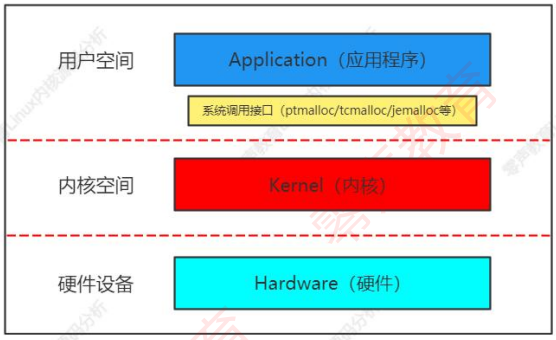
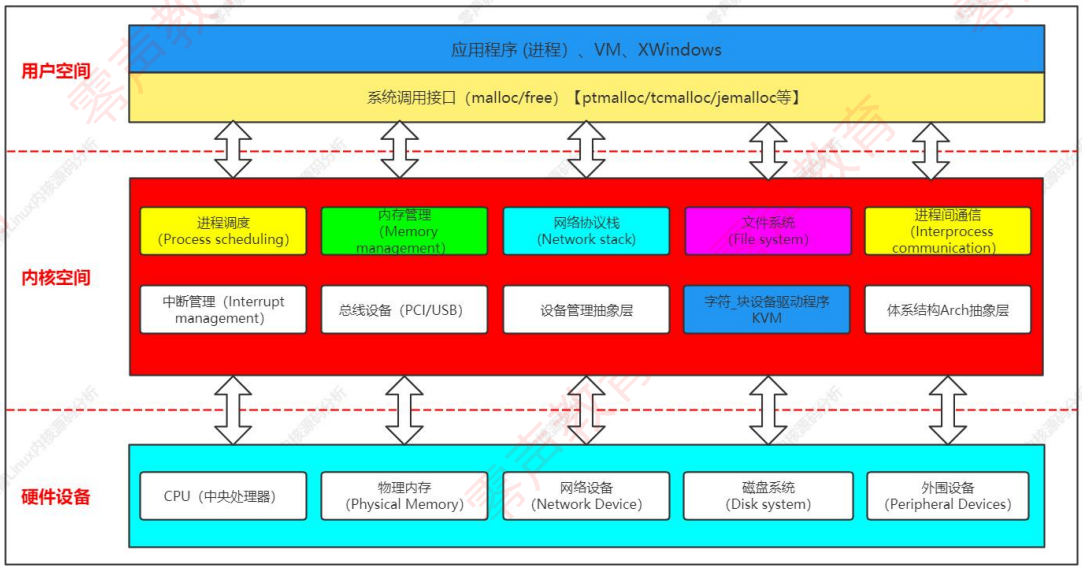
第 004 讲 Linux 内核《文件系统专题》

Linux 内核源码分析架构图





一、虚拟文件系统(VFS)实现原理

1、文件系统概述

在 Linux 操作系统当中，一切皆是文件，除了通常所说狭义的文件（文本文件和二进制文件）以外，目录、套接字、设备及管道等都是文件。

- 在存储设备缓存文件的方法，包括数据结构和访问方法。
- 按照某种文件系统类型格式化的一块存储介质。
- 内核中负责管理和存储文件的模块，即文件系统模块。

2、虚拟文件系统数据结构

虽然不同文件系统类型的物理结构不同，但是虚拟文件系统定义一套统一的数据结构。超级块、索引节点、目录项。

a、超级块

文件系统的第一块是超级块，用来描述文件系统的总体信息。当我们把文件系统挂载到内存中目录树的一个目录下时，就会读取文件系统的超级块，在内存中创建超级块的副本，结构体内核源码，主要成员如下：

include > linux > C fs.h

```
1384 struct super_block {
1385     struct list_head    s_list;     /* Keep this first */
1386     dev_t           s_dev;      /* search index; _not_ kdev_t */
1387     unsigned char       s_blocksize_bits;
1388     unsigned long       s_blocksize;
1389     loff_t          s_maxbytes; /* Max file size */
1390     struct file_system_type *s_type;
1391     const struct super_operations   *s_op;
1392     const struct dquot_operations   *dq_op;
1393     const struct quotactl_ops   *s_qcop;
```

超级块操作集合的数据结构是，结构体 super_operations，主要成员如下：

include > linux > C fs.h

```
1899 struct super_operations {
1900     struct inode *(*alloc_inode)(struct super_block *sb);
1901     void (*destroy_inode)(struct inode *);
1902
1903     void (*dirty_inode) (struct inode *, int flags);
1904     int (*write_inode) (struct inode *, struct writeback_control *wbc);
1905     int (*drop_inode) (struct inode *);
1906     void (*evict_inode) (struct inode *);
1907     void (*put_super) (struct super_block *);
1908     int (*sync_fs)(struct super_block *sb, int wait);
```

b、挂载描述符

一个文件系统，只有挂载到内存中目录树的一个目录下，进程才能访问这个文件系统。每次挂载文件系统，虚拟文件系统就会创建一个挂载描述符：mount 结构体。挂载描述符用来描述文件系统的一个挂载实例，同一个存储设备上的文件系统可以多次挂载，每次挂载到不同的目录下。

结构体 mount 挂载描述符的主要成员如下

fs > C mount.h

```
33
34 struct mount {
35     struct hlist_node mnt_hash;
36     struct mount *mnt_parent;
37     struct dentry *mnt_mountpoint;
38     struct vfsmount mnt;
39     union {
40         struct rcu_head mnt_rcu;
41         struct llist_node mnt_llist;
42     };
43 #ifdef CONFIG_SMP
44     struct mnt_pcp __percpu *mnt_pcp;
45 #else
46     int mnt_count;
47     int mnt_writers;
48 #endif
```

include > linux > C mount.h

```
66
67  struct vfsmount {
68      struct dentry *mnt_root;    /* root of the mounted tree */
69      struct super_block *mnt_sb; /* pointer to superblock */
70      int mnt_flags;
71  } __randomize_layout;
72
```

fs > C mount.h > mountpoint

```
26
27  struct mountpoint {
28      struct hlist_node m_hash;
29      struct dentry *m_dentry;
30      struct hlist_head m_list;
31      int m_count;
32  };
33
```

c、文件系统类型

因为每种文件系统类型的超级块的格式不同，所以每种文件系统需要向虚拟文件系统注册文件系统类型 file_system_type，并且实现 mount 方法用来读取和解析超级块。内核源码如下：

include > linux > C fs.h > file_system_type

```
2166
2167  struct file_system_type {
2168      const char *name;
2169      int fs_flags;
2170  #define FS_REQUIRES_DEV     1
2171  #define FS_BINARY_MOUNTDATA 2
2172  #define FS_HAS_SUBTYPE      4
2173  #define FS_USERNS_MOUNT     8   /* Can be mounted by userns root */
2174  #define FS_RENAME_DOES_D_MOVE   32768   /* FS will handle d_move() during rename() */
2175      struct dentry *(*mount) (struct file_system_type *, int,
2176                  const char *, void *);
```

d、索引节点

在文件系统中，每个文件对应一个索引节点，索引节点描述两类信息。文件的属性，也称为元数据（metadata）。文件数据的存储位置，每个索引节点有一个唯一的编号。当内核访问存储设备上的一个文件时，会在内存中创建索引节点的一个副本，内核结构体 inode 源码如下：

include > linux > C fs.h > inode

```
597
598 /*
599  * Keep mostly read-only and often accessed (especially for
600  * the RCU path lookup and 'stat' data) fields at the beginning
601  * of the 'struct inode'
602  */
603 struct inode {
604     umode_t         i_mode;
605     unsigned short      i_opflags;
606     kuid_t          i_uid;
607     kgid_t          i_gid;
608     unsigned int        i_flags;
609
610 #ifdef CONFIG_FS_POSIX_ACL
611     struct posix_acl    *i_acl;
612     struct posix_acl    *i_default_acl;
613 #endif
```

e、目录项

文件系统把目录当作文件，这种文件的数据是由目录项结构组成的，每个目录项存储一个子目录或文件的名称以及对应的索引节点号。当内核访问存储设备上的一个目录项时，会在内核中创建目录项的一个副本，结构体 dentry 主要成员如下：

include > linux > C dcache.h

```
89
90  struct dentry {
91      /* RCU lookup touched fields */
92      unsigned int d_flags;       /* protected by d_lock */
93      seqcount_t d_seq;       /* per dentry seqlock */
94      struct hlist_bl_node d_hash;    /* lookup hash list */
95      struct dentry *d_parent;    /* parent directory */
96      struct qstr d_name;
97      struct inode *d_inode;      /* Where the name belongs to - NULL is
98                                   * negative */
99      unsigned char d_iname[DNAME_INLINE_LEN];    /* small names */
100
```

f、文件打开实例及打开文件表

当进程打开一个文件的时候，虚拟文件系统就会创建一个打开实例：file 结构体。

include > linux > C fs.h > ...

```
902
903 struct file {
904     union {
905         struct llist_node   fu_llist;
906         struct rcu_head     fu_rcuhead;
907     } f_u;
908     struct path     f_path;
909     struct inode        *f_inode;   /* cached value */
910     const struct file_operations    *f_op;
911
```

文件系统信息结构主要成员如下：

include > linux > C fs_struct.h

```
 9  struct fs_struct {
10      int users;
11      spinlock_t lock;
12      seqcount_t seq;
13      int umask;
14      int in_exec;
15      struct path root, pwd;
16  } __randomize_layout;
17
```

打开文件表的数据结构如下：

include > linux > C fdtable.h

```
26  struct fdtable {
27      unsigned int max_fds;
28      struct file __rcu **fd;      /* current fd array */
29      unsigned long *close_on_exec;
30      unsigned long *open_fds;
31      unsigned long *full_fds_bits;
32      struct rcu_head rcu;
33  };
34
```

二、Ext2/Ext3/Ext4 文件系统

1、基础概念

Linux 上的文件系统一般来说就是 EXT2 或 EXT3，现代计算机大部分文件存储功能都是由机械硬盘这种设备提供的。Linux 以文件的形式对计算机中的数据和硬件资源进行管理，也就是彻底的一切皆文件，反映在 Linux 的文件类型上就是：普通文件、目录文件（也就是文件夹）、设备文件、链接文件、管道文件、套接字文件（数据通信的接口）等等。

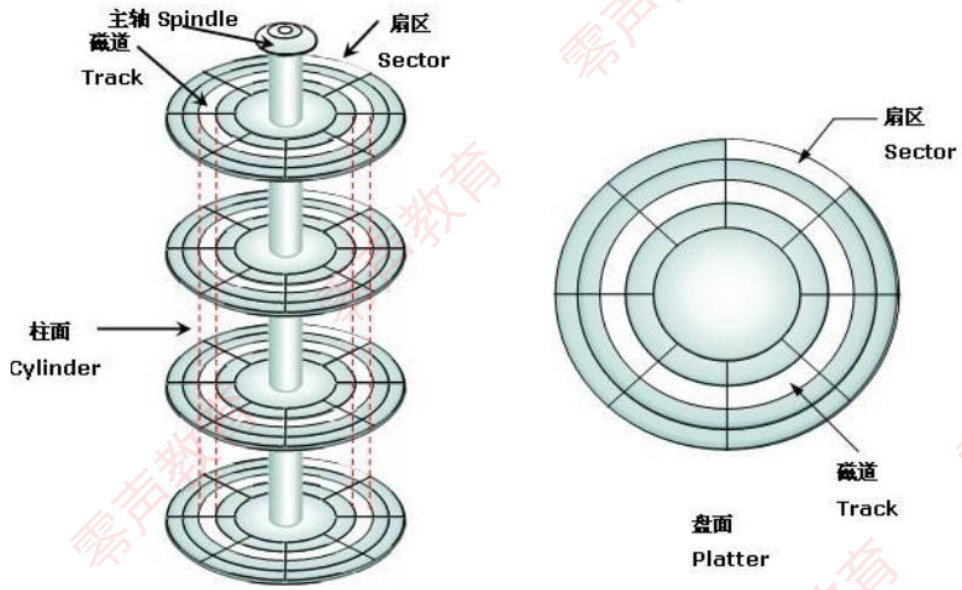
EXT 是延伸文件系统（英语：Extended file system，缩写为 ext 或 ext1），也译为扩展文件系统，一种文件系统，于 1992 年 4 月发表，是为 linux 核心所做的第一个文件系统。

2、文件系统类型

Linux: 存在几十个文件系统类型：ext2，ext3，ext4(1EB，最大单文件 16TB)，xfs，brtfs(固态盘)等，不同文件系统采用不同的方法来管理磁盘空间，各有优劣。**Windows:** FAT16、FAT32、NTFS 等。其他：RAMFS（内存文件系统）、NFS（网络文件系统）、Linux swap（交换分区用以提供虚拟内存）

a、机械硬盘组成结构以西部数据为案例：（SSD，HDD）

B、逻辑结构：



盘面，磁道、柱面、扇区。80 年。网络存储设计构架、台式机游戏硬盘架构、企业级硬盘架构（4k 视频）。

3、Ext2 文件系统

EXT2 第二代扩展文件系统（英语：second extended filesystem，缩写为 ext2），是 LINUX 内核所用的文件系统。它开始由 Rémy Card 设计，用以代替 ext，于 1993 年 1 月加入 linux 核心支持之中。ext2 的经典实现为 LINUX 内核中的 ext2fs 文件系统驱动，最大可支持 2TB 的文件系统，至 linux 核心 2.6 版时，扩展到可支持 32TB。

a、Ext2 文件系统格式

The Second Extended File System(ext2)文件系统是 Linux 系统中的标准文件系统，是通过对 Minix 的文件系统进行扩展而得到的，其存取文件的性能极好。在 ext2 文件系统中，文件由 inode（包含有文件的所有信息）进行唯一标识。一个文件可能对应多个文件名，只有在所有文件名都被删除后，该文件才会被删除。此外，同一文件在磁盘中存放和被打开时所对应的 inode 是不同的，并由内核负责同步。

b、磁盘组织结构

在 ext2 系统中，所有元数据结构的大小均基于“块”，而不是“扇区”。块的大小随文件系统的大小而有所不同。而一定数量的块又组成一个块组，每个块组的起始部分有多种多样的描述该块组各种属性的元数据结构。

超级块（每个 ext2 文件系统都必须有一个超级块），ext2_super_block 对应内核源码如下：

fs > ext2 > C ext2.h > ext2_super_block

```
414  /*
415   * Structure of the super block
416   */
417  struct ext2_super_block {
418      __le32  s_inodes_count;     /* Inodes count */
419      __le32  s_blocks_count;     /* Blocks count */
420      __le32  s_r_blocks_count;   /* Reserved blocks count */
```

块组描述符（一个块组描述符用以描述一个块组的属性），ext2_group_desc 对应内核源码如下：

```
fs > ext2 > C ext2.h > ...
197  /*
198   * Structure of a blocks group descriptor
199   */
200  struct ext2_group_desc
201  {
202      __le32  bg_block_bitmap;        /* Blocks bitmap block */
203      __le32  bg_inode_bitmap;        /* Inodes bitmap block */
204      __le32  bg_inode_table;     /* Inodes table block */
205      __le16  bg_free_blocks_count;   /* Free blocks count */
206      __le16  bg_free_inodes_count;   /* Free inodes count */
207      __le16  bg_used_dirs_count; /* Directories count */
208      __le16  bg_pad;
209      __le32  bg_reserved[3];
210  };
```

inode 表（inode 表用于跟踪定位每个文件，包括位置、大小等（但不包括文件名），一个块组只有一个 inode 表）。ext2_inode 对应内核源码如下：

```
fs > ext2 > C ext2.h > ext2_inode
300  /*
301   * Structure of an inode on the disk
302   */
303  struct ext2_inode {
304      __le16  i_mode;     /* File mode */
305      __le16  i_uid;      /* Low 16 bits of Owner Uid */
306      __le32  i_size;     /* Size in bytes */
307      __le32  i_atime;    /* Access time */
308      __le32  i_ctime;    /* Creation time */
309      __le32  i_mtime;    /* Modification time */
```

目录结构（在 ext2 文件系统中，目录是作为文件存储的），ext2_dir_entry_2 对应内核源码如下：

```
fs > ext2 > C ext2.h > ext2_dir_entry
582  /*
583   * Structure of a directory entry
584   */
585
586  struct ext2_dir_entry {
587      __le32  inode;          /* Inode number */
588      __le16  rec_len;        /* Directory entry length */
589      __le16  name_len;       /* Name length */
590      char    name[];         /* File name, up to EXT2_NAME_LEN */
591  };
592
```

4、Ext3 文件系统

EXT3 是第三代扩展文件系统（英语：Third extended filesystem，缩写为 ext3），是一个日志文件系统，常用于 Linux 操作系统。它是很多 Linux 发行版的默认文件系统。 Linux ext3 文件系统包括 3 个级别的日志：日记、顺序和回写。

Linux ext3 日志文件系统我：高可用性、数据的完整性、文件系统的速度、数据转换和多种日志模式。

5、Ext4 文件系统

EXT4 是第四代扩展文件系统（英语：Fourth extended filesystem，缩写为 ext4）是 Linux 系统下的日志文件系统，是 ext3 文件系统的后继版本。Ext4 产生原因是开发人员在 Ext3 中加入了新的高级功能，但在实现的过程出现了几个重要问题：

- （1）一些新功能违背向后兼容性
- （2）新功能使 Ext3 代码变得更加复杂并难以维护
- （3）新加入的更改使原来十分可靠的 Ext3 变得不可靠。

6、Linux ext4 文件系统特性：

- 更大的文件系统和更大的文件（Ext3-32TB 文件系统和 2TB 文件。Ext4-1EB 文件系统和 16TB 文件 容量）
- 更多的子目录数（Ext3 支持 32000 个子目录，Ext4 取消此限制）
- 更多的块和 i-节点数量（32 位空间记录块数量，64 位空间记录块数量）
- 多块分配
- 日志校验功能
- 支持“无日志”模式
- 在线碎片整理

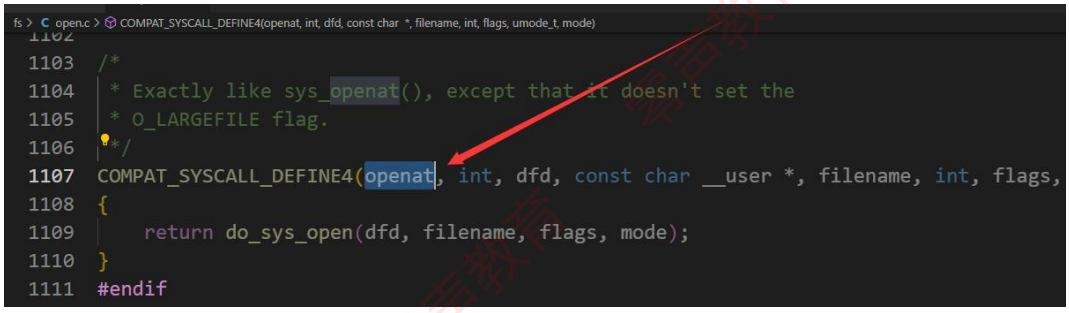
7、打开文件技术原理/关闭文件流程

a、需要在父目录的数据中查找文件对应的目录项，从目录项得到索引节点的编号，然后在内存中创建索引节点的副本。

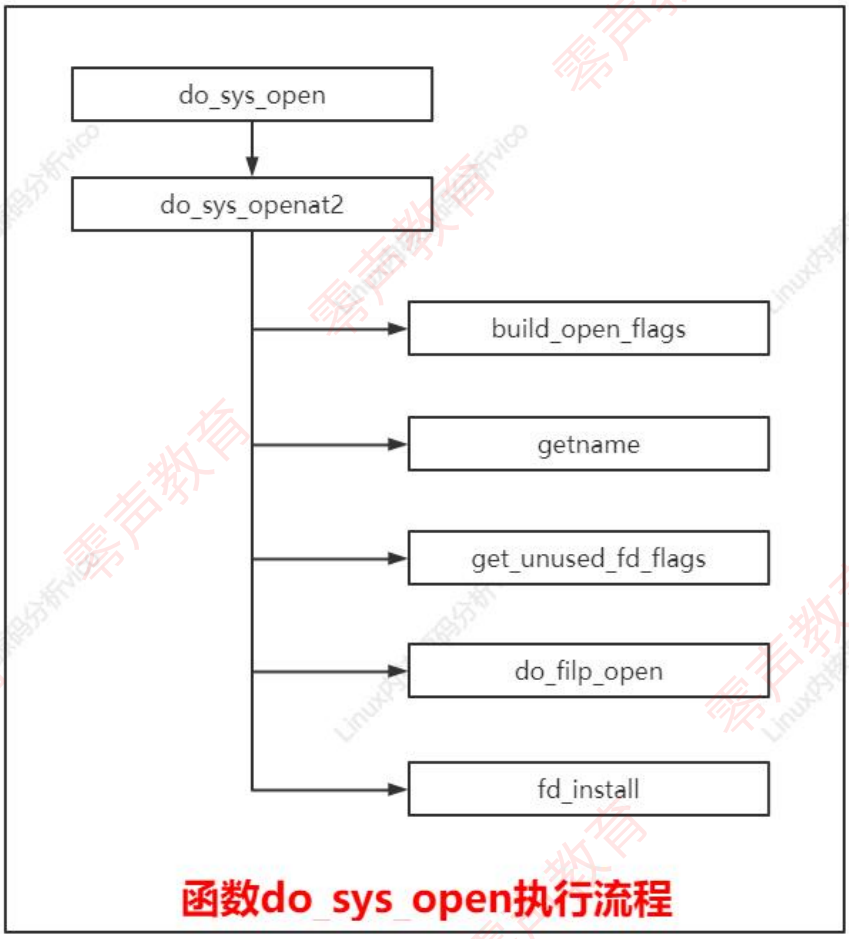
b、需要分配文件的一个打开实例--file 结构体，关联到文件的索引节点。

c、在进程的打开文件表中分配一个文件描述符，把文件描述符和打开实例的映射添加到进程的打开文件表中。

系统调用 open 和 openat 都把主要工作委托给函数 do_sys_open，open 传入特殊的文件描述符 AT_FDCWD，表示“如果文件路径是相对路径，就解释为相对调用进程的当前工作目录”。



【函数 do_sys_open()的执行流程】

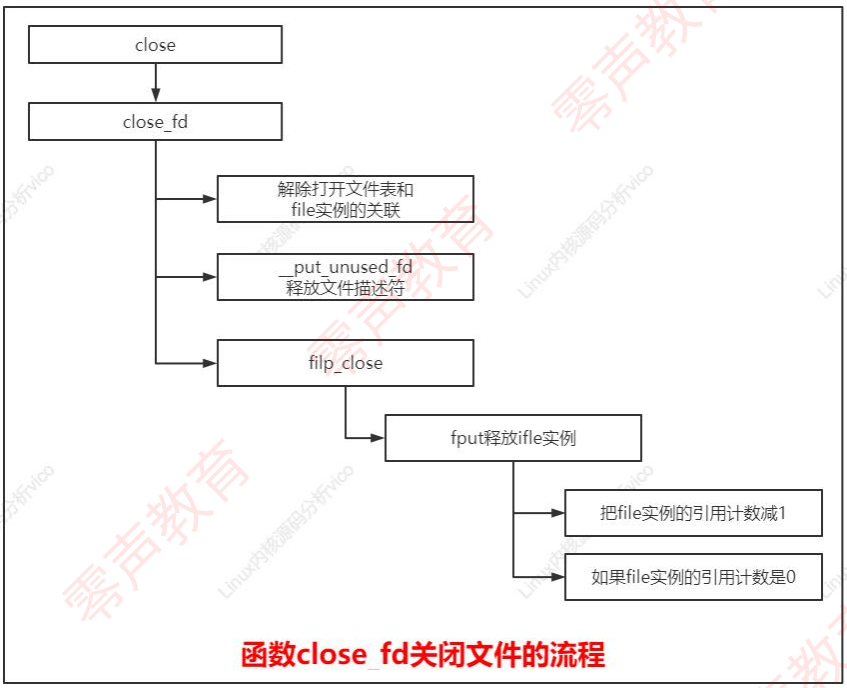


【关闭文件流程】

进程可以使用系统调用 close 关闭文件，进程退出时，内核将会把进程打开的所有文件关闭。

fs > C open.c > SYSCALL_DEFINE1(close, unsigned int, fd)

```
1151
1152  /*
1153   * Careful here! We test whether the file pointer is NULL before
1154   * releasing the fd. This ensures that one clone task can't release
1155   * an fd while another clone is opening it.
1156   */
1157  SYSCALL_DEFINE1(close, unsigned int, fd)
1158  {
1159      int retval = __close_fd(current->files, fd);
1160
1161      /* can't restart close syscall because file table entry was cleared */
1162      if (unlikely(retval == -ERESTARTSYS ||
1163                   retval == -ERESTARTNOINTR ||
1164                   retval == -ERESTARTNOHAND ||
1165                   retval == -ERESTART_RESTARTBLOCK))
1166          retval = -EINTR;
1167
1168      return retval;
1169  }
1170
```



函数close_fd关闭文件的流程

三、注册文件系统类型及实现

因为每种文件系统的超级块的格式不同，所以每种文件系统需要向虚拟文件系统注册文件系统类型 file_system_type，实现 mount 方法来读取和解析超级块。函数 register_filesystem 用来注册文件系统类型：

fs > C filesystems.c

```
71
72  int register_filesystem(struct file_system_type * fs)
73  {
74      int res = 0;
75      struct file_system_type ** p;
76
77      BUG_ON(strchr(fs->name, '.'));
78      if (fs->next)
79          return -EBUSY;
80      write_lock(&file_systems_lock);
81      p = find_filesystem(fs->name, strlen(fs->name));
82      if (*p)
83          res = -EBUSY;
84      else
85          *p = fs;
86      write_unlock(&file_systems_lock);
87      return res;
88  }
89
90  EXPORT_SYMBOL(register_filesystem);
91
```

管理员可以执行命令：cat /proc/filesystems --查看已经注册的文件系统类型。