Linux操作系统编程

Linux文件系统概述

VFS-目录项对象（dentry）

- 每个文件除了有一个索引节点inode数据结构外，还有一个目录项dentry数据结构。
- dentry结构代表的是逻辑意义上的文件，描述的是文件逻辑上的属性，目录项对象在磁盘上并没有对应的映像
- inode结构代表的是物理意义上的文件，记录的是物理上的属性，对于一个具体的文件系统，其inode结构在磁盘上就有对应的映像
- 一个索引节点对象可能对应多个目录项对象

VFS-目录项对象（dentry）

```
struct dentry {
  atomic_t d_count;          /* 目录项引用计数器 */
  unsigned int d_flags;      /* 目录项标志 */
  struct inode  * d_inode;    /* 与文件名关联的索引节点 */
  struct dentry * d_parent;       /* 父目录的目录项 */
  struct list_head d_hash;        /* 目录项形成的哈希表 */
  struct list_head d_lru;         /*未使用的 LRU 链表 */
  struct list_head d_child;       /*父目录的子目录项所形成的链表 */
  struct list_head d_subdirs;     /* 该目录项的子目录所形成的链表*/
  struct list_head d_alias;       /* 索引节点别名的链表*/
  int d_mounted;                  /* 目录项的安装点 */
  struct qstr d_name;             /* 目录项名（可快速查找） */
  struct dentry_operations  *d_op;/* 操作目录项的函数*/
  struct super_block * d_sb;     /* 目录项树的根（即文件的超级块）*/
  unsigned long d_vfs_flags;
  void * d_fsdata;                /* 具体文件系统的数据 */
unsigned char d_iname[DNAME_INLINE_LEN]; /* 短文件名 */
.....
};
```

VFS-文件对象（file）

- 进程是通过文件描述符来访问文件的
- Linux中专门用了一个file文件对象来保存打开文件的文件位置，这个对象称为打开的文件描述（open file description）
- 文件描述符是用来描述打开的文件的。每个进程用一个files_struct结构来记录文件描述符的使用情况，这个files_struct结构称为用户打开文件表，它是进程的私有数据
- file结构中主要保存了文件位置，此外，还把指向该文件索引节点的指针也放在其中。file结构形成一个双链表，称为系统打开文件表。

VFS-文件对象（file）

```
struct file {
        struct list_head      f_list; /*所有的打开的文件形成的链表*/
        struct dentry         *f_dentry; /*与该文件相关的dentry*/
        struct vfsmount     *f_vfsmnt; /*该文件在这个文件系统中的安装点*/
        struct file_operations  *f_op; /*文件操作*/
        atomic_t              f_count; /*引用计数*/
        unsigned int            f_flags; /*打开文件时候指定的标识*/
        mode_t                f_mode; /*文件的访问模式*/
        loff_t              f_pos; /*目前文件的相对开头的偏移*/
        unsigned long           f_reada, f_ramax, f_raend, f_ralen, f_rawin;
        /*预读标志、要预读的最多页面数、上次预读后的文件*/
        /*指针、预读的字节数以及预读的页面数*/
        struct fown_struct      f_owner; /*进程ID,信号*/
....
}
```

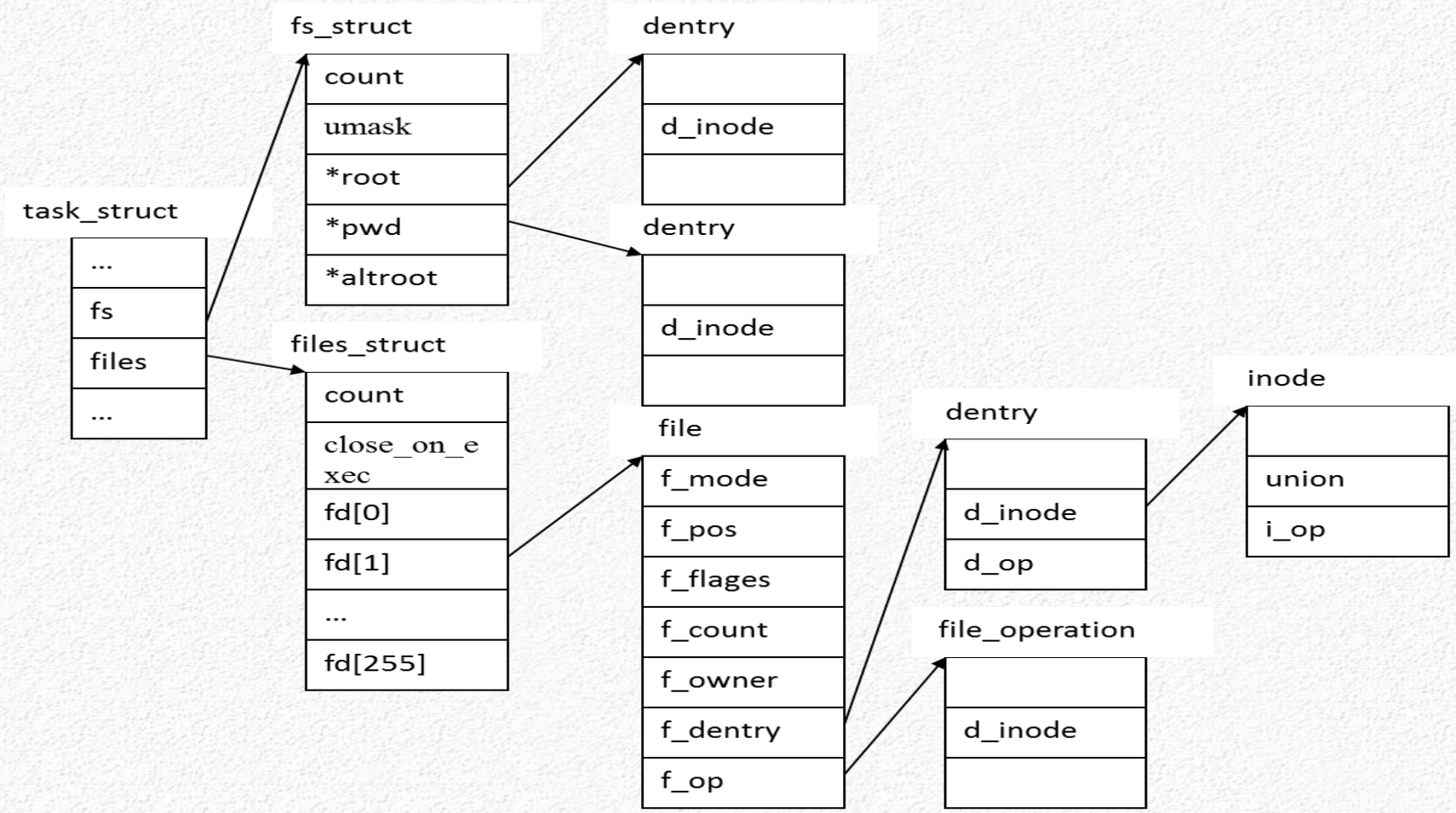
VFS-文件对象（file）

```
struct files_struct {
        atomic_t count; /* 引用计数*/
        rwlock_t file_lock;   /*锁，保护下面的字段 */
        int max_fds; /* 当前文件对象的最大的数量*/
        int max_fdset; /* 文件描述符最大数*/
        int next_fd; /* 已分配的最大的文件描述符+1*/
        struct file ** fd;      /*指向文件对象指针数组的指针 */
        fd_set *close_on_exec;/*执行exec()时候需要关闭的文件描述符 */
        fd_set *open_fds;/* 指向打开的文件描述符的指针*/
        fd_set close_on_exec_init;/*执行exec()时候需要关闭的文件描述符初始化值*/
        fd_set open_fds_init;/* 文件描述符初值集合*/
        struct file * fd_array[NR_OPEN_DEFAULT];/*文件对象指针的初始化数组*/
;
```

VFS数据结构之间的关系

- 超级块是对一个文件系统的描述
- 索引节点是对一个文件物理属性的描述
- 目录项是对一个文件逻辑属性的描述
- 一个进程所处的位置是由fs_struct来描述的，而一个进程（或用户）打开的文件是由files_struct来描述的，而整个系统所打开的文件是由file结构来描述

VFS数据结构之间的关系



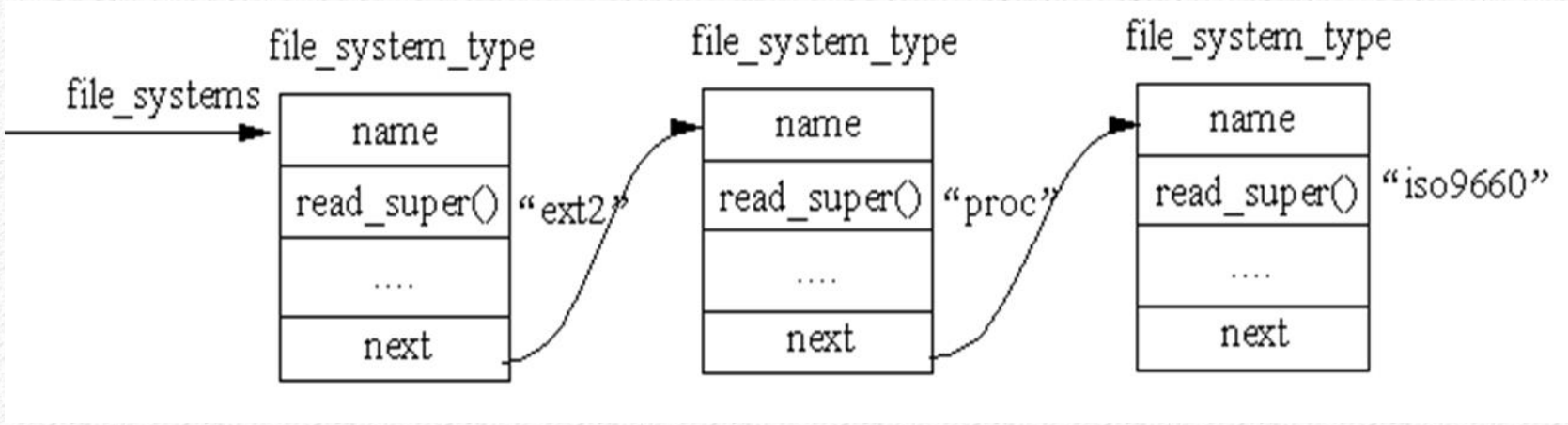
文件系统的注册和注销

●当内核被编译时，就已经确定了可以支持哪些文件系统，这些文件系统在系统引导时，在 VFS 中进行注册。

●VFS的初始化函数用来向VFS注册，即填写文件注册表file_system_type数据结构

●注册调用register_filesystem（）函数

●注销即删除一个file_system_type 结构，需调用 unregister_filesystem()函数



文件系统的注册和注销

```
struct file_system_type {
    const char *name;  /*文件系统的类型名*/
    int fs_flags;      /*文件系统的一些特性*/
    struct super_block *(*read_super)
            (struct super_block *, void *, int);
             /*文件系统读入其超级块的函数指针*/
    struct module *owner;/*确定是否把文件系统作为模块来安装*/
    struct file_system_type * next;
      };
```

文件系统的安装

- 安装一个文件系统实际上是安装一个物理设备
- 自己（一般是超级用户）安装文件系统时，需要指定三种信息：文件系统的名称、包含文件系统的物理块设备、文件系统在已有文件系统中的安装点。
- $ mount -t iso9660 /dev/hdc /mnt/cdrom 其中，iso9660是光驱文件系统的名称，/dev/hdc是包含文件系统的物理块设备，/mnt/cdrom就是将要安装到的目录，即安装点。
- 在用户程序中要安装一个文件系统则可以调用mount（）系统调用。安装过程主要工作是创建安装点对象，将其挂接到根文件系统的指定安装点下，然后初始化超级块对象，从而获得文件系统基本信息和相关的操作。

文件系统的卸载

- 如果文件系统中的文件当前正在使用，该文件系统是不能被卸载的
- 否则，查看对应的 VFS 超级块，如果该文件系统的 VFS 超级块标志为“脏”，则必须将超级块信息写回磁盘
- 之后，对应的 VFS 超级块被释放，vfsmount 数据结构将从vfsmntlist 链表中断开并被释放
- 具体的实现代码为fs/super.c中的sys_umount（）函数