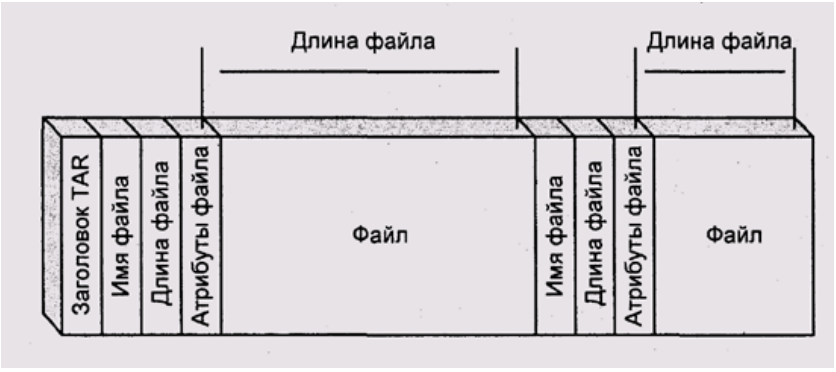
3.1 Простые файловые системы.

3.1.1. TAR FS.

Наиболее простую файловую систему создаёт архиватор UNIX — tar (Tape ARchive — архив на [магнитной] ленте).

Tar просто пишет файлы один за другим помещая в начале каждого файла заголовок с его именем, длиной и атрибутами. Аналогичную структуру имеют файлы, создаваемые архиваторами типа arj; в отличие от них, tar не сжимает файлы.



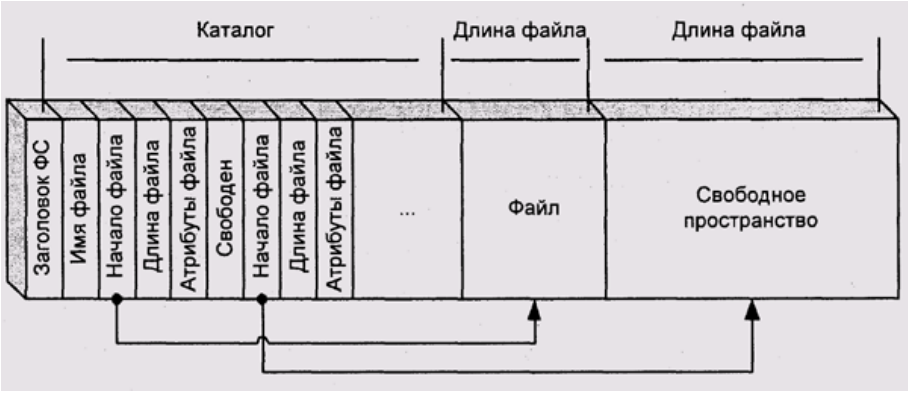
Для поиска какого-то определенного файла вы должны прочитать первый заголовок; если это не тот файл, то переместить указатель до его конца, прочитать новый заголовок и т. д. Это достаточно медленная операция поиска файла, которая используется для чтения файлов, создания файлов (нужно убедиться, что имя свободно), чтения атрибутов, изменения атрибутов.

Изменение же длины файла в середине архива или его удаление вызывает значительные трудности для ОС.

Поэтому tar используется для того чтобы собрать файлы с диска в некую единую сущность, например, для передачи по сети, или для распространения инсталляционных пакетов программ (tarball) или для резервного копирования, а для работы tar-архивы обычно распаковываются на диск или другое устройство с произвольным доступом.

3.1.2. RT-11.

Чтобы ОС при поиске файла не просматривала весь диск (ленту), можно разместить каталог файлов, как это сделано в ОС RT-11. В RT-11 помимо каталога существует понятие свободного участка диска.



В RT-11 каждому файлу и свободному участку диска выделяется непрерывная область на диске. Благодаря этому в каталоге достаточно хранить адрес первого блока файла (участка), его длину измеренную в блоках и атрибуты.

В RT-11 порядок записей в каталоге совпадает с порядком файлов и участков на диске, и началом файла считается окончание предыдущего файла. При создании файла система ищет первый свободный участок подходящего размера.

RT-11 имеет следующие недостатки:

1. При создании файла программа должна указать его длину, а это не всегда известно заранее.

2. Невозможно увеличивать размер уже созданного файла, вместо удлинения старого файла приходится создавать новый файл нужной длины и копировать содержимое старого файла в него.

3. При хаотическом создании и удалении файлов возникает проблема фрагментации свободного пространства.

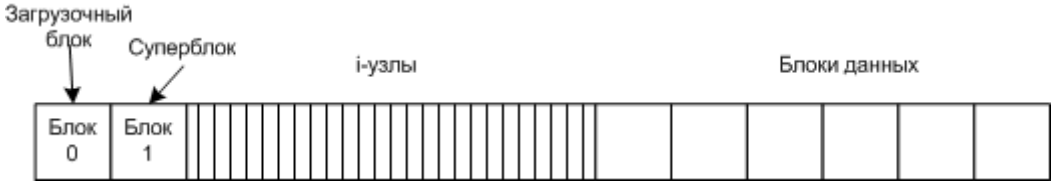
Периодически можно проводить дефрагментацию - переписывать файлы так, чтобы объединить все свободные фрагменты, но, такая работа требует много времени и опасна при сбоях.

3.2. S5FS - файловая система UNIX System 7.

Многие технические решения этой «старой» файловой системы используются в файловых системах всех современных POSIX системах — UNIX, Linux, Android, macOS, iOS.

Особенности:

- Имена файлов ограничены 14 символами ASCII, запрещены "/" и NUL-символ.
- Имеется поддержка ссылок (hard-link).
- Реализован контроль доступа к файлам и каталогам.
- Имена чувствительны к регистру, my.txt и MY.TXT это разные файлы.
- Используется схема адресации списком блоков в i-узлах, с использованием прямой и косвенной адресации.
- Не делается различий между разными типами файлов (текстовыми, двоичными и др.).
- Поддерживаются специальные символьные файлы (для символьных устройств ввода-вывода), например:
  - если открыть файл /dev/lp и записать в него символы, то символы будут печататься на принтере;
  - если открыть файл /dev/tty и прочитать из него данные, то получим символы, введенные с клавиатуры.
- Поддерживаются специальные блочные файлы (для блочных устройств ввода-вывода, например, скопировать MBR с /dev/hda: dd if=/dev/hda of=bootloader.mbr bs=512 count=1).
- Позволяет монтировать разделы в любое место дерева системы.



Структура файловой системы S5FS

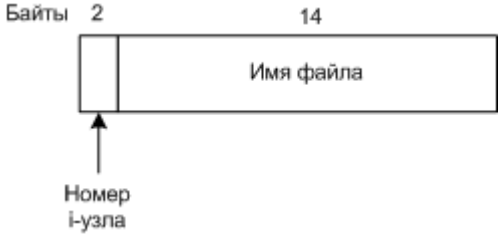
Суперблок содержит:

- Количество i-узлов
- Количество дисковых блоков
- Начало списка свободных блоков диска

При уничтожении суперблока, файловая система становится нечитаемой.

Каждый i-узел имеет 64 байта в длину и описывает один файл (в том числе каталог).

Каталог содержит по одной записи для каждого файла.



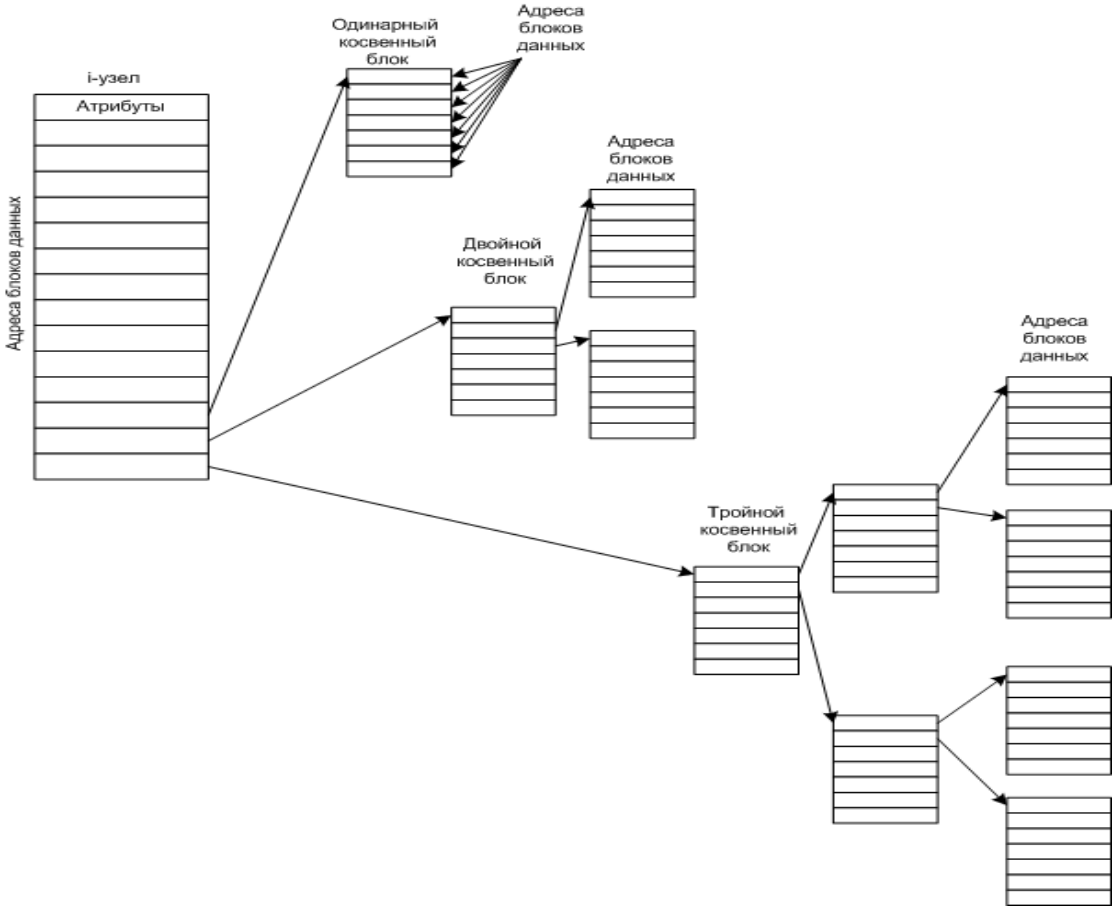
Каталоговая запись S5FS в 16 байт

Структура i-узла

Поле	Байты	Описание
Mode	2	Тип файла, биты защиты, биты setuid и setgid
Nlinks	2	Количество каталоговых записей, указывающий на этот i-узел
Uid	2	Идентификатор (номер) владельца
Gid	2	Идентификатор (номер) группы
Size	4	Размер файла в байтах
DirAddr	30	Адреса первых 10 дисковых блоков файла
IndirAddr	9	Адреса первичного, вторичного, третичного косвенных блоков
Atime	4	Время последнего доступа файла
Mtime	4	Время последнего изменения файла
Ctime	4	Время последнего изменения i-узла

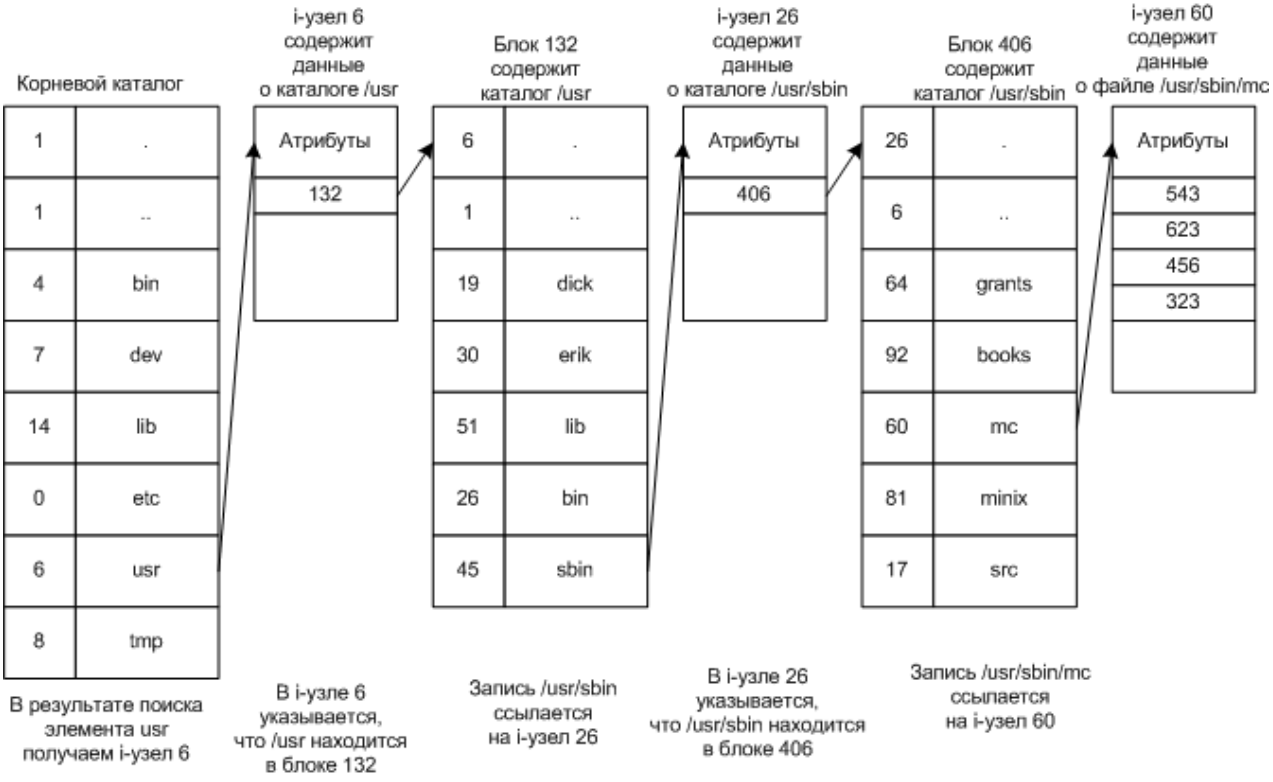
Номера первых 10 блоков файла хранятся в самом i-узле, при блоке в 1 Кбайт, файл до 10Кбайт. Дополнительные блоки для i-узла, в случае больших файлов:

- **Первичный косвенный блок -** дополнительный блок с адресами блоков файла, если файл большой, то эта запись в i-узле указывает на дополнительный блок с адресами. Файл может быть 266Кбайт=10Кбайт+256Кбайт (256Кбайт <= 256 (2^8)-адресов блоков = 1Кбайт-размер блока / 4байта-размер адреса).
- **Вторичный косвенный блок** - дополнительный блок с адресами одинарных косвенных блоков, если одного дополнительного блока не хватает. Файл может быть 65Мбайт=10Кбайт+2^8Кбайт+2^16Кбайт.
- **Третичный косвенный блок** - дополнительный блок с адресами двойных косвенных блоков, если одного одинарного косвенного блока не хватает. Файл может быть 16Гбайт=10Кбайт+2^8Кбайт+2^16Кбайт+2^24Кбайт.


i-узел
Атрибуты
Адреса блоков данных
Одинарный косвенный блок
Адреса блоков данных
Двойной косвенный блок
Адреса блоков данных
Тройной косвенный блок
Адреса блоков данных

i-узел S5FS

3.2.1. Поиск файла.



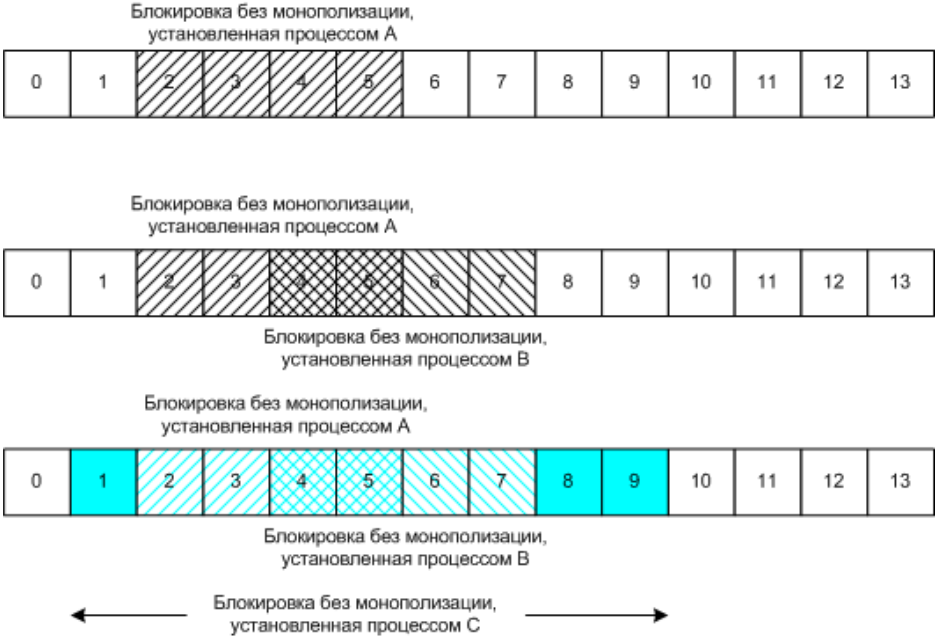
Этапы поиска файла по абсолютному пути /usr/sbin/mc

При использовании относительного пути, например sbin/mc, поиск начинается с рабочего каталога /usr.

3.2.2. Блокировка данных файла.

Блокирование осуществляется поблочно. Стандарт POSIX описывает два типа блокировки:

- Блокировка с монополизацией - больше ни один процесс эти блоки блокировать не может.
- Блокировка без монополизации - могут блокировать и другие процессы.



Блокировки данных файла без монополизации

Если процесс **К** попытается блокировать блок 6 с монополизацией, то сам процесс будет заблокирован до разблокирования блока 6 всеми процессами.

3.2.3 Создание и работа с файлом.

fd=creat("abc", mode) - Пример создания файла **abc** с режимом защиты, указанном в переменной **mode** (какие пользователи имеют доступ). Используется системный вызов creat.

Успешный вызов возвращает целое число fd - **дескриптор файла**.

Который хранится в **таблице дескрипторов файла**, открывшего процесса.

После этого можно работать с файлом, используя системные вызовы **write** и **read**.

n=read(fd, buffer, nbytes)

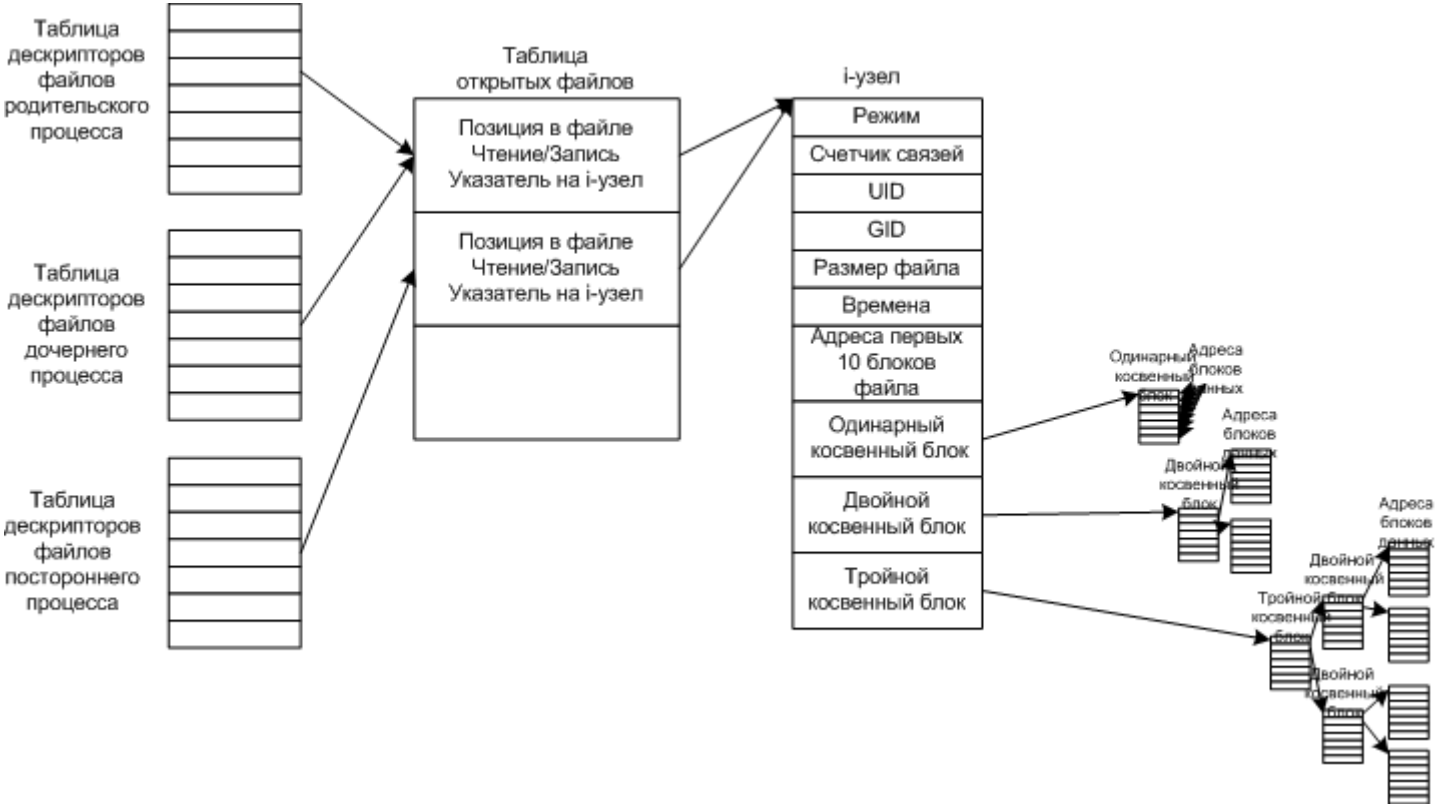
n=write(fd, buffer, nbytes)

У обоих вызовов всего по три параметра:

- fd - дескриптор файла, указывающий на открытый файл
- buffer - адрес буфера, куда писать или откуда читать данные
- nbytes - счетчик байтов, сколько прочитать или записать байт

Теперь нужно по дескриптору получить указатель на i-узел и указатель на позицию в файле для записи или чтения.

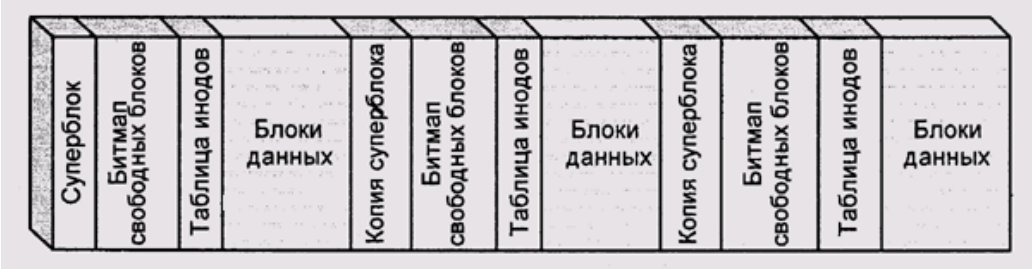
Таблица открытых файлов - создана для хранения указателей на i-узел и на позицию в файле. И позволяет родительскому и дочернему процессам совместно использовать один указатель в файле, но для посторонних процессов выделять отдельные указатели.



Связь между таблицей дескрипторов файлов, таблицей открытых файлов и таблицей i-узлов.

3.3. FFS - файловая система BSD.

Основу составляет классическая файловая система UNIX.

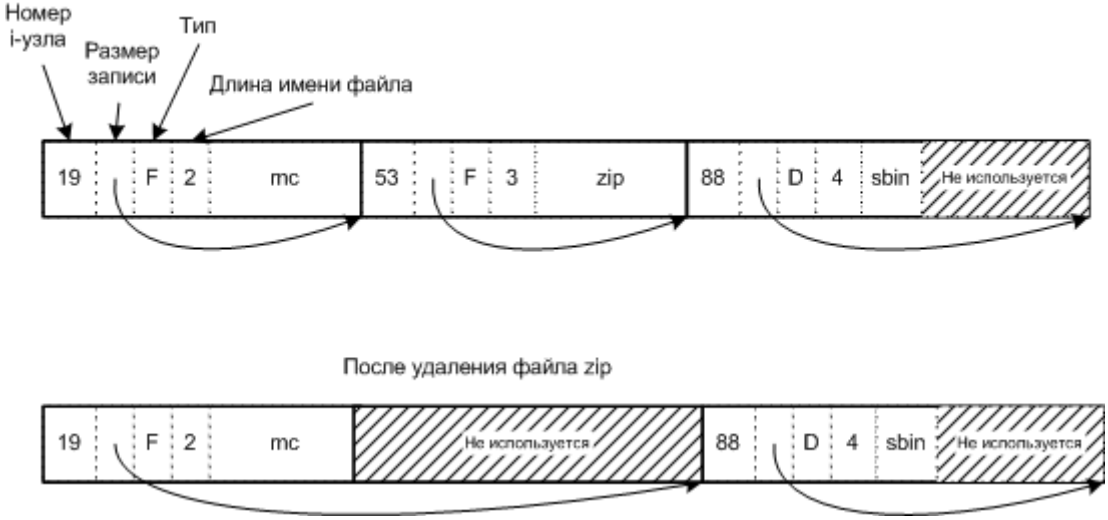


Особенности (отличие от предыдущей системы):

- Увеличена длина имени файла до 255 символов
- Реорганизованы каталоги
- Было добавлено кэширование имен файлов, для увеличения производительности.
- Применено разбиение диска на группы цилиндров, чтобы i-узлы и блоки данных были поближе друг к другу, для каждой группы были свои:
  - суперблок
  - i-узлы
  - блоки данных.

  Это сделано для уменьшения перемещений галовок.
- Используются блоки двух размеров, для больших файлов использовались большие блоки, для маленьких маленькие.

Каталоговые записи никак не отсортированы и следуют друг за другом.



Каталог FFS с тремя каталоговыми записями для трех файлов и тот же каталог после удаления файла zip, увеличивается длина первой записи.

3.4. Файловые системы LINUX.

Изначально использовалась файловая система **MINIX** с ограничениями: 14 символов для имени файла и размер файла 64 Мбайта.

После была создана файловая система **EXT** с расширением: 255 символов для имени файла и размер файла 2Гбайта.

Система была достаточно медленной, но, быстро эволюционировала до EXT2 и дальше.

3.4.1 Файловая система EXT2

Эта файловая система стала основой для LINUX, она очень похожа на BSD FFS систему.

Вместо групп цилиндров используются группы блоков.

Размещение файловой системы EXT2 на диске

Суперблок содержит:

Значительная часть информации одинакова для всех суперблоков, что позволяет восстанавливаться после сбоев.

Magis Number – 0xEF53 (тип ФС чтобы отличать ext2, ext3, ext4)

Revision Level – правка ошибок

Mount Count и Max Mount Count – для проверки целостности

Block Group Number – номер группы которой принадлежит данный суперблок

Block Size – 1024 байта

Block Per Group – количество блоков в группе, при создании ФС, постоянное число

Free Blocks – количество свободных блоков в ФС

Free Inodes – количество свободных инодов в ФС

First Inode – номер первого инода в этой группе

Group Descriptor содержит Free Blocks Count, Free Inodes Count, Used Directory Count

Block Bitmap размером 1 кластер (при 1KB кластере можно отразить 8192 блоков в группе)

Inode Bitmap размером 1 кластер (при 1KB кластере можно отразить 8192 инодов в группе)

Catalog Count – чтобы распределить каталоги по всему разделу равномерно

Inode Table содержит дескрипторы размером по 128 байт, похожие на S5FS. 12 прямых указателей, 3 косвенных, длина указателей по 4 байта ➔ 2^32 адресуемых блоков. Остаток записи в инодах зарезервирован на расширения.

Директории как в FFS в виде связного списка записей переменной длины. Запись содержит номер inode, длину записи, длину имени, имя файла. Первые две записи это . и ..

Другие особенности:

1) Поддерживаются регулярные файлы, директории, спецфайлы устройств, символьные ссылки, программные каналы.
2) Длина имени файла 255 символов, имя пути до 1012 символов.
3) Размер блока 1 Кбайт
4) Размер каждого i-узла 128 байт.
5) i-узел содержит 12 прямых и 3 косвенных адресов, длина адреса в i-узле стала 4 байта, что обеспечивает поддержку размера файла чуть более 16Гбайт.
6) Максимальный размер раздела 4 TB.
7) Особенности работы файловой системы благодаря которым её работа не способствует фрагментации файлов:
   - ✓ Создание новых каталогов распределяется равномерно по группам блоков, чтобы в каждой группе было одинаковое количество каталогов.
   - ✓ Новые файлы старается создавать в группе, где и находится каталог.
   - ✓ При увеличении файла система старается новые блоки записывать ближе к старым.

8) По умолчанию резервируется до 5% блоков для root (по умолчанию) для восстановления ФС при её полном заполнении пользовательскими и журнальными файлами. Меняется командой tune2fs.

9) Возможность выбора семантики BSD или SystemVR4 при монтировании. BSD-семантика – файлы создаются с GID родительского каталога, SVR4-семантика – комплексная, если директория имеет SGID-бит, то файлы принимают GID от директории, а поддиректории также наследуют GID и SGID-бит. Если SGID не установлен, то файлы наследуют GID от процесса, создавшего их.

10) EXT2FS следит за состоянием файловой системы. Когда ФС монтируется в режиме read/write, то в суперблоке устанавливается специальный флаг «Not Clean». При размонтировании и монтировании в режиме read only, этот флаг устанавливаетс в «Clean». Вов ремя загрузки наличие «Not Clean» говорит о сбое в последнем сеансе работы и проводится проверка целостности ФС командой e2fsck.

11) Долгий пропуск проверок целостности может скрывать тонкие сбои ФС, поэтому в суперблоке запоминается дата последней проверки и ведётся счётчик монтирований ФС, растущий при любом режиме монтирования (и при RW и при RO). При достижении Max Mout Count запускается e2fsck. При наступлении события Max Day check Interval запускается e2fsck.

12) Можно установить режим безопасного удаления файлов, при котором блоки ранее принадлежащие файлу заполняются мусором.

13) Есть дополнительные атрибуты для файлов и каталогов (Append Only, Immunization и др.).

Команды для работы с ФС.

Для файловых системах EXT2, EXT3, EXT4 и других

mount, umount – монтирование и размонтирование

df, /etc/fstab, /etc/mtab – просмотр

mke2fs /dev/hdb2 [-b 1024|...|4096] – создание

stat /etc/passwd – информация об иноде

stat -f /etc/passwd – информация о ФС

#stat –f /tmp

ID:0 0 Namelen:255 Type EXT2

Blocks: Total:721445 Free:585216 Available:548569 Size:4096

Inodes: Total:366528 Free:316752

где ID – владелец и группа зарезервированного пространства 5%, Namelen – max имя файла

tune2fs, dump2fs, debugfs – информация и настройка ФС

#debugfs –R stats /ev/sda1

#tune2fs –l /dev/sda1

#dump2fs –l /dev/sda1

выдадут по несколько страниц информации.

3.4.2 Файловая система EXT3

В отличие от EXT2, **EXT3** является **журналируемой** файловой системой, т.е. не попадет в противоречивое состояние после сбоев. Но она полностью совместима с EXT2.

Разработанная в Red Hat. В данный момент является основной для LINUX.

Драйвер Ext3 хранит полные точные копии модифицируемых блоков (1КБ, 2КБ или 4КБ) в памяти до завершения операции. Это может показаться расточительным. Полные блоки содержат не только изменившиеся данные, но и не модифицированные.

Такой подход называется "**физическим журналированием**", что отражает использование "физических блоков" как основную единицу ведения журнала. Подход, когда хранятся только изменяемые байты, а не целые блоки, называется "**логическим журналированием**" (используется XFS). Поскольку ext3 использует "физическое журналирование", журнал в ext3 имеет размер больший, чем в XFS. За счет использования в ext3 полных блоков, как драйвером, так и подсистемой журналирования нет сложностей, которые возникают при "логическом журналировании".

Типы журналирования поддерживаемые Ext3, которые могут быть активированы из файла /etc/fstab:

- **data=journal** (full data journaling mode) - все новые данные сначала пишутся в журнал и только после этого переносятся на свое постоянное место. В случае аварийного отказа журнал можно повторно перечитать, приведя данные и метаданные в непротиворечивое состояние. Самый медленный, но самый надежный.
- **data=ordered** - записываются изменения только мета-данных файловой системы, но логически metadata и data блоки группируются в единый модуль, называемый transaction. Перед записью новых метаданных на диск, связанные data блоки записываются первыми. Этот режим

журналирования ext3 установлен по умолчанию.
При добавлении данных в конец файла режим data=ordered гарантированно обеспечивает целостность (как при full data journaling mode). Однако если данные в файл пишутся поверх существующих, то есть вероятность перемешивания "оригинальных" блоков с модифицированными. Это результат того, что data=ordered не отслеживает записи, при которых новый блок ложится поверх существующего и не вызывает модификации метаданных.

- **data=writeback** (metadata only) - записываются только изменения мета-данных файловой системы. Самый быстрый метод журналирования. С подобным видом журналирования вы имеете дело в файловых системах XFS, JFS и ReiserFS.

3.4.3 Файловая система XFS

XFS - журналируемая файловая система разработанная Silicon Graphics, но сейчас выпущенная открытым кодом (open source).

Официальная информация на http://oss.sgi.com/projects/xfs/

XFS была создана в начале 90-ых (1992-1993) фирмой Silicon Grapgics (сейчас SGI) для мультимедийных компьютеров с ОС Irix. Файловая система была ориентирована на очень большие файлы и файловые системы. Особенностью этой файловой системы является устройство журнала - в журнал пишется часть метаданных самой файловой системы таким образом, что весь процесс восстановления сводится к копированию этих данных из журнала в файловую систему. Размер журнала задается при создании системы, он должен быть не меньше 32 мегабайт; а больше и не надо - такое количество незакрытых транзакций тяжело получить.

Некоторые особенности:

- Более эффективно работает с большими файлами.

- Имеет возможность выноса журнала на другой диск, для повышения производительности.
- Сохраняет данные кэша только при переполнении памяти, а не периодически как остальные.
- В журнал записываются только мета-данные.
- Используются B+ trees для каталогов.
- Используется логическое журналирование

3.4.4 Файловая система RFS

RFS (RaiserFS) - журналируемая файловая система разработанная Namesys. Официальная информация на RaiserFS .

Некоторые особенности:

- Более эффективно работает с большим количеством мелких файлов, в плане производительности и эффективности использования дискового пространства.
- Использует оптимизированные b* balanced tree (усовершенствованная версия B+ дерева)
- Динамически ассигнует i-узлы вместо их статического набора, образующегося при создании "традиционной" файловой системы.
- Поддерживает динамические размеры блоков.

3.4.4 Файловая система JFS

JFS (Journaled File System) - журналируемая файловая система разработанная IBM для ОС AIX, но сейчас выпущенная как открытый код. Официальная информация на Journaled File System Technology for Linux

Некоторые особенности:

- Журналы JFS соответствуют классической модели транзакций, принятой в базах данных
- В журнал записываются только мета-данные
- Размер журнала не больше 32 мегабайт.
- Асинхронный режим записи в журнал - производится в моменты уменьшения трафика ввода/вывода
- Используется логическое журналирование.

3.5. Сравнительная таблица современных файловых систем.

Характеристика	S5FS, MinixFS	NTFS	EXT4	RFS	XFS	JFS
Хранение информации о файлах	inode	MFT	inode	inode	inode	inode
Максимальный размер раздела	64 MB	16 Эбайт (2^{60})	1 Эбайт	4 гигаблоков (т.к. блоки динамические до 256 Тбайт	16 Эбайт	32 Пбайт (2^{50})
Размеры блоков	512 B	от 512 байт до 64 Кбайт	1 Кбайт – 4 Кбайт	До 64 Кбайт (сейчас фиксированы 4 Кбайт)	от 512 байт до 64 Кбайт	512/1024/ 2048/4096 байт
Максимальное число блоков		2^48	2^32			2^32
Максимальный размер файла	64 MB	2^64	16 Тбайт (для 4KB блоков)	8 Тбайт	8 Эбайт	4 Пбайт (2^{50})
Максимальная длина имени файла	14	255	255			
Журналирование	Нет	Да	Да	Да	Да	Да
Управление свободными блоками	Нет		На основе битовой карты	На основе битовой карты	B-деревья, индексиро-ванные по смещению и по размеру	Дерево+ Binary Buddy

Экстенты для свободного пространства	Нет		Нет	Нет	Да	Нет
B-деревья для элементов каталогов	Нет	Да	Нет	Как поддерево основного дерева файл. системы	Да	Да
B-деревья для адресации блоков файлов	Нет		Нет	Внутри основ-ного дерева файловой системы	Да	Да
Экстенты для адресации блоков файлов	Нет		Нет	Да (с 4 версии)	Да	Да
Данные внутри inode (небольшие файлы)	Нет		Нет	Да	Да	Нет
Данные симво-льных ссылок внутри inode	Да		Да	Да	Да	Да
Элементы каталогов внутри inode (небольшие каталоги)	Нет		Нет	Да	Да	Да
Динамическое выделение inode/MFT	Нет	Да	Нет	Да	Да	Да
Структуры управления динамически выделяемыми inode	Нет		Нет	Общее B*дерево	B+дерево	B+дерево с непрерыв-ными областями inode
Поддержка разреженных файлов	Нет	Да	Нет	Да	Да	Да

The physical size of Big Data

Name	Symbol	Binary	Decimal
byte	B	2^0=1 byte	10^0=1
kilobyte	KB	2^{10}=1.024 byte (B)	10^3=1.000
megabyte	MB	2^{20}=1.048.576 B	10^6=1.000.000
gigabyte	GB	2^{30}=1.073.741.824 B	10^9=1.000.000.000
terabyte	TB	2^{40}=1.099.511.627.776 B	10^{12}=1.000.000.000.000
petabyte	PB	2^{50}=1.125.899.906.842.624 B	10^{15}=1.000.000.000.000.000
exabyte	EB	2^{60}=1.152.921.504.606.846.976 B	10^{18}=1.000.000.000.000.000.000
zettabyte	ZB	2^{70}=1.180.591.620.717.411.303.424 B	10^{21}=1.000.000.000.000.000.000.000
yottabyte	YB	2^{80}=1.208.925.819.614.629.174.706.176 B	10^{24}=1.000.000.000.000.000.000.000.000
brontobyte	BB	2^{90}=1.237.940.039.285.380.274.899.124.224 B	10^{27}=1.000.000.000.000.000.000.000.000.000
geopbyte	GeB	2^{100}=1.267.650.600.228.229.401.496.703.205.376 B	10^{30}=1.000.000.000.000.000.000.000.000.000.000

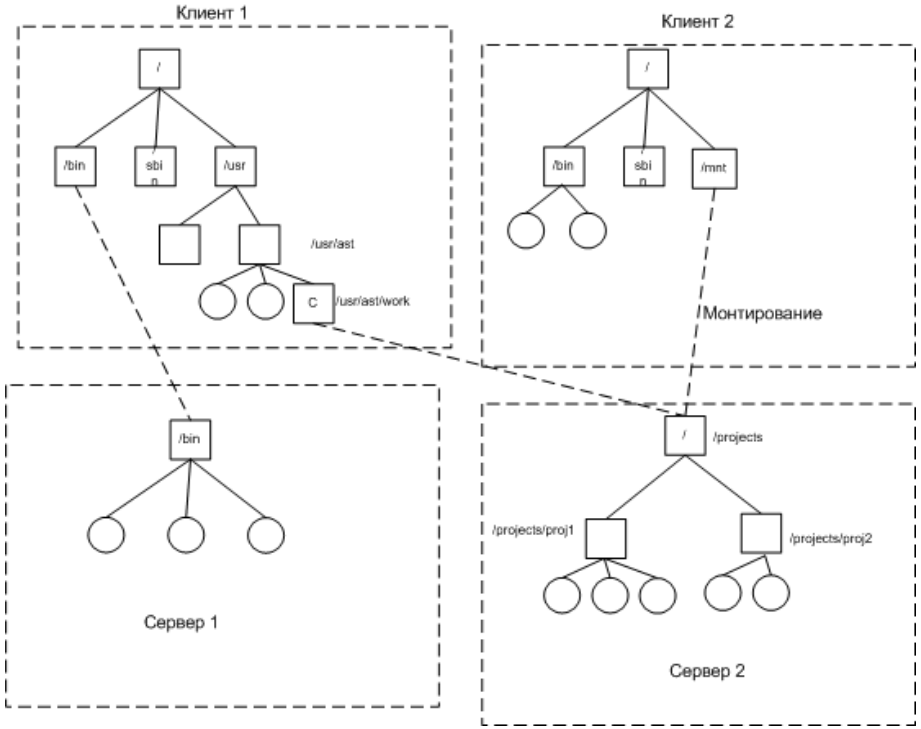
3.6. Файловая система NFS.

NFS (Network File System) - сетевая файловая система. Создана для объединения файловых систем по сети.

3.6.1 Архитектура файловой системы NFS

Предоставляется доступ к каталогу (экспортируется) с подкаталогами. Информация об экспортируемых каталогах хранится в /etc/exports. При подключении эти каталоги монтируются к локальной файловой системе.

Примеры монтирования удаленных файловых систем →



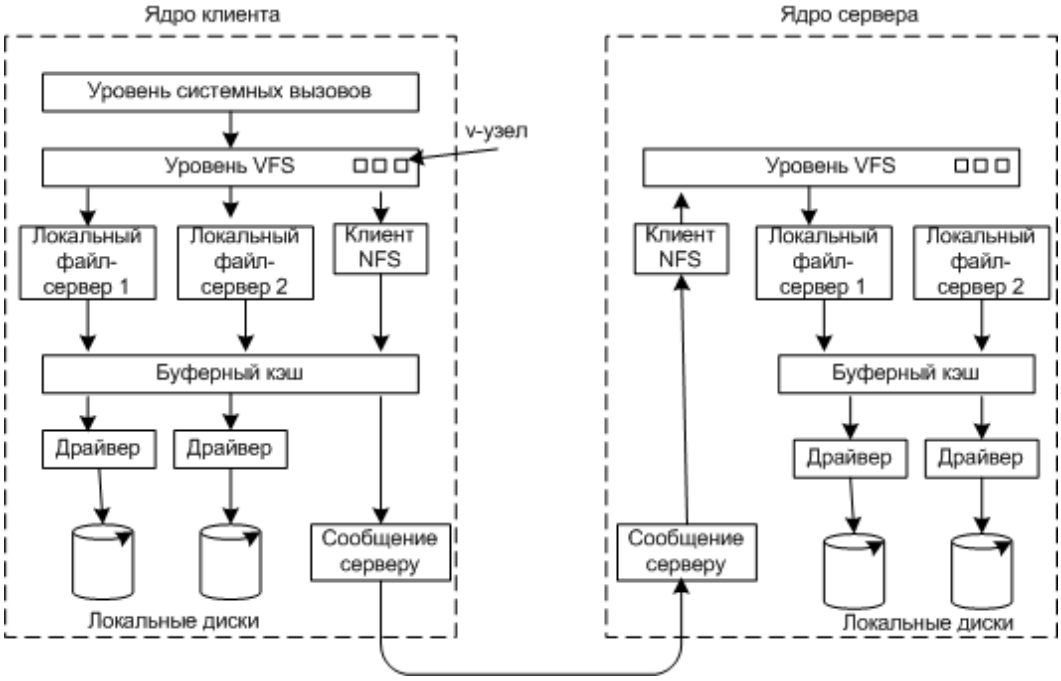
3.6.2 Протоколы файловой системы NFS

Протокол - набор запросов и ответов, клиента и сервера.

Используется два протокола:

1. Протокол управления монтирования каталогов
2. Протокол управления доступа к каталогам и файлам

3.6.3 Реализация файловой системы NFS



Структура уровней файловой системы NFS

3.6.4. VFS (Virtual File System).

VFS - виртуальная файловая система. Необходима для управления таблицей открытых файлов. VFS используется не только для NFS, но и для работы с инородными файловыми системами (FAT, /proc и т.д.) Записи для каждого открытого файла называются **v-узлами** (virtual i-node).

Алгоритм работы NFS (последовательность системных вызовов mount, open и read):

1. Вызывается программа mount, ей указывается удаленный каталог и локальный каталог для монтирования.
2. Программа ищет сервер, соединяется с ним.
3. Запрашивает дескриптор каталога.
4. Программа mount обращается к системному вызову mount для монтирования полученного каталога.
5. Ядро формирует v-узел для открытого удаленного каталога.
6. Ядро формирует **r-узел** (удаленный i-узел) для удаленного каталога в своих внутренних таблицах. В результате v-узел указывает либо на r-узел для удаленного каталога, либо на i-узел одной из локальных файловых систем.
7. Система просит программу клиента NFS открыть файл.
8. Создаются v-узел и r-узел для удаленного файла.
9. Вызывающему процессу выдается дескриптор удаленного файла.
10. Теперь этот процесс может работать с файлом, используя вызов read