**7.1. Устройства ввода-вывода**

Внешние устройства, выполняющие операции ввода-вывода, можно разделить на три группы:

* устройства, работающие с пользователем. Используются для связи пользователя с компьютером. Сюда относятся принтеры, дисплеи, клавиатура, манипуляторы (мышь, трекбол, джойстики) и т.п.;
* устройства, работающие с компьютером. Используются для связи с электронным оборудованием. К ним можно отнести дисковые устройства и устройства с магнитными лентами, датчики, контроллеры, преобразователи;
* коммуникации. Используются для связи с удаленными устройствами. К ним относятся модемы и адаптеры цифровых линий.

По другому признаку устройства ввода-вывода можно разделить на блочные и символьные [10]. Блочными являются устройства, хранящие информацию в виде блоков фиксированного размера, причем у каждого блока есть адрес и каждый блок может быть прочитан независимо от остальных блоков. *Символьные устройства* принимают или передают поток символов без какой-либо блочной структуры (принтеры, сетевые карты, мыши и т.д.).

Однако некоторые из устройств не попадают ни в одну из этих категорий, например, часы, мониторы и др. И все же модель блочных и символьных устройств является настолько общей, что может использоваться в качестве основы для достижения независимости от устройств некоторого программного обеспечения операционных систем, имеющего дело с вводом-выводом. Например, файловая система имеет дело с абстрактными *блочными устройствами*, а зависимую от устройств часть оставляет программному обеспечению низкого уровня.

Следует также отметить существенные различия между устройствами ввода-вывода, принадлежащими к разным классам, и в рамках каждого класса. Эти различия касаются следующих характеристик:

* скорость передачи данных (различия на несколько порядков);
* применение. Каждое действие, поддерживаемое устройством, оказывает влияние на программное обеспечение и стратегии операционной системы (например, диск, используемый для хранения файлов или для страниц виртуальной памяти, требует различного программного обеспечения);
* сложность управления. Для принтера требуется относительно простой интерфейс управления, для диска – намного сложнее. Влияния этих отличий на ОС сглаживается усложнением контроллеров ввода-вывода;
* единицы передачи данных. Данные могут передаваться блоками или потоками байтов или символов;
* представления данных. Различные устройства используют разные схемы кодирования данных, включая разную кодировку символов и контроль четности;
* условия ошибки. Природа ошибок, способ сообщения о них, их последствия и возможные ответы резко отличаются при переходе от одного устройства к другому.

Такое разнообразие внешних устройств приводит, по сути, к невозможности разработки единого и согласованного подхода к проблеме ввода-вывода как с точки зрения операционной системы, так и с точки зрения пользовательских процессов.

Устройства ввода-вывода, как правило, состоят из электромеханической и электронной части. Обычно их выполняют в форме отдельных модулей – собственно устройство и контроллер (адаптер). В ПК контроллер принимает форму платы, вставляемой в слот расширения. Плата имеет разъем, к которому подключается кабель, ведущий к самому устройству. Многие контроллеры способны управлять двумя, четырьмя и даже более идентичными устройствами. Интерфейс между контроллером и устройством является официальным стандартом (ANSI, IEEE или ISO) или фактическим стандартом, и различные компании могут выпускать отдельно котроллеры и устройства, удовлетворяющие данному интерфейсу. Так, многие компании производят диски, соответствующие интерфейсу IDE или SCSI, а наборы схем системной логики материнских плат реализуют IDE и SCSI-контроллеры.

Интерфейс между контроллером и устройством часто является интерфейсом очень низкого уровня, т.е. очень специфичным, зависящим от типа внешнего устройства. Например, видеоконтроллер считывает из памяти байты, содержащие символы, которые следует отобразить, и формирует сигналы управления лучом электронной трубки, сигналы строчной и кадровой развертки и т.п.

Каждый контроллер взаимодействует с драйвером системным программным модулем, предназначенным для управления данным устройством. Для работы с драйвером контроллер имеет несколько регистров, кроме того, он может иметь буфер данных, из которого операционная система может читать данные, а также записывать данные в него. Каждому управляющему регистру назначается номер порта ввода-вывода. Используя регистры контроллера, ОС может узнать состояние устройства (например, готово ли оно к работе), а также выдавать команды управления устройством (принять или передать данные, включиться, выключиться и т.п.).

**7.2. Назначение, задачи и технологии подсистемы ввода-вывода**

Обмен данными между пользователями, приложениями и периферийными устройствами компьютера выполняет специальная подсистема ОС – подсистема ввода-вывода. Собственно, для выполнения этой задачи и были разработаны первые системные программы, послужившие прототипами операционных систем.

Основными компонентами подсистемы ввода-вывода являются драйверы, управляющие внешними устройствами, и файловая система. В работе подсистемы ввода-вывода активно участвует диспетчер прерываний. Более того, основная нагрузка диспетчера прерываний обусловлена именно подсистемой ввода-вывода, поэтому диспетчер прерываний иногда считают частью подсистемы ввода-вывода.

Файловая система – это основное хранилище информации в любом компьютере. Она активно использует остальные части подсистемы ввода-вывода. Кроме того, модель файла лежит в основе большинства механизмов доступа к периферийным устройствам.

На подсистему ввода-вывода возлагаются следующие функции [5, 17]:

* организация параллельной работы устройств ввода-вывода и процессора;
* согласование скоростей обмена и кэширование данных;
* разделение устройств и данных между процессами (выполняющимися программами);
* обеспечение удобного логического интерфейса между устройствами и остальной частью системы;
* поддержка широкого спектра драйверов с возможностью простого включения в систему нового драйвера;
* динамическая загрузка и выгрузка драйверов без дополнительных действий с операционной системой;
* поддержка нескольких различных файловых систем;
* поддержка синхронных и асинхронных операций ввода-вывода.

Эволюция ввода-вывода может быть представлена следующими этапами [17].

1. Процессор непосредственно управляет периферийным устройством.
2. Устройство управляется контроллером. Процессор использует программируемый ввод-вывод без прерываний (переход к абстракции интерфейса ввода-вывода).
3. Использование контроллера прерываний. Ввод-вывод, управляемый прерываниями.
4. Использование модуля (канала) прямого доступа к памяти. Перемещение данных в память (из нее) без применения процессора.
5. Использование отдельного специализированного процессора ввода-вывода, управляемого центральным процессором.
6. Использование отдельного компьютера для управления устройствами ввода-вывода при минимальном вмешательстве центрального процессора.

Проследив описанный путь развития устройств ввода-вывода, можно заметить, что вмешательство процессора в функции ввода-вывода становится все менее заметным. Центральный процессор все больше освобождается от задач, связанных с вводом-выводом, что приводит к повышению общей производительности компьютерной системы.

Для персональных компьютеров операции ввода-вывода могут выполняться тремя способами.

1. С помощью программируемого ввода-вывода. В этом случае, когда процессору встречается команда, связанная с вводом-выводом, он выполняет ее, посылая соответствующие команды контроллеру ввода-вывода. Это устройство выполняет требуемое действие, а затем устанавливает соответствующие биты в регистрах состояния ввода-вывода и не посылает никаких сигналов, в том числе сигналов прерываний. Процессор периодически проверяет состояние модуля ввода-вывода с целью проверки завершения операции ввода-вывода.

Таким образом, процессор непосредственно управляет операциями ввода-вывода, включая опознание состояния устройства, пересылку команд чтения-записи и передачу данных. Процессор посылает необходимые команды контроллеру ввода-вывода и переводит текущий процесс в состояние ожидания завершения операции ввода-вывода. Недостатки такого метода – большие потери процессорного времени, связанные с управлением вводом-выводом.

1. Ввод-вывод, управляемый прерываниями. Процессор посылает необходимые команды контроллеру ввода-вывода и продолжает выполнять текущий процесс, если нет необходимости в ожидании выполнения операции ввода-вывода. В противном случае текущий процесс приостанавливается до получения *сигнала прерывания* о завершении ввода-вывода, а процессор переключается на выполнение другого процесса. Наличие прерываний процессор проверяет в конце каждого цикла выполняемых команд.

Такой ввод-вывод намного эффективнее, чем программируемый ввод-вывод, так как при этом исключается ненужное ожидание с бесполезным простоем процессора. Однако и в этом случае ввод-вывод потребляет еще значительное количество процессорного времени, потому что каждое слово, которое передается из памяти в модуль ввода-вывода (контроллер) или обратно, должно пройти через процессор.

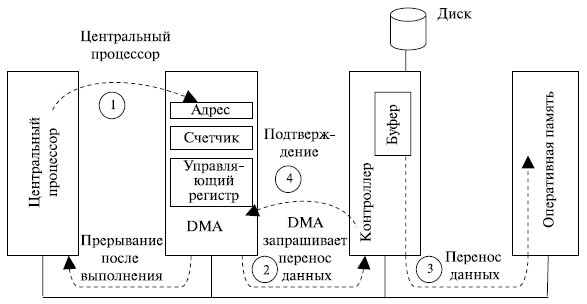
1. Прямой доступ к памяти (*direct memory access* – DMA). В этом случае специальный модуль прямого доступа к памяти управляет обменом данных между основной памятью и контроллером ввода-вывода. Процессор посылает запрос на передачу блока данных модулю прямого доступа к памяти, а прерывание происходит только после передачи всего блока данных.

В настоящее время в персональных и других компьютерах используется третий способ ввода-вывода, поскольку в структуре компьютера имеется DMA-контроллер или подобное ему устройство, обслуживающее, как правило, запросы по передаче данных от нескольких устройств ввода-вывода на конкурентной основе.

DMA-контроллер имеет доступ к системной шине независимо от центрального процессора, как показано на рис. 7.1. Контроллер содержит несколько регистров, доступных центральному процессу для чтения и записи (*регистр адреса* памяти, счетчик байтов, управляющие регистры). Управляющие регистры задают порт ввода-вывода, который должен быть использован, направление переноса данных (чтение или запись в устройство ввода-вывода), единицу переноса (побайтно, пословно), а также число байтов, которые следует перенести за одну операцию.

Перед выполнением операции обмена ЦП программирует DMA-контроллер, устанавливая его регистры (шаг 1 на рис. 7.1). Затем ЦП дает команду дисковому контролеру прочитать внести данные во внутренний буфер и проверить контрольную сумму. После этого процессор продолжает свою работу. Когда данные получены и проверены контроллером диска, DMA может начинать работу.

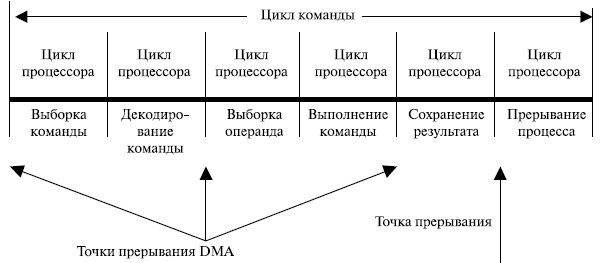
DMA-контроллер начинает перенос данных, посылая дисковому контроллеру по шине запрос чтения (шаг 2). Адрес памяти уже находится на адресной шине, так что контроллер знает, куда пересылать следующее слово из своего буфера. Запись в память является еще одним стандартным циклом шины (шаг 3). Когда запись закончена, контроллер диска посылает сигнал подтверждения контролеру DMA (шаг 4). Затем контроллер DMA увеличивает используемый адрес памяти и уменьшает значение счетчика байтов. После этого шаги 2, 3 и 4 повторяются, пока значение счетчика не станет равным нулю. По завершению цикла копирования контроллер DMA инициирует прерывание процессора, сообщая ему о завершении операции ввода-вывода.



**Рис. 7.1.**  Работа DMA

Необходимо обратить внимание на работу шины в этом процессе обмена данными. Шина может работать в двух режимах: пословном и поблочном. В первом случае контроллер DMA выставляет запрос на перенос одного слова и получает его. Если процессору также нужна эта шина (не забывайте, в основном он работает с кэш-памятью), ему приходится подождать. Этот механизм называется захватом цикла, потому, что контроллер устройства периодически забирает случайный цикл шины у центрального процессора, слегка тормозя его.

Ниже на рис. 7.2 показана позиция *цикла команд*, в которых работа процессора может быть приостановлена. В любом случае приостановка процессора происходит только при необходимости использования шины. После этого устройство DMA выполняет передачу слова и возвращает управление процессору. Однако это не является прерыванием: процессор не сохраняет контекст с переходом к выполнению другого задания. Он просто делает паузу на время одного цикла шины.



**Рис. 7.2.**  Точки прерывания DMA

В блочном режиме работы контроллер DMA занимает шину на серию пересылок (пакет). Этот режим более эффективен, однако при переносе большого блока центральный процессор и другие устройства могут быть заблокированы на существенный промежуток времени.

При большом количестве устройств ввода-вывода от подсистемы ввода-вывода требуется спланировать в реальном масштабе времени (в котором работают внешние устройства) запуск и приостановку большего количества разных драйверов, обеспечив при этом время реакции каждого драйвера на независимые события контролеров внешних устройств. С другой стороны, необходимо минимизировать загрузку процессора задачами ввода-вывода.

Решение этих задач достигается на основе многоуровневой приоритетной схемы обслуживания прерываний. Для обеспечения приемлемого уровня реакции все драйверы распределяются по нескольким приоритетным уровням в соответствии с требованиями по времени реакции и временем использования процессора. Для реализации приоритетной схемы задействуется общий диспетчер прерываний ОС.

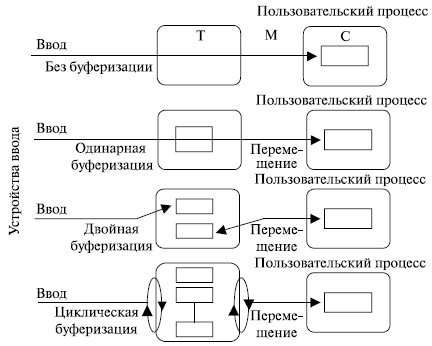
**7.3. Согласование скоростей обмена и кэширования данных**

При обмене данными всегда возникает задача согласования скоростей работы устройств. Решение этой задачи достигается буферизацией данных [10, 17]. В подсистеме ввода-вывода часто используется буферизация в оперативной памяти. Однако буферизация только на основе оперативной памяти часто оказывается недостаточной из-за большой разницы скоростей работы оперативной памяти, и внешнего устройства объема оперативной памяти может просто не хватить. В этих случаях часто используют в качестве буфера дисковый файл, называемый спул-файлом. Типичный пример применения спулинга – вывод данных на принтер (для печатаемых документов объем в несколько Мбайт – не редкость, поэтому временное хранение такого файла в течение десятков минут в оперативной памяти нецелесообразно).

Другим решением проблемы является использование большой буферной памяти в контроллерах внешних устройств. Такой подход полезен в тех случаях, когда помещение данных на диск слишком замедляет обмен (или когда данные выводятся на сам диск). Например, в контроллерах графических дисплеев применяется буферная память, соизмеримая по обмену с оперативной памятью, и это существенно ускоряет вывод графика на экран.

При рассмотрении различных методов буферизации нужно учитывать, что существует, как отмечалось, два типа устройств – блочные и символьные. Первые сохраняют информацию блоками фиксированного размера и передают поблочно (диски, ленты). Вторые выполняют передачу в виде неструктурированных потоков байтов (терминалы, принтеры, манипулятор мыши, сканеры и др.).

Возможные схемы буферизации ввода-вывода приведены на рис. 7.3.



**Рис. 7.3.**  Варианты буферизации

Простейший тип поддержки со стороны ОС – одинарный буфер. В тот момент, когда пользовательский процесс выполняет запрос ввода-вывода, операционная система назначает ему буфер в системной части оперативной памяти. Работа одинарного буфера для блочно-ориентированных устройств может быть описана следующим образом. Сначала осуществляется передача входных данных в системный буфер. Когда она завершается, процесс перемещает блок в пользовательское пространство и немедленно производит запрос следующего блока. Такая процедура называется опережающим считыванием, или упреждающим вводом.

Подобный подход по сравнению с отсутствием буферизации обеспечивает повышение быстродействия, поскольку пользовательский процесс может обрабатывать один блок данных в то время, когда происходит считывание следующего блока.

Пусть Т – время, необходимое для ввода одного блока, а С – для вычислений, выполняющихся между запросами на ввод-вывод. Без буферизации время выполнения, приходящееся на один блок, будет равно Т + С. При использовании одинарной буферизации время будет равно max [С, T] + M, где М – время перемещения данных из системного буфера в пользовательскую память. В большинстве случаев T + C > max [C, T] + M.

Схема одинарного буфера может быть применена и при поточно-ориентированном вводе-выводе – построчно или побайтно ( в строчных принтерах, терминалах и др.). Например, при операции вывода пользовательский процесс может разместить в буфере строку и продолжить работу. Улучшить схему одинарной буферизации можно путем использования двух буферов. Теперь процесс выполняет передачу данных в один буфер (или считывает из него), в то время как ОС освобождает (или заполняет) другой. Эта технология известна как двойная буферизация, или сменный буфер.

Время выполнения при блочно-ориентированной передаче можно грубо оценить как max [C, T]. Таким образом, если C <= T, то блочно-ориентированное устройство может работать с максимальной скоростью. Если C > T, то двойная буферизация избавляет процесс от необходимости ожидания завершения ввода-вывода.

Двойной буферизации может оказаться недостаточно, если процесс часто выполняет ввод или вывод. Решить проблему помогает наращивание количества буферов. Если буфер больше двух, схема именуется циклической буферизацией.

Буферизация данных позволяет не только согласовать скорости работы процессора и внешних устройств, но и решить другую задачу – сократить количество реальных операций ввода-вывода за счет кэширования данных. Дисковый кэш является непременным атрибутом подсистем ввода-вывода практически всех операционных систем и значительно сокращает время доступа к хранимым данным.

**7.4. Разделение устройств и данных между процессами**

Устройства ввода-вывода могут предоставляться процессам как в монопольном, так и разделенном режиме. При этом ОС должна обеспечивать контроль доступа теми же способами, что и при доступе процессов к другим ресурсам вычислительной системы, – путем проверки прав пользователя или группы пользователей, от имени которых действует процесс, на выполнение той или иной операции над устройством.

ОС может контролировать доступ не только к устройству в целом, но и к отдельным порциям данных, хранимых этим устройством. Диск является типичным примером такого устройства, где важно контролировать доступ к файлам и каталогам. В последнем случае непременным является задание режима совместного использования устройства в целом.

Одно и то же устройство в разные периоды времени может работать как в разделяемом, так и в монопольном режимах. Тем не менее, существуют устройства, для которых характерен один из этих режимов, например, последовательные порты и алфавитно-цифровые терминалы чаще используются в монопольном режиме, а диск – в режиме совместного доступа.

В случае совместного использования ОС должна оптимизировать последовательность операций ввода-вывода для различных процессов в целях повышения общей производительности. Например, при обмене данными нескольких процессов с диском можно так упорядочить последовательность операций, что непроизводительные затраты времени на перемещение головок существенно уменьшаются (при этом для отдельных процессов возможно некоторое замедление операции ввода-вывода).

При разделении устройства между процессами может возникнуть необходимость в разграничении данных процессов друг от друга. Обычно такая потребность появляется при совместном использовании последовательных устройств, которые, в отличие от устройств прямого доступа, не адресуются. Типичный представитель такого устройства – принтер. Для таких устройств организуется очередь заданий на вывод, при этом каждое задание представляет собой порцию данных, которую нельзя разрывать, например, документ для печати.

Для хранения очереди заданий используется спул-файл, который согласует скорость работы принтера и оперативной памяти и позволяет организовать разбиение данных на логические порции. Процессы могут одновременно выполнять вывод на принтер, помещая данные в свой раздел спул-файла.

**7.5. Обеспечение логического интерфейса между устройствами и системой**

Разнообразие устройств ввода-вывода делает актуальной функцию операционной системы по созданию экранирующего логического интерфейса между периферийными устройствами и приложениями.

Практически все современные ОС поддерживают в качестве такого интерфейса файловую модель периферийных устройств, когда любое устройство выглядит для прикладного программиста последовательным набором байт, с которым можно работать с помощью унифицированных системных вызовов (например, read, write), задавая имя файла-устройства и смещение от начала последовательности байт.

Привлекательность модели файла-устройства состоит в ее простоте и унифицированности для устройств любого типа, однако во многих случаях для программирования операций ввода-вывода некоторого устройства она является слишком бедной. Поэтому данная модель часто используется в качестве базиса, над которым подсистема ввода-вывода строит более содержательную модель устройства конкретного типа.

**7.6. Поддержка широкого спектра драйверов**

Разнообразный набор драйверов для широкого круга популярных периферийных устройств – непременное условие популярности ОС у пользователей.

Для разработки драйверов производителями внешних устройств необходимо наличие четкого, удобного, открытого и хорошо документированного интерфейса между драйверами и другими компонентами ОС. Драйвер взаимодействует, с одной стороны, с модулями ядра ОС (модулями подсистемы ввода-вывода, модулями системных вызовов, модулями подсистем управления процессами и памятью), а с другой стороны – с контроллерами внешних устройств. Поэтому существует два вида интерфейсов: интерфейс "драйвер-ядро" (Driver Kernel Interface, DKI) и интерфейс "драйвер-устройство" (Driver Device Interface).

Интерфейс "драйвер-ядро" должен быть стандартизован в любом случае. Подсистема ввода-вывода может поддерживать несколько различных интерфейсов DKI/DDI, предоставляя специфический интерфейс для устройств определенного класса. К наиболее общим классам относятся *блочные устройства*, например, диски, и *символьные устройства*, такие как клавиатура и принтеры. Может существовать класс сетевых адаптеров и др. В большинстве современных ОС определен стандартный интерфейс, который должен поддерживать все блочные драйверы, и второй стандартный интерфейс, поддерживаемый всеми символьными адаптерами. Эти интерфейсы включают наборы процедур, которые могут вызываться остальной операционной системой для обращения к драйверу. К этим процедурам относятся, например, процедуры чтения блока или записи символьной строки.

Кроме того, подсистема ввода-вывода поддерживает большое количество системных функций, которые драйвер может вызывать для выполнения некоторых типовых действий. Например, это операции обмена с регистрами контроллера, ведения буферов промежуточного хранения данных ввода-вывода, взаимодействия с DMA-контроллером и контроллером прерываний и др.

У драйверов устройств есть множество функций, перечисленных ниже [17].

1. Обработка запросов записи-чтения от программного обеспечения управления устройствами. *Постановка запросов в очередь*.
2. Проверка входных параметров запросов и обработка ошибок.
3. Инициализация устройства и проверка статуса устройства.
4. Управление энергопотреблением устройства.
5. Регистрация событий в устройстве.
6. Выдача команд устройству и ожидание их выполнения, возможно, в блокированном состоянии, до поступления прерывания от устройства.
7. Проверка правильности завершения операции.
8. Передача запрошенных данных и статуса завершенной операции.
9. Обработка нового запроса при незавершенном предыдущем запросе (для реентерабельных драйверов).

Наиболее очевидная функция состоит в обработке абстрактных запросов чтения и записи независимого от устройств программного обеспечения, расположенного над ними. Но, кроме этого, они должны выполнять еще несколько функций. Например, драйвер должен при необходимости инициализировать устройство. Ему может понадобиться управлять энергопотреблением устройства и регистрацией событий.

Многие драйверы обладают сходной общей структурой. Типичный драйвер начинает работу с проверки входных параметров. Если они не удовлетворяют определенным критериям, драйвер возвращает ошибку. В противном случае драйвер преобразует абстрактные термины в конкретные. Например, дисковый драйвер преобразует линейный номер кластера в номер головки, дорожки и сектора.

Затем драйвер может проверить, не используется ли это устройство в данный момент. Если устройство занято, запрос может быть поставлен в очередь. Если устройство свободно, проверяется статус устройства, чтобы понять, может ли запрос быть обслужен прямо сейчас. Может оказаться необходимым включить устройство или запустить двигатель, прежде чем начнется перенос данных. Как только устройство включено и готово, начинается собственно управление устройством.

Управление устройством подразумевает выдачу ему серии команд. Именно в драйвере определяется последовательность команд в зависимости от того, что должно быть сделано. Определившись с командой, драйвер начинает записывать их в регистры *контроллера устройства*. После записи каждой команды в контроллер, возможно, будет нужно проверить, принял ли контроллер команду и готов ли принять следующую. Такая последовательность действий продолжается до тех пор, пока контроллеру не будут переданы все команды. Некоторые контроллеры способны принимать связные списки команд, находящихся в памяти. Они сами считывают и выполняют их без дальнейшей помощи ОС.

После того как драйвер передал все команды контроллеру, ситуация может развиваться по двум сценариям. Во многих случаях драйвер устройства должен ждать, пока контроллер не выполнит для него определенную работу, поэтому он блокируется до тех пор, пока прерывание от устройства не разблокирует его. В других случаях операция завершается без задержек и драйверу не нужно блокироваться. В любом случае по завершении выполнения операции драйвер должен проверить, завершилась ли операция без ошибок. Если все в порядке, драйверу, возможно, придется передать данные (например, только что прочитанный блок) независимому от устройств программному обеспечению. Наконец, драйвер возвращает информацию о состоянии для информирования вызывающей программы о*статусе завершения* операции. Если в очереди находились другие запросы, один из них теперь может быть выбран и запущен. В противном случае драйвер блокируется в ожидании следующего запроса.

Для поддержки процесса разработки драйверов операционной системы выпускается так называемый пакет *DDK* (Driver Development Kit), представляющий собой набор инструментальных средств-библиотек, компиляторов и отладчиков.

**7.7. Динамическая загрузка и выгрузка драйверов**

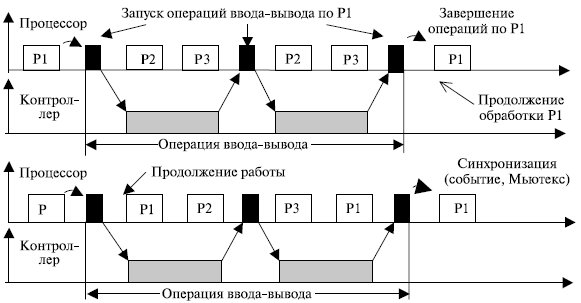
Так как набор потенциально поддерживаемых данных ОС периферийных устройств всегда шире набора устройств, которыми ОС должна управлять при установке на конкретной машине, то ценным свойством ОС является возможность динамически загружать в оперативную память требуемый драйвер (без остановки ОС) и выгружать его, если надобность в драйвере отпала. Такое свойство ОС может существенно сэкономить *системную область* памяти.

Альтернативой динамической загрузке драйверов при изменении текущей конфигурации внешних устройств компьютера является повторная компиляция кода ядра с требуемым набором драйверов, что создает между всеми компонентами ядра статические связи вместо динамических. Например, таким образом решалась данная проблема в ранних версиях ОС UNIX. При статистических вызовах между ядром и драйверами структура ОС упрощается, но этот подход требует наличия исходных кодов модулей ОС, доступность которых скорее является исключением (для некоммерческих версий UNIX). Кроме того, в этом варианте работающую версию ОС надо остановить и заменить новой, что не всегда допустимо в некоторых применениях.

Поэтому поддержка динамической загрузки драйверов является практически обязательным требованием для современных универсальных ОС.

**7.8. Поддержка синхронных и асинхронных операций ввода-вывода**

Операция ввода-вывода может выполняться по отношению к программному модулю, запросившему операцию, в синхронном или асинхронном режимах [10]. Синхронный режим означает, что программный модуль приостанавливает свою работу до тех пор, пока операция ввода-вывода не будет завершена (рис. 7.4, верхняя диаграмма). При асинхронным режиме программный модуль продолжает выполняться в мультипрограммном режиме одновременно с операцией ввода-вывода (рис. 7.4, нижняя диаграмма).



**Рис. 7.4.**  Варианты выполнения операций ввода-вывода

Отличие заключается в том, что операция ввода-вывода может быть инициирована не только пользовательским процессом – в этом случае операция выполняется в рамках системного вызова, – но и кодом ядра, например, кодом подсистемы виртуальной памяти для считывания отсутствующей страницы.

Системы вызовы ввода-вывода чаще оформляются как синхронные процедуры в связи с тем, что такие операции длятся долго и пользовательскому процессу или потоку все равно придется ждать получения результатов потоков операции, для того чтобы продолжить свою работу.

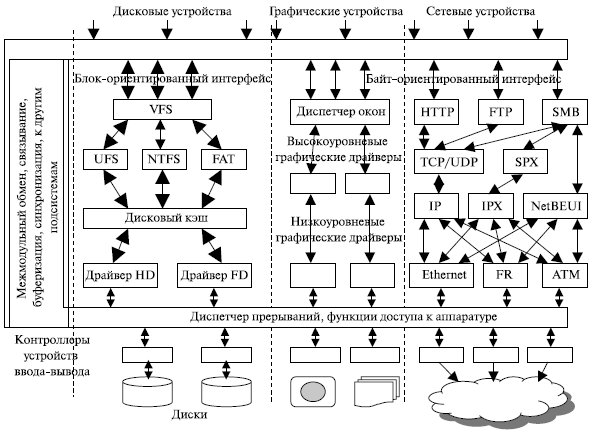
Внутренние вызовы операций ввода-вывода из модулей ядра обычно выполняются в виде асинхронных процедур, так как кодам ядра нужна свобода в выборе дальнейшего поведения после запроса ввода-вывода.

**7.9. Многослойная (иерархическая) модель подсистемы ввода-вывода**

При большом разнообразии устройств ввода-вывода, обладающих существенно различными характеристиками, иерархическая структура подсистемы ввода-вывода позволяет соблюсти баланс между двумя противоречивыми требованиями. С одной стороны, необходимо учесть все особенности каждого устройства, а с другой стороны – обеспечить единое логическое представление и унифицированный интерфейс для устройств всех типов. При этом нижние слои подсистемы ввода-вывода должны включать индивидуальные драйверы, написанные для конкретно физических устройств, а верхние слои должны обобщать процедуры управления этими устройствами, предоставляя общий интерфейс если не для всех устройств, то, по крайней мере, для группы устройств, обладающих некоторыми общими характеристиками, например, для принтеров определенного производителя или для всех матричных принтеров и т.п.

Многослойность структуры, безусловно, облегчает решение большинства перечисленных в предыдущем разделе задач подсистемы ввода-вывода. Обобщенная структура подсистемы ввода-вывода показана на рис. 7.5 [13]. Как видно из рисунка, программное обеспечение подсистемы ввода-вывода делится не только на горизонтальные слои, но и на вертикальные. В данном случае в качестве примера приведены три вертикальные подсистемы управления дисками, графическими устройствами и сетевыми адаптерами. Естественно, таких подсистем может быть больше. Например, сюда можно добавить подсистему управления текстовыми терминалами или подсистему управления специализированными устройствами, такими как аналого-цифровые и цифро-аналоговые преобразователи.

В каждой вертикальной подсистеме – несколько слоев модулей. Нижний слой образует аппаратные драйверы, управляющие аппаратурой внешних устройств, осуществляя обмен байтами и блоками байтов. Как правило, этот слой программного обеспечения не имеет дела с вопросами логической организации данных, например, с файлами или сложными графическими объектами. Функции вышележащих слоев в значительной степени зависят от типа вертикальной подсистемы.



**Рис. 7.5.**  Иерархическая структура подсистемы ввода-вывода

Наряду с модулями, отражающими специфику внешних устройств, в подсистеме ввода-вывода имеются модули универсального назначения. Эти модули организуют согласованную работу всех остальных компонентов подсистемы ввода-вывода, взаимодействие с пользовательскими процессами и другими подсистемами ОС. Эти организующие функции распределяются по всем уровням, образуя оболочку, называемую менеджером ввода-вывода.

Верхний слой менеджера составляют системные вызовы ввода-вывода, которые принимают от пользовательских процессов запросы на ввод-вывод и переадресуют их отвечающим за определенный класс устройств модулям и драйверам, а также возвращают процессам результаты операций ввода-вывода. Таким образом, этот слой поддерживает пользовательский интерфейс ввода-вывода, создавая для прикладных программистов максимум удобств по манипулированию внешними устройствами и расположенными на них данными.

Нижний слой менеджера реализует непосредственное взаимодействие с контроллерами внешних устройств, экранируя драйверы от особенностей аппаратной платформы компьютера – шин ввода-вывода, системы прерываний и т.п. Этот слой принимает от драйверов запросы на обмен данными с регистрами контроллеров в некоторой обобщенной форме с использованием независимых от шины ввода-вывода адресации и формата, а затем преобразует эти запросы в зависящий от аппаратной платформы формат.

Диспетчер прерываний может входить в состав менеджера ввода-вывода или представлять отдельный модуль ядра. В последнем случае менеджер ввода-вывода выполняет для диспетчера прерываний первичную обработку запросов прерываний, передавая диспетчеру обобщения сведения об источнике запроса.

Важной функцией менеджера ввода-вывода является создание некоторой среды для остальных компонентов системы, которая бы облегчала их взаимодействие друг с другом. Эта задача решается созданием стандартного внутреннего интерфейса взаимодействия модулей ввода-вывода между собой. Это облегчает включение новых драйверов и файловых систем в состав ОС. Кроме того, разработчики драйверов и других программных компонентов освобождаются от написания общих процедур, таких как буферизация данных и синхронизация нескольких модулей между собой при обмене данными. Все эти функции берет на себя менеджер ввода-вывода.

Еще одной функцией менеджера ввода-вывода является организация взаимодействия модулей ввода-вывода с модулями других подсистем ОС, таких как подсистема управления процессами, виртуальной памятью и другими.

**7.10. Драйверы**

Первоначально термин "драйвер" применялся в достаточно узком смысле – под драйвером понимается программный модуль, который:

* входит в состав ядра ОС, работая в привилегированном режиме;
* непосредственно управляет внешним устройством, взаимодействуя с его контроллером с помощью команд ввода-вывода компьютера;
* обрабатывает прерывания от *контроллера устройства*;
* предоставляет прикладному программисту удобный логический интерфейс работы с устройством, экранируя от него низкоуровневые детали управления устройством и организации его данных;
* взаимодействует с другими модулями ядра ОС с помощью строго оговоренного интерфейса, описывающего формат передаваемых данных, структуру буферов, способы включения драйвера в состав ОС, способы вызова драйвера, набор общих процедур подсистемы ввода-вывода, которыми драйвер может пользоваться и т.п.

Согласно этому определению драйвер вместе с контроллером устройства и прикладной программой воплощали идею многослойного подхода к организации программного обеспечения. Контроллер представлял низкий слой управления устройством, выполняющий операции в терминах блоков и агрегатов устройства (например, передвижение головки дисковода, побитную передачу байта по двухпроводному кабелю). Драйвер выполнял более сложные операции, преобразуя данные, адресуемые в терминах номеров цилиндров, головок и секторов диска, в линейную последовательность блоков. В результате прикладная программа работала с данными, преобразованными в достаточно понятную форму, – файлами, таблицами баз данных, текстовыми окнами на мониторе и т.п., не вдаваясь в детали представления этих данных в устройствах ввода-вывода.

В описанной схеме драйверы не делились на слои. Постепенно, по мере развития операционных систем и усложнения структуры подсистемы ввода-вывода, наряду с традиционными драйверами в ОС появились так называемые высокоуровневые драйверы, которые располагаются в общей модели подсистемы ввода-вывода над традиционными драйверами. Появление таких драйверов можно считать развитием идеи многоуровневой организации подсистемы ввода-вывода, когда ее функции декомпозируются между несколькими модулями в соседних слоях иерархии (таких примеров много, например семиуровневая модель сетевых протоколов).

Традиционные драйверы, которые стали называть аппаратными, низкоуровневыми или драйверами устройств, освобождаются от высокоуровневых функций и занимаются только низкоуровневыми операциями. Эти низкоуровневые операции составляют фундамент, на котором можно построить тот или иной набор операций в драйверах более высоких уровней.

При таком подходе повышается гибкость и расширяемость функции по управлению устройством. Например, если различным приложениям необходимо работать с различными логическими модулями одного и того же физического устройства, то для этого в системе достаточно установить несколько драйверов на одном уровне, работающих над одним аппаратным драйвером. Несколько драйверов, управляющих одним устройством, но на разных уровнях, можно рассматривать как один многоуровневый драйвер.

На практике используют от двух до пяти уровней драйверов, поскольку с увеличением числа уровней снижается скорость выполнения операций ввода-вывода.

Высокоуровневые драйверы оформляются по тем же правилам и придерживаются тех же внутренних интерфейсов, что и аппаратные драйверы. Как правило, высокоуровневые драйверы не вызываются по прерываниям, так как взаимодействуют с устройством через посредничество аппаратных драйверов.

В модулях подсистемы ввода-вывода, кроме драйверов, могут присутствовать и другие модули, например, дисковый кэш. Достаточно специфичные функции кэша делают нецелесообразным оформление его в виде драйвера, взаимодействующего с другими модулями ОС только с помощью услуг менеджера ввода-вывода. Другим примером модуля, который чаще всего не оформляется в виде драйвера, является *диспетчер окон* графического интерфейса. Иногда этот модуль вообще выносится из ядра ОС и реализуется в виде пользовательского интерфейса. Таким образом, был реализован *диспетчер окон* в Windows NT 3.5 и 3.51, но этот микроядерный подход заметно замедляет графические операции, поэтому в Windows 4.0 *диспетчер окон* и высокоуровневые графические драйверы, а также графическая библиотека GDI были перенесены в пространство ядра.

Аппаратные драйверы после запуска операции ввода-вывода должны своевременно реагировать на завершение контроллером заданного действия путем взаимодействия с системой прерывания. Драйверы более высоких уровней вызываются не по прерываниям, а по инициативе аппаратных драйверов или драйверов вышележащего уровня. Не все процедуры аппаратного драйвера нужно вызывать по прерываниям, поэтому драйвер обычно имеет определенную структуру, в которой выделяется секция обработки прерываний (*Interrupt Service* Routine, *ISR*), которая и вызывается от соответствующего устройства диспетчером прерываний.

В унификацию драйверов большой вклад внесла ОС UNIX, в которой все драйверы были разделены на два класса: блок-ориентированные (Block-oriented) и байт-ориентированные (Character-oriented) драйверы. Это более общее деление, чем деление на вертикальные подсистемы. Например, драйверы графических устройств и сетевых устройств относятся к классу байт-ориентированных.

Блок-ориентированные драйверы управляют устройствами прямого доступа, которые хранят информацию в блоках фиксированного размера, каждый из которых имеет свой адрес. Адресуемость блоков приводит к тому, что для дисков, являющихся устройствами прямого доступа, появляется возможность кэширования данных в оперативной памяти. Это обстоятельство значительно влияет на общую организацию ввода-вывода для блок-ориентированных драйверов.

Устройства, с которыми работают байт-ориентированные драйверы, не адресуют данные и не позволяют производить операции поиска данных, они генерируют или потребляют последовательность байта (терминалы, принтеры, сетевые адаптеры и т.п.).

Однако не все устройства, управляемые подсистемой ввода-вывода, можно разделить на блок и байт-ориентированные. Для таких устройств (например, таймер) нужен специфический драйвер.

В свое время ОС UNIX сделала очень важный шаг по унификации операций и структуризации программного обеспечения ввода-вывода. В ОС UNIX все устройства рассматриваются как виртуальные (специальные) файлы, что дает возможность использовать общий набор базовых операций ввода-вывода для любых устройств независимо от их специфики. Подобная идея реализована позже в MS-DOS, где последовательные устройства – монитор, принтер и клавиатура – считаются файлами со специальными именами: con, *prn*, con.

**7.11. Файловые системы. Основные понятия**

Цели и задачи файловой системы

Любое компьютерное приложение получает, хранит и выводит данные. Во время работы процесс может хранить ограниченное количество данных в собственном адресном пространстве, поскольку его емкость ограничена рамками виртуального адресного пространства. Для некоторых приложений, например, систем резервирования авиабилетов, систем *банковского учета* и др., однако, только виртуального адресного пространства будет недостаточно.

Кроме того, после завершения работы процесса информация, хранящаяся в его адресном пространстве, теряется. В это же время для ряда приложений (например, баз данных) ее надо хранить длительное время, а иногда даже вечно. Исчезновение данных после завершения процесса для таких приложений неприемлемо. Информация должна сохраняться и при аварийном завершении процесса в случае сбоя компьютера.

Третья проблема состоит в том, что часто необходимо разным процессам одновременно получать доступ к одним и тем же данным (или части данных). Для решения этой проблемы необходимо отделить информацию от процесса.

Таким образом, необходимо хранить данные на устройствах компьютеров (диски, ленты и др.) с соблюдением следующих требований [13].

1. Устройства должны позволять хранить очень большие объемы данных. К таким устройствам относятся жесткие магнитные диски, магнитные ленты, оптические и магнитооптические диски.
2. Информация должна длительно и надежно сохраняться после прекращения работы процесса, использующего эту информацию. Долговременность хранения обеспечивается применением запоминающих устройств, не зависящих от электропитания, а высокая надежность определяется соответствующей организацией операционной системы.
3. Несколько процессов должны иметь возможность получения одновременного доступа к информации, т.е. должно быть обеспечено совместное использование данных.

Решение этих проблем состоит в хранении информации, организованной в файлы. Файл – это именованная совокупность данных, хранящаяся на каком-либо носителе информации.

При рассмотрении отдельных файлов и их совокупностей используются следующие понятия.

1. Поле (Field) – основной элемент данных. Поле содержит единственное значение, такое как имя служащего, дату, значение некоторого показателя и т.п. Поле характеризуется длиной и типом данных и может быть фиксированной или переменной длины, т.е. состоять из нескольких подполей: имя поля, значение, длина поля.
2. Запись (Record) – набор связанных между собой полей, которые могут быть обработаны как единое целое некоторой прикладной программой (например, запись о сотруднике, содержащая такие поля, как имя, должность, оклад и т.д.). В зависимости от структуры записи могут быть фиксированной или переменной длины.
3. Файл (File) – совокупность однородных записей. Файл рассматривается как единое целое приложениями и пользователем. Обращение к файлу осуществляется по его имени. Пользователь (программист) должен иметь удобные средства работы с файлами, включая каталоги-справочники, объединяющие файлы в группы, средства поиска файла по различным признакам, набор команд для создания, модификации и удаления файлов. Файл может быть создан одним пользователем, а затем использоваться другим, при этом создатель файла или администратор могут определить права доступа к нему других пользователей. В некоторых системах управления доступом осуществляется на уровне записи, а иногда и на уровне поля.
4. База данных (database) – набор связанных между собой данных, представленных совокупностью файлов одного или несколько типов. Обычно существует отдельная система управления базой данных (СУБД), независимая от операционной системы, но, тем не менее, она почти всегда использует некоторые программы управления файлами.

Обычно единственным способом работы с файлами является применение *системы управления файлами* или иначе – файловой системы (ФС).

Файловая система – это часть операционной системы, включающая:

* совокупность всех файлов на носителе информации (магнитном или оптическом диске, магнитной ленте и др.);
* наборы структур данных, используемых для управления файлами, каталоги и дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске и др.);
* комплекс системных программных средств, реализующих различные операции над файлами (создание, уничтожение, чтение, запись и др.).

Задачи, решаемые файловой системой, во многом определяются способом организации вычислительного процесса (наиболее простые – в однопрограммных и однопользовательских ОС, наиболее сложные – в сетевых ОС.).

В мультипрограммных, многопользовательских ОС задачами файловой системы являются [10]:

* соответствие требованиям управления данными и требованиям со стороны пользователей, включающим возможность хранения данных и выполнения операций с ними;
* гарантирование корректности данных, содержащихся в файле;
* оптимизация производительности, как с точки зрения системы (пропускная способность), так и с точки зрения пользователя (время отклика);
* поддержка ввода-вывода для различных типов устройств хранения информации;
* минимизация или полное исключение возможных потерь или повреждений данных;
* защита файлов от несанкционированного доступа;
* обеспечение поддержки совместного использования файлов несколькими пользователями (в том числе средства блокировки файла и его частей, исключение тупиков, согласование копий и т.п.);
* обеспечение стандартизированного набора подпрограмм интерфейса ввода-вывода.

Минимальным набором требований к файлам системы со стороны пользователя диалоговой *системы общего назначения* можно считать следующую совокупность возможностей, предоставляемую пользователю:

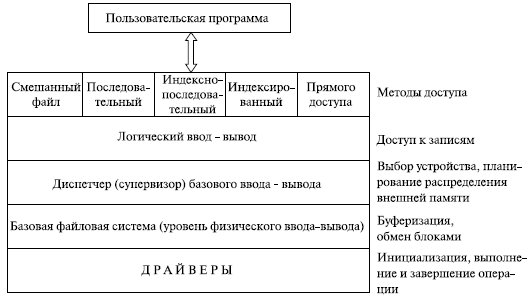
1. создание, удаление, чтение и изменение файлов;
2. контролируемый доступ к файлам других пользователей;
3. управление доступом к своим файлам;
4. реструктурирование файлов в соответствии с решаемой задачей;
5. перемещение данных между файлами;
6. резервирование и восстановление файлов в случае повреждения;
7. доступ к файлам по символьным именам.

**7.12. Архитектура файловой системы**

Файловая система позволяет программам обходиться набором достаточно простых операций для выполнения действий над некоторым абстрактным объектом, представляющим файл. При этом программистам не нужно иметь дело с деталями действительного расположения данных на диске, буферизацией данных и другими низкоуровневыми проблемами передачи данных с запоминающего устройства. Все эти функции файловая система берет на себя. Файловая система распределяет дисковую память, поддерживает именование файлов, отображает имена файлов в соответствующие адреса во внешней памяти, обеспечивает доступ к данным, поддерживает разделение, защиту и восстановление данных.

Таким образом, файловая система играет роль промежуточного слоя, экранизирующего все сложности физической организации долговременного хранилища данных и создающего для программ более простую логическую модель этого хранилища, а затем предоставляет им набор удобных в использовании команд для манипулирования файлами.

Классическая схема организации программного обеспечения файловой системы представлена на рис. 7.6.



**Рис. 7.6.**  Организация программного обеспечения файловой системы

На нижнем уровне драйверы устройств непосредственно связаны с периферийными устройствами или их котроллерами либо каналами. Драйвер устройства отвечает за начальные операции ввода-вывода устройства и за обработку завершения запроса ввода-вывода. При файловых операциях контролируемыми устройствами являются дисководы и *стримеры* (накопители на МЛ). Драйверы устройств рассматриваются как часть операционной системы.

Следующий уровень называется базовой файловой системой, или уровнем физического ввода-вывода. Это первичный интерфейс с окружением (периферией) компьютерной системы. Он оперирует блоками данных, которыми обменивается с дисками, магнитной лентой и другими устройствами. Поэтому он связан с размещением и буферизацией блоков в оперативной памяти. На этом уровне не выполняется работа с содержимым блоков данных или структурой файлов. Базовая файловая система обычно рассматривается как часть операционной системы (в MS-DOS эти функции выполняет BIOS, не относящийся к ОС).

Диспетчер базового ввода-вывода отвечает за начало и завершение файлового ввода-вывода. На этом уровне поддерживаются управляющие структуры, связанные с устройством ввода-вывода, планированием и статусом файлов. Диспетчер осуществляет выбор устройства, на котором будет выполняться операция файлового ввода-вывода, планирование обращения к устройству (дискам, лентам), назначение буферов ввода-вывода и распределение внешней памяти. Диспетчер базового ввода-вывода является частью ОС.

Логический ввод-вывод предоставляет приложениям и пользователям доступ к записям. Он обеспечивает возможности общего назначения по вводу-выводу записей и поддерживает информацию о файлах. Наиболее близкий к пользователю уровень ФС часто называется методом доступа. Он обеспечивает стандартный интерфейс между приложениями и файловыми системами и устройствами, содержащими данные. Различные методы доступа отражают различные структуры файлов и различные пути доступа и обработки данных.

**7.13. Организация файлов и доступ к ним**

Типы, именование и атрибуты файлов

Файловые системы поддерживают несколько функционально различных типов файлов, в число которых входят обычные файлы, содержащие информацию произвольного характера (текст, графика, звук и др.), файлы-каталоги, специальные файлы, именованные конвейеры, отображаемые в память файлы и др.

Обычные файлы, или просто файлы, или регулярные файлы, содержат информацию, которую в них заносит пользователь или которая образуется в результате работы системных и пользовательских программ. Большинство ОС не контролируют содержимое и структуру *регулярных файлов*, которые в основном являются ASCII-файлами либо двоичными файлами. ASCII-фалы состоят из текстовых строк. Они могут отображаться на экране и выводиться на печать без какого-либо преобразования, и могут редактироваться практически любым текстовым редактором. Двоичные файлы имеют определенную внутреннюю структуру, которая известна программе, использующей данный файл. При выводе двоичного файла на принтер получается случайный набор символов.

Каталоги – это системные файлы, обеспечивающие поддержку структуры файловой системы. Они содержат системную справочную информацию о наборе файлов, сгруппированных пользователем по какому-либо неформальному признаку (договоры, рефераты, курсовые проекты и т.п.). Во многих ОС в каталог могут входить другие файлы, в том числе другие каталоги, за счет чего образуется древовидная структура, удобная для поиска требуемого файла. Каталоги устанавливают соответствие между именами файлов и их характеристиками, используемыми файловой системой для управления файлами. В число таких характеристик входят тип файла, права доступа к файлу, его распоряжение на диске, размер, дата и время создания и др.

Специальные файлы – это фиктивные файлы, ассоциированные с устройствами ввода-вывода, которые используются для унификации механизма доступа к последовательным устройствам ввода-вывода, таким как терминалы, принтеры и др. (например, MS-DOS рассматривает монитор и клавиатуру как файлы со стандартным именем con – консоль, а принтер – как файл *prn*). *Блочные специальные файлы* используются для моделирования дисков.

Именованные конвейеры (каналы) представляют собой циклические буферы, позволяющие выходной файл одной программы соединить со входным файлом другой программы.

Наконец, отображаемые файлы – это обычные файлы, отображенные на адресное пространство процесса по указанному виртуальному адресу.

Файлы относятся к абстрактному механизму. Они предоставляют способ сохранять информацию на запоминающем устройстве и считывать ее позднее снова. При этом от пользователя должны скрываться такие детали, как способ и место хранения информации, а также детали работы устройства.

Наиболее важной характеристикой любого механизма абстракции является именование управляемых объектов. Правила именования файлов меняются от одной ОС к другой, но, как правило, все современные операционные системы поддерживают использование в качестве имен файлов 8-символьные текстовые строки. Часто в именах разрешается использование цифр и специальных символов. В некоторых файловых системах различаются прописные и строчные символы, тогда как в других, например, MS-DOS, – нет.

Во многих операционных системах имя файла состоит из двух частей, разделенных точкой. Часть имени после точки называется расширением файла и обычно означает его тип. Так, в MS-DOS имя файла может содержать от 1 до 8 символов, а расширение от 0 (отсутствует) до 3.

В некоторых ОС, например, Windows, расширение указывает на программу, создавшую файл. Другие ОС, например, UNIX, не принуждают пользователя строго придерживаться расширений. Некоторые типичные расширения файлов приведены ниже.

|  |  |
| --- | --- |
| **Расширение** | **Значение** |
| file.*bak* | Резервная копия файла |
| file.cpp | Исходный текст программы на С++ |
| file.gif | Изображение формата GIF |
| file.hlp | Файл справки |
| file.html | Документ в формате HTML |
| file.jpg | Неподвижное изображение стандарта JPEG |
| file.mp3 | Музыка в формате MPEG-1 уровень 3 |
| file.mpg | Фильм в формате MPEG |
| file.obj | Объектный файл |

В иерархически организованных файловых системах обычно используются три типа имен файлов: простые, составные и относительные.

Простое (короткое) символьное имя идентифицирует файл в пределах одного каталога. Несколько файлов могут иметь одно и то же простое имя, если они принадлежат разным каталогам.

Составное (полное) символьное имя представляет собой цепочку, содержащую имя диска и имена всех каталогов, через которые проходит путь от корневого каталога до данного файла.

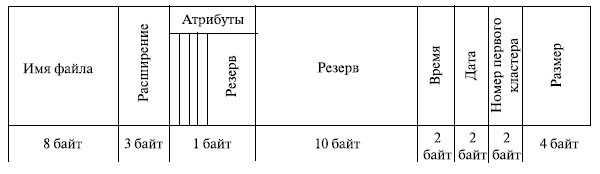
*Относительное имя файла* определяется через текущий каталог, т.е. каталог, в котором в данный момент времени работает пользователь. Таким образом, относительных имен у файла может быть достаточно много, и все они являются частью полного имени.

Понятие файла включает не только хранимые им данные и имя, но и информацию, описывающую свойства файла. Эта информация составляет атрибуты файла. Список атрибутов может быть различным в различных ОС. Пример возможных атрибутов приведен ниже.

|  |  |
| --- | --- |
| **Атрибут** | **Значение** |
| Тип файла | Обычный, каталог, специальный и т. д. |
| *Владелец файла* | Текущий владелец |
| Создатель файла | Идентификатор пользователя, создавшего файл |
| Пароль | Пароль для получения доступа к файлу |
| Время | Создания, последнего доступа, последнего изменения |
| Текущий размер файла | Количество байт в записи |
| Максимальный размер | Количество байт, до которого можно увеличивать размер файла |
| Флаг "только чтение" | 0 – чтение / запись, 1 – только чтение |
| Флаг "скрытый" | 0 – нормальный, 1 – не показывать в перечне файлов каталога |
| Флаг "системный" | 0 – нормальный, 1 – системный |
| Флаг "архивный" | 0 – заархивирован, 1 – требуется архивация |
| Флаг ASCII / двоичный | 0 – ASCII, 1 – двоичный |
| Флаг произвольного доступа | 0 – только последовательный доступ, 1 – произвольный доступ |
| Флаг "временный" | 0 – нормальный, 1 – удаление после окончания работы процесса |
| Позиция ключа | Смещение до ключа в записи |
| Длина ключа | Количество байт в поле ключа |

Пользователь может получить доступ к атрибутам, используя средства, предоставляемые для этой цели файловой системой. Обычно разрешается читать значение любых атрибутов, а изменять – только некоторые.

Значения атрибутов файлов могут содержаться в каталогах, как это сделано, например, в MS-DOS (рис. 7.7). Другим вариантом является размещение атрибутов в специальных таблицах, в этом случае в каталогах содержатся ссылки на эти таблицы.



**Рис. 7.7.**  Атрибуты файлов MS DOS

Логическая организация файла

В общем случае данные, содержащиеся в файле, имеют некоторую логическую структуру. Эта структура (организация) файла является базой при разработке программы, предназначенной для обработки этих данных. Поддержание структуры данных может быть целиком возложено на приложение либо в той или иной степени эту работу может взять на себя файловая система.

В первом случае, когда все действия, связанные со структуризацией и интерпретацией содержимого файла, целиком относятся к ведению приложения, файл представляется файловой системе неструктурированной последовательностью данных. Приложение формирует запросы к файловой системе на ввод-вывод, используя общие для всех приложений системные средства, например, указывая смещение от начала файла и количество байт, которые необходимо считать или записать. Поступивший к приложению поток байт интерпретируется в соответствии с заложенной в программе логикой. Следует подчеркнуть, что интерпретация данных никак не связана с действительным способом их хранения в файловой системе.

Модель файла, в соответствии с которой содержимое файла представляется неструктурированной последовательностью байт, стала популярной вместе с ОС UNIX, и теперь широко используется в современных ОС. Неструктурированная модель файла позволяет легко организовать разделение файла между несколькими приложениями, поскольку разные приложения могут по-своему структурировать и интерпретировать данные, содержащиеся в файле.

Другая модель файла – структурированный файл. В этом случае поддержание структуры файла поручается файловой системе. Файловая система видит файл как упорядоченную последовательность логических записей. ФС предоставляет приложению доступ к записи, а вся дальнейшая обработка данных, содержащихся в этой записи, выполняется приложением!

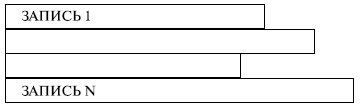
Известно пять фундаментальных способов организации файлов [10]:

* смешанный файл,
* *последовательный файл*,
* индексно-*последовательный файл*,
* индексируемый файл,
* файл прямого доступа.

При выборе способа организации файла нужно учитывать несколько критериев:

* быстрота доступа,
* легкость обновления,
* экономность хранения,
* простота обслуживания,
* надежность.

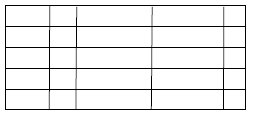
Смешанный файл. Это наименее сложная форма организации файла. Данные накапливаются в порядке поступления. Запись состоит из одного пакета данных. Записи могут иметь различные или одинаковые поля, расположенные в различном порядке (рис. 7.8). Каждое поле описывает само себя, включая как имя, так и значение. Длина каждого поля должна быть указана явно либо посредством применения разделителя.



**Рис. 7.8.**  Смешанный файл

Поскольку смешанный файл не имеет никакой структуры, доступ к записи осуществляется полным перебором всех записей файла. Смешанные файлы применяются в том случае, когда данные накапливаются и сохраняются перед обработкой, или если данные неудобны для организации. Файлы этого типа рационально используют дисковое пространство, хорошо подходят для полного набора. Обновление записей достаточно сложно, так же как и вставка записи.

*Последовательный файл*. Для записей используется фиксированный формат. Все записи имеют одинаковую длину (но иногда и не одинаковую) и состоят из одинакового количества полей фиксированной длины, организованных в определенном порядке (рис. 7.9). Поскольку длина и позиция каждого поля известны, сохранению подлежат только значения полей. Атрибутами файловой структуры является имя и длина каждого поля.



**Рис. 7.9.**  Последовательный файл

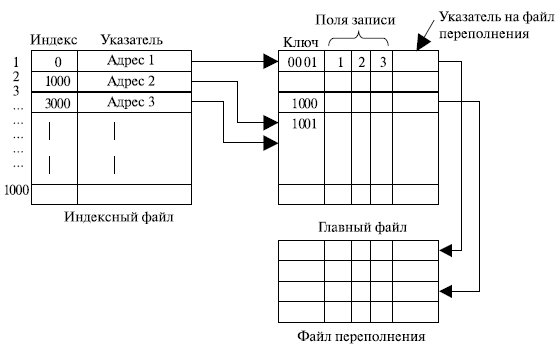
Одно определенное поле (или несколько полей) называется ключевым. Оно однозначно идентифицирует запись, так как это поле различно для каждой записи. Более того, записи сохраняются в "ключевой" последовательности: в алфавитном порядке для текстового ключа и в числовом – для числового. Последовательные файлы часто используются пакетными приложениями и обычно являются оптимальным вариантом, если эти приложения выполняют обработку всех записей. Удобно и то, что такой файл можно хранить как на ленте, так и на магнитном диске.

Для диалоговых приложений *последовательный файл* малоэффективен, поскольку для нахождения нужной записи требуется последовательный перебор записи файла. Правда, если в оперативную память загрузить весь файл, возможен более эффективный метод поиска. Дополнения к файлу или изменения в записях создают проблемы.

Обычно *последовательный файл* сохраняется с последовательной организацией записей внутри блока, т.е. физическая организация файла в точности соответствует логической. Новые записи размещаются в отдельном смешанном файле, называемом журнальным файлом, или файлом транзакции. Периодически в пакетном режиме выполняется слияние основного и журнального файлов в новый файл с корректной последовательностью ключей.

Альтернативной организацией может быть физическая организация в виде списка с использованием указателей. В каждом физическом блоке сохраняется одна или несколько записей, и каждый блок содержит указатель на следующий блок. Для вставки новых записей достаточно изменить указатели, и нет необходимости в том, чтобы новые записи занимали определенную физическую позицию. Это удобство достигается за счет определенных накладных расходов и дополнительной работы. Если в последовательном файле записи имеют одну и ту же длину, то можно вычислить адрес требуемой записи по ее номеру, номеру текущей записи и длине записи. Если записи имеют переменную длину, такой подход невозможен.

Индексно-*последовательный файл*. Одним из методов преодоления недостатков *последовательного файла* является индексно-последовательная организация файла. В этом случае файл состоит из трех частей (файлов): главный файл, содержащий записи с последовательно идущими ключами, индексный файл, содержащий индексное поле, и указатель в главный с ключами, файл переполнения (рис. 7.10).



**Рис. 7.10.**  Индексно-последовательный файл

Для поиска нужной записи по ее ключу сначала выполняется поиск в индексном файле. После того как в нем найдено наибольшее значение ключа, которое не превышает искомое, продолжается поиск в главном файле. Например, пусть *последовательный файл* (главный) содержит 1 млн записей. Для поиска определенного ключевого значения необходимо в среднем 0,5 млн операций доступа к записям. Если создать индексный файл, содержащий 1000 элементов, то потребуется в среднем 500 операций доступа к индексному файлу, после чего еще нужно в среднем 500 операций доступа к главному файлу. В результате средняя длина поиска уменьшилась с 0,5 млн до 1000. Еще лучшего результата можно достичь, используя многоуровневую индексацию. При этом нижний уровень индексного файла рассматривается как *последовательный файл*, для которого создается индексный файл верхнего уровня.

Дополнения к файлу обрабатываются следующим образом. В каждой записи главного файла содержится дополнительное поле, невидимое для приложения и являющееся указателем на файл переполнения. Если в файле производится вставка новой записи, она добавляется в файл переполнения. Запись в главном файле, непосредственно предшествующая новой записи в логической последовательности, обновляется и указывает на новую запись в файле переполнения. Время от времени выполняется слияние индексно-*последовательного файла* с файлом переполнения.

Индексированный файл. Индексно-*последовательный файл* сохраняет одно ограничение *последовательного файла*: эффективная работа с файлом ограничена работой с ключевым полем. Если необходимо производить поиск записи по какой-либо иной характеристике, отличной от ключевого поля, то оказываются непригодными обе организации *последовательного файла*, в то время как в некоторых приложениях эта гибкость крайне желательна.

Для достижения гибкости необходимо применение большого количества индексов, по одному для каждого типа поля, которое может быть объектом поиска. В обобщенном индексированном файле доступ к записям осуществляется только по их индексам. В результате в размещении записей нет никаких ограничений до тех пор, пока указатель по крайней мере в одном индексе ссылается на эту запись. Кроме того, в таком файле легко реализуются записи переменной длины.

Используется два типа индексов. Полный индекс содержит по одному элементу для каждого типа записей главного файла. Сам по себе индекс организовывается в виде *последовательного файла* для облегчения поиска. Частный индекс содержит элементы для записей, в которых имеется интересующее пользователя поле. При добавлении новой записи в главный файл необходимо обновлять все индексные файлы.

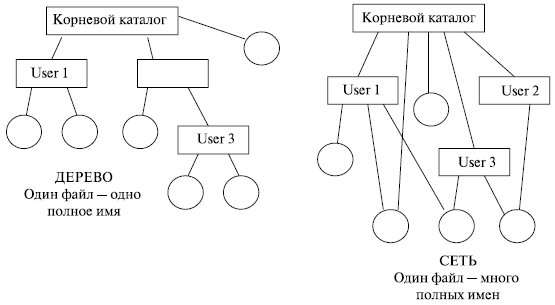
Индексированные файлы применяются теми приложениями, в которых время доступа к информации является критической характеристикой и редко требуется обработка всех записей в файле.

Файл прямого доступа. Такой файл использует возможность прямого доступа к блоку с известным адресом при хранении файлов на диске. В каждой записи в этом случае также имеется ключевое поле.

**7.14. Каталоговые системы**

Связующим звеном между *системой управления файлами* и набором файлов служит файловый каталог. Простейшая форма системы каталогов состоит в том, что имеется один каталог, в котором содержатся все файлы. Каталог содержит информацию о файлах, включая атрибуты, местоположение, принадлежность. Пользователи обращаются к файлам по символьным именам. Однако способности человеческой памяти ограничивают количество имен объектов, к которым пользователь может обращаться по именам. Иерархическая организация пространства имен позволяет значительно расширить эти границы. Именно поэтому каталоговые системы имеют иерархическую структуру. Граф, описывающий иерархию каталогов, может быть деревом или сетью. Каталоги образуют дерево, если файлу разрешено входить только в один каталог (рис. 7.11), и сеть, если файл может входить в несколько каталогов.

Например, в Ms-Dos и Windows каталоги образуют древовидную структуру, а в UNIX – сетевую. В общем случае вычислительная система может иметь несколько дисковых устройств, даже в ПК всегда имеется несколько дисков: гибкий, винчестер, CD-ROM (DVD). Как организовать хранение файлов в этом случае?



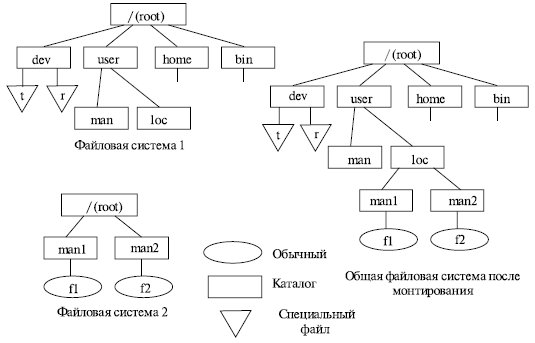
**Рис. 7.11.**  Каталоговые системы

Первое решение состоит в том, что на каждом из устройств размещается автономная файловая система, т.е. файлы, находящиеся на этом устройстве, описываются деревом каталогов, никак не связанным с деревьями каталогов на других устройствах. В таком случае для однозначной идентификации файла пользователь вместе с составным символьным именем файла должен указывать идентификатор логического устройства. Примером такого автономного существования может служить MS-DOS, Windows 95/98/Me/XP.

Другим решением является такая организация хранения файлов, при которой пользователю предоставляется возможность объединить файловые системы, находящиеся на разных устройствах, в единую файловую систему, описываемую единым деревом каталогов. Такая операция называется монтированием.

В ОС UNIX монтирование осуществляется следующим образом. Среди всех имеющихся логических дисковых устройств выделяется одно, называемое системным. Пусть имеются две файловые системы, расположенные на разных логических дисках, причем один из дисков является системным (рис. 7.12).

Файловая система, расположенная на системном диске, называется корневой. Для связи иерархий файлов в *корневой файловой системе* выбирается некоторый существующий каталог, в данном примере – каталог *loc*. После выполнения монтирования выбранный каталог *loc* становится корневым каталогом второй файловой системы. Через этот каталог монтируемая файловая система подсоединяется как поддерево к общему дереву.



**Рис. 7.12.**  Монтирование

**7.15. Физическая организация файловой системы**

Информационная структура магнитных дисков

Представление пользователей о файловой системе как об иерархически организованном множестве информационных блоков имеет мало общего с порядком хранения файлов на диске. Файл, имеющий образ цельного, непрерывающегося набора байт, на самом деле разбросан своими частями по всему диску, причем это разбиение никак не связано с логической структурой файла. Точно так же логически объединенные файлы из одного каталога совсем не обязательно соседствуют на диске. Принципы размещения файлов, каталогов и системной информации на реальном устройстве описываются физической организацией файловой системы. При этом ясно, что разные файловые системы имеют разную физическую организацию.

Основным устройством для хранения файлов являются жесткие и гибкие магнитные диски. Жесткие диски состоят из одной или нескольких стеклянных или металлических пластин, каждая из которых покрыта с одной стороны или двух сторон магнитным материалом.

На каждой стороне каждой пластины размечены тонкие концентрические кольца – дорожки (treks), на которых хранятся данные. Нумерация дорожек начинается с 0 от *внешнего края* к центру диска. Когда диск вращается, магнитные головки, имеющиеся над (под) каждой поверхностью диска, считывают или записывают двоичные данные на магнитные дорожки. Головки могут позиционировать над каждой дорожкой, если на одну поверхность диска в устройстве имеется одна головка. Некоторые диски имеют по отдельной головке на каждую дорожку, тогда позиционирование головок не требуется, что повышает быстродействие диска.

Совокупность дорожек одного радиуса на всех поверхностях пластин пакета называется цилиндром (*cylinder*). Каждая дорожка разбивается на фрагменты, называемые секторами (sectors) или блоками (blocks), так что все дорожки имеют равное число секторов, в которые можно максимально записать одно и то же число байт. Сектор имеет фиксированный для данной системы размер, выражающийся степенью двойки (чаще всего 512 байт).

Сектор – наименьшая адресуемая единица обмена данными диска с оперативной памятью. Для того чтобы контроллер мог найти на диске нужный сектор, ему необходимо задать все составляющие адреса сектора: номер цилиндра, номер поверхности и номер сектора. Типичный запрос включает чтение (запись) нескольких секторов, содержащих наряду с требуемыми избыточные данные.

Операционная система при работе с диском использует, как правило, единицу дискового пространства, называемую кластером (cluster) и содержащую несколько секторов в числе, кратном степени двойки. Это связано с тем, что применение более мелкой единицы дискового пространства – сектора – усложняет учет свободного и занятого пространства диска при современных больших емкостях дисков, исчисляющихся десятками и сотнями Гбайт.

Дорожки и секторы создаются в результате выполнения процедуры физического (низкоуровнего) форматирования диска, предшествующей использованию диска. Для определения *границ блоков* на диск записывается идентификационная информация. Низкоуровневый формат диска не зависит от типа ОС, которая с этим диском будет работать.

Разметку диска под конкретный тип файловой системы выполняют процедуры высокоуровнего, или логического, форматирования. При высокоуровневом форматировании определяется *размер кластера*, записываются информация, необходимая для работы файловой системы, и загрузчик ОС – небольшая программа, которая начинает процесс инициализации операционной системы после включения питания.

Прежде чем форматировать диск под определенную файловую систему, он может быть разбит на разделы. Раздел – это непрерывная часть физического диска, которую операционная система представляет пользователю как логическое устройство (логический диск или логический раздел). На каждом разделе может создаваться только одна файловая система.

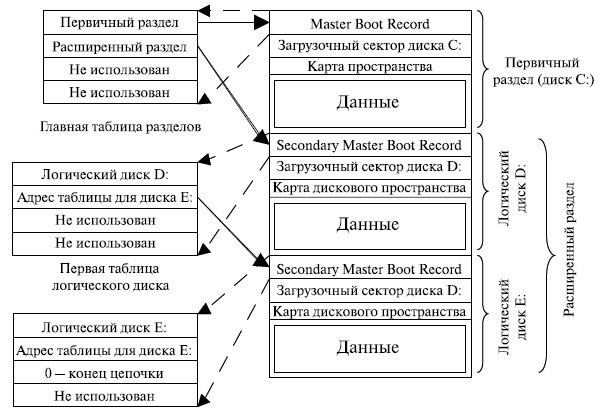
В IBM-совместных ПК сектор 1 диска называется главной загрузочной записью (*MBR* – *Master Boot Record*) и используется для загрузки компьютера. В конце *MBR*содержится таблица разделов. В ней хранятся начальные и конечные адреса (номера блоков) каждого раздела. Один из разделов помечен в таблице как активный. При загрузке компьютера BIOS считывает и исполняет *MBR*-запись, после чего загрузчик в *MBR*-записи определяет активный раздел диска, считывает его первый блок (загрузчик) и исполняет его. Программа, находящаяся в загрузочном блоке, загружает операционную систему, содержащуюся в этом разделе. Для единообразия каждый дисковый раздел начинается с загрузочного блока, даже если в нем не содержится операционной системы. К тому же в этом разделе может быть в дальнейшем установлена операционная система, поэтому зарезервированный *загрузочный блок* оказывается полезным.

Таблица разделов располагается в *MBR* по смещению 0х1BE и содержит четыре элемента. Структура записи элемента таблицы разделов приведена ниже.

|  |  |
| --- | --- |
| **Наименование 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 относительно номера начального сектора | 4 |
| Младшее и старшее двухбайтовые слова размера раздела в секторах | 4 |

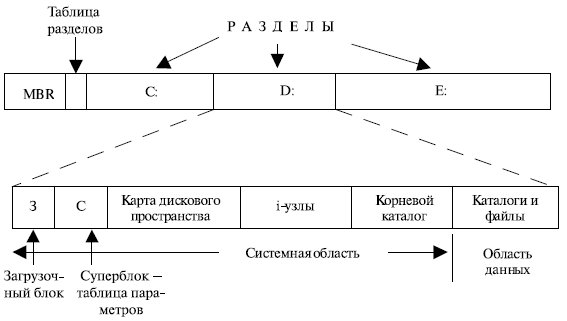
Каждый элемент таблицы описывает один раздел, причем двумя способами: через координаты C-H-S начального и конечного секторов, а также через номер первого сектора в спецификации *LBA* (Logical *Block Addressing*) и общее число секторов в разделе [10]. Последние два байта *MBR* имеют значение 55AAh, т.е. чередующиеся значения 0 и 1. Эта сигнатура выбрана для того, чтобы проверить работоспособность всех линий передачи данных. Значение 55AAh, присвоенное последним двум байтам, имеется во всех загрузочных секторах.

Разделы дисков могут быть двух типов: первичные (primary) и расширенные (extended). Максимальное число первичных разделов равно четырем. Из них только один может быть активным. Именно загрузчику, расположенному в активном разделе, передается управление при включении компьютера с помощью внесистемного загрузчика. Согласно принятым спецификациям на одном жестком диске может быть только один *расширенный раздел*, который может быть разделен на логические диски (рис. 7.13). *Расширенный раздел* содержит вторичную запись *MBR*, в состав которой вместо таблицы разделов входит аналогичная ей таблица логических дисков (logical Disks Table, *LDT*). Эта таблица описывает размещение и характеристики раздела, содержащего единственный логический диск, а также может специфицировать следующую запись SMBR (Secondary *MBR*).



**Рис. 7.13.**  Разделы диска

Во всем остальном строение раздела диска меняется от системы к системе. Часто файловая система содержит некоторые элементы, показанные на рис. 7.14. Один из таких элементов называется *суперблоком* и содержит ключевые параметры файловой системы, и считывается в память при загрузке компьютера. Следом располагается информация о свободных блоках файловой системы. За этими данными может следовать информация об i-узлах, содержащих информацию о файлах. Следом может размещаться каталог и затем – остальные файлы и каталоги.



**Рис. 7.14.**  Структура раздела

На разных логических устройствах одного и того же физического диска могут располагаться файловые системы разного типа. Все разделы одного диска имеют одинаковый размер блока, определенный для данного диска в результате низкого уровневого форматирования. Однако в результате высокоуровневого форматирования в разных разделах одного и того же диска могут быть установлены различные файловые системы с различными разделами кластеров.

**7.16. Физическая организация и адресация файла**

Физическая организация выделяет способ размещения файлов на диске и учет соответствия блоков диска файлам. Основными критериями эффективности физической организации файлов являются:

* скорость доступа к данным;
* объем адресной информации файла;
* степень фрагментированности дискового пространства;
* максимально возможный размер файла.

Наиболее часто используются следующие схемы размещения файлов:

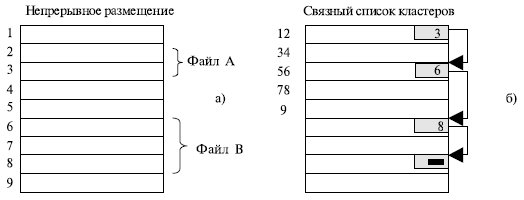
* непрерывное размещение (непрерывные файлы);
* сводный список блоков (кластеров) файла;
* сводный список индексов блоков (кластеров) файла;
* перечень номеров блоков (кластеров) файла в структурах, называемых i-узлами (index-node – индекс-узел).

Простейший вариант физической организации – непрерывное размещение в наборе соседних кластеров (рис. 7.15a). Достоинство этой схемы – высокая скорость доступа и минимальный объем адресной информации, поскольку достаточно хранить номер первого кластера и объем файла. Размер файла при такой организации не ограничивается.

Однако у этой схемы имеется серьезный недостаток – фрагментация, возрастающая по мере удаления и записи файлов. Кроме того, возникает вопрос, какого размера область нужно выделить файлу, если при каждой модификации он может увеличить свой размер.

И все-таки есть ситуации, в которых непрерывные файлы могут эффективно использоваться и действительно широко применяются – на компакт-дисках. Здесь все размеры файлов заранее известны и не могут меняться.

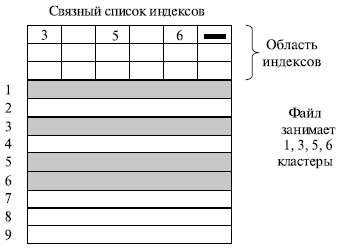
Второй метод размещения файлов состоит в представлении файла в виде связного списка кластеров дисковой памяти (рис. 7.15б). Первое слово каждого кластера используется как указатель на следующий кластер. В этом случае адресная информация минимальна, поскольку расположение файла задается номером его первого кластера.



**Рис. 7.15.**  Варианты физической организации файлов

Кроме того, отсутствует фрагментация на уровне кластеров, а файл легко может изменять размер наращиванием или удалением цепочки кластеров. Однако доступ к такому файлу может оказаться медленным, так как для получения доступа к кластеру n операционная система должна прочитать первые n-1 кластеры. Кроме того, *размер кластера* уменьшается на несколько байтов, требуемых для хранения. Указателю это не очень важно, но многие программы читают и пишут блоками, кратными степени двойки.

Оба недостатка предыдущей схемы организации файлов могут быть устранены, если указатели на следующие кластеры хранить в отдельной таблице, загружаемой в память. Таким образом, образуется связный список не самих блоков (кластеров) файла, а индексов, указывающих на эти блоки (рис. 7.16).



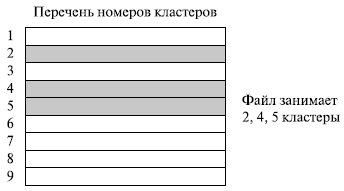
**Рис. 7.16.**  Вариант физической организации файлов

Такая таблица, называемая FAT-таблицей (File Allocation Table), используется в файловых системах MS-DOS и Windows (FAT 16 и FAT 32). Файлу выделяется память на диске в виде связного списка кластеров. Номер первого кластера запоминается в записи каталога, где хранятся характеристики этого файла. С каждым кластером диска связывается индекс. Индексы располагаются в FAT-таблице в отдельной области диска. Когда память свободна, все индексы равны нулю. Если некоторый кластер N назначен файлу, то индекс этого кластера либо становится равным номеру M следующего кластера файла, либо принимает специальное значение, являющееся признаком того, что кластер является последним для файла. Вообще индексы могут содержать следующую информацию о кластере диска (для FAT 32):

* не используется (Unused) – 0000.0000;
* используется файлом (Cluster in use by a file) – значение, отличное от 000.000, FFFF.FFFF и FFFF.FFF7;
* плохой кластер (*Bad cluster*) – FFFF.FFF7;
* последний кластер файла (Last cluster in a file) – FFFF.FFFF.

При такой организации сохраняются все достоинства второго метода организации файлов: отсутствие фрагментации, отсутствие проблем при изменении размера. Кроме того, данный способ обладает дополнительными преимуществами: для доступа к произвольному кластеру файла не требуется последовательно считывать его кластеры, достаточно прочитать FAT-таблицу, отсчитать нужное количество кластеров файла по цепочке и определить номера нужного кластера. Во-вторых, данные файла заполняют кластер целиком в объеме, кратном степени двойки.

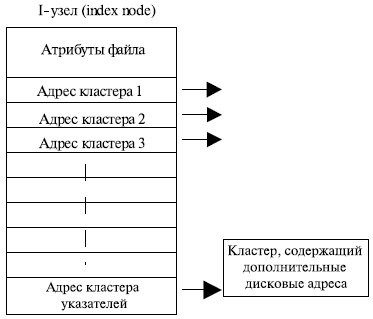
Еще один способ заключается в простом перечислении номеров кластеров, занимаемых файлом (рис. 7.17). Этот перечень и служит адресом файла. Недостаток такого подхода – длина адреса зависит от размера файла. Достоинства – высокая скорость доступа к произвольному кластеру благодаря прямой адресации, отсутствие внешней фрагментации.



**Рис. 7.17.**  Вариант физической организации файлов

Эффективный метод организации файлов, используемый в Unix-подобных операционных системах, состоит в связывании с каждым файлом структуры данных, называемой i-узлами. Такой узел содержит атрибуты файла и адреса кластеров файла (рис. 7.18). Преимущество такой схемы перед FAT-таблицей заключается в том, что каждый конкретный i-узел должен находиться в памяти только тогда, когда открыт соответствующий ему файл. Если каждый узел занимает n байт, а одновременно может быть открыто k файлов, то массив i-узлов займет в памяти k \* n байт, что значительно меньше, чем FAT-таблица.

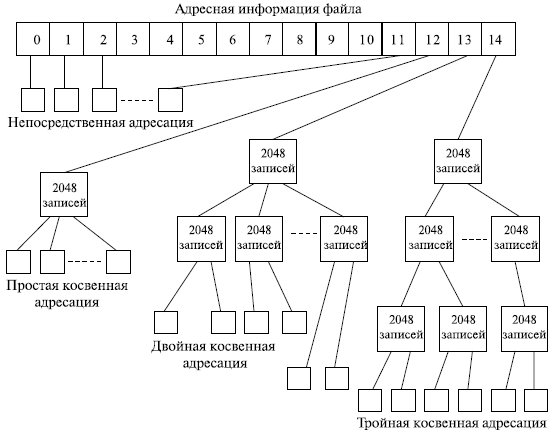
Это объясняется тем, что размер FAT-таблицы растет линейно с размером диска и даже быстрее, чем линейно, так как с увеличением количества кластеров на диске может потребоваться увеличить разрядность числа для хранения их номеров.



**Рис. 7.18.**  вариант физической организации файлов

Достоинством i-узлов является также высокая скорость доступа к произвольному кластеру файла, так как здесь применяется прямая адресация. Фрагментация на уровне кластеров также отсутствует.

Однако с такой схемой связана проблема, заключающаяся в том, что при выделении каждому файлу фиксированного количества адресов кластеров этого количества может не хватить. Выход из этой ситуации может быть в сочетании прямой и косвенной адресации. Такой поход реализован в файловой системе *ufs*, используемой в ОС UNIX, схема адресации в которой приведена на рис. 7.19.



**Рис. 7.19.**  Файловая система ufs

Для хранения адреса файла выделено 15 полей, каждое из которых состоит из 4 байт. Если размер файлов меньше или равен 12 кластерам, то номера этих кластеров непосредственно перечисляются в первых двенадцати полях адреса. Если кластер имеет размер 8 Кбайт, то можно адресовать файл размеров до 8 Кбайт \* 12 = 98304 байт. Если *размер кластера* превышает 12 кластеров, то следующее 13 поле содержит адрес кластера, в котором могут быть расположены номера следующих кластеров, и размер файла может возрасти до 8192 \* (12 + 2048) = 16.875.520 байт.

Следующий уровень адресации, обеспечиваемый 14-м полем, позволяет адресовать до 8192 \* (12 + 2048 + 20482) = 3,43766\*1020 байт. Если и этого недостаточно, используется следующее 15-е поле. В этом случае максимальный размер файла может составить 8192 \* (12 + 2048 + 20482 + 20483) = 7,0403\*1013 байт.

При этом объеме самой адресной информации составит всего 0,05% от объема адресуемых данных (задачи!!!).

Метод перечисления адресов кластеров файла задействован и в файловой системе NTFS, применяемой в Windows NT/2000/2003. Для сокращения объема адресной информации в NTFS адресуются не кластеры файла, а непрерывные области, состоящие из смежных кластеров диска. Каждая такая область называется экстентом (extent) и описывается двумя числами: номером начального кластера и количеством кластеров.

**7.17. Физическая организация FAT-системы**

Для обеспечения доступа приложений к файлам операционная система с файловой системой FAT использует следующие структуры:

* загрузочные секторы главного и *дополнительных разделов*;
* загрузочные секторы логических дисков (разделов);
* корневой каталог;
* область данных;
* цилиндр для выполнения диагностических операций чтения-записи.

На дискетах, в отличие от жесткого диска, нет загрузочных секторов главного и *дополнительных разделов* и диагностического цилиндра. Эти структуры создаются программой Fdisk, которая не применяется для дискет, так как они на разделы не разбиваются. Чтобы установить на один жесткий диск несколько операционных систем, его надо разбить на разделы. В загрузочном секторе главного раздела создается таблица списка разделов.

Загрузочный сектор главного раздела (называемый главной загрузочной записью – *Master Boot Record* – *MBR*) является первым сектором на жестком диске (цилиндр 0, головка 0, сектор 1) и состоит из двух элементов [10]:

* таблица главного раздела, содержащая список разделов (максимум четыре) и расположение загрузочных секторов соответствующих логических дисков (первая и последняя головки, первый и последний цилиндры с соответствующими значениями секторов, а также количество секторов);
* главный загрузочный код – небольшая программа, которая выполняется системой BIOS. Основная функция этого кода – передача управления в раздел, который обозначен как активный (загрузочный).

Загрузочный сектор раздела содержит:

* блок параметров диска, в котором содержится информация о разделе (размер, количество секторов, *размер кластера*, метка тома и др.);
* загрузочный код – программу, с которой начинается процесс загрузки операционной системы (для Ms-Dos и Windows 9x – файл Io.sys).

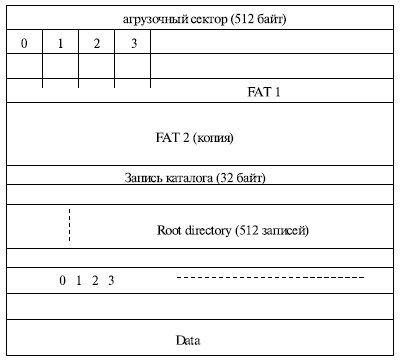
Загрузочные секторы логических дисков создаются программой Format. Они похожи на загрузочные диски разделов. Однако при загрузке выполняется код только того сектора, который находится в активном разделе.

Логический диск, отформатированный программой Fdisk, состоит из следующих областей (рис. 7.20):

* загрузочный сектор;
* основная FAT-таблица, содержащая информацию о размещении файлов и каталогов на диске;
* копия FAT-таблицы;
* корневой каталог – фиксированная область (16 Кбайт для жесткого диска), позволяющая хранить 512 записей о файлах и каталогах (каждая запись состоит из 32 байтов);
* область данных для размещения всех файлов и каталогов, кроме корневого каталога.

Первые две записи FAT зарезервированы и содержат информацию о самой FAT, все остальные указывают на соответствующие кластеры диска. Индексный указатель принимает значение, характеризующее состояние связанного с ним кластера (для FAT 16):

* кластер свободен (0000h);
* кластер используется (любое значение, кроме специальных);
* последний кластер файла (FFF8h – FFFFh);
* кластер поврежден (FFF7h);
* резервный кластер (FFF6h).



**Рис. 7.20.**  FAT-система

Размер FAT-таблицы определяется количеством кластера. Разрядность индексного указателя FAT-таблицы должна быть такой, чтобы можно было задать максимальный номер кластера диска определенного объема. В соответствии с разрядностью дискового указателя существуют несколько разновидностей FAT: FAT12, *FAT16*, FAT32 (соответственно 212, 216 и 232 кластеров). Тип используемой FAT определяется программой Fdisk, хотя и записываются они в процессе форматирования высокого уровня программы Format. На всех дискетах применяется FAT 12, на жестких дисках до 512 Мбайт – *FAT16*, на жестких дисках, имеющих большую емкость при использовании Windows 95 OSR2 и Windows98 – FAT 32 (вообще *размер кластера* может быть от 1 до 128 секторов или от 512 байт до 64 Кбайт). Максимальный размер раздела *FAT16* ограничен объемом 4 Гбайт (216 = 65536 кластеров по 64 Кбайт). Максимальный размер раздела FAT 32 практически не ограничен (232 кластеров по 32 Кбайт).

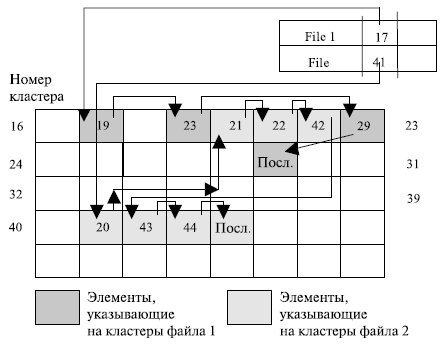
За копией FAT-таблицы следует корневой каталог – база данных, содержащая информацию о записанных на диске данных. Каждая запись в ней имеет длину 32 байта и содержит всю информацию о файле, которой располагает операционная система. Формат записи приведен ниже.

|  |  |  |  |
| --- | --- | --- | --- |
| **Смещение** | | | **Описание** |
| Hex | Dec | Длина поля |
| 00h | 0 | 8 байт | Имя файла |
| 08h | 8 | 3 байт | Расширение файла |
| 0Bh | 11 | 1 байт | Атрибуты файла |
| 0Ch | 12 | 10 байт | Зарезервировано |
| 16h | 22 | 2 байт | Время создания |
| 18h | 24 | 2 байт | Дата создания |
| 1Ah | 26 | 2 байт | Начальный кластер |
| 1Ch | 28 | 4 байт | Размер файла в байтах |

Информация о расположении файла, то есть о расположении оставшихся кластеров, содержится в FAT-таблице. В процессе работы системы кластеры файла могут оказаться не в смежных областях, а будут чередоваться с кластерами других файлов. Однако эту цепочку кластеров легко выделить, зная начальный кластер файлов. На рис. 7.21 показан пример размещения двух файлов.

В корневом каталоге имеются записи не только о файлах, но и подкаталогах. Эти записи имеют точно такую же структуру, что и записи корневого каталога. Признак подкаталогов указывается в атрибутах файла, т.е. можно считать, что подкаталог – это специальный файл. Структура *атрибутивного* байта показана ниже.

|  |  |  |
| --- | --- | --- |
| **Позиция бита в шестнадцатеричном формате** | **Значение** | **Описание** |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01h | Только чтение |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 02h | Скрытый |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 04h | Системный |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 08h | Метка тома |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 10h | Подкаталог |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 20h | Архивный (измененный) |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 40h | Зарезервировано |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 80h | Зарезервировано |



**Рис. 7.21.**  Пример размещения двух файлов

Файловые системы FAT 12 и *FAT16* оперируют с именами файлов, составленных по схеме 8.3 (имя, расширение). В Windows 95 с появлением 32-разрядной виртуальной FAT-*VFAT* (Virtual file allocation table) поддерживаются имена длиной 255 символов (заметим, что изменился лишь программный код, поддерживающий*FAT16*, он стал 32-м). Для обеспечения обратной совместимости ОС создает его псевдоним, удовлетворяющий стандарту 8.3. Делается это следующим образом.

1. Первые 3 символа после последней точки в длинном имени файла становятся расширением псевдонима.
2. Первые шесть символов длинного имени файла, за исключением пробелов, которые игнорируются, преобразуются в символы верхнего регистра и становятся шестью символами стандартного имени файла. Недопустимые символы (+ , ; = [28] ), которые могут использоваться в Windows 95, преобразуются в символы подчеркивания.
3. Добавляются символы ~1 (седьмой и восьмой) к псевдониму имени файла.

Если первые шесть символов нескольких файлов одни и те же, то добавляются символы ~2, ~3 и т.д.

*VFAT* хранит псевдонимы длинных имен в поле стандартных имен файлов записи каталога файла. Таким образом, все версии DOS и Windows могут получить доступ к файлу под длинным именем с помощью его псевдонима. Остается проблема: как хранить 255 символов имени файлов 32 байт записи каталога? Разработчики файловой системы решили эту проблему следующим образом: были добавлены дополнительные записи каталога для хранения длинных имен файлов. Чтобы предыдущие версии не повредили эти дополнительные записи каталога, *VFAT* устанавливает для них атрибуты, которые нельзя использовать для обычного файла: только для чтения, скрытый, системный и метка тома. Такие атрибуты DOS игнорирует, а следовательно, длинные имена файлов остаются нетронутыми. Подобным же образом решается проблема длинных имен в Windows NT/2000/2003/XP, применяющих для хранения имен двухбайтовый формат на каждый символ – Unicode.

Как уже отмечалось, выбор типа FAT-системы во многом определяется емкостью жесткого диска. При использовании *FAT16* нельзя создать раздел емкостью более 2-х Гбайт. Для устранения этого ограничения фирма Microsoft разработала FAT 32. Она работает как FAT 16, но имеет отличие в организации хранения данных. Кроме того, FAT 32 можно установить с помощью программы Fdisk. Впервые FAT 32 была реализована в Windows 95 OEM *Service Release* 2 (OSR2). Она встроена и в Windows 98/Ме/NT/2000.

Основное преимущество FAT 32 – возможность использования 32-разрядных записей вместо 16-разрядных, что приводит к увеличению кластеров (вместо 216=65536) до 268 435 456 в разделе. Это значение в Windows 95 OSR2 эквивалентно 228, а не 232, поскольку 4 бита из 32 зарезервированы для других целей.

При работе в FAT 32 размер раздела может достигать 2 Tбайт при кластере размером 8, 16 или 32 Кбайт. Новая файловая система может иметь 232 кластеров размером 512 байт, а размер единичного файла может составить 4 Гбайт. Реально FAT 32 поддерживает максимальный размер тома до 32 Гбайт. Это связано с тем, что в Windows 2000 это ограничение обусловлено программой Format. Вообще максимально возможный том – 2 Tбайт при кластере 32 Кбайт.

Существует важное отличие FAT 32 от ее предшественниц – положение корневого каталога: он может располагаться в любом месте раздела и иметь любой раздел. Это обеспечивает динамическое изменение размера раздела. Независимые разработчики использовали это свойство. Так, фирма Power-*Quest* создала программу Partion Magic, позволяющую переопределять разделы после их создания.

Файловая система FAT 32 также использует преимущество двух копий FAT. Как и в FAT 16, в FAT 32 первая копия является основной и периодически копирует данные в дополнительную копию FAT. При проблемах с главной копией FAT системы переключаются в дополнительную копию, которая становится главной.

Примечание: программа Fdisk автоматически определяет *размер кластера* на основе выбранной файловой системы и размерам раздела. Однако существует недокументированный параметр команды Format, позволяющий явно указать *размер кластера*: Format/z:n, где n –*размер кластера* в байтах, кратный 512.

**7.18. Файловые операции**

Набор файловых операций

Файловая система ОС должна предоставлять пользователям набор операций для работы с файлами, оформленный в виде системных вызовов. В различных ОС имеются различные наборы файловых операций. Наиболее часто встречающимися системными вызовами для работы с файлами являются [13, 17]:

1. Create (создание). Файл создается без данных. Этот системный вызов объявляет о появлении нового файла и позволяет установить некоторые его атрибуты;
2. Delete (удаление). Ненужный файл удаляется, чтобы освободить пространство на диске;
3. Open (открытие). До использования файла его нужно открыть. Данный вызов позволяет прочитать атрибуты файла и список дисковых адресов для быстрого доступа к содержимому файла;
4. Close (закрытие). После завершения операций с файлом его атрибуты и дисковые адреса не нужны. Файл следует закрыть, чтобы освободить пространство во внутренней таблице;
5. Read (чтение). Файл читается с текущей позиции. Процесс, работающий с файлом, должен указать (открыть) буфер и количество читаемых данных;
6. Write (запись). Данные записываются в файл в текущую позицию. Если она находится в конце файла, его размер автоматически увеличивается. В противном случае запись производится поверх существующих данных;
7. Append (добавление). Это усеченная форма предыдущего вызова. Данные добавляются в конец файла;
8. Seek (поиск). Данный системный вызов устанавливает файловый указатель в определенную позицию;
9. Get attributes (получение *атрибутов). Процессам* для работы с файлами бывает необходимо получить их атрибуты;
10. Set attributes (установка атрибутов). Этот вызов позволяет установить необходимые атрибуты файлу после его создания;
11. Rename (переименование). Этот системный вызов позволяет изменить имя файла. Однако такое действие можно выполнить копированием файла. Поэтому данный системный вызов не является необходимым;
12. Execute (выполнить). Используя этот системный вызов, файл можно запустить на выполнение.

Рассмотрим примеры файловых операций в ОС Windows 2000 и UNIX. Как и в других ОС, в Windows 2000 есть свой набор системных вызовов, которые она может выполнять. Однако корпорация Microsoft никогда не публиковала список системных вызовов Windows, кроме того, она постоянно меняет их от одного выпуска к другому [17]. Вместо этого Microsoft определила набор функциональных вызовов, называемый Win 32 API (Win 32 Application Programming Interface). Эти вызовы опубликованы и полностью документированы. Они представляют собой библиотечные процедуры, которые либо обращаются к системным вызовам, чтобы выполнить требуемую работу, либо выполняют ее прямо в пространстве пользователя.

Философия Win 32 API заключается в предоставлении всеобъемлющего интерфейса, с возможностью выполнить одно и то же требование несколькими (тремя-четырьмя) способами. В ОС UNIX все системные вызовы формируют *минимальный интерфейс*: удаление даже одного из них приведет к снижению функциональности ОС.

Многие вызовы API создают объекты ядра того или иного типа (файлы, процессы, потоки, каналы и т.д.). Каждый вызов, создающий объект, возвращает вызывающему процессу результат, называемый дескриптором (небольшое целое число). Дескриптор используется впоследствии для выполнения операций с объектами. Он не может быть передан другому процессу и использован им. Однако при определенных обстоятельствах дескриптор может быть дублирован и передан другому процессу защищенным способом, что предоставляет второму процессу контролируемый доступ к объекту, принадлежащему первому процессу. С каждым объектом ассоциирован дескриптор безопасности, описывающий, кто и какие действия может, а какие не может выполнять с данным объектом.

Основные функции Win 32 API для файлового ввода-вывода и соответствующие системные вызовы ОС UNIX приведены ниже.

|  |  |  |
| --- | --- | --- |
| Функция Win 32 API | Системные вызовы UNIX | Описание |
| CreateFile | open | Создать или открыть файл; вернуть дескриптор файла |
| DeleteFile | unlink | Удалить существующий файл |
| CloseHandle | close | Закрыть файл |
| *ReadFile* | read | Прочитать данные из файла |
| WriteFile | write | Записать данные в файл |
| SetFilePointer | lseek | Установить указатель в файле в определенную позицию |
| GetFileAttributes | stat | Вернуть атрибуты файла |
| LockFile | fcntl | Заблокировать область файла для обеспечения взаимного исключения |
| UnlockFile | fcntl | Отменить блокировку области файла |

Аналогично файловым операциям обстоит дело с операциями управления каталогами. Основные функции Win 32 API и системные вызовы UNIX для управления каталогами приведены ниже.

|  |  |  |
| --- | --- | --- |
| Функция Win 32 API | Системные вызовы UNIX | Описание |
| CreateDirectory | mkdir | Создать новый каталог |
| RemoveDirectory | rmdir | Удалить пустой каталог |
| FindFirstFile | opendir | Инициализация, чтобы начать чтение записей каталога |
| FindNextFile | readdir | Прочитать следующую запись каталога |
| MoveFile | rename | Переместить файл из одного каталога в другой |
| SetCurrentDirectory | chdir | Изменить текущий рабочий каталог |

Способы выполнения файловых операций

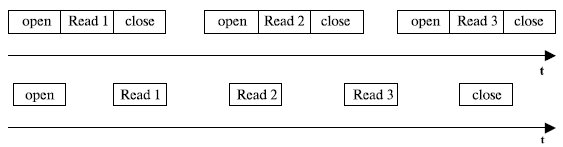
Чаще всего с одним и тем же файлом пользователь выполняет не одну, а последовательность операций. Независимо от набора этих операций операционной системе необходимо выполнить ряд постоянных (универсальных) для всех операций действий.

1. По символьному имени файла найти его характеристики, которые хранятся в файловой системе на диске.
2. Скопировать характеристики в оперативную память, поскольку только в этом случае программный код может их использовать.
3. На основании характеристик файла проверить права пользователя на выполнение запрошенной операции.
4. Очисть область памяти, отведенную под временное хранение характеристик файла.

Кроме того, каждая операция включает ряд уникальных для нее действий, например, чтение определенного набора кластеров диска, удаление файла, изменение его атрибутов и т.п.

ОС может выполнить последовательность действий над файлами двумя способами (см. рис. рис. 7.22).

1. Для каждой операции выполняются как универсальные, так и уникальные действия. Такая схема иногда называется схемой без заполнения состояния операции (*stateless*).
2. Все универсальные действия выполняются в начале и конце последовательности операций, а для каждой промежуточной операции выполняются только уникальные действия.



**Рис. 7.22.**  Варианты выполнения последовательности действий над файлами

Подавляющее большинство файловых систем поддерживает второй способ, как более экономичный и быстрый. Однако первый способ более устойчив к сбоям в работе системы, так как каждая операция является самодостаточной и не зависит от результата предыдущей. Поэтому первый способ иногда применяется в распределенных сетевых файловых системах, когда сбои из-за потерь пакетов или отказов одного из сетевых узлов более вероятны, чем при локальном доступе к данным.

При втором способе в ФС вводится два специальных системных вызова: open и close. Первый выполняется перед началом любой последовательности операций с файлом, а второй – после окончания работы с файлом.

Основной задачей вызова open является преобразование символьного имени файла в его уникальное числовое имя, копирование характеристик файла из дисковой области в буфер оперативной памяти и проверка прав пользователя на выполнение запрошенной операции. Вызов close освобождает буфер с характеристиками файла и делает невозможным продолжение операций с файлами без его повторного открытия.

Приведем несколько примеров системных вызовов для работы с файлами. Системный вызов create в ОС UNIX работает с двумя аргументами: символьным именем открываемого файла и режимом защиты. Так команда

fd = create ("abc", mode);

создает файл abc с режимом защиты, указанным в переменной mode. Биты mode определяют круг пользователей, которые могут получить доступ к файлам, и уровень предоставляемого им доступа. Системный вызов create не только создает новый файл, но также открывает его для записи. Чтобы последующие системные вызовы могли получить доступ к файлу, успешный системный вызов create возвращает небольшое неотрицательное целое число – дескриптор файла – fd. Если системный вызов выполняется с существующим файлом, длина этого файла уменьшается до 0, а все содержимое теряется.

Чтобы прочитать данные из существующего файла или записать в него данные, файл сначала нужно открыть с помощью системного вызова open с двумя аргументами: символьным именем файла и режимом открытия файла (для записи, чтения или того т другого), например

fd = open ("file", how);

Системные вызовы create и open возвращают наименьший неиспользуемый в данный момент дескриптор файла. Когда программа начинает выполнение стандартным образом, файлы с дескрипторами 0, 1 и 2 уже открыты для стандартного ввода, стандартного вывода и стандартного потока сообщений об ошибках.

В стандарте языка Си отсутствуют средства ввода-вывода. Все операции ввода-вывода реализуются с помощью функций, находящихся в библиотеке языка, поставляемой в составе системы программирования Си. На *стандартный поток ввода* ссылаются через указатель stdin, вывода – stdout, сообщений об ошибках –stderr. По умолчанию потоку ввода stdin ставится в соответствие клавиатура, а потокам stdout и stderr – экран дисплея.

Для ввода-вывода данных с помощью стандартных потоков в библиотеке Си определены функции:

* getchar ( )/putchar ( ) – ввод-вывод отдельного символа;
* gets ( )/ puts ( ) – ввод-вывод строки;
* scanf ( )/ printf ( ) – ввод-вывод в режиме форматирования данных.

Процесс в любое время может организовать ввод данных из стандартного файла ввода, выполнить символьный вызов:

read (stdin, buffer, nbyts);

Аналогично организуется вывод в стандартный файл вывода

write (stdout, buffer, nbytes).

При работе в Windows 2000 с помощью функции CreateFile можно создать файл и получить дескриптор к нему. Эту же функцию следует применять и для открытия уже существующего файла, так как в Win 32 API нет специальной функции File Open. Параметры функций, как правило, многочисленны, например, функция CreateFile имеет семь параметров:

1. указатель на имя файла, который нужно создать или открыть;
2. флаги (биты), указывающие, может ли с этим файлом выполняться чтение, запись или то и другое;
3. флаги, указывающие, может ли этот файл одновременно открываться несколькими процессами;
4. указатель на описатель защиты, сообщение, кто может получать доступ к файлу;
5. флаги, сообщающие, что делать, если файл существует или, наоборот, не существует;
6. флаги, управляющие архивацией, сжатием и т.д.;
7. дескриптор файла, чьи атрибуты должны быть клонированы для нового файла,

Fd = CreateFile ("data", GENERIC\_READ, O, NULL, OPEN\_EXSTING, O, NULL).

**7.19. Контроль доступа к файлам**

Файлы – один из видов разделяемых ресурсов, доступ к которым ОС должна контролировать. Существуют и другие виды ресурсов, с которыми пользователи работают в режиме совместного использования: принтеры, модемы, графопостроители и т.п. Во всех этих случаях пользователи или процессы пытаются выполнить с разделяемым ресурсом определенные операции, а ОС должны решить, имеют ли пользователи на это право. Пользователи являются субъектами доступа, а разделяемые ресурсы – объектами. Пользователь осуществляет доступ к объектам не непосредственно, а c помощью прикладных процессов, которые запускаются от его имени.

Для каждого типа объекта существует набор операций, которые можно с ним выполнять. Система контроля доступа ОС должна предоставлять средства для задания прав пользователей по отношению к объектам дифференцированно по операциям.

В качестве субъектов доступа могут выступать как отдельные пользователи, так и группы пользователей. Объединение пользователей с одинаковыми правами в группу и задания прав доступа в целом для группы является одним из основных приемов администрирования в больших системах.

У каждого объекта доступа существует владелец. Владелец объекта имеет право выполнить с ним любые допустимые для данного объекта операции. Во многих ОС существует особый пользователь – администратор "superuser", который имеет все права по отношению к объектам системы, не обязательно являясь их владельцем. Эти права (полномочия) необходимы администратору для управления политикой доступа.

Различают два основных подхода к определению прав доступа [13].

1. Избирательный доступ – ситуация, когда владелец объекта определяет допустимые операции с объектом. Этот подход называется также произвольным доступом, так как позволяет администратору и владельцам объекта определить права доступа произвольным образом, по их желанию. Однако администратор по умолчанию наделен всеми правами.
2. Мандатный доступ (от mandatory – принудительный) – подход к определению прав доступа, при котором система (администратор) наделяет пользователя или группу определенными правами по отношению к каждому разделяемому ресурсу. В этом случае группы пользователей образуют строгую иерархию, причем каждая группа пользуется всеми правами группы более низкого уровня иерархии.

Мандатные системы доступа считаются более надежными, но менее гибкими. Обычно они применяются в системах с повышенными требованиями к защите информации.

Каждый пользователь (группа) имеет символьное имя, а также уникальный числовой идентификатор. При выполнении процедуры логического входа в систему пользователь сообщает свое символьное имя или пароль. Все идентификационные данные, а также сведения о вхождении пользователя в группы хранятся в специальном файле (UNIX) или базе данных (Windows NT).

Вход пользователя в систему порождает процесс – оболочку, который поддерживает диалог с пользователем и запускает для него другие процессы. Любой порождаемый процесс наследует идентификаторы пользователя и групп от процесса родителя.

В разных ОС для одних и тех же типов ресурсов может быть определен свой список дифференцируемых операций доступа. Для файловых объектов этот список может включить операции, которые рассмотрены выше.

Набор файловых операций может включать всего несколько укрупненных операций, например, для файлов и каталогов: читать, писать и выполнять.

Возможна комбинация двух подходов – детальный уровень и укрупненный. Например, в Windows NT/2000/2003 администратор работает на укрупненном уровне, а при желании может перейти на детальный.

В самом общем случае права доступа могут быть описаны матрицей прав доступа, в которой столбцы соответствуют всем файлам системы, а строки – всем пользователям. На пересечении строк и столбцов указываются разрешенные операции. Однако реально для тысяч и десятков тысяч файлов в системе пользоваться такой матрицей неудобно. Поэтому она хранится по частям, т.е. для каждого файла и каталога создается список управления доступом (Access Control List, ACL), в котором описываются права на выполнение операций пользователей и групп пользователей по отношению к этому файлу или каталогу. Список управления доступом является частью характеристик файла или каталога и хранится на диске в соответствующей области. Не все файловые системы поддерживают списки управления доступом, например, FAT не поддерживает, поскольку разрабатывалась для однопрограммной, однопользовательской ОС MS-DOS.

Обобщено формат списка управления доступом (ACL) можно представить в виде набора идентификаторов пользователей и групп пользователей, в котором для каждого идентификатора указывается набор разрешенных операций над объектом. Сам список ACL состоит из элементов управления доступом (Access Control Element, ACE), которые соответствуют одному идентификатору. Список ACL с добавлением идентификатора владельца называют характеристиками безопасности.

Рассмотрим организацию контроля доступа в ОС Windows NT/2000/XP. Система управления доступом в этой операционной системе отличается высокой степенью гибкости, которая достигается за счет большого разнообразия субъектов и объектов доступа и детализации операции доступа.

Для разделяемых ресурсов в Windows XP применяется общая модель объекта, которая содержит такие характеристики безопасности, как набор допустимых операций, идентификатор владельца, список управления доступом.

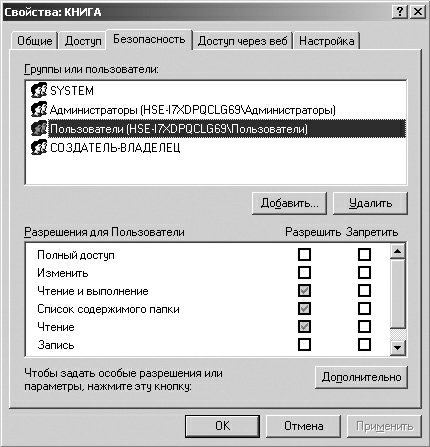
Проверки прав доступа для объектов любого типа выполняются централизованно с помощью монитора безопасности (Security Reference Monitor), работающего в привилегированном режиме.

Для системы безопасности Windows характерно большое количество различных встроенных (предопределенных) субъектов доступа – отдельных пользователей и групп (Administrator, System, Guest, группы Users, Administrators, Account, Operators и др.). Смысл этих встроенных пользователей и групп состоит в том, что они наделены определенными правами. Это облегчает работу администратора по созданию эффективной системы разграничения доступа. Во-первых, за счет того, что нового пользователя можно внести в какую-то группу. Во-вторых, можно добавлять (изымать) права встроенных групп. Наконец, можно создавать новые группы с уникальным набором прав.

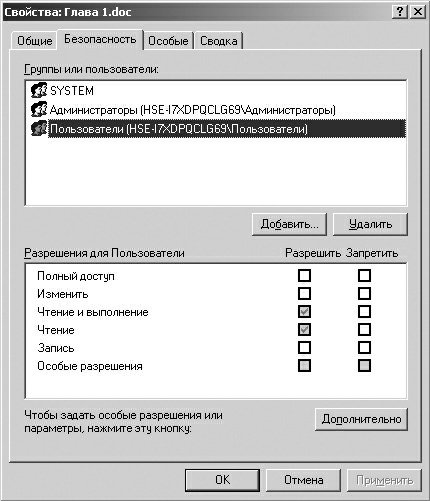
Все объекты при создании снабжаются дескрипторами безопасности, содержащими список управления доступом и список пользователей и групп, имеющих доступ к данному объекту. Владелец объекта, обычно пользователь, который его создал, обладает возможностью изменять ACL объекта, чтобы позволить или не позволить другим осуществлять доступ к объекту. Он может выполнить требуемую операцию с объектом, став его владельцем (такая возможность предусмотрена), а затем как владелец получить полный набор разрешений. Однако вернуть владение предыдущему владельцу объекта администратор не может, поэтому пользователь всегда может узнать о том, что с его файлом (принтером и т.п.) работал администратор.

В Windows NT/2000/XP администратор может управлять доступом пользователей к каталогам и файлам только в разделах диска, в которых установлена файловая система NTFS. Разделы FAT не поддерживаются, так как в этой ФС у файлов и каталогов отсутствуют атрибуты для хранения списков управления доступом.

Разрешения в Windows бывают индивидуальные (специальные) и стандартные. Индивидуальные относятся к элементарным операциям над каталогами и файлами, а стандартные разрешения являются объединением нескольких индивидуальных разрешений. На рис. 7.23 и рис. 7.24 приведены шесть стандартных разрешений (элементарных операций), смысл которых отличается для каталогов и файлов.

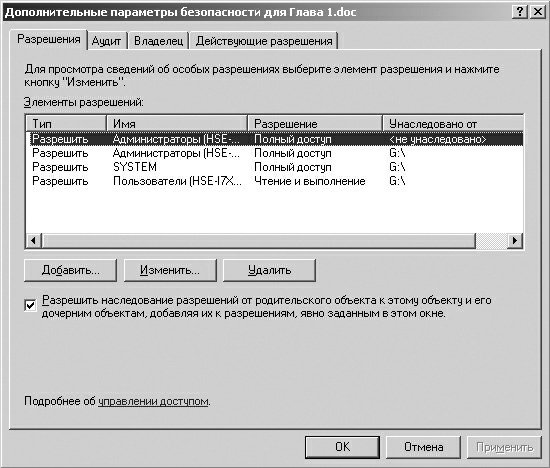


**Рис. 7.23.**  Стандартные разрешения для каталогов



**Рис. 7.24.**  Стандартные разрешения для файлов

На рис. 7.25 показана возможность установки индивидуальных разрешений для файлов.



**Рис. 7.25.**  Индивидуальные разрешения для файлов