -l common.jnl
-rw-rw-r-- 1 finn finn 0 ноя 18 01:37 common.jnl
finn@ubuntu:~$ umask g-w,o-rwx
finn@ubuntu:~$ umask
0027
finn@ubuntu:~$ umask -S
u=rwx,g=rx,o=
finn@ubuntu:~$ touch group.jnl
finn@ubuntu:~$ ls -l group.jnl
-rw-r----- 1 finn finn 0 ноя 18 01:37 group.jnl
finn@ubuntu:~$ umask g=
finn@ubuntu:~$ umask
0077
finn@ubuntu:~$ umask -S
u=rwx,g=,o=
finn@ubuntu:~$ touch private.jnl
finn@ubuntu:~$ ls -l private.jnl
-rw------- 1 finn finn 0 ноя 18 01:38 private.jnl
```

Изменять режима доступа разрешено непосредственному пользователю — владельцу файла, но не членами группы-владельцев, что иллюстрирует листинг 3.37 при помощи команды chmod(1).

Листинг 3.37. Изменение режима доступа к файлу

```
finn@ubuntu:~$ id finn
uid=1000(finn) gid=1000(finn) группы=1000(finn),4(adm),24(cdrom),...,131(sambashare)
finn@ubuntu:~$ ls -l README.*
-rw-r--r-- 1 finn        adm     2471 окт.  11 01:12 README.finn
-rw-r--r-- 1 jake        adm      776 окт.  11 01:12 README.jake
                  ...            ...              ...              ...
```

```
finn@ubuntu:~$ chmod g-r,o-r README.finn
finn@ubuntu:~$ ls -la README.finn
-rw------- 1 finn        admin     2471 окт.  11 01:12 README.finn
finn@ubuntu:~$ chmod g-r,o-r README.jake
chmod: изменение прав доступа для 'README.jake': Операция не позволена
```

Семантика режима доступа разных типов файлов

Права доступа `r`, `w`, `x` для обычных файлов представляются чем-то интуитивно понятным, но для других типов файлов это не совсем так. Например, каталог (см. рис. 3.2) содержит *список имен* файлов, поэтому право `w` для каталога — это право записи в этот список и право стирания из этого списка, что трансформируется в право *удаления* файлов из каталога и *создания* файлов в каталоге. Аналогично, право `r` для каталога — это право *просмотра списка имен* его файлов. И наконец, право `x` для каталога является правом *прохода* в каталог, т. е. позволяет обращаться к файлам внутри каталога по их имени (листинг 3.38).

Листинг 3.38. Права доступа к каталогу

```
finn@ubuntu:~$ mkdir folder
finn@ubuntu:~$ ls -lad folder/
drwxrwxr-x 2 finn finn 4096 окт.  12 00:37 folder/
finn@ubuntu:~$ cp /etc/magic folder
finn@ubuntu:~$ chmod u-w folder
finn@ubuntu:~$ ls -lad folder/
dr-xrwxr-x 2 finn finn 4096 окт.  12 00:40 folder/
finn@ubuntu:~$ cp /etc/localtime folder/
cp: невозможно создать обычный файл «folder/localtime»: Отказано в доступе
finn@ubuntu:~$ rm folder/magic
rm: невозможно удалить «folder/magic»: Отказано в доступе
finn@ubuntu:~$ ls -li folder/
итого 4
20318203 -rw-r--r-- 1 finn finn 111 окт.  12 00:40 magic
finn@ubuntu:~$ chmod u-r folder
finn@ubuntu:~$ ls -lad folder/
d--xrwxr-x 2 finn finn 4096 окт.  12 00:40 folder/
finn@ubuntu:~$ ls -li folder/
ls: невозможно открыть каталог folder/: Отказано в доступе
finn@ubuntu:~$ ls -li folder/magic
! 20318203 -rw-r--r-- 1 finn finn 111 окт.  12 00:40 folder/magic
finn@ubuntu:~$ chmod u-x folder/
```

```
finn@ubuntu:~$ ls -lad folder/
d---rwxr-x 2 finn finn 4096 окт.  12 00:40 folder/
finn@ubuntu:~$ ls -li folder/magic
ls: невозможно получить доступ к folder/magic: Отказано в доступе
finn@ubuntu:~$ chmod u+rw folder/
finn@ubuntu:~$ ls -lad folder/
drw-rwxr-x 2 finn finn 4096 окт.  12 00:40 folder/
finn@ubuntu:~$ cp /etc/localtime folder/
cp: не удалось выполнить stat для «folder/localtime»: Отказано в доступе
finn@ubuntu:~$ rm folder/magic
rm: невозможно удалить «folder/magic»: Отказано в доступе
finn@ubuntu:~$ ls -l folder/
ls: невозможно получить доступ к folder/magic: Отказано в доступе
итого 0
-????????? ? ? ? ?            ? magic
finn@ubuntu:~$ chmod u=rwx folder/
finn@ubuntu:~$ ls -ld folder/
drwxrwxr-x 2 finn finn 4096 окт.  12 00:40 folder/
finn@ubuntu:~$ cd folder/
finn@ubuntu:~/folder$ ls -l
итого 4
-rw-r--r-- 1 finn finn  111 окт.  12 00:40 magic
finn@ubuntu:~/folder$ chmod a= magic
finn@ubuntu:~/folder$ ls -l magic
---------- 1 finn finn 111 окт.  12 00:40 magic
finn@ubuntu:~/folder$ rm magic
rm: удалить защищенный от записи обычный файл «magic»? y
finn@ubuntu:~/folder$ ls -l
итого 0
```

Для жестких ссылок права доступа не существуют вовсе — они просто являются теми же правами, что и права целевого файла, в силу того что права доступа хранятся в метаданных. Для символических ссылок семантика прав сохранена такой же, как и у жестких ссылок, с тем лишь различием, что права символических ссылок существуют отдельно от целевых файлов, но никогда не проверяются (см. **symlink(7)**). Для изменения прав доступа самих символических ссылок даже не существует специальной команды — при использовании **chmod(1)** со ссылкой всегда будут изменяться права целевого файла (листинг 3.39).

Листинг 3.39. Права доступа ссылок

```
finn@ubuntu:~$ ls -l README.finn
-rwxr--r-- 1 finn finn 2471 окт.  11 01:13 README.finn
finn@ubuntu:~$ ln README.finn read.me
finn@ubuntu:~$ ln -s README.finn readme.1st
finn@ubuntu:~$ ls -l README.finn read.me readme.1st
-rwxr--r-- 2 finn finn 2471 окт.  11 01:13 read.me
lrwxrwxrwx 1 finn finn   11 окт.  12 01:19 readme.1st -> README.finn
-rwxr--r-- 2 finn finn 2471 окт.  11 01:13 README.finn
finn@ubuntu:~$ chmod g+w read.me
finn@ubuntu:~$ ls -l README.finn read.me readme.1st
-rwxrw-r-- 2 finn finn 2471 окт.  11 01:13 read.me
lrwxrwxrwx 1 finn finn 11 окт.  12 01:19 readme.1st -> README.finn
-rwxrw-r-- 2 finn finn 2471 окт.  11 01:13 README.finn
finn@ubuntu:~$ chmod o-r readme.1st
finn@ubuntu:~$ ls -l README.finn read.me readme.1st
-rwxrw---- 2 finn  finn  2471 окт.  11 01:13 read.me
lrwxrwxrwx 1 finn  finn  11 окт.  12 01:19 readme.1st -> README.finn
-rwxrw---- 2 finn  finn  2471 окт.  11 01:13 README.finn
```

Для специальных файлов устройств, именованных каналов и сокетов право **x** не определено, а права **r** и **w** стоит воспринимать как права ввода и вывода информации на устройство и как права передачи и приема информации через средство взаимодействия.

Дополнительные атрибуты

Помимо базовых прав доступа **r**, **w** и **x**, для решения отдельных задач разграничения доступа используют дополнительные атрибуты **s**, Set user/group ID (SUID Set User ID или SGID, Set Group ID) — атрибут неявного делегирования полномочий и **t**, sTicky — «липучка», атрибут ограниченного удаления.

Типичной задачей, требующей неявного делегирования полномочий, является проблема невозможности изменения пользователями свойств своих учетных записей, которые хранятся в двух файлах-таблицах — **passwd(5)** и **shadow(5)**, доступных на запись ❶ (и чтение ❷) только суперпользователю **root**. Однако (листинг 3.40) команды **passwd(1)**, **chsh(1)** и **chfn(1)**, будучи запущены обычным пользователем, прекрасно изменяют (!) пароль в таблице **/etc/shadow** и свойства пользовательской записи в таблице **/etc/passwd** за счет передачи полномочий ❸ *пользователя* — *владельца* программы тому пользователю, который ее запускает.

Листинг 3.40. Дополнительный атрибут SUID

```
finn@ubuntu:~$ ls -la /etc/passwd /etc/shadow
❶ -rw-r--r-- 1 root root   2867 ноя 17 11:16 /etc/passwd
❷ -rw-r----- 1 root shadow 1617 ноя 17 11:17 /etc/shadow
finn@ubuntu:~$ passwd
Смена пароля для finn.
(текущий) пароль UNIX:
Введите новый пароль UNIX:
Повторите ввод нового пароля UNIX:
passwd: пароль успешно обновлён
finn@ubuntu:~$ ls -la /etc/passwd /etc/shadow
-rw-r--r-- 1 root root   2867 ноя 17 11:16    /etc/passwd
-rw-r----- 1 root shadow 1617 ноя 18 01:46 ➥ /etc/shadow
finn@ubuntu:~$ chfn
Пароль:
Изменение информации о пользователе finn
Введите новое значение или нажмите ENTER для выбора значения по умолчанию
        Полное имя:
        Номер комнаты []:
        Рабочий телефон []: +7(812)703-02-02
        Домашний телефон []:
finn@ubuntu:~$ ls -l /etc/passwd /etc/shadow
-rw-r--r-- 1 root root   2883 ноя 18 01:47 ➥ /etc/passwd
-rw-r----- 1 root shadow 1617 ноя 18 01:46    /etc/shadow
finn@ubuntu:~$ ls -la /usr/bin/passwd /usr/bin/chfn
❸ -rwsr-xr-x 1 root root 84848 авг 29 16:00 /usr/bin/chfn
-rwsr-xr-x 1 root root 67992 авг 29 16:00 /usr/bin/passwd
```

За счет использования атрибута SUID получается, что пользователям, запускающим программы **chfn(1)**, **chsh(1)** и **passwd(1)**, для их исполнения временно делегируются права владельца этих программ (суперпользователя **root**) так, как будто сам суперпользователь их запустил.

Листинг 3.41. Дополнительный атрибут SGID

Сеанс finn — tty1

```
finn@ubuntu:~$ w
 00:03:53 up 12 days, 13:53,  7 users,  load average: 0,53, 0,51, 0,91
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
jake     tty2                      00:03    9.00s  0.52s  0.43s -bash
finn     tty1                      00:03   17.00s  0.51s  0.45s -bash
```

```
finn@ubuntu:~$ ls -l /dev/tty1 /dev/tty2
crw------- 1 finn tty 4, 1 окт.  20 00:03 /dev/tty1
crw------- 1 jake tty 4, 2 окт.  20 00:03 /dev/tty2
finn@ubuntu:~$ write jake
write: jake has messages disabled
```

```
Сеанс jake                                                    tty2
jake@ubuntu:~$ mesg y
```

```
finn@ubuntu:~$ ls -l /dev/tty2
crw--w---- 1 jake tty 4, 2 окт.  20 00:07 /dev/tty2
finn@ubuntu:~$ write jake
write: write: you have write permission turned off.


Hi, buddy, wazzzup?
^D
```

```
Сеанс jake                                                    tty2
jake@ubuntu:~$
Message from finn@ubuntu on tty1 at 00:10 ...
Hey buddy, wazzup?
EOF
```

```
finn@ubuntu:~$ ls -Ll /usr/bin/write
-rwxr-sr-x 1 root tty 14328 мая  3  2018 /usr/bin/write
```

Аналогично (см. листинг 3.41) при использовании атрибута SGID, при передаче сообщений от пользователя к пользователю командой **write(1)** или **wall(1)**, запускающему эти программы пользователю делегируются полномочия группы **tty**, имеющей доступ на запись к терминалам (специальным файлам устройств **/dev/ttyN**), владельцы которых разрешили такой доступ.

Именно за счет механизма SUID/SGID различные команды позволяют обычным, непривилегированным пользователям, выполнять сугубо суперпользовательские действия. Так, например, **su(1)** и **sudo(1)** позволяют выполнять команды одним пользователям от лица других пользователей, **mount(8)**, **umount(8)** и **fusermount(1)** — монтировать и размонтировать файловые системы, **ping(8)** и **traceroute(1)** — выполнять диагностику сетевого взаимодействия, **at(1)** и **crontab(1)** — сохранять в «системных» каталогах отложенные и периодические задания, и т. д.

Однако для каталогов атрибут SGID имеет совсем другой смысл. По умолчанию владельцем файла становится ❷ тот пользователь (и его первичная группа), который запустил программу, создавшую файл. Но для файлов, создаваемых в «об-

щих» ❶ для какой-то группы пользователей, каталогах, логичнее было бы назначать группой-владельцем создаваемых файлов эту общую группу ❸.

Листинг 3.42. Дополнительный атрибут SGID для каталога

```
bubblegum@ubuntu:~$ cd /srv/kingdom
bubblegum@ubuntu:/srv$ id
uid=1005(bubblegum) gid=1005(bubblegum) группы=1005(bubblegum),1007(candy)
bubblegum@ubuntu:/srv/kingdom$ ls -ld .
drwxr-xr-x 2 bubblegum bubblegum 4096 окт.  21 22:02 .
bubblegum@ubuntu:/srv/kingdom$ touch bananaguard1
bubblegum@ubuntu:/srv/kingdom$ ls -l
итого 0
-rw-rw-r-- 1 bubblegum bubblegum 0 окт.  21 22:02 bananaguard1
bubblegum@ubuntu:/srv/kingdom$ chgrp candy .
bubblegum@ubuntu:/srv/kingdom$ chmod g+ws .
bubblegum@ubuntu:/srv/kingdom$ ls -ld .
drwxrwsr-x 2 bubblegum candy 4096 окт.  21 22:02 .
                    ...          ...          ...          ...
finn@ubuntu:/srv/kingdom$ id
uid=1001(finn) gid=1001(finn) группы=1001(finn),1007(candy)
finn@ubuntu:/srv/kingdom$ touch bananaguard2
finn@ubuntu:/srv/kingdom$ ls -l
итого 0
-rw-rw-r-- 1 bubblegum bubblegum 0 окт.  21 22:02 bananaguard1
❸ -rw-rw-r-- 1 finn     candy     0 окт.  21 22:02 bananaguard2
```

В примере из листинга 3.42 за счет SGID-атрибута каталога владельцем всех файлов, помещаемых в этот каталог, автоматически назначается группа-владелец самого каталога, а создатель (владелец) файла может теперь назначать нужные права доступа для всех членов *этой* группы к *своему* файлу — либо неявно при помощи реверсивной маски доступа, либо явно при помощи команды **chmod(1)**.

Атрибут-«липучка» **t** (sTicky) служит для ограничения действия базового разрешения **w** записи в каталоге. Например, временный каталог **/tmp** предназначается для хранения временных файлов любых пользователей и поэтому доступен на запись всем пользователям. Однако право записи в каталог дает возможность не только создавать в нем новые файлы, но и удалять *любые* существующие файлы (любых пользователей), что совсем не кажется логичным. Именно атрибут **t** ограничивает возможность удалять чужие файлы, т. е. файлы, не принадлежащие пользователю, пытающемуся их удалить.

Листинг 3.43. Дополнительный атрибут sTicky для каталога

```
finn@ubuntu:/srv/kingdom$ id
uid=1001(finn) gid=1001(finn) группы=1001(finn),1007(candy)
finn@ubuntu:/srv/kingdom$ ls -la
итого 8
drwxrwsr-x 2 bubblegum candy  4096 окт.  23 20:57 .
drwxr-xr-x 3 root      root   4096 окт.  21 21:57 ..
-rw-rw-r-- 1 bubblegum bubblegum 0 окт.  21 23:15 bananaguard1
-rw-rw-r-- 1 finn      candy     0 окт.  21 23:24 bananaguard2
                  ...          ...           ...           ...
finn@ubuntu:/srv/kingdom$ rm bananaguard1
rm: удалить защищенный от записи пустой обычный файл «bananaguard1»? y
finn@ubuntu:/srv/kingdom$ ls -l
итого 0
-rw-rw-r-- 1 finn candy 0 окт.  21 23:24 bananaguard2
                  ...          ...           ...           ...
bubblegum@ubuntu:/srv/kingdom$ chmod +t .
bubblegum@ubuntu:/srv/kingdom$ touch bananaguard1
bubblegum@ubuntu:/srv/kingdom$ ls -la
итого 8
drwxrwsr-t 2 bubblegum candy 4096 окт.  23 21:19 .
drwxr-xr-x 3 root      root  4096 окт.  21 21:57 ..
-rw-rw-r-- 1 bubblegum candy    0 окт.  23 21:19 bananaguard1
-rw-rw-r-- 1 finn      candy    0 окт.  23 21:19 bananaguard2
                  ...          ...           ...
finn@ubuntu:/srv/kingdom$ rm bananaguard1
rm: невозможно удалить «bananaguard1»: Операция не позволена
```

3.5.3. Списки контроля доступа POSIX

Режим доступа к файлу (access mode), определяющий базовые разрешения **r**, **w** и **x** только для трех субъектов доступа (владельца, группы-владельца и всех остальных), не является достаточно гибким и удобным инструментом разграничения доступа. *Списки* контроля доступа (**W**:[**ACL**], **a**ccess **c**ontrol **l**ists), согласно стандарту POSIX.1e, расширяют классический *режим* доступа к файлу дополнительными записями (рис. 3.4), определяющими права доступа для явно указанных пользователей и групп. Для просмотра и модификации записей в списках доступа используются утилиты **getfacl(1)** и **setfacl(1)**, соответственно.

В примере из листинга 3.44 для всех «остальных» (не входящих в группу **candy**) пользователей отзываются все права на каталог **/srv/kingdom/stash**, но для отдельного пользователя **jake** (не являющегося членом группы **candy**) назначаются права чтения, модификации и прохода в него **rwx**.

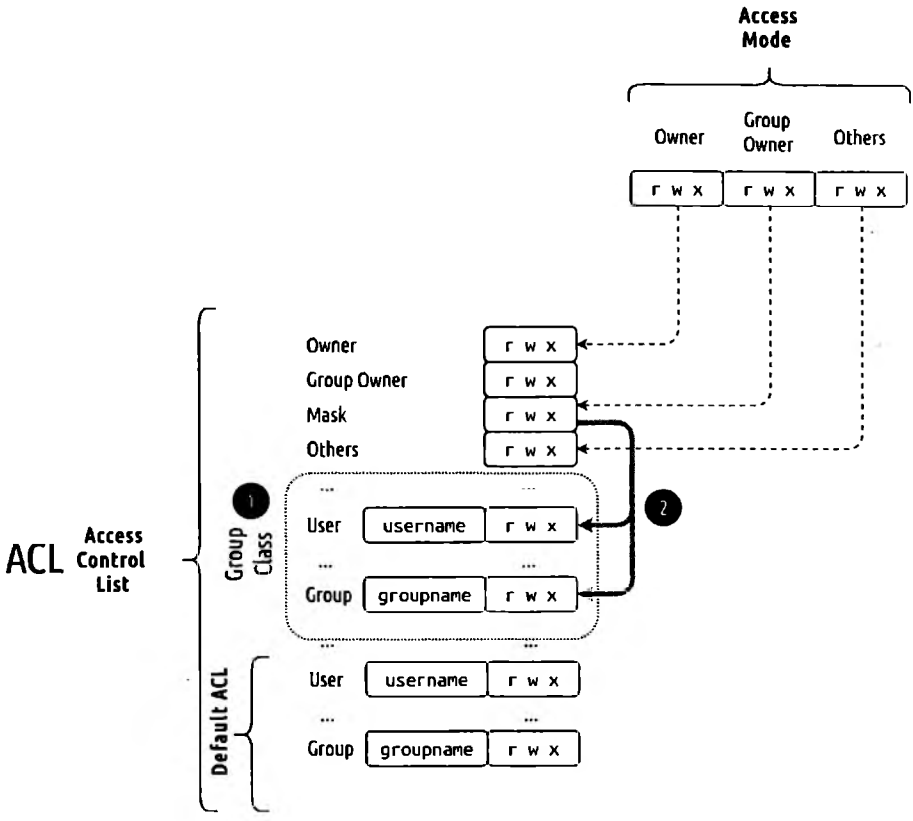


Рис. 3.4. Списки контроля доступа к файлам

Листинг 3.44. Списки контроля доступа

```
finn@ubuntu:/srv/kingdom$ ls -lad .
drwxrwsr-t 2 bubblegum candy 4096 окт.  23 21:19 .
finn@ubuntu:/srv/kingdom$ id
uid=1001(finn) gid=1001(finn) группы=1001(finn),1007(candy)
finn@ubuntu:/srv/kingdom$ mkdir stash
finn@ubuntu:/srv/kingdom$ chmod o= stash/
finn@ubuntu:/srv/kingdom$ ls -lad stash/
drwxrws--- 2 finn candy 4096 нояб.  4 13:05 stash/
finn@ubuntu:/srv/kingdom$ id jake
? uid=1002(jake) gid=1002(jake) группы=1002(jake)
finn@ubuntu:/srv/kingdom$ setfacl -m u:jake:rwx stash/
finn@ubuntu:/srv/kingdom$ ls -lad stash/
drwxrws---+ 2 finn candy 4096 нояб.  4 13:05 stash/
finn@ubuntu:/srv/kingdom$ getfacl stash/
...
```

```
user::rwx
user:jake:rwx
group::rwx
mask::rwx
other::---
```

Групповая маска

С расширением множеств субъектов, для которых определены права доступа при помощи списков контроля доступа, возникает вопрос об их смысловой совместимости с режимом доступа, в котором определены всего три множества: пользователь-владелец, группа-владелец и все остальные.

Программы, работающие в соответствии с режимом доступа, считают, что если и существует некоторое количество «выделенных» субъектов со специально определенными правами, отличными от «всех остальных», то *все* эти субъекты входят в группу владельцев. Списки контроля доступа позволяют «выделять» субъектов из числа любых пользователей и групп и определять их права произвольным образом. Когда программа, работающая в соответствии с режимом доступа, назначает права группе-владельцу, она вправе считать, что *все* «выделенные» субъекты будут ограничены этими правами. Именно поэтому в список контроля доступа добавлена групповая *маска* прав, определяющая ограничения «выделенных» субъектов (называемых групповым *классом* субъектов) в их индивидуальных правах.

В примере из листинга 3.45 каталогу, которому определены индивидуальные права **rwx** для пользователя **jake** в списке контроля доступа, изменяют права группы-владельцев ❶ классического режима доступа. Получившийся *режим* доступа ❶ означает, что *никому*, кроме владельца файла, не разрешено записывать в этот файл, что противоречит индивидуальным правам *списка* доступа. Противоречия устраняются маской ❷ *списка* доступа, ограничивающей эффективные права пользователя **jake** так, чтобы это соответствовало *режиму* доступа *по смыслу*.

Листинг 3.45. Групповая маска ACL

```
finn@ubuntu:/srv/kingdom$ ls -lad stash/
drwxrws---+ 2 finn candy 4096 нояб.  4 13:05 stash/
finn@ubuntu:/srv/kingdom$ getfacl stash/
...          ...              ...
user::rwx
user:jake:rwx
group::rwx
mask::rwx
other::---
```

```
❶ finn@ubuntu:/srv/kingdom$ chmod g-w stash/
  finn@ubuntu:/srv/kingdom$ ls -lad stash/
❷ drwxr-s---+ 2 finn candy 4096 нояб.  4 13:05 stash/
  finn@ubuntu:/srv/kingdom$ getfacl stash/
  ...          ...          ...          ...          ...
  user::rwx
! user:jake:rwx              #effective:r-x
! group::rwx                 #effective:r-x
❸ mask::r-x
  other::---
```

Права по умолчанию

При создании новых файлов в каталогах с индивидуальными правами пользователей в списках доступа зачастую складывается ситуация, когда пользователи (имеющие доступ в каталоги) не получают нужных прав доступа к создаваемым файлам в этих каталогах. В большинстве случаев это противоречит здравому смыслу, т. к. все файлы некоторого каталога являются в определенном смысле «общими» для множества пользователей, которым разрешен доступ в сам каталог.

В примере из листинга 3.46 в каталоге **stash**, куда пользователю **jake** предоставлен индивидуальный доступ (см. листинг 3.45) при создании файла **README**, он в силу SGID для каталога (см. листинг 3.42) передается группе **candy**. В группу **candy** пользователь **jake** не входит (именно поэтому ему назначены индивидуальные права в листинге 3.44), в результате чего файл ему никак не будет доступен.

Проблема решается назначением ❷ каталогу **stash** прав доступа «по умолчанию» (default), которые будут унаследованы ❸ файлами, создающимися в этом каталоге.

Листинг 3.46. Права по умолчанию

```
  finn@ubuntu:/srv/kingdom$ cd stash/
  finn@ubuntu:/srv/kingdom$ umask 0007
  finn@ubuntu:/srv/kingdom/stash$ touch README
  finn@ubuntu:/srv/kingdom/stash$ ls -la README
  -rw-rw---- 1 finn      candy 0 нояб.  4 14:16 README
  finn@ubuntu:/srv/kingdom/stash$ id jake
❶ uid=1002(jake) gid=1002(jake) группы=1002(jake)
❷ finn@ubuntu:/srv/kingdom/stash$ setfacl -m d:u:jake:rw .
  finn@ubuntu:/srv/kingdom/stash$ getfacl .
  # file: .
  ...          ...                                    ...
  default:user::rwx
```

```
default:user:jake:rw-
default:group::rwx
default:mask::rwx
default:other::---
finn@ubuntu:/srv/kingdom/stash$ touch README.jake
finn@ubuntu:/srv/kingdom/stash$ ls -l README.jake
-rw-rw----+ 1 finn      candy    0 нояб.  4 14:17 README.jake
finn@ubuntu:/srv/kingdom/stash$ getfacl README.jake
# file: README.jake
...          ...              ...
user:jake:rw-
...                                    ...              ...
```

3.6. Мандатное (принудительное) разграничение доступа

В Linux дискреционные механизмы разграничения доступа (DAC, discretionary access control) являются основными и всегда активны. Их использование предполагает, что владельцы объектов правильно распоряжаются правами доступа к находящимся в их владении объектам. Например, пользовательские закрытые ключи, используемые службой W:[SSH] *(см. разд. 6.4.1)* в каталоге **~/.ssh** или ключи W:[GnuPG] в каталоге **~/.gnupg** и прочие секретные данные (подобные ключи доступа в банковские информационные системы), должны быть *недоступны* никому, кроме их владельца.

Запускаемые пользователем программы выполняются от лица запустившего их пользователя и имеют доступ к файлам согласно установленным режимам или спискам доступа. В примере из листинга 3.47 клиент **ssh(1)**, браузер **firefox(1)** и коммуникатор **skype** имеют абсолютно равные возможности по чтению и модификации (!) пользовательского закрытого ключа **~/.ssh/id_rsa**, тогда как настоящим «владельцем» ключей является только **ssh(1)**.

Листинг 3.47. Необходимость MAC

```
finn@ubuntu:~$ ls -l .ssh
итого 8
-rw------- 1 finn finn 1675 нояб.  4 16:06 id_rsa
-rw-r--r-- 1 finn finn  393 нояб.  4 16:06 id_rsa.pub

finn@ubuntu:~$ ps fux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
finn     20650  0.0  0.1  13488  8576 pts/0    S    16:08   0:00 -bash
finn     20943  0.0  0.0   6668  2284 pts/0    S+   16:19   0:00  \_ ssh jake@grex.org
                ...                 ...
```

```
finn     21094 13.1  1.5 592676 127820 pts/2   Rl+  16:27   0:09 /usr/lib/firefox/firefox
...        ...                  ...
finn     21790 29.7  2.1 575000 172084 ?       Sl   16:38   0:09 skype
```

Абсолютно естественно предполагать, что программы **firefox(1)** и **skype(1)** не имеют никаких *намерений* доступа к пользовательским ключам SSH. Можно даже *доверять* программе **firefox(1)**, штатно установленной из доверенного источника (дистрибутива), где она была изготовлена из открытых *исходных* текстов, подлежащих верификации. Однако нет никаких оснований доверять закрытому **skype**, поставляемому в бинарном виде.

Более того, предоставлять доступ программам **firefox(1)** и **skype(1)** к SSH-ключам пользователя нет никакой необходимости, во-первых, просто потому, что это выходит за рамки набора минимально необходимых условий их *целевого* функционирования. Во-вторых, практически в любой программе есть ошибки, используя которые злоумышленник может осуществлять *непреднамеренные* действия в свою пользу. Таким воздействиям особенно подвержены программы, использующие сетевой обмен с *недоверенной* внешней средой — клиенты и серверы сетевых служб операционной системы. Тем временем дискреционный подход и механизмы служат для разграничения доступа разных пользователей к файлам, но никак не предназначены для разграничения доступа программ одного и того же пользователя к разным файлам этого пользователя.

Для разграничения доступа *субъектов* — программ к *объектам* — файлам дерева каталогов используют так называемый *мандатный* (от англ. *mandatory* — обязательный или принудительный) подход (MAC, **m**andotary **a**ccess **c**ontrol), предполагающий следование обязательным *правилам доступа* к файлам, назначаемым *администраторами* системы. Правила доступа строятся на основе знания о внутреннем устройстве программ и представляют собой описание набора минимально необходимых условий их *целевого* функционирования.

Таким образом, в мандатных правилах, ограничивающих доступ к SSH-ключам пользователя, только программе **ssh(1)** должен быть *разрешен* доступ для непосредственного выполнения своих прямых функций, а программам **firefox(1)** и **skype(1)** в доступе к SSH-ключам должно быть *отказано*.

3.6.1. Модуль принудительного разграничения доступа AppArmor

В Linux мандатные механизмы разграничения являются дополнительными и активируются по желанию пользователя или дистрибьютора. Так, например, в Ubuntu Linux по умолчанию устанавливается и активируется модуль принудительного разграничения доступа **apparmor(7)**. Модуль **W:[AppArmor]** имеет некоторое количество готовых к употреблению наборов мандатных правил (называемых *профилями*) для ограничения *(confine)* доступа субъектов — программ к объектам операционной

системы — файлам, сетевым протоколам, портам TCP/UDP и пр. Правила AppArmor идентифицируют программы и файлы на основе их полных путевых имен. При этом каждый профиль описывает ограничения для одной определенной запускаемой программы (а также для других, запускаемых этой программой, т. е. «подчиненных» программ), а программы, для которых профили не определены, никак не ограничиваются *(uniconfined)*.

В примере из листинга 3.48 проиллюстрировано использование команды **aa-status(8)**, показывающей статус модуля принудительного контроля доступа AppArmor. В распоряжении модуля имеются профиль ❶ программы **skype** в режиме принуждения (**enforce**) и профиль ❷ программы **firefox(1)** в режиме оповещения (**complain**). Более того, процесс **skype** с идентификатором **30173** принуждается (**enforce**) модулем контроля к выполнению правил профиля, а процесс **firefox** с идентификатором **10335** только отслеживается (**complain**) на предмет нарушений правил.

Листинг 3.48. Модуль мандатного разграничения доступа AppArmor

```
finn@ubuntu:~$ sudo aa-status
apparmor module is loaded.
50 profiles are loaded.
26 profiles are in enforce mode.
❶    /usr/bin/skype
                    ...          ...          ...          ...
24 profiles are in complain mode.
                    ...          ...          ...          ...
❷    /usr/lib/firefox/firefox{,*[^s][^h]}
                    ...          ...          ...          ...
18 processes have profiles defined.
3 processes are in enforce mode.
                    ...          ...          ...          ...
🔒   /usr/bin/skype (30173)
6 processes are in complain mode.
🔓   /usr/lib/firefox/firefox{,*[^s][^h]} (10335)
                    ...          ...          ...          ...
0 processes are unconfined but have a profile defined.
```

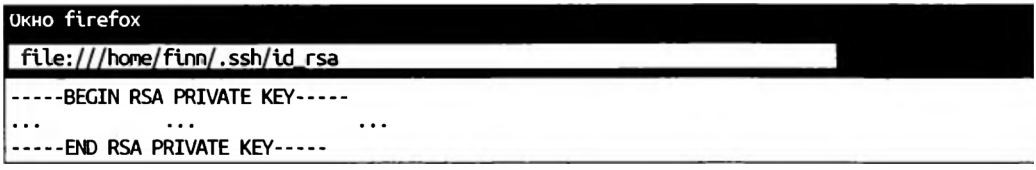
Нарушения мандатных правил процессами, находящимися в режиме оповещения (**complain**), приводят только к журнализации сообщений аудита об обнаруженных нарушениях. Попытки нарушения мандатных правил процессами, находящимися в режиме принуждения (**enforce**), пресекаются модулем контроля доступа в виде отказа в доступе к тому или иному запрашиваемому объекту.

Выполненный (листинг 3.49) при помощи команды **apparmor_parser(8)** анализ полного (**-p**) набора правил профиля программы **/usr/bin/firefox** показывает, что обращения к любым файлам каталога **.ssh** явно запрещены указанием **deny** и подлежат обязательному аудиту согласно указанию **audit**. При помощи команды **aa-enforce(8)**

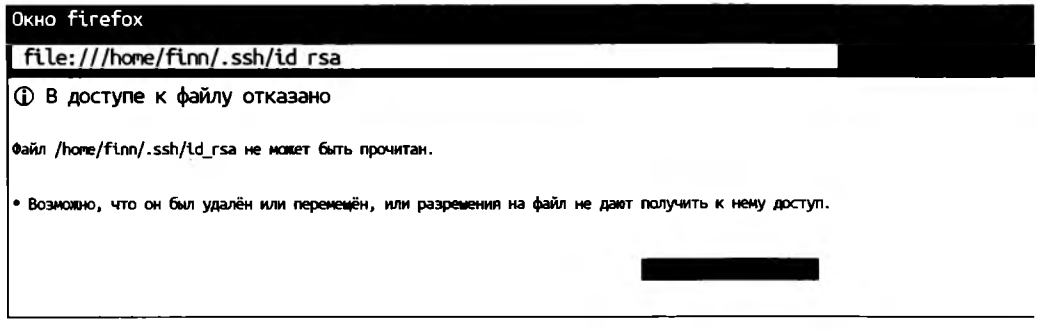
профиль переводится в режим **enforce**, в результате чего доступ **firefox(1)** к ключам пользователя оказывается запрещенным.

Листинг 3.49. Мандатные правила профиля firefox

```
finn@ubuntu:~$ apparmor_parser -p /etc/apparmor.d/usr.bin.firefox
...          ...              ...            ...              ...
  audit deny @{HOME}/.ssh/** mrwkl,
...          ...              ...
finn@ubuntu:~$ cat ~/.ssh/id_rsa
-----BEGIN RSA PRIVATE KEY-----
...          ...              ...            ...
-----END RSA PRIVATE KEY-----
finn@ubuntu:~$ firefox ~/.ssh/id_rsa
```




```
finn@ubuntu:~$ sudo aa-enforce firefox
Profile for /usr/lib/firefox/firefox.sh not found, skipping
finn@ubuntu:~$ pgrep firefox
16392
finn@ubuntu:~$ ps up 16392
USER     PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
finn   16392  2.4  5.9 2974256 238292 pts/2  Sl   23:33   0:03 /usr/lib/firefox/firefox
finn@ubuntu:~$ sudo aa-enforce /usr/lib/firefox/firefox
Назначение /usr/lib/firefox/firefox принудительного режима.
Warning: profile /usr/lib/firefox/firefox{,*[^s][^h]} represents multiple programs
finn@ubuntu:~$ pkill firefox
...          ...              ...
finn@ubuntu:~$ firefox ~/.ssh/id_rsa
```



Стоит отметить, что команды **aa-enforce(8)** и **aa-status(8)** выполняются от лица *суперпользователя*, единолично управляющего модулем принудительного контроля дос-

тупа AppArmor, но детальное рассмотрение синтаксиса правил и процедур управления модулем относится к задачам администрирования операционной системы, которые выходят за рамки этой книги.

3.6.2. Модуль принудительного разграничения доступа SELinux

Еще одна известная реализация мандатных механизмов разграничения доступа в RedHat/CentoOS/Fedora Linux называется W:[SELinux] (Security Enhanced Linux) и имеет несколько моделей применения и соответствующих им наборов мандатных правил, называемых *политиками*. Политики SELinux описывают ограничения доступа субъектов — программ к объектам операционной системы — файлам, сетевым портам протоколов TCP и UDP и пр. на основе *меток* безопасности.

Самая распространенная из моделей применения, похожая на AppArmor, представлена целевой *(target)* политикой, предполагает ограничение прав доступа только *определенных* программ, тогда как остальные программы никак не ограничиваются. Другие модели применения представлены многоуровневой (MLS, multilayer security и ее вариантом MCS, multicategory security) и строгой *(strict)* политиками.

В многоуровневой политике определяются уровни (и категории) доступа, например «не секретно», «для служебного пользования», «секретно», «совершенно секретно» и т. д., и реализуется принцип «no read up, no write down», т. е. запрещение читать файлы с меткой более высокого уровня и запрещение изменять файлы с меткой более низкого уровня. В строгой политике, ограничивающей права доступа *всех* процессов, запрещается все, что не разрешено явно.

В листинге 3.50 при помощи команды **sestatus(8)** показан статус модуля принудительного контроля доступа SELinux. Аналогично AppArmor, у модуля SELinux имеются разрешительный *(permissive)* и принудительный *(enforcing)* режимы ❷ работы. В разрешительном режиме нарушения мандатных правил процессами приводят только к журнализации сообщений аудита об обнаруженных нарушениях. Попытки нарушения мандатных правил процессами, находящимися в принудительном режиме, пресекаются модулем контроля доступа в виде отказа в доступе к тому или иному запрашиваемому объекту.

Листинг 3.50. Модуль мандатного разграничения доступа SELinux

```
[lich@fedora ~]$ sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
❶ Loaded policy name:           targeted
❷ Current mode                  enforcing
```

```
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Max kernel policy version:      29
```

В отличие от AppArmor, модуль SELinux оперирует не профилями отдельных программ, а общим набором правил ❶, называемых политикой *(policy)*. Сами правила политики описывают возможные отношения между процессами и файлами согласно их меткам безопасности. Каждая метка состоит из четырех компонент **user:role:type:level**, которые могут быть использованы лишь частично. Например, в целевой политике используются только составляющая «тип» **(type)** и принцип «соблюдения типов» (TE, type enforcement), согласно которому доступ программы к файлу разрешается, если они имеют «совместимые» типы.

В примере из листинга 3.51 показаны метки безопасности процессов **bash** и **ps** сеанса пользователя **lich** и процесса **httpd** системной службы Web-сервера. В *целевой (targeted)* политике программы, запускаемые в пользовательских сеансах, и, соответственно, их процессы никак не ограничиваются, что и обозначает тип **unconfined_t** их метки безопасности. Наоборот, процессы всех системных служб и являются собственно «целями» принудительного контроля доступа целевой политики, поэтому выполняются со своими типами, например **httpd_t** для службы Web-сервера.

Листинг 3.51. Мандатные метки процессов

```
[lich@centos ~]$ ps -Z
LABEL                              PID TTY          TIME CMD
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 8893 pts/0 00:00:00 bash
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 9454 pts/0 00:00:00 ps
[lich@centos ~]$ ps -ZC httpd
LABEL                              PID TTY          TIME CMD
system_u:system_r:httpd_t:s0      3262 ?        00:00:00 httpd
...          ...                  ...             ...              ...
system_u:system_r:httpd_t:s0      3267 ?        00:00:00 httpd
```

Политика содержит правила назначения меток *процессам* тех *программ*, исполняемые файлы которых тоже имеют свои метки, проиллюстрированные в листинге 3.52. Так, например, целевая политика содержит правила, согласно которым процессы Web-сервера **httpd** получают метку с типом **httpd_t**, потому что выполняют программы **/usr/sbin/httpd** с меткой **httpd_exec_t**.

Листинг 3.52. Мандатные метки файлов программ

```
[lich@centos ~]$ ls -Z /usr/sbin/httpd
-rwxr-xr-x. root root system_u:object_r:httpd_exec_t:s0 /usr/sbin/httpd
```

Файлы с данными тоже размечаются своими метками. Например, в листинге 3.53 иллюстрируются метки файлов в домашнем каталоге пользователя **lich**, где при помощи меток дифференцированы разные пользовательские данные *по смыслу*.

Листинг 3.53. Мандатные метки файлов

```
[lich@centos ~]$ ls -Za ~
drwxr-xr-x. lich lich unconfined_u:object_r:user_home_t:s0 Desktop
                      ...          ...             ...              ...
drwxr-xr-x. lich lich unconfined_u:object_r:audio_home_t:s0 Music
                      ...          ...             ...              ...
drwx------. lich lich unconfined_u:object_r:ssh_home_t:s0 .ssh
                      ...          ...             ...              ...
```

В частности, пользовательские ключи SSH в каталоге **~/.ssh** имеют метку с типом **ssh_home_t**, на основании чего и можно ограничить к ним доступ таких программ как **skype** или **firefox**. Для этого нужно просто выполнять эти программы в процессах с такими типами в их метках, доступ которых к файлам с метками, имеющими тип **ssh_home_t**, ограничен правилами политики.

В листинге 3.54 при помощи команды **sesearch(1)** производится поиск разрешительных (**-A**, allow) правил в политике безопасности, разрешающих (**-p**, permissions) всем субъектам с исходным (**-s**, source) типом **unconfined_t** открывать (**open**) объекты файлового (**file**) класса (**-c**, class), имеющие целевой (**-t**, target) тип **ssh_home_t**.

Листинг 3.54. Мандатные правила типа unconfined_t

```
[lich@centos ~]$ sesearch -A -s unconfined_t -t ssh_home_t -c file -p open
Found 2 semantic av rules:
 allow files_unconfined_type file_type: file { ioctl read ... open audit_access } ;
❶ allow unconfined_t user_home_type: file { ioctl read write create ... rename open } ;
[lich@centos ~]$ seinfo -tssh_home_t -x
      ssh_home_t
                                                                    ...
❷     user_home_type
[lich@centos ~]$ seinfo -auser_home_type -x
   user_home_type
      audio_home_t
```

```
❸       ssh_home_t
                    ...                    ...          ...
        audio_home_t
```

В результате найдено правило ❶, разрешающее доступ субъектов с типом **unconfined_t** к объектам, типы которых «обобщаются» атрибутом **user_home_type**.

При помощи команды **seinfo(1)** уточняется, что тип (**-t**, type) **ssh_home_t** действительно обобщен атрибутом ❷ **user_home_type**, и, наоборот, атрибут **user_home_type** обобщает такие «пользовательские» типы ❸, как **ssh_home_t**, **audio_home_t**, **user_home_t** и пр.

Другими словами, всем процессам пользовательского сеанса (т. к. они имеют тип **unconfined_t**) разрешено обращаться к пользовательским SSH-ключам (имеющим тип **ssh_home_t**) и любым другим пользовательским данным (с типами **audio_home_t**, **user_home_t** и пр.).

Для ограничения программ в доступе к SSH-ключам пользователей потребуется некоторый другой тип (отличный от **unconfined_t**), которому правилами политики будет запрещено обращаться к файлам с типом **ssh_home_t**. Такими типами в целевой политике являются несколько типов «песочниц» (sandbox), например **sandbox_t** или **sandbox_net_t** (разрешающий сетевые соединения).

В примере из листинга 3.55 поиск ❶ разрешающих правил показывает, что процессам с типом **sandbox_net_t** не разрешено открытие файлов с типом **ssh_home_t**, но разрешены ❷ TCP-соединения. Таким образом, тип **sandbox_net_t** оказывается подходящим кандидатом процессов тех программ, которым нужно ограничить доступ к SSH-ключам и другим файлам пользователя.

Листинг 3.55. Мандатные правила типа sandbox_net_t

```
❶ [lich@centos ~]$ sesearch -A -s sandbox_net_t -t ssh_home_t -c file -p open
!
❷ [lich@centos ~]$ sesearch -A -s sandbox_net_t -c tcp_socket -p connect
   Found 4 semantic av rules:
      allow sandbox_net_t sandbox_net_t: tcp_socket { ... read write ... connect ... shutdown } ;
                          ...          ...          ...          ...
    [lich@centos ~]$ links -dump ~/.ssh/id_rsa
?  -----BEGIN RSA PRIVATE KEY-----
                          ...          ...          ...          ...
   -----END RSA PRIVATE KEY-----
    [lich@centos ~]$ sandbox -t sandbox_net_t links -dump ~/.ssh/id_rsa
❸ ELinks: Отказано в доступе
```

Запустить программу в процессе с соответствующей меткой позволяет команда **sandbox(8)**, специально предназначенная для формирования «песочниц» при помощи

модуля SELinux и соответствующих **sandbox_*_t**-типов. В примере из листинга 3.55 текстовый Web-браузер **links(1)**, выполненный в процессе с типом **sandbox_net_t**, в действительности оказывается ограниченным ❸ в доступе к SSH-ключам пользователей, что и требовалось получить.

3.7. Дополнительные свойства файлов

3.7.1. Расширенные атрибуты файлов

Как было показано выше (см. листинг 3.4), базовые права доступа, дополнительные атрибуты SUID/SGID, владельцы, счетчик имен и другие *основные свойства* файлов хранятся в их метаданных. Кроме этого, файлам могут быть назначены списки контроля доступа (см. листинг 3.44) и метки безопасности SELinux (см. листинг 3.50), которые являются их *дополнительными свойствами* и хранятся за пределами метаданных, при помощи расширенных атрибутов **attr(5)**.

Каждый расширенный атрибут имеет имя вида **namespace.attrname**, при этом пространства имен **namespace** определяют назначение атрибута. Пространство имен **system** используется системными (ядерными) компонентами, например, для списков контроля доступа POSIX ACL. Пространство имен **security** используется системными компонентами безопасности, в частности для хранения привилегий исполняемых программ (**capabilities(7)**, *см. разд. 4.5.2*) и меток модуля принудительного контроля доступа SELinux (см. листинги 3.52 и 3.53). Пространства имен **trusted** и **user** предназначены для атрибутов внеядерных компонент — программ, выполняющихся привилегированным и обычными пользователями, соответственно.

Для просмотра и назначения внеядерных (пользовательских) расширенных атрибутов используются утилиты **getfattr(1)** и **setfattr(1)**. Читать *пользовательские расширенные атрибуты* файла разрешено тем же субъектам, которым разрешено чтение *данных* этого файла. Аналогично, устанавливать (изменять) и удалять *пользовательские расширенные атрибуты* файла могут субъекты, допущенные к записи данных этого файла.

Ядерные (системные) атрибуты обычно управляются специально предназначенными командами: например, **getfacl(1)** и **setfacl(1)** предназначены для списков контроля доступа, команды **getcap(8)** и **setcap(8)** — для привилегий исполняемых программ, команды **chcon(1)** и **ls(1) -Z** — для мандатных меток. Системные атрибуты, как правило, всегда доступны для чтения, но их установка и изменение требуют определенных привилегий процесса *(см. разд. 4.5.2)*.

Листинг 3.56. Расширенные системные атрибуты файлов

```
finn@ubuntu:~$ getcap /usr/bin/gnome-keyring-daemon
/usr/bin/gnome-keyring-daemon = cap_ipc_lock+ep ❶
```

```
finn@ubuntu:~$ getfattr -d -m - /usr/bin/gnome-keyring-daemon
getfattr: Удаление начальных '/' из абсолютных путей
# file: usr/bin/gnome-keyring-daemon
security.capability=0sAQAAAgBAAAAAAAAAAAAAAAAAAAA=

finn@ubuntu:~$ cd /srv/kingdom/stash
finn@ubuntu:/srv/kingdom/stash$ ls -l README.jake
-rw-rw----+ 1 finn      candy    0 нояб.  4 14:17 README.jake
finn@ubuntu:/srv/kingdom/stash$ getfacl README.jake
# file: README.jake
# owner: finn
# group: candy
user::rw-
user:jake:rw- ❷
group::rwx                      #effective:rw-
mask::rw-
other::---
finn@ubuntu:/srv/kingdom/stash$ getfattr -d -m - README.jake
# file: README.jake
system.posix_acl_access=0sAgAAAAEABgD/////AgAGAOoDAAAEAAcA/////xAABgD/////IAAAAP////8=

finn@ubuntu:/srv/kingdom/stash$ setfattr -n user.color -v orange README.jake
finn@ubuntu:/srv/kingdom/stash$ setfattr -n user.flavour -v vanilla README.jake
finn@ubuntu:/srv/kingdom/stash$ getfattr -d README.jake
# file: README.jake
user.color="orange"
user.flavour="vanilla"
```

В примере из листинга 3.56 показано, что привилегии исполняемых программ ❶ на самом деле сохранены в атрибуте **security.capability**, списки контроля доступа ❷ — в атрибуте **system.posix_acl_access**, а в атрибутах пространства имен **user** можно разместить любые значения. Наиболее известным применением пользовательских атрибутов (листинг 3.57) являются атрибуты **user.xdg.origin.url** и **user.xdg.referrer.url**, используемые браузером **chromium-browser(1)** для сохранения URL файлов, которые были загружены из Интернета.

Листинг 3.57. Расширенные пользовательские атрибуты файлов

```
finn@ubuntu:~/Downloads$ getfattr -d TLCL-13.07.pdf
# file: TLCL-13.07.pdf
```

```
user.xdg.origin.url="http://freefr.dl.sourceforge.net/project/linuxcommand/TLCL/13.07/
TLCL-13.07.pdf"
user.xdg.referrer.url="http://sourceforge.net/projects/linuxcommand/files/TLCL/13.07/
TLCL-13.07.pdf/download"
```

3.7.2. Флаги файлов

Кроме «общих» расширенных атрибутов, которые используются разными компонентами операционной системы, каждая файловая система зачастую имеет собственные атрибуты файлов, управляющие ее поведением и функциями при доступе к файлам. Так, файловые системы **ext2/ext3/ext4** управляются специальными атрибутами-флагами, например **a** (append only), **i** (immutable), **s** (secure deletion), S (synchronous updates) и пр.

Флаг **s** заставляет файловую систему при удалении файла не только высвобождать принадлежащие ему блоки, но и обнулять их, а флаг **S** заставляет операции записи в файл выполняться синхронно (немедленно), минуя отложенную запись с использованием дискового кэша. Флаг **i** делает файл «неприкасаемым» (❶, листинг 3.58) — его нельзя ни изменить ❷, ни удалить ❸ никому, даже суперпользователю ❹ **root**. Флаг **a** делает файл «накопительным», т. е. никому не дает изменять имеющуюся в файле информацию или удалять файл, а позволяет только добавлять данные ❷ в его конец.

Устанавливать флаги файлов разрешено их владельцам, а установка отдельных флагов, например **a** или **i**, требует определенных привилегий процесса *(см. разд. 4.5.2)*. Для просмотра флагов файлов предназначена утилита **lsattr(1)**, а для их изменения — утилита **chattr(1)**, что иллюстрирует листинг 3.58 на примере флага **i**.

Листинг 3.58. Флаги файлов

```
finn@ubuntu:/srv/kingdom/stash$ lsattr
------------e- ./README.jake
------------e- ./README
finn@ubuntu:/srv/kingdom/stash$ chattr +i README.jake
chattr: Операция не позволяется while setting flags on README.jake
finn@ubuntu:/srv/kingdom/stash$ sudo chattr +i README.jake
finn@ubuntu:/srv/kingdom/stash$ lsattr README.jake
----i--------e- README.jake
finn@ubuntu:/srv/kingdom/stash$ date >[1] README.jake
-bash: README.jake: Операция не позволяется
```

[1] О перенаправлениях подробнее *см. разд. 5.3.*

```
finn@ubuntu:/srv/kingdom/stash$ rm README.jake
rm: невозможно удалить «README.jake»: Операция не позволяется
finn@ubuntu:/srv/kingdom/stash$ sudo rm README.jake
rm: невозможно удалить «README.jake»: Операция не позволяется ☚!
```

3.8. В заключение

Всестороннее рассмотрение разнообразных файлов, их свойств, атрибутов и контекстов использования неизбежно должно приводить к выводу, что файл является универсальной сущностью, позволяющей организовать однородный доступ к информации, вне зависимости от свойств ее источника. Специальные файлы устройств, именованные каналы и сокеты имеют файловую природу и могут обрабатываться совершенно «обычными» программами за счет идентичности их файлового программного интерфейса, наравне с файлами «обычных» (дисковых), сетевых, псевдофайловых и внеядерных файловых систем.

Подсистема управления процессами, о которой пойдет речь в следующей главе, тоже не обходится без файлов и использует механизм их отображения в память для организации виртуальной памяти и средств межпроцессного взаимодействия, таких как разделяемая память, очереди сообщений[1] и семафоры. Даже сетевые сокеты, рассмотрение которых отложено до *главы 6*, тоже на поверку оказываются файлами.

Таким образом, понимание файла как основополагающей компоненты операционной системы дает ключ к пониманию многих других ее частей, а навыки мониторинга файлов или трассировки файлового интерфейса позволяют заглянуть в корень практически всех ее механизмов.

[1] В реализации POSIX, но, к сожалению, не в реализации SYSV.

Глава 4

Управление процессами и памятью

Процессы операционной системы в большинстве случаев отождествляются с выполняющимися программами, что не совсем верно, точнее — совсем не верно. В современных операционных системах, включая Linux, между программой и процессом есть очевидная взаимосвязь, но далеко не такая непосредственная, как кажется на первый взгляд.

4.1. Программы и библиотеки

Программа представляет собой *алгоритм*, записанный на определенном языке, понятном исполнителю программы[1]. Различают *машинный* язык, понятный центральному процессору, и языки более *высоких уровней* (алгоритмические), понятные составителю программы — программисту.

Программы, составленные на языке высокого уровня, в любом случае перед исполнением должны быть транслированы (переведены) на язык исполнителя, что реализуется при помощи специальных средств — *трансляторов*. Различают два вида трансляторов программ — *компиляторы* и *интерпретаторы*. Компилятор транслирует в машинный код сразу целиком всю программу и не участвует в ее исполнении. Интерпретатор, наоборот, пошагово транслирует отдельные инструкции программы и немедленно выполняет их. Например, *командный интерпретатор* при интерактивном режиме пошагово выполняет команды, вводимые пользователем, а в пакетном режиме *(см. главу 5)* так же пошагово выполняет команды, записанные в файле сценария.

Алгоритм, в свою очередь, есть некоторый набор инструкций, выполнение которых приводит к решению конкретной задачи. В большинстве случаев инструкции алгоритма имеют причинно-следственные зависимости и выполняются исполнителем

[1] Программа политической партии и программа научной конференции имеют тот же смысл, что и компьютерная программа, но предназначены для других исполнителей.

последовательно. Однако если выделить «независимые» поднаборы инструкций (независимые ветви), то их можно выполнять несколькими исполнителями одновременно — *параллельно*. Поэтому различают *последовательные* и *параллельные* алгоритмы и соответствующие им последовательные и параллельные программы. Некоторые программы реализуют алгоритмы *общего* назначения, например алгоритмы сжатия или шифрования информации, алгоритмы сетевых протоколов и т. д. Такие программы, востребованные не столько конечными пользователями, сколько другими программами, называют *библиотеками*.

Согласно **hier(7)**, откомпилированные до машинного языка программы размещаются в каталогах **/bin**, **/sbin**, **/usr/bin**, **/usr/sbin**, **/usr/local/bin**, **/usr/local/sbin**, а библиотеки — в каталогах **/lib**, **/usr/lib**, **/usr/local/lib**. Программы имеют специальный бинарный «запускаемый» формат **W:[ELF] executable** и зависят от библиотек, что проиллюстрировано в листинге 4.1 при помощи команды **ldd(1)** (loader dependencies). Каждая зависимость отображается именем библиотеки ❶ (SONAME, shared object **name**), найденным в системе файлом библиотеки ❷ и адресом[1] в памяти процесса ❸ (32- или 48-битным, в зависимости от платформы), куда библиотека будет загружена.

Листинг 4.1. Программы и библиотеки

```
fitz@ubuntu:~$ which ls
/usr/bin/ls
fitz@ubuntu:~$ file /usr/bin/ls
/usr/bin/ls: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=2f15ad836be3339dec0e2e6a3c637e08e48aacbd, for GNU/Linux 3.2.0, stripped
fitz@ubuntu:~$ ldd /usr/bin/ls
        linux-vdso.so.1 (0x00007ffcb529d000)
        libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007fb02f58d000)
    ❶ libc.so.6 => ❷ /lib/x86_64-linux-gnu/libc.so.6 (0x00007fb02f39c000) ❸
        libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0 (0x00007fb02f317000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fb02f311000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fb02f5f1000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fb02f2ee000)

fitz@ubuntu:~$ file /lib/x86_64-linux-gnu/libc.so.6
❹ /lib/x86_64-linux-gnu/libc.so.6: symbolic link to libc-2.30.so
```

[1] Для противодействия эксплуатации уязвимости в программах адрес выбирается случайным образом, см. **W:[ASLR]**.

Нужно заметить, что файла библиотеки **linux-vdso.so.1** (реализующей интерфейс системных вызовов к ядру) не существует, т. к. она является виртуальной (VDSO, virtual dynamic shared object), т. е. предоставляется и отображается в память процесса самим ядром, «как будто» является настоящей библиотекой. Кроме того, библиотека **ld-linux-x86-64.so.2** указана абсолютным путевым именем, поэтому поиск ее файла не производится.

Для большинства библиотек зависимость устанавливается при помощи SONAME вида **lib*NAME*.so.*X***, где **lib** — стандартный префикс (***library***, библиотека), **.so** — суффикс (***shared object***, разделяемый объект), **NAME** — имя «собственное», а **.*X*** — номер версии ее интерфейса (листинг 4.2). По имени SONAME в определенных (конфигурацией компоновщика — см. **ld.so(8)** и **ldconfig(8)**) каталогах производится поиск одноименного файла библиотеки, который на самом деле оказывается символической ссылкой ❹ на «настоящий» файл библиотеки. Например, для **6**-й версии интерфейса динамической библиотеки языка **C** (**libc.so.6**) настоящий файл библиотеки называется **libc-2.30.so**, что указывает на версию самой библиотеки как **2.30**.

Листинг 4.2. Версии библиотек

```
fitz@ubuntu:~$ file /lib/x86_64-linux-gnu/libpcre2-8.so.0
/lib/x86_64-linux-gnu/libpcre2-8.so.0: symbolic link to libpcre2-8.so.0.7.1 ❧
```

Аналогично, в листинге 4.2 показано, что для **0**-й версии интерфейса динамической библиотеки регулярных perl-выражений **pcre2** (**libpcre2-8.so.0**) настоящий файл библиотеки называется **libpcre2-8.so.0.7.1**, а это указывает на версию самой библиотеки как **0.7.1**.

Такой подход позволяет заменять (исправлять ошибки, улучшать неэффективные алгоритмы и пр.) библиотеки (при условии неизменности их интерфейсов) *отдельно* от программ, зависящих от них. При обновлении библиотеки **libc-2.30.so**, например, до **libc-2.32.so** достаточно установить символическую SONAME-ссылку **libc.so.6** на **libc-2.32.so**, в результате чего ее начнут использовать все программы с зависимостями от **libc.so.6**. Более того, в системе может быть одновременно установлено любое количество версий одной и той же библиотеки, реализующих одинаковые или разные версии интерфейсов, выбор которых будет указан соответствующими SONAME-ссылками.

Листинг 4.3. Библиотеки — это незапускаемые программы

```
fitz@ubuntu:~$ file /lib/x86_64-linux-gnu/libc-2.30.so
/lib/x86_64-linux-gnu/libc-2.30.so: ELF 64-bit LSB shared object, x86-64, version 1
(GNU/Linux), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=2155f455ad56bd871c8225bcca85ee25c1c197c4, for GNU/Linux 3.2.0, stripped
```

```
fitz@ubuntu:~$ file /lib/x86_64-linux-gnu/libpcre2-8.so.0.7.1
/lib/x86_64-linux-gnu/libpcre2-8.so.0.7.1: ELF 64-bit LSB shared object, x86-64, version 1
(SYSV), dynamically linked, BuildID[sha1]=815e1acbcc22015f05d62c17fe982c1b573125b1, stripped

fitz@ubuntu:~$ ldd /lib/x86_64-linux-gnu/libpcre2-8.so.0.7.1
    linux-vdso.so.1 (0x00007ffe22093000)
    libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f8ec2bdd000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f8ec29ec000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f8ec2c99000)
```

Библиотеки имеют тот же бинарный формат **W:[ELF]**, что и «запускаемые» программы, но не «запускаемый» **executable**, а «совместно используемый» **shared object**. Библиотеки, являясь пусть и незапускаемыми, но программами, естественным образом тоже зависят от других библиотек, что показано в листинге 4.3. Практически «запускаемость» ELF-файлов (листинг 4.4) зависит не от их типа, а от прав доступа и осмысленности точки входа — адреса первой инструкции, которой передается управление при попытке запуска. Например, библиотеку **libc-2.30.so** можно запустить, в результате чего будет выведена статусная информация.

Листинг 4.4. Запускаемые библиотеки

```
fitz@ubuntu:~$ ls -l /lib/x86_64-linux-gnu/libc-2.30.so
-rwxr-xr-x 1 root root 2025032 сен 16 17:56 /lib/x86_64-linux-gnu/libc-2.30.so
fitz@ubuntu:~$ /lib/i386-linux-gnu/libc-2.15.so
GNU C Library (Ubuntu GLIBC 2.30-0ubuntu2) stable release version 2.30.
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
Compiled by GNU CC version 9.2.1 20190909.
libc ABIs: UNIQUE IFUNC ABSOLUTE
For bug reporting instructions, please see:
<https://bugs.launchpad.net/ubuntu/+source/glibc/+bugs>.
```

4.1.1. Ядро Linux

Не стоит забывать, что самой главной *программой* операционной системы является ее ядро, которое в Linux состоит из статического стартового модуля (листинг 4.5) в формате **ELF executable** и динамически пристыковываемых программных модулей формата **ELF relocatable** (листинг 4.6). Для выполнения процедуры начальной загрузки стартовый модуль упакован в «самораспаковывающийся» **gzip**-архив формата

bzImage (big zipped image), который состоит из программы распаковки и собственно запакованного стартового модуля.

В листинге 4.5 проиллюстрирован процесс извлечения стартового модуля из архива **/boot/vmlinuz-3.13.0-49-generic** формата bzImage ❶, который предварительно копируется ❶ в **/tmp/vmlinuz**. Для извлечения используется сценарий **extract-vmlinux** ❷ из пакета заголовочных файлов ядра. Распакованный ❸ стартовый модуль **/tmp/vmlinux** ожидаемо оказывается статически скомпонованной (т. е. не использующей библиотеки ELF shared object) исполняемой **ELF**-программой.

Листинг 4.5. Ядро операционной системы

```
fitz@ubuntu:~$ uname -r
5.3.0-23-generic
fitz@ubuntu:~$ file /boot/vmlinuz-5.3.0-23-generic
/boot/vmlinuz-5.3.0-23-generic: regular file, no read permission
fitz@ubuntu:~$ ls -l /boot/vmlinuz-5.3.0-23-generic
-rw------- 1 root root 11399928 ноя 12 11:51 /boot/vmlinuz-5.3.0-23-generic
fitz@ubuntu:~$ sudo file /boot/vmlinuz-5.3.0-23-generic
❶ /boot/vmlinuz-5.3.0-23-generic: Linux kernel x86 boot executable bzImage, version 5.3.0-23-
generic (buildd@lgw01-amd64-002) #25-Ubuntu SMP Tue Nov 12 09:22:33 UTC 2019, RO-rootFS,
swap_dev 0xA, Normal VGA
❶ fitz@ubuntu:~$ sudo cat /boot/vmlinuz-5.3.0-23-generic > /tmp/vmlinuz
❷ fitz@ubuntu:~$ /usr/src/linux-headers-5.3.0-23/scripts/extract-vmlinux /tmp/vmlinuz >
/tmp/vmlinux
fitz@ubuntu:~$ file /tmp/vmlinux
❸ /tmp/vmlinux: ELF 64-bit LSB executable ❸, x86-64, version 1 (SYSV), statically linked,
BuildID[sha1]=b23ff3f6790319ec538278e3269af619ba2ca642, stripped
```

Динамические модули загружаются в пространство ядра и пристыковываются к стартовому модулю позднее, уже при работе операционной системы при помощи системных утилит insmod(8) или modprobe(8). Для отстыковки и выгрузки ненужных модулей предназначена системная утилита rmmod(8), для просмотра списка ❶ (см. листинг 4.6) загруженных модулей — lsmod(8), а для идентификации свойств и параметров ❷ модулей — утилита modinfo(8). Загрузка и выгрузка модулей реализуется специальными системными вызовами init_module(2) и delete_module(2), доступ к списку загруженных модулей — при помощи файла **/proc/modules** псевдофайловой системы proc(5), а идентификация свойств и параметров модулей — чтением специальных секций ELF-файлов модулей.

Листинг 4.6. Модули ядра

```
❶ fitz@ubuntu:~$ lsmod
Module                  Size  Used by
                        ...           ...             ...
```

```
i915                  1949696  4
                       ...          ...          ...
btusb                   57344  0
                       ...          ...          ...
uvcvideo                98304  0
                       ...          ...          ...
e1000e                 258048 0
                       ...          ...          ...
❷ fitz@ubuntu:~$ modinfo i915
filename:       /lib/modules/5.3.0-23-generic/kernel/drivers/gpu/drm/i915/i915.ko
license:        GPL and additional rights
description:    Intel Graphics
                       ...          ...          ...
fitz@ubuntu:~$ file /lib/modules/5.3.0-23-generic/kernel/drivers/gpu/drm/i915/i915.ko
❸ /lib/modules/5.3.0-23-generic/kernel/drivers/gpu/drm/i915/i915.ko: ELF 64-bit LSB
relocatable, x86-64, version 1 (SYSV),
BuildID[sha1]=49e59590c1a718074b76b6541702f6f794ea7eae, not stripped
```

Динамические модули ядра зачастую являются драйверами устройств, что проиллюстрировано в листинге 4.7 при помощи утилит **lspci(8)** и **lsusb(8)**, которые сканируют посредством псевдофайловой системы **sysfs** списки обнаруженных ядром на шинах PCI и USB устройств и обслуживающих их драйверов.

Листинг 4.7. Драйверы устройств

```
fitz@ubuntu:~$ lspci -k
                       ...          ...          ...
00:02.0 VGA compatible controller: Intel Corporation 2nd Generation Core Process
or Family Integrated Graphics Controller (rev 09)
        Subsystem: Dell 2nd Generation Core Processor Family Integrated Graphics
 Controller
        Kernel driver in use: i915
        Kernel modules: i915
                       ...          ...          ...
00:19.0 Ethernet controller: Intel Corporation 82579LM Gigabit Network Connection
(Lewisville) (rev 04)
        Subsystem: Dell 82579LM Gigabit Network Connection (Lewisville)
        Kernel driver in use: e1000e
        Kernel modules: e1000e
fitz@ubuntu:~$ lsusb -t
                       ...          ...          ...
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=ehci-pci/3p, 480M
    |__ Port 1: Dev 2, If 0, Class=Hub, Driver=hub/6p, 480M
        |__ Port 4: Dev 3, If 2, Class=Vendor Specific Class, Driver=, 12M
        |__ Port 4: Dev 3, If 0, Class=Wireless, Driver=btusb, 12M
        |__ Port 4: Dev 3, If 3, Class=Application Specific Interface, Driver=, 12M
        |__ Port 4: Dev 3, If 1, Class=Wireless, Driver=btusb, 12M
        |__ Port 5: Dev 4, If 0, Class=Video, Driver=uvcvideo, 480M
        |__ Port 5: Dev 4, If 1, Class=Video, Driver=uvcvideo, 480M
```

4.2. Процессы и нити

Сущность *процесса* неразрывно связана с мультипрограммированием и *многозадачностью* операционной системы. Например, в однозадачных операционных системах программы существуют, а процессы — нет. В *однозадачных* операционных системах единовременно *одна* последовательная программа выполняется *одним* исполнителем (центральным процессором), имея возможность безраздельно использовать все доступные ресурсы (память, устройства ввода-вывода и пр.).

В любой программе можно выделить перемежающиеся блоки инструкций, использующих или центральный процессор (ЦП), или устройства ввода-вывода (УВВ). При этом центральный процессор вынужден простаивать при выполнении программой операций ввода-вывода, например, при ожидании окончания записи (или чтения) блока данных на внешний носитель, или при ожидании окончания передачи (или приема) сетевого кадра, или при ожидании событий с устройств человеко-машинного взаимодействия. С другой стороны, устройства ввода-вывода тоже вынуждены простаивать при выполнении программой вычислительных операций, например ожидая результата, подлежащего выводу, или ожидая возникновения у программы потребности в новых исходных данных.

Используя такую модель поведения программ, можно провести анализ потребления ими ресурсов при выполнении. Например, компрессоры **gzip(1)**, **bzip(2)** и **xz(1)** считывают очередной блок данных исходного файла, относительно долго упаковывают его и записывают в результирующий файл, а затем повторяют процедуру до исчерпания блоков исходного файла. Количество времени, потраченного на вычислительные операции упаковки, будет много больше количества времени, потраченного на чтение исходных данных и запись результатов, поэтому нагрузка на ЦП будет высокой, а на УВВ — нет. Такой же анализ можно привести и для дубликатора **dd(1)**, копировщика **rsync(1)** или архиватора **tar(1)**, которые, наоборот, почти не выполняют никаких вычислений, а сосредоточены на вводе-выводе больших объемов данных, поэтому при их использовании нагрузка на ЦП будет довольно низкой, а на УВВ — высокой.

Для командного интерпретатора **bash(1)**, текстовых редакторов **nano(1)** и **vim(1)** и других *интерактивных* программ, взаимодействующих с пользователем, характерны длительные ожидания ввода небольших команд, простая и недолгая их обработка и вывод короткого результата. В результате коэффициент полезного использования и ЦП, и УВВ будет приближен к нулю.

Подобный анализ и желание увеличить коэффициенты полезного использования ресурсов привели к созданию *многозадачных* операционных систем, основывающихся на простой идее псевдоодновременного выполнения *нескольких* последовательных программ *одним* исполнителем. Для этого вместо *простоя* в ожидании окончания операции ввода-вывода, начатой некоторой программой, центральный

процессор переключается на *выполнение* другой программы, тем самым увеличивая интегральный коэффициент его полезного использования.

С появлением *мультипрограммной смеси*[1] каждая из ее программ больше не может безраздельно использовать все доступные ресурсы (например, всю память — она одновременно нужна всем программам смеси), в связи с чем операционная система берет на себя задачи диспетчеризации (распределения) ресурсов между ними. В Linux, как и во многих других операционных системах, программы изолируются друг от друга в специальных «виртуальных» средах, обеспечивающих их *процесс* выполнения. Каждая такая среда называется *процессом* и получает долю доступных ресурсов — выделенный участок памяти, выделенные промежутки процессорного времени. Процесс эмулирует для программы «однозадачный» режим выполнения, словно программа выполняется в одиночку, и «безраздельное» использование ресурсов процесса, как будто это все доступные ресурсы.

Параллельные программы, как указывалось ранее, состоят из независимых ветвей, каждая из которых сама по себе укладывается в модель поведения последовательной программы, поэтому *одну* параллельную программу можно выполнять в *нескольких* процессах в псевдоодновременном режиме. *Процессы* операционной системы, таким образом, являются контейнерами для многозадачного выполнения программ, как последовательных, так и параллельных.

В листинге 4.8 при помощи команды **ps(1)** показаны процессы пользователя, упорядоченные в дерево, построенное на основе дочерне-родительских отношений между процессами. Уникальный идентификатор, отличающий процесс от других, выведен в столбце **PID** (**p**rocess **id**entifier), а имя и аргументы программы, запущенной в соответствующем процессе — в столбце **COMMAND**.

В столбце **STAT** показано текущее состояние процесса, например **S** (сон, sleep) или **R** (выполнение, running, или готовность к выполнению, runnable). Процессы, ожидающие завершения их операций ввода-вывода, находятся в состоянии *сна*, в противном случае либо *выполняются*, либо *готовы* к выполнению, т. е. ожидают, когда текущий выполняющийся процесс заснет и процессор будет переключен на них. В столбце **TIME** показано чистое потребленное процессом процессорное время от момента запуска программы, увеличивающееся только при нахождении им в состоянии выполнения.

Листинг 4.8. Дерево процессов пользователя

```
fitz@ubuntu:~$ ps fx
  PID TTY      STAT   TIME COMMAND
                ...          ...          ...
```

[1] Набор программ, между которыми переключается процессор.

```
  17764 tty3     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script ...
  17766 tty3     Sl+    3:09  \_ /usr/lib/xorg/Xorg vt3 -displayfd 3 ...
  17774 tty3     Sl+    0:00  \_ /usr/lib/gnome-session/gnome-session-binary ...
                                 ...          ...          ...
   2987 ?        Ss     0:04 /lib/systemd/systemd --user
   2992 ?        S      0:00  \_ (sd-pam)
  17373 ?        Ssl    0:08  \_ /usr/bin/pulseaudio --daemonize=no
  17444 ?        Ss     0:02  \_ /usr/bin/dbus-daemon --session --address=systemd: ...
                                 ...          ...          ...
  17921 ?        Ssl   10:04  \_ /usr/bin/gnome-shell
                                 ...          ...          ...
❶ 30192 ?        Ssl    0:00  \_ /usr/libexec/gnome-terminal-server
❷ 30202 pts/1    Ss     0:00      \_ bash
❸ 30226 pts/1    S+     0:00      |   \_ man ps
  30236 pts/1    S+     0:00      |       \_ pager
❹ 30245 pts/3    Ss     0:00      \_ bash
❺ 30251 pts/3    R+     0:00          \_ ps fx
❻ 30315 ?        Sl     0:04  \_ /usr/lib/firefox/firefox -new-window
  30352 ?        Sl     0:02      \_ /usr/lib/firefox/firefox -contentproc -childID 1 ...
  30396 ?        Sl     0:00      \_ /usr/lib/firefox/firefox -contentproc -childID 2 ...
  30442 ?        Sl     0:00      \_ /usr/lib/firefox/firefox -contentproc -childID 3 ...
```

Управляющий терминал процесса, показанный в столбце TTY, используется для доставки ему интерактивных *сигналов (см. разд. 4.8)* при вводе управляющих символов *(см. разд. 2.3)* intr ^C, quit ^\ и пр. У части процессов ❶, ❻ управляющий терминал отсутствует, потому что они выполняют приложения, взаимодействующие с пользователем не посредством терминалов, а через графическую систему *(см. главу 7)*.

Процесс по своему определению изолирует свою программу от других выполняющихся программ, что затрудняет использование процессов для выполнения таких параллельных программ, ветви которых не являются полностью независимыми друг от друга и должны обмениваться данными. Использование предназначенных для этого средств межпроцессного взаимодействия *(см. разд. 4.9)* при интенсивном обмене приводит к обременению неоправданными накладными расходами, поэтому для эффективного выполнения таких параллельных программ используются легковесные процессы (LWP, light-weight processes), они же нити[1] (threads). Механизм

[1] Существует еще один (неудачный, на мой взгляд) перевод понятия thread на русский язык — поток. Во-первых, он конфликтует с переводом понятия stream — поток, а во-вторых, в отличие от stream, thread никуда не течет. А вот процесс (process) содержит в себе нити (thread) абсолютно таким же образом, как и обычная веревка состоит из... нитей.

нитей позволяет переключать центральный процессор между параллельными ветвями одной программы, размещаемыми в *одном* (!) процессе. Нити никак не изолированы друг от друга, и им доступны абсолютно все ресурсы своего процесса, поэтому задача обмена данными между нитями попросту отсутствует, т. к. все данные являются для них общими.

В примере из листинга 4.9 показаны нити процесса в BSD-формате вывода. Выбор процесса производится по его идентификатору **PID**, предварительно полученному командой **pgrep(1)** по имени программы, выполняющейся в искомом процессе. В выводе наличие нитей процесса отмечает флаг **l** (lwp) в столбце состояния STAT, а каждая строчка без идентификатора **PID** символизирует одну нить. Так как в многонитевой программе переключение процессора производится между нитями, то и состояния сна S, выполнения или ожидания R приписываются отдельным нитям.

Листинг 4.9. Нити процессов, BSD-формат вывода

```
fitz@ubuntu:~$ pgrep firefox
30315
fitz@ubuntu:~$ ps mp 30315
  PID TTY      STAT   TIME COMMAND
  PID TTY      STAT   TIME COMMAND
30315 ?        -      0:05 /usr/lib/firefox/firefox -new-window
    - -      ← Sl     0:03 -
                          ...          ...          ...
    - -        Sl     0:00 -
    - -        Sl     0:00 -
    - -        Sl     0:00 -
```

В листинге 4.10 показаны нити процесса в SYSV-формате вывода. Выбор процесса производится по имени его программы. Общий для всех нитей идентификатор их процесса отображается в столбце PID, уникальный идентификатор каждой нити — в столбце LWP (иногда называемый TID, thread identifier), а имя процесса (или собственное имя нити, если задано) — в столбце CMD.

Листинг 4.10. Нити процессов, SYSV-формат вывода

```
fitz@ubuntu:~$ ps -LC firefox
    PID ← LWP TTY          TIME CMD
  30315 30315 ?        00:00:04 firefox
  30315 30320 ?        00:00:00 gmain
  30315 30321 ?        00:00:00 gdbus
                          ...          ...          ...
```

```
30315 30328 ?        00:00:00 Socket Thread
30315 30332 ?        00:00:00 Cache2 I/O
30315 30333 ?        00:00:00 Cookie
                      ...          ...          ...
30315 30371 ?        00:00:00 HTML5 Parser
30315 30373 ?        00:00:00 DNS Resolver #3
                      ...          ...          ...
```

4.3. Порождение процессов и нитей, запуск программ

Несмотря на очевидные различия, историю[1] возникновения и развития, нити и процессы объединяет общее назначение — они являются примитивами выполнения некоторого набора последовательных инструкций.

Процессы выполняют или разные последовательные программы целиком, или ветви одной параллельной программы, но в изолированном окружении со своим «частным» (private) набором ресурсов. Нити, наоборот, выполняют ветви одной параллельной программы в одном окружении с «общим» (shared) набором ресурсов. В многозадачном ядре Linux вообще используется универсальное понятие «задача», которая может иметь как общие ресурсы (память, открытые файлы и т. д.) с другими задачами, так и частные ресурсы для своего собственного использования.

Порождение нового процесса (рис. 4.1, *а*) реализуется при помощи системного вызова **fork(2)**, в результате которого ядро операционной системы создает новый дочерний (child) процесс PID_2 — полную копию (**COPY**) процесса-родителя (parent) PID_1. Вся (за небольшими исключениями) память процесса — состояние, свойства, атрибуты (кроме идентификатора **PID**) и даже содержимое (программа с ее библиотеками) — наследуется дочерним процессом. Даже выполнение порожденного и порождающего процесса продолжится с одной и той же инструкции их одинаковой программы. Такое клонирование обычно используют параллельные программы с ветвями *(см. разд. 4.3.1)*, выполняющимися в дочерних *процессах*.

Уничтожение процесса (например, при штатном окончании программы) производится с помощью системного вызова **exit(3)**. При этом родительскому процессу доставляется сигнал **SIGCHILD**, оповещающий о завершении дочернего процесса *(см. разд. 4.8)*. Статус завершения **status**, переданный дочерним процессом через аргументы **exit(3)**, будет сохраняться ядром до момента его востребования родительским процессом при помощи системного вызова **wait(2)**, а весь этот промежуток времени дочерний процесс будет находиться в состоянии **Z** (**z**ombie[2] (см. столбец **STAT** в листинге 4.8). Родительский процесс может завершиться раньше

[1] Откровенно говоря, нити, в общем, появились в операционных системах раньше, чем изолированные UNIX-процессы, в которые со временем вернулись UNIX-нити...

ше своих дочерних процессов, тогда логично предположить, что все «осиротевшие» процессы окажутся зомби по завершении, потому как просто некому будет востребовать их статус завершения. На самом деле этого не происходит, потому что «осиротевшим» процессам назначается приемный родитель, в качестве которого выступает прародитель всех процессов **init(1)** с идентификатором **PID = 1**.

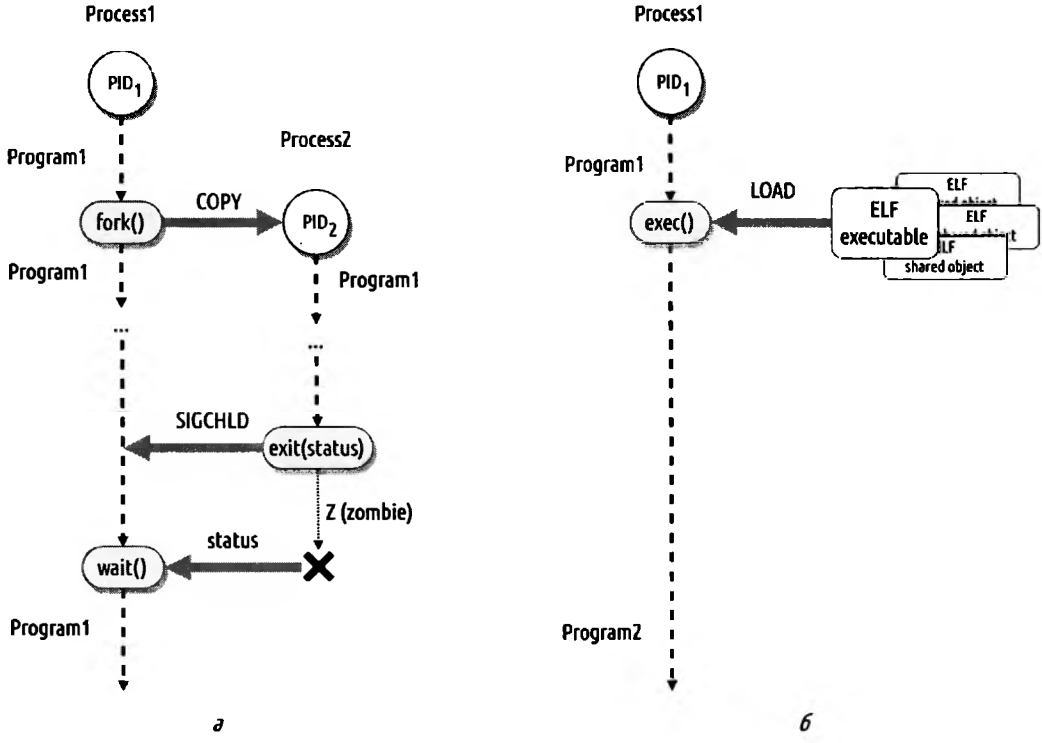


Рис. 4.1. Порождение процессов *(а)* и запуск программ *(б)*

Запуск новой программы (см. рис. 4.1, *б*) реализуется при помощи системного вызова **exec(3)**, в результате которого содержимое процесса PID_1 полностью замещается запускаемой программой и библиотеками, от которых она зависит, а свойства и атрибуты (включая идентификатор PID) остаются неизменными. Такое замещение обычно используется программами, устанавливающими нужные значения свойств и атрибутов процесса и подготавливающими ресурсы процесса к выполнению запускаемой программы. Например, обработчик терминального доступа **getty(8)** (см. главу 2) открывает заданный терминал, устанавливает режимы работы порта терминала, перенаправляет на терминал стандартные потоки ввода-вывода, а затем замещает себя программой аутентификации **login(1)**.

Для запуска новой программы в новом процессе используются оба системных вызова **fork(2)** и **exec(3)** согласно принципу **fork-and-exec** «раздвоиться и запустить», показанного на рис. 4.2. В примере из листинга 4.8 дерево процессов сформиро-

вано именно на основе дочерне-родительских отношений между процессами, формирующимися при использовании принципа **fork-and-exec**. Например, командный интерпретатор **bash(1)** по командам **ps fx** или **man ps** порождает дочерние процессы ❷ и замещает их программами **ps(1)** и **man(1)**. Тем же образом действует ❶ графический эмулятор терминала **gnome-terminal-server** — запуская новый сеанс пользователя ❶ на каждой из своих вкладок, он замещает свои дочерние процессы программой интерпретатора **bash(1)**.

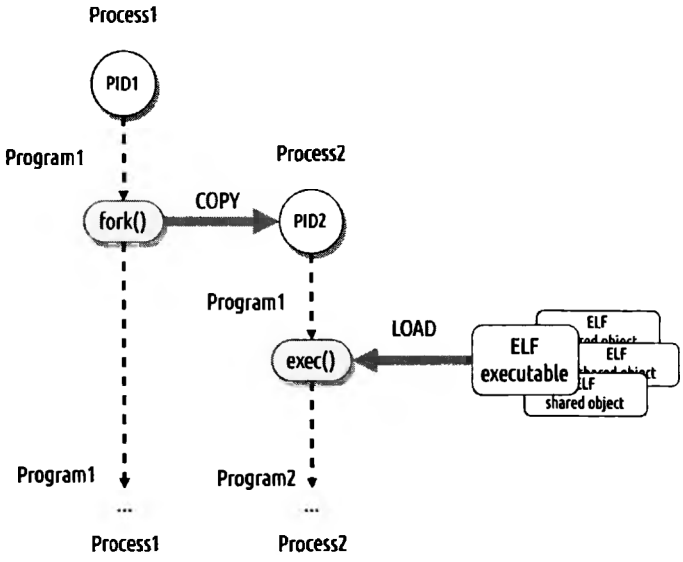


Рис. 4.2. Запуск программы в отдельном процессе

Листинг 4.11 иллюстрирует команду интерпретатора, запущенную в «фоновом» режиме при помощи конструкции асинхронного списка (*см. разд. 5.6*). Аналогично всем предыдущим командам, интерпретатор использует **fork-and-exec** для запуска программы в дочернем процессе с идентификатором **23228**, но не дожидается его завершения при помощи системного вызова **wait(2)**, как обычно, а немедленно ❶ продолжает интерактивное взаимодействие с пользователем, сообщив ему **PID** порожденного процесса и «номер задания» **[1]** команды «заднего фона» (*см. разд. 4.8.1*). Оповещение о завершении своего дочернего процесса интерпретатор получит позже, при помощи сигнала **SIGCHLD**, и отреагирует соответствующим сообщением ❷ об окончании команды «заднего фона».

Листинг 4.11. Фоновое выполнение программ

```
fitz@ubuntu:~$ dd if=/dev/dvd of=plan9.iso &
❶ [1] 23228
```

```
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
23025 pts/1    S      0:00 -bash
23228 pts/1    R      1:23  \_ dd if=/dev/dvd of=plan9.iso
23230 pts/1    R+     0:00  \_ ps f
fitz@ubuntu:~$
                              ...          ...          ...
fitz@ubuntu:~$ 586896+0 записей получено
586896+0 записей отправлено
300490752 байт (300 MB, 286 MiB) скопирован, 14,6916 с, 20,5 MB/с

❷ [1]+  Завершён              dd if=/dev/dvd of=plan9.iso
```

В листинге 4.12 показана конвейерная *(см. разд. 5.3)* конструкция интерпретатора, при помощи которой осуществляется поиск самого большого файла с суффиксом **.html** вниз по дереву каталогов, начиная с **/usr/share/doc**. Эта конструкция реализуется при помощи **fork-and-exec** четырьмя параллельно порожденными дочерними процессами интерпретатора, в каждом из которых запущена программа соответствующей части конвейера, при этом дочерние процессы связаны неименованным каналом **pipe(2)** — простейшим средством межпроцессного взаимодействия *(см. разд. 4.9)*. Встроенная команда интерпретатора **wait** реализует одноименный системный вызов **wait(2)** и используется для ожидания окончания всех дочерних процессов конвейера, целиком запущенного в «фоновом» режиме.

Листинг 4.12. Параллельный запуск взаимодействующих программ

```
          ①↴                                    ②↴          ③↴          ④↴
fitz@ubuntu:~$ find /usr/share/doc -type f -name '*.html' | xargs -n1 wc -l | sort -k 1 -nr | head -1 &
                                                  ↳          ↳          ↳     ↳

[1] 12827
fitz@ubuntu:~$ ps fj
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
11715 11716 11716  9184 pts/0    14699 S     1006   0:01 -bash
11716 12824 12824  9184 pts/0    14699 R     1006   0:00  \_ find ... -type f -name *.html
11716 12825 12824  9184 pts/0    14699 R     1006   0:00  \_ xargs -n1 wc -l
11716 12826 12824  9184 pts/0    14699 S     1006   0:00  \_ sort -k 1 -nr
11716 12827 12824  9184 pts/0    14699 S     1006   0:00  \_ head -1
11716 14699 14699  9184 pts/0    14699 R+    1006   0:00  \_ ps fj
fitz@ubuntu:~$ wait
15283 /usr/share/doc/xterm/xterm.log.html
[1]+  Завершён find /usr/share/doc -type f -name '*.html' | xargs -n1 wc -l | sort -k 1 -nr |
head -1
```

4.3.1. Параллельные многопроцессные программы

Как указывалось ранее, параллельные программы зачастую используют процессы для выполнения отдельных ветвей. В эту категорию часто попадают программы сетевых служб, например сервер баз данных W:[PostgreSQL], служба удаленного доступа W:[SSH] и подобные. Листинг 4.13 иллюстрирует программу **postgres(1)**, выполняющуюся в шести параллельных процессах, один из которых — диспетчер ❶, четыре служебных ❷ и еще один ❸ вызван подключением пользователя **fitz** к одноименной базе данных **fitz**. При последующих подключениях пользователей к серверу будут порождены дополнительные дочерние процессы для обслуживания их запросов — по одному на каждое подключение.

Листинг 4.13. Параллельные многопроцессные сервисы

```
   fitz@ubuntu:~$ ps f -C postgres
     PID TTY      STAT   TIME COMMAND
❶   6711 ?        S      0:00 /usr/lib/postgresql/11/bin/postgres -D /var/lib/postgresql...
❷   6713 ?        Ss     0:00  \_ postgres: 11/main: checkpointer
|   6714 ?        Ss     0:00  \_ postgres: 11/main: background writer
|   6715 ?        Ss     0:00  \_ postgres: 11/main: walwriter
|   6716 ?        Ss     0:00  \_ postgres: 11/main: autovacuum launcher
|   6717 ?        Ss     0:00  \_ postgres: 11/main: stats collector
↳   6718 ?        Ss     0:00  \_ postgres: 11/main: logical replication launcher
❸   9443 ?        Ss     0:00  \_ postgres: 11/main: fitz ◄• fitz [local] idle
   fitz@ubuntu:~$ ssh ubuntu
   fitz@ubuntu's password:
                                  ...          ...          ...
   Last login: Sat Nov 21 13:29:33 2015 from localhost
   fitz@ubuntu:~$ ps f -C sshd
     PID TTY      STAT   TIME COMMAND
①    655 ?        Ss     0:00 /usr/sbin/sshd -D
②  21975 ?        Ss     0:00  \_ sshd: fitz [priv]
③  22086 ?        S      0:00      \_ sshd: ◄• fitz@pts/1
   fitz@ubuntu:~$ ^Dвыход
   Connection to ubuntu closed.
```

Аналогично, при удаленном доступе по протоколу SSH программа **sshd(8)**, работая в качестве диспетчера ① в одном процессе, на каждое подключение порождает один свой клон ②, который, выполнив аутентификацию и авторизацию пользователя в системе, порождает еще один свой клон ③, имперсонирующийся в пользователя и обслуживающий его запросы.

4.3.2. Параллельные многонитевые программы

Для управления нитями в Linux используют стандартный POSIX-интерфейс **pthreads(7)**, реализующийся библиотекой W:[NPTL], которая является частью библиотеки **libc**. Интерфейс предоставляет «нитевой» вызов создания нити **pthread_create(3)**, который является условным аналогом «процессных» **fork(2)** и **exec(3)**, вызов завершения и уничтожения нити **pthread_exit(3)**, условно аналогичный **exit(2)**, и вызов для получения статуса завершения нити **pthread_join(3)**, условно аналогичный **wait(2)**.

В качестве типичных примеров применения нитей можно привести сетевые сервисы, которые для параллельного обслуживания клиентских запросов используют нити вместо процессов. Например, WEB-сервер **apache(8)**, как показано в листинге 4.14, использует два многонитевых процесса по 27 нитей в каждом, что позволяет экономить память (за счет работы всех нитей процесса с общей памятью) при обслуживании большого количества одновременных клиентских подключений.

Листинг 4.14. Параллельные многонитевые сервисы

```
fitz@ubuntu:~$ ps f -C apache2
  PID TTY      STAT   TIME COMMAND
10129 ?        Ss     0:00 /usr/sbin/apache2 -k start
10131 ?        Sl     0:00  \_ /usr/sbin/apache2 -k start
10132 ?        Sl     0:00  \_ /usr/sbin/apache2 -k start
fitz@ubuntu:~$ ps fo pid,nlwp,cmd -C apache2
  PID NLWP CMD
10129    1 /usr/sbin/apache2 -k start
10131   27  \_ /usr/sbin/apache2 -k start
10132   27  \_ /usr/sbin/apache2 -k start

fitz@ubuntu:~$ ps -fLC rsyslogd
UID        PID  PPID  LWP  C NLWP STIME TTY          TIME CMD
syslog     606     1  606  0    4 ноя18 ?        00:00:00 /usr/sbin/rsyslogd -n -iNONE
syslog     606     1  680  0    4 ноя18 ?        00:00:00 /usr/sbin/rsyslogd -n -iNONE
syslog     606     1  681  0    4 ноя18 ?        00:00:00 /usr/sbin/rsyslogd -n -iNONE
syslog     606     1  682  0    4 ноя18 ?        00:00:00 /usr/sbin/rsyslogd -n -iNONE
```

Аналогично, сервис централизованной журнализации событий **rsyslogd(8)** использует нити для параллельного сбора событийной информации из разных источников, ее обработки и журнализации. Одна нить считывает события ядра из **/proc/kmsg**, вторая принимает события других служб из файлового сокета **/run/systemd/journal/syslog** (**/dev/log** в ранних, до **systemd** системах), третья фильтрует поток принятых

событий и записывает в журнальные файлы каталога **/var/log/*** и т. д. Параллельная обработка потоков поступающих событий при помощи нитей производится с минимально возможными накладными расходами, что позволяет достигать колоссальной производительности по количеству обрабатываемых сообщений в единицу времени.

Распараллеливание используется не только для псевдоодновременного выполнения ветвей параллельной программы, но и для их настоящего одновременного выполнения несколькими центральными процессорами. В примере из листинга 4.15 показано, как сокращается время сжатия ISO-образа файла при использовании параллельного упаковщика **pbzip2(1)** по сравнению с последовательным **bzip2(1)**. Для измерения времени упаковки применяется встроенная команда интерпретатора **time**, при этом сначала измеряется время упаковки ❶ и время распаковки ❷ последовательным упаковщиком, а затем — время упаковки ① и время распаковки ② параллельным упаковщиком. Команды упаковки запускаются на «заднем фоне», оценивается наличие процессов и нитей паковщиков, после чего они переводятся на «передний фон» встроенной командой интерпретатора **fg** (**fore**ground) и оцениваются затраты времени.

Листинг 4.15. Параллельные многонитевые утилиты

```
fitz@ubuntu:~$ ls -lh plan9.iso
-rw-r--r-- 1 fitz fitz 287M нояб. 28 15:47 plan9.iso
❶ fitz@ubuntu:~$ time bzip2 plan9.iso &
[1] 5545
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
 4637 pts/0    S      0:00 -bash
 5545 pts/0    S      0:00  \_ -bash
 5546 pts/0    R      0:12  |   \_ bzip2 plan9.iso
 5548 pts/0    R+     0:00  \_ ps f
fitz@ubuntu:~$ ps -fLp 5546
UID        PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
fitz      5546  5545  5546 96    1 10:50 pts/0    00:00:22 bzip2 plan9.iso
fitz@ubuntu:~$ fg
time bzip2 plan9.iso
                                  ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪
real 0m54.780s
user 0m51.772s
sys  0m0.428s
fitz@ubuntu:~$ ls -lh plan9.iso.bz2
-rw-r--r-- 1 fitz fitz 89M нояб. 28 15:47 plan9.iso.bz2
```

```
❷ fitz@ubuntu:~$ time bzip2 -d plan9.iso.bz2
                                  ① ① ① ① ① ① ① ① ① ① ①
  real 0m20.705s ←
  user 0m19.044s
  sys  0m1.168s
①  fitz@ubuntu:~$ time pbzip2 plan9.iso &
  [1] 5571
  fitz@ubuntu:~$ ps f
    PID TTY      STAT   TIME COMMAND
   4637 pts/0    S      0:00 -bash
   5571 pts/0    S      0:00  \_ -bash
   5572 pts/0    Sl ←   0:03  |   \_ pbzip2 plan9.iso
   5580 pts/0    R+     0:00  \_ ps f
  fitz@ubuntu:~$ ps -fLp 5572
  UID        PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
  fitz      5572  5571  5578 92 ←- 8 10:52 pts/0    00:00:43 pbzip2 plan9.iso
                                ...          ...             ...
  fitz      5572  5571  5579  1    8 10:52 pts/0    00:00:00 pbzip2 plan9.iso
  fitz@ubuntu:~$ fg
  time pbzip2 plan9.iso
                                  ① ① ① ① ① ① ① ① ① ① ①
  real 0m24.259s ←
  user 1m22.940s
  sys  0m1.888s
  fitz@ubuntu:~$ ls -lh plan9.iso.bz2
  -rw-r--r-- 1 fitz fitz 89M нояб. 28 15:47 plan9.iso.bz2
②  fitz@ubuntu:~$ time pbzip2 -d plan9.iso.bz2
                                  ① ① ① ① ① ① ① ① ① ① ①
  real 0m7.384s ←
  user 0m25.972s
  sys  0m1.396s
```

В результате оценки оказывается, что последовательный упаковщик **bzip2(1)** использует один однонитевой процесс и затрачивает ≈54,7 с реального времени на упаковку, из них ≈51,7 с проводит в пользовательском режиме **user** и лишь ≈0,4 с в режиме ядра **sys** (выполняя системные вызовы, например **read(2)** или **write(2)**). Соотношение между временем режимов говорит о вычислительном характере программы, т. е. о существенном превалировании времени вычислительных операций упаковки над временем операций ввода-вывода для чтения исходных данных и записи результатов (что подтверждает анализ разд. 4.2). Это означает, что нагрузка

последовательного упаковщика на центральный процессор (в случае, если бы он был единственный) близка к максимальной, и его параллельная реализация для псевдоодновременного выполнения ветвей (которые практически никогда не спят) лишена смысла.

Параллельный упаковщик **pbzip2(1)** использует один многонитевой процесс из восьми нитей и затрачивает ≈24,4 с реального времени на упаковку, при этом ≈1 мин 22,9 с (!) проводит в пользовательском режиме и ≈1,8 с в режиме ядра. Прирост производительности упаковки и, как следствие, сокращение времени упаковки достигаются за счет настоящего параллельного выполнения нитей на нескольких процессорах (разных ядрах процессора). Соотношение между реальным временем упаковки и суммарно затраченным временем режима пользователя, которое примерно в 3 раза больше, означает использование в среднем трех процессоров для параллельного выполнения вычислительных операций упаковки.

4.3.3. Двойственность процессов и нитей Linux

Как указывалось ранее, процессы и нити в ядре Linux сводятся к универсальному понятию «задача». Задача, все ресурсы которой (память, открытые файлы и т. д.) используются совместно с другими такими же задачами, является нитью. И наоборот, процессами являются такие задачи, которые обладают набором своих частных, индивидуальных ресурсов.

Универсальный системный вызов **clone(2)** позволяет указать, какие ресурсы станут общими в порождаемой и порождающей задачах, а какие — частными. Системные вызовы порождения **POSIX**-процессов **fork(2)** и **POSIX**-нитей **pthread_create(3)** оказываются в Linux всего лишь «обертками» над **clone(2)**, что проиллюстрировано в листингах 4.16 и 4.17.

В примере из листинга 4.16 архиватор **tar(1) PID = 11801** создает при помощи системного вызова **clone(2)** дочерний процесс **PID = 11802**, в который помещает программу компрессора **gzip(1)**, используя системный вызов **execve(2)**. В результате параллельной работы двух взаимодействующих процессов будет создан компрессированный архив **docs.tgz** каталога **/usr/share/doc**.

Листинг 4.16. Системный вызов clone — порождение нового процесса

```
fitz@ubuntu:~$ strace -fe clone,fork,execve tar czf docs.tgz /usr/share/doc
execve("/usr/bin/tar", ["tar", "czf", "docs.tgz", "/usr/share/doc"], ... ) = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID| ...) = 12403
tar: Удаляется начальный `/' из имен объектов
strace: Process 12403 attached
[pid 12403] execve("/bin/sh", ["/bin/sh", "-c", "gzip"], 0x7ffd8dd597c0 ...) = 0
```

```
[pid 12403] clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID| ...) = 12404
strace: Process 12404 attached
[pid 12404] execve("/usr/bin/gzip", ["gzip"], 0x55e2e45bbb48 /* 35 vars */) = 0
                                    ⓿ ❶ ❷ ❸ ❹ ❺ ❻ ❼ ❽ ❾
                                ...         ...          ...
+++ exited with 0 +++
```

В примере из листинга 4.17 компрессор **pbzip2(1)** создает при помощи системного вызова **clone(2)** семь «дочерних» нитей ❶...❼ PID = 12514→12520, которые имеют общую память CLONE_VM, общие открытые файлы CLONE_FILES и прочие общие ресурсы.

Листинг 4.17. Системный вызов clone — порождение новой нити

```
fitz@ubuntu:~$ strace -fe clone,fork,execve pbzip2 plan9.iso
execve("/usr/bin/pbzip2", ["pbzip2", "plan9.iso"], 0x7ffd28884938 /* 35 vars */) = 0
❶ clone(child_stack=0x7f1d46d38fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|... ) = 12514
5✕              ...              ...
❼ clone(child_stack=0x7f1d46537fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|...) = 12520
strace: Process 12520 attached
5✕              ...              ...
strace: Process 12514 attached
[pid 12514] +++ exited with 0 +++
5✕              ...              ...
[pid 12520] +++ exited with 0 +++
+++ exited with 0 +++
```

4.4. Дерево процессов

Процессы, попарно связанные дочерне-родительскими отношениями, формируют дерево процессов операционной системы. Первый процесс **init(1)**, называемый *прародителем процессов*, порождается ядром операционной системы после инициализации и монтирования корневой файловой системы, откуда и считывается программа **/sbin/init** (в современных системах является символической ссылкой на актуальный **/lib/systemd/systemd**). Прародитель процессов всегда имеет PID = 1, а его основной задачей является запуск разнообразных системных служб, включая запуск обработчиков *(см. разд. 2.2.1)* алфавитно-цифрового терминального доступа **getty(8)**, менеджера дисплеев *(см. разд. 7.5.3)* графического доступа, службы дистанционного доступа SSH *(см. разд. 6.4.1)* и прочих *(см. главу 10)*. Кроме того, **systemd(1)** назначается приемным родителем для «осиротевших» процессов, а также отслеживает аварийные завершения запускаемых им служб и перезапускает их.

В примере из листинга 4.18 показано дерево процессов, построенное при помощи специальной команды **pstree(1)**, а в листинге 4.19 — «классическое» представление дерева процессов при помощи команды **ps(1)**.

Листинг 4.18. Дерево процессов

```
fitz@ubuntu:~$ pstree -cnAhT
systemd-+-systemd-journal
    ❷ |-systemd-udevd
    ❸ |-systemd-resolve
    ❹ |-rsyslogd
                          ...          ...          ...
      |-gdm3---gdm-session-wor-+-gdm-session-wor
      |                        |-gdm-x-session-+-Xorg
      |                        |               `-gnome-session-b
                          ...          ...          ...
      |-systemd-+-(sd-pam)
                          ...          ...          ...
      |         |-gnome-terminal--+-bash---man---pager ❺
      |         |                 `-bash
                          ...          ...          ...
    ❻ |-postgres-+-postgres
      |          |-postgres
      |          |-postgres
      |          |-postgres
      |          |-postgres
      |          `-postgres
    ❼ |-apache2-+-apache2
      |         `-apache2
                          ...          ...          ...
    ❽ |-sshd-+-sshd---sshd---bash
      |      `-sshd---sshd---bash
                          ...          ...          ...
      |-agetty
      |-login---bash---pstree ❶
                          ...          ...          ...
```

Процессы операционной системы принято классифицировать на *системные* (ядерные), *демоны* и *прикладные*, исходя из их назначения и свойств (см. листинг 4.19).

Прикладные процессы ❺ выполняют обычные *пользовательские* программы (например, утилиту **man**), для чего им выделяют индивидуальную память, объем которой указан в столбце **VSZ** вывода команды **ps(1)**. Такие процессы обычно интерактивно взаимодействуют с пользователем посредством управляющего терминала (за исключением графических программ), указанного в столбце **TTY**.

Демоны (daemons) ❷ выполняют *системные* программы, реализующие те или иные службы операционной системы. Например, **cron(8)** реализует службу периодического выполнения заданий, **atd(8)** — службу отложенного выполнения заданий, **rsyslogd(8)** — службу централизованной журнализации событий, **sshd(8)** — службу дистанционного доступа, **systemd-udevd(8)** — службу «регистрации» подключаемых устройств, и т. д. Демоны запускаются на ранних стадиях загрузки операционной системы и взаимодействуют с пользователем не интерактивно при помощи терминала, а опосредованно — при помощи своих утилит. Таким образом, отсутствие управляющего терминала в столбце TTY отличает[1] их от прикладных процессов.

Листинг 4.19. Процессы ядра, демоны, прикладные процессы

```
fitz@ubuntu:~$ ps faxu
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         2  0.0  0.0      0     0 ?        S    ноя18   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        I<   ноя18   0:00  \_ [rcu_gp]
root         4  0.0  0.0      0     0 ?        I<   ноя18   0:00  \_ [rcu_par_gp]
root         6  0.0  0.0      0     0 ?        I<   ноя18   0:00  \_ [kworker/0:0H...]
root         8  0.0  0.0      0     0 ?        I<   ноя18   0:00  \_ [mm_percpu_wq]
root         9  0.0  0.0      0     0 ?        S    ноя18   0:09  \_ [ksoftirqd/0]
                                  ...              ...           ...
root         1  0.0  0:2 168400 11684 ?        Ss   ноя18   0:12 /sbin/init splash
                                  ...              ...           ...
root       333  0.0  0.1  21844  5348 ?        Ss   ноя18   0:07 /lib/systemd/systemd-udevd
syslog     606  0.0  0.1 224360  4244 ?        Ssl  ноя18   0:01 /usr/sbin/rsyslogd -n -i...
                                  ...              ...           ...
root       649  0.0  0.0  20320  3036 ?        Ss   ноя18   0:00 /usr/sbin/cron -f
daemon     675  0.0  0.0   3736  2184 ?        Ss   ноя18   0:00 /usr/sbin/atd -f
                                  ...              ...           ...
root     21545  0.0  0.0   5560  3420 tty4     Ss   ноя18   0:00 /bin/login -p --
fitz     28152  0.0  0.0   2600  1784 tty4     S    01:38   0:00  \_ -sh
fitz     28162  0.0  0.0  12948  3584 tty4     S+   01:38   0:00      \_ bash
finn     12989  0.2  0.0  12092  3988 tty4     S+   13:47   0:00          \_ man ps
finn     13000  0.0  0.0  10764  2544 tty4     S+   13:47   0:00              \_ pager
```

Системные (ядерные) ❶ процессы[2] выполняют параллельные части *ядра* операционной системы, поэтому не обладают ни индивидуальной виртуальной памятью **VSZ**, ни управляющим терминалом **TTY**. Более того, ядерные процессы не выполняют отдельную программу, загружаемую из **ELF**-файла, поэтому их имена **COMMAND** яв-

[1] Зачастую демоны имеют суффикс **d** в конце названия, например **sshd** — это **s**ecure **sh**ell **d**aemon, а **rsyslogd** — **r**ocket **sys**tem **log**ging **d**aemon, и т. д.

[2] Правильнее — ядерные нити, т. к. выполняются они в общей памяти ядра операционной системы.

ляются условными и изображаются в квадратных скобках, а кроме того, они имеют особое состояние `I` в столбце **STAT**.

4.5. Атрибуты процесса

Процесс в операционной системе является основным активным субъектом, взаимодействующим с окружающими его объектами — файлами и файловыми системами, другими процессами, устройствами и пр. Возможности процесса выполнять те или иные действия по отношению к другим объектам определяются его специальными свойствами — атрибутами процесса.

4.5.1. Маркеры доступа

Возможности процесса по отношению к объектам, доступ к которым разграничивается при помощи дискреционных механизмов (в частности, к файлам дерева каталогов — *см. разд. 3.5*) определяются значениями его атрибутов, формирующих его DAC-маркер доступа, а именно — атрибутами **RUID**, **RGID**, **EUID**, **EGID**, см. **credentials(7)**.

Эффективные идентификаторы **EUID** (**e**ffective **u**ser **id**entifier) и **EGID** (**e**ffective **g**roup **id**entifier) указывают на «эффективных» пользователя и группу, использующихся дискреционными механизмами для определения прав доступа процесса к файлам и другим объектам согласно назначенному им режиму или списку доступа *(см. разд. 3.5.3)*. Атрибуты **RUID** (**r**eal **u**ser **id**entifier) и **RGID** (**r**eal **g**roup **id**entifier) указывают на «настоящих» пользователя и группу, «управляющих» процессом.

Первому процессу пользовательского сеанса (в случае регистрации в системе с использованием алфавитно-цифрового терминала — командному интерпретатору) назначают атрибуты **RUID/EUID** и **RGID/EGID** равными идентификаторам зарегистрировавшегося пользователя и его первичной группы. Последующие процессы пользовательского сеанса наследуют значения атрибутов, т. к. порождаются в результате клонирования при помощи **fork(2)**. В примере из листинга 4.20 при помощи команды **id(1)** показаны значения **EUID/EGID** пользовательского сеанса и их наследование от командного интерпретатора, что явным образом подтверждает команда **ps(1)**.

Листинг 4.20. DAC-маркер доступа процесса — атрибуты RUID, EUID, RGID, EGID

```
fitz@ubuntu:~$ id
uid=1006(fitz) gid=1008(fitz) группы=1008(fitz)
fitz@ubuntu:~$ ps fo euid,ruid,egid,rgid,user,group,tty,cmd
 EUID  RUID  EGID  RGID  USER  GROUP  TT       CMD
1006  1006  1008  1008  fitz  fitz   pts/2    -bash
1006  1006  1008  1008  fitz  fitz   pts/2     \_ ps fo euid,uid,egid,...,tty,cmd
```

Изменение идентификаторов **EUID/EGID** *процесса* происходит при срабатывании механизма неявной передачи полномочий, основанном на дополнительных атрибутах **SUID/SGID** файлов *программ*. При запуске таких программ посредством системного вызова **exec(3)** атрибуты **EUID/EGID** запускающего процесса устанавливаются равными идентификаторам **UID/GID** владельца запускаемой программы. В результате процесс, в который будет загружена такая программа, будет обладать правами владельца программы, а не правами пользователя, запустившего эту программу.

В листинге 4.21 приведен типичный пример использования механизма неявной передачи полномочий при выполнении команд **passwd(1)** и **wall(1)**. При смене пароля пользователем при помощи программы **/usr/bin/passwd** ее процесс получает необходимое право записи ① в файл **/etc/shadow** (см. листинг 3.40) в результате передачи полномочий ❶ суперпользователя **root** (**UID=0**). При передаче широковещательного сообщения всем пользователям при помощи **/usr/bin/wall** необходимо иметь право записи ② в их файлы устройств **/dev/tty***, которое появляется ❷ в результате передачи полномочий группы **tty** (GID = 5).

Листинг 4.21. Атрибуты файла SUID/SGID и атрибуты процесса RUID, EUID, RGID, EGID

```
fitz@ubuntu:~$ who
fitz     pts/0        2019-11-22 00:52 (:0.0)
fitz     pts/1        2019-11-22 00:53 (:0.0)
fitz     pts/2        2019-11-22 01:06 (:0.0)

fitz@ubuntu:~$ ls -la /etc/shadow /dev/pts/*
crw--w---- 1 fitz tty    136, 2 ноя 19 12:00 /dev/pts/1
② crw--w---- 1 fitz tty    136, 3 ноя 19 13:53 /dev/pts/2
c--------- 1 root root     5, 2 ноя 17 03:30 /dev/pts/ptmx
① -rw-r----- 1 root shadow   1647 ноя 19 12:27 /etc/shadow

fitz@ubuntu:~$ ls -l /usr/bin/passwd /usr/bin/wall
-rwsr-xr-x 1 ☛ root   root 67992 авг 29 16:00 /usr/bin/passwd
-rwxr-sr-x 1    root ☛ tty 35048 авг 21 16:19 /usr/bin/wall

fitz@ubuntu:~$ ls -ln /usr/bin/passwd /usr/bin/wall
-rwsr-xr-x 1 ☛ 0    0 67992 авг 29 16:00 /usr/bin/passwd
-rwxr-sr-x 1    0 ☛ 5 35048 авг 21 16:19 /usr/bin/wall

fitz@ubuntu:~$ ps ft pts/1,pts/2 o pid,ruid,rgid,euid,egid,tty,cmd
  PID  RUID  RGID  EUID  EGID TT       CMD
27883  1006  1008  1006  1008 pts/2    bash
```

```
❷ 27937  1006  1008  1006  ☛ 5 pts/2     \_ wall
  27124  1006  1008  1006  1008 pts/1    bash
❶ 27839  1006  1008  ☛ 0  1008 pts/1     \_ passwd
```

По отношению к объектам, доступ к которым ограничивается при помощи мандатных механизмов *(см. разд. 3.6)*, возможности процесса определяются значениями его MAC-маркера доступа, а именно — атрибутом мандатной метки **LABEL**. Как и **RUID/EUID/RGID/EGID**, атрибут **LABEL** назначается первому процессу сеанса пользователя явным образом, а затем наследуется при клонировании процессами-потомками от процессов-родителей. В примере из листинга 4.22 при помощи команды **id(1)** показан атрибут **LABEL** сеанса пользователя, а при помощи команды **ps(1)** — его явное наследование от процесса-родителя. Аналогично изменениям **EUID/EGID** процесса, происходящим при запуске **SUID**-ной/**SGID**-ной программы, изменение метки **LABEL** *процесса* происходит (согласно мандатным правилам ❶) в системном вызове **exec(3)** при запуске *программы*, помеченной соответствующей мандатной меткой файла. Так, например, при запуске программы **/usr/sbin/dhclient** с типом **dhcpc_exec_t** ее мандатной метки ❷ процесс приобретает тип **dhcpc_t** своей мандатной метки ❹, в результате чего существенно ограничивается в правах доступа к разным объектам операционной системы.

Листинг 4.22. MAC-маркер доступа процесса — мандатная метка selinux

```
fitz@ubuntu:~$ ssh lich@fedora
lich@fedora's password:
Last login: Sat Nov 21 14:25:16 2015

[lich@centos ~]$ id -Z
staff_u:staff_r:staff_t:s0-s0:c0.c1023

[lich@centos ~]$ ps Zf
LABEL                             PID TTY      STAT   TIME COMMAND
staff_u:staff_r:staff_t:s0-s0:c0.c1023 31396 pts/0 Ss   0:00 -bash
staff_u:staff_r:staff_t:s0-s0:c0.c1023 31835 pts/0 R+   0:00  \_ ps Zf
staff_u:staff_r:staff_t:s0-s0:c0.c1023 31334 tty2 Ss+   0:00 -bash
[lich@centos ~]$ sesearch -T -t dhcpc_exec_t -c process
Found 19 semantic te rules:
                              ...          ...              ...
❶    type_transition NetworkManager_t dhcpc_exec_t : process dhcpc_t;
                              ...          ...              ...

[lich@centos ~]$ ls -Z /usr/sbin/dhclient
❷ -rwxr-xr-x. root root system_u:object_r:dhcpc_exec_t:s0 /usr/sbin/dhclient
```

```
[lich@centos ~]$ ps -ZC dhclient
LABEL                             PID TTY          TIME CMD
❸ system_u:system_r:dhcpc_t:s0    2120 ?        00:00:00 dhclient
  system_u:system_r:dhcpc_t:s0    4320 ?        00:00:00 dhclient
```

4.5.2. Привилегии

Еще одним важным атрибутом процесса, определяющим его возможности по использованию системных вызовов, являются привилегии процесса **cababilities(7)**. Например, обладание привилегией **CAP_SYS_PTRACE** разрешает процессам трассировщиков **strace(1)** и **ltrace(1)**, использующих системный вызов **ptrace(2)**, трассировать процессы *любых* пользователей (а не только «свои», **EUID** которых совпадает с **EUID** трассировщика). Аналогично, привилегия **CAP_SYS_NICE** разрешает изменять приоритет, устанавливать привязку к процессорам и назначать алгоритмы планирования *(см. разд. 4.6)* процессов и нитей *любых* пользователей, а привилегия **CAP_KILL** разрешает посылать сигналы *(см. разд. 4.8)* процессам *любых* пользователей.

Явная привилегия «владельца» **CAP_FOWNER** позволяет процессам изменять режим и списки доступа *(см. разд. 3.5.1)*, мандатную метку *(см. разд. 3.6.2)*, расширенные атрибуты *(см. разд. 3.7.1)* и флаги *(см. разд. 3.7.2)* любых файлов так, словно процесс выполняется от лица владельца файла. Привилегия **CAP_LINUX_IMMUTABLE** разрешает управлять флагами файлов **t**, **i**mmutable и **a**, **a**ppend *(см. разд. 3.7.2)*, а привилегия **CAP_SETFCAP** — устанавливать «файловые» привилегии *(см. далее)* запускаемых программ.

Необходимо отметить, что именно обладание полным набором привилегий делает пользователя **root** (**UID=0**) в Linux суперпользователем. И наоборот, обычный, непривилегированный пользователь (в смысле **UID≠0**) не обладает никакими явными[1] привилегиями. Назначение[2] привилегий процесса происходит при запуске программы при помощи системного вызова **exec(3)**, исполняемый файл которого помечен «файловыми» привилегиями.

В примере из листинга 4.23 иллюстрируется получение списка привилегий процесса при помощи утилиты **getpcaps**. Как и ожидалось, процесс **postgres** (**PID=6711**), работающий от лица обычного (непривилегированного, в смысле **UID≠0**) псевдопользователя **postgres**, не имеет ❶ никаких привилегий, а процесс **apache2** (**PID=10129**), работающий от лица суперпользователя **root** (**UID=0**), имеет полный ❷ набор привилегий. Однако процесс **NetworkManager** (**PID=646**) выполняется от лица

---

[1] Неявно он обладает привилегией владельца для всех своих объектов.

[2] Здесь допущено намеренное упрощение механизма наследования и назначения привилегий при **fork(2)** и **exec(3)** без потери смысла.

«суперпользователя», лишенного ❸ большинства своих привилегий, т. к. ему их умышленно уменьшили при его запуске (см. **systemd(1)** в *главе 8*) до минимально необходимого набора, достаточного для выполнения его функций[1].

Листинг 4.23. Привилегии (capabilities) процесса

```
fitz@ubuntu:~$ ps fo user,pid,cmd -C NetworkManager,postgres,apache2
USER       PID CMD
root     10129 /usr/sbin/apache2 -k start
www-data 10131  \_ /usr/sbin/apache2 -k start
www-data 10132  \_ /usr/sbin/apache2 -k start
postgres  6711 /usr/lib/postgresql/11/bin/postgres -D /var/lib/postgresql/11/main ...
postgres  6713  \_ postgres: 11/main: checkpointer
postgres  6714  \_ postgres: 11/main: background writer
postgres  6715  \_ postgres: 11/main: walwriter
postgres  6716  \_ postgres: 11/main: autovacuum launcher
postgres  6717  \_ postgres: 11/main: stats collector
postgres  6718  \_ postgres: 11/main: logical replication launcher
root       646 /usr/sbin/NetworkManager --no-daemon

fitz@ubuntu:~$ getpcaps 6711
❶ Capabilities for `6711': =
❷ fitz@ubuntu:~$ getpcaps 10129
Capabilities for `10129': =
cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,
cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_net_broadcast,cap_net_
admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_rawio,cap_sys_chroot,
cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,cap_sys_nice,cap_sys_resource,cap_
sys_time,cap_sys_tty_config,cap_mknod,cap_lease,cap_audit_write,cap_audit_control,cap_
setfcap,cap_mac_override,cap_mac_admin,cap_syslog,cap_wake_alarm,cap_block_suspend,
cap_audit_read+ep
❸ fitz@ubuntu:~$ getpcaps 646
Capabilities for `646': =
cap_dac_override,cap_kill,cap_setgid,cap_setuid,cap_net_bind_service,cap_net_admin,
cap_net_raw,cap_sys_module,cap_sys_chroot,cap_audit_write+ep
```

В листинге 4.24 показан типичный пример применения *отдельных* привилегий там, где классически применяется неявная передача *всех* полномочий суперпользователя при помощи механизма **SUID/SGID**. Например, «обычная» утилита **ping(1)** для выполнения своей работы должна создать «необработанный» **raw(7)** сетевой сокет, что

[1] Что способствует обеспечению защищенности операционной системы.

является с точки зрения ядра привилегированной операцией. В старых системах[1] программа **/bin/ping** наделялась атрибутом **SUID** ❶ и находилась во владении суперпользователя **root**, чьи права и передавались при ее запуске. С точки зрения защищенности системы это не соответствует здравому смыслу, подсказывающему наделять программы минимально необходимыми возможностями, достаточными для их функционирования. Для создания «необработанных» **raw(7)** и пакетных **packet(7)** сокетов достаточно только привилегии **CAP_NET_RAW**, а весь суперпользовательский набор привилегий более чем избыточен.

Листинг 4.24. Делегирование привилегий программы ping(1)

```
fitz@ubuntu-1804:~$ ls -l /bin/ping
❶ -rwsr-xr-x 1 root root 64424 Jun 28 11:05 /bin/ping
fitz@ubuntu-1804:~$ ping ubuntu-1804
PING ubuntu-1804 (127.0.1.1) 56(84) bytes of data.
64 bytes from ubuntu-1804 (127.0.1.1): icmp_req=1 ttl=64 time=0.074 ms
^C
--- ubuntu ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.074/0.074/0.074/0.000 ms

❶ fitz@ubuntu-1804:~$ sudo chmod u-s /bin/ping
fitz@ubuntu-1804:~$ ls -l /bin/ping
-rwxr-xr-x 1 root root 64424 Jun 28 11:05 /bin/ping
fitz@ubuntu-1804:~$ ping ubuntu-1804
🔒 ping: icmp open socket: Operation not permitted

❸ fitz@ubuntu-1804:~$ sudo setcap cap_net_raw+ep /bin/ping
fitz@ubuntu-1804:~$ getcap /bin/ping
/bin/ping = cap_net_raw+ep
fitz@ubuntu-1804:~$ ping ubuntu-1804
PING ubuntu (127.0.1.1) 56(84) bytes of data.
🔓 64 bytes from ubuntu (127.0.1.1): icmp_req=1 ttl=64 time=0.142 ms
^C
--- ubuntu ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.142/0.142/0.142/0.000 ms
```

[1] Актуально для **Ubuntu** ● до версии 18.10 включительно. Начиная с 19.04 все уже «правильно из коробки».

При отключении передачи полномочий ❷ программа **/bin/ping** лишается возможности выполнять свои функции, а при назначении ей при помощи команды **setcap(8)** «файловой» привилегии **CAP_NET_RAW** ❸ функциональность возвращается в полном объеме, т. к. приводит к установке «процессной» привилегии **CAP_NET_RAW** при запуске этой программы. Для просмотра привилегий, делегируемых при запуске программ, используется парная команда **getcap(8)**.

Аналогично, при использовании анализаторов сетевого трафика **tshark(1)** (листинг 4.25) и/или **wireshark(1)**, вызывающих для захвата сетевых пакетов утилиту **dumpcap(1)**, требуется открывать как «необработанные» **raw(7)**, так и пакетные **packet(7)** сетевые сокеты, что требует той же привилегии **CAP_NET_RAW**. Классический способ применения анализаторов пакетов состоит в использовании явной передачи *всех* полномочий суперпользователя (при помощи **su(1)** или **sudo(1)**) при их запуске, что опять не соответствует минимально необходимым и достаточным требованиям к разрешенным возможностям программ.

Листинг 4.25. Делегирование привилегий программе tshark(1)

```
fitz@ubuntu:~$ tshark
tshark: There are no interfaces on which a capture can be done
itz@ubuntu:~$ strace -fe execve tshark
execve("/usr/bin/tshark", ["tshark"], [/* 23 vars */]) = 0
Process 8951 attached
[pid  8951] execve("/usr/bin/dumpcap", ["/usr/bin/dumpcap", "-D", "-Z", "none"],...) = 0
Process 8951 detached
--- SIGCHLD (Child exited) @ 0 (0) ---
tshark: There are no interfaces on which a capture can be done
fitz@ubuntu:~$ ls -la /usr/bin/dumpcap
-rwxr-xr-x 1 root root 104688 Sep  5 19:43 /usr/bin/dumpcap
fitz@ubuntu:~$ getcap /usr/bin/dumpcap
☛
fitz@ubuntu:~$ sudo setcap cap_net_raw+ep /usr/bin/dumpcap
fitz@ubuntu:~$ getcap /usr/bin/dumpcap
/usr/bin/dumpcap = cap_net_raw+ep
fitz@ubuntu:~$ tshark -i wlan0
Capturing on wlan0
  0.307205 fe80::895d:9d7d:f0b3:a372 -> ff02::1:ff96:2df6 ICMPv6 86 Neighbor Solicitation
  0.307460 SuperMic_74:0e:90 -> Spanning-tree-(for-bridges)_00 STP 60 Conf. Root =
32768/0/00:25:90:74:0e:90  Cost = 0  Port = 0x8001
                              ...          ...          ...
```

Для эффективного использования анализаторов трафика непривилегированными пользователями достаточно делегировать их процессам захвата пакетов привилегию

CAP_NET_RAW при помощи «файловых» привилегий CAP_NET_RAW для программы захвата **/usr/bin/dumpcap**, что и проиллюстрировано[1] в листинге 4.25.

4.5.3. Другие атрибуты

Переменные окружения (листинг 4.26) и текущий рабочий каталог (листинг 4.27) на поверку тоже оказываются атрибутами процесса, которые можно получить при помощи команд **ps(1)** и **pwdx(1)** соответственно.

Листинг 4.26. Переменные окружения процесса

```
fitz@ubuntu:~$ ps fe
  PID TTY      STAT   TIME COMMAND
21872 pts/2    S      0:00 -bash USER=fitz LOGNAME=fitz HOME=/home/fitz PATH=/usr/...
22904 pts/2    R+     0:00  \_ ps fe LANGUAGE=ru:ko:en LC_ADDRESS=ru_RU.UTF-8 ...
```

Листинг 4.27. Текущий рабочий каталог процесса

```
fitz@ubuntu:~$ ps fx
  PID TTY      STAT   TIME COMMAND
22984 pts/0    S      0:00 -bash
23086 pts/0    S+     0:00  \_ man ps
23097 pts/0    S+     0:00      \_ pager
21872 pts/2    S      0:00 -bash
23103 pts/2    R+     0:00  \_ ps fx

fitz@ubuntu:~$ pwdx 23097 22984
23097: /home/fitz
22984: /home/fitz
```

4.6. Классы и приоритеты процессов

4.6.1. Распределение процессора между процессами

Переключение центрального процессора между задачами (процессами и нитями) выполняет специальная компонента подсистемы управления процессами, называемая *планировщиком* (scheduler). Именно планировщик определенным образом

[1] Необходимо заметить, что все это уже достаточно давно умеет проделывать инсталлятор при установке пакета **wireshark-common** (от которого зависят пакеты **tshark** и **wireshark**), если утвердительно ответить на вопрос инсталлятора *'Should non-superusers be able to capture packets?'*. Однако для более простого **tcpdump(8)** такой услуги не предоставлено ☺.

выбирает из множества неспящих, готовых к выполнению (runable) задач одну, которую переводит в состояние выполнения (running). Процедуры, определяющие способ выбора и моменты выполнения выбора, называются *алгоритмами планирования*. Выбор задачи, подлежащей выполнению, естественным образом происходит в моменты времени, когда текущая выполнявшаяся задача переходит в состояние сна (sleep) в результате выполнения операции ввода-вывода. *Вытесняющие* алгоритмы планирования, кроме всего прочего, ограничивают непрерывное время выполнения задачи, принудительно прерывая ее выполнение по исчерпанию выданного ей кванта времени (timeslice) и *вытесняя* ее во множество готовых, после чего производят выбор новой задачи, подлежащей выполнению.

По умолчанию для пользовательских задач используется вытесняющий алгоритм CFS (completely fair scheduler), согласно которому процессорное время распределяется между неспящими задачами справедливым (fair) образом. Для каждой задачи определяется выделяемая справедливая (в соответствии с ее относительным «приоритетом») доля процессорного времени, которую она должна получить при конкуренции за процессор. Для двух задач с любыми *одинаковыми* приоритетами должны быть выделены *равные* доли (в 50% процессорного времени), а при *различии* в приоритетах на *одну* ступень разница между выделяемыми долями должна составить ≈10% процессорного времени (т. е. 55 и 45% соответственно). Для удовлетворения этого требования алгоритм планирования CFS назначает каждой ступени приоритета соответствующий[1] вес задачи, а процессорное время делит между всеми неспящими задачами пропорционально их весам. Таким образом, две задачи с любыми одинаковыми приоритетами будут иметь равные веса $w_i = w_j = w$, а доли процессорного времени составят $\mu_i = w_i/(w_i + w_j) = 1/2$ и $\mu_j = w_j/(w_i + w_j) = 1/2$. Для двух задач с приоритетами, отличающимися на одну ступень, $w_i \neq w_j$, а $\mu_i - \mu_j = 1/10$, откуда несложно получить, что $w_i/w_j = 11/9$ — правило построения шкалы[2] весов, а $\mu_1 = 11/20 = 0{,}55$ и $\mu_2 = 9/20 = 0{,}45$, что и требовалось получить.

Для дифференциации задач используют 40 относительных POSIX-приоритетов на шкале от −20 до +19, называемых «любезностью» задачи NICE. Относительный приоритет буквально определяет, насколько «любезна» будет задача по отношению к остальным готовым к выполнению задачам при конкуренции за процессорное время освободившегося процессора. Наименее «любезным», с относительным приоритетом −20 (наивысшим) планировщик выделит большую долю процессорного времени, а наиболее «любезным», с приоритетом +19 (наинизшим) — меньшую.

---

[1] Шкала весов учитывает только требование 10% разницы в выделении времени CPU для задач с различием в их относительных приоритетах на одну ступень.

[2] Только $w_i/w_j = 11/9 = 1{,}2(2)$, тогда как в ядре Linux взято $w_i/w_j = 1{,}25$.

При отсутствии конкуренции, когда количество готовых к выполнению задач равно количеству свободных процессоров, приоритет не будет играть никакой роли.

В примере из листинга 4.28 при помощи команды **bzip(2)** запущены два процесса сжатия ISO-образа с наилучшим качеством одновременно друг другу на «заднем фоне». В выводе свойств **pcpu** (**p**ercent **cpu**, процент потребляемого процессорного времени), **pri** (**pri**ority), **ni** (**ni**ce) и **psr** (**p**rocessor number) их процессов при помощи **ps(1)** оказывается, что они потребляют практически одинаковые ❶ доли (проценты) процессорного времени, и это для одинаковых программ вполне соответствует интуитивным ожиданиям. После повышения любезности (понижения относительного приоритета) одного из них ❷ при помощи команды **renice(1)** до значения +10 отношение потребляемых долей процессорного времени не изменилось ❸, что означает отсутствие конкуренции за процессор, подтверждаемое как столбцом PSR, показывающим номер процессора, выполняющего программу, так и командами **nproc(1)** и **lscpu(1)**.

Листинг 4.28. Относительный приоритет NICE

```
   fitz@ubuntu:~$ bzip2 --best -kf plan9.iso &
   [1] 12944
   fitz@ubuntu:~$ bzip2 --best -kf plan9.iso &
   [2] 12945
   fitz@ubuntu:~$ ps fo pid,pcpu,pri,ni,psr,cmd
     PID %CPU PRI  NI PSR CMD
   12808  0.1  19   0   0 -bash
❶ 12944 94.5  19   0   2  \_ bzip2 --best -kf plan9.iso
   12945 96.0  19   0   1  \_ bzip2 --best -kf plan9.iso
   12946  0.0  19   0   3  \_ ps fo pid,pcpu,pri,ni,psr,cmd

❷ fitz@ubuntu:~$ renice +10 12945
   12945 (process ID) old priority 0, new priority 10
   fitz@ubuntu:~$ ps fo pid,pcpu,pri,ni,psr,cmd
     PID %CPU PRI  NI PSR CMD
   12808  0.1  19   0   0 -bash
❸ 12944 94.8  19   0 ☛2  \_ bzip2 --best -kf plan9.iso
❓ 12945 97.0   9  10 ☛0  \_ bzip2 --best -kf plan9.iso
   12948  0.0  19   0   1  \_ ps fo pid,pcpu,pri,ni,psr,cmd
   fitz@ubuntu:~$ nproc
   4
   fitz@ubuntu:~$ lscpu
   Архитектура:                    x86_64
   CPU op-mode(s):                 32-bit, 64-bit
```

```
Порядок байт:                    Little Endian
Address sizes:                   36 bits physical, 48 bits virtual
CPU(s):                          4
On-line CPU(s) list:             0-3
                                 ...          ...          ...

❹ fitz@ubuntu:~$ taskset -p -c 3 12808
pid 12808's current affinity list: 0-3
pid 12808's new affinity list: 3

❺ fitz@ubuntu:~$ nice -n 5 time bzip2 --best -kf  plan9.iso &
[1] 29331
❺ fitz@ubuntu:~$ nice -n 15 time bzip2 --best -kf  plan9.iso &
[2] 29333
fitz@ubuntu:~$ ps fo pid,pcpu,pri,ni,psr,cmd
  PID %CPU PRI  NI PSR CMD
28573  0.0  19   0   3 -bash
29331  0.0   9   5   3 \_ time bzip2 --best -kf plan9.iso
❻ 29332 91.4  9   5   3 |   \_ bzip2 --best -kf plan9.iso
29333  0.0   4  15   3 \_ time bzip2 --best -kf plan9.iso
! 29334  9.7  4  15   3 |   \_ bzip2 --best -kf plan9.iso
29336  0.0  19   0   3 \_ ps fo pid,pcpu,pri,ni,psr,cmd
fitz@ubuntu:~$ wait
51.10user 0.22system 0:56.86elapsed 90%CPU (0avgtext+0avgdata 31248maxresident)k
0inputs+180560outputs (0major+1004minor)pagefaults 0swaps
[1]-  Завершён              nice -n 5 time bzip2 --best -kf plan9.iso
53.79user 0.20system 1:43.08elapsed 52%CPU (0avgtext+0avgdata 31520maxresident)k
0inputs+180560outputs (0major+1515minor)pagefaults 0swaps
[2]+  Завершён              nice -n 15 time bzip2 --best -kf plan9.iso
```

Проиллюстрировать действие относительного приоритета **NICE** на многопроцессорной системе можно, создав искусственную конкуренцию двух процессов за один процессор. Для этого при помощи команды **taskset(1)** устанавливается привязка (affinity) ❹ командного интерпретатора (**PID = 12808**) к процессору **3** (привязка, как и прочие свойства и атрибуты процесса, наследуется потомками). Затем при помощи команды **nice(1)** запускаются ❺ две программы упаковки с относительными приоритетами **5** и **15** в режиме измерения потребления времени при помощи команды **time(1)**. В результате доли процессорного времени распределяются неравномерно, причем их разница зависит от *разницы* в относительных приоритетах (и от свойств конкурирующих процессов, но в примере они одинаковые). Дождавшись завершения процессов заднего фона, при помощи встроенной команды **wait**

можно оценить разницу в реальном времени выполнения упаковки, вызванную неравным распределением процессора между процессами упаковщиков.

Кроме приоритетной очереди, планировщик Linux позволяет использовать еще три алгоритма планирования — FIFO, RR и EDF, предназначенные для задач реального времени. Вытесняющий алгоритм RR (round robin) организует простейшее *циклическое* обслуживание с фиксированными квантами времени, тогда как FIFO (first in first out) является его невытесняющей модификацией, позволяя задаче выполняться непрерывно долго, до момента ее засыпания. По сути, оба алгоритма организуют задачи в одну приоритетную очередь (PQ, priority queue) со статическими приоритетами на шкале от 1 до 99, выбирая для выполнения всегда самую высокоприоритетную из множества готовых. Алгоритм EDF (Earliest Deadline First) предназначен для обеспечения гарантий периодическим задачам реального времени, которым важно получать периодическое обслуживание так, чтобы задача не была вытеснена в течение определенного времени.

Перевод задачи под управление тем или алгоритмом планирования производится при помощи назначения ей политики планирования (scheduling policy) посредством команды chrt(1). Различают шесть политик планирования, три из которых — SCHED_OTHER, SCHED_BATCH и SCHED_IDLE, реализуются алгоритмом CFS, политики SCHED_FIFO, SCHED_RR — одноименными алгоритмами FIFO и RR, а политика SCHED_DEADLINE — алгоритмом EDF. Политика SCHED_OTHER, она же SCHED_NORMAL, применяется по умолчанию и обслуживает класс задач TS (time sharing), требующих «интерактивности», а политика SCHED_BATCH предназначается для выполнения «вычислительных» задач класса пакетной B (batch) обработки.

Разница между политиками состоит в том, что планировщик в случае необходимости всегда прерывает задачи класса B в пользу задач класса TS, но никогда наоборот. Политика SCHED_IDLE формирует класс задач, выполняющихся только при «простое» (idle) центрального процессора за счет выделения им планировщиком CFS очень небольшой доли процессорного времени. Для этого задачам IDL-класса назначают минимально возможный вес — меньший, чем вес самых «любезных» (NICE = +19) задач TS-класса.

В примере из листинга 4.29 проиллюстрировано распределение процессорного времени алгоритмом планирования CFS между (конкурирующими за один процессор) одинаковыми процессами TS-, B- и IDL-классов, назначенных им при запуске упаковщиков bzip2(1) посредством команды chrt(1).

Листинг 4.29. Классы процессов

```
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
```

```
  12058 pts/0    Ss     0:00 bash
  12065 pts/0    R+     0:00  \_ ps f
fitz@ubuntu:~$ taskset -p -c 3 12058
pid 12058's current affinity list: 0-3
pid 12058's new affinity list: 3

fitz@ubuntu:~$ chrt -b 0 time bzip2 --best -kf plan9.iso &
[1] 12410
fitz@ubuntu:~$ chrt -o 0 time bzip2 --best -kf plan9.iso &
[2] 12412
fitz@ubuntu:~$ chrt -i 0 time bzip2 --best -kf plan9.iso &
[3] 12414
fitz@ubuntu:~$ ps fo pid,pcpu,class,pri,ni,psr,cmd
  PID %CPU CLS   PRI  NI PSR CMD
12058  0.1 TS     19   0   3 -bash
12410  0.0 B      19   -   3  \_ time bzip2 --best -kf plan9.iso
12411 50.0 B      19   -   3  |   \_ bzip2 --best -kf plan9.iso
12412  0.0 TS     19   0   3  \_ time bzip2 --best -kf plan9.iso
12413 49.7 TS     19   0   3  |   \_ bzip2 --best -kf plan9.iso
12414  0.0 IDL    19   -   3  \_ time bzip2 --best -kf plan9.iso
12415  0.1 IDL    19   -   3  |   \_ bzip2 --best -kf plan9.iso
12471  0.0 TS     19   0   3  \_ ps fo pid,pcpu,class,pri,ni,psr,cmd
fitz@ubuntu:~$ wait
53.85user 0.26system 1:45.98elapsed 51%CPU (0avgtext+0avgdata 31248maxresident)k
0inputs+180560outputs (0major+1004minor)pagefaults 0swaps
53.96user 0.22system 1:46.04elapsed 51%CPU (0avgtext+0avgdata 31248maxresident)k
0inputs+180560outputs (0major+1004minor)pagefaults 0swaps
52.74user 0.27system 2:41.54elapsed 32%CPU (0avgtext+0avgdata 31248maxresident)k
0inputs+180560outputs (0major+1004minor)pagefaults 0swaps
[1]   Завершён          chrt -b 0 time bzip2 --best -kf plan9.iso
[2]-  Завершён          chrt -o 0 time bzip2 --best -kf plan9.iso
[3]+  Завершён          chrt -i 0 time bzip2 --best -kf plan9.iso
```

В листинге 4.30 показана конкуренция процессов под управлением RR-планировщика, использующего статические приоритеты. Так как операция назначения политик планирования «реального времени» FIFO и RR является привилегированной, то сначала ❶ командный интерпретатор переводится в RR-класс (`-r`) с наивысшим статическим приоритетом **99** при помощи команды chrt(1), выполняемой от лица суперпользователя **root**. При последующих запусках упаковщиков класс будет унаследован и не потребует повышенных привилегий. Два процесса упаковщиков

запускаются ❷ командой **chrt(1)** с одинаковыми статическими приоритетами **1**, привязанные одному процессору командой **taskset(1)**, в результате чего получают равные доли процессорного времени, что вполне соответствует интуитивным ожиданиям от вытесняющего циклического планировщика **RR**. Нужно отметить, что шкала статических приоритетов **1→99** классов **RR** и **FIFO**, как и шкала «любезности» **NICE +19→-20** классов **TS** и **B**, отображаются на общую шкалу приоритетов **PRI** так, что верхняя часть шкалы **PRI: 41→139** соответствует статическим приоритетам, а нижняя часть шкалы **PRI: 0→39** соответствует «любезности».

Листинг 4.30. Классы процессов реального времени

```
   fitz@ubuntu:~$ ps f
     PID TTY      STAT   TIME COMMAND
   15313 pts/0    S      0:00 -bash
   15520 pts/0    R+     0:00  \_ ps f
❶  fitz@ubuntu:~$ sudo chrt -pr 99 15313
   fitz@ubuntu:~$ ps fo pid,psr,cls,ni,pri,pcpu,comm
     PID PSR CLS  NI PRI   %CPU COMMAND
   15313   0 RR   - 139   0.1 bash
   15550   1 RR   - 139   0.0  \_ ps

❷  fitz@ubuntu:~$ chrt -r 1 taskset -c 2 bzip2 --best -kf plan9.iso &
   [1] 15572
❸  fitz@ubuntu:~$ chrt -r 1 taskset -c 2 bzip2 --best -kf plan9.iso &
   [2] 15573
   fitz@ubuntu:~$ ps fo pid,psr,cls,ni,pri,pcpu,comm
     PID PSR  CLS  NI   PRI %CPU   COMMAND
   15313   0   RR   -   139  0.1   bash
   15572   2   RR   -    41 51.8    \_ bzip2
   15573   2   RR   -    41 48.8    \_ bzip2
   15597   1   RR   -   139  0.0    \_ ps

❹  fitz@ubuntu:~$ chrt -r 2 taskset -c 2 bzip2 --best -kf plan9.iso &
   [3] 15628
   fitz@ubuntu:~$ ps fo pid,psr,cls,ni,pri,pcpu,comm
     PID PSR CLS  NI   PRI %CPU    COMMAND
   15313   0 RR   -   139  0.1    bash
   15572   2 RR   -    41 48.3    \_ bzip2
★  15573   2 RR   -    41 47.5    \_ bzip2
   15628   2 RR   -    42 93.3    \_ bzip2
   15630   1 RR   -   139  0.0    \_ ps
```

```
fitz@ubuntu:~$ top -b -n1 -p 15572,15573,15628
top - 14:44:01 up  4:27,  2 users,  load average: 5.07, 3.42, 2.50
Tasks:   3 total,   3 running,   0 sleeping,   0 stopped,   0 zombie
Cpu(s): 18.5%us,  3.4%sy,  0.6%ni, 76.0%id,  1.4%wa,  0.0%hi,  0.1%si,  0.0%st
МиБ Mem :   3935,6 total,   2629,5 free,    425,1 used,    880,9 buff/cache
МиБ Swap:    448,5 total,    448,5 free,      0,0 used.   3265,3 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
15628 fitz      -3   0    9536   7880    468 R   100  0.1   1:14.96 bzip2
15572 fitz      -2   0    9536   7880    460 R     0  0.1   1:30.95 bzip2
15573 fitz      -2   0    9536   7884    464 R     0  0.1   1:30.01 bzip2
```

Добавление ❸ третьего процесса упаковщика со статическим приоритетом **2** приводит к резкому перекосу выделяемой доли процессорного времени в его пользу. Это объясняется тем, что алгоритм планирования RR (равно как и FIFO) всегда выбирает процесс с самым высоким статическим приоритетом из множества готовых, поэтому процессам с более низкими приоритетами процессорное время будет выделено только при засыпании[1] всех процессов с большими приоритетами.

Странный результат ★, изображаемый командой **ps(1)**, объясняется «несовершенством» ее способа расчета доли процессорного времени **%CPU**, выделяемой процессу. Расчет производится как отношение чистого потребленного процессорного времени (за все время существования процесса) к промежутку реального времени, прошедшему с момента порождения процесса, что соответствует *среднему*, но не *мгновенному* значению потребляемой доли. Гораздо более ожидаемый результат получает команда **top(1)**, выполняющая расчет *мгновенной* доли процессорного времени как отношение чистого потребленного процессорного времени (за небольшой промежуток наблюдения) к реальному времени наблюдения.

4.6.2. Распределение устройств ввода-вывода между процессами

В любой *многозадачной* операционной системе кроме вопроса распределения (между ее задачами) центрального процессора, рассмотренного выше, неизбежно возникают вопросы распределения и других устройств, например памяти *(см. разд. 4.7)* и устройств ввода-вывода. В Linux все устройства ввода-вывода принято подразделять на «поблочные» устройства (block devices), которые зачастую являются накопителями, «посимвольные» устройства (character devices), — как правило, устрой-

[1] А упаковщик, хоть и выполняет операции ввода-вывода, в реальности практически не засыпает в связи с использованием дискового кэша.

ства взаимодействия с пользователями (мыши, клавиатуры, терминалы и пр.) и «попакетные» устройства, в большинстве своем — сетевые интерфейсы.

В каждом классе устройств способы их распределения часто связаны с эффективностью их работы, т. к. решение задачи «в лоб» приводит к неприемлемым результатам. Например, при распределении доступа к дисковым накопителям оказывается, что количество считываемых данных в единицу времени (пропускная способность, *throughput*) существенно зависит от того, в каком порядке обслуживать запросы отдельных процессов на чтение/запись накопителя. Такой результат в основном обусловлен свойствами самих накопителей, все еще являющихся механическими дисками. Для проведения операции чтения или записи дискового блока контроллер накопителя должен потратить время на перемещение головки над нужной дорожкой, а затем дождаться «приезда» нужного сектора этой дорожки, который содержит данные искомого блока. На все это нужно колоссальное (с точки зрения центрального процессора и процессов) время[1], что и ограничивает количество данных, обрабатываемых в единицу времени.

Если представить себе операции линейного считывания или записи, когда обрабатываются дисковые блоки, находящиеся в последовательных секторах одной дорожки, затем последовательно в секторах соседней дорожки, то можно добиться максимальной производительности, но это невозможно по двум причинам. Во-первых, процессы работают с абстракциями более высокого уровня — с файлами, данные которых могут размещаться файловыми системами в произвольных дисковых блоках. Даже если последовательные блоки файла размещены в последовательных дисковых блоках, программы в принципе могут читать произвольные файлы в каком угодно порядке. Во-вторых, сама мультипрограммная смесь в принципе генерирует общий поток запросов к произвольным местам диска. При попытке обслуживать этот поток «как есть» количество накладных расходов на хаотичный поиск (seek time) нужных дисковых блоков будет достаточно велико, а результирующая пропускная способность диска — мала. Именно эта задача приводит к появлению в подсистеме блочных устройств ядра *планировщика ввода-вывода (I/O scheduler)*, т. е. специальной компоненты, обслуживающей очередь запросов к накопителю особенным образом, изначально направленным на оптимизацию пропускной способности накопителя.

Подобные планировщики часто используют алгоритмы, подобные тем, что задействованы в лифтовых системах зданий для оптимизации перемещения лифтов *(elevator)*, и носят название лифтовых алгоритмов. Наиболее известными алгоритмами являются так называемые SCAN, C-SCAN, LOOK и C-LOOK, см. **W:[Elevator algorithm]**, **W:[LOOK algotithm]**. Принцип их действия (упрощенно) состоит в том, что

[1] Что, однако, и привело в свое время к многозадачным ОС.

начавший движение лифт всегда едет в одном направлении, подбирая новых пассажиров, только направляющихся в сторону его движения, после чего, достигнув точки назначения, ждет нового вызова.

Планировщики ввода-вывода аналогично пытаются обслуживать поступающие запросы к накопителю не в порядке их поступления, а в порядке их номеров блоков (в порядке номеров этажей, лифт всегда едет снизу вверх), что естественно уменьшает суммарное время поиска и, как следствие, увеличивает общую пропускную способность. Кроме этого, планировщики пытаются укрупнять мелкие запросы на доступ к единичным, расположенным последовательно дисковым блокам, объединяя их в меньшее количество крупных запросов на доступ к «несколькоблочным» дисковым областям. Это опять позволяет экономить время при обращении к диску, что тоже увеличивает количество данных, обрабатываемых в единицу времени. Две такие применяемые планировщиками операции принято называть сортировкой (sorting) и слиянием (merging) соответственно.

Простейший планировщик в ядрах версии 2.4 организовывал общую очередь запросов к накопителю, отсортированную в порядке их номеров блоков, помещая новые запросы в нужное место в середине очереди, а обслуживал запросы всегда из головы очереди, отправляя их на обработку драйверу контроллера накопителя. Организованная таким образом обработка, однако, должна страдать от эффекта голодания *(starvation)* запросов со старшими номерами, т. к. если представить, что новые запросы будут непрерывно и достаточно интенсивно поступать с младшими номерами блоков, то запросы со старшими номерами вообще никогда не будут обработаны. Для решения проблемы голодания этот классический планировщик использовал возрастные отметки запросов, заставляя новые запросы размещаться всегда в конце очереди, если в ней был найден хотя бы один запрос старше определенного возраста. Такая эвристика уменьшала проблему, но в принципе не избавляла от нее, т. к. запросы, конечно, не застревали в очереди «навсегда», но задержка их обработки была непредсказуемой и ожидала желать лучшего.

Более того, она никак не касалась особенного случая эффекта голодания — так называемого голодания запросов на чтение, вызываемого запросами на запись *(writes starving reads)*. Этот эффект основывается на том, что при чтении в подавляющем большинстве случаев по тем или иным причинам программы ждут реального завершения одного запроса, прежде чем отправят другой. При записи, наоборот, они практически всегда генерируют кучу запросов пачкой, отчасти и потому, что ядро воображаемо «завершает» для процесса операцию записи немедленно после копирования данных запроса в свои внутренние структуры (например, в страничный кэш). Таким образом, в очереди запросов оказывается куча запросов на запись, обработка которых неминуемо ведет к задержкам обработки немногих запросов на чтение. Ситуация еще больше усугубляется тем, что подобные пачки запросов на запись обычно адресуют последовательные блоки, что, к сожалению

(для эффекта голодания), согласуется с лифтовыми алгоритмами классического планировщика.

Одним из первых планировщиков ввода-вывода, который был направлен на решение вышеобозначенных проблем, стал появившийся в ядрах версии 2.6 так называемый **deadline**, доступный и по сей день[1]. Принцип использования «лифтовых» алгоритмов остался без изменения, и в планировщике также организуется общая очередь запросов, отсортированная в порядке их *номеров* дисковых блоков. Однако для решения проблем голодания планировщик организует еще две отдельные очереди — отдельную для запросов на чтение и отдельную для запросов на запись, в которых запросы размещены в порядке *поступления*. Более того, у каждой из дополнительных очередей есть определенные преследуемые «сроки» обработки, определенные в пользу запросов на чтение. По умолчанию они составляют 500 мс для чтения и 5000 мс (5 с) для запросов на запись. В основном режиме планировщик обслуживает запросы в порядке общей очереди, что направлено на повышение пропускной способности накопителя, но по истечении определенного «срока» обработки запросов в одной из дополнительных очередей планировщик переходит в режим обработки этой дополнительной очереди, тем самым пытаясь обеспечить обозначенные «сроки». Несмотря на свою простоту, **deadline**-планировщик справился с эффектами голодания, обеспечив вполне предсказуемые значения задержек обработки запросов.

Однако надо заметить, что **deadline**-планировщик делает свою работу за счет уменьшения общей пропускной способности, т. к. каждый раз при истечении срока обработки запроса на чтение он прерывает обслуживание в «лифтовом» порядке и тратит дополнительное время на поиск блока этого истекшего запроса, после чего возвращается к обычной работе и опять тратит дополнительное время на поиск того блока, на котором он прервался. Такие наблюдения привели к эвристике «предвосхищающего» **anticipatory**-планировщика, который основывается на поведении **deadline**-планировщика и предвидении запросов на чтение, последующих за обработанными запросами с истекшим «сроком». Другими словами, каждый раз, когда **anticipatory**-планировщик отвлекается на запрос чтения с истекшим «сроком», по окончании его обслуживания он не спешит возвращаться к основной очереди, а выжидает некоторое непродолжительное время (6 мс по умолчанию) в надежде на получение еще одного запроса на чтение с номером блока, следующим за только что обработанным. В силу широко распространенного последовательного чтения из файлов такое предвидение оправдывается с высокой вероятностью, что приводит к повышению пропускной способности при сохранении предсказуемых задержек операций чтения.

---

[1] Претерпевший некоторые трансформации, но все же.

Впоследствии **anticipatory**-планировщик был удален из ядра в пользу планировщика **CFQ** (**C**ompletely **F**air **Q**ueuing, совершенно справедливая очередь), который позволяет получать результаты сравнимые (и даже лучше) с **anticipatory**-планировщиком, но обладает еще и массой других полезных свойств. В основе **CFQ**-планировщика лежит желание обеспечить справедливое распределение пропускной способности накопителя между процессами, вне зависимости от их поведения. Для этого планировщик организует для запросов на чтение по одной очереди на процесс, плюс общую очередь для запросов на запись, а сами запросы во всех очередях, как и всегда, сортируются в порядке номеров их дисковых блоков. Кроме этого, подобно **deadline**-планировщику, **CFQ** определяет максимальные «сроки» обработки запросов, а подобно **anticipatory**-планировщику, после обработки запроса на чтение выжидает некоторое время, предвосхищая появление еще одного запроса с номером блока, следующего за только что обработанным.

Кроме этого, планировщик предусматривает возможность дифференциации процессов (и, как следствие, их очередей чтения) по классам и приоритетам, позволяя выделить долю пропускной способности для определенных процессов чуть «справедливее», чем для других. Очереди записи для всех процессов общие, но тоже подразделяются по классам и приоритетам.

Порядок выбора очереди для обработки запросов определяется сначала ее классом, а внутри класса — при помощи специальных отметок «времени начала» обработки, назначаемых в зависимости от приоритета. При этом запросы в каждой из них обрабатываются в течение времени, ограниченного сверху некоторым интервалом *(slice)* времени, так же определяемым приоритетом очереди.

Различают три класса обслуживания: **realtime**, **best-effort**, **idle**, которые задаются процессам явно при помощи системного вызова **ioprio_set(2)** и утилиты **ionice(1)**. В случае, если для процесса явно не определен класс обслуживания (по умолчанию все процессы отнесены к «классу» **none**), он неявно «зеркалируется» на приоритет планировщика центрального процессора **CFS** *(см. разд. 4.6.1)*. Так, например, процессы из **CFS** класса **SCHED_IDLE** неявно расцениваются как находящиеся в **CFQ**-классе **idle**, а процессы из классов **SCHED_FIFO**, **SCHED_RR** и **SCHED_DEADLINE** — как в классе **realtime**. Для остальных процессов используется класс **best-effort**, при этом не заданные явно приоритеты так же неявно высчитываются пропорционально «любезности» **NICE**.

В классах **realtime** и **best-effort** различают по 8 приоритетов (min 7→0 max), от которых зависит длительность интервала времени, выделяемого очередям на обработку. Базовый интервал времени S обычно равняется 100 мс (см. ❶ в листинге 4.32), который назначается «среднему» приоритету $p = 4$, а для остальных они масштабируются от 40 мс (для приоритета 7) до 180 мс (для приоритета 0), как $S_p = S + S \cdot k \cdot (4 - p)$, где $k = 1/5$ — масштабный множитель шкалы приорите-

тов. Для обработки запросов на запись базовый интервал составляет всего лишь 40 мс (см. ❷ в листинге 4.32).

Кроме того, приоритеты так же используются для определения «времени начала» обработки запросов каждой очереди *(slice offset)*, как $SO_p = (n - 1) \cdot (S_0 - S_p)$, где n — количество непустых очередей планировщика. Например, если есть три активно читающих процесса $n = 3$, то для процессов с приоритетом 0 время начала обработки определяется как $SO_0 = 0$ мс, а для процессов с приоритетом 7 — как $SO_7 = 2 \cdot (180 - 40) = 360$ мс соответственно.

Запросы в очередях класса **realtime** обрабатываются в первую очередь, и если все запросы исчерпаны, то планировщик переходит к обработке **best-effort**-очередей, а при их исчерпании — к очередям класса **idle**. Внутри каждого класса очереди обрабатываются в порядке «времени начала» обработки и в течение интервала обработки, а сами запросы — в порядке номеров их дисковых блоков. Вместе с тем, как было сказано раньше, **CFQ**-планировщик попроцессно отслеживает максимально установленные «сроки» обработки запросов, что составляет 250 мс для запросов записи и 125 мс для запросов чтения (см. ❸ в листинге 4.32). При исчерпании запросов в обрабатываемой очереди (если еще не истек ее интервал) **CFQ**-планировщик выжидает 8 мс (см. ❹ в листинге 4.32), предвидя появление нового запроса на чтение, «продолжающего» только что обработанный.

Внутреннее устройство **CFQ**-планировщика, надо заметить, изначально не было направлено (в отличие от «классического», **dealine** и **anticipatory**) на увеличение пропускной способности накопителя, а преследовало несколько иные цели. Однако его современная реализация оказалась весьма эффективной для самых разнообразных применений, что и сделало его планировщиком по умолчанию вплоть до ядер версии 5.*x* (см. листинги 4.31 и 4.32), где он был заменен планировщиком **BFQ** (**B**udget **F**air **Q**ueue).

Листинг 4.31. I/O планировщики устройств (single-queue)

```
❶ fitz@ubuntu:~$ lsblk -S
  NAME HCTL       TYPE VENDOR   MODEL             REV TRAN
  sda  0:0:0:0    disk ATA      WDC WD2500BEKT-7 1A01 sata
  sr0  1:0:0:0    rom  TSSTcorp DVD+-RW TS-U633J D600 sata
❶ fitz@ubuntu:~$ cat /sys/block/{sda,sr0}/queue/scheduler
  noop deadline [cfq]
  noop deadline [cfq]

  fitz@ubuntu:~$ uname -r
  4.18.0-25-lowlatency
```

```
❷ fitz@ubuntu:~$ echo deadline | sudo tee /sys/block/sr0/queue/scheduler
  deadline

  fitz@ubuntu:~$ cat /sys/block/sr0/queue/scheduler
  noop [deadline] ☚ cfq
```

Листинг 4.31 показывает, что назначенный блочному устройству ❶ планировщик можно узнать только при помощи «прямого» чтения файловой системы **sysfs** ❶, а выбрать иной планировщик при помощи «прямой» записи в ее файлы. Нужно отметить, что планировщик noop (**no ope**ar**t**ion), как можно догадаться из названия, (почти) ничего с поступающими запросами не делает, но именно поэтому идеально подходит для недисковых устройств, например SSD-накопителей, задержки доступа к данным которых никак не определяются механической составляющей, присущей «обычным», дисковым накопителям.

Листинг 4.32. Параметры I/O планировщика CFQ

```
  fitz@ubuntu:~$ ls /sys/block/sda/queue/iosched/
  back_seek_max      group_idle_us   slice_async_us  target_latency
  back_seek_penalty  low_latency     slice_idle      target_latency_us
  fifo_expire_async  quantum         slice_idle_us
  fifo_expire_sync   slice_async     slice_sync
  group_idle         slice_async_rq  slice_sync_us
❶ fitz@ubuntu:~$ cat /sys/block/sda/queue/iosched/slice_sync
  100
❷ fitz@ubuntu:~$ cat /sys/block/sda/queue/iosched/slice_async
  40
❸ fitz@ubuntu:~$ cat /sys/block/sda/queue/iosched/fifo_expire_*
  250
  125
❹ fitz@ubuntu:~$ cat /sys/block/sda/queue/iosched/slice_idle
  8
```

Широкое распространение сверхбыстрых твердотельных накопителей, например W:[NVMe], поставило перед разработчиками ядра Linux новые задачи, т. к. оказалось, что подсистема блочных устройств в силу внутреннего строения в принципе не способна обрабатывать большое количество запросов на чтение/запись в единицу времени W:[IOPS], что не позволяет использовать такие накопители на «полную катушку». При разработке ядер серии 4.*x* блочную систему перепроектировали и изменили принцип организации запросов, поступающих к блочному устройству на обработку. Вместо одной *общей* входящей очереди (**single queue**) к каждому уст-

ройству, за которую ранее *состязались* процессы, выполняющиеся на разных процессорах, организовали *индивидуальные* очереди (**multi-queue**) по числу процессоров, что позволило существенно повысить эффективность работы подсистемы. Это привело к переделке драйверов устройств и планировщиков ввода-вывода к новому виду, которая закончилась к началу ядер серии 5.*x*. Вместе с тем планировщик **deadline** превратился в **mq-deadline**, место **CFQ** занял **BFQ**, а позднее был добавлен планировщик **kyber** (листинг 4.33) для работы с высокоскоростными накопителями.

Листинг 4.33. I/O планировщики устройств (multi-queue)

```
fitz@ubuntu:~$ cat /sys/block/{sda,sr0}/queue/scheduler
mq-deadline [bfq] none
[mq-deadline] bfq none
fitz@ubuntu:~$ uname -r
5.3.0-24-lowlatency
fitz@ubuntu:~$ modinfo bfq
filename:       /lib/modules/5.3.0-26-generic/kernel/block/bfq.ko
description:    MQ Budget Fair Queueing I/O Scheduler
                ...          ...          ...
fitz@ubuntu:~$ modinfo kyber-iosched
filename:       /lib/modules/5.3.0-26-generic/kernel/block/kyber-iosched.ko
description:    Kyber I/O scheduler
                ...          ...          ...
```

Планировщик **BFQ** изначально базировался на коде **CFQ** и до сих пор сохраняет практически все его свойства, но вместо интервалов времени, отводимых на обработку запросов, использует понятие «бюджета» обработки, исчисляемого в количестве (дисковых) секторов, которые будут обслужены из той или иной очереди. Бюджет **BFQ** (как и интервалы обработки **CFQ**) пропорционален приоритетам процессов, что делает планировщик «справедливым»[1] в отношении доли пропускной способности устройства, выделяемой отдельным процессам. Кроме этого, планировщик следит, чтобы потребление выделенного «бюджета» не занимало много времени (например, процесс может читать секторы, далеко отстоящие друг от друга), в противном случае очередь «наказывается» путем снятия с обработки до исчерпания своего бюджета.

В листинге 4.34 показано управление классами и приоритетами планировщиков **CFQ** и **BFQ**, а в листинге 4.35 — пропорциональное распределение пропускной способности между конкурирующими процессами.

---

[1] Процессам в одном классе с одинаковыми приоритетами будет выделена одинаковая доля пропускной способности, а процессам с разными приоритетами — разная, но пропорциональная их приоритетам. И пусть никто не уйдет обиженным.

Листинг 4.34. I/O классы процессов планировщиков CFQ и BFQ

```
fitz@ubuntu:~$ ionice -p $$
none: prio 0
fitz@ubuntu:~$ ionice -c best-effort -n 5 -p $$
fitz@ubuntu:~$ ionice -p $$
best-effort: prio 5
fitz@ubuntu:~$ sudo ionice -c realtime -n 3 -p $$
fitz@ubuntu:~$ ionice -p $$
realtime: prio 3
```

В листинге 4.35 проведены два опыта, в которых сравнивается распределение пропускной способности при чтении с дискового накопителя **/dev/sdc** между двумя одинаковыми процессами. Если процессы имеют одинаковый класс и приоритет ❶, то в результате и получают одинаковую долю пропускной способности диска ⓪, а если разные ❶ приоритеты, то пропорциональную ①. В принципе доля пропускной способности должна была распределиться как 40 мс/(40 мс + 180 мс) ≈ 0,18 и 180 мс/(40 мс + 180 мс) ≈ 0,82 согласно длительностям интервалов обработки, выделяемой очередям планировщиком (см. выше). Полученный результат отличается от прогнозируемого потому, что один процесс заканчивает копирование раньше другого на целых 4 с, что составляет примерно 30% общего времени копирования, поэтому второй заканчивает свое чтение, уже используя полную пропускную способность диска. Поэтому *средние* (!) скорости копирования в 1,5 и 2,2 Мбит/с и не соотносятся как 0,18 к 0,82.

Листинг 4.35. Распределение пропускной способности планировщиками CFQ и BFQ

```
fitz@ubuntu:~$ findmnt -T .
TARGET SOURCE      FSTYPE OPTIONS
/      /dev/sda4 ➥ ext4   rw,relatime,errors=remount-ro
fitz@ubuntu:~$ cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
fitz@ubuntu:~$ dd if=/dev/urandom of=big1 ➥ bs=16384 count=1024
1024+0 записей получено
1024+0 записей отправлено
16777216 bytes (17 MB, 16 MiB ➥) copied, 0,089367 s, 188 MB/s
fitz@ubuntu:~$ dd if=/dev/urandom of=big2 ➥ bs=16384 count=1024
                              ...          ...          ...
fitz@ubuntu:~$ sync
⓪ fitz@ubuntu:~$ dd if=big1 of=/dev/null iflag=direct &
fitz@ubuntu:~$ dd if=big2 of=/dev/null iflag=direct &
```

```
fitz@ubuntu:~$ wait
[1] 6452
[2] 6453
32768+0 записей получено
32768+0 записей отправлено
⓪ 16777216 bytes (17 MB, 16 MiB) copied, ☛ 12,6594 s, 1,3 MB/s
32768+0 записей получено
32768+0 записей отправлено
⓪ 16777216 bytes (17 MB, 16 MiB) copied, ☛ 13,366 s, 1,3 MB/s
[1]- Завершён        dd if=big1 of=/dev/null iflag=direct
[2]+ Завершён        dd if=big2 of=/dev/null iflag=direct

❶ fitz@ubuntu:~$ ionice -c best-effort -n 7 dd if=big1 of=/dev/null iflag=direct &
fitz@ubuntu:~$ ionice -c best-effort -n 0 dd if=big2 of=/dev/null iflag=direct &
fitz@ubuntu:~$ wait
                                  ...          ...          ...
① 16777216 bytes (17 MB, 16 MiB) copied, ☛ 7,45967 s, 2,2 MB/s
                                  ...          ...          ...
① 16777216 bytes (17 MB, 16 MiB) copied, ☛ 11,4372 s, 1,5 MB/s
                                  ...          ...          ...
```

4.7. Память процесса

Еще одним ресурсом, подлежащим распределению между процессами, является оперативная память. В Linux, как и во многих других современных операционных системах, для управления памятью используют механизм *страничного отображения*, реализуемого ядром операционной системы при помощи устройства управления памятью — W:[MMU]. При этом процессы работают с *виртуальными* адресами (virtual address) «воображаемой» памяти, отображаемыми устройством MMU на физические адреса (physical address) настоящей оперативной памяти. Для отображения (рис. 4.3) вся оперативная память (RAM) условно разбивается на «гранулы» — *страничные кадры* размером 4 Кбайт, которые затем выделяются процессам. Таким образом, память процесса условно состоит из *страниц* (page), которым в специальных *таблицах страниц* (page table) сопоставлены выделенные страничные кадры (page frame). При выполнении процесса преобразование его виртуальных адресов в физические выполняется устройством MMU «на лету» при помощи его индивидуальной таблицы страниц.

Именно механизм страничного отображения позволяет эффективно распределять память между процессами путем размещения страниц процессов в *произвольные* свободные страничные кадры. Кроме этого, механизм страничного отображения позволяет выделять процессам память *по требованию*, добавляя дополнительные страницы и отображая их на свободные страничные кадры. Аналогично, ненужные

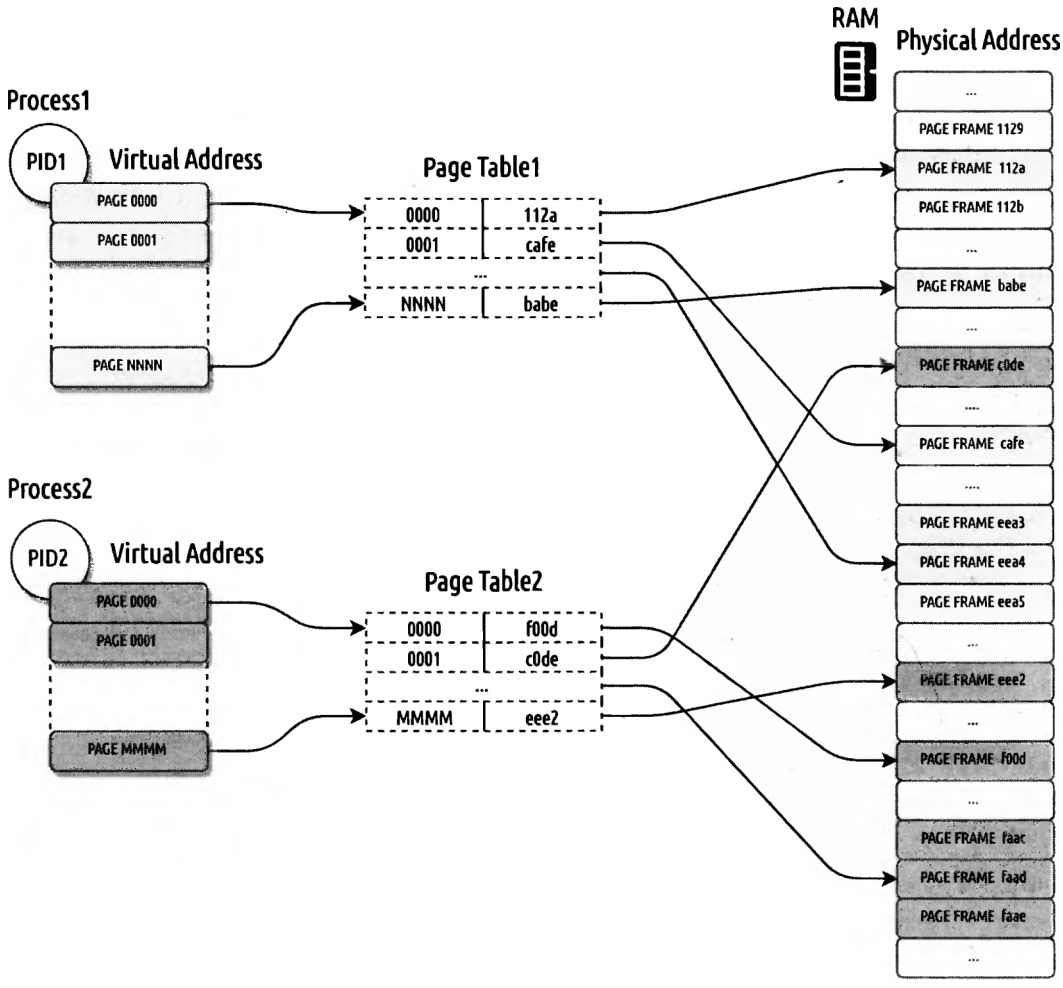


Рис. 4.3. Страничное отображение и распределение памяти

процессу страницы могут быть удалены, а соответствующие страничные кадры высвобождены для использования другими процессами.

4.7.1. Виртуальная память

Помимо задачи распределения памяти между процессами, механизм страничного отображения используется ядром операционной системы и для решения задачи нехватки оперативной памяти. При определенных обстоятельствах имеющиеся в распоряжении свободные страничные кадры оперативной памяти могут быть исчерпаны. Одновременно с этим оказывается, что бо́льшую часть времени процессы используют лишь малую часть выделенной им памяти, а находясь в состоянии сна, не используют память вовсе.

Увеличить коэффициент полезного использования памяти позволяет еще одна простая идея[1] (рис. 4.4) — высвобождать ❶ страничные кадры при помощи выгрузки ❷ (page out) неиспользуемых страниц процессов во вторичную память (в специальную область «подкачки» SWAP, например, на диске), а при обращении к выгруженной странице — загружать (page in) ее обратно перед использованием. За счет такого *страничного обмена*[2] (paging или page swapping) организуется W:[виртуальная память], т. е. видимость большего количества (оперативной) памяти для размещения процессов, чем есть на самом деле.

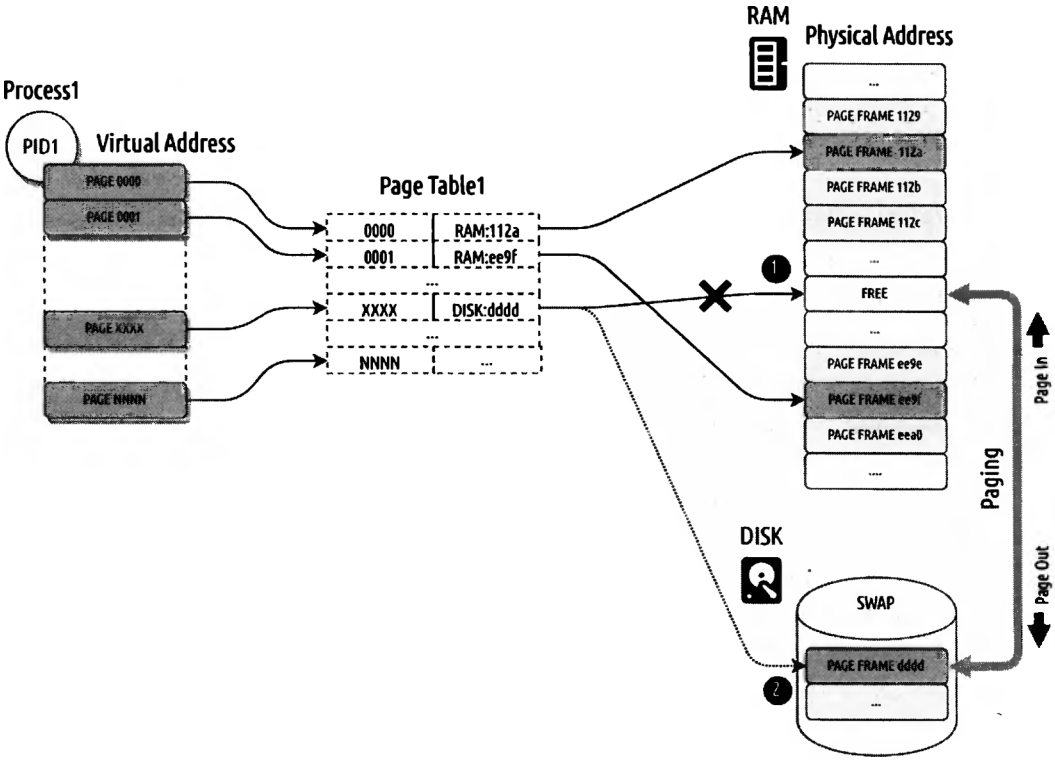


Рис. 4.4. Страничный обмен

В примере из листинга 4.36 столбцах VSZ и RSS вывода команды ps(1) показано потребление памяти процессами (в килобайтах). В столбце VSZ (virtual size) указывается суммарный объем всех страниц процесса (в том числе и выгруженных), а в столбце RSS (resident set size) — суммарный объем всех его страничных кадров в оперативной памяти, т. е. ее *реальное* потребление процессом.

---

[1] Подобная идее многозадачности *(см. разд. 4.2)*.

[2] Зачастую используют термин W:[подкачка страниц].

Листинг 4.36. Виртуальная и резидентная память процесса

```
fitz@ubuntu:~$ ps fux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
jake      4257  1.4  1.2 962016 49072 ?        Ssl  20:46   0:00 \_ .../gnome-terminal-server
jake      4267  0.0  0.1  12948  4808 pts/0    Ss+  20:46   0:00 |   \_ bash
jake      4276 20.5  5.9 2930660 240788 ?      Sl   20:46   0:05 \_ .../firefox -new-window
jake      4378  8.6  3.5 2427016 141300 ?      Sl   20:46   0:01     \_ .../firefox ...
jake      4475  6.8  2.6 2390580 107920 ?      Sl   20:46   0:01     \_ .../firefox ...
jake      4521  3.2  2.0 2374856 84480  ?      Sl   20:46   0:00     \_ .../firefox ...
```

4.7.2. Отображение файлов в память

Страничный обмен, помимо организации виртуальной памяти, имеет еще одно важнейшее применение. Именно на его основе реализуется незаменимый механизм отображения файлов в память процесса, доступный при помощи системных вызовов **mmap(2)/munmap(2)** (и дополнительных **mlock(2)**, **mprotect(2)**, **msync(2)**, **madvise(2)** и др.).

Для отображения файла (рис. 4.5) в память процесса ему выделяют страницы ❶ в необходимом количестве (**VSZ**), но не страничные кадры (**RSS**). Вместо этого в таблице страниц формируются такие записи ❶, как будто эти страницы уже были выгружены ранее (!) в отображаемый файл ❷. При последующем обращении (on demand) процесса к какой-либо странице отображенной памяти под нее выделяют ❶ страничный кадр и заполняют ❷ (read) соответствующим содержимым файла. Любые последующие изменения, сделанные процессом в отображенных страницах, сохраняются ❶ обратно (write back) в файл, если отображение выполнено «разделяемым»[1] (shared) способом. Для страниц, отображенных «частным» (private) способом, используется принцип **COW** (copy-on-write) ❶, согласно которому любые попытки их изменения (write) приводят к созданию их копий (copy), куда и попадают изменения.

Таким образом, страницы отображенного файла, которые никогда не были востребованы процессом, не будут вовсе занимать оперативной памяти (не попадут в **RSS**). Это обстоятельство широко используется для «загрузки» в процесс его программы и библиотек. В листинге 4.37 при помощи команды **pmap(1)** показана карта (отображения файлов) памяти процесса командного интерпретатора **bash(1)**.

[1] Такие понятия и поведение механизма отображения выбраны не случайно, ведь файл может быть отображен в несколько процессов одновременно.

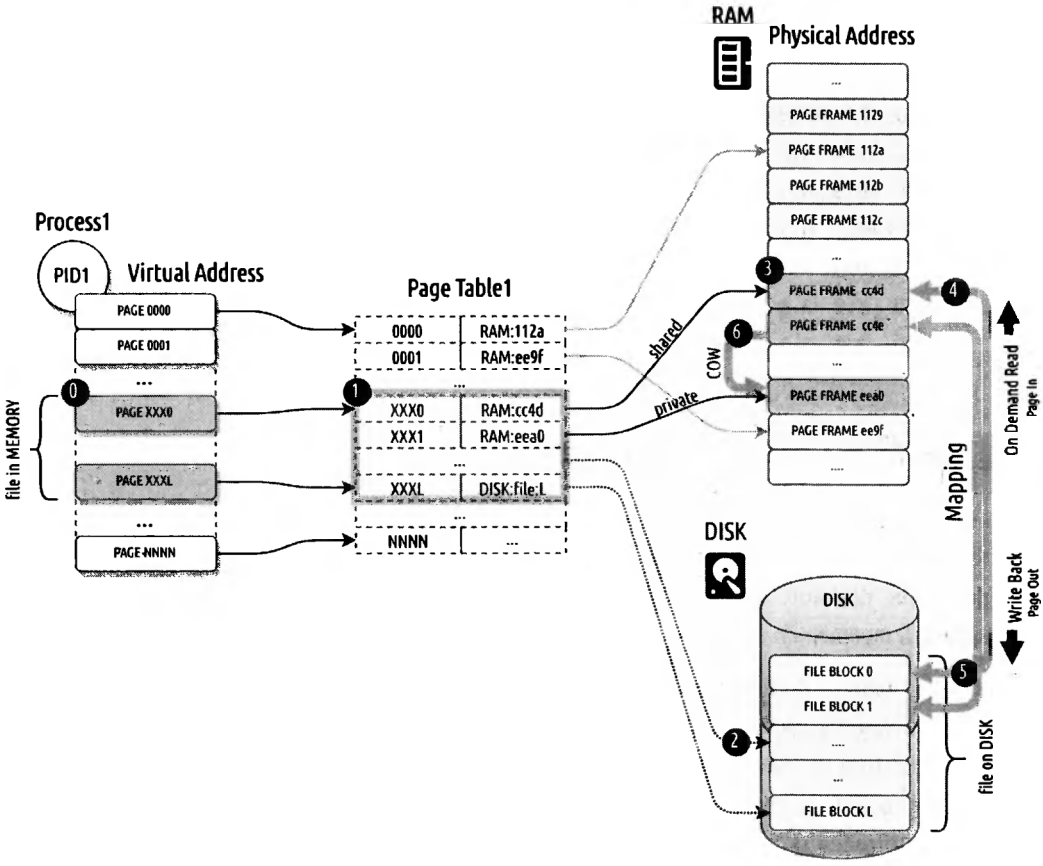


Рис. 4.5. Отображение файла в память

Листинг 4.37. Карта памяти процесса

```
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
26958 pts/0    Ss     0:00 bash
28540 pts/0    R+     0:00  \_ ps f
fitz@ubuntu:~$ which bash
/usr/bin/bash

fitz@ubuntu:~$ readelf -l /usr/bin/bash
                              ...            ...                ...
Заголовки программы:
  Тип           Смещ.             Вирт.адр           Физ.адр
                Рзм.файл          Рзм.пм             Флаги  Выравн
                              ...            ...                ...
① LOAD          0x0000000000002d000 0x0000000000002d000 0x0000000000002d000
                0x00000000000ad78d 0x00000000000ad78d  R E    0x1000
```

```
② LOAD          0x0000000000110cf0 0x0000000000111cf0 0x0000000000111cf0
                0x000000000000b914 0x00000000000155a8  RW     0x1000
                        ...              ...              ...

fitz@ubuntu:~$ pmap -d 26958
26958:   bash     ❹
Адрес               Kб Mode  Offset           Device    Mapping
0000563b6b512000    180 r---- 0000000000000000 008:00002 bash
❶ 0000563b6b53f000  696 r-x-- 000000000002d000 008:00002 bash
0000563b6b5ed000    216 r---- 000000000000db000 008:00002 bash
0000563b6b623000     16 r---- 0000000000110000 008:00002 bash
❷ 0000563b6b627000   36 rw--- 0000000000114000 008:00002 bash
0000563b6b630000     40 rw--- 0000000000000000 000:00000   [ anon ]
❸ 0000563b6b8d6000 1168 rw--- 0000000000000000 000:00000   [ anon ]
                        ...              ...              ...
00007f3b6267a000    148 r---- 0000000000000000 008:00002 libc-2.30.so
❸ 00007f3b6269f000 1504 r-x-- 0000000000025000 008:00002 libc-2.30.so
00007f3b62817000    296 r---- 000000000019d000 008:00002 libc-2.30.so
00007f3b62861000     12 r---- 00000000001e6000 008:00002 libc-2.30.so
00007f3b62864000     12 rw--- 00000000001e9000 008:00002 libc-2.30.so
                        ...              ...              ...
00007fff05018000    132 rw--- 0000000000000000 000:00000   [ stack ]
mapped: 12932K    writeable/private: 1468K    shared: 28K
```

В память процесса интерпретатора отображен исполняемый ELF-файл его программы ❶❷ и ELF-файлы всех библиотек ❸, от которых она зависит. Отображение ELF-файлов выполняется частями — *сегментами* (при помощи **readelf(1)** можно получить их список), в зависимости от их назначения. Так, например, сегмент программного кода ① отображен в страницы ❶, доступные на чтение **r** и выполнение **x**, сегмент данных ② отображен в страницы ❷, доступные на чтение **r** и запись **w**, и т. д.

Более того, выделение страниц памяти по требованию (*heap*, куча) в процессе работы процесса ❹ реализуется при помощи «воображаемого» отображения некоторого несуществующего, «*анонимного файла*» **[anon]** на страничные кадры. Необходимо отметить, что механизм виртуальной памяти при освобождении неиспользуемых страниц выгружает в специальную область подкачки SWAP (см. рис. 4.4) только «анонимные» страничные кадры и «анонимизированные», полученные копированием при изменении (согласно принципу COW). Неанонимные измененные кадры выгружаются непосредственно в соответствующие им файлы, а неизмененные освобождаются вовсе без выгрузки, т. к. уже «заранее выгружены».

В примере из листинга 4.38 иллюстрируются два способа выделения памяти по требованию: *явный* — при помощи системного вызова **mmap(2)** с аргументом

MAP_ANONYMOUS, и *неявный*[1] — при помощи системного вызова brk(2). Явный способ ❸ позволяет выделять *новые* сегменты памяти процесса, тогда как неявный способ ❹ изменяет размер предопределенного «сегмента данных» процесса, позволяя увеличивать и уменьшать его по желанию, перемещая так называемый «break» — адрес конца этого сегмента.

Листинг 4.38. Системные вызовы mmap/munmap и brk — выделение и высвобождение памяти

```
fitz@ubuntu:~ $ ldd /usr/bin/hostname
        linux-vdso.so.1 (0x00007ffca19a6000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f5cd5da6000)
☛       /lib64/ld-linux-x86-64.so.2 (0x00007f5cd5fb3000)

fitz@ubuntu:~$ strace hostname
❶ execve("/usr/bin/hostname", ["hostname"], 0x7ffc63e59df0 /* 31 vars */) = 0
  brk(NULL)                                 = 0x55fb3dc7b000
                                ...             ...             ...
┌ openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
❶ fstat(3, {st_mode=S_IFREG|0644, st_size=71063, ...}) = 0
└▶ mmap(NULL, 71063, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f48ae3b7000 ①
  close(3)                                  = 0
                                ...             ...             ...
┌ openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
❷ read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360r\2\0\0\0\0"..., 832) = 832
│    ...           ...             ...             ...
│ fstat(3, {st_mode=S_IFREG|0755, st_size=2025032, ...}) = 0
│    ...           ...             ...             ...
├▶ mmap(NULL, 2032984, PROT_READ, MAP_PRIVATE|..., 3, 0) = 0x7f48ae1c4000
├▶ mmap(0x7f48ae1e9000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|..., 3, 0x25000) = 0x7f48ae1e9000
├▶ mmap(0x7f48ae361000, 303104, PROT_READ, MAP_PRIVATE|..., 3, 0x19d000) = 0x7f48ae361000
└▶ mmap(0x7f48ae3ab000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|..., 3, 0x1e6000) = 0x7f48ae3ab000
❸ mmap(0x7f48ae3b1000, 13656, PROT_READ|PROT_WRITE, MAP_PRIVATE|...|MAP_ANONYMOUS, -1, 0) = 0x7f48ae3b1000
  close(3)                                  = 0
                                ...             ...             ...
② mprotect(0x7f48ae3ab000, 12288, PROT_READ) = 0
  mprotect(0x55fb3beea000, 4096, PROT_READ) = 0
② mprotect(0x7f48ae3f5000, 4096, PROT_READ) = 0
① munmap(0x7f48ae3b7000, 71063)             = 0                  ↰ /lib64/ld-linux-x86-64.so.2
  brk(NULL)                                 = 0x55fb3dc7b000     ↱ /bin/hostname
❹ brk(0x55fb3dc9c000)                       = 0x55fb3dc9c000
  uname({sysname="Linux", nodename="ubuntu", ...}) = 0
```

[1] Доставшийся в наследство от классической ОС UNIX.

```
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...}) = 0
write(1, "ubuntu\n", 7)       = 7
ubuntu
exit_group(0)                           = ?
+++ exited with 0 +++
```

Трасса команды **hostname(1)**, показанная в листинге 4.38, поясняет работу загрузчика и компоновщика (loader, ld) динамических библиотек **ld-linux(8)**.

Системный вызов **exec(2)** ❶ отображает для запуска в память процесса не только заданный ELF-файл **/bin/hostname**, но и (указанный в этом ELF-файле) загрузчик библиотек **/lib64/ld-linux-x86-64.so.2**, которому и передаются управление до самой программы.

Загрузчик библиотек, в свою очередь, отображает ❶ в процесс свой «конфигурационный» файл **/etc/ld.so.cache**, а затем посегментно ❷ отображает файлы всех библиотек и выделяет им ❸ требуемую дополнительную память. Загруженные библиотеки присоединяются (линкуются или же компонуются, linking) к программе **/bin/hostname**, после чего страницам их отображенных сегментов назначается ② соответствующий режим доступа системным вызовом **mprotect(2)**. По завершении компоновки отображение конфигурационного файла **/etc/ld.so.cache** снимается ① при помощи **munmap(2)**, а управление передается исходной программе.

4.7.3. Потребление памяти

Суммарное распределение страниц памяти по сегментам процесса можно получить при помощи третьего набора столбцов (активировав его клавишами **G**,**3**, а затем добавив столбец SWAP клавишей **F**) команды **top(1)**, как показано в листинге 4.39. В столбце VIRT (VSZ в терминах **ps(1)**) изображается суммарный объем (в килобайтах) всех страниц процесса, а в столбце RES (RSS в терминах **ps(1)**) — объем резидентных страниц (находящихся в страничных кадрах оперативной памяти). В столбце SWAP указывается объем всех страниц, находящихся во вторичной памяти — как «анонимных» страниц, выгруженных в специальную область подкачки, так и «файловых» страниц, возможно никогда не загружавшихся в оперативную память.

Столбцы CODE и DATA показывают объемы (в килобайтах) памяти, выделенных под сегменты кода и данных, а столбец SHR — объем резидентных страниц, которые используется (или могут быть использованы) совместно с другими процессами.

Листинг 4.39. Распределение памяти по назначению

```
fitz@ubuntu:~$ top -p 26958
G3top - 22:10:12 up  1:48,  2 users,  load average: 0,01, 0,02, 0,00
```

```
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
МиБ Mem :   3935,6 total,   1356,2 free,    974,9 used,   1604,4 buff/cache
МиБ Swap:    448,5 total,    448,5 free,      0,0 used.   2679,9 avail Mem

  PID  %MEM    VIRT   SWAP   RES    CODE    DATA   SHR  nMaj nDRT  %CPU COMMAND
 7019   0,1   13060      0  5108     876    1600  3600     0    0   0,0 bash
```

Механизм отображения считывает содержимое файла в страничные кадры только один раз, вне зависимости от количества процессов, отображающих этот файл в свою память. В случае отображения одного файла разными процессами (рис. 4.6) их страницы *совместно* отображаются на одни и те же страничные кадры, за исключением страниц, скопированных (согласно принципу COW) при изменении.

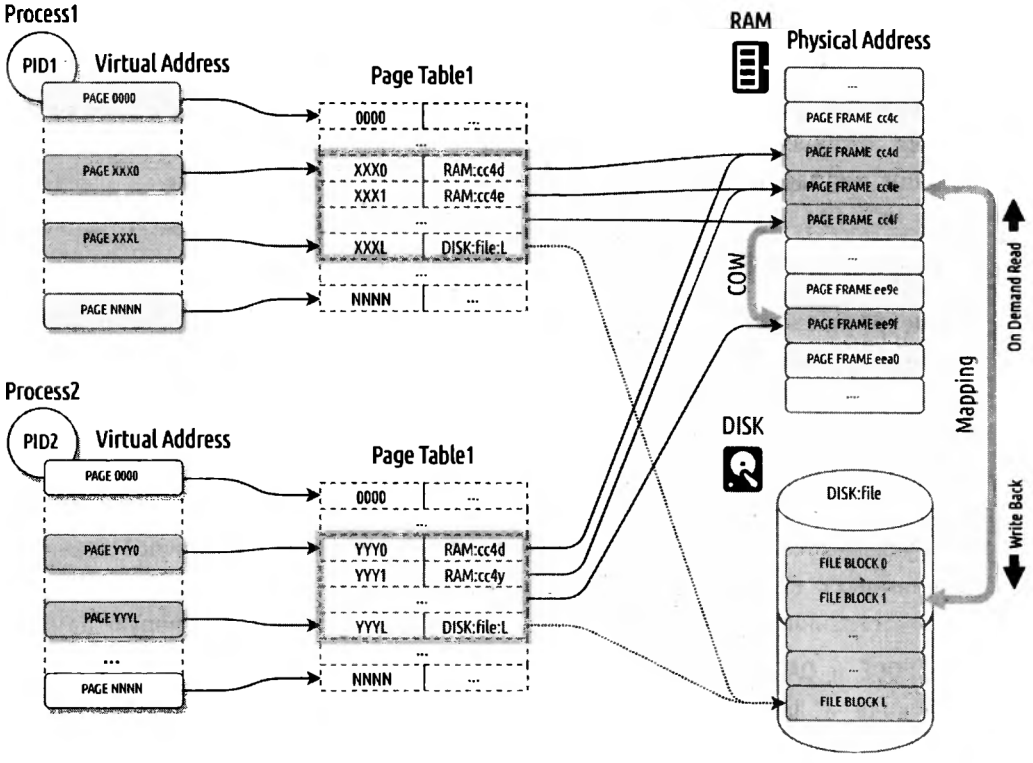


Рис. 4.6. Совместное использование памяти

Такое поведение механизма отображения позволяет эффективно расходовать оперативную память за счет использования разными программами одинаковых разделяемых библиотек. Так как ELF-файлы библиотек «загружаются» в память процессов

при помощи отображения **mmap(2)**, то в результате каждая библиотека размещается в оперативной памяти лишь единожды, вне зависимости от количества ее «использований».

Интегральная статистика по использованию виртуальной памяти может быть получена при помощи команды **free(1)**, как показано в листинге 4.40.

Листинг 4.40. Статистика использования памяти

```
fitz@ubuntu:~$ free -mw
              всего       занято     свободно       общая      буферы  временные    доступно
Память:        3935          975         1354           39          71        1534        2679
Подкачка:       448            0          448

fitz@ubuntu:~$ LANGUAGE=en free -mw
              total        used        free      shared     buffers       cache   available
Mem:           3935          975         1354          39          71        1534        2679
Swap:           448            0          448
```

Строка **Mem:** содержит статистику использования оперативной памяти, а строка **Swap:** — статистику специальной области подкачки. В столбце **total** указан суммарный объем всех доступных страничных кадров, а в столбцах **used** (вычисляется как **used = total - free - buffers - cache**) и **free** — суммарные объемы использованных и свободных страничных кадров соответственно.

В столбце **cache** указан объем *страничного кэша* (page cache), т. е. суммарный объем страничных кадров оперативной памяти, использованных под отображение файлов в память. Необходимо заметить, что именно страничный кэш является неким резервом памяти, которую можно высвободить при первой необходимости, т. к. содержимое этих страниц в реальности всегда можно считать из отображаемых файлов заново (и то, если понадобится) — *см. разд. 4.7.2*.

Аналогично, в столбце **buffers** указывается объем *буферного кэша*, т. е. суммарный объем памяти, использованной ядром для кэширования «неотображаемых» сущностей: метаданных файлов, дисковых блоков при прямом вводе-выводе на устройства и пр.

В столбце **available** предсказывается объем доступной памяти для новых процессов, без вытеснения страниц старых.

В листинге 4.41 показан пример потребления памяти процессом текстового редактора **vi(1)** при попытке редактирования громадного файла в 1 Гбайт. Процесс загружает файл целиком, в результате чего он целиком оказывается в резидентных страницах ❶ сегмента данных процесса, что естественным образом увеличивает

«чистый» расход оперативной памяти системы ❷. После принудительного завершения процесса при помощи команды **kill(1)** память естественным образом высвобождается. Однако, так как под процесс редактора был высвобожден страничный кэш ❸, повторное его наполнение ❹ будет происходить позже, при обращении процессов к своим отображенным файлам (и то, если потребуется).

Листинг 4.41. Потребление памяти процессами

```
                                                                    pts/0
fitz@ubuntu:~$ dd if=/dev/urandom of=big bs=4069 count=262144
262144+0 записей получено
262144+0 записей отправлено
1066663936 байт (1,1 GB, 1017 MiB) скопирован, 13,3567 s, 79,9 MB/s
fitz@ubuntu:~$ ls -lh big
-rw-rw-r-- 1 fitz fitz 1018M ноя 19 23:03 big
fitz@ubuntu:~$ free -mw
               total        used        free      shared     buffers       cache   available
Mem:            3935         987         509          39          71        2366        2656
Swap:            448           0         448
fitz@ubuntu:~$ vi big
```

```
                                                                    pts/1
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
20595 pts/1    S      0:00 -bash
21087 pts/1    R+     0:00  \_ ps f
20437 pts/0    S      0:00 -bash
21085 pts/0    Rl+    0:08  \_ vi big
                             ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪
fitz@ubuntu:~$ ps up 21085
USER         PID %CPU %MEM    VSZ   RSS      TTY      STAT START   TIME COMMAND
❶ fitz     21085 96.4 21.8 1816164 1807248 ➧ pts/0 Sl+  21:14   0:18 vi big

fitz@ubuntu:~$ free -mw
               total        used        free      shared     buffers       cache   available
Mem:            3935      ❷ 2752         133          38          55      ❸ 993         913
Swap:            448           6         442

fitz@ubuntu:~$ top -b -n1 -p 21085
top - 23:11:05 up  2:49,  3 users,  load average: 0,00, 0,09, 0,07
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
```

```
MiB Mem :   3935,6 total,    133,0 free,   2753,0 used,   1049,6 buff/cache
MiB Swap:    448,5 total,    442,0 free,      6,5 used.    913,0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
❶  7680 fitz      20   0 1816164  1.7g   3424 S   0,0  44,8   0:14.90 vi

fitz@ubuntu:~$ kill 21085
fitz@ubuntu:~$ free -m
               total        used        free      shared  buffers     cache   available
Mem:            3935         986        1908          38       56     ❷ 984        2679
Swap:            448           6         442
```

4.8. Механизм сигналов

Механизм сигналов **signal(7)** является простейшей формой межпроцессного взаимодействия и предназначен для внешнего управления процессами. Каждый сигнал имеет свой обработчик, определяющий поведение процесса при отсылке ему этого сигнала. Этот обработчик является неким набором инструкций программы (подпрограммой), которым передается управление при доставке сигнала процессу. Каждому процессу назначаются обработчики «по умолчанию», в большинстве случаев приводящие к завершению процесса.

В примере из листинга 4.42 показано применение сигнала штатного прерывания процесса **№ 2 (SIGINT)**, отсылаемого драйвером терминала всем процессам «переднего» фона при получении символа **^C (ETX)**, т. е. нажатии клавиш Ctrl+C ❶ на этом терминале. Для процессов «заднего» фона сигнал может быть отослан явно ❷, при помощи команды **kill(1)**, предназначенной, несмотря на ее название[1], для отсылки сигналов.

Листинг 4.42. Штатное прерывание процесса (^C, intr, SIGINT)

```
fitz@ubuntu:~$ stty -a
speed 38400 baud; rows 24; columns 80; line = 0;
① intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; ...
                                  ...          ...          ...
② isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop ... -extproc

fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null
❶ ^C782349+0 записей получено
```

[1] В большинстве случаев обработчик завершит процесс, отсюда и название.

```
782349+0 записей отправлено
скопировано 400562688 байт (401 MB), 0,531532 с, 754 MB/с

fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null &
[1] 23418
fitz@ubuntu:~$ ^C
                                ...          ...          ...
fitz@ubuntu:~$ ^C
fitz@ubuntu:~$ jobs -l
[1]+ 23418 Запущен dd if=/dev/zero of=/dev/null &

❷ fitz@ubuntu:~$ kill -SIGINT 23418
fitz@ubuntu:~$ 25318811+0 записей получено
25318810+0 записей отправлено
скопировано 12963230720 байт (13 GB), 17,1001 с, 758 MB/с

[1]+  Прерывание      dd if=/dev/zero of=/dev/null
```

Нужно отметить, что настройки драйвера терминала позволяют как переопределить ① символ, получение которого приведет к выполнению действия **intr** (посылка сигнала **SIGINT**), так и совсем выключить ② отсылку подобных сигналов управления процессами при получении управляющих символов.

В листинге 4.43 показано аналогичное применение сигнала аварийного завершения процесса **№ 3 (SIGQUIT)**, посылаемого процессам «переднего» фона при получении драйвером управляющего символа **^\ (FS)**, генерируемого терминалом при нажатии Ctrl+\. Обработчик сигнала не только завершит процесс, но и попытается (если позволяют настройки) сохранить дамп памяти процесса в файл **core(5)** для последующей отладки.

Листинг 4.43. Аварийное прерывание процесса (^\, quit, SIGQUIT)

```
fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null
^\Выход (стек памяти сброшен на диск)
fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null &
[1] 23429
fitz@ubuntu:~$ kill -SIGQUIT 23429
[1]+  Выход                   (стек памяти сброшен на диск) dd if=/dev/zero of=/dev/null
```

Некоторые процессы (например, демоны, или графические приложения) не имеют управляющего терминала, поэтому не могут быть завершены интерактивно при по-

мощи Ctrl+C или Ctrl+\. В этом случае используется сигнал штатного завершения № 15 (SIGTERM), что проиллюстрировано в листинге 4.44.

Листинг 4.44. Штатное завершение процесса (SIGTERM)

```
fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null &
[1] 23444
fitz@ubuntu:~$ kill -SIGTERM 23444
fitz@ubuntu:~$
[1]+  Завершено       dd if=/dev/zero of=/dev/null
```

Для приостановки процесса, т. е. временного исключения его из процедур распределения процессорного времени планировщиком, предназначен сигнал № 19 (SIGSTOP), а для возобновления процесса — сигнал № 18 (SIGCONT). В листинге 4.45 показано, что приостановленный (sTopped) процесс не потребляет процессорного времени.

Листинг 4.45. Приостановка и возобновление процесса (SIGSTOP и SIGCONT)

```
fitz@ubuntu:~$ pbzip2 big &
[1] 1647
fitz@ubuntu:~$ top -b -n1 -p 1647
                              ...             ...                ...
  PID USER      PR  NI  VIRT  RES  SHR S    %CPU %MEM     TIME+  COMMAND
 1647 fitz      20   0  102m  33m  900 S ☛387,0  0,4   0:25.09 pbzip2

fitz@ubuntu:~$ pkill -STOP pbzip2
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
 1640 pts/2    Ss     0:00 -bash
 1647 pts/2  ☛ Tl     0:24  \_ pbzip2 big
 1651 pts/2    R+     0:00  \_ ps f
fitz@ubuntu:~$ jobs -l
[1]+  1647 Остановлен      pbzip2 big
fitz@ubuntu:~$ top -b -n1 -p 1647
                              ...             ...                ...
  PID USER      PR  NI  VIRT  RES  SHR    S    %CPU %MEM     TIME+  COMMAND
 1647 fitz      20   0  102m  34m  900 ☛ T  ☛ 0,0  0,4   0:42.90 pbzip2

fitz@ubuntu:~$ kill -CONT 1647
fitz@ubuntu:~$ top -b -n1 -p 1647
                              ...             ...                ...
  PID USER      PR  NI  VIRT  RES  SHR S    %CPU %MEM     TIME+  COMMAND
 1647 fitz      20   0  102m  34m  900 S ☛370,0  0,4   0:51.82 pbzip2
```

Сигналы могут быть перехвачены, если процесс назначит собственный обработчик, а также проигнорированы, тогда при их доставке вообще никакой обработчик не вызывается. Исключение составляют некоторые «безусловные» сигналы, такие как № 9 (SIGKILL) — безусловное завершение или № 19 (SIGSTOP) — безусловная приостановка процесса. Игнорирование и перехват сигналов показаны в листинге 4.46 на примере командного интерпретатора bash(1). Ни попытки «интерактивного» завершения ❶ при помощи сигналов SIGINT и SIGQUIT, ни явная посылка ❷ сигналов SIGINT и SIGTERM не приводят к завершению командного интерпретатора. К желаемому результату приводит только явная отсылка ❸ сигнала SIGKILL.

Листинг 4.46. Игнорирование и перехват сигналов

```
   fitz@ubuntu:~$ ps f
     PID TTY      STAT   TIME COMMAND
   23025 pts/1    S      0:00 -bash
   23771 pts/1    R+     0:00  \_ ps f
   fitz@ubuntu:~$ bash
   fitz@ubuntu:~$ ps f
     PID TTY      STAT   TIME COMMAND
   23025 pts/1    S      0:00 -bash
   23636 pts/1    S      0:00  \_ bash
   23692 pts/1    R+     0:00      \_ ps f
❶  fitz@ubuntu:~$ ^C
❶  fitz@ubuntu:~$ ^\
   fitz@ubuntu:~$ ps f
     PID TTY      STAT   TIME COMMAND
   23025 pts/1    S      0:00 -bash
!  23636 pts/1    S      0:00  \_ bash
   23692 pts/1    R+     0:00      \_ ps f

❷  fitz@ubuntu:~$ kill -SIGINT 23636
   fitz@ubuntu:~$ ps f
     PID TTY      STAT   TIME COMMAND
   23025 pts/1    S      0:00 -bash
!  23636 pts/1    S      0:00  \_ bash
   23701 pts/1    R+     0:00      \_ ps f
❷  fitz@ubuntu:~$ kill -SIGTERM 23636
   fitz@ubuntu:~$ ps f
     PID TTY      STAT   TIME COMMAND
```

```
  23025 pts/1    S      0:00 -bash
! 23636 pts/1    S      0:00  \_ bash
  23708 pts/1    R+     0:00      \_ ps f
❶ fitz@ubuntu:~$ kill -SIGKILL 23636
  Убито
  fitz@ubuntu:~$ ps f
    PID TTY      STAT   TIME COMMAND
  23025 pts/1    S      0:00 -bash
  23771 pts/1    R+     0:00  \_ ps f
```

Диспозицию сигналов, т. е. информацию об игнорировании (**IGNORED**), перехвате (**CAUGHT**), временной блокировке (**BLOCKED**) или ожидании доставки (**PENDING**) сигналов процессов можно получить при помощи команды **ps(1)**, как показано в листинге 4.47. Диспозиция изображается битовой маской, где каждый N-й бит маски (нумерация от младших к старшим) соответствует сигналу N. Например, командный интерпретатор игнорирует сигналы, представленные (шестнадцатеричной) маской 00384004_{16}, что в двоичном представлении составляет $11100001000000000000100_2$ и указывает на игнорируемые 3-й, 15-й, 20-й, 21-й и 22-й сигналы, т. е. **SIGQUIT**, **SIGTERM**, **SIGTSTP**, **SIGTTIN** и **SIGTTOU**. Для перевода из шестнадцатеричного в двоичное представление использован стековый калькулятор **dc(1)**, которому было велено из входной **i** (input) системы счисления по основанию **16** в выходную **o** (output) систему счисления по основанию **2** напечатать **p** (print) число **00384004**, а для просмотра имен сигналов по их номерам использована встроенная команда **kill**.

Листинг 4.47. Диспозиция сигналов

```
fitz@ubuntu:~$ ps s
  UID   PID   PENDING   BLOCKED   IGNORED    CAUGHT STAT TTY        TIME COMMAND
 1006  23025 00000000  00010000  00380004  4b813efb Ss   pts/5      0:00 -bash
 1006  23773 00000000  00000000  00000000 <f3d1fef9 R+   pts/5      0:00 ps s
fitz@ubuntu:~$ dc -e 16i2o00380004p
11100000000000000000100
222120----------87654321
fitz@ubuntu:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT       4) SIGILL       5) SIGTRAP
                                  ...              ...             ...
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE      14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT      19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG       24) SIGXCPU     25) SIGXFSZ
                                  ...              ...             ...
63) SIGRTMAX-164) SIGRTMAX
```

4.8.1. Сеансы и группы процессов: управление заданиями

Одним из основных применений сигналов при интерактивной работе пользователя в системе является механизм управления «заданиями», которыми пользуются командные интерпретаторы и подобные интерактивные программы, например **lftp(1)**.

Для удобства управления процессами при помощи сигналов они объединяются в *группы* и *сеансы* (см. **credentials(7)**), проиллюстрированные в листинге 4.48 при помощи команды **ps(1)** атрибутами **PGID** (**p**rocess **g**roup **id**entifier) и **SID** (session **id**entifier). Процесс (создавший группу), чей идентификатор **PID** совпадает ② с идентификатором **PGID** группы, носит название *лидера группы*. Процесс (создавший сеанс), чей идентификатор **PID** совпадает ① с идентификатором **SID** сеанса, называется *лидером сеанса*. Нужно отметить, что лидер сеанса в столбце **STAT** отмечается флагом **s**, а процессы группы переднего фона — флагом **+**. Только одна группа сеанса, называемая «терминальной» **TGPID**, является группой «переднего» (foreground) фона, остальные группы сеанса являются группами «заднего» (background) фона.

Командный интерпретатор формирует свои задания «заднего» ❶ или «переднего» фона ❷, помещая процессы заданий в соответствующие группы. Механизм управления заданиями всегда посылает «терминальные» сигналы **^C SIGINT**, **^\SIGQUIT** всем процессам текущей «терминальной» группы. Для смены терминальной группы используется сигнал **№ 20 SIGTSTP** (**t**erminal **st**o**p** **sig**nal), также отсылаемый всем процессам «терминальной» группы при получении драйвером терминала управляющего символа **^Z** (**SUB**), генерируемого клавишами Ctrl+Z. Обработчик сигнала **SIGTSTP** по умолчанию приостанавливает процессы, и управление возвращается к командному интерпретатору ❸, группа которого становится «терминальной». При помощи встроенных команд `fg` (**fg**round), `bg` (**bg**round) можно продолжить (**SIGCONT**) выполнение указанного задания (всех процессов его группы) на «переднем» или «заднем» фоне, а при помощи команды `jobs -l` получить список всех заданий вместе с номерами их групп процессов.

Листинг 4.48. Сеансы и группы процессов — задания интерпретатора

```
❶ fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null &
  [1] 3181
❷ fitz@ubuntu:~$ ps jf
   PPID   PID  PGID   SID  TTY      TPGID STAT   UID   TIME COMMAND
①  3094  3099••3099••3099 pts/0     3159 Ss     1000   0:00 bash
②  3099  3181••3181  3099 pts/0     3182 R      1000   0:08  \_ dd if=/dev/zero of=...
②  3099  3182••3182  3099 pts/0     3182 R+     1000   0:00  \_ ps jf
```

```
fitz@ubuntu:~$ man dd
                ...     ...     Ctrl+Z                  ...
❸ [2]+  Остановлен  man dd
fitz@ubuntu:~$ jobs -l
[1]-  3181 Запущен          dd if=/dev/zero of=/dev/null &
[2]+  3193 Остановлено      man dd
fitz@ubuntu:~$ ps jf
PPID   PID  PGID   SID  TTY      TPGID STAT   UID   TIME COMMAND
 3094  3099  3099  3099┐ pts/0     3310 Ss    1000   0:00 bash
 3099  3181  3181  3099│ pts/0     3310 R+    1000   4:48  \_ dd if=/dev/zero of=
 3099  3193  3193┐ 3099│ pts/0     3310 T+    1000   0:00  \_ man dd
 3193  3203  3193┘ 3099│ pts/0     3310 T+    1000   0:00  |   \_ pager -s
 3099  3310  3310  3099┘ pts/0     3310 R+    1000   0:00  \_ ps jf
fitz@ubuntu:~$ fg %1
dd if=/dev/zero of=/dev/null
^Z
[1]+  Остановлен     dd if=/dev/zero of=/dev/null
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
 3099 pts/0    Ss     0:00 bash
 3181 pts/0    T+    26:44  \_ dd if=/dev/zero of=/dev/null
 3193 pts/0    T      0:00  \_ man dd
 3203 pts/0    T      0:00  |   \_ pager -s
 3937 pts/0    R+     0:00  \_ ps f
fitz@ubuntu:~$ bg 1
[1]+ dd if=/dev/zero of=/dev/null &
fitz@ubuntu:~$ fg %2
     ...     ...     Q       ...     ...
man dd
fitz@ubuntu:~$ jobs
[1]+  Запущен     dd if=/dev/zero of=/dev/null &
fitz@ubuntu:~$ fg
dd if=/dev/zero of=/dev/null
^C11771330744+0 записей получено
11771330743+0 записей отправлено
скопировано 6026921340416 байт (6,0 TB), 8400,83 с, 717 MB/с
```

Кроме переключения группы «переднего» фона, механизм управления заданиями координирует «совместный» доступ процессов к управляющему терминалу. При

«одновременном» вводе информации с одного терминала несколькими процессами результат оказывается непредсказуем, т. к. нет возможности предугадать порядок и объемы считываемой информации. Поэтому ввод (input) разрешен только процессам группы «переднего» фона, а группа формируется так, что только один из них в реальности будет производить чтение. Процессам группы «заднего» фона ввод запрещен, а любые попытки подавляются при помощи сигнала SIGTTIN (terminal stop on input signal), доставка которого приводит к приостановке процесса.

В примере из листинга 4.49 при составлении текста письма посредством команды mail(1) ее процесс был временно приостановлен ❶ при помощи ^Z и SIGTSTP для получения доступа к командному интерпретатору. Попытка продолжить ❷ выполнение задания mail на «заднем» фоне не увенчалась успехом, т. к. была подавлена ❸ за чтение терминала. Продолжение задания на «переднем» фоне ❹ дает возможность закончить ввод текста письма и завершить ввод управляющим символом ^Z (EOT).

Листинг 4.49. Приостановка при вводе из заднего фона (SIGTTIN)

```
   fitz@ubuntu:~$ mail dketov@gmail.com
   Subject: schedtool(1) вместо taskset, chrt и nice/renice
   ^Z
❶  [2]+  Остановлен      mail dketov@gmail.com
   fitz@ubuntu:~$ which schedtool
   /usr/bin/schedtool
   fitz@ubuntu:~$ dpkg -S /usr/bin/schedtool
   schedtool: /usr/bin/schedtool
❷  fitz@ubuntu:~$ bg
?  [1]+ mail dketov@gmail.com &
?  (continue)
?  fitz@ubuntu:~$ jobs -l
❸  [1]+ 12025 Остановлено (вывод на терминал)     mail dketov@gmail.com
   fitz@ubuntu:~$ ps f
     PID TTY      STAT   TIME COMMAND
    8992 pts/2    Ss     0:00 bash
   12025 pts/2    T      0:00  \_ mail dketov@gmail.com
   12063 pts/2    R+     0:00  \_ ps f
    8897 pts/0    Ss+    0:00 bash
❹  fitz@ubuntu:~$ fg
   mail dketov@gmail.com
   (continue)
```

```
Утилита schedtool(1) из одноименного пакета schedtool заменяет "стандартные"
taskset/nice/renice/chrt↵
^D
Cc:↵
fitz@ubuntu:~$
```

При «одновременном» выводе информации на один терминал несколькими процессами результат точно так же непредсказуем, как и при вводе. В итоге будет получена смесь перемежающихся строчек разных процессов, однако по умолчанию вывод (output) разрешен как процессам группы «переднего» фона, так и процессам всех групп «заднего» фона. Настроечный флаг драйвера терминала **tostop** (terminal output stop) позволяет запретить вывод из заднего фона так же, как и ввод. При запрещенном выводе из заднего фона все попытки будут подавляться сигналом **SIGTTOU** (terminal stop on output signal), приостанавливающим процесс. В листинге 4.50 проиллюстрировано действие сигнала **SIGTTOU** при включении настроечного флага **tostop** посредством команды **stty(1)**.

Листинг 4.50. Приостановка при выводе из заднего фона (SIGTTOU)

```
   fitz@ubuntu:~$ find / -type f -size 0 &
!?      ...      ^C      ...      ^\      ...      ^Z      ...              ...
                                  ① ① ① ① ① ① ① ① ① ① ①
   fitz@ubuntu:~$ stty -a
   speed 38400 baud; rows 24; columns 80; line = 0;
                                  ...          ...          ...
   isig icanon iexten echo ... -echonl -noflsh -xcase -tostop -echoprt echoctl echoke
   fitz@ubuntu:~$ stty tostop
   fitz@ubuntu:~$ find / -type f -size 0 &
   [1] 356
   fitz@ubuntu:~$ jobs -l
   [1]+   356 Остановлено (вывод на терминал)      find / -type f -size 0
   fitz@ubuntu:~$ ps f
     PID TTY      STAT   TIME COMMAND
   32535 pts/1    S      0:00 -bash
     356 pts/1    T      0:00  \_ find / -type f -size 0
     360 pts/1    R+     0:00  \_ ps f
```

4.9. Межпроцессное взаимодействие

Кроме сигналов, которые могут использоваться как простейшие средства межпроцессного взаимодействия (IPC, inter-process communication), для эффективного обмена информацией между процессами применяются каналы, сокеты, очереди со-

общений и разделяемая память, а для синхронизации действий процессов над совместно используемыми объектами — семафоры.

4.9.1. Неименованные каналы

Самое простое средство обмена информацией между родственными процессами (родитель и любые его потомки) — *неименованные каналы*. Канал является «двусторонним однонаправленным безымянным файлом», с одного конца в который можно только записывать информацию, а с другого конца — только считывать. В отличие от «обычного» файла, при открытии которого создается только один файловый дескриптор (и для чтения, и для записи файла), при создании канала создаются сразу два дескриптора — один для передачи (записи) в канал, а другой — для приема (чтения) из канала. При порождении дочерних процессов файловые дескрипторы наследуются, что и позволяет им взаимодействовать как с родительским процессом, так и между собой.

Использование неименованных каналов в системе широко распространено. Например, командный интерпретатор применяет их для организации конвейерной обработки *(см. разд. 5.3)*. Архиватор **tar(1)** аналогичным способом — для упаковки архивов «на лету» при помощи «внешних» упаковщиков. В примере из листинга 4.51 посредством трассировщика **strace(1)** отслеживается системный вызов **pipe(2)**, при помощи которого **tar(1)** создает неименованный канал для связи с упаковщиком **xz(1)** в дочернем процессе. Вся группа процессов архиватора **tar(1)** приостановлена сигналом **^Z SIGTSTP**, после чего при помощи команды **lsof(1)** показаны файловые дескрипторы открытых файлов обоих процессов.

Листинг 4.51. Неименованные каналы

```
fitz@ubuntu:~$ strace -fe pipe,execve tar cJf /tmp/docs.tgz  /usr/share/doc
execve("/usr/bin/tar", ["tar", "cJf", "/tmp/docs.tgz", "/usr/share/doc"], ...) = 0
pipe([3, 4])                            = 0
strace: Process 9274 attached
tar: Удаляется начальный `/' из имен объектов
[pid  9274] execve("/bin/sh", ["/bin/sh", "-c", "xz"], 0x7fff7fd6d060 ...) = 0
strace: Process 9275 attached
[pid  9275] execve("/usr/bin/xz", ["xz"], 0x5557bddfdb48 /* 31 vars */) = 0
[pid  9275] pipe([3, 4])                = 0
                          ...           ...             ...
^Z
[1]+  Остановлен  strace -fe pipe,execve tar cJf /tmp/docs.tgz /usr/share/doc
fitz@ubuntu:~$ ps f
 8547 pts/1    Ss     0:00 -bash
```

```
 9270 pts/1    T      0:03  \_ strace -fe pipe,execve tar cJf /tmp/docs.tgz /usr
 9273 pts/1    t      0:03  |   \_ tar cJf /tmp/docs.tgz /usr/share/doc
 9274 pts/1    t      0:00  |       \_ /bin/sh -c xz
 9275 pts/1    t      0:27  |           \_ xz
 9321 pts/1    R+     0:00  \_ ps f
fitz@ubuntu:~$ lsof -p 9273
COMMAND  PID USER   FD   TYPE  DEVICE SIZE/OFF     NODE NAME
tar     9273 fitz  cwd    DIR     8,2     4096   417609 /home/fitz
tar     9273 fitz  rtd    DIR     8,2     4096        2 /
tar     9273 fitz  txt    REG     8,2   452048   397186 /usr/bin/tar
                              ...          ...          ...
tar     9273 fitz   0u   CHR  136,1      0t0        4 /dev/pts/1
tar     9273 fitz   1u   CHR  136,1      0t0        4 /dev/pts/1
tar     9273 fitz   2u   CHR  136,1      0t0        4 /dev/pts/1
tar     9273 fitz   3r   DIR     8,2    69632   409070 /usr/share/doc
tar     9273 fitz   4w   FIFO    0,13     0t0    75234 pipe
                             ...          ...          ...
fitz@ubuntu:~$ lsof -p 9275
COMMAND  PID USER   FD   TYPE  DEVICE SIZE/OFF     NODE NAME
xz      9275 fitz  cwd    DIR     8,2     4096   417609 /home/fitz
xz      9275 fitz  rtd    DIR     8,2     4096        2 /
xz      9275 fitz  txt    REG     8,2    80224   397427 /usr/bin/xz
                             ...          ...          ...
xz      9275 fitz   0r   FIFO    0,13     0t0    75234 pipe
xz      9275 fitz   1w   REG     8,2 19365888   394591 /tmp/docs.tgz
xz      9275 fitz   2u   CHR  136,1      0t0        4 /dev/pts/1
xz      9275 fitz   3r   FIFO    0,13     0t0    75937 pipe
xz      9275 fitz   4w   FIFO    0,13     0t0    75937 pipe
                             ...          ...          ...
```

Нужно отметить, что в процессе архиватора **PID = 9273** файловый дескриптор передающей части канала сохранил свой номер, а в дочернем процессе упаковщика **PID = 9375** файловый дескриптор принимающей части канала был перенаправлен на **STDIN**, как того и ожидает упаковщик **xz(1)**.

4.9.2. Именованные каналы

Именованные каналы повторяют поведение неименованных каналов, но предназначены для обмена информацией между *неродственными* процессами. Любые две запущенные программы могут организовать однонаправленный канал передачи путем открытия файла канала по заранее согласованному *имени* на запись «с одной стороны» и на чтение «с другой». Файл канала должен быть предварительно создан в дереве каталогов при помощи специального системного вызова **mkfifo(3)**, а

его последующее открытие осуществляется «обычным» системным вызовом **open(2)**. Именованные каналы сейчас используются крайне редко и практически вытеснены именованными локальными сокетами, но в некоторых редких случаях все еще являются вполне достаточным средством взаимодействия.

В качестве примера на листинге 4.52 показан именованный канал **/run/initctl** (ранее размещался в **/dev/initctl**), через который **systemd(1)** все еще для совместимости принимает устаревшие *(legacy)* команды устаревшего **init(8)** для переключения «уровней исполнения» системы *(run-level)*, см. **boot(7)**.

Листинг 4.52. Именованные каналы

```
fitz@ubuntu:~$ ls -l /run/initctl /dev/initctl
lrwxrwxrwx 1 root root 12 ноя 22 22:01 /dev/initctl -> /run/initctl
prw------- 1 root root 0 ноя 22 19:24 /run/initctl
fitz@ubuntu:~$ sudo lsof /run/initctl
COMMAND PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
systemd   1 root   26u  FIFO   0,23      0t0  276 /run/initctl
```

4.9.3. Неименованные локальные сокеты

Каналы являются однонаправленными средствами взаимодействия процессов, поэтому слабо подходят для двунаправленного обмена, например для организации обратной связи между процессами. В большинстве случаев именованные каналы позволяют эффективно реализовать только простейшую модель взаимодействия «поставщик → потребитель», тогда как для реализации модели «клиент ⇆ сервер» используют специальное средство взаимодействия, называемое *сокетом*[1]. Неименованные локальные (файловые) сокеты, как и неименованные каналы, являются «безымянными файлами», только *двунаправленными*, и так же предназначены для взаимодействия родственных процессов. Для создания пары соединенных сокетов используется системный вызов **socketpair(2)**, создающий пару файловых дескрипторов, каждый из которых используется одновременно и для приема (чтения), и для передачи (записи) информации. Применение неименованных файловых сокетов проиллюстрировано в листинге 4.53 на примере синхронизирующего копировщика **rsync(1)**. Копировщик оказывается параллельной программой, использующей сокеты для взаимодействия своих параллельных ветвей, первая из которых работает с исходными файлами, а вторая — с результирующими. При этом сокет используется как для обмена информацией о свойствах файлов для детектирования разни-

[1] Сокет — устоявшаяся русская калька с англ. *socket*, буквально означающая «разъем», например, 220В розетку и вилку, или сетевую розетку и вилку RJ-45, или 3,5 мм гнездо для наушников и соответствующий штекер.

цы между ними, так и для последующей передачи самих данных различающихся файлов.

Листинг 4.53. Неименованные локальные (файловые) сокеты

```
fitz@ubuntu:~$ strace -fe socketpair,execve rsync -a /usr/share/doc /tmp/
execve("/usr/bin/rsync", ["rsync", "-a", "/usr/share/doc", "/tmp/"], ...) = 0
socketpair(AF_UNIX, SOCK_STREAM, 0, [3, 4]) = 0
socketpair(AF_UNIX, SOCK_STREAM, 0, [5, 6]) = 0
strace: Process 5268 attached
[pid  5268] socketpair(AF_UNIX, SOCK_STREAM, 0, [3, 4]) = 0
strace: Process 5269 attached
^Z
[1]+  Остановлен      strace -fe socketpair,execve rsync -a /usr/share/doc /tmp/
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
 4838 pts/5    Ss     0:00 bash
 5266 pts/5    T      0:00  \_ strace -fe socketpair execve rsync -a /usr/share/
 5267 pts/5    t      0:00  |   \_ rsync -a /usr/share/doc /tmp/
 5268 pts/5    t      0:00  |       \_ rsync -a /usr/share/doc /tmp/
 5269 pts/5    t      0:00  |           \_ rsync -a /usr/share/doc /tmp/
 5301 pts/5    R+     0:00  \_ ps f

fitz@ubuntu:~$ lsof -p 5269
COMMAND  PID   USER   FD   TYPE             DEVICE SIZE/OFF     NODE NAME
rsync   5269   fitz  cwd    DIR              252,0   139264 26869761 /tmp
...
rsync   5269   fitz   0u  unix 0x0000000000000000      0t0  1752340 type=STREAM
...
rsync   5269   fitz   4u  unix 0x0000000000000000      0t0  1751443 type=STREAM

fitz@ubuntu:~$ lsof -p 5268
COMMAND  PID   USER   FD   TYPE             DEVICE SIZE/OFF     NODE NAME
rsync   5268   fitz  cwd    DIR              252,0   139264 26869761 /tmp
...
rsync   5268   fitz   1u  unix 0x0000000000000000      0t0  1752343 type=STREAM
...
rsync   5268   fitz   3u  unix 0x0000000000000000      0t0  1751442 type=STREAM

fitz@ubuntu:~$ lsof -p 5267
COMMAND  PID   USER   FD   TYPE             DEVICE SIZE/OFF     NODE NAME
rsync   5267   fitz  cwd    DIR              252,0    20480  3801093 /usr/share
...
```

```
rsync   5267   fitz    4u  unix 0x0000000000000000      0t0  1752341 type=STREAM
rsync   5267   fitz    5u  unix 0x0000000000000000      0t0  1752342 type=STREAM
```

4.9.4. Именованные локальные сокеты

Именованные локальные сокеты, как и именованные каналы, предназначаются для взаимодействия неродственных процессов и широко распространены в системе.

В примере из листинга 4.54 показан терминальный мультиплексор **W:[GNU screen]**, который использует именованные сокеты для взаимодействия между своими отключенным ❶ (detached) и повторно подключающимся (reattach, **-r**) ❷ экземплярами.

Терминальные мультиплексоры **screen(1)** и **tmux(1)** позволяют запускать несколько пользовательских «вторичных» сеансов одновременно, переключаться между ними, отсоединяться от них и снова присоединяться к ним из «первичного» сеанса. Например, запустив терминальный мультиплексор в сеансе алфавитно-цифрового виртуального терминала и отключившись от него, можно подключиться к нему и продолжить работу уже из графического эмулятора терминала или по сети с использованием службы удаленного доступа **SSH** *(см. разд. 6.4.1)*.

Листинг 4.54. Именованные локальные (файловые) сокеты

Первичный сеанс — tty1

```
fitz@ubuntu:~$ tty
tty1
fitz@ubuntu:~$ screen
```

Вторичный сеанс #1 — pts/2

```
fitz@ubuntu:~$ tty
/dev/pts/2
fitz@ubuntu:~$ youtube-dl https://www.youtube.com/watch?v=kbEKbmpZKzo
...
[download] Destination: Основы Linux - командная строка-kbEKbmpZKzo.f248.webm
[download]   3.8% of ~81.04MiB at  3.52MiB/s ETA 04:08
                              ... Ctrl+A c          ...
```

Вторичный сеанс #2 — pts/4

```
fitz@ubuntu:~$ tty
/dev/pts/4
fitz@ubuntu:~$          ...     Ctrl+A D          ...
```

```
❶ [detached from 22261.pts-14.ubuntu]
fitz@ubuntu:~$ tty
tty1
fitz@ubuntu:~$ ps fp 22261 t pts/2,pts/4
  PID TTY      STAT   TIME COMMAND
```

```
22261 ?        Ss     0:00 SCREEN
22262 pts/2    Ss+    0:00  \_ /bin/bash
22263 pts/2    S+     0:10  \_ /usr/bin/python3 /usr/bin/youtube-dl ...
28020 pts/4    Ss+    0:00  \_ /bin/bash
fitz@ubuntu:~$ logout
```

Первичный сеанс tty1

```
fitz@somewhere:~$ ssh ubuntu
No mail.
Last login: Sat Nov 23 01:01:04 2019 from 10.0.2.2
fitz@ubuntu:~$ screen -ls
There is a screen on:
        22261.pts-14.ubuntu     (23.11.2019 01:01:06)   (Detached)
1 Socket in /run/screen/S-fitz.
fitz@ubuntu:~$ ls -l /run/screen/S-fitz
итого 0
srwx------ 1 fitz fitz 0 ноя 20 00:40 22261.pts-14.ubuntu
fitz@ubuntu:~$ screen -r
```

Продолжаем вторичный сеанс №2 pts/4

```
fitz@ubuntu:~$ tty
/dev/pts/4
fitz@ubuntu:~$ ^D
```

Продолжаем вторичный сеанс №1 pts/2

```
[download] 100% of 219.30MiB in 06:28
[dashsegments] Total fragments: 246
[download] Destination: Основы Linux - командная строка-kbEKbmpZKzo.f251.webm
[download] 100% of 26.45MiB in 00:19
[ffmpeg] Merging formats into "Основы Linux - командная строка-kbEKbmpZKzo.webm"
Deleting original file Основы Linux - командная строка-kbEKbmpZKzo.f248.webm (pass -k to
keep)
Deleting original file Основы Linux - командная строка-kbEKbmpZKzo.f251.webm (pass -k to
keep)
fitz@ubuntu:~$ tty
/dev/pts/2
fitz@ubuntu:~$ ^D
```

```
[screen is terminating]
```

4.9.5. Разделяемая память, семафоры и очереди сообщений

Разделяемая память

Каналы и сокеты являются удобными средствами обмена информацией между процессами, но их использование при интенсивном обмене или обмене объемными

данными приводит к значительным накладным расходам. *Разделяемая память* является специализированным средством взаимодействия, имеющим минимальные издержки использования.

В листинге 4.55 при помощи команды **ipcs(1)** показаны созданные в системе сегменты разделяемой памяти, массивы семафоров и очереди сообщений — средства межпроцессного взаимодействия **svipc(7)**, «унаследованные» Linux от W:[UNIX System V]. Экземпляры средств **System V IPC** идентифицируются при помощи глобально уникальных ключей ❶, или локальных[1] идентификаторов ❷ **shmid** (**sh**ared **m**emory **id**entifier), **semid** (**sem**aphore **id**entifier) и **msqid** (**m**essage **q**ueue **id**entifier), и сродни файлам имеют владельцев и права доступа.

Листинг 4.55. Разделяемая память, очереди сообщений и семафоры (System V IPC)

```
fitz@ubuntu:~$ ipcs

------ Очереди сообщений --------
ключ   msqid      владелец права исп. байты сообщения

------ Сегменты совм. исп. памяти --------
ключ ❶     shmid ❷    владелец  права     байты      nattch     состояние
0x0052e2c1 0          postgres  600       56         6
0x00000000 11567105   fitz      600       393216     2          назначение
0x00000000 10944514   fitz      700       1694000    2          назначение
                      ...       ...       ...                   ...
0x00000000 11206666   fitz      600       393216     2          назначение

------ Массивы семафоров --------
ключ   semid      владелец права nsems
```

Разделяемая память **System V IPC** реализуется ядром на основе механизма страничного отображения при помощи совместного использования страничных кадров страницами разных процессов (см. рис. 4.6). При помощи системного вызова **shmget(2)** один из взаимодействующих процессов создает сегмент памяти, состоящий из страничных кадров без отображения на какой-либо файл. Впоследствии кадры этого сегмента при помощи системного вызова **shmat(2)** (**sh**ared **m**emory **at**tach) отображаются на страницы всех взаимодействующих процессов, за счет чего эти процессы и используют выделенную память совместно.

Иллюстрация применения разделяемой памяти приведена в листинге 4.56, где сегменты разделяемой памяти с идентификаторами 18_{16} и 12_{16} были созданы (cpid,

[1] Идентификаторы IPC (сродни индексным и файловым дескрипторам файлов) изменяются при пересоздании/переоткрытии экземпляра, а ключи IPC (подобно именам файлов) — нет.

creator **pid**) процессом **PID = 3828** программы X-клиента **gnome-shell**, а последнее обращение к нему (lpid, last operation **pid**) осуществлялось процессом **PID = 3450** программы X-сервера **Xorg(1)**. В данном конкретном случае X-клиент и X-сервер *(см. разд. 7.1)* для эффективного обмена значительными объемами растровых изображений используют расширение X-протокола **W:[MIT-SHM]** *(см. разд. 7.6)*, основанное на применении разделяемой памяти.

Листинг 4.56. Разделяемая память (System V IPC)

```
fitz@ubuntu:~$ ipcs -m -p
------ Shared Memory Creator/Last-op PIDs --------
shmid      владелец  cpid       lpid
                     ...        ...              ...              ...
23         fitz      3887       5007
26         fitz      3887       5007
❶ 38       fitz      3887       3541
40         fitz      3887       3541
                     ...        ...              ...              ...
fitz@ubuntu:~$ dc -e 16o10i38p
☛ 26
fitz@ubuntu:~$ dc -e 16o10i40p
☛ 28
fitz@ubuntu:~$ ps up 3887,3541,5007
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
fitz      3541  0.0  1.7 376008 68532 tty2     Sl+  15:12   0:04 /usr/lib/xorg/Xorg ...
fitz      3887  0.4  7.9 2646616 322356 ?      Ssl  15:12   0:50 /usr/bin/gnome-shell
fitz      5007  0.2  7.1 3072068 286592 ?      Sl   15:16   0:26 /usr/lib/.../firefox ...

fitz@ubuntu:~$ pmap 3887
3887:   /usr/bin/gnome-shell
                     ...        ...                         ...
00007ff280123000   3952K rw-s-   [ shmid=0x1a ]
❶ 00007ff2a11d7000   3744K rw-s-   [ shmid=0x28 ]
00007ff2a157f000    512K rw-s-   [ shmid=0x26 ]
00007ff2c413e000     16K rw-s-   [ shmid=0x17 ]
                     ...        ...              ...              ...
fitz@ubuntu:~$ pmap 3541
3541:   /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority ...
                     ...        ...              ...              ...
❷ 00007f70a7c58000   3744K rw-s-   [ shmid=0x28 ]
00007f70b4591000    512K rw-s-   [ shmid=0x26 ]
                     ...        ...              ...              ...
00007f70bbe32000   3952K rw-s-   [ shmid=0x1a ]
                     ...        ...              ...              ...
```

Анализ карт памяти взаимодействующих процессов при помощи команды **pmap(1)** подтверждает, что страничные кадры сегмента разделяемой **s** памяти 26_{16} ($1A_6$) и 40_{10} (28_{16}) были отображены на страницы обоих взаимодействующих процессов по разным виртуальным адресам ❶ и ❷.

Еще один вариант реализации разделяемой памяти, пришедший в Linux из **W:[BSD]**, основывается на «разделяемом» (shared) отображении одного и того же файла в память разных процессов при помощи «штатного» механизма **mmap(2)**. В листинге 4.57 показан типичный пример применения совместного отображения файла **/var/cache/nscd/hosts** в память процесса **nscd** (name service cache daemon) и многих других процессов, пользующихся его услугами.

Служба имен (name service) предназначена для извлечения свойств различных каталогизируемых сущностей по их имени. Например, по имени интернет-узла из каталога системы доменных имен DNS служба имен извлекает IP-адрес этого узла, и наоборот, по IP-адресу — имя. При большом количестве повторяющихся запросов к службе имен в течение промежутка времени, в который изменения запрашиваемой информации в соответствующем каталоге объектов не происходит, «бесполезные» повторные запросы к каталогу могут быть сокращены при помощи сохранения и использования предыдущих ответов (кэширования), чем и занимается демон **nscd(8)**.

В частности, ответы из DNS, сохраняются процессом **nscd** в страничных кадрах памяти отображенного файла **/var/cache/nscd/hosts** ❶, а остальные процессы совместно их используют, отображая тот же файл в страницы своей памяти ❷.

Листинг 4.57. Разделяемая память (BSD)

```
fitz@ubuntu:~ $ sudo fuser -v /var/cache/nscd/hosts
                     ПОЛЬЗ-ЛЬ    PID ДОСТУП КОМАНДА
/var/cache/nscd/hosts:
                     root        668 ....m NetworkManager
                     syslog      680 ....m rsyslogd
                     root        683 ....m snapd
                     root        712 F...m nscd ☚
                     root        784 ....m cups-browsed
                     whoopsie    903 ....m whoopsie
                     root        919 ....m apache2
                     ...           ...           ...           ...
                     root       1103 ....m sshd
                     postgres   1144 ....m postgres
                     ...           ...           ...           ...
                     postgres   1149 ....m postgres
                     root       1280 ....m master
```

```
                    fitz       3129 ....m sshd
                    root       3170 ....m sudo
fitz@ubuntu:~$ sudo pmap -p 712
712:   /usr/sbin/nscd
                    ...         ...          ...          ...
❶ 00007fe044f94000 32768K rw-s- ➥ /var/cache/nscd/hosts
                    ...         ...          ...          ...

fitz@ubuntu:~$ sudo pmap -p 919
919:   /usr/sbin/apache2 -k start
                    ...         ...          ...          ...
❷ 00007fb0e2f36000   212K r--s- ➥ /var/cache/nscd/hosts
                    ...         ...          ...          ...
```

Организация межпроцессного взаимодействия при помощи разделяемой памяти на основе «штатного» механизма отображения файлов (в общем случае размещаемых на внешней, *дисковой* памяти) в память процессов имеет один значительный недостаток. Так как изменяемые страничные кадры такого общего сегмента памяти требуют сохранения в *дисковый* файл (что, к счастью, выполняется ядром не немедленно при их изменении), то в общем смысле на производительность такой разделяемой памяти влияют задержки операций *дискового* ввода-вывода (!), а не только скорость работы оперативной памяти.

В примере с демоном **nscd(8)** этого вполне достаточно, тем более что создаваемый им кэш все равно должен сохраняться при перезагрузках. В случаях, когда взаимодействие процессов требует максимальной производительности, разделяемая память на основе отображения *дисковых* файлов в память является не лучшим механизмом. Элегантное решение данной проблемы используется в Linux-реализации разделяемой памяти стандарта **POSIX**[1], что проиллюстрировано в листинге 4.58. Псевдофайловая система **tmpfs** специально придумана для «временного» размещения файлов непосредственно в оперативной памяти, что в совокупности со «штатным» механизмом их отображения в память взаимодействующих процессов и дает желаемые характеристики производительности.

Листинг 4.58. Разделяемая память (POSIX)

```
fitz@ubuntu:~$ findmnt /dev/shm
TARGET   SOURCE FSTYPE OPTIONS
/dev/shm        tmpfs  rw,nosuid,nodev
fitz@ubuntu:~$ sudo fuser -v /dev/shm/*
                     ПОЛЬЗ-ЛЬ    PID ДОСТУП КОМАНДА
```

[1] См. **shm_overview(7)**, а также **POSIX**-семафоры **sem_overview(7)** и **POSIX**-очереди сообщений **mq_overview(7)**.

```
/dev/shm/PostgreSQL.1587464325:
                    postgres   1023 ....m postgres
                    postgres   1144 ....m postgres
                    postgres   1145 ....m postgres
                    postgres   1146 ....m postgres
                    postgres   1147 ....m postgres
                    postgres   1149 ....m postgres
fitz@ubuntu:~$ ps p 1023,1144,1145,1146,1147,1149
  PID TTY      STAT   TIME COMMAND
 1023 ?        S      0:00 /usr/lib/postgresql/11/bin/postgres ...
 1144 ?        Ss     0:00 postgres: 11/main: checkpointer
 1145 ?        Ss     0:01 postgres: 11/main: background writer
 1146 ?        Ss     0:01 postgres: 11/main: walwriter
 1147 ?        Ss     0:00 postgres: 11/main: autovacuum launcher
 1149 ?        Ss     0:00 postgres: 11/main: logical replication launcher
fitz@ubuntu:~$ sudo pmap -p 1023
1023:   /usr/lib/postgresql/11/bin/postgres -D /var/lib/postgresql/11/main ...
00007feafa398000      8K rw-s- /dev/shm/PostgreSQL.1587464325
                      ...      ...          ...          ...
fitz@ubuntu:~$ sudo pmap -p 1145
1145:   postgres: 11/main: background writer
00007feafa398000      8K rw-s- /dev/shm/PostgreSQL.1587464325
                      ...      ...          ...          ...
```

В примере из листинга 4.58 показано, как разделяемую память **POSIX** использует SQL-сервер **postgres(1)** для взаимодействия между своими параллельными процессами.

Семафоры и очереди сообщений

Разделяемая память требует синхронизации действий процессов из-за эффекта гонки (race), возникающего между конкурентными, выполняющимися параллельно процессами. Для синхронизации процессов при совместном доступе к разделяемой памяти и прочим разделяемым ресурсам предназначено еще одно специализированное средство их взаимодействия — *семафоры*. В большинстве случаев семафорами **System V** или **POSIX** пользуются многопроцессные сервисы, использующие разделяемую память, такие как, например, **postgres(1)**, проиллюстрированный выше.

Очереди сообщений являются средством взаимодействия между процессами, реализующим еще один интерфейс передачи сообщений (message passing inerface), подобно каналам и сокетам. По своей природе они похожи на дейтаграммный **SOCK_DGRAM** режим передачи поверх именованных локальных сокетов **unix(7)**. Основное отличие очередей сообщений от сокетов заключается в том, что время их жизни не ограничивается временем существования процессов, которые их создали.

На практике семафоры и очереди сообщений являются настолько малораспространенными, что их иллюстрация (листинги 4.59 и 4.60) на среднестатистической инсталляции Linux практически невозможна.

Листинг 4.59. Семафоры и очереди сообщений (System V IPC)

```
fitz@ubuntu:~$ ipcs -q
------ Очереди сообщений --------
ключ   msqid      владелец права исп. байты сообщения
fitz@ubuntu:~$ ipcs -s
------ Массивы семафоров --------
ключ   semid      владелец права nsems
```

Листинг 4.60. Семафоры и очереди сообщений (POSIX)

```
fitz@ubuntu:~$ findmnt /dev/mqueue
TARGET      SOURCE FSTYPE OPTIONS
/dev/mqueue mqueue mqueue rw,nosuid,nodev,noexec,relatime

fitz@ubuntu:~$ ls -l /dev/mqueue
☹ итого 0

fitz@ubuntu:~$ ls -l /dev/shm/sem.*
☹ ls: невозможно получить доступ к '/dev/shm/sem.*': Нет такого файла или каталога
```

4.10. В заключение

Выполняющиеся программы являются основными активными сущностями, инструкции которых при помощи механизма системных вызовов потребляют ресурсы, находящиеся под управлением операционной системы. Распределением этих ресурсов и занимаются подсистемы управления процессами, управления памятью и ввода-вывода, в достаточно детальной мере рассмотренные в этой главе. Основной задачей этих подсистем является организация *эффективного* распределения ресурсов между массой их потребителей — процессами и нитями.

Фактическая эффективность их работы при прочих равных будет во многом зависеть от понимания пользователем их внутренних алгоритмов и значений конфигурационных параметров этих алгоритмов, в зависимости от характеристик самих потребителей и желаемых результатов. Например, эффективность распределения процессорного времени будет напрямую зависеть от свойств процессов и нитей, их приоритетов, их классов и процессорных привязок.

Кроме того, понимание алгоритмов работы подсистем может ответить на многие вопросы о количестве потребляемых ресурсов и дать ответ об их достаточности или недостатке. Например, важно понимать, что недостаток ресурса оперативной памяти вовсе не определяется суммарными размерами виртуальной памяти, потребленной процессами, а напрямую связан с суммарными размерами их резидентной памяти.

Навыки мониторинга и трассировки потребления ресурсов процессами помогут сделать массу полезнейших выводов о свойствах выполняющихся в них программ, что чрезвычайно полезно при разработке качественного программного обеспечения. Нелишними эти навыки будут и при выборе качественного программного обеспечения для эксплуатации в заданных условиях и с требуемыми характеристиками.

Глава 5

Программирование на языке командного интерпретатора

Командный интерпретатор является основой интерфейса командной строки, первой и главной программой, запускающейся в *интерактивном* сеансе пользователя. Кроме этого, он широко используется и в *пакетном* режиме работы, когда команды записываются в файл *сценария* «пьесы» и «проигрываются по ролям» при его запуске. В этом случае сценарий является простейшей интерпретируемой программой на языке соответствующего командного интерпретатора.

5.1. Интерпретаторы и их сценарии

В настоящее время существует достаточное количество диалектов языка командного интерпретатора: POSIX-совместимые **ash(1)** и **dash(1)**, авторские диалекты W:[Korn shell] **ksh(1)** и W:[Bourne shell] **bash(1)**, диалекты с синтаксисом, подобным языку программирования Си, — **csh(1)** и **tcsh(1)** и прочие[1].

Кроме языка командного интерпретатора, языки W:[Perl], W:[Python] или W:[Tcl] так же имеют свои интерпретаторы и практически всегда применяются в пакетном режиме обработки своих сценариев.

Для запуска нужного интерпретатора используют универсальный комментарий W:[shebang], записываемый в первую строчку сценария (листинг 5.1) и указывающий полный путь к программе интерпретатора, которая вызывается для интерпретации запускаемого сценария.

Листинг 5.1. Интерпретаторы и sha-bang

```
bender@ubuntu:$ file /bin/which
/bin/which: POSIX shell script, ASCII text executable
```

[1] Ультрасовременные **zsh(1)** или **fish(1)** хороши для интерактивной работы в системе, но для пакетной обработки команд не имеют особенного смысла.

```
bender@ubuntu:$ head -1 /bin/which
#! /bin/sh
bender@ubuntu:~$ file /bin/gunzip
/bin/gunzip: Bourne-Again shell script, ASCII text executable
bender@ubuntu:~$ head -1 /bin/gunzip
#!/bin/bash
bender@ubuntu:~$ file /usr/sbin/iotop
/usr/sbin/iotop: Python script, ASCII text executable
bender@ubuntu:~$ head -1 /usr/sbin/iotop
#!/usr/bin/python3
bender@ubuntu:~$ file /usr/bin/lsdev
/usr/bin/lsdev: Perl script text executable
bender@ubuntu:~$ head -1 /usr/bin/lsdev
#!/usr/bin/perl
bender@ubuntu:~$ file /usr/bin/netwag
/usr/bin/netwag: a /usr/bin/wish script, ASCII text executable, with very long lines,
with CRLF, LF line terminators, with overstriking
bender@ubuntu:~$ head -1 /usr/bin/netwag
#!/usr/bin/wish
```

Сами сценарии представляют собой обычные текстовые файлы, подготавливаемые в любом текстовом редакторе, однако размещаются в каталогах и имеют права (см. ❶ и ❷, листинг 5.2) подобно «обычным» исполняемым W:[ELF]-программам.

Листинг 5.2. Сценарии интерпретаторов

Сеанс работы tty1

```
bender@ubuntu:~$ cat hello.sh
#!/bin/sh

echo "Hello, World!"
bender@ubuntu:~$ hello.sh
hello.sh: команда не найдена
bender@ubuntu:~$ printenv PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:...:/snap/bin
bender@ubuntu:~$ pwd
/home/bender
bender@ubuntu:~$ mkdir /home/bender/bin
bender@ubuntu:~$ logout
```

```
Новый сеанс                                                    tty1
bender@ubuntu:~$ printenv PATH
! /home/bender/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:...:/snap/bin
❶ bender@ubuntu:~$ mv hello.sh /home/bender/bin
  bender@ubuntu:~$ hello.sh
  bash: /home/bender/bin/hello.sh: Отказано в доступе
  bender@ubuntu:~$ ls -la bin/hello.sh
  -rw-rw-r-- 1 bender bender 32 янв.  17 15:23 bin/hello.sh
❷ bender@ubuntu:~$ chmod a+x bin/hello.sh
  bender@ubuntu:~$ ls -la bin/hello.sh
  -rwxrwxr-x 1 bender bender 32 янв.  17 15:23 bin/hello.sh
  bender@ubuntu:~$ hello.sh
  Hello, World!
```

5.2. Встроенные и внешние команды

Основное назначение любого командного интерпретатора в интерактивном или пакетном режиме — запускать команды, которые приводят или к запуску программы, «внешней» по отношению к самому интерпретатору, или к выполнению каких-либо «встроенных» действий самим командным интерпретатором (листинг 5.3). Например, команда **cd**, изменяющая текущий каталог, является встроенной (и по-другому реализована быть не может, потому что должна изменить атрибут **CWD** процесса самого интерпретатора). Команда **pwd**, наоборот, может быть внешней (и показывать при запуске атрибут **CWD** своего процесса, унаследованного в момент запуска от командного интерпретатора), но для интерпретаторов Bourne/Korn shell зачастую имеет и встроенную реализацию.

Листинг 5.3. Встроенные и внешние команды

```
? bender@ubuntu:~$ which -a cd
  bender@ubuntu:~$ type -a cd
  cd – это встроенная команда bash
  bender@ubuntu:~$ which -a pwd
  /usr/bin/pwd
  /bin/pwd
  bender@ubuntu:~$ type -a pwd
  pwd – это встроенная команда bash
  pwd является /usr/bin/pwd
  pwd является /bin/pwd
```

5.3. Перенаправление потоков ввода-вывода

Для программирования сценариев на языке командного интерпретатора одной из важнейших его способностей является возможность организации *сохранения результатов* в файлы и возможность *считывания исходных* данных из файлов при выполнении команд.

Командный интерпретатор организует перенаправления потоков ввода-вывода внешних и встроенных команд при помощи конструкций **[n]>file** или **[n]>>file** и **[n]<file** (рис. 5.1). Символы **<** и **>** естественным образом идентифицируют направление выполняемых перенаправлений — ввода или вывода, а необязательное число **n** уточняет номер перенаправляемого потока (при умалчивании **n** перенаправляется стандартный поток вывода № 1, **stdout(3)** при выводе и стандартный поток ввода № 0, **stdin(3)** при вводе). Перенаправление вывода **>** выполняется с *усечением* старого содержимого, а перенаправление **>>** — с *добавлением* к старому содержимому файла **file**.

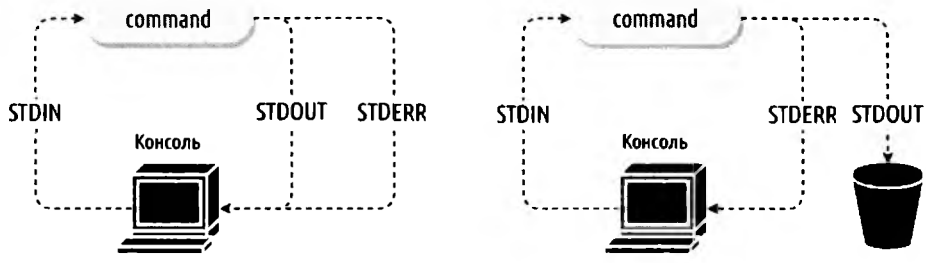


Рис. 5.1. Стандартные потоки ввода-вывода и перенаправление потока вывода STDOUT

Для перенаправления заданного потока **N** некоторой запускаемой команды **command** в файл **file** командный интерпретатор порождает дочерний процесс при помощи системного вызова **fork(2)**, затем в новом процессе открывает файл **file** при помощи системного вызова **open(2)**, перенаправляет поток **N** в открытый файл посредством системного вызова **dup2(2)** и, наконец, запускает в этом процессе программу **command** при помощи системного вызова **execve(2)**.

В примере из листинга 5.4 при помощи «разрезателя» текста **cut(1)** вырезается по разделителю (**-d**) **:** (двоеточие) **1**-е поле (**-f**) файла **passwd(5)**, содержащего свойства учетных записей пользователей, зарегистрированные в системе, а *результат сохраняется* в файле **users** при помощи перенаправления потока STDOUT. Затем при помощи потокового редактора файлов **sed(1)** выводятся (**5,8p**) только (**-n**) с **5**-й по **8**-ю строки полученного файла, что дает логины с 5-го по 8-й пользователя.

Листинг 5.4. Перенаправление стандартного потока вывода

```
bender@ubuntu:~$ sed -n 5,8p /etc/passwd
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
bender@ubuntu:~$ cut -f 1 -d : /etc/passwd 1> users
bender@ubuntu:~$ sed -n 5,8p users
sync
games
man
lp
```

При выполнении некоторых команд, например как в листинге 5.5, при поиске файлов посредством **find(1)** может выводиться достаточное количество сообщений об «ошибках» доступа в тот или иной каталог, которые мешают анализировать результаты поиска. В этом случае их удобно убрать с терминала путем перенаправления потока № 2, **stderr(3)**, например в файл **eaccess**. В результате на терминале будут отражены только имена (**-name**) файлов, соответствующие критерию имени ***.xml**, найденные в каталоге **/etc** и всех его подкаталогах.

Листинг 5.5. Перенаправление стандартного потока ошибок

```
bender@ubuntu:~$ find /etc -name '*.xml'
/etc/cupshelpers/preferreddrivers.xml
                ...          ...          ...          ...
? find: '/etc/cups/ssl': Отказано в доступе
find: '/etc/polkit-1/localauthority': Отказано в доступе
find: '/etc/ssl/private': Отказано в доступе
                ...          ...          ...          ...
/etc/thermald/thermal-cpu-cdev-order.xml
bender@ubuntu:~$ find /etc -name '*.xml' 2>eaccess
/etc/cupshelpers/preferreddrivers.xml
                ...          ...          ...          ...
/etc/ImageMagick-6/log.xml
/etc/thermald/thermal-cpu-cdev-order.xml
```

На практике часто оказывается, что мешающий вывод сообщений об «ошибках» доступа, как в листинге 5.6, оказывается ненужным вовсе, тогда его *подавляют* при помощи перенаправления потока STDERR в файл **/dev/null** специального всепожирающего псевдоустройства **null(4)**. Подсчитав количество строк (**-l**) в результирующем файле **empty** и зная, что одна строка в нем соответствует одному найден-

ному ранее командой **find(1)** имени файла, размер (**-size**) которого равен 0 байт, можно получить количество «пустых» файлов в системе.

Листинг 5.6. Подавление стандартного потока ошибок

```
bender@ubuntu:~$ find / -size 0 1>empty 2>/dev/null
bender@ubuntu:~$ wc -l empty
97602 empty
```

В случаях, когда необходимо использовать содержимое файла в качестве *исходных данных* для выполнения программ, вместо ввода этих данных с терминала, как в примере из листинга 5.7, применяют перенаправление потока **STDIN** (рис. 5.2). Таким образом, например, сохраненный в файл весьма внушительный список установленных в системе пакетов (полученный командой **dpkg(1)**) можно отправить по почте командой **mail(1)** в качестве текста сообщения.

Листинг 5.7. Перенаправление стандартного потока ввода

```
bender@ubuntu:~$ dpkg -l > installed-pacakges
bender@ubuntu:~$ mail dketov@gmail.com 0< installed-pacakges
bender@ubuntu:~$ rm -f installed-pacakges
```

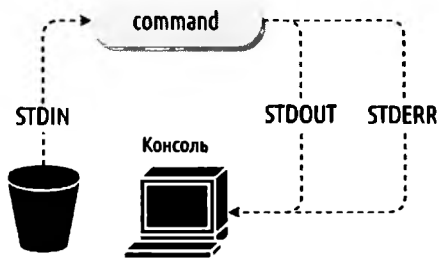


Рис. 5.2. Перенаправление потока ввода STDIN

Однако чаще на практике необходимо использовать *результат выполнения* одной команды в качестве *исходных данных* другой команды, не сохраняя эти данные в промежуточный (ненужный) файл. В этом случае используют конструкцию **command$_1$ | command$_2$ | ...**, называемую *конвейерной*[1] обработкой (рис. 5.3). Нужно заметить, конструкцией конвейерной обработки можно сцеплять более чем две команды, причем все они будут выполняться параллельно в дочерних процессах командного интерпретатора. При этом стандартные потоки ввода и вывода пар процессов будут связаны простейшим средством IPC, реализуемым ядром ОС — неименован-

[1] Или просто *конвейером*, или *каналом* (pipe), или даже *трубой*.

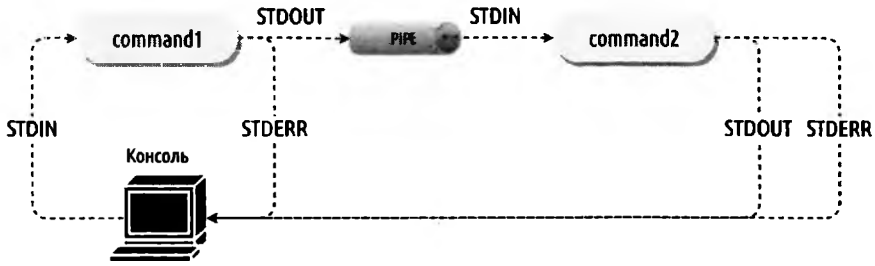


Рис. 5.3. Конвейерная обработка

ным каналом, создаваемым системным вызовом **pipe(2)**, куда и будут перенаправлены потоки команд при помощи системного вызова **dup2(2)**.

При использовании конвейерной обработки в примере из листинга 5.8 отправка списка пакетов, установленных в системе почтовым сообщением, происходит без промежуточного файла, ровно как и подсчет количества файлов, размер которых равен 0 байт (листинг 5.9).

Листинг 5.8. Конвейерная обработка: отправка списка установленных пакетов по электронной почте

```
bender@ubuntu:~$ dpkg -l | mail dketov@gmail.com
```

Листинг 5.9. Конвейерная обработка: подсчет количества пустых файлов

```
bender@ubuntu:~$ find / -size 0 2>/dev/null | wc -l
98940
```

Листинг 5.10. Конвейерная обработка: вывод части файла

```
bender@ubuntu:~$ getent passwd | cut -f 1 -d : | head -8 | tail -5
sys
sync
games
man
lp
```

В примере из листинга 5.10 конвейерная обработка используется для последовательного получения строкового представления свойств всех доступных[1] пользова-

[1] Учетные записи пользователей, зарегистрированных непосредственно в системе (локально), хранятся в файле **/etc/passwd**, см. **passwd(5)**. Все доступные в системе учетные записи, в том числе, например, централизованные в (сетевых) LDAP-каталогах организации, могут быть получены при помощи команды **getent(1)** благодаря службе имен.

тельских учетных записей при помощи **getent(1)**, затем вырезания из этих строк при помощи **cut(1)** 1-го поля (**-f**) посредством разделителя : (**-d**), далее выбора 8 первых строк командой **head(1)**, а затем выбора 5 последних строк предыдущего результата командой **tail(1)**.

Листинг 5.11. Конвейерная обработка: генерация трех случайных восьмизначных паролей

```
bender@ubuntu:~$ tr -dc a-zA-Z0-9 </dev/urandom | fold -w 8 | head -3
Ta3Su0vJ
1vK7H7Fw
Vs7FK8md
```

В листинге 5.11 конвейерная обработка в совокупности с перенаправлением потока **STDIN** применяется для генерации трех случайных строк, которые могут использоваться, например, в качестве начальных паролей пользовательских учетных записей. Для этого из специального файла псевдоустройства **urandom(4)**, генерирующего случайную последовательность байтов, отбираются командой транслитерации **tr(1)** только строчные и прописные (заглавные) буквы латинского алфавита и цифры **a-zA-Z0-9** путем удаления (**-d**) из выходного потока символов, не (**-c**) попавших в заданный набор. Затем команда **fold(1)** разбивает свой входной поток по 8 символов в строке выходного потока, после чего команда **head(1)** выбирает только 3 первые строки.

Листинг 5.12. Конвейерная обработка: удаление пустых файлов

```
bender@ubuntu:~$ find /tmp -user bender -size 0 -print0 2>/dev/null | xargs -0 rm -f
```

В листинге 5.12 конвейерная обработка вместе с командой **xargs(1)** используется для того, чтобы во временном каталоге **/tmp** удалить все файлы, принадлежащие (**-user**) пользователю **bender**, размер (**-size**) которых равен нулю. Сначала при помощи команды **find(1)** производится поиск файлов согласно указанным критериям и вывод списка их имен с разделением имен нулевым символом (**-print0**). Затем команда **xargs(1)** последовательно «применяет» команду **rm(1)** безусловного (**-f**) удаления файла для каждого имени из списка. Следует отметить, что такая конструкция работает с абсолютно любыми именами файлов, включая файлы с пробелами, табуляциями, переводами строк и другими управляющими символами в имени, т. к. имена файлов в списке разделяются нулевым символом, а он единственный запрещен к использованию в именах файлов.

Аналогично, в листинге 5.13, используя конвейер, при помощи команды **groups(1)** можно вывести групповое членство всех пользователей, доступных в системе.

Листинг 5.13. Конвейерная обработка: просмотр группового членства пользователей системы

```
bender@ubuntu:~$ getent passwd | cut -f 1 -d : | xargs groups
                    ...          ...           ...           ...
finn : finn candy
jake : jake

bubblegum : bubblegum candy
                    ...          ...           ...           ...
fitz : fitz sudo
skillet : skillet
bender : bender
                    ...          ...           ...           ...
```

С помощью конвейеров и универсального спискового «применятеля» команд **xargs(1)** можно организовать (листинг 5.14) параллельный запуск (**-P**) упаковки целого списка отобранных командой **find(1)** файлов, использовав параллельный упаковщик **pbzip2(1)**, в несколько упаковочных нитей (**-p**) на каждый файл. В результате получим запускаемые парами процессы по две активные нити на каждый, что эффективно загружает четырехъядерный процессор в течение упаковки всего списка файлов.

Листинг 5.14. Конвейерная обработка: параллельная упаковка ISO-образов

```
bender@ubuntu:~$ find . -name '*.iso' | xargs -P 2 pbzip2 -p2 &
bender@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
 4723 pts/1    S      0:00 -bash
 4903 pts/1    S      0:00  \_ xargs -P 2 -n1 pbzip2 -p2
 4904 pts/1    Sl     0:08  |   \_ pbzip2 -p2 ./dvd.iso
 4905 pts/1    Sl     0:08  |   \_ pbzip2 -p2 ./plan9.iso
 4856 pts/1    R+     0:00  \_ ps f
bender@ubuntu:~$ ps -fL
UID        PID  PPID  LWP  C NLWP STIME TTY          TIME CMD
bender    4723  4722  4723  0    1 01:18 pts/1    00:00:00 -bash
bender    4903  4723  4903  0    1 01:21 pts/1    00:00:00 xargs -P 2 -n1 pbzip2
                  ...           ...              ...           ...
bender    4904  4903  4910 99    6 01:21 pts/1    00:00:01 pbzip2 -p2 ./dvd.iso
bender    4904  4903  4913 99    6 01:21 pts/1    00:00:01 pbzip2 -p2 ./dvd.iso
bender    4904  4903  4914  0    6 01:21 pts/1    00:00:00 pbzip2 -p2 ./dvd.iso
                  ...           ...              ...           ...
bender    4905  4903  4911 99    6 01:21 pts/1    00:00:01 pbzip2 -p2 ./plan9.is
bender    4905  4903  4912 99    6 01:21 pts/1    00:00:01 pbzip2 -p2 ./plan9.is
bender    4905  4903  4915  0    6 01:21 pts/1    00:00:00 pbzip2 -p2 ./plan9.is
bender    4916  4723  4916  0    1 01:21 pts/1    00:00:00 ps -fL
```

5.4. Подстановки командного интерпретатора

Еще одной важной способностью командного интерпретатора, необходимой для разработки сценариев, является возможность подставлять различные вычисляемые значения в качестве аргументов запускаемых команд.

5.4.1. Подстановки имен файлов

Простейшими значениями, вычисляемыми командным интерпретатором, являются имена файлов, соответствующие некоторым шаблонам. Язык шаблонных выражений крайне прост и основывается на понятии *метасимволов*, т. е. символов со специальными значениями (табл. 5.1).

Таблица 5.1. Шаблонные метасимволы

Метасимвол	*Значение*
`?`	Один любой символ
`*`	Любое количество любых других символов
`[ab...z]`	Любой символ из набора a, b, ..., z
`[!ab...z]` или `[^ab...z]`	Любой символ *не* из набора a, b, ..., z
`~`	Домашний каталог пользователя

Использование метасимволов в команде интерпретатора заставляет его искать файлы, чьи имена соответствуют составленному шаблонному выражению, и *подставлять* их вместо самих шаблонных выражений.

В примере из листинга 5.15 шаблонное выражение `y*` используется для подстановки имен файлов, начинающихся с буквы `y`, за которой следует любое количество любых символов. Выражение `???.*` означает имена файлов, которые содержат три любых символа, потом символ `.` (точка), за которым следует любое количество любых символов. Сама работа подстановок интерпретатора может быть отслежена в режиме трассировки, который включается ❶ командой `set -x` (а выключается, как ни странно, командой `set +x`). В этом режиме интерпретатора видно, что выражение `[0-9]*`, означающее имена файлов, начинающиеся с цифры, сначала вычисляется ❶ до соответствующего списка имен, который подставляется вместо самого шаблонного выражения, и только потом выполняется команда `ls(1)`, в аргументах которой оно было использовано. Аналогично, вычисляется ❷ и подставляется шаблонное выражение `[!a-z0-9]*`, означающее имена файлов, начинающиеся

с символов, не являющихся ни буквой, ни цифрой. Таким образом, ни одна команда самостоятельно не вычисляет подстановки имен файлов, а пользуется результатами вычислений так, как будто они были заданы непосредственно в качестве ее аргументов.

Листинг 5.15. Просмотр файлов man-страниц по критерию имени

```
bender@ubuntu:~$ cd /usr/share/man/man1
bender@ubuntu:/usr/share/man/man1$ ls y*
ybmtopbm.1.gz  yes.1.gz         ypdomainname.1.gz   yuvtoppm.1.gz
yelp.1.gz      youtube-dl.1.gz  yuvsplittoppm.1.gz
bender@ubuntu:/usr/share/man/man1$ ls ???.*
'[.1.gz'       dsa.1ssl.gz          ico.1.gz   rcp.1.gz          tee.1.gz
 apg.1.gz      dwp.1.gz             lcf.1.gz   red.1.gz          tgz.1.gz
                ...                  ...        ...               ...
 dir.1.gz      gtf.1.gz             pwd.1.gz   tbl.1.gz          zip.1.gz
bender@ubuntu:/usr/share/man/man1$ set -x ❶
bender@ubuntu:/usr/share/man/man1$ ls -l [0-9]*
+ ls --color=auto ❷ -l 2to3-2.7.1.gz 411toppm.1.gz ❶
-rw-r--r-- 1 root root 563 окт 10 13:26 2to3-2.7.1.gz
-rw-r--r-- 1 root root 592 апр 23  2016 411toppm.1.gz
bender@ubuntu:/usr/share/man/man1$ ls [!a-z0-9]*
+ ls --color=auto '[.1.gz' CA.pl.1ssl.gz GET.1p.gz HEAD.1p.gz ImageMagick-im6.q16.1.gz
POST.1p.gz Xephyr.1.gz Xmark.1.gz Xorg.1.gz Xorg.wrap.1.gz Xserver.1.gz ❸
'[.1.gz'          HEAD.1p.gz                 Xephyr.1.gz   Xorg.wrap.1.gz
 CA.pl.1ssl.gz    ImageMagick-im6.q16.1.gz   Xmark.1.gz    Xserver.1.gz
 GET.1p.gz        POST.1p.gz                 Xorg.1.gz
```

В режиме трассировки командного интерпретатора видна еще и подстановка псевдонимов ❷, которая заменяет «псевдоним» команды пользователя, например **ls** (листинг 5.16), ее «настоящим значением».

Листинг 5.16. Псевдонимы командного интерпретатора

```
bender@ubuntu:~$ alias
                  ...          ...          ...          ...
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

5.4.2. Подстановки параметров

Более важным видом подстановок командного интерпретатора являются подстановки значений *параметров* — специальных сущностей, имеющих эти самые значения. Различают три типа параметров: *переменные*, *позиционные* параметры и *специальные* параметры. Значения переменных могут быть изменены в любой момент времени при помощи операции присваивания, тогда как значения позиционных параметров задаются один раз при их инициализации, а значения специальных параметров вычисляются предопределенным образом в зависимости от окружающей среды и обстоятельств.

Переменные — именованные параметры

Самый простой тип параметров — это переменные командного интерпретатора, и их «глобальное» подмножество — переменные окружения. *Переменные окружения (см. главу 2)* параметризируют пользовательский сеанс работы в системе и видны абсолютно всем командам сеанса, тогда как переменные командного *интерпретатора* видны только ему.

В примере из листинга 5.17 командами **env(1)** и **set** показано количество переменных окружения и переменных командного интерпретатора соответственно. При помощи операции присваивания **VARIABLE=value** вводится новая переменная интерпретатора ❶, а посредством команды **export** ❷ эта переменная «экспортируется» в переменные окружения, что видно во флагах объявления переменных на выводе команды **typeset**. Стоит отметить, что команда **set** интерпретатора **bash(1)** по умолчанию выводит не только объявленные переменные, но и объявленные функции, что отключается ❶ установкой POSIX-совместимого режима работы.

Листинг 5.17. Переменные интерпретатора и переменные окружения

```
❶ bender@ubuntu:~$ set -o posix
  bender@ubuntu:~$ env | wc -l; set | wc -l; typeset -p VARIABLE
  34
  80
  -bash: typeset: VARIABLE: не найден
❶ bender@ubuntu:~$ VARIABLE=value
  bender@ubuntu:~$ env | wc -l; set | wc -l; typeset -p VARIABLE
  34
  81
  declare -- VARIABLE="value"
❷ bender@ubuntu:~$ export VARIABLE
  bender@ubuntu:~$ env | wc -l; set | wc -l; typeset -p VARIABLE
```

```
35
81
declare -x VARIABLE="value"
bender@ubuntu:~$ set +o posix
```

Присваивание значения «новой» переменной, как показано в листинге 5.18, устанавливает именно переменную командного *интерпретатора*, которая не видна «глобально» в сеансе пользователя и требует экспортирования для работы в качестве переменной *окружения*.

Листинг 5.18. Присвоение значений переменным

```
bender@ubuntu:~$ LC_TIME=ko_KR.utf8
bender@ubuntu:~$ date
2019. 11. 23. (토) 13:09:29 MSK
bender@ubuntu:~$ TZ=Europe/Stockholm
bender@ubuntu:~$ date
2019. 11. 23. (토) 13:11:49 MSK
bender@ubuntu:~$ declare -p LANG TZ
declare -x LANG="ko_KR.utf8"
declare -- TZ="Europe/Stockholm"
bender@ubuntu:~$ export TZ
2019. 11. 23. (토) 11:12:39 CET
```

Так как имена и значения переменных располагаются в оперативной памяти процесса командного интерпретатора, то их *время жизни* и *область видимости* ограничиваются процессом их интерпретатора. Это означает, что переменные теряют свои значения при завершении интерпретатора, в котором были установлены, а переменные, установленные в одном интерпретаторе, не *видны* в других интерпретаторах. Только установленные переменные *окружения* экспортируются (копируются) процессам других команд при запуске, но любые их последующие изменения происходят отдельно и локально в каждом процессе.

Для разового экспорта переменных окружения предназначается утилита **env(1)**, хотя в большинстве диалектов языка командного интерпретатора используется просто оператор присвоения значения переменным перед запускаемой командой (листинг 5.19).

Листинг 5.19. Временное присвоение значений переменным окружения

```
bender@ubuntu:~$ env LC_TIME=be_BY.utf8 TZ=Europe/Minsk ls -l
итого 222976
```

```
-rw-r--r-- 1 bender bender 114650029 ☛ лис 23 10:36 dvd.iso.gz
-rw-r--r-- 1 bender bender 113666114    лис 23 00:02 plan9.iso.gz
bender@ubuntu:~$ LC_TIME=fi_FI.utf8 TZ=Europe/Helsinki ls -l
итого 222976
-rw-r--r-- 1 bender bender 114650029 ☛ marras 23 09:36 ☚dvd.iso.gz
-rw-r--r-- 1 bender bender 113666114    marras 22 23:02 plan9.iso.gz
bender@ubuntu:~$ typeset -p LC_TIME TZ
declare -x LC_TIME="ru_RU.UTF-8"
-bash: typeset: TZ: не найден
bender@ubuntu:~$ ls -l
итого 222976
-rw-r--r-- 1 bender bender 114650029 ☛ ноя 23 10:36 dvd.iso.gz
-rw-r--r-- 1 bender bender 113666114    ноя 23 00:02 plan9.iso.gz
```

Для подстановки значения параметров при выполнении команд используется конструкция **$parameter**, где символ **$** является требованием подстановки, а **parameter** — идентификатором параметра, например *именем* переменной, *номером* позиционного параметра или *символом* специального параметра.

В режиме трассировки команд интерпретатора из листинга 5.20 видно, что командный интерпретатор вместо указанной переменной подставляет ее значение до выполнения команды, после чего команда выполняется так, как будто ее аргументы были заданы непосредственно подставленными значениями.

Листинг 5.20. Подстановка значений переменных

```
bender@ubuntu:/tmp$ env
SSH_AGENT_PID=3260
SHELL=/bin/bash
USER=bender
                ...         ...         ...         ...
USERNAME=bender
MAIL=/var/mail/bender
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PWD=/tmp
LANG=ru_RU.UTF-8
                ...         ...         ...         ...
HOME=/home/bender
                ...         ...         ...         ...
LOGNAME=bender
bender@ubuntu:/tmp$ set -x
bender@ubuntu:/tmp$ file $SHELL
```

```
+ file /bin/bash
/bin/bash: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses
shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0xf199a4a89ac968c2e0e99f2410600b9d7e995187,
stripped
bender@ubuntu:/tmp$ ps p $SSH_AGENT_PID
+ ps p 3260
  PID TTY      STAT   TIME COMMAND
 3260 ?        Ss     0:00 /usr/bin/ssh-agent /usr/bin/im-launch env GNOME_SHELL
bender@ubuntu:/tmp$ cd $HOME
+ cd /home/bender
bender@ubuntu:~$
```

Позиционные параметры

Позиционные параметры используются сценариями командного интерпретатора для передачи их фактических параметров и идентифицируются номерами, нумерующими аргументы сценария, указанные при его запуске. Например, в листинге 5.21 показан достаточно простой сценарий **tgz(1)**, применяемый для создания **.tgz**-архивов. В тексте сценария используются подстановки **$1** и **$0**, символизирующие первый формальный аргумент ①, переданный в сценарий при его запуске, и нулевой формальный аргумент ⓪ — имя самого сценария. При последующем запуске сценария в режиме трассировки (явно указывая интерпретатор **sh** с опцией **-x**) ему передаются фактические аргументы ❶ и ❶, которые соответствующим образом подставляются в командах, выполняемых интерпретатором согласно тексту сценария.

Листинг 5.21. Подстановка значений позиционных параметров

```
bender@ubuntu:~$ which tgz
/usr/bin/tgz
bender@ubuntu:~$ file /usr/bin/tgz
/usr/bin/gunzip: POSIX shell script, ASCII text executable
bender@ubuntu:~$ less /usr/bin/gunzip
#!/bin/sh
                    ...          ...          ...          ...
Error ()
{       echo "Error: $0: ${@-}." >&2 ❶
        exit 1
}

if [ $# = 0 ]; then
                    ...          ...          ...          ...
else
        dest=$1 ❶
```

```
        shift
        src="${@-}"
fi
                    ...         ...         ...         ...
elif [ -f "$dest" ]; then
        Error "Destination \"$dest\" already exists as a file"
                        ❷↴         ❶↴
bender@ubuntu:~$ sh -x /usr/bin/tgz ~/bin.tgz ~/bin
                        $0↥        $1↥
                    ...         ...         ...         ...
+ dest=/home/bender/bin.tgz ❶
                    ...         ...         ...         ...
tar: Удаляется начальный `/' из имен объектов
/home/bender/bin/
Всего записано байт: 10240 (10KiB, 5,9MiB/s)
 99.0%
+ ls -l /home/bender/bin.tgz
-rw-rw-r-- 1 bender bender 118 ноя 23 13:52 /home/bender/bin.tgz
+ exit 0
                        ❷↴         ❶↴
bender@ubuntu:~$ sh -x /usr/bin/tgz ~/bin.tgz ~/bin
                        $0↥        $1↥
                    ...         ...         ...         ...
+ dest=/home/bender/bin.tgz ❶
                    ...         ...         ...         ...
+ [ -f /home/bender/bin.tgz ]
+ Error Destination "/home/bender/bin.tgz" already exists as a file
+ echo Error: /usr/bin/tgz ❷: Destination "/home/bender/bin.tgz" already exists as a file.
Error: /usr/bin/tgz: Destination "/home/bender/bin.tgz" already exists as a file.
+ exit 1
```

Специальные параметры

Специальные параметры идентифицируются символами **#**, **?**, **!**, **@**, **$** и вычисляются в зависимости от окружения и обстоятельств выполнения сценария в заранее предопределенные значения. Так, например, в листинге 5.22 показана подстановка параметра **$**, вычисляющегося в идентификатор процесса самого командного интерпретатора, а в листинге 5.23 — подстановка параметра **!**, вычисляющегося в идентификатор процесса последнего асинхронного (параллельного с самим идентификатором) дочернего процесса. В режиме трассировки команд видно, что, как и любые другие подстановки, они выполняются до запуска команд, в которых были использованы. В результате сами команды выполняются так, словно параметры их были заданы непосредственным образом.

Листинг 5.22. PID текущего процесса

```
bender@ubuntu:~$ set -x
bender@ubuntu:~$ ps up $$
+ ps up 32649
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
bender   32649  0.1  0.1  13536  8604 pts/1    Ss   10:28   0:00 -bash
```

Листинг 5.23. PID последнего дочернего асинхронного процесса

```
bender@ubuntu:~$ set -x
bender@ubuntu:~$ dd if=/dev/dvd of=dvd.iso &
[1] 399
+ dd if=/dev/dvd of=dvd.iso
bender@ubuntu:~$ ps up $!
+ ps up 399
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
bender     399 14.0  0.0   5604   588 pts/1    D    10:36   0:00 dd if=/dev/dvd ...
```

Используя специальный параметр **?**, можно узнать статус завершения (код возврата) программы, выполнившейся последней. При этом *нулевой* статус завершения символизирует успешное ее выполнение, а любое другое, *отличное от нулевого*, значение есть «номер» ошибки. В листинге 5.24 команда **which(1)** завершилась с успехом при поиске программы, запускающейся по внешней команде **ls**, и неудачей — при поиске программы, соответствующей встроенной команде **cd**.

Листинг 5.24. Статус завершения процесса/программы

```
bender@ubuntu:~$ which ls
/usr/bin/ls
bender@ubuntu:~$ echo $?
0
bender@ubuntu:~$ which cd
bender@ubuntu:~$ echo $?
1
```

5.4.3. Подстановки вывода команд

Еще одним видом подстановок, выполняемых командным интерпретатором, являются подстановки *вывода команд*. Конструкции вида **$(command)** (или ее более старая форма **`command`**) используются для подстановки результата вывода команды **command** на поток STDOUT в место ее использования.

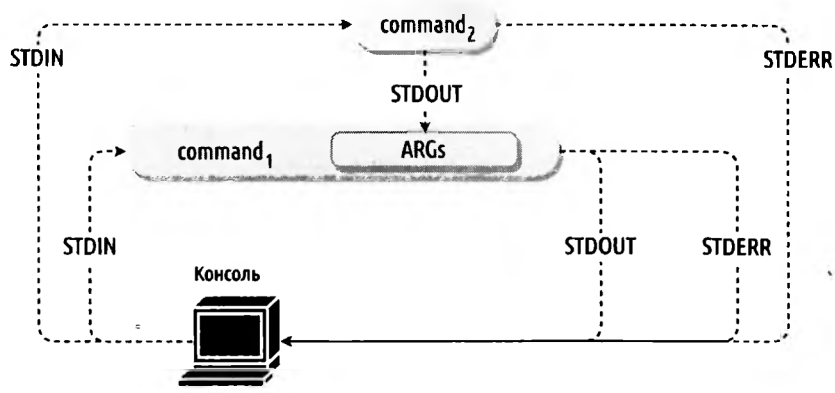


Рис. 5.4. Подстановка вывода команд

На рис. 5.4 показана широко распространенная подстановка *результата* работы команды **command$_2$** в качестве *аргументов*[1] команды **command$_1$**, используемая в виде **command$_1$ $(command$_2$)**. Так, например, можно, зная полный путь (который можно получить при помощи **which(1)**) к определенной утилите, выяснить при помощи **dpkg(1)** пакет программного обеспечения, которому он принадлежит, что показано в примере из листинга 5.25 в режиме трассировки.

Листинг 5.25. В каком пакете утилита?

```
bender@ubuntu:~$ which lspci
/usr/bin/lspci
bender@ubuntu:~$ set -x
❶ bender@ubuntu:~$ dpkg -S `which lspci`
++ which lspci
+ dpkg -S /usr/bin/lspci
pciutils: /usr/bin/lspci
❷ bender@ubuntu:~$ dpkg -S $(which lspci) | cut -f 1 -d : | xargs dpkg -s
+ xargs dpkg -s
+ cut -f 1 -d :
++ which lspci
+ dpkg -S /usr/bin/lspci
Package: pciutils
                    ...          ...          ...          ...
Description: PCI utilities
This package contains various utilities for inspecting and setting of
 devices connected to the PCI bus.
```

[1] Для сравнения: при конвейерной обработке *результаты* выполнения одной команды передаются в качестве *исходных* данных другой программе.

Стоит заметить, что команда **xargs(1)** выполняет подобную работу, что и подстановка вывода команд, т. е. передает в качестве *аргументов* одной команды результаты вывода другой. В примере ❷ из листинга 5.25 используются оба варианта, где подстановка вывода **which(1)** выполняется командным интерпретатором, а вывод первой команды **dpkg(1)** подставляется при помощи **xargs(1)**, будучи предварительно отфильтрованным при помощи **cut(1)**.

Подстановка вида **$(command)**, в отличие от **`command`**, может без особых ухищрений многократно вкладываться сама в себя. В примере из листинга 5.26 имя текущего пользователя ①, полученное при помощи команды **id(1)**, подставляется в качестве аргумента команды поиска **find(1)** для поиска его пустых файлов ②, список которых подставляется в качестве аргумента команды **rm(1)** для их удаления.

Листинг 5.26. Удаление пустых файлов пользователя

```
bender@ubuntu:~$ set -x
                                        ①↴
bender@ubuntu:~$ rm -f $(find /tmp -user $(id -un) -size 0)
                       ②↴
+++ id -un
❶ ++ find /tmp -user bender -size 0
find: `/tmp/.org.chromium.Chromium.N29MCl': Отказано в доступе
find: `/tmp/.wine-1000': Отказано в доступе
                    ...          ...          ...          ...
❷ + rm -f /tmp/wireshark_eth0_20151204134033_OMZ6wa /tmp/wireshark_eth0_20151204134033_...
```

Важно понимать, что подстановки заменяются их вычисляемыми значениями в любом месте команды, как, например, в листинге 5.27 — в части имени архива, получаемого при помощи команды **tar(1)** из файлов каталога **~/.config**.

Листинг 5.27. Архив с датой создания в имени

```
bender@ubuntu:~$ set -x
bender@ubuntu:~$ tar cjf dotconfig-$(date +%F).tbz2 ~/.config
+ tar cjf dotconfig-2015-12-13.tbz2 /home/bender/.config
tar: Удаляется начальный `/' из имен объектов
                    ...          ...          ...          ...
```

5.4.4. Подстановки арифметических выражений

Подстановки вывода команд и параметров являются для программирования на языке командного интерпретатора практически самыми важными конструкциями. Например, используя операцию присвоения, подстановку вывода и внешнюю

команду **expr(1)**, предназначенную для вычисления *арифметических* выражений, можно вычислять значения одних переменных на основе других, как показано в примере ❶ из листинга 5.28. Нужно отметить, что аргументы команды **expr(1)** отделяются пробелами, как и у любой другой команды, например **find(1)**, а специальные символы, например *, подлежат экранированию *(см. разд. 5.5)*.

Запуск внешних команд влечет за собой накладные расходы в виде системных вывозов **fork(2)** и **execve(2)**, поэтому во многих диалектах языка командного интерпретатора реализованы аналогичная встроенная команда *арифметических* вычислений **let** и *арифметическая* подстановка в виде стандартной POSIX-конструкции **$((expression))**, а в некоторых еще и в нестандартном виде **$[expression]**. Использование команды **let** в примере ❷ из листинга 5.28, наоборот, исключает пробелы в арифметическом выражении, т. к. оно целиком является одним ее аргументом. В случае использования арифметической подстановки в примере ❸ пробельные символы могут быть расставлены произвольным образом.

Листинг 5.28. Арифметические действия командного интерпретатора

```
   bender@ubuntu:~$ set -x
   bender@ubuntu:~$ RADIUS=10
❶  bender@ubuntu:~$ CIRCLE=`expr 2 \* $RADIUS \* 355 / 113`
   ++ expr 2 '*' 10 '*' 355 / 113
   + CIRCLE=62
❷  bender@ubuntu:~$ let CIRCLE=2*RADIUS*355/113; echo $CIRCLE
   + let 'CIRCLE=2*RADIUS*355/113'
   + echo 62
   62
❸  bender@ubuntu:~$ CIRCLE=$((2 * RADIUS * 355/113))
   + CIRCLE=62
```

Подстановки арифметических выражений полезно применять для пересчета значений в аргументах разных команд, использующих «неудобные» единицы измерения. Так, например, в листинге 5.29 при поиске библиотек больше 10 Мбайт нужно задавать количество байтов (символов), которое и составит искомые десять килобайт килобайтов.

Листинг 5.29. Библиотеки больше 10 Мбайт

```
   bender@ubuntu:~$ set -x
   bender@ubuntu:~$ find /lib /usr/lib -name '*.so.*' -size +$((1024*1024*10))c
☛  + find /lib /usr/lib -name '*.so.*' -size +10485760c
   /usr/lib/x86_64-linux-gnu/libicudata.so.63.2
                    ...          ...          ...          ...
   /usr/lib/x86_64-linux-gnu/libQt5WebKit.so.5.212.0
```

Пример из листинга 5.30 содержит две вложенные друг в друга подстановки — подстановку вывода команды **date(1)** в арифметическую подстановку. Команда **find(1)** при поиске модифицированных (**-mtime**, modification time) файлов использует единицы измерения «дней назад», и при поиске файлов в каталоге **/var/lib/dpkg/info**[1], изменившихся в период с 19.11.2019 по 22.11.2019, приходится рассчитывать количество «дней назад» до этих дат. Для расчета используются команды **date(1)** и **W:[UNIX-время]**, вычисляющая как целое количество секунд, прошедших от 01.01.1970 00:00 UTC до нужного момента, и арифметическая подстановка, вычисляющая количество «дней назад». Сначала вычисляется количество «секунд назад» как разница между текущим UNIX-временем и UNIX-временем указанной даты, после чего вычисляется количество «дней назад» путем деления на 60 секунд в минуте, 60 минут в часе и 24 часа в сутках. При помощи команды **stat(1)** подтверждается правильность поиска путем анализа даты модификации найденных файлов.

Листинг 5.30. Пакеты программного обеспечения, установленные в определенный период

```
bender@ubuntu:~$ set -x
bender@ubuntu:~$ find /var/lib/dpkg/info -name '*.list' \ ↵
> -mtime +$[ ([redacted] - [redacted]) / (24*60*60) ] \ ↵
> -mtime -$[ ([redacted] - [redacted]) / (24*60*60) ] | ↵
+ xargs stat -c %y:%n
++ date +%s
++ date -d 2019-11-22 +%s
++ date +%s
++ date -d 2019-11-19 +%s
+ find /var/lib/dpkg/info -name '*.list' -mtime +1 -mtime -4
2019-11-19 19:10:05.219764913 +0300:/var/lib/dpkg/info/tshark.list
2019-11-20 19:59:56.743180406 +0300:/var/lib/dpkg/info/libjq1:amd64.list
2019-11-20 00:54:46.292860010 +0300:/var/lib/dpkg/info/nscd.list
                    ...          ...          ...          ...
```

В этом примере строка первой *команды* конвейера для удобства ввода разбивается на три *экранные строки* посредством *экранирования* (см. ниже) управляющего символа перевода строки ↵ при помощи метасимвола одиночного экранирования ****. В этом случае командный интерпретатор каждую последующую экранную строку

[1] Это база данных пакетного менеджера **dpkg(1)** об установленных в систему пакетах программного обеспечения.

предваряет «вторичным» приглашением **PS2**, а не «первичным» приглашением **PS1**. Символ перевода строки после символа конвейера **|** экранирования не требует, т. к. командному интерпретатору очевидна необходимость ввода второй команды конвейера.

5.5. Экранирование

Перед запуском введенной команды интерпретатор анализирует ее на наличие так называемых *метасимволов*, т. е. символов, имеющих *специальное* значение. Так, например, *пробел* отделяет аргументы команды друг от друга, *шаблонные* выражения на основе **?**, *****, и **[]** вычисляются в имена файлов, *доллар* **$** активирует подстановку команд, параметров или арифметических выражений, а символы **<**, **>**, **|** — перенаправления потоков. В определенных командах требуется использовать *литеральное* (буквальное) значение метасимволов, как, например, в листинге 5.31 для манипулирования файлами, в именах которых содержатся пробелы. Для отмены специального значения метасимволов используется их *экранирование* при помощи конструкций ****m, **'**$m_1m_2m_3..m_N$**'** и **"**$m_1m_2m_3..m_N$**"**, «отменяющих» метасимволы m, m_1, ..., m_N.

Листинг 5.31. Экранирование пробельных символов

```
bender@ubuntu:~$ ls -l
итого 454292
                ...          ...          ...          ...
drwxr-xr-x  2 bender bender     4096 ноя 16 15:08 'Рабочий стол'
                ...          ...          ...          ...
❶ bender@ubuntu:~$ file Рабочий стол
? Рабочий: cannot open `Рабочий' (No such file or directory)
? стол:    cannot open `стол' (No such file or directory)
❷ bender@ubuntu:~$ file Рабочий\ стол
Рабочий стол: directory
❸ bender@ubuntu:~ $ file "Рабочий стол"
Рабочий стол: directory
❹ bender@ubuntu:~$ file 'Рабочий стол'
Рабочий стол: directory
```

В примере из листинга 5.31 показано экранирование символа пробела в имени файла, который привел ❶ к разбиению имени файла на два аргумента команды **file(1)**, ни один из которых, естественно, не является именем какого-либо файла. В примере ❷ используется одиночное экранирование пробела при помощи ****, а в примерах ❸ и ❹ — множественное экранирование всех символов в кавычках **''** и **""**, включая пробел.

В менее тривиальных случаях, как в примере ❶ из листинга 5.32, попытка выполнения команды **find(1)** с вполне валидными аргументами приводит к неожиданным и странным результатам. В режиме трассировки команд интерпретатора становится очевидно, что шаблонное выражение ***.gz**, предназначавшееся *самой* команде поиска файлов, было подставлено интерпретатором *до* запуска команды, что привело ее в недоумение. Для правильной передачи шаблонных выражений самим командам их следует экранировать, как в примере ❷.

Листинг 5.32. Экранирование шаблонных выражений

```
❶ bender@ubuntu:~$ find . -name *.gz
find: paths must precede expression: `plan9.iso.gz'
find: possible unquoted pattern after predicate `-name'?
bender@ubuntu:~$ set -x
bender@ubuntu:~$ find . -name *.gz ③
+ find . -name dvd.iso.gz plan9.iso.gz
    ①↴   ②↴            ?↴
find: paths ① must precede expression ②: `plan9.iso.gz'
find: possible unquoted ③ pattern after predicate `-name'?
❷ bender@ubuntu:~$ find . -name "*.gz"
+ find . -name '*.gz'
./dvd.iso.gz
./plan9.iso.gz
```

Аналогично, в примере ❶ из листинга 5.33 при попытке скачивания с Web-сервера ресурса со сложным **W**:[URL] при помощи команды **wget(1)** командный интерпретатор постарался разбить команду на список заданий, встретив метасимвол **&**, формирующий асинхронные задания *(см. разд. 4.8.1)*. Для правильной передачи строк, содержащих метасимволы, их следует экранировать, как в примере ❷.

Листинг 5.33. Экранирование символов списка команд &

```
bender@ubuntu:~$ youtube-dl -g https://www.youtube.com/watch?v=n1F_MfLRlX0
https://r4---sn-n3toxu-axqe.googlevideo.com/videoplayback?expire=1574532639&ei=vyHZXfo3kp3JBfWbqzg&ip=93.100.207.82&id=o-ABldbr46QqdVsKX_53HJb4sbFnPU60Ki_oStnKOvfARv&itag=136&aitags=133%2C134%2C135%2C136%2C160%2C242%2C243%2C244%2C247%2C278&source=youtube&requiressl=yes&mm=31%2C29&mn=sn-n3toxu-axqe%2Csn-axq7sn7s&ms=au%2Crdu&mv=m&mvi=3&pl=23&nh=%2CIgpwcjAyLmxlZDAzKgkxMjcuMC4wLjE&initcwndbps=1630000&mime=video%2Fmp4&gir=yes&clen=380099033&dur=5496.290&lmt=1540876699100611&mt=1574510962&fvip=4&keepalive=yes&fexp=23842630&beids=9466585&c=WEB&txp=5432432&sparams=expire%2Cei%2Cip%2Cid%2Caitags%2Csource%2Crequiressl%2Cmime%2Cgir%2Cclen%2Cdur%2Clmt&sig=ALgxI2wwRgIhAIDBEEOHhkEBZZoFPqTEUWrbzgQQ-7BR0dfzBSl9Tws2AiEAmF6e0aCTtR2L2YcBZRIGACeUnoVuB5cPlSo-l6iJHGUX3D&lsparams=mm%2Cmn%2Cms%2Cmv%2Cmvi%2Cpl%2Cnh%2Cinitcwndbps&lsig=AHylml4wRQIhAPjk5cS7desGzT5Vx4QPsOtAsg6wtY2ShvGeCmwojtnUAiA8fkeGc76lPA-USegzGJCzc4qvvLUMF763x1u5yXSVxQ%3D%3D&ratebypass=yes

❶ bender@ubuntu:~$ wget https://r4---sn-n3toxu-axqe.googlevideo.com/videoplayback?expire=1574532639&ei=vyHZXfo3kp3JBfWbqzg&ip=93.100.207.82&id=o-ABldbr46QqdVsKX_53HJb4sbFnPU60Ki_oStnKOvfARv&itag=136&aitags=133%2C134%2C135%2C136%2C160%2C242%2C243%2C244%2C247%2C278&source=youtube&requiressl=yes&mm=31%2C29&mn=sn-n3toxu-axqe%2Csn-axq7sn7s&ms=au%2Crdu&mv=m&mvi=3&pl=
```

```
23&nh=%2CIgpwcjAyLnxlZDAzKgloHjcuMC4wLjE&initcwndbps=1630000&mime=video%2Fmp4&gir=yes&clen=380099033&dur=5496.290&lmt=15400
76699100611&mt=1574510962&fvip=4&keepalive=yes&fexp=23842630&beids=9466585&c=WEB&txp=5432432&sparams=expire%2Cei%2Cip%2Cid%
2Caitags%2Csource%2Crequiressl%2Cmime%2Cgir%2Cclen%2Cdur%2Clmt&sig=ALgxI2wAgIhAIDBEEOMhkI3Z2oFPqTEUMrbzgQQ-7BR0dfzBSl9Tws2
AiEAmF6e0aCTtR2L2YcBZRIGACeUnoVu85cPlSo-l6iJHGUK3D&lsparams=mh%2Cmm%2Cms%2Cmv%2Cpl%2Cnh%2Cinitcwndbps&lsig=AHylml4wRQ
IhAPjk5c57desGzTSVx4QPs0tAsg6wtY2ShvGeCnwojtnUAlA8fkaGc76lPA-USegzGJCzc4qvvLUMF763x1u5yXSVxQ%3D%3D&ratebypass=yes
-O revolution-os.mp4

[1] 1263
                    ...         ...         ...         ...
? [31] 12663

HTTP-запрос отправлен. Ожидание ответа... 403 Forbidden

2015-12-26 19:00:05 ОШИБКА 403: Forbidden.

bender@ubuntu:~$ wget 'https://r4---sn-n3toxu-axqe.googlevideo.com/videoplayback?expire=1574532639&ei=vyHZXfo3kp3JB
fWbqzg&ip=93.100.207.82&id=o-AB1dbr46QqdVsKX_53HJb4sbFnPU68Ki_oStnKOvfARv&itag=136&aitags=133%2C134%2C135%2C136%2C160%2C242
%2C243%2C244%2C247%2C278&source=youtube&requiressl=yes&mm=31%2C29&mn=sn-n3toxu-axqe%2Csn-axq7sn7s&ms=au%2Crdu&mv=m&mvi=3&pl
=23&nh=%2CIgpwcjAyLnxlZDAzKgloHjcuMC4wLjE&initcwndbps=1630000&mime=video%2Fmp4&gir=yes&clen=380099033&dur=5496.290&lmt=1540
0766991006118mt=1574510962&fvip=4&keepalive=yes&fexp=23842630&beids=9466585&c=WEB&txp=5432432&sparams=expire%2Cei%2Cip%2Cid
%2Caitags%2Csource%2Crequiressl%2Cmime%2Cgir%2Cclen%2Cdur%2Clmt&sig=ALgxI2wAgIhAIDBEEOMhkI3Z2oFPqTEUMrbzgQQ-7BR0dfzBSl9Tws
2AiEAmF6e0aCTtR2L2YcBZRIGACeUnoVu85cPlSo-l6iJHGUK3D&lsparams=mh%2Cmm%2Cms%2Cmv%2Cpl%2Cnh%2Cinitcwndbps&lsig=AHylml4wR
QIhAPjk5c57desGzTSVx4QPs0tAsg6wtY2ShvGeCnwojtnUAlA8fkaGc76lPA-USegzGJCzc4qvvLUMF763x1u5yXSVxQ%3D%3D&ratebypass=yes'
-O revolution-os.mp4
                    ...         ...         ...         ...
Распознаётся r4---sn-axq7sn7s.googlevideo.com (r4---sn-axq7sn7s.googlevideo.com)...
173.194.2.22, 2a00:1450:4012:2::16

Подключение к r4---sn-axq7sn7s.googlevideo.com (r4---sn-
axq7sn7s.googlevideo.com)|173.194.2.22|:443... соединение установлено.

HTTP-запрос отправлен. Ожидание ответа... 200 OK

Длина: 380099033 (362M) [video/mp4]

Сохранение в: «revolution-os.mp4»

revolution-os.mp4       0%[                    ]   2,02M  1,69MB/s
```

В листингах 5.32 и 5.33 использованы разные способы множественного экранирования '' и "", хотя в этих примерах между ними нет разницы. Экранирование метасимволов в двойных кавычках "" носит название *слабого* экранирования, потому что не распространяется на метасимволы подстановки **$** и `` и метасимвол одиночного экранирования ****. Экранирование метасимволов в одинарных кавычках '' носит название *сильного* экранирования, потому как распространяется на абсолютно любые метасимволы.

Листинг 5.34. Сильное и слабое экранирование

```
bender@ubuntu:~$ cal
     Ноября 2019
Вс Пн Вт Ср Чт Пт Сб
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

```
❶ bender@ubuntu:~$ notify-send $(date) $(cal)
  Invalid number of options.
  bender@ubuntu:~$ set -x
❷ bender@ubuntu:~$ notify-send $(date) $(cal)
  ++ date
  ++ cal
                ①1②1③1      ④1     ... 1
  + notify-send Сб ноя 23 15:20:23 MSK 2019 Ноября 2019 Вс Пн Вт Ср Чт Пт Сб 1 2 3 4 5 6 7 8 9
  10 11 12 13 14 15 16 17 18 19 20 21 22 $'_\b2_\b3' 24 25 26 27 28 29 30
  Invalid number of options.
❸ bender@ubuntu:~$ notify-send '$(date)' '$(cal)'
  + notify-send '$(date)' '$(cal)'
❹ bender@ubuntu:~$ notify-send "$(date)" "$(cal)"
  ++ date
  ++ cal                                    ①1
  + notify-send 'Сб ноя 23 15:22:12 MSK 2019'  '     Ноября 2019
  Вс Пн Вт Ср Чт Пт Сб
                  1  2
   3  4  5  6  7  8  9
  10 11 12 13 14 15 16
  17 18 19 20 21 22 23
  24 25 26 27 28 29 30
                     '
```

В примерах из листинга 5.34 необходимо текущее время и календарь на текущий месяц отправить пользователю в качестве уведомления в сеансе графического интерфейса. При помощи команды **notify-send(1)** можно послать уведомление, заголовок которого передается первым ее параметром, а текст уведомления — вторым. Прямое решение ❶ с подстановками вывода команд **date(1)** и **cal(1)** не срабатывает и завершается со странным результатом. В режиме трассировки команд интерпретатора ❷ видно, что пробелы ①②③... в выводе команд **date(1)** и **cal(1)** были естественным образом подставлены как аргументы команды **notify-send(1)**, что привело к передаче более чем двух ожидаемых аргументов. Решение с сильным экранированием ❸ посылает уведомление с литеральным написанием подстановок, без их выполнения, что тоже не является ожидаемым результатом. Слабое экранирование ❹ работает, как ожидается, с выполнением подстановок и экранированием пробелов после выполнения подстановки, в результате чего формируются два правильных аргумента команды **notify-send(1)**.

5.6. Списки команд

Перенаправления и подстановки командного интерпретатора позволяют производить различные вычисления, но не разрешают *управлять ходом* вычислений — выполнять различные действия в зависимости от результата вычислений или циклически повторять вычисления. Для управления ходом выполнения сценариев на языке командного интерпретатора служат *списки* команд.

Простейшие списки формирует пользователь при интерактивной работе с командным интерпретатором, последовательно запуская команды при помощи управляющего символа перевода строки ↵ или параллельно запуская задания *(см. разд. 4.8.1)* посредством метасимвола **&** запуска задач в «фоновом» режиме.

В пакетном режиме работы такие списки называются *простыми* и формируются конструкциями **command$_1$; command$_2$; ...** и **command$_1$ & command$_2$ & ...**, называемыми *простым синхронным* (листинг 5.35) и *простым асинхронным* (листинг 5.36) списками соответственно.

Листинг 5.35. Простой последовательный (синхронный) список команд

```
bender@ubuntu:~$ ① dd if=/dev/dvd of=dvd.iso ↵
                                              ↓
                                ⑦ ⑥ ⑤ ④ ③ ② ① ⑧ ⑨ ⑩ ⑪
150772+0 записей получено
150772+0 записей отправлено
77195264 байт (77 MB, 74 MiB) скопирован, 1,83868 s, 42,0 MB/s
bender@ubuntu:~$ ② ls -lh dvd.iso
-rw-rw-r-- 1 bender bender 74M ноя 23 15:27 dvd.iso
              ①↴              ②↴
bender@ubuntu:~$ bzip2 -v dvd.iso ; ls -lh dvd.iso.bz2
                                  ↓
❶   dvd.iso:  1.271:1,  6.293 bits/byte, 21.34% saved, 77195264 in, 60721096 out.
❷ -rw-rw-r-- 1 bender bender 58M ноя 23 15:27 dvd.iso.bz2
```

Интерактивный режим взаимодействия ①② с командным интерпретатором заставляет пользователя дожидаться завершения текущей выполняющейся программы и появления приглашения командного интерпретатора для того, чтобы запустить последующую команду. Последовательный список позволяет организовать подобное последовательное выполнение посредством командного интерпретатора.

Листинг 5.36. Простой параллельный (асинхронный) список команд

```
                ①↴                          ②↴                          ③↴
bender@ubuntu:~$ time xz --best -k plan9.iso -S .b.xz & time xz --fast -k plan9.iso -S .f.xz & ps f
                                              ↓                           ↓
```

```
[1] 11978
[2] 11979
  PID TTY      STAT   TIME COMMAND
11716 pts/0    Ss     0:00 -bash
11978 pts/0    S      0:00  \_ -bash
❶ 11981 pts/0    R      0:00  |   \_ xz --best -k plan9.iso -S .b.xz
11979 pts/0    S      0:00  \_ -bash
❷ 11982 pts/0    R      0:00  |   \_ xz --fast -k plan9.iso -S .f.xz
11980 pts/0    R+     0:00  \_ ps f
   binder@ubuntu:~$ wait ; ls -lh plan9.*
                                ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
❷ real 0m37.791s
user 0m37.476s
sys  0m0.264s
                                ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
❶ real 3m7.409s
user 3m6.656s
sys  0m0.472s
[1]-  Готово                  time xz --best -k plan9.iso -S b.xz
[2]+  Готово                  time xz --fast -k plan9.iso -S f.xz
-rw-r--r-- 1 binder bender 287M    нояб. 28 15:47 plan9.iso
❶ -rw-r--r-- 1 binder binder  61M ➧ нояб. 28 15:47 plan9.iso.b.xz
❷ -rw-r--r-- 1 binder binder  89M ➧ нояб. 28 15:47 plan9.iso.f.xz
```

При помощи асинхронного списка (см. листинг 5.36), наоборот, организуется запуск сразу нескольких процессов в «фоновом» режиме, дождаться завершения которых позволяет встроенная команда интерпретатора **wait**.

5.6.1. Условные списки

Условные списки команд представляют собой компактные, но достаточно выразительные конструкции, управляющие ходом выполнения сценария. Как указывалось выше (см. специальный параметр **?** в *разд. 5.4.2*), каждая команда имеет статус завершения — нулевой при *успешном* выполнении и отличный от нуля при *неуспешном*. Условный список «И» формируется конструкцией вида **command$_1$ && command$_2$ && ...** и выполняет команды *последовательно*, запуская очередную команду, только если предыдущая закончилась *успешно* (заканчивая свое выполнение после первой *неуспешной* команды). Условный список «ИЛИ» вида **command$_1$ || command$_2$ || ...**, наоборот, очередную команду запускает только в случае *неуспеха* предыдущей (заканчивая свое выполнение после первой *успешно* выполнившейся команды).

В примере из листинга 5.37 безусловная попытка ❶ смонтировать ISO-образ диска неуспешна, потому что эта операция уже была выполнена ранее. Условный список «ИЛИ» ❷ организует проверку «смонтированности» файловой системы в указанный каталог. Если первая команда списка **findmnt(1)** завершится неуспехом (если никакая файловая система в указанный каталог не смонтирована), то в этом случае будет запущена команда монтирования **fuseiso(1)**. В противном случае результатом успешно выполнившейся команды **findmnt(1)** будет вывод информации о файловой системе, уже смонтированной в целевой каталог.

Листинг 5.37. Список «ИЛИ»

```
bender@ubuntu:~$ fuseiso dvd.iso ~/.dvd
❶ bender@ubuntu:~$ fuseiso dvd.iso ~/.dvd
fuse: mountpoint is not empty
fuse: if you are sure this is safe, use the 'nonempty' mount option
❷ bender@ubuntu:~$ set -x
bender@ubuntu:~$ findmnt ~/.dvd || fuseiso dvd.iso ~/.dvd
+ findmnt /home/bender/.dvd
TARGET          SOURCE  FSTYPE      OPTIONS
/home/bender/.dvd fuseiso fuse.fuseiso rw,nosuid,nodev,relatime,user_id=1000,group_id=1010
```

Условный список «И» в примере из листинга 5.38, наоборот, пытается размонтировать файловую систему, только если она была смонтирована ранее.

Листинг 5.38. Список «И»

```
bender@ubuntu:~$ fusermount -u ~/.dvd
fusermount: entry for /home/bender/.dvd not found in /etc/mtab
bender@ubuntu:~$ set -x
bender@ubuntu:~$ findmnt ~/.dvd && fusermount -u ~/.dvd
+ findmnt /home/bender/.dvd
```

Условные списки можно комбинировать, например, в виде **command$_1$ && command$_2$ || command$_3$ ...** или **command$_1$ || command$_2$ && command$_3$**. Сначала будет выполнена первая команда, и начнется анализ ее статуса завершения по условиям списка. Если после команды указано условие «И» **&&**, то при ее успешном завершении будет выполнена *следующая команда*, а при неуспешном анализ продолжится на *следующем условии* и т. д. И наоборот, если после команды указано условие «ИЛИ», то при ее неуспешном завершении будет выполнена *следующая команда*, а при успешном анализ продолжится на *следующем условии*. Как только

будет найдена очередная команда для выполнения, она будет выполнена и начнется анализ *ее статуса* выполнения по условиям списка, следующим за ней.

Так, например, в листинге 5.39 комбинированный список «ИЛИ-И» всегда приводит к выводу списка файлов из каталога, куда смонтирован ISO-образ, вне зависимости от того, был ли он туда смонтирован до запуска списка или был смонтирован командами при его выполнении.

Листинг 5.39. Комбинация списков «И» и «ИЛИ»

```
bender@ubuntu:~$ set -x
bender@ubuntu:~$ findmnt ~/.dvd || fuseiso dvd.iso ~/.dvd && ls -a ~/.dvd
❶ + findmnt /home/bender/.dvd
❷ + fuseiso dvd.iso /home/bender/.dvd
❸ + ls --color=auto -a /home/bender/.dvd
      acme    bootdisk.img  env      LICENSE.afpl  mips    pbsraw  sparc
..    adm     cfg           fd       LICENSE.gpl   mnt     power   sys
386   amd64   cron          lib      lp            n       power64 tmp
9load arm     dist          LICENSE  mail          NOTICE  rc      usr

bender@ubuntu:~$ findmnt ~/.dvd || fuseiso dvd.iso ~/.dvd && ls -a ~/.dvd
❶ + findmnt /home/bender/.dvd
TARGET            SOURCE  FSTYPE       OPTIONS
/home/bender/.dvd fuseiso fuse.fuseiso rw,nosuid,nodev,relatime,user_id=1008,...
❸ + ls --color=auto -a /home/bender/.dvd
      acme    bootdisk.img  env      LICENSE.afpl  mips    pbsraw  sparc
..    adm     cfg           fd       LICENSE.gpl   mnt     power   sys
386   amd64   cron          lib      lp            n       power64 tmp
9load arm     dist          LICENSE  mail          NOTICE  rc      usr
```

5.6.2. Составные списки: ветвление

Условные списки «И» и «ИЛИ» являются простейшей формой *ветвления* хода выполнения сценария в зависимости от успеха или неудачи выполнения той или иной команды. При помощи специальной команды[1] **test(1)**, позволяющей выполнять проверки *логических выражений*, можно осуществлять ветвление сценария, например, в зависимости от определенных условий или значений тех или иных параметров.

[1] Аналогично специальной команде **expr(1)**, предназначенной для вычисления арифметических выражений.

В листинге 5.40 показано, что команда **test(1)** и ее «красивая» форма **[** изначально являются внешними командами, что также влечет за собой накладные расходы на системные вызовы **fork(2)** и **execve(2)**. Поэтому в большинстве интерпретаторов обе формы команды **test(1)** реализованы еще и как встроенные команды.

Листинг 5.40. Команды test и [

```
bender@ubuntu:~$ which test
/usr/bin/test
bender@ubuntu:~$ which [
/usr/bin/[
bender@ubuntu:~$ type -a test
test – это встроенная команда bash
test является /usr/bin/test
test является /bin/test
bender@ubuntu:~$ type -a [
[ – это встроенная команда bash
[ является /usr/bin/[
[ является /bin/[
bender@ubuntu:~$ test -f /etc/passwd
bender@ubuntu:~$ echo $?
0
bender@ubuntu:~$ [ -w /etc/passwd
-bash: [: отсутствует символ «]»
bender@ubuntu:~$ [ -w /etc/passwd ]
bender@ubuntu:~$ echo $?
1
```

Команда **test(1)** выполняет проверку логических выражений и заканчивается успехом, если проверяемое выражение *истинно*, и неуспехом, если проверяемое выражение *ложно*. Именно это ее свойство используется для реализации ветвления. Например, во втором примере из листинга 5.41 проверятся наличие блочного (**-b**) файла устройства **/dev/cdrom** и при наличии запускается команда **eject(1)**, предписывающая драйверу устройства открыть лоток привода CD/DVD.

Листинг 5.41. Ветвление при помощи условных списков

```
bender@ubuntu:~$ which clear && clear || tput clear
bender@ubuntu:~$ [ -b /dev/cdrom ] && eject /dev/cdrom
```

Условные списки удобно использовать для ветвления, если в каждой ветви выполняется по одной команде, как в первом примере из листинга 5.41, где посредством **which(1)** проверяется наличие команды **clear(1)**, которая при наличии и вызывается для очистки терминала. В противном случае терминал очищается при помощи управляющей последовательности, которую выводит команда **tput(1)**.

При необходимости выполнить в каждой ветви сценария несколько команд используется конструкция ветвления «ЕСЛИ» вида

if [!] list; then list; [elif [!] list; then list;] ... [else list;] fi,

являющаяся *составным списком* и использующая *ключевые слова* языка командного интерпретатора: **if**, **then**, **elif**, **else**, **fi**. Стоит заметить, что в самой конструкции *логические выражения* отсутствуют, а ветвление основано на статусах завершения списков команд, указывающихся после ключевых слов **if** и **elif**. В листинге 5.42 из ISO-образа извлекается каталог **sys/src**, но двумя разными способами. Если установлен архиватор **7z(1)** и обнаружен командой **which(1)**, то выполняется «успешная ветвь» ① с его использованием, иначе выполняется «неуспешная» ветвь с использованием монтирования/размонтирования ISO-образа при помощи **fuseiso(1)**.

Листинг 5.42. Простое ветвление по результату выполнения команд

```
❶ bender@ubuntu:~$ if which 7z ↵
  > then ↵
①>    7z x plan9.iso sys/src ↵
  > else ↵
②>    fuseiso dvd.iso ~/.dvd ↵
  >    cp -a ~/dvd/sys/src . ↵
  >    fusermount -u ~/.dvd ↵
  > fi ↵

❷ bender@ubuntu:~$ if ! findmnt ~/.dvd ↵
  > then ↵
  >    fuseiso dvd.iso ~/.dvd ↵
  > fi ↵

❸ bender@ubuntu:~$ if test -b /dev/cdrom; then eject /dev/cdrom; fi
```

В примере ❷ из листинга 5.42 используется признак отрицания **!**, который заставляет **if** действовать «наоборот», т. е. запускать ветвь **then**, если список после

if закончился *неуспешно*, и ветвь **else** в противном случае. В примере ❸ стоит обратить внимание на то, что списки, использующиеся в конструкции **if**, могут отделяться символом **;**, а не символом перевода строки ↵, как в примерах ❶ и ❹.

В листинге 5.43 приведен маленький сценарий на языке командного интерпретатора, исполняющийся интерактивно в режиме трассировки команд. В нем используются практически все конструкции командного интерпретатора: перенаправление потоков, конвейер, подстановки команд, параметров, арифметических выражений, экранирование и составной список ветвления.

Листинг 5.43. Простое ветвление по результату проверки условий

```
   bender@ubuntu:~$ set -x
❶  bender@ubuntu:~$ load=$(awk '{printf "%d", $1*100 + $2*10 + $3}' < /proc/loadavg)
   ++ awk '{printf "%d", $1*100 + $2*10 + $3}'
   + load=86
❷  bender@ubuntu:~$ power=$(( $(nproc)*100 ))
   ++ nproc
   + power=400
❸  bender@ubuntu:~$ temp=$(( $(cat /sys/class/thermal/thermal_zone0/temp)/1000 ))
   ++ cat /sys/class/thermal/thermal_zone0/temp
   + temp=55
❹  bender@ubuntu:~$ if [ $load -ge $power -a $temp -gt 80 ] ↵
   > then ↵
   >   notify-send "Обнаружена перегрузка: t=$temp C" "$(top -bn1 | head -5)" ↵
   > fi ↵
⑦  + '[' 86 -ge 400 -a 55 -gt 80']'
```

Сначала ❶ вычисляется «интегральная» нагрузка на систему **load** путем «взвешивания» с весами 100, 10 и 1 статистики за последние 1, 5 и 15 мин о среднем количестве процессов, стоящих в очереди на получение процессорного времени или доступа к устройству ввода-вывода. Измеряемая ядром статистика считывается из *трех* столбцов *одной* строки файла **/proc/loadavg** псевдофайловой системы **proc(5)**. Взвешивание выполняется при помощи процессора текстовых таблиц **W:[AWK]** *(см. разд. 5.8.3)*, основное назначение которого состоит в разбиении строк на столбцы и построчном выполнении указанных действий над ними. В данном случае **awk(1)** выводит в формате (**printf**) целого числа (**%d**) результат взвешивания на основе значений статистики из первого (**$1**), второго (**$2**) и третьего (**$3**) столбцов.

Затем ❷ вычисляется «мощность» вычислительной системы **power** путем умножения 100 (процентов) на количество ядер процессора. Следующим шагом ❸ вычисляет-

ся температура **temp** процессора в градусах °C, рассчитываемая путем деления нацело измеряемой ядром (из псевдофайловой системы **sysfs**) температуры на 1000 (миллиградусов).

Завершает сценарий составной список ветвления **if** ❹, проверяющий при помощи «красивой» формы **[** команды **test(1)** логическое выражение |*нагрузка* ≥ *мощность* И *температура* > 80°C| на истинность и уведомляющий при помощи **nodtify-send(1)** пользователя о перегрузке, в сообщении которого будет приведена текущая статистика потребления ресурсов из первых 5 строк вывода команды **top(1)**.

Еще один вид составного списка, предназначенного для организации *множественного* ветвления, реализуется конструкцией

case word in [[(] pattern$_1$ [| pattern$_2$]...) list$_1$;;]... esac

с ключевыми словами **case**, **in**, **esac** и признаком окончания ветви **;;**. При множественном ветвлении используются не логические[1], а *шаблонные* выражения, заимствованные у подстановок имен файлов. Для определения ветви исполнения слово **word** проверяют на соответствие шаблонам **pattern**$_i$ слева направо (сверху вниз), при совпадении с одним из которых выполняется соответствующий список команд **list**$_j$, и дальше никакие проверки не производятся.

В примере из листинга 5.44 сильно заэкранированный текст небольшого сценария командного интерпретатора присваивается переменной **PROMPT_COMMAND**, выполнение команд которой происходит каждый раз перед выводом первичного приглашения **PS1**. В роли такой команды выступает составной список множественного ветвления **case**, который анализирует статус завершения последней выполненной интерпретатором команды при помощи подстановки значения специального параметра **$?**.

Если статус завершения равен **0** (успех) или **127** (команда не найдена), то вспомогательной «выдуманной» переменной **PROMPT_COLOR** присваивается управляющая последовательность *(см. разд. 2.4)* «нормального» (**sgr0**) режима вывода на терминал. Если статус завершения равен **130** (штатное завершение по сигналу **SIGINT**, например, при нажатии на **^C**) или **131** (аварийное завершение по сигналу **SIGQUIT**, например, при нажатии на **^**), то используется управляющая последовательность «негативного» (**rev**) вывода ❶. Если же команда завершилась со статусом **1** или с любым другим ошибочным статусом, то будет использована управляющая последовательность «жирного» (**bold**) начертания ❷.

---

[1] При «обычном» ветвлении **if-then-else** для определения ветви исполнения тоже используются не логические выражения, а статусы завершения списков.

Листинг 5.44. Множественное ветвление

```
bender@ubuntu:~$ PROMPT_COMMAND='↵
> case $? in ↵
>   0|127) PROMPT_COLOR=`tput sgr0`;; ↵
>   13[01]) PROMPT_COLOR=`tput rev`;; ↵
>   1) PROMPT_COLOR=`tput bold`;; ↵
>   *) echo "exit code = $?"; PROMPT_COLOR=`tput bold`;; ↵
> esac'↵

bender@ubuntu:~$ PS1='`echo $PROMPT_COLOR`\u@\h \w\$ `tput sgr0`'
bender@ubuntu ~$ dd
^C0+0 записей получено
0+0 записей отправлено
скопировано 0 байт (0 B), 0,900673 с, 0,0 kB/c
bender@ubuntu ~$

❶ bender@ubuntu ~$ date
Сб ноя 23 15:55:19 MSK 2019
bender@ubuntu ~$

bender@ubuntu ~$ grep bender /etc/shadow
grep: /etc/shadow: Отказано в доступе
exit code = 2
❷ bender@ubuntu ~$
```

5.6.3. Составные списки: циклы

Последний важный вид составных списков предназначен для многократного циклического выполнения команд в сценариях командного интерпретатора. Различают цикл с *параметром*, реализуемый конструкцией

for name in [words ...]; do list; done,

и циклы с *условием* «ПОКА»

while [!] list; do list; done

и «ДО»

unitl [!] list do list; done

с ключевыми словами **for**, **in**, **while**, **until**, **do**, **done** соответственно.

В примере из листинга 5.45 цикл с параметром используется для создания галереи миниатюр фотографий при помощи составного списка **for** и подстановки вывода

команд. В режиме трассировки команд интерпретатора видно, что команда **convert(1)** в теле цикла выполняется столько раз, сколько слов (найденных имен файлов) было подставлено из вывода команды **find(1)**. При этом переменная **file** на каждой итерации цикла **()** принимает очередное значение из подставленного списка. В результате последовательных подстановок разных значений переменной **$file** были выполнены одинаковые преобразования разных файлов изображений.

Листинг 5.45. Цикл с параметром: создание галереи миниатюр фотографий

```
bender@ubuntu:~$ set -x
bender@ubuntu:~$ for file in $(find DCIM -name '*.jpg') ↵
> do ↵
>   convert $file -resize 100x $(basename $file .jpg).mini.jpg ↵
> done ↵
❶ ++ find DCIM -iname '*.jpg'
() + for file in '$(find DCIM -name '\''*.jpg'\'')'
   ++ basename DCIM/DSC_0067.jpg .jpg
   + convert DCIM/DSC_0067.jpg -resize 100x DSC_0067.mini.jpg
() + for file in '$(find DCIM -name '\''*.jpg'\'')'
   ++ basename DCIM/DSC_0189.jpg .jpg
   + convert DCIM/DSC_0189.jpg -resize 100x DSC_0189.mini.jpg
                ...          ...          ...          ...
() + for file in '$(find DCIM -iname '\''*.jpg'\'')'
   ++ basename DCIM/DSC_0062.jpg .jpg
   + convert DCIM/DSC_0062.jpg -resize 100x DSC_0062.mini.jpg
⃠
```

В этом примере для формирования результирующих имен файлов используется подстановка вывода команды **basename(1)** для отрезания «расширений» **.jpg** от имен файлов, к которым затем приклеиваются новые «расширения» **.mini.jpg**.

В примере из листинга 5.46 составной список **for** используется вместе с подстановкой команды **seq(1)**, формирующей список чисел 1, ..., 254, очередные значения из которого принимает переменная **node**. На каждой итерации цикла **()** используется подстановка значения **$node** для формирования IP-адреса очередного узла локальной сети, доступность которого проверяется при помощи команды **ping(1)**.

Листинг 5.46. Цикл с параметром: проверка доступности узлов локальной сети

```
bender@ubuntu:~$ for node in $(seq 1 254) ↵
> do ↵
```

```
   >  ping -c 1 -W 1 192.168.1.$node ↵
   > c ↵
   ++ seq 1 254
() + for node in '$(seq 1 254)'
   + ping -c 1 -W 1 192.168.1.1
   PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.

   --- 192.168.1.1 ping statistics ---
   1 packets transmitted, 0 received, 100% packet loss, time 0ms
                  ...          ...          ...          ...
() + for node in '$(seq 1 254)'
   + ping -c 1 -W 1 192.168.1.45
   PING 192.168.16.45 (192.168.16.45) 56(84) bytes of data.
   64 bytes from 192.168.16.45: icmp_req=1 ttl=62 time=5.25 ms

   --- 192.168.16.45 ping statistics ---
   1 packets transmitted, 1 received, 0% packet loss, time 0ms
   rtt min/avg/max/mdev = 5.257/5.257/5.257/0.000 ms
                  ...          ...          ...          ...
() + for node in '$(seq 1 254)'
   + ping -c 1 -W 1 192.168.1.254
   PING 192.168.1.1 (192.168.1.254) 56(84) bytes of data.

   --- 192.168.1.254 ping statistics ---
   1 packets transmitted, 0 received, 100% packet loss, time 0ms
⦸
```

В примере из листинга 5.47 на основе составного списка **while** организован «индикатор прогресса» процесса фонового сжатия ISO-образа диска при помощи опроса его состояния. Цикл выполняется, *пока* успешна команда **ps(1)**, опрашивающая процесс, PID которого передан при помощи подстановки специального параметра **$!**. При каждой итерации цикла **()** команда **ls(1)** выводит размер выходного файла **dvd.iso.bz2**, а команда **sleep(1)** приостанавливает выполнение на **1** секунду.

Листинг 5.47. Цикл «ПОКА»: ожидание завершения процесса

```
   bender@ubuntu:~$ bzip2 -kf dvd.iso &
   [1] 5773
   bender@ubuntu:~$ set -x
   bender@ubuntu:~$ while ps p $! ; do ls -lh dvd.iso.bz2; sleep 1 ; done
() + ps p 5773
   PID TTY      STAT   TIME COMMAND
```

```
  5773 pts/0    R      0:05 bzip2 -kf dvd.iso
+ ls --color=auto -lh dvd.iso.bz2
-rw------- 1 bender bender 9,6M  дек.  12 13:38 dvd.iso.bz2
                    ...          ...          ...          ...
+ sleep 1
+ ps p 5773
PID TTY      STAT   TIME COMMAND
  5773 pts/0    R      0:58 bzip2 -kf dvd.iso
+ ls --color=auto -lh dvd.iso.bz2
-rw------- 1 bender bender 87M  дек.  12 13:39 dvd.iso.bz2
[1]+  Готово                 bzip2 -kf dvd.iso
+ sleep 1
+ ps p 4523
PID TTY      STAT   TIME COMMAND
```

В примере из листинга 5.48 составной список **while** используется в качестве генератора пар имен файлов для массового параллельного переименования. Для этого стандартный поток ввода построчно считывается в цикле при помощи встроенной команды интерпретатора **read**, причем на каждой итерации цикла переменная **FN** принимает значение очередной строки, а команда **read** завершается успешно до исчерпания строк. В результате выполнения встроенной команды **echo** и при помощи подстановки значения переменной **$FN** и подстановок вывода команд **dirname(1)** и **basename(1)** на стандартном потоке вывода составного списка формируются пары имен файла — исходное считанное путевое имя и синтезированное путевое имя с измененным «расширением файла». Исходный поток имен формируется командой **find(1)**, а результирующий поток передается команде **xargs(1)**, которая в свою очередь передает пары имен (**-n2**) командам **mv(1)**, запускаемым параллельно в количестве ядер процессора, которое было получено командой **nproc(1)**.

Листинг 5.48. Цикл «ПОКА»: массовое параллельное переименование файлов

```
bender@ubuntu:~$ find DCIM -name '*.jpg' | ↵
> while read FN ; do echo $FN $(dirname $FN)/$(basename $FN .jpg).jpeg ; done | ↵
> xargs -n2 -P $(nproc) mv ↵
```

Аналогично, в примере из листинга 5.49 составной список **while** используется в качестве генератора ❶ классификатора файлов по их контрольным MD5-суммам. При помощи команд **find(1)**, **xargs(1)** и **md5sum(1)** формируется поток строк, в первом столбце которых будет выведена контрольная сумма W:[MD5] файла, а во втором столбце — его имя. Полученный поток построчно считывается в цикле при помо-

щи встроенной команды интерпретатора **read**, причем на каждой итерации переменные **sum** и **file** принимают значения соответственно первого и второго столбцов строки, при этом во временном каталоге создаются файлы классификатора, именованные значениями **$sum**, в содержимое которых добавляются имена **$file**. Если контрольные суммы нескольких обрабатываемых файлов одинаковые, то их имена будут добавлены в разные строки одного и тот же файла классификатора. Остается только найти те файлы классификатора, которые содержат более одной строки, они и будут содержать имена дубликатов. Для поиска по классификатору ❷ используется аналогичная связка **find(1)**, **xargs(1)** и **wc(1)**, формирующая на потоке вывода текстовую таблицу по два столбца в строке, в первом из которых указано количество строк в файле классификатора, а во втором — его имя. Затем при помощи процессора текстовых таблиц **awk(1)** на стандартный поток вывода печатаются только те имена файлов классификатора из второго столбца строк таблицы (**print $2**), в первом столбце которых содержится число, большее единицы (**$1 > 1**). Полученный поток имен файлов отправляется на **xargs(1)** для вывода имен (**-t**) и содержимого при помощи **cat(1)**. Необходимо заметить, что сам классификатор размещен в каталоге файловой системы **W:[tmpfs]**, использующей для хранения оперативную память, и имеет случайное имя, сгенерированное при помощи **mktemp(1)**.

Листинг 5.49. Цикл «ПОКА»: поиск дублирующихся фотографий

```
bender@ubuntu:~$ df -h /run/shm
Файл.система    Размер Использовано  Дост Использовано % Смонтировано в
none`              4,0G          26M  3,9G             1% /run/shm
kotoff@ubuntu:~$ findmnt /run/shm
TARGET   SOURCE FSTYPE OPTIONS
/run/shm        tmpfs  rw,nosuid,nodev,relatim
bender@ubuntu:~$ T=$(mktemp -d /run/shm/XXXXXX)
bender@ubuntu:~$ find DCIM -type f -print0 | xargs -0 -n1 md5sum |↵
❶ > while read sum file ; do echo $file >> $T/$sum ; done↵
❷ bender@ubuntu:~$ find $T -type f | xargs -n1 wc -l |↵
  > awk '$1 > 1 { print $2 }' | xargs -n1 -t cat
☛ cat /dev/shm/fjYfn/4f7ba191573ded309bc0f04e08309d8a
  DCIM/Camera/IMG_20140731_212625+.jpg
  DCIM/Camera/IMG_20140731_212625.jpg
☛ cat /dev/shm/fjYfn/4329ae8006b6935d0c7fd66b57e07c0c
  DCIM/DSC_0046+.JPG
  DCIM/DSC_0046.JPG
```

В листинге 5.50 проиллюстрирована вторая форма цикла с условием — составной список **until**, который применяется для ожидания доступности подключения к Ин-

тернету путем опроса наличия связи с узлом **8.8.8.8** (публичный DNS-сервер компании Google) при помощи команды **ping(1)**. Нужно отметить, что использование **until list do ...; done** может быть заменено эквивалентным **while ! list do ...; done** с указанием признака отрицания **!**.

Листинг 5.50. Цикл «ДО»: ожидание доступности подключения к Интернету

```
bender@ubuntu:~$ set -x
bender@ubuntu:~$ until ping -c1 -w1 8.8.8.8 ; do sleep 1;date ; done
() + ping -c1 -w1 8.8.8.8
☛ connect: Network is unreachable
+ sleep 1
+ date
Сб. дек. 19 09:11:32 MSK 2015
                    ...          ...          ...          ...
() + ping -c1 -w1 8.8.8.8
☛ connect: Network is unreachable
+ sleep 1
+ date
Сб. дек. 19 09:11:39 MSK 2015
⃠ + ping -c1 -w1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
☛ 64 bytes from 8.8.8.8: icmp_req=1 ttl=56 time=11.6 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 11.670/11.670/11.670/0.000 ms
```

5.6.4. Функции

Как и во многих языках программирования, командный интерпретатор имеет средства структуризации сценариев при помощи функций. Составной *именованный* список команд, называемый *функцией*, объявляется при помощи (Bourne- и POSIX-диалекты) конструкций

name() compound-list

или (Korn-диалект)

function name compound-list

с использованием ключевого слова **function**, где **compound-list** — это составной список, например **if**, **case**, **for** или **while**. Сформировать составной список из конвейера, простого или условного списка можно при помощи конструкций **{ list; }**,

или **(list)**, позволяющих выполнить **list** в том же или в отдельном дочернем процессе интерпретатора.

После объявления функция может быть неоднократно вызвана (как и любая другая внешняя или встроенная команда) с разными фактическими параметрами, значения которых в самой функции доступны при помощи подстановки позиционных параметров *(см. разд. 5.4.2)*.

В примере из листинга 5.51 объявляется, а затем вызывается без параметров функция, выполняющая вывод идентификаторов пользовательских учетных записей, доступных в системе (по аналогии с листингом 5.13).

Листинг 5.51. Список пользователей, зарегистрированных в операционной системе

```
bender@ubuntu:~$ function getusers↵
> {↵
>   getent passwd | cut -f 1 -d : | xargs -n1 id↵
> }↵
bender@ubuntu:~$ type -a getusers
getusers — это функция
getusers ()
{
    getent passwd | cut -f 1 -d : | xargs -n1 id
}
bender@ubuntu:~$ getusers
uid=0(root) gid=0(root) группы=0(root)
                    ...          ...          ...          ...
uid=1001(finn) gid=1001(finn) группы=1007(candy),1001(finn)
uid=1002(jake) gid=1002(jake) группы=1002(jake)
                    ...          ...          ...          ...
uid=1003(iceking) gid=1003(iceking) группы=1003(iceking)
uid=1004(marceline) gid=1004(marceline) группы=1004(marceline)
uid=1005(bubblegum) gid=1005(bubblegum) группы=1007(candy),1005(bubblegum)
uid=1006(fitz) gid=1008(fitz) группы=27(sudo),1008(fitz)
uid=1007(skillet) gid=1009(skillet) группы=1009(skillet)
                    ...          ...          ...          ...
uid=1008(bender) gid=1010(bender) группы=1010(bender)
                    ...          ...          ...          ...
```

В листинге 5.52 объявляется функция, являющаяся универсальным экстрактором «архивов». Сначала ❶ при помощи «красивой» формы **[** команды **test(1)** определяется наличие файла **(-f)**, путевое имя которого будет передано первым фактическим параметром при вызове функции. Затем ❷, если заданный файл найден, при помощи множественного ветвления и шаблонов, «примеряемых» к имени заданного

файла, определяется и выполняется в соответствующей ветви команда распаковки tar(1), gunzip(1), bunzip2(1), rar(1), unzip(1), uncompress(1) или 7z(1). При запуске функции в режиме трассировки видно, как работает передача аргументов в функцию, как выполняются списки ветвления и множественного ветвления.

Листинг 5.52. Универсальный экстрактор архивов

```
bender@ubuntu:~$ extract () ↵
❶ > if [ -f $1 ] ↵
> then ↵
❷ >   case $1 in ↵
>       *.tar)            tar xf  $1;; ↵
>       *.tar.bz2|*.tbz2) tar xjf $1;; ↵
>       *.tar.gz|*.tgz)   tar xzf $1;; ↵
>       *.gz)  gunzip     $1;; ↵
>       *.bz2) bunzip2    $1;; ↵
>       *.rar) rar x      $1;; ↵
>       *.zip) unzip      $1;; ↵
>       *.Z)   uncompress $1;; ↵
>       *.7z|*.iso) 7z x  $1;; ↵
>       *) echo "Неизвестен распаковщик для '$1'"; return 1;; ↵
>   esac ↵
> else ↵
>   echo "'$1' не является файлом"; return 1 ↵
> fi
bender@ubuntu:~$ type -a extract
extract является функцией
extract ()
{
                    ...          ...          ...          ...
}
bender@ubuntu:~$ set -x
bender@ubuntu:~$ extract dvd.iso
+ extract dvd.iso
➧ + '[' -f dvd.iso ']'
+ case $1 in
+ 7z x dvd.iso

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=ru_RU.UTF-8,Utf16=on,HugeFiles=on,64 bits,...,ASM,AES-NI)
```

```
Processing archive: dvd.iso
                    ...         ...         ...         ...

bender@ubuntu:~$ extract /tmp
+ extract /tmp
+ '[' -f /tmp ']'
+ echo ''\''/tmp'\'' не является файлом'
'/tmp' не является файлом
+ return 1

bender@ubuntu:~$ extract lynx_2.8.8dev.9-2ubuntu0.12.04.1_all.deb
+ extract lynx_2.8.8dev.9-2ubuntu0.12.04.1_all.deb
+ '[' -f lynx_2.8.8dev.9-2ubuntu0.12.04.1_all.deb ']'
+ case $1 in
+ echo 'Неизвестен распаковщик для '\''lynx_2.8.8dev.9-2ubuntu0.12.04.1_all.deb'\'''
Неизвестен распаковщик для 'lynx_2.8.8dev.9-2ubuntu0.12.04.1_all.deb'
+ return 1
```

Объявленные функции, как и переменные, располагаются в оперативной памяти процесса командного интерпретатора, в силу чего их *время жизни* и *область видимости* так же ограничиваются процессом их интерпретатора. В этом смысле объявленные функции практически неотличимы от встроенных команд интерпретатора и могут расцениваться как его «расширения».

Сохранить объявленные функции и присвоенные переменные нельзя, но можно их повторно объявить и присвоить, воспользовавшись инициализационными dot-файлами *(см. разд. 2.8)* интерпретатора, например сценариями **.bashrc** или **.profile**.

5.7. Сценарии на языке командного интерпретатора

Сценарий на языке командного интерпретатора представляет собой текстовый файл со списком команд, подлежащих выполнению в пакетном режиме. Достаточное количество программного обеспечения в системе написано на языке командного интерпретатора и представлено сценариями в каталогах **/usr/bin** и **/bin**. Сценарии неотличимы от любых других программ и доступны пользователями как внешние команды, формируя таким образом «расширения» операционной системы.

Пользователи, расширяющие таким образом операционную систему за счет собственных сценариев, располагают их обычно в каталоге **~/bin** и наделяют правом исполнения (см. ❶ и ❷ в листинге 5.2). Аналогично, «локальные» расширения, выполняемые системными администраторами для всех пользователей операционной системы, располагаются в **/usr/local/bin** (см. листинг 5.68).

В листинге 5.53 приведен сценарий **ldr** (loader requirements), обратный утилите **ldd(1)** (loader dependencies). Если при помощи **ldd(1)** можно увидеть библиотеки, от которых зависит заданная программа, то сценарий **ldr**, наоборот, показывает программу, использующую заданную библиотеку (листинг 5.54).

В сценарии объявлена ❶ функция **usage**, формирующая сообщение пользователю о правильных параметрах запуска сценария и использующаяся при некорректно заданных параметрах ❷ и ❺-①. В соответствии с соглашениями о выводе сообщение об «ошибке» перенаправлено ❶-① на поток **stderr** при помощи перенаправления потока **stdin** конструкцией **[n]>&m**. По соглашению о статусе завершения с «ошибкой» встроенная команда **exit** возвращает ненулевое значение ❶-②. Корректность параметра **$1** — имени искомой библиотеки — проверяется на «пустоту» ❹ при помощи «красивой» формы **[** команды **test(1)**.

Сам сценарий выполняется следующим образом: сначала ❸ переменной **ldpath** присваивается список имен каталогов, используемых динамическим компоновщиком **ld.so(8)** для поиска библиотек. Для этого из конфигурационных файлов компоновщика **/etc/ld.so.conf.d/*.conf** фильтруются (см. листинг 5.57) строки, незакомментированные символом **#**. Затем ❹, в зависимости от формы задания имени целевой библиотеки, переменной **what** присваивается имя ее файла. Если библиотека была сразу задана абсолютным путевым именем своего файла, соответствующим шаблонному выражению **/*/lib*.so.[0-9]**, то оно используется непосредственно. Если библиотека была задана «предметным» именем ***NAME***, то при помощи команды **find(1)** организуется поиск абсолютного путевого имени файла библиотеки по шаблону **lib*NAME*.so.[0-9]** в каталогах компоновщика **ld.so(8)**, перечисленных в переменной **$ldpath**, а найденные имена присваиваются переменной **what**. Корректность результата «вычисления» имен файлов библиотеки проверяется ❺ при помощи «красивой» формы **[** команды **test(1)** на предмет непустоты (**-z**) результатов поиска. На очередном шаге сценария ❻ переменной **where** присваивается список каталогов (перечаленных в переменной окружения **PATH**), содержащих «доступные» пользователю исполняемые файлы программ. Для этого при помощи транслитератора **tr(1)** символы-разделители **PATH**-списка — двоеточия — заменяются требуемыми символами-разделителями — пробелами.

Листинг 5.53. Сценарий поиска программ, использующих указанную библиотеку

```
    bender@ubuntu:~$ cd bin
    bender@ubuntu:~/bin$ cat ldr
    #!/bin/sh
❶ usage() {
|①  echo "Usage: $(basename $0) name | /.../libname.so.0" >&2 ➥
|②  exit 1
↳ }
```

```
❷ [ -z "$1" ] && usage

❸ ldpath=$(grep -h '^[^#]' /etc/ld.so.conf.d/*.conf)

❹ case $1 in
|①   /*/lib*.so.[0-9]) what=$1;;
|②   *) what=$(find $ldpath -name "lib$1.so.[0-9]" 2>/dev/null);;
└→ esac

❺ if [ -z "$what" ]
|  then
|    echo "Library $1 is not found in $LDPATH"
|①   usage
└→ fi

❻ where=$(printenv PATH | tr ':' ' ')

❽     for lib in $what
|     do
|     ❼⓪         find $where -type f |
|     |①           xargs file -L |
|     |②           grep ':.*ELF' |
|     |③           cut -f 1 -d : |
|     |④           while read exe
|     |    do
|     |⑤             ldd $exe | grep -q $lib && echo $exe
|     └→   done
└→    done
```

Целевая работа сценария выполняется одним конвейером ❼, в котором при помощи **find(1)** ищутся ⓪ регулярные файлы (**-type f**) в каталогах **$where**, список имен которых при помощи **xargs(1)** передается ① для классификации **file(1)**. Построенный классификатор **имя:класс** фильтруется ② при помощи **grep(1)**, из которого затем отделяются имена ③ посредством **cut(1)**. Последняя команда конвейера — составной список **while** циклично перебирает ④ имена файлов, считывая их встроенной командой **read** в переменную **exe**, где на каждой итерации цикла проверятся зависимость исполняемого файла **$exe** от заданной библиотеки. Зависимость устанавливается ⑤ по факту успешного завершения **grep(1)**, определяющего только наличие (**-q**) в выводе **ldd(1)** строки с именем путевого файла библиотеки **$lib**.

Кроме того, конвейер ❼ вложен в цикл **for** ❽, который пробегает переменной **lib** по всем найденным ранее ❻② именам файлов библиотек, сохраненных в переменной **$what**.

Листинг 5.54. Утилита ldd(1) и сценарий ldr

```
bender@ubuntu:~$ ldr gtk-3
/usr/bin/eog
/usr/bin/evince
                ...          ...          ...          ...
/usr/bin/gnome-disk-image-mounter
bender@ubuntu:~$ ldd /usr/bin/eog
        linux-vdso.so.1 (0x00007ffd211bf000)
                ...          ...          ...          ...
        libgtk-3.so.0 => /lib/x86_64-linux-gnu/libgtk-3.so.0 (0x00007ff9735e3000)
                ...          ...          ...          ...

bender@ubuntu:~$ ldr pthread | wc -l
❼ ❻ ❺ ❹ ❸ ❷ ❶ ❾ ❽ ❼ ❻
617
```

5.8. Инструментальные средства обработки текста

Данные, которые генерируют, обрабатывают и потребляют внешние и встроенные команды и конструкции интерпретатора, представляют собой *текстовые* потоки с произвольной структурой. Чаще всего в потоке можно выделить *строки*, отделяемые друг от друга управляющим символом перевода строки **\n** (**CR**, **^J** с кодом **0x0A**). Иногда в каждой строке выделяют *поля*, отделяемые друг от друга пробельными символами — управляющим символом горизонтальной табуляции **\t** (**HT**, **^I** с кодом **0x09**) или символом пробела (**SPC** с кодом **0x20**) либо каким-то другим символом, зачастую символом двоеточия **:** или символом вертикальной черты **|**.

Для манипуляции текстовыми данными используют **W:[регулярные выражения]** — формальный язык *поиска подстрок*, удовлетворяющих определенным правилам (табл. 5.2). Регулярные выражения подобны шаблонным выражениям, которые применяются в подстановках имен файлов *(см. разд. 5.4.1)* и в команде **find(1)** при поиске файлов по критерию имени (**-name**).

Регулярные выражения (RE, **R**egular **E**xpressions) учитывают *строковую* структуру обрабатываемых данных (табл. 5.3) и по сравнению с шаблонными выражениями вводят два дополнительных метасимвола — **^** и **$**, обозначающих начало и конец строки соответственно. Различают традиционные для UNIX, *базовые*[1] (BRE, **B**asic

[1] Во всех иллюстративных листингах используются **BRE**, а изучение **ERE** и **PCRE** выходит за рамки этой книги.

RE), *расширенные*[1] (ERE, **E**xtended RE) и *Perl-совместимые* регулярные выражения (PCRE, **P**erl **C**ompatible RE).

Таблица 5.2. Базовые регулярные выражения

Метасимвол	*Значение*
.	Любой одиночный символ
c*	Любое количество символов c
.*	Любое количество любых символов
[ab...z]	Любой символ из набора a, b, ..., z
[^ab...z]	Любой символ НЕ из набора a, b, ..., z
^	Начало строки
$	Конец строки

Таблица 5.3. Регулярные выражения в сравнении с шаблонными выражениями

Шаблонные выражения	*Регулярные выражения*	*Значение*
?	.	Любой одиночный символ
*	.*	Любое количество любых символов

Основной набор инструментальных средств, используемых для обработки текстовых потоков, включает в себя фильтр строк **grep(1)**, транслитератор **tr(1)**, фильтр символов и полей **cut(1)**, склейщик строк и полей **paste(1)**, процессор таблиц **awk(1)** и потоковый редактор **sed(1)**.

5.8.1. Фильтр строк grep

Имя команды **grep(1)** восходит к команде **g** древнейшего текстового редактора **ed(1)**, которая использовала регулярные выражения **re** для глобального поиска строк и применения других команд редактора к ним, например команды печати **p**. В целом встроенная команда редактора **ed(1)** записывалась как **g/re/p**, что затем и дало название эквивалентной внешней команде **grep(1)**, фильтрующей (и печатающей на поток вывода) строки, соответствующие заданному регулярному выражению (рис. 5.5).

[1] В стандарте **W:[POSIX]**.

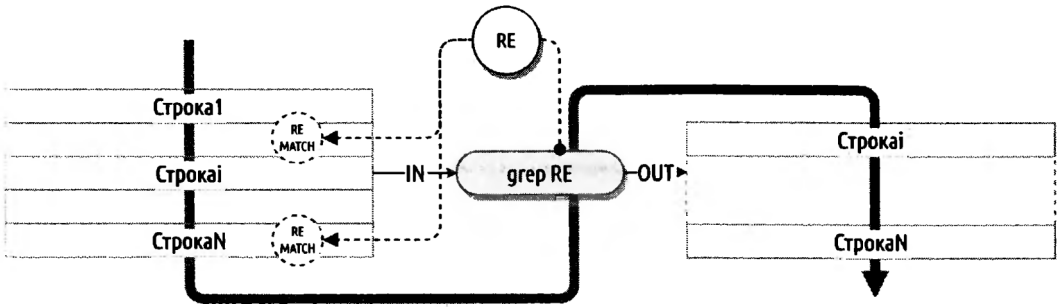


Рис. 5.5. Фильтр строк grep(1)

В примерах из листингов 5.55 и 5.56 fgrep(1) и grep(1) используются для фильтрации строк, содержащих (-F) фиксированные слова, что быстрее, чем обработка полных регулярных выражений.

Листинг 5.55. Выборка строк: список сценариев интерпретатора

```
bender@ubuntu:~ $ file -Li /*bin/* /usr/*bin/* | fgrep shellscript
/bin/bzcmp:                    text/x-shellscript; charset=us-ascii
...          ...               ...                 ...
/bin/setupcon:                 text/x-shellscript; charset=utf-8
...          ...               ...                 ...
/sbin/dkms:                    text/x-shellscript; charset=us-ascii
```

Листинг 5.56. Выборка строк: RSS память процессов браузера chromium

```
bender@ubuntu:~$ ps axo rss,comm | grep -F chrome
137812 chrome
45872 chrome
13068 chrome
104368 chrome
72052 chrome
97244 chrome
50984 chrome
19936 chrome
```

В примере из листинга 5.57 отфильтровываются строки конфигурационного файла команды wget(1), закомментированные символом # в их начале. Для этого выбираются строки, соответствующие регулярному выражению **^[^#]**, которое требует в начале строки ^ наличия символа, не ^ входящего в набор [#].

Листинг 5.57. Выборка строк: фильтрация комментариев

```
bender@ubuntu:/etc$ wc -l /etc/wgetrc
138 /etc/wgetrc
bender@ubuntu:/etc$ grep '^[^#]' /etc/wgetrc
passive_ftp = on
```

В листинге 5.58 при помощи регулярного выражения **[^0-9][0-9][0-9][^0-9]** выбираются строки с двузначными числами — содержащие два символа подряд из набора **[0-9]** (цифры), непосредственно перед которыми и после которых находятся символы не **^** из набора **[0-9]** (не цифры).

Листинг 5.58. Выборка строк с двузначными числами

```
bender@ubuntu:/etc$ grep [^0-9][0-9][0-9][^0-9] /etc/services
systat                11/tcp          users
daytime               13/tcp
                     ...             ...             ...                 ...
z3950           210/tcp         wais            # NISO Z39.50 database
#> Ports are used in the TCP [45,106] to name the ends of logical
rtcm-sc104      2101/tcp                        # RTCM SC-104 IANA 1/29/99
x11             6000/tcp        x11-0           # X Window System
x11             6000/udp        x11-0
x11-1           6001/tcp
                    ...             ...             ...                 ...
# The following is probably Kerberos v5  --- ajt@debian.org (11/02/2000)
```

5.8.2. Фильтр символов и полей cut

Команда **cut(1)** применяется для «вырезания» указанных (порядковым номером) символов или полей (по заданному разделителю) каждой строки (рис. 5.6). Несмотря на название команды, с содержимым файла никаких действий не производится, а символы и поля «выделяются» на поток вывода, что позволяет считать команду фильтром *полей*, аналогичным фильтру *строк* **grep(1)**.

В примере из листинга 5.59 при помощи **cut(1)** отфильтровывается первое поле (**-f 1 -d :**) тех строк классификатора файлов **file(1)**, которые были предварительно отфильтрованы по наличию слова **shellscript**. В результате на поток **stdout** будут выведены только имена файлов, классифицированных как сценарии командного интерпретатора.

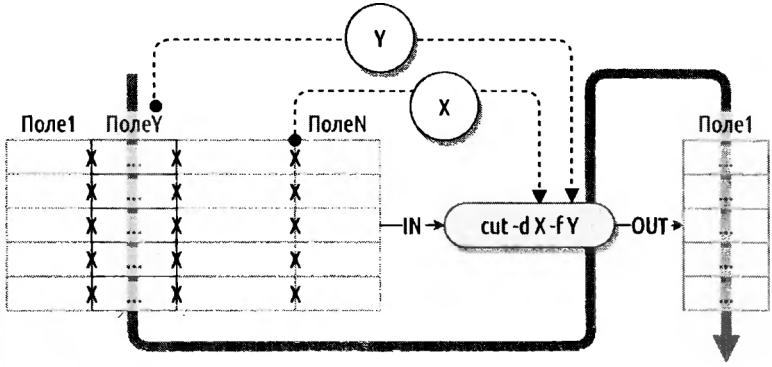


Рис. 5.6. Фильтр столбцов и полей cut(1)

Листинг 5.59. Выборка строк и полей с помощью grep и cut: список имен сценариев интерпретатора

```
bender@ubuntu:~ $ file -Li /*bin/* /usr/*bin/* | grep shellscript | cut -f 1 -d :
/bin/bzcmp
                ...         ...         ...         ...
/bin/setupcon
                ...         ...         ...         ...
/sbin/dkms
```

Аналогично, в примере из листинга 5.60 из вывода команды **ps(1)** отбираются строки свойств процессов с именем **chromium**, затем выделяются символы с **1**-го по **8**-й каждой строки (содержащие суммарное количество резидентной памяти процесса **rss**, помещенное в первые **:8** символов). Отобранные статистические данные склеиваются командой **paste(1)** в одну строку (**-s**) посредством символа **+** в качестве разделителя (**-d**), и полученное таким образом арифметическое выражение отправляется на калькулятор **bc(1)** для подсчета. В результате будет получено суммарное потребление физической памяти всеми процессами Web-браузера **chromium**.

Листинг 5.60. Выборка столбцов и склейка полей: суммарная RSS-память процессов браузера chromium

```
bender@ubuntu:~$ ps axo rss:8,comm | grep chrome | cut -c 1-8 | paste -s -d + | bc
580040
```

5.8.3. Процессор текстовых таблиц awk

Команда **awk(1)** предназначается для обработки текстовых таблиц, столбцы которых разделяются пробельными символами, а строки — символом перевода строки (рис. 5.7). Язык процессора **W:[AWK]**[1] использует регулярные выражения **RE** для

[1] Имя процессора образовано от заглавных букв фамилий его разработчиков — Aho, Weinberger и Kernighan.

выделения *строк*, подлежащих обработке, и подстановочные выражения **$1**, **$2**, ..., **$N** для выделения *столбцов*. Каждая инструкция языка AWK записывается как **/RE/{ ACTION $i ... }** и заставляет процессор выполнить действие **ACTION** со столбцами **$i** каждой строки, соответствующей регулярному выражению **RE**.

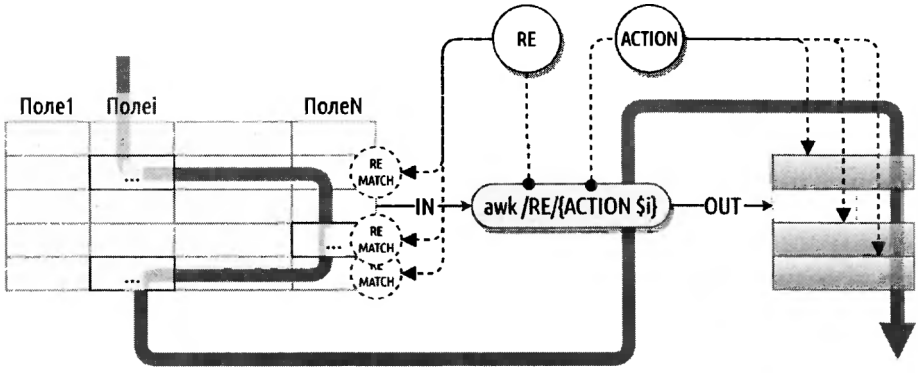


Рис. 5.7. Процессор текстовых таблиц awk(1)

Так, например, в листинге 5.61 при помощи **awk(1)** вместо **grep(1)** и **cut(1)** выделяются имена классифицированных командой **file(1)** файлов, являющихся сценариями командного интерпретатора.

Листинг 5.61. Выборка строк и полей с помощью awk: список имен сценариев интерпретатора

```
bender@ubuntu:~$ file -Li /*bin/* /usr/*bin/* | awk -F: '/shellscript/ { print $1 }'
/bin/bzcmp
                ...         ...         ...         ...
/bin/setupcon
                ...         ...         ...         ...
/sbin/dkms
```

В примере из листинга 5.62 при помощи **awk(1)** подсчитывается суммарное потребление **RSS**-памяти процессами **chromium** при помощи простейшей AWK-программы с тремя инструкциями. Инструкция **BEGIN { sum = 0 }** выполняется *до* анализа строк, инструкция **END { print sum }** — *после* анализа всех строк, а инструкция **/chromium/ { sum += $1 }** выполняется *при* анализе строк и прибавляет к значению переменной **sum** целочисленное значение первого столбца **$1** тех строк, в содержимом которых будет найдена строка **chromium**.

Листинг 5.62. Суммирование по строкам и полям: суммарная RSS-память процессов браузера chromium

```
bender@ubuntu:~$ ps axo rss,comm | ↵
> awk 'BEGIN { sum = 0 } /chrome/ { sum += $1 } END { print sum }'
580108
```

5.8.4. Потоковый редактор текста sed

Потоковый редактор **sed(1)** (stream **ed**itor) применяется для пакетного (неинтерактивного) редактирования текстовых файлов, имеющих строчную структуру (рис. 5.8). Инструкции языка **W:[SED]** используют регулярные выражения **RE** для выделения *строк*, подлежащих редактированию при помощи команд **CMD**: вставки и добавления строк **i** (insert) и **a** (append), замены строк **c** (change), удаления строк **d** (delete), замены подстрок **s** (substitute) и др.

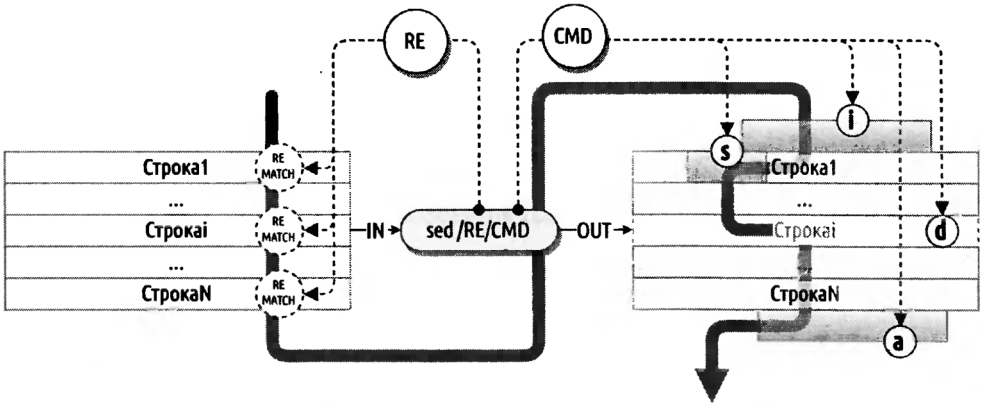


Рис. 5.8. Потоковый редактор sed(1)

В примере из листинга 5.63 при помощи редактора **sed(1)** эмулируется поведение **grep(1)** путем удаления (команда **d**) тех строк потока, которые не (отрицание регулярного выражения **!**) содержат подстроку **chromium**.

Листинг 5.63. Выборка строк: удаление лишних

```
bender@ubuntu:~$ ps axo rss,comm | sed '/chrome/!d'
171820 chrome
45872 chrome
...    ...    ...    ...
51124 chrome
19936 chrome
```

Аналогично, в листинге 5.64 фильтруются строки файла **/etc/wgetrc** путем удаления из вывода строк, не соответствующих регулярному выражению **^[^#]** (в начале строки содержащих знак, не похожий на знак комментария **#**).

Листинг 5.64. Выборка строк: фильтрация комментариев

```
bender@ubuntu:~$ sed '/^[^#]/!d' /etc/wgetrc
passive_ftp = on
```

Кроме выбора отдельных строк инструкциями вида /RE/CMD, sed(1) позволяет адресовать блоки строк, подлежащих редактированию. При помощи инструкций вида /RE1/,/RE2/CMD команда редактирования применяется ко всем строкам, начиная с той, которая соответствует выражению RE1, заканчивая той, которая соответствует выражению RE2.

Так, например, в листинге 5.65 из потока вывода команды lspci(8) выделяется блок текста, начиная со строки, содержащей слово **Ethernet**, вплоть до следующей пустой строки (**^$** между началом и концом которой не содержится ни одного символа).

Листинг 5.65. Выборка блоков текста

```
bender@ubuntu:~$ lspci -v | sed '/Ethernet/,/^$/!d'
00:19.0 Ethernet controller: Intel Corporation 82579LM Gigabit Network Connection
(Lewisville) (rev 04)
        Subsystem: Dell 82579LM Gigabit Network Connection (Lewisville)
                ...         ...             ...             ...
        Kernel driver in use: e1000e
        Kernel modules: e1000e
```

В листинге 5.66 показан типичный пример самого распространенного варианта использования sed(1) с командой **s** — замены одних подстрок другими. При помощи трех нехитрых подстановок отсортированный командой sort(1) вывод команды find(1) превращается в дерево файлов и каталогов, аналогичное тому, что строит команда tree(1). Команда подстановки подстрок имеет форму **/RE/s/FND/RPL/F**, при этом во всех строках, соответствующих RE, будут найдены подстроки, соответствующие FND, и заменены на RPL согласно флагам замены F. Регулярное выражение RE может отсутствовать, тогда замены производятся во всех строчках, а регулярные выражения FND и RPL могут быть ограничены любым другим символом, например **:** вместо **/**. Флаги уточняют место замены. Например, флаг **g** (global) требует произвести столько замен, сколько будет найдено слева направо соответствий выражению FND, а флаг **i** (case **i**nsensitive) указывает на нечувствительность выражения FND к регистру букв.

Таким образом, подстановка **s:^..::** выполняет удаление **^..** двух любых символов в начале строки (замену на «ничто»). Это отрежет ненужные префиксы **./** относительных путевых имен, выводимых как результат find(1). Подстановка **s:[^/.][^/]*$:+-- &:** заменяет подстроки, состоящие из любого символа, кроме слеша и точки **[^/.]**, и следующих за ним любого количества ***** любых, кроме слеша **[^/]**, символов вплоть до конца строки **$** на тоже самое значение **&**, перед которым поставлены символы **+--** Такое действие изобразит имена найденных find(1) файлов и каталогов как **+-- file**. Последняя подстановка **s:[^/]*/:| :g**

глобально заменит подстроки из любого количества * символов, кроме слеша [^/], заканчивающиеся слешем /, на вертикальную черту с тремя пробелами | , что заменит элементы путевого имени найденных файлов в нужное количество уровней иерархии дерева в выводе.

Листинг 5.66. Замена подстрок: дерево файлов и каталогов

```
bender@ubuntu:~$ cd /lib/terminfo
bender@ubuntu:/lib/terminfo$ find . | sort -f |
> sed -e 's:^..::' -e 's:[^/.][^/]*$:+-- &:' -e 's:[^/]*/:|   :g'
.
+-- a
|   +-- ansi
+-- c
|   +-- cons25
                ...         ...         ...         ...
+-- v
|   +-- vt100
|   +-- vt102
|   +-- vt220
|   +-- vt52
                ...         ...         ...         ...
+-- x
|   +-- xterm
                ...         ...         ...         ...
|   +-- xterm-vt220
|   +-- xterm-xfree86
```

Еще один типичный пример применения потокового редактора sed(1) проиллюстрирован в листинге 5.67, где во всех модулях некоторой программы на языке W:[python] имя модуля **module** нужно заменить его новым именем **Module**. Найденные командой find(1) имена модулей ***.py** проверяются командой grep(1) на предмет наличия в их тексте слова (**-w**) **module**, затем их имена (**-l**) отправляются потоковому редактору, который выполняет замену подстрок, замещая (**-i**) содержимое непосредственно обрабатываемого файла. Замены производятся в строках, содержащих слово **import**, при этом только подстроки **module**, слева **\<** и справа **\>** от которых находятся границы слов, т. е. только слова **module**, заменяются на **Module**.

Листинг 5.67. Массовая замена подстрок в файлах

```
bender@ubuntu:~$ find . -name '*.py' | xargs grep -wl module |
> xargs sed -i '/import/ s:\<module\>:Module:g'
```

В очень большом количестве случаев **sed(1)** применяется для редактирования конфигурационных файлов системы, например в сценариях установки и удаления пакетов программного обеспечения. В примере из листинга 5.68 показан небольшой сценарий интерпретатора, позволяющий «включать»/«выключать» параметры в конфигурационном файле **/etc/sysctl.conf**, используя удаление/установку символа комментария **#** в начало строк при помощи **sed(1)**. Для этого в трех инструкциях **sed(1)** используются две команды подстановки **s** и команда ветвления **t** (test). Первая подстановка **s/^#//** удаляет комментарий из начала строки, а вторая подстановка **s/^/#/** добавляет его в начало. Команда **t** тестирует результат выполнения первой инструкции подстановки и в случае успеха передает управление на конец списка инструкций. В результате выполнения инструкций либо комментарий удаляется из начала строки первой инструкцией, если он там присутствует (на чем список инструкций завершается), либо комментарий добавляется в начало строки второй инструкцией.

Листинг 5.68. Редактирование конфигурационных файлов

```
bender@ubuntu:~$ grep net.ipv4.ip_forward /etc/sysctl.conf
#net.ipv4.ip_forward=1
bender@ubuntu:~$ sudo sed -i '/net.ipv4.ip_forward/ s/^#//' /etc/sysctl.conf
bender@ubuntu:~$ grep net.ipv4.ip_forward /etc/sysctl.conf
net.ipv4.ip_forward=1

bender@ubuntu:~$ cat /usr/local/bin/toggle-comment
#!/bin/sh
[ -n "$1" -a -f "$2" ] &&
  sed -i -e "/$1/ s/^#//" -e t -e "/$1/ s/^/#/" $2 ||
  echo "Usage: toggle-comment RE file"
bender@ubuntu:~$ sudo toggle-comment net.ipv4.ip_forward /etc/sysctl.conf
bender@ubuntu:~$ grep net.ipv4.ip_forward /etc/sysctl.conf
#net.ipv4.ip_forward=1
```

Несмотря на простоту инструментальных средств обработки текста, таких как **awk(1)** и **sed(1)**, энтузиасты реализуют с их помощью (вкупе с управляющими **ESC**-последовательностями терминала и символами кодировки **utf-8**) даже простые игры, такие как крестики-нолики (**tic-tac-toe**), тетрис (**tetris**), **sokoban**, **arkanoid**, **2048** и даже (!) шахматы[1].

[1] См. https://habrahabr.ru/post/191006/.

Более подробное изложение концепций и практик применения регулярных выражений, sed(1) и awk(1), можно найти в книгах ISBN:[1-56592-225-5], ISBN:[5-93286-121-5], а полноценный курс программирования на языке командного интерпретатора — в ISBN:[5-7315-0114-9] и ISBN:[5-93286-029-4].

5.9. В заключение

Командный интерпретатор вместе с утилитами обработки текста формирует среду, позволяющую практически без ограничений решать разнообразные задачи автоматизации, что и находит широкое применение в виде соответствующих сценариев в операционной системе. Сценарии применяются практически повсеместно — при запуске и остановке служб операционной системы, при постинсталляционном конфигурировании установленных пакетов программного обеспечения, при компиляции и компоновке программ и т. д.

Располагая таким могучим инструментом, командный интерфейс перестает быть для пользователя просто интерактивным способом взаимодействия с операционной системой, а превращается в полноценное средство разработки решений его произвольных задач. Вместе с освоением языка командного интерпретатора сам пользователь шаг за шагом превращается из чужака и пришельца в нативного обитателя этой экосистемы, аборигена цифровых джунглей Linux. Для такого пользователя командный интерфейс больше не представляется рудиментом и тяжким наследием прошлого, а *дополняет* графический интерфейс до единого гармоничного целого.

Основной вид профессиональной деятельности такого пользователя не имеет особого значения. Фотографы и дизайнеры, аудио- и видеоинженеры, 3D-моделеры и инженеры САПР, типографские работники и прочие профессионалы находят свою прелесть в написании и использовании сценариев пакетной обработки своих фотографий, аудио- и видеоматериалов, моделей, чертежей и массы другой информации. Вручив ежедневную рутину командному интерпретатору, они переходят на следующий уровень развития, где в полную силу посвящают себя решению творческих задач.

Не желаете присоединиться?

Глава 6

Сетевая подсистема

Сетевая подсистема Linux организует сетевой обмен пользовательских приложений и, как следствие, сетевое взаимодействие самих пользователей. Часть сетевой подсистемы, выполняющаяся в режиме *ядра*, естественным образом ответственна за управление сетевыми *устройствами* ввода-вывода, но, кроме этого, на нее также возложены задачи маршрутизации и транспортировки пересылаемых данных, которые решаются при помощи соответствующих сетевых *протоколов*. Таким образом, именно ядерная часть сетевой подсистемы обеспечивает процессы средствами *сетевого* межпроцессного взаимодействия (network IPC).

Внеядерная часть сетевой подсистемы отвечает за реализацию сетевых служб, предоставляющих пользователям *прикладные* сетевые услуги, такие как передача файлов, обмен почтовыми сообщениями, удаленный доступ и т. д.

6.1. Сетевые интерфейсы, протоколы и сетевые сокеты

Непосредственное, *физическое* взаимодействие сетевых узлов через каналы связи между ними реализуется аппаратурой сетевых адаптеров. Сетевые адаптеры, как и любые другие устройства ввода-вывода, управляются соответствующими драйверами (листинг 6.1), реализуемыми в большинстве случаев в виде динамических модулей ядра.

Листинг 6.1. Драйвера сетевых устройств

```
lumpy@ubuntu:~$ lspci
                ...             ...             ...             ...
02:00.0 Network controller: Intel Corporation Centrino Advanced-N 6205 [Taylor Peak] (rev 34)
00:19.0 Ethernet controller: Intel Corporation 82579LM Gigabit Network Connection
(Lewisville) (rev 04)
lumpy@ubuntu:~$ lspci -ks 02:00.0
02:00.0 Network controller: Intel Corporation Centrino Advanced-N 6205 [Taylor Peak] (rev 34)
        Subsystem: Intel Corporation Centrino Advanced-N 6205 AGN
```

```
        Kernel driver in use: iwlwifi
        Kernel modules: iwlwifi
lumpy@ubuntu:~$ lspci -ks 02:00.0
00:19.0 Ethernet controller: Intel Corporation 82579LM Gigabit Network Connection
(Lewisville) (rev 04)
        Subsystem: Dell 82579LM Gigabit Network Connection (Lewisville)
        Kernel driver in use: e1000e
        Kernel modules: e1000e
lumpy@ubuntu:~$ modinfo iwlwifi e1000e | grep ^description
description:    Intel(R) Wireless WiFi driver for Linux
description:    Intel(R) PRO/1000 Network Driver
```

В отличие от несетевых устройств, большинство которых имеют специальный файл в каталоге **/dev**, сетевые устройства представляются в системе своими *интерфейсами*. Список доступных интерфейсов, их параметры и статистику можно получить при помощи «классической» UNIX-команды **ifconfig(8)** или специфичной для Linux команды **ip(8)** (листинги 6.2 и 6.3).

Листинг 6.2. Сетевые интерфейсы (UNIX ifconfig(8))

```
lumpy@ubuntu:~$ ifconfig -a
    wlp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
            inet 192.168.0.101  netmask 255.255.255.0  broadcast 192.168.0.255
            inet6 fe80::5f2d:68c3:2c09:9a18  prefixlen 64  scopeid 0x20<link>
            ether 08:11:96:29:19:70  txqueuelen 1000  (Ethernet)
                                  ...                ...

    lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
            inet 127.0.0.1  netmask 255.0.0.0
            inet6 ::1  prefixlen 128  scopeid 0x10<host>
                     ...            ...                ...

    eno1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
            ether 5c:26:0a:85:a9:1a  txqueuelen 1000  (Ethernet)
                     ...            ...                ...
```

Листинг 6.3. Сетевые интерфейсы (Linux ip(8))

```
lumpy@ubuntu:~$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN ... qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eno1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc ... state DOWN ... qlen 1000
    link/ether 5c:26:0a:85:a9:1a brd ff:ff:ff:ff:ff:ff
```

```
3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP ... qlen 1000
    link/ether 08:11:96:29:19:70 brd ff:ff:ff:ff:ff:ff

lumpy@ubuntu:~$ ip addr show dev wlp2s0
3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen
1000
    link/ether 08:11:96:29:19:70 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.101/24 brd 192.168.0.255 scope global dynamic noprefixroute wlp2s0
       valid_lft 7121sec preferred_lft 7121sec
    inet6 fe80::5f2d:68c3:2c09:9a18/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

За *логическое* взаимодействие (адресацию, маршрутизацию, обеспечение надеждой доставки и пр.) отвечают сетевые протоколы, тоже в большинстве случаев реализуемые соответствующими модулями ядра. Нужно отметить, что в примере из листинга 6.4 показан список *динамически* загруженных модулей, среди которых присутствует «нестандартный» TCP vegas, но нет IP, TCP, UDP и прочих «стандартных» протоколов стека TCP/IP. На текущий момент времени сложно вообразить применение Linux без подключения к IP-сети, поэтому модули стандартных протоколов TCP/IP стека скомпонованы в ядро *статически* и являются частью «стартового» *(см. разд. 4.1.1)* модуля.

Листинг 6.4. Драйвера сетевых протоколов

```
lumpy@ubuntu:~$ lsmod
Module                  Size  Used by
...                      ...     ...
iwldvm                229376  0
mac80211              786432  1 iwldvm
iwlwifi               290816  1 iwldvm
cfg80211              622592  3 iwldvm,iwlwifi,mac80211
...                      ...     ...                ...
e1000e                249856  0
ptp                    20480  1 e1000e
...                      ...     ...

lumpy@ubuntu:~$ modinfo tcp_vegas bnep mac80211 | grep ^description
description:    PTP clocks support
description:    Intel(R) Wireless WiFi Link AGN driver for Linux
description:    IEEE 802.11 subsystem
```

Доступ процессов к услугам ядерной части сетевой подсистемы реализует интерфейс *сетевых сокетов* **socket(7)**, являющихся основным (и единственным) средством сетевого взаимодействия процессов в Linux.

Разные *семейства* (address family) сокетов соответствуют различным стекам сетевых протоколов. Например, стек TCP/IP v4 представлен семейством **AF_INET**, см. **ip(7)**, стек TCP/IP v6 — семейством **AF_INET6**, см. **ipv6(7)**, и даже локальные (файловые) сокеты имеют собственное семейство — **AF_LOCAL**, см. **unix(7)**.

Для просмотра статистики по использованию сетевых сокетов используют «классическую» UNIX-команду **netstat(8)** или специфичную для Linux команду **ss(8)**. В листингах 6.5 и 6.6 иллюстрируется использование этих команд для вывода информации обо всех (**-a**, all) сокетах протоколов (**-u**, udp) UDP и (**-t**, tcp) TCP стека TCP/IP v4 (**-4**), порты и адреса которых выведены в числовом (**-n**, numeric) виде, а также изображены процессы (**-p**, process), их открывшие.

Листинг 6.5. Сетевые сокеты (UNIX netstat(8))

```
lumpy@ubuntu:~$ sudo netstat -4autpn
Активные соединения с Интернетом (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      864/sshd
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      697/cupsd
tcp        0      0 127.0.0.1:5432          0.0.0.0:*               LISTEN      1039/postgres
tcp        0      0 0.0.0.0:25              0.0.0.0:*               LISTEN      1250/master
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      638/systemd-resolve
...        ...    ...                       ...                     ...         ...
tcp        0      0 192.168.0.101:22        192.168.0.103:57622     ESTABLISHED 6152/sshd: lumpy [pr
...        ...    ...                       ...                     ...         ...
udp        0      0 127.0.0.53:53           0.0.0.0:*                           638/systemd-resolve
udp        0      0 192.168.0.101:68        0.0.0.0:*                           684/NetworkManager
                    ...                       ...                     ...
```

Сетевые сокеты идентифицируются парой адресов (собственным, local, и чужим[1], foreign), принятыми в их семействе. Например, для семейства TCP/IP адрес сокета состоит из (сетевого) IP-адреса и (транспортного) номера порта, причем нули имеют специальное — «неопределенное» значение. Так, для прослушивающего[2] (**LISTEN**) сокета **0.0.0.0** в собственном IP-адресе означает, что он принимает соединения, направленные на любой адрес любого сетевого интерфейса, а **0.0.0.0** в чужом адресе указывает на то, что взаимодействие еще не установлено. Для сокетов с установленным (**ESTABLISHED**) взаимодействием оба адреса имеют конкретные значения, определяющие участников взаимодействия, например **192.168.100.105:22** и **192.168.100.103: 57622**.

---

[1] Адрес удаленного приложения, с которым установлено соединение.

[2] Прослушивающие сокеты используются «серверными» приложениями, пассивно ожидающими входящие соединения с ними.

Листинг 6.6. Сетевые сокеты (Linux ss(8))

```
lumpy@ubuntu:~$ sudo ss -4atupn
Netid  State      Recv-Q  Send-Q  Local Address:Port       Peer Address:Port
udp    UNCONN     0       0       127.0.0.53%lo:53         0.0.0.0:*          users:(("systemd-resolve",pid=638,fd=12))
udp    UNCONN     0       0       192.168.0.101% wlp2s0:68 0.0.0.0:*          users:(("NetworkManager",pid=684,fd=19))
...    ...        ...     ...     ...                      ...
tcp    LISTEN     0       128     0.0.0.0:22               0.0.0.0:*          users:(("sshd",pid=864,fd=3))
tcp    LISTEN     0       5       0.0.0.1:631              0.0.0.0:*          users:(("cupsd",pid=697,fd=7))
tcp    LISTEN     0       128     127.0.0.1:5432           0.0.0.0:*          users:(("postgres",pid=1039,fd=3))
tcp    LISTEN     0       100     0.0.0.0:25               0.0.0.0:*          users:(("master",pid=1250,fd=13))
tcp    LISTEN     0       128     127.0.0.53%lo:53         0.0.0.0:*          users:(("systemd-resolve",pid=638,fd=13))
tcp    ESTAB      0       0       192.168.0.101:22         192.168.0.103:57622
                                                                               users:(("sshd",pid=6234,fd=4),("sshd",pid=6152,fd=4))
                          ...             ...                    ...                    ...
```

Из примеров листингов 6.5 и 6.6. видны 5 «слушающих» (**LISTEN**) сокетов TCP и 2 «несоединенных» (**UNCONN**) сокета UDP, открытых разными службами операционной системы. Например, 22-й порт TCP открыл сервер **sshd**, **PID = 864** службы удаленного доступа W:[SSH], а 5432-й порт TCP — сервер **postgres**, **PID = 1039** службы реляционной СУБД W:[SQL].

6.2. Конфигурирование сетевых интерфейсов и протоколов

6.2.1. Ручное конфигурирование

Для функционирования разных стеков протоколов сетевым интерфейсам должны быть предварительно назначены корректные сетевые адреса и сконфигурированы прочие параметры, что может быть выполнено вручную администратором или автоматически специальными службами этих стеков.

Ручное назначение сетевых адресов стека TCP/IP выполняется при помощи команды **ifconfig(8)** или **ip(8)**, а простейшая диагностика — при помощи команды **ping(1)**, как проиллюстрировано в листинге 6.7.

Листинг 6.7. Ручное конфигурирование сетевых интерфейсов

```
lumpy@ubuntu:~$ sudo ifconfig eno1 10.0.0.10 up
lumpy@ubuntu:~$ sudo ifconfig eno1
eno1:   flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.10  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::66f6:6415:7455:6a0f  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:c0:67:8f  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
```

```
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 94  bytes 14932 (14.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
lumpy@ubuntu:~$ ping -c 1 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data.
64 bytes from 10.0.0.10: icmp_seq=1 ttl=64 time=0.032 ms

--- 10.0.0.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.032/0.032/0.032/0.000 ms

lumpy@ubuntu:~$ sudo ip address add 172.16.16.172/16 dev eno1
lumpy@ubuntu:~$ ip address show dev eno1
3: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
qlen 1000
    link/ether 08:00:27:c0:67:8f brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.10/8 brd 10.255.255.255 scope global eno1
       valid_lft forever preferred_lft forever
    inet 172.16.16.172/16 scope global eno1
       valid_lft forever preferred_lft forever

lumpy@ubuntu:~$ ping -c 1 172.16.16.172
PING 172.16.16.172 (172.16.16.172) 56(84) bytes of data.
64 bytes from 172.16.16.172: icmp_seq=1 ttl=64 time=1.27 ms

--- 172.16.16.172 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.270/1.270/1.270/0.000 ms
```

Просмотр таблиц маршрутизации и ручное конфигурирование IP-маршрутов выполняется посредством команды **route(8)** или **ip(8)**, а простейшая диагностика — при помощи команды **traceroute(1)**. В примере из листинга 6.8 показана процедура ручного добавления маршрута «по умолчанию» ❶ через интернет-шлюз **10.0.0.1** ❷ с последующей диагностикой ❸ доступности узлов за ним ❹.

Листинг 6.8. Ручное конфигурирование таблицы маршрутизации

```
lumpy@ubuntu:~$ sudo ip route add 0.0.0.0/0 ❶ via 10.0.0.1 ❷
lumpy@ubuntu:~$ route -n
```

```
Таблица маршрутизации ядра протокола IP
Destination     Gateway         Genmask         Flags Metric Ref Use  Iface
0.0.0.0 ❶       10.0.0.1    ❷ 0.0.0.0         UG    0      0        0 eno1
10.0.0.0        0.0.0.0         255.0.0.0       U     0      0        0 eno1
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 eno1
172.16.0.0      0.0.0.0         255.255.0.0     U     0      0        0 eno1

lumpy@ubuntu:~$ ip route show
default ❶ via 10.0.0.1 ❷ dev eno1 proto static
10.0.0.0/8 dev eno1 proto kernel  scope link  src 10.0.0.1
172.16.0.0/16 dev eno1 proto kernel  scope link  src 172.16.0.1
169.254.0.0/16 dev eno1 scope link metric 1000

❸ lumpy@ubuntu:~$ traceroute -m 50 bad.horse
traceroute to bad.horse (162.252.205.157), 30 hops max, 60 byte packets
❹  1  10.0.0.1 (10.0.0.1)  1.025 ms  1.080 ms  1.236 ms
                    ...          ...          ...          ...
   22  bad.horse (162.252.205.130)  191.988 ms  191.997 ms  178.848 ms
   23  bad.horse (162.252.205.131)  190.805 ms  195.160 ms  194.807 ms
   24  bad.horse (162.252.205.132)  199.199 ms  187.907 ms  201.063 ms
   25  bad.horse (162.252.205.133)  209.814 ms  203.633 ms  199.866 ms
   26  he.rides.across.the.nation (162.252.205.134)  209.300 ms  204.059 ms  211.089 ms
   27  the.thoroughbred.of.sin (162.252.205.135)  211.454 ms  208.207 ms  212.280 ms
   28  he.got.the.application (162.252.205.136)  208.660 ms  222.452 ms  210.522 ms
   29  that.you.just.sent.in (162.252.205.137)  215.037 ms  223.553 ms  223.043 ms
   30  it.needs.evaluation (162.252.205.138)  229.889 ms  231.036 ms  234.139 ms
   31  so.let.the.games.begin (162.252.205.139)  226.369 ms  230.170 ms  223.952 ms
   32  a.heinous.crime (162.252.205.140)  232.431 ms  231.814 ms  240.990 ms
   33  a.show.of.force (162.252.205.141)  241.706 ms  233.070 ms  272.973 ms
   34  a.murder.would.be.nice.of.course (162.252.205.142)  264.821 ms  239.704 ms  247.930 ms
   35  bad.horse (162.252.205.143)  250.081 ms  244.279 ms  254.147 ms
   36  bad.horse (162.252.205.144)  247.903 ms  258.675 ms  257.366 ms
   37  bad.horse (162.252.205.145)  261.103 ms  253.861 ms  261.307 ms
   38  he-s.bad (162.252.205.146)  267.135 ms  266.860 ms  258.031 ms
   39  the.evil.league.of.evil (162.252.205.147)  262.974 ms  262.773 ms  275.656 ms
   40  is.watching.so.beware (162.252.205.148)  278.567 ms  276.382 ms  277.577 ms
   41  the.grade.that.you.receive (162.252.205.149)  276.158 ms  283.299 ms  284.268 ms
   42  will.be.your.last.we.swear (162.252.205.150)  286.575 ms  286.470 ms  278.213 ms
   43  so.make.the.bad.horse.gleeful (162.252.205.151)  283.040 ms  292.280 ms  290.042 ms
   44  or.he-ll.make.you.his.mare (162.252.205.152)  300.872 ms  299.806 ms  289.688 ms
   45  o_o (162.252.205.153)  294.548 ms  295.458 ms  295.030 ms
```

```
46 you-re.saddled.up (162.252.205.154)  310.332 ms  308.226 ms  307.612 ms
47 there-s.no.recourse (162.252.205.155)  316.779 ms  315.848 ms  311.799 ms
48 it-s.hi-ho.silver (162.252.205.156)  320.175 ms  311.570 ms  319.072 ms
49 signed.bad.horse (162.252.205.157)  313.125 ms  311.095 ms  321.902 ms
```

6.2.2. Автоматическое конфигурирование

За автоматическое конфигурирование сетевых интерфейсов отвечает менеджер сетевых подключений — системная служба **networkmanager(8)**, отслеживающая физическую активацию сетевых адаптеров (подключение сетевого кабеля Ethernet или подключения к сети Wi-Fi) и взаимодействующая[1] с другими службами, например со службой **wpa_supplicant(8)** (для ассоциации с Wi-Fi точками доступа и аутентификации) или с локальной службой **W:[DNS] systemd-resolved(1)** *(см. разд. 6.3)*. Кроме этого, менеджер сетевых подключений может запускать «подчиненные» службы, например **W:[DHCP]**-клиента **dhclient(8)** или **dhcpcd(8)** для автоматического получения IP-адреса, простейший локальный DNS-сервер **dnsmasq(8)** и т. д.

Для взаимодействия с менеджером сетевых подключений предназначены команда **nmcli(1)** (см. также **nmcli-examples(7)**), TUI-приложения **nmtui(1)**, **nmtui-edit(1)**, **nmtui-connect(1)** и GUI-приложения **nm-applet(1)**, **nm-connection-editor(1)**, позволяющие опрашивать его состояние и управлять его действиями.

Листинг 6.9. Конфигурирование сетевых интерфейсов (автоматически)

```
lumpy@ubuntu:~$ nmcli dev
DEVICE   TYPE      STATE         CONNECTION
wlp2s0   wifi      подключено    474+
eno1     ethernet  отключено     --
lo       loopback  не настроенно --

lumpy@ubuntu:~$ nmcli dev show eno1
GENERAL.DEVICE:                         eno1
GENERAL.TYPE:                           ethernet
GENERAL.HWADDR:                         08:00:27:C0:67:8F
GENERAL.MTU:                            1500
GENERAL.STATE:                          30 (отключено)
GENERAL.CONNECTION:                     --
GENERAL.CON-PATH:                       --
WIRED-PROPERTIES.CARRIER:               вкл.
```

[1] При помощи механизмов службы **W:[D-Bus]**, требующей отдельного разговора, выходящего за рамки этой книги.

```
lumpy@ubuntu:~$ nmcli conn
NAME                UUID                                  TYPE      DEVICE
464+                57cd20ce-098e-3b31-acd6-f50fd2e4db41  wifi      wlp2s0
lumpy@ubuntu:~$ sudo nmcli conn add type ethernet ifname eno1
Соединение «ethernet-eno1» (eadac6e2-eb71-43ee-9b34-86c2910a382c) добавлено.

lumpy@ubuntu:~$ nmcli conn
NAME                UUID                                  TYPE      DEVICE
464+                57cd20ce-098e-3b31-acd6-f50fd2e4db41  wifi      wlp2s0
ethernet-eno1       eadac6e2-eb71-43ee-9b34-86c2910a382c  ethernet  eno1

lumpy@ubuntu:~$ sudo nmcli conn up ethernet-eno1
Соединение успешно активировано (адрес действующего D-Bus:
/org/freedesktop/NetworkManager/ActiveConnection/6)

lumpy@ubuntu:~$ nmcli dev show eno1
GENERAL.DEVICE:                         eno1
GENERAL.TYPE:                           ethernet
GENERAL.HWADDR:                         08:00:27:C0:67:8F
...                     ...                     ...             ...
IP4.ADDRESS[1]:                         10.0.2.4/24 ❶
IP4.GATEWAY:                            10.0.2.1 ❷
IP4.ROUTE[1]:                           dst = 0.0.0.0/0, nh = 10.0.2.1, mt = 101
IP4.ROUTE[2]:                           dst = 10.0.2.0/24, nh = 0.0.0.0, mt = 10
IP4.DNS[1]:                             10.0.2.1

lumpy@ubuntu:~$ ip a show dev eno1
① 3: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP ... qlen 1000
    link/ether 08:00:27:c0:67:8f brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.4/24 ❶ brd 10.0.2.255 scope global dynamic noprefixroute eno1
       valid_lft 954sec preferred_lft 954sec
    inet6 fe80::9eca:df0a:5401:d34f/64 scope link noprefixroute
       valid_lft forever preferred_lft forever

lumpy@ubuntu:~$ ip route show
② default via 10.0.2.1 ❷ dev enp0s8 proto dhcp metric 101
    10.0.2.0/24 dev enp0s8 proto kernel scope link src 10.0.2.4 metric 101
    169.254.0.0/16 dev enp0s3 scope link metric 1000
```

В листинге 6.9 проиллюстрирован результат работы менеджера подключений при подключении сетевого интерфейса **eno1**. Полученные при помощи DHCP-клиента

конфигурационные параметры были активированы автоматически: IP-адреса и маска ❶ назначены на интерфейс ①, а шлюз «по умолчанию» ❷ задан в соответствующем маршруте ②.

6.3. Служба имен и DNS/mDNS-резолверы

Основными идентификаторами сетевого взаимодействия в стеке протоколов TCP/IP являются числа — IP-*адреса* узлов и *номера* портов TCP/UDP, «человеческое» использование которых довольно неудобно[1]. Использование строковых *имен* для узлов и портов приводит к необходимости отображать «человеческие» имена в «протокольные» числа и обратно, что возложено на *службу имен* (name service). Как указывалось ранее, служба имен вообще предназначается для организации доступа приложений к свойствам каталогизируемых сущностей по их имени: к UID пользователя по имени его учетной записи, к IP-адресу по имени узла, к номеру порта TCP/UDP по имени сетевой службы, использующей его, и т. д.

Отображение имен сущностей на их свойства зачастую выполняется различными способами в разных каталогах, что определяется конфигурацией коммутатора службы имен **nsswitch.conf(5)** (name service switch **conf**iguration) и наличием ее соответствующих модулей **libnss_*.so.?**.

В листинге 6.10 иллюстрируется такая конфигурация отображения имен *узлов* **hosts** ⓪, которая использует сначала файловую таблицу ① **/etc/hosts** — см. **hosts(5)**, а затем — службы **W:[mDNS]** ② и **W:[DNS]** ③.

Аналогично, номера портов сетевых *служб* **services** отображаются сначала с использованием файла индексированной базы данных (соответствующий модуль **libnss_db.so.2** оказался не установлен), а затем с использованием файловой таблицы **/etc/services** — см. **services(5)**.

Листинг 6.10. Служба имен и ее модули

```
lumpy@ubuntu:~$ grep hosts /etc/nsswitch.conf
⓪             ①↴  ②↴                          ③↴
hosts:          files mdns4_minimal [NOTFOUND=return] dns
lumpy@ubuntu:~$ grep services /etc/nsswitch.conf
services:       db files
lumpy@ubuntu:~$ find /lib/ -name 'libnss_*'
                  ...          ...             ...             ...
```

[1] Если с 32-битным IPv4-адресом еще можно было совладать, то 128-битный адрес IPv6 не оставляет человеку практически никаких шансов.

❸ /lib/x86_64-linux-gnu/libnss_dns.so.2
❶ /lib/x86_64-linux-gnu/libnss_files.so.2
... ...
❷ /lib/x86_64-linux-gnu/libnss_mdns4_minimal.so.2
... ...

Файловые таблицы имен **/etc/hosts** и **/etc/services** имеют тривиальный формат ❶❷, сопоставляющий имена узлов и сервисов — их IP-адресам и портам протоколов TCP и UDP, что проиллюстрировано в листинге 6.11. Утилита службы имен **getent(1)**, позволяющая выбирать указанную сущность по ее типу и имени, используется в качестве диагностики ❸ коммутатора службы имен и его модулей.

Листинг 6.11. Файловые таблицы имен

```
lumpy@ubuntu:~$ cat /etc/hosts
❶ 127.0.0.1     localhost
  127.0.1.1     ubuntu

  # The following lines are desirable for IPv6 capable hosts
  ::1     ip6-localhost ip6-loopback
  fe00::0 ip6-localnet
  ff00::0 ip6-mcastprefix
  ff02::1 ip6-allnodes
  ff02::2 ip6-allrouters

  lumpy@ubuntu:~$ grep http /etc/services
  # Updated from https://www.iana.org/assignments/service-names-port-numbers/service-names-
  port-numbers.xhtml .
❷ http            80/tcp          www             # WorldWideWeb HTTP
  https           443/tcp                         # http protocol over TLS/SSL
  http-alt        8080/tcp        webcache        # WWW caching service
  http-alt        8080/udp

❸ lumpy@ubuntu:~$ getent hosts ubuntu
  127.0.1.1       ubuntu
  lumpy@ubuntu:~$ getent services 53/udp
  domain                  53/tcp
```

Соответствия IP-адресов именам «серверных» узлов, например публичных Web-, почтовых и прочих серверов, обычно регистрируются их администраторами в «таблицах» на *ответственных*[1] (authoritative) серверах службы DNS. Для доступа

[1] Подробнее о том, как устроена система DNS, можно узнать здесь: **https://tiny.cc/niwqgz**.

к ним соответствующий модуль службы имен (см. ③, листинг 6.10) использует стандартный DNS-клиент (он же **resolver(3)**, DNS-резолвер), в конфигурационном файле **resolv.conf(5)** которого при *статических* настройках указываются IP-адреса ближайших *кэширующих* DNS-серверов, например серверов провайдера услуг Интернета.

Однако в примере из листинга 6.12 показано, что в качестве кэширующего DNS-сервера указан адрес `127.0.0.53` некоторой локальной службы DNS. Такой подход позволяет *динамически* управлять настройками (не меняя каждый раз файл **resolv.conf(5)**) при реконфигурировании сетевых подключений, например при присоединении к другой Wi-Fi-сети. В этом случае именно менеджер сетевых подключений *(см. разд. 6.2.2)* сообщает новые DNS-параметры нового активированного подключения этой самой локальной службе DNS (которой на проверку оказывается **systemd-resolved(1)**).

Для диагностики DNS-модуля службы имен (равно как и любого другого ее модуля) используется команда **getent(8)**, а для непосредственной диагностики DNS-серверов — команда **host(1)**.

Листинг 6.12. DNS-клиент

```
lumpy@ubuntu:~$ cat /etc/resolv.conf
# This file is managed by man:systemd-resolved(8). Do not edit.
...          ...          ...          ...
# See man:systemd-resolved.service(8) for details about the supported modes of
# operation for /etc/resolv.conf.

nameserver 127.0.0.53
options edns0

lumpy@ubuntu:~$ sudo ss -4autpn sport = 53
Netid  State  Recv-Q  Send-Q  Local Address:Port  Peer Address:Port
udp    UNCONN 0       0       127.0.0.53%lo:53    0.0.0.0:*  users:(("systemd-resolve",pid=5932,...))
tcp    LISTEN 0       128     127.0.0.53%lo:53    0.0.0.0:*  users:(("systemd-resolve",pid=5932,...))

lumpy@ubuntu:~$ getent hosts bhv.ru
91.244.162.162  bhv.ru

lumpy@ubuntu:~$ host bhv.ru
bhv.ru has address 91.244.162.162
bhv.ru mail is handled by 50 relay2.peterlink.ru.
bhv.ru mail is handled by 30 relay1.peterlink.ru.

lumpy@ubuntu:~$ host 8.8.8.8
8.8.8.8.in-addr.arpa domain name pointer dns.google.
```

Сетевые устройства (принтеры, камеры, видеорегистраторы и пр.) и «клиентские» узлы локальных сетей, динамически получающие случайные IP-адреса при помощи DHCP, на ответственных серверах DNS почти никогда не регистрируются, а использование их имен в локальной сети (в «домене» **.local**) становится возможным благодаря службе W:[mDNS].

Серверы mDNS запускаются на каждом «клиентском» узле и регистрируют у себя соответствия собственных IP-адресов своему имени, а затем используют многоадресную (multicast) рассылку стандартных запросов DNS для получения информации друг у друга. Сервером mDNS, как показано в примере из листинга 6.13, является **avahi-daemon(8)**, реализующий еще и службу W:[DNS-SD] (DNS service discovery), которая позволяет узлам локальной сети обнаруживать (discovery) услуги (service), предоставляемые другими узлами. При помощи **avahi-browse(1)** проиллюстрирован список всех (**-a**, all) имен ❶ и типов ❷ услуг, объявленных узлами сети и сохраненных в локальном кэше (**-c**, cache), а также результаты (**-r**, resolve) запросов на получение информации об услугах.

Листинг 6.13. mDNS/DNS-SD-клиент

```
lumpy@ubuntu:~$ sudo ss -4autpn sport = :mdns
sudo ss -4autpn sport = :mdns
Netid   State   Recv-Q  Send-Q  Local Address:Port  Peer Address:Port
udp     UNCONN  0       0       0.0.0.0:5353        0.0.0.0:*       users:(("avahi-daemon",pid=678,...))

lumpy@ubuntu:~$ avahi-browse -arcl
+   eth0 IPv4 HP LaserJet 700 M712 [4C6BF5] ❶ ❷ UNIX Printer                  local
+   eth0 IPv4 AXIS 211M - 00408C81D401          RTSP Realtime Streaming Server local
+   eth0 IPv4 NVR(SMB)                          Microsoft Windows Network      local
+   eth0 IPv4 NVR(NFS)                          Network File System            local
=   eth0 IPv4 HP LaserJet 700 M712 [4C6BF5]     UNIX Printer                   local
   hostname = [NPI4C6BF5.local]
   address = [192.168.17.68]
   port = [515]
   txt = ["Scan=F" "UUID=56ff35dd-1065-4e5e-9385-3c26b8946b79" "Color=F" "Duplex=T" "Binary=T"
"Transparent=T" "note=" "adminurl=http://NPI4C6BF5.local." "priority=40" "usb_MDL=HP LaserJet
700 M712" "usb_MFG=Hewlett-Packard" "product=(HP LaserJet 700 M712)" "ty=HP LaserJet 700 M712"
"URF=V1.1,CP99,RS600,MT1-2-3-5-12,W8,PQ4,IS20-21-22-23,DM1,OB1" "pdl=application/postscript"
"rp=BINPS" "qtotal=4" "txtvers=1"]
=   eth0 IPv4 AXIS 211M - 00408C81D401          RTSP Realtime Streaming Server local
   hostname = [axis-00408c81d401.local]
   address = [192.168.17.142]
```

```
   port = [554]
   txt = ["path=mpeg4/1/media.amp"]
=  eth0 IPv4 NVR(SMB)                          Microsoft Windows Network      local
   hostname = [NVR.local]
   address = [192.168.17.90]
   port = [445]
   txt = []
=  eth0 IPv4 NVR(NFS)                          Network File System            local
   hostname = [NVR.local]
   address = [192.168.17.90]
   port = [2049]
   txt = []

lumpy@ubuntu:~$ avahi-resolve --name NVR.local NPI4C6BF5.local axis-00408c81d401.local
NVR.local                192.168.17.90
NPI4C6BF5.local          192.168.17.68
axis-00408c81d401.local 192.168.17.142

lumpy@ubuntu:~$ getent hosts NVR.local NPI4C6BF5.local axis-00408c81d401.local
192.168.17.90    NVR.local
192.168.17.68    NPI4C6BF5.local
192.168.17.142   axis-00408c81d401.local
```

Для диагностики mDNS-модуля службы имен неизменно используется команда **getent(8)**, а для непосредственной диагностики mDNS-сервера **avahi-daemon(8)** — специальная команда **avahi-resolve(1)**.

6.4. Сетевые службы

6.4.1. Служба SSH

Служба **W:[SSH]** предназначена[1] для организации *безопасного* (secure) доступа к сеансу командного интерпретатора (**shell**) удаленных сетевых узлов. Изначально разрабатывалась как замена *небезопасным* R-утилитам **W:[Rlogin]**, **W:[Rsh]** и протоколу сетевого алфавитно-цифрового терминала **W:[telnet]**.

---

[1] О SSH протоколе с картинками можно узнать здесь: **https://tiny.cc/ocxqgz**.

При сетевом взаимодействии в «открытой» публичной сети, такой как Интернет, «безопасность» обычно понимают как *конфиденциальность* (плюс *целостность*) передаваемых данных и *аутентичность* (подлинность) взаимодействующих сторон.

Конфиденциальность данных, т. е. их недоступность некоторой третьей стороне, не участвующей во взаимодействии, обеспечивается в протоколе SSH при помощи *симметричного* шифрования с помощью общего *сеансового* ключа. Симметричные алгоритмы шифрования используют для зашифровки и расшифровки информации один и тот же ключ, поэтому конфиденциальность целиком и полностью сводится к *секретности* ключа, т. е. его недоступности третьей стороне.

Сеансовый ключ устанавливается обеими взаимодействующими сторонами при помощи *асимметричного* алгоритма открытого согласования ключей (**W:[протокол Диффи — Хеллмана]**), использующего две пары дополнительных ключей. Обе стороны взаимодействия случайным образом выбирают *закрытые* ключи и на их основе вычисляют парные *открытые* ключи, которыми публично обмениваются. Общий (сеансовый) ключ *вычисляется* каждой стороной на основе *своего закрытого* ключа и *чужого открытого* ключа, что не может повторить третья сторона, т. к. может подслушать только передачу открытых ключей и не владеет[1] закрытыми ключами участников взаимодействия.

При активном вмешательстве третьей стороны во взаимодействие путем перехвата и подмены сообщений согласования ключей злоумышленник может выдавать себя за другую сторону каждому из участников взаимодействия, став посредником, — см. **W:[MITM]** (**man in the middle**). В этом случае взаимодействующие стороны согласуют свои сеансовые ключи со злоумышленником (!), предполагая, что согласовали их с подлинным участником взаимодействия. Как следствие, все передаваемые данные «естественным» образом станут доступными злоумышленнику, причем наличие посредника никак не будет обнаружено.

Обеспечение подлинности (аутентичности) взаимодействующих сторон в протоколе SSH основывается на *аутентификации сервера* клиентом при помощи *асимметричных* алгоритмов цифровой подписи с использованием закрытого и открытого ключей *сервера*. Закрытый ключ используется для подписывания сообщений, а парный ему открытый ключ — для проверки подписи, корректность которой удостоверяет в том, что сообщение было сгенерировано подлинным владельцем закрытого ключа.

В сообщение протокола Диффи — Хеллмана сервер добавляет свой открытый ключ цифровой подписи (так называемый host key), а само сообщение подписывает закрытым ключом. Клиент извлекает этот открытый ключ из сообщения и

[1] Обратная задача вычисления закрытых ключей на основе открытых ключей практически не решаема, на чем и основывается вся асимметричная криптография.

с помощью проверки корректности его цифровой подписи удостоверяется в подлинности владельца вложенного ключа. Остается только убедиться, что владельцем вложенного ключа и является целевой сервер.

В примере из листинга 6.14 иллюстрируется первое присоединение к SSH-серверу **grex.org (162.202.67.158)**, в результате чего был получен открытый ключ алгоритма **W:[ECDSA]**, который, *возможно*, действительно принадлежит этому серверу, а не злоумышленнику посередине соединения. Единственный способ это проверить — *заранее знать* действительный открытый ключ SSH-сервера **grex.org** и побитно *сверить* его с присланным ключом, что довольно затруднительно сделать человеку. Поэтому на практике сверяют короткие хэш-суммы действительного и присланного ключей, называемые «отпечатками пальца» (fingerprint), совпадение которых гарантирует совпадение[1] ключей. Хэш-суммы открытых ключей SSH-серверов заранее известны и открыто публикуются, например для **grex.org** — на Web-странице **http://grex.cyberspace.org/faq.xhtml#sshfinger**. После ручной сверки ключа ❶ при первом подключении он сохраняется в файле **~/.ssh/known_hosts** для автоматической сверки при последующих подключениях.

Листинг 6.14. Первое присоединение к SSH-серверу

```
lumpy@ubuntu:~$ ssh jake@grex.org
The authenticity of host 'grex.org (75.61.90.157)' can't be established.
ECDSA key fingerprint is SHA256:pMO3fe6UTyqtqzUMq5SmTmH5tqUuN9WdvlwdpcEJhSU.
Are Are you sure you want to continue connecting (yes/no/[fingerprint])? yes ↵ ❶
Warning: Permanently added 'grex.org,75.61.90.157' (ECDSA) to the list of known hosts.
jake@grex.org's password: P@s$w0rd ↵ ❷
                    ...          ...          ...
grex$ uname -a ↵
OpenBSD grex.org 6.3 GENERIC#9 i386
grex$ whoami ↵
jake
grex$ w ↵
8:44PM  up 41 days,  3:15, 8 users, load averages: 1.44, 1.46, 1.60
USER    TTY FROM              LOGIN@  IDLE WHAT
cross    p0 166.84.136.80    01Nov19 2days -bash
mcd      p1 37.59.109.123    12Nov19 9days -zsh
fernan   pa 24.78.62.91      11Nov1911days -ksh
cross    pi 166.84.136.80    Tue04PM 2days -bash
```

[1] При этом подобрать другой ключ с такой же хэш-суммой практически невозможно.

```
mattias  pq 81.233.210.67    18Oct19     0 /suid/bin/party
pbbl     ps 24.59.50.208      7:17PM    41 alpine
jake ❸ pB 93.100.207.82 ❹ 8:44PM     0 w
lerxst   pG 47.50.84.206     03Nov1920days screen -RDD
grex$ tty
/dev/ttypB ❸
grex$ logout ↵
Connection to grex.org closed.
lumpy@ubuntu:~$
```

После успешного установления соединения SSH пользовательский сеанс аутентифицируется одним из способов, например с помощью пароля ❷, а затем в интерактивном режиме запускается начальный командный интерпретатор аутентифицированного пользователя. При этом на стороне сервера используется псевдотерминал ❸ и эмулируется терминальный вход в систему *(см. разд. 2.2.1)*, считающийся сетевым ❹.

Листинг 6.15. Выполнение отдельной команды

```
lumpy@ubuntu:~$ ssh jake@grex.org uptime
jake@grex.org's password: P@s$W0rd ↵
8:47PM  up 41 days,  3:17, 7 users, load averages: 2.34, 1.65, 1.64
lumpy@ubuntu:~$
```

Кроме интерактивного сетевого входа, SSH позволяет удаленно выполнять отдельные команды (см. листинг 6.15), при этом псевдотерминал не используется, а терминальный вход не эмулируется. Вместо этого стандартные потоки ввода-вывода **STDIN**, **STDOUT** и **STDERR** выполняемой команды просто перенаправляются через сетевое соединение ssh-клиенту. Это зачастую используется для удаленного копирования файлов. Например, в листинге 6.16 при помощи архиватора **tar(1)** создается сжатый архив каталога **/usr/src/sys** на *удаленном* узле **grex.org**, а результат перенаправляется в *локальный* файл **openbsd-kernel-source.tgz**.

Листинг 6.16. Копирование при помощи ssh

```
lumpy@ubuntu:~$ ssh jake@grex.org tar czf - /usr/src/sys > openbsd-kernel-source.tgz
jake@grex.org's password: P@s$W0rd ↵
tar: Removing leading / from absolute path names in the archive
                                ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰ ⑱
lumpy@ubuntu:~$
```

Аутентификация пользовательского сеанса по паролю имеет некоторое неудобство — необходимость вводить пароль заново при каждом новом соединении SSH. Кроме того, пароль передается внутри соединения SSH, поэтому существует угроза подбора пароля злоумышленником. Более прогрессивный способ аутентификации пользователя основан на использовании асимметричных алгоритмов цифровой подписи (аналогично аутентификации сервера в протоколе Диффи — Хелмана) и ключей *пользователя*. Для этого открытый ключ пользователя единожды регистрируется на сервере, а при аутентификации подписывается его закрытым ключом и высылается серверу повторно. Сервер в свою очередь проверяет цифровую подпись присланного ключа и удостоверяется в подлинности его владельца, а побитное сравнение с заранее зарегистрированным ключом пользователя указывает на то, что владелец присланного ключа и есть целевой пользователь.

В листинге 6.17 иллюстрируется процедура применения ключей аутентификации пользователя. Сначала при помощи команды **ssh-keygen(1)** генерируется закрытый (private) W:[RSA]-ключ в локальном файле **~/.ssh/id_rsa** и парный ему открытый (public) ключ в локальном файле **~/.ssh/id_rsa.pub**. Закрытый ключ защищается (шифруется) от несанкционированного использования парольной фразой (passphrase) ⓪, которая в отличие от пароля никогда по сети не передается, а лишь обеспечивает доступ к самому ключу. Нужно заметить, что использование пустой парольной фразы (как в примере) потенциально *небезопасно*, т. к. в случае кражи или разглашения ключей ими может воспользоваться любая третья сторона.

Затем при помощи команды **ssh-copy-id(1)** открытый ключ регистрируются на уделенном сервере **grex.org** в файле **~/.ssh/authorized_keys**, что происходит с предъявлением пароля ❷. Последующие SSH-соединения ❸ аутентифицируются парой ключей, в результате отпадает необходимость вводить пароль[1].

Листинг 6.17. Доступ по ключу

```
❶ lumpy@ubuntu:~$ ssh-keygen
  Generating public/private rsa key pair.
  Enter file in which to save the key (/home/lumpy/.ssh/id_rsa): ↵
  Enter passphrase (empty for no passphrase): ↵ ⓪
  Enter same passphrase again: ↵ ⓪
  Your identification has been saved in /home/lumpy/.ssh/id_rsa.
  Your public key has been saved in /home/lumpy/.ssh/id_rsa.pub.
  The key fingerprint is:
  SHA256:ztlK3kUagoDuY4sUxqFMSZFnqVxQQhPOQ4yBtDMov6w lumpy@ubuntu
                    ...          ...          ...          ...
```

[1] Появляется необходимость вводить парольную фразу, но в примере она (непозволительно) пустая.

```
❷ lumpy@ubuntu:~$ ssh-copy-id jake@grex.org
  /usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/lumpy/.ssh/id_rsa.pub"
  /usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that
  are already installed
  /usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is
  to install the new keys
🗝 jake@grex.org's password: P@s$w0rd ↵

  Number of key(s) added: 1

  Now try logging into the machine, with:   "ssh 'jake@grex.org'"
  and check to make sure that only the key(s) you wanted were added.

❸ lumpy@ubuntu:~$ ssh jake@grex.org ls
! OPENME.txz
  README.gz
❸ lumpy@ubuntu:~$ ssh jake@grex.org zcat README.gz
  А в файле OPENME.txz находится кое-что полезное ;)
❸ lumpy@ubuntu:~$ ssh jake@grex.org file OPENME.txz
  OPENME.txz: XZ compressed data
```

Использовать зашифрованные парольными фразами ключи и одновременно не вводить парольную фразу при каждом подключении позволяет SSH-агент **ssh-agent(1)**, который удерживает в оперативной памяти расшифрованные единожды закрытые ключи пользователя и генерирует с их помощью цифровые подписи по запросу. Листинг 6.18 иллюстрирует аутентификацию пользователя по ключу, защищенному парольной фразой ❶, а затем передачу этого расшифрованного ключа агенту при помощи команды **ssh-add(1)**, что позволяет избавиться от необходимости ввода парольной фразы ключа все время, пока запущен процесс **ssh-agent** (обычно — до окончания сеанса). Наличие SSH-агента, запущенного в сеансе пользователя, обнаруживают две переменные окружения — **SSH_AGENT_PID** и **SSH_AUTH_SOCK**, содержащие соответственно PID агента и имя локального сокета для связи с ним.

Листинг 6.18. SSH-агент

```
🔒 lumpy@ubuntu:~$ ssh jake@grex.org file /usr/bin/ssh
🗝 Enter passphrase for key '/home/lumpy/.ssh/id_rsa': ... ... ... ... ... ↵ ❶
  /usr/bin/ssh: ELF 32-bit LSB shared object, Intel 80386, version 1

  lumpy@ubuntu:~$ ssh-add ❷
🗝 Enter passphrase for /home/lumpy/.ssh/id_rsa: ... ... ... ... ... ↵
  Identity added: /home/lumpy/.ssh/id_rsa (/home/lumpy/.ssh/id_rsa)
```

```
lumpy@ubuntu:~$ ssh jake@grex.org ldd /usr/bin/ssh
/usr/bin/ssh:
        Start    End      Type  Open Ref GrpRef Name
        17645000 37681000 exe   1    0   0      /usr/bin/ssh
        0b348000 2b3c5000 rlib  0    1   0      /usr/lib/libcrypto.so.43.1
        06757000 2675c000 rlib  0    1   0      /usr/lib/libutil.so.13.0
        00799000 207a1000 rlib  0    1   0      /usr/lib/libz.so.5.0
        0f668000 2f698000 rlib  0    1   0      /usr/lib/libc.so.92.3
        0ab98000 0ab98000 ld.so 0    1   0      /usr/libexec/ld.so

lumpy@ubuntu:~$ env | grep SSH
SSH_AGENT_PID=21655
SSH_AUTH_SOCK=/tmp/ssh-Eehhsbx21654/agent.21654

lumpy@ubuntu:~$ ls -l /tmp/ssh-Eehhsbx21654/agent.21654
srw------- 1 lumpy lumpy 0 марта 28 23:07 /tmp/ssh-Eehhsbx21654/agent.21654
```

Протокол SSH получил широкое распространение далеко за рамками своего изначального предназначения. Кроме непосредственного удаленного доступа, он используется другими командами для своих нужд. Например (см. листинг 6.19), команды **scp(1)** и **sftp(1)** используют **ssh(1)** для безопасного сетевого копирования файлов ❶❷, а команда **rsync(1)** — для сетевой синхронизации (копирование изменившихся) файлов ❸.

Все эти (и другие, подобные им) команды используют **ssh(1)** для запуска некоторой «серверной» программы на удаленном узле (в частности, **scp** и **rsync** запускают сами себя, а **sftp** запускает **sftp-server(8)**), с которой и взаимодействуют через защищенное соединение.

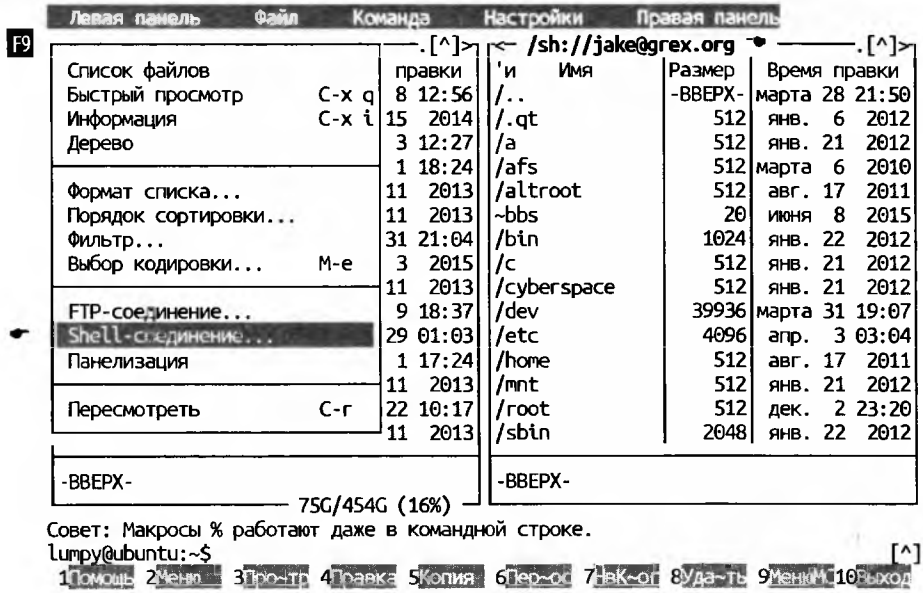
Листинг 6.19. Копирование файлов поверх SSH

```
❶ lumpy@ubuntu:~$ scp *.pdf jake@grex.org:
  tcpdump.pdf                                   100%   37KB  37.3KB/s   00:00
  Wireshark_Display_Filters.pdf                 100%   38KB  38.0KB/s   00:00
❷ lumpy@ubuntu:~$ sftp jake@grex.org
  Connected to grex.org.
  sftp> ls
  OPENME.txz                    README.gz
  Wireshark_Display_Filters.pdf linuxperftools.png
  tcpdump.pdf
  sftp> exit
```

```
❷ lumpy@ubuntu:~$ rsync -avrz jake@grex.org:/usr/share/man/man1 .
  receiving incremental file list
  man1/
  man1/acme-client.1
  man1/addr2line.1
                    ...          ...          ...
  man1/i386/fdformat.1
  man1/sparc64/
  man1/sparc64/fdformat.1
  man1/sparc64/mksuncd.1

  sent 9,529 bytes  received 4,180,838 bytes  239,449.54 bytes/sec
  total size is 12,970,760  speedup is 3.10
```

Как оказывается на практике, использование «файловой» надстройки **sftp-server(1)** для безопасного доступа к дереву каталогов удаленных узлов имеет достаточно широкое распространение. В частности, терминальный файловый менеджер **mc(1)**, **W:[Midnight Commander]**, тоже является SSH:sftp-клиентом, что иллюстрирует листинг 6.20. Более того, при помощи FUSE-файловых систем **sshfs(1)** (см. листинг 3.29) или **gvfs** файлы SSH:sftp-серверов могут быть смонтированы в дерево каталогов так, что вообще любые программы смогут ими воспользоваться.

Листинг 6.20. Файловый менеджер mc(1) — клиент SSH (sftp)



6.4.2. Почтовые службы SMTP, POP/IMAP

Электронная почта, пожалуй, является самым ранним приложением сетевой подсистемы операционных систем семейства UNIX. Изначально электронные письма пересылались непосредственно между конечными сетевыми узлами при помощи службы **W:[sendmail]** с использованием протокола **W:[SMTP]**, а для отправки или чтения писем применялась утилита **mail(1)**.

Вместо **sendmail** может быть использована абсолютно любая реализация агента пересылки почты (**W:[mail transport agent]**, MTA[1]), например **W:[postfix]** или **W:[exim]**, но обычно его функцию делегируют почтовым серверам провайдера услуг Интернета или серверам публичных сервисов типа **yandex.ru** или **gmail.com**.

Листинг 6.21 иллюстрирует «классическую» схему электронной почты с «локальным» MTA, использующую команду **mail(1)** для составления исходящих ❶ и чтения входящих ❸ писем и команду **mailq(1)** для просмотра очередей обработки почты ❷.

Листинг 6.21. Обработка почты локальной почтовой системой

```
❶ lumpy@ubuntu:~$ mail -s Тест dketov@gmail.com
   Cc: ↵
   Это тест ;) ↵
   ^D
❷ lumpy@ubuntu:~$ mailq
   Mail queue is empty
❸ lumpy@ubuntu:~$ mail
   Mail version 8.1.2 01/15/2001.  Type ? for help.
   "/var/mail/lumpy" ☚: 1 message 1 new
   >N  1 MAILER-DAEMON@ubu  Thu Jan  7 15:09   77/2653  Undelivered Mail Returned t
☚ & 1 ↵
   Message 1:
   From MAILER-DAEMON  Thu Jan  7 15:09:35 2016
   X-Original-To: lumpy@ubuntu
   Date: Thu,  7 Jan 2016 15:09:35 +0300 (MSK)
   From: MAILER-DAEMON@ubuntu (Mail Delivery System)
   Subject: Undelivered Mail Returned to Sender
   To: lumpy@ubuntu
                    ...          ...          ...          ...
   This is the mail system at host ubuntu.☚
```

[1] О всех этих MTA, MDA и MRA см. подробнее здесь: **https://tiny.cc/n1yqgz**.

```
   I'm sorry to have to inform you that your message could not
   be delivered to one or more recipients. It's attached below.
                    ...          ...          ...          ...
★  <dketov@gmail.com>: host gmail-smtp-in.l.google.com[74.125.205.27] said:
|      550-5.7.1 [95.55.94.237] The IP you're using to send mail is not
|      Authorized to 550-5.7.1 send email directly to our servers. Please use the
|    ☛ SMTP relay at your 550-5.7.1 service provider ☚ instead. Learn more at 550
|      5.7.1 https://support.google.com/mail/answer/10336 tw4si41552991lbb.77 -
└      gsmtp (in reply to end of DATA command)
                    ...          ...          ...          ...
   & q ↵
   Saved 1 message in /home/lumpy/mbox ☚
❶  lumpy@ubuntu:~$ mail handy@happytreefriends.ru
   Cc: ↵
   Subject: См. hands(4) ... ↵
   ... про /dev/hands ;) ↵
   ^D
❷  lumpy@ubuntu:~$ mailq
   -Queue ID-  --Size-- ----Arrival Time---- -Sender/Recipient-------
   8B85027304*     341 Sun Nov 24 00:38:27  lumpy@ubuntu
                                            handy@happytreefriends.ru

   -- 0 Kbytes in 1 Request.
```

Современные требования к условиям корректной пересылки почтовых сообщений (например, ★ в листинге 6.21) зачастую оказываются чересчур строгими, а содержание собственной локальной почтовой системы — необправданно сложным. В большинстве случаев конечные пользователи пользуются услугами «внешних» почтовых серверов, организующих полный цикл обработки почты — от приема исходящих сообщений до обслуживания почтовых ящиков. Исходящие сообщения отправляются таким серверам с помощью протокола **W:[SMTP]**, а доступ к почтовым ящикам — при помощи протоколов **W:[POP3]** или **W:[IMAP]**.

В примере из листинга 6.22 показан терминальный клиент современных почтовых систем **mutt(1)**, поддерживающий «защищенные» протоколы электронной почты **SMTPs**, **IMAPs** и **POPs**, использующие протокол **W:[SSL]**[1] для криптозащиты сетевых соединений.

[1] Использующий весьма похожие на SSH способы обеспечения конфиденциальности данных и аутентичности взаимодействующих сторон.

Для отправки ❶ сообщений используются учетная запись пользователя **lumpy.moose** и сервер «исходящих» сообщений **smtp.yandex.ru**, принимающий почту по протоколу SMTPs ①, а для чтения ❷ писем из почтового ящика пользователя **lumpy.moose** могут быть использованы протоколы IMAPs ② или POPs ③ и серверы «входящих» сообщений **imap.yandex.ru** и **pop.yandex.ru**, соответственно.

Листинг 6.22. Обработка почты публичной почтовой службой

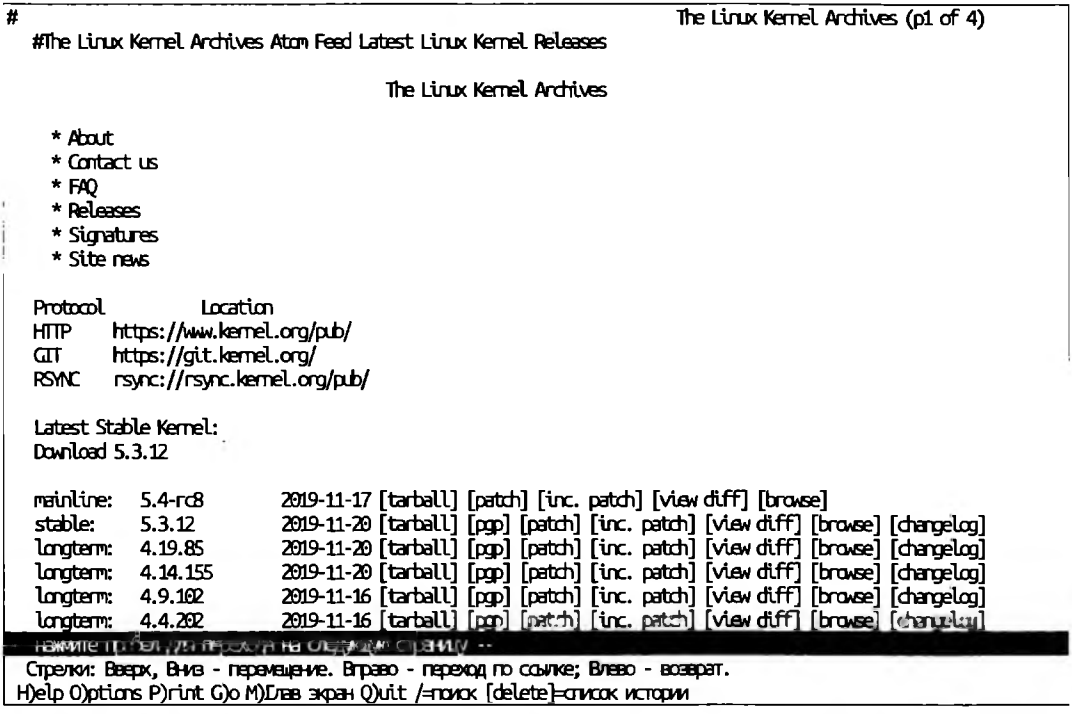
```
❶ lumpy@ubuntu:~$ mutt -s Тест dketov@gmail.com
                  ...          ...                    ...
  lumpy@ubuntu:~$ cat ~/.muttrc
  set from=lumpy.moose@yandex.ru
  set smtp_url=smtps://lumpy.moose@smtp.yandex.ru
                ①
❷ lumpy@ubuntu:~$ mutt -f imaps://lumpy.moose@imap.yandex.ru
                       ②
  Определяется адрес сервера imap.yandex.ru...
  Устанавливается соединение с imap.yandex.ru...
  SSL/TLS-соединение с использованием TLS1.3 (ECDHE-RSA/AES-256-GCM/AEAD)
  Пароль для lumpy.moose@imap.yandex.ru: ... ... ... ... ... ↵
q:Выход  d:Удалить  u:Восстановить  s:Сохранить  m:Создать  r:Ответить  g:Всем
   1     Jan 07 Яндекс            (9,9K) Соберите всю почту в этот ящик
   2     Jan 07 Команда Яндекс. ( 15K) Как читать почту с мобильного
   3 N + Mar 30 Яндекс.Паспорт  (8,4K) Доступ к аккаунту восстановлен

---Mutt: imaps://lumpy.moose@imap.yandex.ru/INBOX [Msgs:3 New:1 33K]---(threads/date)-(all)---

❷ lumpy@ubuntu:~$ mutt -f pops://lumpy.moose@pop.yandex.ru
                       ③
                  ...                    ...
```

6.4.3. Служба WWW

Служба W:[WWW] знакома каждому современному пользователю и в комментариях особо не нуждается. Одной ее заметной особенностью в Linux, пожалуй, является существование терминальных Web-браузеров **links(1)**, **lynx(1)**, **elinks(1)** и **w3m(1)**, позволяющих работать с «текстовой» частью гипертекстовых Web-ресурсов, что проиллюстрировано с помощью **lynx(1)** в примере из листинга 6.23.

Листинг 6.23. Терминальные браузеры lynx, links и w3m

```
lumpy@ubuntu:~$ lynx http://www.kernel.org
```



Кроме Web-браузеров, предназначенных для интерактивной работы пользователей, в сценариях на языке командного интерпретатора зачастую используются неинтерактивные пользовательские агенты **wget(1)** и **curl(1)**, позволяющие автоматизировать Web-взаимодействие. Так, например, в листинге 6.24 при помощи **wget(1)** показано скачивание файла в режиме «с докачкой» (**-c**, continue), а **curl(1)** применяется для обращения к Google Geocoding API.

Листинг 6.24. Пользовательские агенты wget и curl

```
lumpy@ubuntu:~$ wget -c http://www.brendangregg.com/Perf/linuxperftools.png
--2019-11-24 00:54:08-- http://www.brendangregg.com/Perf/linuxperftools.png
Распознаётся www.brendangregg.com (www.brendangregg.com)... 184.168.188.1
Подключение к www.brendangregg.com (www.brendangregg.com)|184.168.188.1|:80... соединение установлено.
HTTP-запрос отправлен. Ожидание ответа... 200 OK
Длина: 523561 (511K) [image/png]
Сохранение в: «linuxperftools.png»

42% [=========================>                ] 224 157      --.-K/s   за 11s
```

```
2019-11-24 00:54:19 (19,5 KB/s) - Ошибка чтения, позиция 224157/523561 (Время ожидания соединения истекло).
Продолжение попыток.

--2019-11-24 00:54:20--  (попытка: 2) http://www.brendangregg.com/Perf/linuxperftools.png
Подключение к www.brendangregg.com (www.brendangregg.com)|184.168.188.1|:80... соединение установлено.
HTTP-запрос отправлен. Ожидание ответа... 206 Partial Content
Длина: 523561 (511K), 299404 (292K) осталось [image/png]
Сохранение в: «linuxperftools.png»

100%[++++++++++++++++++++++++++=============================>] 523 561      213K/s   за 1,4s

2019-11-24 00:54:22 (747 KB/s) - «linuxperftools.png» сохранён [523561/523561]

lumpy@ubuntu:~$ curl -I http://man7.org/tlpi/download/TLPI-24-Process_Creation.pdf
HTTP/1.1 200 OK
Date: Sat, 23 Nov 2019 22:08:17 GMT
Server: Apache
Last-Modified: Thu, 21 Nov 2019 14:26:50 GMT
ETag: "171-31f5f5-597dc16857e80"
Accept-Ranges: bytes
Content-Length: 3274229
Connection: close
Content-Type: application/pdf
```

6.4.4. Служба FTP

Протокол W:[FTP] является «ископаемым[1]» даже по сравнению с W:[SMTP], однако все еще широко используется для организации доступа к обширным файловым хранилищам. Основная особенность протокола — отделение потока команд от потоков данных, что позволяет организовать параллельную передачу нескольких файлов одновременно. За счет этой особенности появляется возможность (практически не используемая, как небезопасная) передачи файлов не между файловым сервером и клиентом (как «обычно»), а между двумя (!) файловыми серверами, см. W:[FXP].

В листинге 6.25 иллюстрируется **lftp(1)** — один из самых распространенных терминальных FTP-клиентов, имеющий массу полезных возможностей, как то: «задачи» заднего фона ❷, зеркалирование файловых поддеревьев (включая FXP) ❷, неинтерактивная работа ❸ и т. д.

---

[1] Первая публикация спецификации протокола (RFC141) датируется 1971 годом, см. https://tools.ietf.org/html/rfc114.

Листинг 6.25. FTP-клиент lftp

```
lumpy@ubuntu $ lftp cdimage.debian.org
lftp cdimage.debian.org:~> cd /cdimage/ports/latest/hurd-i386/current ↵
cd ok, каталог=/cdimage/ports/latest/hurd-i386/current
lftp cdimage.debian.org:/.../hurd-i386/current> ls *.iso ↵
-rw-r--r--    1 ftp    ftp    670121984 Feb 20  2019 debian-sid-hurd-i386-CD-1.iso
-rw-r--r--    1 ftp    ftp    1764358144 Feb 20  2019 debian-sid-hurd-i386-DVD-1.iso
-rw-r--r--    1 ftp    ftp    174952448 Feb 20  2019 debian-sid-hurd-i386-NETINST-1.iso
-rw-r--r--    1 ftp    ftp    30199808 Feb 20  2019 mini.iso
lftp cdimage.debian.org:/.../hurd-i386/current> get debian-sid-hurd-i386-DVD-1.iso & ↵
                                                                                ❶↑
[0] get debian-sid-hurd-i386-DVD-1.iso &
        `debian-sid-hurd-i386-DVD-1.iso' в позиции 0
lftp cdimage.debian.org:/.../hurd-i386/current> get mini.iso & ↵
                                                          ↑❷
[1] get mini.iso &
        `mini.iso' в позиции 0
lftp cdimage.debian.org:/.../hurd-i386/current> jobs ↵
[1] get mini.iso  -- 907.5 Киб/с
        `mini.iso' в позиции 1573976 (5%) 907.5Кб/с овп:31с [Получение данных]
[0] get debian-sid-hurd-i386-DVD-1.iso  -- 2.95 Миб/с
        `debian-sid-hurd-i386-DVD-1.iso' в позиции 18610948 (1%) 2.95Мб/с овп:9м [Получение
                                                                                данных]
lftp cdimage.debian.org:/.../hurd-i386/current> quit ↵
❸ [12334] Переход в фоновый режим для завершения работы заданий...

lumpy@ubuntu $ lftp -c attach 12334
[12334] Присоединился к терминалу.
lftp cdimage.debian.org:/cdimage/ports/latest/hurd-i386/current> jobs ↵
[0] get debian-sid-hurd-i386-DVD-1.iso  -- 498.6 Киб/с
        `debian-sid-hurd-i386-DVD-1.iso' в позиции 718802944 (40%) овп:6м [Получение данных]
lftp cdimage.debian.org:/cdimage/ports/latest/hurd-i386/current> quit ↵
[12334] Переход в фоновый режим для завершения работы заданий...
```

Кроме массы специализированных FTP-клиентов **ftp(1)**, **lftp(1)**, **ncftp(1)**, **gftp(1)**, протокол FTP поддерживается и другими программными средствами, скажем различными файловыми менеджерами, что иллюстрирует листинг 6.26 на примере **mc(1)**.

Листинг 6.26. Файловый менеджер mc(1) — клиент FTP

```
   Левая панель    Файл    Команда    Настройки    Правая панель
F9 ┌─────────────────────────────┐──.[^]>┐┌< /ftp://mirror.yandex.ru ─.[^]>┐
   │ Список файлов               │ правки ││'и   Имя          │Размер │Время правки │
   │ Быстрый просмотр      C-x q │ 2 05:02││/..               │-ВВЕРХ-│марта 28 21:50│
   │ Информация            C-x i │18 21:00││/.ping            │  4096│марта 31  2014│
   │ Дерево                      │30 14:08││/altlinux         │  4096│ апр.  2 05:02│
   ├─────────────────────────────┤31 21:00││/altlinux-beta    │  4096│марта 18 21:00│
   │ Формат списка...            │ 2 13:33││/altlinu~ightly   │  4096│марта 30 14:08│
   │ Порядок сортировки...       │12 13:36││/altlinu~erkits   │  4096│марта 31 21:00│
   │ Фильтр...                   │18 21:00││/archlinux        │  4096│ апр.  2 13:33│
   │ Выбор кодировки...    M-e   │17  2007││/archlinux-arm    │  4096│ окт. 12 13:36│
   ├─────────────────────────────┤15 20:51││/archserver       │  4096│марта 18 21:00│
 ● │ FTP-соединение...           │ 2 02:34││/asplinux-tigro   │  4096│сент. 17  2007│
   │ Shell-соединение...         │ 2 12:06││/astra            │  4096│марта 15 20:51│
   │ Панелизация                 │ 2 13:18││/calculate        │  4096│ апр.  2 02:34│
   ├─────────────────────────────┤13 09:26││/centos           │  4096│ апр.  2 12:06│
   │ Пересмотреть          C-r   │ 2 09:36││/debian           │  4096│ апр.  2 13:18│
   └─────────────────────────────┘ 2 01:20││/debian-~kports   │  4096│марта 13 09:26│
   │/archlinux                           ││-ВВЕРХ-                                 │
Совет: Внешний просмотрщик можно выбрать с помощью переменной оболочки PAGER.
lumpy@ubuntu:~$                                                            [^]
 1Помощь 2Меню 3Про~тр 4Правка 5Копия 6Пер~ос 7НвК~ог 8Уда~ть 9МенюМС10Выход
```

Кроме всего прочего, клиентами FTP являются еще и внеядерные FUSE-файловые системы **curlftpfs** (см. листинг 3.29) или **gvfs**, позволяющие монтировать файлы FTP-серверов в дерево каталогов для их использования вообще любыми программами.

6.4.5. Служба NFS

Служба W:[**Network File System**] изначально разрабатывалась для *прозрачного* сетевого использования файловых систем сервера так, как будто они были непосредственно примонтированы в дерево каталогов клиента. В отличие от FTP (File Transfer Protocol)-протокола, предназначенного для скачивания (transfer) файлов, протокол NFS является ретранслятором системных вызовов **open(2)**, **close(2)**, **read(2)**, **write(2)**, **seek(2)** и прочих с клиента на сервер. Это позволяет клиенту выполнять операции чтения/записи с любой частью файла, без его передачи целиком.

NFS-клиент

Клиент протокола NFS непосредственно реализован в ядре Linux при помощи модулей **nfs**, **nfsv2**/**nfsv3**/**nfsv4** и используется с помощью штатной операции монтирования **mount(8)**, тем самым делая доступными серверные файлы любым клиентским программам.

В примере из листинга 6.27 показана процедура монтирования файловой системы **/Qmultimedia** NFS-сервера **NVR.local** ❷ в каталог **/mnt/network/nvr/Qmm** при помощи

протокола NFS v3 (**-t nfs -o vers=3**). Для получения списка экспортируемых (**-e**, export) сервером файловых систем ❶ вызывается команда **showmount(8)**, которая также является специализированным NFS-клиентом.

Листинг 6.27. Монтирование NFS

```
❶ lumpy@ubuntu:~$ showmount -e NVR.local
  Export list for NVR.local:
  /Qweb *
  /Qusb *
  /Qrecordings *
  /Qmultimedia *
  /Qdownload *
  /Public *
  /Network Recycle Bin 1 *
  lumpy@ubuntu:~$ sudo mkdir -p /mnt/network/nvr/Qmm
❷ lumpy@ubuntu:~$ sudo mount -t nfs -o vers=3 NVR.local:/Qmultimedia /mnt/network/nvr/Qmm
  lumpy@ubuntu:~$ findmnt -t nfs
  TARGET                      SOURCE                 STYPE  OPTIONS
  /mnt/network/nvr/Qmm        NVR.local:/Qmultimedia nfs    rw,relatime,vers=3,...
  lumpy@ubuntu:~$ ls /mnt/network/nvr/Qmm
❸ -rw-r--r-- 1 toothy users  678696 авг.  20  2014 IMG_20140719_125651.jpg
  -rw-r--r-- 1 toothy users  649685 авг.  20  2014 IMG_20140719_125713.jpg
                  ...        ...          ...          ...
  -rw-r--r-- 1 toothy users  814607 авг.  20  2014 IMG_20140820_111355.jpg
  lumpy@ubuntu:~$ id toothy
  uid=1010(toothy) gid=1012(toothy) группы=1012(toothy)
```

Необходимо отметить, что права доступа и идентификаторы **UID/GID** владельцев файлов ❸ передаются NFS-протоколом в неизменном виде, поэтому базы пользовательских учетных записей всех клиентов (и сервера) должны быть согласованы, например, при помощи их централизованного хранения в каталоге LDAP.

NFS-сервер

Функционирование NFS-сервера имеет свою специфику, связанную с использованием NFS-протоколом принципа RPC (**r**emote **p**rocedure **c**all, удаленного вызова процедур) в реализации **W:[SUN RPC]**. Серверы, использующие **SUN RPC**, не имеют «зафиксированного»[1] номера порта TCP/UDP, а используют произвольный, случай-

[1] Как, например, порт 22 закреплен за службой SSH, порт 25 — за SMTP, а порт 80 — за HTTP протоколом службы WWW.

но выбираемый порт. Вместо этого в **SUN RPC** закрепляются номера «программ» (program), предоставляющих определенные «услуги» (service), а соответствующий порт регистрируется в служебной RPC-программе **portmapper**[1], что проиллюстрировано в листинге 6.28 при помощи утилиты **rpcinfo(8)**.

Листинг 6.28. Удаленный вызов процедур RPC: динамические номера портов и portmapper

```
lumpy@ubuntu:~$ rpcinfo -p NVR.local
   program vers proto   port  service
    100000    3   tcp    111  portmapper
    100000    3   udp    111  portmapper
                ...     ...          ...          ...
    100003    3   tcp   2049  nfs
    100003    3   udp   2049  nfs
                ...     ...          ...          ...
    100005    3   udp  53964  mountd
    100005    3   tcp  39835  mountd
                ...     ...          ...          ...
```

При помощи **portmapper** организуется обнаружение NFS-серверов в локальной сети посредством широковещательного (**-b**, broadcast) поиска ❶ зарегистрированных RPC-программ NFS v3 (листинг 6.29). Кроме этого, некоторые серверы NFS (например, сетевые устройства хранения данных **W:[NAS]** или сетевые видеорегистраторы **W:[NVR]**) регистрируются в службе mDNS/DNS-SD, что обнаруживается ❷ при помощи **avahi-browse(1)**.

Листинг 6.29. Обнаружение NFS-серверов

```
❶ lumpy@ubuntu:~$ rpcinfo -b nfs 3
   192.168.17.90.8.1       NVR.local
                    ...          ...          ...          ...
❷ lumpy@ubuntu:~$ avahi-browse -rcl _nfs._tcp
                    ...          ...          ...          ...
   +   eth0 IPv4 NVR(NFS)                      Network File System          local
                    ...          ...          ...          ...
   =   eth0 IPv4 NVR(NFS)                      Network File System          local
      hostname = [NVR.local]
      address = [192.168.17.90]
      port = [2049]
      txt = []
```

[1] Порт самой RPC-программы **portmapper** все же закреплен за номером 111 TCP/UDP, что позволяет клиентам обращаться **portmapper** сервера и находить порты других RPC-программ по их номерам.

Сервер NFS предоставляет клиентам некоторое количество RPC-услуг (**-s**, services), показанных в листинге 6.30, при помощи **rpcinfo(8)**. Выделяют базовые RPC-программы **mountd** ② и **nfs** ①, позволяющие монтировать файловые системы NFS и обращаться к их файлам, и дополнительные RPC-программы[1] **nlockmgr** ③ и **status** ④, реализующие механизм блокировки файлов.

Листинг 6.30. RPC-программы службы NFS

```
lumpy@ubuntu:~$ rpcinfo -s NVR.local
   program version(s) netid(s)                         service     owner
    100000  4,3,2     tcp6,tcp,udp,udp6                 portmapper  superuser
    100003  4,3,2     udp6,tcp6,udp,tcp           ①     nfs         superuser
    100005  3,2,1     tcp6,udp6,tcp,udp           ②     mountd      superuser
    100021  4,3,2,1   tcp6,udp6,tcp,udp           ③     nlockmgr    superuser
    100024  1         tcp6,udp6,tcp,udp           ④     status      superuser
```

6.4.6. Служба SMB/CIFS

Служба W:[CIFS] (common internet file system), заимствованная из семейства операционных систем Windows, предназначена (аналогично «родной» NFS) для совместного использования файлов. Основным протоколом службы CIFS является протокол SMB (server message blocks), который аналогично NFS ретранслирует системные вызовы к файлам.

Имена NetBIOS

Отличительной особенностью протокола SMB, доставшейся ему в наследство от транспорта W:[NetBIOS], является возможность использования еще одного вида имен узлов (в дополнение к DNS- и mDNS-именам, *см. разд. 6.3*) — так называемых «имен NetBIOS» — и собственной службы NBNS[2] (netbios name service), отображающей имена NetBIOS на IP-адреса.

Листинг 6.31. Имена узлов NetBIOS и их IP-адреса

```
lumpy@ubuntu:~$ nmblookup WINXP
querying WINXP on 192.168.100.255
192.168.100.3 WINXP<00>
```

[1] См. W:[NLM], network lock manager и W:[NSM], network status monitor.

[2] Служба NBNS похожа на DNS и mDNS одновременно, но несовместима с ними.

В листинге 6.31 иллюстрируется использование утилиты **nmblookup(1)**, предназначенной для диагностики службы NBNS, при помощи которой «плоское» имя NetBIOS **WINXP** отображается на соответствующий ему IP-адрес. Именно при помощи службы NBNS и широковещательного поиска специальных «групповых» имен NetBIOS реализуется основной способ обнаружения CIFS-серверов локальной сети, что выполняет утилита **smbtree(1)**, иллюстрируемая ❶ в листинге 6.32.

В редких отдельных случаях серверы CIFS обнаруживаются ❷ зарегистрированными в службе mDNS/DNS-SD, что характерно (как и в случае серверов NFS) для сетевых устройств хранения данных W:[NAS] или сетевых видеорегистраторов W:[NVR].

Листинг 6.32. Обнаружение CIFS-серверов

```
❶ lumpy@ubuntu:~$ smbtree -N
  WORKGROUP
      \\WINXP
              \\WINXP\C$              Стандартный общий ресурс
              \\WINXP\ADMIN$          Удаленный Admin
              \\WINXP\media           Фото, видео, и т. д.
              \\WINXP\D$              Стандартный общий ресурс
              \\WINXP\IPC$            Удаленный IPC
      \\NVR
              \\NVR\Qrecordings
              \\NVR\Qmultimedia
              \\NVR\Qdownload
              \\NVR\IPC$

❷ lumpy@ubuntu:~$ avahi-browse -rcl _smb._tcp
  +   eth0 IPv4 NVR(SMB)                    Microsoft Windows Network      local
  =   eth0 IPv4 NVR(SMB)                    Microsoft Windows Network      local
     hostname = [NVR.local]
     address = [192.168.17.90]
     port = [445]
     txt = []
```

CIFS-клиенты

Различают две разные реализации клиента CIFS — внеядерную **smbclient(1)** (аналогичную «интерактивным» FTP-клиентам) и ядерную (аналогичную NFS-клиенту), реализуемую модулем ядра **cifs**. Использование ядерного модуля позволяет монти-

ровать общие файловые ресурсы (share) серверов CIFS непосредственно в дерево каталогов клиента, что дает возможность любым его программам использовать серверные файлы.

В примерах из листинга 6.33 показано использование CIFS-клиента **smbclient(1)** для получения списка (**-L**, list) разделяемых ресурсов ❶ (share) узла **WINXP**, равно как и для подключения ❷ к его публичному (**-N**, no password) разделяемому ресурсу **media** с последующим скачиванием файлов целиком.

Листинг 6.33. Клиент SMB/CIFS

```
❶ lumpy@ubuntu:~$ smbclient -NL //WINXP
  Domain=[WINXP] OS=[Windows 5.1] Server=[Windows 2000 LAN Manager]

        Sharename       Type      Comment
        ---------       ----      -------
        C$              Disk      Стандартный общий ресурс
        D$              Disk      Стандартный общий ресурс
        ADMIN$          Disk      Удаленный Admin
        IPC$            IPC       IPC Service
        media           Disk      Фото, видео, и т. д.

❷ lumpy@ubuntu:~$ smbclient -N //WINXP/media
  Domain=[WINXP] OS=[Windows 5.1] Server=[Windows 2000 LAN Manager]
  smb: \> cd DCIM\ ↵
  smb: \DCIM\> dir ↵
    .                                   D        0  Thu Jan  7 20:57:36 2016
    ..                                  D        0  Thu Jan  7 20:57:36 2016
    Dd595.jpg                           A  4494092  Sun Feb 13 16:24:04 2011
    Dd596.jpg                           A  3842680  Sun Feb 13 16:14:56 2011
                  ...           ...           ...                 ...
    Dd680.jpg                           A  4087313  Sun Feb 13 03:10:45 2011
    Dd681.jpg                           A  4108278  Sun Feb 13 15:58:38 2011
                61192 blocks of size 1048576. 10915 blocks available
  smb: \DCIM\> get Dd680.jpg ↵
  getting file \DCIM\Dd680.jpg of size 4087313 as Dd680.jpg (72572,9 KiloBytes/sec)
  (average 72573,0 KiloBytes/sec)
  smb: \DCIM\> quit ↵
```

Листинг 6.34 иллюстрирует использование ядерного модуля **cifs** для монтирования публичного (**-o guest**) ресурса **media** с узла **WINXP** в каталог **/mnt/network/winxp/media**.

Клиент **smbclient(1)** имеет встроенный механизм NBNS, поэтому без проблем подключается к узлу **WINXP**, а при монтировании **cifs** механизм NBNS недоступен ❶, что требует подсказки ❷ в виде IP-адреса сервера.

Листинг 6.34. Монтирование ресурса SMB/CIFS

```
lumpy@ubuntu:~$ sudo mkdir -p /mnt/network/winxp/media
lumpy@ubuntu:~$ sudo mount -t cifs -o guest //WINXP/media /mnt/network/winxp/media
❶ mount error: could not resolve address for WINXP: Unknown error
lumpy@ubuntu:~$ nmblookup WINXP
querying WINXP on 192.168.100.255
192.168.100.3 WINXP<00>
❷ lumpy@ubuntu:~$ sudo mount -t cifs -o guest,ip=192.168.100.3 //WINXP/media /mnt/network/winxp/media
lumpy@ubuntu:~$ findmnt -t cifs
TARGET                    SOURCE         FSTYPE OPTIONS
/mnt/network/winxp/media  //WINXP/media cifs   rw,relatime,vers=1.0,cache=strict,...
lumpy@ubuntu:~$ ls -l /mnt/network/winxp/media/DCIM/
итого 346228
-rwxr-xr-x 1 root root 4494092 февр. 13  2011 Dd595.jpg
                  ...        ...          ...          ...
-rwxr-xr-x 1 root     root    4108278 февр. 13  2011 Dd681.jpg
```

6.5. Средства сетевой диагностики

Диагностика сетевого обмена существенно облегчает решение разнообразных задач, связанных с эксплуатацией или разработкой сетевых приложений. К сетевым диагностическим специальным средствам относят анализаторы пакетов и сетевые сканеры, которые применяются самостоятельно или вместе с трассировщиками системных и библиотечных вызовов.

6.5.1. Анализаторы пакетов tcpdump и tshark

Анализаторы пакетов предназначены для перехвата данных, поступающих из сети на сетевые интерфейсы или отправляющиеся в сеть с сетевых интерфейсов. Современные анализаторы пакетов, кроме собственно захвата пакетов, осуществляют их детальный протокольный разбор, а также позволяют отфильтровывать подлежащие анализу пакеты по ряду критериев.

В листинге 6.35 показан пример использования наиболее распространенного, «классического» анализатора пакетов **tcpdump(1)**, анализирующего пакеты на интерфейсе (**-i**, interface) **wlp2s0**, адресованные порту **port 53**.

Листинг 6.35. Анализатор пакетов tcpdump

```
                                                                    pts/1
lumpy@ubuntu:~$ tcpdump -i wlp2s0 port 53
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlp2s0, link-type EN10MB (Ethernet), capture size 65535 bytes
                    ...         ...         ...         ...
                                                                    pts/2
lumpy@ubuntu:~$ host 2x2tv.ru
2x2tv.ru ① has address 146.158.12.222 ②
2x2tv.ru mail is handled by 10 fm.tnt-tv.ru.
                                                                    pts/1
                    ...         ...         ...         ...
15:11:32.139394 IP ubuntu.local.40694 > 192.168.100.1.domain: 2656+ [1au] A? 2x2tv.ru. (37)
15:11:32.144674 IP 192.168.100.1.domain > ubuntu.local.40694: 2656 1/0/1 A 146.158.12.222 (53)
15:11:32.145260 IP ubuntu.local.22022 > 192.168.100.1.domain: 63760+ [1au] MX? 2x2tv.ru. (37)
15:11:32.147959 IP 192.168.100.1.domain > ubuntu.local.48132: 63760 1/0/1 MX fm.tnt-tv.ru. 10 (63)
                    ...         ...         ...         ...
^C
4 packets captured
4 packets received by filter
0 packets dropped by kernel
```

Анализу подвергается работа DNS-клиента **host(1)**, отображающего имя домена **2x2tv.ru** на IP-адрес и имена его почтовых адресов (MX-записи DNS). В результате анализа захваченных пакетов наблюдаются запросы и ответы DNS-протокола к локальному кэширующему серверу **192.168.100.1**, который на запрос ① **A?** адреса IPv4, соответствующего имени **2x2tv.ru**, отвечает ② адресом **A 146.158.12.222**.

Терминальный[1] анализатор пакетов **tshark(1)**, позволяющий проводить детальный анализ прикладных протоколов, таких как SSH, HTTP, FTP, NFS и пр., проиллюстрирован в листинге 6.36. Здесь анализируется работа пользовательского агента **curl(1)**, запрашивающего Web-ресурс по адресу **http://ipinfo.io/city**. В результате захвата пакетов на интерфейсе (**-i**, interface) **wlp2s0**, адресованных порту **port 80**, просматриваются (**-R**, read filter) пакеты, содержащие **http**-запросы, при этом детальному анализу (**-V**, view) подвергается только (**-O**, only) их **http**-содержимое.

В результате анализа, например, можно сделать вывод ★ о программном обеспечении Web-сервера, обслуживающего сайт **http://ipinfi.io**.

[1] Гораздо удобнее, конечно, использовать его графический вариант — **wireshark(1)**.

Листинг 6.36. Анализатор пакетов wireshark

```
pts/1
lumpy@ubuntu:~$ tshark -i wlp2s0 -V -O http,data-text-lines -Y http port 80
Capturing on wlp2s0
                    ...         ...         ...         ...
pts/2
lumpy@ubuntu:~$ curl http://ipinfo.io/city
Saint Petersburg
pts/1
                    ...         ...         ...         ...
Frame 4: 131 bytes on wire (1048 bits), 131 bytes captured (1048 bits) on interface 0
Ethernet II, Src: PcsCompu_a9:78:36 (08:00:27:a9:78:36), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 216.239.36.21
Transmission Control Protocol, Src Port: 38534, Dst Port: 80, Seq: 1, Ack: 1, Len: 77
Hypertext Transfer Protocol
    GET /city HTTP/1.1\r\n
        [Expert Info (Chat/Sequence): GET /city HTTP/1.1\r\n]
            [GET /city HTTP/1.1\r\n]
            [Severity level: Chat]
            [Group: Sequence]
        Request Method: GET
        Request URI: /city
        Request Version: HTTP/1.1
    Host: ipinfo.io\r\n
    User-Agent: curl/7.65.3\r\n
    Accept: */*\r\n
    \r\n
    [Full request URI: http://ipinfo.io/city]
    [HTTP request 1/1]
                    ...         ...         ...         ...
Frame 6: 441 bytes on wire (3528 bits), 441 bytes captured (3528 bits) on interface 0
Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_a9:78:36 (08:00:27:a9:78:36)
Internet Protocol Version 4, Src: 216.239.36.21, Dst: 10.0.2.15
Transmission Control Protocol, Src Port: 80, Dst Port: 38534, Seq: 1, Ack: 78, Len: 387
Hypertext Transfer Protocol
    HTTP/1.1 200 OK\r\n
        [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
            [HTTP/1.1 200 OK\r\n]
            [Severity level: Chat]
            [Group: Sequence]
        Response Version: HTTP/1.1
        Status Code: 200
        [Status Code Description: OK]
        Response Phrase: OK
    Date: Sat, 23 Nov 2019 23:27:18 GMT\r\n
    Content-Type: text/html; charset=utf-8\r\n
    Content-Length: 7\r\n
        [Content length: 7]
```

```
    x-cloud-trace-context: 8fa9d5d63ef04965a6a136652cac2f6b/7935058721407822980\r\n
    Access-Control-Allow-Origin: *\r\n
    X-Frame-Options: DENY\r\n
    X-XSS-Protection: 1; mode=block\r\n
    X-Content-Type-Options: nosniff\r\n
    Referrer-Policy: strict-origin-when-cross-origin\r\n
    Via: 1.1 google\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.145039080 seconds]
    [Request in frame: 4]
    [Request URI: http://ipinfo.io/city]
    File Data: 7 bytes
Line-based text data: text/html (1 lines)
Saint Petersburg\n

^C2 packets captured
```

6.5.2. Сетевой сканер nmap

Сетевой сканер W:[nmap] предназначен для поиска служб по их открытым портам на указанных узлах сети. В примере из листинга 6.37 показан процесс и результаты сканирования узла **192.168.0.1** (беспроводной маршрутизатор, арендованный у провайдера Интернета).

Сканирование выполнялось способом TCP connect scan ❶, т. е. предпринималась попытка установить соединение TCP с каждым из 1000 «популярных» портов. В результате оказывается, что на узле открыты 4 порта, доступные ❷ для присоединения.

Листинг 6.37. Сетевой сканер nmap

```
    lumpy@ubuntu:~$ nmap -n -vvv --reason 192.168.0.1
    Starting Nmap 7.80 ( https://nmap.org ) at 2019-11-24 11:46 MSK
    Initiating Ping Scan at 11:46
    Scanning 192.168.0.1 [2 ports]
    Completed Ping Scan at 11:46, 0.01s elapsed (1 total hosts)
❶   Initiating Connect Scan at 11:46
    Scanning 192.168.0.1 [1000 ports]
    Discovered open port 80/tcp on 192.168.0.1
    Discovered open port 22/tcp on 192.168.0.1
    Discovered open port 1900/tcp on 192.168.0.1
    Discovered open port 49152/tcp on 192.168.0.1
    Completed Connect Scan at 11:46, 0.57s elapsed (1000 total ports)
```

```
Nmap scan report for 192.168.0.1
Host is up, received syn-ack (0.054s latency).
Scanned at 2019-11-24 11:46:18 MSK for 1s
Not shown: 996 closed ports
Reason: 996 conn-refused
❷ PORT      STATE SERVICE REASON
22/tcp    open  ssh     syn-ack
80/tcp    open  http    syn-ack
1900/tcp  open  upnp    syn-ack
49152/tcp open  unknown syn-ack
```

6.5.3. Мониторинг сетевых соединений процессов

Интерфейс сокетов в целом является продолжением идеи «файлов», специально предназначенных для межпроцессного взаимодействия, некоторым развитием «файловой» природы простейших именованных и неименованных каналов.

Таким образом, сокеты сетевых семейств **ip(7)**, **ipv6(7)** и прочие обладают «файловыми» свойствами точно так же, как и локальные сокеты **unix(7)**. Например, каждый сетевой сокет идентифицируется при помощи файлового (!) дескриптора в таблице открытых файлов процесса, что проиллюстрировано в листинге 6.38 при помощи **lsof(1)** и **ss(8)**.

Листинг 6.38. Файловые дескрипторы сетевых сокетов

```
lumpy@ubuntu:~$ pgrep lftp
6056
lumpy@ubuntu:~$ lsof -i 4 -a -p 6056
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
lftp    6056 lumpy ☛ 4u  IPv4  88150      0t0  TCP ubuntu.local:43578->napoleon.ftp.acc.umu.se:ftp (ESTABLISHED)

lumpy@ubuntu:~$ ss -p dport = :ftp
Netid  State   Recv-Q  Send-Q      Local Address:Port      Peer Address:Port
tcp    ESTAB   0       0           192.168.0.101:43578     194.71.11.173:ftp  users:(("lftp",pid=6056,fd=4☛))
```

Более детально проследить за жизненным циклом сетевого сокета позволяет трассировка «сокетных» системных вызовов **socket(2)**, **connect(2)**, **bind(2)**, **listen(2)**, **accept(2)**, **send(2)** и **recv(3)**.

В примере из листинга 6.39 показана трассировка «клиентского» сокета пользовательского агента **curl(1)**, загружающего Web-ресурс **http://www.gnu.org/graphics/agnuheadterm-xterm.txt**.

Системный вызов **socket(2)** ❶ создает потоковый сокет семейства **ip(7)**, которому назначается первый свободный файловый дескриптор **FD=3**, после чего системный вызов **connect(2)** инициирует установку соединения ❷ этого сокета с портом **80** узла **209.51.188.148**. После установки соединения системный вызов **sendto(2)** отсылает Web-серверу команду **W:[HTTP]** протокола **GET** на получение запрашиваемого ресурса, а несколько системных вызовов **recvfrom(2)** получают запрошенный ресурс.

Листинг 6.39. Сетевой клиент: системные вызовы socket(2), connect(2), sendto(2) и recvfrom(2)

```
lurpy@ubuntu:~$ strace -fe trace=network curl http://www.gnu.org/graphics/agnuheadterm-xterm.txt
                    ...          ...          ...          ...
❶ socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
  setsockopt(3, SOL_TCP, TCP_NODELAY, [1], 4) = 0
                    ...          ...          ...          ...
❷ connect(3, {sa_family=AF_INET, sin_port=htons(80), sin_addr=inet_addr("209.51.188.148")}, 16) = -1
  EINPROGRESS (Операция выполняется в данный момент)
                    ...          ...          ...          ...
❸ sendto(3, "GET /graphics/agnuheadterm-xterm"..., 106, MSG_NOSIGNAL, NULL, 0) = 106
❹ recvfrom(3, "HTTP/1.1 200 OK\r\nDate: Sun, 24 N"..., 102400, 0, NULL, NULL) = 1382
  recvfrom(3, "249m\342\226\204\33[38;5;246m\342\226"..., 17588, 0, NULL, NULL) = 12438
                    ...          ...          ...          ...
```

Жизненный цикл «серверных» сокетов чуть более сложен и предполагает использование одного «слушающего» сокета для клиентских подключений и по одному «обслуживающему» сокету на каждого подключенного клиента.

В примере из листинга 6.40 показана трассировка простейшего Web-сервера, реализованного модулем **SimpleHTTPServer** языка программирования **W:[python]**. Web-сервер запускается из «рабочего» каталога **/usr/share/doc** и предоставляет Web-доступ ко всем файлам этого каталога, используя «нестандартный» порт **8000**.

Для начала при помощи **socket(2)** создается ❶ потоковый сокет семейства **ip(7)**, которому назначается первый свободный файловый дескриптор **FD=3**. Этот сокет и будет выступать в роли «слушающего», т. е. принимающего клиентские соединения, поэтому ему «привязывается» ❷ адрес **0.0.0.0** и порт **8000** (куда и будут поступать клиентские соединения) при помощи системного вызова **bind(2)**, а сам сокет переводится в слушающее состояние ❸ системным вызовом **listen(2)**. Все входящие клиентские соединения ставятся в очередь «слушающего» сокета и изымаются из нее системным вызовом **accept(2)**, который создает для каждого клиентского соединения собственный сокет (клонируя слушающий). При поступлении клиентского соединения ❹ был создан новый «обслуживающий» сокет с файловым дескриптором **FD=4**, используя который при помощи **recv(2)** была получена **W:[HTTP]**-команда **GET** на доступ к «корневому» ресурсу сервера, а с помощью нескольких системных вызовов **send(2)** в ответ был направлен сформированный HTML-список файлов в каталоге **/usr/share/doc**.

Листинг 6.40. Сетевой сервер: системные вызовы socket(2), bind(2), listen(2), accept(2), send(2) и recv(2)

pts/1

```
lumpy@ubuntu:~$ cd /usr/share/doc
lumpy@ubuntu:/usr/share/doc$ strace -fe trace=network python -m SimpleHTTPServer
❶ socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 3
setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
❷ bind(3, {sa_family=AF_INET, sin_port=htons(8000), sin_addr=inet_addr("0.0.0.0")}, 16) = 0
                    ...          ...          ...          ...
❸ listen(3, 5)                  = 0
                    ...          ...          ...          ...
Serving HTTP on 0.0.0.0 port 8000 ...
❹ accept(3, {sa_family=AF_INET, sin_port=htons(55094), sin_addr=inet_addr("127.0.0.1")}, [16]) = 4 ↵
❺ recvfrom(4 ↵, "GET / HTTP/1.1\r\nHost: localhost:"..., 8192, 0, NULL, NULL) = 78
127.0.0.1 - - [24/Nov/2019 12:14:54] "GET / HTTP/1.1" 200 -
❻ sendto(4 ↵, "HTTP/1.0 200 OK\r\n", 17, 0, NULL, 0) = 17
sendto(4, "Server: SimpleHTTP/0.6 Python/2."..., 41, 0, NULL, 0) = 41
sendto(4, "Date: Sun, 24 Nov 2019 09:14:54 "..., 37, 0, NULL, 0) = 37
sendto(4, "Content-type: text/html; charset"..., 40, 0, NULL, 0) = 40
sendto(4, "Content-Length: 86602\r\n", 23, 0, NULL, 0) = 23
sendto(4, "\r\n", 2, 0, NULL, 0)       = 2
sendto(4, "<!DOCTYPE html PUBLIC \"-//W3C//D"..., 8192, 0, NULL, 0) = 8192
                    ...          ...          ...          ...
shutdown(4, SHUT_WR)            = 0^C
```

pts/2

```
lumpy@ubuntu:~$ wget http://ubuntu:8000
--2019-11-24 12:18:05--  http://ubuntu:8000/
Распознаётся ubuntu (ubuntu)... 127.0.1.1
Подключение к ubuntu (ubuntu)|127.0.1.1|:8000... соединение установлено.
HTTP-запрос отправлен. Ожидание ответа... 200 OK
Длина: 86602 (85K) [text/html]
Сохранение в: «index.html»

100%[=====================================>] 84,57K     --.-K/s   за 0,002s

2019-11-24 12:18:05 (54,6 MB/s) - «index.html» сохранён [86602/86602]
```

6.6. В заключение

Сетевая подсистема ОС Linux чрезвычайно развита на всех ее уровнях — от сетевых интерфейсов и протоколов и до прикладных сетевых служб. На сегодняшний день колоссальное количество сетевых устройств работают под управлением

Linux — маршрутизаторы, сетевые хранилища, медиаплееры, TV-боксы, планшеты, смартфоны и прочие «встраиваемые» и мобильные устройства.

К сожалению, рассмотреть весь пласт сетевых возможностей в рамках этой книги не представляется возможным, т. к. потребует от читателя серьезного понимания устройства и функционирования самих сетевых протоколов стека TCP/IP, что не является предметом настоящего рассмотрения.

Основополагающим результатом текущей главы должно стать понимание принципов организации сетевого взаимодействия в Linux, необходимое и достаточное в качестве базы для последующего самостоятельного расширенного и углубленного изучения. Не менее полезными в практике администратора и программиста будут навыки использования инструментов трассировки и мониторинга сетевых сокетов, а в особенно «непонятных» ситуациях — навыки применения анализаторов пакетов.

Исключительное место (эдакий «швейцарский нож») среди прочих сетевых инструментов Linux займет служба SSH, применение которой найдется уже в следующей главе, при распределенном использовании оконной системы X Window System, являющейся основой современного графического интерфейса пользователя.

Глава 7

Графическая система X Window System

Графическая система W:[X Window System] предоставляет приложениям операционной системы возможность представления графической информации на графических устройствах вывода, в большинстве случаев — на дисплеях растровых видеотерминалов.

Основной принцип графической системы X(7) позволяет множеству графических приложений осуществлять «одновременный» графический вывод за счет *окон*, совместно отображаемых на дисплеях. Содержимое окон определяется их владельцами — графическими приложениями, а управление окнами (их создание, уничтожение, отображение на дисплеях, перекрытие другими окнами, перемещение, изменение размеров и пр.) возложено на *оконную* систему.

Действительной особенностью оконной системы X(7) является ее сетевая прозрачность, т. е. возможность различным компонентам выполняться в виде отдельных процессов на разных узлах сети и взаимодействовать при помощи соответствующих средств, в большинстве случаев — с использованием сетевых сокетов семейств ip(7) и ipv6(7). При выполнении компонент оконной системы на одном узле в целях повышения производительности используют локальные (файловые) сокеты unix(7) и даже разделяемую память (*см. разд. 4.9.5* или *7.6*).

Как любая другая сетевая служба, оконная система X(7) состоит из X-сервера и X-клиентов, взаимодействующих между собой посредством W:[X протокола].

7.1. X-сервер

Первичной компонентой оконной системы X(7) является X-сервер, основная задача которого заключается в управлении оборудованием графического вывода и ввода. Под управлением X-сервера находятся графические дисплеи (видеоадаптеры и подключенные к ним мониторы), устройства «графического» интерфейса с пользователем (манипуляторы «мышь», трекболы, тачпады, графические планшеты и пр.), а кроме того, устройства «символьного» взаимодействия с пользователем — клавиатуры.

Именно X-сервер принимает подключения от X-клиентов и согласно их запросам создает окна, изменяет их размер, отображает или скрывает окна на дисплеях, сообщает положение курсора, рисует текст, линии, точки, прямоугольники, дуги, полигоны и пр. В обратную сторону X-сервер отправляет X-клиентам информацию о *событиях* (events) нажатия клавиш, кнопок мыши и планшетов, оповещает о движении курсора и т. д.

Листинг 7.1. Аппаратный X-сервер

```
homer@ubuntu:~$ pgrep -l Xorg
6892 Xorg
homer@ubuntu:~$ ps o pid,tty,cmd p 6892
 PID TT       CMD
6892 tty3     /usr/lib/xorg/Xorg vt3 -displayfd 3 -auth /run/user/1000/gdm/Xauthority ...

❶ homer@ubuntu:~$ lsof -p 6892 -a /dev
COMMAND  PID USER  FD   TYPE DEVICE SIZE/OFF NODE NAME
Xorg    6892 homer mem   CHR  226,0          439 /dev/dri/card0
Xorg    6892 homer 0u    CHR    4,3      0t0  24 /dev/tty3
Xorg    6892 homer 11u   CHR    4,3      0t0  24 /dev/tty3 ⓪
Xorg    6892 homer 12u   CHR  226,0      0t0 439 /dev/dri/card0
                  ...           ...        ...          ...
Xorg    6892 homer 16u   CHR  226,0      0t0 439 /dev/dri/card0
Xorg    6892 homer 24u   CHR  13,64      0t0 149 /dev/input/event0
                  ...           ...        ...          ...
Xorg    6892 homer 32u   CHR  13,68      0t0 336 /dev/input/event4

❷ homer@ubuntu:~$ lsof -p 6892 -a -U
COMMAND  PID USER   FD   TYPE             DEVICE SIZE/OFF   NODE NAME
Xorg    6892 homer   1u  unix 0x0000000000000000      0t0 118649 type=STREAM
Xorg    6892 homer   2u  unix 0x0000000000000000      0t0 119748 type=STREAM
Xorg    6892 homer   3u  unix 0x0000000000000000      0t0 119927 @/tmp/.X11-unix/X0 type=STREAM
Xorg    6892 homer   6u  unix 0x0000000000000000      0t0 118653 @/tmp/.X11-unix/X0 type=STREAM
Xorg    6892 homer   7u  unix 0x0000000000000000      0t0 118654 /tmp/.X11-unix/X0① type=STREAM
                    ...           ...                   ...                  ...
Xorg    6892 homer  47u  unix 0x0000000000000000      0t0 124222 @/tmp/.X11-unix/X0 type=STREAM

❸ homer@ubuntu:~$ lsof -p 6892 -a -i
!☛
```

Листинг 7.1 иллюстрирует сервер **Xorg(1)**, использующий специальные файлы ❶ устройств видеоускорителя **/dev/dri/card0** и устройств пользовательского ввода

/dev/input/* для организации на виртуальном терминале **/dev/tty3** графического дисплея ⓪.

Для взаимодействия с X-клиентами сервер создает ❷ локальный (файловый) сокет ① **/tmp/.X11-unix/X0**, но не создает по умолчанию сетевой сокет ❸, поэтому подключение X-клиентов по сети к такому серверу невозможно.

Основные свойства X-сервера и управляемых им графических дисплеев можно получить при помощи утилиты **xdpyinfo(1)**, см. листинг 7.2.

Листинг 7.2. Свойства дисплея X-сервера

```
homer@ubuntu:~$ xdpyinfo | grep -A 4 screen
default screen number:    0
number of screens:    1

screen #0:
  dimensions:    1600x900 pixels (423x238 millimeters)
  resolution:    96x96 dots per inch
  depths (7):    24, 1, 4, 8, 15, 16, 32
  root window id:    0x17c
```

7.2. X-клиенты и X-протокол

Второй важной компонентой оконной системы **X(7)** являются X-клиенты — приложения, получающие в свое распоряжение окна и отображающие в них графическую информацию. Точнее, X-клиенты всего лишь взаимодействуют с сервером при помощи X-протокола и могут вовсе не создавать окон, как это делают «простейшие» **xdpyinfo(1)**, **xrandr(1)** и **glxinfo(1)**, **xlsclients(1)**, **xwininfo(1)**, **xprop(1)**, но «полновесные» X-клиенты — всегда создают хотя бы одно окно.

Список созданных окон (упорядоченный в дерево[1]) и системные *атрибуты* каждого из них можно запросить у X-сервера при помощи **xwininfo(1)**. В листинге 7.3 построено полное дерево окон (начиная с «корневого» окна самого X-сервера), среди которых отобраны окна с именем **gnome-terminal** (предположительно принадлежащие **gnome-terminal(1)**, узнать явно нет возможности). При этом оказывается, что только одно окно изображается ❸ X-сервером на дисплее, а еще одно находится в скрытом состоянии ❶.

---

[1] Окна в X Window System связаны дочерне-родительскими отношениями, определяемыми при создании окна. Суть этих отношений состоит в том, что координаты окон всегда определяют их положение относительно родительского окна, при этом изображаются только части окон, видимые в пределах своего родительского окна.

Листинг 7.3. Дерево окон X-сервера и их атрибуты

```
homer@ubuntu:~$ xwininfo -tree -root | grep gnome-terminal
   0x2a0000a "homer@ubuntu: ~": ("gnome-terminal-server" "Gnome-terminal")  786x527+74+134  +74+134
   0x2a00001 "Терминал": ("gnome-terminal-server" "Gnome-terminal-server")  10x10+10+10  +10+10

homer@ubuntu:~$ xwininfo -id 0x2a00001 | grep Map
  Map State: IsUnMapped ❶

homer@ubuntu:~$ xwininfo -id 0x2a0000a | egrep 'Map|Width|Height'
  Width: 786
  Height: 527
  Map State: IsViewable ❷
```

Кроме системных атрибутов, каждое окно наделяется *свойствами (properties)*, широко используемыми для взаимодействия между X-клиентами (см. W:[ICCCM]), особенно между «обычными» клиентами и оконным менеджером *(window manager)*, *см. разд. 7.3*. Оконный менеджер является «особенным» X-клиентом, обрабатывающим события создания окон «обычных» X-клиентов. Именно он добавляет к «чужим» создаваемым окнам «свои» декорирующие элементы: заголовок (*title*) окна — для его перемещения, бордюр (*border*) — для изменения его размеров и т. д. Делает он это одним незамысловатым способом — создает собственное окно с декором, и делает его родительским для декорируемого окна.

В листинге 7.4 показаны некоторые свойства окна, установленные программой-владельцем окна для оконного менеджера. Например, свойство WM_NAME используется оконным менеджером для текста заголовка (отображаемых) окон, свойство WM_LOCALE_NAME указывает на язык и кодировку текста, содержащегося в WM_NAME, свойство WM_CLIENT_MACHINE содержит имя сетевого узла X-клиента, а свойство WM_COMMAND — команду, при помощи которой был запущен клиент.

Листинг 7.4. Свойства окон X-сервера

```
homer@ubuntu:~$ xprop -id 0x2a00001 | grep ^WM_
WM_CLASS(STRING) = "gnome-terminal-server", "Gnome-terminal-server"
WM_COMMAND(STRING) = { "gnome-terminal-server" }
WM_CLIENT_LEADER(WINDOW): window id # 0x2800001
WM_LOCALE_NAME(STRING) = "ru_RU.UTF-8"
WM_CLIENT_MACHINE(STRING) = "ubuntu"
WM_NORMAL_HINTS(WM_SIZE_HINTS):
WM_PROTOCOLS(ATOM): protocols  WM_DELETE_WINDOW, WM_TAKE_FOCUS, _NET_WM_PING
WM_ICON_NAME(COMPOUND_TEXT) = "Терминал"
WM_NAME(COMPOUND_TEXT) = "Терминал"
```

Свойства окон X-клиентов могут использоваться не только оконным менеджером, но и другими клиентами, например **xlsclients(1)**, который опрашивает X-сервер (листинг 7.5) и выводит список «клиентов». На самом деле он отображает список окон только со свойствами **WM_CLIENT_MACHINE** и **WM_COMMAND**, которые в действительности устанавливаются (и то не всегда) X-клиентами для своих «основных» окон.

Листинг 7.5. Список клиентов X-сервера

```
homer@ubuntu:~$ xlsclients -l | grep -C 3 gnome-terminal
Window 0x2a00001:
  Machine:  ubuntu
  Name:  <unknown type COMPOUND_TEXT (502) or format 8>
  Icon Name:  <unknown type COMPOUND_TEXT (502) or format 8>
  Command:  gnome-terminal-server
  Instance/Class:  gnome-terminal-server/Gnome-terminal-server
```

Взаимодействие X-клиентов и X-сервера происходит при помощи локальных или сетевых сокетов в зависимости от их взаимного месторасположения и согласно адресу подключения, указываемому на стороне X-клиентов при помощи переменной окружения **DISPLAY** в формате **host:number**.

Адрес подключения состоит из имени (или IP-адреса) узла X-сервера **host** и номера его дисплея **number** (например, **DISPLAY=ubuntu.local:0** или **192.168.0.5:0**). В качестве **host** может выступать любое имя, для которого можно получить IP-адрес, используя службу имен *(см. разд. 6.3)*, а в случае локального взаимодействия **host** не указывается вовсе (т. е. **DISPLAY=:0**).

Номер дисплея указывает на экземпляр X-сервера, запущенного на узле **host** для управления некоторым набором оборудования (графическими дисплеями — видеокартами и мониторами, клавиатурами и пр.).

Проиллюстрировать разные виды X-взаимодействия проще всего при помощи «виртуальных» X-серверов **Xnest(1)** и **Xephyr(1)**. В отличие от «аппаратного» X-сервера, организующего доступ X-клиентов к аппаратному дисплею, эти серверы организуют доступ своих клиентов к виртуальному «дисплею», в качестве которого выступает их собственное окно аппаратного сервера.

В листинге 7.6 показан запуск X-сервера **Xnest(1)** с номером дисплея **:1** (т. к. дисплей **:0** обычно занят ❶ аппаратным X-сервером), который создает сокет ❷ семейства **ip(7)** для подключения сетевых клиентов и сокет ❸ семейства **unix(7)** для локальных.

Листинг 7.6. Виртуальный X-сервер Xnest

```
homer@ubuntu:~$ Xnest :0 &
_XSERVTransSocketUNIXCreateListener: ...SocketCreateListener() failed
_XSERVTransMakeAllCOTSServerListeners: server already running
(EE)
Fatal server error:
❶ (EE) Cannot establish any listening sockets - Make sure an X server isn't already running(EE)

homer@ubuntu:~$ Xnest :1 -listen tcp &
[1] 10950
homer@ubuntu:~$ lsof -p 10950 -a -i
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
❷ Xnest   10950 homer  4u  IPv6 146673      0t0  TCP *:x11-1 (LISTEN) ➥
  Xnest   10950 homer  5u  IPv4 146674      0t0  TCP *:x11-1 (LISTEN)
homer@ubuntu:~$ lsof -p 10950 -a -U
COMMAND   PID USER   FD   TYPE             DEVICE SIZE/OFF   NODE NAME
  Xnest   10950 homer  6u  unix 0x0000000000000000      0t0 146675 @/tmp/.X11-unix/X1   type=STREAM
❸ Xnest   10950 homer  7u  unix 0x0000000000000000      0t0 146676 /tmp/.X11-unix/X1 ➥ type=STREAM
  Xnest   10950 homer  9u  unix 0x0000000000000000      0t0 146679 type=STREAM
```

В примере из листинга 7.7 показана трассировка демонстрационного X-клиента **xeyes(1)**, иллюстрирующая использование клиентом локального ❶ и сетевого ❷ сокетов в зависимости от значения переменной окружения **DISPLAY**.

Листинг 7.7. Сетевые и локальные подключения клиента X-сервера

```
homer@ubuntu:~$ DISPLAY=:1 strace -fe connect xeyes
❶ connect(3, {sa_family=AF_UNIX, sun_path=@"/tmp/.X11-unix/X1" ➥}, 20) = 0
^C
homer@ubuntu:~$ DISPLAY=ubuntu.local:1 strace -fe connect xeyes
                ...          ...             ...             ...
❷ connect(3, {sa_family=AF_INET, sin_port=htons(6001), sin_addr=inet_addr("192.168.0.105")}, 16) = 0
^C
```

При работе в оконной системе переменную **DISPLAY** обычно устанавливают в начале сеанса, а затем при запуске X-клиентов их вывод будет попадать на нужный X-сервер, как это показано на примере[1] сервера **Xnest(1)** и клиентов **xterm(1)**, **xcalc(1)** и **xeyes(1)** в листинге 7.8 и на рис. 7.1.

---

[1] Все иллюстрации дальше сделаны именно при помощи «виртуальных» X-серверов **Xnest(1)** или **Xephyr(1)** и могут быть без проблем воспроизведены пользователем, уже работающим в оконной системе.

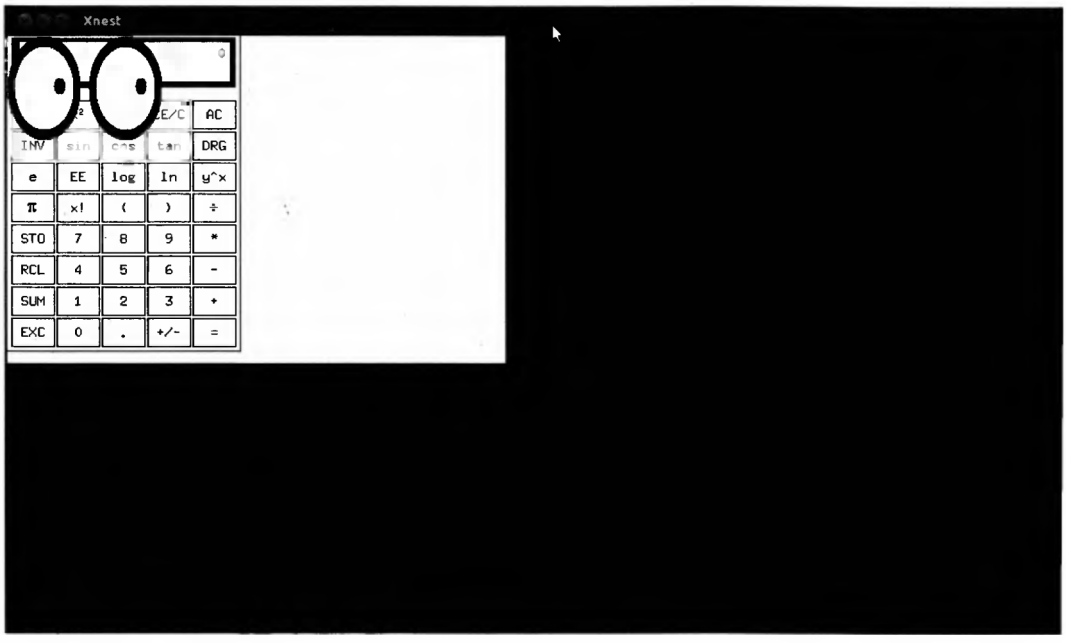


Рис. 7.1. X-клиенты на дисплее X-сервера: X-сеанс пользователя

Впоследствии при запуске одними X-клиентами других X-клиентов (например, по команде пользователя из эмулятора терминала **xterm(1)**, см. листинг 7.8) переменная окружения наследуется *(см. разд. 4.5.3)* порожденными процессами, и вывод запускаемых клиентов «интуитивным» (так же, как и стандартные потоки ввода-вывода) образом попадает на тот же X-сервер.

Листинг 7.8. X-сеанс пользователя

```
homer@ubuntu:~$ export DISPLAY=:1
homer@ubuntu:~$ xterm & xcalc & xeyes &
[3] 2803
[3] 2804
[4] 2805
```

Нужно заметить, что все окна по умолчанию открываются в левом верхнем углу (координата +0+0) экрана, а возможность управлять их размером и местоположением отсутствует. Каждый X-клиент может самостоятельно управлять размером и местоположением своих окон, и даже многие из них запоминают при выходе эти параметры в своих dot-файлах *(см. разд. 2.8)*, а при последующем запуске восстанавливают окна в прежних размерах и местах, но интерактивным перемещением окон и изменением их размера занимается *оконный менеджер*.

7.3. Оконные менеджеры

Третья важная составляющая оконной системы, позволяющая пользователю *интерактивно* манипулировать окнами, — это оконный менеджер. Одним из самых ранних[1] оконных менеджеров является W:[twm], проиллюстрированный на рис. 7.2 и в листинге 7.9.

Листинг 7.9. Оконный менеджер twm

```
homer@ubuntu:~$ twm &
[5] 2880
```

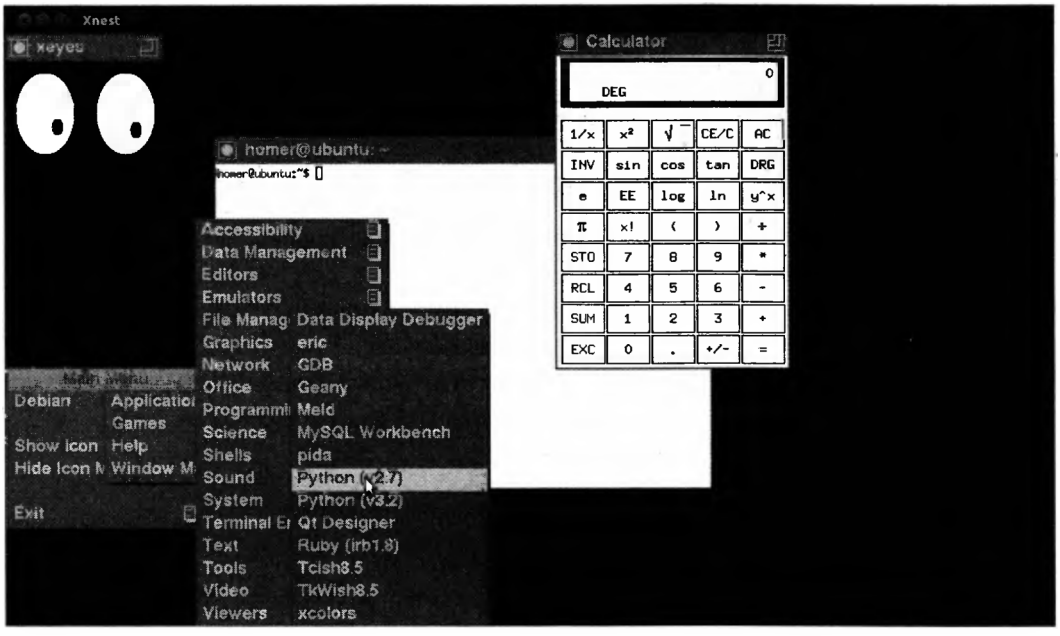


Рис. 7.2. Оконный менеджер twm (Tab Window Manager)

Кроме возможности интерактивного перемещения, изменения размеров окон, сворачивания их в значок на «рабочем столе» и разворачивания[2], оконный менеджер **twm(1)** имеет «главное меню» (вызываемое нажатием левой кнопки мыши непосредственно на «рабочем столе»), позволяющее запускать графические приложения.

Другим, не менее древним, оконным менеджером является W:[olwm], доставшийся в наследство от настольного окружения W:[OpenWindows] из операционной системы

[1] В Ubuntu ® 19.10 заработал только после установки пакета древних растровых шрифтов **xfonts-75dpi**.

[2] Аналогом привычной панели задач в **twm(1)** являются приложения, свернутые в значки, которые располагаются непосредственно на «рабочем столе», т. е. корневом окне X-сервера.

W:[SunOS]. Начиная с Ubuntu Linux 18.04 olwm(1) (что ожидаемо), больше не доступен (листинг 7.10), но можно взглянуть на его W:EN:[look and feel] на рис. 7.3.

Листинг 7.10. Оконный менеджер olwm

```
homer@ubuntu:~$ olwm
☹ Команда "olwm" не найдена. Возможно, вы имели в виду:

  command 'olam' from snap olam (0+git.c66238a)
  command 'lvm' from deb lvm2 (2.03.02-2ubuntu6)
  command 'rlvm' from deb rlvm (0.14-3build1)

See 'snap info <snapname>' for additional versions.
homer@ubuntu:~$ apt search olwm
☹ Сортировка... Готово
Полнотекстовый поиск... Готово
homer@ubuntu:~$ apt-file search bin/olwm
☹
```

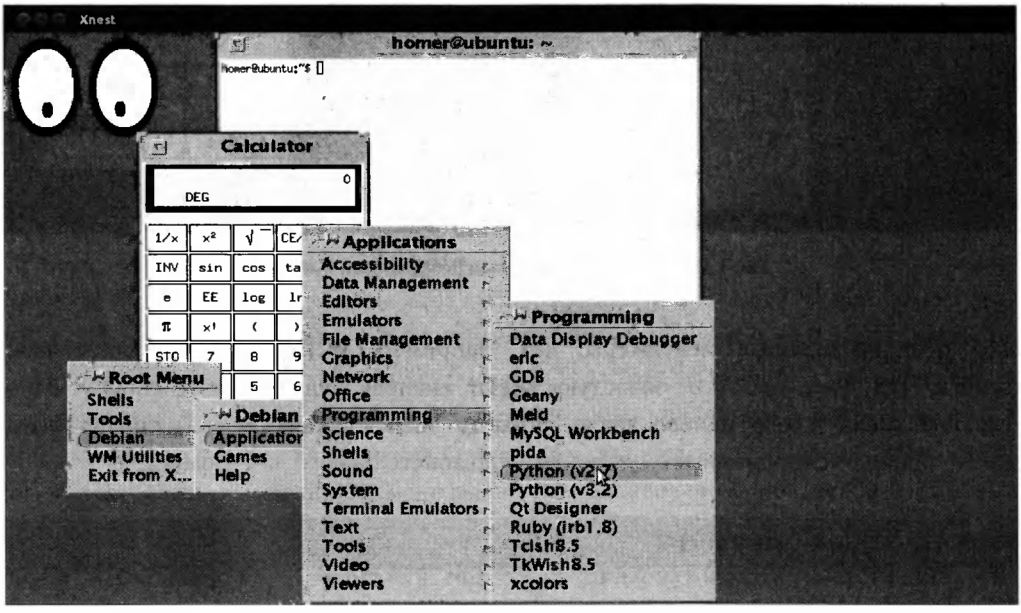


Рис. 7.3. Оконный менеджер olwm (OpenLook Window Manager)

Еще один оконный менеджер «из прошлого», W:[Motif Window Manager], mwm(1), являвшийся частью настольного окружения W:[CDE], проиллюстрирован в листинге 7.11 и на рис. 7.4. Как и twm(1) и olwm(1), окна приложений под управлением mwm(1) при минимизации сворачиваются в значки на «рабочем столе», а главное меню вызывается правой кнопкой мыши.

Листинг 7.11. Оконный менеджер mwm

```
homer@ubuntu:~$ kill %twm
[5]+ Завершён          twm
homer@ubuntu:~$ mwm &
[5] 14090
```

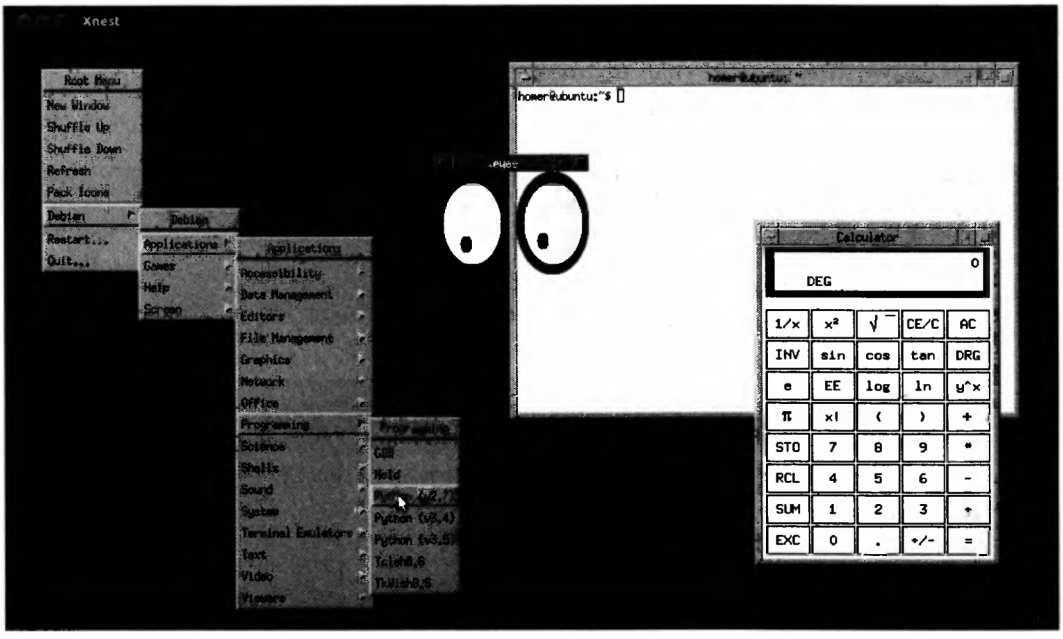


Рис. 7.4. Оконный менеджер mwm (Motif Window Manager)

Более поздние оконные менеджеры, как, например, **W:[IceWM]**, представленный в листинге 7.12 и на рис. 7.5, зачастую имеют «панель задач» снизу, кнопку «Пуск» с главным меню слева панели задач, область уведомлений («трей») справа панели задач и прочие «современные» элементы пользовательского интерфейса.

Листинг 7.12. Оконный менеджер icewm

```
homer@ubuntu:~$ kill -9 %mwm
[5]+ Убито              mwm
homer@ubuntu:~$ icewm-session &
[5] 15315
homer@ubuntu:~$ pstree 15315
icewm-session─┬─icewm
              └─icewmbg
```

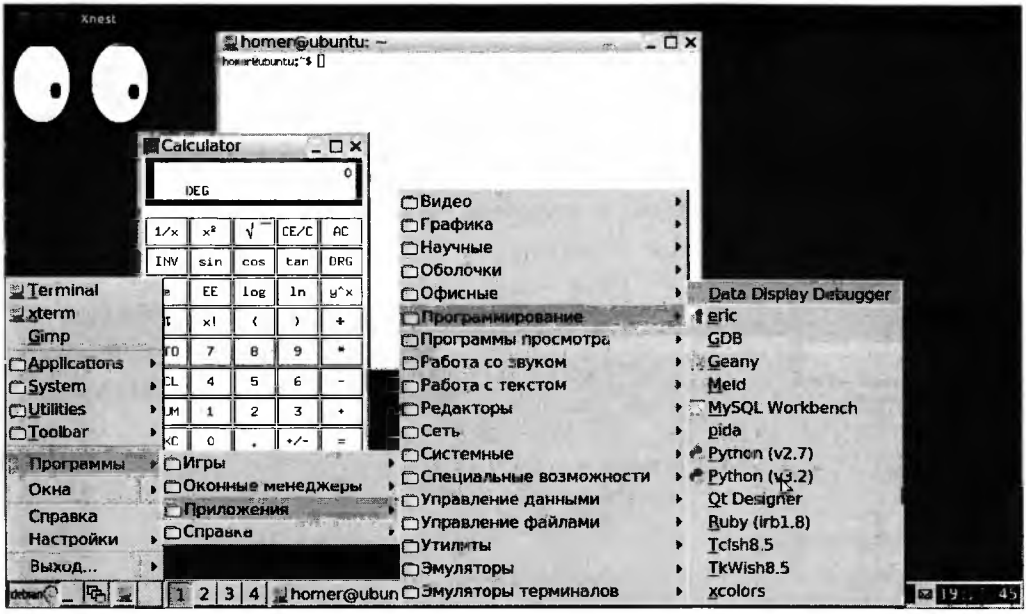


Рис. 7.5. Оконный менеджер icewm (ICE Window Manager)

Нужно заметить ★, что сеанс **icewm(1)** запускается командой **icewm-session(1)** и состоит из двух компонент: самого оконного менеджера **icewm(1)** и «подчиненного» **icewmbg(1)**, управляющего фоном рабочего стола.

7.3.1. Декорирование на клиентской стороне

Оконный менеджер является классической компонентой X Window System, являющейся ее неотъемлемой частью с самого начала развития. Однако в последнее время наблюдается постепенный переход от «серверного» декорирования окон к «клиентскому» декорированию (CSD, **W:[Client-Side Decoration]**). Эти два способа различаются тем, кто́ изображает весь декор окна приложения (заголовок для перетаскивания, бордюр для изменения размера, кнопки сворачивания, разворачивания и закрытия) — оконный менеджер «централизовано» или каждый отдельный X-клиент самостоятельно.

Клиентский способ декорирования позволяет приложениям соответствовать принципам дизайна (на основе) взаимодействия с пользователем (UX, **W:[User experience design]**). Проще говоря, декор окна (например, ранее пустовавший заголовок, зря съедавший свободное место экрана) становится частью приложения и играет активную роль в его жизненном цикле, выполняя не только функции «рукоятки» для перетаскивания.

На текущий момент времени в настольном окружении **GNOME** «клиентское» декорирование является основным, а функции оконного менеджера сохранены только для

«классических» X-клиентов (рис. 7.6). Окна «классических» X-клиентов **xedit(1)** и **xcalc(1)** в отсутствие оконного менеджера размещаются в левом верхнем углу экрана, а перетаскивание и изменение размера невозможно, тогда как **gedit(1)** и **gnome-calculator(1)** имеют заголовок, в котором размещены их главное меню (!) и кнопки сворачивания, разворачивания и закрытия. К тому же они самостоятельно обрабатывают события перетаскивания и изменения размеров, без перекладывания этой задачи на «серверный» оконный менеджер.

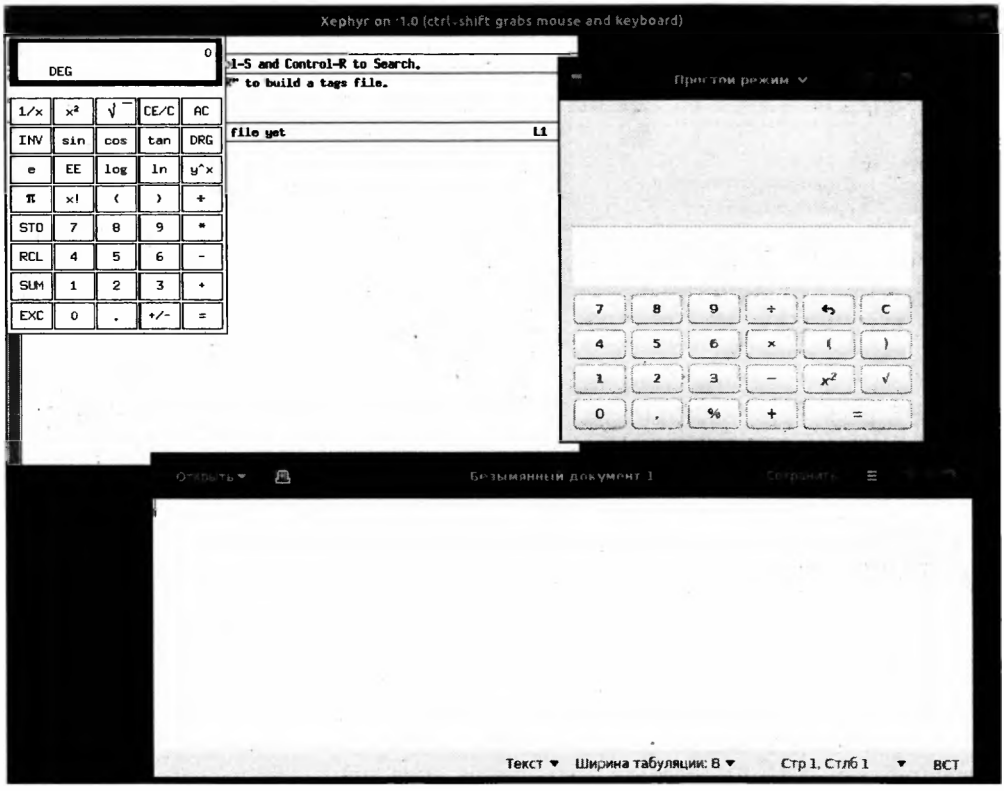


Рис. 7.6. Декорирование на клиентской стороне

7.4. Настольные пользовательские окружения

Современные графические пользовательские среды[1], такие как **W:[GNOME]**, **W:[KDE]**, **W:[XFCE]** и **W:[LXDE]**, с точки зрения оконной системы **X(1)** являются всего лишь набором X-клиентов, таким же как и набор из листингов 7.8 и 7.9 — **xterm(1)**, **xcalc(1)**, **xeyes(1)** и оконный менеджер, например **twm(1)**.

[1] Зачастую используют метафору «рабочего стола», за что именуются «настольным окружением», **W:[Desktop Environment]**.

Существенное отличие состоит в том, что приложения пользовательских сред много *взаимодействуют* между собой (и другими компонентами операционной системы) по некоторым законам[1] и запускаются в определенном порядке *менеджером сеансов*.

На рис. 7.7 и в листинге 7.13 проиллюстрирована среда GNOME, запущенная менеджером сеансов ❶ **gnome-session(1)**, состоящая из достаточно большого ❷ количества компонент, включая композитный *(подробнее см. разд. 7.6.1)* оконный менеджер ❸ **gnome-shell(1)**.

Рис. 7.7. Настольное окружение Ubuntu GNOME

Листинг 7.13. Настольное окружение GNOME

```
homer@ubuntu:~$ DISPLAY=:0 Xephyr :1 -listen tcp &
[1] 24417
```

[1] В современных средах практически безальтернативно используется **W**:**[D-Bus]**, но в более ранних версиях использовались **W**:**[DCOP]** в KDE и **W**:**[CORBA]** в GNOME. Нужно признаться, что ранние X-клиенты тоже взаимодействовали между собой, и особенно — с оконным менеджером при помощи некоторых механизмов, например **W**:**[ICCCM]** и **W**:**[EWMH]**.

```
homer@ubuntu:~$ DISPLAY=:1 dbus-launch --exit-with-session gnome-session &
[2] 26024
homer@ubuntu:~$ pstree -Tp 26024
❶ gnome-session-b(26024)─┬─evolution-alarm(26347)
                         ❸ ├─gnome-shell(26062)─┬─ibus-daemon(26172)─┬─ibus-dconf(26178)
                           │                     │                    ├─ibus-engine-sim(26309)
                           │                     │                    └─ibus-extension-(26179)
                           │                     └─polkit-agent-he(26409)
                           ├─gsd-a11y-settin(26220)
                         ❷ ├─gsd-color(26229)
                           ...                   ...                  ...
                           ├─gsd-xsettings(26226)
                           ├─tracker-extract(26349)
                           └─tracker-miner-f(26361)
```

Стоит отметить, что в листинге 7.13 менеджер сеансов **gnome-session(1)** запускается через посредника — **dbus-launch(1)**, который запустит новую шину для взаимодействия компонент сеанса так, чтобы предыдущий сеанс, работающий с «аппаратным» X-сервером, не пересекался с новым сеансом, запускаемым для «виртуального» **Xephyr(1)**.

Среда **KDE**, представленная в листинге 7.14 и на рис. 7.8, запускается при помощи специального сценария **startkde(1)** и тоже состоит из отдельных компонент ❶, среди которых неизменно присутствуют менеджер сеансов **ksmserver** ❷ и композитный *(см. разд. 7.6.1)* оконный менеджер **kwin** ❸.

Листинг 7.14. Настольное окружение KDE

```
homer@ubuntu:~$ DISPLAY=:0 Xephyr :1 -listen tcp &
[1] 6215
homer@ubuntu:~$ DISPLAY=:1 startkde >& /dev/null
[1] 6236
homer@ubuntu:~$ pstree -Tp 6236
?  startkde(6236)──kwrapper5(6291)
homer@ubuntu:~$ pgrep -l ^kde*
21 kdevtmpfs
6270 kdeinit5
❶ 6274 kded5
❶ 6345 kdeconnectd
homer@ubuntu:~$ pstree -Tp 6270
kdeinit5(6270)─┬─file.so(6618)
               ├─kaccess(6289)
               ├─kded5(6274)
             ❶ ├─klauncher(6271)
               ├─korgac(6358)
             ❷ └─ksmserver(6293)──kwin_x11(6311) ❸
```

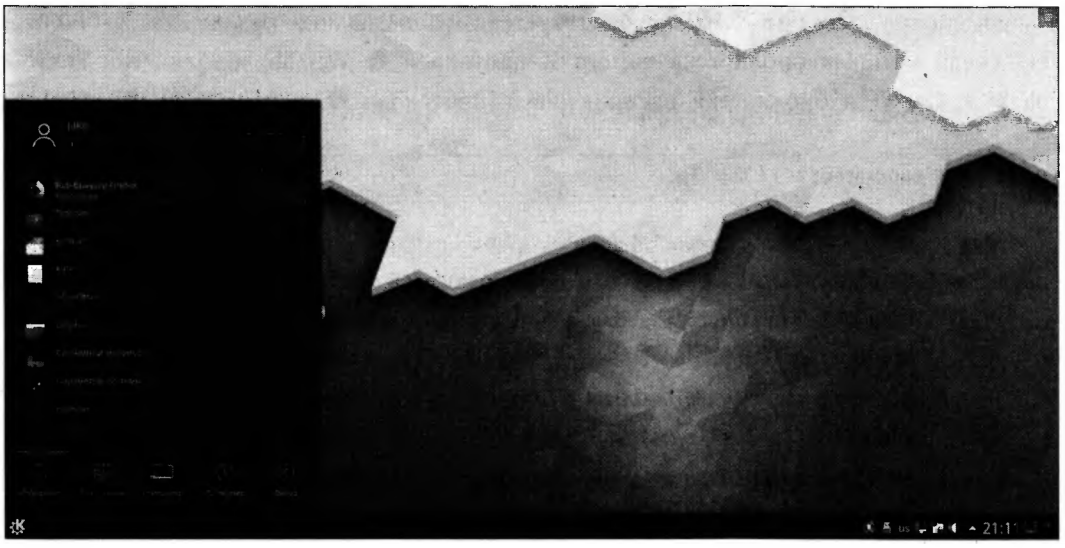


Рис. 7.8. Настольное окружение KDE plasma

7.5. Библиотеки интерфейсных элементов

Возможности X-протокола позволяют прорисовывать лишь простейшие графические примитивы: линии, точки, прямоугольники, дуги, полигоны, растровые изображения и текст — при помощи растровых или векторных шрифтов, поддерживаемых X-сервером. Все интерфейсные элементы управления приложениями, такие как кнопки, полосы прокрутки, закладки, меню, списки и прочее, должны быть самостоятельно составлены X-клиентами из простейших примитивов. Для решения этой задачи X-клиенты используют специальные *библиотеки интерфейсных элементов*[1] (*виджетов*, widget), такие как **W:[Xaw]**, **Xview**, **W:[Motif]**, **W:[Tk]**, **W:[Qt]**, **W:[Gtk]** и др.[2]

Самими «простыми» (доставшимися в наследство от UNIX) библиотеками виджетов являются библиотеки **Xaw** и **Motif**, разработанные на основе базовой **Xt**, **W:[X Toolkit Intrinsics]**, а также «альтернативная»[3] **XView**[4].

Более «современным» видом виджетов обладают наиболее распространенные на сегодняшний день библиотеки **Qt** и **Gtk**, на основе которых разрабатываются пользовательские среды **KDE** и **GNOME**.

---

[1] Так же называемые тулкитами от англ. widget *toolkit* или GUI *toolkit*.

[2] См. **W:[wxWidgets]**, **W:[FLTK]**.

[3] См. **W:[OpenLook]**

[4] Отсутствующая в ● 19.10 вместе с OpenLook Window Manager.

В примере из листинга 7.15 иллюстрируется использование различных X-библиотек, среди которых библиотека самого X-протокола ❶ W:[Xlib], библиотеки виджетов Xt и Xaw ❷ и библиотеки расширений X-протокола ❸, например W:[XRender].

Листинг 7.15. Библиотеки Xlib и Xt и Xaw

```
homer@ubuntu:~$ ldd $(which xeyes) | grep -i libX
        libXext.so.6 => /lib/x86_64-linux-gnu/libXext.so.6 (0x00007f9f16776000)
        libXmu.so.6 => /lib/x86_64-linux-gnu/libXmu.so.6 (0x00007f9f1675a000)
❷       libXt.so.6 => /lib/x86_64-linux-gnu/libXt.so.6 (0x00007f9f164f1000)
❶       libX11.so.6 => /lib/x86_64-linux-gnu/libX11.so.6 (0x00007f9f163b3000)
❸       libXrender.so.1 => /lib/x86_64-linux-gnu/libXrender.so.1 (0x00007f9f161a9000)
        libxcb.so.1 => /lib/x86_64-linux-gnu/libxcb.so.1 (0x00007f9f15e15000)
        libXau.so.6 => /lib/x86_64-linux-gnu/libXau.so.6 (0x00007f9f15de4000)
        libXdmcp.so.6 => /lib/x86_64-linux-gnu/libXdmcp.so.6 (0x00007f9f15ddc000)

homer@ubuntu:~$ ldd $(which xcalc) | grep -i Xaw
❷       libXaw.so.7 => /lib/x86_64-linux-gnu/libXaw.so.7 (0x00007fbd4f743000)
```

Аналогично, в листинге 7.16 показан X-клиент, использующий библиотеку виджетов Motif, а в листинге 7.17 — X-клиенты, использующие Tk, Qt и Gtk.

Листинг 7.16. Библиотека декорирования Motif

```
homer@ubuntu:~$ ldd $(which mwm) | grep -i libXm
        libXm.so.4 => /lib/x86_64-linux-gnu/libXm.so.4 (0x00007fdd8686a000)
        libXmu.so.6 => /lib/x86_64-linux-gnu/libXmu.so.6 (0x00007fdd862a1000)
```

Листинг 7.17. Библиотеки декорирования Tk, Gtk и Qt

```
homer@ubuntu:~$ ldd $(which wish) | grep -i libTk
    libtk8.6.so => /lib/x86_64-linux-gnu/libtk8.6.so (0x00007fa8a09eb000)
homer@ubuntu:~$ ldd $(which gnome-shell) | grep -i libGtk
    libgtk-3.so.0 => /lib/x86_64-linux-gnu/libgtk-3.so.0 (0x00007f7b3e3de000)
homer@ubuntu:~$ ldd $(which kcalc) | grep -i libQt.Gui
    libQt5Gui.so.5 => /lib/x86_64-linux-gnu/libQt5Gui.so.5 (0x00007f69b75cf000)
```

В соответствии с идеями, заложенными в X Window System, первые тулкиты, такие как Xaw, Motif или Tk, пользовались иерархией окон для формирования своих виджетов, т. е. каждый элемент управления как минимум являлся X-окном (и мог содержать дочерние окна). Впоследствии в Qt и Gtk отказались от такого подхода,

потому что он приводил к «фликерингу[1]» при отрисовке в силу независимости окон и неодновременности их обработки.

В современных тулкитах (листинг 7.18) в большинстве случаев приложение имеет всего лишь одно серверное X-окно, а элементы управления отрисовываются целиком в его пределах только тулкитом X-клиента, без участия X-сервера.

Листинг 7.18. Окна X-клиентов

```
   homer@ubuntu:~$ xcalc & gnome-calculator &
   [1] 1668
   [2] 1669
homer@ubuntu:~$ xwininfo -tree -root
       0x400005b "Calculator": ("xcalc" "XCalc")  226x308+10+38  +50+107
          1 child:
          0x400005c (has no name): ()  226x308+0+0  +50+107
             41 children:
             0x40000d9 (has no name): ()  214x48+4+4  +54+111
                1 child:
                0x40000da (has no name): ()  204x38+4+4  +59+116
                   7 children:
                   0x40000e1 (has no name): ()  10x15+4+2  +64+119
5x                      ...                 ...                 ...
                   0x40000db (has no name): ()  18x15+127+21  +187+138
             0x40000d6 (has no name): ()  40x26+4+66  +54+173
             0x40000d3 (has no name): ()  40x26+48+66  +98+173
37x                     ...                 ...                 ...
             0x4000061 (has no name): ()  40x26+180+276  +230+383
          ...                 ...                 ...
    0x4200007 "Калькулятор": ("gnome-calculator" "Gnome-calculator")  448x467+70+62  +70+62
       1 child:
       0x4200008 (has no name): ()  1x1+-1+-1  +69+61
       ...                 ...                 ...
    0x4200001 "gnome-calculator": ("gnome-calculator" "Gnome-calculator")  10x10+10+10  +10+10
```

[1] См. http://tiny.cc/dgb5hz.

7.6. Расширения X-протокола

Оконная система X(1) и ее X-протокол на текущий момент времени обросли массой *расширений*, одним из которых является расширение W:[MIT-SHM], широко используемое современными библиотеками виджетов для организации наиболее эффективного обмена растровыми изображениями между X-клиентами и X-сервером. Ранние тулкиты, например Xaw и Motif, практически полностью полагались на примитивы векторной графики X-протокола, тогда как современные Gtk или Qt в гораздо большей степени используют растровые изображения (для визуальных эффектов типа выпуклых кнопок, градиентной заливки и т. д.), подготавливаемые целиком на X-клиенте и затем пересылаемые X-серверу для вывода.

Очевидно, что пересылка большого количества растровых изображений с использованием «последовательных» средств межпроцессного взаимодействия, таких как сокеты[1], заставляет жертвовать либо визуальными эффектами, либо производительностью оконной системы. Однако если заметить, что в большинстве «настольных» применений оконной системы X-клиенты и X-сервера в реальности выполняются на одном и том же узле, то очевидно, что разделяемая память является наиболее эффективным средством IPC. Вместе с тем необходимо констатировать факт утери пресловутой «сетевой прозрачности» в угоду приемлемой производительности визуальных эффектов.

Листинг 7.19. Расширения X-сервера: MIT-SHM

```
homer@ubuntu:~$ xdpyinfo | grep -i shm
    MIT-SHM

homer@ubuntu:~$ ipcs -mp
------ Shared Memory Creator/Last-op PIDs --------
shmid      owner      cpid       lpid
           ...        ...        ...        ...
15         homer      9190       8498
20         homer      8588       8498
           ...        ...        ...
27         homer      9514       8498
31         homer      9661       8498

homer@ubuntu:~$ ps up 8498,9190,8588,9514,9661
USER  PID   %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
homer 8498  0.0  1.4 256652 53136 tty2     Sl+  06:07   0:28 /usr/lib/xorg/Xorg vt2 ...
homer 8588  0.1  6.0 2775368 222432 ?      Ssl  06:07   1:10 /usr/bin/gnome-shell
```

[1] Даже при использовании файловых сокетов, не говоря уже о сетевых.

```
homer 9190   0.0  4.3 1031120 158448 ?      SLl  06:08   0:08 /usr/bin/gnome-software ...
homer 9514   0.2  1.4 822552 52580 pts/0    Sl   16:07   0:00 gedit
homer 9661   0.4  1.1 590088 42332 pts/0    Sl   16:08   0:01 gnome-disks

homer@ubuntu:~$ pmap 8498 | grep shmid
00007f3b3d792000    512K rw-s-   [ shmid=0x14 ]
00007f3b3d93c000    512K rw-s-   [ shmid=0x1f ]
00007f3b3d9bc000    512K rw-s-   [ shmid=0x1b ]
00007f3b3da3c000    512K rw-s-   [ shmid=0x17 ]
00007f3b3dabc000    512K rw-s-   [ shmid=0xf ]
00007f3b3db3c000    512K rw-s-   [ shmid=0x18 ]

homer@ubuntu:~$ pmap 9514 | grep shmid
00007f09bc196000    512K rw-s-   [ shmid=0x1b ]
```

В листинге 7.19 показано, что практически все современные X-клиенты имеют для взаимодействия с X-сервером помимо файлового сокета (см. листинг 7.7) еще и сегмент разделяемой памяти System V IPC.

Расширение **W**:[RANDR] (**R**esize **AND** **R**otate) позволят управлять сменой графического разрешения, ориентацией, подключением и отключением видеовыходов дисплея X-сервера и т. д. В примере из листинга 7.20 иллюстрируется утилита **xrandr(1)**, реализующая расширение RANDR, которая может использоваться как для запроса ❶ списка видеовыходов (и поддерживаемых ими видеорежимов работы), так и для переключения ❷ заданного видеовыхода в тот или иной видеорежим и смены ориентации изображения.

Листинг 7.20. Расширения X-сервера: RANDR

```
homer@ubuntu:~$ xdpyinfo | grep -i randr
    RANDR
❶ homer@ubuntu:~$ xrandr
Screen 0: minimum 1 x 1, current 1600 x 900, maximum 8192 x 8192
LVDS1 connected 1600x900+0+0 (normal left inverted right x axis y axis) 423mm x 238mm
   1600x900      60.00*+
   1440x900      59.89
                  ...          ...          ...          ...
   800x600       60.32
   640x480       59.94
VGA1 disconnected (normal left inverted right x axis y axis)
HDMI1 disconnected (normal left inverted right x axis y axis)
DP1 disconnected (normal left inverted right x axis y axis)
```

```
homer@ubuntu:~$ xrandr --output LVDS1 --mode 1024x768 --rotate left
```

Расширение W:[Render] (листинг 7.21) широко используется современными тулкитами, такими как Gtk или Qt, для отрисовки полупрозрачных фигур и сглаживания контуров *(anti-aliasing)*, а также библиотекой отрисовки векторных шрифтов W:[Xft]) для их сглаживания.

Листинг 7.21. Расширения X-сервера: RENDER

```
homer@ubuntu:~$ xdpyinfo | grep -i render
    RENDER
homer@ubuntu:~$ ldd /lib/x86_64-linux-gnu/libgtk-3.so.0 | grep -i xrender
    libXrender.so.1 => /lib/x86_64-linux-gnu/libXrender.so.1 (0x00007fc107a6c000)
homer@ubuntu:~$ ldd /usr/lib/x86_64-linux-gnu/libXft.so.2 | grep Xrender
    libXrender.so.1 => /lib/x86_64-linux-gnu/libXrender.so.1 (0x00007fe48a4ca000)
```

7.6.1. Расширение Composite и композитный менеджер

Расширение **Composite** (листинг 7.22) заслуживает отдельного внимания. Изначально оконная система была устроена так[1], что любые команды рисования в окне непосредственно изменяли содержимое в том месте экранного буфера[2] (*frame buffer*, он же *front buffer*), который соответствует этому окну. Для окон, перекрытых другими окнами, команды рисования выполнялись X-сервером только в открытых их частях, и не существовало никакой возможности узнать полное содержимое окна, поскольку оно попросту отсутствовало и в памяти X-клиента, и в памяти X-сервера. Однако с развитием настольных окружений и простых оконных менеджеров появился запрос на визуальные эффекты, такие как полупрозрачность окон при перекрытии, размытие, вращение, проявления или растворения окон при их появлении или сокрытии, анимация, масштабирование и пр. Все эти эффекты требуют постоянного знания полных содержимых всех окон, из которых затем можно скомбинировать *(composite)* любую визуальную сцену. Именно расширение **Composite** позволяет попросить X-сервер отрисовывать окна во внеэкранных буферах[3] *(off screen buffer)*, которые затем доступны композитному менеджеру (compositing manager) для составления результирующей визуальной сцены. Простейшим композитным менеджером является **xcompmgr(1)**, иллюстрирующий как само расширение **Composite**, так и принцип композитинга в целом.

---

[1] В первую очередь в силу ограниченности ресурса оперативной памяти.

[2] Буфер, который немедленно изображается видеокартой на дисплее.

[3] Естественно, за счет потребления индивидуальной памяти под каждое окно, но память больше не является сверхдорогим ресурсом.

Важно понимать, что *композитный* менеджер — такая же отдельная компонента оконной системы, как и *оконный* менеджер *(см. разд. 7.3)*. Однако очень часто функции оконного и композитного менеджеров совмещаются в *композитных оконных менеджерах* W:[Compositing window manager], таких как kwin из настольного окружения KDE или gnome-shell(1) из настольного окружения GNOME *(см. разд. 7.4)*.

Листинг 7.22. Расширения X-сервера: Composite

```
homer@ubuntu:~$ xdpyinfo | grep -i composite
    Composite
homer@ubuntu:~$ ldd /usr/bin/gnome-shell | grep -i composite
    libXcomposite.so.1 => /lib/x86_64-linux-gnu/libXcomposite.so.1 (0x00007f21b3b55000)
```

Кроме того, широко распространено заблуждение, что композитные менеджеры обязательно нуждаются в поддержке OpenGL в общем и в аппаратно-ускоренном 3D-рендеринге в частности, что совсем не соответствует действительности. Примером тому служит все тот же xcompmgr(1), полагающийся только на расширение RENDER при составлении результирующей визуальной сцены. Несомненно, если композитный менеджер реализует визуальные 3D-эффекты, как это делает W:[Compiz], то поддержка OpenGL будет необходимым условием, но это не имеет никакого отношения к самому принципу композитинга.

7.6.2. GLX, DRI и 3D-графика

Расширение W:[GLX], получившее свое название по имени специального программного интерфейса GLX[1], предназначено для передачи «команд» OpenGL внутри X-протокола. Программный интерфейс OpenGL используется приложениями, интенсивно работающими с 3D-графикой (а иногда и 2D, в большинстве случаев[2] — играми, системами моделирования и визуализации) и нацеленными на аппаратно-ускоренный рендеринг «команд» OpenGL.

Утилита glxinfo(1), показанная в листинге 7.23, предназначена для запроса свойств GLX-расширения и свойств самого OpenGL-рендерера.

---

[1] GLX (OpenGL for X Window System) сопрягает W:[OpenGL] с оконной системой X(1), т. к. спецификации OpenGL касаются только предмета 3D-рендеринга, но никак не затрагивают вопросы непосредственного отображения результата, например в окно X Window System).

[2] Хотя на сегодняшний день даже «обычные» пользовательские окружения GNOME и KDE интенсивно используют OpenGL для спецэффектов.

Листинг 7.23. Расширения X-сервера: GLX

```
homer@ubuntu:~$ xdpyinfo | grep GLX
    GLX

homer@ubuntu:~$ glxinfo | grep -E "render(er|ing)|version"
direct rendering: Yes
server glx version string: 1.4
client glx version string: 1.4
GLX version: 1.4
                ...          ...          ...          ...
OpenGL renderer string: Mesa DRI Intel(R) Sandybridge Mobile
OpenGL version string: 2.1 Mesa 19.2.1
OpenGL shading language version string: 1.20
OpenGL ES profile version string: OpenGL ES 2.0 Mesa 19.2.1
OpenGL ES profile shading language version string: OpenGL ES GLSL ES 1.0.16
```

Различают два режима работы GLX — косвенный *(indirect rendering)* и прямой *(direct rendering)*. Косвенный режим действительно предполагает передачу команд OpenGL от X-клиента через GLX-расширение X-протокола к X-серверу, который, в свою очередь, отрисует 3D-сцену. Сам отрисовка (рендеринг) будет выполнена либо с использованием графического процессора (GPU, **G**raphics **P**rocessing **U**nit), либо программно, что потребует значительных ресурсов центрального процессора (CPU, **C**entral **P**rocessing **U**nit). Такой режим все еще сохраняет свойство сетевой прозрачности, т. е. позволяет X-клиенту и X-серверу работать на разных узлах сети, но крайне неэффективен[1] при значительных объемах данных (текстуры, массивы вершин и пр.), которые клиент должен передать на сервер для запуска рендеринга.

Режим прямой отрисовки предполагает, что X-клиенты и X-сервер имеют прямой (непосредственный) доступ *к одному и тому же* GPU, что позволяет существенно увеличить производительность отрисовки, за счет уменьшения накладных расходов на работу GLX-расширения и на передачу команд от клиентов к серверу. В таком случае даже результат отрисовки клиента не требует передачи серверу, т. к. напрямую доступен ему для отображения из GPU. Однако в очередной раз возникает побочная задача разделения устройства между потребителями, а именно GPU видеоускорителя и его памяти между X-сервером и некоторым количеством X-клиентов. Вместе с тем, надо заметить, исчезает и сетевая прозрачность, т. к. теперь X-клиенты и X-серверы вынуждены работать на одном и том же узле в угоду производительности.

---

[1] С развитием возможностей 3D-графики, OpenGL и аппаратных видеоускорителей стало очевидно, что X-протокол крайне неэффективен в трансляции OpenGL-команд.

Задачу разделения GPU и памяти видеоускорителей между любыми программами операционной системы (не только между X-клиентами и X-сервером) решает специальная компонента ядра Linux, называемая менеджером прямой отрисовки — W:[DRM] *(Direct Renedering Manager)*. Для согласования взаимодействия между X-клиентами и X-сервером при доступе к ресурсам DRM понадобилось еще одно расширение X-протокола, называемое W:[DRI] *(Direct Renedering Infrastructure)*.

Листинг 7.24. Расширения X-сервера: DRI

```
homer@ubuntu:~$ xdpyinfo | grep DRI
    DRI2
    DRI3
homer@ubuntu:~$ xdriinfo
Screen 0: i965
```

В листинге 7.24 показано, что при помощи **xdpyinfo(1)** можно узнать, поддерживает ли X-сервер расширение DRI и каких версий, а **xdriinfo(1)** демонстрирует, какие видеоускорители доступны X-серверу и идентификаторы их DRI-драйверов, используемые программным интерфейсом GLX при прямой отрисовке.

Стоит заметить, что все вышеперечисленные утилиты — **xdpyinfo(1)**, **xrandr(1)**, **glxinfo(1)** и даже **xdriinfo(1)** — также являются X-клиентами и используют X-протокол для опроса X-сервера.

7.7. Запуск X Window System

7.7.1. Локальный запуск X-клиентов

Изначально в UNIX-системах, в том числе в Linux, основным интерфейсом пользователя был командный, на алфавитно-цифровом терминале, а графический интерфейс запускался после входа пользователя в систему при помощи **xinit(1)** и/или **startx(1)**, что совсем утратило свою актуальность на сегодняшний день. Основным интерфейсом пользователя стал GUI, запускающийся менеджерами X-дисплеев *(см. разд. 7.7.3)* вместе со стартом операционной системы (см. листинг 10.3.1).

7.7.2. Дистанционный запуск X-клиентов

Оконная система X изначально проектировалась для распределенной работы ее компонент на различных узлах сети, что достаточно широко используется на практике, когда нужно запускать графические приложения на удаленном узле, их окна отображать на локальном дисплее.

В большинстве случаев для дистанционного запуска используется **ssh(1)**, что проиллюстрировано в листинге 7.25. Попытка ❶ «прямого» запуска **xeyes(1)** на узле **centos** от лица пользователя **lich** не увенчалась успехом потому, что в большинстве инсталляций аппаратный X-сервер на дисплее **:0** не принимает сетевые соединения (см. листинг 7.25). Попытка ❷ запуска локального виртуального сервера **Xnest(1)** на дисплее **:1** с перенаправлением ему вывода дистанционного **xeyes(1)** тоже оказалась неудачной, но уже по другим причинам *(см. далее)*.

Листинг 7.25. Запуск дистанционного X-клиента

```
❶ homer@ubuntu:~$ ssh -f lich@centos "DISPLAY=ubuntu.local:0 xeyes"
  Error: Can't open display: ubuntu.local:0
❷ homer@ubuntu:~$ Xnest :1 &
                  ...          ...          ...          ...
  homer@ubuntu:~$ ssh -f lich@centos "DISPLAY=ubuntu.local:1 xeyes"
☛ No protocol specified
  Error: Can't open display: ubuntu.local:1
```

При сетевом взаимодействии X-клиентов и X-сервера для аутентификации клиентских подключений используется механизм, основанный на предъявлении общего (известного обеим сторонам) «секрета», называемого «волшебной печенькой» (см. W:[magic cookie]), использование которой проиллюстрировано в примере из листинга 7.26.

На стороне сервера «печеньки» всех клиентов, которым разрешено подключение, размещаются при помощи утилиты **xauth(1)** в «банке с печеньками» (jar) ❶, откуда извлекаются сервером для проверки при подключении клиента. На стороне клиента «печеньки» при помощи той же утилиты **xauth(1)** размещаются ❷ в «банке» **~/.Xauthority**, откуда извлекаются библиотекой **Xlib** для предъявления серверу при соединении с ним.

Листинг 7.26. Аутентификация дистанционного X-клиента

```
  homer@ubuntu:~$ mcookie
  8f36c904dc0c9934c506c21ea7860eb2
❶ homer@ubuntu:~$ xauth -f cookie-jar ☚
  xauth:  file cookie-jar does not exist
  Using authority file cookie-jar
  xauth> add ubuntu:1 MIT-MAGIC-COOKIE-1 ☚ 8f36c904dc0c9934c506c21ea7860eb2 ↵
  xauth> exit↵
  Writing authority file cookie-jar
```

```
homer@ubuntu:~$ Xnest :1 -auth cookie-jar ➧ &
                ...      ...      ...      ...
homer@ubuntu:~$ ssh lich@centos
                                                    lich@centos:
Last login: Fri Jan  8 17:43:04 2016 from ubuntu
[lich@centos ~]$ xauth
❷ xauth:  file /home/lich/.Xauthority does not exist
Using authority file /home/lich/.Xauthority
xauth> add ubuntu.local:1 MIT-MAGIC-COOKIE-1 ➧ 8f36c904dc0c9934c506c21ea7860eb2 ↵
xauth> exit↵
Writing authority file /home/lich/.Xauthority
[lich@centos ~]$ logout
Connection to centos closed.
❸ homer@ubuntu:~$ ssh -f lich@centos "DISPLAY=ubuntu.local:1 xeyes"
```

Необходимость установки общего секрета, переменной окружения **DISPLAY** и запуска дополнительного виртуального X-сервера (или активации приема сетевых соединений аппаратным сервером) делают «ручной» запуск дистанционных X-клиентов неудобным «чуть более чем полностью». Вместе с этим передача «волшебных печенек» (как и любых других сообщений X-протокола) от X-клиента к X-серверу по сети происходит незащищенным образом, что может быть легко использовано злоумышленником. Именно поэтому на практике используют *туннелирующие* возможности протокола SSH, позволяющие удобным автоматизированным способом решить все вышеперечисленные задачи и проблемы.

В примере из листинга 7.27 показано поведение SSH-сервера при туннелировании X-протокола. По запросу (**-X**) от SSH-клиента SSH-сервер начинает эмулировать ❹ поведение X-сервера, устанавливает ❶ переменную окружения **DISPLAY**, указывающую на «дисплей» **:10** на «том же» узле **localhost**, и создает ❷ «волшебную печеньку» для этого дисплея.

При последующем запуске ❸ X-клиента **xeyes(1)** им будет установлено соединение с «SSH-эмулятором» X-сервера на **localhost:10**, а SSH-сервер перенаправит (туннелирует) это соединение X-протокола обратно SSH-клиенту внутри зашифрованного соединения SSH.

Листинг 7.27. SSH-туннелирование X-протокола (SSH-сервер)

```
homer@ubuntu:~$ ssh -X lich@centos
Last login: Fri Jan  8 17:48:34 2016 from ubuntu
[lich@centos ~]$ echo $DISPLAY
❶ localhost:10.0
```

```
[lich@centos ~]$ xauth list
❷ centos/unix:10  MIT-MAGIC-COOKIE-1  80c749073282be2001c33bd43e577aa4 ↩
❸ [lich@centos ~]$ strace -fe connect xeyes
                   ...             ...             ...             ...
connect(3, {sa_family=AF_INET, sin_port=htons(6010), sin_addr=inet_addr("127.0.0.1")}, 16) = 0
                   ...             ...             ...             ...
+++ exited with 0 +++
[lich@centos ~]$ sudo lsof -nP -i 4:6010
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
❹ sshd    14221 lich   10u  IPv4  44578      0t0  TCP 127.0.0.1:6010 (LISTEN)
[lich@centos ~]$ logout
Connection to centos closed.
```

Листинг 7.28 иллюстрирует поведение SSH-клиента в режиме (**-X**) X-туннелирования, при котором он эмулирует «X-клиента» и соединяется с аппаратным (!) X-сервером через локальный сокет **/tmp/.X11-unix/X0**. При получении от SSH-сервера перенаправленных соединений X-протокола они ретранслируются SSH-эмулятором «X-клиента» локальному X-серверу, тем самым вывод *дистанционных* X-клиентов изображается в окнах *локального* X-сервера.

Листинг 7.28. SSH-туннелирование X-протокола (SSH-клиент)

```
homer@ubuntu:~$ strace -fe connect ssh -f -X lich@centos xeyes
                ...             ...             ...             ...
connect(3, {sa_family=AF_INET, sin_port=htons(22), sin_addr=inet_addr("10.0.0.99")}, 16) = 0
                ...             ...             ...             ...
❺ connect(7, {sa_family=AF_FILE, path="/tmp/.X11-unix/X0"↩}, 110) = 0
```

Так как SSH-туннелирование X-протокола позволяет перенаправлять любое количество X-соединений внутри одного SSH-соединения, то дистанционно можно запускать целые сеансы пользовательских сред, таких как GNOME (листинг 7.29), с использованием **xinit(1)** и виртуального X-сервера **Xnest(1)**.

Листинг 7.29. Запуск дистанционного сеанса GNOME с отображением на локальном сервере Xnest

```
homer@ubuntu:~$ xinit /usr/bin/ssh -X lich@centos gnome-session -- /usr/bin/Xnest :4 &
                ...             ...             ...             ...
homer@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
27759 pts/0    Ss     0:00 bash
27829 pts/0    S      0:00  \_ xinit /usr/bin/ssh -X lich@centos gnomes-session -- ...
27830 pts/0    Sl     0:00  |   \_ /usr/bin/Xnest :4
27837 pts/0    S      0:00  |   \_ /usr/bin/ssh -X lich@centos gnome-session
27928 pts/0    R+     0:00  \_ ps f
```

7.7.3. Управление X-дисплеями: XDMCP-менеджер и протокол

В большинстве современных инсталляций Linux работа пользователей в системе осуществляется сразу с использованием оконной системы X и какой-либо пользовательской среды, например GNOME. Если при работе в (автоматически запущенной) оконной системе (листинг 7.30) проследить дерево процессов от аппаратного X-сервера **Xorg(1)** до прародителя процессов **init(1)**, то обнаружится *менеджер дисплеев* ❶ (например, **gdm3(1)**), осуществивший запуск ❷ X-сервера. Более того, пользовательский сеанс ❸ (**gnome-session(1)**) тоже окажется запущенным этим менеджером.

Листинг 7.30. Локальный графический вход

```
homer@ubuntu:~$ pgrep Xorg
21261
homer@ubuntu:~$ pstree -sTp 21261
systemd(1)—gdm3(742)—gdm-session-wor(21227)—gdm-x-session(21259)—Xorg(21261)
                ❶                                    ←                  ❷

homer@ubuntu:~$ pstree -Tp 742
gdm3(742)—gdm-session-wor(21227)—gdm-x-session(21259)─┬─Xorg(21261)
                                                      └─gnome-session-b(21267)
  ❶                                  →                          ❸

homer@ubuntu:~$ ps fp 742,21227,21259,21261,21267
  PID TTY      STAT   TIME COMMAND
  742 ?        Ssl    0:00 /usr/sbin/gdm3
21227 ?        Sl     0:00  \_ gdm-session-worker [pam/gdm-password] ②
21259 tty7     Ssl+   0:00    \_ /usr/lib/gdm3/gdm-x-session ...
21261 tty7     Sl+    0:00    \_ /usr/lib/xorg/Xorg vt7 -displayfd 3 ... ①
21267 tty7     Sl+    0:00      \_ /usr/lib/gnome-session/gnome-session-binary --systemd... ③

homer@ubuntu:~$ pgrep -l gnome-shell
21604 gnome-shell
21655 gnome-shell-cal

homer@ubuntu:~$ pstree -sTp 21604
systemd(1)—systemd(5527)—gnome-shell(21604)—ibus-daemon(21707)┬ibus-dconf(21714)
                                                               ├ibus-...-sim(21922)
                                                               └ibus-...-(21716)
```

Менеджер дисплеев является специальной компонентой оконной системы, управляющей автоматическим запуском ее X-серверов и X-сеансов. Именно он запускает аппаратные X-серверы для обслуживания дисплеев в указанном количестве (по

умолчанию один дисплей ①), «графическим» образом производит аутентификацию ② (*см. разд. 2.2.1*, рис. 2.4) пользователя в системе и запускает менеджер сеансов ③ пользовательской среды.

Кроме этого, менеджер дисплеев предназначался ранее и для «автоматического» запуска *дистанционных (см. разд. 7.7.2)* пользовательских X-сеансов при помощи специального протокола **W:[XDMCP]**, а не посредством протокола SSH, как при «ручном» дистанционном их запуске.

Надо заметить, что при использовании дистанционного XDMCP-запуска данные XDMCP- и X-протоколов (включая согласование «волшебной печеньки» и пароля пользователя) при передаче в публичной сети оказываются никак не защищены. Поэтому в современной практике безальтернативно используют ручной SSH-запуск дистанционных сеансов с SSH-туннелированием X-протокола, отключая поддержку XDMCP насовсем.

7.8. Программный интерфейс X Window System

7.8.1. Трассировка X-библиотек и X-протокола

Программный интерфейс оконной системы X, являющейся обычной сетевой службой, представлен библиотеками и соответствующими (библиотечными) вызовами к ним. Естественным образом, при наблюдении за работой разнообразных компонент утилита **strace(1)** трассировки системных вызовов (к ядру) оказывается далеко не лучшим инструментом, а ее место занимает утилита трассировки библиотечных вызовов **ltrace(1)**.

В примере из листинга 7.31 показан пример трассировки библиотечных вызовов библиотек **Xt**, **Xlib** и **Xrender** при работе простейшего демо-клиента **xeyes(1)**.

Листинг 7.31. Трассировка библиотек X Window System

```
homer@ubuntu:~$ ldd $(which xeyes) | grep libX
        libXext.so.6 => /lib/x86_64-linux-gnu/libXext.so.6 (0x00007f2c63c4e000)
        libXmu.so.6 => /lib/x86_64-linux-gnu/libXmu.so.6 (0x00007f2c63c32000)
        libXt.so.6 => /lib/x86_64-linux-gnu/libXt.so.6 (0x00007f2c639c9000)
        libX11.so.6 => /lib/x86_64-linux-gnu/libX11.so.6 (0x00007f2c6388b000)
        libXrender.so.1 => /lib/x86_64-linux-gnu/libXrender.so.1 (0x00007f2c63681000)
        libXau.so.6 => /lib/x86_64-linux-gnu/libXau.so.6 (0x00007f2c632bc000)
        libXdmcp.so.6 => /lib/x86_64-linux-gnu/libXdmcp.so.6 (0x00007f2c632b4000)

homer@ubuntu:~$ ltrace -x \
> XOpenDisplay+XCreateWindow+XQueryPointer+XFillRectangle+XRenderCompositeDoublePoly \
> xeyes
```

```
   XOpenDisplay@libX11.so.6(nil)                                    = 0x558cb64e8a70
❶  XCreateWindow@libX11.so.6(0x558cb64e8a70, 380, 0, 0)                    = 0xe0000a
   XCreateWindow@libX11.so.6(0x558cb64e8a70, 0xe0000a, 0, 0)               = 0xe0000b
   XFillRectangle@libX11.so.6(0x558cb64e8a70, 0xe0000c, 0x558cb65087a0, 0) = 1
   XRenderCompositeDoublePoly@libXrender.so.1(0x558cb64e8a70, 3, 0xe00008, 0xe0000e) = 0
                    ...          ...          ...          ...
❷  XQueryPointer@libX11.so.6(0x558cb64e8a70, 0xe0000b, 0x7fffd96f4fb0, 0x7fffd96f4fb8) = 1
|  XFillRectangle@libX11.so.6(0x558cb64e8a70, 0xe0000b, 0x558cb6505ee0, 45) = 1
|• XRenderCompositeDoublePoly@libXrender.so.1(0x558cb64e8a70, 3, 0xe00009, 0xe0000e) = 0
|  XFillRectangle@libX11.so.6(0x558cb64e8a70, 0xe0000b, 0x558cb6505ee0, 124) = 1
└→ XRenderCompositeDoublePoly@libXrender.so.1(0x558cb64e8a70, 3, 0xe00009, 0xe0000e) = 0
❸  XQueryPointer@libX11.so.6(0x558cb64e8a70, 0xe0000b, 0x7fffd96f4fb0, 0x7fffd96f4fb8) = 1
|  XFillRectangle@libX11.so.6(0x558cb64e8a70, 0xe0000b, 0x558cb6505ee0, 43) = 1
|  XRenderCompositeDoublePoly@libXrender.so.1(0x558cb64e8a70, 3, 0xe00009, 0xe0000e) = 0
|  XFillRectangle@libX11.so.6(0x558cb64e8a70, 0xe0000b, 0x558cb6505ee0, 122) = 1
└→ XRenderCompositeDoublePoly@libXrender.so.1(0x558cb64e8a70, 3, 0xe00009, 0xe0000e) = 0
                    ...          ...          ...          ...
```

Зная назначение библиотечных вызовов **XQueryPointer(3)**, **XFillRecangle(3)** и **XRenderCompositeDoublePoly(3)**, можно составить модель функционирования X-клиента, в котором легко увидеть создание окна ❶, цикл ❷ опроса положения курсора при помощи **XQueryPointer(3)** и цикл ❸ перерисовки глаз стиранием их старого изображения при помощи **XFillRectangle(3)** и отрисовки нового положения посредством **XRenderCompositeDoublePoly(3)**. Можно даже предположить, что две пары **XFill/XRender** используются на каждой итерации, потому что перерисовываются два глаза.

Еще одним инструментом, разрешающим трассировать сообщения самого X-протокола, является **xtrace(1)**, позволяющая увидеть обмен между X-клиентом и X-сервером (листинг 7.32).

Листинг 7.32. Трассировка X-протокола

```
homer@ubuntu:~$ xtrace -n xeyes | grep -E 'QueryPointer|FillRectangle|RENDER'
                ...          ...          ...          ...
000:<:003c:  8: Request(38): QueryPointer window=0x02e0000b

000:>:003c:32: Reply to QueryPointer: same-screen=true(0x01) root=0x0000017c
child=None(0x00000000) root-x=913 root-y=1 win-x=813 win-y=-638 mask=0
                ...          ...          ...          ...
000:<:0053: 20: Request(70): PolyFillRectangle drawable=0x02e0000b gc=0x02e00006
rectangles={x=42 y=28 w=13 h=18};

000:<:0054:304: RENDER-Request(140,10): Trapezoids op=Over(0x03) src=0x02e00009 xSrc=0 ySrc=0
dst=0x02e0000e maskFormat=0x00000024 trapezoids={...};};
                ...          ...          ...          ...
```

Кроме **xtrace(1)**, для анализа сообщений X-протокола при передаче по сети можно использовать анализатор сетевых пакетов **wrieshark(1)** ровно таким же образом, как и для анализа сетевых сообщений любых других сетевых служб.

Большинство современных X-клиентов не прибегают напрямую к помощи низкоуровневых библиотек, таких как **Xlib** и **Xrender**, а используют высокоуровневые библиотеки виджетов, отрисовывающие их при помощи сообщений X-протокола. В примере из листинга 7.33 показана простейшая программа на языке программирования C, использующая библиотеку виджетов **Gtk**, а в листинге 7.34 приведены ее компиляция[1] и трассировка ее библиотечных вызовов к **libgtk-x11-2.0.so.0**.

Листинг 7.33. Язык C и библиотека виджетов Gtk

```
homer@ubuntu:~$ cat hello.c
#include <gtk/gtk.h>

static void terminate(GtkWidget *widget, gpointer data) {
❾    gtk_main_quit();
}

int main(int argc, char *argv[]) {
    GtkWidget *window, *button, *label, *vbox;

❶    gtk_init (&argc, &argv);

❷    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
❸    vbox = gtk_vbox_new(TRUE, 0);
③    gtk_container_add(GTK_CONTAINER (window), vbox);

❹    label = gtk_label_new("Hello, world!");
④    gtk_container_add(GTK_CONTAINER (vbox), label);

❺    button = gtk_button_new_with_label("Quit");
⑤    gtk_container_add(GTK_CONTAINER (vbox), button);

❻    g_signal_connect(button, "clicked", G_CALLBACK (terminate), NULL);
❼    gtk_widget_show_all (window);
❽    gtk_main ();

    return 0;
}
```

При работе программы сначала инициализируется библиотека виджетов ❶, которая инициирует соединение с X-сервером и аутентифицирует клиента при помощи «волшебной печеньки».

[1] При наличии установленного пакета **libgtk2.0-dev**.

Интерфейс программы выстраивается из виджетов окна (window) ❷ и вертикального контейнера (vbox) ❸, который добавляется в это окно ③. В контейнер последовательно добавляются ④ и ⑤ виджеты текстовой метки (label) ❹ и кнопки (button) ❺, сигнал нажатия (**clicked**) на которую связывается ❻ с обработчиком — функцией **terminate**.

По запросу отображения окна ❼ библиотека прорисовывает виджеты при помощи X-протокола, после чего запускается главный цикл ❽ обработки поступающих событий X-протокола (щелчки кнопкой мыши, нажатия клавиш и т. д.), который прекращается ❾ при срабатывании обработчика сигнала нажатия (**clicked**) на виджет кнопки.

Листинг 7.34. Трассировка библиотеки Gtk

```
homer@ubuntu:~$ gcc hello.c -o hello $(pkg-config --cflags --libs gtk+-2.0)
homer@ubuntu:~$ lddtree hello | grep libgtk
    libgtk-x11-2.0.so.0 => /lib/x86_64-linux-gnu/libgtk-x11-2.0.so.0
homer@ubuntu:~$ ltrace -n2 -l libgtk-x11-2.0.so.0 ./hello
❶ gtk_init(0xbfc98440, 0xbfc98444, 0x8049ff4, 0x80488b1, -1)      = 1
❷ gtk_window_new(0, 0xbfc98444, 0x8049ff4, 0x80488b1, -1)         = 0x8302000
❸ gtk_vbox_new(1, 0, 0x8049ff4, 0x80488b1, -1)                    = 0x82ab858
③ gtk_container_get_type(1, 0, 0x8049ff4, 0x80488b1, -1)          = 0x82e29b0
③ gtk_container_add(0x8302000, 0x82ab858, 0x8049ff4, 0x80488b1, -1) = 0
❹ gtk_label_new(0x8048960, 0x82ab858, 0x8049ff4, 0x80488b1, -1)   = 0xb5b06608
④ gtk_container_get_type(0x8048960, 0x82ab858, 0x8049ff4, 0x80488b1, -1) = 0x82e29b0
④ gtk_container_add(0x82ab858, 0xb5b06608, 0x8049ff4, 0x80488b1, -1) = 0
❺ gtk_button_new_with_label(0x804896e, 0xb5b06608, 0x8049ff4, 0x80488b1, -1) = 0x830a010
⑤ gtk_container_get_type(0x804896e, 0xb5b06608, 0x8049ff4, 0x80488b1, -1) = 0x82e29b0
⑤ gtk_container_add(0x82ab858, 0x830a010, 0x8049ff4, 0x80488b1, -1) = 0
❼ gtk_widget_show_all(0x8302000, 0x8048973, 0x8048764, 0, 0)      = 2
❽ gtk_main(0x8302000, 0x8048973, 0x8048764, 0, 0 <unfinished ...>
❾ gtk_main_quit(0x82ab858, 0xb73d0f30, 0xb72e5160, 0xb72e5243, 0x830a010) = 0
<... gtk_main resumed> )                                          = 0
+++ exited (status 0) +++
```

Аналогичным образом выглядят и работают программы, использующие Gtk и на других языках программирования, например на Python[1], что проиллюстрировано в листинге 7.35.

[1] При наличии установленного пакета python-gtk2.

Листинг 7.35. Интерпретатор Python и библиотека интерфейсных элементов Gtk

```
homer@ubuntu:~$ cat gtk-hello.py
#!/usr/bin/python

import gtk

w = gtk.Window()
box = gtk.VBox()
w.add(box)

l = gtk.Label("Hello, world!")
box.add(l)

b = gtk.Button("Quit")
box.add(b)

def terminate(o):
  gtk.main_quit()

b.connect("clicked", terminate)
w.show_all()
gtk.main()
```

Каждая библиотека виджетов при ее использовании для построения пользовательского интерфейса обладает своей спецификой, но в целом имеет много общего с другими библиотеками интерфейсных элементов. Для сравнения в листинге 7.36 приведена программа на языке Python, использующая библиотеку Qt[1], в которой можно найти много общего с программой из листинга 7.35, написанной на том же языке, но использующей библиотеку Gtk.

Листинг 7.36. Интерпретатор Python и библиотека интерфейсных элементов QtGui

```
homer@ubuntu:~$ cat qt-hello.py
#!/usr/bin/python

import sys
from PyQt4 import QtGui, Qt

a = QtGui.QApplication(sys.argv)

w = QtGui.QWidget()
box = QtGui.QVBoxLayout()
w.setLayout(box)
```

[1] Требует установленного пакета python-qt4.

```
l = QtGui.QLabel("Hello, world!")
w.layout().addWidget(l)

b = QtGui.QPushButton("Quit")
w.layout().addWidget(b)

def terminate():
  a.quit()

w.show()
a.connect(b, Qt.SIGNAL("clicked()"), terminate)
a.exec_()
```

Библиотека виджетов W:[Tk] среди прочих других библиотек занимает, пожалуй, особенное место за счет языка W:[Tcl], в качестве расширения которого специально и разрабатывалась. В отличие от языков C и Python, приведенных выше, и многих других «настоящих» систем программирования, Tcl (Tool Command Language) является инструментальным языком, родственным семейству языков командного интерпретатора **bash(1)**, **ksh(1)**, **csh(1)**, **zsh(1)** и пр.

Tcl-интерпретатор имеет «обычную» оболочку **tclsh(1)** (**Tcl shell**) и «оконную» оболочку **wish(1)**, **wi**ndowing **sh**ell, со встроенными командами к библиотеке виджетов Tk[1]. Среди прочих вариантов X-клиента с использованием различных языков программирования и библиотек виджетов пример из листинга 7.37 имеет самый простой и очевидный синтаксис, похожий на синтаксис языка командного интерпретатора.

Листинг 7.37. Командный интерпретатор wish: язык Tcl и библиотека интерфейсных элементов Tk

```
homer@ubuntu:~$ cat hello.tcl
#!/usr/bin/wish

label .l -text "Hello, world!"
button .b -text Quit -command exit

pack .l .b
```

7.8.2. 3D-графика и инфраструктура прямого рендеринга DRI

Как показывает практика, современные графические приложения все чаще прибегают к использованию программного интерфейса OpenGL как для аппаратно-ускоренной отрисовки непосредственно 3D-графики, так и для получения аппаратного ускорения при решении других графических задач, например композитинга *(см. разд. 7.6.1)*.

[1] Интерпретатор wish можно найти в пакете **tk**.

Как было проиллюстрировано выше, для эффективной работы приложений с GPU и памятью аппаратных ускорителей им необходимо обеспечить прямой доступ к этим устройствам, что реализуется менеджером прямого отрисовки — **W:[DRM]**, а для согласования взаимодействия между X-клиентами и X-сервером при доступе к DRM используется инфраструктура прямой отрисовки (рендеринга) **W:[DRI]**.

Кроме того, необходимы компоненты, реализующие расширение X-протокола GLX, сам программный интерфейс OpenGL и драйверы, умеющие транслировать OpenGL-команды в обращения к GPU и видеопамяти, специфические для того или иного видеоускорителя. Эти роли обычно выполняют *проприетарные* библиотеки от производителей самих видеоускорителей или библиотеки проекта **W:[Mesa]**, изначально задуманного только как программный рендерер OpenGL (и расширение GLX), но впоследствии интегрировавшего в себя *свободные* DRI-драйверы аппаратного рендеринга для видеоускорителей Intel, AMD и NVIDIA.

Листинг 7.38. Трассировка GLX-клиента

```
homer@ubuntu:~$ lddtree `which glxdemo`
glxdemo => /usr/bin/glxdemo (interpreter => /lib64/ld-linux-x86-64.so.2)
    libGL.so.1 => /lib/x86_64-linux-gnu/libGL.so.1
        libGLX.so.0 => /lib/x86_64-linux-gnu/libGLX.so.0
                    ...          ...          ...
    libX11.so.6 => /lib/x86_64-linux-gnu/libX11.so.6
                   ...          ...          ...

homer@ubuntu:~$ ltrace -n4 -x dlopen -e gl*@MAIN+X*@MAIN -s 256 glxdemo
❶ glxdemo->XOpenDisplay(nil)                                          = 0x55a43bc9bd40
                    ...          ...          ...          ...
❷ glxdemo->glXChooseVisual(0x55a43bc9bd40, 0, 0x7ffea01b4700, 0 <unfinished ...>
    dlopen@libdl.so.2("libGLX_mesa.so.0", 1)                          = 0x55a43bca9890
                    ...          ...          ...          ...
❸   dlopen@libdl.so.2("/usr/lib/x86_64-linux-gnu/dri/i965_dri.so", 258) = 0x55a43bcb6ec0
<... glXChooseVisual resumed> )                                       = 0x55a43bcb7830
                    ...          ...          ...
❹ glxdemo->XCreateWindow(0x55a43bc9bd40, 380, 0, 0)                   = 0xe00002
glxdemo->glXCreateContext(0x55a43bc9bd40, 0x55a43bcb7830, 0, 1)       = 0x55a43bcb7d90
❺ glxdemo->glXMakeCurrent(0x55a43bc9bd40, 0xe00002, 0x55a43bcb7d90, 0) = 1
                    ...          ...          ...          ...
❻ glxdemo->XMapWindow(0x55a43bc9bd40, 0xe00002)                       = 1
                    ...          ...          ...          ...
❼ glxdemo->XNextEvent(0x564f9b53cd40, ..., 0x564f9b60a58c)              = 0
Redraw event
❽ glxdemo->glClear(0x4000, 0x55a43bcb4760, 0, 0x7ff3e5e88317)         = 0
❾ glxdemo->glColor3f(0, 0, 1, 0)                                      = 0x55a43bdea424
❿ glxdemo->glRectf(12, 4, 5460, 0)                                    = 1
```

```
❶ glxdemo->glXSwapBuffers(0x55a43bc9bd40, 0xe00002, 519, 4)          = 0
  glxdemo->XNextEvent(0x55a43bc9bd40, 0x7ffea01b4700, 3, 0^C <no return ...>
  --- SIGINT (Interrupt) ---
  +++ killed by SIGINT +++
```

В листинге 7.38 показана работа простейшего X-клиента **glxdemo(1)**, использующего библиотеки **libGL**, **libGLX** и **libX11**. Как и любой другой X-клиент, он для начала подключается ❶ к X-серверу, затем при выборе режима изображения ❷ «оберточной» библиотекой **libGLX** загружается «настоящая»[1] OpenGL/GLX-библиотека **libGLX_mesa.so.0**, которая, в свою очередь, ❸ загружает DRI-драйверы (выполненные как разделяемые библиотеки), в данном случае — для видеоускорителя на чипе Intel **i965**. После инициализации создается обычное окно ❹, которое затем прикрепляется ❺ к контексту вывода и изображается ❻ на экране. При получении события на перерисовку содержимого окна ❼ начинается собственно OpenGL-рендеринг ❽, который завершается отправкой результата рендеринга ❾ в окно. В **glxdemo(1)**, как и большинстве OpenGL-приложений, используется контекст вывода с двойной буферизацией, т. е. рендеринг происходит во внеэкранный буфер (back buffer), изображается всегда экранный (front buffer), а по завершении рендеринга они быстро меняются местами (swap buffers), что позволяет изображать только полностью отрисованные сцены.

Тем не менее из результата трассировки в листинге 7.38 не очевидно, что был использован GLX-режим прямой отрисовки *(direct rendering)*, что можно явно увидеть при совместной трассировке библиотечных вызовов и команд X-протокола (листинг 7.39).

Листинг 7.39. Трассировка GLX- и DRI3-расширений X-протокола

```
homer@ubuntu:~$ xtrace -n ltrace -n4 -x dlopen -e gl*@MAIN+X*@MAIN -s 256 glxdemo
❶ glxdemo->XOpenDisplay(nil
     000:<: am lsb-first want 11:0 authorising with '' of length 0
     000:>: Success, version is 11:0 vendor='The X.Org Foundation' ... roots={root=0x00000137 ...};
                ...          ...          ...          ...
❷ glxdemo->glXChooseVisual(0x55c68d75dcd0, 0, 0x7ffe0507cc90, <unfinished ...>
                ...          ...          ...          ...
  ① 000:<:0007: 12: Request(98): QueryExtension name='GLX'
     000:>:0007:32: Reply to QueryExtension: present=true(0x01) ...
                ...          ...          ...
  ② 000:<:0009: 12: GLX-Request(155,19): glXQueryServerString ...
     000:>:0009:40: Reply to glXQueryServerString: string='mesa'
  dlopen@libdl.so.2("libGLX_mesa.so.0", 1)     = 0x55c68d76ac30
                ...          ...          ...
```

[1] В этом месте могла бы загружаться проприетарная библиотека GL/GLX.

```
    ③ 000:<:000d: 12: Request(98): QueryExtension name='DRI3'
       000:>:000d:32: Reply to QueryExtension: present=true(0x01) ...
       000:<:000e: 16: Request(98): QueryExtension name='Present'
       000:>:000e:32: Reply to QueryExtension: present=true(0x01) ...
                      ...          ...          ...          ...
    ④ 000:<:001a: 12: DRI3-Request(149,1): Open drawable=0x00000137 provider=0x00000000
       000:>:001a:32: Reply to Open: nfd=1
❸ dlopen@libdl.so.2("/usr/lib/x86_64-linux-gnu/dri/i965_dri.so", 258) = 0x55c68d77b480
<... glXChooseVisual resumed> )                 = 0x55c68d774dd0
                      ...          ...          ...
❹ glxdemo->XCreateWindow(0x55c68d75dcd0, 311, 0, ...) = 0x3600002
       000:<:001d: 48: Request(1): CreateWindow depth=0x18 window=0x03600002 parent=0x00000137 ...
    glxdemo->glXCreateContext(0x55c68d75dcd0, 0x55c68d774dd0, 0, ...
       000:<:001e: 24: GLX-Request(155,3): glXCreateContext context=0x03600003 ...
       000:<:001f:  8: GLX-Request(155,6): glXIsDirect context=0x03600003
       000:>:001f:32: Reply to glXIsDirect: is_direct=1
) = 0x55c68d775330
❺ glxdemo->glXMakeCurrent(0x55c68d75dcd0, 0x3600002, 0x55c68d775330,
                      ...          ...          ...          ...
    ⑤ 000:<:0026: 24: DRI3-Request(149,2): PixmapFromBuffer pixmap=0x03600006 drawable=0x03600002 ...
    ⑥ 000:<:0027: 16: DRI3-Request(149,4): FenceFromFD drawable=0x03600006 fence=0x03600007 ...
                      ...          ...          ...          ...
) = 1
                      ...          ...
❻ glxdemo->XMapWindow(0x55c68d75dcd0, 0x3600002
       0000:<:0028:  8: Request(8): MapWindow window=0x03600002
       000:>:0028: Event MapNotify(19) event=0x03600002 window=0x03600002 ...
) = 1
                      ...          ...          ...          ...
       000:>:0028: Event Expose(12) window=0x03600002 x=0 y=0 width=300 height=300 count=0x0000
❼ glxdemo->XNextEvent(0x55b3b9d62cd0, 0x7fffa3e90a00, 0x7fffa3e908f0, 0x55b3ba01919c) = 0
❽ glxdemo->glClear(0x4000, 0x6520776172646552, 0x7fee08ec18c0, 0x746e657665207761
    ⑤ 000:<:008f: 24: DRI3-Request(149,2): PixmapFromBuffer pixmap=0x03600008 drawable=0x03600002 ...
    ⑥ 000:<:0090: 16: DRI3-Request(149,4): FenceFromFD drawable=0x03600008 fence=0x03600009 ...
       000:<:0098: 28: Request(62): CopyArea src-drawable=0x03600006 dst-drawable=0x03600008 ...
       000:<:0099:  8: SYNC-Request(134,15): TriggerFence fid=0x03600009
       000:<:009a:  8: Request(54): FreePixmap drawable=0x03600006
       000:<:009b:  8: SYNC-Request(134,17): DestroyFence fid=0x03600007
) = 0
❾ glxdemo->glColor3f(0x7fee04f53c5b, 2, 0x7fee04f6bed8, 0) = 0x55c68da5ad04
❿ glxdemo->glRectf(0, 0, 5126, 2)                 = 1
⓫ glxdemo->glXSwapBuffers(0x55c68d75dcd0, 0x3600002, 519, 4
                      ...          ...          ...          ...
    ⑦ 000:<:002a: 72: Present-Request(148,1): Pixmap window=0x03600002 pixmap=0x03600008
       idle_fence=0x03600009 ...
) = 1
                      ...          ...          ...          ...
    ⑧ 0000:>:008e: Event Generic(35) Present(148) IdleNotify(2) event=0x03600005 window=0x03600002 serial=19
       pixmap=0x03600008 idle_fence=0x03600009
    ⑨ 000:>:009c: Event Generic(35) Present(148) CompleteNotify(1) kind=Pixmap(0x00) mode=Copy(0x00)
       event=0x03600005 window=0x03600002 ...
```

```
glxdemo->XNextEvent(0x55c68d75dcd0, 0x7ffe0507cc90, 0x55c68da6c880, 0...^C <no return ...>
--- SIGINT (Interrupt) ---
+++ killed by SIGINT +++
```

Анализ трассы показывает, что при выборе режима изображения ❹ X-сервер при помощи GLX-расширения X-протокола ① сообщает идентификатор «настоящей» OpenGL/GLX-библиотеки **mesa** ②, а затем при помощи DRI-расширения X-протокола ③ производится подключение к DRM ④ и загружается DRI-драйвер видеоускорителя **i965**.

Каждый раз при получении X-клиентом события на перерисовку содержимого его окна ❼ создаются объект **pixmap** ⑤ и некий примитив **W:[memory fence]** ⑥ для синхронизации доступа к этому **pixmap**.

Как и ожидалось, при выполнении операций OpenGL-рендеринга ❽ X-протокол никак не задействован, т. е. задействован режим прямой отрисовки. По окончании рендеринга ❿ полученные результаты «презентуются» X-серверу (при помощи соответствующего расширения X-протокола **Present**) ⑦, которые в конце концов помещаются им в окно X-клиента. При этом сам X-клиент оповещается о событиях высвобождения объекта **pixmap** ⑧ и об окончании отображения **pixmap** в его окно ⑨.

Сама OpenGL-отрисовка при помощи загруженного DRI-драйвера, выделение DRM-буфера под рендеринг в памяти видеоускорителя и его связь с **pixmap** естественным образом остаются за рамками X-протокола, т. к. прямой рендеринг и разрабатывался для вынесения процедур OpenGL-отрисовки за его рамки *(см. разд. 7.6.2)*. Несложно предположить, что *прямое* взаимодействие X-клиента и DRM можно наблюдать на интерфейсе системных вызовов (листинг 7.40).

Листинг 7.40. Трассировка DRM интерфейса

```
homer@ubuntu:~$ xtrace strace -e socket,connect,sendmsg,recvmsg,openat,ioctl glxdemo
                ...       ...       ...       ...                          glXChooseVisual()
  000:<:0000: 12: GLX-Request(155,19): glXQueryServerString ...
  000:>:0009:40: Reply to glXQueryServerString: string='mesa'
  openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/libGLX_mesa.so.0", O_RDONLY|O_CLOEXEC) = 5...
                ...       ...       ...       ...
  000:<:001a:12: DRI3-Request(149,1): Open drawable=0x00000137 provider=0x00000000
❶ 000:>:001a:32: Reply to Open: nfd=1
↳ recvmsg(4, {... msg_control=[{..., cmsg_type=SCM_RIGHTS, cmsg_data=[5]}], ...}, 0) = 32
                          ...       ...       ...
❷ ioctl(5, DRM_IOCTL_VERSION, 0x555930e37770) = 0
❸ openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/dri/i965_dri.so", O_RDONLY|O_CLOEXEC) = 6
                ...       ...       ...       ...
❹ ioctl(5, DRM_IOCTL_I915_GEM_CREATE, 0x7ffe4afc40c0) = 0
❺ ioctl(5, DRM_IOCTL_I915_GEM_MMAP, 0x7ffe4afc40d0) = 0
                    ...       ...       ...       ...
```

```
                                                                    glXMakeCurrent()
❹ ioctl(5, DRM_IOCTL_I915_GEM_CREATE, 0x7ffe4afc4460) = 0
❻ ioctl(5, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffe4afc462c) = 0
❼ sendmsg(4, {... msg_control=[{... cmsg_type=SCM_RIGHTS, cmsg_data=[7, 6]}], ...}, 0) = 40
↳ 000:<:0026:24: DRI3-Request(149,2): PixmapFromBuffer pixmap=0x03600006 drawable=0x03600002
                                                                    glClear()
❶ ioctl(5, DRM_IOCTL_I915_GEM_CREATE, 0x7ffe4afc4580) = 0
❻ ioctl(5, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffe4afc474c) = 0
❼ sendmsg(4, {... msg_control=[{... cmsg_type=SCM_RIGHTS, cmsg_data=[7, 6]}], ...}
↳ 000:<:008f: 24: DRI3-Request(149,2): PixmapFromBuffer pixmap=0x03600008 drawable=0x03600002 ...
                                                    glColor3f() → glRectf() → glXSwapBuffers()
❺ ioctl(5, DRM_IOCTL_I915_GEM_EXECBUFFER2, 0x7fff9c009060) = 0
  ioctl(5, DRM_IOCTL_I915_GEM_WAIT or DRM_IOCTL_RADEON_GEM_OP, 0x7fff9c008ff0) = 0
  ioctl(5, DRM_IOCTL_I915_GEM_BUSY, 0x7fff9c008eb0) = 0
   000:<:002a: 72: Present-Request(148,1): Pixmap window=0x03600002 pixmap=0x03600008 ...
```

Из трассы в листинге 7.40 видно, что для обращений к менеджеру DRM и DRM-драйверу видеоускорителя используются системные вызовы **ioctl(2)**, как и всегда для взаимодействия с любыми драйверами устройств (*см. разд. 3.3* и листинг 3.19). Кроме того, интересным оказывается тот факт, что файловый дескриптор для работы X-клиента с видеоускорителем дисплея X-сервера возвращается сразу в «готовом» виде, при ответе ❶ на запрос подключения этого клиента к DRI-инфраструктуре. Для этого задействуется механизм передачи файловых дескрипторов через специальные служебные (ancillary) сообщения **SCM_RIGHTS** файловых сокетов (см. **cmsg(3)** и **unix(7)**). Идентификатор нужного DRI-драйвера ❷ выясняется специальным запросом непосредственно к DRM, после чего соответствующий драйвер загружается ❸ для выполнения последующего рендеринга. Перед началом рендеринга у DRM запрашивается несколько буферов (буфер команд GPU, z-буфер, stencil-буфер и пр.) в памяти видеоускорителя ❹, которые отображаются в память процесса X-клиента ❺ (листинг 7.41).

Идентификатор буфера, в который непосредственно будет рендерится изображение, запрашивается у DRM ❻ в виде файлового дескриптора, который так же при помощи служебного сообщения **SCM_RIGHTS** отправляется ❼ X-серверу. В результате сервер создаст объект **pixmap**, который затем будет использоваться в качестве источника изображения окна клиента. В результате получается, что клиент и сервер

напрямую работают с изображением в общей памяти видеоускорителя, накладные расходы сведены к нулю, а производительность OpenGL отрисовки максимальна.

Рисующие вызовы OpenGL-библиотеки (например, **glRectf(3)** и **glColor3f(3)**) превращаются DRI-драйвером в набор команд, понятных GPU видеоадаптера, и упаковываются в буфер команд, который потом отправляется на исполнение ❽.

Для полноты картины в листинге 7.41 показаны ресурсы, используемые демонстрационным X-клиентом **glxgears(1)** при прямом рендеринге, а именно: локальный сокет ❶ для взаимодействия с X-сервером, файловый дескриптор ❷ для доступа к DRM-менеджеру, а также отображения DRM-буферов ❸ в память процесса.

Листинг 7.41. DRM-буфера прямого рендеринга

```
homer@ubuntu:~$ glxgears &
[1] 31456
homer@ubuntu:~$ pmap 31456
COMMAND    PID  USER  FD   TYPE             DEVICE SIZE/OFF    NODE NAME
glxgears 31456 homer  cwd    DIR              8,4    53248  786434 /home/homer
glxgears 31456 homer  rtd    DIR              8,4     4096       2 /
glxgears 31456 homer  txt    REG              8,4    22616 8259843 /usr/bin/glxgears
glxgears 31456 homer  DEL    REG              0,45            453 /i915 ❸
                       ...                ...                ...
glxgears 31456 homer  DEL    REG              0,45            522 /i915
glxgears 31456 homer  DEL    REG              0,45            253 /i915
                       ...                ...                ...
glxgears 31456 homer  DEL    REG              0,45            386 /i915
glxgears 31456 homer  DEL    REG              0,45            370 /i915
glxgears 31456 homer  DEL    REG              0,45            247 /i915
                       ...                ...                ...
glxgears 31456 homer    0u   CHR              136,2      0t0       5 /dev/pts/2
glxgears 31456 homer    1u   CHR              136,2      0t0       5 /dev/pts/2
glxgears 31456 homer    2u   CHR              136,2      0t0       5 /dev/pts/2
glxgears 31456 homer    3u   unix 0x0000000000000000      0t0 1444944 type=STREAM ❶
glxgears 31456 homer    4u   CHR              226,0      0t0     442 /dev/dri/card0 ❷
```

7.9. В заключение

Резюмируя современное состояние *оконной* системы X, нужно признать, что со временем, в силу возникших новых условий или требований, она претерпела значительные изменения и на текущий момент очень далека от исходной архитектуры.

Современные тулкиты практические не пользуются иерархией окон и их дочерне-родительскими отношениями (см. листинг 7.18). Для отрисовки «по-современному» выглядящих виджетов с тенями и градиентами широко используются растровые

изображения вместо векторных примитивов X-протокола. Вместе с тем с целью сохранения производительности при работе с растровой графикой востребовано использование разделяемой памяти (см. листинг 7.19) для обмена изображениями между X-клиентами и X-сервером, что свело к нулю изначально заложенное свойство сетевой прозрачности оконной системы.

Для визуальных эффектов просвечивающих окон при перекрытии, проявления или растворения окон, масштабирования и анимации и прочего потребовалась новая компонента системы в виде *композитного* менеджера *(см. разд. 7.6.1)*, тогда как классическая компонента *оконного* менеджера, наоборот, потеряла свою актуальность (см. рис. 7.6) из-за популяризации дизайна (на основе) взаимодействия с пользователем и «клиентского» декорирования.

Наконец, приложения, требующие доступа к процессору и памяти видеоускорителей для аппаратно ускоренного 3D-рендеринга или аппаратно ускоренного (де-)кодирования сжатого видео *(см. разд. 7.8.2)* вообще перестали использовать X-протокол для какой-либо отрисовки содержимого окон.

Все это привело к тому, что X-протокол превратился из *основного*, *предметного*, протокола системы во *вспомогательное средство* взаимодействия (IPC) X-клиентов и X-сервера. При всем этом далеко не самое удобное и адекватное современным реалиям средство взаимодействия, т. к. при развитии оконной системы X требовалось сохранить обратную совместимость со всеми старыми версиями и компонентами.

Именно такое положение вещей побудило разработчиков X Window System к созданию новой графической системы, которая бы унаследовала все современные завоевания оконной системы X, безжалостно распрощавшись с ее неактуальным наследием.

И имя этой новой графической системе — **W:[Wayland]**.

Глава 8

Графическая система Wayland

На самом деле **W:[Wayland]** — это протокол взаимодействия между графическими приложениями, Wayland-клиентами и так называемым Wayland-композитором, т. е. дисплейным сервером (он же Wayland-сервер).

Wayland-сервер (как и любой другой дисплейный сервер, включая X-сервер), решает задачу разделения устройств вывода (дисплеев) и ввода (клавиатуры, манипуляторы «мышь», сенсорные панели и пр.) между одновременно выполняющимися графическими приложениями. Вместе с тем архитектура и внутреннее устройство Wayland-протокола и сервера основывается на современных достижениях X Window System, отбросив все их недостатки и сохранив преимущества.

Вторая задача, решение которой возложено на Wayland-композитор, — это, собственно, композитинг (аналогично композитному менеджеру X), т. е. составление конечной визуальной сцены, изображаемой на дисплее пользователя из содержимого окон клиентов, которые принято называть поверхностями (surface).

Протокол Wayland не предоставляет клиентам никаких возможностей рисования векторных примитивов, как это делал X-протокол, вместо чего за основу протокола взята пересылка от клиентов к композитору растровых изображений. Пересылаемые изображения содержат образ окна целиком или образ изменившейся части окна и доставляются при помощи буферов в разделяемой памяти или через DRM-буферы видеоускорителей, задействуя инфраструктуру DRI.

Сам протокол спроектирован так, чтобы произвольно расширяться без потери совместимости, для чего вводят понятие интерфейсов, каждый из которых является некоторым набором предоставляемых услуг. Каждый интерфейс имеет набор методов, которые приложение может вызвать, и событий, на которые приложение может подписываться и реагировать.

На листинге 8.1 показан вывод[1] утилиты **weston-info(1)**, которая, будучи простейшим Wayland-клиентом, подключается к композитору посредством Wayland-протокола и

[1] Все примеры в этой главе работают, только если войти в систему с использованием **Ubuntu on wayland** сеанса при входе в систему.

выводит список глобальных объектов композитора, предоставляющих соответствующие интерфейсы.

Листинг 8.1. Интерфейсы Wayland-протокола

```
homer@ubuntu:~$ weston-info
interface: 'wl_drm', version: 2, name: 1
interface: 'wl_compositor', version: 4, name: 2
interface: 'wl_shm', version: 1, name: 3
        formats: XRGB8888 ARGB8888
                ...                     ...
interface: 'zxdg_shell_v6', version: 1, name: 10
interface: 'wl_shell', version: 1, name: 11
interface: 'gtk_shell1', version: 3, name: 12
                ...                     ...
interface: 'wl_seat', version: 5, name: 16
        name: seat0
        capabilities: pointer keyboard
        keyboard repeat rate: 33
        keyboard repeat delay: 500
                ...                     ...                     ...
interface: 'zwp_linux_dmabuf_v1', version: 3, name: 20
                ...                     ...                     ...
```

Интерфейс **wl_compositor** является основой протокола, позволяющей клиентам создавать окна (surfaces), в которые они при помощи буферов в разделяемой памяти и специального интерфейса **wl_shm** отправляют изображения, подлежащие композитингу. Интерфейсы **wl_drm** и/или **zwp_linux_dmabuf_v1** используются при прямом OpenGL-рендеринге и обеспечивают поддержку DRI для доставки результата рендеринга от клиента к композитору при помощи DRM-буферов в памяти видеоадаптера. Кроме того, интерфейс **zwp_linux_dmabuf_v1** (как и расширение X-протокола DRI3, проиллюстрированное в *разд. 7.8.2*) позволяет доставлять от клиента к композитору DMA-BUF-буферы, которые широко используются драйверами устройств, например видеокамер. Это позволяет с минимальными накладными расходами изображать видео, например захватываемое с веб-камеры.

Интерфейс **wl_seat** доставляет клиентам события указателя (мыши), нажатия клавиш и события тачскрина, если такие устройства обнаружены композитором, а shell-интерфейсы **wl_shell**, **zxdg_shell_v6** и **gtk_shell1** организуют управление местоположением и размерами окон на экране. Другими словами, реализуют функции оконного менеджера X Window System, за исключением декорирования окон, которое в Wayland возложено на самих клиентов (см. CSD, **W:[Client-Side Decoration]**).

8.1. Wayland-композитор

Как уже сказано выше, основной компонентой Wayland является композитор. Однако, в отличие от оконной системы X, спецификации Wayland не накладывают вообще никаких ограничений на внутреннее устройство композитора. Вместо этого стандартизуется всего лишь протокол взаимодействия и функции, предоставляемые клиентам при помощи тех или иных интерфейсов.

Для тестирования жизнеспособности и уточнения концепции разработчики Wayland-протокола и спецификаций поддерживают клиентскую и серверную библиотеки основного *(core)* протокола (куда как раз входят интерфейсы **wl_compositor**, **wl_shm** и пр.) и прототип *(reference)* композитора под названием **W:[Weston]**. Кроме этого, коллегиально стандартизуются расширения протокола (наборы интерфейсов), предложенные другими заинтересованными разработчиками, которые на основе этого «конструктора» и создают законченные решения, такие как **W:[Mutter (software)]**, являющиеся основой для **gnome-shell(1)**) из настольного окружения **GNOME**, **kwin** из настольного окружения **KDE** (листинг 8.2) и пр.

Листинг 8.2. Wayland-композиторы weston, gnome-shell и kwin

```
homer@ubuntu:~$ ldd $(which weston) | grep wayland
    libwayland-server.so.0 => /lib/x86_64-linux-gnu/libwayland-server.so.0 (0x00007f9d4ee4e000)

homer@ubuntu:~$ ldd $(which gnome-shell) | grep wayland
    libwayland-server.so.0 => /lib/x86_64-linux-gnu/libwayland-server.so.0 (0x00007f8c30150000)
    libwayland-cursor.so.0 => /lib/x86_64-linux-gnu/libwayland-cursor.so.0 (0x00007f8c2dfa1000)
    libwayland-egl.so.1 => /lib/x86_64-linux-gnu/libwayland-egl.so.1 (0x00007f8c2df9c000)
    libwayland-client.so.0 => /lib/x86_64-linux-gnu/libwayland-client.so.0 (0x00007f8c2df8b000)

homer@ubuntu:~$ ldd $(which kwin_wayland) | grep wayland
    libwayland-client.so.0 => /lib/x86_64-linux-gnu/libwayland-client.so.0 (0x00007f1dc6c52000)
    libwayland-server.so.0 => /lib/x86_64-linux-gnu/libwayland-server.so.0 (0x00007f1dc6c3d000)
```

В листинге 8.3 показано, что при запуске Wayland-композитора он действует ровно так же, как и X-сервер (для сравнения см. листинг 7.1), т. е. инициализирует устройства ввода (видеоадаптер) ❶, устройства ввода (мышь и пр.) ❷ и открывает локальный сокет для взаимодействия с клиентами ❸.

Листинг 8.3. Wayland-композитор

```
homer@ubuntu:~$ pgrep -l gnome-shell
5628 gnome-shell
5672 gnome-shell-cal
```

```
homer@ubuntu:~$ lsof -p 5628 -a /dev
COMMAND     PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
gnome-she 5628 homer  mem    CHR  226,0          398 /dev/dri/card0
gnome-she 5628 homer  0r   CHR    1,3      0t0    6 /dev/null
gnome-she 5628 homer  10u   CHR  226,0      0t0  398 /dev/dri/card0 ❶
                      ...                  ...          ...              ...
gnome-she 5628 homer  20u   CHR  13,64      0t0  149 /dev/input/event0 ❷
                      ...                  ...          ...              ...
gnome-she 5628 homer  26u   CHR  13,68      0t0  336 /dev/input/event4

homer@ubuntu:~$ lsof -p 5628 -a -U
COMMAND     PID USER  FD   TYPE             DEVICE SIZE/OFF  NODE NAME
                      ...                  ...          ...              ...
gnome-she 5628 homer 29u  unix 0x000000...00000000      0t0 54619 /run/user/1000/wayland-0 ❸ type=STREAM
                      ...                  ...          ...              ...
```

8.2. Wayland-клиенты и Wayland-протокол

Как было сказано выше, Wayland-протокол по сравнению с X-протоколом весьма прост, т. к. большинство задач возложено на самого клиента или клиентские библиотеки виджетов, в качестве которых выступают все те же **Gtk+** и **Qt**. В стеке тулкита **Gtk** (**g**imp **t**oolkit) изначально присутствовала библиотека рисования **W:[gdk]** (**g**imp **d**rawing **k**it), абстрагирующая саму **Gtk** от оконной системы. Кроме этого, современная **gdk** базируется на библиотеке 2D-рендеринга **W:[cairo (graphics)]**, что вместе составляет достаточную базу для работы поверх Wayland, т. к. позволяет получать именно растровые изображения окна клиентского приложения, отправляемые Wayland-композитору.

Более того, сами приложения совсем (!) не нуждаются в адаптации для работы в графической системе Wayland. Однако таким свойством **Gtk** обладает, только начиная с версии 3, поэтому запустить *откомпилированный* ранее пример из листинга 7.33 не получится[1]. А вот пересобрать тот же пример с библиотекой `libgtk-3` можно так, как показано в листинге 8.4, причем получившаяся программа за счет свойств **Gtk/Gdk** сможет выполняться как в X, так и в Wayland.

Листинг 8.4. Wayland-клиент на основе Gtk 3

```
homer@ubuntu:~$ gcc hello.c -o hello $(pkg-config --cflags --libs gtk+-3.0)
hello.c: In function 'main':
hello.c:13:6: warning: 'gtk_vbox_new' is deprecated: Use 'gtk_box_new' instead [-Wdeprecated-declarations]
     vbox = gtk_vbox_new(TRUE, 0);
     ^~~
```

[1] Если быть точнее, то получится, но только в режиме совместимости, т. к. практически любой Wayland-композитор имеет в своем составе **Xwayland** — X-сервер, конвертирующий X-окна в Wayland-поверхности (surfaces).

```
In file included from /usr/include/gtk-3.0/gtk/gtk.h:282:0,
                 from hello.c:1:
/usr/include/gtk-3.0/gtk/deprecated/gtkvbox.h:61:13: note: declared here
 GtkWidget * gtk_vbox_new      (gboolean homogeneous,
             ^~~~~~~~~~~

homer@ubuntu:~$ lddtree hello
hello => ./hello (interpreter => /lib64/ld-linux-x86-64.so.2)
    libgtk-3.so.0 => /usr/lib/x86_64-linux-gnu/libgtk-3.so.0
        libgdk-3.so.0 => /usr/lib/x86_64-linux-gnu/libgdk-3.so.0
            libXrandr.so.2 => /usr/lib/x86_64-linux-gnu/libXrandr.so.2
                libXrender.so.1 => /usr/lib/x86_64-linux-gnu/libXrender.so.1
                ...                ...                ...                ...
            libwayland-cursor.so.0 => /usr/lib/x86_64-linux-gnu/libwayland-cursor.so.0
            libwayland-egl.so.1 => /usr/lib/x86_64-linux-gnu/libwayland-egl.so.1
            libwayland-client.so.0 => /usr/lib/x86_64-linux-gnu/libwayland-client.so.0
            ...                ...                ...                ...
        libX11.so.6 => /usr/lib/x86_64-linux-gnu/libX11.so.6
```

Для наблюдения за работой программы можно было бы воспользоваться трассировкой библиотечных вызовов так же, как это ранее было показано в листинге 7.34, однако `libwayland-client.so.0` устроена (в угоду эффективности) так, что результат практически невозможно будет трактовать. К счастью, сама библиотека предоставляет великолепные возможности трассировки (активируемые переменной окружения **`WAYLAND_DEBUG`**), показанные в листинге 8.5.

Листинг 8.5. Трассировка программного интерфейса Wayland

```
homer@ubuntu:~$ WAYLAND_DEBUG=1 ltrace -n2 -l libgtk-3.so.0./hello
hello->gtk_init(0x7fffda3942ac, 0x7fffda3942a0, 0x7fffda3942a0, 0
❶ [1406864,197]  -> wl_display@1.get_registry(new id wl_registry@2)
                 ...               ...               ...               ...
① [1406874,203] wl_registry@2.global(2, "wl_compositor", 4)
↳ [1406874,248]  -> wl_registry@2.bind(2, "wl_compositor", 3, new id [unknown]@4)
② [1406874,404] wl_registry@2.global(3, "wl_shm", 1)
↳ [1406874,418]  -> wl_registry@2.bind(3, "wl_shm", 1, new id [unknown]@5)
                 ...               ...               ...               ...
③ [1406891,543] wl_registry@2.global(15, "wl_seat", 5)
↳ [1406891,556]  -> wl_registry@2.bind(15, "wl_seat", 5, new id [unknown]@15)
                 ...               ...               ...               ...
❷ [1406894,912]  -> wl_display@1.sync(new id wl_callback@21)
                 ...               ...               ...               ...
④ [1406895,627] wl_seat@15.capabilities(3)
|  [1406895,632]  -> wl_seat@15.get_pointer(new id wl_pointer@8)
                 ...               ...               ...               ...
↳ [1406895,680]  -> wl_seat@15.get_keyboard(new id wl_keyboard@26)
                 ...               ...               ...               ...
⑤ [1406895,713] wl_callback@21.done(106554)
❸ [1406895,721]  -> wl_registry@2.bind(9, "xdg_wm_base", 1, new id [unknown]@21)
)=1
```

```
hello->gtk_window_new(0, 0x7f294f2493a0, 0, 0x7f294f2493a0) = 0x55d553e24520
hello->gtk_vbox_new(1, 0, 1, 0)                 = 0x55d553faa180
hello->gtk_container_get_type(0x55d553f07f20, 1, 0x55d553f07f20, 0) = 0x55d553e4d8a0
hello->gtk_container_add(0x55d553e24520, 0x55d553faa180, 0x55d553e24520, 3) = 0x55d553e6b0a0
hello->gtk_label_new(0x55d55301cc84, 0x55d553faa180, 1, 0) = 0x55d5540d0210
hello->gtk_container_get_type(0x55d5540d0220, 80, 1, 0) = 0x55d553e4d8a0
hello->gtk_container_add(0x55d553faa180, 0x55d5540d0210, 0x55d553faa180, 3) = 0x55d553e6b0a0
hello->gtk_button_new_with_label(0x55d55301cc92, 0x55d5540d0210, 1, 0) = 0x55d5540d1180
hello->gtk_container_get_type(0x55d553f22bb0, 0x55d553ddd010, 0x55d553f22bb0, 0) = 0x55d553e4d8a0
hello->gtk_container_add(0x55d553faa180, 0x55d5540d1180, 0x55d553faa180, 3) = 0x55d553e6b0a0
hello->gtk_widget_show_all(0x55d553e24520, 1, 212, 0
❹ [1406978,365]  -> wl_compositor@4.create_surface(new id wl_surface@27)
   [1406978,599]  -> wl_surface@27.set_buffer_scale(1)
❺ [1406979,800]  -> xdg_wm_base@21.get_xdg_surface(new id xdg_surface@28, wl_surface@27)
                    ...          ...             ...             ...
❻ [1406979,881]  -> wl_surface@27.commit()
) = 2
hello->gtk_main(0x55d553e24530, 80, 1, 0
                    ...          ...             ...             ...
⑥ [1406997,924] xdg_surface@28.configure(106556)
   [1406997,959]  -> xdg_surface@28.ack_configure(106556)
❼ [1407014,540]  -> wl_shm@5.create_pool(new id wl_shm_pool@31, fd 13, 272800)
❽ [1407014,580]  -> wl_shm_pool@31.create_buffer(new id wl_buffer@32, 0, 275, 248, 1100, 0)
❾ [1407018,097]  -> wl_surface@27.attach(wl_buffer@32, 0, 0)
   [1407018,131]  -> wl_surface@27.set_buffer_scale(1)
   [1407018,140]  -> wl_surface@27.damage(0, 0, 275, 248)
                    ...          ...             ...             ...
   [1407018,293]  -> wl_surface@27.frame(new id wl_callback@35)
❿ [1407018,303]  -> wl_surface@27.commit()
                    ...          ...             ...             ...
<unfinished ...>
hello->gtk_main_quit(0x55d5540d1180, 0, 0, 0)    = 0
                    ...          ...             ...             ...
<... gtk_main resumed> )                        = 0x55d553dde280
+++ exited (status 0) +++
```

Wayland-взаимодействие с композитором начинается с подключения к нему и получения «дисплея» `wl_display` (на трассе не показано). Затем при помощи его метода `get_registry()` извлекается реестр глобальных интерфейсов ❶, в котором зарегистрированы интерфейсы `wl_compositor` ①, `wl_shm` ②, `wl_seat` ③ и др. На события всех этих интерфейсов производится подписка при помощи метода `bind()` реестра, а затем у дисплея запрашивается ❹ доставка события «синхронизация», которое доставляется в момент, когда нет больше других событий в очереди доставки.

При получении события «возможности» (capabilities) ④ интерфейса wl_seat, при помощи его методов get_pointer() и get_keyboard() извлекаются интерфейсы wl_pointer и wl_keyboard, на события которых производится подписка (на трассе не показано) для получения информации о движении указателя мыши и нажатия клавиш на поверхностях (surface), созданных приложением.

По окончании «синхронизации» ⑤ производится подписка ❸ еще на один глобальный интерфейс xdg_wm_base, предназначенный для базового управления окнами (window management), а затем создается объект «поверхности» приложения ❹ методом create_surface() интерфейса wl_compositor. Эта поверхность «превращается» в «xdg-окно» ❺ при помощи метода get_xdg_surface() интерфейса xdg_wm_base, т. е. пользователь сможет изменять ее размер или местоположение на экране. Последним шагом все сделанные изменения применяются ❻ при помощи метода commit() интерфейса wl_surface. Надо заметить, что все вышеперечисленные интерфейсы на самом деле реализуются объектами, начиная с объектов «дисплея» и реестра глобальных объектов и заканчивая объектами поверхностей. И вообще, Wayland-протокол и его программный интерфейс спроектированы в объектно-ориентированном *стиле*, несмотря на то что их базовая реализация выполнена для необъектно-ориентированного языка C.

Последняя часть трассы листинга показывает, как на самом деле растровые изображения из клиента доставляются в композитор. При получении события конфигурирования xdg-окна ⑥ приложение посредством wl_shm интерфейса создает пул ❼ разделяемой (между процессами клиента и композитора) памяти на основе отображения файла с файловым дескриптором fd = 13 и размером 272 800 байт. Такой дескриптор зачастую получают либо[1] при помощи shm_open(3) в семантике разделяемой памяти POSIX (*см. разд. 4.9.5* и листинг 4.58), либо при помощи Linux-специфичного системного вызова memfd_create(2). Затем в пуле создается wl_buffer буфер ❽ размером 245×278 пиксела (по 4 байта каждый), который присоединяется к «поверхности» ❾ в качестве «заднего» (back buffer), а затем все сделанные изменения приводятся к исполнению ❿, т. е. буфер становится передним (front buffer) и начинает изображаться на экране композитором. Между шагами ❽ и ❾ производится отображение файла с fd = 13 в память клиента и наверняка производится рендеринг изображения окна приложения, что заметно по задержке между ними.

При анализе файлов, отображенных в память процесса, видно, что libgdk действительно использует memfd_create(2) для создания временного файла в оперативной памяти, который затем разделяемым образом отображается в память процессов клиента и композитора (листинг 8.6).

---

[1] Так и поступит libgdk на другой платформе, например FreeBSD.

Листинг 8.6. Трассировка библиотек Wayland

```
homer@ubuntu:~$ ./hello &
[1] 28237
homer@ubuntu:~$ pgrep -l gnome-shell
5628 gnome-shell

homer@ubuntu:~$ lsof -p 28237 | grep memfd
hello    28237 homer DEL       REG              0,5        ☛ 387481 /memfd:gdk-wayland
homer@ubuntu:~$ pmap 28237 | grep memfd
00007f3068202000 ☛ 268K rw-s- memfd:gdk-wayland (deleted)

homer@ubuntu:~$ lsof -p 5628 | grep 387481
gnome-she 5628 homer DEL       REG              0,5        ☛ 387481 /memfd:gdk-wayland
homer@ubuntu:~$ pmap 5628 | grep 268K
00007f2974fde000    268K rw-s- memfd:gdk-wayland (deleted)
```

Практически идентичным образом работают и «продвинутые» Wayland-клиенты, задействующие OpenGL-рендеринг, для которого, однако, в Wayland в принципе нет специального протокольного расширения, подобного GLX. Вместо этого есть интерфейсы, позволяющие доставлять композитору DRM-буферы (в виде файловых дескрипторов DMA-BUF примерно так же, как ему доставляются «обычные» буферы в виде файловых дескрипторов разделяемой памяти). Такой подход позволяет в принципе охватить целый класс приложений, работающих с растровыми изображениями, получаемыми аппаратным образом, будь то результат рендеринга 3D-сцены GPU видеоускорителя, аппаратно декодированный кадр **W:[H.264]/W:[H.265]** видео или видеокадр, полученный видеокамерой. В листинге 8.7 показана работа демонстрационного Wayland-клиента **weston-simple-egl**, который использует программный интерфейс **W:[EGL (API)]** (аналог GLX) для OpenGL-рендеринга и **zwp_linux_dmabuf_v1** интерфейс Wayland для отправки результата отрисовки композитору.

Листинг 8.7. OpenGL-рендеринг при помощи программного интерфейса EGL в Wayland

```
homer@ubuntu:~$ WAYLAND_DEBUG=1 strace -fe ioctl /usr/lib/weston/weston-simple-egl
                ...          ...          ...          ...
❶ [2890473.319]  -> wl_display@1.get_registry(new id wl_registry@3)
                ...          ...          ...          ...
① [2890474.242] wl_registry@3.global(1, "wl_drm", 2)
↳ [2890474.417]  -> wl_registry@3.bind(1, "wl_drm", 2, new id [unknown]@10)
                ...          ...          ...          ...
② [2890479.041] wl_registry@3.global(20, "zwp_linux_dmabuf_v1", 3)
↳ [2890479.284]  -> wl_registry@3.bind(20, "zwp_linux_dmabuf_v1", 3, new id [unknown]@11)
                ...          ...          ...          ...
```

```
③ [2890482.729] wl_drm@10.device("/dev/dri/card0")
❷ ioctl(5, DRM_IOCTL_GET_MAGIC, 0x7ffe85f71eb4) = 0
↳ [2890483.635]  -> wl_drm@10.authenticate(16)
                      ...          ...          ...          ...
   ioctl(5, DRM_IOCTL_VERSION, 0x55acc6ed9a20) = 0
   ioctl(5, DRM_IOCTL_VERSION, 0x55acc6ed9a20) = 0
   strace: Process 13734 attached
                      ...          ...          ...          ...
❸ [pid 13733] ioctl(5, DRM_IOCTL_I915_GEM_CREATE, 0x7ffe85f71a40) = 0
                      ...          ...          ...          ...
                      ...          ...          ...          ...
❸ [pid 13733] ioctl(5, DRM_IOCTL_I915_GEM_MMAP, 0x7ffe85f71a80) = 0
                      ...          ...          ...          ...
❹ [2890542.462]  -> wl_compositor@4.create_surface(new id wl_surface@9)
                      ...          ...          ...          ...
   [2890572.504]  -> zwp_linux_dmabuf_v1@11.create_params(new id zwp_linux_buffer_params_v1@18)
❺ [pid 13733] ioctl(5, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffe85f7223c) = 0
   [2890572.853]  -> zwp_linux_buffer_params_v1@18.add(fd 8, 0, 0, 1024, 16777216, 2)
❻ [2890573.500]  -> zwp_linux_buffer_params_v1@18.create_immed(new id wl_buffer@19, 250, 250, ..., 0)
   [2890573.928]  -> zwp_linux_buffer_params_v1@18.destroy()
❼ [2890574.009]  -> wl_surface@9.attach(wl_buffer@19, 0, 0)
   [2890574.299]  -> wl_surface@9.damage(0, 0, 2147483647, 2147483647)
❽ [pid 13733] ioctl(5, DRM_IOCTL_I915_GEM_EXECBUFFER2, 0x7ffe85f722a0) = 0
   [pid 13733] ioctl(5, DRM_IOCTL_I915_GEM_WAIT or DRM_IOCTL_RADEON_GEM_OP, 0x7ffe85f72230) = 0
                      ...          ...          ...          ...
   [pid 13733] ioctl(5, DRM_IOCTL_I915_GEM_BUSY, 0x7ffe85f720f0) = 0
                      ...          ...          ...          ...
❾ [2890575.166]  -> wl_surface@9.commit()
                      ...          ...          ...          ...
```

После обычного разбора реестра глобальных интерфейсов ❶ и подписки на нужные интерфейсы ① и ②, при получении события ③ доступности файла «устройства» DRM производится подключение ❷ к нему. Затем при помощи уже известного (по листингу 7.40) ioctl-интерфейса к DRM ❸ создаются буферы в памяти видеоускорителя и отображаются в память процесса Wayland-клиента. После чего создается «поверхность» ❹ для отображения, DRM-буфер с результатом будущего рендеринга ❺ превращается в файловый дескриптор `fd = 8`, на основе которого создается `wl_buffer` ❻ и уже известным способом присоединяется к «поверхности» ❼. На последнем этапе при помощи DRM задействуется ❽ GPU видеоускорителя, а затем полученный результат отрисовки поступает композитору на отображение ❾.

8.3. Запуск графической среды на основе Wayland

За долгое время развития оконной системы X сложилась устоявшаяся инфраструктура запуска ее компонент и обеспечения графического входа пользователей в систему на основе так называемого менеджера дисплеев *(см. разд. 7.3.3)*. Вместе с тем, как было показано выше, Wayland-композитор является всего лишь совре-

менной реинкарнаций дисплейного сервера и композитного менеджера окон, тесно интегрированных друг с другом в одной компоненте.

В этом смысле запуск графической среды на основе Wayland мало чем отличается от среды на основе X Window System. В листинге 8.8 показан сеанс работы пользователя, запущенный менеджером **gdm3(1)**.

Листинг 8.8. Графический сеанс Wayland

```
homer@ubuntu:~$ pgrep gdm3
742
homer@ubuntu:~$ pstree -Tp 742
gdm3(742)—gdm-session-wor(20003)—gdm-wayland-ses(20042)—gnome-session-b(20046)

homer@ubuntu:~$ ps fp 742,20003,20042,20046
  PID TTY      STAT   TIME COMMAND
  742 ?        Ssl    0:00 /usr/sbin/gdm3
20003 ?        Sl     0:00  \_ gdm-session-worker [pam/gdm-launch-environment]
20042 tty1     Ssl+   0:00    \_ /usr/lib/gdm3/gdm-wayland-session ...
20046 tty1     Sl+    0:00      \_ /usr/lib/gnome-session/gnome-session-binary --systemd ...

homer@ubuntu:~$ pgrep -l gnome-shell
5628 gnome-shell
5672 gnome-shell-cal

homer@ubuntu:~$ pstree -sTp 5628
systemd(1)—systemd(5527)—gnome-shell(5628)┬Xwayland(5641)
                                           └ibus-daemon(5721)┬ibus-dconf(5726)
                                                             ├ibus-...-sim(5892)
                                                             └ibus-...-(5727)
```

Основное наблюдается в том, что в сессии отсутствует дисплейный сервер (**Xorg(1)**), т. к. его роль выполняет композитор **gnome-shell(1)**, а остальные процессы остались практически без изменения.

8.4. В заключение

Графический интерфейс, с которого начинается первое «визуальное» знакомство любого начинающего пользователя с современной операционной системой Linux, на поверку оказывается самой «сложной» ее подсистемой.

Оконные системы X Window System и Wayland и используют в своей работе практически все[1] рассмотренные сущности операционной системы — программы и

[1] Даже выход первой версии ядра Linux был практически «приурочен» к моменту успешного запуска и стабильной работы оконной системы X.

библиотеки, процессы и нити, драйверы устройств и их специальные файлы, локальные (файловые) сокеты, сетевую подсистему и сетевые сокеты, разделяемую память, отображение файлов в память, и даже службу SSH. Не менее изощренными (а может быть, и более) в использовании услуг, предоставляемых приложениям разнообразными компонентами операционной системы, являются графические среды пользователей, такие как GNOME или KDE.

Таким образом, понимание происходящих в недрах Linux процессов, стоящих за графическим интерфейсом, пожалуй, сможет служить хорошей мерой проникновения в предмет внутреннего устройства этой замечательной операционной системы.

Глава 9

Контейнеры и виртуальные машины

Как уже неоднократно упоминалось в предыдущих главах, кроме задач по управлению ресурсами центрального процессора, памяти, устройств ввода-вывода и прочих, любая операционная система, включая Linux, решает задачи по распределению их между потребителями и разграничению доступа к ним. Одним из видов разграничения доступа, помимо рассмотренных в *разд. 3.5* и *3.6*, является *изоляция*, т. е. создание определенных сред исполнения, называемых *контейнерами*, в которых потребители ресурсов «заключаются»[1] для лишения свобод обращения к полному набору ресурсов, доступных в операционной системе.

На практике контейнеры используются для разных целей, например для обеспечения безопасности. Так, «заключенная» сетевая служба, подвергшаяся атаке **W:[Denial-of-service attack]**, сможет употребить только ограниченное количество времени центрального процессора и ограниченное количество памяти, оставив системе в целом возможность для функционирования и способность обнаружения подобных ситуаций и противодействия им. Точно в том же смысле, потенциально уязвимая к атакам повышения привилегий программа (например, Web-сервер или даже Web-браузер) может быть «заключена» в контейнер для исключения ситуаций полного захвата системы злоумышленником, использовавшим уязвимости этой программы.

Кроме этого, контейнеры могут быть использованы просто для разделения вычислительных ресурсов между группами абсолютно несвязанных потребителей, например для организации совместного размещения (виртуального хостинга серверов, **W:[VPS]**) слабо- или средненагруженных информационных систем заказчиков на сверхмощных серверах провайдера. Помимо этого, т. к. контейнеры позволяют организовывать почти произвольные[2] среды исполнения, они являются удобным средством одновременного размещения программ, требующих для своей работы

---

[1] Как заключенные в местах ограничения свободы, недаром один из BSD-механизмов носит название «тюрьма» (*jail*), см. **W:[FreeBSD jail]**.

[2] Кроме ядра ОС, которое предоставляется контейнеру хост-системой.

несовместимого набора программных средств (библиотек и пр.), другими словами, дают возможность одновременно на одной системе исполнять программы, жестко привязанные к очень разным версиям операционной системы или к версиям ОС от разных поставщиков (дистрибутивам).

9.1. Чрутизация

Самым древним средством изоляции, известным со времен классического UNIX, является системный вызов chroot(2), позволяющий назначать процессам корень дерева каталогов, от которого вычисляются все абсолютные путевые имена *(см. разд. 3.1.1)*. Другими словами, каждый процесс, помимо атрибута cwd *(current working directory, см. разд. 4.5.3)*, относительно которого вычисляются относительные путевые имена, имеет еще и атрибут rtd *(root directory)*, относительно которого вычисляются абсолютные путевые имена, т. е. корень дерева каталогов оказывается абсолютом только для ядра ОС[1], а для процессов это вполне установленный каталог, выше которого процесс не имеет доступа, иными словами, *изолирован в части* настоящего дерева каталогов.

Листинг 9.1. Чрутизация

```
rick@ubuntu:~$ pgrep -l avahi-daemon
587 avahi-daemon
663 avahi-daemon
rick@ubuntu:~$ lsof -p 587 | grep rtd
avahi-dae 587 avahi  rtd       DIR                8,2     4096 262180 /etc/avahi
```

В листинге 9.1 показано типичное применение так называемой чрутизации, т. е. добровольного заключения сетевыми службами себя в некоторый каталог (например, в каталог с конфигурационными файлами или в их специальный каталог «времени исполнения»). В случае, если злоумышленник воспользуется уязвимостью такой службы, он не сможет заставить ее выдать ему содержимое файлов системы, находящихся за пределами этого каталога.

Чрутизация реализуется как раз посредством chroot(2), когда сетевая служба, закончившая свою инициализацию, назначает своим *новым* корневым каталогом некоторый каталог, все дерево выше которого должно быть ей (и потенциальному злоумышленнику) недоступно.

---

[1] Да и то, если честно, это не так. Ядро может даже для себя определять каталог, который является корнем в данный определенный момент времени.

Используя чрутизацию, уже можно организовать протоконтейнеры при помощи одноименной утилиты **chroot(1)**, которая при запуске назначает своему процессу указанный корневой каталог, а затем замещает себя при помощи системного вызова **exec(2)** на указанную программу. В результате программа оказывается изолированной в определенной части дерева каталогов. Естественно, при таком поведении утилиты **chroot(1)** в изолированном окружении можно запускать только программы, сами там расположенные. Более того, не стоит забывать, что программы при запуске компонуются с библиотеками, от которых зависят, поэтому и библиотеки, и сам компоновщик тоже должны располагаться в том же окружении.

В листинге 9.2 показано создание протоконтейнера **c-137**, в который помещаются командный интерпретатор **sh** и все необходимые компоненты для его работы, а затем он запускается в этом изолированном окружении.

Листинг 9.2. Протоконтейнер, созданный при помощи chroot

```
rick@ubuntu:~$ mkdir c-137
rick@ubuntu:~$ chroot c-137 sh
chroot: cannot change root directory to 'c-137': Operation not permitted
rick@ubuntu:~$ sudo chroot c-137 sh
? chroot: failed to run command 'sh': No such file or directory
rick@ubuntu:~$ sudo strace -fe chroot,chdir,execve chroot c-137 sh
execve("/usr/sbin/chroot", ["chroot", "c-137", "sh"], ...) = 0
chroot("c-137")                         = 0
chdir("/")                              = 0
...                ...                  ...                  ...
execve("/usr/bin/sh", ["sh"], ...) = -1 ENOENT (No such file or directory)
execve("/sbin/sh", ["sh"], ...) = -1 ENOENT (No such file or directory)
execve("/bin/sh", ["sh"], ...) = -1 ENOENT (No such file or directory)
...                ...                  ...                  ...
chroot: failed to run command 'sh': No such file or directory
+++ exited with 127 +++
rick@ubuntu:~$ mkdir c-137/bin
rick@ubuntu:~$ cp /bin/sh c-137/bin
rick@ubuntu:~$ sudo chroot c-137 sh
? chroot: failed to run command 'sh': No such file or directory
rick@ubuntu:~$ ldd /bin/sh
        linux-vdso.so.1 (0x00007ffe64cdd000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fb508897000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fb508abf000)
rick@ubuntu:~$ tar ch /lib/x86_64-linux-gnu/libc.so.6 /lib64/ld-linux-x86-64.so.2 | ↵
> tar x -C c-137
tar: Removing leading `/' from member names
```

```
tar: Removing leading `/' from hard link targets
rick@ubuntu:~$ find c-137 -ls
12312      4 drwxrwxr-x   5 rick   rick     4096 янв 11 14:11 c-137/
12315      4 drwxrwxr-x   3 rick   rick     4096 янв 11 14:11 c-137/lib
12316      4 drwxrwxr-x   2 rick   rick     4096 янв 11 14:17 c-137/lib/x86_64-linux-gnu
12317   1980 -rwxr-xr-x   1 rick   rick  2025032 сен 16 17:56 c-137/lib/x86_64-linux-gnu/libc.so.6
12318      4 drwxrwxr-x   2 rick   rick     4096 янв 11 14:17 c-137/lib64
12319    184 -rwxr-xr-x   1 rick   rick   187376 сен 16 17:56 c-137/lib64/ld-linux-x86-64.so.2
12313      4 drwxrwxr-x   2 rick   rick     4096 янв 11 14:11 c-137/bin
12314    128 -rwxr-xr-x   1 rick   rick   129816 янв 11 14:11 c-137/bin/sh
rick@ubuntu:~$ sudo chroot c-137 sh
# pwd
/
# ls
sh: 7: ls: not found
```

Как и ожидалось, в новом окружении ни одна внешняя команда интерпретатора не доступна, а само окружение не очень пригодно для интерактивной работы, поэтому зачастую вместо обычного командного интерпретатора в подобных окружениях используют интерпретатор из проекта W:[busybox], который реализует массу полезных POSIX.1 команд встроенным образом (листинг 9.3).

Листинг 9.3. Протоконтейнер, созданный при помощи chroot и busybox

```
rick@ubuntu:~$ which busybox
/usr/bin/busybox
rick@ubuntu:~$ cp /usr/bin/busybox c-137/bin
rick@ubuntu:~$ sudo chroot c-137/ busybox sh
BusyBox v1.30.1 (Ubuntu 1:1.30.1-4ubuntu4) built-in shell (ash)
Enter 'help' for a list of built-in commands.

/ # ls
bin    lib    lib64
/ # help
Built-in commands:
------------------
        . : [ [[ alias bg break cd chdir command continue echo eval exec
        exit export false fg getopts hash help history jobs kill let
                    ...          ...          ...                ...
        logname logread losetup ls lsmod lsscsi lzcat lzma lzop md5sum
                    ...          ...          ...                ...
```

```
      mktemp modinfo modprobe more mount mt mv nameif nc netstat nl
                  ...          ...          ...              ...
      paste patch pidof ping ping6 pivot_root poweroff printf ps pwd
                  ...          ...          ...              ...
      whoami xargs xxd xz xzcat yes zcat
/ # ps
PID   USER     COMMAND
ps: can't open '/proc': No such file or directory
/ # lsmod
lsmod: /proc/modules: No such file or directory
Module                  Size  Used by    Not tainted
/ # lsscsi
lsscsi: can't change directory to '/sys/bus/scsi/devices': No such file or directory
```

Однако многие утилиты, например **ps(1)**, отказываются работать в таком изолированном окружении в силу изоляции от той части дерева каталогов (каталоги /proc и /sys), где при помощи псевдофайловой системы **proc** или **sysfs** *(см. разд. 3.4.4)* ядром экспортируется информация о процессах и устройствах. Это в принципе и есть ожидаемый результат изоляции в случаях, когда она используется для предотвращения утечки информации третьим лицам, однако для организации работы хостинга это не тот результат, который требуется получить. Естественным решением в таком случае является повторное монтирование файловых систем **proc** и/или **sysfs** внутрь окружения.

Листинг 9.4. Протоконтейнер, созданный при помощи chroot, busybox, proc и sysfs

```
rick@ubuntu:~$ sudo chroot c-137/ busybox sh
                  ...          ...          ...              ...
/ # ps
PID   USER     COMMAND
ps: can't open '/proc': No such file or directory
/ # mount
? sh: mount: not found
! / # ln -s /bin/busybox /bin/mount
/ # mount
mount: no /proc/mounts
/ # mkdir /proc
❶ / # mount -t proc none /proc
❷ / # mount
① none on /proc type proc (rw,relatime)
❸ / # ps
③ PID   USER     COMMAND
    1 0        {systemd} /sbin/init splash
                  ...          ...          ...              ...
```

```
   9743 0        {ps} busybox sh
   / # exit
   rick@ubuntu:~$ mount -t proc
   proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
② none on /home/rick/c-137/proc type proc (rw,relatime)
```

В листинге 9.4 показано монтирование ❶ файловой системы /proc внутри протоконтейнера, после чего утилиты ps(1) ❸ и mount(8) ❷ начинают исправно функционировать. Однако нужно заметить, что если внутри контейнера видны только файловые системы, смонтированные ниже его корневого каталога ①, то в самой системе видны абсолютно все смонтированные в ее дерево файловые системы, включая те, которые смонтированы внутри и для контейнеров ②, что доставляет некоторые неудобства. Более того, в протоконтейнере видны абсолютно все процессы операционной системы ③, что в принципе ожидаемо, т. к. механизм чрутизации изначально предназначался для изоляции в *пространстве* дерева каталогов и никакого отношения к пространству процессов не имеет.

Еще одно замечание по поводу busybox в листинге 9.4 касается того, что далеко не все команды реализуются им как «встроенные» в его интерпретатор. Многие из них, например mount или cat и подобные, вызываются напрямую другими программами и, как следствие, должны быть «внешними» в окружении, как это обычно бывает в «нормальных» случаях. Именно поэтому они должны иметь имена в каталогах /bin или /sbin, которые просто линкуются на сам busybox, который, как и в случае со встроенными командами, их и реализует.

9.2. Пространства имен

Для «настоящей» контейнеризации, т. е. изоляции в других пространствах, отличных от дерева каталогов, в Linux появились так называемые *пространства имен (namespaces)*, которые на деле оказываются достаточно простым механизмом, не многим сложнее chroot(2).

Например, пространства имен процессов (*PID namespaces*, нужно заметить, что процессы «именуются» своими идентификаторами) позволяют изолировать процессы так, что они «видят» друг друга, только если находятся в одном пространстве. Более того, в каждом пространстве нумерация процессов начинается с единицы, а идентификаторы разных процессов из разных пространств могут совпадать.

Точно так же, пространства монтирований *(mount namespaces)* позволяют процессам видеть только файловые системы, смонтированные в их пространстве. Аналогично, сетевые пространства *(net namespaces)* изолируют сетевые интерфейсы и сокеты, пространства средств межпроцессного взаимодействия *(IPC namespaces)*

изолируют разделяемую память, очереди сообщений и семафоры *(см. разд. 4.9.5)*, а пространства пользователей *(user namespaces)* изолируют пользовательские идентификаторы, которые могут в каждом пространстве начинаться с 0 (!), т. е. обеспечивать контейнерам собственного «виртуального» пользователя root. На текущий момент различают 8 различных типов пространств, подробную информацию о которых можно найти в странице руководства namespaces(7).

По умолчанию все процессы выполняются в одних и тех же пространствах и, как следствие, совместно используют *(share)* ресурсы, но при порождении процесса системным вызовом clone(2)[1] или позднее при работе процесса при помощи системного вызова unshare(2) можно создать новое пространство, в которое и будет перемещен процесс. А по умолчанию, согласно традициям UNIX, процессы просто унаследуют пространства своих родителей при порождении, т. е. продолжат выполняться в одних и тех же пространствах.

Как и большинство сущностей в Linux, пространства имен являются файлами в некоторой псевдофайловой системе ❶, однако, в отличие от «нормальных» файлов, они не имеют имен и идентифицируются только номерами ❷ их индексных дескрипторов inode. Показать номера пространств процессов может утилита ps(1), поскольку она получает любую информацию о процессах (в том числе идентификаторы их пространств) через псевдофайловую систему /proc (листинг 9.5).

Листинг 9.5. Идентификаторы пространств имен процесса

```
rick@ubuntu:~$ ps up $$
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
rick      2314  0.0  0.2  19844  5788 pts/0    Ss   13:07   0:01 -bash
rick@ubuntu:~$ ls -la /proc/$$/ns
total 0
dr-x--x--x 2 rick rick 0 янв 11 16:25 .
dr-xr-xr-x 9 rick rick 0 янв 11 13:07 ..
                ...          ...          ...          ...
lrwxrwxrwx 1 rick rick 0 янв 11 16:25 mnt -> 'mnt:[4026531840]'
lrwxrwxrwx 1 rick rick 0 янв 11 16:25 net -> 'net:[4026531992]' ❷
lrwxrwxrwx 1 rick rick 0 янв 11 16:25 pid -> 'pid:[4026531836]'
                ...          ...          ...          ...
rick@ubuntu:~$ ps o pid,netns,mntns,pidns,comm p $$
  PID       NETNS          MNTNS       PIDNS COMMAND
10149 4026531992 ❷ 4026531840 4026531836 bash
```

[1] Который определяет, будут ли совместно использованы память, файловые дескрипторы и другие ресурсы между порождающим и порожденным процессами.

```
rick@ubuntu:~$ stat -L /proc/$$/ns/net
  File: /proc/10149/ns/net
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: 4h/4d ❶         Inode: 4026531992 ❷ Links: 1
                      ...          ...          ...              ...
```

С возникновением системного вызова **unshare(2)** появилась и одноименная утилита **unshare(1)**, позволяющая воспользоваться механизмом пространств для организации «настоящих» контейнеров. Подобно утилите **chroot(1)**, она создает заданные новые пространства (`-mnp`), затем порождает процесс (`-f`) и выполняет в нем заданную программу (листинг 9.6).

Листинг 9.6. Создание изолированного контейнера при помощи unshare

```
rick@ubuntu:~$ ps o pid,netns,mntns,pidns,comm p $$
  PID       NETNS      MNTNS      PIDNS COMMAND
❶ 10149 4026531992 4026531840 4026531836 bash
rick@ubuntu:~$ sudo unshare -mnp -f -R c-137 --mount-proc busybox sh
unshare: unshare failed: Operation not permitted
rick@ubuntu:~$ sudo unshare -mnp -f -R c-137 --mount-proc busybox sh
/ # ls -l /proc/$$/ns
                    ...          ...          ...              ...
lrwxrwxrwx    1 0        0                0 Jan 11 14:08 mnt -> mnt:[4026532251]
lrwxrwxrwx    1 0        0                0 Jan 11 14:08 net -> net:[4026532254]
lrwxrwxrwx    1 0        0                0 Jan 11 14:08 pid -> pid:[4026532252]
                    ...          ...          ...              ...
/ # ps afx
❷ PID   USER     COMMAND
!☛    1 0        busybox sh
      2 0        {ps} busybox sh
❸ / # ip l
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
/ # exit

rick@ubuntu:~$ ip l
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN ... qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
❹ 2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel ... qlen 1000
    link/ether 08:00:27:a9:78:36 brd ff:ff:ff:ff:ff:ff
❺ rick@ubuntu:~$ mount -t proc
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
```

Ожидаемым результатом создания новых пространств (процессов -p, монтирования -m и сети -n) стала недоступность информации о процессах других пространств ❶ кроме своего собственного ❷, недоступность ❸ в нем сетевых интерфейсов других пространств ❹ (за исключением одноименного, но собственного экземпляра loopback интерфейса) и недоступность другим пространствам смонтированных файловых систем ❺ (сравните с ② в листинге 9.4). Особенным комментарием к листингу 9.6 нужно отметить, что утилита unshare(2) умеет самостоятельно использовать системный вызов chroot(1) для изоляции дерева каталогов (-R) и монтировать файловую систему /proc внутри контейнера, что очень удобно.

Получаемые таким образом контейнеры почти полностью[1] изолированы от основной операционной системы (и друг от друга), что позволяет использовать подобный подход для изолированного запуска сетевых служб или организации виртуального хостинга. Однако вместе с тем возникает обратная задача организации связи этих изолированных окружений с внешним миром. Например, для сетевого взаимодействия контейнеру должны быть так или иначе доступны сетевые интерфейсы. Для связи с основной ОС[2] или другими контейнерами зачастую удобно использовать общие каталоги и файлы, в том числе файлы локальных сокетов *(см. разд. 4.9.4)*. Кроме того, возникает необходимость и просто выполнять те или иные программы в уже *существующих* пространствах, где работают изолированные программы.

В листинге 9.7 показано обеспечение сетевой связи между изолированным контейнером c-137 и хост-системой при помощи создания виртуального Ethernet-коммутатора ❷ (type veth) с двумя виртуальными интерфейсами net0 ② и c137net0 ① для контейнера и хост-системы соответственно. Затем интерфейс net0 был помещен ❸ в то сетевое пространство имен, которое было создано при запуске первой программы контейнера ❶ и которому впоследствии было назначено имя anotherdimension ❹.

Интерфейсы контейнера и хост-системы конфигурируются обычным способом *(см. разд. 6.2.1)*, им назначаются IP-адреса ❺ и ❻, мастер-интерфейс активируется ❼, после чего проверяется работоспособность соединения ❽ и ❾.

Листинг 9.7. Перемещение сетевых интерфейсов между сетевыми пространствами имен

pts/1

```
rick@ubuntu:~$ sudo unshare -mpn -f -R c-137 --mount-proc busybox sh ❶
/ # ip link
1: lo: <LOOPBACK> mtu 65536 qdisc noop qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

[1] Осталось еще несколько пространств, подлежащих изоляции, которые таким же образом можно создать при запуске изолируемой программы утилитой unshare(1).

[2] Называемой зачастую «хост»-системой, другими словами, «ведущей» системой.

pts/2

```
rick@ubuntu:~$ pgrep busybox
12089
rick@ubuntu:~$ sudo ip netns attach anotherdimension 12089
rick@ubuntu:~$ ip netns list
anotherdimension ❶
                                  ↓ ①                          ↓ ②
rick@ubuntu:~$ sudo ip link add c137net0 type veth peer name net0 ❶
rick@ubuntu:~$ sudo ip link show
                    ...              ...              ...              ...
4: net0@c137net0: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN ... qlen 1000
    link/ether a6:71:65:45:fd:c2 brd ff:ff:ff:ff:ff:ff
5: c137net0@net0: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN ... qlen 1000
    link/ether e2:bd:81:ee:fe:bf brd ff:ff:ff:ff:ff:ff
rick@ubuntu:~$ sudo ip link set net0 netns anotherdimension ●
```

pts/1

```
/ # ip link show
4: net0@if20: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop qlen 1000
    link/ether a6:71:65:45:fd:c2 brd ff:ff:ff:ff:ff:ff
/ # ip addr add 10.0.0.137/24 dev net0 ●
/ # ip addr show dev net0
4: net0@if20: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop qlen 1000
    link/ether a6:71:65:45:fd:c2 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.137/24 scope global net0
       valid_lft forever preferred_lft forever
```

pts/2

```
rick@ubuntu:~$ ip link show dev c137net0
5: c137net0@if19: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN ... qlen 1000
    link/ether e2:bd:81:ee:fe:bf brd ff:ff:ff:ff:ff:ff link-netns anotherdimension
rick@ubuntu:~$ sudo ip addr add 10.0.0.1/24 dev c137net0 ●
rick@ubuntu:~$ ip link set dev c137net0 up ●
rick@ubuntu:~$ ip addr show dev c137net0
20: c137net0@if19: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop state UP ...
    link/ether e2:bd:81:ee:fe:bf brd ff:ff:ff:ff:ff:ff link-netns anotherdimension
    inet 10.0.0.1/24 scope global c137net0
       valid_lft forever preferred_lft forever
rick@ubuntu:~$ ping 10.0.0.137
PING 10.0.0.137 (10.0.0.137) 56(84) bytes of data.
64 bytes from 10.0.0.137: icmp_seq=1 ttl=64 time=0.085 ms ❻
64 bytes from 10.0.0.137: icmp_seq=2 ttl=64 time=0.116 ms
^C
```

```
--- 10.0.0.137 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1007ms
rtt min/avg/max/mdev = 0.085/0.100/0.116/0.015 ms
```

```
/ # ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1): 56 data bytes
64 bytes from 10.0.0.1: seq=0 ttl=64 time=0.165 ms ❶
64 bytes from 10.0.0.1: seq=1 ttl=64 time=0.195 ms
^C
--- 10.0.0.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.165/0.180/0.195 ms
```

В листинге 9.8 в контейнере исполняется интерактивный командный интерпретатор, что позволяет запускать команды конфигурирования «изнутри». На практике же в контейнере сразу запускают определенную системную службу, а конфигурационные команды — в интерпретаторе хост-системы, перемещая их в нужное пространство имен нужного контейнера при помощи утилиты **nsenter(1)**, которая использует для этой цели системный вызов **setns(2)**.

Листинг 9.8. Выполнение программ в целевых пространствах имен

```
rick@ubuntu:~$ pgrep busybox
12089
rick@ubuntu:~$ ip link
                ...             ...             ...             ...
5: c137net0@if19: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ... state UP ... qlen 1000
    link/ether e2:bd:81:ee:fe:bf brd ff:ff:ff:ff:ff:ff link-netns anotherdimension
                ...             ...             ...             ...
rick@ubuntu:~$ sudo nsenter -t 12089 -n ip link
                ...             ...             ...             ...
4: net0@if20: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ... state UP ... qlen 1000
    link/ether a6:71:65:45:fd:c2 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

9.3. Контейнеризация: runc и docker

Изолированные окружения для выполнения программ, создаваемые при помощи механизмов изоляции процессов операционной системы, принято называть *контейнерами*. Сам принцип такого предоставления ресурсов ОС приложениям принято называть *контейнеризацией* и считать одним из видов виртуализации[1]. Одна-

[1] См. **W:[Virtualization]** для отличия аппаратной виртуализации, паравиртуализации и виртуализации на уровне ОС (контейнеризации).

ко прямое использование механизмов и базовых утилит, как это иллюстрируется в предыдущем разделе, требует выполнения колоссального количества типовых действий для организации каждого контейнера, что и привело к созданию специализированных систем управления контейнерами, таких как W:[Docker (software)] или W:[LXC] (в том числе W:[OpenVZ], хотя она использует свои механизмы изоляции и поэтому требует специально модифицированного ядра).

Более того, изначальная разрозненность систем контейнеризации привела в конце концов к созданию инициативы OCI (W:[Open Container Initiative]), под эгидой которой была согласована спецификация «исполнителя» контейнеров *(runtime)* и спецификация формата образов *(image)* контейнеров, которые должен понимать это «исполнитель». В Linux обе OCI-спецификации реализуются утилитой runc(8), которая, в свою очередь, основывается на сервисах ядра Linux, включая механизмы изоляции, рассмотренные выше. Сами же системы контейнеризации, например Docker, используют утилиту runc(8) для непосредственного запуска контейнеров.

Листинг 9.9. OCI-исполнитель runc и запуск OCI-контейнера

```
❶ rick@ubuntu:~$ runc spec
  rick@ubuntu:~$ ls
  config.json
  rick@ubuntu:~$ sed -n '/root.*{/,/}/p' config.json
      "root": {
❷             "path": "rootfs",
              "readonly": true
      },
❸ rick@ubuntu:~$ sudo debootstrap --variant=minbase xenial rootfs
  I: Retrieving InRelease
  I: Checking Release signature
  I: Valid Release signature (key id 790BC7277767219C42C86F933B4FE6ACC0B21F32)
  I: Retrieving Packages
  I: Validating Packages
                ⓫  ⓬      ⓭      ⓮      ⓯      ⓰      ⓱      ⓲      ⓳
  I: Base system installed successfully.
  rick@ubuntu:~$ ps o pid,netns,mntns,pidns,comm p $$
    PID      NETNS      MNTNS      PIDNS COMMAND
⓪ 10149 4026531992 4026531840 4026531836 bash

❶ rick@ubuntu:~ $ sudo runc run c-132
  # ps o pid,netns,mntns,pidns,comm p $$
    PID      NETNS      MNTNS      PIDNS COMMAND
⓵     1 4026532375 4026532251 4026532373 sh
  # ps axf
    PID TTY      STAT   TIME COMMAND
```

```
    ☛ 1 pts/0    Ss     0:00 sh
      9 pts/0    R+     0:00 ps axf
  # ip l
② 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode ... qlen 1000
      link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

В листинге 9.9 показано, насколько утилита **runc(1)** облегчает запуск контейнера. На первом шаге ❶ создается шаблонная OCI-конфигурация образа контейнеров в файле `config.json`, которая по умолчанию предполагает наличие **(ch)root**-каталога ❷ с именем `rootfs`. Совокупность конфигурационного файла и корневого каталога контейнера (упрощая) и есть его образ (image).

На втором шаге ❸ при помощи штатной **Ubuntu/Debian**-утилиты **debootstrap(8)** в каталог `rootfs` инсталлируется минимальный набор пакетов **Ununtu 16.04** (`xenial`), включающий командный интерпретатор, базовые утилиты и все необходимые им библиотеки. Кроме того, инсталлируется дополнительный пакет `iproute2`, поставляющий утилиту **ip(8)** для конфигурирования сетевых интерфейсов. На последнем, завершающем, шаге на основе полученного образа собственно запускается ❹ один экземпляр контейнера с именем `c-132`, первым процессом в котором запускается командный интерпретатор `sh` (согласно конфигурации по умолчанию).

Как и ожидалось, процессы внутри контейнера выполняются в своих пространствах имен ①, а в сетевое пространство не добавлен ни один интерфейс. Конфигурированием самого контейнера, например созданием сетевых интерфейсов и помещением их в соответствующее сетевое пространство имен утилита **runc(1)** не занимается, т. к. ее основное назначение согласно спецификации «исполнять» контейнер, а не конфигурировать. Именно системы контейнеризации, подобные **Docker**, являются законченным набором инструментов для создания, конфигурирования, запуска, останова и удаления контейнеров, а также манипулирования их образами.

Листинг 9.10. Запуск Docker-контейнера

```
rick@ubuntu:~ $ docker image ls
Got permission denied while trying to connect to the Docker daemon socket at
unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.40/images/json:
dial unix /var/run/docker.sock: connect: permission denied
rick@ubuntu:~ $ ls -l /var/run/docker.sock
❶ srw-rw---- 1 root docker 0 янв 11 11:11 /var/run/docker.sock=
rick@ubuntu:~ $ id
? uid=1000(rick) gid=1000(rick) groups=1000(rick),4(adm),...,27(sudo),...,131(sambashare)
❷ rick@ubuntu:~ $ sudo gpasswd -a rick docker
rick@ubuntu:~ $ logout
```

```
   rick@ubuntu:~ $ id
?  uid=1000(rick) gid=1000(rick) groups=1000(rick),4(adm),...,27(sudo),...,132(docker)
❶  rick@ubuntu:~ $ docker image ls
?  REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
❹  rick@ubuntu:~ $ docker image pull ubuntu:16.04
   16.04: Pulling from library/ubuntu
   3386e6af03b0: Downloading [==================================================>  ]  41.38MB/44.12MB
   49ac0bbe6c8e: Download complete
   d1983a67e104: Download complete
   1a0f3a523f04: Download complete
                 ...         ...         ...         ...
   Digest: sha256:181800dada370557133a502977d0e3f7abda0c25b9bbb035f199f5eb6082a114
   Status: Downloaded newer image for ubuntu:16.04
   docker.io/library/ubuntu:16.04
   rick@ubuntu:~ $ docker image ls
   REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
   ubuntu              16.04               c6a43cd4801e        3 weeks ago         123MB
❺  rick@ubuntu:~ $ docker run -ti -h c-123 -n c-123 ubuntu:16.04
   root@c-123:/#
```

В листинге 9.10 показана процедура запуска Docker-контейнера при помощи утилиты docker(1). В реальности эта утилита является всего лишь пользовательским интерфейсом к Docker-службе dockerd, которая использует, в свою очередь, службу управления контейнерам containerd, запускающую контейнеры при помощи все той же runc(1), эксплуатирующей механизмы изоляции ядра[1].

Обращаться к службе dockerd посредством файлового сокета /var/run/docker.sock, оказывается, можно только членам группы docker ❶, поэтому после добавления пользователя rick в эту группу ❷ у непривилегированного пользователя появляется право ❸ управлять контейнерами. Для запуска контейнера, как обычно, требуется иметь его образ, но при использовании Docker его можно скачать ❹ из централизованного «хаба» docker.io, куда публикуются как официальные образы от поставщиков программного обеспечения, так и образы контейнеров, от всех желающих поделиться. Запуск контейнера производится одной командой ❺, но сам контейнер гораздо более подготовлен к конечному использованию (листинг 9.11).

Листинг 9.11. Анализ Docker-контейнера

```
root@c-123:/# ps o pid,netns,mntns,pidns,comm p $$
  PID        NETNS      MNTNS      PIDNS COMMAND
```

[1] В доме, который построил Джек. Ну, в смысле в ядре, которое написал W:[Linus Torvalds].

```
①      1 4026532378 4026532373 4026532376 bash  •
   root@c-123:/# apt update
                  ...              ...              ...              ...
   root@c-123:/# apt install iproute2
                  ...              ...              ...              ...
•  root@c-123:/# ip link
   1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN ... qlen 1000
       link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
②  26: eth0@if27: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP ...
       link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
❶  root@c-123:/# Ctrl+P Ctrl+Q

   rick@ubuntu:~ $ docker ps
   CONTAINER ID  IMAGE         COMMAND      CREATED        STATUS        PORTS      NAMES
   d54bd8ab3100  ubuntu:16.04  "/bin/bash"  4 minutes ago  Up 4 minutes             c-123
   rick@ubuntu:~ $ ps o pid,netns,mntns,pidns,comm p $$
     PID      NETNS      MNTNS      PIDNS COMMAND
③  21357 4026531992 4026531840 4026531836 bash
   rick@ubuntu:~ $ ip link
                  ...              ...              ...              ...
④  27: veth72908b6@if26: <BROADCAST,MULTICAST,UP,LOWER_UP>  ... master docker0 state UP ...
       link/ether 2a:4a:c7:0b:9d:1f brd ff:ff:ff:ff:ff:ff link-netnsid 1
❷  rick@ubuntu:~ $ docker attach c-123
   root@c-132:/#
```

Анализ Docker-контейнера в листинге 9.11 в очередной раз показывает, что в его основе лежат базовые механизмы изоляции на основе пространств имен ① и ③, только сетевое пространство имен контейнера и хост-системы оказываются ② ④ предварительно сконфигурированными. В них помещены парные интерфейсы veth72908b6 и eth0 виртуального Ethernet-коммутатора, обеспечивающие связь между виртуальной машиной и хост-системой по сценарию, похожему на изложенный ранее в листинге 9.6. Кроме того, Docker предоставляет удобный способ отключиться от интерактивной консоли контейнера ❶, а затем подключиться к ней обратно ❷.

9.4. Группы управления (cgroups)

Вместе с появлением в операционной системе новых сущностей — контейнеров, между которыми разделяются ее ресурсы, неизбежно возникает запрос на управление количественным распределением ресурсов центрального процессора, памяти и устройств ввода-вывода между ними. Контейнеры — по сути, это группы процессов, выполняющиеся в отдельных пространствах имен (и с отдельными корневыми каталогами). Именно для количественного распределения ресурсов между группами

процессов и был разработан механизм групп управления (*control groups*, они же *cgroups*, *см.* cgroups(7)), который нашел применение как для ограничения в ресурсах группы процессов, выполняющихся в контейнере, так и для ограничения других групп, например групп процессов, принадлежащих тому или иному сервису или пользовательскому сеансу (*см. разд. 10.3*).

По умолчанию все процессы находятся в одной, так называемой «корневой» группе, но механизм позволяет произвольно создавать подгруппы в любых группах, образуя иерархию групп и перемещать туда индивидуальные процессы (на самом деле задачи, т. е. ядерные нити процессов, *см. разд. 4.2*). Затем на группы этой иерархии накладываются ресурсные ограничения, зависящие от так называемого «контроллера» (*controller*), назначенного этой иерархии[1].

На текущий момент различают чёртову дюжину различных контроллеров: cpu, накладывающий ограничения на справедливую долю (cpu share) процессорного времени; cpuacct, просто подсчитывающий потребленные ресурсы ЦП; cpuset, привязывающий процессы к конкретным процессорам; memory, ограничивающий потребление памяти; devices, ограничивающий доступ к файлам устройств; freezer, позволяющий приостанавливать и возобновлять работу групп процессов; net_cls и net_prio, назначающий сетевому трафику процессов класс и приоритет обработки для использования сетевыми фильтрами и планировщиками; blkio, ограничивающий количество запросов к блочным (дисковым) устройствам при помощи планировщика CFQ (*см. разд. 4.6.2*); pids, накладывающий ограничение на количество процессов в группе. Кроме вышеперечисленных контроллеров, имеется еще несколько экзотических и наименее распространенных на практике: perf_event, huge_tlb и rdma (см. cgroups(7)).

Интерфейс, при помощи которого иерархию групп управления можно создавать, модифицировать и накладывать конкретные значения ограничений, выполнен при помощи очередной псевдофайловой системы cgroup, подобной proc и sysfs, а ее анализ использования системой контейнеризации Docker показан в листинге 9.12.

Листинг 9.12. Использование групп управления службой Docker

```
rick@ubuntu:~ $ mount -t cgroup
         ...          ...          ...          ...
                      ↓ ①                        ↓   ↓ ①
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
          ...       ...          ...          ...
```

---

[1] Различают две версии механизма cgroups(7): v1, в которой создаются независимые иерархии групп задач (процессов и нитей) по каждому из контроллеров в отдельности, и v2, где гораздо более удобным образом (но менее гибким) создается одна иерархия групп процессов (но не нитей), а затем уже каждой группе назначают набор контроллеров.

```
                                                                    ↓ ②
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
                ...                  ...                ...                ...
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
rick@ubuntu:~ $ ls /sys/fs/cgroup
cgroup.clone_children              memory.limit_in_bytes
                ...                   ...              ...              ...
❶ docker                           memory.oom_control
❷ init.scope                       memory.pressure_level
                ...                   ...              ...              ...
memory.kmem.tcp.limit_in_bytes     system.slice ❸
memory.kmem.tcp.max_usage_in_bytes tasks
memory.kmem.tcp.usage_in_bytes     user.slice ❹
memory.kmem.usage_in_bytes
rick@ubuntu:~ $ find /sys/fs/cgroup/docker -type d
/sys/fs/cgroup/memory/docker/
❺ /sys/fs/cgroup/memory/docker/d54bd8ab3100ca63041fe5780bc7095e2a0f6729df5e71d7c6474ac685bf5bc7
rick@ubuntu:~ $ docker ps
CONTAINER ID    IMAGE          COMMAND       CREATED         STATUS         PORTS          NAMES
d54bd8ab3100    ubuntu:16.04   "/bin/bash"   13 hours ago    Up 13 hours                   c-123
rick@ubuntu:~ $ cd /sys/fs/cgroup/memory/docker/d54bd8ab3100ca6304...d7c6474ac685bf5bc7
rick@ubuntu:/sys/fs/.../d54b...5bc7$ ls
cgroup.clone_children              memory.limit_in_bytes ④
cgroup.event_control               memory.max_usage_in_bytes
                ...                   ...              ...              ...
memory.kmem.limit_in_bytes         memory.soft_limit_in_bytes ④
                ...                   ...              ...              ...
memory.kmem.slabinfo               memory.swappiness
memory.kmem.tcp.failcnt            memory.usage_in_bytes ⑤
memory.kmem.tcp.limit_in_bytes     memory.use_hierarchy
memory.kmem.tcp.max_usage_in_bytes notify_on_release
memory.kmem.tcp.usage_in_bytes     tasks ③
memory.kmem.usage_in_bytes
rick@ubuntu:/sys/fs/.../d54b...5bc7$ cat tasks
22358
rick@ubuntu:/sys/fs/.../d54b...5bc7$ cat memory.limit_in_bytes memory.soft_limit_in_bytes
9223372036854771712
9223372036854771712
rick@ubuntu:/sys/fs/.../d54b...5bc7$ cat memory.usage_in_bytes
33243136
```

Из листинга видно, что в системе определены 11 иерархий групп процессов, каждой из которых назначено по одному ②, реже по два ① контроллера. Подключение иерархии производится просто путем монтирования файловой системы cgroup ⓪ в какое-либо место дерева каталогов (по соглашению, в подкаталоги каталога /sys/fs/cgroup).

В самих иерархиях, например в /sys/fs/cgroup/memory, созданы группы с названиями docker ❶, init.scope ❷, system.slice ❸ и init.slice ❹. Группа docker и ее подгруппы

создаются и управляются службой **Docker** для ограничения контейнеров в доступных им ресурсах (в данной иерархии — для ограничения в использовании оперативной памяти). Остальные группы в тех же целях созданы службой **systemd**, заведующей запуском и остановом всех служб операционной системы и пользовательских сеансов *(см. разд. 10.3)*. В группе **docker** создана всего одна подгруппа ❻, совпадающая с UUID контейнера, создание которого иллюстрировалось в листинге 9.11, а в самой группе находится ③ всего один процесс **PID = 22358**, при этом группе не назначены[1] ни мягкое, ни жесткое ограничения ④ по памяти, при этом группа потребила ⑤ порядка 31 Мб памяти.

Листинг 9.13. Использование групп управления для собственных нужд

```
❶ rick@ubuntu:~ $ ps o pid,cgroup:512,cmd p $$
    PID CGROUP                                                    CMD
  23746 ...,3:cpu,cpuacct:/user.slice,...                          -bash
❷ rick@ubuntu:~ $ sudo mkdir /sys/fs/cgroup/cpu,cpuacct/mygroup
❸ rick@ubuntu:~ $ echo $$ | sudo tee /sys/fs/cgroup/cpu,cpuacct/mygroup/tasks
  rick@ubuntu:~ $ cat /sys/fs/cgroup/cpu,cpuacct/mygroup/tasks
  23746
★ 25110
  rick@ubuntu:~ $ ps o pid,cgroup:512,cmd p $$
    PID CGROUP                                                    CMD
  23746 ...,3:cpu,cpuacct:/mygroup,...                             -bash
  rick@ubuntu:~ $ ls /sys/fs/cgroup/cpu,cpuacct/mygroup/cpu.*
  /sys/fs/cgroup/cpu,cpuacct/mygroup/cpu.cfs_period_us
  /sys/fs/cgroup/cpu,cpuacct/mygroup/cpu.shares
  /sys/fs/cgroup/cpu,cpuacct/mygroup/cpu.cfs_quota_us
  /sys/fs/cgroup/cpu,cpuacct/mygroup/cpu.stat
  rick@ubuntu:~ $ cat /sys/fs/cgroup/cpu,cpuacct/mygroup/cpu.shares
  1024
❹ rick@ubuntu:~ $ echo 256 | sudo tee /sys/fs/cgroup/cpu,cpuacct/mygroup/cpu.shares
```

В листинге 9.13 проиллюстрирована процедура создания группы, помещения в нее процесса и накладывания ограничений на использование центрального процессора. Утилита **ps(1)** умеет ❶ показывать членство процессов в группах управления, хотя и делает это не очень удобным образом. Оказывается, что по умолчанию **systemd(1)** (и его напарник **systemd-logind(8)**) размещают интерактивные пользовательские процессы в группах с названием **user.slice**, что, например, удобно использовать для

[1] На самом деле они «как бы» назначены, но кого сможет ограничить 8 Эиб (экзабайт, 2^{60})?

ограничения всех пользовательских процессов. Однако, если необходимо дифференцировать отдельные процессы в определенную группу, то ее для начала нужно создать ❷, затем поместить туда нужные процесс ❸, а впоследствии определить количественные значения нужных ограничений ❹. В данном примере все процессы группы не смогут потребить больше 256/1024 = 1/4 процессорного времени. Этими процессами являются явно помещенный в группу командный интерпретатор с PID = 23746 и любые его потомки, например процесс команды cat ★, выполнявшийся после помещения командного интерпретатора в группу.

9.5. В заключение

Реализация принципа контейнеризации в Linux, как может показаться на первый взгляд, является не таким уж сложным и запутанным делом. Вокруг достаточно простых механизмов изоляции, группировки и ограничения изолированных групп процессов, в ресурсах построены современные и чрезвычайно удобные системы, такие как Docker и ему подобные. Эти системы широко используются на практике — от организации виртуального хостинга серверов до систем сборки и тестирования программного обеспечения.

Кроме этих, очевидных на первый взгляд приложений, контейнеризация находит неявное применение даже на обычных настольных системах обычных пользователей. Так, например, в современном ❶ Ununtu Linux она используется системой установки дистрибутиво-независимого программного обеспечения W:[Snappy][1], где каждый пакет программного обеспечения по сути представляет собой контейнер с бинарным образом необходимых компонент (библиотек и пр.), программы которого запускаются в изолированном окружении.

Таким образом, контейнеризация становится неотъемлемой частью современного Linux, а понимание принципов ее работы — неотъемлемым знанием современного технического специалиста.

[1] Рассмотрение самого Snappy выходит за рамки книги хотя бы потому, что его распространенность на текущий день ограничена только Ubuntu Linux, а будущее сомнительно в силу постоянной и к тому же заслуженной критики.

Глава 10

От отдельных компонент — к системе

В предыдущих главах рассматривались отдельные компоненты операционной *системы* и некоторые «теоретические» принципы их функционирования, понимание чего имеет и самостоятельную ценность. Однако гораздо более глубокое проникновение в предмет обычно проявляется при выстраивании тех или иных связей между отдельными сущностями. Ответы на многие «практические» вопросы об этих связях (которые неминуемо обязаны были возникать у внимательного читателя при осваивании предыдущего материала) содержатся в настоящей главе, подводящей итоги под всем экскурсом во внутреннее устройство операционной системы Linux.

10.1. Как Linux загружается

Процесс загрузки абсолютно любой операционной системы начинается с той компоненты, которая собственно операционной системой[1] и является — с ядра. Все остальные, внеядерные ее компоненты в принципе нужны не всем и не всегда. Например, в случае применения Linux для рабочей станции набор компонент будет слегка отличаться от серверного применения и существенно отличаться от набора компонент в применениях для встраиваемых систем: маршрутизаторов, холодильников и мобильных телефонов.

Таким образом, первоочередной задачей при загрузке является размещение самой главной программы ОС — ядра в оперативной памяти и передача ей управления. Такую задачу всегда решает загрузчик операционной системы, который в принципе не является ее частью (т. к. зачастую умеет и используется для загрузки разных операционных систем), а скорее является частью аппаратной платформы, на которой выполняется ОС. Загрузчик очень сильно зависит от аппаратной платформы и услуг, которые она предоставляет, а также вынужден соответствовать ее спецификациям, таким как способы разделения носителей на разделы, собственное разме-

[1] В узком смысле, конечно, но тем не менее.

щение на этих носителях, порядок передачи ему управления при старте платформы и т. д.

Платформа персонального компьютера (она же W:[IBM PC compatible]), для которой изначально разрабатывался Linux, предоставляла W:[BIOS] — набор базовых услуг (в виде подпрограмм или же функций) для работы программ поверх «голого железа», например загрузчиков ОС. Согласно BIOS носители должны содержать таблицу разделов определенного вида в своем первом блоке (W:[boot sector]), так называемой основой загрузочной записи W:[MBR]. Именно подпрограмма, называемая «аппаратный загрузчик» W:[BIOS], получала управление первой (после подпрограмм инициализации и обнаружения и тестирования оборудования), затем считывала MBR, анализировала таблицу разделов, выбирала «активный» раздел, считывала его PBR в память (он же VBR, см. W:[volume boot record]) и передавала ему управление. Установленная на этот «активный» раздел операционная система обычно размещала свой загрузчик (в большинстве случаев всего лишь его малую часть) в PBR, который и выполнял дальнейшую загрузку себя (своих модулей, не поместившихся в PBR) и ядра операционной системы.

С развитием платформы W:[BIOS] превратился в W:[EFI] (позднее W:[UEFI]), MBR-таблица разделов превратилась в W:[GUID Partition Table], а процедура загрузки теперь предполагает наличие специального «системного раздела» ESP (см. W:[EFI System Partition]), содержащего файловую систему FAT (W:[File Allocation Table]), на которой в виде файлов размещаются загрузчики установленных операционных систем. Один из этих загрузчиков является для «аппаратного загрузчика» (он же EFI boot manager) «текущим» (current), по аналогии с «активным» разделом MBR, и именно ему при старте передается управление по умолчанию (листинг 10.1).

При любом типе загрузки, будь то W:[BIOS] или W:[UEFI], загрузчик, получив управление, умеет правильно считать в оперативную память ядро ОС и передать ему управление. Самым распространенным загрузчиком ядра Linux на платформе PC является загрузчик W:[GRUB], используемый в большинстве дистрибутивов.

Листинг 10.1. UEFI-загрузчики

```
morty@ubuntu:~$ efibootmgr
BootCurrent: 0003
Timeout: 1 seconds
BootOrder: 0003,0000,0004,0001,0002,0007,000D
Boot0000* Diskette Drive        BBS(Floppy,,0x0)
Boot0001* CD/DVD/CD-RW Drive    BBS(CDROM,,0x0)P1: TSSTcorp DVD+/-RW TS-U633J.
Boot0002* Internal HDD          BBS(HD,,0x0)P0: WDC WD2500BEKT-75PVMT0
```

```
Boot0003* ubuntu          HD(1,GPT,...,0x22,0x3d06f)/File(\EFI\ubuntu\grubx64.efi)
Boot0004* Onboard NIC     BBS(Network,,0x0)
Boot0007* USB Storage Device  BBS(USB,,0x0)USB Storage Device.
Boot000D* UEFI: INT13(,0x80)  PciRoot(0x0)/Pci(0x19,0x0)/VenHw(...)/HD(1,GPT,...,0x22,...)
```

В современных инсталляциях Linux этот ESP раздел смонтирован в каталог /boot/efi (см. листинг 10.2, в котором показан стартовый файл загрузчика GRUB).

Листинг 10.2. Загрузчики, установленные в ESP-раздел

```
morty@ubuntu:~$ sudo find /boot/efi -iname '*.efi'
/boot/efi/EFI/ubuntu/grubx64.efi
...                ...             ...             ...
/boot/efi/EFI/BOOT/BOOTX64.EFI
```

Остальные компоненты (модули *.mod и конфигурационный файл grub.cfg) загрузчика размещаются уже внутри операционной системы на корневой файловой системе (хотя для пущей надежности иногда для каталога /boot применяют и отдельную файловую систему) в каталоге /boot/grub, как показано в листинге 10.3.

Листинг 10.3. Компоненты загрузчика GRUB

```
morty@ubuntu:~$ find /boot/grub/
/boot/grub/
...          ...          ...          ...
/boot/grub/grub.cfg
...          ...          ...          ...
/boot/grub/x86_64-efi/lspci.mod
...          ...          ...          ...
/boot/grub/x86_64-efi/gfxterm_menu.mod
...          ...          ...          ...
```

Получив управление, загрузчик GRUB считывает свой конфигурационный файл и выполняет его директивы. В современных инсталляциях (в угоду пользователям) в силу директив конфигурационного файла загрузчика изображается красивая, зачастую анимированная заставка самого процесса загрузки, в то время как сам загрузчик производит загрузку ядра «по умолчанию», указанного теми же директивами. Сами ядра располагаются в каталоге /boot и обычно состоят из двух частей *(см. разд. 4.1.1)* — основной части vmlinuz (остова) и архива модулей initrd.img (листинг 10.4). Загрузчик на самом деле загружает обе части, а затем передает управление остову ядра вместе с параметрами ядра (см. **bootparam(7)**), которые потом будут использованы разными подсистемами ядра, модулями и драйверами.

Листинг 10.4. Установленные ядра Linux

```
morty@ubuntu:~$ ls /boot
...              ...                ...                ...
initrd.img                          System.map-5.3.0-24-generic
initrd.img-5.3.0-18-generic         vmlinuz
initrd.img-5.3.0-24-generic         vmlinuz-5.3.0-18-generic
initrd.img.old                      vmlinuz-5.3.0-24-generic
```

Ядро (если точнее, то его остов, т. е. vmlinuz), получив управление от загрузчика, должно выполнить два простейших, но очень важных действия: смонтировать корневую файловую систему и запустить первую программу, по традиции /sbin/init в процессе с PID = 1. В силу модульности ядра и того факта, что большинство драйверов устройств (включая дисковые накопители) являются его модулями, а остов не содержит ни одного модуля, смонтировать корневую файловую систему невозможно, т. к. на этом этапе загрузки ядру недоступны ни драйверы накопителей, где она размещается, ни сам драйвер файловой системы. Поэтому остов ядра распаковывает архив модулей initrd.img и «загружает» оттуда нужные модули — драйверы накопителей корневой файловой системы и самой файловой системы.

На самом деле все происходит немного не так. Образ «временной» корневой файловой системы initrd.img — это не просто пассивный «архив модулей» (см. W:[initial ramdisk]), а микроверсия операционной системы, содержащая набор утилит и компонент (листинг 10.5), достаточных для обнаружения дисковых устройств, загрузки их драйверов, инициализации (если нужно) программных массивов дисков, поиска «настоящей» корневой файловой системы, загрузки драйвера этой ФС, инициализации драйвера ее расшифровки (если зашифрована) и, наконец, ее монтирования. После монтирования образа «временной» корневой ФС ядро находит на ней и запускает «временный» /init, задача которого и состоит в исполнении всех вышеперечисленных действий, приводящих к монтированию «настоящей» корневой ФС, после чего «временный» /init подменяет себя настоящим /sbin/init с «настоящей» корневой ФС. Кроме всего, эта микроверсия ОС содержит командный интерпретатор и набор утилит (обычно на основе busybox, см. листинг 9.3) на случай аварийного останова на одном из этапов монтирования «настоящей» корневой ФС, перечисленных выше.

Листинг 10.5. Содержимое образа «временной» корневой ФС

```
morty@ubuntu:~$ lsinitramfs /boot/initrd.img-5.3.0-18-generic | grep .ko$ | wc -l
1122
morty@ubuntu:~$ lsinitramfs /boot/initrd.img-5.3.0-18-generic | grep bin/
...              ...                ...                ...
```

```
usr/bin/busybox
...          ...          ...          ...
usr/bin/sh
...          ...          ...          ...
usr/bin/modinfo
...          ...          ...          ...
usr/sbin/modprobe
...          ...          ...          ...
usr/sbin/rmmod
...          ...          ...          ...
usr/bin/udevadm
```

В листинге 10.5 показано, что для просмотра содержимого `initrd.img` используется утилита **lsinitramfs(8)**, тогда как для распаковки и запаковки образа предназначаются утилиты **mkinitramfs(8)** и **unmkinitramfs(8)**.

10.2. Как обнаруживаются драйверы устройств

Как уже упоминалось в *разд. 4.1.1*, большинство драйверов устройств выполнено в виде динамически загружаемых модулей ядра, которые могут пристыковываться к ядру и отстыковываться от него в любой момент времени исполнения. Однако открытым остается вопрос о том, каким образом нужные модули загружаются при обнаружении того или иного устройства.

Когда-то очень давно (когда динамически загружаемых драйверов еще не существовало) для добавления каждого нового устройства необходимо было пересоздавать новый бинарный образ ядра (из исходного кода), куда бы вошел нужный драйвер. Практиковался также вариант с бинарным образом ядра, поддерживающим все (ну или почти все) возможные драйверы, жертвуя размером бинарного образа и потреблением памяти. В те времена ядро при загрузке пробовало проинициализировать все включенные в него драйверы в надежде обнаружить соответствующие устройства. При добавлении новых устройств все равно приходилось перезагружать операционную систему, потому что для их добавления обычно обесточивали аппаратные средства.

С появлением динамически загружаемых модулей и драйверов стало возможным создать бинарный образ ядра (точнее, его остова) и все его модулей единожды, а затем загружать драйверы по мере необходимости. Необходимость в перезагрузке существенно уменьшилась с появлением все большего количества устройств с «горячим» подключением — на шинах PCMCIA, USB, SATA и даже PCI. Вместе с тем перестал себя оправдывать подход с загрузкой всех драйверов подряд, а с устройствами «горячего» подключения он вообще не работоспособен как с точки зрения здравого смысла, так и просто в силу количества модулей (❶ в листинге 10.6), являющихся драйверами устройств.

Листинг 10.6. Модули драйверов устройств

```
morty@ubuntu:~$ find /lib/modules/`uname -r`/kernel/drivers/ -name '*.ko' |  wc -l
❶ 4268

morty@ubuntu:~$ lspci -d ::0401 -k
                                    ①↓            ②↓
00:05.0 Multimedia audio controller: Intel Corporation 82801AA AC'97 Audio Controller (rev 01)
        Subsystem: Dell 82801AA AC'97 Audio Controller
        Kernel driver in use: snd_intel8x0
        Kernel modules: snd_intel8x0

morty@ubuntu:~$ lspci -d ::0401 -n
              ③↓   ④↓
❶ 00:05.0 0401: 8086:2415 (rev 01)

morty@ubuntu:~$ modinfo snd_intel8x0 | grep alias
...           ...              ...              ...
alias:          pci:v00008086d00002445sv*sd*bc*sc*i*
alias:          pci:v00008086d00002425sv*sd*bc*sc*i*
                        ③↓      ④↓
❷ alias:        pci:v00008086d00002415sv*sd*bc*sc*i*
...           ...              ...              ...
```

В реальности проблема имеет простое и элегантное решение, основанное на том, что практически все устройства на современных шинах идентифицируют себя как минимум при помощи идентификатора вендора устройства (vendor) и идентификатора модели устройства (device). Драйвера шины сканируют устройства и сообщают о событиях устройств (присоединении и отсоединении и пр.) всем желающим при помощи специального сокета «связи с ядром» **netlink(7)**, а кроме того, экспортируют информацию о текущем состоянии шин при помощи псевдофайловой системы **sysfs(5)**.

Утилиты **lspci(8)**, **lscsi(8)**, **lsusb(8)** и им подобные умеют считывать с **sysfs(5)** информацию (см. ❶ в листинге 10.6) об идентификаторах ③④ устройств, подключенных к шинам в текущий момент времени, а затем, используя таблицу расшифровки идентификаторов **/usr/share/misc/pci.ids**, показывают эту информацию в удобоваримом ①② виде. События устройств, присылаемые драйверами шин из ядра, отслеживаются специальной службой **udev(7)**, которая получает в каждом событии идентификаторы ③④ устройства, с которым это событие связано и, собственно, выполняет *загрузку* драйверов при подключении устройств и *выгрузку* при их отключении.

Для сопоставления *идентификаторов* устройств и *имен* модулей драйверов последние содержат в себе псевдонимы (alias) устройств, которые они поддерживают ❷.

Так, например, модуль snd_intel8x0 поддерживает много конкретных PCI-устройств от Intel (v00008086), включая AC 97 Audio Controller (d00002415), причем остальные идентификаторы устройств (sv, subvendor, sd — subdevice и пр.) не имеют значения (*).

Информация о псевдонимах извлекается из модулей драйверов при их инсталляции и помещается в файлы /lib/modules/`uname -r`/modules.alias и /lib/modules/`uname -r`/modules.alias.bin для дальнейшего использования утилитой modprobe(8) и службой udev(7).

В листинге 10.7 показан мониторинг событий, которые доставляются из ядра в udev(7) при подключении накопителя USB-флэш.

Листинг 10.7. Служба udev

```
morty@ubuntu:~$ udevadm monitor -k
monitor will print the received events for:
KERNEL - the kernel uevent

KERNEL[20373.565718] add      /devices/pci0000:00/0000:00:0b.0/usb1/1-1 (usb)
KERNEL[20373.572489] add      /devices/pci0000:00/0000:00:0b.0/usb1/1-1/1-1:1.0 (usb)
KERNEL[20373.573204] bind     /devices/pci0000:00/0000:00:0b.0/usb1/1-1 (usb)
KERNEL[20373.599217] add      /module/usb_storage (module)
...                    ...                    ...                    ...
KERNEL[20373.601776] add      /bus/usb/drivers/usb-storage (drivers)
KERNEL[20373.604020] add      /module/uas (module)
KERNEL[20373.604112] add      /bus/usb/drivers/uas (drivers)
...                    ...                    ...                    ...
morty@ubuntu:~$ mount -t sysfs
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)

morty@ubuntu:~$ cat /sys/devices/pci0000:00/0000:00:0b.0/usb1/1-1/1-1:1.0/uevent
DEVTYPE=usb_interface
DRIVER=usb-storage
PRODUCT=58f/6387/100
TYPE=0/0/0
INTERFACE=8/6/80
MODALIAS=usb:v058Fp6387d0100dc00dsc00dp00ic08isc06ip50in00

morty@ubuntu:~$ grep -i v058Fp6387 /lib/modules/`uname -r`/modules.*
/lib/modules/5.3.0-24-generic/modules.alias:alias usb:v058Fp6387d0141dc*dsc*dp*...    ① usb_storage
morty@ubuntu:~$ grep -i ic08isc06ip50 /lib/modules/`uname -r`/modules.*
/lib/modules/5.3.0-24-generic/modules.alias:alias usb:v*p*d*dc*dsc*dp*ic08isc06ip50in* uas ②
/lib/modules/5.3.0-24-generic/modules.alias:alias usb:v*p*d*dc*dsc*dp*ic08isc06ip50in* usb_storage
```

Нужно заметить, что абсолютно все сущности (как сами устройства, так и модули ядра) иерархически идентифицируются путевыми файлами в псевдофайловой системе sysfs(5), которая по соглашению монтируется в каталог /sys. Сами события «сохраняется» в файле uevent в каталоге этой сущности, отведенном ему на sysfs, а псевдоним устройства может определять сразу несколько драйверов ❸ ❹, подлежащих загрузке. Так, в приведенном примере по событию подключения ❶ udev(7) узнает псевдоним устройства и определяет модули, подлежащие загрузке ①②, т. е. драйвер самого устройства usb_storage (v, производитель 058F и p, продукт 6387) и драйвер uas, т. к. устройство поддерживает универсальный интерфейс 8/6/80 (класс, подкласс, протокол ic08isc06ip50), т. е. W:[USB Attached SCSI]. Успешная загрузка модулей драйверов тоже приводит к доставке событий ❶ и ❷ (подобных событиям физических устройств), которые используются udev(7) как сигнал для создания специальных файлов устройств в каталоге /dev.

Кроме «горячего» подключения устройств, проиллюстрированного в листинге 10.7, служба udev(7) ответственна также за их «холодное» подключение. Поскольку ядро стартует первым и задолго до самой службы udev(7)[1], то для устройств, обнаруженных на ранних стадиях инициализации ядра, будет некому загрузить из драйвера. «Холодное» подключение устройств и загрузка их драйверов начинаются при старте самой службы, которая обходит файловую систему sysfs в поисках uevent файлов и «проигрывает» их абсолютно так же, как если бы устройства подключались «горячим» образом.

10.3. Как запускаются системные службы

После того как ядро загружено, а корневая файловая система смонтирована, управление передается процессу init (PID = 1), в который помещается программа /sbin/init, выполняющая несколько функций. Первоочередная задача, стоящая перед init, — это обработка всех осиротевших процессов *(см. разд. 4.3)*, для которых он назначается приемным родителем. Вторая его немаловажная задача состоит в *инициализации* операционной системы и последующем *запуске* ее *служб*, которые в большинстве своем работают в классе процессов-демонов *(см. разд. 4.4)*. При инициализации ОС обычно выполняется несколько нехитрых подготовительных действий: монтирование всех оставшихся файловых систем в дерево каталогов (за исключением корневой, которая уже смонтирована ядром), конфигурирование сетевых интерфейсов и маршрутов и прочие сопутствующие действия, прежде чем можно будет запускать[2] службы.

---

[1] Честно говоря, ее облегченный аналог включен даже в образ «временной» корневой файловой системы initrd.img.

[2] Поскольку сетевым службам нужны проинициализированные сетевые интерфейсы, службы СУБД могут размещать свои базы данных на отдельных файловых системах и т. д.

Последняя задача, возложенная на init, — это управление остановом и перезагрузкой системы.

За время развития UNIX W:[init] претерпел несколько реинкарнаций. Классический и BSD init были устроены чрезвычайно просто и использовали большие («монолитные») сценарии на языке командного интерпретатора (так называемые rc[1]-сценарии), один из которых был «главным» (зачастую /sbin/rc или /etc/rc) и запускался самим init, а несколько «подчиненных» сценариев, например rc.local, запускались главным сценарием или друг другом. В AT&T-версиях init (сначала UNIX System III, а позднее и в UNIX System V) добавили понятие «уровней исполнения» системы, так называемых runlevels. При этом службы получили «модульные» сценарии запуска (индивидуальные для каждой службы), размещаемые в специальном каталоге /etc/init.d[2], а каталоги /etc/rc*N*.d, где *N* — номер уровня исполнения (0—6) стали определять набор сценариев служб, запускаемых на том или ином уровне исполнения, и порядок их *последовательного* запуска.

Ранние дистрибутивы Linux использовали BSD init и «монолитные» сценарии запуска служб, позднее практически везде распространились System V init и «модульные» сценарии, но при дальнейшем увеличении количества служб последовательный их запуск стал занимать большое время[3], хотя многие из них можно запускать параллельно. Это привело к созданию нескольких специфичных систем запуска со своей реализацией init и системы сценариев, например W:[upstart], W:[OpenRC] и пр. Все эти системы пытались так или иначе учесть зависимости (причинно-следственные связи) между самими сценариями запуска служб и выстраивать дерево зависимостей, позволяющее запускать службы параллельно настолько, насколько это возможно.

Передовой технологией запуска служб при старте операционной системы на текущий момент развития является система W:[systemd], исповедующая принцип интенсивной параллелизации запуска. Основная идея systemd(1) состоит в разделении работы на довольно мелкие единицы, так называемые юниты (*unit*, см. systemd.unit(5)), которые существенно мельче, чем ранние сценарии System V или upstart. Между юнитами, несомненно, существуют зависимости, но в целом, в силу использующегося подхода, гораздо больше действий можно выполнить параллельно. Например, в ситуации, когда одна служба при запуске обращается к другой

---

[1] Аббревиатура RC восходит к предшественнику UNIX, операционной системе W:[CTSS], в которой списки команд (предшественники сценариев) выполнялись программой-предшественником современных интерпретаторов, называемой RUNCOM (run commands).

[2] Этот рудимент можно наблюдать и поныне для тех служб, которые в полной мере не адаптированы к современному способу запуска в systemd(1) стиле.

[3] Большее, чем хотелось, и что самое главное — большее, чем можно было получить.

(при помощи сетевых сокетов) и отказывается работать, если обращение было неудачным. При использовании классических «модульных» сценариев приходилось сначала запускать первую, затем *дожидаться*[1], пока она откроет свой (серверный) сокет, и только потом запускать вторую. В этом случае **systemd** берет задачу по *одновременному* открытию сокетов служб на себя, а затем *одновременно* запускает обе службы, вручая каждой из них ее уже открытый сокет (в результате службы даже становятся проще). Даже если вторая служба немедленно обратится к сокету первой и сразу отправит запрос, то *ей* придется всего лишь *подождать*[2], пока первая не будет готова его обработать. Важно отметить: несмотря на то что службы в данном примере будут зависеть от открытости сокетов (первая служба зависит только от своего сокета, а вторая еще и от чужого), сами службы становятся *независимыми* друг от друга, что и позволяет запускать их одновременно.

Несмотря на то что в примере выше службы должны быть адаптированы к использованию такого **systemd**-подхода, он хорошо иллюстрирует разделение работы на мелкие единицы — в данном случае работа по открытию сокета (см. **systemd.socket(5)**) и работа по запуску (см. **systemd.exec(5)**) собственно службы (см. **systemd.service(5)**) разделены в угоду эффективности общего процесса. Кроме того, такое разделение еще и позволяет организовать так называемую сокет-активацию служб, когда они запускаются *только* при поступлении подключений на этот сокет, а в их отсутствие остановлены, тем самым экономя ресурсы операционной системы.

Еще одна иллюстрация эффективности разбиения работы по запуску операционной системы на небольшие единицы и их параллелизации может быть проведена относительно инициализации системы, проводимой перед стартом служб. Как уже упоминалось, типичными действиями на этом этапе являются формирование дерева каталогов путем монтирования туда файловых систем, упомянутых в **fstab(5)**, или конфигурирование сетевых интерфейсов. При классическом подходе все файловые системы монтировались в дерево последовательно, и если что-то шло не так, то инициализация останавливалась, приглашая администратора к ручному исправлению проблем посредством спасательного (rescue) командного интерпретатора. В **systemd** монтирование каждой файловой системы — это отдельный юнит (автоматически создаваемый чтением **fstab(5)**, см. **systemd.mount(5)** и **systemd-fstab-generator(8)**), а зависят от нее только те другие юниты, которым она на самом деле нужна, а не судьба инициализации целиком. Поскольку для монтирования файловых систем необходимо иметь созданные файлы устройств (и загруженные драйвера устройств), а эту работу асинхронно выполняет **udev(7)**, то **systemd** считает файлы устройств отдельными юнитами (см. **systemd.device(5)**), от которых, в свою очередь, зависят юниты

[1] Обычно путем циклического опроса, что достаточно неэффективно.

[2] При этом будут работать эффективные событийные механизмы ядра по работе с сокетами, а не циклический опрос.

файловых систем. В результате такой трактовки файловые системы (да и вся «инициализация») монтируются по мере доступности, а вся работа над юнитами происходит событийно и максимально параллельно.

В листинге 10.8 показано «дерево юнитов» активированных **systemd** при старте ОС и находящихся в выполняющемся (running) состоянии. Нужно заметить, что оно далеко не всегда повторяет дерево процессов, т. к. процессы зачастую теряют дочерне-родительские отношения при своей работе. Для группировки юнитов **systemd** использует иерархию групп управления ❶ *(см. разд. 9.4)*, позволяющую ему группировать процессы «по смыслу». Все процессы пользователей размещаются в группе управления `user.slice` ❶, а процессы системных служб — в группе `system.slice` ❶. Важно заметить, что сами `user.slice` и `system.slice` естественно являются юнитами (см. **systemd.slice(5)**), т. к. **systemd** манипулирует только юнитами и ничем, кроме юнитов, и под каждый из них создает свою группу управления.

Каждая проиллюстрированная[1] системная служба ❶ выполняется в своей индивидуальной группе управления, что естественным образом позволяет легко и удобно ограничить ее в ресурсах (изменив или дополнив конфигурацию **systemd**). Процессы интерактивных пользовательских сеансов ❶ помещаются в «динамически создаваемые» группы (см. **systemd.scope(5)**), в данном случае в так называемые сеансовые группы, чем заведует служба-компаньон **systemd-logind(8)**. В листинге 10.8 можно наблюдать группу процессов графического сеанса `session-7.scope` и группу процессов сеанса удаленного доступа SSH `session-25.scope`.

Листинг 10.8. Система systemd

```
morty@ubuntu:~$ systemctl status
● ubuntu
    State: running
     Jobs: 0 queued
   Failed: 0 units
    Since: Sun 2020-01-12 18:14:57 MSK; 1 day 6h ago
❶ CGroup: /
❶           ├─user.slice
            │ └─user-1000.slice
❶           │   ├─user@1000.service
            │ ① │ ├─gnome-terminal-server.service
            │   │ │ ├─3587 /usr/libexec/gnome-terminal-server
            │   │ │ └─3597 bash
            │   │ ├─init.scope
            │ ② │ │ ├─1750 /lib/systemd/systemd --user
            │   │ │ └─1754 (sd-pam)
                ...       ...
```

[1] Все эти службы описывались ранее, см. `containerd` и `dockerd` в *главе 9*, `NetworkManager` и `avahi-daemon` в *главе 6*, `gdm` в *главах 8* и *7*.

```
|  ① |  ├─gnome-shell-x11.service
|    |  |  ├─2227 /usr/bin/gnome-shell
|    |  |  ...
|    |  |  └─2525 /usr/lib/ibus/ibus-engine-simple
|  ② |  ├─dbus.service                    ...          ...          ...
|    |  |  ├─1787 /usr/bin/dbus-daemon --session --address=systemd: --nofork ...
|    |  |  ├─2205 /usr/lib/dconf/dconf-service
|  ③ |  |  ├─2252 /usr/lib/gnome-shell/gnome-shell-calendar-server
|  ③ |  |  ├─2295 /usr/lib/gnome-online-accounts/goa-daemon
|  ③ |  |  ├─2305 /usr/lib/gnome-online-accounts/goa-identity-service
|    |  |  └─2346 /usr/lib/ibus/ibus-portal
❺    |  ├─session-25.scope
|    |  |  ├─ 7810 sshd: rick [priv]
|    |  |  ├─ 7923 sshd: rick@pts/1
|    |  |  └─ 7926 -bash
❺    |  └─session-7.scope
|       ├─ 1992 gdm-session-worker [pam/gdm-password]
|       ├─ 2010 /usr/bin/gnome-keyring-daemon --daemonize --login
|       ├─ 2014 /usr/lib/gdm3/gdm-x-session ...
|       ├─ 2017 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth ...
|       ├─ 2024 /usr/lib/gnome-session/gnome-session-binary --systemd ...
|       ├─ 2094 /usr/bin/ssh-agent /usr/bin/im-launch ...
|       ├─16191 gdm-session-worker [pam/gdm-password]
|       └─16375 gdm-session-worker [pam/gdm-password]
├─init.scope
|  └─1 /sbin/init splash
❺ └─system.slice
                ...
❹    ├─containerd.service
|    |  └─709 /usr/bin/containerd
❹    ├─systemd-udevd.service
|    |  └─333 /lib/systemd/systemd-udevd
                ...          ...
❹    ├─docker.service
|    |  └─16520 /usr/bin/dockerd -H fd:// ...
                ...          ...
❹    ├─NetworkManager.service
|    |  └─586 /usr/sbin/NetworkManager --no-daemon
                ...          ...          ...
❹    ├─gdm.service
|    |  └─731 /usr/sbin/gdm3
                ...
❹    ├─avahi-daemon.service
|    |  ├─597 avahi-daemon: running [ubuntu-2.local]
|    |  └─650 avahi-daemon: chroot helper
     └─systemd-logind.service ...
        └─593 /lib/systemd/systemd-logind
```

Кроме управления инициализацией и запуском *служб операционной системы*, **systemd** позволяет в полной мере воспользоваться всеми своими возможностями в рамках отдельного пользователя ❺, т. е. управлять *пользовательскими службами*. Это свойство широко используется в современном настольном окружении GNOME ①②③. Управлением пользовательскими службами занимается отдельный «пользова-

тельский» экземпляр[1] **systemd** ⓪, запускаемый «системным» **systemd** каждому пользователю при его входе в систему.

Несмотря на то что рассмотрение задач администрирования **systemd** выходит за рамки данной книги, листинги 10.9–10.13 прекрасно иллюстрируют его возможности и назначение. На практике чрезвычайно часто возникает задача мгновенно «поделиться» файлами в «случайной» сети, например с участниками конференции, с партнерами на встрече или заказчиками на воркшопе и в прочих подобных ситуациях. Никогда не известно, какую операционную систему и на каких устройствах они (заказчики) используют — ноутбуки, планшеты, смартфоны, iOS, Android, Windows. Безотказно работает HTTP и MDNS *(см. главу 6)* в публичной Wi-Fi-сети, однако использование в этой ситуации полнофункционального HTTP-сервера, например **W:[Apache HTTP Server]** или **W:[nginx]**, абсолютно лишено всякого смысла. В целом задача не нова, и даже существует определенное количество решений, например **woof**[2] (**w**eb **o**ffer **o**ne **f**ile), но **systemd** позволяет решить эту задачу в два счета и на языке командного интерпретатора.

Листинг 10.9. Пользовательский Web-сервер на основе bash

```
morty@ubuntu:~$ cat ~/bin/peerweb
#!/bin/bash
documentroot=$1

❶ read method uri ver
❷ if [ -f $documentroot/$uri ]
then
        echo "HTTP/1.0 200 Ok"
❸       echo "Content-Type: `file -bi $documentroot/$uri`"
❹       echo "Content-Size: `stat -c %s $documentroot/$uri`"
        echo
❺       cat $documentroot/$uri
else
        echo "HTTP/1.0 404 Not found"
        echo
fi
```

[1] Звучит запутанно, но в понятиях **systemd** все так и есть. Пользовательский экземпляр **systemd**, управляющий пользовательскими службами, сам является системной службой с названием **user@.service**.

[2] См. http://www.home.unix-ag.org/simon/woof.

В качестве службы Web-сервера, который может обрабатывать только GET-запросы[1], будет выступать сценарий на языке командного интерпретатора (см. листинг 10.9). Он считывает три лексемы протокольного запроса ❶, проверяет наличие запрошенного файла, имя которого было передано во второй лексеме (uri) ❷, и возвращает его содержимое ❺, предваряя двумя протокольными заголовками о типе ❸ и размере ❹ файла. Сценарий предполагает, что запросы протокола поступают на стандартный поток ввода, а ответы отправляются на стандартный поток вывода, и ни с какими сетевыми сокетами работать не умеет. Всю работу по открытию и обслуживанию сетевых сокетов, а также запуску сценария, возьмет на себя **systemd**.

Для вручения **systemd** новых юнитов работы необходимо разместить конфигурационные файлы определенного содержания (**systemd.socket(5)** и **systemd.service(5)**) в предопределенных каталогах. В листинге 10.10 показано, как можно попросить об этом команду **systemctl(1)**, указав, что нужно изменять «пользовательскую» --user конфигурацию.

Листинг 10.10. Служба Web-сервера на основе systemd: юнит-файлы

```
morty@ubuntu:~$ EDITOR=nano systemctl --user edit --full peerweb.socket --force
[Socket]
❶ ListenStream=0.0.0.0:8080
❷ Accept=yes
morty@ubuntu:~$ EDITOR=nano systemctl --user edit --full peerweb@.service --force
[Service]
❺ ExecStart=/home/morty/bin/peerweb /home/morty/Public
❶ StandardInput=socket
```

Сокет-юнит peerweb.socket заставит **systemd** слушать ❶ потоковый (TCP) сетевой сокет на порту 8080 и принимать подключения ❷, после чего они будут переданы одноименной «шаблонной» службе peerweb@.service, которая и будет обслуживать каждое подключение. Для этого служба должна запускать ❸ сценарий ~/bin/peerweb и передавать ему в качестве параметра путь к каталогу ~/Public[2] и сокет принятого подключения в качестве стандартных потоков ❹ ввода-вывода. В результате HTTP-запрос будет отправлен на stdin сценария, а его stdout будет возвращен в сетевой сокет в качестве HTTP-ответа.

---

[1] https://www.youtube.com/watch?v=6ziuDdudUkI.

[2] Который по соглашению и так используется для публикации файлов, например, настольным окружением GNOME.

Листинг 10.11. Служба Web-сервера на основе systemd: запуск

```
❺ morty@ubuntu:~$ systemctl --user start peerweb.socket
❻ morty@ubuntu:~$ systemctl --user status peerweb.socket
   ● peerweb.socket
①    Loaded: loaded (/home/morty/.config/systemd/user/peerweb.socket; static; ...)
      Active: active (listening) since Tue 2020-01-14 03:28:47 MSK; 8s ago
      Listen: 0.0.0.0:8080 (Stream)
    Accepted: 0; Connected: 0;
      CGroup: /user.slice/user-1000.slice/user@1000.service/peerweb.socket

② янв 14 03:28:47 ubuntu systemd[18739]: Listening on peerweb.socket.
```

После запуска ❺ сокет-юнита в работу (листинг 10.11) можно узнать его состояние ❻ и расположение самих юнит-файлов ① и журнальные записи ② о последних событиях. Работу службы можно протестировать, разместив в каталоге ~/Public какой-либо файл и запросив[1] его получение при помощи Web-браузера или простейшего Web-клиента curl(1), как проиллюстрировано в листинге 10.12.

Листинг 10.12. Служба Web-сервера на основе systemd: использование

```
morty@ubuntu:~$ cat ~/Public/README
:)
morty@ubuntu:~$ curl -v http://ubuntu.local:8080/README
*   Trying 172.17.0.1:8080...
* TCP_NODELAY set
* Connected to ubuntu-2.local (172.17.0.1) port 8080 (#0)
> GET /README HTTP/1.1
> Host: ubuntu-2.local:8080
> User-Agent: curl/7.65.3
> Accept: */*
>
* Mark bundle as not supporting multiuse
* HTTP 1.0, assume close after body
< HTTP/1.0 200 Ok
< Content-Type: text/plain; charset=us-ascii
< Content-Size: 3
<
```

[1] Необходимо убедиться, что служба MDNS (см. avahi в *главе 6*) исправно функционирует, иначе придется воспользоваться IP-адресами вместо имен в домене .local.

```
:)
* Recv failure: Connection reset by peer
* Closing connection 0
curl: (56) Recv failure: Connection reset by peer
```

Листинг 10.13 иллюстрирует, что «шаблонные» сервис-юниты не имеют собственных состояний ❼, но порождают при запуске свои конкретные ❿ экземпляры, что отмечается ③ в журнале ❾ событий.

Листинг 10.13. Служба Web-сервера на основе systemd: диагностика

```
❼ morty@ubuntu:~$ systemctl --user status peerweb@.service
? Failed to get properties: Unit name peerweb@.service is neither a valid invocation ID nor unit name.
❾ morty@ubuntu:~$ journalctl --user -e
   ...                ...                ...
   янв 14 03:50:44 ubuntu systemd[18739]: Started 192.168.0.103:40930.    ¶
③ янв 14 03:50:44 ubuntu systemd[18739]: peerweb@27-172.17.0.1:8080-192.168.0.103:40930.service: Succeeded.
❿ morty@ubuntu:~$ systemctl --user status peerweb@*.service
● peerweb@27-172.17.0.1:8080-192.168.0.103:40930.service - 192.168.0.103:40930
   Loaded: loaded (/home/morty/.config/systemd/user/peerweb@.service; static; ...)
   Active: active (running) since Tue 2020-01-14 03:53:06 MSK; 10s ago
 Main PID: 20563 (peerweb)
   CGroup: /user.slice/user-1000.slice/user@1000.service
           /peerweb.slice/peerweb@27-172.17.0.1:8080-192.168.0.103:40930.service
           └─20563 /bin/bash /home/morty/bin/peerweb /home/morty/Public

янв 14 03:53:06 ubuntu systemd[18739]: Started 192.168.0.103:40930.
```

Остается только привязать ⓫ активацию сокет-юнита к входу пользователя в систему (листинг 10.14), иначе каждый раз придется активировать его вручную (см. ❺ в листинге 10.11).

Листинг 10.14. Встраивание юнита в дерево зависимостей

```
morty@ubuntu:~$ systemctl --user list-dependencies
default.target
● ├─pulseaudio.service
● ├─ubuntu-report.path
● └─basic.target
●   ├─paths.target
●   ├─sockets.target
●   │ ├─dbus.socket
●   │ ...
●   │ ├─pulseaudio.socket
●   │ └─snapd.session-agent.socket
●   └─timers.target
```

```
⓾ morty@ubuntu:~$ systemctl --user add-wants sockets.target peerweb.socket
  Created symlink /home/morty/.config/systemd/user/sockets.target.wants/peerweb.socket →
  /home/morty/.config/systemd/user/peerweb.socket.
  morty@ubuntu:~$ systemctl --user list-dependencies
  default.target
● └─basic.target
●   ├─sockets.target
●   │ ├─peerweb.socket
```

10.4. Linux своими руками

Теперь, когда в принципе становится ясно, какие компоненты составляют систему и участвуют в процессе ее запуска от включения питания и до получения доступа в ее командную строку, графический интерфейс или к ее сетевым сервисам, остается только один вопрос: где, каким образом и каким способом все эти компоненты системы размещаются, прежде чем они смогут начать последовательно загружаться и выполняться, выстраиваясь в систему?

Как правило, на эти вопросы отвечает программа-инсталлятор, которая, выполняясь в операционной системе, загруженной с одного (инсталляционного) носителя, например DVD-диска, устанавливает эту же операционную систему на другой (целевой) носитель, постоянный дисковый или SSD-накопитель. Вооружившись знаниями об ОС и набором нехитрых инструментов, эту же процедуру можно повторить и вручную, без специализированного инсталлятора. В примерах из листингов 10.15–10.22 проиллюстрирован процесс инсталляции Linux на внешний накопитель USB-флэш, который можно затем использовать для самых разнообразных целей: от «носимой ОС», которая вместе с любимым инструментарием «всегда с собой», до «спасательной флэшки» на черный день.

Первым делом необходимо определить специальный файл устройства накопителя (❶ в листинге 10.15), что можно сделать как командой **lsblk(8)**, так и **lsscsi(8)**, поскольку современные драйверы всех **W:[SATA]**, **W:[SAS]**, **W:[USB MSC]**- и **W:[UAS]**-накопителей работают поверх **SCSI**-подсистемы ядра. Следующим шагом ❷ создается таблица разделов ❸ формата **W:[GUID Partition Table]**, в которой определяются два раздела: один размером 100 Мбайт ❹ для загрузчика и второй ❺ на все оставшееся пространство для корневой файловой системы. Раздел для размещения загрузчика помечается ❻ согласно спецификации **W:[UEFI]** как **W:[ESP]**.

Листинг 10.15. Разбиение носителя на разделы

```
❶ morty@ubuntu:~$ lsscsi
  [0:0:0:0]    disk    ATA      WDC WD2500BEKT-7 1A01  /dev/sda
```

```
    [1:0:0:0]    cd/dvd  TSSTcorp DVD+-RW TS-U633J D600  /dev/sr0
    [4:0:0:0]    disk    JetFlash Transcend 2GB    8.07  /dev/sdc
❶ morty@ubuntu:~$ sudo parted /dev/sdc
    GNU Parted 3.2
    Using /dev/sdc
    Welcome to GNU Parted! Type 'help' to view a list of commands.
❸ (parted) mklabel gpt
    Warning: The existing disk label on /dev/sdc will be destroyed and all data on
    this disk will be lost. Do you want to continue?
    Yes/No? Yes
❹ (parted) mkpart loaders fat16 0 128m
    Warning: The resulting partition is not properly aligned for best performance.
    Ignore/Cancel? Ignore
❺ (parted) mkpart rootfs ext4 128m 100%
    Warning: The resulting partition is not properly aligned for best performance.
    Ignore/Cancel? Ignore
    (parted) print
    Model: JetFlash Transcend 2GB (scsi)
    Disk /dev/sdc: 2020MB
    Sector size (logical/physical): 512B/512B
    Partition Table: gpt
    Disk Flags:

    Number  Start   End     Size    File system  Name     Flags
     1      17,4kB  128MB   128MB   fat16        loaders
     2      128MB   2020MB  1892MB  ext4         rootfs
❻ (parted) set 1 esp on
    (parted) print
    Number  Start   End     Size    File system  Name     Flags
     1      17,4kB  128MB   128MB   fat16        loaders  boot, esp
     2      128MB   2020MB  1892MB  ext4         rootfs
    (parted) quit
    Information: You may need to update /etc/fstab.
```

После создания эти разделы «форматируются» (листинг 10.16), т. е. на них создаются файловые системы: ❶ FAT для ESP и ❷ EXT4 для корневой ФС. Нужно отметить, что конкретные экземпляры файловых систем в Linux идентифицируются или универсально-уникальными идентификаторами W:[UUID], или просто уникальным Volume ID. Эти идентификаторы позволяют найти файловую систему вне зависимо-

сти от порядковых номеров устройства и раздела, на котором она расположена, что крайне полезно при использовании на съемном накопителе, таком как USB-флэш.

Листинг 10.16. Форматирование разделов (создание файловых систем)

```
❶ morty@ubuntu:~$ sudo mkfs -t vfat -v /dev/sdc1
  mkfs.fat 4.1 (2017-01-24)
  ...          ...          ...          ...
  Volume ID is 08c8bdb9 ➥, no volume label.
❷ morty@ubuntu:~$ sudo mkfs -t ext4 /dev/sdc2
  mke2fs 1.45.3 (14-Jul-2019)
  Creating filesystem with 461801 4k blocks and 115680 inodes
  Filesystem UUID: 71002c88-0b02-4ac0-8cd4-65c4eb0e454b ➥
  Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912

  Allocating group tables: done
  Writing inode tables: done
  Creating journal (8192 blocks): done
  Writing superblocks and filesystem accounting information: done
```

Следующим логичным шагом является, собственно, помещение компонент ОС в корневую файловую систему, что можно сделать вручную, дублируя их программы, библиотеки, конфигурационные файлы и данные из исходной системы (например, как было показано в листинге 9.2). При необходимости выборочного копирования компонент ручной подход чрезвычайно трудоемок, поэтому на данном этапе используют разнообразные автоматизированные средства, например (уже знакомый по *главе 9*) **debootstrap(9)**, что проиллюстрировано ❶❷ в листинге 10.17.

Кроме инсталляции минимального набора компонент будущей системы ее необходимо минимально сконфигурировать, а именно — указать ❸ в таблице файловых систем **fstab(5)** то, какая ФС является «настоящей» корневой ①, т. е. должна монтироваться в корень дерева каталогов. Точно так же поступают и с файловой системой ② ESP-раздела, который по соглашению монтируют в каталог **/boot/efi**.

Листинг 10.17. Инсталляция минимальной системы

```
❶ morty@ubuntu:~$ sudo mount /dev/sdc2 /mnt
❷ morty@ubuntu:~$ sudo debootstrap --variant=minbase --include=lsb-release eoan /mnt
  I: Retrieving InRelease
  I: Checking Release signature
```

```
I: Valid Release signature (key id F6ECB3762474EDA9D21B7022871920D1991BC93C)
I: Retrieving Packages
I: Validating Packages
I: Resolving dependencies of required packages...
I: Resolving dependencies of base packages...
                ⑦  ⑦       ⑦       ⑦       ①       ①       ①       ①
I: Base system installed successfully.
morty@ubuntu:~$ sudo chroot /mnt
root@ubuntu:/# apt update
Hit:1 http://archive.ubuntu.com/ubuntu eoan InRelease
Get:2 http://archive.ubuntu.com/ubuntu eoan/main Translation-en [505 kB]
Fetched 505 kB in 2s (306 kB/s)
Reading package lists... Done
Building dependency tree... Done
All packages are up to date.
root@ubuntu:/# apt install nano
                ...
❶ root@ubuntu:/# nano /etc/fstab
① /dev/disk/by-uuid/71002c88-0b02-4ac0-8cd4-65c4eb0e454b  /          ext4   defaults 0 0
② /dev/disk/by-uuid/08C8-BDB9                             /boot/efi  vfat   defaults 0 0
❸ root@ubuntu:/# useradd -m -s /bin/bash morty
root@ubuntu:/# passwd morty
New password: plumbus
Retype new password: plumbus
passwd: password updated successfully
root@ubuntu:/# apt install sudo
                ...             ...
root@ubuntu:/# gpasswd -a morty sudo
Adding user morty to group sudo
root@ubuntu:/# exit
```

Необходимо отметить, что на разделы носителей, содержащие нужные файловые системы, лучше всегда ссылаться не при помощи специальных файлов устройств /dev/sdc1 и /dev/sdc2, а посредством их псевдонимов ① и ② т. е. их символических ссылок из каталога /dev/disk/by-uuid, которые там создает служба **udev(7)**. Так как заранее неизвестно, к какому USB-разъему (контроллеру, каналу) и в каком аппаратном окружении будет подключен этот накопитель для загрузки, и, как следствие, неизвестно, в каком порядке он будет опрошен и обнаружен, то и имя специального файла его устройства заранее определить невозможно. Несмотря на такую неопределенность, идентификаторы файловых систем останутся неизменными, и, как следствие, псевдонимы их разделов тоже.

Завершает минимальную конфигурацию ❸ процесс создания первой пользовательской учетной записи, назначения ей пароля и добавления ее в группу sudo для использования **sudo(1)** в устанавливаемой системе.

После того как корневая ФС наполнена требуемым содержимым, нужно припомнить, что согласно процессу загрузки *(см. разд. 10.1)* корневая файловая система монтируется в дерево каталогов после загрузки ядра, ядро загружается загрузчиком, а он, в свою очередь, получает управление от UEFI. Таким образом, необходимо инсталлировать загрузчик и ядро ОС, что проиллюстрировано в листингах 10.18 и 10.19 соответственно.

Инсталляция загрузчика начинается с формирования дерева каталогов инсталлируемой ОС так, как оно будет сформировано в процессе ее работы. Для этого ниже точки монтирования ее корневой ФС (каталог /mnt, см. ❶ в листинге 10.17) монтируется ESP-раздел ❶ в /mnt/boot/efi, а затем псевдофайловые системы ❶ proc, sysfs и специальные файлы устройств /dev. При этом они попросту «заимствуются» из исходной ОС путем «связующего» --bind монтирования. Другими словами, каталоги /sys, /proc и /dev просто отображаются в еще одно место дерева, а именно ниже каталога /mnt. Для инсталляции загрузчика осуществляется чрутизация *(см. разд. 9.1)* в сформированное окружение, где сначала устанавливается ❸ пакет с компонентами загрузчика в систему, а затем сам загрузчик ❹ на ESP-раздел.

Листинг 10.18. Инсталляция загрузчика GRUB

```
   morty@ubuntu:~$ sudo mkdir /mnt/boot/efi
❶  morty@ubuntu:~$ sudo mount /dev/sdc1 /mnt/boot/efi
❷  morty@ubuntu:~$ sudo mount --bind /dev /mnt/dev
❷  morty@ubuntu:~$ sudo mount --bind /proc /mnt/proc
❷  morty@ubuntu:~$ sudo mount --bind /sys /mnt/sys
   morty@ubuntu:~$ sudo chroot /mnt
❸  root@ubuntu:/# apt install grub-efi
                              ...             ...             ...
❹  root@ubuntu:/# grub-install --removable --no-uefi-secure-boot -v
                              ...             ...             ...
   Installing for x86_64-efi platform.
                              ...             ...             ...             ...
①  grub-install: info: copying `/.../grub/x86_64-efi/ext2.mod' -> `/boot/grub/x86_64-efi/ext2.mod'.
①  grub-install: info: copying `/.../grub/x86_64-efi/part_gpt.mod' -> `/boot/grub/x86_64-efi/part_gpt.mod'.
                              ...             ...             ...             ...
①  grub-install: info: copying `/.../grub/x86_64-efi/efi_gop.mod' -> `/boot/grub/x86_64-efi/efi_gop.mod'.
①  grub-install: info: copying `/.../grub/x86_64-efi/font.mod' -> `/boot/grub/x86_64-efi/font.mod'.
                              ...             ...             ...             ...
①  grub-install: info: copying `/usr/share/grub/unicode.pf2' -> `/boot/grub/fonts/unicode.pf2'.
                              ...             ...             ...             ...
                              ...             ...             ...             ...
②  grub-install: info: grub-mkimage --directory '/usr/lib/grub/x86_64-efi' --prefix '(,gpt2)/boot/grub'
   --output '/boot/grub/x86_64-efi/core.efi'  --dtb '' --format 'x86_64-efi' --compression 'auto'  'ext2' 'part_gpt'
                              ...             ...             ...             ...
②  grub-install: info: reading /usr/lib/grub/x86_64-efi/kernel.img.
②  grub-install: info: reading /usr/lib/grub/x86_64-efi/fshelp.mod.
②  grub-install: info: reading /usr/lib/grub/x86_64-efi/ext2.mod.
```

```
② grub-install: info: reading /usr/lib/grub/x86_64-efi/part_gpt.mod.
② grub-install: info: kernel_img=0x5636d82a7000, kernel_size=0x1a000.
② grub-install: info: the core size is 0x1e280.
② grub-install: info: writing 0x21000 bytes.
                ...          ...          ...          ...
③ grub-install: info: copying `/boot/grub/x86_64-efi/core.efi' -> `/boot/efi/EFI/BOOT/BOOTX64.EFI'.
                ...          ...          ...          ...
   Installation finished. No error reported.
```

Загрузчик W:[GRUB] сам по себе имеет модульное устройство, поэтому при его инсталляции сначала в каталог /boot/grub копируются ① его модули *.mod (и другие компоненты, например шрифты .pf2) на *корневую* ФС, а затем формируется стартовый файл core.efi путем компоновки нескольких модулей ②, а затем он «правильным образом» помещается[1] на ESP-раздел ③. Таким образом, получается, что при загрузке стартового файла с ESP он содержит только минимально необходимые модули (драйвер ФС ext2.mod и таблицы разделов gpt_part.mod), но позволяющие динамически догружать любые другие модули с *корневой* ФС. Нужно отметить, что такую (классическую) схему инсталлятор загрузчика выбрал потому, что его опции указывают, что инсталляция ведется на «съемное» --removable устройство и не требуется соответствовать спецификациям «безопасной загрузки» --no-uefi-secure-boot, когда проверяются цифровые подписи загружаемых компонент (и компоновка неподписанного core.efi на лету уже не сработает). Для «стационарных» устройств используются похожие, но немного другие схемы, что можно наблюдать в листингах 10.1 и 10.2.

Процедура инсталляции ядра, показанная в листинге 10.19, намного проще и обычно ограничивается инсталляцией пакета ❶ ядра. Этот пакет, как правило, зависит от необходимых утилит ❶ работы с образами «временной» корневой ФС (см. листинг 10.5), модулей ядра ❷ и службы ❸ udev, которая начинает работу на самых ранних этапах загрузки, еще при поиске и монтировании «настоящей» корневой ФС.

Листинг 10.19. Инсталляция ядра

```
❶ root@ubuntu:/# apt install linux-image-5.3.0-18-generic
   Reading package lists... Done
   Building dependency tree
   Reading state information... Done
                ...
```

[1] Под именем BOOTX64.EFI, которое по умолчанию ищет «аппаратный» UEFI-загрузчик на подключенных накопителях, если ему вообще разрешено с них загружаться.

```
The following NEW packages will be installed:
 busybox-initramfs cpio initramfs-tools ❶ initramfs-tools-bin
 initramfs-tools-core klibc-utils kmod libklibc
 libkmod2 linux-base linux-image-5.3.0-18-generic  linux-modules-5.3.0-18-generic ❷
 lz4 udev ❸
0 upgraded, 14 newly installed, 0 to remove and 0 not upgraded.
Need to get 24.6 MB of archives.
After this operation, 93.9 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
                    ...          ...          ...
Setting up linux-image-5.3.0-18-generic (5.3.0-18.19+1) ...
                    ...          ...          ...
Processing triggers for initramfs-tools (0.133ubuntu10) ...
❹ update-initramfs: Generating /boot/initrd.img-5.3.0-18-generic
root@ubuntu:/# ls /boot/*5.3.0*
/boot/System.map-5.3.0-18-generic  /boot/initrd.img-5.3.0-18-generic ①
/boot/config-5.3.0-18-generic      /boot/vmlinuz-5.3.0-18-generic ②
```

Нужно отметить, что образ «временной» корневой ФС, в который включаются модули ядра, необходимые для ее монтирования, создается «на лету» ❹ сразу при его установке. Остается только сообщить об установленном ядре загрузчику GRUB и указать параметры **bootparam(7)**, которые должны сообщаться при передаче ему управления, что проиллюстрировано в листинге 10.20.

Листинг 10.20. Конфигурирование загрузчика

```
root@ubuntu:/# nano /boot/grub/grub.conf
❶ insmod efi_uga
  insmod efi_gop
❷ set gfxpayload=auto

❸ menuentry 'My Flash Linux' {
    echo Loading /boot/vmlinuz-5.3.0-18-generic...          ↓ ①                    ② ↓
❹   linux /boot/vmlinuz-5.3.0-18-generic root=/dev/disk/by-uuid/e74422dd-b74a-40bf-a6c4-e1abd516c570 ro

    echo Loading /boot/initrd.img-5.3.0-18-generic...
❺   initrd /boot/initrd.img-5.3.0-18-generic

    echo Booting.
❻   boot
  }
```

Конфигурационный файл загрузчика является сценарием на его командном языке. При отсутствии такового загрузчик перейдет при старте в режим интерактивного взаимодействия с пользователем, выведет приглашение к вводу команд grub> и бу-

дет выполнять вводимые команды. Тремя самыми важными командами являются linux ❶, initrd ❷ и boot ❸. Первая команда загружает в память ядро и параметры ①②, вторая — образ «временной» корневой ФС, а третья передает ядру управление. Команда insmod ❹, как нетрудно догадаться, загружает указанный модуль GRUB, команда set ❺ устанавливает значение переменной окружения GRUB, а команда echo вообще не нуждается в пояснениях. Кроме того, директива menuentry ❻ заставляет GRUB изобразить простейшее меню с таким количеством пунктов, сколько таких директив было найдено, а при выборе одного из них выполнить команды из соответствующей секции {...}.

Приведенный в листинге 10.20 конфигурационный файл указывает GRUB загрузить модули ❹, реализующие спецификации EFI UGA (Universal Graphic Adapter) и UEFI GOP (Graphics Output Protocol), которые организуют доступ к видеоадаптеру, а переменная окружения gfxpayload сообщает ❺ загрузчику в формате WxH, какой видеорежим нужно выбрать для передачи его параметров в ядро. Так как загрузка со «съемного» накопителя может быть в принципе производиться в любом аппаратном окружении, то список поддерживаемых видеорежимов заранее неизвестен, и используется специальное значение auto.

Самым последним этапом при загрузке операционной системы является запуск процесса init, на который возложена обязанность запуска системных служб, в том числе служб терминального, графического и сетевого входа в операционную систему. Для этого остается только добавить пакет systemd на корневую ФС (листинг 10.21), и все будет готово для запуска.

Листинг 10.21. Установка /sbin/init (systemd)

```
root@ubuntu:/# apt install systemd
                    ...                ...
root@ubuntu:/# apt clean
root@ubuntu:/# df -h /
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdc2       1.8G  365M 1.3G  23% /
root@ubuntu:/# exit
morty@ubuntu:~$ sudo umount -R /mnt
```

Тестирование полученной инсталляции можно производить на «голом железе», т. е. попробовать выполнить перезагрузку и загрузить инсталлированную ОС с USB-флэш, но это не очень удобно, т. к. если что-то «пошло не так», то потребуется повторная загрузка исходной ОС для исправления. Куда удобнее использовать виртуальные машины с пара- или полной виртуализацией, например W:[QEMU] или W:[Virtual PC]. В листинге 10.22 показано, как, используя QEMU, можно запустить

«виртуальный PC» с подключенным к нему накопителем USB-флэш. При этом для паравиртуализации используется W:[гипервизор] W:[KVM] (можно и без него, но будет медленнее), а в качестве UEFI-прошивки используется OVFM[1] (Open Virtual Machine Firmware).

Листинг 10.22 Тестирование инсталляции

```
morty@ubuntu:~$ ls -l /dev/sdc*
brw-rw---- 1 root disk 8, 32 янв 17 01:14 /dev/sdc
brw-rw---- 1 root disk    8, 33 янв 17 01:14 /dev/sdc1
brw-rw---- 1 root disk    8, 34 янв 17 01:14 /dev/sdc2
❶ morty@ubuntu:~$ sudo gpasswd -a morty disk
morty@ubuntu:~$ ls -la /dev/kvm
crw-rw---- 1 root kvm 10, 232 янв 18 01:46 /dev/kvm
❷ morty@ubuntu:~$ sudo gpasswd -a morty kvm
morty@ubuntu:~$ logout
                ...                ...
morty@ubuntu:~$ id
uid=1001(morty) gid=1001(morty) группы=1001(morty),...,6(disk),...,135(kvm),...
morty@ubuntu:~$ qemu-system-x86_64 -enable-kvm -m 512 \
> -usbdevice disk:format=raw:/dev/sdc -bios OVMF.fd
```

10.5. В заключение

Искренне надеюсь, что рассмотрение процессов, происходящих в операционной системе Linux от включения питания и до входа в ее графический интерфейс, добавит пониманию ее внутреннего устройства некоторого рода «завершенность» и полноту. Вместе с тем нужно признаться, еще много интересного осталось за кадром и в силу тех или иных причин выходит за рамки данной книги «для начинающих».

Одна из целей книги, которая ставилась при ее написании, — дать начинающим толчок в правильном направлении, научить самостоятельно докапываться до сути вещей, преумножать и укреплять свои знания, опираясь на излюбленные мной способы и приемы, проиллюстрированные здесь.

Буду польщен, если этот процесс доставит вам такое же искреннее удовольствие, как и мне.

Д. К.

[1] Требуется установка одноименного пакета ovmf.

Заключение

Изначально содержимое этой книги задумывалось как практическая иллюстрация внутреннего устройства различных компонент операционной системы, в равной степени подкрепляемая примерами их анализа и синтеза.

Аналитическая часть удалась примерно в той степени, в которой и задумывалась, а примеры синтеза выходили достаточно блеклыми, особенно по сравнению с теми, которые уже представлены в соответствующих изданиях, целиком посвященных программному окружению и разработке. Более того, сделать эти примеры доступными для их успешного понимания читателем, не владеющим базовыми знаниями по программированию на языке C (Си), не удалось вовсе.

Поэтому все эти не столь многочисленные примеры перекочевали в электронное приложение к книге, которое в виде архива доступно на сайте издательства **www.bhv.ru** со страницы книги или по ссылке:

ftp://ftp.bhv.ru/9785977566308.zip

Большинство примеров являются простейшими и достаточно грубыми моделями программ, уже существующих в операционной системе.

Так, например, работу с файлами и ссылками иллюстрируют модели программ **touch(1)**, **ln(1)**, **rm(1)**, **mv(1)**, **cp(1)**, а работу со специальными файлами устройств — модель программы **eject(1)**, открывающая лоток привода CD/DVD, и специальная программа **mouse**, обрабатывающая перемещения манипулятора «мышь». Управление процессами продемонстрировано при помощи программы **shell**, моделирующей простейший командный интерпретатор, а управление нитями представляет программа **pdu**, параллельная модель измерителя файлов в дереве каталогов **du(1)**. Сетевое взаимодействие иллюстрируется простейшими TCP-клиентом, моделирующим **telnet(1)** (или **netcat(1)**), и TCP-сервером, моделирующим сервер службы удаленного доступа, похожей на **telnetd(8)** (или **sshd(8)**, только без криптозащиты).

Завершаются иллюстрации четырьмя вариантами простейшей GUI-программы **W:[Hello,_world!]**, выполненной на языках C, Python и Tcl и являющейся X-клиентом оконной системы X Window, использующим ее графические библиотеки интерфейсных элементов Qt, Gtk+ и Tk.

Кроме собственно иллюстрационных программ, архив содержит вспомогательные сценарии на языке командного интерпретатора, формирующие «программного гида», который поможет откомпилировать и запустить эти программы и вдобавок прокомментирует результат их выполнения в тех частях, где анализ результата не будет тривиальным. Сами сценарии «программного гида» тоже являются живыми примерами практик и приемов программирования на языке командного интерпретатора, анализируя которые можно расширить восприятие материала соответствующей главы, посвященной этой теме.

Как и ожидалось с самого начала, эта книга является лишь иллюстративным введением во внутреннее устройство ОС Linux, оставляющим за кадром массу специфичных деталей реализации конкретных версий ее ядер или ее дистрибутивов. Выстроив «правильную» начальную ментальную модель операционной системы, все эти тонкости можно без значительных усилий почерпнуть из собственной практики, документации, интернет-общения или других книг (см. список литературы).

Список литературы

Для удовольствия

Торвальдс Л., Даймонд Д. Just for Fun. Рассказ нечаянного революционера: Пер. с англ. — М.: Эксмо-Пресс, 2002. 288 с. — ISBN 5-04-009285-7.

Начинающим

Граннеман С. Linux. Карманный справочник: Пер. с англ. — М.: Вильямс, 2015. 416 с. — ISBN 978-5-8459-1956-4.

Пайк Р., Керниган Б. UNIX. Программное окружение: Пер. с англ. — СПб.: Символ-плюс, 2003. 416 с. — ISBN 5-93286-029-4.

Реймонд Э. Искусство программирования для UNIX: Пер. с англ. — М.: Вильямс, 2016. 544 с. — ISBN 978-5-8459-2064-5.

Тейнсли Д. Linux и UNIX: программирование в shell. Руководство разработчика: Пер. с англ. — СПб.: БХВ-Петербург, 2001. 464 с. — ISBN 5-7315-0114-9.

Фридл Дж. Регулярные выражения: Пер. с англ. — СПб.: Символ-плюс, 2008. 608 с. — ISBN 5-93286-121-5.

Dougherty D., Robbins A. sed & awk. — O'Reilly Media, 1997. 432 p. — ISBN: 1-56592-225-5.

Программистам

Керриск М. Linux API. Исчерпывающее руководство: Пер. с англ. — СПб.: Питер, 2019. 1248 с. — ISBN: 978-5-4461-0985-2.

Лав Р. Linux. Системное программирование: Пер. с англ. — СПб.: Питер, 2016. 448 с. — ISBN 978-5-496-01684-1.

Роббинс А. Linux: программирование в примерах: Пер. с англ. — СПб.: КУДИЦ-Пресс, 2008. 656 с. — ISBN:978-5-91136-056-6.

Стивенс У. Р., Рago С. UNIX. Профессиональное программирование: Пер. с англ. — СПб.: Символ-плюс, 2014. 1104 с. — ISBN 978-5-93286-216-2.

Бесстрашным

Бовет Д., Чезати М. Ядро Linux: Пер. с англ. — СПб.: БХВ-Петербург, 2007. 1104 с. — ISBN 0-596-00565-2, 978-5-94157-957-0.

Лав Р. Ядро Linux. Описание процесса разработки: Пер. с англ. — М.: Вильямс, 2014. 496 с. — ISBN 978-5-8459-1944-1.

Kroah-Hartman G. Linux Kernel in a Nutshell. — O'Reilly Media, 2006. 202 p. — ISBN 978-0-596-10079-7.

McKellar J., Rubini A., Corbet J., Kroah-Hartman G. Linux Device Drivers, 3rd Edition. — O'Reilly Media, 2005. 640 p. — ISBN 978-0-596-00590-0.

Предметный указатель

Н

О

П

Р

С

Т

У

Ф

Ц

Ш

Э

Я

www.bhv.ru

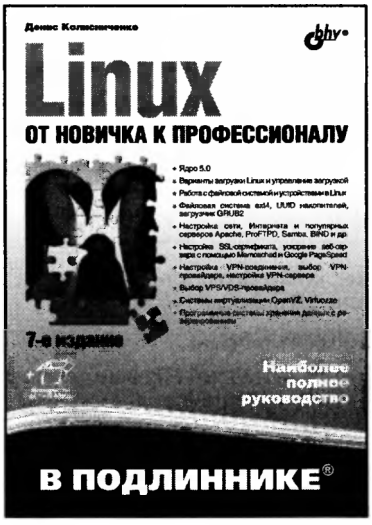
Колисниченко Д.

Linux. От новичка к профессионалу

7-е изд.

Отдел оптовых поставок:
e-mail: opt@bhv.ru

Гарантия эффективной работы в Linux



- Ядро 5.0
- Варианты загрузки Linux и управление загрузкой
- Работа с файловой системой и устройствами в Linux
- Файловая система ext4, UUID накопителей, загрузчик GRUB2
- Настройка сети, Интернета и популярных серверов Apache, ProFTPD, Samba, BIND и др.
- Настройка SSL-сертификата, ускорение веб-сервера с помощью Memcached и Google PageSpeed
- Настройка VPN-соединения, выбор VPN-провайдера, настройка VPN-сервера
- Выбор VPS/VDS-провайдера
- Системы виртуализации OpenVZ, Virtuozzo
- Программные системы хранения данных с резервированием

Книга предназначена для широкого круга пользователей Linux и поможет им самостоятельно настроить и оптимизировать эту операционную систему. Даны ответы на все вопросы, возникающие при работе с Linux: от установки и настройки этой ОС до настройки сервера на базе Linux. Материал книги максимально охватывает все сферы применения Linux от запуска Windows-игр под управлением Linux до настройки собственного Web-сервера. Материал ориентирован на последние версии дистрибутивов Fedora, openSUSE, Slackware, Ubuntu. В седьмом издании книги много внимания уделяется веб-серверам; в частности, добавлены описание настройки SSL-сертификата и рекомендации по ускорению работы с помощью Google-сервиса PageSpeed и системы кэширования данных Memcached.

Колисниченко Денис Николаевич, инженер-программист, системный администратор и IT-консультант. Имеет богатый опыт эксплуатации и создания локальных сетей от домашних до уровня предприятия на базе операционной системы Linux. Автор более 70 книг компьютерной тематики, в том числе «Самоучитель системного администратора», «PHP и MySQL. Разработка веб-приложений», «Самоучитель Linux», «Самоучитель Microsoft Windows 10», «Планшет и смартфон на базе Android для ваших родителей», «Самоучитель системного администратора Linux» и др.

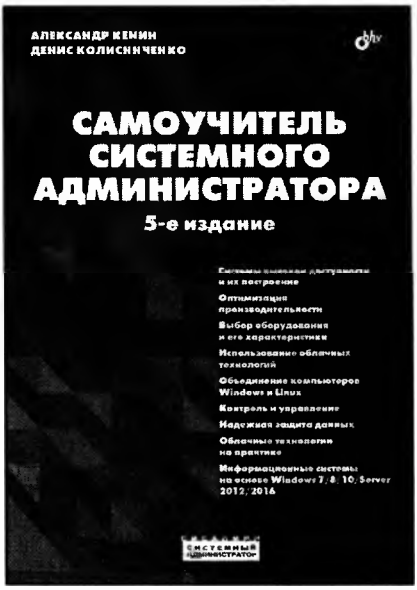
www.bhv.ru

Кенин А., Колисниченко Д.

Самоучитель системного администратора, 5-е изд.

Отдел оптовых поставок:
e-mail: opt@bhv.ru

Настольная книга администратора



- Системы высокой доступности и их построение
- Оптимизация производительности
- Выбор оборудования и его характеристики
- Использование облачных технологий
- Объединение компьютеров Windows и Linux
- Контроль и управление
- Надежная защита данных
- Облачные технологии на практике
- Информационные системы на основе Windows 7/8/10/Server 2012/2016

Есть области человеческой деятельности, где фактор личного опыта и мастерства играет решающую роль. Как ни парадоксально, вышедшее из самых точных наук компьютерное дело относится к таковым. Дружественность интерфейсов часто бывает столь же обманчива, как дружелюбие страховых агентов, а инструкции для сисадминов написаны в расчете на профессионала, ответы так называемых специалистов еще более неясны.

К счастью, исключения есть. Авторы этой книги охотно делятся своим опытом, консультируют, объясняют, обучают. Эти специалисты не только знают свое дело, но и умеют рассказать о нем легко и доходчиво. Книга, которую вы держите в своих руках, — яркое тому подтверждение.

Кенин Александр Михайлович, автор более 10 книг компьютерной тематики, вышедших общим тиражом более 500 000 экз., делится с читателями своим двадцатилетним опытом проектирования и управления информационными системами.

Колисниченко Денис Николаевич, инженер-программист и системный администратор. Имеет богатый опыт эксплуатации и создания локальных сетей от домашних до уровня предприятия. Автор более 50 книг компьютерной тематики, в том числе по операционным системам Linux и Microsoft Windows 7/8/10.

www.bhv.ru

Шлее М.
Qt 5.10. Профессиональное программирование на C++

Отдел оптовых поставок:
e-mail: opt@bhv.ru

Платформно-независимая реализация приложений —
это уже сегодняшний и завтрашний день программной индустрии.
И эта книга станет вашим путеводителем в будущее.

- Кроссплатформенная реализация приложений для Windows, Mac OS X и Linux
- Разработка мобильных приложений для Android и iOS
- Программирование 2D- и 3D-графики, мультимедиа, веб-приложений, баз данных, сети, таймера, многопоточности, XML, QML и JavaScript
- 240 завершенных программ

Книга подробно знакомит с библиотекой Qt 5.10, являющейся не только средством для создания пользовательских интерфейсов, но и позволяющей разрабатывать приложения практически любой сложности. Недаром Qt широко используется многими организациями и компаниями, такими как ADOBE, AMAZON, AMD, BOSCH, BLACKBERRY, CANNON, CISCO SYSTEMS, DISNEY, INTEL, IBM, PANASONIC, PIONEER, PHILIPS, ORACLE, HP, GOOBER, GOOGLE, NASA, NEC, NEONWAY, NOKIA, SAMSUNG, SIEMENS, SONY, XEROX, XILINX, YAMAHA и др.
Если вы хотите идти в ногу со временем, то вам без этой книги просто не обойтись, поскольку она является исчерпывающим пособием по написанию Qt 5-программ на C++ и QML.

Макс Шлее (Max Schlee) — закончил Университет прикладных наук в городе Кайзерлаутерн (Германия). Сооснователь компании NEONWAY по разработке программного обеспечения на Qt. Работал разработчиком программного обеспечения в фирмах THOMSON, Grass Valley, DigitalFilmTechnology Weiterstadt, Goober Ltd. и Advancis. Эксперт в области объектно-ориентированного проектирования, специализирующийся на C++ и Qt. Создатель более 40 программ и приложений для Windows, Mac OS X, iPhone, iPad и Android. Увлеченно занимается проектами в области программирования графики, звука и анализа финансовых рынков. Является автором ряда статей, научных докладов на международных конференциях по генеративному программированию пользовательского интерфейса. Автор книг «Qt 4/4.5/4.8. Профессиональное программирование на C++» и др. Связаться с автором можно по адресу электронной почты **Max.Schlee@neonway.com** или через его персональный блог **www.maxschlee.com**.

www.bhv.ru

Прохоренок Н.

Основы Java
2-е изд.

Отдел оптовых поставок
E-mail: opt@bhv.ru

Просто о сложном

- Базовый синтаксис языка Java
- Объектно-ориентированное программирование
- Работа с файлами и каталогами
- Stream API
- Функциональные интерфейсы
- Лямбда-выражения
- Работа с базой данных MySQL
- Получение данных из Интернета
- Интерактивная оболочка JShell

Если вы хотите научиться программировать на языке Java, то эта книга для вас. В книге описан базовый синтаксис языка Java: типы данных, операторы, условия, циклы, регулярные выражения, лямбда-выражения, ссылки на методы, объектно-ориентированное программирование. Рассмотрены основные классы стандартной библиотеки, получение данных из сети Интернет, работа с базой данных MySQL. Во втором издании приводится описание большинства нововведений: модули, интерактивная оболочка JShell, инструкция var и др.

Книга содержит большое количество практических примеров, помогающих начать программировать на языке Java самостоятельно. Весь материал тщательно подобран, хорошо структурирован и компактно изложен, что позволяет использовать книгу как удобный справочник.

Прохоренок Николай Анатольевич, профессиональный программист, автор книг «HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера», «Python 3. Самое необходимое», «Python 3 и PyQt 5. Разработка приложений», «OpenCV и Java. Обработка изображений и компьютерное зрение» и др.

ВНУТРЕННЕЕ УСТРОЙСТВО LINUX

- Пользовательское окружение и интерфейс командной строки CLI
- Файлы, каталоги и файловые системы
- Дискреционное, мандатное разграничение доступа и привилегии
- Процессы и нити
- Виртуальная память и отображаемые файлы
- Каналы, сокеты и разделяемая память
- Сетевая подсистема и служба SSH
- Графический интерфейс GUI: оконные системы X Window и Wayland
- Программирование на языке командного интерпретатора
- Контейнеры и виртуализация
- Linux своими руками

Книга, которую вы держите в руках, адресована студентам, начинающим пользователям, программистам и системным администраторам операционной системы Linux. Она представляет собой введение во внутреннее устройство Linux — от ядра до сетевых служб и от утилит командной строки до графического интерфейса.

Все части операционной системы рассматриваются в контексте типичных задач, решаемых на практике, и поясняются при помощи соответствующего инструментария пользователя, администратора и разработчика.

Все положения наглядно проиллюстрированы примерами, разработанными и проверенными автором с целью привить читателю навыки самостоятельного исследования постоянно эволюционирующей операционной системы Linux.

Кетов Дмитрий Владимирович, инженер в Санкт-Петербургском исследовательском центре LG Russia R&D Lab. Профессионально занимается теорией построения и практикой разработки операционных систем и системного программного обеспечения. Имеет многолетний опыт преподавания в Санкт-Петербургском политехническом университете (СПбПУ) в области операционных систем и сетевых технологий.

ISBN 978-5-9775-6630-8

191036, Санкт-Петербург,
Гончарная ул., 20
Тел.: (812) 717-10-50,
339-54-17, 339-54-28
E-mail: mail@bhv.ru
Internet: www.bhv.ru

bhv®