Ответы

Фортунки: • 6 точек входа

• почему для передачи данных из kernelUser в kernelUser нужны спец. ф-ции ядра

Семинар 14.05.

dentry - directory entry

Данные dentry создаются на лету на основе информации, которая сохраняется в файле dentry. Файл этот находится во вторич. памяти.

• Dentry - это элемент пути в полном имени файла. Имя файла начинается с корневого каталога "/" и дальше элемент пути, до след. слэша.

• На лекции показывалось содержание файла dir-file. Это всё хранится на диске. На базе этой инф-ции выполняется заполнение кэша dentry.

• Dentry создаются на лету - при обращении к файлу система проходит по элементам пути и полученную инф-цию сохраняет в кэше.

Имя файла ≠ идентификатор файла. Один и тот же файл может иметь несколько имён - это жесткие ссылки (hard link)

• Сначала элементы пути ищутся в кэше; если они в кэше не найдены, то производится обращение к диску.

! идентификатор процесса - integer (не long)
(можно создать столько пр-ссов, какова велечина типа integer)

kstrint() - в лабе при передаче pid

• Кэш нужен чтобы существенно сократить время доступа к файлу. Кэш в ядре находится.

смысл:

• Поскольку чтение записи каталога с диска и создание соотв. объекта dentry требует значительного времени, имеет смысл хранить в памяти объекты dentry, с которыми работа была закончена (т.е. они перестали нам встречаться, мы начали с другими файлами работать), но они могут понадобиться повторно, чтобы обратиться к тому же файлу, или другому файлу в этот же подкаталог.

• Для максимально эффективности работы с элементами пути (dentries) в Linux используется кэш dentry, состоящий из 2х видов структур данных:

1) • Наборы объектов dentry в 1 из 3х состояний: in_use (использ-ся), unused (не исп-ся), negative (отриц.)

2) • Хэш таблицы для быстрого получения объекта dentry, связанного с заданным именем файла и заданной директорией. Если требуемый объект dentry не включён в кэш dentry, то ф-ция хэширования (hashing function) возвращает null!

• Кэш dentry и кэш inode связаны м/у собой. (Как минимум файлы директ имеют inode) (КАК УПРАВЛЯЮЩИЙ)

• Кэш dentry действует как контроллер для кэша inode.

В кэше dentry находится инф-ция об объектах dentry, а в кэше inode инф-ция об объектах inode (struct inode)

• айноды в памяти ядра, связанные с unused dentries, не удаляются, пока кэш dentry их ещё использует.

Т.о. объекты inode хранятся в оперативн. памяти (RAM) и на них можно быстро ссылаться, напредставлял соотв-щих dentries.

D — файл директ

D — uninterr. sleep

Файлы нужны для долговременного хранения данных

Короткое имя файла (shortname) это фактически последний dentry в имени файла

• Все dentries unused включены в дву-связный (struct list_head) список least recently used (LRU)
наименее используемые в послед. время

• Dentry objects кот. были освобождены последними, помещаются вперед. (in front of list)
Т.о. least recently used dentry objects всегда находятся ближе к концу списка

(Мыслить надо стеком)

• Адреса первого и последнего элементов списка LRU хранятся в полях next и prev в dentry unused variable.

struct dentry
{
..
struct list_head d_lru;
..
}

• Поле d_lru (объекта dentry) содержит указатель на соседа в списке dentries ((в данном случае на двух: prev и next). Каждый используемый (in_use) объект dentry включен в двусвяз. список, который определяется указателем в struct inode ? i_dentry. (это union)

struct hlist_head в объекте inode

• На этот список ссылается поле в объекте inode. !! До тех пор пока inode связан с несколькими hard link-ами этот список востребован.

• В struct dentry есть поле d_alias - это поле в объекте denty хранит адрес соседних элементов в списке (inode alias list)

LRU: когда происходит обращение к странице, она в начало (наиболее используемые) перемещается; а удаляем мы из головы (из конца)

· Объект dentry в состоянии in use может стать negative, когда на него будет удалена последняя жесткая ссылка. В этом случае объект dentry перемещается в список LRU unused dentries.

· Все эти связные списки и кэши хранятся в области данных ядра системы.

· Хэш-таблица представляется как массив dentry hash table array. Каждый элемент массива явл. указателем на список dentries, имеющих одно (и то же значение) hash-table (the hash to the same hash-table value). Размер этого массива зависит от доступного объема оперативн. памяти (RAM)

· В struct dentry есть поле d_hash, которое содержит указатель на соседний элемент в связ. с этим списке по величине hash-table.

Хэш ф-ция задает это значение из обоих адресов dentry object дир-рии и имени файла.

· Ф-ция d_lookup() (её нет в struct dentry_operations)

Она есть в коде ядра

Slab кэш

slab — от. англ. — брусок (кирпич)

\# cat /proc/slabinfo — здесь слабы посмотреть можно

· Со (слабами) slab работают только объекты ядра.

· В ядре имеется огранич. кол-во типов объектов (экземпляров структур), но при создании объекта поле структ-ры д.б. проинициализированы — это время занимает.

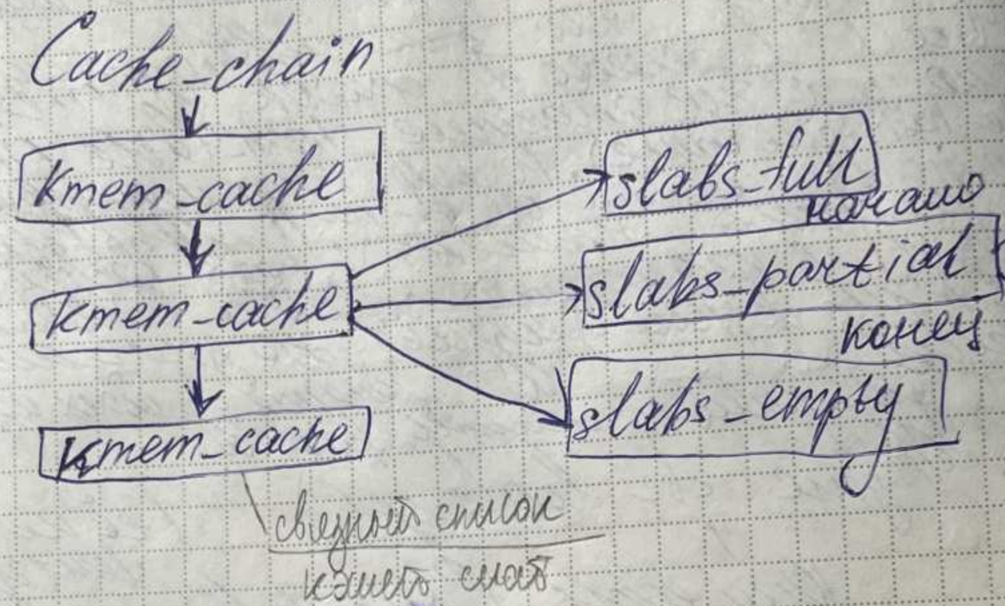
· Идея: не удалять проинициализир. структуру, а сохранять её, чтобы повторно использовать, поскольку кол-во объектов, которыми оперирует ядро, ограничено. Чтобы не создавать каждый раз заново объекты, инициализируя поле соотв. струк-ры, сложно эту

фрагментированную стр-ру сохранить: …так много

• Разные стр-ры имеют разные размеры, поэтому так много слаб-кэшей.

Доп. выгоды от испол. слабов — устранение фрагментации (в тем виде: более стр-ура кода, но она заполнена не полностью — скрытая фрагментация) деле страниц. преобразование ядра

Пример: время стр-ры размером 16 байт. Под нее будет выделена стр-ра (т.к. памяти ф-ции выделение). А slab позволяет в одну стр-цу поместить много таких объектов



Задание на лабу VFS

В ядре 3 ф-ции:
- generic
- symple
- litter

— это всё работающие коды

loop, кот. мы создавали — вирт. блочное устр-во

Надо написать свою ф-цию mount. В нашей ф-ции mount в заголовке вызыв-ие ф-ции fill super

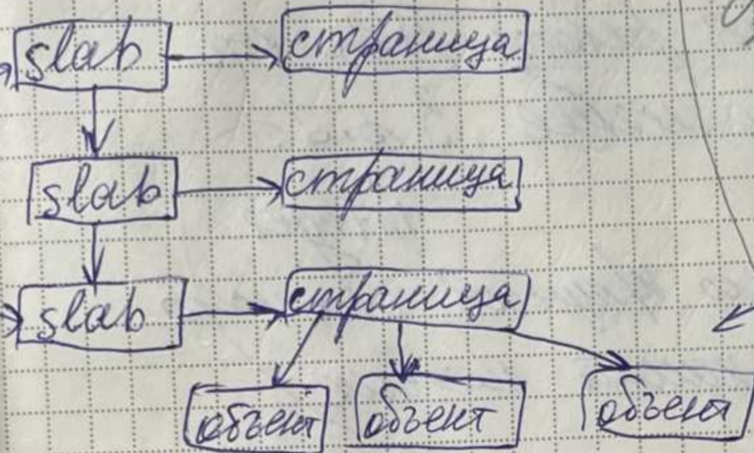
- За основу берём ф-цию из ядра (а не методички), т.е. берём код ядра (generic или symple)
- Удаление superblock можем сами не писать, а взять generic/symple?
- В ядре есть ф-ция создания slab кэша, записи в него

fill_libber_super

- Нам нужен один inode; в ядре создаём слабы

- Если потоки detach, то они завершатся когда завершится главный поток и detach потоки успевали дочитать.

потоки detach можно заменить на join или заставить глав. поток ждать завершения доп. потоков



в странице находятся проинициализир-ые объекты

15.05 Замечания по лабам: Буф/Небуф. вывод

struct stat — для анализа использ. эту стр-ру

Если пишем по 2м независ. файл. дескрипторам

00
11 ← выводим pos
22

буква записанная по fd1 будет затёрта той, что записана по fd2

Лаба 1го модуль task struct.

on_cpu везде 0, когда мы выводим инф-ю

это флаг. Чтобы номер процессора вывести, на

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 ссылок. Т.е. м.б. несколько объектов dentry, имеющих один и тот же inode

II часть: двусвязный список dentry-ви

(least recently used)
наименее используемых 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-ции хэширования

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

· Хэш-таблица представляется в виде массива dentry_hashtable, в котором каждый элемент явл-ся указателем на список dentries, которые хэшируют в одно и то же значение.

· Размер хэш-таблицы зависит от доступного объема физ. памяти в системе (т.е. оперативн. памяти)

· 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

Пример: полное имя файла /home/mydir/c_f/myserver.c

При каждом обращении к файлу myserver.c виртуал. 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) задан объект dentry валидный и эта функция 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 сравниваются 2 имени файла ("name" и "namee" например)

(4) когда d_count становится = 0, вызывается ф-ция d_delete

(5) вызывается VFS когда опред. dentry переходит в состояние free

(6) вызывается VFS когда объект dentry теряет связанный inode.

Inode

- Объект inode описывает физич. файл т.е. реально существующий файл, в котором находятся записанные в него данные. Такой файл может находиться во вторич. памяти или в физич. памяти, но в ∀ случае файл будет иметь представление в системе в виде объекта inode.
- Объект inode — это проинициализированная стр-ра inode

. В отличие от Unix, в Linux нет понятия vinode (виртуальный inode). В Linux ∃ 2 типа стр-ры struct inode:

1) стр-ра inode ядра

Эта стр-ра содержит инф-цию об