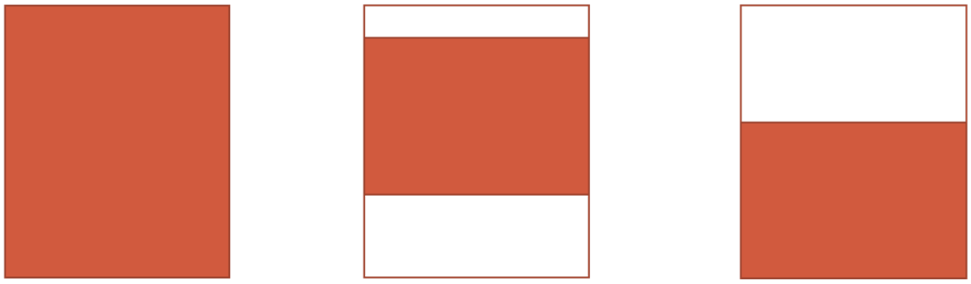
设备管理例题 2

同济大学计算机系操作系统课程作业　　邓蓉

基础题

一、不借助缓存写磁盘，需要先读后写吗？
答：需要。这是因为扇区是磁盘读写的原子单位。写命令会更新整个扇区（512 字节）。所以，当写入的字符量不足 512 字节时，必须先读以保护不会写操作改写的区域。

二、针对下图中的所有情况，写数据块进程是否会入睡，需要执行几次 IO 操作？假设系统缓存队列非空。



答：

（1） 缓存块不命中。需要为数据块分配缓存块，自由缓存队列非空，进程不会睡。

- 子图 (a)，无需先读，不延迟写。一次异步写 io，进程不睡。
- 子图 (b)，需要先读，延迟写。一次同步读 io。进程会睡，等待读操作结束。
- 子图 (c) ，需要先读，不延迟写。两次 IO 操作，一次同步读，一次异步写。进程等待读操作结束时会睡。

（2） 缓存块命中。不需要为数据块分配缓存块，但是需要与其它进程竞争缓存块的使用权，其间进程可能会入睡等待缓存块解锁。

- 子图 (a)，无需先读，不延迟写。一次异步写 io。进程不睡。
- 子图 (b)，需要先读，延迟写。没有 IO 操作。进程不睡。
- 子图 (c) ，需要先读，不延迟写。一次异步写 io。进程不睡。

注：这里的先读指的是执行 Bread( )函数。参看外设管理 3.pptx，第 3 页的伪代码。

三、仅考虑 556#数据块。请问，完成序列中全部的写操作，需要执行几次磁盘 IO 操作？总耗时多少？平均耗时？ 读写序列：
556（写 40#字节），556（写 1#字节），556（写 0~127#字节），600，782，891，900，556（写 128~255#字节），556（写 256~383#字节），556（写 384~511#字节）。

（1）不使用磁盘高速缓存。已知发出 IO 请求至 IO 操作完成，进程平均需要等待 T。
【参考答案】每次写 566#扇区均先读后写。6*2=12 次 IO。总耗时 12*T+6t; 平均耗时 2*T+t。

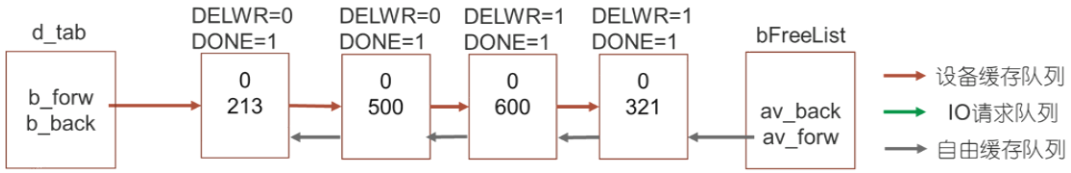
（2）使用高速缓存。已知自由缓存队列中有足够多的干净缓存。将 IO 请求放入 IO 请求队列后至 IO 操作完成，进程平均需要等待 T。从缓存复制数据到用户空间（含上锁、解锁操作），平均耗时 t。
【参考答案】使用缓存，2 次 IO。第一次写 556#块的时候，先读会 IO，最后一次写 556#块的时候，写满了，一次异步写 IO。总耗时 2*T+6t；平均耗时 T/3+t。

四、磁臂调度算法的名称

FIFS，先来先服务
SSTF，最短寻道时间有限
SCAN，电梯 elevator，磁头必需到达最外圈或最内圈磁道才能向相反方向移动。
　　LOOK，磁头到达最远端磁道就可返回。现在默认的电梯就是 LOOK。
C_SCAN，C 是循环 circular。磁头单向移动提供服务，折返时不提供服务快速移动至起始端。

综合题：

一、看图，回答问题



（1）图中为什么没有 IO 请求队列?
答：所有缓存块 B_DONE 是 1，IO 已完成。IO 请求队列是空的。

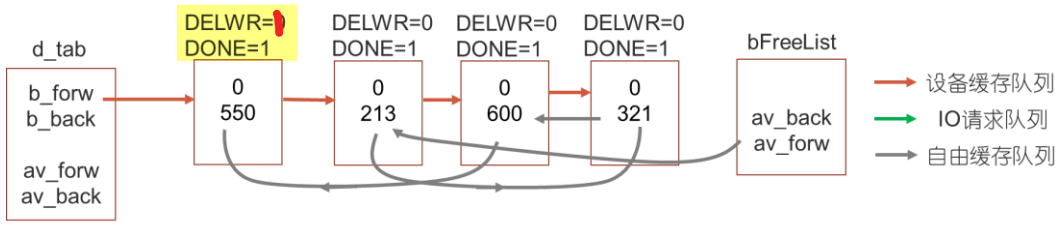
（2）读 213，500，600，321 块。不管是否脏缓存（是否有延迟写标记），进程不会睡，没 IO。

（3）写 213，500，600，321 块。不管是否脏缓存（是否有延迟写标记），进程不会睡。缓存写满了，异步写回磁盘，一次 IO。没写满，没有 IO。

2、3，注意自由缓存队列的变化。GetBlk，锁住某个缓存块之后，会把它从自由缓存队列里抽出来。Brelse 释放缓存块的时候，送自由缓存队列末尾。

（4）写 550#块，偏移量为 8 的字节。
答：缓存不命中。分配自由缓存队列第一个不脏的 Buffer（不带延迟写标记 B_DELWR），将沿途所有脏缓存写回磁盘。321，600 号数据块异步写回磁盘。为 550#数据块分配 500#数据块原先占据的缓存块。先读后写，一次同步读 io。

3个IO请求块送入IO队列（设备缓存队列，队首是新近IO的数据块。脏缓存块，队列中位置不变）



550#块写操作完毕解锁

二、T1 时刻，PA、PB、PC 进程先后访问文件 A，PA read 4#字节，PB write 200#字节，PC read 500#字节。已知文件 A 的 0#逻辑块 存放在 55#扇区。T1 时刻缓存不命中。自由缓存队列不空，所有自由缓存不脏（不带延迟写标识），队首缓存块 Buffer[7]。

（1）请分析如下时刻进程 PA，PB 的调度状态 和 Buffer[i]的使用状态。

- PA 执行 read 系统调用
- PB 执行 write 系统调用
- PC 执行 read 系统调用
- 55#扇区 IO 完成

（3） 题干所有部分不变，PB write 511#字节

【提示】需要一点耐心。关键是这么几点：

- 55#扇区 IO 没有完成前，所有进程执行 sleep(& m_buf[7], -50)入睡。但是，它们继续运行的判断条件是不一样的：PA，IOWait 入睡，等 B_DONE 变 1；另外的 2 个进程，GetBlk 入睡，等 B_BUSY 变 0。IO 完成时，3 个进程都被唤醒了，它们会互斥、串行使用缓存中的数据。PA 先用。所以，万一另 2 个进程先于 PA 上台，它们会再次入睡。PA 读入文件 4#字节之后，PB、PC 串行互斥访问缓存 Buffer[7]中的数据。

探索

N-Step-SCAN，PPT 上表述有错。新请求只要进 io 请求队列，就不能防止饿死现象。

算法性能比较 和 进一步的改进

- SSTF，SCAN和C-SCAN会出现磁臂粘着，IO请求被饿死的情况。
- 磁臂粘着（磁头粘滞）：程序对某些磁道频繁访问，导致 io队列中存在很多针对少数磁道的IO请求，其它磁道的请求被搁置、长期得不到服务的现象。
- 可以防止磁臂粘着的算法

(1) FSCAN（fair）：维护2个IO请求集合，active 和 pending。算法调度active集合中的请求，在该集合变空之前，新请求放入pending集合。active集合变空的时候，交换 active 和 pending指针。 被调度的是固定集合，所以算法不会产生饿死现象。

(2) N-Step-SCAN算法：算法维护一条最大长度为N的io请求队列 和 一条不限长度的io等待队列。算法调度io请求队列中的请求。新进的请求（1）io请求队列 队长小于N时，进io请求队列（2）否则，进io等待队列。io请求队列变空时，算法从等待队列中取最多N个元素，加入io请求队列。 FIFS，无饿死现象；其余，会改善磁盘的io吞吐率。

操作系统 电信学院计算机系 邓蓉 17

以下是查到的有关 N-Step-SCAN 和 FSCAN 的描述：

1、参考文献： 6.8 磁盘存储管理_n-step-scan 算法-CSDN 博客

5.N-step-SCAN算法

N步SCAN算法是将磁盘请求队列分成若干个长度为N的子队列，磁盘调度按照FCFS算法依次处理这些子队列，处理每个子队列用SCAN算法，一个队列处理好了，再处理下一个队列。 当正在处理某子队列时，如果又出现新的磁盘I/O请求，便将新请求进程放入其他队列，这样就可避免出现粘着现象。 当N值取得很大时，会使N步扫描法的性能接近于SCAN算法的性能； 当N=1时， N步SCAN算法便蜕化为FCFS算法。

6.FSCAN算法

FSCAN算法实质上是N步SCAN算法的简化， 即FSCAN只将磁盘请求队列分成两个子队列。一个是由当前所有请求磁盘I/O的进程形成的队列，由磁盘调度按SCAN算法进行处理。在扫描期间，将新出现的所有请求磁盘I/O的进程， 放入另一个等待处理的请求队列。这样，所有的新请求都将被推迟到下一次扫描时处理。

考试写这个。

2、wiki

N-Step-SCAN (also referred to as N-Step LOOK) is a disk scheduling algorithm to determine the motion of the disk's arm and head in servicing read and write requests. It segments the request queue into subqueues of length *N*. Breaking the queue into segments of *N* requests makes service guarantees possible. Subsequent requests entering the request queue won't get pushed into *N* sized subqueues which are already full by the elevator algorithm. As such, starvation is eliminated and guarantees of service within *N* requests is possible.

Another way to look at N-step SCAN is this: A buffer for *N* requests is kept. All the requests in this buffer are serviced in any particular sweep. All the incoming requests in this period are not added to this buffer but are kept up in a separate buffer. When these top *N* requests are serviced, the IO scheduler chooses the next *N* requests and this process continues. This allows for better throughput and avoids starvation.

3、参考文献： N-Step-SCAN disk scheduling - GeeksforGeeks

Algorithm for N-Step-SCAN Disk Scheduling

Step 1: A buffer is created for N requests.

Step 2: All the requests that are kept in this buffer are serviced in any specific wipe.

Step 3: During this time all the new incoming requests can not be added to this buffer, these new requests will be kept in a separate buffer.

Step 4: Now here comes the role of the I/O (Input Output) scheduler because when these top N requests are serviced, I/O (Input Output) scheduler chooses the next N requests and this process goes on and on.

Step 5: By doing this N-Step-SCAN allows better throughput and it is devoid of thrust.

好像没有写透。