操作系统 Operating Systems

L30 文件使用磁盘的实现

Files Implementation

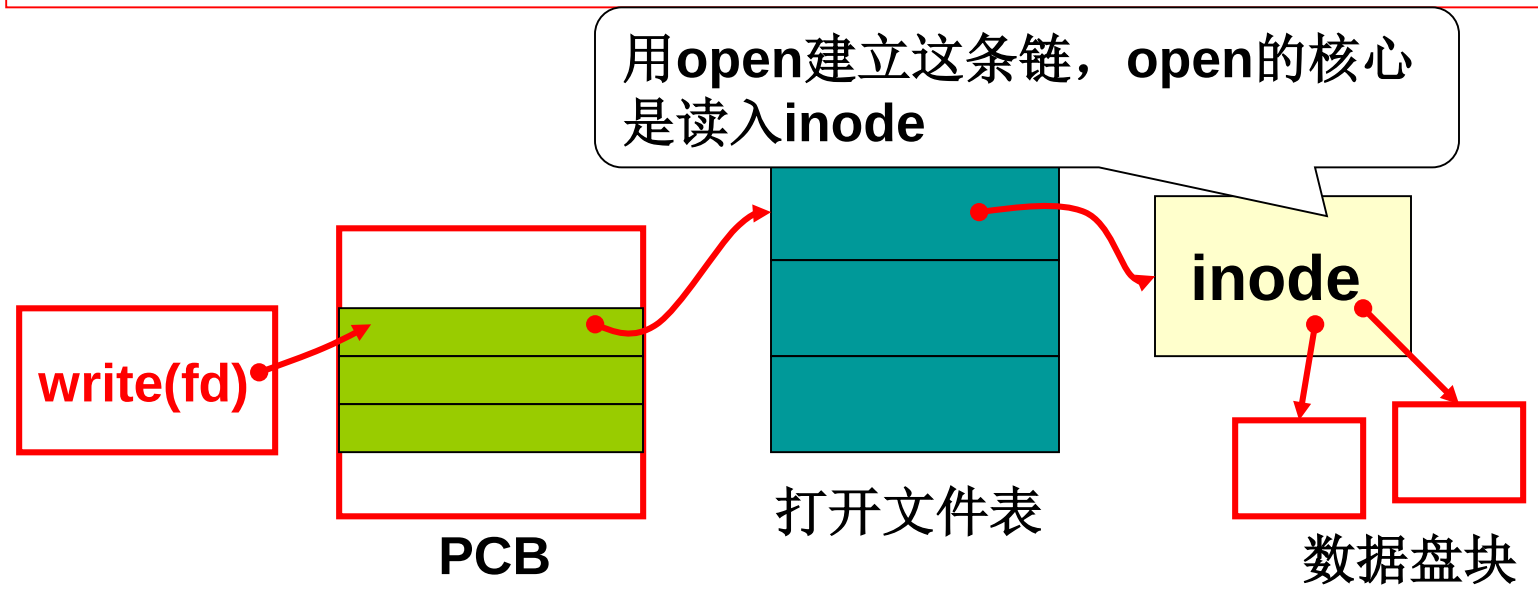
lizhijun_os@hit.edu.cn
综合楼411室

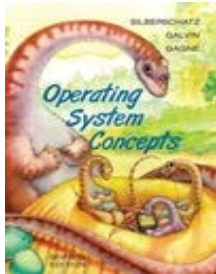
授课教师：李治军

再一次使用磁盘，通过文件使用

在fs/read_write.c中

```
int sys_write(int fd, const char* buf, int count)
{ struct file *file = current->filp[fd];
  struct m_inode *inode = file->inode;
  if(S_ISREG(inode->i_mode))
      return file_write(inode, file, buf, count); }
```



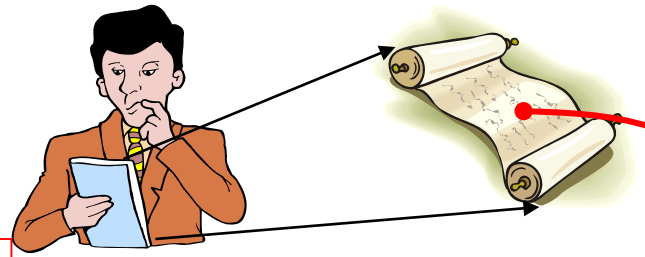


file_write的工作过程应该就是...

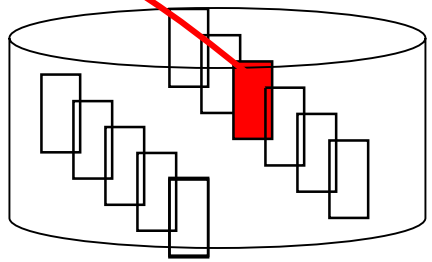
```
file_write(inode, file, buf, count);
```

- (1)首先需要知道是些哪段字符?

file中一个读写指针，是开始地址(fseek就是修改它)，再加上count



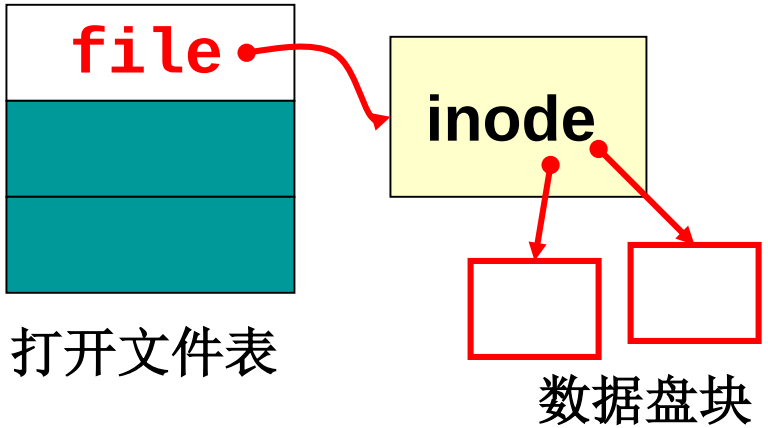
test.c中的200-212字符对应盘块789



将200-212字符删去

- (2)找到要写的盘块号?

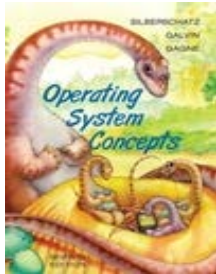
inode就是用来干这事的



打开文件表

数据盘块

- (3)用盘块号、buf等形成request放入“电梯”



file_write的实现

```
int file_write(struct m_inode *inode, struct file
*filp, char *buf, int count)
{  off_t pos;
   if(filp->f_flags&O_APPEND)
     pos=inode->i_size; else pos=filp->f_pos;
```

用file知道文件流读写的字符区间，从哪到哪!

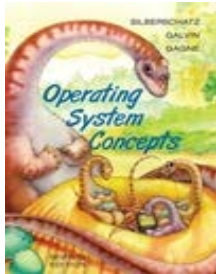
```
while(i<count){
  block=create_block(inode, pos/BLOCK_SIZE);
  bh=bread(inode->i_dev, block);
  int c=pos%BLOCK_SIZE; char *p=c+bh->b_data;
  bh->b_dirt=1; c=BLOCK_SIZE-c; pos+=c;
  ... while(c-->0) *(p++)=get_fs_byte(buf++);
  brelse(bh); }
filp->f_pos=pos; }
```

算出对应的块!

放入“电梯”队列!

一块一块拷贝用户字符，并且释放写出!

修改pos，使之总是对!



create_block算盘块，文件抽象的核心

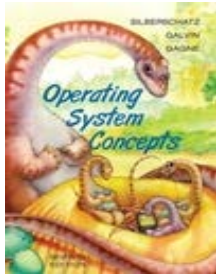
```
while(i<count){
  block=create_block(inode, pos/BLOCK_SIZE);
  bh=bread(inode->i_dev, block);
```

create=1的_bmap，没有映射时创建映射

```
int _bmap(m_inode *inode, int block, int create)
{ if(block<7){ if(create&&!inode->i_zone[block])
  { inode->i_zone[block]=new_block(inode->i_dev);
    inode->i_ctime=CURRENT_TIME; inode->i_dirt=1;}
    return inode->i_zone[block];}
  block-=7; if(block<512){
    bh=bread(inode->i_dev,inode->i_zone[7]);
    return (bh->b_data)[block];} ...
```

一个盘块号2个字节

```
struct d_inode{ unsigned short i_mode;...
  unsigned short i_zone[9];
//(0-6):直接数据块，(7):一重间接，(8):二重间接 }
```



m_inode，设备文件的inode

```
struct m_inode{                    //读入内存后的inode
    unsigned short i_mode;         //文件的类型和属性
    ...
    unsigned short i_zone[9];      //指向文件内容数据块
    struct task_struct *i_wait;
    unsigned short i_count;
    unsigned char i_lock;
    unsigned char i_dirt;   ... }
```

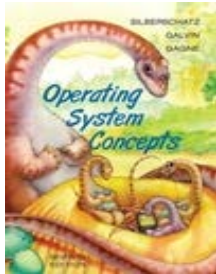
前几项和d_inode一样!

多个进程共享的打开这个inode，有的进程等待...

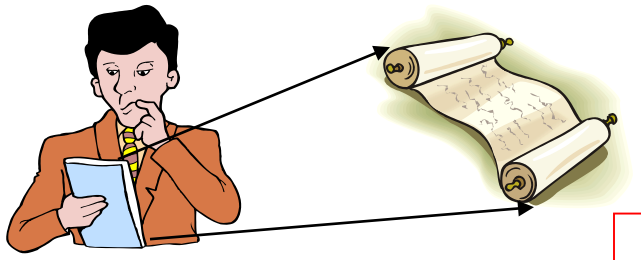
```
int sys_open(const char* filename, int flag)
{ if(S_ISCHR(inode->i_mode)) //字符设备
    { if(MAJOR(inode->i_zone[0])==4)
      current->tty=MINOR(inode->i_zone[0]);}
```

设备文件

```
#define MAJOR(a)(((unsigned)(a))>>8)) //取高字节
#define MINOR(a)((a)&0xff)            //取低字节
```



伟大的文件视图...



789

磁盘控制器

cyl

sec

head

电梯算法

请求队列

将200-212字符删去

用inode的i_zone算出200-212字符对应的盘块是789

第一条路：读写磁盘

write(fd)

PCB

打开文件表

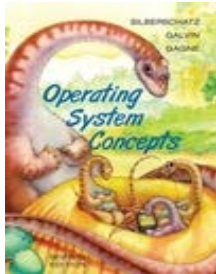
inode

```
S_ISCHR(inode->i_mode)
crw_table[MAJOR(dev)]
 tty_write(minor,buf)
 con_write
```

```
mov ax,pos
```

hello

第二条路：输出到显示器



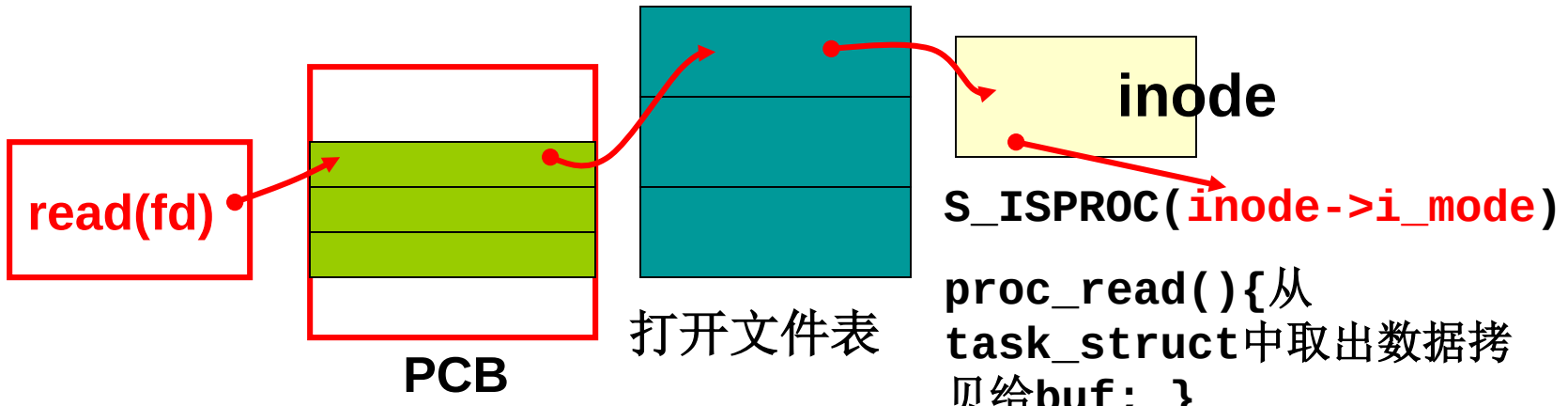
实践项目—实现proc文件

```
# cat /proc/psinfo
pid state   father  counter        start_time
  0     1      -1        0                 0
  1     1       0       28                 1
```

- 这些信息显然不在磁盘上，是特殊文件

cat命令对应的程序

```
main(char *argv[])
{  fd=open(argv[1]); while(文件没有结束){
     read(fd,buf,100); printf(buf); }
```



让inode告诉我们从哪条路走?

```
void init()
{  setup((void *) &drive_info);
   mkdir("/proc", 0755);
   mknod("/proc/psinfo", S_IFPROC|0444);
}
```

```
#define S_IFCHR  0020000
#define S_IFPROC  0040000
```

mode_t mode

```
int sys_read(unsigned int fd,char * buf,int count)
{  if(S_ISCHR(inode->i_mode)) ..
   if(S_ISPROC(inode->i_mode))
      proc_read(file,buf,count);
```

#define S_ISCHR(m) (((m) & S_IFMT) == S_IFCHR)

```
int proc_read(file,char * buf,int count)
{ task_struct *p;
  sprintf(krnbuf,"%d,%d",p[0]->pid...);
  按照file->f_pos和count将krnbuf拷贝到buf中;
  修改file->f_pos;
```

实际读出的数量<要读的count，
就认为文件结束了