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。

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。

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。

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。

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。

2、 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。

3、 写磁盘数据块，缓存命中：不会立即引发 IO，进程不会睡眠。

4、 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。

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。

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

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=？

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

2、 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
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？

5044

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:

序号	柱面号	磁头号	扇区号
(1)	9	6	3
(2)	7	5	6
(3)	15	20	6
(4)	9	4	4
(5)	20	9	5
(6)	7	15	2

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。

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（**设备驱动程序和文件系统**）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。