Лекция 15

Организация системы ввода-вывода

1. Логические принципы организации ввода-вывода

Рассмотренные ранее физические механизмы взаимодействия устройств ввода-вывода с вычислительной системой позволяют понять, почему разнообразные внешние устройства легко могут быть добавлены в существующие компьютеры. Все, что необходимо сделать пользователю при подключении нового устройства, – это отобразить порты устройства в соответствующее адресное пространство, определить, какой номер будет соответствовать прерыванию, генерируемому устройством, и, если нужно, закрепить за устройством некоторый канал DMA. Для нормального функционирования hardware этого будет достаточно. Однако мы до сих пор ничего не сказали о том, как должна быть построена подсистема управления вводом-выводом в операционной системе для легкого и безболезненного добавления новых устройств и какие функции вообще обычно на нее возлагаются.

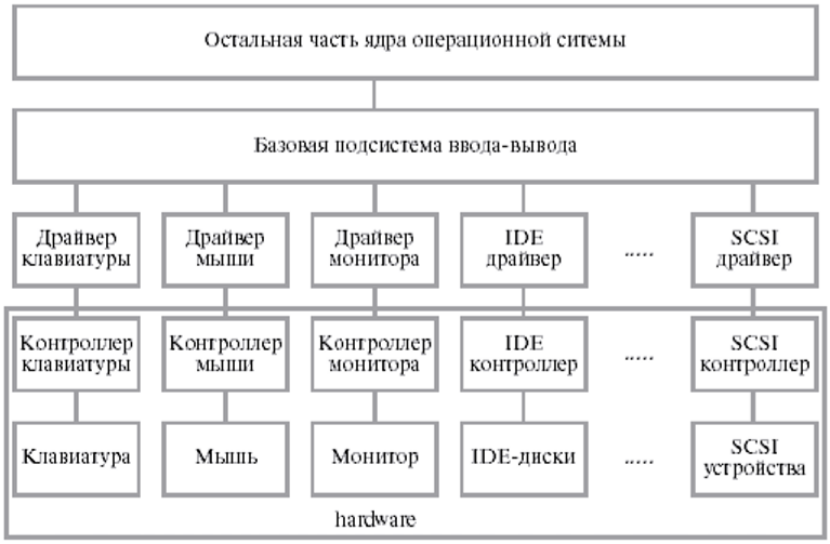
Если поручить неподготовленному пользователю сконструировать систему ввода-вывода, способную работать со всем множеством внешних устройств, то, скорее всего, он окажется в ситуации, в которой находились биологи и зоологи до появления трудов Линнея. Все устройства разные, отличаются по выполняемым функциям и своим характеристикам, и кажется, что принципиально невозможно создать систему, которая без больших постоянных переделок позволяла бы охватывать все многообразие видов. Вот перечень лишь нескольких направлений (далеко не полный), по которым различаются устройства:

- Скорость обмена информацией может варьироваться в диапазоне от нескольких байтов в секунду (клавиатура) до нескольких гигабайтов в секунду (сетевые карты).
- Одни устройства могут использоваться несколькими процессами параллельно (являются разделяемыми), в то время как другие требуют монопольного захвата процессом.
- Устройства могут запоминать выведенную информацию для ее последующего ввода или не обладать этой функцией. Устройства, запоминающие информацию, в свою очередь, могут дифференцироваться по формам доступа к сохраненной информации: обеспечивать к ней последовательный доступ в жестко заданном порядке или уметь находить и передавать только необходимую порцию данных.
- Часть устройств умеет передавать данные только по одному байту последовательно (символьные устройства), а часть устройств умеет передавать блок байтов как единое целое (блочные устройства).
- Существуют устройства, предназначенные только для ввода информации, устройства, предназначенные только для вывода информации, и устройства, которые могут выполнять и ввод, и вывод.

В области технического обеспечения удалось выделить несколько основных принципов взаимодействия внешних устройств с вычислительной системой, т. е. создать единый интерфейс для их подключения, возложив все специфические действия на контроллеры самих устройств. Тем самым конструкторы вычислительных систем переложили все хлопоты, связанные с подключением внешней аппаратуры, на разработчиков самой аппаратуры, заставляя их придерживаться определенного стандарта.

Похожий подход оказался продуктивным и в области программного подключения устройств ввода-вывода. Подобно тому, как Линнею удалось заложить основы систематизации знаний о растительном и животном мире, разделив все живое в природе на относительно небольшое число классов и отрядов, мы можем разделить устройства на относительно небольшое число типов, отличающихся по набору операций, которые могут быть ими выполнены, считая все остальные различия несущественными. Мы можем затем

специфицировать интерфейсы между ядром операционной системы, осуществляющим некоторую общую политику ввода-вывода, и программными частями, непосредственно управляющими устройствами, для каждого из таких типов. Более того, разработчики операционных систем получают возможность освободиться от написания и тестирования этих специфических программных частей, получивших название драйверов, передав эту деятельность производителям самих внешних устройств. Фактически мы приходим к использованию принципа уровневого или слоеного построения системы управления вводом-выводом для операционной системы (рис.).



Два нижних уровня этой слоеной системы составляет hardware: сами устройства, непосредственно выполняющие операции, и их контроллеры, служащие для организации совместной работы устройств и остальной вычислительной системы. Следующий уровень составляют драйверы устройств ввода-вывода, скрывающие от разработчиков операционных систем особенности функционирования конкретных приборов и обеспечивающие четко определенный интерфейс между hardware и вышележащим уровнем – уровнем базовой подсистемы ввода-вывода, которая, в свою очередь, предоставляет механизм взаимодействия между драйверами и программной частью вычислительной системы в целом.

2. Систематизация внешних устройств и интерфейс между базовой подсистемой ввода-вывода и драйверами

Как и система видов Линнея, система типов устройств является далеко не полной и не строго выдержанной. Устройства обычно принято разделять по преобладающему типу интерфейса на следующие виды:

- символьные (клавиатура, модем, терминал и т. п.);
- блочные (магнитные и оптические диски и ленты, и т. д.);
- сетевые (сетевые карты);
- все остальные (таймеры, графические дисплеи, телевизионные устройства, видеокамеры и т. п.).

Такое деление является весьма условным. В одних операционных системах сетевые устройства могут не выделяться в отдельную группу, в некоторых других – отдельные группы составляют звуковые устройства и видеоустройства и т. д. Некоторые группы в свою очередь могут разбиваться на подгруппы: подгруппа жестких дисков, подгруппа мышек, подгруппа принтеров. Нам понадобится эта классификация для иллюстрации того положения, что устройства могут быть разделены на группы по выполняемым ими функциям, и для понимания функций драйверов, и интерфейса между ними и базовой

подсистемой ввода-вывода. Для этого мы рассмотрим только две группы устройств: символьные и блочные. Символьные устройства – это устройства, которые умеют передавать данные только последовательно, байт за байтом, а блочные устройства – это устройства, которые могут передавать блок байтов как единое целое.

К символьным устройствам обычно относятся устройства ввода информации, которые спонтанно генерируют входные данные: клавиатура, мышь, модем, джойстик. К ним же относятся и устройства вывода информации, для которых характерно представление данных в виде линейного потока: принтеры, звуковые карты и т. д. По своей природе символьные устройства обычно умеют совершать две общие операции: ввести символ (байт) и вывести символ (байт) – get и put.

Для блочных устройств, таких как магнитные и оптические диски, ленты и т. п. естественными являются операции чтения и записи блока информации – read и write, а также, для устройств прямого доступа, операция поиска требуемого блока информации – seek.

Драйверы символьных и блочных устройств должны предоставлять базовой подсистеме ввода-вывода функции для осуществления описанных общих операций. Помимо общих операций, некоторые устройства могут выполнять операции специфические, свойственные только им – например, звуковые карты умеют увеличивать или уменьшать среднюю громкость звучания, дисплеи умеют изменять свою разрешающую способность. Для выполнения таких специфических действий в интерфейс между драйвером и базовой подсистемой ввода-вывода обычно входит еще одна функция, позволяющая непосредственно передавать драйверу устройства произвольную команду с произвольными параметрами, что позволяет задействовать любую возможность драйвера без изменения интерфейса. В операционной системе Unix такая функция получила название ioctl (от input-output control).

Помимо функций read, write, seek (для блочных устройств), get, put (для символьных устройств) и ioctl, в состав интерфейса обычно включают еще следующие функции:

- Функцию инициализации или повторной инициализации работы драйвера и устройства – open.
- Функцию временного завершения работы с устройством (может, например, вызывать отключение устройства) – close.
- Функцию опроса состояния устройства (если по каким-либо причинам работа с устройством производится методом опроса его состояния, например, в операционных системах Windows NT и Windows 9x так построена работа с принтерами через параллельный порт) – poll.
- Функцию останова драйвера, которая вызывается при останове операционной системы или выгрузке драйвера из памяти, halt.

Существует еще ряд действий, выполнение которых может быть возложено на драйвер, как правило, это действия базовой подсистемы ввода-вывода. Приведенные выше названия функций, конечно, являются условными и могут меняться от одной операционной системы к другой, но действия, выполняемые драйверами, характерны для большинства операционных систем, и соответствующие функции присутствуют в интерфейсах к ним.

3. Функции базовой подсистемы ввода-вывода

Базовая подсистема ввода-вывода служит посредником между процессами вычислительной системы и набором драйверов. Системные вызовы для выполнения операций ввода-вывода трансформируются ею в вызовы функций необходимого драйвера устройства. Однако обязанности базовой подсистемы не сводятся к выполнению только действий трансляции общего системного вызова в обращение к частной функции драйвера. Базовая подсистема предоставляет вычислительной системе такие услуги, как

поддержка блокирующихся, неблокирующихся и асинхронных системных вызовов, буферизация и кэширование входных и выходных данных, осуществление spooling'a и монопольного захвата внешних устройств, обработка ошибок и прерываний, возникающих при операциях ввода-вывода, планирование последовательности запросов на выполнение этих операций. Давайте остановимся на этих услугах подробнее.

3.1. Блокирующиеся, неблокирующиеся и асинхронные системные вызовы

Все системные вызовы, связанные с осуществлением операций ввода-вывода, можно разбить на три группы по способам реализации взаимодействия процесса и устройства ввода-вывода:

- К первой, наиболее привычной для большинства программистов группе относятся блокирующиеся системные вызовы. Как следует из самого названия, применение такого вызова приводит к блокировке инициировавшего его процесса, т. е. процесс переводится операционной системой из состояния исполнение в состояние ожидание. Завершив выполнение всех операций ввода-вывода, предписанных системным вызовом, операционная система переводит процесс из состояния ожидание в состояние готовность. После того как процесс будет снова выбран для исполнения, в нем произойдет окончательный возврат из системного вызова. Типичным для применения такого системного вызова является случай, когда процессу необходимо получить от устройства строго определенное количество данных, без которых он не может выполнять работу далее.

- Ко второй группе относятся неблокирующиеся системные вызовы. Их название не совсем точно отражает суть дела. В простейшем случае процесс, применивший неблокирующийся вызов, не переводится в состояние ожидание вообще. Системный вызов возвращается немедленно, выполнив предписанные ему операции ввода-вывода полностью, частично или не выполнив совсем, в зависимости от текущей ситуации (состояния устройства, наличия данных и т. д.). В более сложных ситуациях процесс может блокироваться, но условием его разблокирования является завершение всех необходимых операций или окончание некоторого промежутка времени. Типичным случаем применения неблокирующегося системного вызова может являться периодическая проверка на поступление информации с клавиатуры при выполнении трудоемких расчетов.

- К третьей группе относятся асинхронные системные вызовы. Процесс, использовавший асинхронный системный вызов, никогда в нем не блокируется. Системный вызов инициирует выполнение необходимых операций ввода-вывода и немедленно возвращается, после чего процесс продолжает свою регулярную деятельность. Об окончании завершения операции ввода-вывода операционная система впоследствии информирует процесс изменением значений некоторых переменных, передачей ему сигнала или сообщения или каким-либо иным способом. Необходимо четко понимать разницу между неблокирующимися и асинхронными вызовами. Неблокирующийся системный вызов для выполнения операции read вернется немедленно, но может прочитать запрошенное количество байтов, меньшее количество или вообще ничего. Асинхронный системный вызов для этой операции также вернется немедленно, но требуемое количество байтов рано или поздно будет прочитано в полном объеме.

3.2. Буферизация и кэширование

Под буфером обычно понимается некоторая область памяти для запоминания информации при обмене данных между двумя устройствами, двумя процессами или процессом и устройством. Обмен информацией между двумя процессами относится к области кооперации процессов, и мы подробно рассмотрели его организацию в

соответствующей лекции. Здесь нас будет интересовать использование буферов в том случае, когда одним из участников обмена является внешнее устройство. Существует три причины, приводящие к использованию буферов в базовой подсистеме ввода-вывода:

- Первая причина буферизации – это разные скорости приема и передачи информации, которыми обладают участники обмена. Рассмотрим, например, случай передачи потока данных от клавиатуры к модему. Скорость, с которой поставляет информацию клавиатура, определяется скоростью набора текста человеком и обычно существенно меньше скорости передачи данных модемом. Для того чтобы не занимать модем на все время набора текста, делая его недоступным для других процессов и устройств, целесообразно накапливать введенную информацию в буфере или нескольких буферах достаточного размера и отсылать ее через модем после заполнения буферов.

- Вторая причина буферизации – это разные объемы данных, которые могут быть приняты или получены участниками обмена единовременно. Возьмем другой пример. Пусть информация поставляется модемом и записывается на жесткий диск. Помимо обладания разными скоростями совершения операций, модем и жесткий диск представляют собой устройства разного типа. Модем является символьным устройством и выдает данные байт за байтом, в то время как диск является блочным устройством и для проведения операции записи для него требуется накопить необходимый блок данных в буфере. Здесь также можно применять более одного буфера. После заполнения первого буфера модем начинает заполнять второй, одновременно с записью первого на жесткий диск. Поскольку скорость работы жесткого диска в тысячи раз больше, чем скорость работы модема, к моменту заполнения второго буфера операция записи первого будет завершена, и модем снова сможет заполнять первый буфер одновременно с записью второго на диск.

- Третья причина буферизации связана с необходимостью копирования информации из приложений, осуществляющих ввод-вывод, в буфер ядра операционной системы и обратно. Допустим, что некоторый пользовательский процесс пожелал вывести информацию из своего адресного пространства на внешнее устройство. Для этого он должен выполнить системный вызов с обобщенным названием write, передав в качестве параметров адрес области памяти, где расположены данные, и их объем. Если внешнее устройство временно занято, то возможна ситуация, когда к моменту его освобождения содержимое нужной области окажется испорченным (например, при использовании асинхронной формы системного вызова). Чтобы избежать возникновения подобных ситуаций, проще всего в начале работы системного вызова скопировать необходимые данные в буфер ядра операционной системы, постоянно находящийся в оперативной памяти, и выводить их на устройство из этого буфера.

Под словом кэш (cache) обычно понимают область быстрой памяти, содержащую копию данных, расположенных где-либо в более медленной памяти, предназначенную для ускорения работы вычислительной системы. В базовой подсистеме ввода-вывода не следует смешивать два понятия, буферизацию и кэширование, хотя зачастую для выполнения этих функций отводится одна и та же область памяти. Буфер часто содержит единственный набор данных, существующий в системе, в то время как кэш по определению содержит копию данных, существующих где-нибудь еще. Например, буфер, используемый базовой подсистемой для копирования данных из пользовательского пространства процесса при выводе на диск, может в свою очередь применяться как кэш для этих данных, если операции модификации и повторного чтения данного блока выполняются достаточно часто.

Функции буферизации и кэширования не обязательно должны быть локализованы в базовой подсистеме ввода-вывода. Они могут быть частично реализованы в драйверах и даже в контроллерах устройств, скрытно по отношению к базовой подсистеме.

3.3. Spooling и захват устройств

О понятии spooling мы ранее говорили, как о механизме, впервые позволившем совместить реальные операции ввода-вывода одного задания с выполнением другого задания. Теперь мы можем определить это понятие более точно. Под словом spool мы подразумеваем буфер, содержащий входные или выходные данные для устройства, на котором следует избегать чередования его использования (возникновения interleaving) различными процессами. Правда, в современных вычислительных системах spool для ввода данных практически не используется, а в основном предназначен для накопления выходной информации.

Рассмотрим в качестве внешнего устройства принтер. Хотя принтер не может печатать информацию, поступающую одновременно от нескольких процессов, может оказаться желательным разрешить процессам совершать вывод на принтер параллельно. Для этого операционная система вместо передачи информации напрямую на принтер накапливает выводимые данные в буферах на диске, организованных в виде отдельного spool-файла для каждого процесса. После завершения некоторого процесса соответствующий ему spool-файл ставится в очередь для реальной печати. Механизм, обеспечивающий подобные действия, и получил название spooling.

В некоторых операционных системах вместо использования spooling для устранения race condition применяется механизм монопольного захвата устройств процессами. Если устройство свободно, то один из процессов может получить его в монопольное распоряжение. При этом все другие процессы при попытке осуществления операций над этим устройством будут либо блокированы (переведены в состояние ожидание), либо получат информацию о невозможности выполнения операции до тех пор, пока процесс, захвативший устройство, не завершится или явно не сообщит операционной системе о своем отказе от его использования.

Обеспечение spooling и механизма захвата устройств является прерогативой базовой подсистемы ввода-вывода.

3.4. Обработка прерываний и ошибок

Если при работе с внешним устройством вычислительная система не пользуется методом опроса его состояния, а задействует механизм прерываний, то при возникновении прерывания, как мы уже говорили раньше, процессор, частично сохранив свое состояние, передает управление специальной программе обработки прерывания. Мы уже рассматривали действия операционной системы над процессами, происходящими при возникновении прерывания, где после возникновения прерывания осуществлялись следующие действия: сохранение контекста, обработка прерывания, планирование использования процессора, восстановление контекста. Тогда мы обращали больше внимания на действия, связанные с сохранением и восстановлением контекста и планированием использования процессора. Теперь давайте подробнее остановимся на том, что скрывается за словами «обработка прерывания».

Одна и та же процедура обработки прерывания может применяться для нескольких устройств ввода-вывода (например, если эти устройства используют одну линию прерываний, идущую от них к контроллеру прерываний), поэтому первое действие собственно программы обработки состоит в определении того, какое именно устройство выдало прерывание. Зная устройство, мы можем выявить процесс, который инициировал выполнение соответствующей операции. Поскольку прерывание возникает как при удачном, так и при неудачном ее выполнении, следующее, что мы должны сделать, – это определить успешность завершения операции, проверив значение бита ошибки в регистре состояния устройства. В некоторых случаях операционная система может предпринять определенные действия, направленные на компенсацию возникшей ошибки. Например, в случае возникновения ошибки чтения с гибкого диска можно попробовать несколько раз

повторить выполнение команды. Если компенсация ошибки невозможна, то операционная система впоследствии известит об этом процесс, запросивший выполнение операции, (например, специальным кодом возврата из системного вызова). Если этот процесс был заблокирован до выполнения завершившейся операции, то операционная система переводит его в состояние готовность. При наличии других неудовлетворенных запросов к освободившемуся устройству операционная система может инициировать выполнение следующего запроса, одновременно известив устройство, что прерывание обработано. На этом, собственно, обработка прерывания заканчивается, и система может приступать к планированию использования процессора.

Действия по обработке прерывания и компенсации возникающих ошибок могут быть частично переложены на плечи соответствующего драйвера. Для этого в состав интерфейса между драйвером и базовой подсистемой ввода-вывода добавляют еще одну функцию – функцию обработки прерывания intr.

3.5. Планирование запросов

При использовании неблокирующегося системного вызова может оказаться, что нужное устройство уже занято выполнением некоторых операций. В этом случае неблокирующийся вызов может немедленно вернуться, не выполнив запрошенных команд. При организации запроса на совершение операций ввода-вывода с помощью блокирующегося или асинхронного вызова занятость устройства приводит к необходимости постановки запроса в очередь к данному устройству. В результате с каждым устройством оказывается связан список неудовлетворенных запросов процессов, находящихся в состоянии ожидания, и запросов, выполняющихся в асинхронном режиме. Состояние ожидание расщепляется на набор очередей процессов, дожидающихся различных устройств ввода-вывода (или ожидающих изменения состояний различных объектов – семафоров, очередей сообщений, условных переменных в мониторах и т. д.).

После завершения выполнения текущего запроса операционная система (по ходу обработки возникшего прерывания) должна решить, какой из запросов в списке должен быть удовлетворен следующим, и инициировать его исполнение. Точно так же, как для выбора очередного процесса на исполнение из списка готовых нам приходилось осуществлять краткосрочное планирование процессов, здесь нам необходимо осуществлять планирование применения устройств, пользуясь каким-либо алгоритмом этого планирования. Критерии и цели такого планирования мало отличаются от критериев и целей планирования процессов.

Задача планирования использования устройства обычно возлагается на базовую подсистему ввода-вывода, однако для некоторых устройств лучшие алгоритмы планирования могут быть тесно связаны с деталями их внутреннего функционирования. В таких случаях операция планирования переносится внутрь драйвера соответствующего устройства, так как эти детали скрыты от базовой подсистемы. Для этого в интерфейс драйвера добавляется еще одна специальная функция, которая осуществляет выбор очередного запроса, – функция strategy.

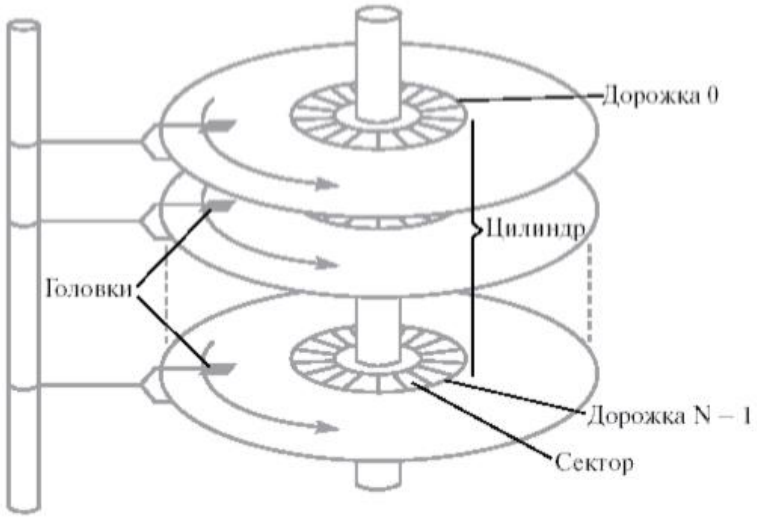
В следующем разделе мы рассмотрим некоторые алгоритмы планирования, связанные с удовлетворением запросов, на примере жесткого диска.

4. Алгоритмы планирования запросов к жесткому диску

Прежде чем приступить к непосредственному изложению самих алгоритмов, давайте вспомним внутреннее устройство жесткого диска и определим, какие параметры запросов мы можем использовать для планирования.

4.1. Строение жесткого диска и параметры планирования

Современный жесткий магнитный диск представляет собой набор круглых пластин, находящихся на одной оси и покрытых с одной или двух сторон специальным магнитным слоем (рис).



Около каждой рабочей поверхности каждой пластины расположены магнитные головки для чтения и записи информации. Эти головки присоединены к специальному рычагу, который может перемещать весь блок головок над поверхностями пластин как единое целое. Поверхности пластин разделены на концентрические кольца, внутри которых, собственно, и может храниться информация. Набор концентрических колец на всех пластинах для одного положения головок (т. е. все кольца, равноудаленные от оси) образует цилиндр. Каждое кольцо внутри цилиндра получило название дорожки (по одной или две дорожки на каждую пластину). Все дорожки делятся на равное число секторов. Количество дорожек, цилиндров и секторов может варьироваться от одного жесткого диска к другому в достаточно широких пределах. Как правило, сектор является минимальным объемом информации, которая может быть прочитана с диска за один раз.

При работе диска набор пластин вращается вокруг своей оси с высокой скоростью, подставляя по очереди под головки соответствующих дорожек все их сектора. Номер сектора, номер дорожки и номер цилиндра однозначно определяют положение данных на жестком диске и, наряду с типом совершаемой операции – чтение или запись, полностью характеризуют часть запроса, связанную с устройством, при обмене информацией в объеме одного сектора.

При планировании использования жесткого диска естественным параметром является время, которое потребуется для выполнения очередного запроса. Время, необходимое для чтения или записи определенного сектора на определенной дорожке определенного цилиндра, можно разделить на две составляющие: время обмена информацией между магнитной головкой и компьютером, которое обычно не зависит от положения данных и определяется скоростью их передачи (transfer speed), и время, необходимое для позиционирования головки над заданным сектором, – время позиционирования (positioning time). Время позиционирования, в свою очередь, состоит из времени, необходимого для перемещения головок на нужный цилиндр, – времени поиска (seek time) и времени, которое требуется для того, чтобы нужный сектор довернулся под головку, – задержки на вращение (rotational latency). Времена поиска пропорциональны разнице между номерами цилиндров предыдущего и планируемого запросов, и их легко сравнивать. Задержка на вращение определяется довольно сложными соотношениями между номерами цилиндров и секторов предыдущего и планируемого запросов и скоростями вращения диска и перемещения головок. Без знания соотношения этих

скоростей сравнение становится невозможным. Поэтому естественно, что набор параметров планирования сокращается до времени поиска различных запросов, определяемого текущим положением головки и номерами требуемых цилиндров, а разницей в задержках на вращение пренебрегают.

4.2. Алгоритм First Come First Served (FCFS)

Простейшим алгоритмом является алгоритм First Come First Served (FCFS) – первым пришел, первым обслужен. Все запросы организуются в очередь FIFO и обслуживаются в порядке поступления. Алгоритм прост в реализации, но может приводить к достаточно длительному общему времени обслуживания запросов. Рассмотрим пример. Пусть у нас на диске из 100 цилиндров (от 0 до 99) есть следующая очередь запросов: 23, 67, 55, 14, 31, 7, 84, 10 и головки в начальный момент находятся на 63-м цилиндре. Тогда положение головок будет меняться следующим образом:

63→23→67→55→14→31→7→84→10

и всего головки переместятся на 329 цилиндров. Неэффективность алгоритма хорошо иллюстрируется двумя последними перемещениями с 7 цилиндра через весь диск на 84 цилиндр и затем опять через весь диск на цилиндр 10. Простая замена порядка двух последних перемещений (7→10→84) позволила бы существенно сократить общее время обслуживания запросов. Поэтому давайте перейдем к рассмотрению другого алгоритма.

4.3. Алгоритм Short Seek Time First (SSTF)

Как мы убедились, достаточно разумным является первоочередное обслуживание запросов, данные для которых лежат рядом с текущей позицией головок, а уж затем далеко отстоящих. Алгоритм Short Seek Time First (SSTF) – короткое время поиска первым – как раз и исходит из этой позиции. Для очередного обслуживания будем выбирать запрос, данные для которого лежат наиболее близко к текущему положению магнитных головок. Естественно, что при наличии равноудаленных запросов решение о выборе между ними может приниматься исходя из различных соображений, например по алгоритму FCFS. Для предыдущего примера алгоритм даст такую последовательность положений головок:

63→67→55→31→23→14→10→7→84

Головки переместятся на 141 цилиндр. Заметим, что наш алгоритм похож на алгоритм SJF планирования процессов, если за аналог оценки времени очередного CPU burst процесса выбирать расстояние между текущим положением головки и положением, необходимым для удовлетворения запроса. И точно так же, как алгоритм SJF, он может приводить к длительному откладыванию выполнения какого-либо запроса. Необходимо вспомнить, что запросы в очереди могут появляться в любой момент времени. Если у нас все запросы, кроме одного, постоянного, группируются в области с большими номерами цилиндров, то этот один запрос может находиться в очереди неопределенно долго.

Точный алгоритм SJF являлся оптимальным для заданного набора процессов с заданными временами CPU burst. Очевидно, что алгоритм SSTF не является оптимальным. Если мы перенесем обслуживание запроса 67-го цилиндра в промежуток между запросами 7-го и 84-го цилиндров, мы уменьшим общее время обслуживания. Это наблюдение приводит нас к идее целого семейства других алгоритмов – алгоритмов сканирования.

4.4. Алгоритмы сканирования (SCAN, C-SCAN, LOOK, C-LOOK)

В простейшем из алгоритмов сканирования – SCAN – головки постоянно перемещаются от одного края диска до другого, по ходу дела обслуживая все встречающиеся запросы. По достижении другого края направление движения меняется, и все повторяется снова. Пусть в предыдущем примере в начальный момент времени головки двигаются в направлении уменьшения номеров цилиндров. Тогда мы и получим порядок обслуживания запросов, подсмотренный в конце предыдущего раздела. Последовательность перемещения головок выглядит следующим образом:

63→55→31→23→14→10→→7→0→67→84

Головки переместятся на 147 цилиндров.

Если мы знаем, что обслужили последний попутный запрос в направлении движения головок, то мы можем не доходить до края диска, а сразу изменить направление движения на обратное:

63→55→31→23→14→10→7→67→84

Головки переместятся на 133 цилиндра. Полученная модификация алгоритма SCAN получила название LOOK.

Допустим, что к моменту изменения направления движения головки в алгоритме SCAN, т.е. когда головка достигла одного из краев диска, у этого края накопилось большое количество новых запросов, на обслуживание которых будет потрачено достаточно много времени (не забываем, что надо не только перемещать головку, но еще и передавать прочитанные данные). Тогда запросы, относящиеся к другому краю диска и поступившие раньше, будут ждать обслуживания несправедливо долго. Для сокращения времени ожидания запросов применяется другая модификация алгоритма SCAN – циклическое сканирование. Когда головка достигает одного из краев диска, она без чтения попутных запросов (иногда существенно быстрее, чем при выполнении обычного поиска цилиндра) перемещается на другой край, откуда вновь начинает движение в прежнем направлении. Для этого алгоритма, получившего название C-SCAN, последовательность перемещений будет выглядеть так:

63→55→31→23→14→10→7→0→99→84→67

По аналогии с алгоритмом LOOK для алгоритма SCAN можно предложить и алгоритм CLOOK для алгоритма C-SCAN:

63→55→31→23→14→10→7→84→67

Существуют и другие разновидности алгоритмов сканирования.