Операционные системы

Курс лекций для гр. 4057/2

Лекция №10

Вопросы к лекции 9

1. Каковы достоинства и недостатки протоколов 1 и 2? Какой из них для каких устройств применяют?
2. Несмотря на то, что процессор не занят обработкой прерываний на каждый пересылаемый элемент данных во время операции обмена блока через DMA, его производительность все же несколько снижается. Почему?
3. Доступ DMA к памяти обычно выполняется с более высоким приоритетом, чем доступ процессора. Почему?
4. Приведите примеры перенаправления вв/вы на различные устройства командами оболочки Unix.
5. Почему быстродействие дисковой памяти оказалось «замороженным», хотя быстродействие электронных компонентов и емкость дисков удваивались, по закону Мура, каждые два года? Что его ограничивает?
6. Предложите идею алгоритма C-SCAN.

Содержание

Основные понятия

- Файл - это именованная совокупность однородных данных, рассматриваемая как единое целое
  - Однородные = имеющие общее происхождение и назначение
- Этимология: File = дело, досье, подшивка документов, картотека, скоросшиватель
- Обычно это данные на внешнем носителе (магнитном диске, ленте, CD-ROM, флэш-памяти), но может быть файл и в основной памяти
- Основная функция файловой системы – организация надежного хранения данных и быстрый доступ к ним (т.е., чтение/запись):
  a) к файлу как единому целому
  b) к компонентам файла - *записям*

Типы файлов

Тип ф. определяется постфиксом его имени: .exe, .jpg, .doc, …

Два больших класса:

1. Символьные – состоят из текстовых строк переменной длины (обычно символы ASCII, тогда тип .txt, .html, cpp, …)
   - легко отображаются на экране и редактируются
   - используются для исходных текстов программ, командной строки и скриптов
2. Двоичные (бинарные) – их внутренняя структура «понимается» соответствующим приложением
   - входные/выходные данные приложений (.jpg, .dxf, .zip, …)
     - в частности, объектные и загрузочные файлы (.obj и .exe)

Структура файлов

- Данные файлов имеют иерархическую структуру:
  биты → байты → символы → поля → записи → файлы → тома
  - поле – это последовательность символов
  - запись - это наименьший элемент данных, который может быть обработан как единое целое прикладной программой
  - том – устройство хранения ф.
- ОС распознает и поддерживает эту структуру, но не всегда полностью
  - Windows и Unix «не видят» полей, и в них записи = символам
    - Символ = 1 или 2 байтам
- Физический обмен с устройством осуществляется большими единицами – *блоками*; записи объединяются в блоки для вывода и разблокируются - для ввода
- Считывание очередного байта производится с текущей позиции, которая характеризуется смещением от начала файла
  - Зная размер блока, легко вычислить номер блока, содержащего текущую позицию

Служебные структуры данных (1)

1. Каталог (directory, folder), определяющий группу ф. – содержащий ссылки на них; виден пользователю
   - Каталог - это тоже ф. (впервые - в Unix'e)
   - Структура системы файлов иерархическая, но не чистое дерево; возможны *псевдонимы* - альтернативные пути к ф. - копирование из одного каталога в другой без физического дублирования: команда link.

2. Дескриптор файла - внутренняя структура данных ОС; в Unix - *индексный узел* (index node, inode), он содержит:
   - *Индекс* ф. - внутренний целочисленный идентификатор
   - Учетную информацию (размер, даты создания/обновления, ...)
   - Атрибуты (создатель, владелец, защита, тип, вид обмена, организация ф. , ...)
   - Физический адрес ф. (недоступный пользователю)

Организация файлов и доступ к ним

1. Последовательный файл – последовательность байтов или записей; обработка подобных ф. - последовательное чтение записей от начала ф. (модель ленты)
2. Файл произвольного (или прямого) доступа – для дисковых ф.: содержимое ф. может быть разбросано по разным блокам диска, которые можно считывать в произвольном порядке
   - номер блока однозначно определяется позицией внутри ф. (относительный номер от начала ф.)
   - два способа задания места, с которого надо начинать чтение:
     - с начала
     - с текущей позиции, которую дает операция seek

В старых ОС поддерживались еще индексный, индексно-последовательный и др. методы доступа к ф., состоящим из многобайтных записей

Сегодня сложно структурированные данные – сфера баз данных

Хранение дескрипторов файлов

- В MS-DOS дескриптор хранится в элементе каталога, соответствующем файлу
- В современных ОС дескрипторы всех файлов диска собраны в одном файле - *индексном файле* диска, обычно продублированном
  - В каталогах же хранятся только отображения имен ф. на их индексы. Например, каталог в Unix - это таблица со строками из двух элементов:

```
# define DIRSIZE 14        // макс. длина имени
...
struct direct {            // элемент каталога
  ino_t d-ino              // целоч. номер индексного узла
  char d_name [DIRSIZE]    // имя файла
};
```

Здесь тип ino_t зависит от реализации Unix (определяется в <sys/dir.h>) например, unsigned short

(Вопрос 1)

Служебные структуры данных (2)

3. Системная таблица открытых файлов – совместно используется всеми процессами. В ней для каждого ф.:
   - счетчик числа процессов, одновременно открывших ф. (несколько для R, но только один – для W)
   - атрибуты, включая владельца, защиту, ...
   - адрес на диске
   - указатели на буферы вв/вы в памяти

4. Таблица открытых файлов процесса - в служебной области памяти процесса:
   - указатель на вход в системной таблице открытых ф.
   - положение указателя текущей позиции в ф. (*смещение)* fp
   - режим доступа (mode): R, W, RW

(количество одновременно открытых файлов ограничено !)

Операции с файлами (на примере Unix)

`Create(name)`

- Выделить дисковое пространство
- Создать дескриптор ф. и добавить его к каталогу

`Delete(name)`

- Найти каталог, содержащий ф.
- Освободить блоки на диске, занятые ф.
- Удалить дескриптор ф. из каталога

`fileId = Open(name, mode)`

- Проверить, не открыт ли ф. другим процессом. Если нет, то:
  - найти ф.
  - скопировать его дескриптор в системную таблицу открытых ф.
- Проверить права доступа; если нет, то прервать операцию
- Увеличить на 1 счетчик процессов, открывших ф.
- Создать вход в таблице откр. ф. процесса, указывающий на вход в сист. таблицу откр. ф.
- Инициализировать смещение: установить его на начало ф.: fp = 0
- Вернуть индекс ф. в таблицу открытых ф. процесса
- Подготовить вв/вы (выполняет подсистема вв/вы ОС)

Операции с файлами (2)

`Close(fileId)`

- Завершить операцию вв/вы
- Удалить вход для ф. в таблице откр. ф. процесса
- Убавить счетчик процессов, открывших ф. Если он == 0, удалить ф. из сист. таблицы откр. Ф.

(Вопрос 2)

`Read(fileId, from, size, bufAddress)` `// прямой доступ:`

```
for (i = from; i < from + size; i++)
  bufAddress[i-from] = file[i];
```

`Read(fileId, size, bufAddress)` `// последовательный доступ:`

```
for(i=0; i < size; i++)
  bufAddress[i] = file[fp + i];
fp += size;
```

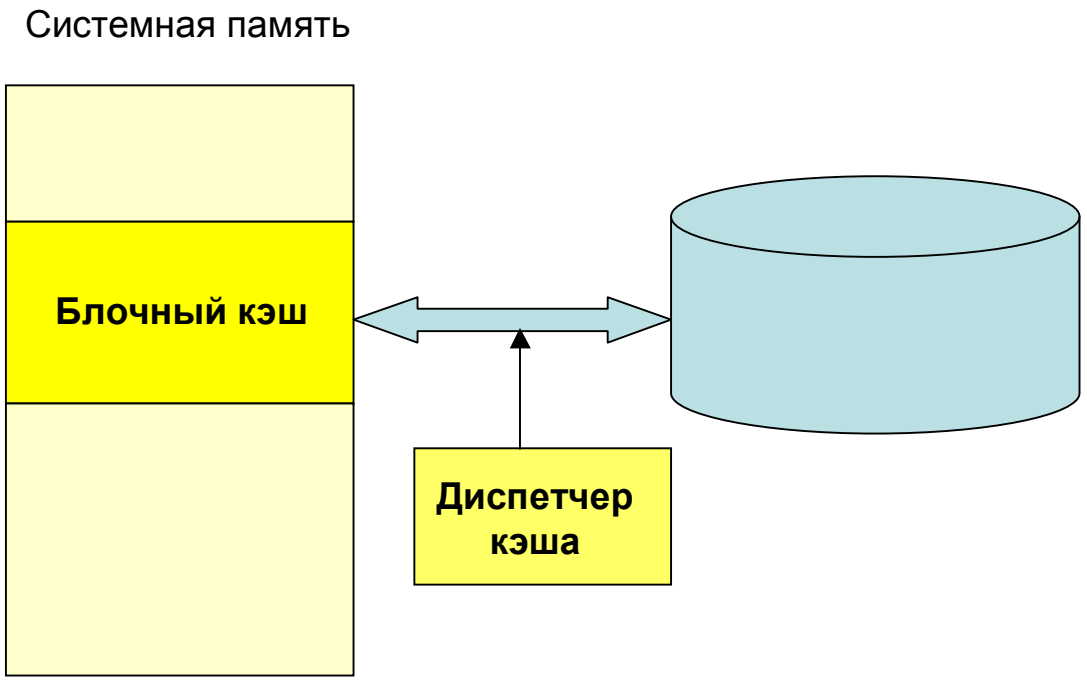
`Write(...):` подобно **`Read`**, но копировать из буфера в файл

`Fp = seek(size)` – переместить указатель текущей позиции на **`size`** байтов вперед

Кэширование файлового вв/вы

Для минимизации числа обращений к диску применяется *блочный кэш* в основной памяти: в нем хранятся считанные недавно дисковые блоки

- **Только благодаря ему возможна обработка 10^3-10^4 запросов вв-вы в сек**

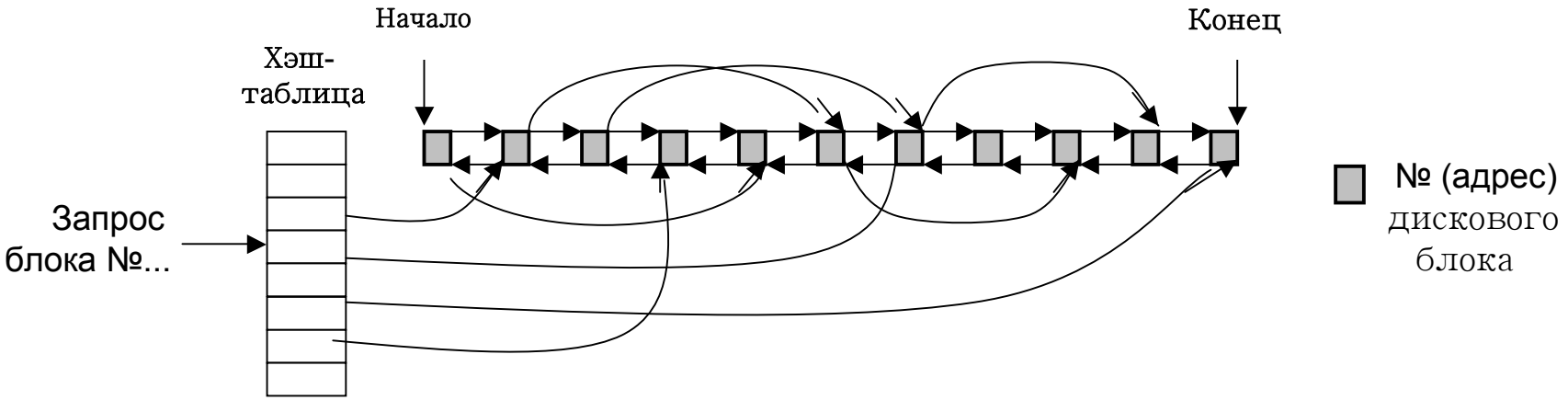


Вариант организации блочного кэша

Буфер с точной LRU-дисциплиной вытеснения: при каждом обращении к блоку он перемещается в конец списка

– в начале списка всегда находится самый старый блок – кандидат на вытеснение

(Вопрос 3)



Для быстрого определения, есть ли нужный блок в кэше (среди тысяч блоков!) - хэширование № блока и поиск в хэш-таблице

Все блоки с одинаковыми хэш-кодами сцепляются вместе в связный список; поиск в нем – по № блока

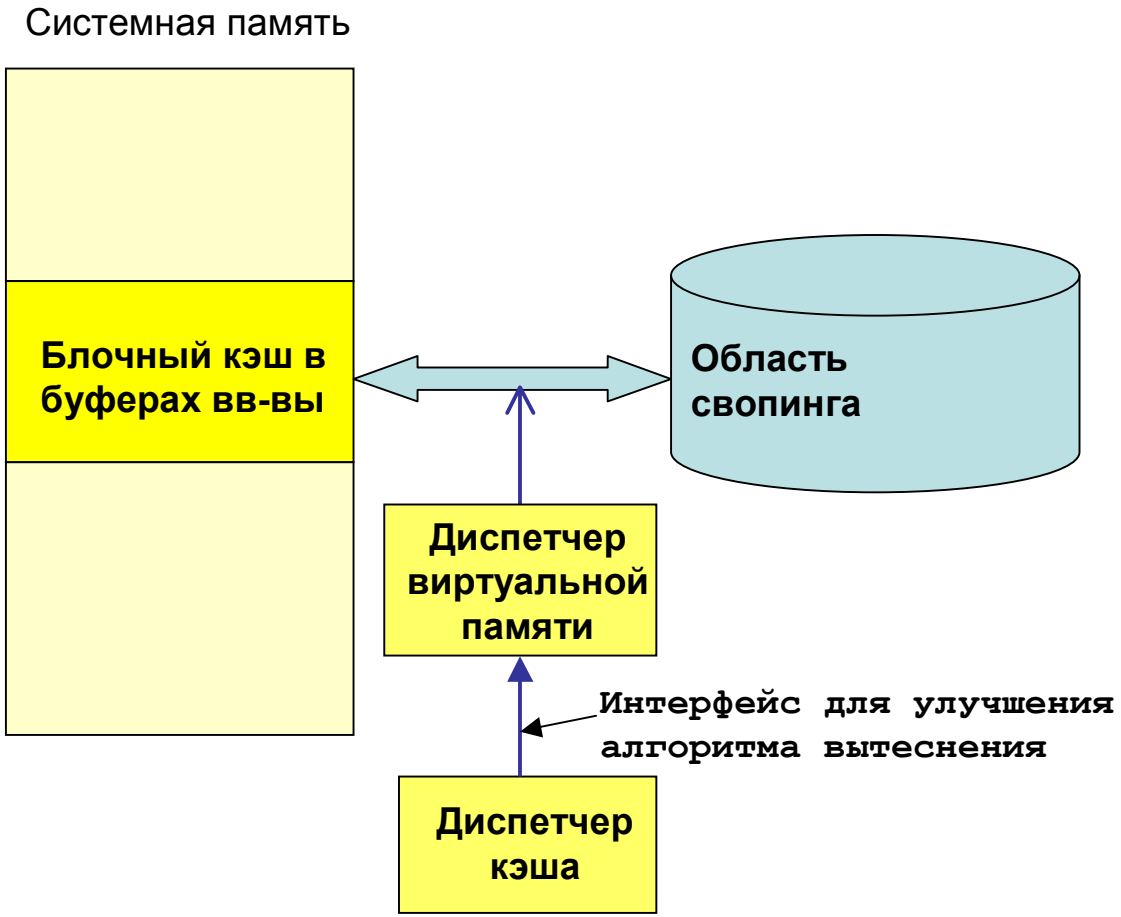
Запись блоков кэша на диск

Модифицированные блоки ф. нужно сохранять на диске. Когда?

- Сразу же после изменения – *кэш со сквозной записью* (в MS-DOS)
  - велико количество обращений к диску
- Периодически - *кэш со отложенной записью*
  - неприменимо для блоков критически важных ф. (напр., индексных узлов)
- Комбинированный способ (в Unix, Win2k):
  - сквозная запись критических блоков (каталогов, дескрипторов ф., косвенных блоков)
  - для остальных блоков – запись периодическая (обычно каждые 30 с) или когда блок вытесняется из кэша

• Современные дисководы имеют аппаратный кэш с простыми алгоритмами вытеснения
• Кэшироваться может не только локальный файловый обмен, но и обмен по сети с файл-сервером. Напр., Internet Explorer кэширует веб-страницы

Кэширование файлового вв/вы в Windows (1)



Кэширование файлового вв/вы в Windows (2)

- - осуществляется *диспетчером кэша* централизованно для всех драйверов файловых систем (как локальных, так и сетевых)
- Ф. проецируются на системную часть виртуального адресного пространства – в буферы вв/вы
- Единица замещения в кэше: страница = 8 дисковым блокам
- Дисциплина замещения – FIFO с отложенным вытеснением – реализуется диспетчером виртуальной памяти ОС
  - Диспетчер кэша может указать, в конец или в начало списка простаивающих страниц (т.е., очереди отложенного вытеснения) следует помещать условно вытесненную страницу (Вопрос 4)
- Системный поток отложенной записи активизируется при достижении порогового числа измененных страниц (dirty page threshold – параметр настройки), но не реже 1 раза в сек
- Для приложений, не терпящих задержки записи, поддерживается режим сквозной записи (write-back cashing), включаемый для конкретного файла

Алгоритмы кэша файлового вв/вы

- Диспетчер кэша поддерживает *интеллектуальное опережающее чтение* (intelligent read-ahead): на основе двух последних запросов он пытается предсказать следующий
  - Например, если приложение обращается сначала к стр. № 400, затем 300, то диспетчер заблаговременно считывает в кэш страницу 200
- Если установить FILE_FLAG_SEQUENTIAL_SCAN при открытии ф., то производится *последовательное* опережающее чтение специальным системным потоком параллельно с потоком приложения
- При установке флага FILE_FLAG_RANDOM_ACCESS опережающее чтение отсутствует
- Размер системного кэша в ВАП зависит от объема физической памяти
- Страницы кэша входят в общий с пулом подкачиваемых страниц системный рабочий набор, физическим размером которого управляет общий для всех рабочих наборов код
  - Диспетчер задач показывает под названием System Cache суммарный размер этого набора, а не физический размер кэша!

Методы распределения дискового пространства (1)

Задача: назначить, какие блоки диска следует выделить файлу

Основные методы: непрерывный, цепочечный и индексный

Критерии: экономное использование дискового пространства и быстрый доступ

1. Непрерывное распределение (contiguous): ф. занимает множество блоков, последовательно расположенных на диске
   - При поиске свободного места на диске применяют алгоритм "первый подходящий"

   Достоинства: простота и быстрый доступ. Так размещают ф. страниц в области свопинга

   Недостатки:
   - Большая внешняя фрагментация диска.
   - Как знать будущий размер при create ? → внутренняя фрагментация

   Применяют схему добавок - экстентов (extents), выделяемых динамически, по мере надобности

Методы распределения дискового пространства (2)

2. Цепочечное распределение (linked): ф. - это список блоков, произвольно расположенных на диске
   - В каждом блоке, кроме последнего, содержится указатель на следующий блок в списке
   - Адрес ф. - это номер его первого блока

   Недостатки непрерывного распределения устраняются, однако произвольный доступ существенно замедляется

Способ ускорения доступа:

таблица отображения: № блока → указатель на следующий блок - размещается целиком в памяти, поэтому просмотр списка при прямом доступе быстрый (FAT в MS-DOS и Win2k, в OS/2)

Недостаток: многочисленные перемещения головки дисковода, если диск не кэширован

Методы распределения дискового пространства (3)

3. Индексное распределение (indexed): *индексный блок* в дескрипторе ф. хранит номера всех его блоков

- Нет фрагментации, и быстрый доступ
- Проблема - очень большие файлы: как ограничить индексный блок разумными размерами?
- Решение в Unix - косвенные блоки на диске

Адресация файловых блоков в Unix

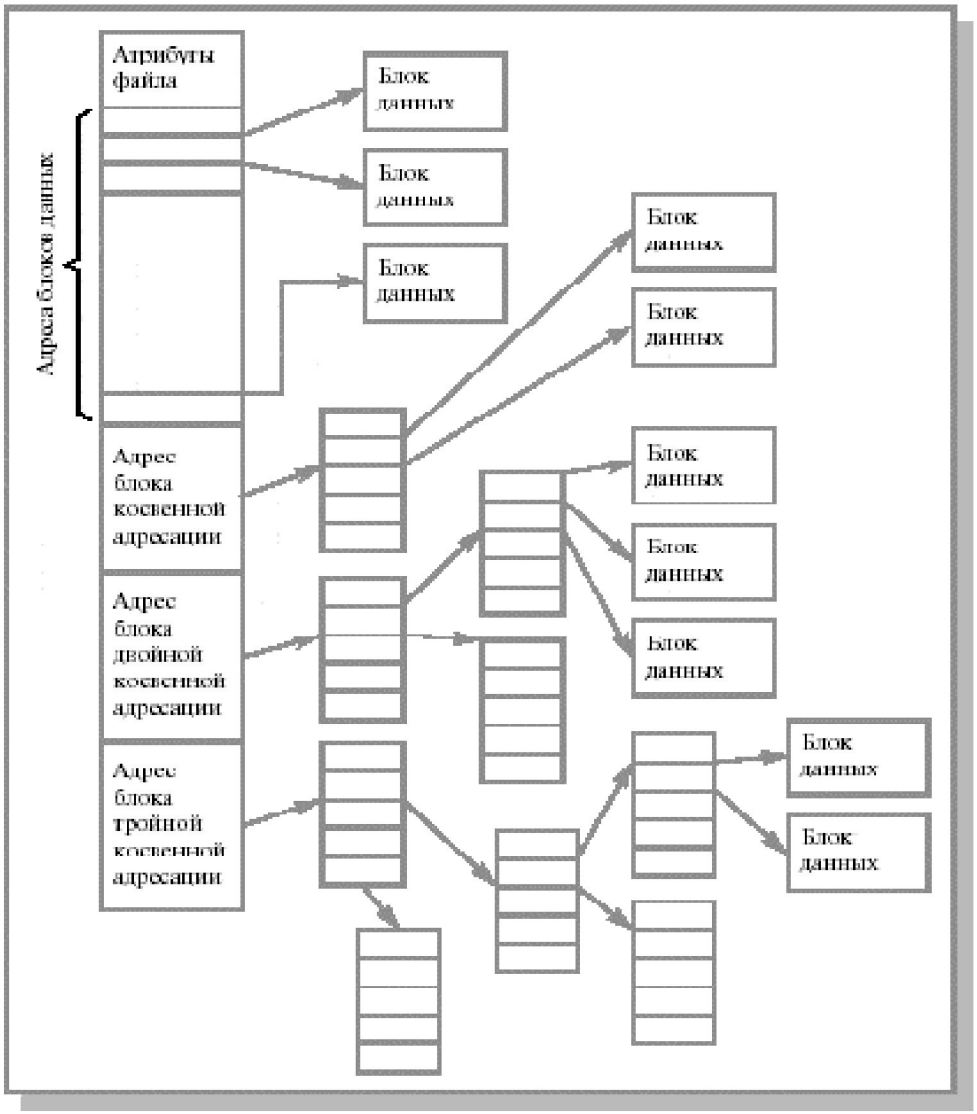


Таблица адресов блоков в индексном узле:

1-12 строка: адреса первых 12 блоков диска

13 строка: адрес блока косвенной адресации, содержащего адреса 256 блоков диска

14 строка: адрес блока двойной косвенной адресации – адреса 256 блоков косвенной адресации

15 строка: адрес блока тройной косвенной адресации - адреса 256 блоков двойной косвенной адресации

В ранних версиях Unix размер дискового блока был 512 байт, в современных - 4 Кбайт

Файловые системы Win2k

1. FAT-16: 16-разрядные дисковые адреса, max 2 Гбайт
2. FAT-32: 32-разрядные адреса, max 2 Тбайт
   - Обе совместимы с MS-DOS
3. NTFS: 64-разрядные адреса, max 2^{64} байт

NTFS (1)

- Новые возможности: безопасность, отказоустойчивость, большие файлы, UNICODE-имена и сжатие файлов
- Базовая сущность - том (volume) - часть диска, целый диск или несколько дисков
- Вся информация о томе - метаданные - хранятся в обычных файлах
- Единица цепочечного распределения дискового пространства – кластер 4 Кбайт = 8 секторов по 0.5 Кбайт (м.б. больше, до 64 Кбайт)

NTFS (2)

- Каждый ф. описан одной или более 1-Кб записями в файле MFT (Master File Table) – главной структуре данных тома
- Файл - структурный объект, состоящий из атрибутов
- Каждый атрибут - независимый поток байтов
  - некоторые из атрибутов - стандартные для всех файлов (имя ф., дата создания, дескриптор защиты, данные, ...)
  - другие специфические; напр., файл каталога имеет атрибуты, описывающие отображение имен ф. на индексы
  - маленькие атрибуты хранятся в самой записи MFT и наз. *резидентными*
  - для крохотных ф. даже атрибуты данных могут храниться в записи MFT
- Каждый ф. тома имеет уникальный ID, называемый *file reference* - 64 бит:
  - 48-битовый номер ф.- номер записи в MFT, описывающей ф.
  - 16-битовый *счетчик обращений (sequence number)* - используется для проверок целостности ф. системы

Файловые системы Linux

- Их несколько – для разных аппаратных платформ
- Общий верхний уровень – VFS (Virtual File System), под ним ф. системы:
  - общего назначения: ext2, ext3, FAT, UDF
  - сетевые: NFS, CIFS, Coda
  - размещенные в памяти: procfs, sysfs, rampfs, tmpfs

VFS

- Ф. представляют различные объекты: ф. дескрипторы, наборы данных, устройства и совместно используемые области памяти
- Индексные узлы (inodes) описывают адреса файловых объектов
- Отображение: ф. дескрипторы → inodes хранится в объекте dentry (directory entry)
- VFS поддерживает два кэша: dcache (directory entry cache) и inode cache

ext3

- 32-битовые адреса 4-Кбайтных блоков – всего до 4100 Гбайт дисковой памяти
- В индексных узлах хранятся атрибуты ф.

Заключение

- Файловая система – главная часть ОС, видная пользователю
- Большое разнообразие файловых систем и их функциональное богатство говорит о том, что большинство компьютеров не столько вычислительные, сколько информационные машины
- Кэш файлового вв/вы аналогичен аппаратной кэш-памяти и программному кэшу страничного обмена, но в отличие от них, в нем возможна реализация точного LRU-алгоритма вытеснения

Вопросы к лекции 10

1. Сравните эти два подхода с точки зрения удобства, эффективности и надежности.
2. Некоторые ОС автоматически открывают файл при первой ссылке на него и закрывают при завершении процесса, если не осталось других процессов, его использующих. Каковы преимущества и недостатки этой схемы по сравнению с более распространенной, когда пользователь должен явно открывать и закрывать файл?
3. Почему здесь оказалось возможным реализовать точный LRU, а в виртуальной памяти – нет?
4. Что следует указать в случае последовательного доступа к файлу? Прямого доступа?