Назначение и функции ОС

ОС выполняет две по существу мало связанные функции: обеспечение пользователю-программисту удобств посредством предоставления для него расширенной машины и повышение эффективности использования компьютера путем рационального управления его ресурсами.

расширенной машины

Программа, которая скрывает от программиста все реалии аппаратуры и предоставляет возможность простого, удобного просмотра указанных файлов, чтения или записи - это, конечно, операционная система. Точно также, как ОС ограждает программистов от аппаратуры дискового накопителя и предоставляет ему простой файловый интерфейс, операционная система берет на себя все малоприятные дела, связанные с обработкой прерываний, управлением таймерами и оперативной памятью, а также другие низкоуровневые проблемы. В каждом случае та абстрактная, воображаемая машина, с которой, благодаря операционной системе, теперь может иметь дело пользователь, гораздо проще и удобнее в обращении, чем реальная аппаратура, лежащая в основе этой абстрактной машины.

управления его ресурсами

функцией ОС является распределение процессоров, памяти, устройств и данных между процессами, конкурирующими за эти ресурсы. ОС должна управлять всеми ресурсами вычислительной машины таким образом, чтобы обеспечить максимальную эффективность ее функционирования. Критерием эффективности может быть, например, пропускная способность или реактивность системы. Управление ресурсами включает решение двух общих, не зависящих от типа ресурса задач:

- планирование ресурса - то есть определение, кому, когда, а для делимых ресурсов и в каком количестве, необходимо выделить данный ресурс;
- отслеживание состояния ресурса - то есть поддержание оперативной информации о том, занят или не занят ресурс, а для делимых ресурсов - какое количество ресурса уже распределено, а какое свободно.

Эволюция ОС

Первый период (1945 -1955)

В середине 40-х были созданы первые ламповые вычислительные устройства. В то время одна и та же группа людей участвовала и в проектировании, и в эксплуатации, и в программировании вычислительной машины. Это была скорее научно-исследовательская работа в области вычислительной техники. Нет ОС – пульт управления. Не было никакого другого системного программного обеспечения, кроме библиотек математических и служебных подпрограмм.

Второй период (1955 - 1965)

новой технической базы - полупроводниковых элементов. Компьютеры второго поколения стали более надежными, теперь они смогли непрерывно работать настолько долго, чтобы на них можно было возложить выполнение действительно практически важных задач. Разделение персонала. Первые алгоритмические языки. Первые системные программы – компиляторы. Появились первые системы пакетной обработки, которые просто автоматизировали запуск одной программ за другой и тем самым увеличивали коэффициент загрузки процессора. Системы пакетной обработки явились прообразом современных операционных систем, они стали первыми системными программами, предназначенными для управления вычислительным процессом. В ходе реализации систем пакетной обработки был разработан формализованный язык управления заданиями, с помощью которого программист сообщал системе и оператору, какую работу он хочет выполнить на вычислительной машине. Совокупность нескольких заданий, как правило в виде колоды перфокарт, получила название пакета заданий.

Третий период (1965 - 1980)

переход от отдельных полупроводниковых элементов типа транзисторов к интегральным микросхемам. создание семейств программно-совместимых машин. Первым семейством программно-совместимых машин, построенных на интегральных микросхемах, явилась серия машин IBM/360. OS/360 и другие ей подобные операционные системы машин третьего поколения действительно удовлетворяли большинству требований потребителей. Важнейшим достижением ОС данного поколения явилась реализация мультипрограммирования. *Мультипрограммирование* - это способ организации вычислительного процесса, при котором на одном процессоре попеременно выполняются несколько программ. Пока одна программа выполняет операцию ввода-вывода, процессор не простаивает, как это происходило при последовательном выполнении программ (однопрограммный режим), а выполняет другую программу (многопрограммный режим). При этом каждая программа загружается в свой участок оперативной памяти, называемый разделом. ***Спулинг*** в то время определялся как способ организации вычислительного процесса, в соответствии с которым задания считывались с перфокарт на диск в том темпе, в котором они появлялись в помещении вычислительного центра, а затем, когда очередное задание завершалось, новое задание с диска загружалось в освободившийся раздел.

Четвертый период (1980 - настоящее время)

появлением больших интегральных схем (БИС). эра персональных компьютеров. На рынке операционных систем доминировали две системы: MS-DOS и UNIX. Однопрограммная однопользовательская ОС MS-DOS широко использовалась для компьютеров, построенных на базе микропроцессоров Intel 8088, а затем 80286, 80386 и 80486. Мультипрограммная многопользовательская ОС UNIX доминировала в среде "не-интеловских" компьютеров, особенно построенных на базе высокопроизводительных RISC-процессоров. Сетевые ОС.

Классификация ОС, виды построения ОС

Операционные системы могут различаться особенностями реализации внутренних алгоритмов управления основными ресурсами компьютера (процессорами, памятью, устройствами), особенностями использованных методов проектирования, типами аппаратных платформ, областями использования и многими другими свойствами.

Особенности алгоритмов управления ресурсами

Так, например, в зависимости от особенностей использованного алгоритма управления процессором, операционные системы делят на многозадачные и однозадачные, многопользовательские и однопользовательские, на системы, поддерживающие многонитевую обработку и не поддерживающие ее, на многопроцессорные и однопроцессорные системы.

Поддержка многозадачности. По числу одновременно выполняемых задач разделены на два класса:

- однозадачные (например, MS-DOS, MSX) и
- многозадачные (ОС ЕС, OS/2, UNIX, Windows 95).

Однозадачные ОС в основном выполняют функцию предоставления пользователю виртуальной машины, делая более простым и удобным процесс взаимодействия пользователя с компьютером. Однозадачные ОС включают средства управления периферийными устройствами, средства управления файлами, средства общения с пользователем.

Многозадачные ОС, кроме вышеперечисленных функций, управляют разделением совместно используемых ресурсов, таких как процессор, оперативная память, файлы и внешние устройства.

Поддержка многопользовательского режима. По числу одновременно работающих пользователей ОС делятся на:

- однопользовательские (MS-DOS, Windows 3.x, ранние версии OS/2);
- многопользовательские (UNIX, Windows NT).

Главным отличием многопользовательских систем от однопользовательских является наличие средств защиты информации каждого пользователя от несанкционированного доступа других пользователей

Вытесняющая и невытесняющая многозадачность.

- невытесняющая многозадачность (NetWare, Windows 3.x);
- вытесняющая многозадачность (Windows NT, OS/2, UNIX).

Основным различием между вытесняющим и невытесняющим вариантами многозадачности является степень централизации механизма планирования процессов. В первом случае механизм планирования процессов целиком сосредоточен в операционной системе, а во втором - распределен между системой и прикладными программами. При невытесняющей многозадачности активный процесс выполняется до тех пор, пока он сам, по собственной инициативе, не отдаст управление операционной системе для того, чтобы та выбрала из очереди другой готовый к выполнению процесс. При вытесняющей многозадачности решение о переключении процессора с одного процесса на другой принимается операционной системой, а не самим активным процессом.

Поддержка многонитевости. Важным свойством операционных систем является возможность распараллеливания вычислений в рамках одной задачи.

Многопроцессорная обработка. Другим важным свойством ОС является отсутствие или наличие в ней средств поддержки многопроцессорной обработки - *мультипроцессирование*. Мультипроцессирование приводит к усложнению всех алгоритмов управления ресурсами.

Многопроцессорные ОС могут классифицироваться по способу организации вычислительного процесса в системе с многопроцессорной архитектурой: асимметричные ОС и симметричные ОС. Асимметричная ОС целиком выполняется только на одном из процессоров системы, распределяя прикладные задачи по остальным процессорам. Симметричная ОС полностью децентрализована и использует весь пул процессоров, разделяя их между системными и прикладными задачами.

Особенности аппаратных платформ

На свойства операционной системы непосредственное влияние оказывают аппаратные средства, на которые она ориентирована. По типу аппаратуры различают операционные системы персональных компьютеров, мини-компьютеров, мейнфреймов, кластеров и сетей ЭВМ

Особенности областей использования

Многозадачные ОС подразделяются на три типа в соответствии с использованными при их разработке критериями эффективности:

- системы пакетной обработки (например, ОС ЕС),
- системы разделения времени (UNIX, VMS),
- системы реального времени (QNX, RT/11).

Системы пакетной обработки предназначались для решения задач в основном вычислительного характера, не требующих быстрого получения результатов. Главной целью и критерием эффективности систем пакетной обработки является максимальная пропускная способность, то есть решение максимального числа задач в единицу времени. *Системы разделения времени* призваны исправить основной недостаток систем пакетной обработки - изоляцию пользователя-программиста от процесса выполнения его задач. Каждому пользователю системы разделения времени предоставляется терминал, с которого он может вести диалог со своей программой. Так как в системах разделения времени каждой задаче выделяется только квант процессорного времени, ни одна задача не занимает процессор надолго, и время ответа оказывается

приемлемым. Если квант выбран достаточно небольшим, то у всех пользователей, одновременно работающих на одной и той же машине, складывается впечатление, что каждый из них единолично использует машину. *Системы реального времени* применяются для управления различными техническими объектами, такими, например, как станок, спутник, научная экспериментальная установка или технологическими процессами, такими, как гальваническая линия, доменный процесс и т.п. Во всех этих случаях существует предельно допустимое время, в течение которого должна быть выполнена та или иная программа, управляющая объектом, в противном случае может произойти авария

Особенности методов построения

При описании операционной системы часто указываются особенности ее структурной организации и основные концепции, положенные в ее основу.

К таким базовым концепциям относятся:

- Способы построения ядра системы - монолитное ядро или микроядерный подход. Большинство ОС использует монолитное ядро, которое компонуется как одна программа, работающая в привилегированном режиме и использующая быстрые переходы с одной процедуры на другую, не требующие переключения из привилегированного режима в пользовательский и наоборот. Альтернативой является построение ОС на базе микроядра, работающего также в привилегированном режиме и выполняющего только минимум функций по управлению аппаратурой, в то время как функции ОС более высокого уровня выполняют специализированные компоненты ОС - серверы, работающие в пользовательском режиме. При таком построении ОС работает более медленно, так как часто выполняются переходы между привилегированным режимом и пользовательским, зато система получается более гибкой - ее функции можно наращивать, модифицировать или сужать, добавляя, модифицируя или исключая серверы пользовательского режима. Кроме того, серверы хорошо защищены друг от друга, как и любые пользовательские процессы.
- Построение ОС на базе объектно-ориентированного подхода дает возможность использовать все его достоинства, хорошо зарекомендовавшие себя на уровне приложений, внутри операционной системы, а именно: аккумуляцию удачных решений в форме стандартных объектов, возможность создания новых объектов на базе имеющихся с помощью механизма наследования, хорошую защиту данных за счет их инкапсуляции во внутренние структуры объекта, что делает данные недоступными для несанкционированного использования извне, структуризованность системы, состоящей из набора хорошо определенных объектов.
- Наличие нескольких прикладных сред дает возможность в рамках одной ОС одновременно выполнять приложения, разработанные для нескольких ОС. Многие современные операционные системы поддерживают одновременно прикладные среды MS-DOS, Windows, UNIX (POSIX), OS/2 или хотя бы некоторого подмножества из этого популярного набора. Концепция множественных прикладных сред наиболее просто реализуется в ОС на базе микроядра, над которым работают различные серверы, часть которых реализуют прикладную среду той или иной операционной системы.
- Распределенная организация операционной системы позволяет упростить работу пользователей и программистов в сетевых средах. В распределенной ОС реализованы механизмы, которые дают возможность пользователю представлять и воспринимать сеть в виде традиционного однопроцессорного компьютера. Характерными признаками распределенной организации ОС являются: наличие единой справочной службы разделяемых ресурсов, единой службы времени, использование механизма вызова удаленных процедур (RPC) для прозрачного распределения программных процедур по машинам, многонитевой обработки, позволяющей распараллеливать вычисления в рамках одной задачи и выполнять эту задачу сразу на нескольких компьютерах сети, а также наличие других распределенных служб.

Управление процессами, состояния процессов, очереди процессов.

Важнейшей частью операционной системы, непосредственно влияющей на функционирование вычислительной машины, является подсистема управления процессами. *Процесс* (или по-другому, задача) - абстракция, описывающая выполняющуюся программу. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных ресурсов. Подсистема управления процессами планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, а также занимается созданием и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие между процессами.

Состояние процессов

В многозадачной (многопроцессной) системе процесс может находиться в одном из трех основных состояний:

ВЫПОЛНЕНИЕ - активное состояние процесса, во время которого процесс обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;

ОЖИДАНИЕ - пассивное состояние процесса, процесс заблокирован, он не может выполняться по своим внутренним причинам, он ждет осуществления некоторого события, например, завершения операции ввода-вывода, получения сообщения от другого процесса, освобождения какого-либо необходимого ему ресурса;

ГОТОВНОСТЬ - также пассивное состояние процесса, но в этом случае процесс заблокирован в связи с внешними по отношению к нему обстоятельствами: процесс имеет все требуемые для него ресурсы, он готов выполняться, однако процессор занят выполнением другого процесса.

В ходе жизненного цикла каждый процесс переходит из одного состояния в другое в соответствии с алгоритмом планирования процессов, реализуемым в данной операционной системе. Типичный граф состояний процесса показан на рисунке 2.1.

В состоянии ВЫПОЛНЕНИЕ в однопроцессорной системе может находиться только один процесс, а в каждом из состояний ОЖИДАНИЕ и ГОТОВНОСТЬ - несколько процессов, эти процессы образуют очереди соответственно ожидающих и готовых процессов. Жизненный цикл процесса начинается с состояния ГОТОВНОСТЬ, когда процесс готов к выполнению и ждет своей очереди. При активизации процесс переходит в состояние ВЫПОЛНЕНИЕ и находится в нем до тех пор, пока либо он сам освободит процессор, перейдя в состояние ОЖИДАНИЯ какого-нибудь события, либо будет насильно "вытеснен" из процессора, например, вследствие исчерпания отведенного данному процессу кванта процессорного времени. В последнем случае процесс возвращается в состояние ГОТОВНОСТЬ. В это же состояние процесс переходит из состояния ОЖИДАНИЕ, после того, как ожидаемое событие произойдет.

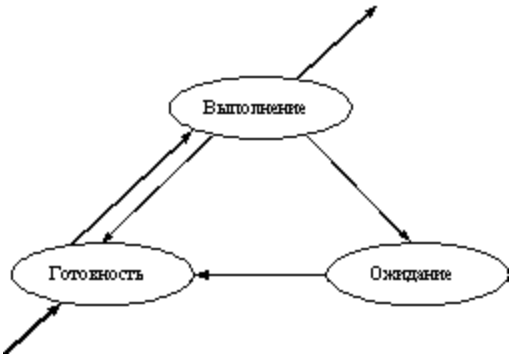


Рис. 2.1. Граф состояний процесса в многозадачной среде

Контекст и дескриптор процесса

На протяжении существования процесса его выполнение может быть многократно прервано и продолжено. Для того, чтобы возобновить выполнение процесса, необходимо восстановить состояние его операционной среды. Состояние операционной среды отображается состоянием регистров и программного счетчика, режимом работы процессора, указателями на открытые файлы, информацией о незавершенных операциях ввода-вывода, кодами ошибок выполняемых данным процессом системных вызовов и т.д. Эта информация называется *контекстом процесса*.

Кроме этого, операционной системе для реализации планирования процессов требуется дополнительная информация: идентификатор процесса, состояние процесса, данные о степени привилегированности процесса, место нахождения кодового сегмента и другая информация. В некоторых ОС (например, в ОС UNIX) информацию такого рода, используемую ОС для планирования процессов, называют *дескриптором процесса*.

Дескриптор процесса по сравнению с контекстом содержит более оперативную информацию, которая должна быть легко доступна подсистеме планирования процессов. Контекст процесса содержит менее актуальную информацию и используется операционной системой только после того, как принято решение о возобновлении прерванного процесса.

Очереди процессов представляют собой дескрипторы отдельных процессов, объединенные в списки. Таким образом, каждый дескриптор, кроме всего прочего, содержит по крайней мере один указатель на другой дескриптор, соседствующий с ним в очереди. Такая организация очередей позволяет легко их переупорядочивать, включать и исключать процессы, переводить процессы из одного состояния в другое.

Программный код только тогда начнет выполняться, когда для него операционной системой будет создан процесс. Создать процесс - это значит:

1. создать информационные структуры, описывающие данный процесс, то есть его дескриптор и контекст;
2. включить дескриптор нового процесса в очередь готовых процессов;
3. загрузить кодовый сегмент процесса в оперативную память или в область свопинга.

Структура сетевой ОС

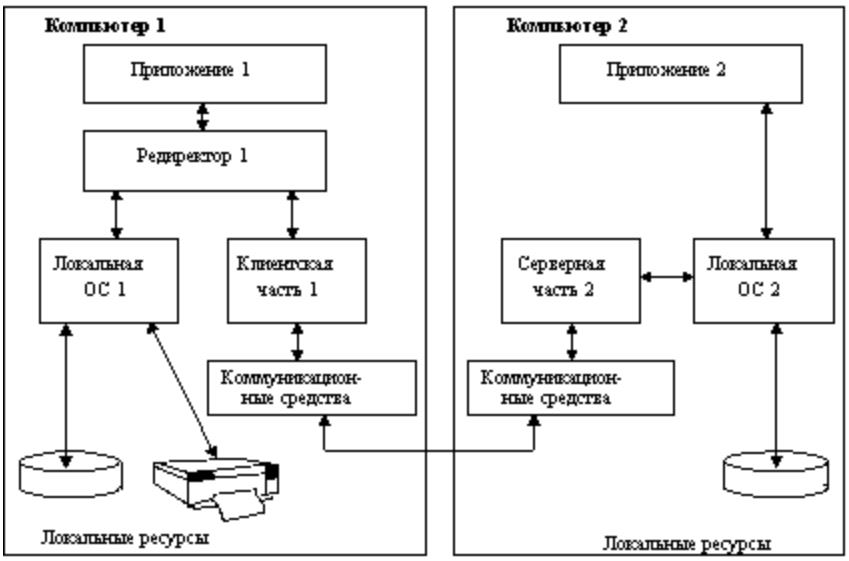
под сетевой операционной системой в широком смысле понимается совокупность операционных систем отдельных компьютеров, взаимодействующих с целью обмена сообщениями и разделения ресурсов по единым правилам - протоколам. В узком смысле сетевая ОС - это операционная система отдельного компьютера, обеспечивающая ему возможность работать в сети.

Рис. 1.1. Структура сетевой ОС

В сетевой операционной системе отдельной машины можно выделить несколько частей (рисунок 1.1):

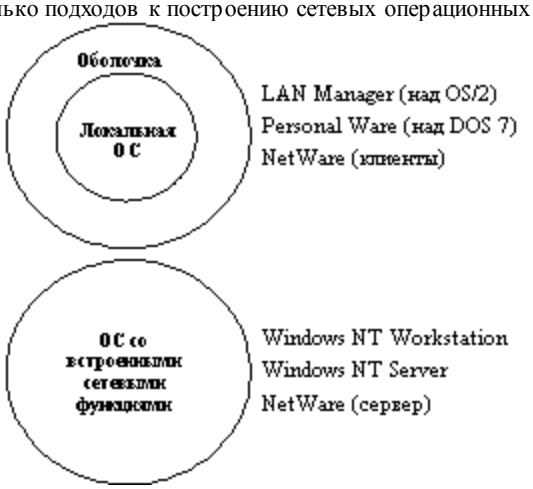
- Средства управления локальными ресурсами компьютера: функции распределения оперативной памяти между процессами, планирования и диспетчеризации процессов, управления процессорами в мультипроцессорных машинах, управления периферийными устройствами и другие функции управления ресурсами локальных ОС.
- Средства предоставления собственных ресурсов и услуг в общее пользование - серверная часть ОС (сервер). Эти средства обеспечивают, например, блокировку файлов и записей, что необходимо для их совместного использования; ведение справочников имен сетевых ресурсов; обработку запросов удаленного доступа к собственной файловой системе и базе данных; управление очередями запросов удаленных пользователей к своим периферийным устройствам.

- Средства запроса доступа к удаленным ресурсам и услугам и их использования - клиентская часть ОС (редиректор). Эта часть выполняет распознавание и перенаправление в сеть запросов к удаленным ресурсам от приложений и пользователей, при этом запрос поступает от приложения в локальной форме, а передается в сеть в другой форме, соответствующей требованиям сервера. Клиентская часть также осуществляет прием ответов от серверов и преобразование их в локальный формат, так что для приложения выполнение локальных и удаленных запросов неразличимо.
- Коммуникационные средства ОС, с помощью которых происходит обмен сообщениями в сети. Эта часть обеспечивает адресацию и буферизацию сообщений, выбор маршрута передачи сообщения по сети, надежность передачи и т.п., то есть является средством транспортировки сообщений.
- На рисунке 1.2 показано взаимодействие сетевых компонентов. Здесь компьютер 1 выполняет роль "чистого" клиента, а компьютер 2 - роль "чистого" сервера, соответственно на первой машине отсутствует серверная часть, а на второй - клиентская. На рисунке отдельно показан компонент клиентской части - редиректор. Именно редиректор перехватывает все запросы, поступающие от приложений, и анализирует их. Если выдан запрос к ресурсу данного компьютера, то он переадресовывается соответствующей подсистеме локальной ОС, если же это запрос к удаленному ресурсу, то он переправляется в сеть. При этом клиентская часть преобразует запрос из локальной формы в сетевой формат и передает его транспортной подсистеме, которая отвечает за доставку сообщений указанному серверу. Серверная часть операционной системы компьютера 2 принимает запрос, преобразует его и передает для выполнения своей локальной ОС. После того, как результат получен, сервер обращается к транспортной подсистеме и направляет ответ клиенту, выдавшему запрос. Клиентская часть преобразует результат в соответствующий формат и адресует его тому приложению, которое выдало запрос.



- *Рис. 1.2. взаимодействие компонентов операционной системы при взаимодействии компьютеров*

- На практике сложилось несколько подходов к построению сетевых операционных систем (рисунок 1.3).



- *Рис. 1.3. Варианты построения сетевых ОС*

Алгоритмы планирования процессов: FSCS, RR, SJF. Гарантированное и приоритетное планирование

First-Come, First-Served (FCFS)

Простейшим алгоритмом планирования является алгоритм, который принято обозначать аббревиатурой FCFS по первым буквам его английского названия – First-Come, First-Served (первым пришел, первым обслужен). Представим себе, что процессы, находящиеся в состоянии готовность, выстроены в очередь. Когда процесс переходит в состояние готовность, он, а точнее, ссылка на его PCB помещается в конец этой очереди. Выбор нового процесса для исполнения осуществляется из начала очереди с удалением оттуда ссылки на его PCB. Очередь подобного типа имеет в программировании специальное наименование – FIFO[1)], сокращение от First In, First Out (первым вошел, первым вышел).

Такой алгоритм выбора процесса осуществляет невытесняющее планирование. Процесс, получивший в свое распоряжение процессор, занимает его до истечения текущего CPU burst. После этого для выполнения выбирается новый процесс из начала очереди.

Таблица 3.1.

Процесс	p_0	p_1	p_2
Продолжительность очередного CPU burst	13	4	1

Преимуществом алгоритма FCFS является легкость его реализации, но в то же время он имеет и много недостатков. Рассмотрим следующий пример. Пусть в состоянии готовность находятся три процесса p_0, p_1 и p_2, для которых известны времена их очередных CPU burst. Эти времена приведены в таблице 3.1. в некоторых условных единицах. Для простоты будем полагать, что вся деятельность процессов ограничивается использованием только одного промежутка CPU burst, что процессы не совершают операций ввода-вывода и что время переключения контекста так мало, что им можно пренебречь.

Если процессы расположены в очереди процессов, готовых к исполнению, в порядке p_0, p_1, p_2, то картина их выполнения выглядит так, как показано на рисунке 3.2. Первым для выполнения выбирается процесс p_0, который получает процессор на все время своего CPU burst, т. е. на 13 единиц времени. После его окончания в состояние исполнение переводится процесс p_1, он занимает процессор на 4 единицы времени. И, наконец, возможность работать получает процесс p_2. Время ожидания для процесса p_0 составляет 0 единиц времени, для процесса p_1 – 13 единиц, для процесса p_2 – 13 + 4 = 17 единиц. Таким образом, среднее время ожидания в этом случае – (0 + 13 + 17)/3 = 10 единиц времени. Полное время выполнения для процесса p_0 составляет 13 единиц времени, для процесса p_1 – 13 + 4 = 17 единиц, для процесса p_2 – 13 + 4 + 1 = 18 единиц. Среднее полное время выполнения оказывается равным (13 + 17 + 18)/3 = 16 единицам времени.

Рис. 3.2. Выполнение процессов при порядке p0,p1,p2

Если те же самые процессы расположены в порядке p_2, p_1, p_0, то картина их выполнения будет соответствовать рисунку 3.3. Время ожидания для процесса p_0 равняется 5 единицам времени, для процесса p_1 – 1 единице, для процесса p_2 – 0 единиц. Среднее время ожидания составит (5 + 1 + 0)/3 = 2 единицы времени. Это в 5 (!) раз меньше, чем в предыдущем случае. Полное время выполнения для процесса p_0 получается равным 18 единицам времени, для процесса p_1 – 5 единицам, для процесса p_2 – 1 единице. Среднее полное время выполнения составляет (18 + 5 + 1)/3 = 8 единиц времени, что почти в 2 раза меньше, чем при первой расстановке процессов.

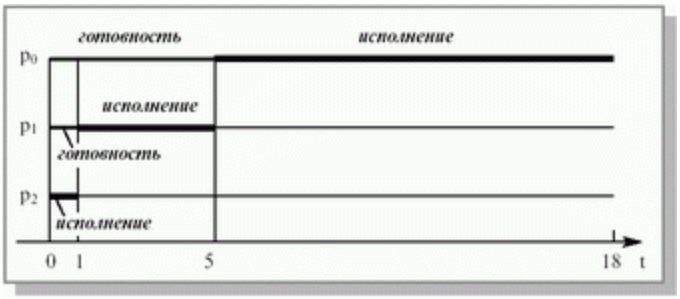


Рис. 3.3. Выполнение процессов при порядке p2, p1, p0

Как мы видим, среднее время ожидания и среднее полное время выполнения для этого алгоритма существенно зависят от порядка расположения процессов в очереди. Если у нас есть процесс с длительным CPU burst, то короткие процессы, перешедшие в состояние готовность после длительного процесса, будут очень долго ждать начала выполнения. Поэтому алгоритм FCFS практически неприменим для систем разделения времени – слишком большим получается среднее время отклика в интерактивных процессах.

Round Robin (RR)

Модификацией алгоритма FCFS является алгоритм, получивший название Round Robin (Round Robin – это вид детской карусели в США) или сокращенно RR. По сути дела, это тот же самый алгоритм, только реализованный в режиме вытесняющего планирования. Можно представить себе все множество готовых процессов организованным циклически – процессы сидят на карусели. Карусель вращается так, что каждый процесс находится около процессора небольшой фиксированный квант времени, обычно 10 – 100 миллисекунд (см. рис. 3.4.). Пока процесс находится рядом с процессором, он получает процессор в свое распоряжение и может исполняться.

Рис. 3.4. Процессы на карусели

Реализуется такой алгоритм так же, как и предыдущий, с помощью организации процессов, находящихся в состоянии готовность, в очередь FIFO. Планировщик выбирает для очередного исполнения процесс, расположенный в начале очереди, и устанавливает таймер для генерации прерывания по истечении определенного кванта времени. При выполнении процесса возможны два варианта.

- Время непрерывного использования процессора, необходимое процессу (остаток текущего CPU burst), меньше или равно продолжительности кванта времени. Тогда процесс по своей воле освобождает процессор до истечения кванта времени, на исполнение поступает новый процесс из начала очереди, и таймер начинает отсчет кванта заново.
- Продолжительность остатка текущего CPU burst процесса больше, чем квант времени. Тогда по истечении этого кванта процесс прерывается таймером и помещается в конец очереди процессов, готовых к исполнению, а процессор выделяется для использования процессу, находящемуся в ее начале.

Рассмотрим предыдущий пример с порядком процессов p_0, p_1, p_2 и величиной кванта времени равной 4. Выполнение этих процессов иллюстрируется таблицей 3.2. Обозначение "И" используется в ней для процесса, находящегося в состоянии исполнение, обозначение "Г" – для процессов в состоянии готовность, пустые ячейки соответствуют завершившимся процессам. Состояния процессов показаны на протяжении соответствующей единицы времени, т. е. колонка с номером 1 соответствует промежутку времени от 0 до 1.

Таблица 3.2.

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
p_0	И	И	И	И	Г	Г	Г	Г	Г	И	И	И	И	И	И	И	И	И
p_1	Г	Г	Г	Г	И	И	И	И										
p_2	Г	Г	Г	Г	Г	Г	Г	Г	И									

Первым для исполнения выбирается процесс p_0. Продолжительность его CPU burst больше, чем величина кванта времени, и поэтому процесс исполняется до истечения кванта, т. е. в течение 4 единиц времени. После этого он помещается в конец очереди готовых к исполнению процессов, которая принимает вид p_1, p_2, p_0. Следующим начинает выполняться процесс p_1. Время его исполнения совпадает с величиной выделенного кванта, поэтому процесс работает до своего завершения. Теперь очередь процессов в состоянии готовность состоит из двух процессов, p_2 и p_0. Процессор выделяется процессу p_2. Он завершается до истечения отпущенного ему процессорного времени, и очередные кванты отмеряются процессу p_0 – единственному не закончившему к этому моменту свою работу. Время ожидания для процесса p_0 (количество символов "Г" в соответствующей строке) составляет 5 единиц времени, для процесса p_1 – 4 единицы времени, для процесса p_2 – 8 единиц времени. Таким образом, среднее время ожидания для этого алгоритма получается равным (5 + 4 + 8)/3 = 5,6(6) единицы времени. Полное время выполнения для процесса p_0 (количество непустых столбцов в соответствующей строке) составляет 18 единиц времени, для процесса p_1 – 8 единиц, для процесса p_2 – 9 единиц. Среднее полное время выполнения оказывается равным (18 + 8 + 9)/3 = 11,6(6) единицы времени.

Легко увидеть, что среднее время ожидания и среднее полное время выполнения для обратного порядка процессов не отличаются от соответствующих времен для алгоритма FCFS и составляют 2 и 8 единиц времени соответственно.

На производительность алгоритма RR сильно влияет величина кванта времени. Рассмотрим тот же самый пример с порядком процессов p_0, p_1, p_2 для величины кванта времени, равной 1 (см. табл. 3.3.). Время ожидания для процесса p_0 составит 5 единиц времени, для процесса p_1 – тоже 5 единиц, для процесса p_2 – 2 единицы. В этом случае среднее время ожидания получается равным (5 + 5 + 2)/3 = 4 единицам времени. Среднее полное время исполнения составит (18 + 9 + 3)/3 = 10 единиц времени.

Таблица 3.3.

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
p_0	И	Г	Г	И	Г	И	Г	И	Г	И	И	И	И	И	И	И	И	И
p_1	Г	И	Г	Г	И	Г	И	Г	И									
p_2	Г	Г	И															

При очень больших величинах кванта времени, когда каждый процесс успевает завершить свой CPU burst до возникновения прерывания по времени, алгоритм RR вырождается в алгоритм FCFS. При очень малых величинах создается иллюзия того, что каждый из n процессов работает на собственном виртуальном процессоре с производительностью ~ 1/n от производительности реального процессора. Правда, это справедливо лишь при теоретическом анализе при условии пренебрежения временами переключения контекста процессов. В реальных условиях при слишком малой величине кванта времени и, соответственно, слишком частом переключении контекста накладные расходы на переключение резко снижают производительность системы.

Shortest-Job-First (SJF)

При рассмотрении алгоритмов FCFS и RR мы видели, насколько существенным для них является порядок расположения процессов в очереди процессов, готовых к исполнению. Если короткие задачи расположены в очереди ближе к ее началу, то общая производительность этих алгоритмов значительно возрастает. Если бы мы знали время следующих CPU burst для процессов, находящихся в состоянии готовность, то могли бы выбрать для исполнения не процесс из начала очереди, а процесс с минимальной длительностью CPU burst. Если же таких процессов два или больше, то для выбора одного из них можно использовать уже известный

нам алгоритм FCFS. Квантование времени при этом не применяется. Описанный алгоритм получил название "кратчайшая работа первой" или Shortest Job First (SJF).

SJF-алгоритм краткосрочного планирования может быть как вытесняющим, так и невытесняющим. При невытесняющем SJF-планировании процессор предоставляется избранному процессу на все необходимое ему время, независимо от событий, происходящих в вычислительной системе. При вытесняющем SJF-планировании учитывается появление новых процессов в очереди готовых к исполнению (из числа вновь родившихся или разблокированных) во время работы выбранного процесса. Если CPU burst нового процесса меньше, чем остаток CPU burst у исполняющегося, то исполняющийся процесс вытесняется новым.

Рассмотрим пример работы невытесняющего алгоритма SJF. Пусть в состоянии готовность находятся четыре процесса, p_0, p_1, p_2 и p_3, для которых известны времена их очередных CPU burst. Эти времена приведены в таблице 3.4. Как и прежде, будем полагать, что вся деятельность процессов ограничивается использованием только одного промежутка CPU burst, что процессы не совершают операций ввода-вывода и что временем переключения контекста можно пренебречь.

Таблица 3.4.

Процесс	p_0	p_1	p_2	p_3
Продолжительность очередного CPU burst	5	3	7	1

При использовании невытесняющего алгоритма SJF первым для исполнения будет выбран процесс p_3, имеющий наименьшее значение продолжительности очередного CPU burst. После его завершения для исполнения выбирается процесс p_1, затем p_0 и, наконец, p_2.

Гарантированное планирование

При интерактивной работе N пользователей в вычислительной системе можно применить алгоритм планирования, который гарантирует, что каждый из пользователей будет иметь в своем распоряжении ~1/N часть процессорного времени. Пронумеруем всех пользователей от 1 до N. Для каждого пользователя с номером i введем две величины: T_i – время нахождения пользователя в системе или, другими словами, длительность сеанса его общения с машиной и τ_i – суммарное процессорное время уже выделенное всем его процессам в течение сеанса. Справедливым для пользователя было бы получение T_i/N процессорного времени. Если

$$\tau_i << T_i/N$$

то i-й пользователь несправедливо обделен процессорным временем. Если же

$$\tau_i >> T_i/N$$

то система явно благоволит к пользователю с номером i. Вычислим для процессов каждого пользователя значение коэффициента справедливости

$$\tau_i N/T_i$$

и будем предоставлять очередной квант времени готовому процессу с наименьшей величиной этого отношения. Предложенный алгоритм называют алгоритмом гарантированного планирования. К недостаткам этого алгоритма можно отнести невозможность предугадать поведение пользователей. Если некоторый пользователь отправится на пару часов пообедать и поспать, не прерывая сеанса работы, то по возвращении его процессы будут получать неоправданно много процессорного времени.

Приоритетное планирование

Алгоритмы SJF и гарантированного планирования представляют собой частные случаи приоритетного планирования. При приоритетном планировании каждому процессу присваивается определенное числовое значение – приоритет, в соответствии с которым ему выделяется процессор. Процессы с одинаковыми

приоритетами планируются в порядке FCFS. Для алгоритма SJF в качестве такого приоритета выступает оценка продолжительности следующего CPU burst. Чем меньше значение этой оценки, тем более высокий приоритет имеет процесс. Для алгоритма гарантированного планирования приоритетом служит вычисленный коэффициент справедливости. Чем он меньше, тем больше у процесса приоритет.

Механизмы синхронизации: семафор, мьютекс, монитор, критическая секция

Мьютексы

Мьютекс (mutex) – средство синхронизации, часто называемое также взаимным исключением. Он позволяет только одному процессу в данное время владеть им. Если продолжать аналогии, то этот объект можно сравнить с эстафетной палочкой. Обычно мьютекс представляет собой целочисленную переменную, все функции для работы с которой (создать мьютекс, удалить, захватить, отдать и проч.) определены как критические секции.

Критические секции

Критические секции (critical section) подобны взаимным исключениям, однако между ними существуют два главных отличия:

- взаимные исключения могут быть совместно использованы в различных процессах;
- если критическая секция принадлежит другому процессу, ожидающий процесс блокируется вплоть до освобождения критической секции; в отличие от этого, взаимное исключение разрешает продолжение по истечении тайм-аута.

Критические секции и взаимные исключения очень схожи. На первый взгляд, выигрыш от использования критической секции вместо взаимного исключения не очевиден. Критические секции, однако, более эффективны, чем взаимные исключения, так как используют меньше системных ресурсов. Взаимные исключения могут быть установлены на определенный интервал времени, по истечении которого выполнение продолжается; критическая секция всегда ждет столько, сколько потребуется.

Семафоры

Семафоры (semaphore) также подобны взаимным исключениям. Разница между ними в том, что семафор может управлять количеством процессов, которые имеют к нему доступ. Семафор устанавливается на предельное количество процессов, которым доступ разрешен. Когда это число достигнуто, последующие процессы будут приостановлены, пока один или более процессов не отсоединятся от семафора и не освободят доступ.

В качестве примера использования семафора рассмотрим случай, когда каждый из группы процессов работает с фрагментом совместно используемого пула памяти. Так как совместно используемая память допускает обращение к ней только определенного числа процессов, все прочие должны быть блокированы вплоть до момента, когда один или несколько пользователей пула откажутся от его совместного использования.

Мониторы представляют собой тип данных, который может быть с успехом внедрен в объектно-ориентированные языки программирования. Монитор обладает собственными переменными, определяющими его состояние. Значения этих переменных извне могут быть изменены только с помощью вызова функций-методов, принадлежащих монитору. В свою очередь, эти функции-методы могут использовать в работе только данные, находящиеся внутри монитора, и свои параметры. На абстрактном уровне можно описать структуру монитора следующим образом:

```
monitor monitor_name {
   описание внутренних переменных ;

   void m1(...){...
   }
   void m2(...){...
   }
   ...
   void mn(...){...
   }

   {
      блок инициализации
      внутренних переменных;
   }
}
```

Распределенные файловые системы.

Две главные цели.

Сетевая прозрачность.
Самая важная цель - обеспечить те же самые возможности доступа к файлам, распределенным по сети ЭВМ, которые обеспечиваются в системах разделения времени на централизованных ЭВМ.

Высокая доступность.
Другая важная цель - обеспечение высокой доступности. Ошибки систем или осуществление операций копирования и сопровождения не должны приводить к недоступности файлов.

Понятие файлового сервиса и файлового сервера.

Файловый сервис - это то, что файловая система предоставляет своим клиентам, т.е. интерфейс с файловой системой.
Файловый сервер - это процесс, который реализует файловый сервис.

Пользователь не должен знать, сколько файловых серверов имеется и где они расположены.

Так, как файловый сервер обычно является обычным пользовательским процессом, то в системе могут быть различные файловые серверы, предоставляющие различный сервис (например, UNIX файл сервис и MS-DOS файл сервис).

Распределенная система обычно имеет два существенно отличающихся компонента - непосредственно файловый сервис и сервис директорий.

5.1.1 Интерфейс файлового сервера

Для любой файловой системы первый фундаментальный вопрос - что такое файл. Во многих системах, таких как UNIX и MS-DOS, файл - не интерпретируемая последовательность байтов. На многих централизованных ЭВМ (IBM/370) файл представляется последовательность записей, которую можно специфицировать ее номером или содержимым некоторого поля (ключом). Так, как большинство распределенных систем базируются на использовании среды UNIX и MS-DOS, то они используют первый вариант понятия файла.

Файл может иметь *атрибуты* (информация о файле, не являющаяся его частью). Типичные атрибуты - владелец, размер, дата создания и права доступа.

Важный аспект файловой модели - могут ли файлы *модифицироваться* после создания. Обычно могут, но есть системы с неизменяемыми файлами. Такие файлы освобождают разработчиков от многих проблем при кэшировании и размножении.

Защита обеспечивается теми же механизмами, что и в однопроцессорных ЭВМ - мандатами и списками прав доступа. Мандат - своего рода билет, выданный пользователю для каждого файла с указанием прав доступа. Список прав доступа задает для каждого файла список пользователей с их правами. Простейшая схема с правами доступа - UNIX схема, в которой различают три типа доступа (чтение, запись, выполнение), и три типа пользователей (владелец, члены его группы, и прочие).

Файловый сервис может базироваться на одной из двух моделей - модели *загрузки/разгрузки* и модели *удаленного доступа*. В первом случае файл передается между клиентом (памятью или дисками) и сервером целиком, а во втором файл сервис обеспечивает множество операций (открытие, закрытие, чтение и запись части файла, сдвиг указателя, проверку и изменение атрибутов, и т.п.). Первый подход требует большого объема памяти у клиента, затрат на перемещение ненужных частей файла. При втором подходе файловая система функционирует на сервере, клиент может не иметь дисков и большого объема памяти.

5.1.2 Интерфейс сервера директорий

Обеспечивает операции создания и удаления директорий, именования и переименования файлов, перемещение файлов из одной директории в другую.

Определяет алфавит и синтаксис имен. Для спецификации ***типа*** информации в файле используется часть имени (расширение) либо явный атрибут.

Все распределенные системы позволяют директориям содержать поддиректории - такая файловая система называется ***иерархической***. Некоторые системы позволяют создавать указатели или ссылки на произвольные директории, которые можно помещать в директорию. При этом можно строить не только деревья, но и произвольные графы (разница между ними очень важна для распределенных систем, поскольку в случае графа удаление связи может привести к появлению недостижимых поддеревьев. Обнаруживать такие поддеревья в распределенных системах очень трудно).

Ключевое решение при конструировании распределенной файловой системы - должны или не должны машины (или процессы) одинаково видеть иерархию директорий. Тесно связано с этим решением наличие единой корневой директории (можно иметь такую директорию с поддиректориями для каждого сервера).

Прозрачность именования.
Две формы прозрачности именования различают - прозрачность расположения (/server/d1/f1) и прозрачность миграции (когда изменение расположения файла не требует изменения имени).

Имеются три подхода к именованию:

- машина + путь;
- монтирование удаленных файловых систем в локальную иерархию файлов;
- единственное пространство имен, которое выглядит одинаково на всех машинах.

Последний подход необходим для достижения того, чтобы распределенная система выглядела как единый компьютер, однако он сложен и требует тщательного проектирования.

Двухуровневое именование.
Большинство систем используют ту или иную форму двухуровневого именования. Файлы (и другие объекты) имеют символические имена для пользователей, но могут также иметь внутренние двоичные имена для использования самой системой. Например, в операции открыть файл пользователь задает символическое имя, а в ответ получает двоичное имя, которое и использует во всех других операциях с данным файлом.

Способы формирования двоичных имен различаются в разных системах:

- если имеется несколько не ссылающихся друг на друга серверов (директории не содержат ссылок на объекты других серверов), то двоичное имя может быть то же самое, что и в ОС UNIX;
- имя может указывать на сервер и файл;
- в качестве двоичных имен при просмотре символьных имен возвращаются мандаты, содержащие помимо прав доступа либо физический номер машины с сервером, либо сетевой адрес сервера, а также номер файла.

В ответ на символьное имя некоторые системы могут возвращать несколько двоичных имен (для файла и его дублей), что позволяет повысить надежность работы с файлом.

Есть ли *разница между клиентами и серверами*? Имеются системы, где все машины имеют одно и то же ПО и любая машина может предоставлять файловый сервис. Есть системы, в которых серверы являются обычными пользовательскими процессами и могут быть сконфигурированы для работы на одной машине с клиентами или на разных. Есть системы, в которых клиенты и серверы являются фундаментально разными машинами с точки зрения аппаратуры или ПО (требуют различных ОС, например).

Второй вопрос - должны ли быть файловый сервер и сервер директорий отдельными серверами или быть объединенными в один сервер. Разделение позволяет иметь разные серверы директорий (UNIX, MS-DOS) и один файловый сервер. Объединение позволяет сократить коммуникационные издержки.

В случае разделения серверов и при наличии разных серверов директорий для различных поддеревьев возникает следующая проблема. Если первый вызванный сервер будет поочередно обращаться ко всем следующим, то возникают большие коммуникационные расходы. Если же первый сервер передает остаток имени второму, а тот третьему, и т.д., то это не позволяет использовать RPC.

Возможный выход - использование кэша подсказок. Однако в этом случае при получении от сервера директорий устаревшего двоичного имени клиент должен быть готов получить отказ от файлового сервера и повторно обращаться к серверу директорий (клиент может не быть конечным пользователем!).

Последний важный вопрос - должны ли серверы хранить информацию о клиентах.

Тупики: условия возникновения и основные направления борьбы

Тупики (взаимные блокировки, дедлоки (deadlocks), клинчи (clinch)) имеют место, когда процесс ожидает ресурс, который в данный момент принадлежит другому процессу.

Рассмотрим пример: Процесс 1 захватывает ресурс А и, для того чтобы продолжать работу, ждет возможности захватить ресурс Б. В тоже время Процесс 2 захватывает ресурс Б и ждет возможности захватить ресурс А. Развитие этого сценария заблокирует оба процесса – ни один из них не будет исполняться.

В зависимости от соотношения скоростей процессов, они могут либо совершенно независимо использовать разделяемые ресурсы, либо образовывать очереди к разделяемым ресурсам, либо взаимно блокировать друг друга. Тупиковые ситуации надо отличать от простых очередей, хотя и те и другие возникают при совместном использовании ресурсов и внешне выглядят похоже: процесс приостанавливается и ждет освобождения ресурса. Однако очередь - это нормальное явление, неотъемлемый признак высокого коэффициента использования ресурсов при случайном поступлении запросов. Она возникает тогда, когда ресурс недоступен в данный момент, но через некоторое время он освобождается, и процесс продолжает свое выполнение. Тупик же, что видно из его названия, является в некотором роде неразрешимой ситуацией.

Проблема тупиков включает в себя следующие задачи:

- предотвращение тупиков,
- распознавание тупиков,
- восстановление системы после тупиков.

Тупики могут быть предотвращены на стадии написания программ, то есть программы должны быть написаны таким образом, чтобы тупик не мог возникнуть ни при каком соотношении взаимных скоростей процессов. Так, если бы в предыдущем примере Процесс 1 и Процесс 2 запрашивали ресурсы в одинаковой последовательности, то тупик был бы в принципе невозможен. Второй подход к предотвращению тупиков называется динамическим и заключается в использовании определенных правил при назначении ресурсов процессам, например, ресурсы могут выделяться в определенной последовательности, общей для всех процессов.

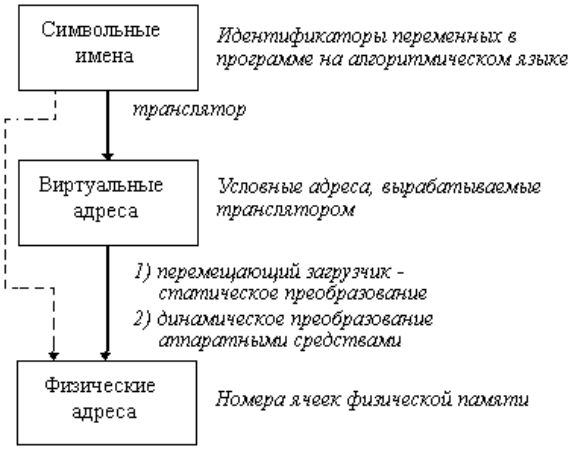
управление памятью, методы распределения памяти

Распределению подлежит вся оперативная память, не занятая операционной системой. Обычно ОС располагается в самых младших адресах, однако может занимать и самые старшие адреса. Функциями ОС по управлению памятью являются: отслеживание свободной и занятой памяти, выделение памяти процессам и освобождение памяти при завершении процессов, вытеснение процессов из оперативной памяти на диск, когда размеры основной памяти не достаточны для размещения в ней всех процессов, и возвращение их в оперативную память, когда в ней освобождается место, а также настройка адресов программы на конкретную область физической памяти.

Для идентификации переменных и команд используются символьные имена (метки), виртуальные адреса и физические адреса (рисунок 2.7).

Символьные имена присваивает пользователь при написании программы на алгоритмическом языке или ассемблере.

Виртуальные адреса вырабатывает транслятор, переводящий программу на машинный язык.



Перемещающий загрузчик на основании имеющихся у него исходных данных о начальном адресе физической памяти, в которую предстоит загружать программу, и информации, предоставленной транслятором об адресно-зависимых константах программы, выполняет загрузку программы, совмещая ее с заменой виртуальных адресов физическими.

Второй способ заключается в том, что программа загружается в память в неизмененном виде в виртуальных адресах, при этом операционная система фиксирует смещение действительного расположения программного кода относительно виртуального адресного пространства. Во время выполнения программы при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический.

Все методы управления памятью могут быть разделены на два класса: методы, которые используют перемещение процессов между оперативной памятью и диском, и методы, которые не делают этого (рисунок 2.8). Начнем с последнего, более простого класса методов.

Распределение памяти фиксированными разделами

Самым простым способом управления оперативной памятью является разделение ее на несколько разделов фиксированной величины. Это может быть выполнено вручную оператором во время старта системы или во время ее генерации. Очередная задача, поступившая на выполнение, помещается либо в общую очередь (рисунок 2.9,а), либо в очередь к некоторому разделу (рисунок 2.9,б).

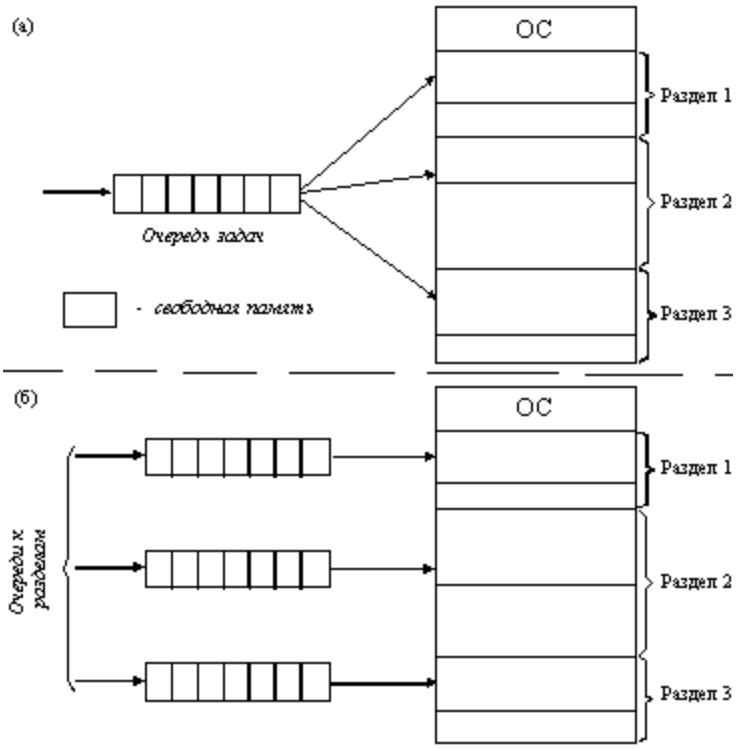


Рис. 2.9. Распределение памяти фиксированными разделами:
а - с общей очередью; б - с отдельными очередями

Подсистема управления памятью в этом случае выполняет следующие задачи:

- сравнивая размер программы, поступившей на выполнение, и свободных разделов, выбирает подходящий раздел,
- осуществляет загрузку программы и настройку адресов.

При очевидном преимуществе - простоте реализации - данный метод имеет существенный недостаток - жесткость. Так как в каждом разделе может выполняться только одна программа, то уровень мультипрограммирования заранее ограничен числом разделов не зависимо от того, какой размер имеют программы. Даже если программа имеет небольшой объем, она будет занимать весь раздел, что приводит к неэффективному использованию памяти. С другой стороны, даже если объем оперативной памяти машины позволяет выполнить некоторую программу, разбиение памяти на разделы не позволяет сделать этого.

Распределение памяти разделами переменной величины

В этом случае память машины не делится заранее на разделы. Сначала вся память свободна. Каждой вновь поступающей задаче выделяется необходимая ей память. Если достаточный объем памяти отсутствует, то задача не принимается на выполнение и стоит в очереди. После завершения задачи память освобождается, и на это место может быть загружена другая задача. Таким образом, в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера. На рисунке 2.10 показано состояние памяти в различные моменты времени при использовании динамического распределения. Так в момент t0 в памяти находится только ОС, а к моменту t1 память разделена между 5 задачами, причем задача П4, завершаясь, покидает память. На освободившееся после задачи П4 место загружается задача П6, поступившая в момент t3.

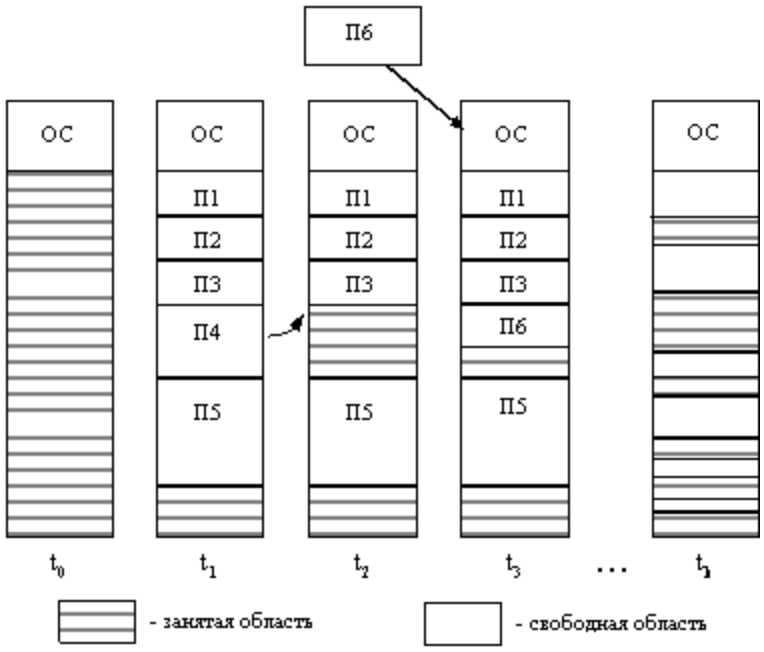


Рис. 2.10. Распределение памяти динамическими разделами

Задачами операционной системы при реализации данного метода управления памятью является:

- ведение таблиц свободных и занятых областей, в которых указываются начальные адреса и размеры участков памяти,
- при поступлении новой задачи - анализ запроса, просмотр таблицы свободных областей и выбор раздела, размер которого достаточен для размещения поступившей задачи,
- загрузка задачи в выделенный ей раздел и корректировка таблиц свободных и занятых областей,
- после завершения задачи корректировка таблиц свободных и занятых областей.

Программный код не перемещается во время выполнения, то есть может быть проведена единовременная настройка адресов посредством использования перемещающего загрузчика.

Выбор раздела для вновь поступившей задачи может осуществляться по разным правилам, таким, например, как "первый попавшийся раздел достаточного размера", или "раздел, имеющий наименьший достаточный размер", или "раздел, имеющий наибольший достаточный размер". Все эти правила имеют свои преимущества и недостатки.

По сравнению с методом распределения памяти фиксированными разделами данный метод обладает гораздо большей гибкостью, но ему присущ очень серьезный недостаток - *фрагментация памяти*. Фрагментация - это наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов). Настолько маленького, что ни одна из вновь поступающих программ не может поместиться ни в одном из участков, хотя суммарный объем фрагментов может составить значительную величину, намного превышающую требуемый объем памяти.

Перемещаемые разделы

Одним из методов борьбы с фрагментацией является перемещение всех занятых участков в сторону старших либо в сторону младших адресов, так, чтобы вся свободная память образовывала единую свободную область (рисунок 2.11). В дополнение к функциям, которые выполняет ОС при распределении памяти переменными разделами, в данном случае она должна еще время от времени копировать содержимое разделов из одного места памяти в другое, корректируя таблицы свободных и занятых областей. Эта процедура называется "сжатием". Сжатие может выполняться либо при каждом завершении задачи, либо только тогда, когда для вновь поступившей задачи нет свободного раздела достаточного размера. В первом случае требуется меньше вычислительной работы при корректировке таблиц, а во втором - реже выполняется процедура сжатия. Так как программы перемещаются по оперативной памяти в ходе своего выполнения, то преобразование адресов из виртуальной формы в физическую должно выполняться динамическим способом.

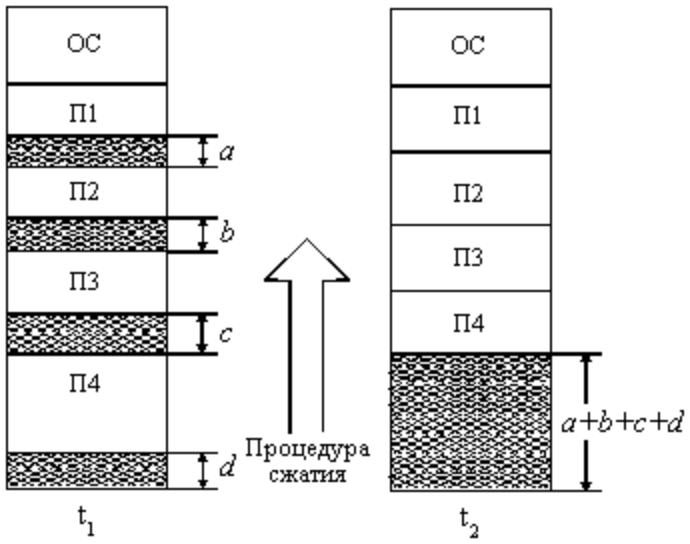


Рис. 2.11. Распределение памяти перемещаемыми разделами

Хотя процедура сжатия и приводит к более эффективному использованию памяти, она может потребовать значительного времени, что часто перевешивает преимущества данного метода.

Каналы

Своппинг

Разновидностью виртуальной памяти является свопинг.

На рисунке 2.16 показан график зависимости коэффициента загрузки процессора в зависимости от числа одновременно выполняемых процессов и доли времени, проводимого этими процессами в состоянии ожидания ввода-вывода.

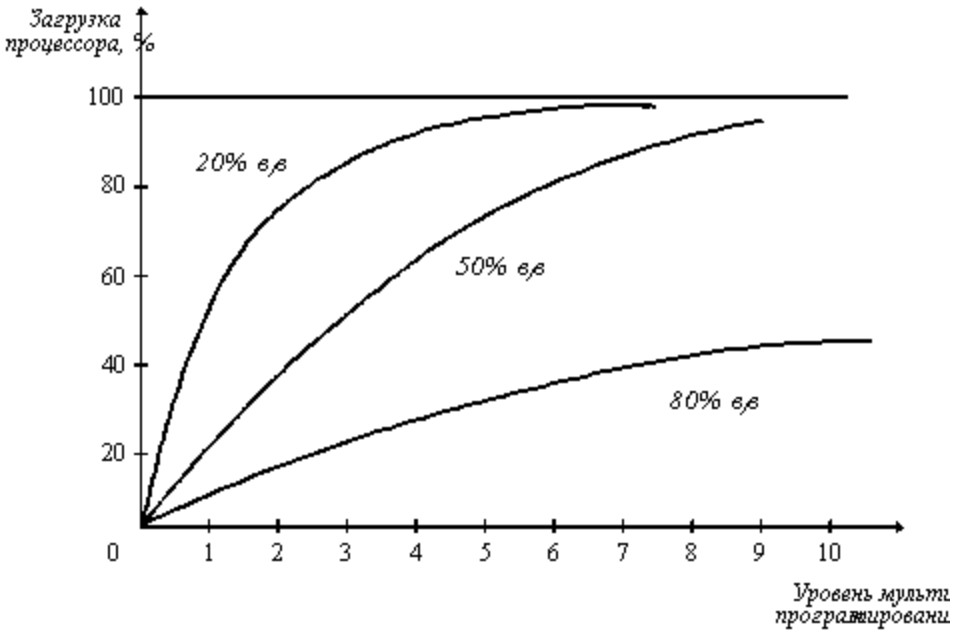


Рис. 2.16. Зависимость загрузки процессора от числа задач и интенсивности ввода-вывода

Из рисунка видно, что для загрузки процессора на 90% достаточно всего трех счетных задач. Однако для того, чтобы обеспечить такую же загрузку интерактивными задачами, выполняющими интенсивный ввод-вывод, потребуются десятки таких задач. Необходимым условием для выполнения задачи является загрузка ее в оперативную память, объем которой ограничен. В этих условиях был предложен метод организации вычислительного процесса, называемый свопингом. В соответствии с этим методом некоторые процессы (обычно находящиеся в состоянии ожидания) временно выгружаются на диск. Планировщик операционной системы не исключает их из своего рассмотрения, и при наступлении условий активизации некоторого процесса, находящегося в области свопинга на диске, этот процесс перемещается в оперативную память. Если свободного места в оперативной памяти не хватает, то выгружается другой процесс.

При свопинге, в отличие от рассмотренных ранее методов реализации виртуальной памяти, процесс перемещается между памятью и диском целиком, то есть в течение некоторого времени процесс может полностью отсутствовать в оперативной памяти. Существуют различные алгоритмы выбора процессов на загрузку и выгрузку, а также различные способы выделения оперативной и дисковой памяти загружаемому процессу.

Файловая система. Логическая организация файлов
Файловая система. Физическая организация и запись файлов
Файловая система. Права доступа к файлу. Кеширование диска

Файловая система - это часть операционной системы, назначение которой состоит в том, чтобы организовать эффективную работу с данными, хранящимися во внешней памяти, и обеспечить пользователю удобный интерфейс при работе с такими данными. Организовать хранение информации на магнитном диске непросто. Это требует, например, хорошего знания устройства контроллера диска, особенностей работы с его регистрами. Непосредственное взаимодействие с диском - прерогатива компонента системы ввода-вывода ОС, называемого драйвером диска. Для того чтобы избавить пользователя компьютера от сложностей взаимодействия с аппаратурой, была придумана ясная абстрактная модель файловой системы. Операции записи или чтения файла концептуально проще, чем низкоуровневые операции работы с устройствами.

Основная идея использования внешней памяти состоит в следующем. ОС делит память на блоки фиксированного размера, например, 4096 байт. Файл, обычно представляющий собой неструктурированную последовательность однобайтовых записей, хранится в виде последовательности блоков (не обязательно смежных); каждый блок хранит целое число записей. В некоторых ОС (MS-DOS) адреса блоков, содержащих данные файла, могут быть организованы в связный список и вынесены в отдельную таблицу в памяти. В других ОС (Unix) адреса блоков данных файла хранятся в отдельном блоке внешней памяти (так называемом индексе или индексном узле). Этот прием, называемый индексацией, является наиболее распространенным для приложений, требующих произвольного доступа к записям файлов. Индекс файла состоит из списка элементов, каждый из которых содержит номер блока в файле и сведения о местоположении данного блока. Считывание очередного байта осуществляется с так называемой **текущей** позиции, которая характеризуется смещением от начала файла. Зная размер блока, легко вычислить номер блока, содержащего текущую позицию. Адрес же нужного блока диска можно затем извлечь из индекса файла. Базовой операцией, выполняемой по отношению к файлу, является чтение блока с диска и перенос его в буфер, находящийся в основной памяти.

Файловая система позволяет при помощи системы справочников (каталогов, директорий) связать уникальное имя файла с блоками вторичной памяти, содержащими данные файла. Иерархическая структура каталогов, используемая для управления файлами, может служить другим примером индексной структуры. В этом случае каталоги или папки играют роль индексов, каждый из которых содержит ссылки на свои подкаталоги. С этой точки зрения вся файловая система компьютера представляет собой большой индексированный файл. Помимо собственно файлов и структур данных, используемых для управления файлами (каталоги, дескрипторы файлов, различные таблицы распределения внешней памяти), понятие "файловая система" включает программные средства, реализующие различные операции над файлами.

Перечислим **основные функции** файловой системы.

1. Идентификация файлов. Связывание имени файла с выделенным ему пространством внешней памяти.
2. Распределение внешней памяти между файлами. Для работы с конкретным файлом пользователю не требуется иметь информацию о местоположении этого файла на внешнем носителе информации. Например, для того чтобы загрузить документ в редактор с жесткого диска, нам не нужно знать, на какой стороне какого магнитного диска, на каком цилиндре и в каком секторе находится данный документ.
3. Обеспечение надежности и отказоустойчивости. Стоимость информации может во много раз превышать стоимость компьютера.
4. Обеспечение защиты от несанкционированного доступа.
5. Обеспечение совместного доступа к файлам, так чтобы пользователю не приходилось прилагать специальных усилий по обеспечению синхронизации доступа.
6. Обеспечение высокой производительности.

Иногда говорят, что файл - это поименованный набор связанной информации, записанной во вторичную память. Для большинства пользователей файловая система - наиболее видимая часть ОС. Она предоставляет механизм для онлайнового хранения и доступа как к данным, так и к программам для всех пользователей системы. С точки зрения пользователя, файл - единица внешней памяти, то есть данные, записанные на диск, должны быть в составе какого-нибудь файла.

Важный аспект организации файловой системы - учет стоимости операций взаимодействия с вторичной памятью. Процесс считывания блока диска состоит из позиционирования считывающей головки над дорожкой, содержащей требуемый блок, ожидания, пока требуемый блок сделает оборот и окажется под головкой, и собственно считывания блока. Для этого требуется значительное время (десятки миллисекунд). В современных компьютерах обращение к диску осуществляется примерно в 100 000 раз медленнее, чем обращение к оперативной памяти. Таким образом, критерием вычислительной сложности алгоритмов, работающих с внешней памятью, является количество обращений к диску.

В данной лекции рассматриваются вопросы структуры, именования, защиты файлов; операции, которые разрешается производить над файлами; организация файлового архива (полного дерева справочников). Проблемы выделения дискового пространства, обеспечения производительной работы файловой системы и ряд других вопросов, интересующих разработчиков системы

Общие сведения о файлах

Имена файлов

Файлы представляют собой абстрактные объекты. Их задача - хранить информацию, скрывая от пользователя детали работы с устройствами. Когда процесс создает файл, он дает ему имя. После завершения процесса файл продолжает существовать и через свое имя может быть доступен другим процессам.

Правила именования файлов зависят от ОС. Многие ОС поддерживают имена из **двух частей** (имя+расширение), например progr.c (файл, содержащий текст программы на языке Си) или autoexec.bat (файл, содержащий команды интерпретатора командного языка). Тип расширения файла позволяет ОС организовать работу с ним различных прикладных программ в соответствии с заранее оговоренными соглашениями. Обычно ОС накладывают некоторые ограничения, как на используемые в имени символы, так и на длину имени файла. В соответствии со стандартом POSIX, популярные ОС оперируют удобными для пользователя длинными именами (до 255 символов).

Типы файлов

Важный аспект организации файловой системы и ОС - следует ли поддерживать и распознавать типы файлов. Если да, то это может помочь правильному функционированию ОС, например не допустить вывода на принтер бинарного файла.

Основные типы файлов: регулярные (обычные) файлы и директории (справочники, каталоги). Обычные файлы содержат пользовательскую информацию. Директории - системные файлы, поддерживающие структуру файловой системы. В каталоге содержится перечень входящих в него файлов и устанавливается соответствие между файлами и их характеристиками (атрибутами). Мы будем рассматривать директории ниже.

Напомним, что хотя внутри подсистемы управления файлами обычный файл представляется в виде набора блоков внешней памяти, для пользователей обеспечивается представление файла в виде линейной последовательности байтов. Такое представление позволяет использовать абстракцию файла при работе с внешними устройствами, при организации межпроцессных взаимодействий и т. д. Так, например, клавиатура обычно рассматривается как текстовый файл, из которого компьютер получает данные в символьном формате. Поэтому иногда к файлам приписывают другие объекты ОС, например специальные символьные файлы и специальные блочные файлы, именованные каналы и сокеты, имеющие файловый интерфейс. Эти объекты рассматриваются в других разделах данного курса.

Далее речь пойдет главным образом об **обычных файлах.**

Обычные (или регулярные) файлы реально представляют собой набор блоков (возможно, пустой) на устройстве внешней памяти, на котором поддерживается файловая система. Такие файлы могут содержать как текстовую информацию (обычно в формате ASCII), так и произвольную двоичную (бинарную) информацию.

Текстовые файлы содержат символьные строки, которые можно распечатать, увидеть на экране или редактировать обычным текстовым редактором.

Другой тип файлов - нетекстовые, или бинарные, файлы. Обычно они имеют некоторую внутреннюю структуру. Например, исполняемый файл в ОС Unix имеет пять секций: заголовок, текст, данные, биты реаллокации и символьную таблицу. ОС выполняет файл, только если он имеет нужный формат. Другим примером бинарного файла может быть архивный файл. Типизация файлов не слишком строгая.

Обычно прикладные программы, работающие с файлами, распознают тип файла по его имени в соответствии с общепринятыми соглашениями. Например, файлы с расширениями .c, .pas, .txt - ASCII-файлы, файлы с расширениями .exe - выполнимые, файлы с расширениями .obj, .zip - бинарные и т. д.

Атрибуты файлов

Кроме имени ОС часто связывают с каждым файлом и другую информацию, например дату модификации, размер и т. д. Эти другие характеристики файлов называются атрибутами. Список атрибутов в разных ОС может варьироваться. Обычно он содержит следующие элементы: основную информацию (имя, тип файла), адресную информацию (устройство, начальный адрес, размер), информацию об управлении доступом (владелец, допустимые операции) и информацию об использовании (даты создания, последнего чтения, модификации и др.).

Список атрибутов обычно хранится в структуре директорий (см. следующую лекцию) или других структурах, обеспечивающих доступ к данным файла.

Организация файлов и доступ к ним

Программист воспринимает файл в виде набора однородных записей. Запись - это наименьший элемент данных, который может быть обработан как единое целое прикладной программой при обмене с внешним устройством. Причем в большинстве ОС размер записи равен одному байту. В то время как приложения оперируют записями, физический обмен с устройством осуществляется большими единицами (обычно блоками). Поэтому записи объединяются в блоки для вывода и разблокируются - для ввода. Вопросы распределения блоков внешней памяти между файлами рассматриваются в следующей лекции.

ОС поддерживают несколько вариантов структуризации файлов.

Последовательный файл

Простейший вариант - так называемый последовательный файл. То есть файл является последовательностью записей. Поскольку записи, как правило, однобайтовые, файл представляет собой **неструктурированную последовательность байтов**.

Обработка подобных файлов предполагает последовательное чтение записей от начала файла, причем конкретная запись определяется ее положением в файле. Такой способ доступа называется последовательным (модель ленты). Если в качестве носителя файла используется магнитная лента, то так и делается. Текущая позиция считывания может быть возвращена к началу файла (rewind).

Файл прямого доступа

В реальной практике файлы хранятся на устройствах прямого (random) доступа, например на дисках, поэтому содержимое файла может быть разбросано по разным блокам диска, которые можно считывать в произвольном порядке. Причем номер блока однозначно определяется позицией внутри файла.

Здесь имеется в виду относительный номер, специфицирующий данный блок среди блоков диска, принадлежащих файлу. О связи относительного номера блока с абсолютным его номером на диске рассказывается в следующей лекции.

Естественно, что в этом случае для доступа к середине файла просмотр всего файла с самого начала не обязателен. Для специфицирования места, с которого надо начинать чтение, используются два способа: с начала или с текущей позиции, которую дает операция seek. Файл, байты которого могут быть считаны в произвольном порядке, называется файлом прямого доступа.

Таким образом, файл, состоящий из однобайтовых записей на устройстве прямого доступа, - наиболее распространенный способ организации файла. Базовыми операциями для такого рода файлов являются считывание или запись символа в текущую позицию. В большинстве языков высокого уровня предусмотрены операторы посимвольной пересылки данных в файл или из него.

Подобную логическую структуру имеют файлы во многих файловых системах, например в файловых системах ОС Unix и MS-DOS. ОС не осуществляет никакой интерпретации содержимого файла. Эта схема обеспечивает максимальную гибкость и универсальность. С помощью базовых системных вызовов (или функций библиотеки ввода/вывода) пользователи могут как угодно структурировать файлы. В частности, многие СУБД хранят свои базы данных в обычных файлах.

Другие формы организации файлов

Известны как другие формы организации файла, так и другие способы доступа к ним, которые использовались в ранних ОС, а также применяются сегодня в больших мэйнфреймах (mainframe), ориентированных на коммерческую обработку данных.

Первый шаг в структурировании - хранение файла в виде **последовательности записей фиксированной длины**, каждая из которых имеет внутреннюю структуру. Операция чтения производится над записью, а операция записи переписывает или добавляет запись целиком. Ранее использовались записи по 80 байт (это соответствовало числу позиций в перфокарте) или по 132 символа (ширина принтера). В ОС CP/M файлы были последовательностями 128-символьных записей. С введением CRT-терминалов данная идея утратила популярность.

Другой способ представления файлов - **последовательность записей переменной длины**, каждая из которых содержит ключевое поле в фиксированной позиции внутри записи (см. рис. 11.1). Базисная операция в данном случае - считать запись с каким-либо значением ключа. Записи могут располагаться в файле последовательно (например, отсортированные по значению ключевого поля) или в более сложном порядке. Метод доступа по значению ключевого поля к записям последовательного файла называется **индексно-последовательным**.

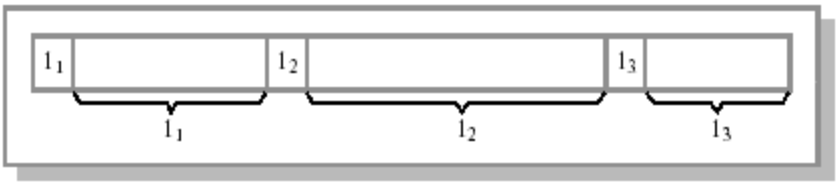


Рис. 11.1. Файл как последовательность записей переменной длины

В некоторых системах ускорение доступа к файлу обеспечивается конструированием **индекса** файла. Индекс обычно хранится на том же устройстве, что и сам файл, и состоит из списка элементов, каждый из которых содержит идентификатор записи, за которым следует указание о местоположении данной записи. Для поиска записи вначале происходит обращение к индексу, где находится указатель на нужную запись. Такие файлы называются **индексированными**, а метод доступа к ним - **доступ с использованием индекса**.

Предположим, у нас имеется большой несортированный файл, содержащий разнообразные сведения о студентах, состоящие из записей с несколькими полями, и возникает задача организации быстрого поиска по одному из полей, например по фамилии студента. Рис. 11.2 иллюстрирует решение данной проблемы - организацию метода доступа к файлу с использованием индекса.

Рис. 11.2. Пример организации индекса для последовательного файла

Следует отметить, что почти всегда главным фактором увеличения скорости доступа является **избыточность** данных.

Способ выделения дискового пространства при помощи индексных узлов, применяемый в ряде ОС (Unix и некоторых других, см. следующую лекцию), может служить другим примером организации индекса.

В этом случае ОС использует древовидную организацию блоков, при которой блоки, составляющие файл, являются листьями дерева, а каждый внутренний узел содержит указатели на множество блоков файла. Для больших файлов индекс может быть слишком велик. В этом случае создают индекс для индексного файла (блоки промежуточного уровня или блоки косвенной адресации).

Списки прав доступа

Наиболее общий подход к защите файлов от несанкционированного использования - сделать доступ зависящим от идентификатора пользователя, то есть связать с каждым файлом или директорией **список прав доступа** (access control list), где перечислены имена пользователей и типы разрешенных для них способов доступа к файлу. Любой запрос на выполнение операции сверяется с таким списком. Основная проблема реализации данного способа - список может быть длинным. Чтобы разрешить всем пользователям читать файл, необходимо всех их внести в список. У такой техники есть два нежелательных следствия.

- Конструирование подобного списка может оказаться сложной задачей, особенно если мы не знаем заранее пользователей системы.
- Запись в директории должна иметь переменный размер (включать список потенциальных пользователей).

Для решения этих проблем создают классификации пользователей, например, в ОС Unix все пользователи разделены на три группы.

- **Владелец** (Owner).
- **Группа** (Group). Набор пользователей, разделяющих файл и нуждающихся в типовом способе доступа к нему.
- **Остальные** (Univers).

Это позволяет реализовать конденсированную версию списка прав доступа. В рамках такой ограниченной классификации задаются только три поля (по одному для каждой группы) для каждой контролируемой операции. В итоге в Unix операции чтения, записи и исполнения контролируются при помощи 9 бит (rwxrwxrwx).

Связывание файлов

Иерархическая организация, положенная в основу древовидной структуры файловой системы современных ОС, не предусматривает выражения отношений, в которых потомки связываются более чем с одним предком. Такая негибкость частично устраняется возможностью реализации связывания файлов или организации линков (link).

Ядро позволяет пользователю связывать каталоги, упрощая написание программ, требующих пересечения дерева файловой системы (см. рис. 12.11). Часто имеет смысл хранить под разными именами одну и ту же команду (выполняемый файл). Например, выполняемый файл традиционного текстового редактора ОС Unix vi обычно может вызываться под именами ex, edit, vi, view и vedit файловой системы. Соединение между директорией и разделяемым файлом называется "связью" или "ссылкой" (link). Дерево файловой системы превращается в циклический граф.

Это удобно, но создает ряд дополнительных проблем.

Простейший способ реализовать связывание файла - просто дублировать информацию о нем в обеих директориях. При этом, однако, может возникнуть проблема совместимости в случае, если владельцы этих директорий попытаются независимо друг от друга изменить содержимое файла. Например, в ОС CP/M запись в директории о файле непосредственно содержит адреса дисковых блоков. Поэтому копии тех же дисковых адресов должны быть сделаны и в другой директории, куда файл линкуется. Если один из пользователей что-то добавляет к файлу, новые блоки будут перечислены только у него в директории и не будут "видны" другому пользователю.

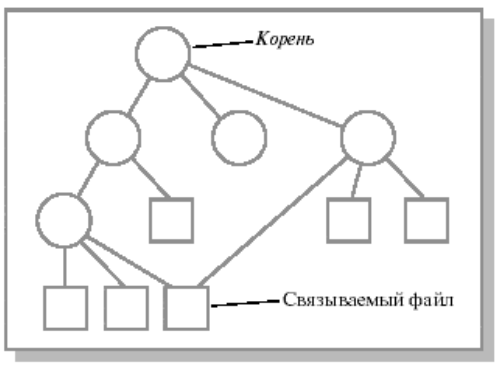


Рис. 12.11. Структура файловой системы с возможностью связывания файла с новым именем

Проблема такого рода может быть решена двумя способами. Первый из них - так называемая **жесткая** связь (hard link). Если блоки данных файла перечислены не в директории, а в небольшой структуре данных (например, в индексном узле), связанной собственно с файлом, то второй пользователь может связаться непосредственно с этой, уже существующей структурой.

Надежность файловых систем. Журнализация

Целостность файловой системы

Важный аспект надежной работы файловой системы - контроль ее целостности. В результате файловых операций блоки диска могут считываться в память, модифицироваться и затем записываться на диск. Причем многие файловые операции затрагивают сразу несколько объектов файловой системы. Например, копирование файла предполагает выделение ему блоков диска, формирование индексного узла, изменение содержимого каталога и т. д. В течение короткого периода времени между этими шагами информация в файловой системе оказывается несогласованной.

И если вследствие непредсказуемой остановки системы на диске будут сохранены изменения только для части этих объектов (нарушена атомарность файловой операции), файловая система на диске может быть оставлена в несовместимом состоянии. В результате могут возникнуть нарушения логики работы с данными, например появиться "потерянные" блоки диска, которые не принадлежат ни одному файлу и в то же время помечены как занятые, или, наоборот, блоки, помеченные как свободные, но в то же время занятые (на них есть ссылка в индексном узле) или другие нарушения.

В современных ОС предусмотрены меры, которые позволяют свести к минимуму ущерб от порчи файловой системы и затем полностью или частично восстановить ее целостность.

Порядок выполнения операций

Очевидно, что для правильного функционирования файловой системы значимость отдельных данных неравноценна. Искажение содержимого пользовательских файлов не приводит к серьезным (с точки зрения целостности файловой системы) последствиям, тогда как несоответствия в файлах, содержащих управляющую информацию (директории, индексные узлы, суперблок и т. п.), могут быть катастрофическими. Поэтому должен быть тщательно продуман порядок выполнения операций со структурами данных файловой системы.

Рассмотрим пример создания жесткой связи для файла [Робачевский, 1999]. Для этого файловой системе необходимо выполнить следующие операции:

- создать новую запись в каталоге, указывающую на индексный узел файла;
- увеличить счетчик связей в индексном узле.

Если аварийный останов произошел между 1-й и 2-й операциями, то в каталогах файловой системы будут существовать два имени файла, адресующих индексный узел со значением счетчика связей, равному 1. Если теперь будет удалено одно из имен, это приведет к удалению файла как такового. Если же порядок операций изменен и, как прежде, останов произошел между первой и второй операциями, файл будет иметь несуществующую жесткую связь, но существующая запись в каталоге будет правильной. Хотя это тоже является ошибкой, но ее последствия менее серьезны, чем в предыдущем случае.

Журнализация

Другим средством поддержки целостности является заимствованный из систем управления базами данных прием, называемый журнализация (иногда употребляется термин "журналирование"). Последовательность действий с объектами во время файловой операции протоколируется, и если произошел останов системы, то, имея в наличии протокол, можно осуществить откат системы назад в исходное целостное состояние, в котором она пребывала до начала операции. Подобная избыточность может стоить дорого, но она оправданна, так как в случае отказа позволяет реконструировать потерянные данные.

Для отката необходимо, чтобы для каждой протоколируемой в журнале операции существовала обратная. Например, для каталогов и реляционных СУБД это именно так. По этой причине, в отличие от СУБД, в файловых системах протоколируются не все изменения, а лишь изменения метаданных (индексных узлов, записей в каталогах и др.). Изменения в данных пользователя в протокол не заносятся. Кроме того, если протоколировать изменения

пользовательских данных, то этим будет нанесен серьезный ущерб производительности системы, поскольку кэширование потеряет смысл.

Журнализация реализована в NTFS, Ext3FS, ReiserFS и других системах. Чтобы подчеркнуть сложность задачи, нужно отметить, что существуют не вполне очевидные проблемы, связанные с процедурой отката. Например, отмена одних изменений может затрагивать данные, уже использованные другими файловыми операциями. Это означает, что такие операции также должны быть отменены. Данная проблема получила название каскадного отката транзакций [Брукшир, 2001]

Общая модель файловой системы (ФС). Отображаемые в память файлы. Современная архитектура ФС

Функционирование любой файловой системы можно представить многоуровневой моделью (рисунок 2.36), в которой каждый уровень предоставляет некоторый интерфейс (набор функций) вышележащему уровню, а сам, в свою очередь, для выполнения своей работы использует интерфейс (обращается с набором запросов) нижележащего уровня.

Отображаемые в память файлы (memory-mapped files) - это мощная возможность операционной системы. Она позволяет приложениям осуществлять доступ к файлам на диске тем же самым способом, каким осуществляется доступ к динамической памяти, то есть через указатели. Смысл отображения файла в память заключается в том, что содержимое файла (или часть содержимого) отображается в некоторый диапазон виртуального адресного пространства процесса, после чего обращение по какому-либо адресу из этого диапазона означает обращение к файлу на диске. Естественно, не каждое обращение к отображенному в память файлу вызывает операцию чтения/записи. Менеджер виртуальной памяти кэширует обращения к диску и тем самым обеспечивает высокую эффективность работы с отображенными файлами.

Сетевое взаимодействие. Протоколы

Протокол доставки пользовательских дейтаграмм UDP

Протокол UDP является одним из двух основных протоколов транспортного уровня, расположенных непосредственно над IP. Он предоставляет прикладным процессам транспортные услуги, которые не многим отличаются от услуг, предоставляемых протоколом IP. Протокол UDP обеспечивает ненадежную доставку датаграмм и не поддерживает соединений из конца в конец. Другими словами, его пакеты могут быть потеряны, продублированы или прийти не в том порядке, в котором они были отправлены. К заголовку IP-пакета он добавляет два поля, одно из которых, поле "порт", обеспечивает мультиплексирование информации между разными прикладными процессами, а другое поле - "контрольная сумма" - позволяет поддерживать целостность данных. Примерами сетевых приложений, использующих UDP, являются NFS и SNMP.

Протокол надежной доставки сообщений TCP

В стеке протоколов TCP/IP протокол TCP работает, как и протокол UDP, на транспортном уровне. Протокол TCP предоставляет транспортные услуги, отличающиеся от услуг UDP. Вместо ненадежной доставки датаграмм без установления соединений, он обеспечивает гарантированную доставку с установлением соединений между прикладными процессами в виде байтовых потоков.

Протокол TCP используется в тех случаях, когда требуется надежная доставка сообщений. Он освобождает прикладные процессы от необходимости использовать таймауты и повторные передачи для обеспечения надежности. Наиболее типичными прикладными процессами, использующими TCP, являются FTP и TELNET. Кроме того, TCP используют система X-Window, rcp (remote copy - удаленное копирование) и другие "r-команды". Большие возможности TCP даются не бесплатно. Реализация TCP требует большой производительности процессора и большой пропускной способности сети. Внутренняя структура модуля TCP гораздо сложнее структуры модуля UDP.

Таблица 16.1. Уровни модели OSI.

Номер уровня	Название уровня	Единица информации
Layer 7	Уровень приложений	Данные (data)
Layer 6	Представительский уровень	Данные (data)
Layer 5	Сессионный уровень	Данные (data)
Layer 4	Транспортный уровень	Сегмент (segment)
Layer 3	Сетевой уровень	Пакет (packet)
Layer 2	Уровень передачи данных	Фрейм (frame)
Layer 1	Физический уровень	Бит (bit)

Хотя сегодня существуют разнообразные модели сетей, большинство разработчиков придерживается именно этой общепризнанной схемы.

Безопасность вычислительных систем

Системы хранения данных. RAID 0, 1,2,3, 4 Системы хранения данных. RAID 5, 6 , 10, 30, 50

Защита хранилища (данных или чего-либо другого) осуществляется независимо от HACMP; для обеспечения высокой доступности хранилища необходимо использовать хранилище с надлежащим уровнем избыточности и отказоустойчивости. HACMP не имеет какого-либо контроля над доступностью хранилища. Для защиты данных можно технологию RAID либо применять (на уровне хранилища или адаптера), либо зеркальное отображение AIX LVM (RAID 1).

Дисковые массивы RAID (Redundant Array of Independent Disks) представляют собой группы совместно работающих дисковых приводов, обеспечивающих более высокую скорость передачи, чем одиночные (независимые) приводы. Массивы также могут обеспечивать избыточность данных, чтобы не возникало потерь данных при отказе одного привода (физического диска) в массиве. В зависимости от уровня RAID выполняется либо зеркальное отображение данных, либо чередование данных, либо совмещение обоих методов. Ниже дается описание подуровней RAID.

- RAID-0. Также называется "чередованием данных" (striping). Обычно файл последовательно записывается на один диск. При чередовании информация разделяется на фрагменты – chunks (при фиксированном размере такие фрагменты обычно называются блоками – blocks), а фрагменты параллельно записываются на набор дисков (или считываются с него). Это дает следующие преимущества с точки зрения производительности: более высокую скорость передачи данных при последовательных операциях в связи с наложением нескольких потоков ввода-вывода; более высокую пропускную способность при произвольном доступе, так как устраняется сдвиг схемы доступа в связи с распределением данных. Это означает, что при равномерном распределении данных по нескольким дискам требуемая информация, скорее всего, будет расположена на нескольких дисках, вследствие чего повышается пропускная способность нескольких приводов. RAID-0 предназначен только для повышения производительности. Избыточность отсутствует, так что каждый диск является единой точкой сбоя.
- RAID-1. RAID-1 также называется "зеркальным отображением дисков". При такой схеме разные диски содержат идентичные копии каждого фрагмента данных; другими словами, у каждого диска есть "двойник", содержащий точную реплику (зеркальный образ) информации. При отказе какого-либо диска в массиве наличие зеркального диска обеспечивает доступность данных. Это также позволяет повысить производительность чтения, так как всегда используется тот диск, у которого привод (головка диска) расположен ближе к требуемым данным, что сокращает время поиска. Время реакции при записи может быть дольше, чем при одном диске, в зависимости от политики записи; операции записи могут выполняться либо параллельно (для более быстрой реакции), либо последовательно (для надежности).
- RAID-2 и RAID-3. Представляют собой массивы с механизмом параллельной обработки, где все приводы в массиве работают в согласии. Подобно чередованию данных, записываемая на диск информация разделяется на фрагменты (фиксированные блоки данных) и каждый фрагмент записывается в одну и ту же физическую позицию на различных дисках (параллельно). При чтении на каждый диск могут направляться одновременные запросы данных. Такая архитектура требует записи данных четности для каждой полосы данных; различие между RAID-2 и RAID-3 состоит в том, что RAID-2 может использовать несколько дисковых приводов для записи данных четности, тогда как RAID-3 может использовать только один дисковый привод. При отказе привода система может воссоздать потерянные данные по оставшимся дискам и данным четности. При больших объемах данных производительность очень высокая, однако на небольших запросах производительность низкая, так как во всех операциях всегда применяются все приводы и не может иметь место наложение или независимое выполнение операций.
- RAID-4. Эта технология призвана решить некоторые недостатки технологии RAID3 путем использования фрагментов данных более крупного размера и чередования данных по всем дискам, кроме одного, зарезервированного для данных четности. Использование чередования данных означает, что запросы ввода-вывода должны только указывать привод, на котором требуемые данные находятся в действительности. Это означает, что возможно одновременное и независимое выполнение операций чтения. Однако запросы записи требуют использования цикла "чтение/изменение/обновление", что создает "узкое место" на одном диске четности. Необходимо считать каждую полосу, вставить новые данные, после чего вычислить новые данные четности, прежде чем записать полосу обратно на диск. Затем на диск четности записываются новые данные четности, и этот диск не может использоваться для других операций записи, пока не будет завершена эта операция. Из-за этого "узкого места" технология RAID-4 не используется настолько часто, как RAID-5, в которой тот же процесс реализован без "узкого места".
- RAID 5. Эта технология очень похожа на RAID-4. Разница состоит в том, что данные четности также распределяются по тем же дискам, которые используются для данных, устраняя, таким образом, "узкое место". Данные четности никогда не сохраняются на том же приводе, на котором сохраняются и защищаемые им фрагменты. Это позволяет выполнять одновременные операции чтения и записи, увеличивая производительность по причине доступности дополнительного диска (диска, который ранее использовался для данных четности). Существуют другие функции, позволяющие еще больше повысить скорость передачи данных, такие как кеширование одновременных операций чтения с дисков и передача этой информации во время чтения следующих блоков. Это позволяет достичь скорости передачи данных на уровне скорости адаптера. Как и в технологии RAID-3, при отказе диска информация может быть воссоздана с других приводов. Массив RAID-5 всегда использует данные четности, однако все равно важно регулярно создавать резервные копии данных в массиве. Массивы RAID-5 распределяют данные по всем дискам массива, по одному сегменту зараз (сегмент может содержать несколько блоков). В массиве из n приводов полоса содержит сегменты данных, записываемые на n-1 дисков, и сегмент четности, записываемый на n-й диск. Использование этого механизма также означает, что не все дисковое пространство доступно для записи данных. Например, в массиве из пяти дисков по 72 Гб, несмотря на то что суммарный размер составляет 360 Гб, только 288 Гб доступны для записи данных.
- RAID 0+1 (RAID-10), также известный как IBM RAID-1 Enhanced или RAID-10, представляет сочетание RAID-0 (чередование данных) и RAID-1 (зеркальное отображение данных). RAID-10 имеет такие же преимущества производительности, как и RAID-0, обеспечивая при этом доступность данных, характерную для RAID-1. В конфигурации RAID-10 осуществляется чередование как для данных, так и для их зеркального отображения по всем дискам массива. Первая полоса представляет полосу данных, тогда как вторая полоса представляет зеркальное отображение, где зеркальное отображение записывается на другой физический диск. Реализации RAID-10 обеспечивают отличную производительность записи, так как им не приходится вычислять и записывать данные четности. RAID-10 может быть реализован программным способом (AIX LVM), аппаратным способом (уровень подсистемы хранения) либо сочетанием аппаратного и программного способов. Выбор подходящего решения зависит от общих требований. RAID-10 имеет такие же стоимостные характеристики, как и RAID-1.

Таблица 2.3. Характеристики широко используемых уровней RAID

Уровень RAID	Доступное дисковое пространство, %	Производительность операций чтения-записи	Стоимость	Защита данных
RAID-0	100	Высокая и для чтения и для записи	Низкая	Нет
RAID-1	50	Средневысокая для чтения, средняя для записи	Высокая	Есть
RAID-5	80	Высокая для чтения, средняя для записи	Средняя	Есть
RAID 0+1	50	Высокая и для чтения и для записи	Высокая	Есть

Важно! Несмотря на то, что на всех уровнях технологии RAID (кроме RAID-0) реализуется избыточность данных, необходимо регулярно осуществлять резервное копирование данных. Это единственный способ восстановления данных в случае случайного повреждения или удаления файла или каталога.

ОС QNX

Как микроядерная операционная система, QNX основана на идее работы основной части своих компонентов, как небольших задач, называемых сервисами. Это отличает её от традиционных монолитных ядер, в которых ядро операционной системы — одна большая программа, состоящая из большого количества «частей», каждая со своими особенностями. Использование микроядра в QNX позволяет пользователям (разработчикам) отключить любую ненужную им функциональность, не изменяя ядро. Вместо этого, можно просто не запускать определённый процесс.

Система достаточно небольшая, чтобы в минимальной комплектации уместиться на одну дискету, вместе с этим она считается очень быстрой и должным образом «законченной» (практически не содержащей ошибок).

QNX Neutrino, выпущенная в 2001 году, перенесена на многие платформы, и сейчас способна работать практически на любом современном процессоре, используемом на рынке встраиваемых систем.

ОС компактных компьютеров