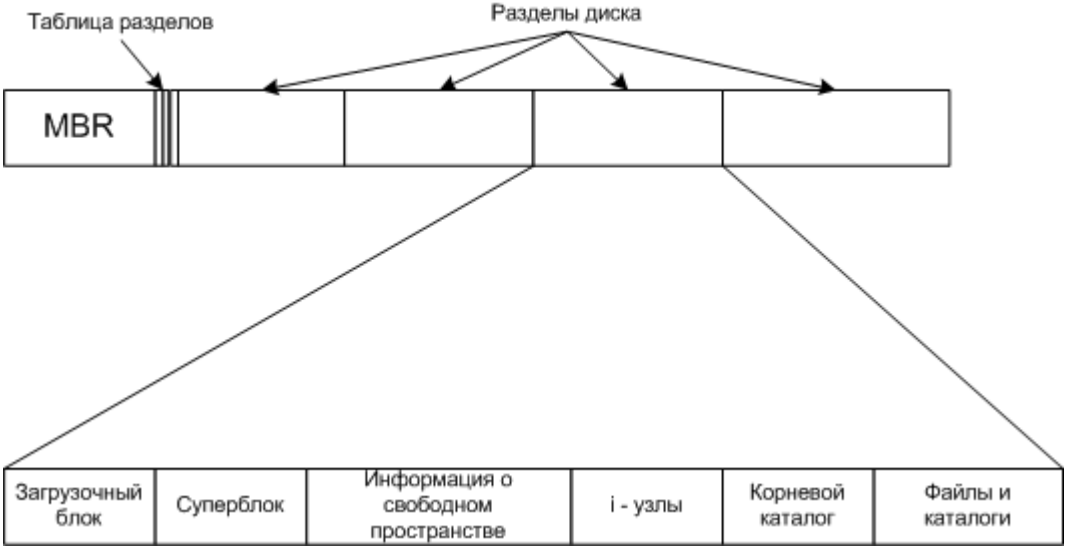
2.3. Структура файловой системы.



Возможная структура файловой системы

Все что до "Загрузочного блока" и включая его одинаково у всех ОС. Дальше начинаются различия.

Суперблок - содержит ключевые параметры файловой системы.

2.4. Физическая организация и адресация файлов.

Критерии эффективности организации хранения данных:

- Скорость доступа к данным;
- Объем адресной информации файловой системы;
- Степень фрагментированности дискового пространства;
- Максимально возможный размер файла и раздела;
- Максимально возможное количество файлов и каталогов;
- Сохранность информации после неожиданного прекращения работы системы или процесса;
- Одновременный доступ к информации нескольких процессов.

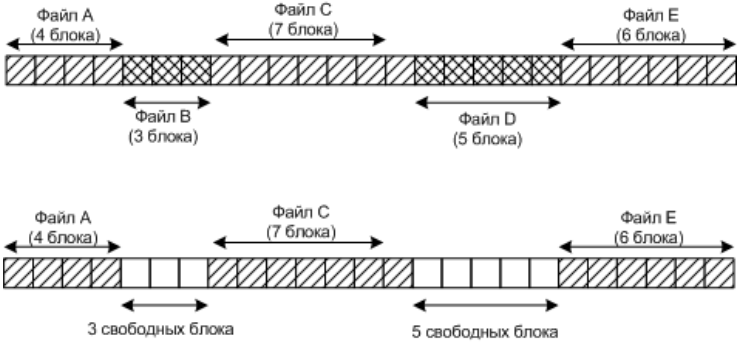
Основная проблема — эффективная адресация - сколько, и какие блоки диска принадлежат тому или иному файлу.

Возможные реализации:

- непрерывное последовательное рамещение блоков файла (cd-rom),
- связный список блоков,
- связный список индексов (fat-12),
- перечисление списка блоков (s5fs),
- перечисление кластеров (fat16, fat32, ufs),
- перечисление extend-ов (ntfs, xfs).

2.4.1. Непрерывные блоки файла.

Выделяется каждому файлу последовательность соседних блоков.



5 непрерывных файлов на диске и состояние после удаления двух файлов

Преимущества такой системы:

- Простота адресации - нужно знать всего два числа, это номер первого блока и число блоков.
- Высокая производительность - требуется только одна операция поиска, и файл может быть прочитан за одну операцию

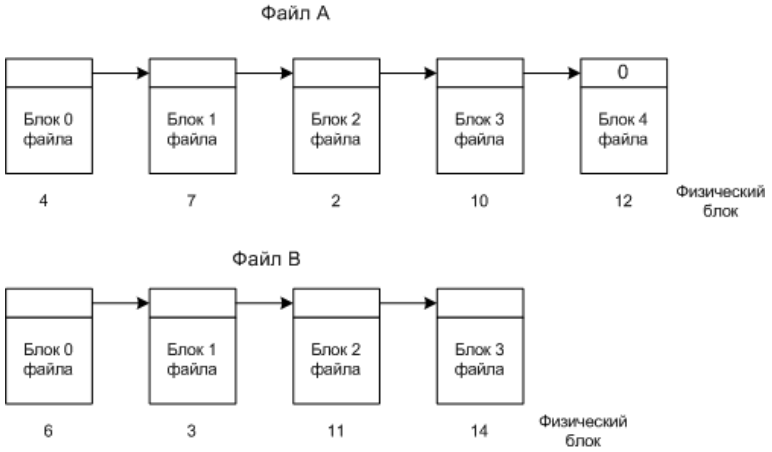
Недостатки:

- Диск сильно фрагментируется.

Сейчас такая запись почти не используется, только на CD-дисках и магнитных лентах.

2.4.2. Связные списки блоков.

Файлы хранятся в непоследовательных блоках, а с помощью связных списков можно последовательно собрать файл.



Размещение файла в виде связного списка блоков диска.

Преимущества:

- Нет потерь дискового пространства на фрагментацию.
- Нужно хранить информацию только о первом блоке.

Недостатки:

- Уменьшение быстродействия - для того чтобы дополнить файл надо перебрать все блоки.
- Уменьшается размер блока из-за хранения номера следующего блока данных.
- Сложность доступа к произвольному месту в файле.

2.4.3. Связные списки индексов (при помощи таблиц размещённых в памяти).

FAT (File Allocation Table) - таблица размещения информации о блоках файлов загружаемая в память. Рассмотри предыдущий пример, но в виде таблицы.

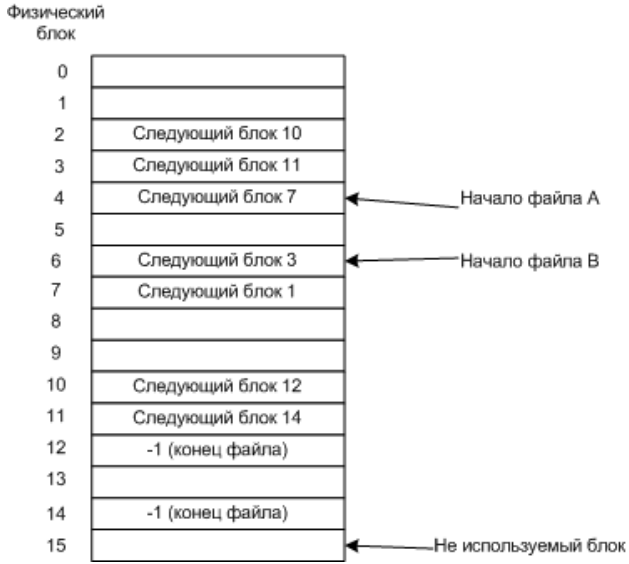


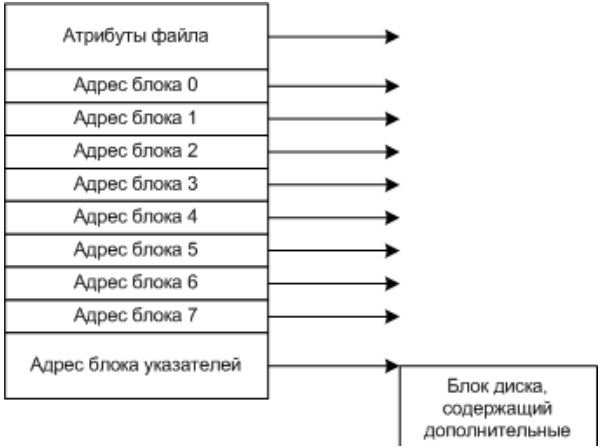
Таблица размещения файлов

Преимущества: Здесь тоже надо собирать блоки по указателям, но работает быстрее, т.к. таблица загружена в память.

Недостаток этого метода - всю таблицу надо хранить в памяти. Например, для 20Гбайт диска, с блоком 1Кбайт (20 млн. блоков), потребовалась бы таблица в 80 Мбайт (при записи в таблице в 4 байта). *Такие таблицы используются в MS-DOS и Windows.*

2.4.4. Пречисление блоков файла в i-узлах.

С каждым файлом связывается отдельная структура данных, называемая i-узлом (index-node-индекс узел), содержащие атрибуты файла и адреса всех блоков файла.



Пример i-узла

Преимущества:

- Быстродействие - имея i-узел можно получить информацию о всех блоках файла, не надо собирать указатели.
- Меньший объем, занимаемый в памяти. В память нужно загружать только те узлы, файлы которых используются.

Если каждому файлу выделять фиксированное количество адресов на диске, то со временем этого может не хватить, поэтому последняя запись в узле является указателем на дополнительный блок адресов.

Такие узлы используются в UNIX.

2.5. Реализация каталогов.

Одна из основных задач каталоговой системы преобразование ASCII-имени в информацию, необходимую для нахождения данных.

При открытии файла используется его путь и имя чтобы найти запись в каталоге указывающую на адреса блоков диска.

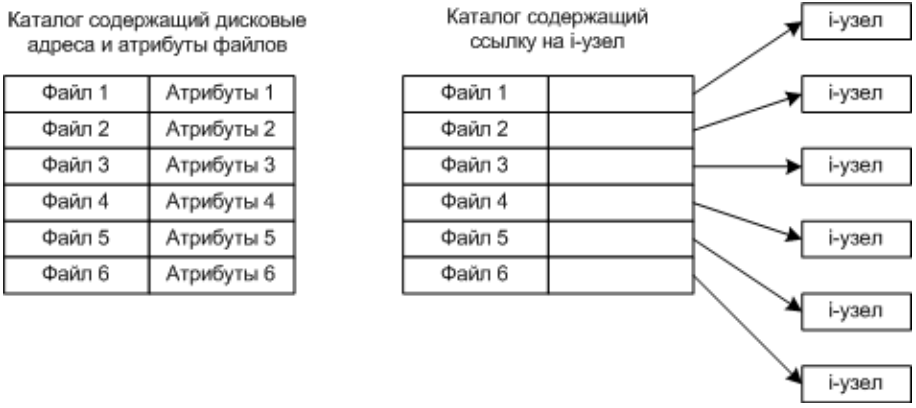
В зависимости от системы это может быть:

- дисковый адрес всего файла (для непрерывных файлов);
- номер первого блока (связные списки);
- номер i-узла.

Также каталог хранит атрибуты файлов.

Варианты хранения атрибутов:

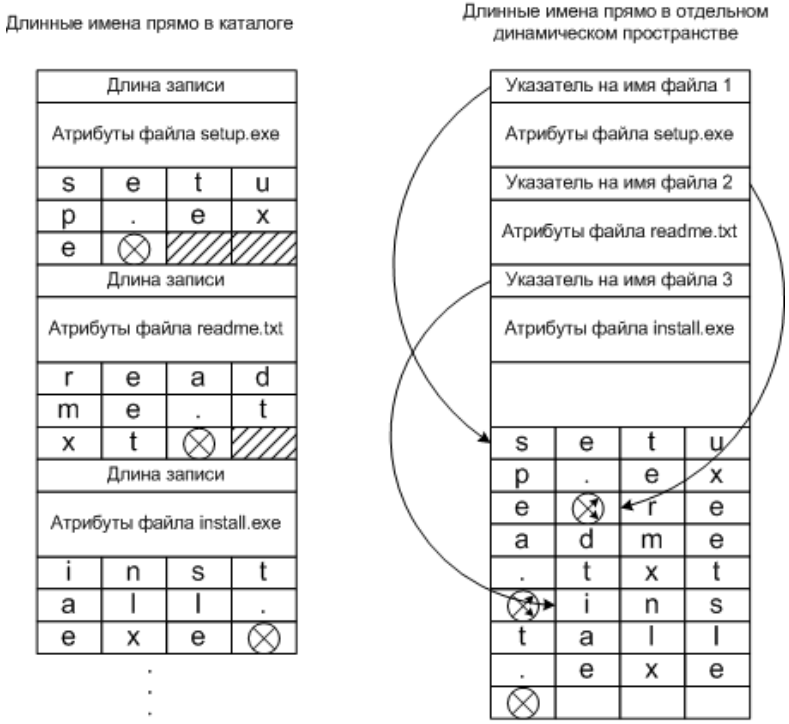
- В каталоговой записи (MS-DOS);
- В i-узлах (UNIX).



Варианты реализации хранения атрибутов файлов

2.5.1. Реализация длинных имен файлов.

Раньше операционные системы использовали короткие имена файлов, MS-DOS до 8+3 символов, в UNIX SVR4 до 14 символов. Теперь используются более длинные имена файлов (до 255 символов и больше).



Реализация длинных имен файлов

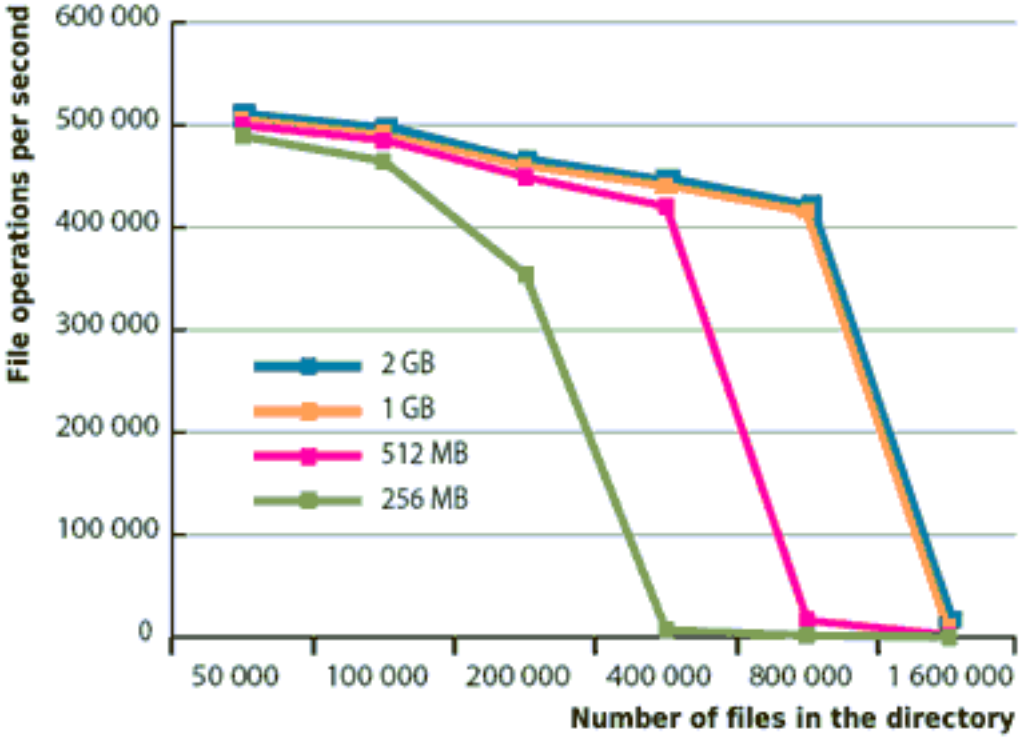
Методы реализации длинных имен файлов:

- Просто выделить место под длинные имена, увеличив записи каталога. Но это займет много места, большинство имен все же меньше 255.
- Применить записи с фиксированной частью (атрибуты) и динамической записью (имя файла).

Второй метод можно реализовать двумя методами:

- Имена записываются сразу после заголовка (длина записи и атрибутов)
- Имена записываются в конце каталога после всех заголовков (указателя на файл и атрибутов).

2.5.2. Ускорение поиска файлов.



Если каталог большой (тысячи файлов), последовательное чтение каталога мало эффективно даже с учётом кэширования в RAM. На рис. пример EXT3FS без опции dir_index.

2.5.2.1. Использование хэш-таблицы для ускорения поиска файла.

Алгоритм записи файла:

- Создается хэш-таблица в начале каталога, с размером n (n записей).
- Для каждого имени файла применяется хэш-функция, дающая при хэш-число от 0 до n-1.
- Исследуется элемент таблицы соответствующий хэш-коду.
- Если элемент не используется, туда помещается указатель на описатель файла (описатели размещены вслед за хэш-таблицей).
- Если используется, то создается связный список, объединяющие все описатели файлов с одинаковым хэш-кодом.

Алгоритм поиска файла:

- Имя файла хэшируется
- По хэш-коду определяется элемент таблицы
- Затем проверяются все описатели файла из связного списка и сравниваются с искомым именем файла посимвольно.
- Если имени файла в связном списке нет, это значит, что файла нет в каталоге.

Такой метод очень сложен в реализации, поэтому используется в тех системах, в которых ожидается, что каталоги будут содержать тысячи файлов.

2.5.2.2. Использование кэширования результатов поиска для ускорения поиска файла.

Алгоритм поиска файла:

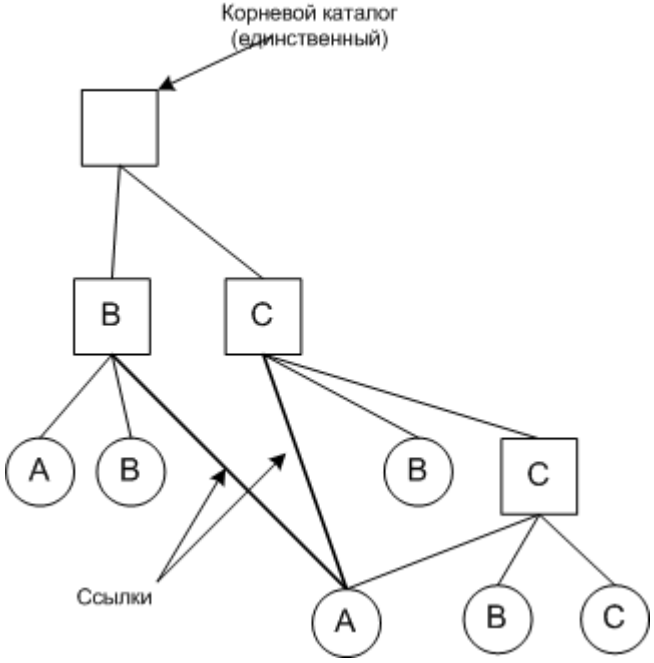
- Проверяется, нет ли имени файла в кэше
- Если нет, то ищется в каталоге, если есть, то берется из кэша

Такой способ дает ускорение только при частом использовании одних и тех же файлов.

2.6. Совместно используемые файлы.

Иногда нужно чтобы файл присутствовал в разных каталогах.

Link (связь, ссылка) - с ее помощью обеспечивается присутствие файла в разных каталогах.



A - совместно используемый файл.

Такая файловая система называется **ориентированный ациклический граф** (DAG, Directed Acyclic Graph).

Возникает проблема, если дисковые адреса содержатся в самих каталоговых записях, тогда при добавлении новых данных к совместно используемому файлу новые блоки будут числится только в каталоге того пользователя, который производил эти изменения в файле.

Есть два решения этой проблемы:

1. Использование i-узлов, в каталогах хранится только указатель на i-узел. Такие ссылки называются **жесткими ссылками**.
2. При создании ссылки, в каталоге создавать реальный Link-файл, новый файл содержит имя пути к файлу, с которым он связан. Такие ссылки называются **символьными ссылками**.

2.6.1. Жесткие ссылки.

Может возникнуть проблема, если владелец файла удалит его (и i-узел тоже), то указатель, каталога содержащего ссылку, будет указывать на не существующий i-узел. Потом может появиться i-узел с тем же номером, а значит, ссылка будет указывать на не существующий файл.

Поэтому в этом случае при удалении файла i-узел лучше не удалять.

Файл будет удален только после того, как счетчик будет равен 0.

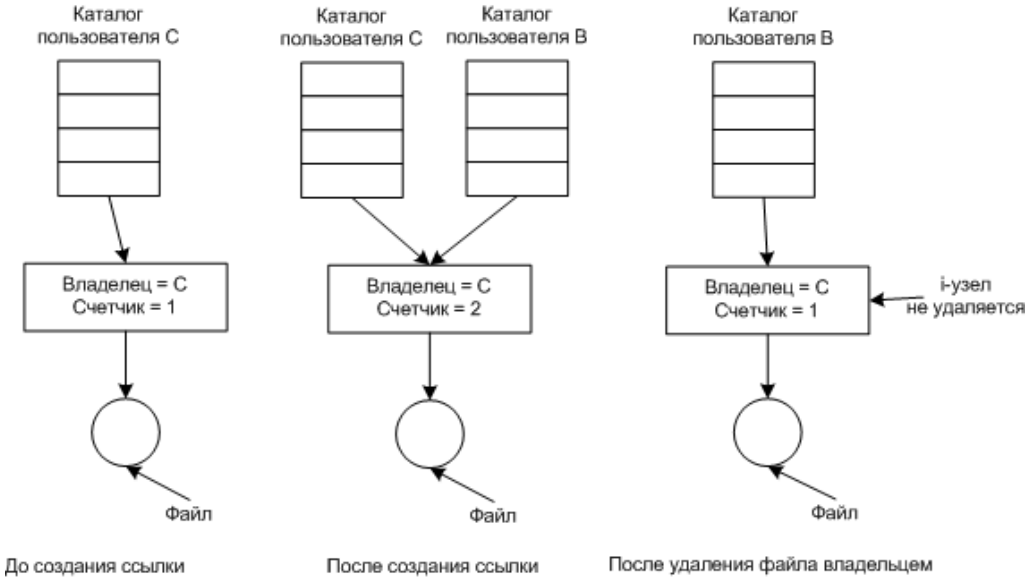


Иллюстрация проблемы, которая может возникнуть

2.6.2. Символьные ссылки.

Удаление файла не влияет на ссылку, просто по ссылке будет не возможно найти файл (путь будет не верен).

Удаление ссылки тоже никак не скажется на файле.

Но возникают накладные расходы, чтобы получить доступ к i-узлу, должны быть проделаны следующие шаги:

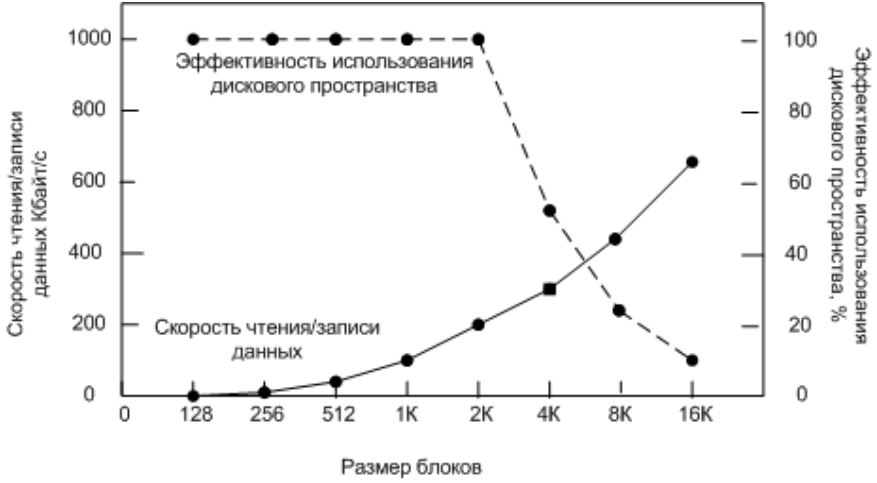
- Прочитать файл-ссылку (содержащий путь)
- Пройти по всему этому путь, открывая каталог за каталогом

2.7. Организация дискового пространства.

2.7.1. Размер блока.

При размещении файла в блоках (кластерах) возникает вопрос о размере этих блоков. Есть две крайности:

- Большие блоки - например, 1Мбайт, то файл даже 1 байт займет целый блок в 1Мбайт.
- Маленькие блоки - чтение файла состоящего из большого числа блоков будет медленным.



Скорости чтения/записи и эффективность использования диска для 2-kB файлов с разными блоками.

В UNIX системах размер блока фиксирован, и, как правило, равен от 1Кбайта до 4Кбайт.

В MS-DOS размер блока от 512 байт до 32 Кбайт в зависимости от размера диска, поэтому FAT16 использовать на дисках больше 500 Мбайт не эффективно, нужно использовать FAT32.

В NTFS размер блока фиксирован (от 512 байт до 64 Кбайт), как правило, равен примерно 2 Кбайтам.

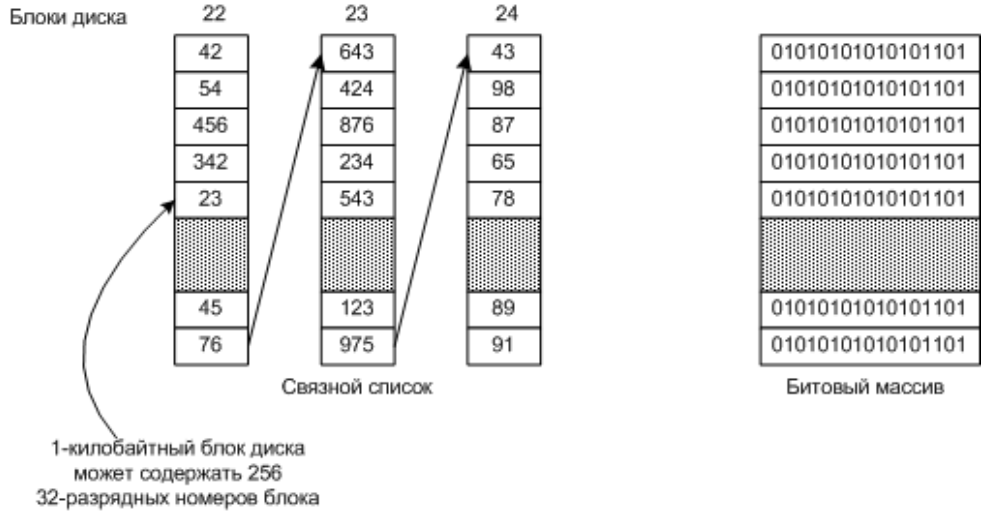
2.7.2. Учет свободных блоков.

Основные два способа учета свободных блоков:

- Битовый массив (бит-карта) - для каждого блока требуется один бит.
- Связный список блоков диска, в каждом блоке содержится столько номеров свободных блоков, сколько вмешается в блок. Часто для списка резервируется нужное число блоков в начале диска.

Недостатки:
- Требует больше места на диске, не один бит, а, например, 32 бита для номера.

- Излишние операции ввода/вывода, т.к. в памяти хранятся не все блоки, а только часть блоков.



Два способа учета свободных блоков

2.7.3. Дисковые квоты.

Чтобы ограничить пользователя, существует механизм квот.

Два вида лимитов:

- Жесткие - превышены быть не могут.
- Гибкие - могут быть превышены, но при выходе пользователь должен удалить лишние файлы. Если он не удалил, то при следующем входе получит предупреждение, после получения нескольких предупреждений дотуп блокируется.

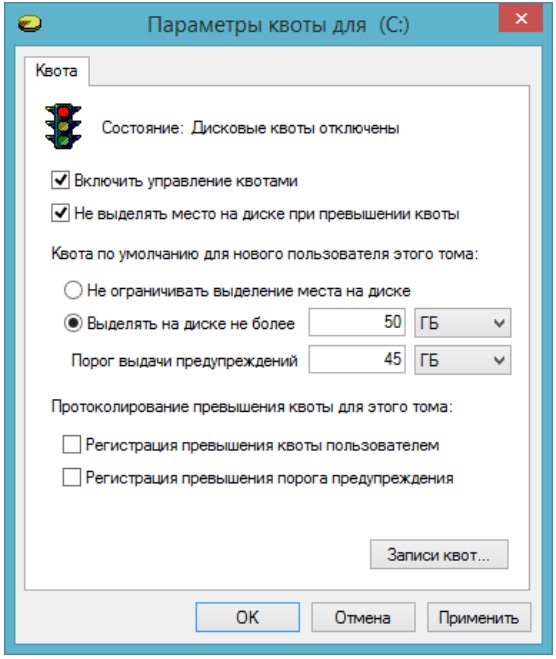
Наиболее распространенные квоты:

- Объем использования диска.
- Количество созданных файлов.
- Количество открытых файлов.
- Количество запущенных процессов.
- Количество занятых страниц памяти.

Программы для администрирования квот:

в UNIX — quotacheck, quotaon, edquota, quota;

в Windows — GUI (Свойства диска-Квоты), см. рис.



2.8. Надежность файловой системы.

2.8.1. Резервное копирование.

Случаи, для которых необходимо резервное копирование:

- Аварийные ситуации, приводящие к потере данных на диске;
- Случайное удаление или программная порча файлов.

Основные принципы создания резервных копий:

- Создавать несколько копий - ежедневные, еженедельные, ежемесячные, ежеквартальные.
- Как правило, необходимо сохранять не весь диск, а только выборочные каталоги.
- Применять **инкрементные резервные копии** - сохраняются только измененные файлы
- Сжимать резервные копии для экономии места
- Фиксировать систему при создании резервной копии, чтобы вовремя резервирования система не менялась.
- Хранить резервные копии в защищенном месте, не доступном для посторонних.

Существует две стратегии:

- Физическая архивация - поблочное копирование диска (копируются блоки, а не файлы)

Недостатки:
- копирование пустых блоков
- проблемы с дефектными блоками
- невозможно применять инкрементное копирование
- невозможно копировать отдельные каталоги и файлы

Преимущества:
- высокая скорость копирования
- простота реализации

- Логическая архивация - работает с файлами и каталогами, применяется чаще физической.

2.8.2. Непротиворечивость файловой системы.

Файловая система может попасть в **противоречивое состояние**, если в системе произойдет сбой, после занятия ресурсов, но, прежде чем модифицированный блок будет записан. Особенно если это операция занятия блока в i-узле, записи в каталоге или в списке свободных блоков.

В большинстве операционных систем есть специальные утилиты, проверяющие непротиворечивость файловых систем: в UNIX — fsck; в Linux – e2fsck, в Windows — scandisk.

Если произошел сбой, то во время загрузки утилиты проверяют файловую систему (если файловая система **журналируемая**, такая проверка не требуется).

Два типа проверки на непротиворечивость файловой системы:

1. **проверка номеров блоков** - проверяется дублирование блоков в файлах или в списке свободных блоков. Потом проверяется, нет ли блока файла, который еще присутствует в списке свободных блоков. Если блока нет в занятых и в незанятых, то блок считается **недостающим** (уменьшается место на диске), такие блоки добавляются к свободным. Также блок может оказаться в двух файлах.
2. **проверка имён файлов (или i-узлов) -** в первую очередь проверяется каталоговая структура. Файл может оказаться; либо в нескольких каталогах, либо ни в одном каталоге (уменьшается место на диске).

При монтировании файловых систем в UNIX на запись в ней устанавливается бит Dirty (грязный), который в случае краха системы не будет снят и приведёт к автоматическому запуску fsck.

2.8.3. Журналирование в файловых системах.

Любое изменение в ФС, например, создание нового файла, отражается во многих местах и представляет собой транзакцию:

- создается новая запись в директории,
- создается новый индексный дескриптор,
- блоки данных и дескриптор помечаются как зарезервированные в битовых соответств. картах,
- изменяется дата последнего доступа в дескрипторе директории,
- обновляется статистика файловой системы в суперблоке,
- сами данные записываются в соответствующие блоки.

Если во время всех этих операций произойдет отключение питания, либо случится сбой системы, то файловая система станет неполноценной, например, будет существовать дескриптор без соответствующей ему записи в директории, что в результате даст безымянный файл в директории lost+found после проверки e2fsck.

Чтобы предотвратить сбои, ОС сначала записывает изменения в журнал ФС. До тех пор пока все перечисленные групповые изменения (транзакция) не войдут в журнал, старые метаданные файловой системы останутся неизменными. Когда транзакция закончится, и новые метаданные будут в журнале, только тогда они могут быть перенесены в файловую систему при первой возможности. Если произойдёт сбой, e2fsck достаточно перенести завершенные транзакции из журнала в файловую систему (накат), чтобы обеспечить целостность и связность системы. Незавершенные транзакции в журнале игнорируются, так как старые метаданные в файловой системе остались в силе.

Режимы журналирования:

1. **writeback**: в журнал записываются только метаданные файловой системы, то есть

информация о её изменении.

Не может гарантировать целостности данных, но уже заметно сокращает время проверки по сравнению с ext2.

2. **ordered**: то же, что и writeback, но запись данных в файл производится гарантированно до записи информации об изменении этого файла.

Немного снижает производительность, также не может гарантировать целостности данных

3. **journal** – полное журналирование как метаданных ФС, так и пользовательских данных. Самый медленный, но и самый безопасный режим; может гарантировать целостность данных при хранении журнала на отдельном разделе.

Журналируемые файловые системы играет очень важную роль в операционной системе, минимизирует задержки при перезагрузке системы и **предотвращает появление ошибок в файловой системе**.

2.9. Производительность файловой системы.

Так как дисковая память достаточно медленная, то используют методы повышающие производительность.

2.9.1. Кэширование в памяти и отложенная запись на диск.

Блочный кэш (буферный кэш) - набор блоков хранящихся в памяти, но логически принадлежащих диску. Перехватываются все запросы чтения к диску, и проверяется наличие требуемых блоков в кэше.

Нужно чтобы измененные блоки периодически записывались на диск:
- в UNIX это выполняет демон update (вызывая системный вызов sync).
- в MS-DOS модифицированные блоки сразу записываются на диск.
- в NetApp применяется энергонезависимая NVRAM.

2.9.2. Опережающее чтение блока.

Если файлы считываются последовательно, и когда получен к-блок, можно считать блок к+1 (если его нет в памяти). Что увеличивает быстродействие.

2.9.3. Снижение времени перемещения блока головок.

Оптимизирующая стратегия - локализация данных и метаданных, при помощи групп блоков.

Если записывать, наиболее часто запрашиваемые файлы, рядом (соседние цилиндры или блоки), то перемещение головок будет меньше.

В случае использования i-узлов если они расположены только в начале диска, то быстродействие будет уменьшено, т.к. сначала головка считает i-узел (в начале диска), а потом будет считывать данные (где-то дальше на диске). Если располагать i-узлы в одной группе с блоками данных, то можно увеличить скорость доступа.

Файлы из одного каталога обычно связаны и часто используются совместно, поэтому желательно файлы из каталога размещать ближе друг к другу – в одной группе блоков.