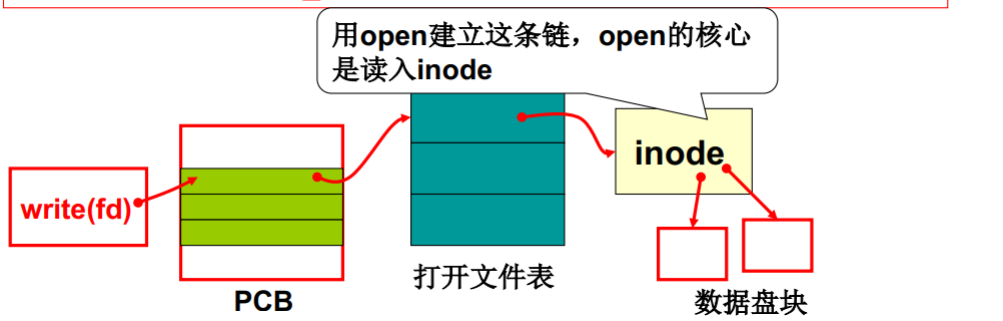
哈工大操作系统-L30文件使用磁盘的实现

1.文件使用file_write写入磁盘

再一次使用磁盘，通过文件使用

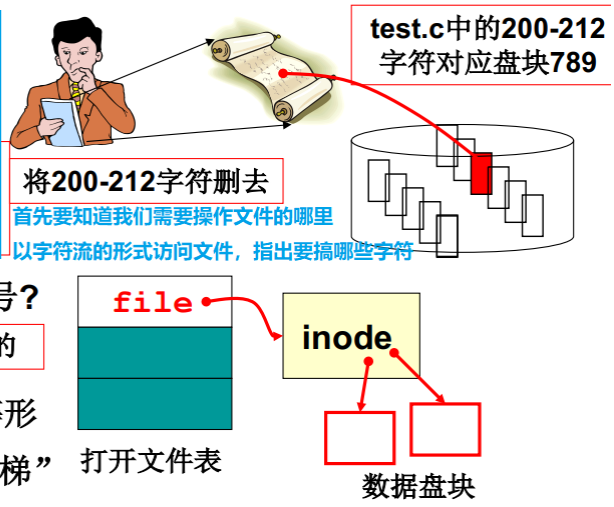
```
在fs/read_write.c中
int sys_write(int fd, const char* buf, int count)
{ struct file *file = current->filp[fd];
  struct m_inode *inode = file->inode; 这个inode就是FCB
  if(S_ISREG(inode->i_mode))
      return file_write(inode, file, buf, count); }写文件的函数
```



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
- (2)找到要写的盘块号?
  - inode就是用来干这事的
- (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;
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; }
```

用file知道文件流读写的字符区间，从哪到哪!

追加模式，则pos在最后

否则，上次读写到哪就是哪

算出对应的块!

计算出盘块号

用create block函数

向磁盘请求访问磁盘

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!

2.需要使用create_block函数计算盘块号

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);
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];} ...
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 }
```

create=1的_bmap，没有映射时创建映射

需要的盘块数目

如果需要的盘块数目小于7，那么我们可以放在直接数据块中

一个盘块号2个字节

存放一个盘块号要两个字节 而一个盘块是两个扇区，一共1024字节

因此一个盘块能存放512个盘块号

因此，在存放完7个盘块之后，如果剩余的小于512个，那么盘块号都能放在7号盘块。

如果多于512个，则放在下一个扇区

- 一个索引块(也是盘块)能存放512个盘块号。一个盘块号2字节，一个盘块两个扇区2*512B。

3.inode提供统一的文件视图

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
```

- 有字符文件的inode也有设备文件的inode
- 这些都为系统提供了统一的文件视图