Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
...льный исследовательский университет)»

СТУДЕНЧЕСКИЙ БИЛЕТ № 184963

Фамилия Мередова
Имя, отчество Мэджахан
Форма обучения ОЧНАЯ
Зачислен приказом от «31» 08 2018 г. №
Дата выдачи «01» 09 2018 г.

(подпись студента)

Руководитель образовательной организации или иное уполномоченное им должностное лицо _____ А. В. Пролетарский
(подпись) (фамилия, имя, отчество)

Действителен по 31.08.2019
Декан/директор В. В. Никитина
(подпись) фамилия, имя, отчество (последнее - при наличии)
М. П.

Действителен по 31.08.2020
Декан/директор декана ИУ
(подпись) фамилия, имя, отчество (последнее - при наличии)
М. П.

Действителен по 31.08.2021
Декан/директор
(подпись) фамилия, имя, отчество (последнее - при наличии)
М. П.

Действителен по "__"__20__г.
Декан/директор
(подпись) фамилия, имя, отчество (последнее - при наличии)
М. П.

Действителен по "__"__20__г.
Декан/директор
(подпись) фамилия, имя, отчество (последнее - при наличии)
М. П.

Действителен по "__"__20__г.
Декан/директор
(подпись) фамилия, имя, отчество (последнее - при наличии)
М. П.

ФАКУЛЬТЕТ РАКЕТНО-КОСМИЧЕСКОЙ ТЕХНИКИ И СИСТЕМЫ УПРАВЛЕНИЯ
(МГТУ ИМ. Н.Э. БАУМАНА)

Операционные системы.
Экзамен.
Мередова Айджахан.
Билет №13.

Файловая подсистема: особенности файловой подсистемы Unix/Linux.: иерархическая структура файловой подсистемы. Виртуальная файловая система VFS в Linux. Четыре структуры VFS – super_block, inode, dentry, file их назначение. Адресация файлов большого размера в файловой системе extX и пример, показывающий доступ к файлу /usr/art/mbox. Монтирование файловых систем. Команда mount и функции монтирования, пример из лаб. раб.

• Файл – это некоторая поименованная совокупность данных, хранящаяся во вторичной памяти.
Система Unix/Linux поддерживают 7 типов файлов:
d - директория
- обычной файл (регулярной файл)
l - символическая ссылка
p - именованное программные каналы.
s - сокет
c - спец. файл символьного устройства
b - бинарной файл блочного устройства
Спец. файл символьного устройства и бинарной файл блочного устройства в ядре представлены структурами cdev и blockdevice.
Существует 2 типа устройств:
- символьное (байт-ориентированное) – к нему можно обращаться как к потоку байтов (последовательной доступ);
- блочное (блок-ориентированное) - допускают обращение к произвольному месту устройства.

Устройства предоставляются в системе как файлы. Файлы устройств находятся в директории /dev.

Файловая подсистема (ФС) - это часть ОС, которая отвечает за возможность длительного хранения информации и доступа к ней; под доступом понимается операции создания, чтения, записи, переименования, удаления, установки прав доступа.

- Основными задачами ФС являются:
  - именование файлов (строковое имя файла не является его идентификатором; один файл может иметь несколько имён - hardlinks);
  - обеспечение прогр. интерфейса для работ с файлами пользователя и приложения;
  - отображение логической модели файлов на физич. организацию хранения данных на внешних носителях;
  - обеспечение длительного хранения файлов, доступа к ним и защиты от несанкционного доступа;
  - обеспечение совместного использования файлов.

ФС в Unix/Linux имеет иерархическую структуру:

символьный уровень
базовый уровень

проверка прав доступа

логический уровень

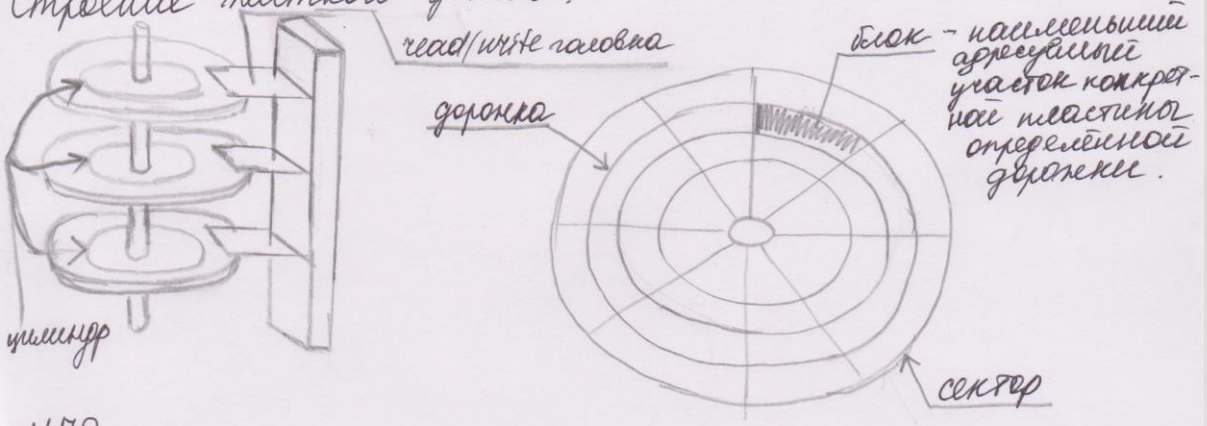
физический уровень

- Символьный уровень - уровень именования файлов, включает в себя организацию дерева каталогов.
- Базовый уровень - уровень формирования дескриптора файла. Файлы имеют inode (Index node); идентификатором файла в системе является номер inode, дескриптором - структура inode; в системе хранится информация, необходимая для преобразования имени файла (строка) в его id (получение inode)
- Проверка прав доступа. У каждого файла существуют права доступа. Эти права отдельно устанавливаются для пользователя, группы пользователей и остальных (с использованием команды chmod, к примеру)

Логический уровень. Адресное пространство файла аналогично адресному прост-ву процесса: начинается с 0-го адреса и представляет собой непрерывную последовательность адресов. Логический уровень позволяет обеспечить доступ к данным в файле в формате, отличном от формата их физического хранения.

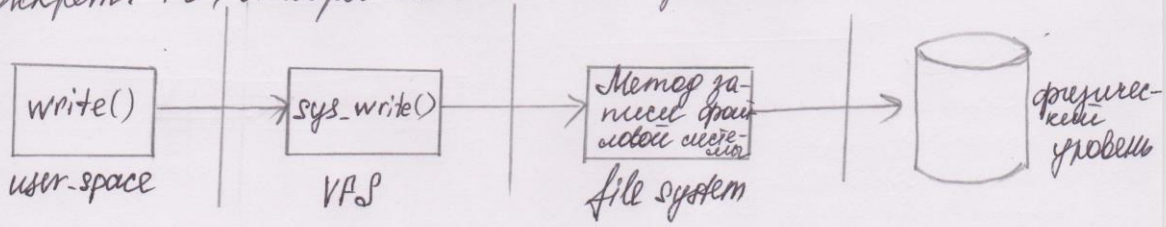
Физический уровень. - уровень, который обеспечивает непосредственный доступ к информации, хранящейся на внешнем носителе.

Строение жесткого диска:



VFS

ОС Linux поддерживает большое число ФС, что стало возможным, т.к. была сформирована абстрактный интерфейс (VFS, Virtual File System), который является операционной конкретными ФС. В результате действия конкрет. ФС, отображаются на действия VFS.



- VFS базируется на 4-х структурах:
  - super_block (SB) - структура, которая содержит информацию для монтирования и управления ФС. Каждая ФС имеет один SB; но для надёжности на диске может храниться несколько копий, т.к. SB - ключевая структура ФС.
  - inode - индексный дескриптор, содержит информацию о файле; каждый файл имеет 1 inode, но может

иметь несколько имён (hardlinks). Существует 2 типа inode:

- дисковой - содержит информацию о файле, а также данные для адресации каждого участка дискового пространства, занимаемого файлом.
- inode ядра - вместо сведений об адресации хранит информацию для нахождения файла, определения принадлежности к ФС, каталогам и подкаталогам.

- dentry - представляет компонент пути к какому-либо файлу. Например, для пути "/mnt/cdrom/foo" компонентами будут являться: mnt, cdrom, foo. Любой dentry может находиться в одном из трёх состояний:
  - используемой (used) - d_inode указывает на объект типа inode (!= NULL), d_count > 0 (число использований).
  - неиспользуемой (unused) - d_inode != NULL, d_count = 0;
  - негативной (negative) (противоречивой) - d_inode = NULL (объект не соответ. никакому inode)
- file - содержит информацию об открытом файле

Ниже представлены ключевые поля описанных структур:

- struct super_block

```
{ struct list_head s_list;
  dev_t s_dev;
  unsigned long s_flags;
  struct dentry *s_root;
  int s_count;
  char s_id[32];
  const struct super_operations *s_op;
  ...
}
```

- struct inode

```
{ struct super_block *i_sb;
  const struct inode_operations *i_op;
  unsigned int i_flags;
  umode_t i_mode;
  atomic_t i_count;
  ...
}
```

- struct dentry

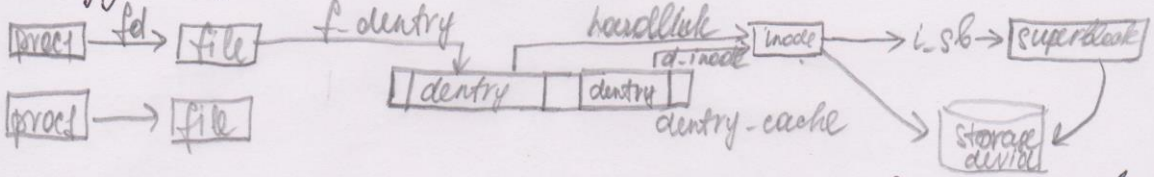
```
{ struct inode *d_inode;
  const struct dentry_operations *d_op;
  struct dentry *d_parent;
  struct list_head d_child;
  struct list_head d_subdirs;
  atomic_t d_count;
  unsigned int d_flags;
  ...
}
```

(d_parent, d_child, d_subdirs — поля служат для организации дерева каталогов.)

- struct file

```
{ mode_t f_mode;
  loff_t f_pos;
  struct file_operations *f_op;
  const char *name;
}
```

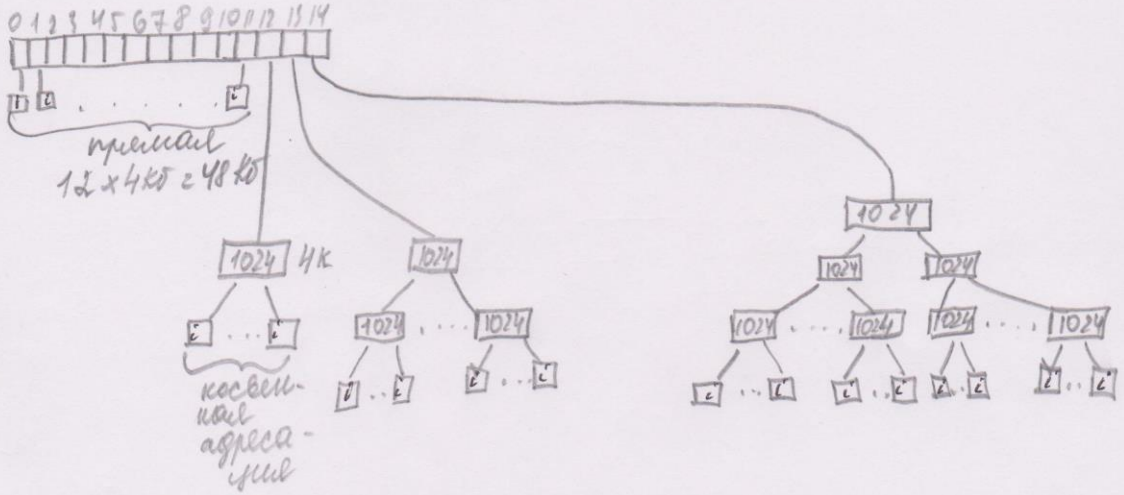
Между структурами существует связь:



В FS ext4 для хранения данных, записываемых в файл, в inode выделено 15 полей по 4 байта, 12 из которых являются блоками прямой адресации, а остальные - блок косвенной, а также двойной и тройной косвенной адресации:

- блок прямой адресации; в блоке нах-ся адрес блока данных
- блоку косвенной адресации хранит указатель на блок, который содержит адреса других блоков (блоков данных или других косвенных блоков)

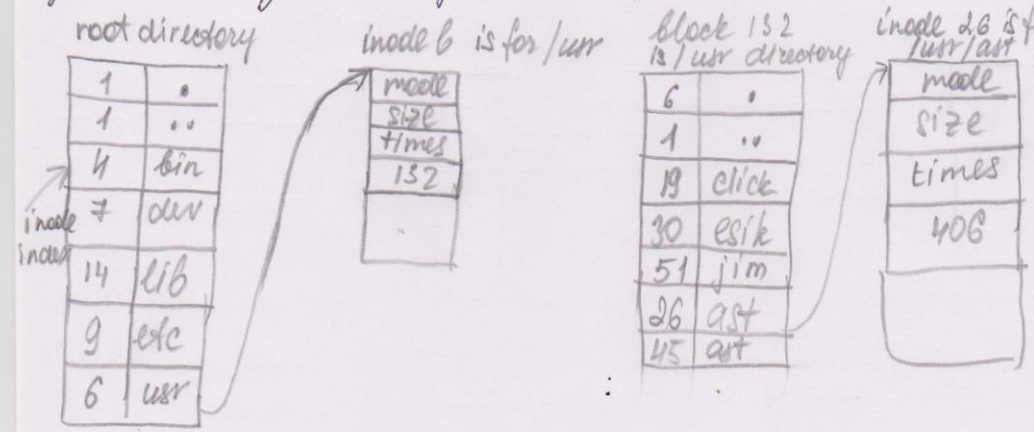
Получилась данной схемой адресации файлов можно поддерживать работу с файлами размером около 2 ТБ, (размер одного блока 4 Кб)



обозначения:

- [i] - блок данных
- [1024] - блок, хранящий адреса других блоков.

В примере ниже демонстрируются выполняемые действия для получения доступа к файлу /usr/ast/mbox



block 408 is for /usr/ast directory

26	.
6	..
64	grants
92	books
60	mbox
81	minix

→ /usr/ast/mbox
inode 60.

- Каждое обращение к файлу – это большой объем операций и затрачиваемое на них время, поэтому ядро кэширует объекты dentry т.е. в ядре имеется кэш dentry, который состоит из 3х частей:
  - список используемых объектов dentry (used)
  - двусвязный список unused и negative объектов по алгоритму LRU (Last Recently Used)
  - хэш-таблицу, хэш-ф-цию, для которой каждая ФС может переопределить: d_hash()

Монтирование ФС

- Mount – утилита командной строки Unix/Linux. Для подключения ФС к конкретному местоположению (точке монтирования) команда mount используется в следующем формате:

```
mount [опции ...] <имя_устройства> <директория>
```

Часто используемая опция -t позволяет указать тип файловой системы. Например:

```
sudo mount -o loop -t my-fs ./image ./dir
```

(тип ФС – my-fs задается в поле name структуры file_system_type)

- Ф-ции монтирования:
  - монтирование блочного устройства:

```
extern struct dentry *mount_bdev (struct file_system_type
        *fs_type, int flags, const char *dev_name,
        void *data, int (*fill_super)
        (struct super_block *, void*, int));
```

Аргументы ф-ции:
- fs_type – тип файловой системы
- flags – флаги монтирования
- dev_name – имя монтируемого устройства
- fill_super – ф-ция, которая выполняет необходимые действия по инициализации суперблока.

- Также есть другие ф-ции монтирования:
  - extern struct dentry * mount_nodev (struct file_system_type *fs_type, int flags, void *data, int (*fill_super) (struct super_block *, void *, int)); – монтирование ФС не привязанной к уст-ву
  - extern struct dentry * mount_single (-"-) – разделяет структуру между всеми монтированиями.
- Когда файловая система монтируется, создается структура struct vfs_mount, которая представляет конкретный экземпляр ФС. Данная структура связывается со списком sb->s_mounts (struct list_head), а также с глобальным vfs mnt list.
- Ниже приведен краткий пример регистрации типа ФС и использования ф-ций монтирования (опущены некоторые действия и проверки ошибок)

```
struct dentry *mount (struct file_system_type *type,
    int flags, const char * dev, void *data)
{ struct dentry *const root_dentry = mount_nodev (type,
        flags, data, fs_fill_sb);
  return root_dentry;
}
int fs_fill_sb (struct super_block *sb, void *data, int
                silent) // инициализация суперблока
{ sb->s_op = &fs_super_ops; // другие наши операции
  struct inode *root = my_fs_make_inode (sb, S_IFDIR|0755);
  // внутри ф-ции вызов new_inode (sb)
```

```
root -> i_op = &simple_dir_inode_operations; // определено
                                  в ядре (VFS)
root -> i_fop = &simple_dir_operations;
sb -> s_root = d_make_root (root); // создание dentry
// создание файлов и директорий ФС
  return 0;
}
struct file_system_type fs_type = {
    .owner=THIS_MODULE;
    .name = "my_fs",
    .mount = fs_mount,
    .kill_sb = kill_block_super; // ф-ия ядра
}
static int _init fs_init (void)
{ int ret = register_filesystem (&fs_type); // регистрация ФС
  return ret;
}
static void _exit fs_exit (void)
{ int ret = unregister_filesystem (&fs_type);
}
module_init (fs_init);
module_exit (fs_exit);
MODULE_LICENCE ("GPL");
MODULE_AUTHOR ("Meredova");
```