ОС

8 сентября 2014 г.

10:11

Михелев Владимир Михайлович

ОС -- это ПО, которое с одной стороны выступает как интерфейс между аппаратурой компьютера и пользователем с его задачей, а с другой стороны предназначено для организации эффективного использования ресурсов этого компьютера и организации надёжных вычислений.

Ни один из компонентов ПО не может получить доступ к аппаратуре, только к ОС, которая изолирует его от аппаратуры.

Поколения ЭВМ:

* 1. Середина 40-х, ламповые.
  2. 55-65-е годы, перфоленты. Появились первые системы пакетной обработки данных.
  3. 65-80-е годы, интегральные микросхемы (OS/360). Базовые механизмы ОС сформулированные в то время:
     1. мультипрограммирование -- возможность одновременного выполнения нескольких программ;
     2. мультипроцессирование -- поддержка нескольких процессоров;
     3. поддержка многотерминального, многопользовательского режима работы;
     4. виртуальная память;
     5. файловая система;
     6. разграничение доступа;
     7. сетевая работа.

Были разработаны также две системы: CTSS, Multics.

Примерно 1980-ый год. Появился UNIX.

* 1. 80-е годы. Большие интегральные схемы. Персональные компьютеры (IBM PC с процессором Intel 8080, 256 Кб оперативной памяти, 160 Кб жесткой памяти (дискеты), черно-белый алфавитно-цифровой монитор). Первая ОС: CP/M.

# Классификация ОС

По количеству пользователей:

* + однопользовательские (MS-DOS);
  + многопользовательские.

По типу организации (доступу):

* + пакетные (MS-DOS);
  + распределения времени (UNIX, Windows);
  + реального времени (QNX).

По количеству решаемых задач:

* + однозадачные (MS-DOS);
  + многозадачные.

По виду интерфейса:

* + символьный;
  + графический.

По назначению:

* + общего;
  + специального;
  + сетевые;

# Функции ОС

* + Представление пользователю вместо реальной аппаратуры компьютера некой расширенной виртуальной машины, с которой удобнее работать и в которой удобнее программировать.
  + Контроль и управление ресурсами компьютера.

# ОС как виртуальная машина

Реальный компьютер может выполнять ограниченный набор команд, а благодаря ОС он превращается в некую виртуальную машину, благодаря чему может выполнять более широкий набор действий.

# ОС как система управления ресурсами

Ресурсы -- совокупность различных составляющих компьютера (оперативная память, процессор, таймер, наборы данных и пр.).

Процесс -- базовое понятие ОС. Программа в стадии выполнения. Это динамический объект, который возникает тогда, когда пользователь запускает программу на выполнение.

Программа -- статический объект, представляющий собой файл с кодами и данными.

ОС распределяет ресурсы между процессами и должна управлять этими ресурсами с целью их наиболее эффективного использования.

Когда ОС управляет ресурсами, решаются следующие задачи:

* + Планирование ресурса -- ранжирование.
  + Удовлетворение запросов на эти ресурсы.
  + Отслеживание состояния ресурсов с учетом их использования.
  + Разрешение конфликтов между процессами.

# Функциональные компоненты ОС

Наиболее важными компонентами являются:

* + подсистема управления процессами;
  + ... памятью;
  + ... файлами и внешними устройствами;
  + подсистема пользовательского интерфейса;
  + защита данных и администрирование.

# Управление процессами

Для каждого вновь создаваемого процесса ОС генерирует специально создаваемую структуру, называемую контекстом процесса, в которой содержатся данные о потребностях процесса в ресурсах.

Чтобы процесс был выполнен ОС должна назначить этому процессу область ОП, в которой будут размещены коды и данные процесса, а также предоставить ему процессорное время.

# Управление памятью

Управление памятью включает в себя распределение физической памяти между всеми существующими в данный момент процессами в ОС, а также загрузку их кода и данных в отведённые области памяти и защиту областей памяти каждого процесса.

# Управление файлами и внешними устройствами

С помощью драйверов

# Интерфейс прикладного программирования

Набор API-функций у каждого производителя свой.

# Защита данных и администрирование

Аудит -- функция фиксации всех событий.

# Требования к современным ОС

* + Расширяемость -- изменение аппаратуры, новые устройства.
  + Переносимость -- функционирование разных процессоров, платформ и пр.
  + Совместимость.
  + Надёжность (отказоустойчивость системы) -- защита от внутренних и внешних ошибок. Определяется построением архитектуры ОС.
  + Безопасность -- защита данных, ресурсов компьютера (аутентификация, авторизация, аудит).
  + Производительность -- хорошее быстродействие и реакция компьютера.

# Архитектура ОС

В настоящее время известно три архитектуры:

* + Монолитная -- каждый компонент включён в ядро и может непосредственно взаимодействовать с другим компонентом ОС (OS/360). Непосредственное взаимодействие компонентов делает ОС быстрой, но определение ошибок -- трудная задача и, более того, когда ОС выполняется с неограниченными правами доступа к ресурсам компьютера, то эти ОС особенно уязвимы для вредоносного кода.
  + Многоуровневая -- основана на организации компонентов ОС в уровни. Причём каждый уровень взаимодействует только с соседним уровнем, расположенным над ним или под ним. Такая ОС обладает высокой степенью модульности. Эта модульность обеспечивает простоту структурной организации ОС и возможность взаимодействия её компонентов, что упрощает проверку работоспособности ОС, её отладку и модификацию. Однако, многоуровневая организация требует, чтобы запросы на обслуживание пользовательского процесса проходили через несколько уровней, прежде чем он будет удовлетворён, что снижает производительность такой ОС.
  + Построенная на микроядре — представляет собой лишь малый набор услуг, который включается в ядро. Обычно в перечень этих услуг входит низкоуровневое управление памятью, межпроцессорное взаимодействие и синхронизация процессов. А большая часть компонентов выполняется вне ядра, с более высокими привилегиями (управление процессами, работа в сети, файловые системы, управление внешними устройствами и пр.). Поэтому сбои в этих компонентах не приводят к краху ОС.

Архитектуру ОС обычно разделяют на две части:

* + Ядро -- модули, выполняющие основные функции ОС.
  + Вспомогательные модули.

Для обеспечения высокой скорости ОС все модули ядра должны быть резидентными, т. е. постоянно находиться в ОП.

Транзитный модуль -- загружается в ОП только во время выполнения программы.

# Ядро в привилегированном режиме

ОС должна обладать исключительными полномочиями, чтобы быть арбитром в споре за ресурсы.

# Многоуровневая структура ОС

Железо -> Ядро -> Вспомогательный модуль

Последние два -- ОС.

Ядро представляет собой сложный многофункциональный комплекс и также обычно имеет многоуровневую структуру. Обычно выделяют 5 базовых слоев в ядре;

* + Средства аппаратной поддержки ОС (система прерываний, таймер, средства защиты памяти).
  + Машинно-независимые компоненты ОС -- отражают специфику конкретной аппаратуры компьютера. И этот слой экранирует следующие слои ОС от особенностей аппаратуры.
  + Базовые механизмы ядра -- наиболее примитивные операции ядра (диспетчеризация прерываний, перемещение страницы из памяти на диск).
  + Менеджеры ресурсов, реализующие стратегические задачи по управлению основными ресурсами компьютера.
  + Интерфейс системных вызовов -- обеспечивает благодаря API-функциям доступ к функциям ОС.

# Архитектура Windows NT

* + 1993 г. -- версия 3.11
  + 1996 г. -- 4.0
  + 1999 г. -- 5.0
  + 2003 г. -- 5.1 (XP)
  + 2006 г. -- 6.0 (Vista)
  + ... -- 6.1 (Seven)
  + ... -- 6.2 (Eight)

К службам уровня hal относится доступ к регистрам устройств, обработка прерываний, операция DMA, управление таймерами, управление BIOS, доступ к CMOS-памяти.

# Уровень ядра

Назначение этого уровня в том, чтобы сделать всю часть ОС независимым от аппаратуры, чтобы ее можно было переносить на другие компьютеры. Над ядром и драйверами находится «Исполнительная система», в которую входят:

* + Менеджер ввода-вывода;
  + Менеджер объектов;
  + Менеджер потоков;
  + Менеджер памяти (реализует виртуальную память со страничной подкачкой);
  + Менеджер безопасности;
  + Менеджер кэша -- хранит в памяти блоки жесткого диска, которые использовались в последнее время, для ускорения доступа к ним.
  + Менеджер plug and play;
  + Менеджер энергопотребления;
  + Менеджер конфигурации -- состояние системного реестра;
  + Менеджер ... процедур;
  + Интерфейс графических устройств (GDI) -- позволяет программам выводить данные на монитор и принтер.

# Системные службы

Предоставление доступа и интерфейсов к исполнительным системам.

# Режим пользователя

Три типа компонентов:

* + Динамические библиотеки -- работа заключается в том, чтобы реализовывать..
  + Системные процессы -- те процессы, которые обслуживает сама ОС.
    - SMSS -- диспетчер сеансов.
    - WINLOGON -- авторизация.
    - Idle -- время простоя.
    - И т. д.
  + Подсистемы.

# Загрузка Windows

BIOS → MBR (0 сектор, 0 цилиндр, 0 поверхность) → ntldr → (boot.ini, bootvid.dll, hal.dll, реестр, NTOSkrnl.exe) → сеансовый менеджер → CSRSS.exe → WINLOGON.exe → lsass.exe → services.exe → explorer.exe

# Процессы и потоки

...

# Взаимоблокировка (deadlock)

Под этим понятием можно понимать постоянное блокирование множества процессов, которые конкурируют в доступе к системным ресурсам компьютера, либо при взаимодействии друг с другом.

## Транспортная блокировка

Перекресток, автомобили, помеха справа

## Близкие технологии

Два потока (а и б) выполняют какую-то работу и им необходимы два ресурса: принтер и ик-порт

Поток А:

* + Занять принтер;
  + Занять порт;
  + ...
  + Освободить принтер;
  + Освободить порт.

Поток Б:

* + Занять порт;
  + Занять принтер;
  + ...
  + Освободить порт;
  + Освободить принтер.

В зависимости от соотношения скоростей выполнения потоков могут возникнуть три ситуации:

* + Взаимоблокировка;
  + Очередь к ресурсам;
  + Независимое использование ресурсов.

Пусть дано два одинаковых процесса, каждый из которых записывает отсканированный документ на какой-то диск.

Поток А:

* + Занять сканер.
  + Занять ДВД.

Поток Б:

* + Занять ДВД.
  + Занять сканер.

Выгружаемый ресурс -- это тот ресурс, который может быть безболезненно забран у процесса и передан другому.

Хоффман определил, что существует четыре условия возникновения 4 условия возникновения взаимоблокировки:

* + Взаимное исключение -- каждый ресурс компьютера, который в нем есть, он или отдан какому-то потоку, или свободен.
  + Удержание и ожидание -- любой процесс, который уже удерживает ресурсы, может запросить еще новые, необходимые ему ресурсы.
  + Условие отсутствия принудительной выгрузки ресурса. У процесса нельзя принудительно забрать ресурс, а процесс освобождает этот ресурс в ходе выполнения своих действий.
  + Условие циклического ожидания. Должна существовать круговая последовательность из двух или более потоков, каждый из которых ждет доступа к ресурсу, удерживаемому следующим потоком в этой последовательности.

Должна существовать замкнутая цепь потоков, каждый из которых удерживает как минимум один ресурс, необходимый потоку следующему в этой цепи после данного.

Чтобы возникла блокировка потоков, необходимо чтобы существовало хотя бы одно из перечисленных условий.

Страусовый алгоритм: Если количество возникающих ошибок соизмеримо с..

Выход из взаимной блокировки:

* + Прекратить все заблокированные процессы и перезапустить их.
  + Сделать откат заблокированных процессов и снова разрешить им выполниться (должен как-то фиксироваться откат).
  + Рекомендуется последовательно прекращать выполнение заблокированных процессов, до тех пор, пока ситуация блокировки не исчезнет. Тут несколько подходов:
    - Снимать тот, который занимает меньше памяти.
    - .. выводит меньше информации.
    - .. занимает больше всех времени и пр.
  + Последовательно выгружать ресурсы, пока взаимоблокировка не прекратится.

Алгоритм банкира -- позволяет определить..

Предполагается, что есть какая-то группа клиентов, которые хотят получить какой-то кредит.

Алгоритм банкира рассматривает каждый запрос по мере поступления и проверяет, к чему он приведет.

# Управление памятью

Память -- это важный ресурс, который требует тщательного управления.

ОС должна решать четыре базовые задачи:

* + Отслеживает свободную и занятую память;
  + Выделяет память процессам и освобождает память после завершения процесса;
  + Вытесняет коды и данные процессов из оперативной памяти, когда размер оперативной памяти недостаточен для размещения в ней всех процессов, и возвращает их обратно в оперативную память, когда в этом возникает необходимость.
  + Настраивает адреса программ на конкретную область физической памяти.

# Типы адресов

Существует три типа адресов:

* + Символьные имена -- те идентификаторы, которые присваивают пользователи при написании программы на алгоритмическим языке.
  + Виртуальные адреса -- их вырабатывает транслятор, переводящий программу в машинный код. Транслятор присваивает переменным и командам виртуальные адреса, при этом обычно начальный адрес программы одинаковый -- нулевой.
  + Физические адреса -- соответствуют номерам ячеек оперативной памяти, где в действительности расположены переменные или команды в ходе выполнения процесса.

Совокупность виртуальных адресов процесса называется **виртуальным адресным пространством**. Максимальный диапазон виртуального адресного пространства всех процессов одинаков: 0..2^32 (4 Гб), 0..FFFFFFFF.

Существует два вида размещения памяти:

* + Сегментное -- адресное пространство делится на сегменты и любой адрес -- сегмент и смещение.
  + Плоское -- адрес -- одно число.

Существует два подхода преобразования виртуального адреса в физический:

* + Замена виртуального адреса на физический выполняется один раз для каждого процесса в момент начальной загрузки виртуального адресного пространства в оперативную память.
  + Программа загружается в оперативную память в неизмененном виде в виртуальных адресах, но при загрузке этой программы в оперативную память фиксируется начальный адрес этой программы в оперативной памяти. Во время выполнения программы при каждом обращении к ОП выполняется преобразование виртуального адреса в физический.

Схема преобразования виртуального адреса в физический

Номер виртуальной страницы (P): Смещение (Sv)

При обращении по какому-либо адресу происходят следующие действия:

* + Из специального регистра процессора извлекается начальный адрес размещения таблицы страниц этого процесса -- AT.

Из адреса мы можем узнать номер виртуальной страницы и, зная размер дескриптора таблицы страниц мы можем получить..

* + Из этого дескриптора извлекаем номер физической страницы в котором размещается даная виртуальная страница.
  + К номеру физической страницы мы присоединяем виртуальное смещение.

Второй алгоритм:

Все множество виртуального адресного пространства делится на разделы (каталоги), а разделы потом делятся на страницы.

m -- количество позиций для номера раздела, на которые разобьется ВАП.

k -- количество позиций для номера страницы в данном разделе.

n -- разрядность (смещение) каждой страницы.

Номер последнего раздела 2^m - 1

Номер последней странички раздела 2^k - 1.

Все страницы имеют одинаковый размер, а все разделы содержат одинаковое количество страниц.

Размер страницы -- 2^n

Количество страниц в разделе -- 2^k

Количество разделов -- 2^m

Виртуальный адрес можно будет представить следующим образом: n разрядов будут представлять смещение виртуальной страницы, k -- номер страницы в разделе, m -- номер раздела.

Для каждого раздела строится собственная таблица страниц, и количество дескрипторов в этой таблице и их размер подбираются. таким образом, чтобы объем этой таблицы был равным объему виртуальной страницы.

На адресацию внутри страницы отводится 4 Кб, значит n = 2^12

k = 2^10

m = 2^10

Рассмотрим схему преобразования адресов для случая двухуровневой структуризации виртуального адресного пространства.

Виртуальный адрес состоит из трех частей:

* 1. Номер раздела.
  2. Номер страницы.
  3. Смещение.

Физический адрес состоит из двух частей:

* 1. Номер страницы.
  2. Смещение.

Предположим, что происходит обращение по какому-то адресу:

* 1. Определяется номер раздела, к которому принадлежит данный виртуальный адрес путем отбрасывания k+n младших разрядов этого адреса.
  2. Из таблицы разделов извлекается дескриптор соответствующей таблицы страниц.

Проверяем, находится ли данная таблица страниц в оперативной памяти. Если нет, то происходит страничное прерывание, данная таблица находится (в смысле ищется и находится:)) в файле подкачки и загружается в ОП.

* 1. Из данной таблицы извлекается дескриптор страницы и совершается переход по адресу нахождения этой страницы.
  2. Из данной таблицы страниц извлекается дескриптор из которого извлекается номер физической страницы.

Оптимальный алгоритм замещения страниц

Предполагается, что в памяти находится какой-то набор страниц. К каждой из этих страниц будет обращаться в какой-то момент времени какой-то процесс. Каждая страница может быть помечена числом равным количеству команд, которые будут выполняться перед первым обращением к этой странице.

Оптимальный алгоритм будет удалять страницы с максимальным числом.

Реально используемые алгоритмы:

NRU (Not recently used) -- исходная информация -- таблица страниц.

Выделяют два бита: R -- обращение, M -- изменение.

Когда какой-то процесс запускается, R == M == 0. В дальнейшем периодически (при каждом, например, прерывании по таймеру) бит R очищается для того, чтобы можно было отличить страницы к которым давно не происходило обращение от тех, на которые были ссылки.

Когда происходит страничное прерывание ОС проверяет дескрипторы всех страниц и делит их на четыре категории (классы) в зависимости от значений дескрипторов R и M.

Класс 0 -- не было обращений и изменений. 00.

Класс 1 -- не было обращений -- но страницы изменены. 01.

Класс 2 -- было обращение, но страница не изменена. 10.

Класс 3 -- изменение и обращение. 11.

Алгоритм NRU удаляет страницу с помощью случайного поиска в наименьшем классе..

# FIFO

Алгоритм замещения страниц памяти. Имеем список, засовываем в конец, убираем из начала.

# Алгоритм «Вторая попытка»

Модификация FIFO. Заключается в том, что FIFO может выгрузить из памяти часто загружаемые страницы, а их желательно не выкидывать.

Поэтому в этом алгоритме используется бит R -- обращение. Если он равен 0, значит к страничке давно не обращались, поэтому ее можно заменить или убрать, с если же R == 1, то ему присваивается значение 0, а страница переносится в конец списка.

# Алгоритм «Часы»

Вторая попытка -- не совсем эффективный алгоритм, так как в нем постоянно перемещаются страницы по списку. Предлагается хранить все страничные блоки в кольцевом списке, в форме часов. Стрелка в часах указывает на старейшую страницу. Если происходит страничное прерывание, то проверяется страница, на которую указана стрелка. Если ее бит R == 0, то она удаляется, иначе -- R = 0, и переход к следующей страницы.

# LRU -- Least Recently Used (страница, используемая меньше всего)

Есть страницы (4) используемые в следующем порядке: 0 1 2 3 2 1 0 3 2 3..

Пусть у нас есть матрица n\*n, где первоначально все ячейки равны 0. Когда же происходит обращение к i-ой странице, то все элементы этой строки 1, а этого столбца -- 0.

Трассировка:

0:

0 1 1 1

0 0 0 0

0 0 0 0

0 0 0 0

1:

0 0 1 1

1 0 1 1

0 0 0 0

0 0 0 0

2:

0 0 0 1

1 0 0 1

1 1 0 1

0 0 0 0

Если возникает страничное прерывание, то удаляется та страница, в которой в строке меньше 1.

# NFU -- Not Frequently Used (редко используемая страница)

Суть в том, что для каждой страницы, находящейся в ОП определяется счетчик, который в начале равен 0, а при каждом прерывании по таймеру проверяется значение бита R каждой страницы, и его значение прибавляется к счетчику.

При страничном прерывании для замещения выбирается страница с наименьшим значением счетчика.

# Алгоритм «Старения»

Доработка NFU, и суть его в том, что во-первых каждый счетчик сдвигается вправо на 1 разряд перед прибавлением бита R. Во-вторых, бит R прибавляется в крайний слева, а не справа разряд этого счетчика.

Например:

Такты: 0. 1. 2.

Счетчик: 101011 110010 110101

Процессы:

0. 1000. 1100. 1110

1. 0000. 1000. 1100

2. 1000. 0100. 0010 del!

3. 0000. 0000. 1000

4. 1000. 1100. 0110

5. 1000. 0100. 1010

# Аномалия Бипэ ди

Возникает вопрос. Сколько должно быть в ОП физических страниц, чтобы система функционировала эффективнее? Лучше больше или меньше?

Рассмотрим пример. Виртуальное адресное пространство разбито на 5 страниц, обращения в таком порядке:

0 1 2 3 0 1 4 0 1 2 3 4

Предположим, что у нас может быть три физические страницы. Выберем алгоритм FIFO:

0. 1. 2. 3

0: 0 1 2 3

1: 0 1 2

2: 0 1

....

Много цифр.

# Сегментное распределение памяти

Имеем два процесса. Виртуальное адресное пространство каждого процесса разбивается на сегменты, размеры каждого из которых определяются с учетом смыслового значения содержащихся в них информации.

Сегмент может представлять подпрограмму, функцию или массив. Деление виртуального адресного пространства на сегменты осуществляет компилятор. Максимальный размер сегмента в 32-разрядной системе может быть 4 Гб.

Av -> n, S, где n -- номер сегмента, а S -- линейный виртуальный адрес внутри этого сегмента. При загрузке процесса в ОП происходит загрузка только часть сегментов, а все виртуальное адресное пространство загружается в файл подкачки.

Если во время выполнения процесса происходит обращение к другому сегменту, который в данный момент отсутствует в ОП, то данный процесс переводится в состояние ожидания, возникает прерывание и ОС загружает необходимый сегмент с диска в ОП. Если необходимой памяти для сегмента в ОП нет, то ОС также должна решить вопрос о выгрузке какого-то сегмента из ОП на диск.

В момент запуска процесса в ОП также создаётся таблица сегментов, которая состоит из дескрипторов. В дескрипторах этой таблицы указывается следующее: базовый физический адрес сегмента в ОП (если он в ОП), размер сегмента, права доступа к сегменту, управляющая информация (признаки модификации, присутствия, обращения и др.).

Зная базовый адрес (по адресу сегмента), прибавляем к нему смещение и получаем физический адрес. Однако, недостаток в том, что сложение -- медленная операция, плюс память на хранение адреса избыточна.

При использовании сегментов память фрагментируется.

# Сегментно-страничное распределение

Сегментно-страничное распределение является комбинацией страничного и сегментного методов управления памятью.

Сначала ВАП процесса делится на сегменты, а затем сегменты делятся на страницы. Причем, делятся на страницы равного размера и перемещение между ОП и диском осуществляется страницами. А т. к. страницы одинакового размера, то фрагментирования нет.

ВАП разбивается на сегменты. А в дальнейшем строится т. н. линейное ВАП. Для каждого ВАП строится таблица дескрипторов сегментов, однако в поле базового адреса указывается не начальный физический адрес этого сегмента в ОП, а начальный линейный виртуальный адрес сегмента в пространстве виртуальных адресов.

Наличие базового виртуального адреса сегмента позволяет однозначно преобразовать адрес заданный в виде пары чисел (номер сегмента и смещение в сегменте) в линейный виртуальный адрес, который затем преобразуется в физический адрес механизмами страничного распределения.

Размер страницы выбирается равным степени двойки, что значительно упрощает механизм преобразования виртуального линейного адреса в физический.

При создании процесса в ОП загружается только часть страниц, а остальные подгружаются по необходимости, согласно алгоритмов замещения страниц.

Базовые адреса сегментов и таблицы страниц процесса являются частью его контекста. При активизации процесса эти адреса загружаются в специальный регистр процесса, которые затем будут использоваться для преобразования адреса виртуального в физический.

Преобразование виртуального адреса в физический будет выполняться в два этапа.

На первом работает механизм сегментации: исходный виртуальный адрес, заданный двумя числами -- n и S преобразуется в линейный виртуальный адрес. Для этого на основании базового адреса в таблице сегментов и номера сегмента вычисляется адрес дескриптора.

На втором этапе работает страничный механизм. Если размер страницы равен 2^k, то мы сразу из найденного ранее адреса можем найти смещение и номер виртуальной страницы.

Далее по номеру виртуальной страницы находится в таблице дескрипторов необходимый и из него берется номер физической страницы, а смещение равно тому, что было найдено.

Олифер. Сетевые ОС.

# Файловая система

Цели использования файлов:

* 1. Долговременное и надежное хранение файлов.
  2. Совместное использование информации.

Файловая система -- часть ОС, которая включает в себя:

* 1. совокупность всех файлов на диске;
  2. наборы структур данных, используемых для управления файлами (дескрипторы файлов, каталоги файлов, таблицы занятого и свободного места в памяти);
  3. комплекс системных программных средств, реализующих действия над файлами (создание, уничтожение, чтение, запись, переименование, поиск файла).

Типы файлов:

* 1. регулярные (обычные) -- содержат информацию произвольного характера, которую заносят в них пользователи или которая появляется в них в результате работы системных или пользовательских приложений;
  2. каталоги -- это особый тип файлов, который содержит системную справочную информацию о наборе файлов; эти каталоги обычно устанавливают связь между именами файлов и их характеристиками, используемых файловой системой для управления этими файлами.

Теоретическая структура файловой системы:

* 1. Дерево -- файл входит только в один каталог.
  2. Сеть -- файл может входить сразу в несколько каталогов.

Имена файлов:

* 1. Простое -- идентифицирует файл в пределах того каталога, где он находится и вид этот простого файла определяется типом ОС.

MS DOS: 8.3

UNIX: 14

NFS, FAT-32: 256

* 1. Составное (полное) -- цепочка имен всех каталогов + простое имя.
  2. Относительное -- через понятие текущего каталога.

Каталог MS DOS, 32 символа: 8 под имя файла, 3 символа расширение, R, A, H, S (1 байт), 10 под резерв, время 2, дата 2, номер первого кластера 2, 4 размер файла

Катало ЮНИКС, 16: 2 номер дескриптора, 14 имя

Атрибуты файла -- информация, описывающая свойства файла. Во всех обычно присутствуют: тип файла, размер, время создания и доступа, время изменения и пр.

В НТФС есть права доступа, а в ФАТ нет.

# Логическая организация файлов

Данные, которые находятся в файлах, могут иметь определенную логическую структуру. Поддержание структуры данных в файлах может быть возложено на приложение, либо на ОС.

Неструктурированная последовательность данных -- о логической структуре знает создавшая ее программа.

Если информация, структурированная ОС имеет записи одинаковой длины, то доступ может быть и прямой, и последовательный. Если разной, то последовательный, а прямой с помощью хранения информации о длине (индексный файл).

# Физическая организация файловой системы

Адресация жесткого диска: Номер цилиндра, номер поверхности и номер сектора.

Каждая пластина состоит из дорожек, нумерация которых от края к центру. Цилиндр -- совокупность дорожек с одинаковыми номерами (одинакового радиуса).

Каждая дорожка делится на сектора. Сектор -- это наименьшая единица объема при записи информации на диск. Обычно сектор 512 байт.

ОС в данном случае не контроллер и при работе с диском использует свою собственную единицу адресного пространства, которая называется кластер. Кластер состоит из нескольких секторов и всегда размер кластера определяется 2^k. Кластер состоит из секторов, потому минимальное k == 9.

Два типа форматирования:

* 1. Физическое -- производителем.
  2. Логическое (высокоуровневое) -- разметка на логические разделы, определяется размер кластера, формируется MBR, в котором таблица разделов, где должен быть определен активный раздел. В каждом разделе есть свой загрузочный сектор.

# Физическая организация и адресация файлов

Основными критериями эффективности физической организации файлов являются:

* 1. Скорость доступа к данным.
  2. Объем адресной информации файла.
  3. Степень фрагментированности дискового пространства.
  4. Максимально возможный размер файла.

## Непрерывное размещение

Доступ к кластерам по номеру.

Выбираем кластера последовательно.

## Размещение файла в виде связанного списка кластеров дисковой памяти

В начале каждого кластера размещается указатель на следующий кластер.

Сложность обращения к любому месту файла.

Количество данных файла, хранящихся в кластере, не кратно степени двойки.

## Использование связного списка индекса

Модификация предыдущего. Файлу так же выделяется память в виде связного списка кластеров, номер первого кластера запоминается в записи каталога, где хранятся все атрибуты этого файла, а номера следующих кластеров этого файла хранятся в специальной таблице индексов. В этой таблице индексов с каждым индексом связан определенный кластер.

Индексы располагаются в отдельной области диска и когда кластер свободен, значение индекса равно 0.

Если некоторый кластер назначается какому-то файлу, то индекс этого кластера становится равным номеру следующего кластера этого файла, либо принимает специальное значение, указывающее, что этот файл закончился.

## Перечисление номеров кластеров

Адресная информация -- номера занятых кластеров.

Недостаток -- нужно отводить больше место под адресацию.

Рассмотрим модификацию последнего метода. Этот метод сперва появился в UNIX-системе UFS.

Для сохранения объема адресной информации прямой способ можно сочетать с косвенной адресацией. Для хранения адреса файла выделяется 15 полей (60 байт).

Если размер файла меньше или равен 12 кластерам, то номера этих кластеров непосредственно перечисляются в первых 12 полях.

Если размер файла превышает 12 кластеров, то тогда используется 13-е поле, которое содержит адрес кластера, в котором могут располагаться следующие адреса кластеров файлов. Этот 13-ый элемент адреса используется для косвенной адресации. Он указывает на кластер с номерами кластеров -- продолжений этого файла (2048 номеров может быть).

Если размер файла превышает 2060 кластеров, то используется 14-е поле, которое указывает на кластер, хранящий 2048 адресных кластеров. Если вдруг что, то в 15 МЫ ПОЙДЕМ ЕЩЕ ГЛУБЖЕ!!!

У этого метода была дальнейшая модификация, благодаря которой появился NTFS. В модификации сокращается объем адресной информации за счёт того, что адресуются не номера кластеров, а непрерывные области, состоящие из смежных кластеров.

# Физическая организация FAT

Рассмотрим MS DOS FAT.

Логический раздел делился на пять областей:

* 1. Загрузочный сектор 512 байт.
  2. Таблица FAT. Содержит информацию о размещении файлов и каталогов на диске. Состоит из массива индексных указателей, количество которых равно количеству кластеров области данных. Между кластерами и индексными указателями имеется взаимно однозначное соответствие.

Индексный указатель может принимать следующие значения:

* 1. Кластер свободен.
  2. Кластер используется файлом и не является последним кластером файла. В этом случае индекс содержит номер следующего кластера файла.
  3. Последний кластер файла.
  4. Дефектный кластер.
  5. Резервный кластер.

Таблица является общей для всех файлов из области данных.

* 1. Резервная копия таблицы FAT.
  2. Корневой каталог. 32 сектора, 16 Кб.
  3. Область данных. Делится на кластера.

При создании файла происходит просмотр таблицы с начала. Если находит свободный кластер, то система фиксирует этот номер в каталоге. В последнем кластере пишем EOF.

# Физическая организация UFS

На диске 4 области:

* 1. Загрузочный блок
  2. Суперблок -- содержит информацию о файловой системе (размер ФС, число дескрипторов, список свободных блоков (кластеров), список свободных индексных дескрипторов).
  3. Область индексных дескрипторов (дескрипторы размещаются по возрастанию).
  4. Область файлов (обычные файлы и файлы-каталоги).

Основное отличие в том, что имя файла хранится отдельно от его характеристик. Имя только в каталоге, а характеристики в индексном дескрипторе.

Индексный дескриптор имел размер 64 байта. В нем содержалась следующая информация: ID владельца, тип файла, права доступа к файлу, время модификации, обращения, адресная информация.

Каждый индексный дескриптор имеет уникальный номер, который по сути является уникальным именем файла. Соответствие между именем и номером определяется при помощи иерархии каталогов.

При создании файла ему выделяется номер свободного индексного дескриптора из списка имеющихся. А при уничтожении этот номер возвращается в список.

Определение местоположения файла на диске:

* 1. Просмотр корневого каталога с целью поиска первой составляющей из имеющихся. Определяется номер ее индексного дескриптора.
  2. Из области индексных дескрипторов считывается дескриптор с полученным ранее номером. Из него определяется физический адрес каталога.
  3. Просматриваем найденный каталог с целью нахождения второй составляющей адреса. Определяется номер индексного дескриптора следующего каталога.
  4. Считывается индексный дескриптор, найденный на предыдущем шаге.
  5. .....
  6. Находится индексный дескриптор необходимого файла.
  7. Из индексного дескриптора файла определяется номер блока, в котором находится информация необходимого файла.

HPFS -- High Performance FS, была до НТФС.

НТФС отличается тем, что:

* 1. Поддерживает большие файлы и диски (до 2^64, дохуя).
  2. Восстановление после сбоев оборудования и отказов программных и аппаратных средств.
  3. Высокая скорость операций, особенно при больших дисках.
  4. Низкий уровень фрагментации.
  5. Гибкая структура, допускающая создание новых атрибутов файлов.
  6. Поддержка длинных символьных имен.
  7. Контроль доступа к каталогам и отдельным файлам.
  8. Возможность автоматического сжатия файла.
  9. Шифрование файлов.
  10. Квотирование пользователей.

# Физическая организация НТФС

Сперва идет загрузочный сектор, потом главная файловая таблица (MFT), системные файлы, потом повторяется загрузочный сектор, а дальше идет область данных.

Основой является главная таблица файлов, строки которой являются записями, имеющими номера. Все записи на диске идентифицируются дескриптором.

Первый отрезок MFT содержит 16 стандартных записей, созданных при форматировании.

* 1. Главная таблица файлов.
  2. Копия
  3. Файл журнала
  4. Имя тома и пр.
  5. Атрибуты
  6. Индекс корневого каталога
  7. Карта кластеров
  8. Загрузочный сектор адрес
  9. Имена битых кластеров
  10. Таблица квот
  11. И еще хуйни чутка..

Сохраняемая информация о файле -- отрезок. Это несколько смежных кластеров. Хранится начало отрезка и их количество. Virtual logic cluster name, Logic cluster name, K.

В системе НТФС адресация происходит по трём числам.

# Запись

Есть четыре различных размера файла. Если файл минимально возможный, то файл может записаться прямо в таблицу.

Если файл большой, то в данных файла в таблице хранится адресная информация.

Если файл еще больше, то в область данных записывается номер блока, в котором записаны адреса.

Если прям пиздец какой большой файл, то в области данных записаны несколько номеров блоков, которые содержат адреса полезной информации.