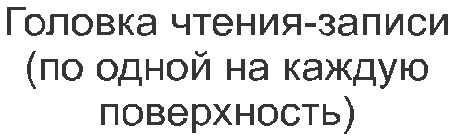
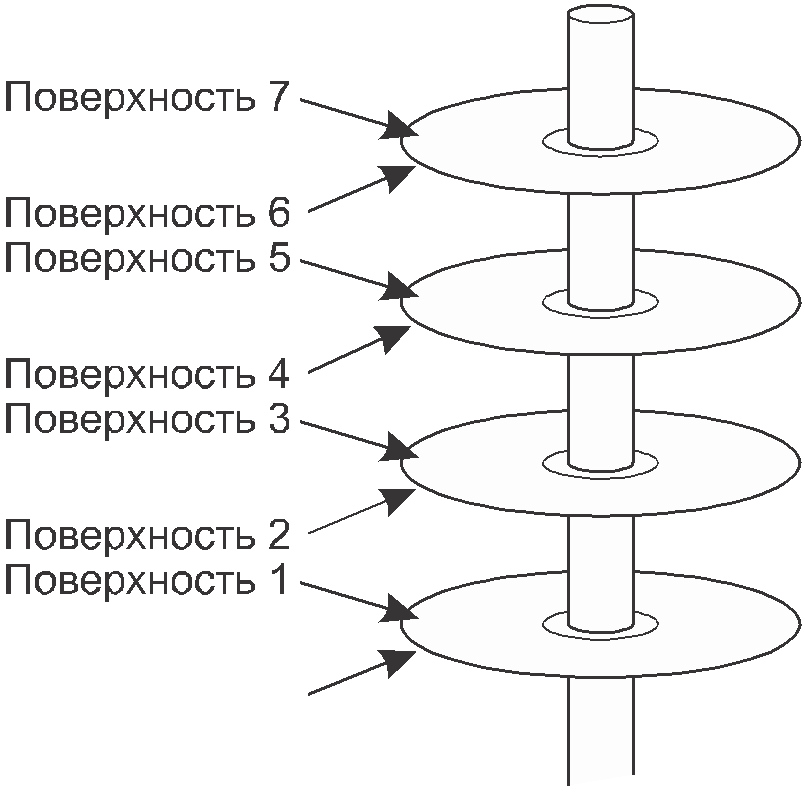
|  |  |
| --- | --- |
|  |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Рис. 1.10.** Схема конструкции жесткого диска



дорожкой, накопитель должен выждать, когда нужный сектор попадет под головку. Это приводит к возникновению еще одной задержки от 5 до 10 мс в зависимости от скорости вращения диска. После попадания требуемого сектора под головку произво- дится операция чтения или записи со скоростью от 50 Мбайт/с (для низкоскоростных дисков) до 160 Мбайт/с (для высокоскоростных).

Порой речь заходит о таких дисках, которые на самом деле дисками не являются, на- пример о твердотельных накопителях — SSD (Solid State Disks). У них нет движущихся частей, дисковых пластин, а данные хранятся во флеш-памяти. Они напоминают диски только тем, что содержат большой объем данных, который при отключении питания не теряется.

Многие компьютеры поддерживают схему, которая называется **виртуальной памятью**. Ее мы довольно основательно рассмотрим в главе 3. Она дает возможность запускать про- граммы, превышающие по объему физическую память компьютера, за счет помещения их на диск и использования оперативной памяти как некой разновидности кэша для наи- более интенсивно исполняемых частей. Эта схема требует прозрачного для программы преобразования адресов памяти, чтобы конвертировать адрес, сгенерированный програм- мой, в физический адрес, по которому слово размещено в ОЗУ. Такое отображение адре- сов осуществляется частью центрального процессора, называется блоком управления памятью (Memory Management Unit (**MMU**)), или **диспетчером памяти** (см. рис. 1.6).

Использование кэширования и MMU может оказать существенное влияние на про- изводительность. При работе в мультипрограммном режиме, когда осуществляется переключение с одной программы на другую, иногда называемое **переключением контекста** (context switch), может потребоваться сброс всех измененных блоков из кэш-памяти и изменение регистров отображения в MMU. Обе эти операции обходятся слишком дорого, и программисты всеми силами стараются их избежать. Некоторые последствия применяемых ими тактических приемов мы рассмотрим чуть позже.

**1.3.5. Устройства ввода-вывода**

Центральный процессор и память не единственные ресурсы, которыми должна управ- лять операционная система. С ней также активно взаимодействуют и устройства ввода-

вывода информации. На рис. 1.6 видно, что устройства ввода-вывода обычно состоят из двух компонентов: самого устройства и его контроллера. Контроллер представляет собой микросхему или набор микросхем, которые управляют устройством на физи- ческом уровне. Он принимает от операционной системы команды, например считать данные с помощью устройства, а затем их выполняет.

Довольно часто непосредственное управление устройством очень сложно и требует высокого уровня детализации, поэтому задачей контроллера является предоставление операционной системе простого (но не упрощенного) интерфейса. Например, контрол- лер диска может получить команду прочитать сектор 11 206 с диска 2. Получив коман- ду, контроллер должен преобразовать этот простой порядковый номер сектора в номер цилиндра, сектора и головки. Операция преобразования может быть затруднена тем, что внешние цилиндры имеют больше секторов, чем внутренние, а номера «плохих» секторов отображаются на другие секторы. Затем контроллер должен определить, над каким цилиндром сейчас находится привод головок, и выдать команду, чтобы он пере- местился вперед или назад на требуемое количество цилиндров. Далее необходимо дождаться, пока нужный сектор не попадет под головку, а затем приступить к чтению и сохранению битов по мере их поступления из привода, удаляя заголовки и подсчи- тывая контрольную сумму. В завершение он должен собрать поступившие биты в слова и сохранить их в памяти. Для осуществления всей этой работы контроллеры часто содержат маленькие встроенные компьютеры, запрограммированные на выполнение подобных задач.

Другим компонентом является само устройство. Устройства имеют довольно про- стые интерфейсы, поскольку они, во-первых, обладают весьма скромными возмож- ностями, а во-вторых, должны отвечать общим стандартам. Соблюдение последнего условия необходимо для того, чтобы, к примеру, любой контроллер SATA-диска смог работать с любым SATA-диском. **SATA** означает Serial ATA (последовательный ATA), а ATA, в свою очередь, означает AT-подключение. Если вы не в курсе того, что именно означает AT, то эта аббревиатура была введена для второго поколения компьютеров IBM — Personal Computer Advanced Technology (персональный компьютер, изготов- ленный по передовым технологиям), построенных на основе очень мощного по тем временам процессора 80286, имевшего тактовую частоту 6 МГц и выпущенного компа- нией в 1984 году. Из этого факта можно сделать вывод, что компьютерная индустрия имеет привычку постоянно дополнять существующие акронимы новыми префиксами и суффиксами. Кроме того, можно прийти к выводу, что такие прилагательные, как

«advanced» (передовая, передовой), должны использоваться весьма осмотрительно, чтобы спустя 30 лет это не выглядело глупо.

В наше время SATA является стандартным типом дисков на многих компьютерах. По- скольку интерфейс устройства скрыт его контроллером, все операционные системы видят только интерфейс контроллера, который может существенно отличаться от интерфейса самого устройства.

Так как все типы контроллеров отличаются друг от друга, для управления ими требу- ется различное программное обеспечение. Программа, предназначенная для общения с контроллером, выдачи ему команды и получения поступающих от него ответов, называется **драйвером устройства**. Каждый производитель контроллеров должен по- ставлять вместе с ними драйверы для каждой поддерживаемой операционной системы. Например, сканер может поступить в продажу с драйверами для операционных систем OS X, Windows 7, Windows 8 и Linux.

Для использования драйвер нужно поместить в операционную систему, предоставив ему тем самым возможность работать в режиме ядра. Вообще-то драйверы могут работать и не в режиме ядра, и поддержка такого режима предлагается в настоящее время в опера- ционных системах Linux и Windows. Подавляющее большинство драйверов по-прежнему запускается ниже границы ядра. В пользовательском пространстве драйверы запуска- ются только лишь в весьма немногих существующих системах, например в MINIX 3. Драйверам в пользовательском пространстве должно быть разрешено получать доступ к устройству неким контролируемым способом, что является весьма непростой задачей.

Существует три способа установки драйвера в ядро. Первый состоит в том, чтобы заново скомпоновать ядро вместе с новым драйвером и затем перезагрузить систему. Многие устаревшие UNIX-системы именно так и работают. Второй способ: создать в специальном файле операционной системы запись, сообщающую ей о том, что тре- буется, и затем перезагрузить систему. Во время загрузки операционная система сама находит нужные ей драйверы и загружает их. Именно так и работает система Windows. При третьем способе — динамической загрузке драйверов — операционная система может принимать новые драйверы в процессе работы и оперативно устанавливать их, не требуя для этого перезагрузки. Этот способ ранее использовался довольно редко, но сейчас он получает все большее распространение. Внешние устройства, работающие по принципу «горячего подключения»1, к которым относятся рассматриваемые далее устройства с интерфейсами USB и IEEE 1394, всегда нуждаются в динамически за- гружаемых драйверах.

В каждом контроллере для связи с ним имеется небольшое количество регистров. Например, простейший контроллер диска может иметь регистры для указания адреса на диске, адреса в памяти, счетчика секторов и направления передачи информации (чтение или запись). Чтобы активизировать контроллер, драйвер получает команду от операционной системы, затем переводит ее в соответствующие значения для записи в регистры устройства. Из совокупности всех регистров устройств формируется про- странство портов ввода-вывода, к которому мы еще вернемся в главе 5.

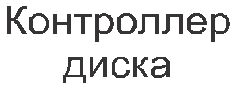
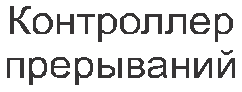
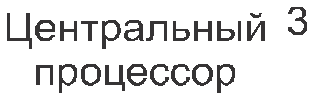
На некоторых компьютерах регистры устройств отображаются в адресное пространство операционной системы (на те адреса, которые она может использовать), поэтому со- стояния регистров можно считывать и записывать точно так же, как и обычные слова в оперативной памяти. На таких компьютерах не требуются какие-то специальные команды ввода-вывода, а пользовательские программы можно держать подальше от оборудования, помещая эти адреса за пределами досягаемости программ (например, за счет использования базовых регистров и регистров границ области памяти). На других компьютерах регистры устройств помещаются в специальное **пространство портов ввода-вывода** (I/O port space), в котором каждый регистр имеет адрес порта. На та- ких машинах в режиме ядра доступны специальные команды ввода-вывода (обычно обозначаемые *IN* и *OUT*), позволяющие драйверам читать и записывать данные в реги- стры. Первая схема исключает необходимость в специальных командах ввода-вывода, но задействует часть адресного пространства. Вторая схема не использует адресное пространство, но требует наличия специальных команд. Обе схемы используются до- вольно широко.

1 То есть устройства, которые могут быть подключены к компьютеру или отключены от него без необходимости останавливать работу операционной системы и отключать компьютер от источника питания. — *Примеч. ред.*

Ввод и вывод данных можно делать тремя различными способами. В простейшем из них пользовательская программа производит системный вызов, который транслируется ядром в процедуру вызова соответствующего драйвера. После этого драйвер присту- пает к процессу ввода-вывода. В это время он выполняет очень короткий цикл, посто- янно опрашивая устройство и отслеживая завершение операции (обычно занятость устройства определяется состоянием специального бита). По завершении операции ввода-вывода драйвер помещает данные (если таковые имеются) туда, куда требуется, и возвращает управление. Затем операционная система возвращает управление вы- зывающей программе. Этот способ называется **активным ожиданием** или **ожиданием готовности**, а его недостаток заключается в том, что он загружает процессор опросом устройства об окончании работы.

Второй способ заключается в том, что драйвер запускает устройство и просит его вы- дать прерывание по окончании выполнения команды (завершении ввода или вывода данных). Сразу после этого драйвер возвращает управление. Затем операционная система блокирует вызывающую программу, если это необходимо, и переходит к вы- полнению других задач. Когда контроллер обнаруживает окончание передачи данных, он генерирует **прерывание**, чтобы просигнализировать о завершении операции.

Прерывания играют очень важную роль в работе операционной системы, поэтому рас- смотрим их более подробно. На рис. 1.11, *а* показан процесс ввода-вывода. На первом этапе драйвер передает команду контроллеру, записывая информацию в его регистры. Затем контроллер запускает само устройство. На втором этапе, когда контроллер за- вершает чтение или запись заданного ему количества байтов, он выставляет сигнал для микросхемы контроллера прерываний, используя для этого определенные линии шины. На третьем этапе, если контроллер прерываний готов принять прерывание (а он может быть и не готов к этому, если обрабатывает прерывание с более высоким уровнем приоритета), он выставляет сигнал на контакте микросхемы центрального процессора, информируя его о завершении операции. На четвертом этапе контроллер прерываний выставляет номер устройства на шину, чтобы процессор мог его считать и узнать, какое устройство только что завершило работу (поскольку одновременно могут работать сразу несколько устройств).



**Рис. 1.11.** Этапы: *a* — запуска устройства ввода-вывода и получения прерывания; *б* —обработки прерывания (включает в себя получение прерывания, выполнение программы обработки прерывания и возвращение управления программе пользователя)

Как только центральный процессор решит принять прерывание, содержимое счетчи- ка команд и слова состояния программы помещаются, как правило, в текущий стек и процессор переключается в режим ядра. Номер устройства может быть использован как индекс части памяти, используемой для поиска адреса обработчика прерываний данного устройства. Эта часть памяти называется **вектором прерываний**. Когда об- работчик прерываний (являющийся частью драйвера устройства, выдающего запрос на прерывание) начинает свою работу, он извлекает помещенные в стек содержимое счетчика команд и слова состояния программы и сохраняет их, а затем опрашивает устройство для определения его состояния. После завершения обработки прерывания обработчик возвращает управление ранее работавшей пользовательской программе — на первую же еще не выполненную команду. Все эти этапы показаны на рис. 1.11, *б*.

При третьем способе ввода-вывода используется специальный контроллер **прямого доступа к памяти** (Direct Memory Access (**DMA**)), который может управлять потоком битов между оперативной памятью и некоторыми контроллерами без постоянного вмешательства центрального процессора. Центральный процессор осуществляет настройку контроллера DMA, сообщая ему, сколько байтов следует передать, какое устройство и адреса памяти задействовать и в каком направлении передать данные, а затем дает ему возможность действовать самостоятельно. Когда контроллер DMA завершает работу, он выдает запрос на прерывание, который обрабатывается в ранее рассмотренном порядке. Более подробно контроллер DMA и аппаратура ввода-вывода будут рассмотрены в главе 5.

Прерывания часто происходят в очень неподходящие моменты, например во время работы обработчика другого прерывания. Поэтому центральный процессор обладает возможностью запрещать прерывания с последующим их разрешением. Пока прерыва- ния запрещены, любые устройства, закончившие свою работу, продолжают выставлять свои запросы на прерывание, но работа процессора не прекращается, пока прерывания снова не станут разрешены. Если за время запрещения прерываний завершится работа сразу нескольких устройств, контроллер решает, какое из них должно быть обработано первым, полагаясь обычно на статические приоритеты, назначенные каждому устрой- ству. Побеждает устройство, имеющее наивысший приоритет, которое и обслуживается в первую очередь. Все остальные устройства должны ожидать своей очереди.

**1.3.6. Шины**

Структура, показанная на рис. 1.6, на протяжении многих лет использовалась в ми- ни-компьютерах, а также в первой модели IBM PC. Но по мере увеличения скорости работы процессоров и памяти возможности единой шины (и, конечно, шины IBM PC) по обеспечению всех процессов обмена данными достигли своего предела. Нужно было что-то делать. В результате появились дополнительные шины как для более быстро- действующих устройств ввода-вывода, так и для обмена данными между процессором и памятью. Вследствие этой эволюции массовая x86-система на данный момент имеет вид, показанный на рис. 1.12.

У этой системы имеется множество шин (например, шина кэш-памяти, шина памяти, а также шины PCIe, PCI, USB, SATA и DMI), каждая из которых имеет свою скорость передачи данных и свое предназначение. Операционная система для осуществления функций настройки и управления должна знать обо всех этих шинах. Основной ши- ной является PCI (Peripheral Component Interconnect — интерфейс периферийных устройств).