```
struct timespec64 i_ctime; // control
u8 i_blksize; u8 i_blkbits
    ...
blkcnt_t i_blocks
    ...
}
```
размер блока в байтах

(1) объект inode представляет конкр. физич. файл, кот. находится на каком-то фактическом устр-ве => в struct inode указан этот device + i_rdev

$\boxed{\text{Лекция 17.04.}}$

В системе ∃ 2 inode:

1) inode ядра

2) дисковый inode (в нём находятся адреса блоков файла, в которых хранятся данные файла)

В struct inode есть:

(информ.)

- i_ino – это поле недоступно через прохождение по пути
- тип loff_t нужен для адресации очень больших файлов
- unsigned long dirtied_when; /* jiffies of first dirtying */

это поле нужно, чтобы контролировать копирование этого инф-ции на диск →

продолжение → Она сначала меняется в ядре, а потом инф-ция об inode переписывается на диск

- unsigned long dirtied_time_when;

привычное нам время

- struct list_head i_lru;

  · Связывает список айнодов по давности по алгоритму lru.
  Т.е. в голове этого списка будут находиться айноды, к которым дольше всего не было обращения

- С открытыми файлами работает стр-ра struct file. Когда файл открыт мы с ним работаем ф-циями, кот. определены в struct file_operations.

поле из struct inode:

· atomic_t i_readcount; /*struct files open RO*/

```
. union{
      const struct file_operations *i_op;
    ...
   }

union{
   struct pipe_inode_info *i_pipe;
   ...
   }
   ...
void *i_private; /*fs or device private pointer */
}
```

стр-ры часто содержат приватное поле

```
struct inode_operations
{
   struct dentry * (*lookup)(struct inode *
                   struct dentry *, unsigned int);
   ...
   int (*create)(struct mnt_idmap, struct inode *,
                 struct dentry *, umode_t bool);
   int (*link)(struct dentry *, struct inode *,
               struct dentry *);
        ...
   int (*symlink)(struct mnt_idmap *, struct inode *,
                  struct dentry *, const char *);
   int (*mkdir)(struct mnt_idmap *, struct inode *,
                struct dentry *, umode_t, dev_t);
} ...
```

создание дескриптора файла

• mknode, rename, rmdir — тоже находятся в inode_operations

здесь надо обратиться к struct inode и struct dentry

• в struct inode есть поле struct rw_semaphore i_rwsem

Зачем?. — потому просматривать инф-цию о файле могут многие процессы, но изменять инф-цию о конкр. файле можно только в режиме монопол. доступа

{Инф-ция о каталогах хранится также как информация об обычных файлах — во вторич. памяти (иначе невозможно будет восстановить инф-цию о дереве каталогов, содержании каталогов)}

Структура inode каталога

идентификатор

inode number	3470036
. DOT	3470036
.. DOT DOT	3470017
folder 1	3470031
file 1	3470043
file 2	3470023
folder 2	3470024
file 3	3470065

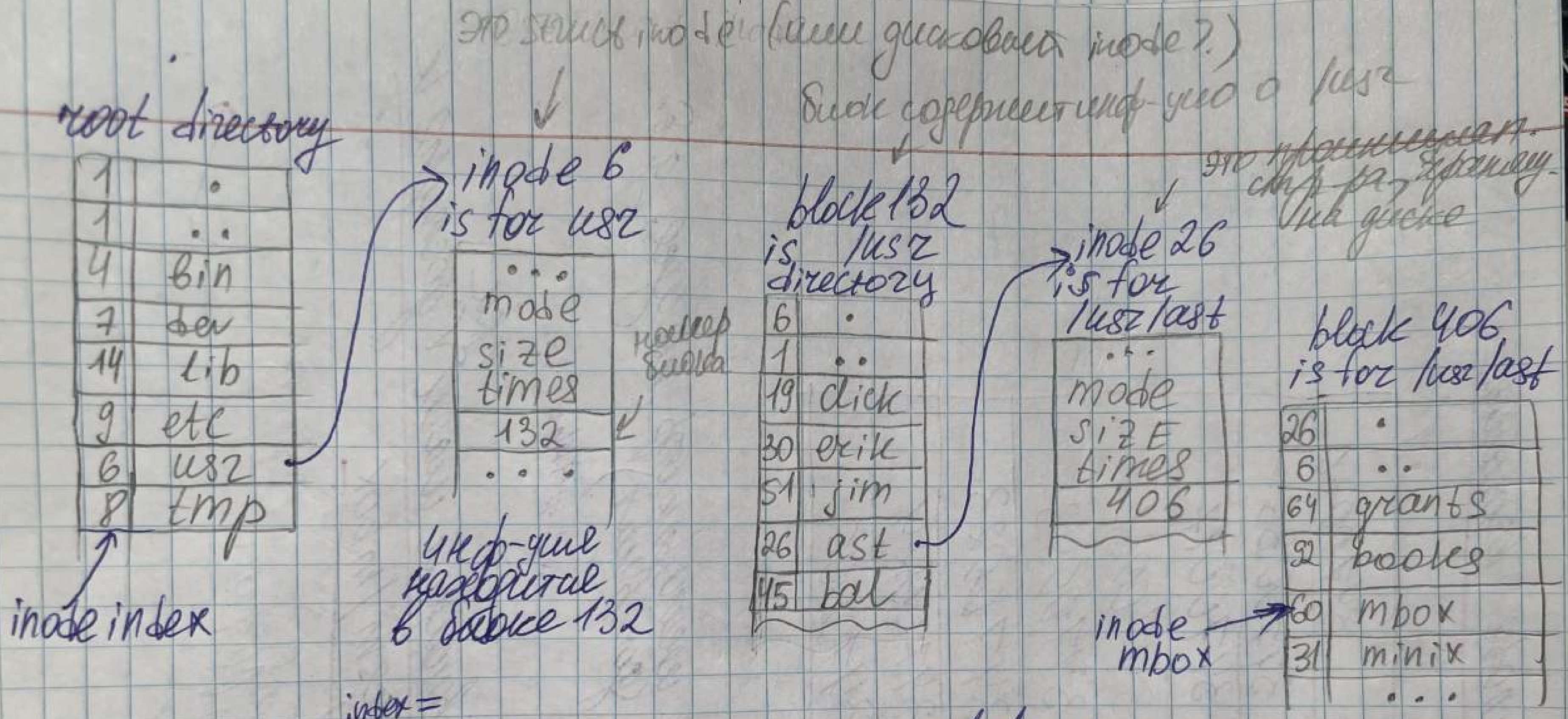
У родител. дир-рии идентификатор, но уже, д.б. меньше, (чем у дочерних)

Пример, показывающий шаги при доступе к файлу /usr/asm/mbox

корн. каталог — /

поддир-рии — usr, asm

файл — mbox



• По inode index= 6 обращаемся к поддир-рии usr

• Из inode с index=6 получаем инф-цию, что инф-ция о поддир-рии usr находится в блоке 132.

· переходим к inode с index=26

• Прежде чем обратиться к инф-ции, хранящ. в файле, надо к этому файлу доступ получить

• Если в кэше dentry не найден, то будет выполнена тяжелая последовательность действий

(Память выделяется страницами (4096 байт) и в стр-цах всегда ∃ незанятые адреса (т.е. эти 4096 байт использ. не очень эфф-но)

• Linux предоставляет возможность создания собств. фс.

• В ядре объявлена стр-ра file_system_type

```
struct file_system_type            версия 6.8.7
{
  const char *name; //НАЗВАНИЕ ФС
  int fs_flags; //флаги
  #define FS_REQUIRES_DEV 1
  ...
  #define FS_USERNS_MOUNT 8
  ...
  #define FS_RENAME_DOES_D_MOVE 32768
  ...
  struct dentry *(*mount)(struct file_system_type*,
                          int, const char *, void *);
  void (*kill_sb)(struct super_block);
struct module *owner;
struct file_system_type *next; // связ. список файл. систем
struct hlist_head fs_supers; — список объектов типа super_block
...
}
```

Разработчик должен инициализ-ть поля стр-ры file_system_type

В системе может существовать только один тип ФС. Но ФС данного типа м.б. смонтирована много раз.
Только смонтированные ФС предоставляют доступ к своим файлам. (это видно из struct super_block: в суперблоке есть поле „все айноды")

· Смонтированная ФС становится частью дерева каталогов.

. name - название ФС

. mount - указ-ль на смонтированную ФС

. kill_sb - эта ф-ция, обеспечивает прекращение доступа к суперблоку

. owner - владелец (владельцем яв-е зачастую модуль ядра THIS_MODULE)

Ядро предоставляет 4 ф-ции mount: (before mount_subtree)
mount_bdev, mount_single, mount_nodev, mount_ns

device
флаг FS_REQUIRES_DEV – требуется устр-во
(мы будем создавать вирт. ФС, поэтому не нужно ~~в~~ устр-во)

block device
1) 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));

Пояснение:

- д.б. проинициализ. на struct filesystem...
- д.б. определены флаги (во флагах требуется девайс)
- fill_super – ф-ция, кот. делает осн. действие: сначала необходимо заполнить поле суперблока, создать inode корневого каталога

не нужно устр-ва
2) extern struct dentry *mount_nodev(struct file_system_type *fs_type, int flags, void *data, int (*fill_super)(struct super_block *, void *, int));

Пример инициализации полей стр-ры:

```
static struct file_system_type lfs_type = {
    .owner = THIS_MODULE,
    .name = "lwnfs",
    .mount = my_mount,
    .kill_sb = kill_litter_super (struct super_block *sb);
    .fs_flag = FS_REQUIRES_DEV,
};
```

минимал. инициализация

— эту ф-цию обязаны написать

— эта ф-ия есть в ядре

В ядре Linux есть 2 вида ф-ций:

- generic
- little

у каждого из этих видов есть свои особенности; этих ф-ций очень много

Из этих ф-ций можно свою ФС написать. Они содержат минимальн. набор необходимых действий

Регистрация собств. ФС:

```
... my_init()
{ return register_filesystem(&lfs_type);
}
```

в эту ф-цию надо передать проинициализир-ую стр-ру file_system_type

Разрегистрация ФС:

```
... my_exit()
{
    return unregister_filesystem(&lfs_type);
}
```

• Минимальная инициализ-ция стр-ры filesystemtype — это поля owner и name.

- Зарегистрировав нашу ФС, с помощью вирт. ФС proc убедимся, что наша ФС введена в перечне зарегистриров-х ФС.