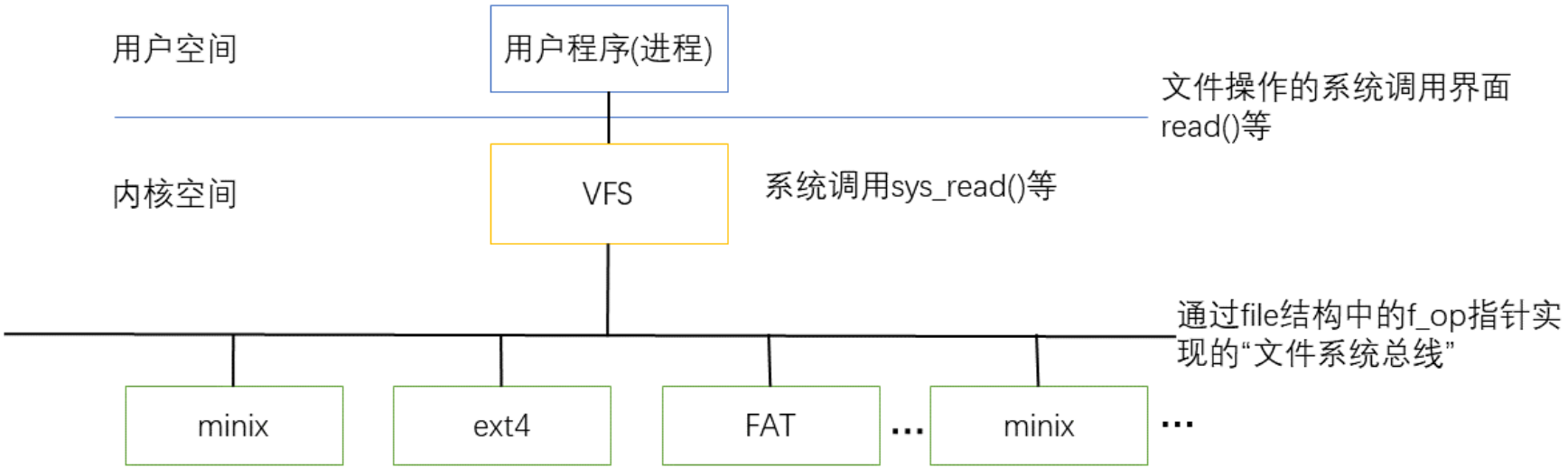
8.2 VFS中的主要数据结构

西安邮电大学

虚拟文件系统中对象的引入

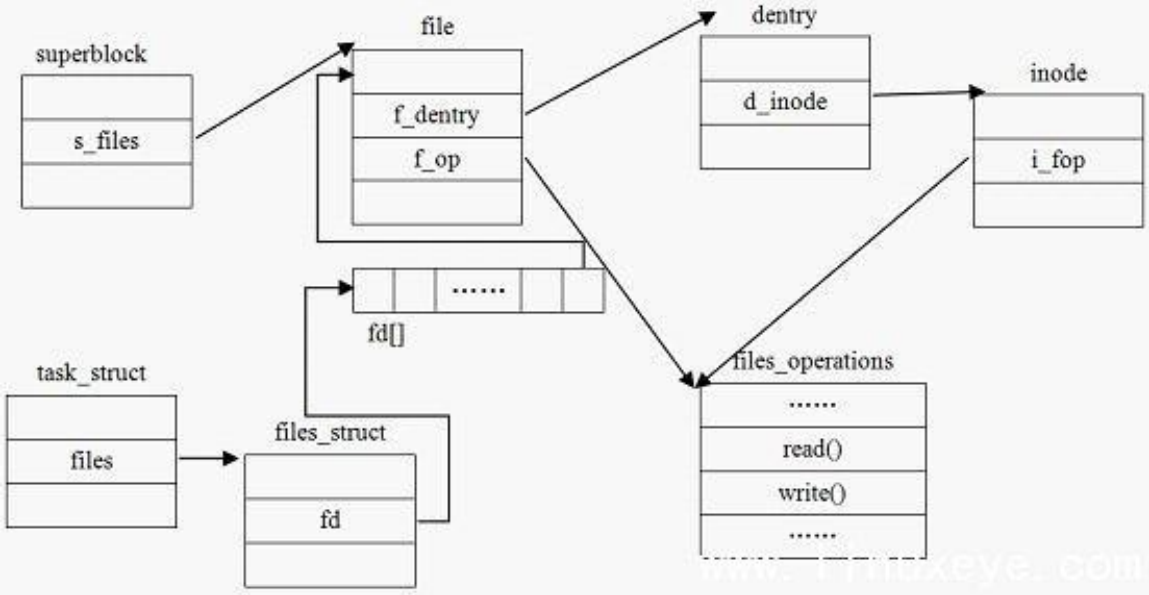


VFS中对象的引入

虚拟文件系统（VFS）的第一个词是“虚拟”，这就意味着，这样的文件系统在磁盘（或其他存储介质上）并没有对应的存储信息。那么，这样一个虚无的文件系统到底怎样形成？尽管Linux支持多达几十种文件系统，但这些真实的文件系统并不是一下子都挂在系统中的，他们实际上是按需被挂载的。老子说：“有无相生”，这个“虚”的VFS的信息都来源于“实”的文件系统，所以VFS必须承载各种文件系统的共有属性。另外，这些实的文件系统只有安装到系统中，VFS才予以认可，也就是说，VFS只管理挂载到系统中的实际文件系统。

既然VFS承担管家的角色，那么我们分析一下它到底要管哪些对象。Linux在文件系统的设计中，全然汲取了Unix的设计思想。Unix在文件系统的设计中抽象出四个概念：文件，目录项，索引节点和超级块。

VFS中共性对象的抽象



超级块（superblock）对象：存放系统中已安装文件系统的有关信息

文件(file)对象：存放打开文件与进程之间进行交互的有关信息

索引节点（inode）对象：存放关于具体文件的一般信息

目录项（dentry）对象：文件目录树中一个路径的组成部分，不管是目录还是普通的文件，都是一个目录项对象。

超级块对象-描述文件系统的属性

```
struct super_block { //超级块数据结构
        struct list_head s_list;                    /*指向超级块链表的指针*/
        ……
        struct file_system_type  *s_type;           /*文件系统类型*/
       struct super_operations  *s_op;             /*超级块方法*/
        ……
        struct list_head         s_instances;       /*该类型文件系统*/
        ……
};

struct super_operations { //超级块方法
        ……
        //该函数在给定的超级块下创建并初始化一个新的索引节点对象
        struct inode *(*alloc_inode)(struct super_block *sb);
       ……
        //该函数从磁盘上读取索引节点，并动态填充内存中对应的索引节点对象的剩余部分
        void (*read_inode) (struct inode *);
       ……
};
```

超级块用来描述整个文件系统的信息。每个具体的文件系统都有各自的超级块

VFS超级块是各种具体文件系统在安装时建立的，并在卸载时被自动删除，其数据结构是 super block

所有超级块对象以双向环形链表的形式链接在一起

与超级块关联的方法就是超级块操作表。这些操作是由数据结构super_operations来描述

索引节点对象-描述文件属性

```
struct inode {//索引节点结构
      ……
       struct inode_operations  *i_op;      /*索引节点操作表*/
      struct file_operations    *i_fop;     /*该索引节点对应文件的文件操作集*/
      struct super_block        *i_sb;      /*相关的超级块*/
      ……
};

struct inode_operations { //索引节点方法
      ……
      //该函数为dentry对象所对应的文件创建一个新的索引节点，主要是由open()系统调用来调用
      int (*create) (struct inode *,struct dentry *,int, struct nameidata *);

      //在特定目录中寻找dentry对象所对应的索引节点
      struct dentry * (*lookup) (struct inode *,struct dentry *, struct nameidata *);
      ……
};
```

索引节点对象存储了文件的相关信息，代表了存储设备上的一个实际的物理文件。当一个 文件首次被访问时，内核会在内存中组装相应的索引节点对象，以便向内核提供对一个文件进行操 作时所必需的全部信息；这些信息一部分存储在磁盘特定位置，另外一部分是在加载时动态填充的。

目录项对象-描述文件的路径

```
struct dentry {//目录项结构
    ……
     struct inode *d_inode;           /*相关的索引节点*/
    struct dentry *d_parent;         /*父目录的目录项对象*/
    struct qstr d_name;              /*目录项的名字*/
    ……
     struct list_head d_subdirs;     /*子目录*/
    ……
     struct dentry_operations *d_op; /*目录项操作表*/
    struct super_block *d_sb;        /*文件超级块*/
    ……
};

struct dentry_operations {
    //判断目录项是否有效;
    int (*d_revalidate)(struct dentry *, struct nameidata *);
    //为目录项生成散列值;
    int (*d_hash) (struct dentry *, struct qstr *);
    ……
};
```

引入目录项的概念主要是出于方便查找文件的目的。一个路径的各个组成部分，不管是目录还是普通的文件，都是一个目录项对象。如，在路径/home/source/test.c中，目录 /，home，source和文件test.c都对应一个目录项对象。不同于前面的两个对象，目录项对象没有对应的磁盘数据结构，VFS在遍历路径名的过程中现场将它们逐个地解析成目录项对象。

文件对象-描述进程打开的文件

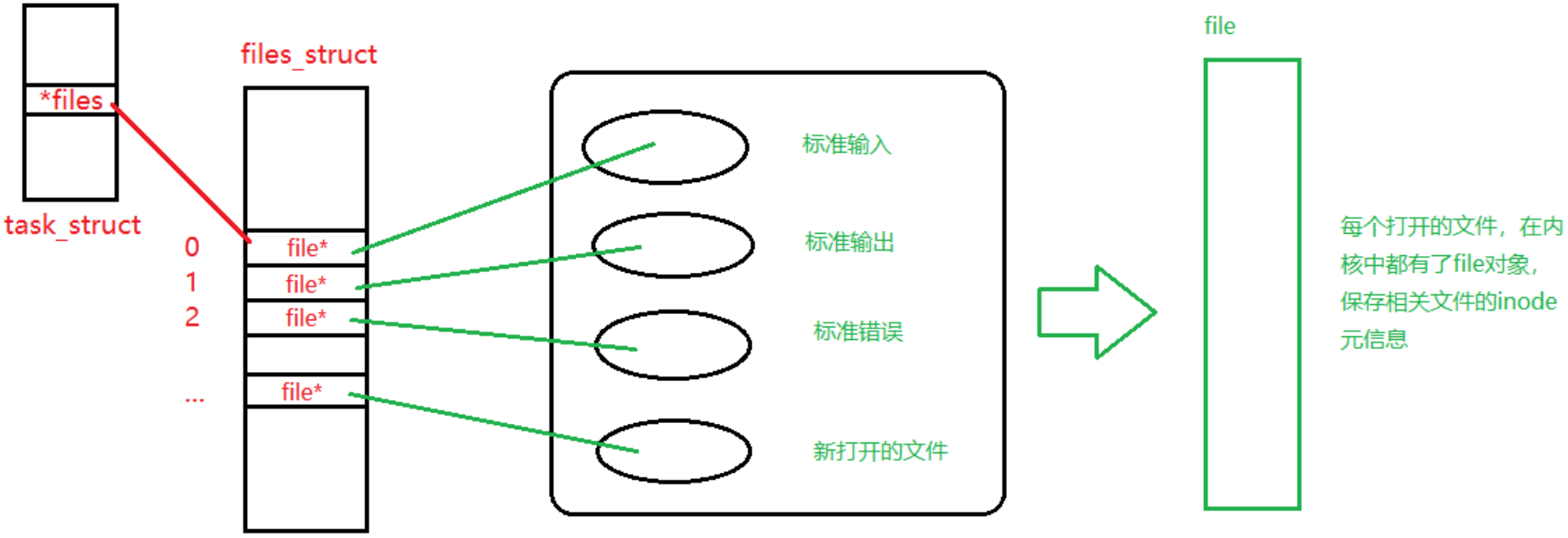
```
struct file {
    ……
     struct list_head        f_list;        /*文件对象链表*/
    struct dentry           *f_dentry;      /*相关目录项对象*/
    struct vfsmount         *f_vfsmnt;      /*相关的安装文件系统*/
    struct file_operations *f_op;           /*文件操作表*/
    ……
};

struct file_operations {
    ……
    //文件读操作
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ……
    //文件写操作
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ……
    int (*readdir) (struct file *, void *, filldir_t);
    ……
    //文件打开操作
    int (*open) (struct inode *, struct file *);
    ……
};
```

文件对象

文件对象是已打开的文件在内存中的表示，主要用于建立进程和磁盘上的文件的对应关系。它由sys_open()现场创建，由sys_close()销毁。文件对象和物理文件的关系有点像进程和程序的关系一样。当我们站在用户空间来看待VFS，我们只需与文件对象打交道，而无须关心超级块，索引节点或目录项。因为多个进程可以同时打开和操作同一个文件，所以同一个文件也可能存在多个对应的文件对象。文件对象仅仅在进程观点上代表已经打开的文件，它反过来指向目录项对象。一个文件对应的文件对象可能不是惟一的，但是其对应的索引节点和目录项对象无疑是惟一的。

与进程相关的文件结构 —用户打开文件表

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结构称为进程打开文件表，它是进程的私有数据。再次强调，每个打开的文件，在内核 中都有file对象。

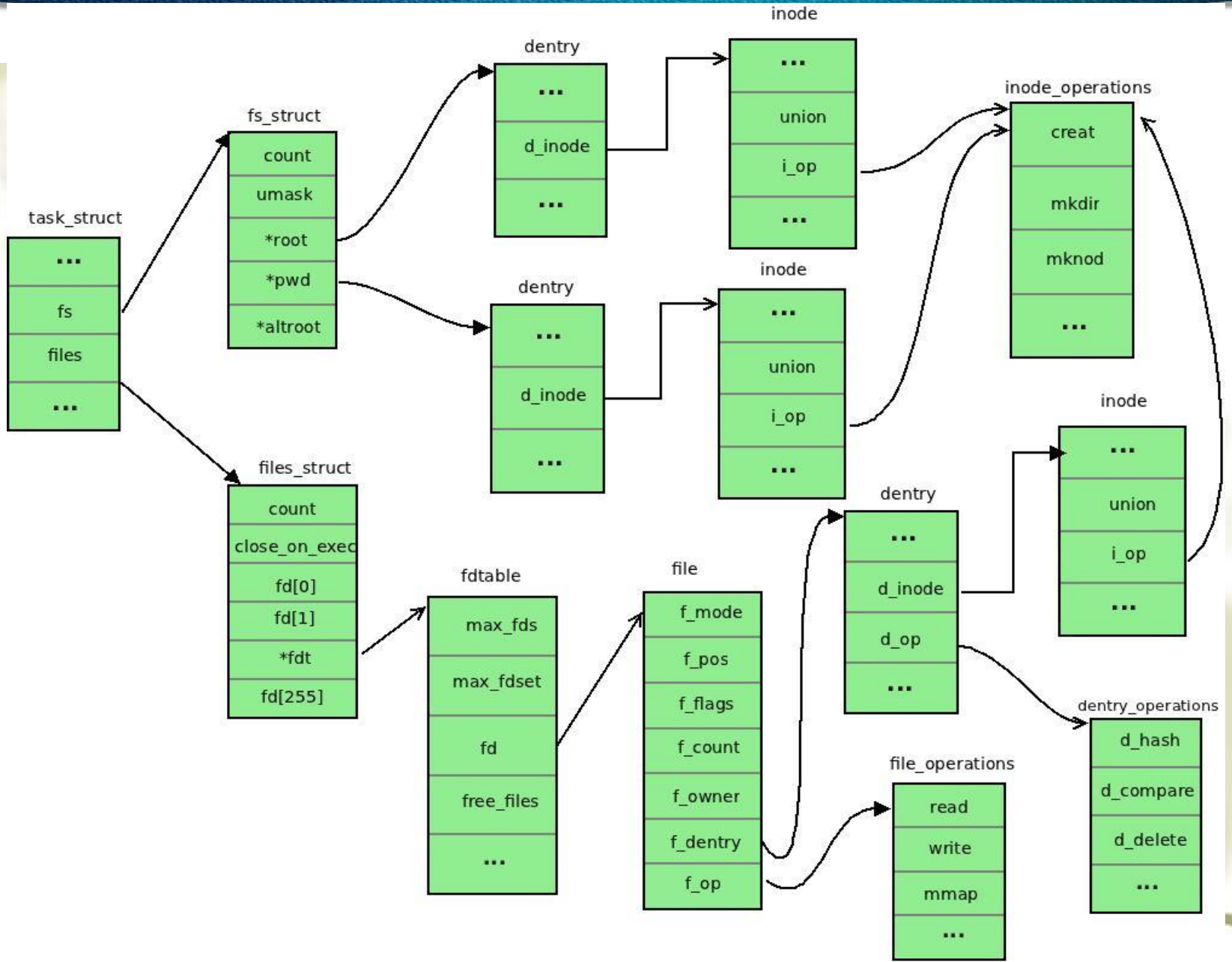


与进程相关的文件结构

```
struct files_struct {//打开的文件集
        atomic_t count;              /*结构的使用计数*/
       ……
        int max_fds;                 /*文件对象数的上限*/
       int max_fdset;               /*文件描述符的上限*/
       int next_fd;                 /*下一个文件描述符*/
       struct file ** fd;           /*全部文件对象数组*/
       ……
 };

struct fs_struct {//建立进程与文件系统的关系
        atomic_t count;              /*结构的使用计数*/
       rwlock_t lock;               /*保护该结构体的锁*/
       int umask;                   /*默认的文件访问权限*/
       struct dentry * root;        /*根目录的目录项对象*/
       struct dentry * pwd;         /*当前工作目录的目录项对象*/
       struct dentry * altroot;     /*可供选择的根目录的目录项对象*/
       struct vfsmount * rootmnt;   /*根目录的安装点对象*/
       struct vfsmount * pwdmnt;    /*pwd的安装点对象*/
       struct vfsmount * altrootmnt;/*可供选择的根目录的安装点对象*/
};
```

file_struct结构用来记录进程打开的文件表，而fs_sturct结构描述进程与文件系统之间的关系。

数据结构关系图


task_struct
...
fs
files
...
fs_struct
count
umask
*root
*pwd
*altroot
dentry
...
d_inode
...
dentry
...
d_inode
...
inode
...
union
i_op
...
inode
...
union
i_op
...
inode_operations
creat
mkdir
mknod
...
files_struct
count
close_on_exec
fd[0]
fd[1]
*fdt
fd[255]
fdtable
max_fds
max_fdset
fd
free_files
...
file
f_mode
f_pos
f_flags
f_count
f_owner
f_dentry
f_op
dentry
...
d_inode
d_op
...
inode
...
union
i_op
...
dentry_operations
d_hash
d_compare
d_delete
...
file_operations
read
write
mmap
...

数据结构间关系图

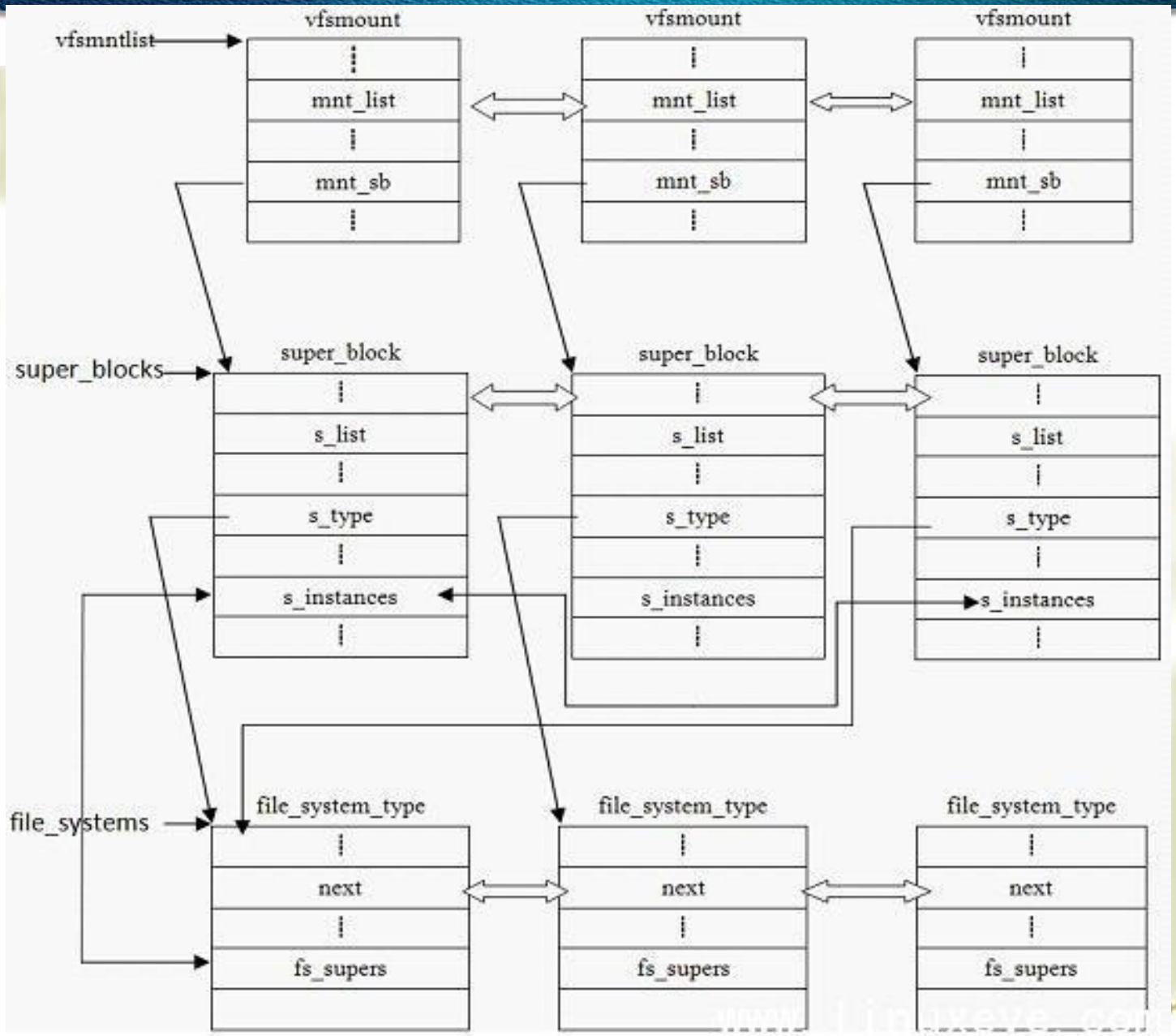
进程通过task_struct中的files域来了解它当前所打开的文件对象；通过fs域了解进程所在的文件系统。文件对象通过域f_dentry找到它对应的目录项对象，再由目录项对象的d_inode域找到它对应的索引结点，这样就建立了文件对象与实际的物理文件的关联。最后，还有一点很重要的是，索引节点对象，目录项对象以及文件对象所对应的操作函数列表是通过相应的操作域得到的。

与文件系统相关的数据结构

```
struct vfsmount
{
        struct list_head mnt_hash;              /*散列表*/
        struct vfsmount *mnt_parent;            /*父文件系统*/
        struct dentry *mnt_mountpoint;          /*安装点的目录项对象*/
        struct dentry *mnt_root;                /*该文件系统的根目录项对象*/
        struct super_block *mnt_sb;             /*该文件系统的超级块*/
        struct list_head mnt_mounts;            /*子文件系统链表*/
        struct list_head mnt_child;             /*子文件系统链表*/
        atomic_t mnt_count;                     /*使用计数*/
        int mnt_flags;                          /*安装标志*/
        char *mnt_devname;                      /*设备文件名*/
        struct list_head mnt_list;              /*描述符链表*/
        struct list_head mnt_fslink;            /*具体文件系统的到期列表*/
        struct namespace *mnt_namespace;        /*相关的名字空间*/
};
```

上一讲介绍了与文件系统相关结构：文件系统类型（file_system_type）和超级块（supe_block），这里给出安装点数据结构VFSmount。

超级块、安装点和具体文件系统的关系



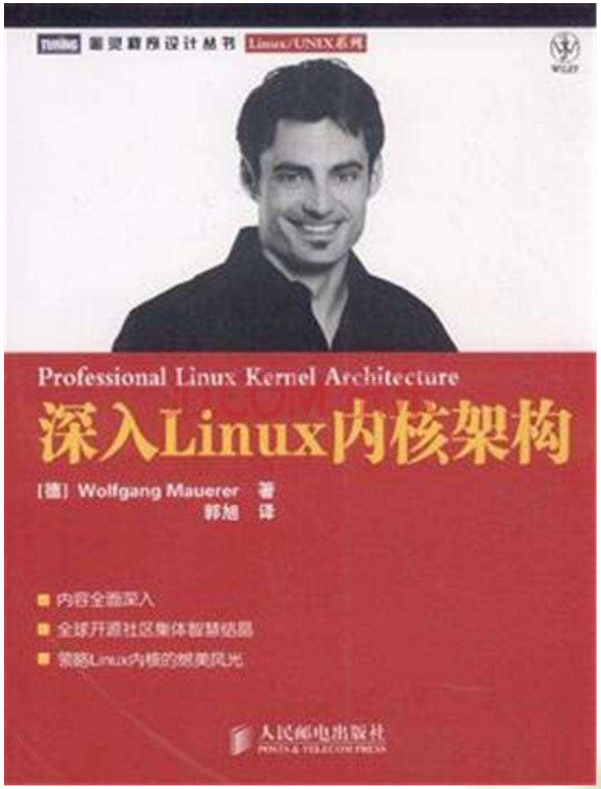
超级块、安装点和具体文件系统的关系

被Linux支持的文件系统，即使都它有零个或多个实例被安装到系统中，但有且仅有一个file_system_type结构。每安装一个文件系统，就对应有一个超级块和安装点。超级块通过它的一个域s_type指向其对应的具体的文件系统类型。具体的文件系统通过file_system_type中的一个域fs_supers，把具有同一种文件类型的超级块链接起来。同一种文件系统类型的超级块通过域s_instances链接。

小结

本讲介绍了文件系统的四种对象，每个对象都对应有两个数据结构，对象的属性和其操作的方法，这是面向对象思想在文件系统设计中的一种体现。与进程相关的结构，除file结构外，还介绍了用户打开文件表和fs结构。文件系统相关的结构，除超级块外，还介绍了与安装点和文件系统类型。这些结构之间都有密切的关系，搞清楚它们之间的关系，对于理解文件系统并阅读源代码有很大的帮助。

参考资料



深入Linux内核架构 第八章

http://www.linuxeye.com/Linux/915.html

带着疑问上路

给定一个文件名，通过文件系统的数据结构关系图，如何查找到相关的文件，请初步阅读open源代码。

谢谢大家！

THANK YOU