# 文件系统例题与习题

同济大学计算机系操作系统 2024-1-2

姓名 2151569 学号 明添识

## 一、完整的文件读写过程

1、

int fd = open(“/usr/ast/Jerry”,3);//以可读可写方式打开文件

char data[300];

seek(fd, 500, 0);//将文件读写指针定位到第500字节

int count = read (fd, data, 300);//从文件读300字节到data

count = write(fd, data, 300);//从data写300字节到文件



图1、缓存队列 和 Jerry文件的索引表

详见文档《文件系统例题与习题2 完整的文件读写过程》

1. **open 系统调用**

**首先open系统调用根据传入的文件名查找 DiskInode 是否存在。若不存在则报错，存在进行下一步**

**用diskInode 号搜索内存中的 Inode表，若存在则对应inode表项的i\_count++，否则需要分配空闲的inode，读入磁盘diskinode 进行初始化**

**检查文件访问权限，若RWX不支持当前mode则报错，否则进行下一步**

**创建文件的打开结构，在进程打开文件表中分配 File 结构和索引号 fd，构建 Ffd->File->Inode的连接关系，最后返回索引号。**

**open 系统调用结束，i\_lastr = -1**

1. **seek文件指针移动**

**seek将File结构中的f\_offset文件指针移动到500，即f\_offset=500**

1. **read系统调用**

**首先初始化参数:m\_offset=f\_offset=500，m\_base=array, m\_count=300**

**由于 i\_last\_t + 1 = 0 = b\_n = m\_offset / 512，查找混合索引表，得到当前块的bn物理块号伪123，下一块的物理块号为124，则调用Breada(123， 124)，当前缓存块123不命中，所以需要Getblk由自由缓存队列分配缓存块，当前进程构造IO访存块，放入IO请求队列中，当前进程入睡等待IO操作完成。IO完成后唤醒当前进程，执行iomove将缓存块中500#-511#字节复制到用户空间data[0]-data[11]，缓存使用完毕之后解锁。修改IO参数，m\_offset=512, m\_base = data+12, m\_count = 300-12非零则仍需要继续读取下一块**

**当前逻辑块号bn = m\_offset / 512 = 1 ，块内偏移量0，读取250-12=238字节。而bn=i\_lastr + 1 breada(124, 0)，缓存命中124则将其上锁并复用缓存块内容，执行 iomove将缓存中 0#-237#字节复制到用户空间 data[12]-data[249]中，解锁缓存内容，修改IO参数，读操作结束，返回实际读入的字数250**

1. **write系统调用：**

**f\_offset ==750，文件长度 750。write 系统调用将 data 数组中的 300 个字节追加写在文 件尾部。写操作完成后，文件长度增加至 1050。**

**初始化 IO 参数：m\_offset = f\_offset = 750，m\_base = data，m\_count = 300 写 1#逻辑块当前逻辑块 bn = m\_offset/512 = 1，块内偏移量 238，写 274 字节至块结束。查混合索引表，得当前块 bn 的物理块号 124。 先读后写，字节数 274 不足 512 字节。124#物理块缓存命中，先读操作只需锁住该缓存块。 iomove 将用户空间中的数据 data[0] ~ data[273]写入这块缓存。 修改 IO 参数，m\_offset = 1024，m\_base = data+274，m\_count = 300-274。写至 1#逻辑块底部，异步写回磁盘。m\_count 非 0，还要继续写。**

**写操作导致文件长度增加，i\_size = 1024。2.3 写 2#逻辑块当前逻辑块 bn = m\_offset/512 = 2，块内偏移量 0，此次写 26 个字节。混合索引表中 2#逻辑块的物理块号是 0，系统为其分配新数据块 new，登记： i\_addr[2] = new。 读后写，这是因为写入的字节数 26 不足 512 字节。系统为新物理块 new 分配缓存 块，启动 IO 操作、同步读入磁盘数据块 new。iomove 将用户空间中的数据 data[274] ~ data[299]写入这块缓存。 修改 IO 参数，m\_offset = 1050，m\_count = 0。未写至 1#逻辑块底部，延迟写。缓存块打脏标记，释放。 写操作导致文件长度增加，i\_size = 1050。m\_count 是 0，write 系统调用结束。返回实际写入文件的字节数 300。**

2、识别文件的顺序读写操作 和 随机读写操作。每个系统调用完成后，**f\_offset =X+Y+Z， 1000 + Z，1000 + Z**？

图片包含 日历

描述已自动生成

## 二、文件系统的静态结构

1、Unix V6++系统，存放一个长102400字节的文件file1，需要使用多少磁盘存储资源 ？

答：一个目录项，存放在父目录文件中。一个DiskInode，存放在Inode区。

数据区：102400/512 = 200扇区用来存放文件数据。**d0 ~ d199 > 6个块**

2个一次间接索引块。i0 ，i1，**一个间接索引块管128个**

共计202个扇区。

混合索引结构如下：

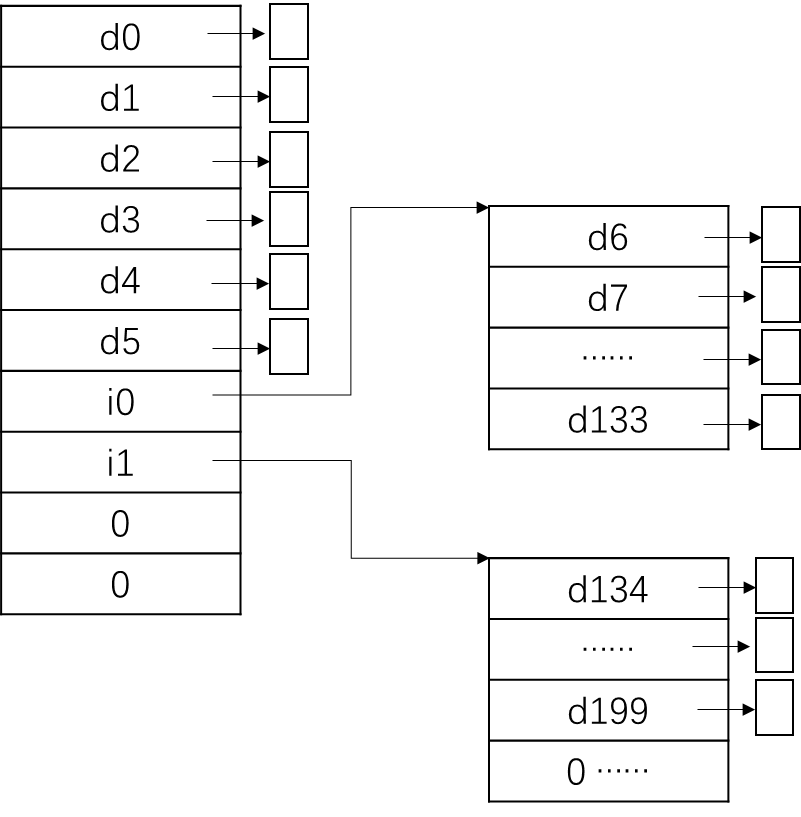


图1

2、Unix V6++系统，如下目录树。已知：目录 /，bin，etc，home，dev，root，user1和user2 分别是 1#，2#，3#，4#，5#，6#，10#和12#文件。请填空补全 / 和 home目录文件，多余的目录项，所有字段填0。

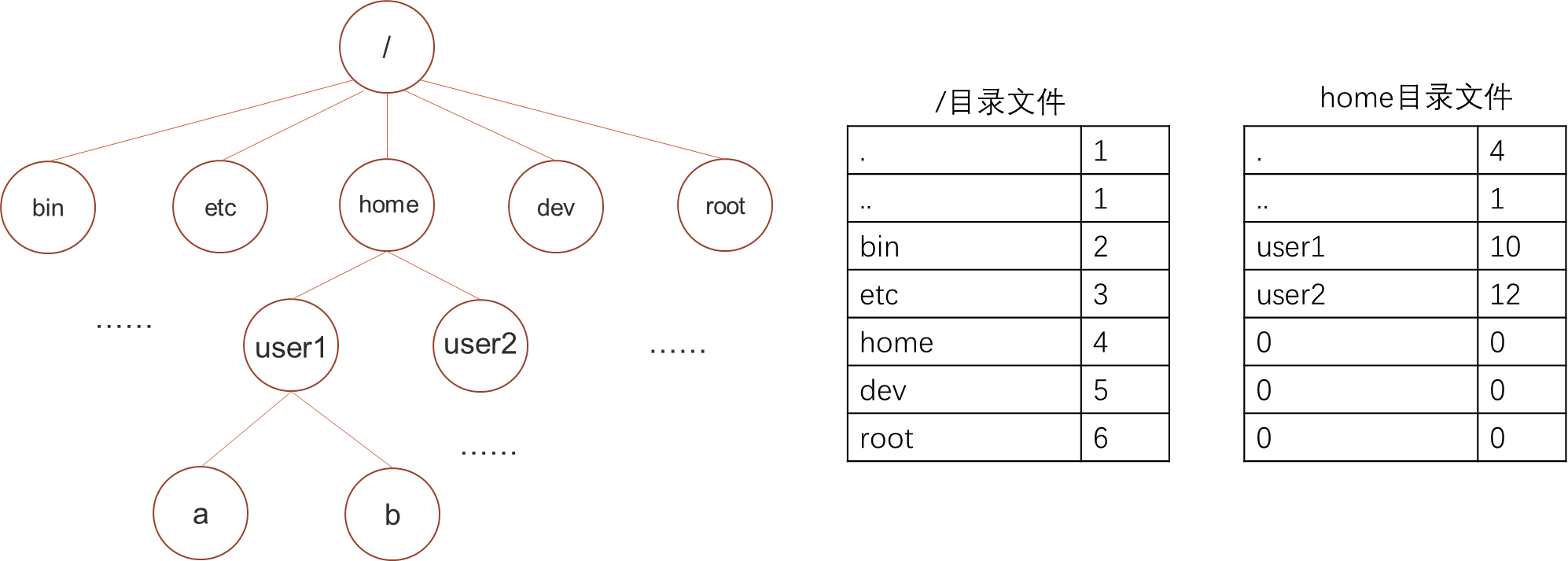


图2

3、Unix V6++系统，超级块1024字节，DiskInode 64字节，每个扇区512字节。简述系统加载100#DiskInode的过程。



图3

答：系统使用100#文件前，需要将100#DiskInode加载进分配给它的内存Inode。具体过程如下：

* 100除以8，商12，余4。100#inode在12#扇区，是这个扇区的4#inode。
* bp=Bread(0,12)，将12#扇区读入缓存块bp。
* IOmove(bp->b\_addr+256, &Inode[i], 64）// Inode[i]是分配给100#DiskInode的内存Inode

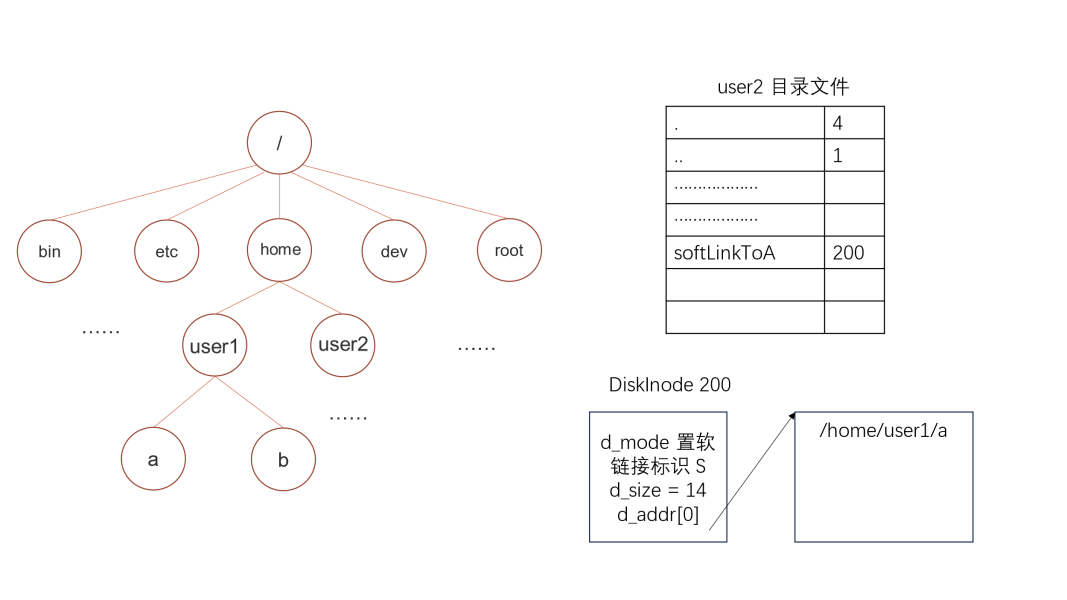
4、某磁盘剩余空间30%，无法创建新文件。这是什么情况？怎样避免这种情况的发生。

答：inode区用完了，磁盘存放了大量的小文件。

避免这种情况发生，最重要的是要明确文件卷的应用场合。如果存的都是小文件，格式化时，inode区要給大点儿。全是大文件，inode区小点儿。磁盘空间足够用，忽略。

5\*、软链接

软链接是一个存有目标文件名的小文件。有自己的DiskInode。分配有数据块。



user2目录下创建一个引用a文件的软链接softLinkToA。200#DiskInode分配给这个软链接文件。

## 三、Unix文件系统的使用

（一）打开文件结构

1、T0 时刻，系统中有两个进程P1 和P2，分别独立打开并同时访问小文件example。则在

内存打开文件结构中有（**A**）个内存Inode指向该文件？（**B**）个File结构记录着进程对文件的访问情况？

A. 1 B. 2

在哪个数据结构中登记有进程对文件的访问方式（读或读写）？(**File结构（f\_flag）**）

文件的读写指针保存在（**对应进程文件打开表项对应的File结构中**）？

组成文件的每个逻辑盘块（信息块）在磁盘上的地址保存在（**AC**）？

A. 内存Inode B. File结构 C. i\_addr数组

若P2 进程向文件追加写入10000 个字符后关闭该文件，引发（**A**）操作；稍后，P1 关

闭example 文件，引发（**ABC**）操作。

A. 释放file 结构 B. 释放内存i 节点

C. 将内存i 节点写回磁盘 D. 不执行任何操作

1. 假设foobar.txt文件的内容是字符串“1234567890”。请问

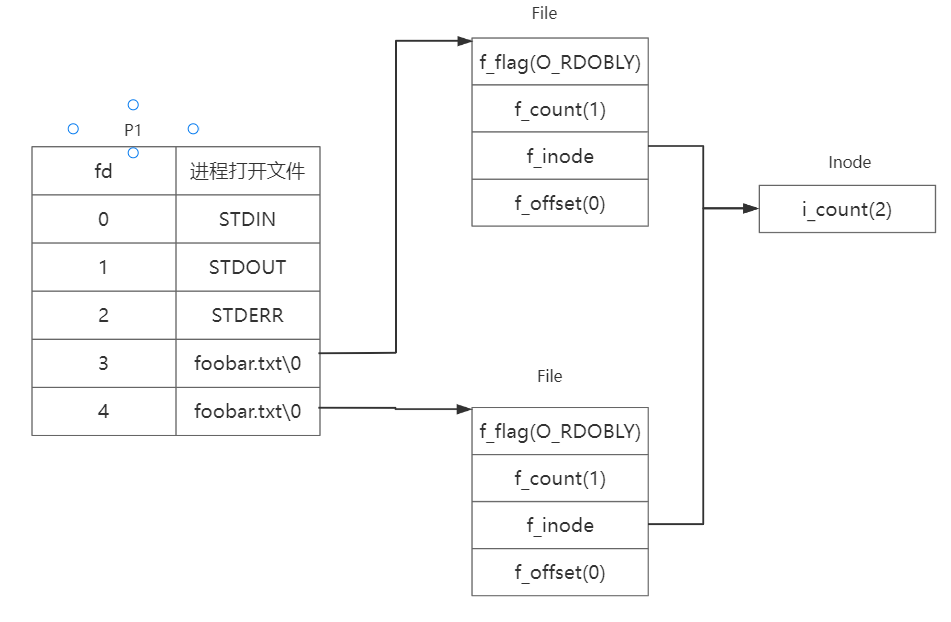
（1）这个程序的输出是什么？（2）画进程的打开文件结构

图形用户界面, 文本

描述已自动生成

1. **输出是：1**

**（2）**



3、fd1，fd2的值是几？

文本, 信件

描述已自动生成

**均为3**

4\*、输入、输出重定向的实现

已知：库函数 printf(“格式化串”，输出的内容)的执行分2步（1）使用格式化串处理输出的内容，生成待显示的字符串formatted\_string（2）执行系统调用

write(1，formatted\_string，sizeof(formatted\_string)。

1，是进程的标准输出文件，默认是终端的输出缓存；即进程打开文件表1#表项引用的File结构指向的是tty的内存inode。

输出重定向是指修改进程的1#文件描述符，让printf将生成的字符串写入一个指定的磁盘文件。例如：以下命令将cat程序的输出重定向至磁盘文件outFile。

$ cat existFile > outFile

cat命令向屏幕输出existFile文件内容。输出重定向后，cat命令不再向屏幕输出，原本输出的内容写入outFile文件。

输出重定向怎么实现呢？以下是一种可行的方法：shell进程创建一个子进程，子进程exec(cat)之前，连续执行系统调用close和open：

if ( fork( )==0 )

{

close(1);

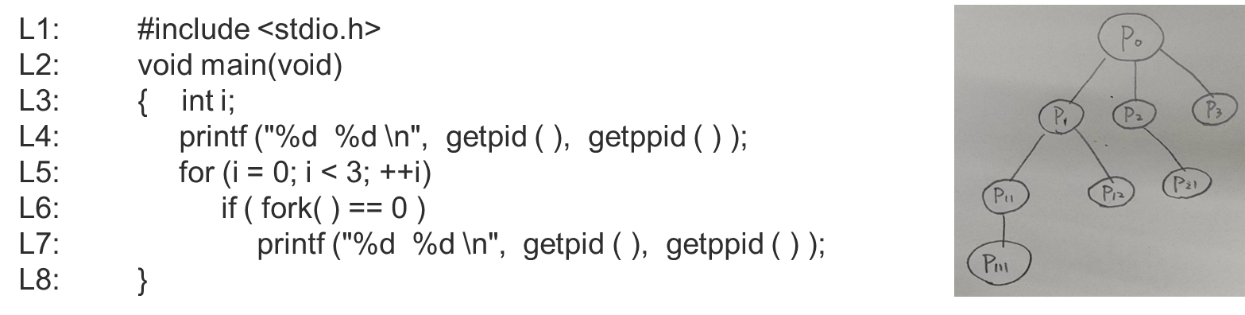
fd = open(outFile, 写)； // fd是1

exec(cat, \*\*\*\*); // cat程序执行时标准输出（1#文件描述符）是outFile文件

}

。。。父进程shell的程序分支。。。

5\*、fork套在for循环中，输出重定向之前，输出是8行，每个进程输出一行。输出重定向之后，输出是20行，为什么会是这样呢？



[参考答案] 库函数printf（其实是标准输入输出库stdio）在用户空间为每个文件描述符维护一个缓存。缓冲的工作模式与文件描述符引用的文件类型(d\_mode)相关。字符设备，也就是终端tty，行缓冲，遇到回车输入、输出。普通磁盘文件，块缓冲，写满4096字节输出。

输出重定向之前，printf向终端屏幕输出，所以1#文件描述符是行缓冲的。每个进程的输出有回车，故，创建子进程之前，父进程会将输出的字符行写入tty输出缓存并且清空库函数使用的用户缓存。也就是，子进程继承来的父进程图像中，没有父进程输出的字符行。。。每个进程输出一行，整个程序一共输出8行。

磁盘文件是块缓冲的，用户空间的这块缓存4096字节。本例（1）执行fork系统调用时，所有父进程的输出不足4096字节，留在用户缓存里（2）子进程继承父进程的输出，所以，前者会重复输出父进程的字符行（3）所有进程最终输出的数据量远远小于4096字节，所以任何进程终止前不会执行write系统调用（4）所有进程终止时执行write系统调用，将用户缓存中留存的字符行写入内核缓存块。综上，进程树上高度为4的节点输出4行，高度为3、2、1的节点分别输出3行、2行和1行。合起来，程序一共输出20行。

（二）系统调用的语义、执行过程和例题。详见文档《Unix V6++的目录和与之有关的系统调用》

1、fd = open(name,mode);

2、fd = creat(name,mode); // creat还是create，无所谓

3、close(fd);

4、unlink(name);

5、link(name1,name2);

（三）系统调用执行时的IO次数

1、以图1中代码为例，线性目录搜索文件 “/usr/ast/Jerry”。如果各级目录文件的内容如图7.32所示，则整个目录搜索的过程如下：



**图7.32：目录搜索示例**

（1）根据1号inode（根目录文件的Inode），打开根目录文件，由i\_mode中的标志位判断该文件确实为目录文件，由i\_addr找到根目录文件在磁盘中的位置，如图7.32所示，从101号盘块开始依次读入根目录文件；在根目录文件中，逐条记录查找到文件名为usr的目录项，该项显示usr文件的磁盘Inode号为56号。根目录DiskInode常驻内存，使用不需要执行IO操作，遍历根目录文件之前需要将101#盘块读入内存，1次IO。

（2）根据56号Inode（user文件的Inode），打开usr文件，由i\_mode中的标志位判断该文件确实为目录文件，则由i\_addr找到usr目录文件在磁盘中的位置，如图7.32所示，从132号盘块开始依次读入user目录文件；在usr目录文件中，逐条记录查找到文件名为ast的目录项，该项显示ast文件的磁盘Inode号为30号。2次IO，分别读入目录文件usr的DiskInode（56#）和数据块（132#盘块）。

（3）根据30号Inode（ast文件的Inode），打开ast文件，由i\_mode中的标志位判断该文件确实为目录文件，则由i\_addr找到ast目录文件在磁盘中的位置，如图7.32所示，从406号盘块开始依次读入ast目录文件；在ast目录文件中逐条记录查找到文件名为Jerry的目录项，该项显示Jerry文件的磁盘Inode号为80号。2次IO，分别读入目录文件ast的DiskInode（30#）和数据块（406#盘块）。

（4）找到80号Inode，即找到“/usr/ast/Jerry”文件。

结论：线性目录搜索好慢好慢呀呀呀。。。一定要想办法优化。

能用，尽量用相对路径名引用文件，会快好多。看下面的例子。

2、用户user1登录后，打开文件a。请问，执行系统调用open(“a”, RDWR)需要执行几次IO操作？

图片包含 文本

描述已自动生成

[参考答案] 2次。

用户登录时，系统自动打开家目录。所以，user1上机的整个过程，家目录/home/user1的DiskInode常驻内存，为其提供相对路径名目录搜索服务。

这个open系统调用使用相对路径名打开文件a，无需加载父目录DiskInode。

需要读取父目录文件user1，数据块缓存不命中，IO一次。

需要读取目标文件a的DiskInode，缓存不命中，IO一次。

所以，open系统调用需要执行2次IO操作。

3、open系统调用结束后，读下图中的文件file1，假设所有块缓存不命中，不考虑预读。（1）读3#逻辑块，几次IO？（2）读6#逻辑块，几次IO？（3）读入6#逻辑块之后，接着读100#逻辑块，几次IO？（4）随后，写198#逻辑块中的第200#字节，几次IO？

形状

中度可信度描述已自动生成[参考答案]

（1）读3#逻辑块，1次IO。0#~5#逻辑块的物理块号在Inode中，文件打开后在内存。所以，读3#逻辑块IO一次，读入物理块d3。

（2）读6#逻辑块，2次IO。第一次读入物理块i0、得到文件的第一个索引块，get物理块号d6。第二次读入物理块d6，get文件数据。

（3）读100#逻辑块，1次IO。100#逻辑块的地址映射关系在第一个索引块中，已经在内存里了。所以，只需一次IO读入物理块d100。

（4）写198#逻辑块中的第200#字节，2次IO。一次读入物理块i1，获得第2个索引块，查询得知198#逻辑块存放在d198#物理块中。第二次IO，执行先读操作，将物理块198读入磁盘高速缓存。没有写IO，因为这个逻辑块没有写满，就让它呆在缓存池里。