操作系统　第五章　外设管理

5.4　磁盘文件读写技术

同济大学计算机系

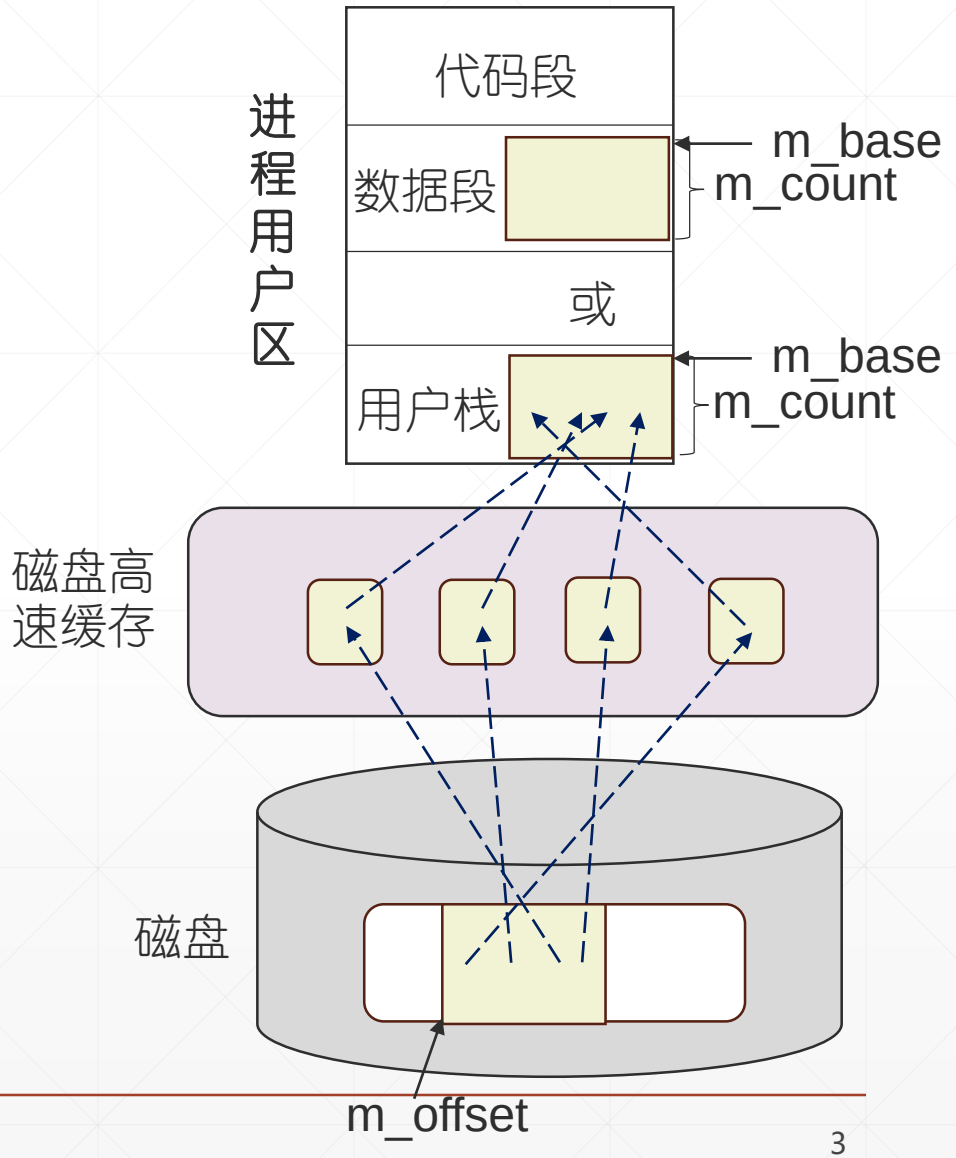
读

Unix V6++的read系统调用

read系统调用同步读入磁盘文件中连续的一块数据，送进程用户区。

- **IO参数，在user结构 的 u_IOParam 字段**
  - **m_offset，数据在文件中的偏移量**
  - **m_base，数据在用户区的首地址**
  - **m_count，需要读入的数据量**
- **返回值**
  - **0，EOF文件尾（m_offset == 文件长度 f_size），未读到数据**
  - **正整数，实际读入的字节数**

预读标识：i_lastr：　每个使用中的文件一个i_lastr，记录上次 IO的逻辑块号。



read( ) 系统调用框架(readi 核心部分)

- 循环，一次处理一个逻辑块

(1) if (m_offset == i_size 或者 m_count == 0)　　// (a) 文件尾，没有数据可读了　(b) 所有数据均已读入

　　返回;

(2) 用 m_offset，m_count 和 文件长度 i_size 计算 当前块的逻辑块号bn，块内偏移量offset 和 需要读取的字节数 n。

bn = m_offset / 512，offset = m_offset % 512，n = min(512 – offset, m_count) 或 min（文件长度 % 512 – offset, m_count)

(3) 地址映射 Bmap(bn)，得当前块的物理块号 blkno 和 预读块的物理块号 rablkno。

(4) 同步读入当前块，异步读入预读块

```
if ( i_lastr+1==bn )
        bp = Breada( dev，blkno，rablkno );
else
        bp = Bread( dev，blkno );
```

(5) 将当前缓存块中的数据送用户区

```
IOmove(bp->b_addr+offset，m_base，n);
```

(6) i_lastr = bn;

(7) 修正IO参数，为读下一个字符块做准备

```
m_offset+=n
m_base+=n
m_count-=n
```

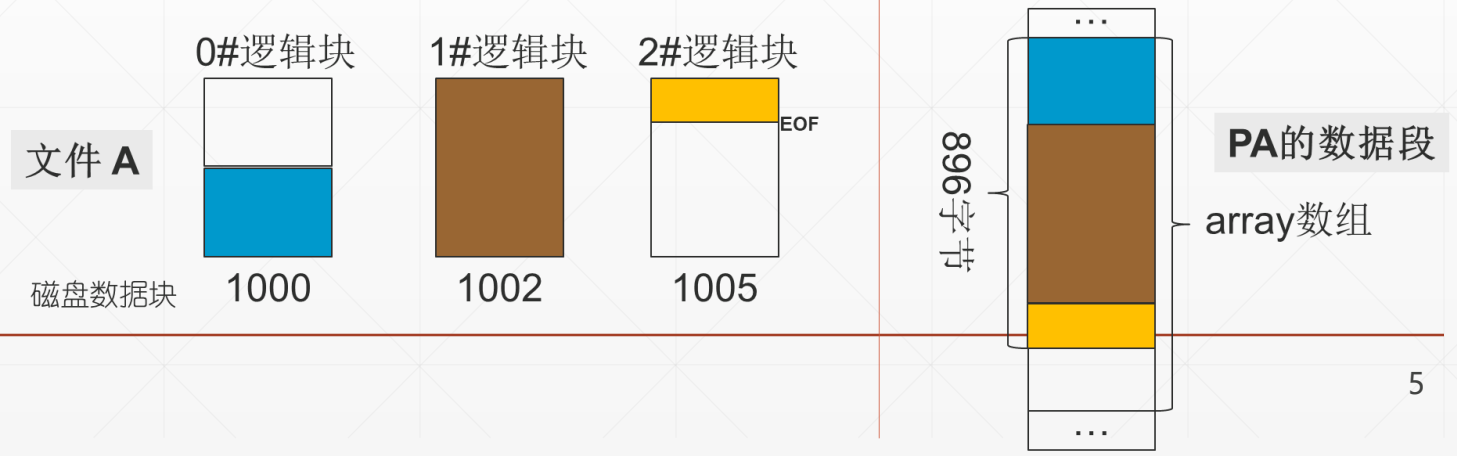
(8) Brelse(bp);

例 1：假设文件A长1152字节，有3个逻辑块：0#块、1#块 512字节，2#块 128字节，分别存放在1000、1002 和 1005#磁盘数据块。系统调用 n=**read(fd,&array,1024)**，从文件偏移量 256 字节 开始连续读1024字节，送数据段 array 数组。假设 read系统调用执行时，i_lastr == -1。试分析read系统调用的执行过程。

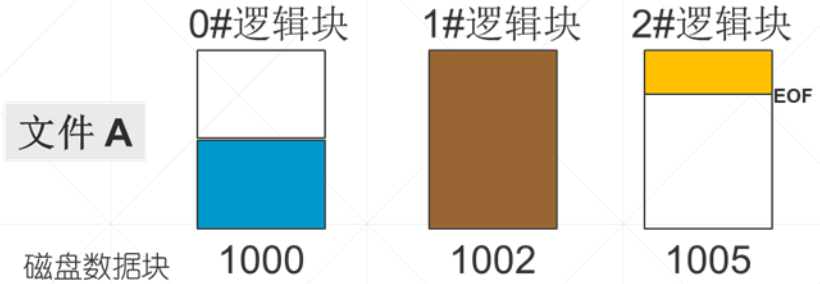
1、初始化user结构中的IO参数：m_offset = 256，m_base = &array，m_count = 1024。

2、循环4轮
前3轮，分别将0#、1# 和 2#数据块中的 256字节、512字节 和128字节送入array数组。
第4轮，文件结束，终止循环。

3、返回读入的字节数 = 256+512+128 = 896字节
array数组1024字节，文件数据只读入896字节。尾部128字节未被读操作覆盖，不是文件数据，是乱码。



i_lastr = = -1
m_offset = 256
m_base = &array
m_count = 1024



每次循环使用的IO参数

逻辑块	m-offset	m-base	m-count	块内偏移量	本块实际读入字节数
0#	256	&array	1024	256	256
1#	512	&array+256	1024-256=768	0	512
2#	1024	&array+768	768-512=256	0	128（文件结束）
	1152 (== i-offset，EOF)		256-128=128		

每次循环使用的缓存读写操作

逻辑块	i_lastr（前	当前块bn	块读写操作	IOmove函数的参数	i_lastr（后
0#	-1	0	breada(0,1000,1002)	bp->b_addr+256，&array，256	0
1#	0	1	breada(0,1002,1005)	bp->b_addr, &array+256，512	1
2#	1	2	breada(0,1005, 0)	bp->b_addr, &array+768，128	2

注：预读块是 0 标识当前块之后没有下一个逻辑块，或查询下个逻辑块的物理块号会引入新的IO操作，不合算。此时，breada会放弃预读。

read系统调用对缓存的使用

逻辑块	i_lastr（前	当前块bn	块读写操作	IOmove函数的参数	i_lastr（后
0#	-1	0	breada(0,1000,1002)	bp->b_addr+256，&array，256	0
1#	0	1	breada(0,1002,1005)	bp->b_addr, &array+256，512	1
2#	1	2	breada(0,1005, 0)	bp->b_addr, &array+768，128	2

- 假设PA执行read系统调用时，所有数据块缓存不命中。
- breada(0,1000,1002)，淘汰自由缓存队列队首的2个缓存，用来装1000#扇区和1002#扇区。放IO请求队列，排一起，睡眠等待1000#扇区IO完成，将数据送入array数组 0~255字节
- breada(0,1002,1005)，淘汰自由缓存队列队首缓存，装1005#扇区，送IO请求队列，breada睡眠，等待复用1002#扇区，置b_wanted标识，被唤醒后将数据送入array数组 256~768字节
- breada(0,1005,0)，预读块是0没有预读操作。等待复用1005#扇区，置b_wanted标识，被唤醒后将最后一块数据送入array数组
- EOF，read系统调用返回。

磁盘中断处理程序：

- 1000#扇区 IO完成，唤醒PA。启动1002#扇区 IO
- 1002#扇区IO完成，异步读缓存块解锁，唤醒等待复用数据的PA。启动1005#扇区 IO
- 1005#扇区IO完成，缓存块解锁，唤醒PA。IO请求队列空，中断处理程序直接返回。

现代操作系统中，预读的作用

- 分配给文件的物理块基本上是相邻的。如果当前块缓存不命中，将 当前块 和 预读块一并送入IO请求队列可以有效减少磁臂移动总距离，减少DMA操作数量，有效提高磁盘工作效率。看示例。
- 预读是提前将预测中未来要用的数据块读入缓存。如果没有明显的收益，放弃。
  - 若预读需要读入索引块、引入新的IO操作，不合算，放弃预读
  - 当前块缓存命中，放弃预读

预读技术的优点

- 减少磁头移动距离，特别有利于改善顺序读时 IO 的平均耗时。
- 例： PA进程顺序读 fileA 的全部内容。与此同时，进程PB需要读取 9999#扇区中另一个文件的内容。假设，所有数据缓存不命中， fileA的0#逻辑块和1#逻辑块分别存放在1000#，1002#物理块。
  - fd = open(fileA,***)
  - read(fd,array,512);
  - read(fd,array,512);

预读，是现代操作系统标配的磁盘性能改善技术

- 使用预读技术，读入当前块1000时，异步读入1002。这2个IO请求一定靠在一起！ 后续读9999#块时，其IO请求不会插在 1000、1002之间，如图：

IO请求队列 …… → 1000,1002 → 9999，读取这3个扇区，磁臂移动8999根磁道。

- 不使用预读技术，读入1000，1002块要执行2次read系统调用。第一次read系统调用进程一定会睡眠放弃CPU，IO完成前，PB一定会上台 IO 9999#扇区 ……，待PA恢复运行读1002时，IO请求队列长这样：
  - IO请求队列 …… → 1000 → 9999 → 1002，读取这3个扇区，磁臂移动 8999 + 8997根磁道。性能远不及预读。

读，为应用程序提供待处理的新数据

预读，是现代操作系统标配的磁盘性能改善技术。
更进一步，在RAM容量可观的系统中，预读的步子可以放得更大，一次读入包含进程所需数据的相邻8个数据块。

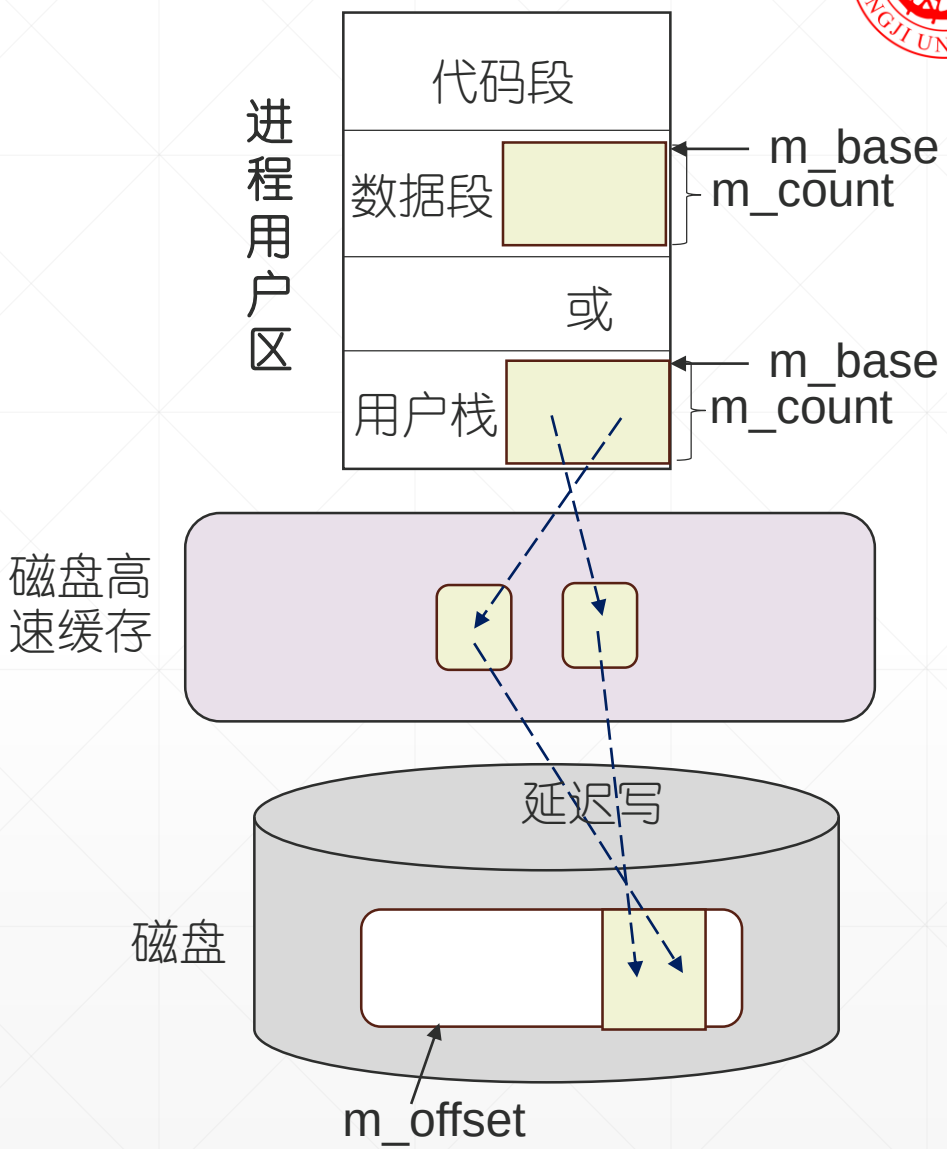
写，持久化应用程序生成的新数据

广泛使用先读后写 和 延迟写技术。

$$n = write(fd, buf, nbytes);$$

write 系统调用将进程用户区中的一块数据异步写入磁盘文件。

- **IO参数，在user结构 的 u_IOParam 字段**
  - **m_offset，数据在文件中的偏移量**
  - **m_base，数据在用户区的首地址**
  - **m_count，需要写入文件的数据量**
- **返回值**
  - **正整数，实际写入的字节数**



write( ) 系统调用框架(Writei 核心部分)

- 循环，一次写一个逻辑块

（1）if (m_count == 0)　　//　（a）所有数据均已写入磁盘

　　返回;

（2）用 m_offset 和 m_count 计算 当前块的逻辑块号bn，块内偏移量offset 和 需要写入本块的字节数 n。bn = m_offset / 512，offset = m_offset % 512，n = min(512 – offset, m_count) 。

（3）地址映射 Bmap(bn)，得 bn 的物理块号 blkno。如果 bn 是新数据块，要先为它分配物理块。

（4）要先读吗？

```
if （ n==512 ）
    bp = GetBlk(dev, blkno);
else
    bp = Bread ( dev，blkno );
```

（5） 将新数据 从用户区 送入 缓存块 bp

```
IOmove(m_base， bp->b_addr+offset，n);
```

（6） 修正IO参数，为写下一个数据块做准备

```
m_offset+=n
m_base+=n
m_count-=n
```

（7） if (m_offset % 512 == 0)

```
Bawrite( bp );  // 启动IO操作。IO完成，磁盘
                中断处理程序释放缓存
else
Bdwrite( bp );  // 打脏标记，释放缓存
```

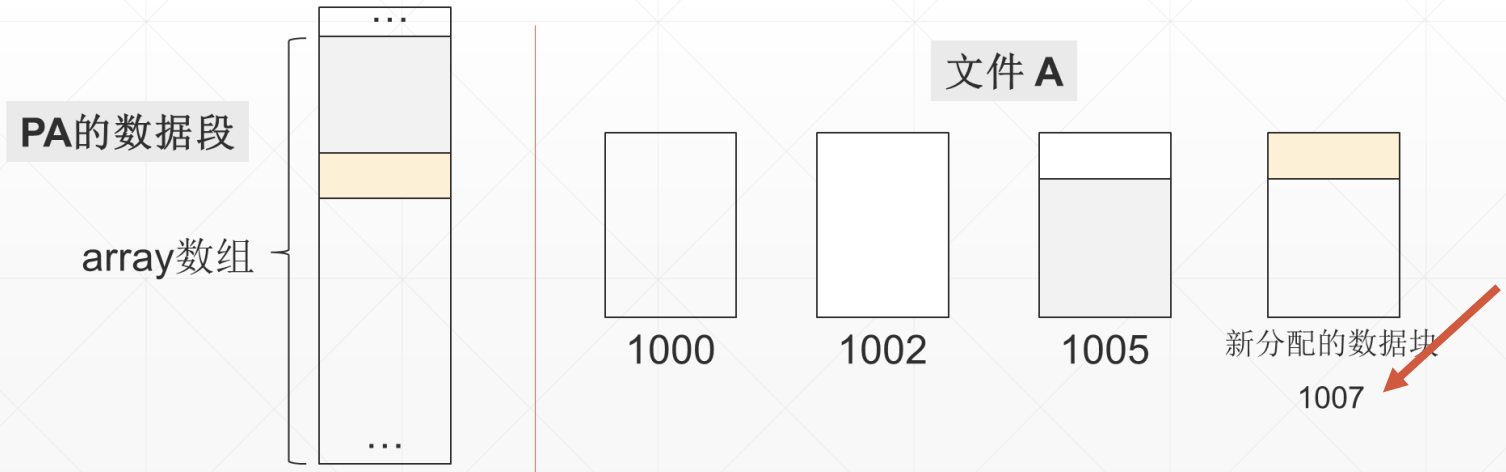
例 2：接例1。进程随后执行系统调用 n=write(fd,&array,512)，将array数组中的前512字节写入文件A的尾部。试分析write系统调用的执行过程。

1、初始化user结构中的IO参数：m_offset = 1152，m_base = &array，m_count = 512。

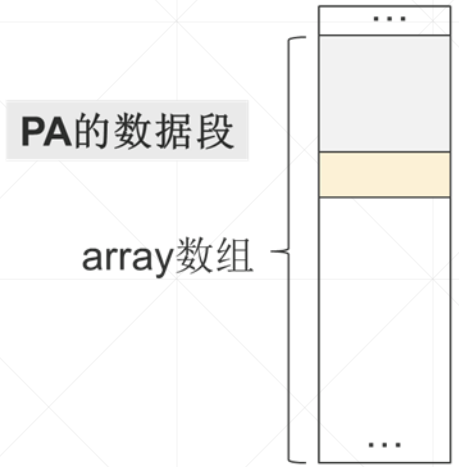
2、循环2轮
第1轮，array数组中的前384个字节写入2#数据块。
第2轮，地址映射表中3#数据块对应的元素为空，这是一个新数据块，先为它分配新物理块 blkno。
向新物理块写入128字节。

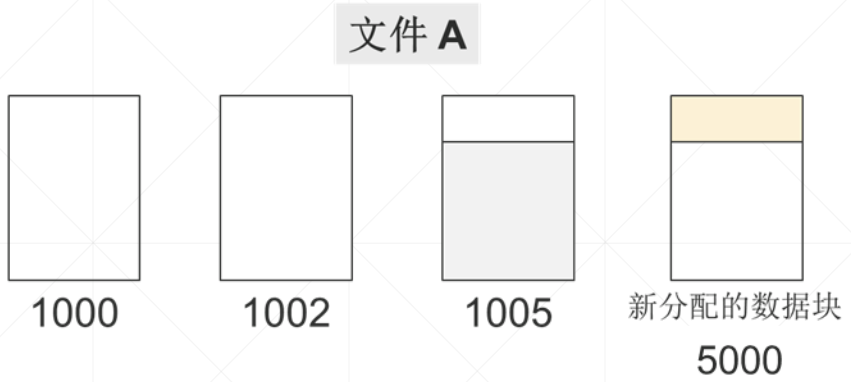
3、……

4、返回实际写入的字节数512字节



m_offset = 1152
m_base = &array
m_count = 512





每次循环使用的IO参数

逻辑块	m-offset	m-base	m-count	块内偏移量	本块实际写入字节数
2#	1152	&array	512	128	384
3#	1536	&array+384	128	0	128
			0，结束		

每次循环使用的缓存读写操作

逻辑块	当前块bn	读操作	IOmove函数的参数	写操作
2#	0	bread(0,1005)	&array，bp->b_addr+128, 384	Bawrite
3#	1	bread(0,5000)	&array+384，bp->b_addr, &128	Bdwrite

注：预读块是0，没有预读操作

Write 系统调用对缓存的使用

逻辑块	当前块bn	读操作	IOmove函数的参数	写操作
2#	0	Bread(0,1005)	&array，bp->b_addr+128, 384	Bawrite
3#	1	Bread(0,1007)	&array+384，bp->b_addr, &128	Bdwrite

- 1005#数据块，缓存命中。写不满，Bread，不睡、锁住这个数据块，IOmove写入384字节后，送IO请求队列末尾。
- 为3#逻辑块分配新数据块1007。写不满，Bread为1007分配缓存块，为磁盘数据块1007构造读IO块，送IO请求队列末尾。
  - IOWait( )，进程PA睡眠等待读操作结束

IOmove写128字节进分配给1007#数据块的缓存块。打脏标记 B_DELWR，解锁1007#数据块。

磁盘中断处理程序：

- 1005#数据块IO完成，解锁。发1007#块的读命令。
- 1007#数据块IO完成，唤醒PA。

Write系统调用，也有这样的一张图。

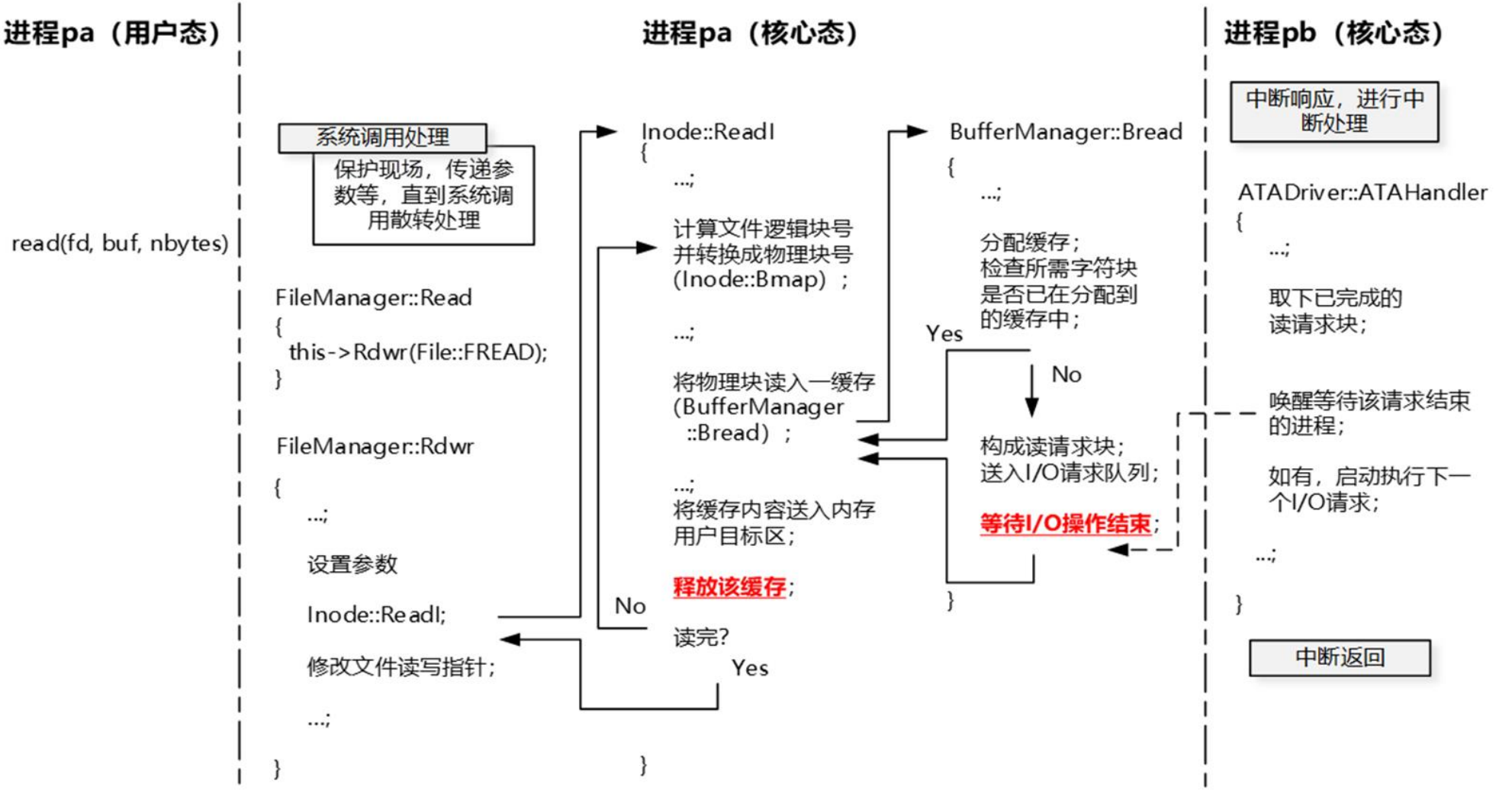


图 7.38：系统调用 read 的基本执行过程