Файловая подсистема.
Взаимодействует практически со всеми модулями ОС, предоставляя пользователю возможность долговременного хранения данных, а также ОС возможность работать с объектами ядра;

ОС явл. загружаемой программой; её не называют файлом, но когда компьютер включается, ОС находится во вторичной памяти;
Затем с помощью нескольких команд, которые находятся в ПЗУ, ОС (программа) загружается в оперативную память;
При этом выполняется огромное кол-во действий, связанных с управлением памятью, и без ф.с. сделать это невозможно;

Любая ОС без файловой подсистемы не м. б. полноценной;

Любой экземпляр (объект) superblock описывает конкретную ф.с, которая м. б. подмонтирована, и только тогда файлы этой ф.с. будут доступны пользователю;

В struct superblock содержится информация, необходимая системе для управления подмонтированной ф.с.;

Если осн. задачей ф.с. (не виртуальной, а той, которая монтируется) явл. обеспечение возможности долговременного хранения файлов

и доступа к ним, то такая ф.с. должна иметь указатели на блоки диска (структура диска неизменна) и на файлы (inode), т.е. физ-файлы, которые описываются struct inode;

Каждый объект inode имеет номер (индекс), который явл. идентификатором файла в системе;

Выполняет задачи ф.с. явл. обеспечение работы с файлами;

Физ. файлы, если говорить об обычных файлах, хранятся на диске;
$\Longrightarrow$ Чтобы создать файл, для него нужно создать inode, а затем для него должно быть выделено адр. пр-во диска;

Файл $\neq$ место на диске, т.к. они уже давно хранятся вразброс

Ф-ция alloc_super создаёт новый superblock:

```
static struct superblock *alloc_super(
    struct file_system_type *type, int flags,
    struct user_namespace *user_ns)
{
    struct superblock *kzalloc(sizeof(struct super_block),
        GFP_USER);
    static const struct super_operations default_ops;
    if (!s) return NULL;
    INIT_LIST_HEAD(&s->s_mounts);
    ...
    INIT_LIST_HEAD(&S->s_inodes);
    ...
}
```

struct file_system_type определена для описания ф.с.; это тип ф.с., которая будет монтироваться (команда mount);
Это один из фокторов возможности ОС Linux работать с большими наборами разных ф.с. (они определены в union);

{Можно создать собственный тип ф.с.}

default_ops:
для любой ф.с. определяется набор операций на суперблоке (система предоставляет разработчику определить эти операции;

s_mounts:
Одна и та же ф.с. м.б. смонтирована много раз, при этом она будет иметь один тип;

struct inode описывает файл;
Если речь идёт об обычных (регулярных) файлах, то struct inode описывает файл на диске;

В ядре имеется 2 вида struct inode:
- дисковый inode содержит информацию об адресах блоков диска, на которых располагаются данные, записанные в файл;
- inode ядра;

Чтобы иметь возможность хранить файлы очень большого размера, ещё в 80-х была предложена схема с прямой и косв. адресацией, двойной и тройной косв. адресацией:
каждый адрес хранит адрес конкретного блока физ. диска, в котором хранится информация, записанная в файл;

Но кроме обычных файлов в UNIX/Linux есть прогр.-каналы, сокеты, гибкие ссылки и внеш. устр-ва (device);
Они имеют inode;
⟹ struct inode содержит union, в котором перечисляются типы файлов, и union, в котором перечисляются inode соответствующих ф.с.;

version 4.10

```
struct inode
{
    struct list_head    i_hash;
    struct list_head    i_list;
    struct list_head    i_dentry;
...
    unsigned long       i_ino;
    atomic_t            i_count;
    kdev_t              i_rdev;
    umode_t             i_mode;
...
    loff_t              i_size;
...
    // информация о времени модификации
    // и доступа к inode
...
    // 6 полей, связ. с блоками (только для ядра)
...
    unsigned int        i_blkbits; // битовая карта блоков
    unsigned long       i_blksize; // размер блоков
    unsigned long       i_blocks;  // кол-во блоков
...
    struct inode_operations  *i_op;  // перечень операций, определённых для
    struct file_operations   *i_fop; // работы с inode и с открытыми файлами
    struct super_block       *i_sb;
...
    struct list_head         i_devices;
    struct pipe_inode_info   *i_pipe;
    struct block_device      *i_bdev;
    struct char_device       *i_cdev;
...
    unsigned long            i_state;
```

```
unsigned int                    i_flags;
...
union // типы ф.с.
{
        struct minix_inode_info     minix_i;
        struct ext2_inode_info      ext2_i;
        ... //ext3 и т.д.
        struct ntfs_inode_info      ntfs_i;
        struct msdos_inode_info     msdos_i;
        ...
        struct nfs_inode_info       nfs_i; //сетевая ф.с.
        struct ufs_inode_info       ufs_i;
        ...
        struct proc_inode_info      proc_i;
        struct socket               socket_i;
        ...
}
}
```

i_hash:
Информацию о dentry (name i_hash и др.) хешируется для ускорения обращения к файлу и его имени (фактически dentry - часть имени файла);

Как правило, пользователь многократно обращается к одному и тому же файлу

Для этого в ядре имеются соотв. односвязные списки;

i_sb:
Любой inode (любой файл) принадлежит конкретной ф.с.
⇒ struct inode должна содержать указатель на superblock;

Ф-ции, определённые для работы с inode:

```
struct inode_operations
{
    int (*create)(struct inode*,
                  structdentry*,
                  struct nameidata *);
    struct dentry *(*lookup)(struct inode *,
                             struct dentry *,
                             struct nameidata*);
    int (*mkdir)(struct inode*, struct dentry *, int);
    int (*rename)(struct inode*, struct dentry *,
                  struct inode*, struct dentry*);
    ...
}
```

mkdir, rename – ф-ции, связанные с именами файлов;

Символьный уровень – самый верхний уровень ф. с., именно он связан с именованием файлов и позволяет пользователю работать с файлами (т.к. помнить inode своих файлов сложно)

Со времён появления письменности самый удобный способ именования – символьные строки

Имя файла ~ полное имя файла, начиная с корневого каталога;

В UNIX/Linux имя файла не явл. его идентификатором;
Идентификатором файла явл. номер inode;

```
struct file_operations
{
    struct module *owner;
    ssize_t (*read)(struct file *, char __user *,
            ssize_t, loff_t *);
    ssize_t (*write)(struct file *, char __user *,
            ssize_t, loff_t *);
    int (*open)(struct inode *, struct file *);
    ...
};
```

Ф-ция open предназначена для открытия файла и на чтение, и на запись (открытие = чтение и запись) (+ добавление в файл);

При создании файла для него должен быть создан inode;

Обращение к файлу ~ обращение к inode; (и уже через экземпляр этой структуры происходит обращение к данным);

В struct file есть ссылка на inode;

Если говорить о диз-файлах, которые хранятся во вторичной памяти (чтобы к ним впоследствии можно было обратиться), то ф.с. необходима информация о директориях,

т.е. о каталоге, который представляет из себя дерево директорий;
Начиная с корневой директории мы, проходя по этому дереву, в конечном итоге попадем в ту директорию, которую используем как рабочую;
К этой директории существует путь, состоящий из поддиректорий, разделённых "/" ("признак");
И только в конце, в самой рабочей директории, находится файл, к которому можно обратиться по имени;
Впоследствии, после окончания работы с файлом и сохранения информации (надолго), м.б. обратиться к этому файлу;

Структура inode каталога

inode number 3470036	
. (текущая директория)	3470036
.. (родительская директория)	3470017
Folder 1	3470031
File 1	3470043
File 2	3470023
Folder 2	3470024
File 3	3470065
...	...

Имя файла сопоставлено с номером inode, имя директории сопоставлено с номером inode (директория - тоже файл);
Именно обычные файлы и директории долговременно хранятся во вторичной памяти;

Невозможно не хранить имена директорий в долговременной памяти, т.к. иначе к ним не будет доступа (выключили компьютер, все имена исчезли и остались одни номера inode...)

Объект dentry

Принято называть объектом, несмотря на то, что ядро написано на C;

struct dentry не имеет mapping, т.е. нигде не отображена;
Именно поэтому имена поддиректорий хранятся как обычные файлы, т.к. эта информация нужна системе, чтобы предоставить в распоряжение пользователя имена директорий и поддиректорий;

Дерева каталогов не существует, т.е. оно строится „на лету" (например, утилитой tree) на основе той информации, которая сохранена на диске;
Чтобы ускорить обращение к этой информации, она вся кешируется;

Т.е. когда происходит первое обращение к каталогу, он кешируется (существует соотв. struct list_head, в которой будет храниться информация об этом каталоге;

struct dentry:

type	field	description
atomic_t	d_count	кол-во usage-ов объекта dentry
unsigned int	d_flags	флаги, опр. для конкр. объекта dentry
struct inode *	d_inode	указывает на inode, связанный с именем файла
struct dentry *	d_parent	Указатель на родительский каталог
struct list_head	d_hash	Указатель на список в хеш-таблице (указатели на соседние эл-ты в списке, связанные с одним и тем же значением хеш-функции)
struct list_head	d_lru (организован по алгоритму LRU → м.к.)	Указатель на список dentry в состоянии unused (очищается по алгоритму LRU, т.е. вытесняются dentry, к которым дольше всего не было обращений) (какое-то время неиспользуемые dentry хранятся в списке для ускорения обращения к файлам)
struct list_head	d_child	Список подкаталогов
...	...	...
int	d_mounted	Флаг, который установлен ⟺ ⟺ dentry явл. точкой монтирования ф.с.
struct qstr	d_name	Имя файла
struct dentry_operations *	d_op	Ф-ции (сист. вызовы) для работы с dentry
struct super_block *	d_sb	любая директория относится к конкретной ф.с., т.е. дерево каталогов – дерево конкретной ф.с. ⟹ объект dentry (как эл-т пути) всегда принадлежит конкретной ф.с.
unsigned long	d_vfs_flags	флаги кеша dentry
struct list_head	d_alias	list of associated inodes