

Рязанова, лекция.

Struct dentry.

В отличие от объектов superblock и inode, dentry это конкретная структура, выделенная на directory entry, не сопоставляется ни с какой структурой на диске. Виртуальная фс создает этот объект как говорят на лету (by fly) из строкового представления пути к файлу (pathname).

Dentry, как и все структуры ядра, претерпевает изменения. Эта структура работает с именами файлов.

Каждая директория, или каждая поддиректория в UNIX/LINUX это файл, специальный файл. То есть интерфейс VFS представляет каталоги как файлы. Но это специальные файлы, на которых определены специальные действия, и у них есть свои задачи, главная из которых обеспечить возможность по символьному имени файла обеспечить доступ к физическому файлу. Но обратиться к файлу мы можем по номеру inode'а, но для человека удобнее использовать какие-то названия (имена) - строка символов, которая разбирается, но по этому пути в конечном итоге надо получить номер inode'а. То есть объект dentry - это определенный компонент пути (к файлу). Причем, все объекты dentry это компоненты (элементы) пути, включая обычные файлы, которые могут включать в себя точки монтирования. Поскольку объекты элементов каталога не хранятся на физическом носителе, например на диске, структура struct dentry не имеет флагов, которые указывали бы на то, что объект был изменен (dirty) потому, что объект dentry не надо переписывать на диск, а отредактированный inode надо, если сопоставить с объектом inode.

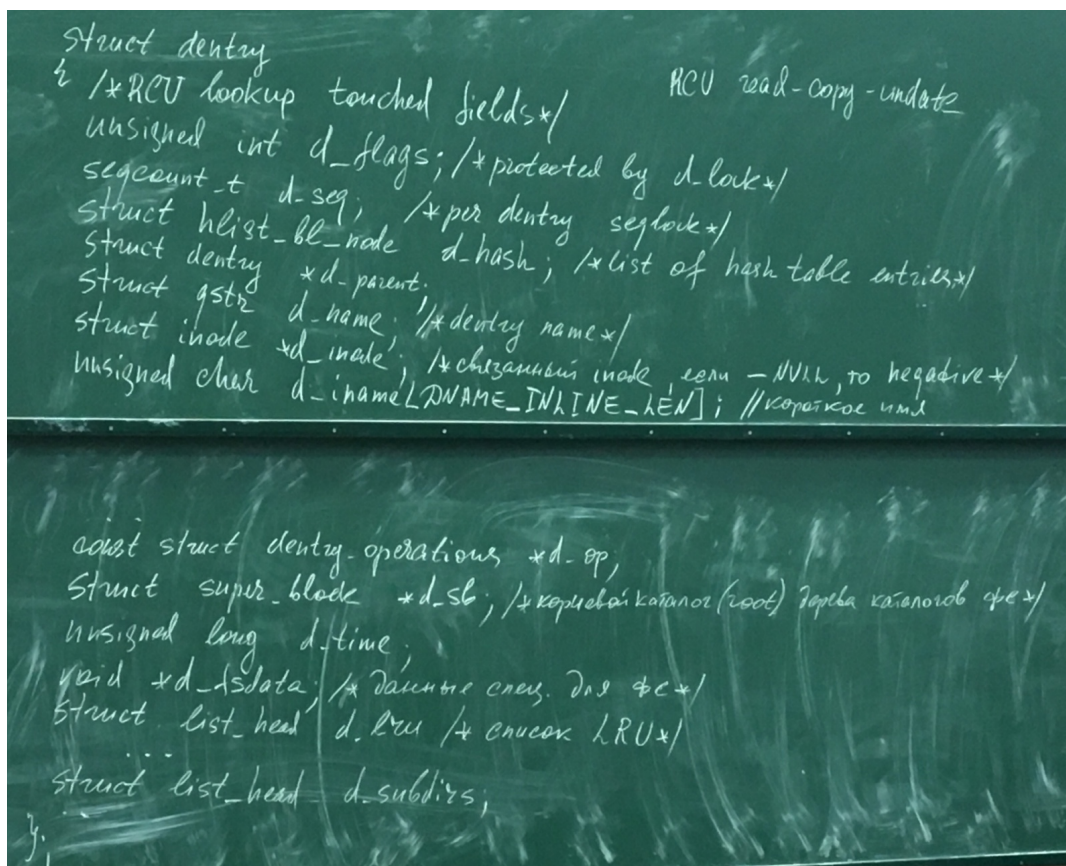
6 апреля, сб

Рассмотрим структуру struct dentry:

Замечание: это код из ядра версии 3.14

RCU - специальный механизм синхронизации (read-copy-update), который обеспечивает монополярный доступ. Синхронизация предполагает, что какой-то процесс заинтересован в действиях другого процесса (сформирует какое-то сообщение, получив которое, процесс сможет продолжить свое выполнение).

STRUCT DENTRY:



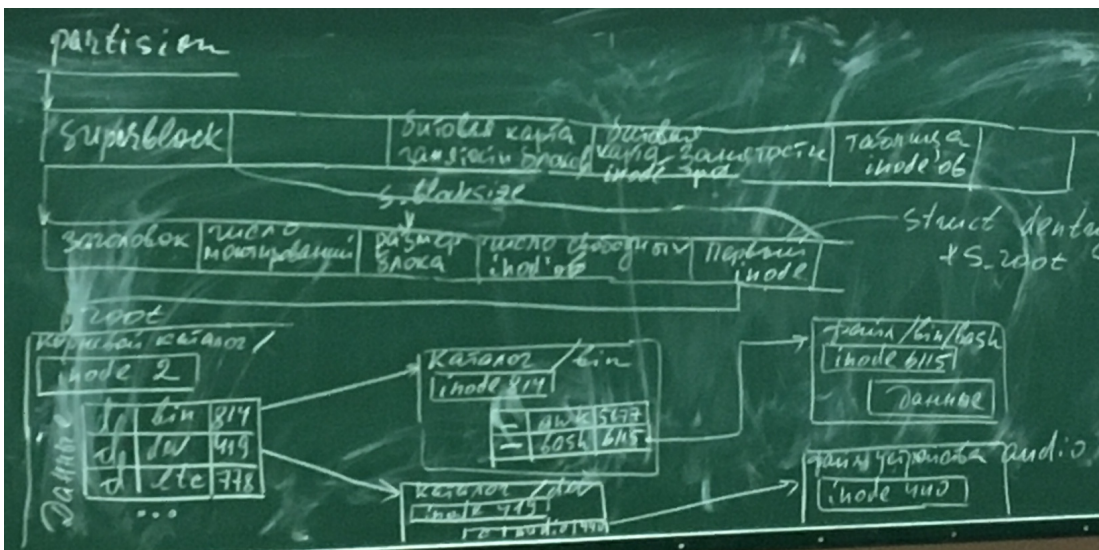
```
struct dentry
/* RCU lookup touched fields */
unsigned int d_flags; /* protected by d_lock */
seqcount_t d_seq; /* per dentry seqlock */
struct hlist_bl_node d_hash; /* list of hash table entries */
struct dentry *d_parent;
struct qstr d_name; /* dentry name */
struct inode *d_inode; /* связанная inode, если - NVL, то negative */
unsigned char d_iname[NAME_MAX]; /* короткое имя

const struct dentry_operations *d_op;
struct super_block *d_sb; /* корневой каталог (root) дерева каталогов */
unsigned long d_time;
void *d_fsdata; /* данные спец. др. ФС */
struct list_head d_lru; /* список LRU */
struct list_head d_subdirs;
};
```

Актуальный объект dentry может находиться в одном из трех состояний - used, unused, negative (используемый, неиспользуемый, противоречивый).

Имеется указатель на корневой каталог (struct super_block *d_sb).

6 апреля, сб



Dentry кэшируются.

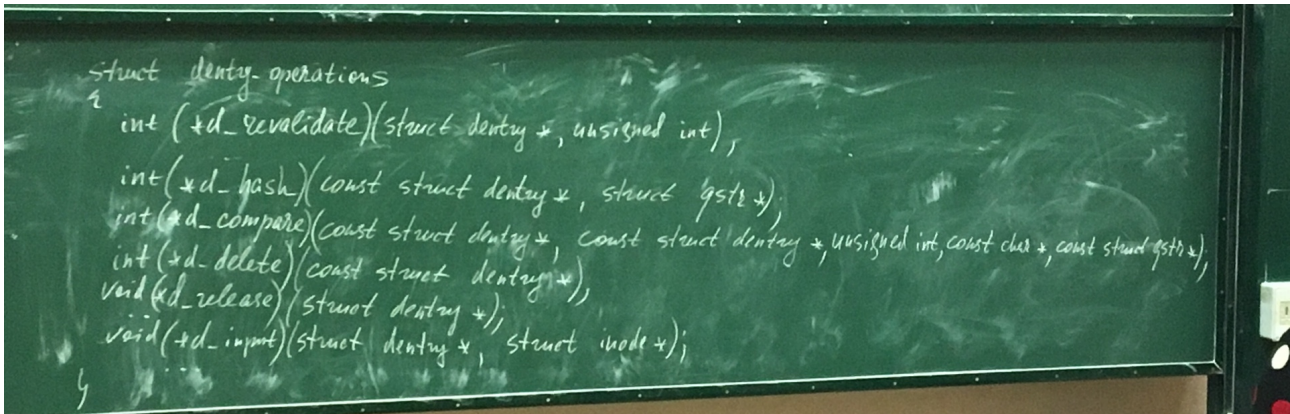
После того, как виртуальная фс прошла по пути (спустилась по пути (полному имени файла)), и для каждого пути создан объект dentry (каждый путь пройдет от начала до конца), то это очень большая работа.

Очевидно, что в системе эту работу не выбрасывают. Вместо этого полученную в результате разбора информация сохраняется в dentry кэш, который состоит из 3х частей - список используемых объектов detnry, которые с inode'ом полем d_inode. Т к отдельный inode может иметь много линков, то может быть много объектов dentry, связанных с этим inode, корреспондирующих этот inode. Второй - двусвязный список list recently used - список неиспользуемых объектов dentry. Соответственно удаление элементов из данного списка выполняется по алгоритму lru.

Третий список - hash table - кэширует функции для быстрого определения заданного пути в ассоциированный объект dentry. Hash table реализована в виде hash_dentrytable массива. Каждый элемент является указателем на список dentries, хэшированных по тому же самому имени.

6 апреля, сб

DENTRY OPERATIONS:



`d_revalidate` - эта функция работает, если заданный `dentry` объект существует. Виртуальная фс вызывает ее, чтобы подготовить `dentry` в хэше. Большинство фс устанавливают его в 0, так как они всегда доступны.

`d_hash` - создает значение хэш, величину хэш для заданного объекта `dentry`. VFS вызывает эту функцию для того, чтобы добавить эту функцию в `hash_tripple`

`d_compare` - сравнивает два имени файла. Большинство файловых систем определяют ее как `found`, `vpzду`.

Inode cache.

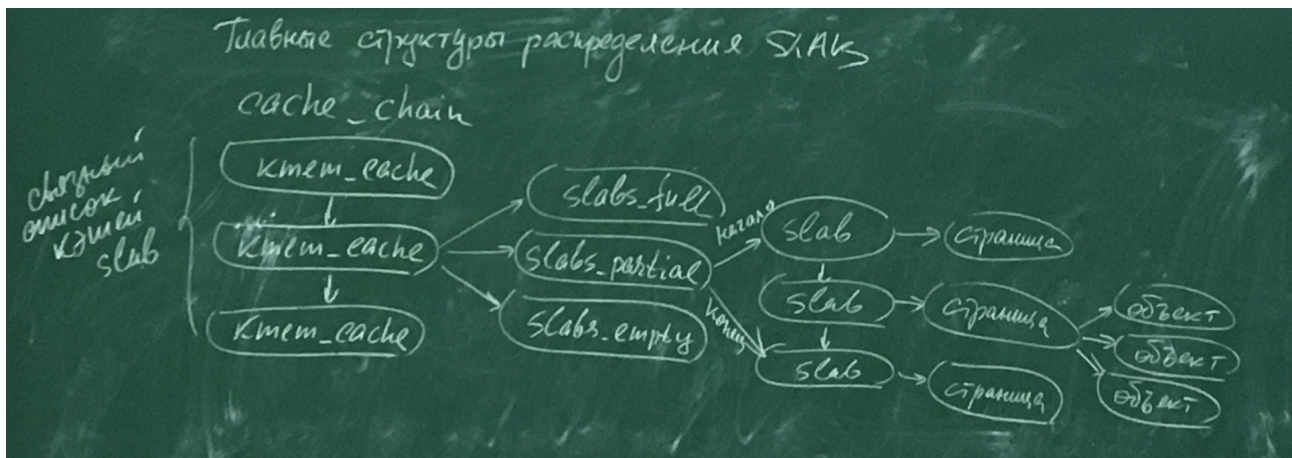
Inode кэш представляет из себя:

1. Глобальный хэш-массив `inode_hashtable`, в котором каждый `inode` хэшируется по значению указателя на суперблок и 32х разрядному номеру `inode`. При отсутствии суперблока, `inode` добавляется к списку анонимных `inode`'ов. Примером таких анонимных `inode`'ов могут служить сокеты, вызванные `sock_alloc()`.
2. Глобальный список `inode_in_use`. В этом списке содержатся допустимые `inode`'ы (`i_count > 0` и `i_nlink > 0`). В этот же список записываются вновь созданные `inode`'ы (объекты `inode`).
3. Глобальный список `inode_unused`, который содержит допустимые `inode`'ы с `i_count` равным нулю.
4. Список для каждого суперблока (`sb -> s_dirty`), которые содержат `inode`'ы с `i_count > 0` и `i_nlink > 0` и состоянием `i_dirty`, то есть измененные `inode`'ы
5. Флаг кэш `inode`'а SLAB cache `inode_cacher`. Объекты `inode` могут освобождаться вставляться и изыматься из SLAB кэшей. Любой

6 апреля, сб

объект inode может находиться только в одном из этих списков. SLAB был введен Джефом Бонвиком для операционной системы SunOS. (SLAB аллокатор) - это кэширующий аллокатор, позволяющий выделять блоки памяти одного и того же размера, но для данных одного и того же типа. SLAB распределение стало использоваться для того, чтобы упростить управление памятью и выделение памяти. Дело в том, что в ядре значительные объемы памяти выделяются на ограниченный набор объектов таких как дескрипторы файлов, inode'ы и т. п. Идея Бонвика базируется на том, что количество времени, нужное для инициализации обычного объекта в ядре, превышает время, которое необходимо для его выделения и освобождения. Вместо того, чтобы возвращать память системе, оставляя ее в проинициализированном состоянии, рассчитывая на то, что в будущем она будет использована для тех же целей. Если надо выделить участок памяти на объект inode, то будем юзать уже выделенную. Например, если память выделена для mutex, то функцию init_mutex надо выполнить только один раз, когда память выделяется впервые. Последующие распределения памяти под mutex не нужны, так как она уже выделена в результате инициализации и последующего освобождения.

Главные структуры распределения SLAB



Алгоритм best-fit - самый подходящий.