ЮУрГУ

Международный факультет

Кафедра международного менеджмента

Дисциплина:

Операционные системы

Тема 4
Файловые системы

Лекция 1
Общие представления о файловой системе

1. Общие сведения
2. Организация файлов и доступ к ним
3. Операции над файлами
4. Директории. Логическая структура файлового архива
5. Защита файлов
6. Интерфейс файловой системы Windows

1. Общие сведения

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.

Файл, обычно представляет собой неструктурированную последовательность однобайтовых записей, хранится в виде последовательности блоков (не обязательно смежных); каждый блок хранит целое число записей. Иногда говорят, что файл – это поименованный набор связанной информации, записанной во вторичную память.

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.

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, 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.

Имена файлов

Правила именования файлов зависят от ОС. Многие ОС поддерживают имена из двух частей (имя + расширение). Тип расширения файла позволяет ОС организовать работу с ним различных прикладных программ в соответствии с заранее оговоренными соглашениями.

Например progr.c (файл, содержащий текст программы на языке Си) или autoexec.bat (файл, содержащий команды интерпретатора командного языка). 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).

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 или других структурах, обеспечивающих доступ к данным файла.

Некоторые из возможных атрибутов файлов

Атрибут	Значение
Защита	Кто и каким образом может получить доступ к файлу
Пароль	Пароль для получения доступа к файлу
Создатель	Идентификатор создателя файла
Владелец	Текущий владелец
Флаг «только для чтения»	0 — для чтения-записи; 1 — только для чтения
Флаг «скрытый»	0 — обычный; 1 — не предназначенный для отображения в перечне файлов
Флаг «системный»	0 — обычный; 1 — системный
Флаг «архивный»	0 — прошедший резервное копирование; 1 — нуждающийся в резервном копировании
Флаг ASCII-двоичный	0 — ASCII; 1 — двоичный
Флаг произвольного доступа	0 — только последовательный доступ; 1 — произвольный доступ
Флаг «временный»	0 — обычный; 1 — удаляемый по окончании работы процесса
Флаги блокировки	0 — незаблокированный; ненулевое значение — заблокированный
Длина записи	Количество байтов в записи
Позиция ключа	Смещение ключа внутри каждой записи
Длина ключа	Количество байтов в поле ключа
Время создания	Дата и время создания файла
Время последнего доступа	Дата и время последнего доступа к файлу
Время внесения последних изменений	Дата и время внесения в файл последних изменений
Текущий размер	Количество байтов в файле
Максимальный размер	Количество байтов, до которого файл может увеличиваться

Типы файлов

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).

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 или UNICODE), так и произвольную двоичную (бинарную) информацию.

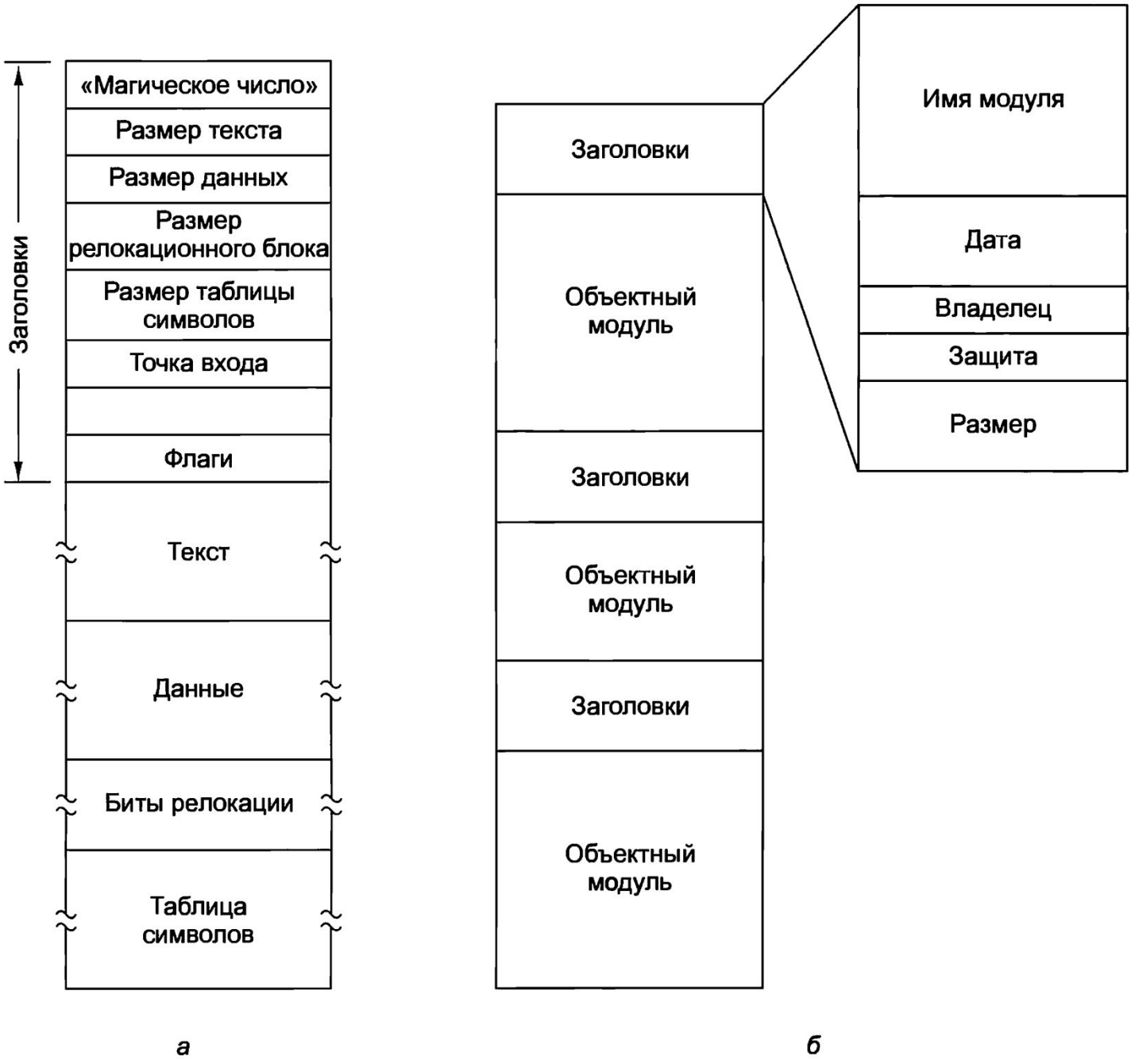
Текстовые файлы содержат символьные строки, которые можно распечатать, увидеть на экране или редактировать обычным текстовым редактором.

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. Та часть имени файла, которая следует за точкой, называется **расширением имени файла** и, как правило, несет в себе некоторую информацию о файле. Например, файлы с расширениями .c, .pas, .txt — ASCII-файлы (или UNICODE-файлы), файлы с расширениями .exe — выполнимые, файлы с расширениями .obj, . zip — бинарные и т. д.

Некоторые типичные расширения файлов файлов

Расширение	Значение
file.bak	Резервная копия файла
file.c	Исходный текст программы на языке C
file.gif	Изображение формата gif
file.hlp	Файл справки
file.html	Документ в формате HTML
file.jpg	Статическое растровое изображение в формате JPEG
file.mp3	Музыка в аудиоформате MPEG layer 3
file.mpg	Фильм в формате MPEG
file.o	Объектный файл (полученный на выходе компилятора, но еще не прошедший компоновку)
file.pdf	Документ формата PDF
file.ps	Документ формата PostScript
file.tex	Входной файл для программы форматирования TEX
file.txt	Обычный текстовый файл
file.zip	Архив, сжатый программой zip



Примеры структур двоичных файлов: исполняемый файл (а), архив *(б)*

2. Организация файлов и доступ к ним

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.

ОС поддерживают несколько вариантов структуризации файлов.

Последовательный файл

Простейший вариант - 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*. Имеется в виду относительный номер, специфицирующий данный блок среди блоков диска, принадлежащих файлу. 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.*

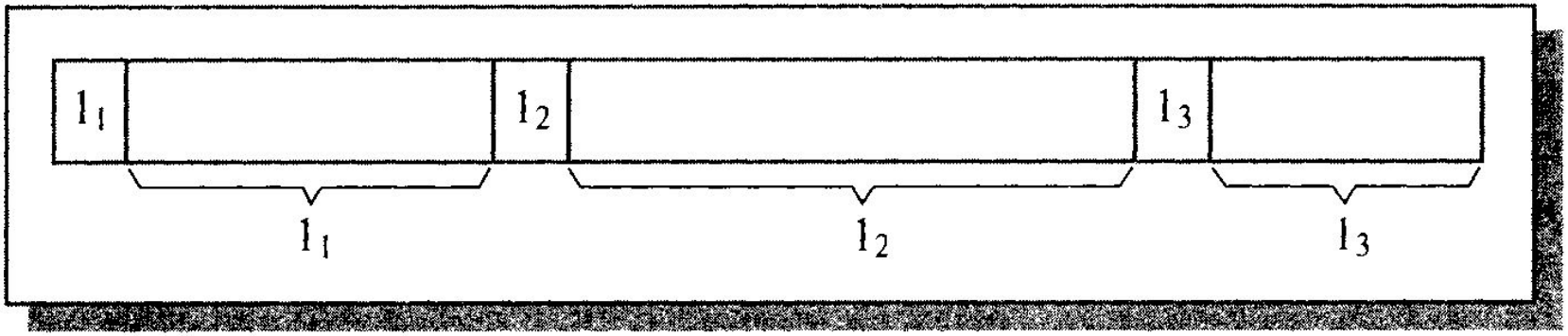
Файл, состоящий из однобайтовых записей на устройстве прямого доступа, — наиболее распространенный способ организации файла. **Базовыми операциями для такого рода файлов** являются считывание или запись символа в текущую позицию. В большинстве языков высокого уровня предусмотрены операторы посимвольной 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.

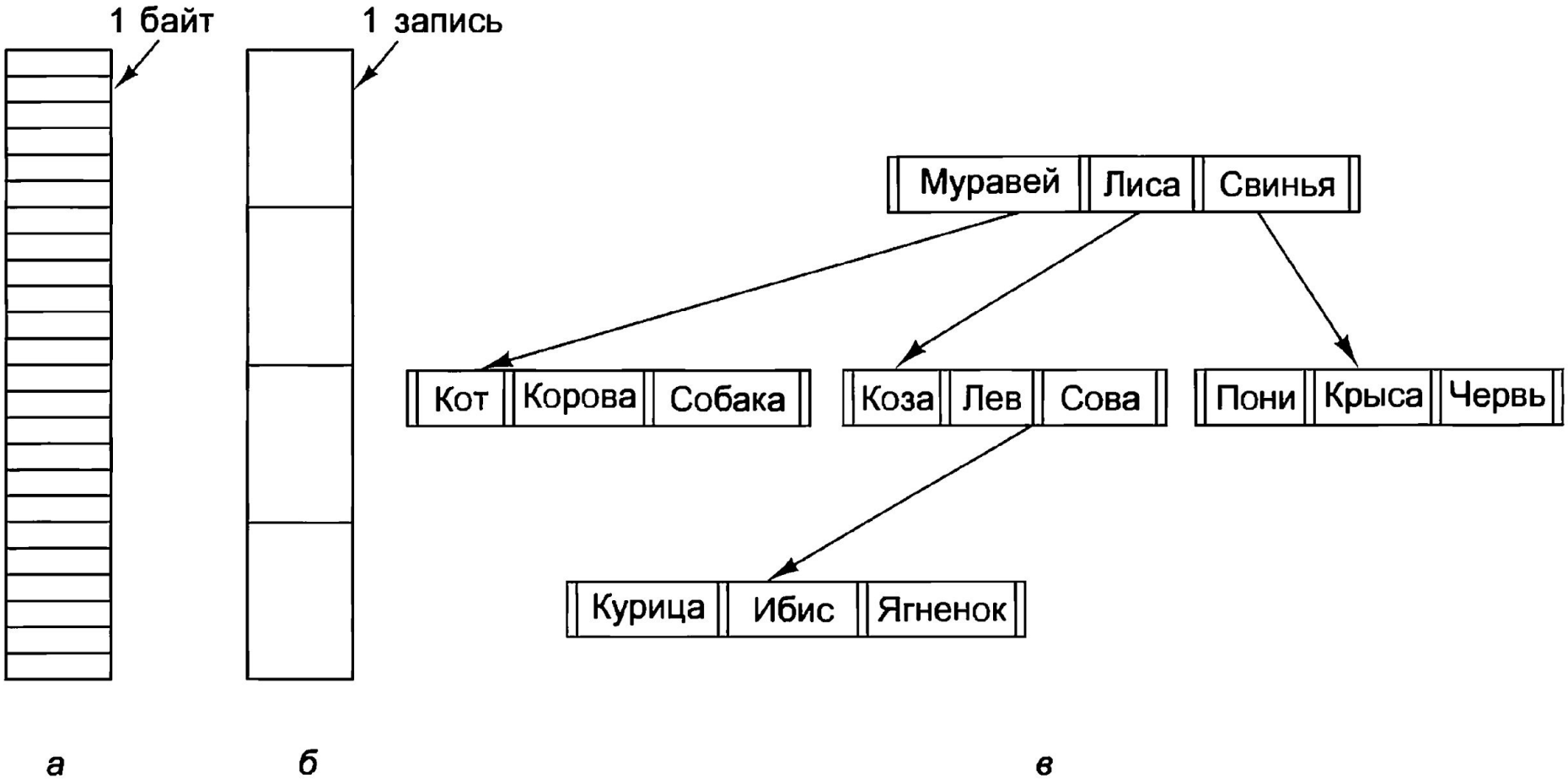
Другие формы организации файлов

1. 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.

2. П*оследовательность записей переменной длины,* каждая из которых содержит ключевое поле в фиксированной позиции внутри записи. 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.*



Файл как последовательность записей переменной длины



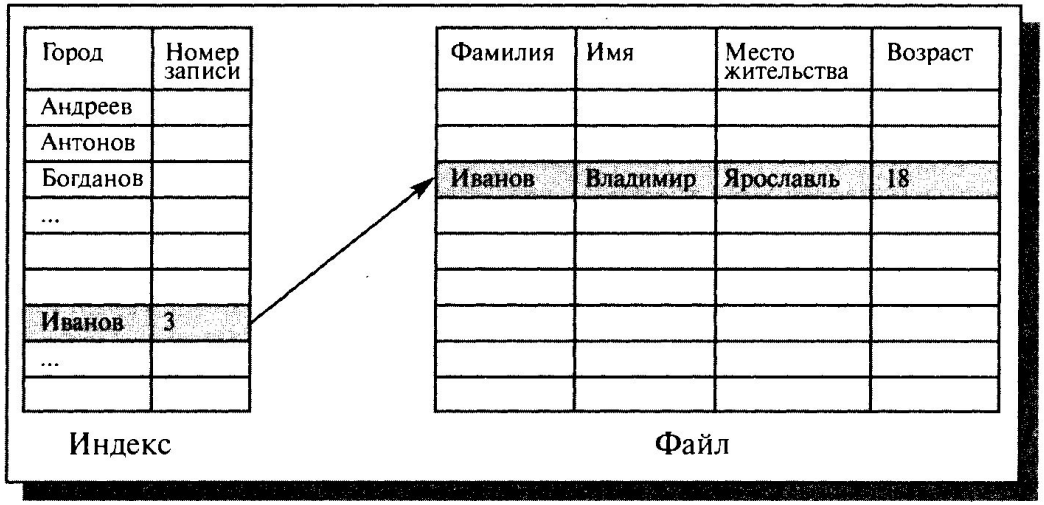
Три типа файлов: последовательность байтов (а); последовательность записей (б); дерево (в)

Другие формы организации файлов

3. Конструирование *индекса* файла для ускорения доступа к файлу. 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.*

Пример

Пусть 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. Рисунок иллюстрирует решение данной проблемы — организацию метода доступа к файлу с использованием индекса.



Пример организации индекса для последовательного файла

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), 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).

3. Операции над файлами

Основные файловые операции ОС Unix:

- **Создание файла, не содержащего данных**. Смысл данного вызова — объявить, что файл существует, и присвоить ему ряд атрибутов. При этом выделяется место для файла на диске и вносится запись в каталог.
- **Удаление файла** и освобождение занимаемого им дискового пространства.
- **Открытие файла**. Перед использованием файла процесс должен его открыть. Цель данного системного вызова — разрешить системе проанализировать атрибуты файла и проверить права доступа к нему, а также считать в оперативную память список адресов блоков файла для быстрого доступа к его данным. Открытие файла является процедурой создания *дескриптора* или управляющего блока файла. Дескриптор (описатель) файла хранит всю информацию о нем. Иногда, в соответствии с парадигмой, принятой в языках программирования, под дескриптором понимается альтернативное имя файла или указатель на описание файла в таблице открытых файлов, используемый при последующей работе с файлом. Например, на языке Си операция открытия файла fd=open(pathname, flags, modes) ; возвращает дескриптор fd, который может быть задействован при выполнении операций чтения (read (fd, buffer, count) ;) или записи.
- **Закрытие файла**. Если работа с файлом завершена, его атрибуты и адреса блоков на диске больше не нужны. В этом случае файл нужно закрыть, чтобы освободить место во внутренних таблицах файловой системы.
- **Позиционирование**. Дает возможность специфицировать место внутри файла, откуда будет производиться считывание (или запись) данных, то есть задать *текущую* позицию.
- **Чтение данных из файла**. Обычно это делается с текущей позиции. Пользователь должен задать объем считываемых данных и предоставить для них буфер в оперативной памяти.
- **Запись данных в файл с текущей позиции**. Если текущая позиция находится в конце файла, его размер увеличивается, в противном случае запись осуществляется на место имеющихся данных, которые, таким образом, теряются.

Другие операции: **переименование файла**, **получение атрибутов файла** и т. д.

Операции с файлами реализуются через системные вызовы.

Существует два способа выполнить последовательность действий над файлами.

1. Для каждой операции выполняются как *универсальные*, так и *уникальные* действия (схема stateless). Например, последовательность операций может быть такой: open, read1, close, ... open, read2, close,... open, read3, close.

2. *Универсальные* действия выполняются в начале и в конце последовательности операций, а для каждой промежуточной операции выполняются только *уникальные* действия. В этом случае последовательность вышеприведенных операций будет выглядеть так: open, read1, ... read2, ... read3, close.

Большинство ОС использует второй способ, более экономичный и быстрый. Первый способ более устойчив к сбоям, поскольку результаты каждой операции становятся независимыми от результатов предыдущей операции; поэтому он иногда применяется в распределенных файловых системах (например, Sun NFS).

4. Директории. Логическая структура файлового архива

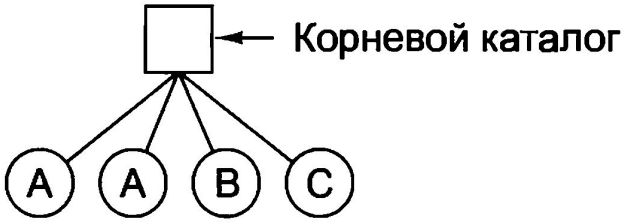
Все современные файловые системы поддерживают многоуровневое именование файлов за счет наличия во внешней памяти дополнительных файлов со специальной структурой — каталогов (или директорий).

Каждый каталог содержит список каталогов и/или файлов, содержащихся в данном каталоге. Каталоги имеют один и тот же внутренний формат, где каждому файлу соответствует одна запись в файле директории.

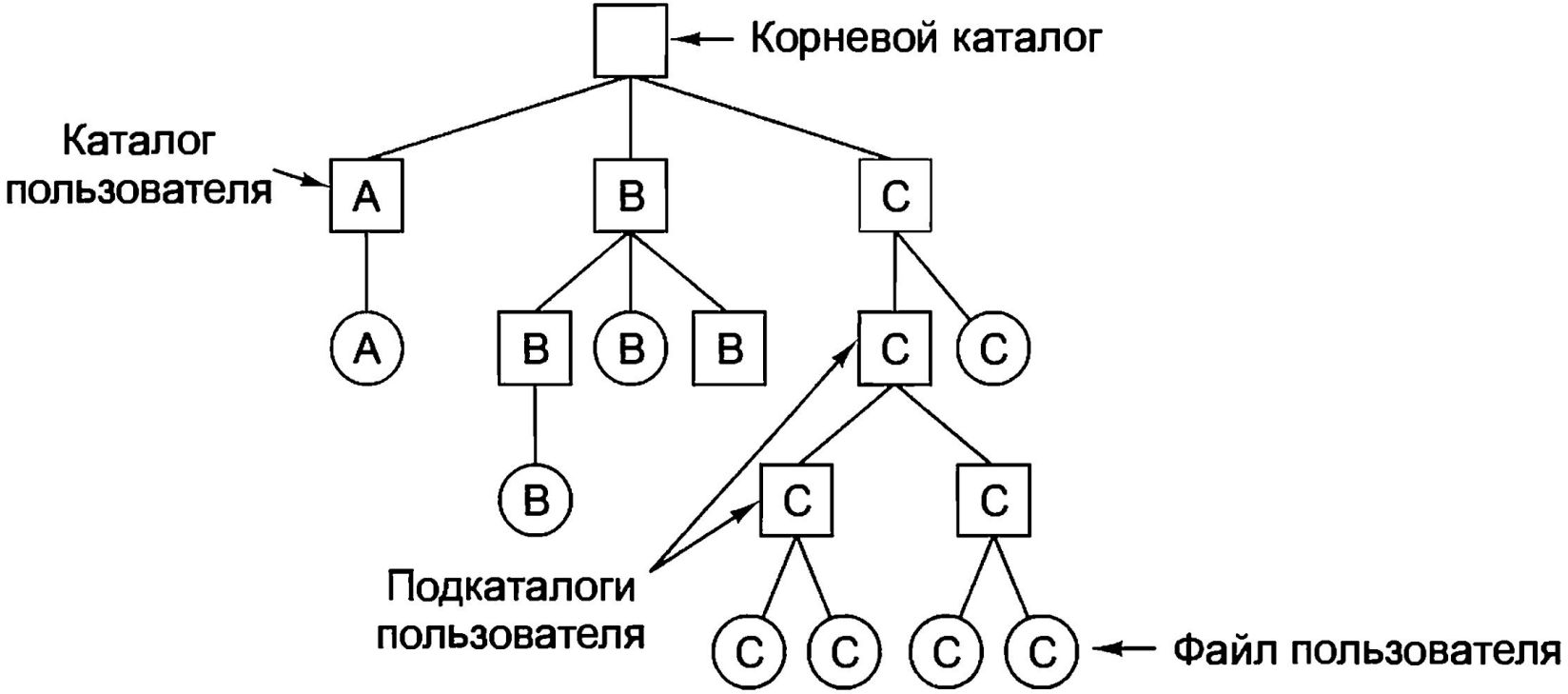
Число директорий зависит от системы. В ранних ОС имелась только одна корневая директория, затем появились директории для пользователей (по одной директории на пользователя). В современных ОС используется произвольная структура дерева директорий.

Имя файла (каталога)	Тип файла (обычный или каталог)	
Anti	K	атрибуты
Games	K	атрибуты
Autoexec.bat	O	атрибуты
mouse.com	O	атрибуты

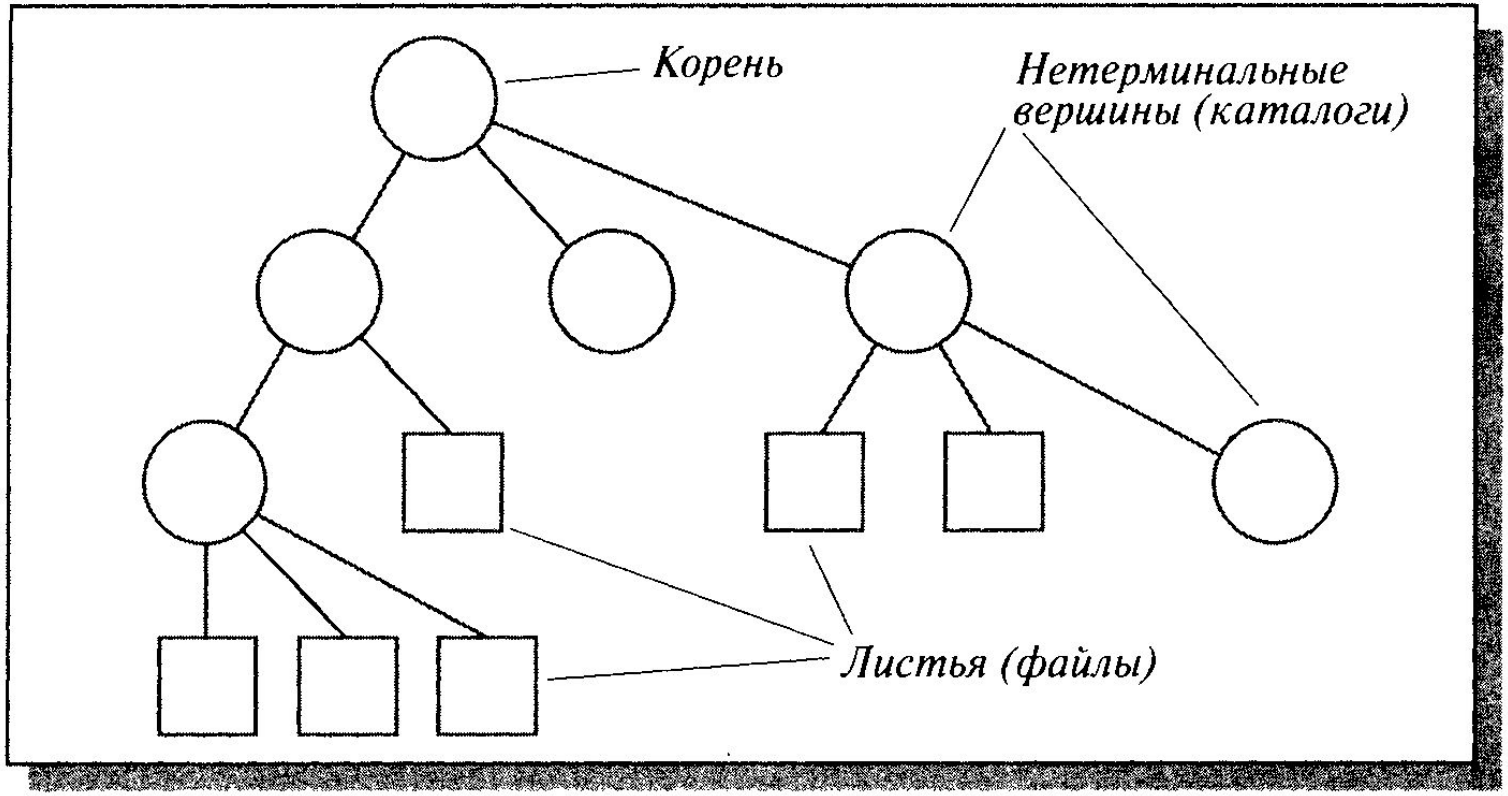
Пример директорий



Система с одноуровневым каталогом, содержащим четыре файла



Иерархическая система каталогов



Иерархическая древовидная структура файловой системы (перевернутое дерево)

Верхняя вершина называется корнем. Если элемент дерева не может иметь потомков, он называется терминальной вершиной или листом. Нелистовые (нетерминальные) вершины – справочники или каталоги – содержат списки листовых и нелистовых вершин. Путь от корня к файлу *однозначно* определяет файл.

Подобные древовидные структуры являются графами, не имеющими циклов. Можно считать, что ребра графа направлены вниз, а корень - вершина, не имеющая входящих ребер.

Внутри одного каталога имена листовых файлов уникальны. Имена файлов, находящихся в разных каталогах, могут совпадать. Для того чтобы однозначно определить файл по его имени (избежать коллизии имен), принято именовать файл так называемым *абсолютным или полным именем (pathname)*, состоящим из списка имен вложенных каталогов, по которому можно найти путь от корня к файлу плюс имя файла в каталоге, непосредственно содержащем данный файл. То есть полное имя включает цепочку имен — путь к файлу, например /usr/games/doom. Такие имена уникальны. Компоненты пути разделяют различными символами: «/» (слэш) в Unix или обратными слэшем «\» в MS-DOS и Windows (в Multics — «>»). Таким образом, использование древовидных каталогов минимизирует сложность назначения уникальных имен.

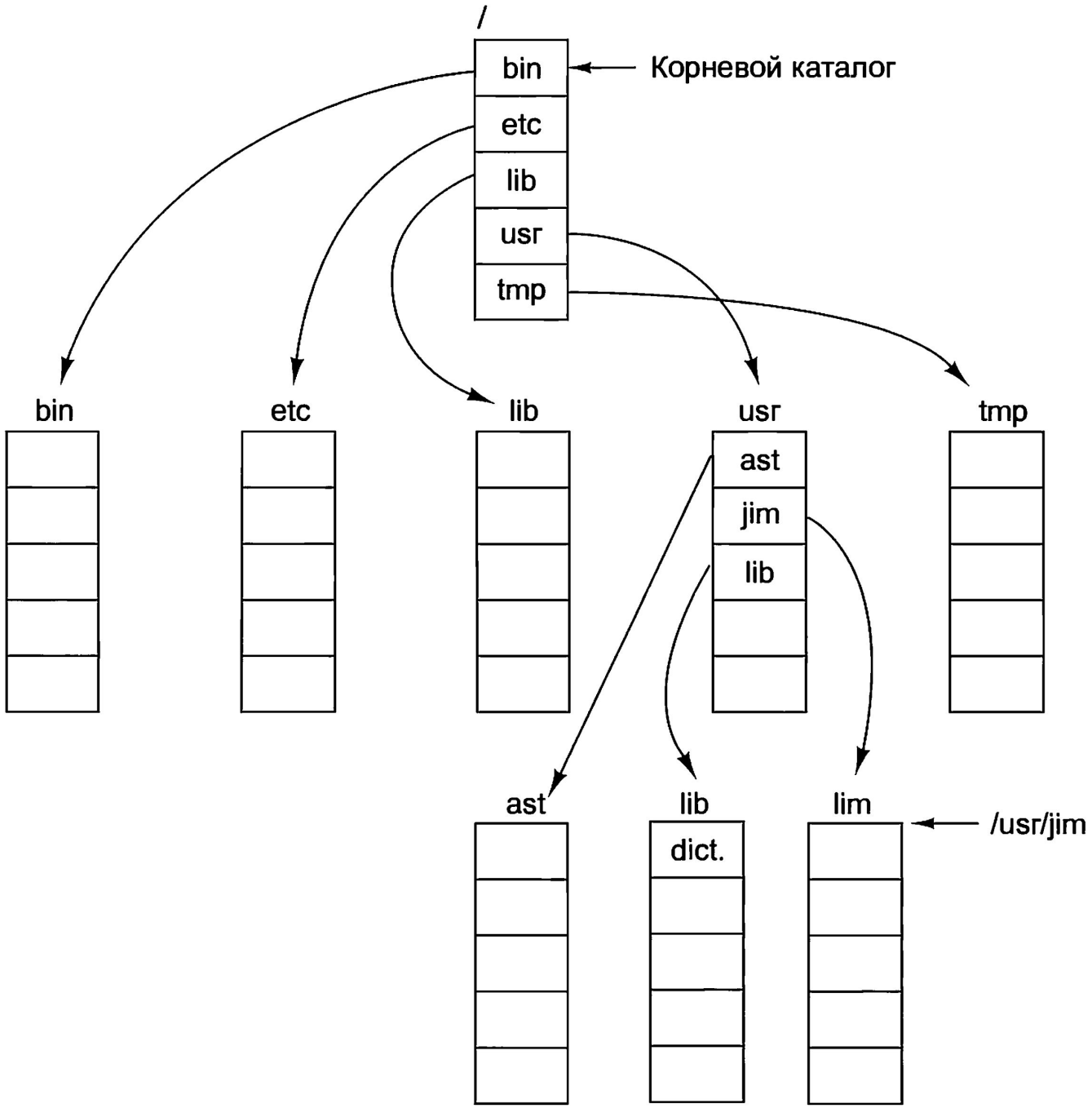
Указывать полное имя не всегда удобно, поэтому применяют другой способ задания имени — относительный путь к файлу. Он использует концепцию рабочей или текущей директории, которая обычно входит в состав атрибутов процесса, работающего с данным файлом. Тогда на файлы в такой директории можно ссылаться только по имени, при этом поиск файла будет осуществляться в рабочем каталоге. Это удобнее, но, по существу, то же самое, что и абсолютная форма.

Для получения доступа к файлу и локализации его блоков система должна выполнить навигацию по каталогам.

Пример: путь /usr/linux/progr.c. Алгоритм одинаков для всех иерархических систем. Сначала в фиксированном месте на диске находится корневая директория. Затем находится компонент пути usr, т. е. в корневой директории ищется файл /usr. Исследуя этот файл, система понимает, что данный файл является каталогом, и блоки его данных рассматривает как список файлов и ищет следующий компонент linux в нем. Из строки для linux находится файл, соответствующий компоненту usr/linux/. Затем находится компонент progr.c, который открывается, заносится в таблицу открытых файлов и сохраняется в ней до закрытия файла.

Отклонение от типовой обработки компонентов pathname может возникнуть в том случае, когда этот компонент является не обычным каталогом с соответствующим ему индексным узлом и списком файлов, а служит точкой связывания (принято говорить «точкой монтирования») двух файловых архивов.

Многие прикладные программы работают с файлами, находящимися в текущей директории, не указывая явным образом ее имени. Это дает пользователю возможность произвольным образом именовать каталоги, содержащие различные программные пакеты. Для реализации этой возможности в большинстве ОС, поддерживающих иерархическую структуру директорий, используется обозначение «.» — для текущей директории и «..» —для родительской.



Дерево каталогов UNIX

Разделы диска. Организация доступа к архиву файлов

Задание пути к файлу в файловых системах некоторых ОС отличается тем, с чего начинается эта цепочка имен. В современных ОС принято разбивать диски на логические диски (это низкоуровневая операция), иногда называем*ые* разделами (partitions). Бывает, что, наоборот, объединяют несколько физических дисков в один логический диск (например, это можно сделать в ОС Windows NT и основанных на ней ОС). Поэтому можно игнорировать проблему физического выделения пространства для файлов и считать, что каждый раздел представляет собой отдельный (виртуальный) диск. Диск содержит иерархическую древовидную структуру, состоящую из набора файлов, каждый из которых является хранилищем данных пользователя, и каталогов или директорий (то есть файлов, которые содержат перечень других файлов, входящих в состав каталога), необходимых для хранения информации о файлах системы.

1. В некоторых системах управления файлами требуется, чтобы каждый архив файлов целиком располагался на одном диске (разделе диска). В этом случае полное имя файла начинается с имени дискового устройства, на котором установлен соответствующий диск (буквы диска). Например, c:\util\nu\ndd.exe. Такой способ именования используется в файловых системах DEC и Microsoft.

2. В других системах (Multics) вся совокупность файлов и каталогов представляет собой единое дерево. Сама система, выполняя поиск файлов по имени, начиная с корня, требовала установки необходимых дисков.

3. В ОС Unix предполагается наличие нескольких архивов файлов, каждый на своем разделе, один из которых считается корневым. После запуска системы можно *«смонтировать»* корневую файловую систему и ряд изолированных файловых систем в одну общую файловую систему. Технически это осуществляется с помощью создания в корневой файловой системе специальных пустых каталогов. Специальный системный вызов mount ОС Unix позволяет подключить к одному из этих пустых каталогов корневой каталог указанного архива файлов. После монтирования общей файловой системы именование файлов производится так же, как если бы она с самого начала была централизованной. Задачей ОС является беспрепятственный проход точки монтирования при получении доступа к файлу по цепочке имен. Если учесть, что обычно монтирование файловой системы производится при загрузке системы, пользователи ОС Unix обычно и не задумываются о происхождении общей файловой системы.

Операции над директориями

Как и в случае с файлами, операции, необходимые для работы с директориями, реализуются через системные вызовы. Несмотря на то что директории — это файлы, логика работы с ними отличается от логики работы с обычными файлами и определяется природой этих объектов, предназначенных для поддержки структуры файлового архива. Совокупность системных вызовов для управления директориями зависит от особенностей конкретной ОС. Операции над каталогами являются прерогативой ОС, то есть пользователь не может, например, выполнить запись в каталог начиная с текущей позиции.

Некоторые системные вызовы, необходимые для работы с каталогами:

- **Создание директории**. Вновь созданная директория включает записи с именами '.' и '..', однако считается пустой.
- **Удаление директории**. Удалена может быть только пустая директория.
- **Открытие директории** для последующего чтения. Например, чтобы перечислить файлы, входящие в директорию, процесс должен открыть директорию и считать имена всех файлов, которые она включает.
- **Закрытие директории** после ее чтения для освобождения места во внутренних системных таблицах.
- **Поиск**. Данный системный вызов возвращает содержимое текущей записи в открытой директории. Для этих целей может использоваться системный вызов read, но в этом случае от программиста потребуется знание внутренней структуры директории.
- **Получение списка файлов в каталоге**.
- **Переименование**. Имена директорий можно менять, как и имена файлов.
- **Создание файла**. При создании нового файла необходимо добавить в каталог соответствующий элемент.
- **Удаление файла**. Удаление из каталога соответствующего элемента. Если удаляемый файл присутствует только в одной директории, то он вообще удаляется из файловой системы, в противном случае система ограничивается только удалением специфицируемой записи.

Создание и удаление файлов предполагает также выполнение соответствующих файловых операций.

Имеются и другие системные вызовы, например, связанные с защитой информации.

5. Защита файлов

Информация в компьютерной системе должна быть защищена как от физического разрушения (reliability), так и от несанкционированного доступа (protection).

Контроль доступа к файлам

Наличие в системе многих пользователей предполагает организацию контролируемого доступа к файлам. Выполнение любой операции над файлом должно быть разрешено только в случае наличия у пользователя соответствующих привилегий. Обычно контролируются следующие операции: чтение, запись и выполнение. Другие операции, например копирование файлов или их переименование, также могут контролироваться. Однако они чаще реализуются через перечисленные. Так, операцию копирования файлов можно представить как операцию чтения и последующую операцию записи.

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 натри 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).

6. Интерфейс файловой системы Windows

Особенности файловой системы Windows

Длина имени файла в Windows соответствует стандарту POSIX (255) и определяется параметром MAX_PATH, значение которого равно 260, но система позволяет преодолеть это ограничение и использовать имена файлов длиной до 32000 символов в формате UNICODE.

Имеется возможность различать строчные и прописные буквы в названии файла, но пользоваться этой возможностью не рекомендуется, поскольку многие приложения и поисковые программы эту возможность не учитывают, поэтому для них данный файл может быть недоступен.

Связь имен файлов с обрабатывающими программами реализована в реестре.

Иногда к файлам приписывают другие объекты ОС, такие как: физические и логические диски, последовательные и параллельные порты, каналы и др., используя абстракцию файла.

Указатель текущей позиции в файле – 64-разрядное число, для задания которого обычно используется два 32-разрядных.

Пользователю предоставляется возможность осуществлять асинхронные операции ввода-вывода наряду с традиционными синхронными.

Имена логических дисков хранятся в каталоге «\??» пространства имен объектов, которые и осуществляют связь логического диска и реального устройства.

Система контроля доступа предполагает наличие у каждого файла *дескриптора защиты*, содержащего список прав доступа, который формирует владелец файла. Каждый процесс имеет *маркер доступа*, который содержит права пользователя, запустившего процесс. Система контроля доступа в момент открытия файла проверяет соответствие прав владельца процесса с теми, которые перечислены в списке прав доступа к файлу. В результате доступ может быть разрешен или отклонен.

Во время выполнения операций чтения, записи и других, проверки прав доступа уже не производятся.

В новых версиях NTFS дескрипторы защиты всех файлов хранятся в отдельном файле метаданных \$Secure, который описывается 9-й записью главной файловой таблицы тома MFT (консолидированная защита).

Атрибуты файлов в Windows

Считается, что файл – это не просто последовательность байтов, а совокупность атрибутов, и данные файла являются лишь одним из атрибутов – так называемым *неименованным потоком данных*. Есть *именованные потоки данных*, которые указываются через двоеточие.

Основные атрибуты файлов в файловой системе NTFS:

- стандартная информация (флаговые биты: только чтение, архивный, временные штампы и т. д.);
- имя файла (кодировка Unicode, может повторяться в MS-DOS);
- описатель защиты;
- данные (именованный и неименованные потоки данных);
- список атрибутов (расположение дополнительных записей MFT, если одной записи о файле оказалось недостаточно);
- идентификатор объекта (64-разрядный идентификатор файла, уникальный для данного тома);
- информация о точке повторного разбора, которая используется для символьных ссылок и монтирования устройств;
- информация о томе;
- информация об индексировании, используемая для каталогов;
- данные EFS (Encryption File System), используемые для шифрования.

Литература:

1. Таненбаум Э. Современные операционные системы. 3-е изд. с. 304…325.
2. Столингс В. Операционные системы. 4-е изд. – М.: Издательский дом «Вильямс», 2002, с. 603…646.
3. Основы операционных систем. Курс лекций. Учебное пособие / В. Е. Карпов, К. А. Коньков / под редакцией В. П. Иванникова. – М.: ИНТУИТ.РУ «Интернет-Университет Информационных Технологий», 2005, с. 169…184.
4. Устройство и функционирование ОС Windows. Практикум к курсу «Операционные системы»: учебное пособие/ К. А. Коньков. – М.: Интернет-Университет Информационных Технологий ; БИНОМ. Лаборатория знаний, 2008, с. 134…146.