Файловая подсистема.
С неё начали писать UNIX (показатель важности);

Файл – важнейшее понятие в файловой подсистеме;

Файл (из Оксфордского словаря) – информация, хранимая во внешней памяти или во вспомогательном ЗУ с целью её сохранения после завершения отдельного задания или преодоления ограничений, связанных с объёмом основного ЗУ;

Задача ФС – обеспечивать сохранение данных и доступ к сохранённым данным.

Файл – поименованная сов-ть данных, хранимая во вторичной памяти (возможно далее целая);

Доступ к файлу осущ. по его имени;

Чтобы обеспечить хранение файла и последующий доступ к нему, ф.с. должна обеспечить изолированность файлов;

Файл должен быть изолирован, т.е. занимать некоторое адр. пр-во, и это адр. пр-во должно быть защищено;

Более того, необходимо обеспечить доступ к файлу, и он обеспечивается по тому, как файлы идентифицируются в системе;

Файловая система – порядок, определяющий способ организации хранения, именования и доступа к данным на вторичных носителях информации;

Вторичная память (стор.)

Дисковая память

долговременное хранение данных

Вся жизнь наша построена на железе, и это железо мы должны понимать

В наших компьютерах все ЗУ – блочные;

В системе 2 типа устройств: символьные и блочные;

Цитата из Оксфордского словаря:

File management (управление файлами) – программные процессы, связанные с общим управлением файлами, т.е. с размещением во вторичной памяти, контролем доступа к файлам, записью резервных копий, ведением справочников (directory);

Осн. ф-ции упр-я файлами обычно выполняются ОС, а дополнительные – системами управления файлами;

Доступ к файлам: open, read, write, rename, delete, remove и т.д.

В UNIX всё файл. Если что-то не файл, то это процесс

→ Справедливо, т.к. в системе имеются спец. файлы, про которые говорят, что они больше, чем файл: программные каналы, сокеты, внеш. устр-ва;

Файловая система работает с регулярными (обычными) файлами и директориями;

При этом UNIX и Linux не делают разницы между файлами и директориями (каталогами);

Директория - файл, который содержит имена других файлов;

7 типов файлов в UNIX:
- "-" - обычный файл (regular);
- "d" - directory;
- "l" - soft link;
- "c" - special character device;
- "d" - block device;
- "s" - socket;
- "p" - named pipe;

∃ Стандарт Filesystem Hierarchy Standart (FHS), который определяет структуру и содержимое каталогов в Linux distribution (Ubuntu поддерживает этот стандарт)

По этому стандарту корень ф. с. обозначается как "/" (корневой каталог) и его ветви обязательно должны составлять единую ф. с., расположенную на одном носителе (диске или дисковом разделе). В нём должны располагаться все компоненты, необходимые для старта системы;

Другое опр-е файла:

Файл - каждая индивидуально идентифицированная единица информации;

Задачи Ф.С.:

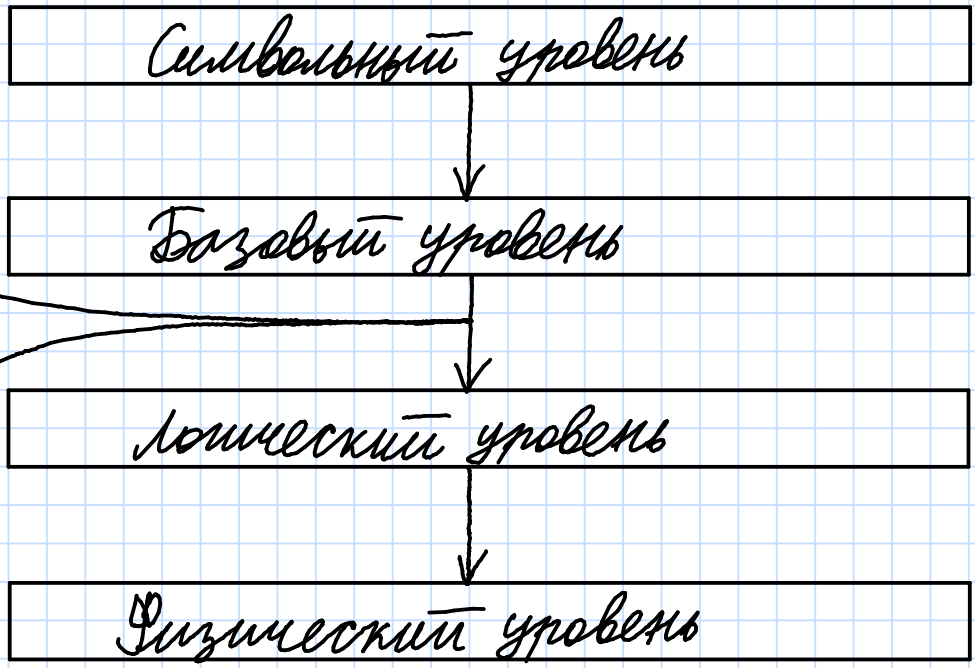
1) Именование файлов;
2) Обеспечение программного интерфейса для работы с файлами пользователей или с приложениями;

Ни ОС, ни ФС никак не интерпретирует содержимое файла;

3) Отображение логической модели (логического представления файлов на физическую организацию хранения данных на соотв. носителях;
4) Обеспечивает надёжное хранение файлов, доступ к ним и защиту от несанкционированного доступа (права доступа в Linux)
5) Обеспечение совм. использования файлов;

(м. б. не в полной мере задача ф. с., но задача ОС)

Иерархическая структура ФС



Уровень именования файлов

В UNIX файл идентифицируется inode
номер inode ~ pid
struct inode ~ struct task_struct

проверка прав доступа
(комплексная задача, решается по-разному для разных типов файлов)

Уровень физ. хранения данных определяется особенностями аппаратных средств (дисками)

файл может занимать разные блоки на диске

(зависит от организации физ. памяти)

В UNIX/Linux файл может иметь много имён (hard link);

В системе поддерживаются байт-ориентированные (direct-oriented) и блок-ориентированные (block-oriented) файлы;

Символьные файлы ~ байт-ориентированные файлы;
(1 символ определяется 1 байтом)
Блочные файлы ~ блок-ориентированные файлы;

Уровень иерархии памяти определяется близостью к процессору;

Файл ≠ место на диске

В мире совр. вычисл. техники файлы имеют настолько большие размеры, что не могут хра-

нимься в непр. физ. адр. пр-ве, они хранятся вразброс;
Файл может занимать разные блоки/сектора/дорожки на диске аналогично тому, как память поделена на страницы; в любой момент м.б. загружена новая страница (как и файл);
Также важно понимать, как это всё адресуется;

Для файлов характерно несвязанное распределение: файлу выделяются участки памяти вразброс;
Соответственно, система должна обеспечить адресацию каждого такого участка;

Файловая система Linux.
Отличается от Ф.С. UNIX предоставляемым интерфейсом: ОС UNIX и Linux позволяют работать с большим кол-вом ф. с.;

Для этого UNIX предоставляет интерфейс VFS (virtual) / vnode, а в Linux – VFS (отказались от vnode);
и там, и там файл идентифицируется номером inode

/root
/bin /boot /etc /usr /var /dev ... /home

„Родные" ФС для Linux – ext (ext2, ext4, ...)

ext2, ext4, XFS, UFS, NTFS, MS-DOS, FAT, FAT32, MPFS, NFS;

Фактически VFS - интерфейс, с помощью которого ОС может работать с большим кол-вом файловых систем;

Основой такой работы (базовым действием) явл. монтирование: прежде чем ф.с. станет доступна (мы сможем увидеть её каталоги и файлы) она должна быть смонтирована;

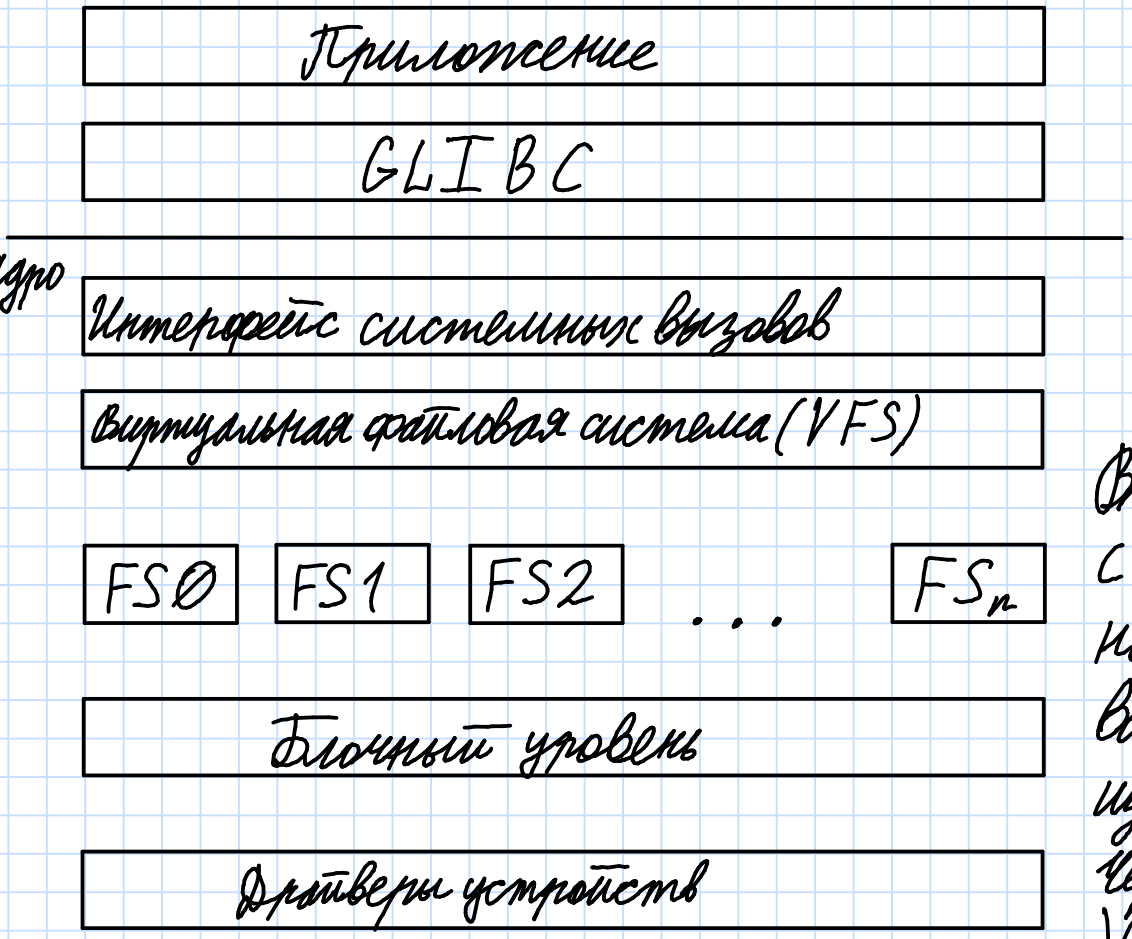
команда mount;

DOS (и впоследствии Windows) освободил пользователя от этих действий;

Это обеспечило её популярность;

Спустя много десятилетий в Windows это исправили: когда мы подключаем, например, флешку, она монтируется автоматически (даже несмотря на то, что на ней м. б. другая ф.с.);

Структура слоёв VFS:



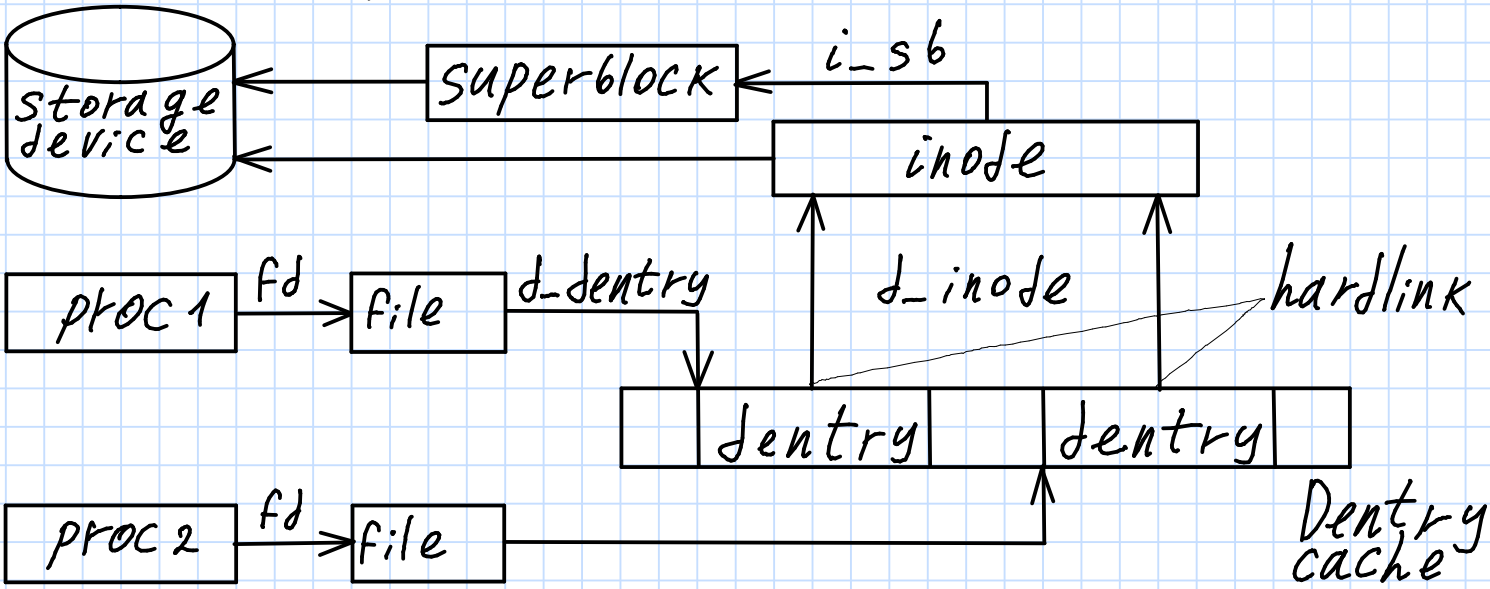
Все операции с файлами на подмонтированных ф.с. осуществляются через интерфейс VFS;

Внутренняя организация VFS базируется на 4 структурах

Struct

1) superblock;
2) dentry;
3) inode;
4) file;

Связь структур:



Все эти структуры взаимодействуют между собой, для каждого файла создаются их экземпляры (каз. объектами, хотя ядро UNIX и Linux написано на языке C без использования объектно-ориентированного подхода);

1) struct superblock описывает подмонтированную ф.с. на диске;
Именно он обеспечивает возможность работы с ф.с.;
Содержит информацию для обеспечения доступа к файлам, которые хранятся на дисках вразброс, т.е. адресацию соотв. участков диска;

2) struct dentry (Directory entry) описывает экземпляр директории;
Инде не хранится, создаётся „на лету" на основе информации, которая хранится в директориях на диске;

3) struct inode - дескриптор файла;
Существует 2 типа inode:
1) Дисковый inode описывает физический файл;
2) inode в ядре („ядрёный" inode), который позволяет предварительно контролировать доступ к файлу;
regular file → обращение к дисковому inode
pipe, socket и др. спец. файлы должны существовать не в superblock (доступ к ним должен осуществляться не через superblock);

Существует 2 ипостаси файла:
1) файл, который лежит на диске;
2) открытый файл (с которым работает процесс);

4) struct file описывает открытый файл;

В ядре имеется сист. табл. открытых файлов.

Каждый процесс имеет собственную таблицу открытых файлов, дескрипторы которой ссылаются на дескрипторы в таблице открытых файлов;

Каждый процесс до того, как он был запущен, был файлом и принадлежал некоторой ф. с., поэтому в struct task_struct имеется указатель на ф. с., которой принадлежит файл программы, и указатель на таблицу открытых файлов процесса;

Стрелки на схеме = „имеет указатель на";

Любой файл открывается с помощью процесса (приложения);

{struct superblock рассматривается для ext2}

Вся ф.с. должна занимать либо диск, либо раздел диска и начинаться с корневого каталога;

Любая ф.с. монтируется к общему дереву каталогов (монтируется в поддиректорию);

И эта подмонтированная ф.с. описывается суперблоком и должна занимать некоторый раздел жёсткого диска („это делается в процессе монтирования")

Разделы жёсткого диска ф.с. ext2:
... не стали рисовать ...