(2) - записывает inode на диск и помечает его как "грязный"

(3) - сбрасывает inode

(4) - вызывается при размонтировании ФС

Лекция 04.04.

dentry и inode

интерфейс

В основе VFS лежат 4 структуры:

- struct super_block
- struct dentry
- struct inode
- struct file

- Ф-ции, опред. на super-block, составляют struct super_operations

- В struct super_operations нет ф-ции
  - создаёт объект суперблок
  alloc_super() — эта ф-ция работает и инициализирует структуру super_block.
- static struct super_block *alloc_super(struct file_system_type *type, int flags, struct user_namespace *user_ns)

{

struct super_block *s = kzalloc(sizeof(struct super_block), GFP_USER);

```
static const struct super_operations default_op;
...
INIT_LIST_HEAD(&s->s_mounts);
...
INIT_LIST_HEAD(&s->s_inodes);
...
}
```

• Проинициализировать объект super_block может описывать смонтированную ФС.

• Любой файл ∈ какой-то опред. ФС и значит эта ФС должна иметь информацию о файлах, которые позволят выполнить обращение к данным, записи в файл.

Узлы inode и dentry

node – "узел"

• struct inode – описывает физический файл. (файлы, кот. реально хранят инф-цию) каждый объект этой стр-ры описывает конкрет. физич. файл

• struct file – каждый объект struct file это дескриптор открытого файла в системе.

• struct dentry – 1) определяет элемент пути в имени файла (с одной стороны). 2) С другой стороны, это стр-ра, кот. пред-

ставиться соответствующую директорию
(имя файла состоит из имени файла соответственно и пути к файлу (все поддиректории, начиная с корневого каталога))

• И объект struct inode, и объект struct dentry это файл (т.е. инф-ция, предназнач. для долговременного хранения)

• Вся инф-ция о директории должна храниться на диске (иначе (мы никогда не сможем получить доступ к файлам)

• Стр-ра struct dentry не соответствует никакой стр-ре на диске, т.е. директории это такие же файлы, как обычные файлы. При этом содержимое файла дир-рии это те файлы и поддир-рии, кот. находятся в данной дир-рии.

• Объект struct dentry - это элемент пути, представляющий собой поддиректорию, но дир. элемент пути (это /.../.../)

• Чтобы получить доступ к файлу, система проходит все элементы пути

struct dentry

Каталог можно удалить, когда на него больше нет ссылок

```
{
  atomic_t d_count; //usage count
  struct inode *d_inode; // associated inodes (связ. inode)
(1) struct list_head d_lru; //unused list
(2) struct list_head d_child; // list of dentries within
  struct list_head d_subdirs; //subdirectories
(3) struct list_head d_alias; // list of alias inodes
  ...
  struct dentry_operations *d_op;
  struct super_block *d_sb; - // файл директории в опред. ФС, кот. д.б. смонтирована
  ...
  int d_mounted; // is this a mount point?
  ...
  struct qstr d_name - //dentry name
(4) struct hlist_node d_hash; // list of hash table entries - вход в список hash table
  struct hlist_head *d_bucket;
  unsigned char d_iname[...]; //shortname
}
```

тут предописанная константа

поскольку инф-я о подкаталогах хранится как файл

действие, опред. над dentry

явл-ся ли директ-я т. монтир-я или нет

(1) тот же элемент: связывает dentry в списке неиспользуемых dentry-записей

(2) - связывает dentry в списке дочерних dentry родительского каталога

(3) список dentry, ссылающихся на один и тот же inode (жесткие ссылки)

(4) связывает dentry в хэш-таблице для быстрого поиска по имени

но: Объекты dentry кэшируются. Когда выполняется проход по пути к файлу, чтобы сократить время этого "прохождения" (при прохождении пути к файлу проверяется существование файлов) все объекты dentry кэшируются

! Вся информация о подкаталогах хранится как файлы

· dentry state – состояние dentry. Допустимость объект dentry может находится в 1 из 3х состояний:

1) используемые (in use)
2) неиспольз-ые (unused)
3) отрицательность (т.е. не связанный с допустимым индексным узлом, т.е. поле d_inode = Null) (negative)

(, он не ошибочный, о нём нет информации на диске)

· Состояние dentry "in use" указывает на то, что данного dentry соответ-ет допустимому inode, т.е. в d_inode указан соотв. индекс-узел. В рез-те можно делать вывод, что у такого dentry есть 1 или несколько пользов-лей, т.е. d_count>0

· Не используемый dentry ("unused") соотв-ует допустимому индекс. узлу (d_inode≠Null), но в данный момент этот dentry не используется и d_count = 0. Поскольку такой объект dentry по-прежнему указывает на допустимый inode, такой dentry хранится в кэше (он может еще понадобится)

· Отрицательность dentry (d_inod = Null) сохраняется в кэше (т.к. какие-то запросы могут к нему обращаться)

Кэш dentry

· Сам объект dentry не имеет стр-ры, которая бы хранилась на диске

· Dentry рассматр-ся системой как файлы, но спец. типа

· Для того, чтобы каждый раз не обращаться к диску за инф-цией, кот. хранится в файле dentry, эта инф-ция, когда происходит проход по пути к файлу, записывается в кэш.

И прежде чем обращаться к диску эта информация об элементе пути всегда сначала ищется в кэше. Если её в кэше нет, то происходит обращение к диску.

Кэш dentry состоит из 3 частей:

I часть: список актуальных (рабочих)

Это dentry, кот. связаны с опред. индексом, но inode может иметь несколько фамил. Т.е. м.б. несколько объектов dentry, имеющих один и тот же inode

II часть: двусвязный список dentry-lru

(least recently used)
наименее используемые в последнее время

Т.е. это список dentry, отсортированный по времени обращения (последнее обращение)

· Есть 2 способа организации алгоритма LRU и один из них это временная метка. При каждом обращении к объекту dentry временная метка корректируется и список перестраивается (если он отсортирован по времени обращения).

Если в ядре возникает необходимость освобождения памяти, то будут удалены объекты dentry, находящиеся в хвосте очереди (т.е. те, к которым дольше всего не было обращения)

III часть: Хэш-таблица и ф-ция хэширования

· Используются для быстрого преобразования заданного пути к указанному файлу (т.е. надо связать указанный путь с соответствующим объектом dentry)

· Хэш-таблица представляется в виде массива dentry_hashtable, в котором каждый элемент (вв-е указателей) на список dentry, которые хэшируют в одно и то же значение.

· Размер хэш-таблицы зависит от доступного объёма физ. памяти в системе

· Фактическое значение хэша определяется ф-цией d_hash().

· Поиск по хэш-таблице выполняется с помощью ф-ции d_lookup(). Если объект

dentry найден, то возвращается указ-ль на него). В противном случае возврат 0

полное имя файла

Пример: /mhome/mydir/c_f/myserver.c

При каждом обращении к файлу myserver.c вертуал. ФС должна пройти по всем dentry данного имени файла (начиная с "/")

```
struct dentry_operations
{
(1) int (*d_revalidate)(struct dentry *, int);
(2) int (*d_hash)(struct dentry *, struct qstr *);
(3) int (*d_compare)(struct dentry *, struct qstr *,
                     struct qstr *);
(4) int (*d_delete)(struct dentry *);
(5) void (*d_release)(struct dentry *);
(6) void (*d_iput)(struct dentry *, struct inode *);
...
}
```

(1) Объект dentry валидный и эта ф-ция вызывается, когда подготавливается для использ-я объект dentry из кэша. Некоторые ФС устанавливают для этого метода значение NULL, т.к. их объекты dentry всегда валидные

(2) Создаёт значение хэш для указанного dentry. ФС вызывает эту функцию

когда надо добавить dentry в хэш-таблицу.
(3) вызывается, когда удаляются данные файла ("name" и "name2" например)
(4) когда d_count становится = 0, вызывается функция d_delete
(5) вызывается VFS когда опред. dentry переходит в состояние free
(6) вызывается VFS когда объект dentry теряет сведения inode.

Inode

- Объект inode описывает физ. файл т.е. реально существующий файл, в котором находятся записанные в него данные. Такой файл может находиться во внеш. памяти или в физ. памяти, но в любом случае файл будет иметь представление в системе в виде объекта inode.
- Объект inode - это произвольная структура inode

- В отличие от Unix, в Linux нет понятия vinode (виртуальный inode). В Linux 2 типа структура struct inode:
  1) структура inode ядра
     Эта структура содержит информацию об

функ. файле. Эта структура позволяет ядру хранить информацию о файле

2) дисковый inode - содержит информацию об адресах блоков диска, в которых хранятся данные этого файла.

- У inode есть идентификатор-ный - номер inode. Обращение к inode выполняется по его номеру. Номер inode - это идентификатор файла

!!!!!!!!!!

inode ядра →

```
struct inode
{
    umode_t i_mode;
    unsigned short i_opflags;
    kuid_t i_uid;
    ...
    unsigned int i_flags;
    ...
    const struct inode_operations *i_op;
    struct super_block *i_sb; // к файлу в данной ФС
    struct address_space *i_mapping;
#ifdef CONFIG_SECURITY
    void *i_security;
#endif
    unsigned long i_ino; // индекс дескриптора (номер)
(1) dev_t i_rdev;
    loff_t i_size; // текущий размер файла в байтах
    struct timespec64 i_atime; // время последнего обращ
    struct timespec64 i_mtime; // modification
```

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

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

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:

- i_ino — это поле недоступно через прохождение по пути (информация)
- 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 →