

操作系统

Operating Systems

L32 目录解析代码实现

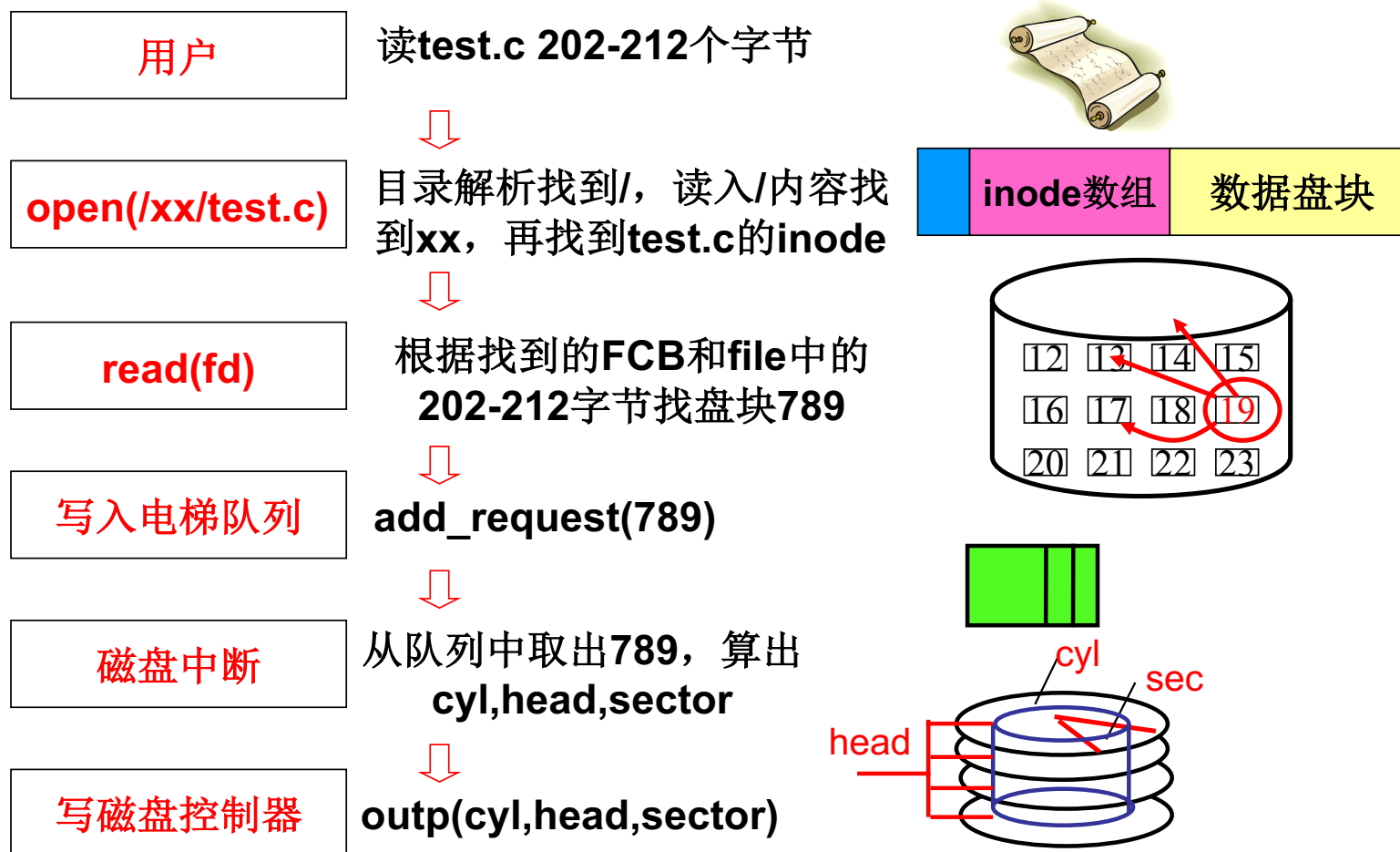
Directory Resolution

lizhijun_os@hit.edu.cn

综合楼411室

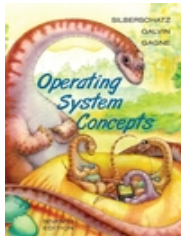
授课教师：李治军

“完成全部映射下”的磁盘使用

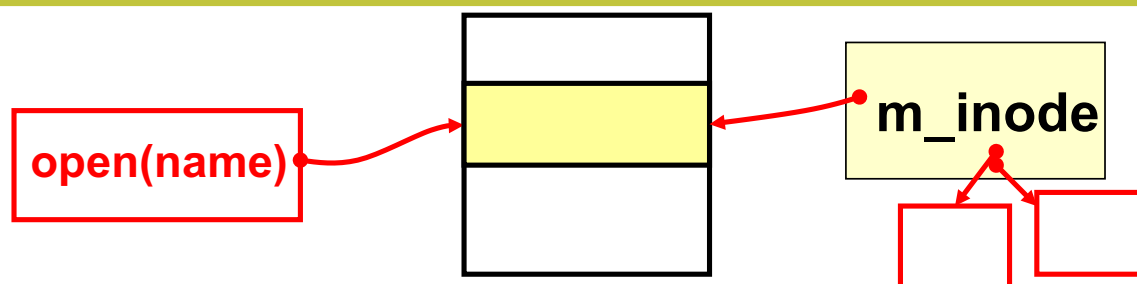




一个实际运转的文件系统!



就是将open弄明白...



在linux/fs/open.c中

```
int sys_open(const char* filename, int flag)
{   i=open_namei(filename, flag, &inode);
    ... }
```

解析路径!

```
int open_namei(...)
{   dir=dir_namei(pathname, &namelen, &basename);
```

```
static struct m_inode *dir_namei()
{   dir=get_dir(pathname); }
```



get_dir完成真正的目录解析

```
static struct m_inode *get_dir(const char *pathname)
{ if((c=get_fs_byte(pathname))== '/')
    {inode=current->root; pathname++;}
  else if(c) inode=current->pwd;
  while(1){if(!c) return inode; //函数的正确出口
    bh=find_entry(&inode,thisname,namelen,&de);
    int inr=de->inode; int idev=inode->i_dev;
    inode=iget(idev,inr); //根据目录项读取下一层inode}}
```

解析从此处开始!

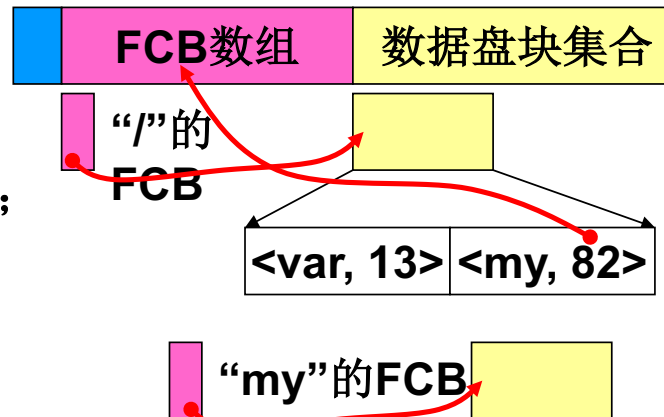
■ 核心的四句话正好对应目录树的四个

重点: (1)root: 找到根目录;

(2)find_entry: 从目录中读取目录项;

(3)inr: 是目录项中的索引节点号;

(4)iget: 再读下一层目录



目录解析 — 从根目录开始

```
inode=current->root;
```

■ 又是current(task_struct), 仍然是拷贝自init进程

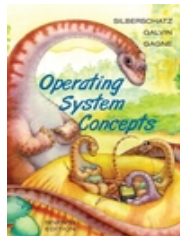
```
void init(void)
{  setup((void *) &drive_info);
```

一句看过无数次, 又略过无数次的语句

```
sys_setup(void * BIOS) //在kernel/hd.c中
{  hd_info[drive].head = *(2+BIOS);
   hd_info[drive].sect = *(14+BIOS);
   mount_root(); ... }
```

```
void mount_root(void) //在fs/super.c中
{
    mi=iget(ROOT_DEV, ROOT_INO);
    current->root = mi;
}
```

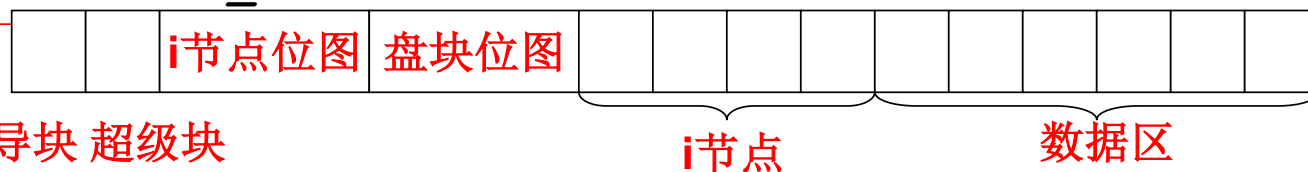
```
#define ROOT_INO 1
```



读取inode — iget

```
struct m_inode * iget(int dev, int nr)
{
    struct m_inode * inode = get_empty_inode();
    inode->i_dev=dev; inode->i_num=nr;
    read_inode(inode); return inode;
}
```

```
static void read_inode(struct m_inode *inode)
{
    struct super_block *sb=get_super(inode->i_dev);
    lock_inode(inode);
    block=2+sb->s_imap_blocks+sb->s_zmap_blocks+
        (inode->i_num-1)/INODES_PER_BLOCK;
    bh=bread(inode->i_dev,block);
    inode=bh->data[(inode->i_num-1)%INODES_PER_BLOCK];
    unlock_inode(inode);
}
```



开始目录解析 — `find_entry(&inode,name,...,&de)`

■ `de`: directory entry(目录项)

```
struct dir_entry{  
    unsigned short inode; //i节点号  
    char name[NAME_LEN]; //文件名 }  
}
```

```
#define NAME_LEN 14
```

在`fs/namei.c`中

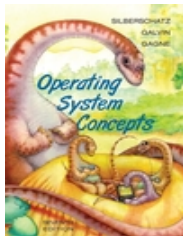
```
static struct buffer_head *find_entry(struct m_inode  
**dir, char *name, ..., struct dir_entry ** res_dir)  
{ int entries=(*dir)->i_size/(sizeof(struct dir_entry));  
  int block=(*dir)->i_zone[0];  
  *bh=bread((*dir)->i_dev, block);  
  struct dir_entry *de =bh->b_data;  
  while(i<entries) {  
      if (match(namelen, name, de) )  
          { *res_dir=de; return bh; } de++; i++; }  
}
```



while(i<entries)...

```
while(i<entries) //entries是目录项数
{ if((char*)de> = BLOCK_SIZE+bh->b_data)
    brelse(bh);
    block=bmap(*dir,i/DIR_ENTRIES_PER_BLOCK);
    bh=bread(*dir->i_dev,block);
    de=(struct dir_entry*)bh->b_data;
} //读入下一块上的目录项继续match
if(match(namelen,name,de))
{ *res_dir=de;return bh;}
de++; i++; }
```

#define BLOCK_SIZE
1024 //两个扇区



操作系统全图

