Файловые системы

Файловая система - это часть операционной системы, обеспечивающая пользователю удобный интерфейс при работе с данными, хранящимися на диске, и совместное использование файлов несколькими пользователями и процессами.

Файловая система включает:

- совокупность всех файлов на диске;
- наборы структур данных, используемых для управления файлами, такие как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске;
- комплекс системных программных средств, реализующих управление файлами: создание, уничтожение, чтение, запись, именование, поиск и другие операции.

Файлы идентифицируются именами. Пользователи дают файлам символьные имена, при этом учитываются ограничения ОС как на используемые символы, так и на длину имени. До недавнего времени эти границы были весьма узкими. Так в популярной файловой системе FAT длина имен ограничивается известной схемой 8.3 (8 символов - собственно имя, 3 символа - расширение имени), а в ОС UNIX System V имя не может содержать более 14 символов. Однако пользователю гораздо удобнее работать с длинными именами, поскольку они позволяют дать файлу действительно мнемоническое название, по которому даже через достаточно большой промежуток времени можно будет вспомнить, что содержит этот файл. Поэтому современные файловые системы, как правило, поддерживают длинные символьные имена файлов. Например, Windows NT в своей новой файловой системе NTFS устанавливает, что имя файла может содержать до 255 символов, не считая завершающего нулевого символа.

При переходе к длинным именам возникает проблема совместимости с ранее созданными приложениями, использующими короткие имена. Чтобы приложения могли обращаться к файлам в соответствии с принятыми ранее соглашениями, файловая система должна уметь предоставлять эквивалентные короткие имена (псевдонимы) файлам, имеющим длинные имена. Таким образом, одной из важных задач становится проблема генерации соответствующих коротких имен.

Длинные имена поддерживаются не только новыми файловыми системами, но и новыми версиями хорошо известных файловых систем. Например, в ОС Windows 95 используется файловая система VFAT, представляющая собой существенно измененный вариант FAT. Среди многих других усовершенствований одним из главных достоинств VFAT является поддержка длинных имен. Кроме проблемы генерации эквивалентных коротких имен, при реализации нового варианта FAT важной задачей была задача хранения длинных имен при условии, что принципиально метод хранения и структура данных на диске не должны были измениться.

Обычно разные файлы могут иметь одинаковые символьные имена. В этом случае файл однозначно идентифицируется так называемым составным именем, представляющем собой последовательность символьных имен каталогов. В некоторых системах одному и тому же файлу не может быть дано несколько разных имен, а в других такое ограничение отсутствует. В последнем случае операционная система присваивает файлу дополнительно **уникальное имя**, так, чтобы можно было установить взаимно-однозначное соответствие между файлом и его уникальным именем. **Уникальное имя** представляет собой **числовой идентификатор** и используется программами операционной системы. Примером такого уникального имени файла является номер индексного дескриптора в системе UNIX.

Файлы бывают разных типов: обычные файлы, специальные файлы, файлы-каталоги.

Обычные файлы подразделяются на текстовые и двоичные.

Текстовые файлы состоят из строк символов, представленных в ASCII-коде. Это могут быть документы, исходные тексты программ и тому подобное. Текстовые файлы можно прочитать на экране и распечатать на принтере. Двоичные файлы не используют ASCII-коды, они, как правило, имеют сложную внутреннюю структуру, например, объектный код программы или архивный файл.

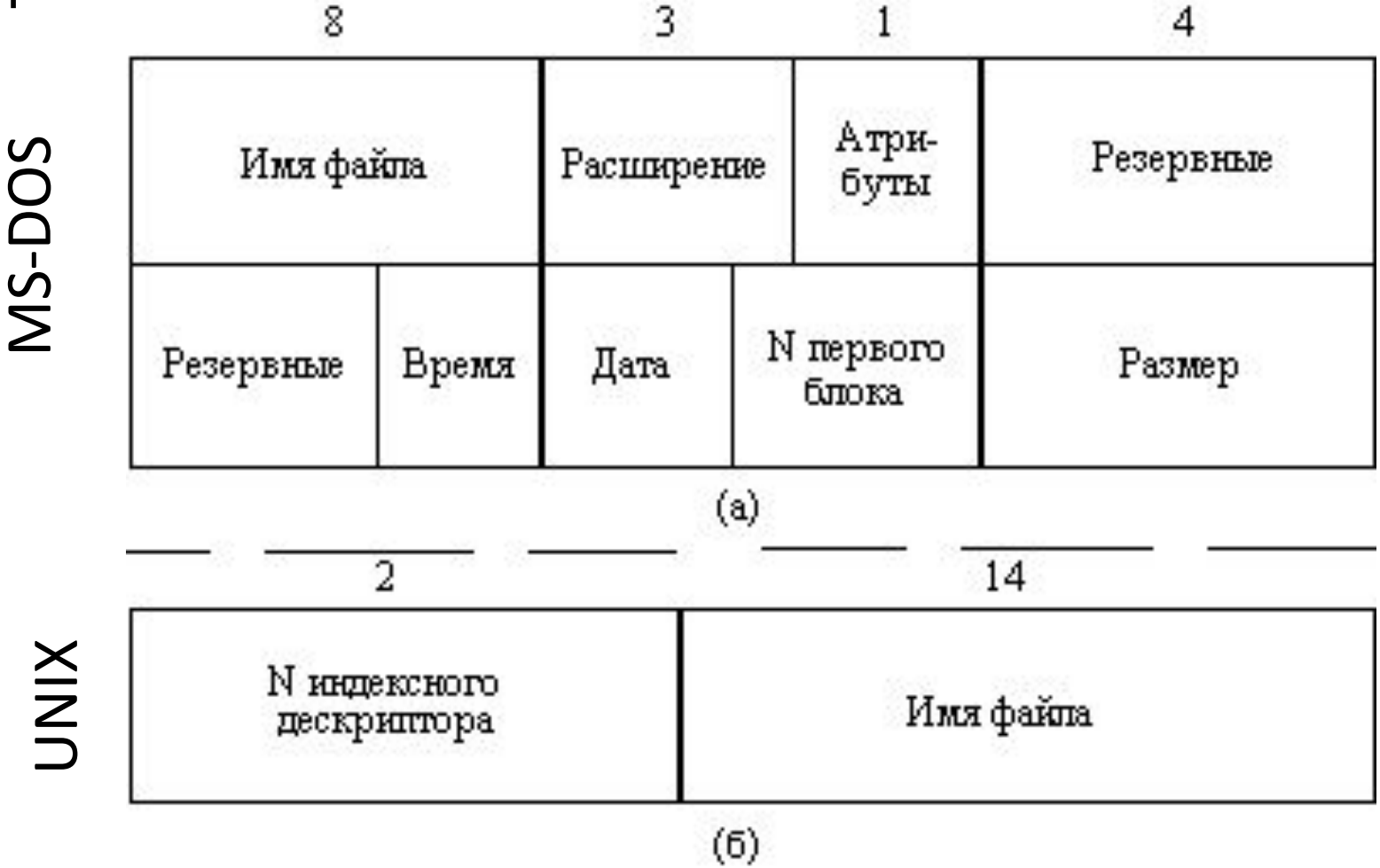
Специальные файлы - это файлы, ассоциированные с устройствами ввода-вывода, которые позволяют пользователю выполнять соответствующие операции, используя обычные команды записи в файл или чтения из файла. Такие команды обрабатываются вначале программами файловой системы, а затем на некотором этапе выполнения запроса преобразуются ОС в команды управления вводом-выводом. Специальные файлы, так же как и устройства ввода-вывода, делятся на *блок-ориентированные и байт-ориентированные*.

Каталог - это, с одной стороны, группа файлов, объединенных пользователем исходя из некоторых соображений, а с другой стороны - это файл, содержащий системную информацию о группе его составляющих файлов. В каталоге содержится список файлов, входящих в него, и устанавливается соответствие между файлами и их характеристиками - атрибутами.

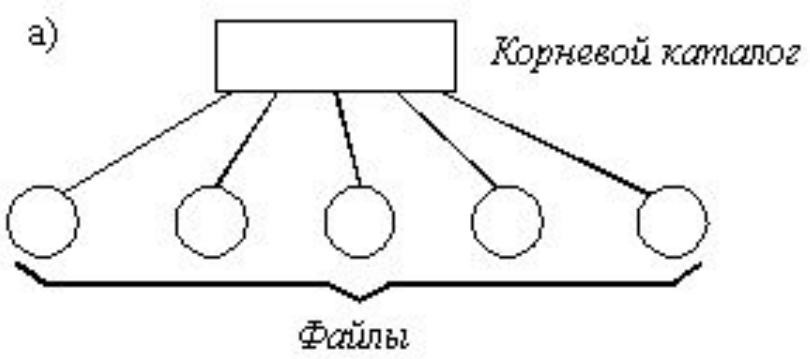
В разных файловых системах могут использоваться в качестве атрибутов разные характеристики, например:

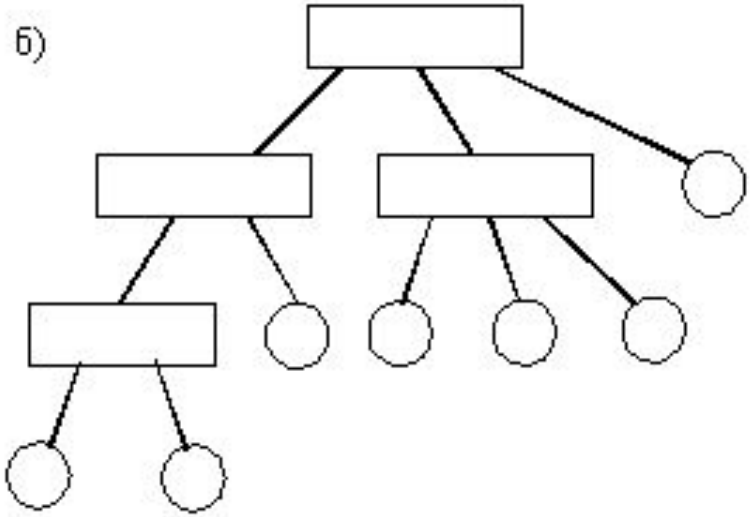
информация о разрешенном доступе;
пароль для доступа к файлу;
владелец файла;
создатель файла;
признак «только для чтения»;
признак «скрытый файл»;
признак «системный файл»;
признак «архивный файл»;
признак «двоичный/символьный»;
признак «временный», что означает «удалить после завершения процесса»;
признак блокировки;
длина записи;
указатель на ключевое поле в записи;
длина ключа;
время создания, последнего доступа и последнего изменения;
текущий размер файла;
максимальный размер файла.

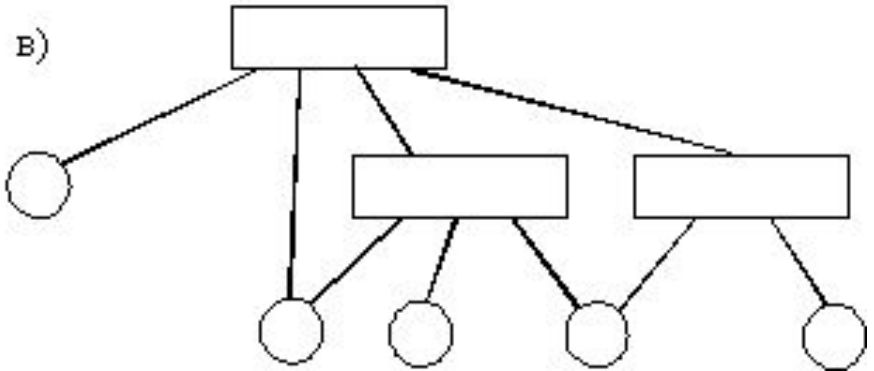
Каталоги могут непосредственно содержать значения характеристик файлов, как это сделано в файловой системе MS-DOS, или ссылаться на таблицы, содержащие эти характеристики, как это реализовано в UNIX. Каталоги могут образовывать иерархическую структуру, когда каталог более низкого уровня может входить в каталог более высокого уровня.



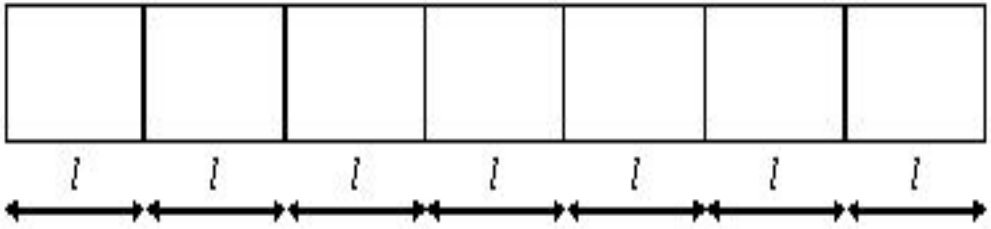
Иерархия каталогов может быть деревом или сетью. Каталоги образуют дерево, если файлу разрешено входить только в один каталог, и сеть - если файл может входить сразу в несколько каталогов. В **MS-DOS** каталоги образуют древовидную структуру, а в **UNIX** сетевую. Как и любой другой файл, каталог имеет символьное имя и однозначно идентифицируется составным именем, содержащим цепочку символьных имен всех каталогов, через которые



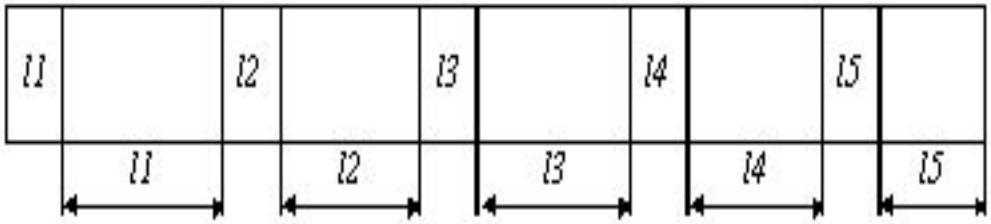




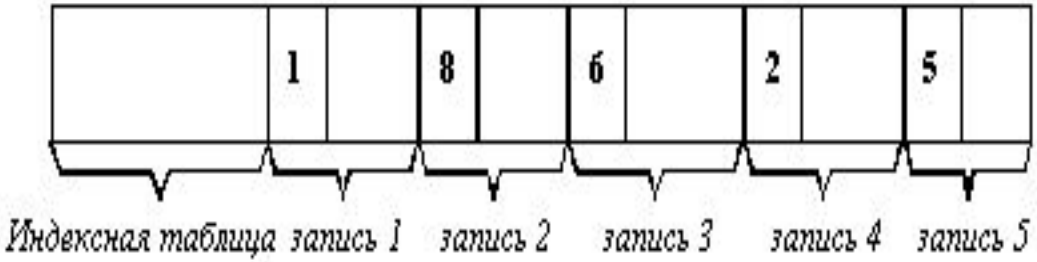
Программист имеет дело с логической организацией файла, представляя файл в виде определенным образом организованных логических записей. Логическая запись - это наименьший элемент данных, которым может оперировать программист при обмене с внешним устройством. Даже если физический обмен с устройством осуществляется большими единицами, операционная система обеспечивает программисту доступ к отдельной логической



Последовательность логических записей фиксированной длины



Последовательность логических записей переменной длины

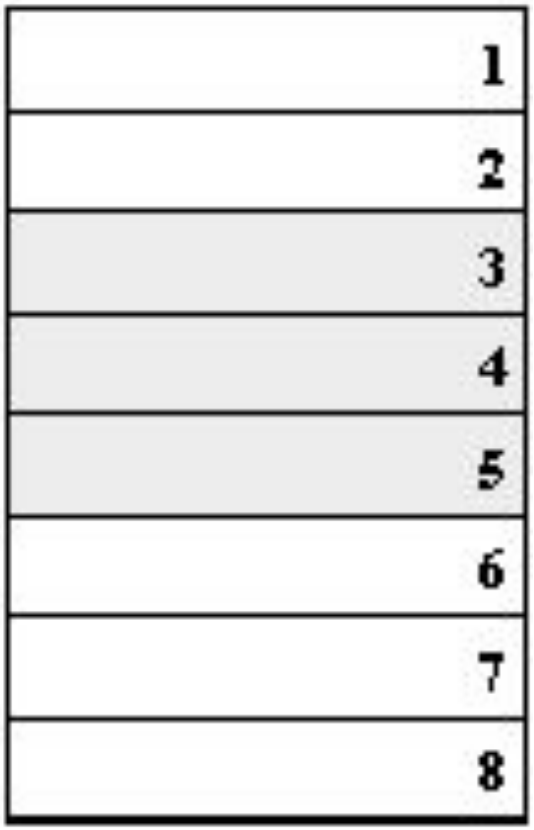


Индексная логическая организация

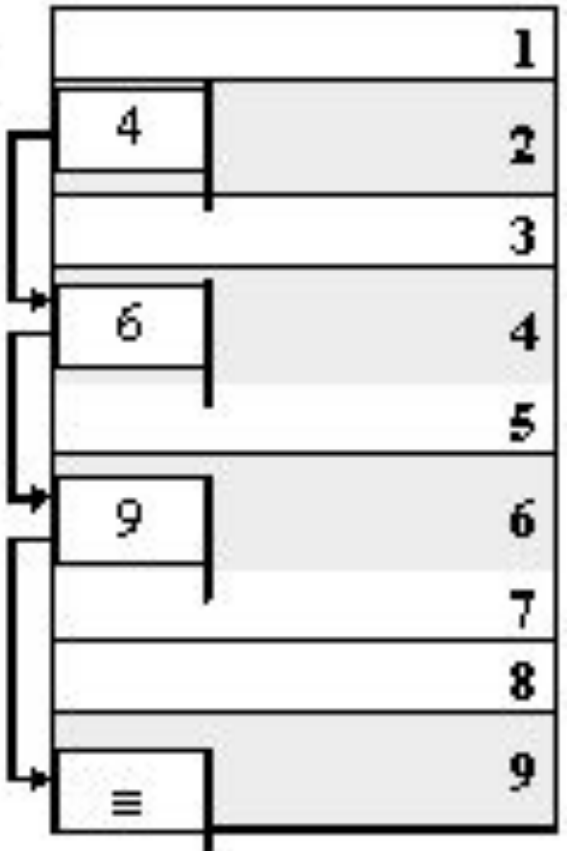
Индекс	1	2	3	4	5	6
Адрес	21	201	315	661	670	715

Индекс ≡ ключ

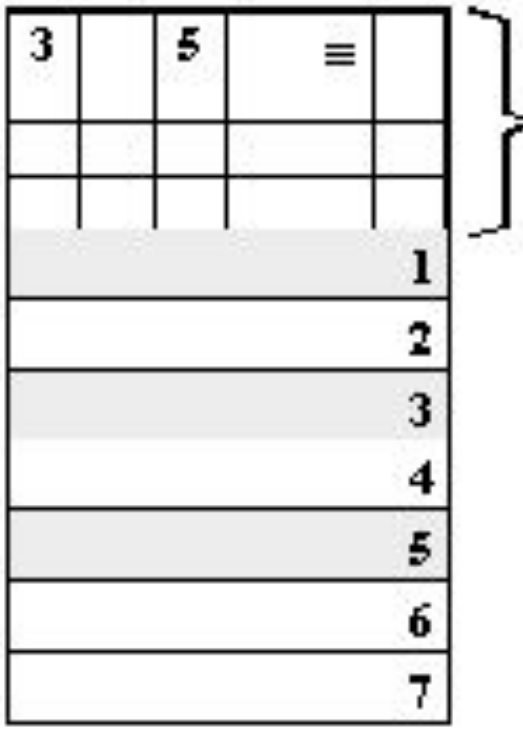
Физическая организация файла описывает правила расположения файла на устройстве внешней памяти, в частности на диске. Файл состоит из физических записей - блоков. Блок - наименьшая единица данных, которой внешнее устройство обменивается с оперативной памятью. Непрерывное размещение - простейший вариант физической организации, при котором файлу предоставляется последовательность блоков диска, образующих единый сплошной участок дисковой памяти. Для задания адреса файла в этом случае достаточно указать только номер начального блока. Другое достоинство этого метода - простота. Но имеются и два существенных недостатка. Во-первых, во время создания файла заранее не известна его длина, а значит не известно, сколько памяти надо зарезервировать для этого файла, во-вторых, при таком порядке размещения неизбежно возникает фрагментация, и пространство на диске используется не эффективно, так как отдельные участки маленького размера (минимально 1 блок) могут остаться не используемыми.



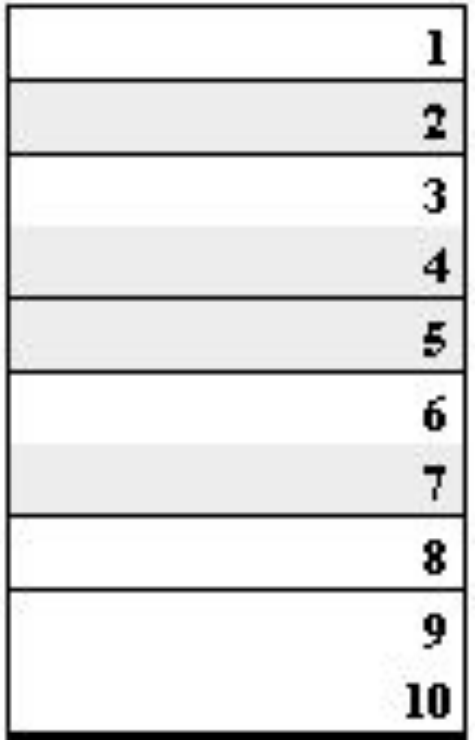
Следующий способ физической организации - размещение в виде связанного списка блоков дисковой памяти. При таком способе в начале каждого блока содержится указатель на следующий блок. В этом случае адрес файла также может быть задан одним числом - номером первого блока. В отличие от предыдущего способа, каждый блок может быть присоединен в цепочку какого-либо файла, следовательно фрагментация отсутствует. Файл может изменяться во время своего существования, наращивая число блоков. Недостатком является сложность реализации доступа к произвольно заданному месту файла: для того, чтобы прочитать пятый по порядку блок файла, необходимо последовательно прочитать четыре первых блока, прослеживая цепочку номеров блоков. Кроме того, при этом способе количество данных файла, содержащихся в одном блоке, не равно степени двойки (одно слово израсходовано на номер следующего блока), а многие программы читают данные блоками, размер которых равен степени двойки.

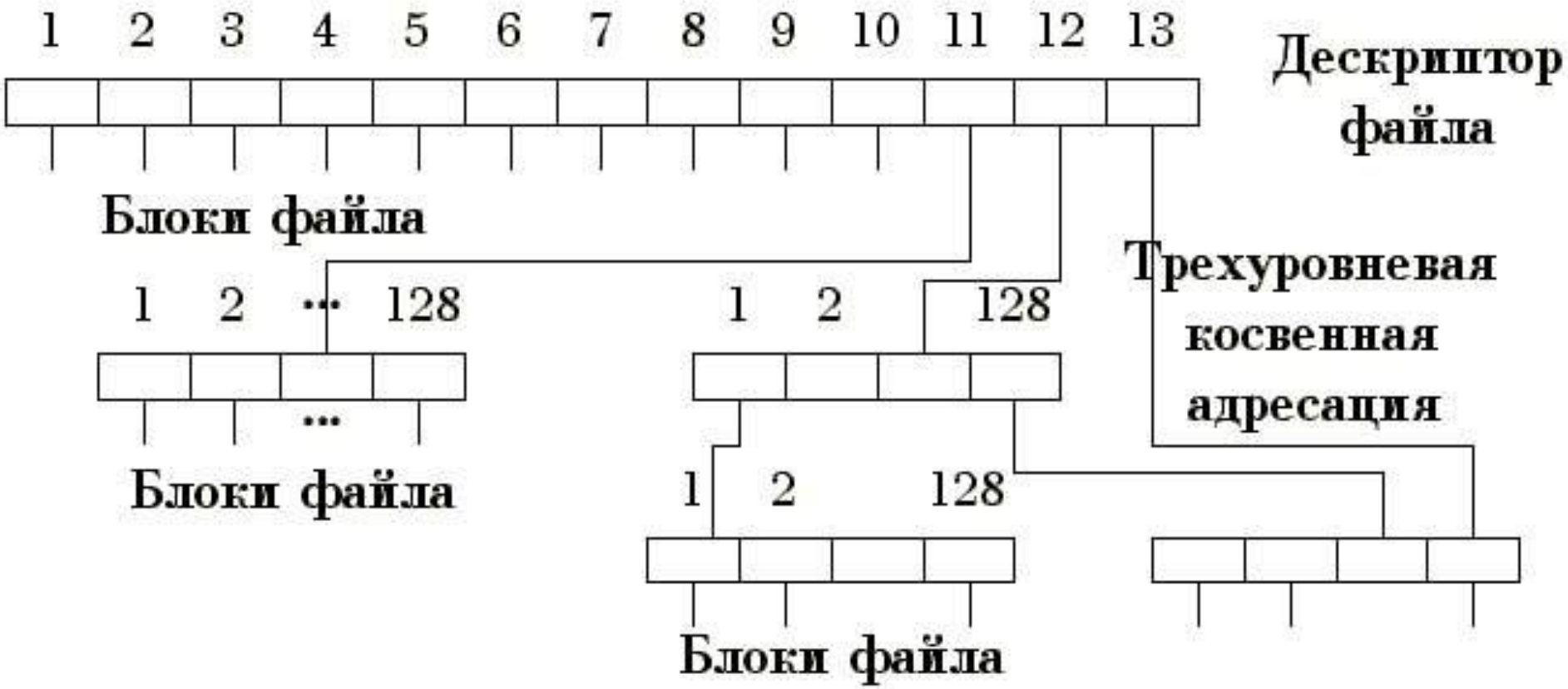


Популярным способом, используемым, например, в файловой системе FAT операционной системы MS-DOS, является использование связанного списка индексов. С каждым блоком связывается некоторый элемент - индекс. Индексы располагаются в отдельной области диска (в MS-DOS это таблица FAT). Если некоторый блок распределен некоторому файлу, то индекс этого блока содержит номер следующего блока данного файла. При такой физической организации сохраняются все достоинства предыдущего способа, но снимаются оба отмеченных недостатка: во-первых, для доступа к произвольному месту файла достаточно прочитать только блок индексов, отсчитать нужное количество блоков файла по цепочке и определить номер нужного блока, и, во-вторых, данные файла занимают блок целиком, а значит имеют объем, равный



Рассмотрим задание физического расположения файла путем простого перечисления номеров блоков, занимаемых этим файлом. ОС UNIX использует вариант данного способа, позволяющий обеспечить фиксированную длину адреса, независимо от размера файла. Для хранения адреса файла выделено 13 полей. Если размер файла меньше или равен 10 блокам, то номера этих блоков непосредственно перечислены в первых десяти полях адреса. Если размер файла больше 10 блоков, то следующее 11-е поле содержит адрес блока, в котором могут быть расположены еще 128 номеров следующих блоков файла. Если файл больше, чем 10+128 блоков, то используется 12-е поле, в котором находится номер блока, содержащего 128 номеров блоков, которые содержат по 128 номеров блоков данного файла. И, наконец, если файл больше 10+128+128(128, то используется последнее 13-е поле для тройной косвенной адресации, что позволяет задать адрес




1 2 3 4 5 6 7 8 9 10 11 12 13
Дескриптор файла
Блоки файла
1 2 ... 128
1 2 128
Трехуровневая косвенная адресация
Блоки файла
1 2 128
Блоки файла

Права доступа к файлу

Определить права доступа к файлу - значит определить для каждого пользователя набор операций, которые он может применить к данному файлу. В разных файловых системах может быть определен свой список дифференцируемых операций доступа. Этот список может включать следующие операции:

- создание файла;
- уничтожение файла;
- открытие файла;
- закрытие файла;
- чтение файла;
- запись в файл;
- дополнение файла;
- поиск в файле;
- получение атрибутов файла;
- установление новых значений атрибутов;
- переименование;
- выполнение файла;
- чтение каталога и другие операции с файлами и каталогами.

В самом общем случае права доступа могут быть описаны матрицей прав доступа, в которой столбцы соответствуют всем файлам системы, строки - всем пользователям, а на пересечении строк и столбцов указываются разрешенные операции. В некоторых системах пользователи могут быть разделены на отдельные категории. Для всех пользователей одной категории определяются единые права доступа. Например, в системе UNIX все пользователи подразделяются на три пы и всех осталь

Имена файлов

Имена пользователей	modem.txt	win.exe	class.dbf	unix.ppt
kira	читать	выполнять	–	выполнять
genya	читать	выполнять	–	выполнять читать
nataly	читать	–	–	выполнять читать
victor	читать писать	–	создать	–

Различают два основных подхода к определению прав доступа:

- избирательный доступ, когда для каждого файла и каждого пользователя сам владелец может определить допустимые операции;
- мандатный подход, когда система наделяет пользователя определенными правами по отношению к каждому разделяемому ресурсу в зависимости от того, к какой группе пользователей он относится.

В некоторых файловых системах запросы к внешним устройствам перехватываются промежуточным программным слоем подсистемой буферизации. Подсистема буферизации представляет собой буферный пул, располагающийся в оперативной памяти, и комплекс программ, управляющих этим пулом. Каждый буфер пула имеет размер, равный одному блоку. При поступлении запроса на чтение некоторого блока подсистема буферизации просматривает свой буферный пул и, если находит требуемый блок, то копирует его в буфер запрашивающего процесса. Операция ввода-вывода считается выполненной, хотя физического обмена с устройством не происходило. Очевиден выигрыш во времени доступа к файлу. Если же нужный блок в буферном пуле отсутствует, то он считывается с устройства и одновременно с передачей запрашивающему процессу копируется в один из буферов подсистемы буферизации. При отсутствии свободного буфера на диск вытесняется наименее используемая информация. Таким образом, подсистема

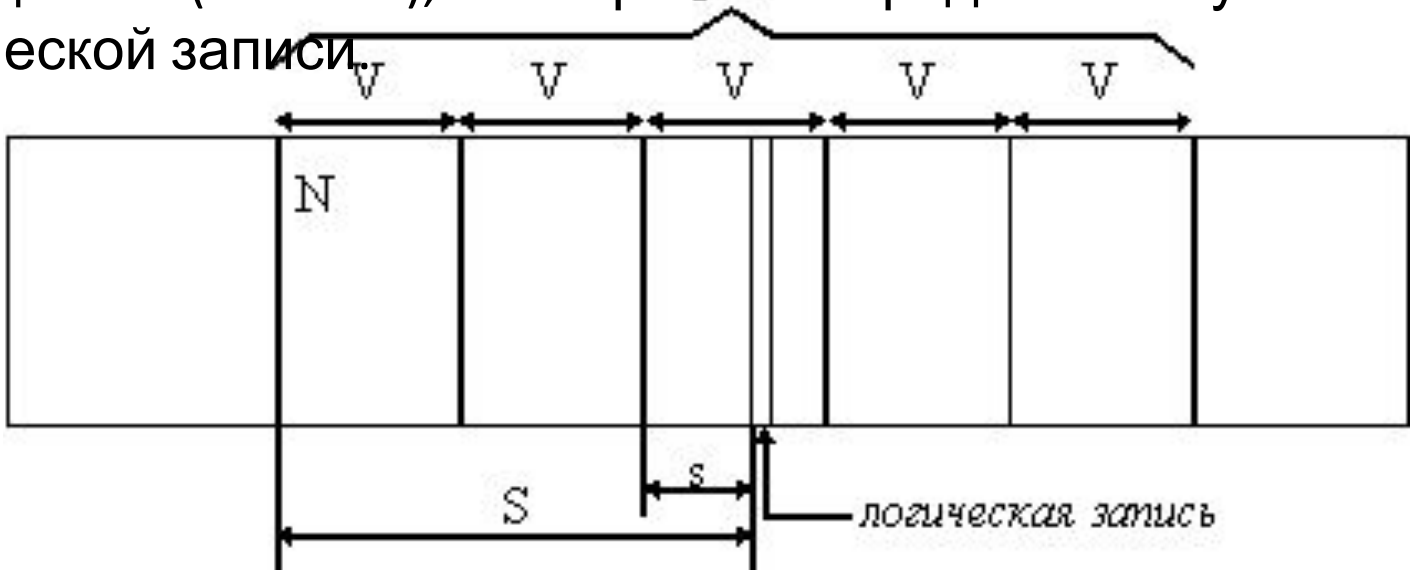
Функционирование любой файловой системы можно представить многоуровневой моделью, в которой каждый уровень предоставляет некоторый интерфейс вышележащему уровню, а сам, в свою очередь, использует интерфейс нижележащ

Задачей **символьного уровня** является определение по символьному имени файла его уникального имени. В файловых системах, в которых каждый файл может иметь только одно символьное имя (например, MS-DOS), этот уровень отсутствует, так как символьное имя, присвоенное файлу пользователем, является одновременно уникальным и может быть использовано операционной системой. В других файловых системах, в которых один и тот же файл может иметь несколько символьных имен, на данном уровне просматривается цепочка каталогов для определения уникального имени файла.

На следующем, **базовом уровне**, по уникальному имени файла определяются его характеристики: *права доступа, адрес, размер и другие*. При открытии файла его характеристики перемещаются с диска в оперативную память, чтобы уменьшить среднее время к его доступу.

Следующим этапом реализации запроса к файлу является проверка прав доступа к нему. Для этого сравниваются полномочия пользователя или процесса, выдавших запрос, со списком разрешенных видов доступа к данному файлу. Если запрашиваемый вид доступа разрешен, то выполнение запроса продолжается, если

На логическом уровне определяются координаты запрашиваемой логической записи в файле, то есть требуется определить, на каком расстоянии (в байтах) от начала файла находится требуемая логическая запись. При этом абстрагируются от физического расположения файла, он представляется в виде непрерывной последовательности байт. Алгоритм работы данного уровня зависит от логической организации файла. Например, если файл организован как последовательность логических записей фиксированной длины **l**, то **n**-ая логическая запись имеет смещение **l((n-1)** байт. Для определения координат логической записи в файле с индексно-последовательной организацией выполняется чтение таблицы индексов (ключей), в которой непосредственно указывается адрес логической записи.



Исходные данные:

V - размер блока

N - номер первого блока файла

S - смещение логической записи в файле

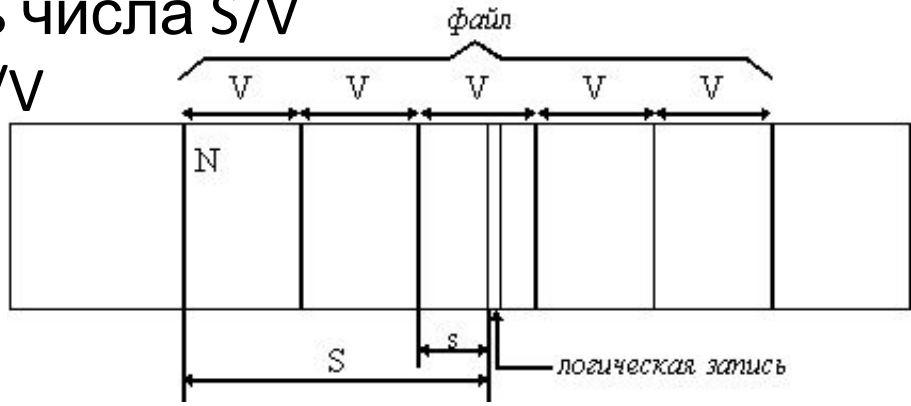
Требуется определить на физическом уровне:

n - номер блока, содержащего требуемую логическую запись

s - смещение логической записи в пределах блока

n = N + [S/V], где [S/V] - целая часть числа S/V

s = R [S/V] - дробная часть числа S/V

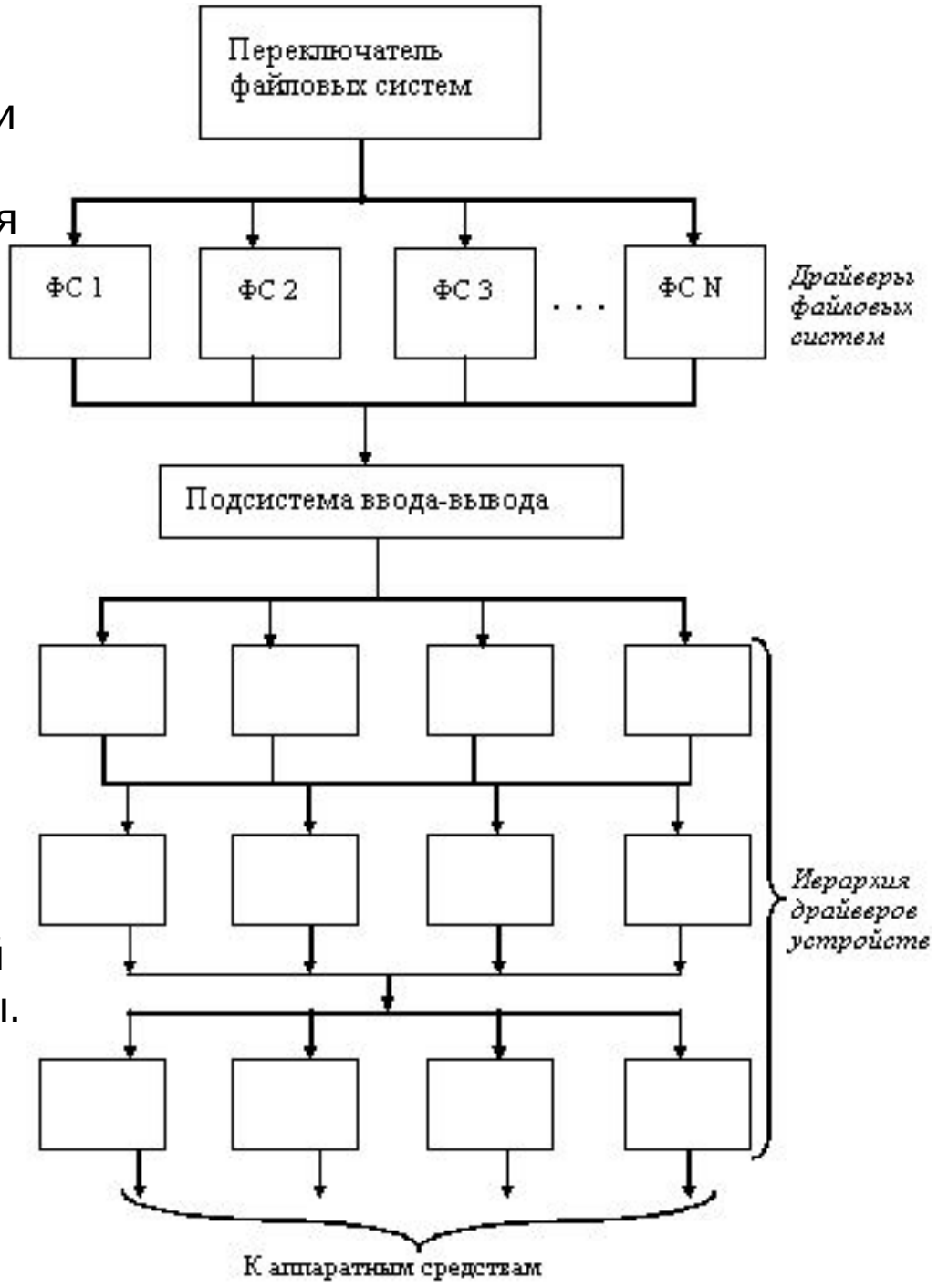


На физическом уровне файловая система определяет номер физического блока, который содержит требуемую логическую запись, и смещение логической записи в физическом блоке. Для решения этой задачи используются результаты работы логического уровня - смещение логической записи в файле, адрес файла на внешнем устройстве, а также сведения о физической организации файла, включая размер блока.

Разработчики операционных систем стремятся обеспечить пользователя возможностью работать сразу с несколькими файловыми системами. В новом понимании файловая система состоит из многих составляющих, в число которых входят и файловые системы в традиционном понимании.

Новые файловые системы имеют многоуровневую структуру, на верхнем уровне которых располагается переключатель файловых систем, который называется устанавливаемым диспетчером файловой системы - installable filesystem manager (**IFS**). Он обеспечивает интерфейс между запросами приложения и конкретной файловой системой, к которой оно обращается. Переключатель файловых систем преобразует запросы в формат, воспринимаемый следующим уровнем - уровнем файловых систем.

Каждый компонент уровня файловых систем выполнен в виде драйвера соответствующей файловой системы и поддерживает определенную её организацию. Переключатель является единственным модулем, который может обращаться к драйверу файловой системы. Приложение не может обращаться к нему напрямую. Каждый драйвер файловой системы в процессе инициализации регистрируется у переключателя, передавая ему таблицу точек входа, которые будут использоваться при последующих обращениях.
Для выполнения своих функций драйверы файловых систем обращаются к подсистеме ввода вывода, образующей следующий слой файловой системы новой архитектуры. Подсистема ввода вывода является составной частью файловой системы, которая отвечает за загрузку, инициализацию и управление всеми модулями низших уровней



Большое число уровней архитектуры файловой системы обеспечивает авторам драйверов устройств большую гибкость - драйвер может получить управление на любом этапе выполнения запроса - от вызова приложением функции, которая занимается работой с файлами, до того момента, когда работающий на самом низком уровне драйвер устройства начинает просматривать регистры контроллера. Многоуровневый механизм работы файловой системы реализован посредством **цепочек вызова**.

В ходе инициализации драйвер устройства может добавить себя к цепочке вызова некоторого устройства, определив при этом уровень последующего обращения. Подсистема ввода-вывода помещает адрес целевой функции в **цепочку вызова** устройства, используя заданный уровень для того, чтобы должным образом упорядочить цепочку. По мере выполнения запроса, подсистема ввода-вывода последовательно вызывает все функции, ранее помещенные в цепочку вызова.

Внесенная в цепочку вызова процедура драйвера может решить передать запрос дальше - в измененном или в неизмененном виде - на следующий уровень, или, если это возможно, процедура может удовлетворить запрос, не передавая его дальше по цепочке.

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).

Архитектура распределенных файловых систем

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

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.

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

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.

Семантика разделения файлов

UNIX-семантика

Естественная семантика однопроцессорной ЭВМ - если за операцией записи следует чтение, то результат определяется последней из предшествующих операций записи. В распределенной системе такой семантики достичь легко только в том случае, когда имеется один файл-сервер, а клиенты не имеют кэшей. При наличии кэшей семантика нарушается. Надо либо сразу все изменения в кэшах отражать в файлах, либо менять семантику разделения файлов.

Еще одна проблема - трудно сохранить семантику общего указателя файла (в UNIX он общий для открывшего файл процесса и его дочерних процессов) - для процессов на разных ЭВМ трудно иметь общий указатель.

Неизменяемые файлы - очень радикальный подход к изменению семантики разделения файлов. Только две операции - создать и читать. Можно заменить новым файлом старый - т.е. можно менять директории. Если один процесс читает файл, а другой его подменяет, то можно позволить первому процессу доработать со старым файлом в то время, как другие процессы могут уже работать

Семантика сессий

Изменения открытого файла видны только тому процессу (или машине), который производит эти изменения, а лишь после закрытия файла становятся видны другим процессам (или машинам). Что происходит, если два процесса одновременно работали с одним файлом - либо результат будет определяться процессом, последним закрывшим файл, либо можно только утверждать, что один из двух вариантов файла станет текущим.

Транзакции

Процесс выдает операцию НАЧАЛО ТРАНЗАКЦИИ, сообщая тем самым, что последующие операции должны выполняться без вмешательства других процессов. Затем выдает последовательность чтений и записей, заканчивающуюся операцией КОНЕЦ ТРАНЗАКЦИИ. Если несколько транзакций стартуют в одно и то же время, то система гарантирует, что результат будет таким, каким бы он был в случае последовательного выполнения транзакций (в неопределенном порядке). Пример - банковские операции.

Реализация распределенных файловых систем

Приступая к реализации очень важно понимать, как система будет использоваться. Приведем результаты некоторых исследований использования файлов (статических и динамических) в университетах. Очень важно оценивать представительность исследуемых данных.

- большинство файлов имеют размер менее 10К (следует перекачивать целиком).
- чтение встречается гораздо чаще записи (кэширование).
- чтение и запись последовательны, произвольный доступ редок (упреждающее кэширование, чтение с запасом, выталкивание после записи следует группировать).
- большинство файлов имеют короткое время жизни (создавать файл в клиенте и держать его там до уничтожения).
- мало файлов разделяются (кэширование в клиенте и семантика сессий).
- существуют различные классы файлов с разными свойствами (следует иметь в системе разные механизмы для разных классов).

Структура системы

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.

Существует две конкурирующие точки зрения.
Первая состоит в том, что сервер не должен хранить такую информацию (**сервер stateless**) Другими словами, когда клиент посылает запрос на сервер, сервер его выполняет, отсылает ответ, а затем удаляет из своих внутренних таблиц всю информацию о запросе. Между запросами на сервере не хранится никакой текущей информации о состоянии клиента. Другая точка зрения состоит в том, что сервер должен хранить такую информацию (**сервер statefull**).
Рассмотрим эту проблему на примере файлового сервера, имеющего команды ОТКРЫТЬ, ПРОЧИТАТЬ, ЗАПИСАТЬ и ЗАКРЫТЬ файл. Открывая файлы, **statefull-сервер** должен запоминать, какие файлы открыл каждый пользователь. Таблица, отображающая дескрипторы файлов на сами файлы, является информацией о состоянии клиентов.
Для сервера **stateless** каждый запрос должен содержать исчерпывающую информацию (полное имя файла, смещение в файле и т.п.), необходимую серверу для выполнения требуемой операции. Очевидно, что эта информация увеличивает длину сообщения.

Серверы с состоянием. Достоинства.

- Короче сообщения (двоичные имена используют таблицу открытых файлов).
- выше эффективность (информация об открытых файлах может храниться в оперативной памяти).
- блоки информации могут читаться с упреждением.
- убедиться в достоверности запроса легче, если есть состояние (например, хранить номер последнего запроса).
- возможна операция захвата файла.

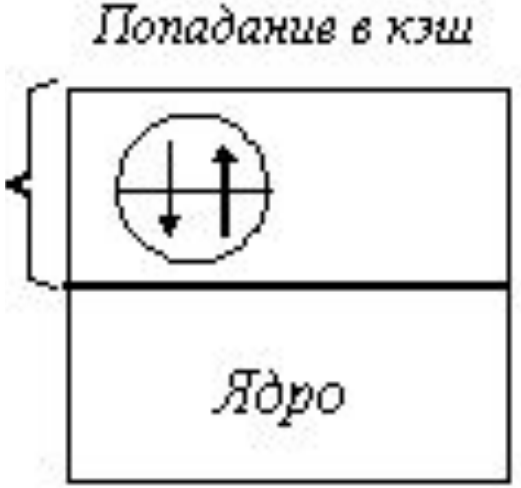
Серверы без состояния. Достоинства.

- устойчивость к ошибкам.
- не требуется операций ОТКРЫТЬ/ЗАКРЫТЬ.
- не требуется память для таблиц.
- нет ограничений на число открытых файлов.
- нет проблем при крахе клиента.

Кэширование

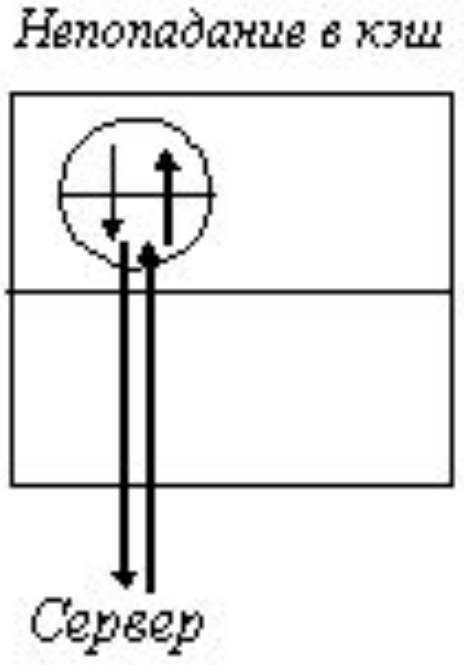
В системе клиент-сервер с памятью и дисками есть четыре потенциальных места для хранения файлов или их частей.
Во-первых, хранение файлов на дисках сервера. Нет проблемы когерентности, так как одна копия файла существует. Главная проблема - эффективность, поскольку для обмена с файлом требуется передача информации в обе стороны и обмен с диском.
Кэширование в памяти сервера. Две проблемы - помещать в кэш файлы целиком или блоки диска, и как осуществлять выталкивание из кэша.
Коммуникационные издержки остаются.
Избавиться от коммуникаций позволяет кэширование в машине клиента.
Кэширование на диске клиента может не дать преимуществ перед кэшированием в памяти сервера, а сложность повышается значительно.

Поэтому рассмотрим подробнее организацию кэширования в памяти клиента.

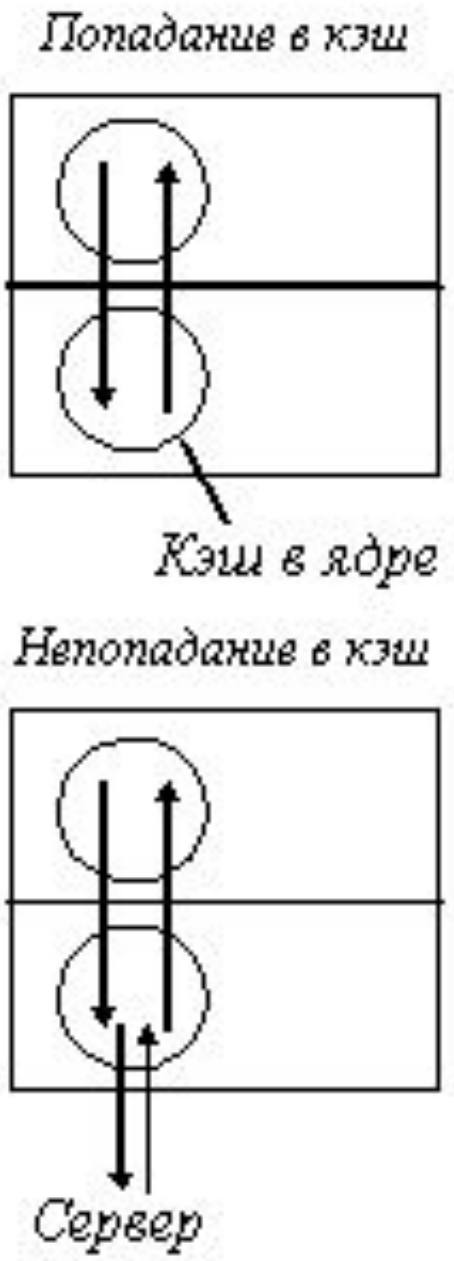


кэширование в каждом процессе.

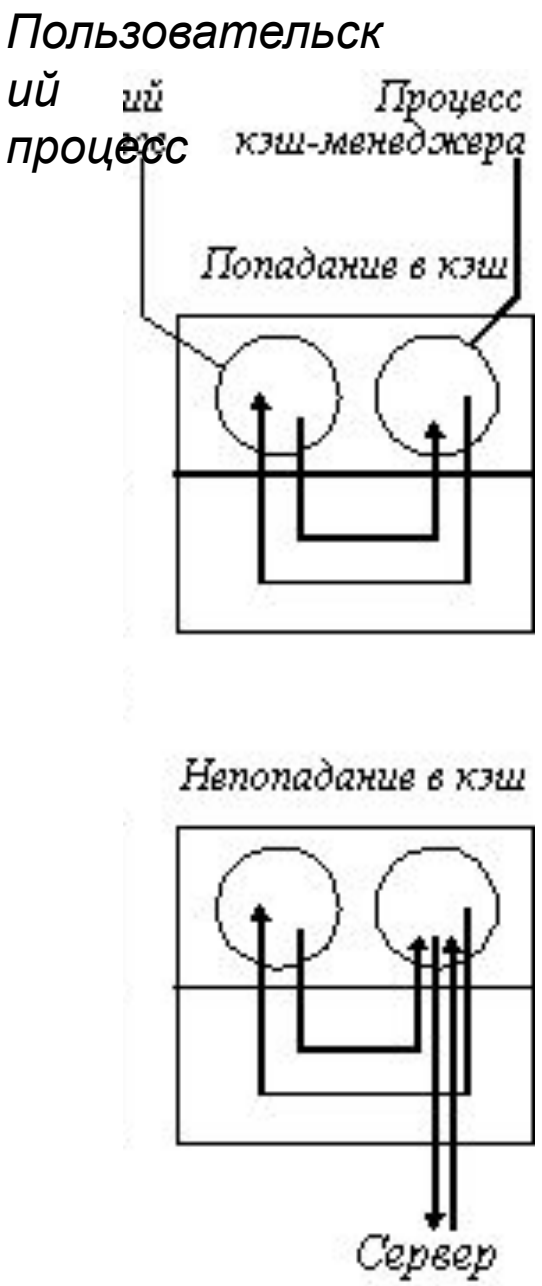
По мере того, как файлы открываются, закрываются, читаются и пишутся, библиотека просто сохраняет наиболее часто используемые файлы. Когда процесс завершается, все модифицированные файлы записываются назад на сервер. Хотя эта схема реализуется с чрезвычайно низкими издержками, она эффективна только тогда, когда отдельные процессы часто повторно открывают и закрывают файлы.



Кэширование в ядре. Недостатком этого варианта является то, что во всех случаях требуется выполнять системные вызовы, даже в случае успешного обращения к кэш-памяти (файл оказался в кэше). Но преимуществом является то, что файлы остаются в кэше и после завершения процессов. Например, предположим, что двухпроходный компилятор выполняется, как два процесса. Первый проход записывает промежуточный файл, который читается вторым проходом. На рисунке показано, что после завершения процесса первого прохода промежуточный файл, вероятно, будет находиться в кэше, так что вызов сервера не потребуется.



Кэш-менеджер в виде отдельного процесса. Ядро освобождается от функций файловой системы и оно может динамически решить, сколько памяти выделить для программ, а сколько для кэша. Когда же кэш-менеджер пользовательского уровня работает на машине с виртуальной памятью, то понятно, что ядро может решить выгрузить некоторые, или даже все страницы кэша на диск, так что для так называемого "попадания в кэш" требуется подкачка одной или более страниц. Однако, если в системе имеется возможность фиксировать некоторые страницы в памяти, то такая парадоксальная ситуация может быть исключена.



Когерентность кэшей.

Алгоритм со сквозной записью.
Необходимость проверки, не устарела ли информация в кэше. Запись вызывает коммуникационные расходы (MS-DOS).
Алгоритм с отложенной записью.
Через регулярные промежутки времени все модифицированные блоки пишутся в файл. Эффективность выше, но семантика непонятная пользователю (UNIX).
Алгоритм записи в файл при закрытии файла.
Реализует семантику сессий. Не намного хуже случая, когда два процесса на одной ЭВМ открывают файл, читают его, модифицируют в своей памяти и пишут назад в файл.
Алгоритм централизованного управления.
Можно выдержать семантику UNIX, но не эффективно, ненадежно, и плохо масштабируется.

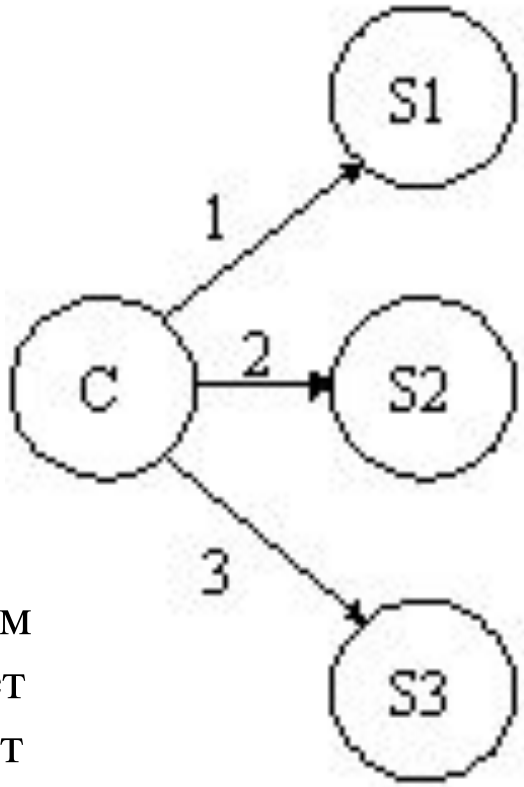
Размножение

Система может предоставлять такой сервис, как поддержание для указанных файлов нескольких копий на различных серверах. Репликация - это асинхронный перенос изменений данных исходной файловой системы в файловые системы, принадлежащие различным узлам распределенной файловой системы. Другими словами, система оперирует несколькими копиями файлов, причем каждая копия находится на отдельном файловом сервере.

Главные цели:

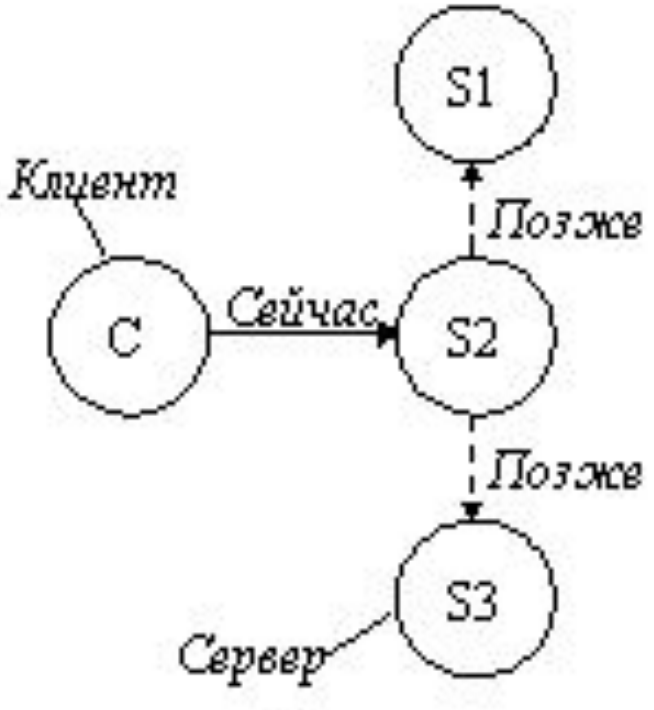
- Повысить надежность.
- Повысить доступность (крах одного сервера не вызывает недоступность размноженных файлов).
- Распределить нагрузку на несколько серверов.

Явное размножение (непрозрачно). В ответ на открытие файла пользователю выдаются несколько двоичных имен, которые он должен использовать для явного дублирования операций с файлами.

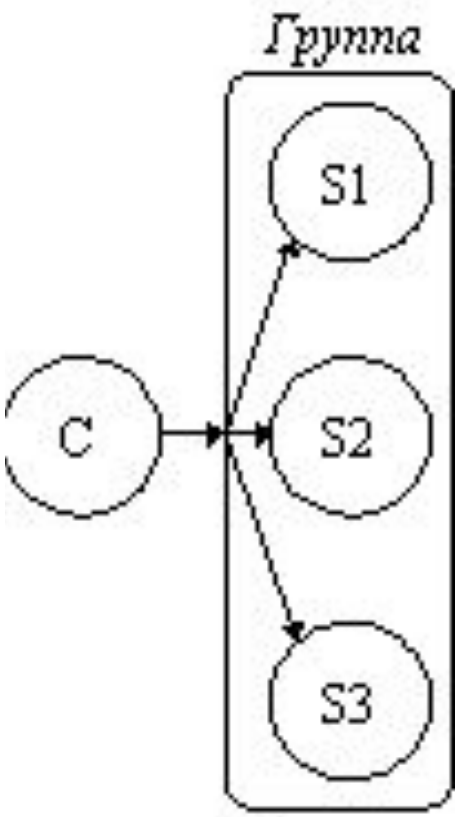


Программист сам управляет всем процессом репликации. Когда процесс создает файл, он делает это на одном определенном сервере. Затем он может сделать дополнительные копии на других серверах. Если сервер каталогов разрешает сделать несколько копий файла, то сетевые адреса всех копий могут быть ассоциированы с именем файла, как показано на рисунке снизу, и когда имя найдено, это означает, что найдены все копии.

Ленивое размножение. Одна копия создается на одном сервере, а затем он сам автоматически создает (в свободное время) дополнительные копии и обеспечивает их поддержание.



Симметричное размножение. Все операции одновременно вызываются в нескольких серверах и одновременно выполняются



В этом методе все системные вызовы ЗАПИСАТЬ передаются одновременно на все серверы, таким образом копии создаются одновременно с созданием оригинала. Имеется два принципиальных различия в использовании групповых связей и ленивой репликации. Во-первых, при ленивой репликации адресуется один сервер, а не группа. Во-вторых, ленивая репликация происходит в фоновом режиме, когда сервер имеет промежуток свободного времени, а при групповой репликации все копии создаются в одно и то же время.

Протоколы коррекции

Рассмотрим, как могут быть изменены существующие реплицированные файлы. Существует два хорошо известных алгоритма решения этой проблемы.

1. Метод размножения главной копии. Один сервер объявляется главным, а остальные - подчиненными. Все изменения файла посылаются главному серверу. Он сначала корректирует свою локальную копию, а затем рассылает подчиненным серверам указания о коррекции. Чтение файла может выполнять любой сервер. Для защиты от краха главного сервера до завершения всех коррекций, до выполнения коррекции главной копии главный сервер запоминает в стабильной памяти задание на коррекцию. Слабость - выход из строя главного сервера не позволяет выполнять коррекции.

2. Метод голосования. Идея - запрашивать чтение и запись файла у многих серверов (запись - у всех!). Запрос может получить одобрение у половины серверов плюс один. При этом должно быть согласие относительно номера текущей версии файла. Этот номер увеличивается на единицу с каждой коррекцией файла. Можно использовать различные значения для кворума чтения (Nr) и кворума записи (Nw). При этом должно выполняться соотношение $Nr+Nw>N$. Поскольку чтение является более частой операцией, то естественно взять $Nr=1$. Однако в этом случае для кворума записи потребуются все серверы.