# Глава 5. Управление вводом-выводом в операционных системах

Побудительной причиной, в конечном итоге приведшей разработчиков к созданию системного программного обеспечения, в том числе операционных систем, стала необходимость предоставить программам средства обмена данными с внешними устройствами, которые бы не требовали непосредственного включения в каждую программу двоичного кода, управляющего устройствами ввода-вывода. Напомним, что программирование ввода-вывода является наиболее сложным и трудоемким, требующим очень высокой квалификации. Поэтому код, реализующий операции ввода-вывода, сначала стали оформлять в виде системных библиотечных процедур, а потом и вовсе вывели из систем программирования, включив в операционную систему. Это позволило не писать такой код в каждой программе, а только обращаться к нему — системы программирования стали генерировать обращения к системному коду ввода-вывода. Таким образом, управление вводом-выводом — это одна из основных функций любой операционной системы.

С одной стороны, организация ввода-вывода в различных операционных системах имеет много общего. С другой стороны, реализация ввода-вывода в ОС так сильно отличается от системы к системе, что очень нелегко выделить и описать именно основные принципы реализации этих функций. Проблема усугубляется еще и тем, что в большинстве ныне используемых систем эти моменты вообще, как правило, подробно не описаны (исключением являются только системы Linux и FreeBSD, для которых имеются комментированные исходные тексты), а детально описываются только функции API, реализующие ввод-вывод. Другими словами, для тех же систем Windows от компании Microsoft мы воспринимаем подсистему ввода-вывода как «черный ящик». Известно, как можно и нужно использовать эту подсистему, но детали ее внутреннего устройства остаются неизвестными. Поэтому в данной главе мы рассмотрим только основные идеи и концепции. Наконец, поскольку такой важный ресурс, как внешняя память, в основном реализуется на устройствах ввода-вывода с прямым доступом, а к ним, прежде всего, относятся накопители на магнитных дисках, мы также рассмотрим логическую структуру дис-

ка, начальную стадию процесса загрузки операционной системы, кэширование операций ввода-вывода, оптимизацию дисковых операций.

# Основные концепции организации ввода-вывода в операционных системах

Как известно, ввод-вывод считается одной из самых сложных областей проектирования операционных систем, в которой сложно применить общий подход и в которой изобилуют частные методы. В действительности, источником сложности является огромное число устройств ввода-вывода разнообразной природы, которые должна поддерживать операционная система. При этом перед создателями операционной системы встает очень непростая задача — не только обеспечить эффективное управление устройствами ввода-вывода, но и создать удобный и эффективный виртуальный интерфейс устройств ввода-вывода, позволяющий прикладным программистам просто считывать или сохранять данные, не обращая внимание на специфику устройств и проблемы распределения устройств между выполняющимися задачами. Система ввода-вывода, способная объединить в одной модели широкий спектр устройств, должна быть универсальной. Она должна учитывать потребности существующих устройств, от простой мыши до клавиатур, принтеров, графических дисплеев, дисковых накопителей, компакт-дисков и даже сетей. С другой стороны, необходимо обеспечить доступ к устройствам ввода-вывода для множества параллельно выполняющихся задач, причем так, чтобы они как можно меньше мешали друг другу.

Поэтому самым главным является следующий принцип: *любые операции по управлению вводом-выводом объявляются привилегированными и могут выполняться только кодом самой операционной системы.* Для обеспечения этого принципа в большинстве процессоров даже вводятся *режимы пользователя* и *супервизора.* Последний еще называют *привилегированным режимом,* или *режимом ядра.* Как правило, в режиме *супервизора* выполнение команд ввода-вывода разрешено, а в пользовательском режиме — запрещено. Обращение к командам ввода-вывода в пользовательском режиме вызывает *исключение*[1], и управление через механизм прерываний передается коду операционной системы. Хотя возможны и более сложные схемы, в которых в ряде случаев пользовательским программам может быть разрешено непосредственное выполнение команд ввода-вывода.

Еще раз подчеркнем, что мы, прежде всего, говорим о мультипрограммных операционных системах, для которых существует проблема разделения ресурсов, и одним из основных видов ресурсов являются устройства ввода-вывода и соответствующее программное обеспечение, с помощью которого осуществляется обмен данными между внешними устройствами и оперативной памятью. Помимо разделяемых устройств ввода-вывода (эти устройства допускают разделение посредством механизма доступа) существуют неразделяемые устройства. Примерами

[1] Исключение — это определенный вид внутреннего прерывания. Этим термином, во-первых, обозначают некоторое множество синхронных прерываний, а во-вторых, подчеркивают, что ситуация, вызвавшая запрос на прерывание, является исключительной, то есть отличается от обычной.

разделяемого устройства могут служить накопитель на магнитных дисках, устройство чтения компакт-дисков. Это устройства с прямым доступом. Примеры неразделяемых устройств — принтер, накопитель на магнитных лентах. Это устройства с последовательным доступом. Операционные системы должны управлять и теми, и другими, предоставляя возможность параллельно выполняющимся задачам их использовать.

Можно назвать три основные причины, по которым нельзя разрешать каждой отдельной пользовательской программе обращаться к внешним устройствам непосредственно.

* *Необходимость разрешать возможные конфликты в доступе к устройствам ввода-вывода.* Например, пусть две параллельно выполняющиеся программы пытаются вывести на печать результаты своей работы. Если не предусмотреть внешнего управления устройством печати, то в результате мы можем получить абсолютно нечитаемый текст, так как каждая программа будет время от времени выводить свои данные, перемежающиеся с данными от другой программы. Либо можно взять ситуацию, когда для одной программы необходимо прочитать данные с одного сектора магнитного диска, а для другой записать результаты в другой сектор того же накопителя. Если операции ввода-вывода не будут отслеживаться каким-то третьим (внешним) процессом-арбитром, то после позиционирования магнитной головки для первой задачи может тут же прийти команда позиционирования головки для второй задачи, и обе операции ввода-вывода не смогут выполниться корректно.
* *Желание увеличить эффективность использования ресурсов ввода-вывода.* Например, у накопителя на магнитных дисках время подвода головки чтения/записи к необходимой дорожке и время обращения к определенному сектору могут значительно (до тысячи раз) превышать время пересылки данных. В результате, если задачи по очереди обращаются к цилиндрам, далеко отстоящим друг от друга, то полезная работа, выполняемая накопителем, может быть существенно снижена.
* *Необходимость избавить программы ввода-вывода от ошибок.* Ошибки в программах ввода-вывода могут привести к краху всех вычислительных процессов, ибо часть операций ввода-вывода требуются самой операционной системе. В ряде операционных систем системный ввод-вывод имеет существенно более высокие привилегии, чем ввод-вывод задач пользователя. Поэтому системный код, управляющий операциями ввода-вывода, очень тщательно отлаживается и оптимизируется для повышения надежности вычислений и эффективности использования оборудования.

Итак, управление вводом-выводом осуществляется компонентом операционной системы, который часто называют *супервизором ввода-вывода.* Перечислим основные задачи, возлагаемые на супервизор.

1. Модуль супервизора операционной системы, иногда называемый *супервизором задач,* получает запросы от прикладных задач на выполнение тех или иных операций, в том числе на ввод-вывод. Эти запросы проверяются на корректность и, если они соответствуют спецификациям и не содержат ошибок, то обрабатыва-

ются дальше. В противном случае пользователю (задаче) выдается соответствующее диагностическое сообщение о недействительности (некорректности) запроса.

2. Супервизор ввода-вывода получает запросы на ввод-вывод от супервизора задач или от программных модулей самой операционной системы.
3. Супервизор ввода-вывода вызывает соответствующие распределители каналов и контроллеров, планирует ввод-вывод (определяет очередность предоставления устройств ввода-вывода задачам, затребовавшим эти устройства). Запрос на ввод-вывод либо тут же выполняется, либо ставится в очередь на выполнение.
4. Супервизор ввода-вывода инициирует операции ввода-вывода (передает управление соответствующим драйверам) и в случае управления вводом-выводом с использованием прерываний предоставляет процессор диспетчеру задач с тем, чтобы передать его первой задаче, стоящей в очереди на выполнение.
5. При получении сигналов прерываний от устройств ввода-вывода супервизор идентифицирует эти сигналы (см. раздел «Прерывания» в главе 1) и передает управление соответствующим программам обработки прерываний.
6. Супервизор ввода-вывода осуществляет передачу сообщений об ошибках, если таковые происходят в процессе управления операциями ввода-вывода.
7. Супервизор ввода-вывода посылает сообщения о завершении операции ввода-вывода запросившей эту операцию задаче и снимает ее с состояния ожидания ввода-вывода, если задача ожидала завершения операции.

В случае, если устройство ввода-вывода является *инициативным*[1], управление со стороны супервизора ввода-вывода будет заключаться в активизации соответствующего вычислительного процесса (перевод его в состояние готовности к выполнению).

Таким образом, прикладные программы (а в общем случае — все обрабатывающие программы) не могут непосредственно связываться с устройствами ввода-вывода независимо от того, в каком режиме используются эти устройства (монопольно или совместно), но, установив соответствующие значения параметров в *запросе на ввод-вывод,* определяющие требуемую операцию и количество потребляемых ресурсов, обращаются к супервизору задач. Последний передает управление супервизору ввода-вывода, который и запускает необходимые логические и физические операции.

Инициативным называют такое устройство ввода-вывода, по сигналу прерывания от которого запускается соответствующая ему программа (обычно это не стандартное устройство ввода-вывода, а набор датчиков) Такая программа, с одной стороны, не является драйвером, поэтому ей не нужно управлять операциями обмена данными, но, с другой стороны, запуск такой программы осуществляется именно по событиям, связанным с генерацией устройством ввода-вывода соответствующего сигнала Разница между драйверами, работающими по прерываниям, и инициативными программами заключается в их статусе Драйвер является компонентом операционной системы и часто выполняется не как вычислительным процесс, а как системный объект, а инициативная программа является обычным вычислительным процессом, только его запуск осуществляется по инициативе внешнего устройства

Упомянутый выше запрос на ввод-вывод должен удовлетворять требованиям API той операционной системы, в среде которой выполняется приложение. Параметры, которые указываются в запросах на ввод-вывод, передаются не только в вызывающих последовательностях, создаваемых по спецификациям API, но и как данные, хранящиеся в соответствующих системных таблицах. Все параметры, которые будут стоять в вызывающей последовательности, предоставляются компилятором и отражают требования программиста, а также постоянные сведения об операционной системе и архитектуре компьютера в целом. Переменные сведения о вычислительной системе (ее конфигурация, состав оборудования, состав и особенности системного программного обеспечения) содержатся в специальных системных таблицах. Процессору, каналам прямого доступа в память и контроллерам необходимо передавать конкретную двоичную информацию, с помощью которой и осуществляется управление оборудованием. Эта конкретная двоичная информация в виде кодов и данных часто готовится с помощью препроцессоров, но часть ее хранится в системных таблицах.

# Режимы управления вводом-выводом

Как известно, имеется два основных режима ввода-вывода: *режим обмена с опросом готовности* устройства ввода-вывода и *режим обмена с прерываниями* (рис. 5. 1).

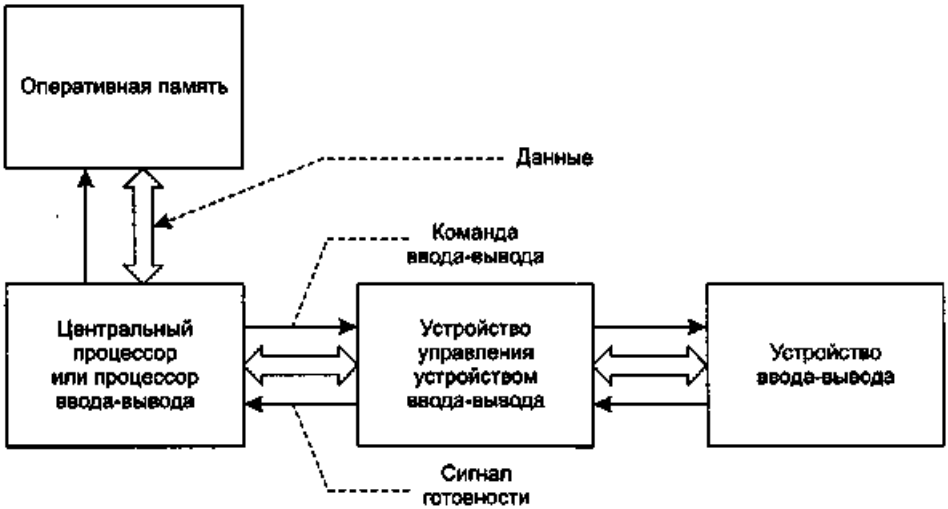



**Рис.** 5. 1. Управление вводом-выводом

Пусть для простоты рассмотрения этих вопросов управление вводом-выводом осуществляет центральный процессор. В этом случае часто говорят о работе *программного канала* обмена данными между внешними устройством и оперативной памятью (в отличие от *канала прямого доступа к памяти,* при котором управление вводом-выводом осуществляет специальное дополнительное оборудование). Итак, пусть центральный процессор посылает команду устройству управления,

требующую, чтобы устройство ввода-вывода выполнило некоторое действие. Например, если мы управляем дисководом, то это может быть команда на включение двигателя или команда, связанная с позиционированием магнитных головок. Устройство управления исполняет команду, транслируя сигналы, понятные ему и центральному устройству, в сигналы, понятные устройству ввода-вывода. После выполнения команды устройство ввода-вывода (или его устройство управления) выдает *сигнал готовности,* который сообщает процессору о том, что можно выдать новую команду для продолжения обмена данными. Однако поскольку быстродействие устройства ввода-вывода намного меньше быстродействия центрального процессора (порой на несколько порядков), то сигнал готовности приходится очень долго ожидать, постоянно опрашивая соответствующую линию интерфейса на наличие или отсутствие нужного сигнала. Посылать новую команду, не дождавшись сигнала готовности, сообщающего об исполнении предыдущей команды, бессмысленно. В режиме опроса готовности драйвер, управляющий процессом обмена данными с внешним устройством, как раз и выполняет в цикле команду «проверить наличие сигнала готовности». До тех пор пока сигнал готовности не появится, драйвер ничего другого не делает. При этом, естественно, нерационально используется время центрального процессора. Гораздо выгоднее, выдав команду ввода-вывода, на время забыть об устройстве ввода-вывода и перейти на выполнение другой программы. А появление сигнала готовности трактовать как запрос на прерывание от устройства ввода-вывода. Именно эти сигналы готовности и являются *сигналами запроса на прерывание* (см. раздел «Прерывания» в главе 1).

Режим обмена с прерываниями по своей сути является режимом асинхронного управления. Для того чтобы не потерять связь с устройством (после выдачи процессором очередной команды по управлению обменом данными и переключения его на выполнение других программ), может быть запущен отсчет времени, в течение которого устройство обязательно должно выполнить команду и выдать-таки сигнал запроса на прерывание. Максимальный интервал времени, в течение которого устройство ввода-вывода или его контроллер должны выдать сигнал запроса на прерывание, часто называют *установкой тайм-аута.* Если это время истекло после выдачи устройству очередной команды, а устройство так и не ответило, то делается вывод о том, что связь с устройством потеряна и управлять им больше нет возможности. Пользователь и/или задача получают соответствующее диагностическое сообщение.

Драйверы, работающие в режиме прерываний, представляют собой сложный комплекс программных модулей и могут иметь несколько секций: *секцию запуска,* одну или несколько *секций продолжения* и *секцию завершения.*

Секция запуска инициирует операцию ввода-вывода. Эта секция запускается для включения устройства ввода-вывода или просто для инициализации очередной операции ввода-вывода.

Секция продолжения (их может быть несколько, если алгоритм управления обменом данными сложный, и требуется несколько прерываний для выполнения одной логической операции) осуществляет основную работу по передаче данных. Секция продолжения, собственно говоря, и является основным обработчиком пре-

рывания. Поскольку используемый интерфейс может потребовать для управления вводом-выводом несколько последовательностей управляющих команд, а сигнал прерывания у устройства, как правило, только один, то после выполнения очередной секции прерывания супервизор прерываний при следующем сигнале готовности должен передать управление другой секции. Это делается путем изменения адреса обработки прерывания после выполнения очередной секции, а если имеется только одна секция продолжения, она сама передает управление в ту или иную часть кода подпрограммы обработки прерывания.

Секция завершения обычно выключает устройство ввода-вывода или просто завершает операцию.

Управление операциями ввода-вывода в режиме прерываний требует значительных усилий со стороны системных программистов — такие программы создавать сложнее. Примером тому может служить существующая ситуация с драйверами печати. Так, в операционных системах Windows (и Windows 9x, и Windows NT/2000) печать через параллельный порт осуществляется не в режиме с прерываниями, как это сделано в других ОС, а в режиме опроса готовности, что приводит к 100-процентной загрузке центрального процессора на все время печати. При этом, естественно, выполняются и другие задачи, запущенные на исполнение, но исключительно за счет того, что упомянутые операционные системы поддерживают вытесняющую мультизадачность, время от времени прерывая процесс управления печатью и передавая центральный процессор остальным задачам.

## Закрепление устройств, общие устройства ввода-вывода

Как известно, многие устройства и, прежде всего, устройства с последовательным доступом не допускают совместного использования. Такие устройства могут стать *закрепленными* за процессом, то есть их можно предоставить некоторому вычислительному процессу на все время жизни этого процесса. Однако это приводит к тому, что вычислительные процессы часто не могут выполняться параллельно — они ожидают освобождения устройств ввода-вывода. Чтобы организовать совместное использование многими параллельно выполняющимися задачами тех устройств ввода-вывода, которые не могут быть разделяемыми, вводится понятие *виртуальных устройств*. Принцип виртуализации позволяет повысить эффективность вычислительной системы.

Вообще говоря, понятие виртуального устройства шире, нежели понятие *спулинга* (spooling — Simultaneous Peripheral Operation On-Line, то есть имитация работы с устройством в режиме непосредственного подключения к нему). Основное назначение спулинга — создать видимость разделения устройства ввода-вывода, которое фактически является устройством с последовательным доступом и должно использоваться только монопольно и быть закрепленным за процессом. Например, мы уже говорили, что в случае, когда несколько приложений должны выводить на печать результаты своей работы, если разрешить каждому такому приложению печатать строку по первому же требованию, то это приведет к потоку строк,

не представляющих никакой ценности. Однако если каждому вычислительному процессу предоставлять не реальный, а виртуальный принтер, и поток выводимых символов (или управляющих кодов для их печати) сначала направлять в специальный файл на диске (так называемый *спул-файл* — spool-file) и только потом, по окончании виртуальной печати, в соответствии с принятой дисциплиной обслуживания и приоритетами приложений выводить содержимое спул-файла на принтер, то все результаты работы можно будет легко читать. Системные процессы, которые управляют спул-файлом, называются *спулером чтения* (spool-reader) или *спулером записи* (spool-writer).

Достаточно рационально организована работа с виртуальными устройствами в системах Windows 9x/NT/2000/XP компании Microsoft. В качестве примера можно кратко рассмотреть подсистему печати. Microsoft различает термины «принтер» и «устройство печати». Принтер — это некоторая виртуализация, объект операционной системы, а устройство печати — это физическое устройство, которое может быть подключено к компьютеру. Принтер может быть *локальным* или *сетевым*.

При установке локального принтера в операционной системе создается новый объект, связанный с реальным устройством печати через тот или иной интерфейс. Интерфейс может быть и сетевым, то есть передача управляющих кодов в устройство печати может осуществляться через локальную вычислительную сеть, однако принтер все равно будет считаться локальным.

Локальность принтера означает, что его спул-файл будет находиться на том же компьютере, что и принтер. Если же некоторый локальный принтер предоставить в сети в общий доступ с теми или иными разрешениями, то для других компьютеров и их пользователей он может стать *сетевым*. Компьютер, на котором имеется локальный принтер, предоставленный в общий доступ, называется *принт-сервером*.

Для получения управляющих кодов принтера устанавливается программное обеспечение (компания Microsoft называет его высокоуровневым драйвером, хотя правильнее было бы называть его иначе: например, препроцессором). Эти управляющие коды посылаются на устройство печати по соответствующему интерфейсу через назначенные принтеру порты и управляют работой устройства печати. При получении операционной системой от приложения запроса на печать она выделяет для этого процесса виртуальный принтер. Можно сказать, что операционная система закрепляет за процессом виртуальный принтер, но никак не устройство печати. Обработанные драйвером принтера данные, посланные на него из приложения, как правило (по умолчанию), направляются в спул-файл, откуда они затем передаются на печать по мере освобождения устройства печати и в соответствии с приоритетом локального принтера. При установке сетевого принтера операционная система устанавливает для этого объекта высокоуровневый драйвер и связывает полученный объект со спулером того компьютера, на котором установлен соответствующий локальный принтер.

Локальных принтеров, связанных с конкретным устройством печати, на компьютере может быть несколько. Каждому локальному принтеру можно назначить тот или иной приоритет, который будет учитываться при формировании очереди пе-

чати в процессе работы спулера. В результате каждый процесс может послать на печать свои данные и не связывать реальное выполнение некоторого задания на печать с занятостью или освобождением самого устройства печати. Приоритетность в печати определяется приоритетом того локального или сетевого принтера, к которому обратилось приложение.

## Основные системные таблицы ввода-вывода

Для управления всеми операциями ввода-вывода и отслеживания состояния всех ресурсов, занятых в обмене данными, операционная система должна иметь соответствующие информационные структуры. Эти информационные структуры, прежде всего, призваны отображать следующую информацию:

* состав устройств ввода-вывода и способы их подключения;
* аппаратные ресурсы, закрепленные за имеющимися в системе устройствами ввода-вывода;
* логические (символьные) имена устройств ввода-вывода, используя которые вычислительные процессы могут запрашивать те или иные операции ввода-вывода;
* адреса размещения драйверов устройств ввода-вывода и области памяти для хранения текущих значений переменных, определяющих работу с этими устройствами;
* области памяти для хранения информации о текущем состоянии устройства ввода-вывода и параметрах, определяющих режимы работы устройства;
* данные о текущем процессе, который работает с данным устройством;
* адреса тех областей памяти, которые содержат данные, собственно и участвующие в операциях ввода-вывода (получаемые при операциях ввода данных и выводимые на устройство при операциях вывода данных).

Эти информационные структуры часто называют таблицами ввода-вывода, хотя они, в принципе, могут быть организованы и в виде списков.

Каждая операционная система ведет свои таблицы ввода-вывода, их состав (и количество, и назначение каждой таблицы) может сильно отличаться. В некоторых операционных системах вместо таблиц создаются списки, хотя использование статических структур данных для организации ввода-вывода, как правило, приводит к более высокому быстродействию. Здесь очень трудно вычленить общие составляющие, тем более что для современных операционных систем подробной документации на эту тему крайне мало, разве что воспользоваться материалами ныне устаревших ОС. Тем не менее попытаемся это сделать, опираясь на идеи семейства простых, но эффективных операционных систем реального времени, разработанных фирмой Hewlett Packard для своих мини-ЭВМ.

Исходя из принципа управления вводом-выводом исключительно через супервизор операционной системы и учитывая, что драйверы устройств ввода-вывода ис-

пользуют механизм прерываний для установления обратной связи центральной части с внешними устройствами, можно сделать вывод о необходимости создания по крайней мере трех системных таблиц.

Первая таблица (или список) содержит информацию обо всех устройствах ввода-вывода, подключенных к вычислительной системе. Назовем ее условно *таблицей оборудования* (equipment table), а каждый элемент этой таблицы пусть называется *UCB* (Unit Control Block — блок управления устройством ввода-вывода). Каждый элемент UCB таблицы оборудования, как правило, содержит следующую информацию об устройстве:

* тип устройства, его конкретная модель, символическое имя и характеристики устройства;
* способ подключения устройства (через какой интерфейс, к какому разъему, какие порты и линия запроса прерывания используются и т. д.);
* номер и адрес канала (и подканала), если такие используются для управления устройством;
* информация о драйвере, который должен управлять этим устройством, адреса секции запуска и секции продолжения драйвера;
* информация о том, используется или нет буферизация при обмене данными с устройством, «имя» (или просто адрес) буфера, если такой выделяется из системной области памяти;
* установка тайм-аута и ячейки для счетчика тайм-аута;
* состояние устройства;
* поле указателя для связи задач, ожидающих устройство;
* возможно, множество других сведений.

Поясним перечисленное. Поскольку во многих операционных системах драйверы могут обладать свойством *реентерабельности* (напомним, это означает, что один и тот же экземпляр драйвера может обеспечить параллельное обслуживание сразу нескольких однотипных устройств), то в элементе UCB должна храниться либо непосредственно сама информация о текущем состоянии устройства и сами переменные для реентерабельной обработки, либо указание на место, где такая информация может быть найдена. Наконец, важнейшим компонентом элемента таблицы оборудования является указатель на дескриптор той задачи, которая в настоящий момент использует данное устройство. Если устройство свободно, то поле указателя будет иметь нулевое значение. Если же устройство уже занято и рассматриваемый указатель не нулевой, то новые запросы к устройству фиксируются посредством образования списка из дескрипторов задач, ожидающих данное устройство.

Вторая таблица предназначена для реализации еще одного принципа виртуализации устройств ввода-вывода — принципа независимости от устройства. Желательно, чтобы программисту не приходилось учитывать конкретные параметры (и/или возможности) того или иного устройства ввода-вывода, которое установлено (или не установлено) в компьютер. Для него должны быть важными только самые общие возможности, характерные для данного класса устройств ввода-вывода. Например,

принтер должен уметь выводить (печатать) символы или графические изображения. А накопитель на магнитных дисках — считывать или записывать порцию данных по указанному адресу, то есть в координатах C-H-S (Cylinder-Head-Sector — номера цилиндра, головки и сектора) или по порядковому номеру блока данных. Хотя чаще всего программист и не использует прямую адресацию при работе с магнитными дисками, а работает на уровне файловой системы (см. главу 6). Однако в таком случае уже разработчики системы управления файлами не должны зависеть от того, каких типа и модели накопитель используется в данном компьютере и кто является его производителем (например, HDD IBM IC35L 120AVV207-0, WD1200JB или еще какой-нибудь). Важным должен быть только сам факт существования накопителя, имеющего некоторое количество цилиндров, головок чтения-записи и секторов на дорожке магнитного диска. Упомянутые значения количества цилиндров, головок и секторов должны быть взяты из элемента таблицы оборудования. При этом для программиста также не должно иметь значения, каким образом то или иное устройство подключено к вычислительной системе. Поэтому в запросе на ввод-вывод программист указывает именно *логическое имя устройства*. Действительное устройство, которое сопоставляется виртуальному (логическому), выбирается супервизором с помощью описываемой таблицы.

Итак, способ подключения устройства, его конкретная модель и соответствующий ей драйвер содержатся в уже рассмотренной таблице оборудования. Но для того чтобы связать некоторое виртуальное устройство, использованное программистом, с системной таблицей, отображающей информацию о том, какое конкретно устройство и каким образом подключено к компьютеру, требуется вторая системная таблица. Назовем ее условно *таблицей виртуальных логических устройств* (Device Reference Table, DRT). Назначение этой второй таблицы — установление связи между виртуальными (логическими) устройствами и реальными устройствами, описанными посредством первой таблицы (таблицы оборудования). Другими словами, вторая таблица позволяет супервизору перенаправить запрос на ввод-вывод из приложения в те программные модули и структуры данных, которые (или адреса которых) хранятся в соответствующем элементе первой таблицы. Во многих многопользовательских системах таких таблиц несколько: одна общая и по одной на каждого пользователя, что позволяет строить необходимые связи между логическими устройствами (символьными именами устройств) и реальными физическими устройствами, которые имеются в системе.

Наконец, третья таблица — *таблица прерываний* — необходима для организации обратной связи между центральной частью и устройствами ввода-вывода. Эта таблица указывает для каждого сигнала запроса на прерывание тот элемент UCB, который сопоставлен данному устройству. Каждое устройство либо имеет свою линию запроса на прерывание, либо разделяет линию запроса на прерывание с другими устройствами, но при этом имеется механизм второго уровня адресации устройств ввода-вывода. Таким образом, таблица прерываний отображает связи между сигналами запроса на прерывания и самими устройствами ввода-вывода. Как и системная таблица ввода-вывода, таблица прерываний в явном виде может и не присутствовать. Другими словами, можно сразу из основной таблицы прерываний

компьютера передать управление на программу обработки (драйвер), связанную с элементом UCB. Важно наличие связи между сигналами прерываний и таблицей оборудования.

В ряде сложных операционных систем, а к ним следует отнести все современные 32-разрядные системы для персональных компьютеров, имеется гораздо больше системных таблиц или списков, используемых для организации управления операциями ввода-вывода. Например, одной из возможных и часто реализуемых информационных структур, сопровождающих практически каждый запрос на ввод-вывод, является *блок управления данными* (Data Control Block, DCB). Назначение DCB — подключение препроцессоров к процессу подготовки данных на ввод-вывод, то есть учет конкретных технических характеристик и используемых преобразований. Это необходимо для того, чтобы имеющееся устройство получало не какие-то непонятные ему коды или форматы данных, не соответствующие режиму его работы, а коды и форматы, созданные специально под данное устройство. Теперь такие препроцессоры часто называют высокоуровневыми драйверами, или просто драйверами, хотя изначально под термином «драйвер» подразумевалась программа управления операциями ввода-вывода.

Взаимосвязи между описанными таблицами изображены на рис. 5.2.

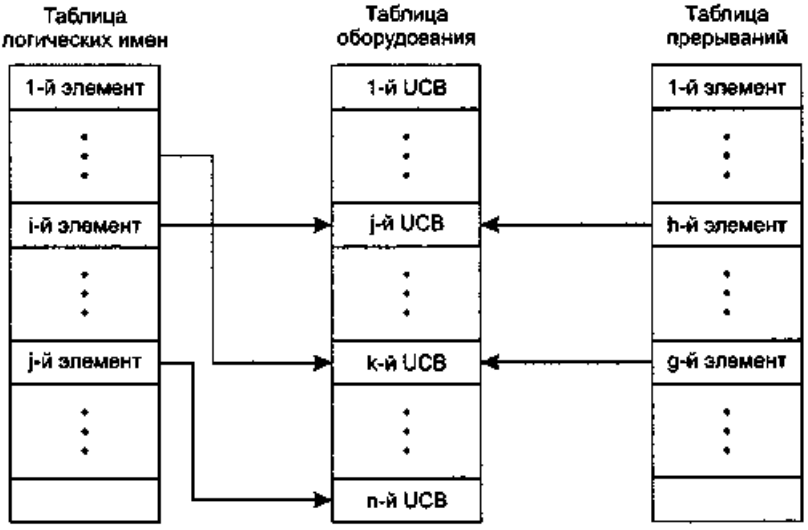



**Рис. 5. 2.** Взаимосвязи системных таблиц ввода-вывода

Нам осталось рассмотреть процесс управления вводом-выводом еще раз, теперь с учетом изложенных принципов (рис. 5.3).

Запрос на операцию ввода-вывода от выполняющейся программы поступает на супервизор задач (шаг 1). Этот запрос представляет собой обращение к операционной системе и указывает на конкретную функцию API. Вызов сопровождается некоторыми параметрами, уточняющими требуемую операцию. Модуль операционной системы, принимающий от задач запросы на те или иные действия, часто

называют *супервизором задач*. Не следует путать его с диспетчером задач. Супервизор задач проверяет системный вызов на соответствие принятым спецификациям и в случае ошибки возвращает задаче соответствующее сообщение (шаг 1-1). Если же запрос корректен, то он перенаправляется в *супервизор ввода-вывода* (шаг 2). Последний по логическому (виртуальному) имени с помощью таблицы DRT находит соответствующий элемент UCB в таблице оборудования. Если устройство уже занято, то описатель задачи, запрос которой обрабатывается супервизором ввода-вывода, помещается в список задач, ожидающих это устройство. Если же устройство свободно, то супервизор ввода-вывода определяет из UCB тип устройства и при необходимости запускает препроцессор, позволяющий получить последовательность управляющих кодов и данных, которую сможет правильно понять и отработать устройство (шаг 3). Когда «программа» управления операцией ввода-вывода будет готова, супервизор ввода-вывода передает управление соответствующему драйверу на секцию запуска (шаг 4). Драйвер инициализирует операцию управления, обнуляет счетчик тайм-аута и возвращает управление супервизору (диспетчеру задач) с тем, чтобы он поставил на процессор готовую к исполнению задачу (шаг 5). Система работает своим чередом, но когда устройство ввода-вывода отработает посланную ему команду, оно выставляет сигнал запроса на прерывание, по которому через таблицу прерываний управление передается на секцию продолжения (шаг 6). Получив новую команду, устройство вновь начинает ее обрабатывать, а управление процессором опять передается диспетчеру задач, и процессор продолжает выполнять полезную работу. Таким образом, получается параллельная обработка задач, на фоне которой процессор осуществляет управление операциями ввода-вывода.

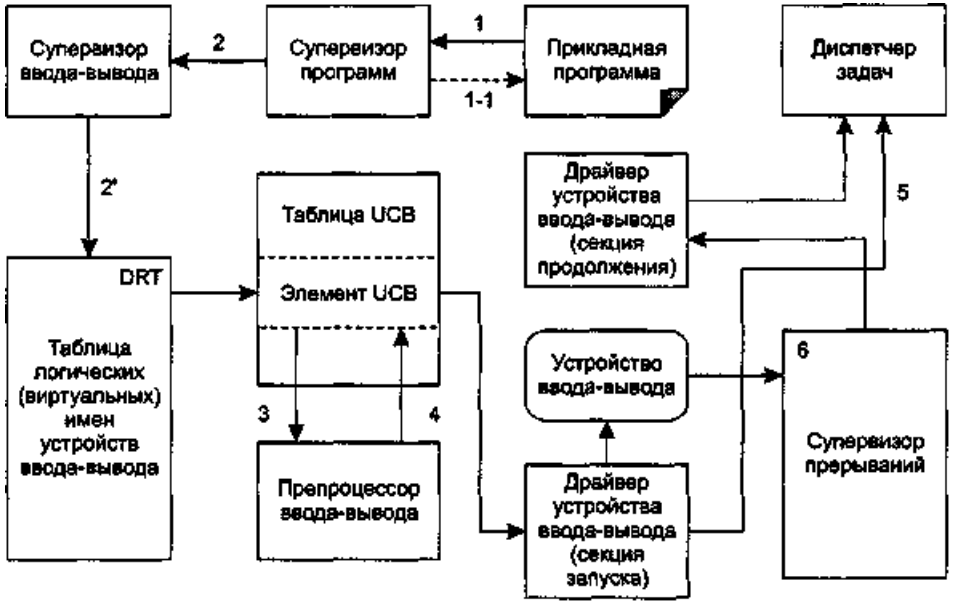



**Рис. 5.3.** Процесс управления вводом-выводом

Очевидно, что если имеются специальные аппаратные средства для управления вводом-выводом (речь идет о каналах прямого доступа к памяти), которые позволяют освободить центральный процессор от этой работы, то в функции центрального процессора будут по-прежнему входить все только что рассмотренные шаги, за исключением последнего — непосредственного управления операциями ввода-вывода. В случае использования каналов прямого доступа в память последние исполняют соответствующие канальные программы и освобождают центральный процессор от непосредственного управления обменом данными между памятью и внешними устройствами.

# Синхронный и асинхронный ввод-вывод

Задача, выдавшая запрос на операцию ввода-вывода, переводится супервизором в состояние ожидания завершения заказанной операции. Когда супервизор получает от секции завершения сообщение о том, что операция завершилась, он переводит задачу в состояние готовности к выполнению, и она продолжает выполняться. Эта ситуация соответствует *синхронному вводу-выводу.* Синхронный ввод-вывод является стандартным для большинства операционных систем. Чтобы увеличить скорость выполнения приложений, было предложено при необходимости использовать *асинхронный ввод-вывод.*

Простейшим вариантом *асинхронного вывода* является так называемый *буферизованный* вывод данных на внешнее устройство, при котором данные из приложения передаются не непосредственно на устройство ввода-вывода, а в специальный системный буфер — область памяти, отведенную для временного размещения передаваемых данных. В этом случае логически операция вывода для приложения считается выполненной сразу же, и задача может не ожидать окончания действительного процесса передачи данных на устройство. Реальным выводом данных из системного буфера занимается супервизор ввода-вывода. Естественно, что выделение буфера из системной области памяти берет на себя специальный системный процесс по указанию супервизора ввода-вывода. Итак, для рассмотренного случая вывод будет асинхронным, если, во-первых, в запросе на ввод-вывод указано на необходимость буферизации данных, а во-вторых, устройство ввода-вывода допускает такие асинхронные операции, и это отмечено в UCB.

Можно организовать и *асинхронный ввод данных.* Однако для этого необходимо не только выделять область памяти для временного хранения считываемых с устройства данных и связывать выделенный буфер с задачей, заказавшей операцию, но и сам запрос на операцию ввода-вывода разбивать на две части (на два запроса). В первом запросе указывается операция на считывание данных, подобно тому как это делается при синхронном вводе-выводе, однако тип (код) запроса используется другой, и в запросе указывается еще по крайней мере один дополнительный параметр — имя (код) системного объекта, которое получает задача в ответ на запрос и которое идентифицирует выделенный буфер. Получив имя буфера (будем так условно называть этот системный объект, хотя в различных операционных системах используются и другие термины, например «класс»), задача продолжает

свою работу. Здесь очень важно подчеркнуть, что в результате запроса на асинхронный ввод данных задача не переводится супервизором ввода-вывода в состояние ожидания завершения операции ввода-вывода, а остается в состоянии выполнения или в состоянии готовности к выполнению. Через некоторое время, выполнив необходимый код, который был определен программистом, задача выдает второй запрос на завершение операции ввода-вывода. В этом втором запросе к тому же устройству, который, естественно, имеет другой код (или имя запроса), задача указывает имя системного объекта (буфера для асинхронного ввода данных) и в случае успешного завершения операции считывания данных тут же получает их из системного буфера. Если же данные еще не успели до конца переписаться с внешнего устройства в системный буфер, супервизор ввода-вывода переводит задачу в состояние ожидания завершения операции ввода-вывода, и далее все напоминает обычный синхронный ввод данных.

Асинхронный ввод-вывод характерен для большинства мультипрограммных операционных систем, особенно если операционная система поддерживает мультизадачность с помощью механизма потоков выполнения. Однако если асинхронный ввод-вывод в явном виде отсутствует, его можно реализовать самому, организовав для вывода данных отдельный поток выполнения.

Аппаратуру ввода-вывода можно рассматривать как совокупность аппаратных процессоров, которые способны работать параллельно друг другу, а также параллельно центральному процессору (процессорам). На таких «процессорах» выполняются так называемые *внешние процессы.* Например, для печатающего устройства (внешнее устройство вывода данных) внешний процесс может представлять собой совокупность операций, обеспечивающих перевод печатающей головки, продвижение бумаги на одну позицию, смену цвета чернил или печать каких-то символов. Внешние процессы, используя аппаратуру ввода-вывода, взаимодействуют как между собой, так и с обычными «программными» процессами, выполняющимися на центральном процессоре. Важным при этом является обстоятельство, что скорости выполнения внешних процессов будут существенно (порой на порядок или больше) отличаться от скорости выполнения обычных *(внутренних)* процессов. Для своей нормальной работы внешние и внутренние процессы обязательно должны синхронизироваться. Для сглаживания эффекта значительного несоответствия скоростей между внутренними и внешними процессами используют упомянутую выше буферизацию. Таким образом, можно говорить о системе параллельных взаимодействующих процессов (см. главу 7).

Буферы (буфер) являются критическим ресурсом в отношении внутренних (программных) и внешних процессов, которые при параллельном своем развитии информационно взаимодействуют. Через буферы данные либо посылаются от некоторого процесса к адресуемому внешнему (операция вывода данных на внешнее устройство), либо от внешнего процесса передаются некоторому программному процессу (операция считывания данных). Введение буферизации как средства информационного взаимодействия выдвигает проблему управления этими системными буферами, которая решается средствами супервизорной части операционной системы. При этом на супервизор возлагаются задачи не только по выделению и освобождению буферов в системной области памяти, но и по синхронизации

процессов в соответствии с состоянием операций заполнения или освобождения буферов, а также по их ожиданию, если свободных буферов в наличии нет, а запрос на ввод-вывод требует буферизации. Обычно супервизор ввода-вывода для решения перечисленных задач использует стандартные средства синхронизации, принятые в данной операционной системе. Поэтому если операционная система имеет развитые средства для решения проблем параллельного выполнения взаимодействующих приложений и задач, то, как правило, она реализует и асинхронный ввод-вывод.

# Организация внешней памяти на магнитных дисках

Для организации внешней памяти желательно использовать относительно недорогие, но достаточно быстродействующие и емкие устройства с прямым доступом к данным. К таким устройствам, прежде всего, относятся накопители на жестких магнитных дисках (НЖМД). Нынче чаще всего такие накопители называют «винчестерами», но мы не будем употреблять это название.

Детальное изучение этих устройств выходит за рамки темы настоящей книги, в основном их изучают в рамках дисциплины «Устройства ввода-вывода». Однако поскольку большинство компьютеров имеет накопители на жестких магнитных дисках и фактически ни одна современная операционная система для повсеместно распространенных персональных компьютеров не обходится без дисковой подсистемы, мы ознакомимся с логической организацией хранения и доступа к данным в этих устройствах, причем применительно к персональным компьютерам.

Действительно, дисковая подсистема для большинства компьютеров является одной из важнейших. Именно на магнитных дисках чаще всего располагается загружаемая в компьютер операционная система, которая и обеспечивает нам удобный интерфейс для работы. Благодаря использованию систем управления файлами, данные на магнитных дисках располагаются в виде именованных наборов данных, называемых файлами. Таким образом, помимо файлов самой операционной системы, на дисках располагаются многочисленные прикладные программы и разнообразные файлы пользователей. Наконец, благодаря тому, что по сравнению с другими устройствами внешней памяти дисковые механизмы обладают большими быстродействием и вместительностью, а также средствами непосредственной (прямой) адресации блоков данных, дисковую подсистему часто используют для организации механизмов виртуальной памяти, что существенно расширяет возможности компьютера.

## Основные понятия

Из оперативной памяти в НЖМД и обратно информация передается байтами, а вот записывается на диск и считывается с него она уже последовательно (побитно). Из-за того что запись и считывание бита данных не являются абсолютно надежными операциями, информация перед записью кодируется с достаточно большой

избыточностью. Для этой цели применяют коды Рида-Соломона. Избыточное кодирование информации позволяет не только обнаруживать ошибки, но и автоматически исправлять их. Следовательно, перед тем как данные, считанные с поверхности магнитного диска, будут переданы в оперативную память, их нужно предварительно обработать (перекодировать). На эту операцию необходимо время, поэтому в ходе обработки данных быстро вращающийся диск успевает повернуться на некоторый угол, и мы можем констатировать, что на магнитном диске данные располагаются не сплошь, а порциями (блоками). Говорят, что НЖМД относится к блочным устройствам. Нельзя прочитать (или записать) байт или несколько байтов. Можно прочитать сразу только блок данных и уже потом извлекать из него нужные байты, использовать их в своих вычислениях и изменять. Записать потом данные обратно тоже можно только сразу блоком.

За счет того что при вращении диска магнитная головка, зафиксированная на некоторое время в определенном положении, образует окружность *(дорожку* — track), блоки данных на таких окружностях называют *секторами* (sectors). С некоторых пор размер сектора стал стандартным и в абсолютном большинстве случаев он равен 512 байт хранимых данных. Все сектора пронумерованы, и помимо данных пользователя на магнитных дисках размещается и служебная информация, с помощью которой можно находить искомый сектор. Служебная информация (сервоинформация), как правило, располагается в межсекторных промежутках.

Группы дорожек (треков) одного радиуса, расположенные на поверхностях магнитных дисков, образуют так называемые *цилиндры* (cylinders). Современные жесткие диски могут иметь по нескольку десятков тысяч цилиндров. Выбор конкретной дорожки в цилиндре осуществляется указанием порядкового номера той *головки* (head) *чтения/записи данных,* которая и образует эту дорожку. Таким образом, адрес конкретного блока данных указывается с помощью уже упоминавшихся трех координат C-H-S — номеров цилиндра, головки и сектора. Устройство управления НЖМД обеспечивает позиционирование блока головок на нужный цилиндр, выбирает заданную поверхность и находит требуемый сектор. Этот способ адресации нынче считается устаревшим и почти не используется. Второй способ адресации блоков данных основывается на том, что все блоки (секторы) пронумерованы.

## Логическая структура магнитного диска

Для того чтобы можно было загрузить с магнитного диска операционную систему, а уже с ее помощью организовать работу с файлами, были приняты специальные системные соглашения о структуре диска. Хранение данных на магнитном диске можно организовать различными способами. Можно поделить все дисковое пространство на несколько частей — *разделов* (partitions), а можно его и не делить. Деление НЖМД на разделы позволяет организовать на одном физическом устройстве несколько логических; в этом случае говорят о *логических дисках.* Следует, однако, заметить, что не во всех операционных системах используется понятие логического диска. Так, UNIX-системы не имеют логических дисков.

Разделение всего дискового пространства на разделы полезно по нескольким соображениям. Во-первых, это структурирует хранение данных. Например, выделе-

ние отдельного раздела под операционную систему и программное обеспечение и другого раздела под данные пользователей позволяет отделить последние от системных файлов и не только повысить надежность системы, но и сделать более удобным ее обслуживание. Во-вторых, на каждом разделе может быть организована своя файловая система, что иногда бывает необходимо. Например, при установке операционной системы Linux нужно иметь не менее двух разделов[1], поскольку файл подкачки (страничный файл) должен располагаться в отдельном разделе. Наконец, в ряде случаев на компьютере может потребоваться установка более одной операционной системы.

Для того чтобы системное программное обеспечение получило информацию о том, как организовано хранение данных на каждом конкретном накопителе, нужно разместить в одном из секторов соответствующие данные. Даже если НЖМД используется как единственный *логический диск,* все равно нужно указать, что имеется всего один диск, и его размер. Структура данных, несущая информацию о логической организации диска, вместе с небольшой программой, с помощью которой можно ее проанализировать, а также найти и загрузить в оперативную память программу загрузки операционной системы, получила название *главной загрузочной записи* (Master Boot Record, MBR). MBR располагается в самом первом секторе НЖМД, то есть в секторе с координатами 0-0-1. Программа, расположенная в MBR, носит название *внесистемного загрузчика* (Non-System Bootstrap, NSB).

Вследствие того что сектор состоит только из 512 байт и помимо программы в нем должна располагаться информация об организации диска, внесистемный загрузчик очень прост, а структура данных, называемая *таблицей разделов* (Partition Table, PT), занимает всего 64 байт. Таблица разделов располагается в MBR по смещению 0xl BE и содержит четыре элемента. Структура записи элемента таблицы разделов приведена в табл. 5. 1. Каждый элемент этой таблицы описывает один раздел, причем двумя способами: через координаты C-H-S начального и конечного секторов, а также через номер первого сектора в спецификации LBA[2] (Logical Block Addressing) и общее число секторов в разделе. Важно отметить, что каждый раздел начинается с первого сектора на заданных цилиндре и поверхности и имеет размер не менее одного цилиндра. Поскольку координаты MBR равны 0-0-1, то первый сектор первого раздела в большинстве случаев получается равным 0-1-1 (в координатах LBA это будет сектор 64).

Первым байтом в элементе таблицы разделов идет флаг *активности раздела* Boot Indicator (значение 0 — не активен, 128 ($80_{(h)}$) — активен). Он позволяет определить, является ли данный раздел системным загрузочным. В результате процесс загрузки операционной системы осуществляется путем загрузки первого сектора

[1] Практика показывает, что Linux и другие UNIX-подобные системы лучше всего устанавливать, разбив НЖМД на 6 разделов. Раздел подкачки (swap partition) служит для размещения файла подкачки. К основному (корневому) разделу, обозначаемому символом /, монтируются разделы /usr, /home, /var и /boot. Такое разбиение диска на разделы считается наиболее технологичным.

[2] Способ указания блока данных, согласно которому все секторы диска считаются пронумерованными по следующему правилу: LBA = c x H + h) x S + s - 1. Здесь H — это максимальное число рабочих поверхностей в цилиндре; S — количество секторов на одной дорожке; c, h и s — «координаты» искомого сектора.

с такого активного раздела и передачи управления на расположенную в нем программу, которая и продолжает загрузку. Активным может быть только один раздел, и это обычно проверяется программой NSB, расположенной в MBR.

**Таблица 5. 1.** Формат элемента таблицы разделов

| Название записи элемента таблицы разделов | Длина, байт |
|---|---|
| Флаг активности раздела | 1 |
| Номер головки начала раздела | 1 |
| Номера сектора и цилиндра загрузочного сектора раздела | 2 |
| Кодовый идентификатор операционной системы | 1 |
| Номер головки конца раздела | 1 |
| Номера сектора и цилиндра последнего сектора раздела | 2 |
| Младшее и старшее двухбайтовые слова относительного номера начального сектора | 4 |
| Младшее и старшее двухбайтовые слова размера раздела в секторах | 4 |

За флагом активности раздела следует байт номера головки, с которой начинается раздел. За ним следуют два байта, означающие соответственно номер сектора и номер цилиндра загрузочного сектора, где располагается первый сектор загрузчика операционной системы. Затем следует кодовый идентификатор System ID (длиной в один байт), указывающий на принадлежность данного раздела к той или иной операционной системе и на установку в этом разделе соответствующей файловой системы. Поскольку крайне сложно найти информацию по этим кодовым идентификаторам, которыми помечаются разделы дисков, в табл. 5.2 приведены не полтора десятка наиболее часто встречающихся, а все известные сигнатуры (кодовые идентификаторы).

**Таблица 5. 2.** Кодовые идентификаторы разделов диска

| Код | Описание | Код | Описание |
|---|---|---|---|
| 000h | Раздел не использован | 085h | Linux Extended, XOSL |
| 001h | FAT12 | 086h | FAT16 volume set |
| 002h | Xenix root | 087h | NTFS volume set |
| 003h | Xenix /usr | 08Ah | AiR-Boot |
| 004h | FAT16(<32Mb) | 08Bh | FAT32 volume set |
| 005h | Extended | 08Ch | FAT32 LBA volume set |
| 006h | FAT16 | 08Dh | FreeFDISKFAT12 |
| 007h | NTFS, HPFS | 08Eh | Linux LVM |
| 008h | AIX Boot | 090h | Free FDISK FAT16 (<32Mb) |
| 009h | AIX Data | 091h | Free FDISK Extended |
| 00Ah | OS/2 Boot Manager | 092h | Free FDISK FAT16 |
| 00Bh | FAT32 | 093h | Amoeba native |

| Код | Описание | Код | Описание |
|---|---|---|---|
| 00Ch | FAT32 LBA | 094h | Amoeba BBT |
| 00Eh | FAT16LBA | 095h | MIT EXOPC |
| 00Fh | Extended LBA | 097h | Free FDISK FAT32 |
| 010h | Opus | 098h | Free FDISK FAT32 LBA |
| 011h | Hidden FAT12 | 099h | DCE376 |
| 012h | Compaq Setup | 09Ah | Free FDISK FAT16 LBA |
| 013h | B-TRON | 09Bh | Free FDISK Extended LBA |
| 014h | Hidden FAT16 (<32Mb) | 09Fh | BSDI |
| 016h | Hidden FAT16 | 0A0h | Laptop hibernation |
| 017h | Hidden NTFS, HPFS | 0A1h | NEC hibernation |
| 018h | AST Windows Swap | OA5h | Free BSD, BSD/386 |
| 019h | Photon | OA6h | Open BSD |
| 01Bh | Hidden FAT32 | OA7h | NextStep |
| 01Ch | Hidden FAT32 LBA | OA8h | Apple UFS |
| 01Eh | Hidden FAT16 LBA | OA9h | Net BSD |
| 020h | OFS1 | 0Aah | Olivetti service |
| 022h | Oxygen | 0Abh | Apple Booter |
| 024h | NEC DOS | 0Aeh | ShagOS native |
| 035h | OS/2 JFS | 0Afh | ShagOS swap |
| 035h | Theos 3 x | 0B0h | BootStar Dummy |
| 039h | Theos 4 x spanned, Plan9 | 0B7h | BSDI old native |
| 03Ah | Theos 4 x 4G | 0B8h | BSDI old swap |
| 03Bh | Theos 4 x Extended | 0BBh | OS Selector |
| 03Ch | Partition Magic | 0Beh | Solaris 8 boot |
| 040h | Venix 80286 | 0C0h | CTOS, REAL/32 smal |
| 041h | Minix, PPC Boot | 0C1h | DR-DOS FAT12 |
| 042h | LinuxSwp/DR-DOS, SFS, Win2K DDM | 0C6h | DR-DOS FAT16, FAT16 set corrupt |
| 043h | LinuxNat/DR-DOS | 0C2h | Hidden Linux swap |
| 045h | Eumel/Ergos 45h, Boot-US | OC3h | Hidden Linux native |
| 046h | Eumel/Ergos 46h | OC4h | DR-DOS FAT16 (<32Mb) |
| 047h | Eumel/Ergos 47h | OC7h | Syrinx boot, NTFS set corrupt |
| 048h | Eumel/Ergos 48h | 0CBh | DR-DOS FAT32 |
| 04Dh | QNX 4 x first | 0CCh | DR-DOS FAT32 LBA |
| 04Eh | QNX 4 x second | 0CDh | CTOS memdump |
| 04Fh | QNX 4 x third, Oberon | 0Ceh | DR-DOS FAT16 LBA |
| 050h | OnTrack DM R/0, Lynx RTOS | 0D0h | REAL/32 big |

*продолжение*

**Таблица 5.2** *(продолжение)*

| Код | Описание | Код | Описание |
|---|---|---|---|
| 051h | DM6 Aux1, DM R/W | 0D1h | Multiuser DOS FAT12 |
| 052h | CP/M, Microport System V | OD4h | Multiuser DOS FAT16 (<32Mb) |
| 053h | OnTrack DM6 Aux3 | OD5h | Multiuser DOS Extended |
| 054h | OnTrack DM6 DD0 | OD6h | Multiuser DOS FAT16 |
| 055h | EZ-Drive | 0D8h | CP/M-86 |
| 056h | GoldenBow Vfeature | 0DBh | Concurrent DOS, CTOS |
| 057h | Drive Pro | 0DDh | Hidden CTOS memdump |
| 05Ch | Priam Edisk | 0DFh | DG/UX |
| 061h | Speed Stor | 0E0h | ST AVFS |
| 063h | Unix | 0E1h | Speed Stor FAT32 |
| 064h | NetWare 2.x, PC-ARMOUR | OE3h | Speed Stor R/O |
| 065h | NetWare 3.x | 0E4h | Speed Stor FAT 16 |
| 067h | Novell 67h | 0Ebh | BeOS |
| 068h | Novell 68h | 0Eeh | EFI header |
| 068h | Novell 69h | 0Efh | EFI file system |
| 070h | DiskSecure Multi-Boot | 0F0h | Linux/PA-RISC boot |
| 074h | ScramDisk | OF1h | Storage Dimensions |
| 075h | PC/AX | OF2h | DOS Secondary |
| 078h | XOSL | OF4h | Speed Stor large, Prologue sing |
| 07Eh | F.I.X | OF5h | Prologue multi |
| 080h | MINIX 1.1-1.4a | 0FBh | VMware native |
| 081h | MINIX 1.4b+, ADM | 0FCh | Vmware swap |
| 082h | Linux swap, Solaris | 0FDh | Linux RAID |
| 083h | Linux native | 0Feh | Speed Stor (>1024), LanStep |
| 084h | Hibernation, OS/2 C: Hidden | 0FFh | Xenix BBT |

Можно сказать, что таблица разделов — одна из наиболее важных структур данных на жестком диске. Если эта таблица повреждена, то не только не будет загружаться ни одна из установленных на компьютере операционных систем, но станут недоступными данные, расположенные в НЖМД, особенно если жесткий диск был разбит на несколько разделов.

Последние два байта MBR имеют значение $55AA_{(h)}$, то есть чередующиеся значения 0 и 1. Эта сигнатура выбрана для того, чтобы проверить работоспособность всех линий передачи данных. Значение $55AA_{(h)}$, присвоенное последним двум байтам, имеется во всех загрузочных секторах.

Разделы диска могут быть двух типов: *первичные* (primary) и *расширенные* (extended). Максимальное число первичных разделов равно четырем. Если первичных разделов несколько, то только один из них может быть активным. Именно загрузчику, расположенному в активном разделе, передается управление при вклю-

чении компьютера с помощью внесистемного загрузчика. Для DOS-систем и иных операционных систем, использующих спецификации DOS, остальные первичные разделы в этом случае считаются невидимыми (hidden). Так ведут себя и операционные системы Windows 9x.

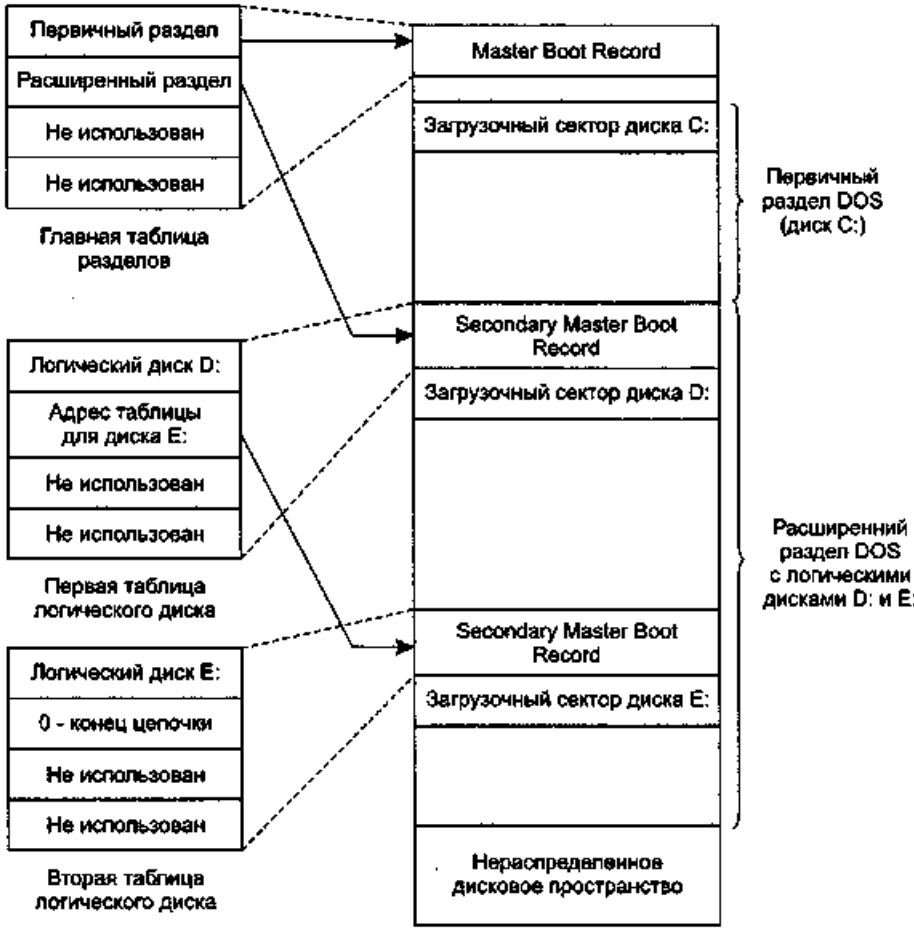
Согласно принятым спецификациям на одном жестком диске может быть только один расширенный раздел, который, в свою очередь, может быть разделен на большое количество подразделов — *логических дисков* (logical disks). В этом смысле термин «первичный» можно признать не совсем удачным переводом слова «primary» — лучше было бы перевести «простейший», или «примитивный». В этом случае становится понятным и логичным термин «расширенный». Расширенный раздел содержит вторичную запись MBR (Secondary MBR, SMBR), в состав которой вместо таблицы разделов входит аналогичная ей *таблица логических дисков* (Logical Disks Table, LDT). Таблица LDT описывает размещение и характеристики раздела, содержащего единственный логический диск, а также может специфицировать следующую запись SMBR. Следовательно, если в расширенном разделе создано K логических дисков, то он содержит K экземпляров SMBR, связанных в список. Каждый элемент этого списка описывает соответствующий логический диск и ссылается (кроме последнего) на следующий элемент списка.

Как мы уже сказали, загрузчик NSB служит для поиска с помощью таблицы разделов активного раздела, копирования в оперативную память компьютера системного загрузчика (System Bootstrap, SB) из выбранного раздела и передачи на него управления, что позволяет осуществить загрузку ОС.

Вслед за сектором MBR размещаются собственно сами разделы (рис. 5.4). В процессе начальной загрузки сектора MBR, содержащего таблицу разделов, работают программные модули BIOS. Начальная загрузка считается выполненной корректно только в том случае, если таблица разделов содержит допустимую информацию.

Рассмотрим еще раз процесс загрузки операционной системы. Процедура начальной загрузки (bootstrap loader) вызывается как программное прерывание (BIOS INT 19h). Эта процедура определяет первое готовое устройство из списка разрешенных и доступных (гибкий или жесткий диск, а в современных компьютерах это могут быть еще и компакт-диск, привод ZIP-drive компании Iomega, сетевой адаптер или еще какое-нибудь устройство) и пытается загрузить с него в оперативную память короткую главную программу-загрузчик. Для накопителей на жестких магнитных дисках — это уже известный нам главный, или внесистемный, загрузчик (NSB) из MBR, и ему передается управление. Главный загрузчик определяет на диске активный раздел, загружает его собственный системный загрузчик и передает управление ему. И наконец, этот загрузчик находит и загружает необходимые файлы операционной системы и передает ей управление. Далее операционная система выполняет инициализацию подведомственных ей программных и аппаратных средств. Она добавляет новые сервисы, вызываемые, как правило, тоже через механизм программных прерываний, и расширяет (или заменяет) некоторые сервисы BIOS. Необходимо отметить, что в современных мультипрограммных операционных системах большинство сервисов BIOS, изначально расположенных в ПЗУ, как правило, заменяются собственными драйверами ОС,

поскольку они должны работать в режиме прерываний, а не в режиме сканирования готовности.




Согласно рассмотренному процессу, каждый раз при запуске компьютера будет загружаться одна и та же операционная система. Это не всегда нас может устраивать. Так называемые *менеджеры загрузки* (boot managers) предназначены для того, чтобы пользователь мог выбрать среди нескольких установленных на компьютере операционных систем желаемую и передать управление на загрузчик выбранной ОС. Имеется большое количество таких менеджеров. Одним из наиболее мощных менеджеров загрузки является OS Selector от фирмы Acronis. Эта программа имеет следующие основные особенности:

* поддержка большого количества операционных систем, включая различные версии DOS (MS DOS, DR-DOS и др.), Windows (9x/ME, NT/2000/XP), OS/2, Linux, FreeBSD, SCO Unix, BeOS и др.;

* возможность установки на любой раздел FAT16/FAT32, в том числе и на отдельный раздел, недоступный другим операционным системам;
* возможность с помощью меню загрузки, предоставляемого менеджером, осуществить загрузку с дискеты;
* автоматическая идентификация операционных систем как на первичных разделах, так и на логических дисках расширенного раздела всех НЖМД, доступных через BIOS компьютера;
* поддержка нескольких операционных систем на одном разделе FAT16/FAT32, при этом предотвращаются конфликты по системным и конфигурационным файлам для систем, установленных на одном разделе;
* возможность дополнительной настройки конфигураций операционных систем и легкого их добавления и удаления;
* встроенная защита от загрузочных вирусов;
* легкое восстановление в случае повреждения MBR;
* поддержка больших жестких дисков во всех режимах современных подсистем BIOS;
* возможность установки паролей отдельно на меню загрузки и на выбранные конфигурации.

формирование таблицы разделов осуществляется с помощью специальных утилит. Обычно их называют FDisk (от слов «Form Disk» — формирование диска). Хотя есть и иные программы, которые могут делать с разделами намного больше, чем простейшие утилиты FDisk от Microsoft. Надо признать, что в последнее время появилось большое количество утилит, которые предоставляют возможность более наглядно представить разбиение диска на разделы, поскольку в них используется графический интерфейс. Эти программы успешно и корректно работают с наиболее распространенными типами разделов (разделы под FAT, FAT32, NTFS). Однако созданы они в основном для работы в среде Win32API, что часто ограничивает возможность их применения. Одной из самых известных и мощных программ для работы с разделами жесткого диска является Partition Magic фирмы Power Quest.

Еще одной мощной утилитой такого рода является Администратор дисков, входящий в состав уже упоминавшегося менеджера загрузки OS Selector от Acronis. Эта утилита позволяет:

* создавать разделы любых типов и форматировать их под файловые системы FAT16, FAT32, NTFS, Ext2FS (Linux), Linux ReiserFS, Linux Swap, при этом можно выбирать точное или произвольное расположение раздела и указывать его параметры;
* получать подробную информацию о разделах и о самих жестких дисках;
* удалять любые разделы;
* преобразовывать разделы из FAT16 в FAT32 и обратно;
* копировать и перемещать разделы с FAT16, FAT32, NTFS, Linux Ext2FS, Linux ReiserFS и Linux Swap;

* изменять размеры разделов с вышеперечисленными файловыми системами;
* выбирать размер кластера вручную во время любой операции создания, копирования, перемещения или изменения размера раздела;
* посекторно редактировать содержимое жестких дисков и разделов с помощью встроенного многооконного редактора дисков.

В популярных операционных системах от Microsoft тоже имеются средства для просмотра и изменения структуры разделов жесткого диска. Так, в Windows NT 4.0 для управления дисками имеется программа Администратор дисков (Disk Manager), а в Windows 2000 и Windows XP — консоль управления с оснасткой под названием Управление дисками (Disk Management). Эти средства имеют графический интерфейс и позволяют создавать новые разделы, удалять разделы, переопределять букву (имя) логического диска и создавать наборы дисков, выступающие как один логический том.

Утилиты формирования дисков, входящие в состав MS DOS и Windows 95/98, а также утилита, встроенная в программу установки Windows NT, первым элементом таблицы разделов всегда делают первичный раздел. Вторым элементом становится расширенный раздел, в котором, в свою очередь, организуется один или несколько логических дисков. При этом создаваемые логические диски помимо известного буквенного именования (диски C:, D:, E: и т. д.) получают еще и так называемые номера разделов. Диск C: получает в этом случае порядковый номер 1, диск D: — 2, диск E: — 3, и т. д. Именно номера разделов используются в файле boot.ini, который указывает системному загрузчику Windows NT/2000/XP, где находятся файлы выбранной операционной системы.

Следует заметить, что в операционных системах типа Linux логические диски и разделы нумеруются и обозначаются иным способом. Жесткий диск с IDE-интерфейсом, подключенный к первому контроллеру как главный (master), имеет имя hda. Если это второй диск на том же шлейфе, то его именуют hdb[1]. Соответственно, имя hdc будет соответствовать диску, подключенному ко второму порту контроллера и имеющему адрес 0, то есть главному. И так далее. Если раздел диска указан посредством таблицы из MBR, то он имеет номер элемента таблицы разделов. Если же речь идет о логических дисках, созданных в пределах расширенного раздела, то их номера уже начинаются с 5. Тем самым указывается, что раздел описан в следующей (вторичной) записи MBR, то есть в SMBR.

Так, для рассматриваемого нами примера (см. рис. 5.4), раздел с номером 1 в Linux тоже будет иметь номер 1. Если мы имеем единственный накопитель, подключенный к первому порту контроллера, то этот раздел обозначается как hda1. А вот логический диск, по умолчанию именуемый в Windows диском D: и имеющий номер раздела 2, в Linux будет обозначаться как hda5. Логический диск E:, имеющий в Windows номер раздела 3, станет в Linux диском с номером раздела 6 и будет обозначаться hda6. Чтобы понять причину такой нумерации, рассмотрим рис. 5.4

[1] Главным является тот накопитель, который имеет адресацию 0 на IDE-интерфейсе, тогда как диск с адресом 1 обозначается как вспомогательный (slave). Адресация выставляется на одной из линий IDE-шлейфа (26 линия).

более внимательно. Вслед за сектором с MBR размещаются собственно сами разделы. Поскольку на рисунке это в явном виде не показано, напомним, что любой раздел начинается с первого сектора. В таблице разделов имеется 4 элемента, но только два из них задействованы. Первый элемент описывает раздел с номером 1 и ему соответствует логический диск C:. Второй элемент указывает на запись SMBR, в которой первый элемент в таблице логических дисков описывает логический диск D:. И этот элемент является уже пятым элементом, если учесть четыре элемента в MBR. А далее нумерация разделов в Linux отходит от этой идеи. Диск E: получает порядковый номер 6, а не 9, как следовало бы ожидать, если подсчитывать все имеющиеся элементы в таблицах разделов. И это логично, поскольку в каждой таблице дисков логический диск описывает только один элемент — первый. Таким образом, если бы расширенный раздел был разбит не на два, а на три логических диска, то последний подраздел (в системе Windows он именовался бы диском F: ) получил бы номер 7.

## Системный загрузчик Windows NT/2000/XP

Операционные системы класса Windows NT имеют возможность загружать не одну операционную систему, а несколько, то есть системный загрузчик Windows NT/2000/XP является менеджером загрузки. Для указания установленных операционных систем и выбора одной из них используется файл boot.ini. Этот файл является текстовым. Он обрабатывается программой ntldr, которая, собственно, и является системным загрузчиком и на которую передается управление из внесистемного загрузчика.

Файл boot.ini состоит из двух секций. Пример такого файла приведен в листинге 5. 1.

**Листинг 5. 1**. Файл boot. ini

```
[boot  loader]
timeout=10
default=multi(0)disk(0)rdisk(0)partition(2)\WINNT
[operating systems]
multi(0)disk(0)rdisk(0)partition(2)\WINNT="IT MTC. EDU Microsoft Windows 2000 Server RUS"
/fastdetect
multi(0)disk(0)rdisk(l)partition(2)\WIN2KP="Staff MTC. EDU Microsoft Windows 2000
Professional RUS" /fastdetect
multi(0)disk(0)rdisk(0)partition(4)\WIN2K_S="SQL  server on M$ Windows 2000 Server RUS" /
fastdetect
multi(0)disk(0)rdisk(2)partition(2)\WIN2K PRO="Microsoft Windows 2000 Professional RUS"
/fastdetect
C \="Microsoft Windows 98"
C\CMDCONS\BOOTSECT DAT="Recovery Console Microsoft Windows 2000" /cmdcons
```

В первой секции этого файла, названной [boot loader], строка timeout задает время в секундах, по истечении которого будет загружаться операционная система, указанная в строке default этой секции. Как мы видим, для выбора одной из операционных систем пользователю дается 10 с. Если бы значение timeout равнялось нулю или во второй секции была бы прописана только одна операционная система, то у пользователя не было бы выбора. В этом случае будет загружаться система,

указанная в строке default Если же значение timeout равняется -1, то загрузка не будет происходить до тех пор, пока пользователь явно не выберет в меню одну из операционных систем и не нажмет клавишу Enter.

Инструкция default указывает, где (на каком накопителе и в каком разделе этого накопителя) располагается операционная система, загружаемая по умолчанию. В большинстве случаев мы можем увидеть там примерно такую строку:

```
default=multi(0)disk(0)rdisk(0)partition(2)\WINNT
```

Слово multi в этой строке означает, что при работе программы ntldr должны использоваться драйверы из BIOS компьютера (используется прерывание int13h). Номер в скобках должен быть равен 0.

Слово disk на персональных компьютерах с подключением накопителей на магнитных дисках через IDE-интерфейс фактически не несет никакой информации, однако оно должно быть записано, а в скобках должен стоять ноль. В случае SCSI-дисков это слово задает идентификатор SCSI ID диска.

Слово rdisk определяет порядковый номер накопителя. Всего при использовании IDE-интерфейса может быть до 4 накопителей на жестких дисках; они нумеруются от 0 до 3.

Наконец, слово partition определяет номер раздела, на который установлена операционная система. После указания раздела записывается имя каталога, в котором расположены файлы этой операционной системы.

Во второй секции, обозначенной как [operating systems], построчно перечисляются пути к установленным операционным системам с текстовыми полями, заключенными в кавычки. Именно тот текст мы и видим при работе загрузчика ntldr, когда он выводит меню с операционными системами. Если на компьютере установлены помимо систем Windows NT/2000/XP еще какие-нибудь операционные системы (например, DOS, Windows 9x, Linux и т. д.), то их можно будет также загрузить. Для этого в секции необходимо указать полный путь к файлу, в котором должен содержаться соответствующий системный загрузчик (загрузочный сектор). Этот файл обязательно должен располагаться на том же диске C:, иначе программа ntldr не сможет его найти. Следует отметить, что для MS DOS и Windows 9x можно не указывать имя файла с загрузочным сектором, а указать только сам корневой каталог диска C:. Но это возможно только в том случае, если имя файла, содержащего системный загрузчик, будет стандартным — bootsect.dos.

# Кэширование операций ввода-вывода при работе с накопителями на магнитных дисках

Как известно, накопители на магнитных дисках обладают крайне низким быстродействием по сравнению с процессорами и оперативной памятью. Разница составляет несколько порядков. Например, современные процессоры за один такт работы, а они работают уже с частотами в несколько гигагерц, могут выполнять по две операции, и, таким образом, время выполнения операции (с позиции внешнего на-

блюдателя, который не видит конвейеризации при выполнении машинных команд, позволяющей увеличить производительность в несколько раз) может составлять менее 0,5 нс (!). В то же время переход магнитной головки с дорожки на дорожку занимает несколько миллисекунд; подобная же задержка требуется и на поиск нужного сектора данных. Как известно, в современных приводах средняя длительность на чтение случайным образом выбранного сектора данных составляет около 20 мс, что существенно медленнее, чем выборка команды или операнда из оперативной памяти и уж тем более из кэш-памяти. Правда, после этого данные читаются большим пакетом (сектор, как мы уже говорили, имеет размер 512 байт, а при операциях с диском часто читаются или записываются сразу несколько секторов). Таким образом, средняя скорость работы процессора с оперативной памятью на 2-3 порядка выше, чем средняя скорость передачи данных из внешней памяти на магнитных дисках в оперативную память.

Для того чтобы сгладить такое сильное несоответствие в производительности основных подсистем, используется буферизация и/или *кэширование* данных в дисковом кэше (disk cache). Простейшим вариантом ускорения дисковых операций чтения данных можно считать использование двойной буферизации. Ее суть заключается в том, что пока в один буфер заносятся данные с магнитного диска, из второго буфера ранее считанные данные могут быть прочитаны и переданы в запросившую их задачу. Аналогично и при записи данных. Буферизация используется во всех операционных системах, но помимо буферизации применяется и кэширование. Кэширование исключительно полезно в том случае, когда программа неоднократно читает с диска одни и те же данные. После того как они один раз будут помещены в кэш, обращений к диску больше не потребуется, и скорость работы программы значительно возрастет.

Упрощая, можно сказать, что под дисковым кэшем можно понимать некий пул буферов, которыми мы управляем с помощью соответствующего системного процесса. Если считывается какое-то множество секторов, содержащих записи того или иного файла, то эти данные, пройдя через кэш, там остаются (до тех пор, пока другие секторы не заменят эти буферы). Если впоследствии потребуется повторное чтение, то данные могут быть извлечены непосредственно из оперативной памяти без фактического обращения к диску. Ускорить можно и операции записи: данные помещаются в кэш, и для запросившей эту операцию задачи получается, что фактически они уже записаны. Задача может продолжить свое выполнение, а системные внешние процессы через некоторое время запишут данные на диск. Это называется *отложенной записью* (lazy write[1]). Если режим отложенной записи отключен, только одна задача может записывать на диск свои данные. Остальные приложения должны ждать своей очереди. Это ожидание подвергает информацию риску не меньшему (если не большему), чем сама отложенная запись, которая к тому же и более эффективна по скорости работы с диском.

Интервал времени, после которого данные будут фактически записываться, с одной стороны, желательно выбрать большим, поскольку это позволило бы не читать (если потребуется) эти данные заново, так как они уже и так фактически на-

[1] Дословно — «ленивая» запись.

ходятся в кэше. И после их модификации эти данные опять же помещаются в быстродействующий кэш. С другой стороны, для большей надежности, желательно поскорее отправить данные во внешнюю память, поскольку она энергонезависима, и в случае какой-нибудь аварии (например, нарушения питания) данные в оперативной памяти пропадут, в то время как на магнитном диске они с большой вероятностью останутся в безопасности.

Поскольку количество буферов, составляющих кэш, ограничено, может возникнуть ситуация, когда считываемые или записываемые данные потребуют замены данных в этих буферах. При этом возможны различные дисциплины выделения буферов под вновь затребованную операцию кэширования.

Кэширование дисковых операций может быть существенно улучшено за счет *упреждающего чтения* (read ahead), которое основано на чтении с диска гораздо большего количества информации, чем на самом деле запрошено приложением или операционной системой. Когда некоторой программе требуется считать с диска только один сектор, программа кэширования читает несколько дополнительных блоков данных. При этом, как известно, операции последовательного чтения нескольких секторов фактически несущественно замедляют операцию чтения затребованного сектора с данными. Поэтому, если программа вновь обратится к диску, вероятность того, что нужные ей данные уже находятся в кэше, будет достаточно высока. Поскольку передача данных из одной области памяти в другую происходит во много раз быстрее, чем чтение их с диска, кэширование существенно сокращает время выполнения операций с файлами.

Итак, путь информации от диска к прикладной программе пролегает как через буфер, так и через дисковый кэш. Когда приложение запрашивает с диска данные, программа кэширования перехватывает этот запрос и читает вместе с необходимыми секторами еще и несколько дополнительных. Затем она помещает в буфер требующуюся задаче информацию и ставит об этом в известность операционную систему. Операционная система сообщает задаче, что ее запрос выполнен, и данные с диска находятся в буфере. При следующем обращении приложения к диску программа кэширования прежде всего проверяет, не находятся ли уже в памяти затребованные данные. Если это так, то она копирует их в буфер, если же их в кэше нет, то запрос на чтение диска передается операционной системе. Когда задача изменяет данные в буфере, они копируются в кэш.

Важно заметить, что простое увеличение объема памяти, отводимого под кэширование файлов, может и не привести к росту быстродействия системы. Другими словами, наблюдается далеко не прямо пропорциональная зависимость ускорения операций с файлами от размера кэша. Кривая этой зависимости достаточно скоро перестает расти, а затем и вовсе эффективность кэширования начинает снижаться. Объяснение этому заключается в том, что поиск нужного фрагмента данных в буферах кэша осуществляется путем их полного перебора. Поэтому с ростом числа буферов кэша затраты на их перебор становятся значительными. И поскольку невозможно обеспечить 100-процентного кэш-попадания искомых данных, то естественно наступает момент, когда среднее время доступа к данным перестает снижаться с увеличением кэша. Очевидно, что оптимальный размер дискового кэша

зависит от очень многих факторов, в том числе и от частоты повторных обращений к недавно прочитанным данным, и от среднего объема обрабатываемых файлов, и от разницы в быстродействии центральной части компьютера и дисковой подсистемы.

В ряде операционных систем имеется возможность указать в явном виде параметры кэширования, в то время как в других за эти параметры отвечает сама операционная система.

Так, в системах семейства Windows 9x мы можем указать и объем памяти, отводимый для кэширования, и объем порции (chunk[1]) данных, из которых набирается кэш, и предельное количество имен файлов, и параметры кэширования каталогов. В файле SYSTEM.INI, расположенном в основном каталоге такой операционной системы (обычно это каталог Windows), в секции [vcache] есть возможность прописать, например, следующие значения:

```
[vcache]
MinFileCache=4096
MaxFileCache=32768
ChunkSize=512
```

Здесь указано, что минимально под кэширование данных зарезервировано 4 Мбайт оперативной памяти, максимальный объем кэша может достигать 32 Мбайт, а размер данных, которыми манипулирует менеджер кэша, равен одному сектору. Следует заметить, что поскольку в современных компьютерах нередко устанавливается большой объем оперативной памяти, порой существенно превосходящий 256 Мбайт, то для обеспечения корректной работы подсистемы кэширования обязательно нужно указывать в явном виде значение MaxFileCache. Оно ни в коем случае не должно превышать величину 262 144 Кбайт. Это ограничение следует из-за особенностей программной реализации подсистемы кэширования[2] — при превышении этого значения происходят нарушения в работе подсистемы памяти и вычислительные процессы могут быть разрушены.

Во всех операционных системах от Microsoft принята стратегия активного кэширования файлов, при которой для кэширования отводится вся свободная память. Поэтому без явного ограничения объема памяти, отводимой под кэширование файлов, мы можем столкнуться с ситуацией, когда рост дискового кэша приводит к значительному росту числа страниц памяти, «сброшенных» в файл подкачки. Последнее может привести к заметному замедлению работы системы, несмотря на то что кэширование имеет целью именно ускорение в работе дисковой подсистемы.

В операционных системах Windows NT 4.0, Windows 2000 и Windows XP также имеется возможность управлять некоторыми параметрами кэширования. Правда, сделать это можно только путем редактирования реестра.

Например, если в разделе [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management] реестра найти параметр IOPageLockLimit и присвоить ему значение 163777216, то это будет означать, что 16 384 Кбайт будут отведе-

[1] Дословно — «кусочек».

[2] Драйвер VCACHE разрабатывался в то время, когда объем памяти в 256 Мбайт казался недостижимым.

ны в физической памяти для хранения буферов дискового кэша. Эта память не может быть выгружена в файл подкачки. Дело в том, что, к большому сожалению, разработчики из Microsoft приняли решение, согласно которому кэшируемые файлы отображаются на виртуальное адресное пространство, а не на физическую память компьютера, как это сделано в других операционных системах. Это означает, что некоторые страничные кадры этого виртуального адресного пространства могут быть отображены не на реальную оперативную память компьютера, а размещены во внешней памяти (попасть в страничный файл подкачки). Очевидно, что это может сильно замедлять работу рассматриваемой подсистемы. Поэтому блокирование некоторого числа страниц файлового кэша от перемещения их во внешнюю память должно приводить к повышению эффективности кэширования. В качестве рекомендации можно заметить, что упомянутое значение в 16 Мбайт можно выделять для компьютеров с объемом памяти более 128 Мбайт.

В других операционных системах можно указывать больше параметров, определяющих работу подсистемы кэширования (см., например, раздел «Файловая система HPFS» в главе 6).

Помимо описанных действий, связанных с кэшированием файлов, операционная система может оптимизировать перемещение головок чтения/записи данных, связанное с выполнением запросов от параллельно выполняющихся задач. Время, необходимое на получение данных с магнитного диска, складывается из времени перемещения магнитной головки на требуемый цилиндр и времени поиска заданного сектора; а временем считывания найденного сектора и временем передачи этих данных в оперативную память мы можем пренебречь. Таким образом, основные затраты времени уходят на поиск данных. В мультипрограммных операционных системах при выполнении многих задач запросы на чтение и запись данных могут идти таким потоком, что при их обслуживании образуется очередь. Если выполнять эти запросы в порядке поступления их в очередь, то вследствие случайного характера обращений к тому или иному сектору магнитного диска потери времени на поиск данных могут значительно возрасти. Напрашивается очевидное решение: поскольку переупорядочивание запросов с целью минимизации затрат времени на поиск данных можно выполнить очень быстро (практически этим временем можно пренебречь, учитывая разницу в быстродействии центральной части компьютера и устройств ввода-вывода), то необходимо найти метод, позволяющий выполнить такое переупорядочивание оптимальным образом. Изучение этой проблемы позволило найти наиболее эффективные дисциплины планирования.

Перечислим известные дисциплины, в соответствии с которыми можно перестраивать очередь запросов на операции чтения/записи данных [11].

* *SSTF* (Shortest Seek Time First — запрос с наименьшим временем позиционирования выполняется первым). В соответствии с этой дисциплиной при позиционировании магнитных головок следующим выбирается запрос, для которого необходимо минимальное перемещение с цилиндра на цилиндр, даже если этот запрос не был первым в очереди на ввод-вывод. Однако для этой дисциплины характерна сильная дискриминация некоторых запросов, а ведь они могут идти от высокоприоритетных задач. Обращения к диску проявляют тен-

денцию концентрироваться, в результате чего запросы на обращение к самым внешним и самым внутренним дорожкам могут обслуживаться существенно дольше, и нет никакой гарантии обслуживания. Достоинством такой дисциплины является максимально возможная пропускная способность дисковой подсистемы.

* *Scan* (сканирование). При сканировании головки поочередно перемещаются то в одном, то в другом «привилегированном» направлении, обслуживая «по пути» подходящие запросы. Если при перемещении головок чтения/записи более нет попутных запросов, то движение начинается в обратном направлении.
* *Next-Step Scan* (отложенное сканирование). Отличается от предыдущей дисциплины тем, что на каждом проходе обслуживаются только те запросы, которые уже существовали на момент начала прохода. Новые запросы, появляющиеся в процессе перемещения головок чтения/записи, формируют новую очередь запросов, причем таким образом, чтобы их можно было оптимально обслужить на обратном ходу.
* *C-Scan* (циклическое сканирование). По этой дисциплине головки перемещаются циклически с самой наружной дорожки к внутренним, по пути обслуживая имеющиеся запросы, после чего вновь переносятся к наружным цилиндрам. Эту дисциплину иногда реализуют таким образом, чтобы запросы, поступающие во время текущего прямого хода головок, обслуживались не попутно, а при следующем проходе, что позволяет исключить дискриминацию запросов к самым крайним цилиндрам. Эта дисциплина характеризуется очень малой дисперсией времени ожидания обслуживания [11]. Ее часто называют «элеваторной».

# Контрольные вопросы и задачи

## Вопросы для проверки

1. Почему создание подсистемы ввода-вывода считается одной из самых сложных областей проектирования операционных систем?
2. Почему операции ввода-вывода в операционных системах объявляются привилегированными?
3. Перечислите основные задачи, возлагаемые на супервизор ввода-вывода?
4. В каких случаях устройство ввода-вывода называется инициативным?
5. Какие режимы управления вводом-выводом вы знаете? Опишите каждый из них.
6. Что означает термин «spooling» и что означает термин «swapping»?
7. Чем обеспечивается независимость пользовательских программ от устройств ввода-вывода, подключенных к компьютеру?
3. Что такое синхронный и асинхронный ввод-вывод?
9. Опишите структуру магнитного диска (разбиение дисков на разделы). Сколько (и каких) разделов может быть на магнитном диске?

10. Как в общем случае осуществляется загрузка операционной системы после включения компьютера? Что такое начальный, системный и внесистемный загрузчики? Где они располагаются?
11. Расскажите о кэшировании операций ввода-вывода при работе с накопителями на магнитных дисках.

## Задания

Используя специально выделенный для этих целей компьютер, изучите структуру диска и освойте работу с программой Disk Editor.

1. Включите компьютер. Во время выполнения программы самотестирования войдите в BIOS и установите возможность загрузки с дискеты.
2. Загрузите операционную систему, расположенную на магнитном диске.
3. Загрузитесь с системной дискеты (на ней должна быть система MS DOS с необходимыми утилитами).
4. Запустите программу Disk Editor из комплекта утилит Питера Нортона. С помощью встроенной справочной системы изучите основные возможности этой утилиты.
5. Посмотрите структуру диска. Сохраните MBR и загрузочный сектор на своей дискете.
6. Найдите таблицы размещения файлов для указанного раздела магнитного диска. Сохраните их на дискете. Выйдите из программы Disk Editor.
7. Запустите программу FDisk. Изучите структуру диска, посмотрите, сколько и каких разделов на нем расположено?
8. Удалите все логические диски с помощью программы FDisk.
9. Перезапустите компьютер и убедитесь, что операционная система, расположенная раньше на магнитном диске, больше не функционирует.
10. Используя программу Disk Editor, восстановите операционную систему, а также файлы, расположенные на магнитном диске и созданные ранее с помощью программы Disk Editor.