

设备管理习题

姓名 郑博远学号 2154312

第一部分 写操作

一、概念题

1、同步 IO

同步 IO，即发出 I/O 请求后进程入睡。等待 I/O 操作完成后唤醒入睡的进程（被内核唤醒），进程方才继续执行后续代码。同步 IO 操作期间，进程阻塞，不能执行其他任务。

2、异步 IO

异步 IO，即 I/O 请求发出后进程不入睡，继续执行后续代码。进程使用异步方式，可以启动多个并行的 IO 操作，可以在等待 IO 传输过程中执行计算任务。异步 IO，是高性能计算利器，但需要进程（应用程序）主动查询 IO 操作状态。

3、为什么读是同步的，写是异步的

因为读操作需要磁盘上的数据，所以必须入睡等待 I/O 从磁盘读入数据之后，再执行 IOMove 将磁盘缓存数据搬入用户空间，以供后续使用；反观写操作，最终是 DMA 和磁盘控制器硬件做的，将进程产生的新数据持久化存储在磁盘上。用户程序不需要等待写入磁盘后再进行额外的操作，因此是异步写。

4、磁盘数据块为什么要先读后写

先读后写发生在写入数据的大小小于缓存块大小的情况下。这是为了保护磁盘数据块中未被新数据覆盖的旧数据，否则这些旧数据可能与磁盘中不同，在写回磁盘时产生错误。

p.s. 步骤：

（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。

5、延迟写操作的优点和不足

优点：减少写回的 I/O 操作，多次写操作合并，提升写操作的平均响应时间，减少磁盘磨损；

不足：破坏了文件系统的一致性，可能导致数据丢失。

6、Unix 系统何时将脏缓存写回磁盘

1. 当写操作写到了缓存块末尾时；

2. 当 LRU 替换选中的缓存块有脏标识时；

3. 卸载 U 盘时，将所有脏缓存写回磁盘；

4. 关机将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。在 Unix V6++ 中是 30s 一次。

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

0 次 IO，进程不入睡。(如果数据块其他进程在用，会睡!)

2、读磁盘数据块，缓存不命中

1 或 2 次 IO，进程入睡。之所以有可能是 2 次，是因为重新分配的缓存块有可能是带有脏标记的缓存，需要写回，额外进行一次 IO。

入睡原因：其一是等待磁盘读操作结束；其二是自由缓存队列为空/没有干净的缓存块，等待分配而入睡。

3、写磁盘数据块，缓存命中

可能是 0、1 次，进程不入睡(还是一样的，如果其他进程在用还是有可能入睡的；但不会因为写操作本身而入睡)。若写数据尺寸小于缓存块尺寸，则需要先读后写，但因为命

中不需要 IO；若写到了缓存块末尾，则需要异步写回，需要一次 IO。

4、写磁盘数据块，缓存不命中

可能是 0、1、2 次 (还有可能缓存分配会产生脏缓存刷回磁盘的异步写 IO)，进程可能入睡也可能不入睡。若写数据尺寸小于缓存块尺寸，则需要先读后写，一次同步读 IO，进程入睡。也可能因为没有自由缓存块队列为空或没有干净的自由缓存块而入睡。若写到了缓存块末尾，则需要异步写回，需要一次 IO。

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

PA 执行 GetBlk 不命中，分配自由缓存队列中的队首缓存块 Buffer[7]。此后执行同步读 IO，PA 进程 IOWait 入睡 sleep(&m_buf[7], -50)，等待唤醒后 B_DONE 为 1 时执行系统调用后半段。

Buffer[7]的使用情况：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

PB 执行 GetBlk 命中，但由于 PA 的 GetBlk 将 B_BUSY 置 1，等待复用 55#数据块，sleep(&m_buf[7], -50)，PB 置 B_WANTED 后入睡。

Buffer[7]的使用情况：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

PC 执行 GetBlk 命中，但由于 PA 的 GetBlk 将 B_BUSY 置 1，PC 置 B_WANTED 后入睡。(同 PB)

- 55#扇区 IO 完成

IO 完成后，会唤醒所有执行 sleep(&m_buf[7], -50) 入睡的进程 PA、PB 和 PC。

PA 发现 B_DONE, 因此上台继续执行系统调用后半段。PB、PC 也会被唤醒, 但若他们先上台会由于 B_BUSY 为 1 会继续入睡。直到 PA 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。进行 B_relse 后清空 B_BUSY 标志, 将 PB、PC 进程唤醒, 此后二者串行互斥运行 (二者先后未可知)。PB 未写到缓存块末尾, 延迟写置脏标记; PC 读命中。二者都无需进行 IO 操作。

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。

其他部分变化不大。重点在于, 由于写到了缓存块尾部, PB 需要进行异步写回。因此若 PB 先上台, 此后 PC 上台则需要等待写 IO 完成后中断处理程序将 B_BUSY 清空。

Buffer[7] 的最终状态: B_DELWR==0, B_DONE==1, B_BUSY==0, 是一块自由的干净缓存。

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)

扇区大小==记录大小, 即每条记录对应一个扇区。

(1) 3680#记录存放的位置。柱面号=?, 磁道号=?, 扇区号=?

柱面号: $3680 / 8 / 8 = 57$

磁道号: $(3680 / 8) \% 8 = 4$

扇区号: $3680 \% 8 = 0$

(2) 78#柱面, 6#磁道, 6#扇区存放该文件的第几个记录?

$(78 * 8 + 6) * 8 + 6 = 5046$

答案正确:)

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=?

扇区号正好是 0, 所以没差

(2) 78#柱面, 6#磁道, 6#扇区存放该文件的第几个记录?

(这里的扇区号应该是物理扇区号, 物理上相邻编号)

错位后数据块的存放顺序是 0、4、1、5、2、6、3、7

6#扇区存放的是第 3 个逻辑扇区号的数据块

因此应该是: $(78 * 8 + 6) * 8 + 3 = 5043$

答案正确:)

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

由于寻道时间>>旋转延迟, 因此以柱面号为主要考察依据, (可以证明选择最远端最近
的一侧为初始方向的电梯算法有最优移动距离) 因此有:

(6) (2) (4) (1) (3) (5)

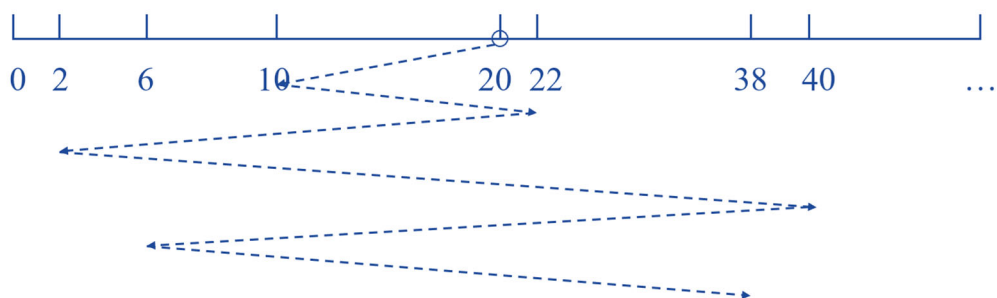
若不考虑扇区号, 下划线部分应该可以交换。 答案正确:)

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

$$(10 + 12 + 20 + 38 + 34 + 32) * 6 = 876 \text{ ms}$$

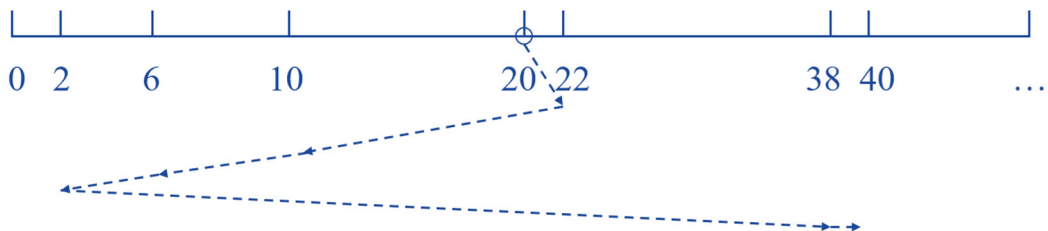


答案正确:)

(2) SSTF (下一个最临近柱面)

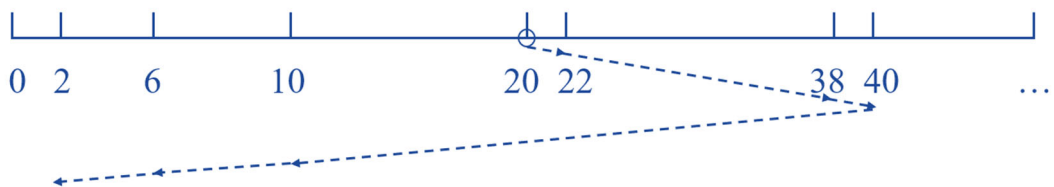
$$(2 + 12 + 4 + 4 + 36 + 2) * 6 = 360 \text{ ms}$$

答案正确:)



(3) 电梯调度算法 (当前状态为向上)

$$(2 + 16 + 2 + 30 + 4 + 4) * 6 = 348 \text{ ms} \quad \text{答案正确:)}$$



3、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?

因为 FIFO 或 SSTF 的磁头都会来回折返, 造成往复的运动浪费, 而电梯调度不会。

(磁臂移动轨迹, 折返距离短)

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?

如果不同的进程同时共享使用外设, 那么多个进程轮番调度上台时可能涉及到多次的数据读/写, 导致数据不一致, 外设状态也难以维护。

标签: 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 (IO 请求队列、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)。

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

题目表述应该错了, 应是“最远端请求磁道距离磁头当前所在磁道距离最近的方向移

动”的电梯理论算法最优。

参考答案如下：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。

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：

1. 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, a]

途径 x 到 a，轨迹长度增加 distance[x, a]。

2. a, b 之间： curr a x b

轨迹长度不变。

3. b 之后： curr a b x

轨迹长度增加 distance[x, b]。

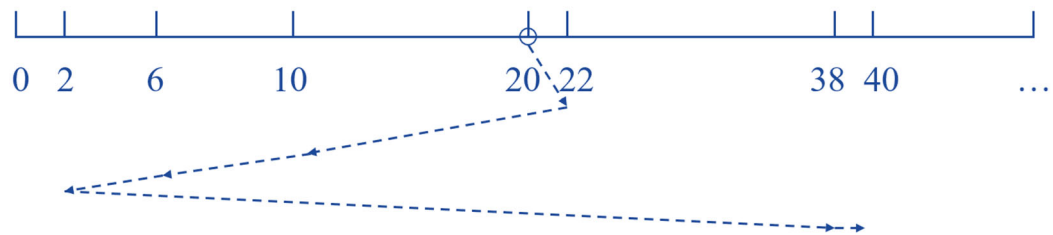
第三步：任选 IO 请求序列中的一个 IO 请求 y，插入从第二步衍生出的任意长度响应序列。得到的响应序列运动轨迹长度不小于已有响应序列。证明方法与第二步相同。

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。

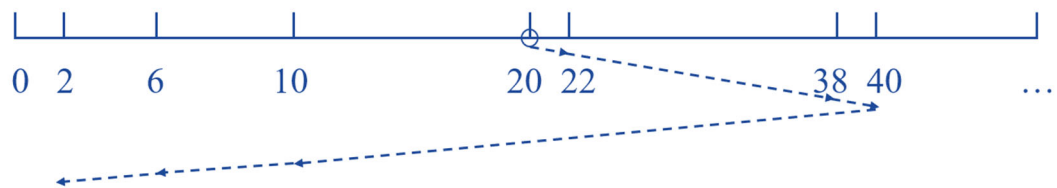
(2) SSTF 不是最优。

上面的题目六中即是反例。

SSTF:



电梯算法:



电梯算法更优，SSTF 不是最优。