操作系统复习提纲

操作系统概述

- 操作系统的概念、特征、功能和提供的服务
  - 概念
  - 特征：并发性、共享性、异步性、虚拟性
  - 功能：资源管理（处理器、存储、设备、文件、联网通信）、控制执行、提供接口
  - 提供的服务：编写和执行程序、数据 I/O 和信息存取、进程通信、差错检测和处理、资源管理、统计、保护等
- 操作系统的发展与分类
  - 多道批处理系统的特点：提高资源利用率和系统吞吐率，但牺牲了用户的响应时间
  - 批处理系统和分时操作系统的区别？追求目标、适应作业、资源利用率
    - 批处理系统：以提高系统资源利用率和作业吞吐率为目标；适应调试好的大型作业；可以合理安排不同负载的作业，使资源利用率达到最佳
    - 分时系统：强调公平性，对于联机用户的立即型命令要快速响应；适应正在调试的小型作业；让多个终端作业使用相同类型的编译系统、运行系统时系统调度开销较小，能够公平地调配 CPU 和内存资源
- 操作系统的运行环境
  - 内核态与用户态
    -
  - 中断、异常
    - （外）中断：时钟中断、控制台中断、输入输出中断、电源故障中断……；来自处理器之外的中断信号；高级中断会屏蔽低级中断
    - 内中断（异常）：主存奇偶校验错、非法操作码、地址越界、缺页、调试、访管、算术溢出……；大部分异常发生在用户态，只有缺页异常发生在内核态；不能被屏蔽
  - 系统调用
    - API，库函数与系统调用？
- 操作系统的体系结构：单体式结构、层次式结构、虚拟机结构、微内核结构

进程管理

- 进程与线程
  - 进程概念：进程(process)是一个可并发执行的具有独立功能的程序关于某个数据集合的一次执行过程，也是操作系统进行资源分配和保护的基本单位。
  - 进程状态与切换：（手写补充）
    - 三态模型：
    - 五态模型
    - 具有挂起状态的状态转换模型
  - 进程控制：FCB，有进程的标识信息、现场信息、控制信息
  - 进程组织：
    - 进程创建：fork（派生、父子进程关系）、vfork（父子进程共享内存空间）、clone

（克隆、对等关系）
    - 进程撤销：exit
    - 进程阻塞和唤醒：wait、waitpid、sleep、write、read
    - 进程挂起和激活：
    - 进程切换≠模式切换，模式切换是用户态和核心态切换，进程切换在核心态完成。进程切换之前一定要有模式切换，模式切换不一定导致进程切换
- 进程通信：实质上是进程中线程之间的通信
    - 信号通信（模拟硬中断，只能传送信号不能传送数据）
    - 管道通信(连接读写进程的一个特殊文件,单向发送,只在一个进程族中使用)
    - 消息传递通信（端口）（原语：send、receive，可以实现进程互斥和进程同步问题）
    - 信号量通信
    - 共享内存通信（两个进程有一块相同的内存区域）
    - 套接字网络进程通信
- 线程的概念与多线程模型
    - 操作系统中能够独立执行的实体，是处理器调度和分配的基本单位
    - 内核级线程（内核空间多线程）、用户级线程（用户空间线程库）、混合式线程（前两者均有）

处理器调度

- 调度的基本准则
    - 资源利用率：CPU 有效工作时间/CPU 总运行时间
    - 响应时间（分时系统、实时系统）：作业提交->得到响应时间
    - 周转时间（批处理系统）：作业提交->作业完成时间（作业在系统中的等待时间+运行时间）
    - 平均周转时间：周转时间/作业数量，衡量不同算法对同一作业流的调度性能
    - 带权周转时间：Σ(每个作业的周转时间/每个作业所需运行时间)/作业数量，衡量同一算法对不同作业流的调度性能
- 调度方式
    - 高级调度：作业管理（位于启动和结束的时候）
    - 中级调度：决定作业（进程）进入内存，并完成内存、外存之间的进程对换工作
    - 低级调度：决定作业（进程、线程）占用处理器
- 典型调度算法：
    - 高级调度算法：（作业调度算法）
        - FCFS：先来先服务
        - SJF：最短作业优先，选运行时间最短的
        - SRTF：最短剩余时间优先，选预测剩余运行时间最短的
        - HRRF：最高响应比优先，选响应比=（作业响应时间或者作业周转时间）/作业处理时间=1+作业等待时间/作业处理时间最短的
        - 优先级调度算法：
        - MLFQ：多级反馈队列调度算法
    - 低级调度算法：
        - FCFS

        - RR：时间片轮转调度算法
        - 优先数调度算法（Priority Scheduling）：
        - 动态优先数调度算法
        - MLFQ：多级反馈队列调度
        - 保证调度算法：不停转向实际获得 CPU/应获得 CPU 时间比最小的进程执行
        - 彩票调度算法：彩票代表一个获得调度的可能性，进程持有彩票越多，调度可能性越高
    - 优先数、优先权、优先级题目自会有规定！务必按照题目的意思做题。
- 同步与互斥
    - 同步：为完成共同任务的并发进程基于某个条件来协调其活动而产生的写作制约关系
    - 互斥：若干进程因相互争夺独占型资源而产生的竞争制约关系
    - 实现临界区互斥的基本方法：
        - 软件方法：
            - Dekker 算法：
            - Peterson 算法：忙等待，任意进程进入临界区的条件是对方不在临界区（inside[i]=1 就在，否则不在）或者对方不想进入临界区(turn 轮到谁)
        - 硬件实现方法：关中断、测试并设置指令、对换指令 XCHG
    - 信号量
        - 公有信号量：互斥；私有信号量：同步。例如生产者和消费者要同步，生产者自己之间以及消费者自己之间要互斥
        - P(s)：信号量 value 值-1，若结果<0，执行 P 操作的进程阻塞，排入 s 队列；≥0，执行 P 操作的进程继续执行。
        - V(s)：信号量 value 值+1，若结果≤0，s 队列释放一进程转就绪态，自己继续执行；>0，执行 V 操作的进程继续执行。
        - 既有同步又有互斥信号量的情况下，原则上，用于互斥信号量上的 P 操作总在后面执行
        - mutex 一般意思是用在互斥信号量上面，即同一类功能的东东和自己互斥
    - 管程
        - 条件变量：进程可以在这个条件变量上等待或唤醒
        - wait 原语：挂起调用进程并释放管程，直至另一个进程在条件变量上执行 signal
        - signal 原语：若有其它进程因对条件变量执行 wait 而挂起，那么释放之，否则相当于空操作

```
type  管程名=monitor{
    <局部变量声明>
    cond <条件变量声明>
    Interface Module IM;
    define <管程内定义，管程外可用的过程列表>
    use enter, leave, wait, signal<管程外定义，管程内可用的过程列表>
}
procedure  过程 1{
    过程体
}
```

procedure 过程 2{

过程体

}

- 经典同步问题
  - 生产者-消费者问题（单生产者单消费者？多类产品的单生产者单消费者？多类产品的多生产者多消费者？）
  - 读者-写者问题（读者优先？写者优先？读写近似公平？）
  - 哲学家进餐问题、理发师问题，etc.
- 死锁
  - 死锁的概念: 如果一个进程集合中的每个进程都在等待只能由此集合中的其它进程才能引发的事件，而无限期陷入僵持的局面称为死锁
  - 死锁条件
    - 互斥条件：让进程可同时访问
    - 占有和等待条件：使用静态分配，但是严重降低资源利用率
    - 不剥夺条件:
      - 法一：申请新资源必放弃旧的;
      - 法二：资源管理程序有则分配，无则剥夺它所有资源
    - 循环等待条件：层次分配，系统资源在不同层次，得到只能申请更高，释放必须将更高的全释放
  - 死锁避免
    - 系统安全状态：要把安全序列写出来!!!
    - 银行家算法
      - 系统每类资源的总数——m 个元素的向量，每个分量对应一类资源的总数：Resource = (R1, R2, ···, Rm);
      - 每类资源未分配数量——m 个元素的向量，每个分量对应一类资源尚可分配的数量：Available = (V1, V2, ···, Vm);
      - 最大需求矩阵——n*m，行代表 n 个进程，列代表 m 类资源，Claim，其中元素 Cij 表示进程 Pi 需要 Rj 类资源最大数;
      - 分配矩阵——n*m，行代表 n 个进程，列代表 m 类资源，Allocation，其中元素 Aij 表示进程 Pi 已分得 Rj 类资源的数量;

资源\进程	CurrentAvil			$C_{ki}-A_{ki}$			Allocation			Currentavil+allocation			Possible
	A	B	C	A	B	C	A	B	C	A	B	C	
P_0	7	4	3	7	4	3	0	1	0	7	5	3	T
P_1	3	3	2	1	2	2	2	0	0	5	3	2	T
P_2	7	5	3	6	0	0	3	0	2	10	5	5	T
P_3	5	3	2	0	1	1	2	1	1	7	4	3	T
P_4	10	5	5	4	3	1	0	0	2	10	5	7	T

- 银行家算法的问题：
  - 进程很难在运行前知道其所需资源的最大值
  - 系统中各进程之间必须是无关的，即没有同步要求，无法处理有同步关系的进程
  - 进程的数量和资源的数目是固定不变的，无法处理进程数量和资源数目动态变化的情况
- 死锁的检测和解除
  - 资源分配图：单资源有环路：死锁；多资源有环路：化简无环的进程，释放出的新资源看看能不能继续供给环路化简，直至所有进程均孤立：无死锁，否则有死锁
- 解除：进程终止、资源抢占

内存管理

- 内存管理基础
  - 内存管理概念：
    - 程序装入与链接、逻辑地址与物理地址、内存保护
  - 内存不足的存储管理技术
    - 移动技术：移动内存当中的作业
    - 对换技术：将当前的阻塞进程和磁盘中的某个进程对换，让其投入运行
    - 覆盖技术：把程序的不同模块在内存中相互替代，实现小内存执行大程序
  - 连续分配管理方式
    - 主存分配算法
      - 最先适应算法：从小到大找未分配表，分配第一个找到的满足长度的空闲区（简单快速，实际上用得较多）
      - 下次适应算法：从未分配表的上次扫描结束处顺序找第一个满足长度的空闲区（其次）
      - 最优适应算法：扫描整个未分配表，选能满足用户进程的最小分区（其次）
      - 最坏适应算法：扫描整个未分配表，选能满足用户进程的最大空闲区
      - 快速适应算法：为经常用到的长度的空闲区设立单独的空闲区链表，按进程长度直接索引能容纳它的最小空闲区链表并取第一块空闲区分配即可
      - 伙伴内存分配算法

	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
$t=0$	1024K															
$t=1$	A-64K	64K	128K		256K				512K							
$t=2$	A-64K	64K	B-128K		256K				512K							
$t=3$	A-64K	C-64K	B-128K		256K				512K							
$t=4$	A-64K	C-64K	B-128K		D-128K		128K		512K							
$t=5$	A-64K	64K	B-128K		D-128K		128K		512K							
$t=6$	128K		B-128K		D-128K		128K		512K							
$t=7$	256K				D-128K		128K		512K							
$t=8$	1024K															

- 非连续分配管理方式
  - 分页管理
    - 一级页表：访问两次主存，一次页表，一次物理地址

        - 二级页表，访问三次主存，同上道理
        - 反置页表：为物理内存建立页表而不是进程建立页表，用哈希函数映射到哈希表，然后从哈希表中读取页表地址取页表表项再拼接成物理地址
            - 优点：减少页表对内存的占用
            - 缺点：不包含未调入的页面，仍然需要未进程建立传统页表；反置页表存放在磁盘上，缺页异常时访问速度会比较慢
    - 分段管理
- 虚拟存储管理
    - 缺页率 f：假定作业 p 共计 n 页，系统分配给它的主存块为 m (m <= n)。如果作业 p 在运行中成功访问页面的次数为 S（所访问的页面在主存中），不成功的访问次数为 F，则总的访问次数为：A = S + F；f = F / A
    - 影响因素：内存页框数、页面大小、页面替换算法、程序特性
    - 页面置换算法
        - OPT：最佳页面替换算法：调入一页而必须淘汰一个旧页时，所淘汰的页应该是以后不再访问的页或距现在最长时间后再访问的页。
        - 随机页面替换算法：要淘汰的页面是由一个随机数产生程序所产生的随机数来确定
        - FIFO（FCFS）：先进先出页面替换算法：总是淘汰最先调入主存（在主存中驻留时间最长）的那一页。
        - SCR：第二次机会页面替换算法：初始引用位为 1，访问时引用位置 1，淘汰时遇 0 淘汰零，遇 1 跳过，遇全 1 全清零并淘汰第一个
        - Clock：同第二次机会页面替换算法，区别是不再是队列而是循环队列
        - 改进的 Clock 算法：淘汰 r=0,m=0 页面，若失败则淘汰 r=0,m=1 页面，扫描过程中将指针所经过的页面 r 置 0，若又失败则指针回起始位，淘汰此时 r=0,m=0 的页面即可
        - LRU：最近最少用页面替换算法：总是淘汰在最近一段时间里较久未被访问的页面
            - NRU：引用位法：访问页则将引用位置 1，周期 t 清所有页引用位 0，淘汰时从引用位为 0 的页面中淘汰，选中后也将其它页面的引用位清 0。
            - NFU：计数法，最不经常使用页面替换算法：访问 1 次计数 1 次，周期 t 将所有计数值清 0，淘汰页引用计数值最小的页面。
            - 记时法：每页增设计时单元，引用一次将当前时间计入计时单元，周期 t 清零，淘汰时间值最小的页面
            - 老化算法：多位寄存器 r，访问 1 次最左边一位为 1，隔时间 t 寄存器右移一位，淘汰数值最小的寄存器 r 对应的页面（常被操作系统采用）
        - MIN：局部最佳页面替换算法：进程在 t 时刻发生缺页，则把该页面装入一个空闲页框。每次访问，均检查在内存中的所有页面引用情况，如果页面在时间间隔(t, t+τ)内未被引用，则移出该页面。
        - WS：工作集替换算法：进程在 t 时刻发生缺页，则把该页面装入一个空闲页框。每次访问，均检查在内存中的所有页面引用情况，如果页面在时间间隔(t -T, t)内未被引用，则移出该页面。
            - 老化算法：1000->0100->0010->0001->0000 此时页面被移出工作集
            - 时间戳：t_offset>t_max 则将页面移出工作集
        - PFF：缺页频率替换算法：如果本次缺页与前次缺页之间的时间超过临界值 T，

那么，所有在这个时间间隔内没有引用的页面都被移出工作集。

- 防止“抖动”（处理器花费大量的时间用于对换页面而不是执行计算任务的现象）发生：增加配给进程的页框数、挑选页面替换算法和改进应用程序结构
- Belady 异常：增加物理页框数量可能导致更多的缺页异常，伴随 FIFO 调度算法出现
- 最佳页面尺寸估计：f(p)=se/p+p/2，p：页面大小；s：进程的平均大小；e：页表项大小

文件管理

- 文件概念
- 文件的存取方法
  - 顺序存取（读写位置指针，依序读写）
  - 直接存取（随机存取，可以任意顺序从文件中的任何位置存取文件内容）
  - 索引存取（记录名/记录键编址）
- FCB、文件目录、文件目录项、目录文件？一个文件包含 FCB 和文件体（作业题）
- 目录文件？普通文件？特殊文件？（P299）
- 文件目录
  - 文件控制块 FCB
  - 层次（树形）目录结构：“.”指出目录自身的 inode 入口，“..”指出父目录的 inode 入口；对于根目录，“.”和“..”都指向同一个 inode
  - 索引到一个文件的方法（书 P304-305 有例子）
- 文件共享
  - 静态共享：文件连接
    - 文件链接 link
      - 检索目录找到 oldnamep 所指向文件的索引节点编号；
      - 再次检索目录找到 newnamep 所指文件的父目录文件，并把上一步中获得的索引节点编号与 newnamep 构成一个新的目录项，加入到此目录文件中；
      - 将对应索引节点连接计数分量 i_nlink 加一。
    - 文件解除链接 unlink
      - 与文件删除是同一个系统调用
  - 动态共享
    - 使用同一文件位移指针
    - 使用不同文件位移指针
- 文件的物理结构
  - 顺序文件
  - 连接文件（文件数据块最后 n Byte 是一个指针，指向下一个磁盘块）
  - 索引文件（索引表）
  - 直接文件（哈希）
- 文件系统实现
  - 活动 inode：解决频繁访问磁盘 inode 表的效率问题
  - 文件系统层次结构
    - 磁盘：超级块、索引节点区、数据区
    - 用户打开文件表、系统打开文件表

- fp->用户打开文件表 file_struct 中 file 指针 fp->系统打开文件表 file_struct（f_flags/f_count/…/f_inode）中某个表项的 f_inode->内存活动 inode 表的一个活动 inode（这个 inode 和磁盘索引节点区的一个 inode 一一对应），活动 inode 中的 i_data 指向磁盘数据块

- 文件实现
  - 创建文件 create
    - 为新文件分配索引节点和活动索引节点，并把索引节点编号与文件分量名组成新目录项，记到目录中。
    - 在新文件所对应的活动索引节点中置初值，如置存取权限 i_mode，连接计数 i_nlink 等。
    - 分配用户打开文件表项和系统打开文件表项，置表项初值。包括在 f_flag 中置“写”标志，读写位移 f_offset 清零。把各表项及文件对应的活动索引节点用指针连接起来，把文件描述字返回给调用者
  - 删除文件 unlink，i_nlink--，如果删除前 i_nlink==1，还要把文件占用的存储空间释放
  - 打开文件 open：
    - 检索目录，把对应的外存索引节点复制到活动索引节点表中。
    - 根据输入参数 mode 值核对权限，如果非法，则打开失败。
    - 为文件分配用户已打开表项和系统已打开表项，并为表项设置初值。通过指针建立表项与活动索引节点间联系。最后，返回用户已打开文件表项的序号（即文件描述字）。
  - 关闭文件 close
    - 根据 fd 找到用户已打开文件表项，再找到系统已打开表项。释放用户已打开文件表项；
    - 将系统已打开文件表项中的 f_count 减 1，若非零，则还有进程共享此表项，直接返回；否则，释放此表项，并找到与之关联的活动索引节点；
    - 将活动索引节点中的 i_count 减 1，若非零，则还有用户进程正使用该文件，直接返回，否则，讲活动索引节点内容写回磁盘索引节点分区，并释放该活动索引节点。
  - 复制文件描述符 dup：可以用于输入输出重定向
    - “echo hello! > foo”, 则在执行时输出将会被重定向到磁盘文件 foo 中 (注：重定向于文件描述符有关)。我们假定在此之前该 shell 进程只有三个标准文件打开，文件号分别为 0、1、2，以上命令行将按如下序列执行：
      - (1) 打开或创建磁盘文件 foo，如果 foo 中原来有内容，则清除原来内容，其文件号为 3。
      - (2) 通过 dup()复制文件 stdout, 即将文件号 1 出的 file 结构指针复制到文件号 4 处，目的是将 stdout 的 file 指针暂时保存一下
      - (3) 关闭 stdout，即 1 号文件，但是由于 4 号文件对 stdout 也同时有个引用，所以 stdout 文件并未真正关闭，只是腾出 1 号文件号位置。
      - (4) 通过 dup()，复制 3 号文件（即磁盘文件 foo），由于 1 号文件关闭, 其位置空缺, 故 3 号文件被复制到 1 号, 即进程中原来指向 stdout 的指针指向了 foo。
      - (5) 通过系统调用 fork()和 exec()创建子进程并执行 echo，子进程在

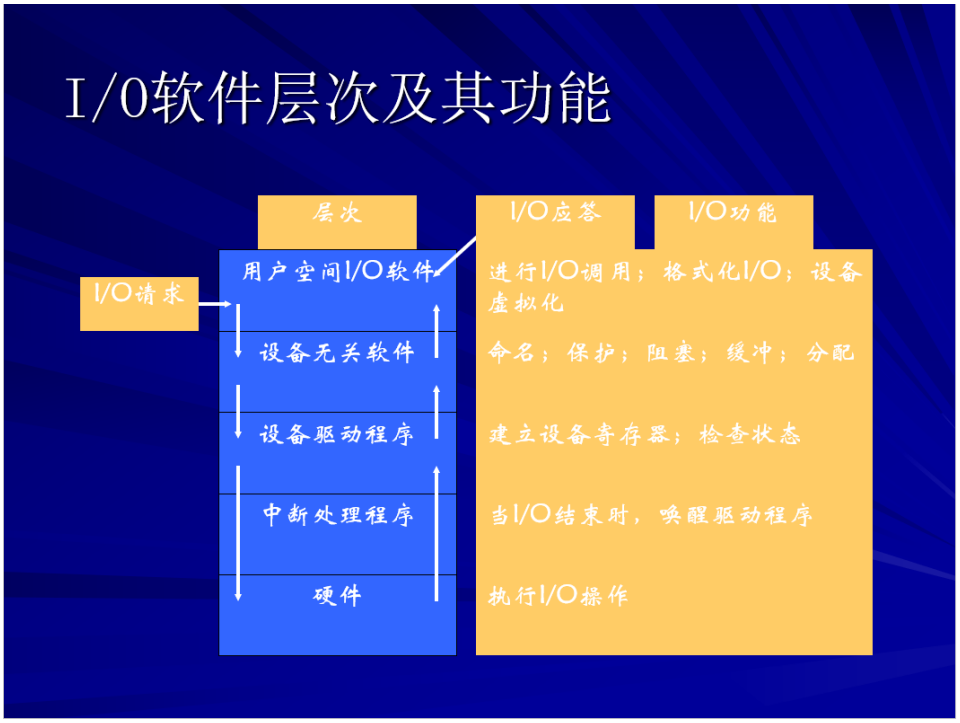
执行 echo 前夕关闭 3 号和 4 号文件，只留下 0、1、2 三个文件，请注意，这 时的 1 号文件已经不是 stdout 而是磁盘文件 foo 了。当 echo 想向 stdout 文件写入“hello!”时自然就写入到了 foo 中。

- (6) 回到 shell 后，关闭指向 foo 的 1 号与 3 号文件文件，再用 dup() 和 close()将 2 号恢复至 stdout，这样 shell 就恢复了 0、1、2 三个标准输入/输出文件。

- 磁盘空间管理
  - 位示图
  - 成组空闲块链表：保持半满
- 主存映射文件
  - 为解决文件读写效率低下的问题，结合虚存管理技术和文件管理技术实现的一种文件访问方法，将磁盘访问转为内存访问。
  - 新增 mmap 和 unmmap 系统调用，分别将文件映射到进程地址空间和断开映射，把内存数据写回磁盘文件。

设备管理

- 输入输出控制方式
  - 轮询方式：程序直接控制方式，轮询设备控制器的忙闲状态位
  - 中断方式：中断驱动的
  - DMA 方式：I/O 设备可直接与主存交换数据
    - 优点：DMA 具有处理器的能力，数据传输过程不需要占用 CPU 时间，因此 CPU 与 I/O 设备之间可以并行工作
    - 缺点：CPU 还需要在块与块之间对 I/O 操作进行干预
  - 通道方式：在开始启动时执行相应指令，并在操作结束时通过中断通知执行代码进行处理，完全并行工作
- 缓冲技术：单缓冲、双缓冲、多缓冲
- I/O 软件的层次和功能



- 驱动调度技术
  - 效率指标：若干个输入/输出请求服务所需的总时间越少，则系统效率越高
  - 存储设备的物理结构
    - 顺序存取存储设备（如磁带）
    - 随机存取存储设备（如磁盘）
  - 磁盘的访问优化
    - 循环排序：按照数据的分布对输入/输出请求进行排序，提高处理的效率
    - 优化分布：按照数据处理的规律，合理安排其磁盘上的分布，以提高处理的效率（在处理时间 4ms 内磁盘也在旋转 20ms/r，就可能略过下一个也要读的逻辑记录，假设 10 条记录，处理 A 时磁盘略过了两条记录，如果让 B 在 A 后 3 块出现，那么 A 处理完，B 刚好就可以开始处理了，这样就更好）
    - 交替地址：通过数据的冗余存放来提高访问的速度
    - 柱面斜进：？？？不懂
    - 搜查定位
      - FCFS 先来先服务：先来的先扫描
      - 最短查找时间优先：总是先执行查找时间最短的请求，可能饥饿
      - Scan Algorithm 扫描算法：扫所有柱面，遇到 I/O 就处理，直到最后一个柱面以后向相反的方向移动回来，如此往复
      - N-steps scan algorithm 分步扫描算法：将 I/O 请求分成长度为 N 的子队列，每个子队列采用扫描算法，扫描完一个以后再服务下一个子队列；扫描过程中新到达的 I/O 请求放入等待队列当中，避免“磁臂粘性”现象
      - LOOK 电梯调度算法：像电梯一样，idle 不动，每次总是选择沿移动彼得移动方向最近的那个柱面；当且仅当当前移动方向上没有且相反方向有访问请求时才改变移动臂的移动方向，然后处理所遇到的最近的 I/O 请求
      - 循环扫描算法：一直从 0 号柱面往最大柱面扫描，到最大以后返回 0 号柱面继续往最大柱面扫描，如此往复
- 设备分配
  - 静态分配：对独占型设备使用，作业执行前他要的所有资源全部分配给他
  - 动态分配：某些例如打印机等独占式使用设备可以动态分配，提高设备利用率
  - 磁盘往往可以多个作业同时使用，一般不必进行分配
- 虚拟设备 SPOOLing 技术：外部设备联机并行操作
  - 是用一类物理设备模拟另一类物理设备的技术
  - 作用
    - 使独立使用的设备变成可共享设备
    - 处理器与外围设备速度匹配
  - 提高了 I/O 速度.从对低速 I/O 设备进行的 I/O 操作变为对输入井或输出井的操作,如同脱机操作一样,提高了 I/O 速度,缓和了 CPU 与低速 I/O 设备速度不匹配的矛盾.
  - 设备并没有分配给任何进程.在输入井或输出井中,分配给进程的是一存储区和建立一张 I/O 请求表.
  - 实现了虚拟设备功能.多个进程同时使用一独享设备,而对每一进程而言,都认为自己独占这一设备,不过,该设备是逻辑上的设备.

一些其它的东西

- 实现按名存取的文件系统的优点：
  - 将用户从复杂的物理存储地址管理中解放出来
  - 可方便地对文件提供各种安全、保密和保护措施
  - 实现文件的共享（同名共享、异名共享）
- 实时调度与调度算法：时间因素非常关键的系统
  - 什么是可调度？
    - 在忽略调度本身所花费 CPU 时间的前提下，系统能够在各事件规定的响应时间处理完这些事件。
    - 对于周期事件，判断系统任务是否可调度的数学公式：C1/P1 + C2/P2 + … + Cm/Pm <= 1，其中 m 为事件总数，Ci 为某个事件的处理时间，Pi 为事件发生的周期
  - 实时调度算法
    - 单比率调度算法（静态）：进程的优先级与对应的事件出现频率成正比（最优）
    - 限期调度算法（动态）：进程就绪队列按照对应事件处理的截止期限排序
    - 最少裕度调度算法（动态）：裕度 = 截止时间 - （就绪时间 + 计算时间），选择裕度最小的进程先执行
- Bernstein 条件：并发进程的无关性
  - R(Pi)={a1, a2, …, an}，指进程 Pi 在执行期间引用的变量集
  - W(Pi)={b1, b2, …, bm}，指进程 Pi 在执行期间改变的变量集
  - 若两个进程的上述变量集合满足(R(P1)∩W(P2))∪(R(P2)∩W(P1))∪(W(P1)∩W(P2))=空集，则并发进程的执行与时间无关
  - 有四个进程：P_1: a = x + y; P_2: b = z + 1; P_3: c = a – b; P_4: w = c + 1，判断哪两个进程可并发执行？
    - $R(P_1)$ = {x, y}, $R(P_2)$ = {z}, $R(P_3)$ = {a, b}, $R(P_4)$ = {c}
    - $W(P_1)$ = {a}, $W(P_2)$ = {b}, $W(P_3)$ = {c}, $W(P_4)$ = {w}
    - P_1 和 P_2 的上述集合满足 Bernstein 条件，可并发执行。

后记：

每一章的本章小结和第五章 4.6（P235）也要看一看！