3 Устройства

В этой главе представлены основные сведения об инфраструктуре устройств, которая обеспечивается ядром в работающей системе Linux. На протяжении истории этой системы многое изменилось в том, как ядро представляет устройства пользователю. Мы начнем с рассмотрения традиционной системы файлов устройств, чтобы понять, каким образом ядро обеспечивает конфигурацию устройства с помощью пути `sysfs`. Наша цель — научиться извлекать информацию об устройствах в системе, чтобы понимать некоторые элементарные операции. В следующих главах будет подробно рассмотрено взаимодействие с особыми типами устройств.

Важно понимать, как ядро взаимодействует с пространством пользователя, когда ему предъявляются новые устройства. Менеджер устройств `udev` позволяет программам из пространства пользователя автоматически конфигурировать и использовать новые устройства. Вы увидите основные моменты того, как ядро отправляет сообщение процессу в пространстве пользователя с помощью менеджера `udev` и какой процесс при этом задействован.

3.1. Файлы устройств

В системе Unix большинством устройств легко управлять, поскольку ядро представляет процессам в виде файлов многие интерфейсы ввода-вывода устройств. Такие файлы устройств иногда называют *узлами устройств*. Не только программист может применять обычные файловые операции для работы с каким-либо устройством: некоторые устройства доступны также стандартным командам вроде `cat`. Вам не обязательно быть программистом, чтобы использовать устройство, однако есть ограничения на то, что вы можете сделать с помощью файлового интерфейса, так что не все устройства или их возможности доступны в стандартном файловом вводе-выводе.

Система Linux применяет ту же схему файлов устройств, какая используется в других вариантах системы Unix. Файлы устройств расположены в каталоге `/dev`, и при запуске команды `ls /dev` обнаружится достаточное количество файлов в этом каталоге. При работе с устройствами для начала попробуйте такую команду:

```
$ echo blah blah > /dev/null
```

Как и положено любой команде с перенаправлением вывода, данная команда отправляет нечто из стандартного вывода в файл. Однако файл `/dev/null` является устройством, и ядро решает, что делать с данными, записываемыми в это устройство. В случае с `/dev/null` ядро просто игнорирует ввод и не использует данные.

Чтобы идентифицировать устройство и просмотреть его права доступа, применяйте команду `ls -l`:

Пример 3.1. Файлы устройств

```
$ ls -l
brw-rw----     1 root disk 8, 1 Sep  6 08:37 sda1
crw-rw-rw-     1 root root 1, 3 Sep  6 08:37 null
prw-r--r--     1 root root    0 Mar  3 19:17 fdata
srw-rw-rw-     1 root root    0 Dec 18 07:43 log
```

Обратите внимание на первый символ в каждой строке (первый символ режима файла) в примере 3.1. Если это символ `b`, `c`, `p` или `s`, такой файл является устройством. Эти буквы обозначают соответственно *блочное устройство*, *символьное устройство*, *канал* и *сокет*, что подробно объяснено ниже.

- **Блочное устройство.** Программы получают доступ к данным на *блочном устройстве* в виде фиксированных порций. В приведенном примере `sda1` является дисковым устройством — одним из типов блочных устройств. Диски можно легко разделить на блоки данных. Поскольку общий объем блочного устройства фиксирован и легко поддается индексации, процессы с помощью ядра получают случайный доступ к любому блоку устройства.
- **Символьное устройство.** Символьные устройства работают с потоками данных. Вы можете лишь считывать символы с таких устройств или записывать символы на них, как было показано в примере с `/dev/null`. Символьные устройства не обладают размером. Когда выполняется чтение или запись, ядро обычно осуществляет операцию чтения или записи на устройство. Принтеры, напрямую подключенные к компьютеру, представлены символьными устройствами. Важно отметить следующее: при взаимодействии с символьным устройством ядро не может выполнить откат данных и повторную их проверку после того, как данные переданы устройству или процессу.
- **Канал.** *Именованные каналы* подобны символьным устройствам, но у них на другом конце потока ввода-вывода располагается другой процесс, а не драйвер ядра.
- **Сокет.** *Сокеты* являются специализированными интерфейсами, которые часто используются для взаимодействия между процессами. Часто они располагаются вне каталога `/dev`. Файлы сокетов представляют сокеты домена Unix (о них вы узнаете из главы 10).

Числа перед датами в первых двух строках примера 3.1 являются *старшим* и *младшим* номерами устройств, помогая ядру при идентификации устройств. Сходные устройства, как правило, снабжены одинаковым старшим номером, например `sda3` и `sdb1` (оба являются разделами жесткого диска).

ПРИМЕЧАНИЕ

Не все устройства обладают файлами устройств, поскольку интерфейсы ввода-вывода для блочных и символьных устройств подходят не для всех случаев. Например, у сетевых интерфейсов нет файлов устройств. Теоретически возможно взаимодействие с сетевым интерфейсом с помощью одного символьного устройства, но, поскольку это оказалось бы исключительно сложным, ядро использует другие интерфейсы ввода-вывода.

3.2. Путь устройств sysfs

Традиционный каталог `/dev` в системе Unix является удобным способом, с помощью которого пользовательские процессы могут обращаться к устройствам, поддерживаемым ядром, а также предоставлять интерфейс для них. Однако такая схема также и слишком упрощена. Название устройства в каталоге `/dev` даст вам некоторую информацию об устройстве, но далеко не всю. Кроме того, ядро назначает устройства в порядке их обнаружения, поэтому они могут получать различные имена после перезагрузки.

Чтобы обеспечить унифицированный обзор присоединенных устройств, взяв за основу их действительные аппаратные характеристики, ядро системы Linux предлагает интерфейс `sysfs` для обозначения файлов и каталогов. Основным путем для устройств является `/sys/devices`. Например, жесткий диск SATA в файле `/dev/sda` мог бы получить следующий путь в интерфейсе `sysfs`:

```
/sys/devices/pci0000:00/0000:00:1f.2/host0/target0:0:0/0:0:0:0/block/sda
```

Как видите, такой путь достаточно длинный по сравнению с именем файла `/dev/sda`, который является также каталогом. Однако в действительности нельзя сравнивать эти пути, поскольку у них разное назначение. Файл `/dev` расположен здесь для того, чтобы пользовательский процесс мог применять устройство, в то время как путь `/sys/devices` использован для просмотра информации об устройстве и для управления им. Если вы составите список содержимого пути устройств, подобного приведенному выше, вы увидите что-либо подобное.

alignment_offset	discard_alignment	holders	removable	size	uevent
bdi	events	inflight	ro	slaves	
capability	events_async	power	sda1	stat	
dev	events_poll_msecs	queue	sda2	subsystem	
device	ext_range	range	sda5	trace	

Названия файлов и подкаталогов предназначены в первую очередь для чтения программами, а не людьми, однако вы сможете понять, что они содержат и представляют, посмотрев какой-либо пример, скажем файл `/dev`. После запуска в этом каталоге команды `cat dev` отобразятся числа `8:0`, которые являются старшим и младшим номерами устройств в файле `/dev/sda`.

В каталоге `/sys` есть несколько ярлыков. Например, ярлык `/sys/block` должен содержать все доступные в системе блочные устройства. Однако это всего лишь символические ссылки. Запустите команду `ls -l /sys/block`, чтобы выяснить их настоящие `sysfs`-пути.

Могут возникнуть затруднения при отыскании sysfs-пути устройства в каталоге /dev. Используйте команду udevadm, чтобы показать путь и другие атрибуты:

```
$ udevadm info --query=all --name=/dev/sda
```

ПРИМЕЧАНИЕ

Команда udevadm расположена в каталоге /sbin; можно указать этот каталог в самом конце командного пути, если его там еще нет.

Подробности о команде udevadm и описание системы udev вы найдете в разделе 3.5.

3.3. Команда dd и устройства

Команда dd чрезвычайно полезна при работе с блочными и символьными устройствами. Единственная функция этой команды заключается в чтении из входного файла или потока и запись в выходной файл или поток, при этом попутно может происходить некоторое преобразование кодировки.

Команда dd копирует данные в блоках фиксированного размера. Пример использования команды dd с некоторыми распространенными параметрами для символического устройства:

```
$ dd if=/dev/zero of=new_file bs=1024 count=1
```

Как видите, формат параметров команды dd отличается от формата большинства других команд системы Unix. Он основан на старом стиле JCL (Job Control Language, язык управления заданиями), применявшемся в компании IBM. Вместо использования дефиса (-) перед параметром вы указываете название параметра, а затем после символа равенства (=) задаете его значение. В приведенном выше примере происходит копирование одного блока размером 1024 байта из потока /dev/zero (непрерывный поток нулевых байтов) в файл new_file.

Приведем важные параметры команды dd.

- if=*file*. Входной файл. По умолчанию применяется стандартный ввод.
- of=*file*. Выходной файл. По умолчанию применяется стандартный вывод.
- bs=*size*. Размер блока. Команда dd будет считывать и записывать указанное количество байтов за один прием. Чтобы сокращенно указать большие объемы данных, можно использовать символы b и k, которые соответствуют значениям 512 и 1024 байт. Так, в приведенном выше примере можно записать bs=1k вместо bs=1024.
- ibs=*size*, obs=*size*. Размеры блоков на вводе и выводе. Если вы можете использовать одинаковые размеры блоков для ввода и вывода, примените параметр bs. Если нет, используйте параметры ibs и obs для ввода и вывода соответственно.
- count=*num*. Общее количество блоков, подлежащих копированию. При работе с большим файлом (или с устройством, которое поддерживает бесконечный поток данных, такой как /dev/zero) следует остановить команду dd в определенной точке, чтобы не израсходовать впустую большое количество дискового

пространства, времени центрального процессора или того и другого сразу. Используйте команду `count` с параметром `skip`, чтобы скопировать небольшой фрагмент из большого файла или устройства.

- `skip=`*num*. Пропускает указанное количество блоков *num* во входном файле или потоке и не копирует их на вывод.

ВНИМАНИЕ

Команда dd является очень мощной, при ее запуске вы должны понимать, что делаете. Очень легко повредить файлы или данные устройств, совершив по небрежности ошибку. Часто может помочь запись вывода в новый файл, если вы не вполне уверены в результатах.

3.4. Сводка имен устройств

Иногда бывает сложно отыскать название устройства (например, при создании разделов диска). Вот несколько способов выяснить это.

- Выполните запрос `udevd` с помощью команды `udevadm` (см. раздел 3.5).
- Поищите устройство в каталоге `/sys`.
- Попробуйте угадать название на основе результатов вывода команды `dmesg` (которая выдает несколько последних сообщений ядра) или из файла системного журнала ядра (см. раздел 7.2). Эти результаты могут содержать описание устройств в вашей системе.
- Для дискового устройства, которое уже видно в системе, можно посмотреть результаты работы команды `mount`.
- Запустите команду `cat /proc/devices`, чтобы увидеть блочные и символьные устройства, для которых ваша система уже имеет драйверы. Каждая строка состоит из номера и имени. Номер является старшим номером устройства, как рассказано в разделе 3.1. Если вы сможете определить устройство по его имени, поищите в каталоге `/dev` символьное или блочное устройство с соответствующим старшим номером. Таким образом вы найдете файлы устройства.

Среди перечисленных методов только первый является надежным, но для него необходим менеджер устройств udev. Если вы окажетесь в ситуации, когда этот менеджер недоступен, попробуйте остальные методы, памятуя о том, что в ядре может не оказаться файла устройства для вашего аппаратного средства.

В следующих разделах перечислены наиболее распространенные устройства Linux и соглашения об их именовании.

3.4.1. Жесткие диски: /dev/sd*

Большинству жестких дисков в современных системах Linux соответствуют имена устройств с префиксом `sd`, например `/dev/sda`, `/dev/sdb` и т. д. Такие устройства представляют диски полностью; для разделов диска ядро создает отдельные файлы устройств, например `/dev/sda1` и `/dev/sda2`.

Соглашение об именовании потребует небольшого объяснения. Префикс `sd` в имени устройства соответствует *диску SCSI*. Интерфейс SCSI (Small Computer

System Interface, интерфейс малых вычислительных систем) изначально был разработан в качестве стандарта для аппаратных средств и протокола коммуникации между устройствами (например, дисками) и другой периферией. Хотя в большинстве современных компьютеров не используются традиционные аппаратные средства SCSI, сам этот протокол присутствует повсюду благодаря своей приспособляемости. Например, USB-накопители применяют его при подключении. В случае с дисками SATA дело немного усложняется, однако ядро системы Linux в определенный момент по-прежнему использует команды SCSI для обращения к таким дискам. Одним из наиболее элегантных инструментов является команда `lsscsi`. Вот пример того, что можно получить в результате ее работы:

```
$ lsscsi
[0:0:0:0]❶     disk❷    ATA      WDC WD3200AAJS-2 01.0  /dev/sda❸
[1:0:0:0]      cd/dvd   Slimtype DVD A  DS8A5SH   XA15  /dev/sr0
[2:0:0:0]      disk     FLASH    Drive UT_USB20   0.00  /dev/sdb
```

В столбце ❶ приводится адрес устройства в системе. В столбце ❷ представлено описание типа устройства, а в последнем столбце ❸ указано, где можно найти файл устройства. Остальная информация содержит сведения о производителе.

Система Linux назначает устройствам файлы устройств в том порядке, в каком их драйверы находят устройства. Так, в приведенном выше примере ядро сначала нашло жесткий диск, затем оптический привод и наконец флеш-накопитель.

К сожалению, такая схема назначения устройств обычно вызывала проблемы при перенастройке аппаратного обеспечения. Допустим, в вашей системе три диска: `/dev/sda`, `/dev/sdb` и `/dev/sdc`. Если диск `/dev/sdb` вышел из строя и вы должны заменить его, чтобы компьютер снова смог работать, то диск, который до этого был диском `/dev/sdc`, превратится в диск `/dev/sdb`, а диска `/dev/sdc` больше не будет. Если вы ссылались непосредственно на названия устройств в файле `fstab` (см. подраздел 4.2.8), то вам придется внести некоторые изменения в данный файл, чтобы вернуть (основную часть) устройства к нормальной работе. Чтобы устранить подобные проблемы, в большинстве современных систем Linux используется идентификатор UUID (Universally Unique Identifier, универсальный уникальный идентификатор, см. подраздел 4.2.4) для устойчивого доступа к дисковым устройствам.

Здесь лишь вкратце рассказано о том, как использовать диски и другие устройства для хранения данных в системе Linux. Дополнительную информацию о применении дисков можно найти в главе 4. Далее в текущей главе мы изучим то, каким образом интерфейс SCSI участвует в работе ядра системы Linux.

3.4.2. Приводы CD и DVD: /dev/sr*

Система Linux распознает большинство оптических приводов как устройства SCSI с именами `/dev/sr0`, `/dev/sr1` и т. д. Однако если такое устройство использует устаревший интерфейс, оно может отобразиться как устройство PATA (об этом пойдет речь ниже). Устройства `/dev/sr*` доступны только для чтения и применяются только для считывания оптических дисков. Для использования возможностей записи и перезаписи в оптических приводах следует применять «обобщенные» SCSI-устройства, такие как `/dev/sg0`.

3.4.3. Жесткие диски PATA: /dev/hd*

Блочные устройства системы Linux `/dev/hda`, `/dev/hdb`, `/dev/hdc` и `/dev/hdd` часто встречаются в старых версиях ядра системы и для устаревших аппаратных средств. Это фиксированные назначения, которые основаны на ведущем и ведомом устройствах с интерфейсами 0 и 1. Иногда вы можете обнаружить, что диск SATA распознан как один из таких дисков. Это означает, что диск SATA работает в режиме совместимости, который снижает производительность. Проверьте настройки системы BIOS, чтобы выяснить, можно ли переключить контроллер диска SATA в его штатный режим.

3.4.4. Терминалы: /dev/tty*, /dev/pts/* и /dev/tty

Терминалы — это устройства для перемещения символов между пользовательским процессом и устройством ввода-вывода, как правило, для вывода текста на экран терминала. Интерфейс терминальных устройств прошел довольно долгий путь с тех пор, когда терминалы были основаны на печатающих аппаратах.

Псевдотерминальные устройства — это эмулированные терминалы, которые понимают функции ввода-вывода реальных терминалов. Однако вместо того, чтобы обращаться к реальному аппаратному средству, ядро представляет интерфейс ввода-вывода посредством какой-либо программы, например окна терминала оболочки, в котором вы, вероятно, печатаете большинство команд.

Двумя общими терминальными устройствами являются `/dev/tty1` (первая виртуальная консоль) и `/dev/pts/0` (первое псевдотерминальное устройство). Каталог `/dev/pts` сам по себе является специализированной файловой системой.

Устройство `/dev/tty` — это управляющий терминал текущего процесса. Если какая-либо команда в данный момент производит чтение или запись в терминале, данное устройство выступает в качестве синонима такого терминала. Процесс не обязательно должен быть присоединен к терминалу.

Режимы отображения и виртуальные консоли

В системе Linux есть два основных режима отображения: текстовый режим и оконная система *«Сервер X Window System»* (графический режим, как правило, с использованием менеджера отображения). Хотя система Linux традиционно загружается в текстовом режиме, сейчас в большинстве дистрибутивов используются параметры ядра и промежуточные механизмы графического отображения (экраны загрузки, такие как plymouth), чтобы полностью скрыть текстовый режим при загрузке системы. В подобных случаях система переключается в полноценный графический режим незадолго до окончания процесса загрузки.

Система Linux поддерживает *виртуальные консоли*, чтобы поддерживать несколько дисплеев. Каждая виртуальная консоль может работать в графическом или в текстовом режиме. В текстовом режиме можно переключаться между консолями с помощью сочетания клавиши Alt с какой-либо функциональной клавишей. Например, сочетание Alt+F1 переведет вас в консоль `/dev/tty1`, сочетание Alt+F2 — в консоль `/dev/tty2` и т. д. Многие из таких сочетаний могут быть

заняты процессом getty, который обслуживает вход в систему (подробнее — в разделе 7.4).

Виртуальная консоль, которая применяется X-сервером в графическом режиме, немного отличается. Вместо того чтобы получить назначение виртуальной консоли из начальной конфигурации, X-сервер занимает свободную виртуальную консоль, если он не был направлен на использование конкретной консоли. Например, если процесс getty запущен в консолях tty1 и tty2, новый X-сервер займет консоль tty3. Кроме того, после перевода виртуальной консоли в графический режим следует, как правило, нажимать сочетание клавиш Ctrl+Alt с функциональной клавишей для переключения к другой виртуальной консоли, а не просто сочетание клавиши Alt с какой-либо функциональной клавишей.

В довершение ко всему, если вы желаете увидеть текстовую консоль после загрузки системы, нажмите сочетание клавиш Ctrl+Alt+F1. Чтобы вернуться в сессию X11, нажимайте сочетания Alt+F2, Alt+F3 и т. д., пока не окажетесь в X-сессии.

Если у вас возникли сложности с переключением консолей в результате неверной работы механизма ввода или по каким-либо другим причинам, можно попытаться принудительно сменить консоли с помощью команды chvt. Например, чтобы переключиться в консоль tty1, запустите эту команду с правами доступа root:

```
# chvt 1
```

3.4.5. Последовательные порты: /dev/ttyS*

Старые последовательные порты RS-232 и подобные им являются специальными терминальными устройствами. Поскольку приходится заботиться о слишком большом количестве параметров, таких как скорость передачи и управление потоком, выполнить с последовательным портом с помощью командной строки можно немногое.

Порт, известный в Windows как COM1, называется /dev/ttyS0; COM2 — это /dev/ttyS1 и т. д. В именах подключаемых последовательных USB-адаптеров будет присутствовать сочетание USB или ACM: /dev/ttyUSB0, /dev/ttyACM0, /dev/ttyUSB1, /dev/ttyACM1 и т. п.

3.4.6. Параллельные порты: /dev/lp0 и /dev/lp1

Представляя тип интерфейса, который теперь широко заменен интерфейсом USB, устройства /dev/lp0 и /dev/lp1 с однонаправленным параллельным портом соответствуют портам LPT1 и LPT2 в Windows. Можно отправлять файлы (например, для распечатки) напрямую на параллельный порт с помощью команды cat, однако может потребоваться выполнить перед этим дополнительную подачу страницы или перезагрузку принтера. Сервер печати, например CUPS, гораздо лучше осуществляет взаимодействие с принтером.

Двунаправленными параллельными портами являются /dev/parport0 и /dev/parport1.

3.4.7. Аудиоустройства: /dev/snd/*, /dev/dsp, /dev/audio и другие

В системе Linux присутствуют два набора аудиоустройств. Это устройства для системного интерфейса *ALSA* (Advanced Linux Sound Architecture, продвинутая звуковая архитектура Linux), а также старый драйвер *OSS* (Open Sound System, открытая звуковая система). Устройства ALSA находятся в каталоге `/dev/snd`, однако работать с ними напрямую трудно. Системы Linux, которые используют интерфейс ALSA, поддерживают обратно совместимые со стандартом OSS устройства, если ядром загружена поддержка OSS.

Некоторые элементарные операции возможны с OSS-устройствами `dsp` и `audio`. Например, компьютер воспроизведет любой файл в формате WAV, если вы отправите его на устройство `/dev/dsp`. Тем не менее аппаратное средство может выполнить не совсем то, что вы ожидаете, вследствие несоответствия частоты. Кроме того, в большинстве систем устройство часто становится занято, как только вы войдете в систему.

ПРИМЕЧАНИЕ

Звук в системе Linux — довольно запутанная вещь вследствие большого количества вовлеченных слоев. Мы только что говорили об устройствах на уровне ядра, но обычно в пространстве пользователя присутствуют такие серверы, как pulse-audio, которые управляют звуком из различных источников и выступают в качестве посредника между звуковыми устройствами и другими процессами пространства пользователя.

3.4.8. Создание файлов устройств

В современных системах Linux вы не создаете файлы устройств сами; это выполняется с помощью файловой системы devtmpfs и менеджера устройств udev (см. раздел 3.5). Однако полезно увидеть, как это было сделано, поскольку в редких случаях вам может понадобиться создать именованный канал.

Команда `mknod` создает одно устройство. Вы должны знать имя устройства, а также его старший и младший номера. Например, чтобы создать устройство `/dev/sda1`, используйте следующую команду:

```
# mknod /dev/sda1 b 8 2
```

Параметры `b 8 2` определяют блочное устройство со старшим номером 8 и младшим номером 2. Для символьных устройств или именованных каналов используйте параметры `c` или `p` вместо `b` (для именованных каналов номера устройства опускаются).

Как отмечалось ранее, команда `mknod` полезна только для преднамеренного создания именованного канала. Одно время она была также полезна при создании отсутствующих устройств, когда производилось восстановление системы в режиме одного пользователя.

В старых версиях систем Unix и Linux обслуживание каталога `/dev` было весьма трудоемким. После каждого существенного обновления ядра или добавления драйвера появлялась возможность поддержки дополнительных типов

устройств, это означало, что именам файлов устройств можно назначать новый набор старшего и младшего номеров устройств. Поддерживать это было трудно, поэтому в каждой системе имелась команда MAKEDEV в каталоге /dev для создания групп устройств. После обновления системы следовало попробовать найти обновленную команду MAKEDEV, а затем запустить ее, чтобы создать новые устройства.

Такая статичная система становилась неуклюжей, поэтому ее пришлось заменить. Первой попыткой исправления ситуации была файловая система devfs, реализация каталога /dev в пространстве ядра, содержащая все устройства, которые поддерживает текущее ядро. Однако в ней присутствовали определенные ограничения, которые привели к разработке менеджера устройств udev и файловой системы devtmpfs.

3.5. Менеджер устройств udev

Излишняя усложненность ядра приводит к нестабильности в системе. Пример тому — управление файлами устройств: можно создавать файлы устройств в пространстве пользователя, так почему бы не выполнять это в ядре? Ядро системы Linux отправляет уведомления процессам в пространстве пользователя (они называются udevd) после обнаружения нового устройства в системе (например, после подключения USB-накопителя). С другой стороны, процесс в пространстве пользователя проверяет характеристики нового устройства, создает файл устройства, а затем выполняет необходимую инициализацию устройства.

Это была теория. К сожалению, на практике такой подход проблематичен: файлы устройств необходимы уже на ранней стадии загрузки системы, поэтому уведомления udevd должны начинать работу рано. Чтобы создать файлы устройств, менеджеру udevd не следует зависеть от устройств, для создания которых предназначен. Он должен выполнять начальный запуск очень быстро, чтобы остальная часть системы не оказалась в подвешенном состоянии, ожидая запуска менеджера udevd.

3.5.1. Файловая система devtmpfs

Файловая система devtmpfs была разработана для решения проблемы с доступностью устройств во время загрузки (см. подробности в разделе 4.2). Эта файловая система подобна старой devfs, но является упрощенной. Ядро создает файлы устройств по мере надобности, а также уведомляет менеджер udevd о том, что доступно новое устройство. После получения такого сигнала менеджер udevd не создает файлы устройств, а выполняет инициализацию устройства и отправляет уведомление процессу. Кроме того, он создает несколько символических ссылок в каталоге /dev для дальнейшей идентификации устройств. Примеры вы можете отыскать в каталоге /dev/disk/by-id, в котором каждому присоединенному диску соответствует одна или несколько записей.

Рассмотрим, например, такой типичный диск:

```
lrwxrwxrwx 1 root root 9 Jul 26 10:23 scsi-SATA_WDC_WD3200AAJS-_WD-WMAV2FU80671 ->
../../sda

lrwxrwxrwx 1 root root 10 Jul 26 10:23 scsi-SATA_WDC_WD3200AAJS-_WD-WMAV2FU80671-
part1 ->

  ../../sda1

lrwxrwxrwx 1 root root 10 Jul 26 10:23 scsi-SATA_WDC_WD3200AAJS-_WD-WMAV2FU80671-
part2 ->

  ../../sda2

lrwxrwxrwx 1 root root 10 Jul 26 10:23 scsi-SATA_WDC_WD3200AAJS-_WD-WMAV2FU80671-
part5 ->

  ../../sda5
```

Менеджер udevd дает имена ссылкам по типу интерфейса, а затем по информации об изготовителе и модели, серийному номеру и разделу (если применяется).

В следующем разделе подробно рассказано о том, как менеджер udevd выполняет свою работу: как узнает о том, какие символические ссылки создать, и как создает их. Эта информация необязательна, чтобы продолжить чтение книги. Если вы впервые встречаетесь с устройствами Linux, можете смело переходить к следующей главе, чтобы начать изучение того, как использовать диски.

3.5.2. Работа и настройка менеджера udevd

Демон `udevd` работает следующим образом.

1. Ядро отправляет демону `udevd` событие-уведомление, называемое `uevent`, через внутреннюю сетевую ссылку.
2. Демон `udevd` загружает все атрибуты, содержащиеся в уведомлении `uevent`.
3. Демон `udevd` проводит разбор правил, а затем предпринимает действия или устанавливает дополнительные атрибуты на основе правил.

Входящее уведомление `uevent`, которое демон `udevd` получает от ядра, может выглядеть так:

```
ACTION=change
DEVNAME=sde
DEVPATH=/devices/pci0000:00/0000:00:1a.0/usb1/1-1/1-1.2/1-1.2:1.0/host4/
  target4:0:0/4:0:0:3/block/sde
DEVTYPE=disk
DISK_MEDIA_CHANGE=1
MAJOR=8
MINOR=64
SEQNUM=2752
SUBSYSTEM=block
UDEV_LOG=3
```

Можно заметить, что здесь присутствует смена устройства. После получения уведомления uevent демон udevd знает путь устройства в файловой системе sysfs, а также некоторые другие атрибуты, связанные с его свойствами, и готов начать обработку правил.

Файлы правил расположены в каталогах /lib/udev/rules.d и /etc/udev/rules.d. Правила из каталога /lib являются правилами по умолчанию, а правила из каталога /etc переопределяют их. Подробное объяснение правил вы сможете найти на странице руководства udev(7).

Взглянем на символические ссылки из примера /dev/sda в поразделе 3.5.1. Эти ссылки были определены с помощью правил из файла /lib/udev/rules.d/60-persistent-storage.rules. В нем вы увидите следующие строки:

```
# ATA devices using the "scsi" subsystem

KERNEL=="sd*[!0-9]|sr*", ENV{ID_SERIAL}!="?*", SUBSYSTEMS=="scsi",
ATTRS{vendor}=="ATA", IMPORT{program}="ata_id --export $tempnode"

# ATA/ATAPI devices (SPC-3 or later) using the "scsi" subsystem

KERNEL=="sd*[!0-9]|sr*", ENV{ID_SERIAL}!="?*", SUBSYSTEMS=="scsi",

  ATTRS{type}=="5", ATTRS{scsi_level}=="[6-9]*", IMPORT{program}="ata_id --export
$tempnode"
```

Эти правила сопоставляют диски ATA, представленные с помощью SCSI-подсистемы ядра (см. раздел 3.6). Можно заметить здесь несколько правил, предназначенных для отслеживания различных способов, с помощью которых могут быть представлены устройства, но суть заключается в том, что демон udevd будет пытаться подобрать устройство, начиная с символов sd или sr без номера (с использованием выражения KERNEL=="sd*[!0-9]|sr*"), а также подсистему (SUBSYSTEMS=="scsi") и, наконец, некоторые другие атрибуты. Если все эти условные выражения окажутся истинными, то демон udevd переходит к следующему выражению:

```
IMPORT{program}="ata_id --export $tempnode"
```

Это уже не условие, а скорее указание выполнить импорт переменных из команды /lib/udev/ata_id. Если у вас есть подобный диск, попробуйте самостоятельно ввести в командной строке следующее:

```
$ sudo /lib/udev/ata_id --export /dev/sda
ID_ATA=1
ID_TYPE=disk
ID_BUS=ata
ID_MODEL=WDC_WD3200AAJS-22L7A0
ID_MODEL_ENC=WDC\x20WD3200AAJS22L7A0\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20
  \x20\x20\x20\x20\x20\x20\x20\x20\x20\x20
ID_REVISION=01.03E10
ID_SERIAL=WDC_WD3200AAJS-22L7A0_WD-WMAV2FU80671
--snip--
```

Теперь импорт настраивает окружение так, чтобы все имена переменных в данном выводе приняли указанные значения. Например, любое правило, которое появится следом, будет распознавать ENV{ID_TYPE} в качестве диска.

Особо следует отметить переменную ID_SERIAL. В каждом из правил на втором месте следует такое условие:

```
ENV{ID_SERIAL}!="?*"
```

Это означает, что оно будет истинным, только если для переменной ID_SERIAL не указано значение. Следовательно, если она *уже* определена, условие будет ложным и все правило также станет ложным, в результате чего демон udevd перейдет к следующему правилу.

В чем же суть? Целью этих двух правил (и многих других в том же файле) является поиск серийного номера дискового устройства. Если значение переменной ENV{ID_SERIAL} установлено, демон udevd может проверить следующее правило:

```
KERNEL=="sd*|sr*|cciss*", ENV{DEVTYPE}=="disk", ENV{ID_SERIAL}=="?*",
  SYMLINK+="disk/by-id/$env{ID_BUS}-$env{ID_SERIAL}"
```

Для этого правила необходимо задать значение переменной ENV{ID_SERIAL}. В нем есть также одна директива:

```
SYMLINK+="disk/by-id/$env{ID_BUS}-$env{ID_SERIAL}"
```

После встречи с этой директивой демон udevd добавляет символическую ссылку на обнаруженное устройство. Итак, теперь вы знаете, откуда берутся символические ссылки на устройства.

Вероятно, вам любопытно узнать, как отличить условное выражение от директивы. Условные выражения записываются с помощью двух знаков равенства (==) или восклицательного знака и знака равенства (!=). Для директив используют либо один знак равенства (=), либо символ «плюс» и знак равенства (+=), либо двоеточие со знаком равенства (:=).

3.5.3. Команда udevadm

Команда udevadm является инструментом администрирования менеджера udevd. С ее помощью можно перезагрузить правила для udevd, а также события-триггеры. Однако, вероятно, самыми мощными функциями команды udevadm являются возможность поиска и обнаружения системных устройств, а также способность отслеживать уведомления uevents, когда демон udevd получает их от ядра. Единственную сложность может составить синтаксис этой команды, который становится немного запутанным.

Начнем с рассмотрения системного устройства. Вернувшись к примеру из подраздела 3.5.2, чтобы взглянуть на все атрибуты менеджера udev, которые использованы и сгенерированы в сочетании с правилами для устройства (такого как /dev/sda), запустите следующую команду:

```
$ udevadm info --query=all --name=/dev/sda
```

Результат будет выглядеть так:

```
P: /devices/pci0000:00/0000:00:1f.2/host0/target0:0:0/0:0:0:0/block/sda
N: sda
S: disk/by-id/ata-WDC_WD3200AAJS-22L7A0_WD-WMAV2FU80671
S: disk/by-id/scsi-SATA_WDC_WD3200AAJS-_WD-WMAV2FU80671
S:  disk/by-id/wwn-0x50014ee057faef84  S:  disk/by-path/pci-0000:00:1f.2-scsi-
0:0:0:0
E: DEVLINKS=/dev/disk/by-id/ata-WDC_WD3200AAJS-22L7A0_WD-WMAV2FU80671 /dev/disk/by-
id/scsi
  -SATA_WDC_WD3200AAJS-_WD-WMAV2FU80671 /dev/disk/by-id/wwn-
0x50014ee057faef84 /dev/disk/by
  -path/pci-0000:00:1f.2-scsi-0:0:0:0
E: DEVNAME=/dev/sda
E: DEVPATH=/devices/pci0000:00/0000:00:1f.2/host0/target0:0:0/0:0:0:0/block/sda
E: DEVTYPE=disk
E: ID_ATA=1
E: ID_ATA_DOWNLOAD_MICROCODE=1
E: ID_ATA_FEATURE_SET_AAM=1
--snip--
```

Префикс в каждой строке указывает на атрибут или другую характеристику устройства. В данном случае `P:` в самом верху содержит путь устройства в файловой системе sysfs; `N:` является узлом устройства (то есть именем, которое присвоено файлу `/dev`), `S:` указывает символическую ссылку на узел устройства, которую демон `udevd` поместил в каталог `/dev` в соответствии со своими правилами; `E:` содержит дополнительную информацию об устройстве, извлеченную из правил `udevd`. В приведенном примере было гораздо больше строк вывода, не показанных здесь; попробуйте применить команду самостоятельно, чтобы получить представление о ее работе.

3.5.4. Отслеживание устройств

Чтобы отслеживать уведомления `uevents` с помощью инструмента `udevadm`, используйте команду `monitor`:

```
$ udevadm monitor
```

Результат (когда вы, например, вставите флеш-накопитель) будет выглядеть как этот сокращенный пример:

```
KERNEL[658299.569485]   add    /devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.2
(usb)
KERNEL[658299.569667]   add    /devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.2/2-
1.2:1.0 (usb)
KERNEL[658299.570614]   add    /devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.2/2-
1.2:1.0/host15
  (scsi)
KERNEL[658299.570645]   add    /devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.2/2-
1.2:1.0/
  host15/scsi_host/host15 (scsi_host)
```

```
UDEV  [658299.622579]  add    /devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.2
(usb)
UDEV  [658299.623014]  add    /devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.2/2-
1.2:1.0 (usb)
UDEV  [658299.623673]  add    /devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.2/2-
1.2:1.0/host15
  (scsi)
UDEV  [658299.623690]  add    /devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.2/2-
1.2:1.0/
  host15/scsi_host/host15 (scsi_host)
--snip--
```

Каждое сообщение здесь присутствует дважды, поскольку по умолчанию выводятся как входящие сообщения от ядра (помеченные словом `KERNEL`), так и те сообщения, которые демон `udevd` отправляет другим программам по окончании обработки и фильтрации событий. Чтобы увидеть только события ядра, добавьте параметр `--kernel`, а чтобы увидеть только исходящие события, используйте параметр `--udev`. Чтобы увидеть входящее уведомление `uevent` полностью, включая те атрибуты, которые показаны в подразделе 3.5.2, применяйте параметр `--property`.

Можно также отфильтровать события для какой-либо подсистемы. Например, чтобы увидеть только сообщения ядра, относящиеся только к изменениям в подсистеме SCSI, используйте следующую команду:

```
$ udevadm monitor --kernel --subsystem-match=scsi
```

Подробно о команде `udevadm` можно прочитать в руководстве на странице udevadm(8).

Помимо менеджера udev, есть и другие. Например, в шине D-Bus системы межпроцессного взаимодействия присутствует демон `udisks-daemon`, который отслеживает исходящие события менеджера `udevd`, чтобы автоматически подключать диски, а затем уведомлять программное обеспечение рабочей станции о доступности нового диска.

3.6. Подробнее: интерфейс SCSI и ядро Linux

В этом разделе мы рассмотрим поддержку интерфейса SCSI в ядре Linux, чтобы исследовать часть архитектуры ядра системы. Если вы торопитесь использовать какой-либо диск, переходите сразу к главе 4, изложенные здесь сведения вам будут не нужны. Кроме того, представленный в данном разделе материал более сложен и абстрактен, поэтому, если вы желаете остаться в практическом русле, вам определенно следует пропустить оставшуюся часть главы.

Начнем с небольшой предварительной информации. Традиционная конфигурация аппаратных средств SCSI состоит из хост-адаптера, который соединен с цепью устройств с помощью шины SCSI, как показано на рис. 3.1. Хост-адаптер подключен к компьютеру. У этого адаптера и у всех устройств есть идентифика-

торы SCSI ID, и в зависимости от версии интерфейса SCSI шина может поддерживать 8 или 16 идентификаторов. Наверное, вам приходилось слышать термин *исполнитель SCSI*, который используется по отношению к устройству и его идентификатору SCSI ID.

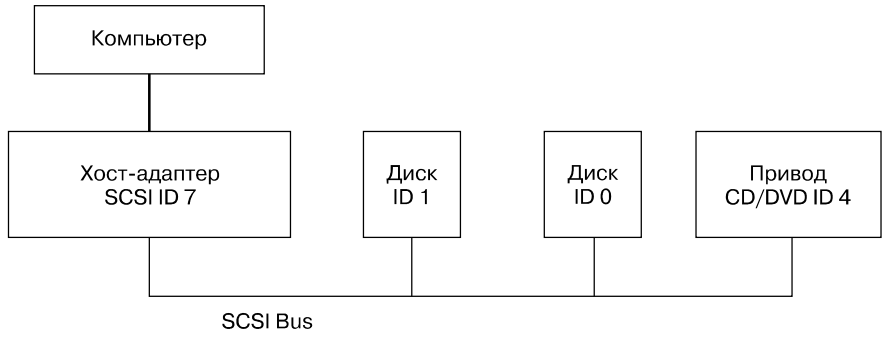

Рис. 3.1. Шина SCSI с хост-адаптером и устройствами

Хост-адаптер взаимодействует с устройствами с помощью набора команд SCSI в одноранговой связи; устройства высылают отклик хост-адаптеру. Компьютер не подключен к цепи устройств напрямую, он должен «пройти» через хост-адаптер, чтобы взаимодействовать с дисками и другими устройствами. Как правило, компьютер для связи с устройствами отправляет SCSI-команды хост-адаптеру, а устройства возвращают отклик также через него.

Новые версии интерфейса SCSI, например *SAS* (Serial Attached SCSI, интерфейс SCSI с последовательным подключением), обеспечивают исключительную производительность, однако вы, вероятно, не отыщете настоящие SCSI-устройства в большинстве компьютеров. Гораздо чаще вы встретите USB-устройства хранения, которые используют команды SCSI. В дополнение к ним устройства, поддерживающие интерфейс ATAPI (например, приводы CD/DVD-ROM), применяют вариант набора команд SCSI.

Диски SATA также присутствуют в системе в качестве устройств SCSI, что достигается с помощью слоя трансляции в библиотеке libata (см. подраздел 3.6.2). Некоторые контроллеры SATA (в особенности высокопроизводительные RAID-контроллеры) осуществляют такую трансляцию аппаратно.

Как же все это уживается вместе? Рассмотрите устройства, показанные в следующей системе:

```
$ lsscsi
[0:0:0:0]   disk    ATA      WDC WD3200AAJS-2 01.0  /dev/sda
[1:0:0:0]   cd/dvd  Slimtype DVD A  DS8A5SH   XA15  /dev/sr0
[2:0:0:0]   disk    USB2.0   CardReader CF    0100  /dev/sdb
[2:0:0:1]   disk    USB2.0   CardReader SM XD 0100  /dev/sdc
[2:0:0:2]   disk    USB2.0   CardReader MS    0100  /dev/sdd
[2:0:0:3]   disk    USB2.0   CardReader SD    0100  /dev/sde
[3:0:0:0]   disk    FLASH    Drive UT_USB20   0.00  /dev/sdf
```

Числа в скобках значат следующее (слева направо): номер хост-адаптера SCSI, номер шины SCSI, идентификатор устройства SCSI ID и номер логического устройства LUN (Logical Unit Number, дальнейшее подразделение устройства).

В данном примере в наличии четыре подключенных адаптера (`scsi0`, `scsi1`, `scsi2` и `scsi3`), каждый обладает единственной шиной (ее номер всюду 0) с одним устройством на ней (номер исполнителя также 0). Флеш-ридер USB с идентификатором 2:0:0 имеет четыре логических устройства — по одному на каждый тип флеш-карты, которая может быть вставлена. Ядро назначило отдельный файл устройства каждому логическому устройству.

На рис. 3.2 показана иерархия драйверов и интерфейсов внутри ядра для приведенной системной конфигурации, начиная от индивидуальных драйверов устройств и заканчивая драйверами блоков. Сюда не включены обобщенные драйверы SCSI (`sg`, SCSI generic).

Поначалу такая обширная структура может показаться необъятной, однако поток данных здесь весьма линейный. Начнем анализировать ее, рассмотрев подсистему SCSI и три ее слоя драйверов.

- Верхний слой отвечает за операции для класса устройств. Например, на этом слое имеется драйвер `sd` (для SCSI-диска); он знает, как переводить запросы от интерфейса блочных устройств в специальные команды протокола SCSI для дисков и наоборот.
- Средний слой анализирует и направляет сообщения SCSI между верхним и нижним слоями, а также отслеживает все шины SCSI и устройства, подключенные к системе.
- Нижний слой отвечает за действия, связанные с аппаратными средствами. Расположенные здесь драйверы отсылают исходящие сообщения протокола SCSI конкретным ведущим адаптерам или аппаратным средствам, а также принимают входящие сообщения от аппаратных средств. Причина для такого отделения от верхнего слоя заключается в том, что, хотя сообщения протокола SCSI и унифицированы для какого-либо класса устройств (например, для дисков), разные типы хост-адаптеров обладают отличающимися процедурами для отправки одинаковых сообщений.

Верхний и нижний слои содержат множество различных драйверов, однако важно помнить о том, что для любого файла устройства в вашей системе ядро применяет один драйвер верхнего слоя и один драйвер нижнего слоя. В нашем примере для диска `/dev/sda` ядро использует драйвер `sd` верхнего слоя и драйвер моста ATA на нижнем слое.

Иногда вам может потребоваться применить более одного драйвера верхнего слоя для одного аппаратного средства (см. подраздел 3.6.3).

Для настоящих аппаратных средств SCSI, таких как диск, подключенный к хост-адаптеру SCSI, или аппаратный RAID-контроллер, драйверы нижнего слоя напрямую «общаются» с расположенными ниже аппаратными средствами. Однако для большинства аппаратных средств, которые подключены к вашей подсистеме SCSI, история совсем другая.

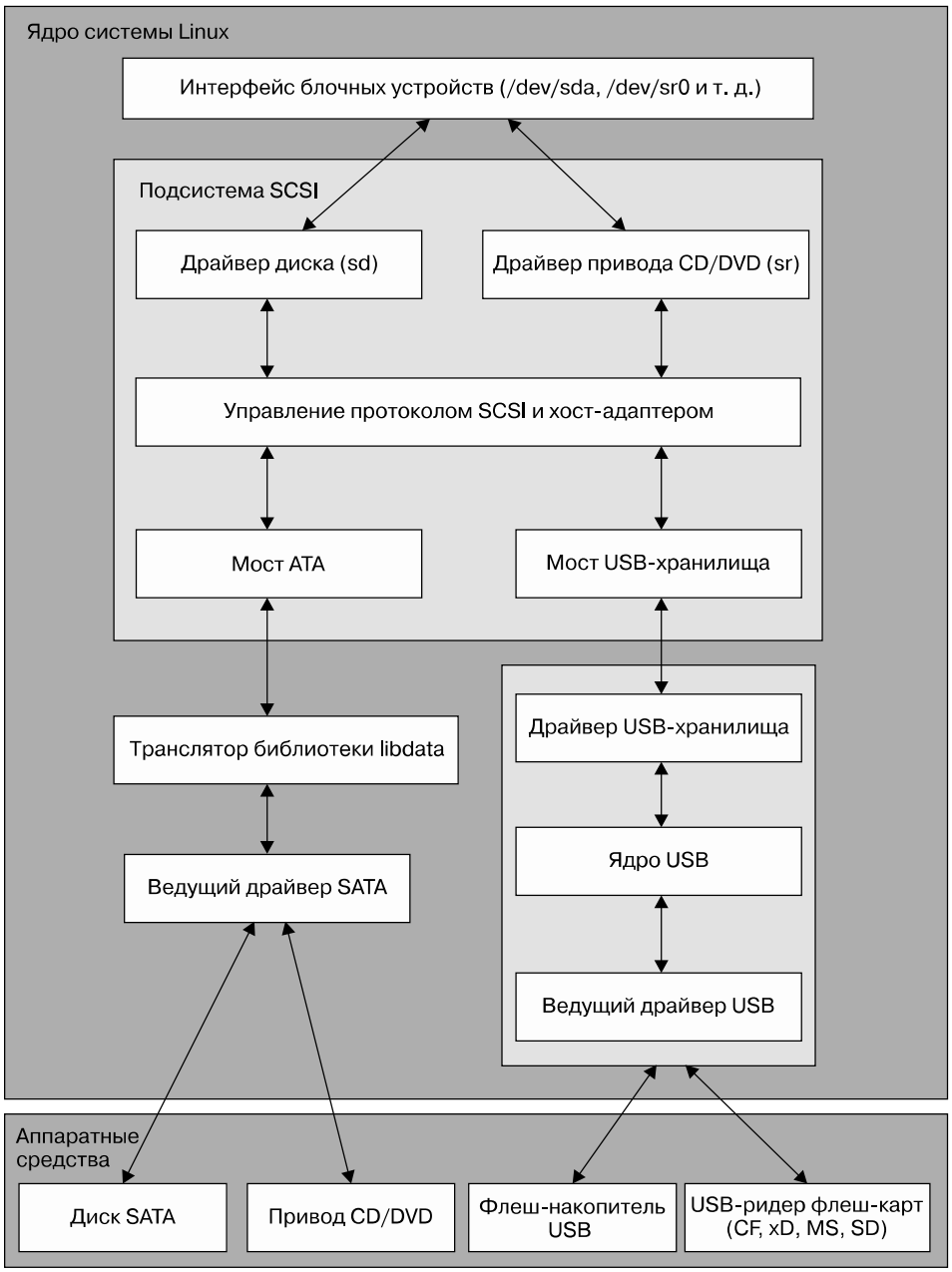

Рис. 3.2. Схема подсистемы SCSI в Linux

3.6.1. USB-хранилища и протокол SCSI

Чтобы подсистема SCSI могла взаимодействовать с обычными USB-накопителями, как показано на рис. 3.2, ядру необходим не только драйвер SCSI на нижнем

слое. Флеш-устройство USB, представленное файлом `/dev/sdf`, понимает команды SCSI, но для реальной связи с устройством ядру необходимо знать, каким образом «общаться» через систему USB.

В абстракции подсистема USB очень похожа на SCSI: у нее есть классы устройств, шины и хост-контроллеры. Следовательно, не должно вызывать удивления то, что ядро системы Linux содержит трехслойную подсистему USB, которая сильно напоминает подсистему SCSI: сверху расположены драйверы классов устройств, в середине находится ядро управления шиной, а внизу — драйверы хост-контроллера. Подобно тому как подсистема SCSI передает команды между своими компонентами, подсистема USB пересылает сообщения между своими компонентами. В ней есть даже команда `lsusb`, которая подобна команде `lsscsi`.

Часть, которая нам здесь особенно интересна, находится вверху. Это драйвер USB-хранилища. Данный драйвер играет роль переводчика. С одной стороны, он «говорит» на языке SCSI, а с другой — на языке USB. Поскольку аппаратные средства для хранения данных включают команды SCSI внутрь сообщений USB, у драйвера довольно простая работа: главным образом он занят переупаковкой данных.

Когда обе подсистемы (SCSI и USB) заняли свои места, у вас в наличии практически все, что необходимо для обращения к флеш-накопителю. Последнее недостающее звено — это драйвер нижнего слоя в подсистеме SCSI, так как драйвер USB-хранилища является частью подсистемы USB, а не подсистемы SCSI. (Из организационных соображений две подсистемы не должны совместно использовать один и тот же драйвер.) Чтобы две подсистемы смогли общаться друг с другом, на нижнем слое простой драйвер моста SCSI выполняет соединение с драйвером хранилища в подсистеме USB.

3.6.2. Интерфейсы SCSI и ATA

Жесткий диск SATA и оптический привод, показанные на рис. 3.2, используют один и тот же интерфейс SATA. Чтобы присоединить драйверы ядра, специфичные для интерфейса SATA, к подсистеме SCSI, ядро задействует драйвер-мост, подобный мосту для USB-накопителей, но с другим механизмом и дополнительными усложнениями. Оптический привод «говорит» на языке ATAPI (это версия команд SCSI, закодированных в протокол ATA). Однако жесткий диск не использует интерфейс ATAPI и не кодирует никаких команд SCSI!

Ядро Linux применяет часть библиотеки libata, чтобы «примирить» приводы SATA (и ATA) с подсистемой SCSI. Для оптических приводов с интерфейсом ATAPI это довольно простая задача, заключающаяся в упаковке и извлечении SCSI-команд, содержащихся в протоколе ATA. Для жесткого диска задача существенно усложняется, поскольку библиотека должна выполнять полную трансляцию команд.

Работа оптического привода подобна работе по набору на компьютере книги на английском языке: вам не обязательно понимать, о чем эта книга, чтобы выполнить работу. Не надо даже понимать английский язык. Задача для жесткого диска напоминает чтение немецкой книги и ее набор на компьютере в виде перевода на английский язык. В этом случае вам необходимо знать оба языка и понимать содержание книги.

Библиотека libata справляется с задачей и дает возможность подключить подсистему SCSI для устройств с интерфейсами ATA/SATA. Как правило, оказывается вовлеченным большее количество драйверов, а не всего лишь один ведущий драйвер SATA, как показано на рис. 3.2. Остальные драйверы не показаны в целях упрощения схемы.

3.6.3. Обобщенные устройства SCSI

Процесс из пространства пользователя взаимодействует с подсистемой SCSI с помощью слоя блочных устройств и/или другой службы ядра, расположенной над драйвером класса устройств SCSI (например, `sd` или `sr`). Другими словами, большинству пользовательских процессов нет нужды знать что-либо об устройствах SCSI или об их командах.

Тем не менее пользовательские процессы могут обходить драйверы классов устройств и отправлять команды протокола SCSI напрямую устройствам с помощью *обобщенных устройств*. Посмотрите, например, на систему, описанную ранее в разделе. Но сейчас взгляните на то, что произойдет, когда вы добавите параметр `-g` в команду `lsscsi`, чтобы отобразить обобщенные устройства:

```
$ lsscsi -g
[0:0:0:0]    disk    ATA      WDC WD3200AAJS-2 01.0  /dev/sda ❶/dev/sg0
[1:0:0:0]    cd/dvd  Slimtype DVD A  DS8A5SH   XA15  /dev/sr0 /dev/sg1
[2:0:0:0]    disk    USB2.0   CardReader CF    0100  /dev/sdb /dev/sg2
[2:0:0:1]    disk    USB2.0   CardReader SM XD 0100  /dev/sdc /dev/sg3
[2:0:0:2]    disk    USB2.0   CardReader MS    0100  /dev/sdd /dev/sg4
[2:0:0:3]    disk    USB2.0   CardReader SD    0100  /dev/sde /dev/sg5
[3:0:0:0]    disk    FLASH    Drive UT_USB20   0.00  /dev/sdf /dev/sg6
```

В дополнение к обычному файлу блочного устройства в каждой строке указан файл обобщенного SCSI-устройства (отмечен символом ❶). Так, обобщенным устройством для оптического привода `/dev/sr0` является `/dev/sg1`.

Зачем может понадобиться обобщенное SCSI-устройство? Ответ обусловлен сложностью кода ядра. Когда задачи становятся более тяжелыми, лучше их вывести за пределы ядра. Представьте запись и чтение CD/DVD. Чтение происходит существенно проще записи, при нем не затрагиваются важные службы ядра. Программа в пространстве пользователя выполнила бы запись чуть менее эффективно, чем служба ядра, однако такую программу гораздо проще создать и поддерживать, чем службу ядра, а ошибки в ней не затронут пространство ядра. Следовательно, чтобы записать оптический диск в системе Linux, мы запускаем программу, которая «разговаривает» с обобщенным SCSI-устройством, таким как `/dev/sg1`. Однако благодаря простоте чтения, по сравнению с записью, считывание с устройства происходит с помощью специального драйвера `sr` в ядре.

3.6.4. Методы коллективного доступа к одному устройству

На рис. 3.3 для SCSI-подсистемы Linux показаны две точки доступа (`sr` и `sg`) к оптическому приводу из пространства пользователя (опущены все драйверы, которые

расположены под самым нижним уровнем SCSI). Процесс А осуществляет чтение с помощью драйвера `sr`, а процесс Б производит запись с помощью драйвера `sg`. Однако такие процессы не могут одновременно получать доступ к одному устройству.

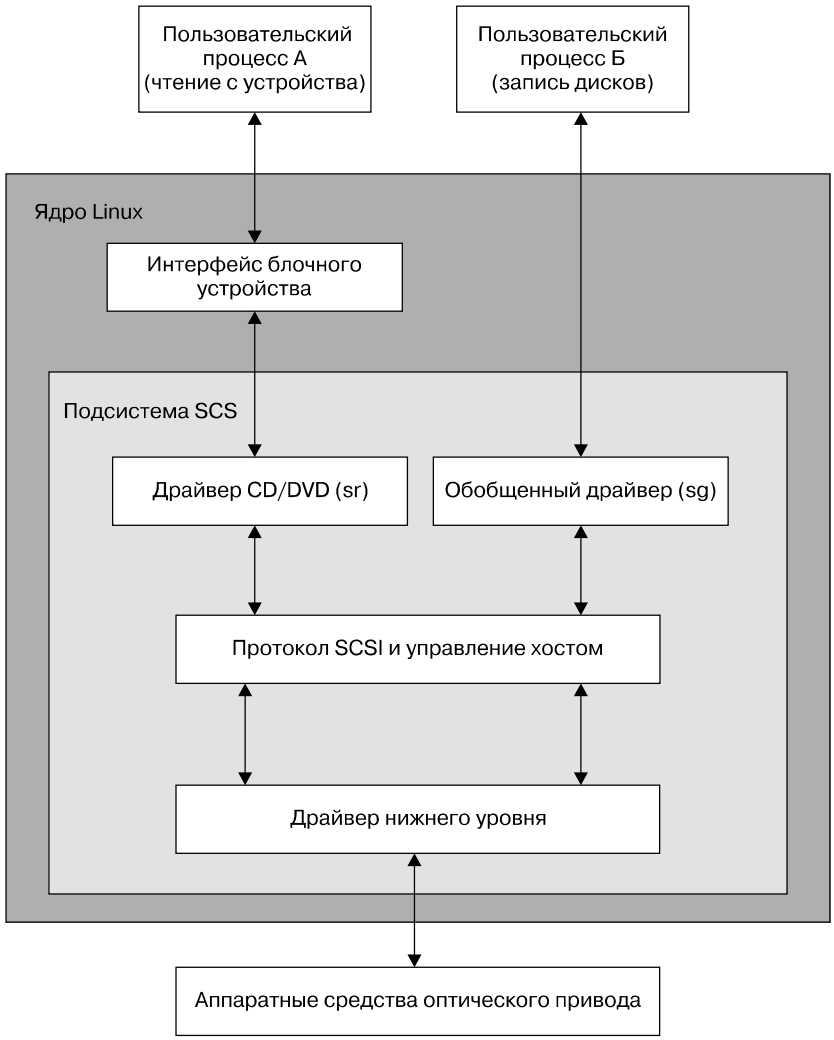

Рис. 3.3. Схема драйверов оптического привода

На рис. 3.3 процесс А осуществляет чтение с блочного устройства. Однако действительно ли пользовательские процессы считывают данные подобным образом? Ответ, как правило, отрицательный: нет, напрямую не считывают. Над блочными устройствами есть дополнительные слои, а для жестких дисков — также и дополнительные точки доступа, как вы узнаете из следующей главы.