# 设备管理习题

姓名 吕博文 学号 2151769

## 第一部分 写操作

一、概念题

1、同步IO

在同步IO操作中，应用程序发起IO请求后，必须等待IO操作完成才能继续执行。

2、异步IO

在异步IO中，应用程序在发起IO请求后可以继续执行其他任务，IO操作完成时，应用程序会收到通知。

3、为什么读是同步的，写是异步的

读操作通常是同步的，因为应用程序需要等待数据读取完成才能继续处理。写操作则可以是异步的，因为数据可以暂时存储在缓存中，而应用程序可以继续执行其他任务。

因为进程要处理从磁盘读入的文件数据。 为什么写操作是异步的？因为写磁盘最终是 DMA 和磁盘控制器硬件做的，它们将进程 产生的新数据持久化存储在磁盘上。进程后续无需，也不能再处理这块数据了。

4、磁盘数据块为什么要先读后写

这是因为在更新磁盘上的部分数据时，必须先读取整个数据块，修改相应部分，然后将整个数据块写回磁盘。

磁盘写操作以数据块为单位，会用内存中的整块数据覆盖磁盘数据块。所以，正确的操 作步骤是（1）先读，确保缓存块中包含磁盘数据块中的旧数据（2）后写，将新数据存入缓 存块（3）硬件（DMA 控制器和磁盘控制器）将既包含新数据，又包含旧数据的缓存块写回 磁盘。如果没有先读操作，内存中的错误信息会覆盖磁盘数据块中的部分数据，导致文件内 容出错。

5、延迟写操作的优点和不足

延迟写可以提高性能，减少磁盘操作次数，但如果在写入前系统崩溃，可能会导致数据丢失。

优点：合并多个写操作，减少写 IO 数量。缩短写操作的平均响应时间，减少磁盘磨损。 缺点：引入文件系统不一致性，增加了丢失数据的风险。

6、Unix系统何时将脏缓存写回磁盘

Unix系统通常在缓存空间不足或者达到一定时间间隔时，将脏缓存（修改过但未写回磁盘的数据）写回磁盘。

（1）数据块写至尾部

（2）LRU 缓存是脏缓存，分配用来装其它数据块之前

（3）关机时所有脏缓存写回磁盘

（4）磁盘卸载时，属于这张磁盘的所有脏缓存写回磁盘

（5）系统定期执行一次 sync 操作，将所有 SuperBlock，脏的 DiskInode 和脏缓存块写回磁 盘。Unix V6++，30s。

二、以下操作引发几次IO？进程会不会睡？不考虑预读。

1. 读磁盘数据块，缓存命中：不会引发IO，进程不会睡眠。

IO：0 次。 数据块被占用时需要入睡等待，其余不需要入睡。

1. 读磁盘数据块，缓存不命中：引发一次IO，进程可能会睡眠等待数据读取。

IO：1 次，读入目标数据块。此外，缓存分配会产生脏缓存刷回磁盘的异步写 IO。 入睡：1 次，等待磁盘读操作结束。此外，自由缓存队列空或没有干净的缓存块时，进 程会因为需要为数据块分配缓存块而入睡。

1. 写磁盘数据块，缓存命中：不会立即引发IO，进程不会睡眠。

写 IO：写至缓存底部，1 次（立即）。未写至缓存底部，0 次（延迟）。 入睡：数据块空闲，不睡。数据块，其它进程在用，会睡。但，不会因为写操作入睡。

1. 写磁盘数据块，缓存不命中：可能不会立即引发IO，进程不会睡眠。

读 IO：需要先读，1 次。此外，缓存分配会产生脏缓存刷回磁盘的异步写 IO。 写 IO：写至缓存底部，1 次（立即）。未写至缓存底部，0 次（延迟）。 入睡：1 次，等待磁盘读操作结束。此外，自由缓存队列空或没有干净的缓存块时，进 程会因为需要为数据块分配缓存块而入睡。

三、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：

调度状态：PA进程发起读操作，由于缓存不命中，需要从磁盘读取数据。此时，PA可能会被置于等待状态，直到所需数据从磁盘读取到缓存中。

Buffer状态：Buffer[7]被分配用于读取55#扇区的数据。

* PB执行write系统调用

调度状态：PB进程发起写操作。由于操作系统采用延迟写策略，写操作一般不会立即导致IO，而是先将数据写入缓存。因此，PB可能不会进入等待状态。

Buffer状态：新的Buffer被分配（假设是Buffer[x]），用于存储PB写入的数据。

* PC执行read系统调用

调度状态：PC进程发起读操作。此时，由于55#扇区的数据已经被读取到Buffer[7]中，如果PC读取的是相同的数据块，则无需再次访问磁盘，PC可以直接从Buffer[7]读取数据；如果是不同的数据块，则可能需要进行另一次磁盘IO。

Buffer状态：如果PC读取的是55#扇区的数据，则使用Buffer[7]；否则，可能需要分配另一个Buffer。

* 55#扇区IO完成

调度状态：一旦55#扇区的数据被读取到Buffer[7]，PA进程可以从等待状态恢复，继续执行。

Buffer状态：Buffer[7]现在包含了55#扇区的数据，可供PA进程读取。

⚫ PA 执行 read 系统调用 ◼ 调度状态。PA 为 55#磁盘数据块申请得到一块缓存，发起读 IO 请求后， sleep(&m\_buf[7], -50)，入睡等待 IO 完成、B\_DONE 变为 1。 ◼ Buffer[7]的使用状态：分配给 55#磁盘数据块，在 IO 请求队列。B\_BUSY 为 1， B\_DONE 为 0。

⚫ PB 执行 write 系统调用 ◼ 调度状态。PB 等待复用 55#数据块，sleep(&m\_buf[7],-50)，入睡等待缓存块解锁、 B\_BUSY 变为 0。 ◼ Buffer[7]的使用状态：分配给 55#磁盘数据块，在 IO 请求队列。B\_BUSY 为 1， B\_DONE 为 0，B\_WANTED 为 1。

⚫ PC 执行 read 系统调用 ◼ 同 PB。

⚫ 55#扇区 IO 完成 调度：中断处理程序执行 IODone( )函数，m\_buf[7] B\_DONE=1 置 1，唤醒 PA，PB 和 PC。 Buffer[7] 的使用状态：上锁(B\_BUSY==1)，不在 IO 请求队列，在设备缓存队列，不在 自由缓存队列；数据可用（B\_DONE==1）。进程 PA 持锁。有进程等待使用其中的数据 （B\_WANTED==1）。 同济大学计算机系操作系统课程作业 邓蓉 2023-12-31 调度：PA 先用缓存块中的数据，执行 IOmove 将 4#字节复制进用户空间。解锁缓存 （B\_BUSY=0，B\_WANTED=0），送自由缓存队列队尾，唤醒 PB 和 PC。PB 和 PC 依次上台， 分别执行 write 系统调用和 read 系统调用的下半段，写、读缓存块中偏移量依次为 200 和 500 的字节。谁先执行，不得而知，对最终结果没有影响。 Buffer[7] 的最终状态：B\_DELWR==1，B\_DONE==1，B\_BUSY==0，是一块自由的脏缓 存。

2、题干所有部分不变，PB write 511#字节。问题2与问题1独立。

调度状态：类似问题1中的分析，PB进程可能不会立即进入等待状态，因为写操作首先会写入缓存。

Buffer状态：根据写入的数据量和逻辑块的大小，可能需要一个或多个Buffer来存储PB写入的数据

和第一小题一致。唯一的不同在于，本小题 PB 写至数据块底部，55#数据块需要异步 写回磁盘。若其先于 PC 执行，PC 会再次入睡，等待中断处理程序解锁 55#数据块。 Buffer[7] 的最终状态：B\_DELWR==0，B\_DONE==1，B\_BUSY==0，是一块自由的不脏 的缓存。

四、一个磁盘组有100 个柱面，每个柱面有8 个磁道（磁道号=磁头号），每根磁道8 个扇区，每个扇区512字节。现有含6400 个记录的文件，每条记录512字节。文件从0 柱面、0 磁道、0 扇区顺序存放。

1. 若同根磁道，相邻磁盘数据块存放在相邻物理扇区（现代硬盘，磁盘数据缓存的尺寸：整根磁道）

（1）3680#记录存放的位置。 柱面号=？，磁道号=？，扇区号=？(注意到记录也是从0#开始编号)

柱面号：57

磁道号：3

扇区号：7

（2）78#柱面，6#磁道，6#扇区存放该文件的第几个记录？

5047

柱面，从最外圈向内递增编号。从 0 开始，先编号 n 号柱面的所有磁道，后编号 n+1 号柱 面的所有磁道。属于同一柱面的多个磁道，按磁头号递增对磁道依次编号。一根磁道，所有 扇区递增编号，之后编号下一个扇区。

（1）每个柱面 8\*8 = 64 个扇区。 3680/64 = 57 柱面号是 57。 3680 % 64 = 32 这个记录存放在 57#柱面的 32#扇区。 每根磁道 8 个扇区。 32 / 8 = 4 32#扇区在 4#磁道（4#读写头管理的那个磁道） 32 % 8 = 0 是 4#磁道的 0#扇区。 3680#记录存放的位置是： 57#柱面，4#磁道，0#扇区。

（2）78\*64+6\*8+6 = 5046。

1. 若同根磁道，相邻磁盘数据块错开一个物理扇区（老式硬盘，磁盘数据缓存的尺寸：一个扇区）

（1）3680#记录存放的位置。 柱面号=？，磁道号=？，扇区号=？

柱面号：57

磁道号：3 (4)

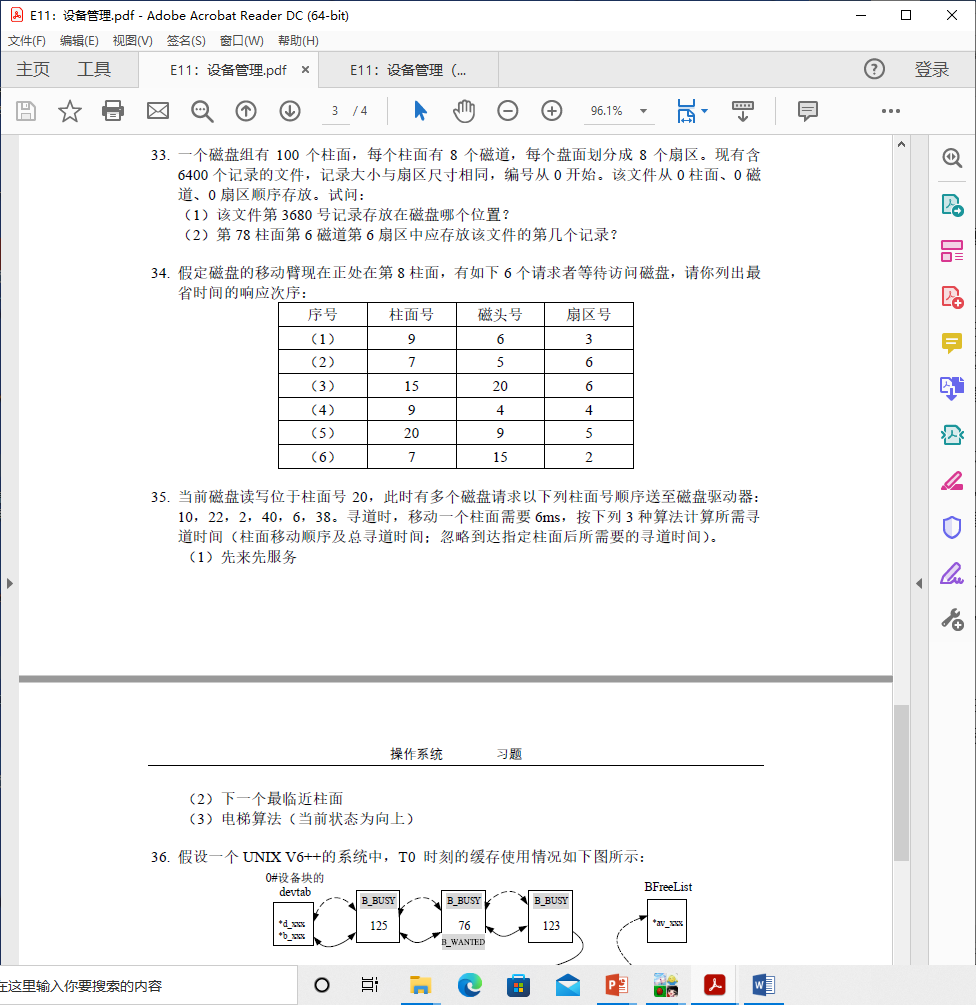
扇区号：7 (0)

（2）78#柱面，6#磁道，6#扇区存放该文件的第几个记录？

5044(5043#)

五、假定磁盘的移动臂现在正处在第8 柱面，有如下6 个请求者等待访问磁盘。

假设寻道时间>>旋转延迟。请列出最省时间的响应次序：



响应次序：7、7,、9、9、15、20

六、当前磁盘读写位于柱面号20，此时有多个磁盘请求以下列柱面号顺序送至磁盘驱动器：10，22，2，40，6，38。寻道时，移动一个柱面需要6ms。

1、按下列3 种算法计算所需寻道时间，画磁头移动轨迹。

（1）先来先服务

寻道时间：876ms

磁头移动轨迹：20 → 10 → 22 → 2 → 40 → 6 → 38。

（2）SSTF（下一个最临近柱面）

寻道时间：360ms

磁头移动轨迹：20 → 22 → 10 → 6 → 2 → 38 → 40

（3）电梯调度算法（当前状态为向上）

寻道时间：348ms

磁头移动轨迹：20 → 22 → 38 → 40 → 10 → 6 → 2

2、为什么电梯调度算法（磁头 向最远请求磁道距离目前磁头位置最近的方向 移动）性能优于另两种算法？

减少寻道时间：电梯调度算法通过在一个方向上连续处理所有请求，减少了磁头在柱面之间频繁来回移动的次数，从而降低了总寻道时间。

公平性：与SSTF相比，SCAN算法避免了“饥饿”现象，即某些磁盘请求因为一直不是最近的请求而长时间得不到服务。

预测性：电梯算法的移动路径更加可预测，有助于优化磁头移动策略和提高整体性能。

电梯算法的主要优点是它为系统提供了更平衡和可预测的性能表现，特别是在请求密集的环境中。

七、外设的独占和共享

1. 从硬件驱动的角度，所有外设必须独占使用。为什么？

从硬件驱动的角度来看，所有外设必须独占使用，原因包括：

资源冲突：多个进程或线程同时访问同一个外设可能会导致资源冲突，比如数据覆盖或损坏。

一致性维护：独占使用确保了对外设的操作可以顺序地、一致地进行，避免了并发访问带来的同步问题。

性能考虑：对于某些外设，比如打印机或特定类型的传感器，同时处理多个请求可能会大大降低效率或无法正常工作。

外设芯片中的数据缓存，命令、状态、数据寄存器只能存放一个请求的 IO 参 数和 IO 数据。另外，对外设而言（以读操作为例），一次典型的 IO 操作包括：CPU 发 IO 命令，外设执行 IO 命令，数据寄存器中的新数据读入内存。全部完成后，CPU 才可以向 外设发下一个 IO 命令。综上，从硬件驱动的角度，所有外设必须独占使用，没有例外。

2、多道系统，硬盘是共享设备。并发执行的多个任务可以同时访问硬盘。

内核引入了（**磁盘高速缓存**）填内核数据结构 将必须独占使用的物理硬盘改造成逻辑上的共享设备。

3、打印机是必需独占使用的外设。并发2个打印任务，两个输出内容交织在一起，打印结果是不可用的。Spooling技术借助硬盘这个共享设备，将原先必须独占使用的打印机改造成能够同时为多个用户提供打印服务的共享设备。思路是：

* 系统维护一个FIFS的打印队列。
* 进程需要打印时，
  + 新建一个临时磁盘文件，命名之。把要打印的内容写入这个文件。
  + 生成一个打印作业控制块，包含进程ID，用户ID，临时文件名……，送打印队列尾。
  + 进程返回。无需等待打印IO完成。
* 打印机完成当前打印任务后，取打印队列队首打印作业控制块，构造针对打印机的新的IO命令。

PS1：相对打印机，磁盘是很快的设备。所以，Spooling技术同时也改善了打印机这个IO子系统的响应速度。Spooling是个很不错的IO优化技术，对照期末自己在打印店看到的现象，体会下Spooling技术。

PS2：教科书上 Spooling技术相关的，有输入井和输出井这两个概念。打印机的输出井就是一个文件夹，用来存放临时文件（上面高亮的那个）。

**解：**

Spooling技术用于将原本需要独占使用的打印机改造成能够同时为多个用户提供打印服务的共享设备。其实现思路如下：

维护打印队列：系统维护一个先入先出（FIFO）的打印队列。

进程打印请求：

进程创建一个临时磁盘文件并写入要打印的内容。

生成一个打印作业控制块，包含进程ID、用户ID、临时文件名等信息，然后将其加入到打印队列尾部。

进程可以在不等待打印IO完成的情况下返回继续执行。

打印任务执行：打印机完成当前任务后，取出打印队列队首的打印作业控制块，并根据其中信息构造针对打印机的新的IO命令。

PS1：由于磁盘的速度相对于打印机来说很快，Spooling技术同时也提高了打印机这个IO子系统的响应速度，是一种有效的IO优化技术。

PS2：在教科书上，Spooling技术涉及到输入井和输出井的概念。对于打印机来说，输出井可以是一个用来存放临时文件的文件夹。

总结来说，Spooling技术通过使用硬盘这个共享设备来间接实现对原本需要独占使用的外设（如打印机）的共享访问，有效提高了资源利用率和系统的整体性能。

八、证明题 选做（1）io请求集合固定时，第一步磁头向 最远端请求磁道距离磁头当前所在磁道距离最远的方向移动。这个版本的电梯算法理论最优（2）SSTF不是最优。

**解：**

（1）电梯调度算法（SCAN）的一个变体是，在一组IO请求集合固定的情况下，首先将磁头移动到最远端请求的方向。这种做法可以被认为是理论上最优的，原因如下：

最小化磁头的反向移动：首先移动到最远端的请求意味着磁头在处理完所有请求之前不需要改变移动方向。这种一次性的遍历最大化了磁头的顺序移动，并最小化了寻道时间。

预防“饥饿”现象：确保磁头访问最远端的请求意味着所有请求都将被公平地处理，没有一个请求会被长时间忽略。

固定的请求集合：在请求集合固定的情况下，电梯算法能够确保在最短的总寻道时间内完成所有请求，因为它按照一定方向顺序处理每个请求，避免了不必要的磁头移动。​

（2）SSTF（最短寻道时间优先）算法选择距当前磁头位置最近的请求进行服务。这个算法虽然在短期内看起来有效，但并非总是最优的，主要原因是：

可能引发“饥饿”现象：如果有一系列距离磁头当前位置较近的请求持续出现，那么距离较远的请求可能长时间得不到服务。这种情况在高负载下尤其明显。

缺乏全局优化：SSTF只考虑了磁头到最近请求的距离，而没有考虑整体的寻道路径。在某些情况下，优先处理最近的请求可能导致磁头在几个位置之间来回移动，增加了总寻道时间。​

证明：磁道 min 和 max 处有 IO 请求。无论是怎样的响应序列，磁头一定要经过这两根 磁道。 同济大学计算机系操作系统课程作业 邓蓉 2023-12-31 第一步，仅考虑 curr、min、max 这 3 根磁道。从 curr 出发，经过 min 和 max，磁头最短 移动距离 = distance[min , max] + min(distance[min , curr], distance[curr , max]) 磁头移动轨迹：curr，离 curr 较近的一端 a，离 curr 较远的一端 b。 磁头在从 a 移动至 b 的过程中，服务全部 IO 请求。 这是电梯调度算法给出的解，命名为 bestSequence。 第二步：任选 IO 请求序列中的一个 IO 请求 x，插入 bestSequence。得到的响应序列长度 不小于第一步产生的响应序列。证明如下： x 的插入位置有 3 处：

⚫ a 之前 curr x a b curr，a 两点间直线距离最短： distance[curr，a]<= distance[curr，x]+ distance[x, y]。 途径 x 到 a，轨迹长度增加 distance[x , a]。

⚫ a ，b 之间 curr a x b 轨迹长度不变。

⚫ b 之后 curr a b x 轨迹长度增加 distance[x , b]。 第三步：任选 IO 请求序列中的一个 IO 请求 y，插入从第二步衍生出的任意长度响应序 列。得到的响应序列运动轨迹长度不小于已有响应序列。证明方法与第二步相同。 依次实施第一步、第二步，之后递归实施第三步可以构造出任意 IO 请求响应序列。由 上述证明可知，生成的 IO 请求响应序列磁头运动轨迹长度不会小于第一步给出的 bestSequence。电梯调度算法给出的解理论最优。 证明有点长，应该还会有简洁的、更好的证明。