### proc\_dir\_entry

| **typedef** **int** (**read\_proc\_t**)(**char** \* page, **char** \*\*start, **off\_t** off, **int** count, **int** \*eof, **void** \*data); **typedef** **int**(**write\_proc\_t**)(struct file \* file, **const** **char** \_\_user. \*buffer, **unsigned** **long** count, **void** \*data); **struct** **proc\_dir\_entry** {  *//!Для файлов и поддиректорий ФС /proc*  **unsigned** **int** low\_ino; *// номер inode*  **unsigned** **short** namelen;  **const** **char** \*name; *// имя виртуального файла*  **mode\_t** mode; *// режим доступа*  **nlink\_t** nlink; *// количество линков - чет странное*  **uid\_t** uid; *// уникальный номер пользователя -- владельца файла*  **gid\_t** gid; *// уникальный номер группы, которой принадлежит файл*  **loft\_t** size; *// loft=unsigned long*  **const** **struct** **inode\_operations** \***proc\_iops**; *// функции-обработчики операций с inode*  **const** **struct** **file\_operations** \***proc\_fops**; *// функции-обработчики операций с файлом*  **struct** **proc\_dir\_entry** \***next**, \***parent**, \***subdir**;  **void** \*data; *// Указатель на локальные данные*  **atomic\_t** count; *// счетчик ссылок на файл*  **read\_proc\_t** \*read\_procc; *// функция чтения из /proc*  **write\_proc\_t** \*write\_proc; *// функция записи в /proc* } **#include <linux/types.h>** **#include <linux/fs.h>** **extern** struct proc\_dir\_entry \***proc\_create\_data**(**const** **char** \*, **umode\_t**, struct proc\_dir\_entry \*, **const** struct file\_operations \*, **void** \*); struct proc\_dir\_entry \***proc\_create**(**const** **char** \*name, **umode\_t** mode, struct proc\_dir\_entry \*parent, **const** struct file\_operations \*proc\_fops); **extern** struct proc\_dir\_entry \***proc\_symlink**(**const** **char** \*, struct proc\_dir\_entry \*, **const** **char** \*); **extern** struct proc\_dir\_entry \***proc\_mkdir**(**const** **char** \*, struct proc\_dir\_entry \*); struct proc\_dir\_entry \***create\_proc\_read\_entry**( **const** **char** \*name, **mode\_t** mode, struct proc\_dir\_entry \*base, **read\_proc\_t** \*read\_proc, **void** \*data ); |
| --- |

### file\_operations

| **struct** **file\_operations** {  *//!старая версия функ работы с файлами*  **struct** **module** \***owner**; *// указатель на модуль рассматриваемой подсистемы.*   **loff\_t** (\*llseek) (struct file \*, **loff\_t**, **int**);  **ssize\_t** (\*read) (struct file \*, **char** \_\_user \*, **size\_t**, **loff\_t** \*);  **ssize\_t** (\*write) (struct file \*, **const** **char** \_\_user \*, **size\_t**, **loff\_t** \*);  ...  **int** (\*open) (struct inode \*, struct file \*);  **int** (\*flush) (struct file \*, **fl\_owner\_t** id);  **int** (\*release) (struct inode \*, struct file \*);  ...  **int** (\*lock) (struct file \*, **int**, struct file\_lock \*); *// специфичная для файловой системы часть механизма блокировки области файла POSIX* } \_\_randomize\_layout; |
| --- |

### proc\_ops

| **struct** **proc\_ops** {  *//!современная версия функц работы с файлом*  **unsigned** **int** proc\_flags;  **int** (\*proc\_open)(struct inode \*, struct file \*);  **ssize\_t** (\*proc\_read)(struct file \*, **char** \_\_user \*, **size\_t**, **loff\_t** \*);  **ssize\_t** (\*proc\_read\_iter)(struct kiocb \*, struct iov\_iter \*);  **ssize\_t** (\*proc\_write)(struct file \*, **const** **char** \_\_user \*, **size\_t**, **loff\_t** \*);  */\* mandatory unless nonseekable\_open() or equivalent is used \*/*  **loff\_t** (\*proc\_lseek)(struct file \*, **loff\_t**, **int**);  **int** (\*proc\_release)(struct inode \*, struct file \*);  **\_\_poll\_t** (\*proc\_poll)(struct file \*, struct poll\_table\_struct \*);  **long** (\*proc\_ioctl)(struct file \*, **unsigned** **int**, **unsigned** **long**);  **long** (\*proc\_compat\_ioctl)(struct file \*, **unsigned** **int**, **unsigned** **long**);  **int** (\*proc\_mmap)(struct file \*, struct vm\_area\_struct \*);  **unsigned** **long** (\*proc\_get\_unmapped\_area)(struct file \*, **unsigned** **long**, **unsigned** **long**, **unsigned** **long**, **unsigned** **long**); } \_\_randomize\_layout; |
| --- |

### inode

| **struct** **inode //ФС** {  **umode\_t** i\_mode; *//права доступа*  **unsigned** **short** i\_opflags;  **kuid\_t** i\_uid;  **kgid\_t** i\_gid;  **unsigned** **int** i\_flags;  **const** **struct** **inode\_operations** \***i\_op**; */\* указатель на ту таблицу операций с индексом которые определены в конкретной системе для работы с этим индексом. Такая структура создается для конкретной файловой системы \*/*  **struct** **super\_block** \***i\_sb**; */\*указатель на связаный суперблок, так как именно он хранит информацию об inode данной файловой системы. \*/*  **struct** **address\_space** \***i\_mapping**;  ...  **unsigned** **long** i\_ino;  */\*! inode содержит информацию о файле. Копя индек, котр наход в памят (в ядре), содер поля, котор нет в диск: Поля, ответств за блокир, Лог номер устр, Номер индекса (unsigned long i\_ino;). В диск это имя не нужно, т.к. на диске индексы хранятся в лин мас, и ядро идентиф индекс по его смещ, Ссылки на др индексы (для орган в ядре хеш-очереди), Счетчик ссылок на файл, в кот содер число, указыв, сколько раз был откр этот файл. Если это ВФС, то может не быть копии!\*/*  **union**{  **const** **unsigned** **int** i\_nlink;  **unsigned** **int** \_i\_nlink;  };  **dev\_t** i\_rdev;  **loff\_t** i\_size;  **struct** **timespec** **i\_atime**;  **struct** **timespec** **i\_mtime**;  **struct** **timespec** **i\_ctime**;  ...  **unsigned** **int** i\_blkbits;  **blkcnt\_t** i\_blocks;  **struct** **hlist\_node** **i\_hash**;  **struct** **list\_head** **i\_io\_list**;  ...  **struct** **list\_head** **i\_lru**; */\*inode LRU list. LRU испол для inode. Если отк файл, то очень вероят что вы будете работать с этим файлом.\*/*  **union**{  **struct** **hlist\_head** **i\_dentry**;  **struct** **rcu\_head** **i\_rcu**;  };  **atomic\_t** i\_count; */\* счётчик ссылок \*/*  **atomic\_t** i\_writecount; */\* счетчик исп-я для записи \*/*  count **struct** **file\_operations** \***i\_top**;  **struct** **address\_space** **i\_data**;  **struct** **list\_head** **i\_devices**; */\* список блочных устройств \*/*  **union**  {  **struct** **pipe\_inode\_info** \***i\_pipe**;  **struct** **block\_device** \***i\_bdev**;  **struct** **cdev** \***i\_cdev**;  **char** \*i\_link;  **unsigned** i\_dir\_seq;  };  ... } |
| --- |

| **struct** **inode** { // упр устр-ми  **umode\_t** i\_mode; ... **const** **struct** **inode\_operations** \***i\_op**; */\* операции, определенные на inode\*/* **struct** **super\_block** \***i\_sb**;  */\*!*  *Следуя парадигме UNIX - в UNIX все – файл, внеш устр представ в сист как спец файлы и имеют inode, кот содер метаданные о файле. Прил обращ к символ и блочным устройст через inode. Когда созд inode устр, он сопостав с номер major и minor.*  *!\*/* **struct** **address\_space** \***i\_mapping**; ... */\* Stat data, not accessed from path walking \*/* **unsigned** **long** i\_ino; */\*номер inode\*/* ... **dev\_t** i\_rdev; */\*фактический номер устройства, содержащий major, minor (как st\_rdev, который описан ниже)\*/* ... **union** { **const** **struct** **file\_operations** \***i\_fop**; */\* former ->i\_op->default\_file\_ops \*/* **void** (\*free\_inode)(struct inode \*); }; ... **struct** **list\_head** **i\_devices**;  **union** { *// вот как раз объединение - может храниться блочное устройство/символьное устройство/(что-то еще, что нас здесь не интересует)*  **struct** **pipe\_inode\_info** \***i\_pipe**; **struct** **block\_device** \***i\_bdev**; **struct** **cdev** \***i\_cdev**; **char** \*i\_link; **unsigned** i\_dir\_seq; }; …  } \_\_randomize\_layout; |
| --- |

*Приложение может извлечь метаданные из inode, используя системный вызов stat(), который возвращает структуру stat.*

| ***struct******stat*** *{* ***dev\_t*** *st\_dev; /\* ID of device containing file - описыв устро, на котором «живет» inode. Это устр идентиф-руется комбин его major идентиф-ром, который идентиф-рует общий класс устр, и minor идентиф-ром, который идентиф-рует конкрет экземл в общ классе.\*/* ***ino\_t*** *st\_ino; /\* Inode number \*/* ***mode\_t*** *st\_mode; /\* File type and mode - определяет тип файла (обычный/директория/символьное устройство/...) и режим \*/* ***nlink\_t*** *st\_nlink; /\* Number of hard links \*/* ***uid\_t*** *st\_uid; /\* User ID of owner \*/* ***gid\_t*** *st\_gid; /\* Group ID of owner \*/* ***dev\_t*** *st\_rdev; /\* Device ID (if special file) - устр, представл-ное данным inode. Если этот файл, имеющ inode, пред-ет устр, тогда inode содер major и minor идентиф этого устр. \*/* ***off\_t*** *st\_size; /\* Total size, in bytes \*/* ***blksize\_t*** *st\_blksize; /\* Block size for filesystem I/O \*/* ***blkcnt\_t*** *st\_blocks; /\* Number of 512B blocks allocated \*/* ***struct******timespec******st\_atim****; /\* Time of last access \*/* ***struct******timespec******st\_mtim****; /\* Time of last modification \*/* ***struct******timespec******st\_ctim****; /\* Time of last status change \*/  /\* ! (т.е. если inode описыв устр, то разн: st\_dev - устр, где расположен сам inode, st\_rdev - устр, кот он описыв) !\*/ };* |
| --- |

### file\_system\_type

| **struct** **file\_system\_type** {  **const** **char** \*name;  **int** fs\_flags; */\*!  явл глоб "определителем" ФС и содержит имя ФС, фун иниц и уничтож суперблока. !\*/* **#define FS\_REQUIRES\_DEV 1 //требуется блочное устройство** **#define FS\_BINARY\_MOUNTDATA 2** **#define FS\_HAS\_SUBTYPE 4** **#define FS\_USERNS\_MOUNT 8 */\* Can be mounted by userns root \*/*** **#define FS\_DISALLOW\_NOTIFY\_PERM 16 */\* Disable fanotify permission events \*/*** **#define FS\_ALLOW\_IDMAP 32 */\* FS has been updated to handle vfs idmappings. \*/*** **#define FS\_THP\_SUPPORT 8192 */\* Remove once all fs converted \*/*** **#define FS\_RENAME\_DOES\_D\_MOVE 32768 */\* FS will handle d\_move() during rename() internally. \*/***  **int** (\*init\_fs\_context)(struct fs\_context \*);  **const** **struct** **fs\_parameter\_spec** \***parameters**;  **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**;  **struct** **hlist\_head** **fs\_supers**;   **struct** **lock\_class\_key** **s\_lock\_key**;  **struct** **lock\_class\_key** **s\_umount\_key**;  **struct** **lock\_class\_key** **s\_vfs\_rename\_key**;  **struct** **lock\_class\_key** **s\_writers\_key**[**SB\_FREEZE\_LEVELS**];   **struct** **lock\_class\_key** **i\_lock\_key**;  **struct** **lock\_class\_key** **i\_mutex\_key**;  **struct** **lock\_class\_key** **i\_mutex\_dir\_key**; }; |
| --- |

### superblock

| **struct** **super\_block** {  **struct** **list\_head** **s\_list**; */\* двусвязный список всех активных суперблоков; \*/*  **dev\_t** s\_dev; */\* предназначено для файловых систем, требующих наличие блочного устройства, т.е. для файловых систем, зарегистрированных с флагом FS\_REQUIRES\_DEV, это поле представляет собой копию i\_dev блочного устройства. \*/*  **unsigned** **char** s\_blocksize\_bits;  **unsigned** **long** s\_blocksize; *// размер блока, минимальная единица адресации информации на диске*  **struct** **file\_system\_type** \***s\_type**;  **const** **struct** **super\_operations** \***s\_op**; *// указатель на структуру (список) super\_operations, которая содержит специфичные для заданной файловой системы методы, такие как чтение/запись inode и пр.*  **unsigned** **long** s\_flags;   **unsigned** **long** s\_magic; *// "магическое" число файловой системы. Используется файловой системой minix для различения разных вариантов ее.*  **struct** **dentry** \***s\_root**; *// dentry корня файловой системы.*   **struct** **rw\_semaphore** **s\_umount**; *//read-write семафоры. Работают по принципу читатели-писатели*  **int** s\_count;  **atomic\_t** s\_active;   **struct** **list\_head** **s\_mounts**; */\* список всех структур vfsmount для каждого смонтированного экземпляра данного суперблока. \*/*  **struct** **block\_device** \***s\_bdev**;   **void** \*s\_fs\_info; */\* Filesystem private info \*/*  /\*!*СБ - структура, описыв смонтир ФС (конкр ФС). Она содер инфо необхо для монтиров и управл ФС. Каждая ФС имеет один СБ, но на диске СБ наход в нескольких экземплярах (для надежности).* !\*/   **char** s\_id[32]; */\* Informational name \*/*  **uuid\_t** s\_uuid; */\* UUID - используется для идентификации разделов диска\*/*   **fmode\_t** s\_mode;  */\*  \* Подтип файловой системы. Если поле типа файловой системы не является пустым  \* в /proc/mounts будет указано "type.subtype".  \*/*  **const** **char** \*s\_subtype;   **const** **struct** **dentry\_operations** \***s\_d\_op**; */\* default d\_op for dentries \*/*   */\* Количество индексов с nlink == 0, но на которые все еще ссылаются \*/*  **atomic\_long\_t** s\_remove\_count;  */\*  \* Владение пользовательским пространством имен и контекстом по умолчанию, в котором для  \* интерпретация UID файловой системы, gid, квот, узлов устройств,  \* xattrs и метки безопасности.  \*/*  **struct** **user\_namespace** \***s\_user\_ns**;   */\*Структура list\_lru по сути является просто указателем на таблицу списков lru для каждого узла, каждый из которых имеет свой собственный spinlock.  \* \* Нет необходимости помещать их в отдельные строки кэша.  \*/*  **struct** **list\_lru** **s\_dentry\_lru**;  **struct** **list\_lru** **s\_inode\_lru**;  **struct** **rcu\_head** **rcu**;  **struct** **work\_struct** **destroy\_work**;   **struct** **mutex** **s\_sync\_lock**; */\* sync serialisation lock \*/*  */\* s\_inode\_list\_lock защищает s\_inodes \*/*  **spinlock\_t** s\_inode\_list\_lock \_\_\_\_cacheline\_aligned\_in\_smp;  **struct** **list\_head** **s\_inodes**; */\* all inodes \*/*   **spinlock\_t** s\_inode\_wblist\_lock;  **struct** **list\_head** **s\_inodes\_wb**; */\* //write back inodes - список грязных inode - измененное значение \*/* } \_\_randomize\_layout; |
| --- |

### dentry

| **struct** **dentry** {  */\*! Объект dentry связывает комп пути с соотв индекс. Напр, при поиске пути /tmp/test ядро созд объект dentry для корн кат, затем для tmp корн кат, затем для test. !\*/*  **unsigned** **int** d\_flags;  ...  */\* ! У объектов dentry нет соотв отобр на диске, ВФС созд об на лету, то есть в оп памяти по строковому представл имени пути.! \*/*  **struct** **hlist\_bl\_node** **d\_hash**; *//lookup hash list*  **struct** **dentry** \***d\_parent**;  **struct** **qstr** **d\_name**;  **struct** **inode** \***d\_inode**; *//where the name belongs to -Null is negative*  **const** **struct** **dentry** **operations** \***d\_op**;  **struct** **super\_block** \***d\_sb**; ...  **union**{  **struct** **list\_head** **d\_lru**; *//LRU list Списки двусвязные. Удаление из начала, добавление в конец, не часто используемая операция.*   **wait\_queue\_head\_t** \*d\_wait;  }  **struct** **list\_head** **d\_cchild**; - список подкаталогов родителя  **struct** **list\_head** **d\_subdirs** - подкаталоги текущего ... } //4 возм сост: Free, Unused, Isused, Negative. Inode **static** **inline** **unsigned** **d\_count**(**const** struct dentry \*dentry) {  **return** dentry->d\_lockref.count; } |
| --- |

### file

| **struct** **file** {  //!Определяет дескриптор открытого файла в системе.  **union** {  **struct** **llist\_node** **fu\_llist**;  **struct** **rcu\_head** **fu\_rcuhead**;  } f\_u;  **struct** **path** **f\_path**;  **struct** **inode** \***f\_inode**; */\* cached value \*/*  **const** **struct** **file\_operations** \***f\_op**; *// указатель на список file\_operations, который содержит адреса методов для работы с файлом.*   */\*Protects f\_ep, f\_flags. Must not be taken from IRQ context.\*/*  **spinlock\_t** f\_lock;  **enum** rw\_hint f\_write\_hint;  **atomic\_long\_t** f\_count; *// счетчик ссылок*  **unsigned** **int** f\_flags; *// флаги O\_XXX системного вызова open(2),*  **fmode\_t** f\_mode; *// режим доступа к файлу*  **struct** **mutex** **f\_pos\_lock**;  **loff\_t** f\_pos; *// f\_pos: текущая позиция чтения/записи в файле.*  *struct fown\_struct f\_owner; //владелец файла*  **const** **struct** **cred** \***f\_cred**;  **struct** **file\_ra\_state** **f\_ra**;   u64 f\_version; **#ifdef CONFIG\_SECURITY**  **void** \*f\_security; **#endif**  */\* needed for tty driver, and maybe others \*/*  **void** \*private\_data;  **#ifdef CONFIG\_EPOLL**  */\* Used by fs/eventpoll.c to link all the hooks to this file \*/*  **struct** **hlist\_head** \***f\_ep**; **#endif */\* #ifdef CONFIG\_EPOLL \*/***  **struct** **address\_space** \***f\_mapping**;  **errseq\_t** f\_wb\_err;  **errseq\_t** f\_sb\_err; */\* for syncfs \*/* } \_\_randomize\_layout |
| --- |

### super\_operations

| **struct** **super\_operations** {  *// создание и инициализация нового объекта inode.*   *// Данная формулировка не имеет никакого отношения к ООП, но подчеркивается, что это не тип, а экземпляр соответствующего типа.*  **struct** **inode** \*(\***alloc\_inode**)(**struct** **super\_block** \***sb**);   *// Удаление объекта*  **void** (\*destroy\_inode)(struct inode \*);   *// dirty - изменённый inode. Вызывается, чтобы обновить информацию в журналируемых файловых системах (например EXT3)*  **void** (\*dirty\_inode)(struct inode \*, **int** flags); */\*! Эта струк описыв опер, опреде на СБ, если например в ФС необх выпол запись, то будет вызыв функ-я write\_superblock следующим образом: sb->s\_op->write\_super(sb); где write\_super передается указатель на СБ в качестве параметра. !\*/* *// Запись inode на диск (второй параметр сейчас указывает на то, как должна выполняться операция, например, синхронно)*  **int** (\* write\_inode)(struct inode \*, struct writeback\_control \* wbc);   *// Вызывается при исчезновении последней ссылки на inode. Обычная ФС Unix никогда не определяет эту фукцию, в этом случае подсистема ВФС удаляет этот inode*   **int** (\* drop\_inode)(struct inode \*);   *// Вызывается при размонтировании ВФС для освобождения указанного суперблока*  **void** (\* put\_super)(struct super\_block \*);   *// Синхронизирует метаданные ФС с данным на диске. Параметр указывает на то, синхронно или асинхронно будет выполняться данное действие.*  **int**(\* sync\_fs)(struct super\_block \*sb, **int** wait);  ... *// Получение статистики для ФС (записывается в структуру statfs)*  **int** (\* statfs)(struct dentry, struct kstatfs \*);   *// Вызывается, когда ФС монтируется с другими параметрами монтирования*  **int** (\*remount\_fs)(struct super\_block, **int** \*, **char** \*);   *// Прерывание монтирования*  **int** (\*umount\_begin)(struct super\_block \*); } |
| --- |

### inode\_operations

| **struct** **inode\_operations** { **struct** **dentry** \*(\***lookup**)(**struct** **inode** \*, **struct** **dentry** \*, **unsigned** **int**); *//- Поиск найденного индекса в указанном каталоге. Для поиска inode требуется, чтобы VFS вызывала метод lookup() родительского каталога inode. Как только VFS находит требуемый dentry (и, следовательно, inode), можно открывать файл системным вызовом open(2) или получать информацию о файле функцией stat(2).* **const** **char** \*(\*get\_link)(struct dentry \*, struct inode \*, struct delayed\_call \*);  **int** (\*permissions)(struct inode \*, **int**); *//- проверяет права доступа к указанному, на который указывает inode. Если норм - 0, иначе -1. Поле intmask для большинства ФС выставляется в 0. Поэтому обычно обращение к методу ... сравнение битовых полей с указанной маской* ... **int** (\*link)(struct dentry \*, struct inode. \*, struct dentry \*); *//- жесткая ссылка, как правило, создается чтобы один и тот же файл был доступен из разных директорий. На старую и новую dentry. Просто еще одно равноправное имя файла* **int** (\*symlink)(struct inode \*, struct \*dentry, **const** **char** \*); *//- имя специального. Файла. Созданный специально файл* **int** (\*create)(struct inode \*, struct dentry \*, **umode\_t**, **bool**); *// вызывается двумм системными вызовами - open(2) и create(2) - только в случае если они вызываются для обычного файлаы. При этом создается новый inode с указанным режимом доступа mode, и этот inode связан с элементом пути (обычно называем подкаталогом), то есть dentry.*  **int** (\*mkdir)(struct inode \*, struct dentry\*, **umode\_t**); *//- создает директорию. Есть указатель на struct inode. Есть права доступа, struct dentry - родительский элемент пути* **int** (\*rmdir)(struct inode \*, struct dentry \*); **int** (\*mknod)(struct inode \*, struct dentry \*, **umode\_t**, **dev\_t**); *//- создание именованного канала было в 1 по линукс. С помощью этой функции можно создать файл устройства. Информация о созданном устройстве будет в dev\_t.*  **int** (\*rename) (struct inode \*, struct dentry \*, struct inode\*, struct dentry \*); **int** (\*atomic\_open)(struct inode \*, struct dentry \*, struct file \*, **unsigned** open\_flag, **umode\_t** create\_mode, **int** opened); *//- вызывается из open, при этом файл или создается, или открывается одной неделимой операцией* ... } |
| --- |

### task\_struct

| **struct** **task\_struct**{  **struct** **thread\_info** **thread\_info**; *// не указатель, а переменная. Ссылается на main\_task\_structure. Потоки ввели разработчики юникс именно, чтобы сократить время переключения контекста. Для единообразия управления всеми потоками всегда в программе есть так называемый основной поток. Можно предположить, что это и есть основной поток.* **unsigned** **int** \_\_state;  ...  **int** prio; *//приоритет, связано с планированием, есть еще много подобных полей. Ps -ajx. Подобно этому в syslog мы выведем информацию*  **int** static\_prio;  */\*! Это ДЕСКРИПТОР проц. Описыв запущ в системе проц, содер всю необход инф, чтобы можно было управ этим проц и выдел ему ресурсы. Создаётся динам, кроме процесса init. Для проц init созд струк init\_task.- стат струк.*  *Ядро испол циклич двусвяз список записей* ***struct******task\_struct*** *хран дескрип проц. В "начале" списка наход процесс* ***init****. С помощью макроса* ***next\_task*** *можно получить след задачу из списка. Макрос* ***current*** *- указ на* ***task\_struct*** *тек исполн проц. !\*/*  ...  **struct** **list\_head** **tasks**;*//ядро поддерживает специальные функции работы со списками. Именно связный список, а не массив, так как массивы приводят к дополнительным накладным расходам. Проблема вставки*  …  *//У процесса виртуальное АП. Есть указатель memory management, которая и описывает ВАП. Нет указателей на физическую память, так как она выделяется при запросах, и это другая работа системы. Struct page описывает физическую память, но она видимо не здесь?*  **struct** **mm\_struct** \***mm**;  **struct** **mm\_struct** \***active\_mm**;  ...  **pid\_t** pid; *//идентификатор процесса*  **pid\_t** tgid;  ...  **struct** **task\_struct** \_\_**reu** \***parent**; *// указатель на дескриптор процесса-предка*  **struct** **pid** \***thread\_pid**;  **struct** **list\_head** **children**;  **struct** **list\_head** **sibling**;  ...  */\* Filesystem information \*/*  **struct** **fs\_struct** \***fs**; *//файловая подсистема, к которой относится файл*  */\* Open file information \*/*  **struct** **files\_struct** \***files**; *//открытые процессом файлы*  */\* Namespaces \*/*  **struct** **nsproxy** \***nsproxy**;  ...  **char** comm[TASK\_COM\_LEN]; *// Имя файла, которым был процесс до того, как мы его запустили*  */\* CPU-specific state of this task: \*/*  **struct** **thread\_struct** **thread**; *// еще в семах есть 2 неизвестных поля, но вот такие комментарии* *//можно установить, чтобы группа процессов получала одни и те же сигналы* *//идентификатор сессии - session\_id* } |
| --- |

### namespace

| **struct** **namespace** {  //!позволяет каждому процессу иметь своё видение смонтированной файловой системы  **atomic\_t** count;  **struct** **vfsmount** \***root**;  **struct** **list\_head** **list**;  **struct** **rw\_semaphore** **sem**; }; |
| --- |

### vfsmount

| *//структура, описывающая подмонтированные ФС.*  **struct** **vfsmount** {  **struct** **dentry** \***mnt\_root**; */\* root of the mounted tree \*/*  **struct** **super\_block** \***mnt\_sb**; */\* pointer to superblock \*/*  **int** mnt\_flags;  **struct** **user\_namespace** \***mnt\_userns**; } \_\_randomize\_layout; |
| --- |

| **typedef** **int** (\***fill\_super\_t**)(struct super\_block \*, **void** \*, **int**); struct dentry \***mount\_bdev**(struct file\_system\_type \*fs\_type, **int** flags, **const** **char** \*dev\_name, **void** \*data, **fill\_super\_t** fill\_super); struct dentry \***mount\_single**(struct file\_system\_type \*fs\_type, **int** flags, **void** \*data, **fill\_super\_t** fill\_super); struct dentry \***mount\_nodev**(struct file\_system\_type \*fs\_type, **int** flags, **void** \*data, **fill\_super\_t** fill\_super); |
| --- |

### nsproxy

| */\* Структура, содержащая указатели на все пространства имен для каждого процесса*  *nsproxy совместно используется задачами, которые совместно используют все пространства имен. Как только одно пространство имен клонируется или не разделяется, копируется nsproxy.  \*/* **struct** **nsproxy** {  **atomic\_t** count;  **struct** **uts\_namespace** \***uts\_ns**;  **struct** **ipc\_namespace** \***ipc\_ns**;  **struct** **mnt\_namespace** \***mnt\_ns**;  **struct** **pid\_namespace** \***pid\_ns\_for\_children**;  **struct** **net** \***net\_ns**;  **struct** **time\_namespace** \***time\_ns**;  **struct** **time\_namespace** \***time\_ns\_for\_children**;  **struct** **cgroup\_namespace** \***cgroup\_ns**; }; |
| --- |

### fs\_struct

| **struct** **fs\_struct** {  **atomic\_t** count; *// количество пользователей (количество ссылок на структуру)*  **rwlock\_t** lock; *// Блокировка чтения/записи - защита структуры.*   **int** umask; *// права доступа к файлу, используемые по умолчанию*  */\*!*  *struct fs\_struct \*fs описывает файловую систему, к которой относится процесс.*  *Исполняемый файл мб запущен с подмонтирвоанной ФС. file system information*  *!\*/*  **struct** **dentry** \***root**; *// корневая директория данного процесса*  **struct** **dentry** \***pwd**; *// текущая (рабочая) директория данного процесса*  **struct** **dentry** \***altroot**; *// альтернативная корневая директория*  *//struct vfsmount - структура, описывающая подмонтированные ФС.*   **struct** **vfsmount** \***rootmnt**; *// объект монтирования корневой директории*   **struct** **vfsmount** \***pwdmnt**; *// объект монтирования текущей директории*  **struct** **vfsmount** \***altrootmnt**; *// объект монтирования альтернативной корневой директории*  }; |
| --- |

### files\_struct

| **struct** **files\_struct** {  **atomic\_t** count;*// счетчик пользователей структуры*  **spinlock\_t** file\_lock; *// средство взаимоисключения*  **unsigned** **int** next\_fd; */\*! struct files\_struct \*files формирует таблицу открытых файлов процесса (собственная у каждого процесса). Если открыть более NR\_OPEN\_DEFAULT(32 или 64) файловых дескрипторов, то ядро выделяет новый массив указателей !\*/*  **unsigned** **long** close\_on\_exec\_init[1]; *//Число файлов, которые дб закрыты системным вызовом exec*  **unsigned** **long** open\_fds\_init[1];  **unsigned** **long** full\_fds\_bits\_init[1];  **struct** **file** \_\_**rcu** \***fd\_array**[**NR\_OPEN\_DEFAULT**]; *//для SMP-архитектур - многозадачность. Это массив дескрипторов файлов, открытых процессом* }; |
| --- |

### path

| **struct** **path** {  **struct** **vfsmount** \***mnt**;  **struct** **dentry** \***dentry**; } |
| --- |

### open\_flags

| **struct** **open\_flags** {  **int** open\_flag;  **umode\_t** mode;  **int** acc\_mode;  **int** intent;  **int** lookup\_flags; }; |
| --- |

### seq\_file

| **struct** **seq\_file** {  **char** \*buf;  **size\_t** size;  **size\_t** from;  **size\_t** count;  **size\_t** pad\_until;  **loff\_t** index;  **loff\_t** read\_pos;  **struct** **mutex** **lock**;  **const** **struct** **seq\_operations** \***op**;  **int** poll\_event;  **const** **struct** **file** \***file**;  **void** \***private**; };  **struct** **seq\_operations** {  **void** \* (\*start) (struct seq\_file \*m, **loff\_t** \*pos);  **void** (\*stop) (struct seq\_file \*m, **void** \*v);  **void** \* (\*next) (struct seq\_file \*m, **void** \*v, **loff\_t** \*pos);  **int** (\*show) (struct seq\_file \*m, **void** \*v); }; |
| --- |

| *//Модуль может опр только ф show(), кот должна созд все* *выход данные, кот будет содерж вирт файл. Затем метод open() файла вызывает:* **int** **single\_open**(struct file \*file, **int** (\*show)(struct seq\_file \*m, **void** \*p), **void** \*data); *//Когда придет время выв, ф show() будет вызв 1раз. Знач данных, заданное single\_open(), можно найти в приват поле струк seq\_file. При исп single\_open() программ должен исп single\_release()* |
| --- |

### 

### socket

| */\*\* \* struct socket - general BSD socket \* @state: socket state (%SS\_CONNECTED, etc) \* @type: socket type (%SOCK\_STREAM, etc) \* @flags: socket flags (%SOCK\_NOSPACE, etc) \* @ops: protocol specific socket operations \* @file: File back pointer for gc \* @sk: internal networking protocol agnostic socket representation \* @wq: wait queue for several uses \* Сокет - абстракция конечной точки соединения (коммуникации, взаимодействия). \*/* ***struct******socket*** *{  socket\_state state; //5 состояний* ***short*** *type; //тип сокета - 2 параметр функции socket* ***unsigned******long*** *flags; //используется для синхронизации доступа (те самые флаги O\_CLOEXEC и O\_NONBLOCK)* ***struct******socket\_wq*** *\_\_****rcu*** *\*****wq****;* ***struct******file*** *\*****file****; //указатель на дескриптор открытого файла (парадигма все файл: сокет - специальный файл), а file имеет указатель на struct inode* ***struct******sock*** *\*****sk****;* ***const******struct******proto\_ops*** *\*****ops****; /\*операции, связанные с протоколом (так называемый сетевой протокол) (например, TCP (transmission control) или UDP (user datagram)). Структура подобна struct file\_operations - функции, определенные на сокете. Зависят от типа и протокола\*/ };* ***#include <sys/types.h>******#include <sys/socket.h>******int******socket****(****int*** *domain,* ***int*** *type,* ***int*** *protocol); // соз кон точ соед и возвр фд, указ на эту точку. // Зад домен соед или сем адр (address family - AF). Домен опр сем проток, кот будет исполь для связи. AF\_UNIX - сем юникс на отд стоящ машин, AF\_INET - сетевые сокеты по протоколу TCP/IP // type опр тип нужного коммуник отнош (семан соед) SOCK\_STREAM упоряд полнодупл лог соед между 2 сок., SOCK\_DGRAM - опр ненадеж служ datagram без устан лог соед, когда пакеты могут перед без сохр лог пор - широковещ перед дан. Огр дл сооб, SOCK\_SEQPACKET - последов двустор кан // Задаётся определённый протокол, используемый с сокетом. Обычно для этого параметра устанавливается значение 0, в этом случае протокол устанавливается по умолчанию для соответствующего семейства и типа.* ***int******bind****(****int*** *sockfd, struct sockaddr \*addr,* ***int*** *addrlen)* ***int******listen****(****int*** *sockfd,* ***int*** *backlog); //адрес не важен. Имеет смысл только для протокола TCP, ориентированного на соединение.* ***int******connect****(****int*** *sockfd, struct sockaddr \*serv\_addr,* ***int*** *addrlen) //serv\_addr - подчеркивается, что выполняется клиентом, и клиент должен указать адрес сервера* ***int******accept****(****int*** *sockfd, struct sockaddr \*cli\_addr,* ***int*** *\*addrlen); // 1) последний аргумент с типом socklen\_t=int, 2)подчеркивается, что адрес клиента* ***#define SS\_FREE - не занят******#define SS\_UNCONNECTED - не соединен******#define SS\_CONNECTING - соединяется в данный момент******#define SS\_CONNECTED - соединен******#define SS\_DISCONNECTING - разъединяется в данный момент*** |
| --- |

| **int** **select**( **int** n, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds, struct timeval \*timeout ); **int** **pselect**( **int** n, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds, **const** struct timespec \*timeout, **sigset\_t** \*sigmask ); |
| --- |

### sockaddr

| **struct** **sockaddr** {  **unsigned** **short** sa\_family; *// Семейство адресов, AF\_xxx. Определяет семейство адресов, но точный формат не определен.*   **char** sa\_data[14]; *// 14 байтов для хранения адреса. Вид адреса зависит от выбранного вами домена.*  }; |
| --- |

### sockaddr\_un

| **struct** **sockaddr\_un**  {  **sa\_family\_t** sun\_family;  **char** sun\_path[ 108 ]; } |
| --- |

### sockaddr\_in

| **struct** **sockaddr\_in**  {  **short** **int** sin\_family; *// Семейство адресов (AF\_INET). Соответствует полю sa\_family в sock\_addr*  **unsigned** **short** **int** sin\_port; *// Номер порта. ПОРТ-ЭТО АДРЕС*  **struct** **in\_addr** **sin\_addr**; *// IP-адрес хоста (адрес в инете, который пишется через . (является структурой ради обратной совместимости, ниже). если указ INADDR\_ANY, то сервер зарег на всех адресах той машины, на которой она выполняется.*  **unsigned** **char** sin\_zero[8]; *// Дополнение до размера структуры sockaddr. почему именно 8: sizeof(struct sockaddr)-sizeof(sa\_family\_t) - sizeof(uint16\_t)-sizeof(struct in\_addr)*   }; |
| --- |

| **struct** **in\_addr** {  **unsigned** **long** s\_addr; *// по мнению Р тип \_u32* }; |
| --- |

### тип dev\_t

Для хранения номеров устройств, как старшего так и младшего, в ядре используется тип dev\_t, определенный в <linux/types.h>.

| **typedef** **\_\_kernel\_dev\_t** **dev\_t**; |
| --- |

Стандарт POSIX.1 определяет существование этого типа, но не оговаривает формат

полей и их содержание. Начиная с версии ядра 2.6.0: dev\_t является 32-х разрядным, 12 бит отведены для старшего номера и 20 - для младшего.

### struct cdev

При создании символьного устройства в данной структуре нужно заполнить только два поля: file\_operation и owner (Обязательное значение - THIS\_MODULE)

Существует два способа выделения и инициализации структуры:

1) Runtime Allocation

2) Own allocation.

| **struct** **cdev** {  **struct** **kobject** **kobj**;  **struct** **module** \***owner**;  **const** **struct** **file\_operations** \***ops**;  **struct** **list\_head** **list**;  **dev\_t** dev; *// номер устройства (эта структура обсуждалась выше)*  **unsigned** **int** count;  ...  MAJOR(**dev\_t** dev); *// получить старший*  MINOR(**dev\_t** dev); *// получить младший*  MKDEV(**int** major, **int** minor); *// страший + младший = dev\_t*  *// Если нужно пол автономную струк cdev во время выпол, то это можно сдел с помощью след кода*  **struct** **cdev** \***my\_cdev** = **cdev\_alloc**();  my\_cdev->ops = &my\_fops;  *// или можно встроить file\_op в cdev*   **void** **cdev\_init**(struct cdev \*cdev, struct file\_operations \*fops);  **int** **cdev\_add**(struct cdev \*dev, **dev\_t** num, **unsigned** **int** count); *// сообщ ядру о структуре cdev* } \_\_randomize\_layout; |
| --- |

### block\_device

| **struct** **block\_device** { */\*! Драйв блоч устр обесп доступ к устр, которые перед произв-но доступ данные блоками фиксир размера, в первую оч на дисках. Очев, что блоч устр принципиал отлич от символ устр, и эти особенн не могут не учитываться в ядре. Драйв блоч устр имеют соответ интерфейс, который опред специф задач и пробл блоч устр. Мож сказ, что драйв блоч устр - это канал между памятью ядра и вторичным хранилищем; следовательно, они могут рассматриваться как составляющие подсистемы виртуальной памяти. Большая часть дизайна блочного слоя сосредоточена на производительности.!\*/* **dev\_t** bd\_dev; */\* not a kdev\_t - it's a search key \*/* **int** bd\_openers; **struct** **inode** \* **bd\_inode**; */\* will die \*/* **struct** **super\_block** \* **bd\_super**; **struct** **mutex** **bd\_mutex**; */\* open/close mutex \*/* **void** \* bd\_claiming; **void** \* bd\_holder; **int** bd\_holders; **bool** bd\_write\_holder; **#ifdef CONFIG\_SYSFS** **struct** **list\_head** **bd\_holder\_disks**; **#endif** **struct** **block\_device** \* **bd\_contains**; **unsigned** bd\_block\_size; u8 bd\_partno; **struct** **hd\_struct** \* **bd\_part**; */\* number of times partitions within this device have been opened. \*/* **unsigned** bd\_part\_count; **int** bd\_invalidated; **struct** **gendisk** \* **bd\_disk**; **struct** **request\_queue** \* **bd\_queue**; **struct** **backing\_dev\_info** \***bd\_bdi**; **struct** **list\_head** **bd\_list**; */\* \* Личные данные. Вы, должно быть, bd\_claimed на block\_device*  *\* чтобы использовать это. ПРИМЕЧАНИЕ: bd\_claim позволяет владельцу требовать*  *\* одно и то же устройство несколько раз, владелец должен принять специальные*  *\* позаботьтесь о том, чтобы не испортить bd\_private для этого случая. \*/* **unsigned** **long** bd\_private; */\* The counter of freeze processes \*/* **int** bd\_fsfreeze\_count; */\* Mutex for freeze \*/* **struct** **mutex** **bd\_fsfreeze\_mutex**;  } \_\_randomize\_layout; |
| --- |

### block\_device\_operations

| **struct** **block\_device\_operations** { */\*! Симв устр для реги в сист нужных операций испол структуру file\_operations. Блоч устр испол для этого struct block\_device\_operations (вот такая иерархия: в struct block\_device есть struct gendisk, в которой есть block\_device\_operations), !\*/*  **int** (\*open) (struct block\_device \*, **fmode\_t**);  **void** (\*release) (struct gendisk \*, **fmode\_t**);  **int** (\*rw\_page)(struct block\_device \*, **sector\_t**, struct page \*, **unsigned** **int**);  **int** (\*ioctl) (struct block\_device \*, **fmode\_t**, **unsigned**, **unsigned** **long**);  **int** (\*compat\_ioctl) (struct block\_device \*, **fmode\_t**, **unsigned**, **unsigned** **long**);  **unsigned** **int** (\*check\_events) (struct gendisk \*disk,  **unsigned** **int** clearing);  */\* ->media\_changed() is DEPRECATED, use ->check\_events() instead \*/*  **int** (\*media\_changed) (struct gendisk \*);  **void** (\*unlock\_native\_capacity) (struct gendisk \*);  **int** (\*revalidate\_disk) (struct gendisk \*);  **int** (\*getgeo)(struct block\_device \*, struct hd\_geometry \*); */\* this callback is with swap\_lock and sometimes page table lock held \*/*  **void** (\*swap\_slot\_free\_notify) (struct block\_device \*, **unsigned** **long**);  **int** (\*report\_zones)(struct gendisk \*, **sector\_t** sector,   struct blk\_zone \*zones, **unsigned** **int** \*nr\_zones, **gfp\_t** gfp\_mask);   **struct** **module** \***owner**;  **const** **struct** **pr\_ops** \***pr\_ops**;  }; |
| --- |

### 

### gendisk

| **struct** **gendisk** { */\*! На структуру block\_device\_operations ссылается struct gendisk (объявлена в <linux/genhd.h>), которая представляет в ядре отдельное дисковое устройство. !\*/* */\* major, first\_minor and minors are input parameters only, \* don't use directly. Use disk\_devt() and disk\_max\_parts(). \*/* **int** major; */\* major number of driver \*/* **int** first\_minor; **int** minors; */\* maximum number of minors, =1 for disks that can't be partitioned. \*/* **char** disk\_name[DISK\_NAME\_LEN]; */\* name of major driver \*/* **char** \*(\*devnode)(struct gendisk \*gd, **umode\_t** \*mode); **unsigned** **int** events; */\* supported events \*/* **unsigned** **int** async\_events; */\* async events, subset of all \*/* */\* Array of pointers to partitions indexed by partno. \* Protected with matching bdev lock but stat and other \* non-critical accesses use RCU. Always access through \* helpers. \*/* **struct** **disk\_part\_tbl** \_\_**rcu** \***part\_tbl**; **struct** **hd\_struct** **part0**; **const** **struct** **block\_device\_operations** \***fops**; **struct** **request\_queue** \***queue**; **void** \*private\_data; **int** flags; **struct** **rw\_semaphore** **lookup\_sem**; **struct** **kobject** \***slave\_dir**; **struct** **timer\_rand\_state** \***random**; **atomic\_t** sync\_io; */\* RAID \*/* **struct** **disk\_events** \***ev**; **#ifdef CONFIG\_BLK\_DEV\_INTEGRITY** **struct** **kobject** **integrity\_kobj**; **#endif */\* CONFIG\_BLK\_DEV\_INTEGRITY \*/*** **int** node\_id; **struct** **badblocks** \***bb**; **struct** **lockdep\_map** **lockdep\_map**; *// выделение статически* **int** **register\_chrdev\_region**(**dev\_t** first, **unsigned** **int** count, **char** \*name); *//first это - начало диапазона номеров устройств, который вы хотели бы выделить* *//count - запрашиваемое общее число смежных номеров устройств.* *//name - имя уст, которое должно быть связано с этим диап чисел; оно будет отобр в /proc/devices*  *// выделение динамически* **int** **alloc\_chrdev\_region**(**dev\_t** \*dev, **unsigned** **int** firstminor, **unsigned** **int** count, **char** \*name); *// dev является только выходным значением, которое при успешном завершении содержит первый номер выделенного диапазона.*  *// firstminor должен иметь значение первого младшего номера для использования; как правило, 0.* *// Параметры count и name аналогичны register\_chrdev\_region.* *// освободить* **void** **unregister\_chrdev\_region**(**dev\_t** first, **unsigned** **int** count) }; |
| --- |

### struct device

| **struct** **device** { *// ! На самом низком уровне каждое устройство в Linux представлено экземпляром struct device. Это базовая структура.* **struct** **kobject** **kobj**; ... **struct** **device** \***parent**; *// //устройство, к которому подключается новое устройство. Обычно шина или хост-контроллер* **const** **char** \*init\_name; */\* исходное название устройства \*/* **const** **struct** **device\_type** \***type**; **struct** **bus\_type** \***bus**; */\* важно, к какой шине подключается устройство \*/* **struct** **device\_driver** \***driver**; */\* какой драйвер разместил это устройство \*/* **void** \*platform\_data; */\* Platform specific data, device core doesn't touch it \*/* **void** \*driver\_data; */\* Driver data, set and get with \*/* ... **#ifdef CONFIG\_GENERIC\_MSI\_IRQ\_DOMAIN** **struct** **irq\_domain** \***msi\_domain**; **#endif** **#ifdef CONFIG\_PINCTRL** **struct** **dev\_pin\_info** \***pins**; **#endif** **#ifdef CONFIG\_GENERIC\_MSI\_IRQ** **struct** **list\_head** **msi\_list**; **#endif** **const** **struct** **dma\_map\_ops** \***dma\_ops**; u64 \*dma\_mask; */\* dma mask (if dma'able device) \*/* **struct** **dma\_coherent\_mem** \***dma\_mem**; *//direct memory access - это метод освобождения процессора от рутинной перекачки данных от устройства в оперативную память. При передаче блоков данных в оперативку, если нет dma, то все через регистры, и процессор загружен. Coherent DMA mapping API. В этом случае нет необходимости предварительно выделять буфер DMA, установка такого поля используется для устойчивого соединения многократно используемых драйверов.* ... **#ifdef CONFIG\_NUMA** **int** numa\_node; */\* NUMA node this device is close to \*/* **#endif** **dev\_t** devt; */\* для создания в sys/fs девайса \*/* u32 id; */\* экземпляр устройства \*/* ...  }; |
| --- |

### struct pci\_dev

*Устройства, подключаемые к шине pci представляются структурой struct pci\_dev, которая содержит строку struct device dev.*

| */\* The pci\_dev structure describes PCI devices \*/* ***struct******pci\_dev*** *{ /\*! Но ядро сод набор подсис таких как pci, pci express, usb. Больш подсис отслеж доп инфо об устр, поэтому устр обычно представ более детал струк (например, struct pci\_device или struct usb\_device), которые в себя включают саму struct device.!\*/* ***struct******list\_head******bus\_list****; /\* Node in per-bus list \*/* ***struct******pci\_bus*** *\*****bus****; /\* Шина, на которой находится устройство \*/* ***struct******pci\_bus*** *\*****subordinate****; /\* Bus this device bridges to \*/ ...* ***unsigned******short*** *vendor; /\*поставщик\*/* ***unsigned******short*** *device; /\*устройство\*/* ***unsigned******short*** *subsystem\_vendor;* ***unsigned******short*** *subsystem\_device;* ***unsigned******int******class****; /\* 3 bytes: (base,sub,prog-if) \*/ ... u8 pin; /\* Interrupt pin this device uses \*/ ...* ***struct******pci\_driver*** *\*****driver****; /\* Драйвер, связанный с конкретным устройством \*/ ...* ***struct******device\_dma\_parameters******dma\_parms****; ...* ***struct******device******dev****; /\* универсальный интерфейс устройства \*/ ... /\* Вместо непосредственного использования линии прерывания и регистров базового адреса используйте значения, хранящиеся здесь. Они могут быть разными! \*/* ***unsigned******int*** *irq;* ***struct******resource******resource****[****DEVICE\_COUNT\_RESOURCE****]; /\* I/O and memory regions + expansion ROMs \*/ ...* ***pci\_dev\_flags\_t*** *dev\_flags; ...* ***unsigned******long*** *priv\_flags; /\* Private flags for the PCI driver \*/  };* |
| --- |

### struct usb\_device

*Аналогично, struct pci\_dev структура struct usb\_dev содержит struct device dev.*

| */\* USB\_DT\_DEVICE: Device descriptor - вспомогательная структура \*/* ***struct******usb\_device\_descriptor*** *{  \_\_u8 bLength; \_\_u8 bDescriptorType; \_\_le16 bcdUSB; \_\_u8 bDeviceClass; \_\_u8 bDeviceSubClass; \_\_u8 bDeviceProtocol; \_\_u8 bMaxPacketSize0; \_\_le16 idVendor; \_\_le16 idProduct; \_\_le16 bcdDevice; \_\_u8 iManufacturer; \_\_u8 iProduct; \_\_u8 iSerialNumber; \_\_u8 bNumConfigurations;  } \_\_attribute\_\_ ((packed));* ***struct******usb\_device*** *{* ***int*** *devnum;* ***char*** *devpath[16]; u32 route;* ***enum*** *usb\_device\_state state; /\*Cостояние устройства: настроено, не подключено и т. д.\*/* ***enum*** *usb\_device\_speed speed; /\*Скорость устройства: высокая / полная / низкая (или ошибка)\*/ ...* ***struct******usb\_device*** *\*****parent****;* ***struct******usb\_bus*** *\*****bus****; /\*шина, частью которого мы являемся\*/* ***struct******usb\_host\_endpoint******ep0****; /\*данные конечной точки 0 (канал управления по умолчанию)\*/* ***struct******device******dev****; /\* универсальный интерфейс устройства \*/* ***struct******usb\_device\_descriptor******descriptor****;* ***struct******usb\_host\_bos*** *\*****bos****;* ***struct******usb\_host\_config*** *\*****config****;* ***struct******usb\_host\_config*** *\*****actconfig****;* ***struct******usb\_host\_endpoint*** *\*****ep\_in****[16]; /\*массив конечных точек IN\*/* ***struct******usb\_host\_endpoint*** *\*****ep\_out****[16]; /\*массив конечных точек OUT\*/ ... u8 portnum; /\*номер родительского порта (источник 1)\*/ u8 level; /\*количество предков USB-концентраторов\*/ u8 devaddr;* ***unsigned*** *can\_submit:1; /\*URB могут быть представлены\*/ ...* ***int*** *string\_langid; /\*идентификатор языка для строк\*/ /\* static strings from the device \*/* ***char*** *\*product; /\*идентификатор продукта, если есть (статический)\*/* ***char*** *\*manufacturer; /\*Строка i-производителя, если имеется (статическая)\*/* ***char*** *\*serial; /\* серийный номер\*/* ***struct******list\_head******filelist****;* ***int*** *maxchild; /\*количество портов в хабе\*/ u32 quirks;* ***atomic\_t*** *urbnum; /\*количество URB, представленных для всего устройства\*/* ***unsigned******long*** *active\_duration;* ***#ifdef CONFIG\_PM******unsigned******long*** *connect\_time; /\*время, когда устройство было впервые подключено/\* ... unsigned port\_is\_suspended:1; #endif  struct wusb\_dev \*wusb\_dev; int slot\_id; enum usb\_device\_removable removable; ...   }* |
| --- |

### 

### struct device\_driver

| **struct** **device\_driver** { *// Аналогично тому, что struct device представляет универсальный интерфейс устройства struct device\_driver представляет универсальную структуру драйвера.* **const** **char** \*name; **struct** **bus\_type** \***bus**; *//шина, если используем, то выше тип по умолчанию (не знаю, что это значит, но так сказали на лекции)* **struct** **module** \***owner**; **const** **char** \*mod\_name; *//имя модуля (для встраиваемых модулей)* **bool** suppress\_bind\_attrs; */\* disables bind/unbind via sysfs \*/* **enum** probe\_type probe\_type; **const** **struct** **of\_device\_id** \***of\_match\_table**; **const** **struct** **acpi\_device\_id** \***acpi\_match\_table**;  *//точки входа- probe, disconnect, suspend, resume, ...Нет open, close .., так как они в struct file\_operations* **int** (\*probe) (struct device \*dev); */\*Вызывается для запроса существования определенного устройства, может ли этот драйвер работать с ним, и связать драйвер с конкретным устройством\*/* **void** (\*sync\_state)(struct device \*dev); **int** (\*remove) (struct device \*dev); */\*Вызывается, когда устройство удаляется из системы, чтобы отсоединить устройство от этого драйвера.\*/* **void** (\*shutdown) (struct device \*dev); */\*Вызывается во время выключения, чтобы отключить устройство\*/* **int** (\*suspend) (struct device \*dev, **pm\_message\_t** state); */\*Вызывается перевести устройство в спящий режим. Обычно в состоянии низкого энергопотребления\*/* **int** (\*resume) (struct device \*dev); */\* Вызывается, чтобы вывести устройство из спящего режима.\*/* **const** **struct** **attribute\_group** \*\***groups**; **const** **struct** **attribute\_group** \*\***dev\_groups**; **const** **struct** **dev\_pm\_ops** \***pm**; **void** (\*coredump) (struct device \*dev); **struct** **driver\_private** \***p**; */\*!Для драйверов устройств, подключенных к шине PCI определена структура pci\_driver. Для USB драйверов определена структура struct usb\_driver (в ядре Linux объявлена еще одна структура struct usb\_device\_driver)!\*/* }; |
| --- |

### struct pci\_driver

*Для драйверов устройств, подключенных к шине PCI определена структура:*

| **struct** **pci\_driver** { **struct** **list\_head** **node**; **const** **char** \*name; **const** **struct** **pci\_device\_id** \***id\_table**; */\* Должен быть ненулевым, чтобы вызывалась probe \*/* **int** (\*probe)(struct pci\_dev \*dev, **const** struct pci\_device\_id \*id); */\* Новое устройство вставлено \*/* **void** (\*remove)(struct pci\_dev \*dev); */\* Устройство удалено (NULL, если драйвер без поддержки «горячей» замены)\*/* **int** (\*suspend)(struct pci\_dev \*dev, **pm\_message\_t** state); */\* Устройство приостановлено \*/* **int** (\*resume)(struct pci\_dev \*dev); */\* Устройство проснулось \*/* **void** (\*shutdown)(struct pci\_dev \*dev); **int** (\*sriov\_configure)(struct pci\_dev \*dev, **int** num\_vfs); */\* On PF \*/* **const** **struct** **pci\_error\_handlers** \***err\_handler**; **const** **struct** **attribute\_group** \*\***groups**; **struct** **device\_driverdriver**; **struct** **pci\_dynids** **dynids**; }; |
| --- |

### softirq\_action

| */\*! Когда ядро вып обр отлож прер, то фун-я action вызыв c указ на соответ струк softirq\_action в кач арг. softirq опр стат при комп ядра. в наст вр их 10 - 0 - HI\_SOFTIRQ и 1 - TIMER\_SOFTIRQ. Доб новый ур softirq можно только путем перекомп ядра, то есть число не может быть изм динам !\*/*  **struct** **softirq\_action** {  **void** (\*action)(struct softirq\_action \*); }; |
| --- |

### tasklet\_struct

| **struct** **tasklet\_struct**  {  **struct** **tasklet\_struct** \***next**; */\* указатель на следующий тасклет в списке (очередь на планирование)\*/*  **unsigned** **long** state; */\* текущее состояние тасклета, enum (см ниже)\*/*  */\*! Если поле count = 0, то таск разрешен и может выпол, если он помечен как запланир (TASKLET\_STATE\_SCHED), иначе таск запрещен и не может выпол.!\*/*  **atomic\_t** count; */\* счетчик ссылок \*/*  **unsigned** **long** data; */\* аргумент функции-обработчика тасклета \*/*   **bool** use\_call\_back;  **union**  {  **void** (\*func) (**unsigned** **long** data);  **void** (\*callback)(struct tasklet\_struct \*t); } ); *// Поле state может принимать одно из следующих значений:* **enum** {  TASKLET\_STATE\_SCHED, *//запланирован на выполнение*  TASKLET\_STATE\_RUN *//выполняется (только в SMP)* } */\* прототип функции-обработчика тасклета выглядит так\*/* **void** **tasklet\_handler**(**unsigned** **long** data)  struct tasklet {  DECLARE\_TASKLET(name, \_callback) *//поле count = 0 и , он разрешен*  DECLARE\_TASKLET\_DISABLED(name, \_callback) *//count = 1 , то есть он "запрещен".*  **extern** **void** **tasklet\_init**(struct tasklet\_struct \*t, **void** (\*func)(**unsigned** **long**), **unsigned** **long** data); *// Дин. При дин созд таск объявл указ на струк struct tasklet\_struct \*t а затем для инициал вызыв функция*  *// Планирование (внутри команда test\_and\_set)*   tasklet\_shedule(struct tasklet\_struct \*t)  tasklet\_hi\_shedule(struct tasklet\_struct \*t)  */\*! Когда tasklet заплан, ему выстав сост TASKLET\_STATE\_SCHED, и он добав в оч. Tasklet не может наход сразу в неск мес очер на планир. После того, как тасклет был заплан, он выпол тол 1 раз.!\*/*  **void** **tasklet\_disable\_nosync**(struct tasklet\_struct \*t); */\* деактивация. возв упр без ожид заверш выполн тасклета \*/*  **void** **tasklet\_disable**(struct tasklet\_struct \*t); */\* деак с ожид завер раб tasklet \*/*  **void** **tasklet\_enable**(struct tasklet\_struct \*t); */\* акт. Эта функция должна быть вызвана для того, чтобы можно было испол тасклет, созд с помощью макроса DECLARE\_TASKLET\_DISABLED() ,,, сколько деактивирован, столько и активирован\*/*  tasklet\_kill (struct tasklet\_struct \*t) *// ждет завер и удал его из оч в контек проц*  tasklet\_trylock(struct tasklet\_struct \*t) *// функ блок*  } |
| --- |

### workqueue\_struct

| *// Передает "выданную" (issued) work\_struct подходящему worker\_poll через своих pool\_workqueues (это мой перевод документации, исключительно для собственного понимания)* **struct** **workqueue\_struct** { **struct** **list\_head** **pwqs**; *// все посредники (pwqs)* **struct** **list\_head** **list**; *//все очереди работ (workqueue). Но на конкретное CPU есть свой список rcu\_head (см последнее поле), чтобы не перебираться все очереди* **struct** **mutex** **mutex**; ... **char** name[WQ\_NAME\_LEN]; */\* Уничтожение workqueue\_struct защищено RU, чтобы разрешить просмотр списка рабочих очередей без захвата wq\_pool\_mutex. Это используется для сброса всех рабочих очередей из sysrq. \*/* **struct** **rcu\_head** **rcu**; ... **int** **alloc\_workqueue**(**char** \*name, **unsigned** **int** flags, **int** max\_active); *// имя, огран число задач из некот очереди, которые могут выпол на одном кпу, флаги, опред как оч будет выполня* destroy\_workqueue(struct workqueue\_struct \* wq); *// уничтожить* } |
| --- |

### work\_struct

| **struct** **work\_struct** {  */\*! Структура work\_struct представляет задачу (обработчик нижней половины) в workqueue (очереди работ).!\*/*  *// создание во время компиляции*  DECLARE\_WORK(name, **void** (\*func)(**void** \*));  DECLARE\_DELAYED\_WORK(name, **void** (\*func)(**void** \*))  *// динамически*  INIT\_WORK(sruct work\_struct \*work, **void** (\*func)(**void**), **void** \*data);  ..**atomic\_long\_t** data;  **struct** **list\_head** **entry**;  **work\_func\_t** func;  **#ifdef CONFIG\_LOCKDEP**  **struct** **lockdep\_map** **lockdep\_map**;  **# endif..**  *// добавление в очер*  queue\_work(struct workqueue\_struct \*wq, struct work\_struct \*work) *// тек проц*  queue\_work\_on(**int** cpu, struct workqueue\_struct \*wq, struct work\_struct \*work); *// конкр проц*  *// для отлож работ*  queue\_delayed\_work(struct workqueue\_struct \*wq, struct delayed\_work \*dwork, **unsigned** **long** delay)   queue\_delayed\_work\_on(**int** cpu, struct workqueue\_struct \*wq, struct delayed\_work \*dwork, **unsigned** **long** delay)  *// глоб оч работ с 2 ф-ями, не нужно опр струк оч работ, в остальном похожи с пред ф-ями*  schedule\_work(struct work\_struct \*work)  schedule\_work\_on(**int** cpu, struct work\_struct \*work)   *//принудительно завершить конкретный элемент work и блокировать прочую обработку прежде, чем работа будет закончена*  **int** **flush\_work**( struct work\_struct \*work );  *//принудительно завершить все работы в данной очереди*  **int** **flush\_workqueue**( struct workqueue\_struct \*wq );  *//принудительно завершить глобальную очередь работ ядра*  **void** **flush\_scheduled\_work**(**void**);  }; |
| --- |

### Флаги

| **enum** {  WQ\_UNBOUND =1<<1,*/\*привязанные или нет к cpu\*/*  WQ\_FREEZABLE =1<<2,*/\*работа будет заморожена, когда система будет приостановлена.\*/*  WQ\_MEM\_RECLAIM =1<<3,*/\*may be used for memory reclaim\*/*  WQ\_HIGHPRI =1<<4,*/\*high priority\*/*  WQ\_CPU\_INTENSIVE =1<<5,*/\*cpu intensive workqueue\*/*  WQ\_SYSFS =1<<6,*/\*visible in sysfs/   WQ\_POWER\_EFFICIENT =1<<7   WQ\_MAX\_ACTIVE =512 ... }* |
| --- |

### Передача данных из ядра в пользователя

| **unsigned** **long** \_\_copy\_to\_user(**void** \_\_user \*to, **const** **void** \*from, **unsigned** **long** n); **unsigned** **long** \_\_copy\_from\_user(**void** \*to, **const** **void** \_\_user \*from, **unsigned** **long** n); |
| --- |

| */\*\*  \* request\_irq - Добавление обработчика для строки прерывания  \* @irq: Линия прерывания для выделения  \* @handler:Функция, которая будет вызываться при возникновении IRQ. Основной обработчик для потоковых прерываний, если значение равно NULL, устанавливается основной обработчик по умолчанию  \* @flags: Обработка флагов  \* @name: Имя устройства, генерирующего это прерывание  \* @dev: Файл cookie, переданный функции-обработчику  \* Этот вызов выделяет прерывание и устанавливает обработчик;  \*/* ***static******inline******int*** *\_\_must\_check* ***request\_irq****(****unsigned******int*** *irq,* ***irq\_handler\_t*** *handler,* ***unsigned******long*** *flags,* ***const******char*** *\*name,* ***void*** *\*dev) {* ***return*** *request\_threaded\_irq(irq, handler, NULL, flags, name, dev); }* ***int*** *ret = request\_irq(IRQ, /\* номер irq \*/  handler, /\* наш обработчик \*/  IRQF\_SHARED, /\* линия может быть разделена, IRQ(разрешено совместное использование)\*/  "my\_irq2\_handler", /\* имя устройства (можно потом посмотреть в /proc/interrupts)\*/  &devID); /\* Последний параметр (идентификатор устройства) irq\_handler нужен для того, чтобы можно отключить с помощью free\_irq \*/* ***#define IRQ\_NONE (0) (если не удалось обработать),******#define IRQ\_HANDLED (1) (если прерывание обработано)******#define IRQ\_RETVAL (x) ((x)!=0)*** |
| --- |