考研计算机408 · 冲刺背诵手册系列

操作系统
冲刺背诵手册

必背必做系列

2025 版

请添加学姐微信：kaoyan1200

前言

相信大家在学习 408 的时候都会出现出现这么一种情况：学着忘着，甚至有时候刚学完一章，再做题的时候就记不清了，这时候再看一遍书和笔记的话效率就会很低。虽然 408 不是文科，但还是有很多需要记忆的内容的。**所以 408 冲刺背诵手册的目的就是把那些需要特别记忆而有容易记错的内容提取出来，便于让大家加深对这些知识点的印象。**

去年（2023 年）我制作了第一版（2024 版）408 冲刺背诵手册，受到了很多同学的好评，也得到了大家的支持。毕竟去年是第一版，所以还存在很多问题，于是在各位同学的建议下，我利用寒假时间，又**为 25 考研的同学专门制作了第二版（2025 版），**这一版改动很大，无论是在外观还是在内容上，下面我将一一介绍：

1. 首先，在外观和排版上，25 版的冲刺背诵手册**全册采用 LaTeX 进行排版**，特别是封面，全部采用 tikz 绘制[1]，在电子版和打印版都做了应有的适配，可以让大家拥有最好的视觉体验；
2. 其次，在内容上，我延续了 24 版的大部分优点，改善了 24 版的一些缺点，在**目录的设置**上，我分别**对不同的内容标注了不同的符号**，更清晰明了；在内容简洁的基础上，我又穿插了三个板块：
   (a). **总结板块（橙色）：** 总结一些记忆技巧；
   (b). **提示板块（绿色）：** 提示一些容易出错的地方；
   (c). **章节手记板块（蓝色）：** 每一章的最后我都为大家划分了一些空白区域，用于记录大家对于本章的独特理解。

说完改动，我想聊一聊这本手册适合那些同学使用，以及什么时候使用：

- **适用人群：**适合所有考 408 的同学使用，**尤其是基础差的，记忆力不好的；**
- **什么时候用：**这本手册最适合在二三轮复习使用，特别是中后期复习阶段甚至考前，由于这个阶段你已经学了很多东西，所以遗忘在所难免，那使用这本手册就能够增强你对那些重要知识点的记忆！

最后，我想提前回答大家的一些疑惑，也是去年问得最多的**两个问题**：

1. **为什么有些知识点没有出现在这本手册上?**
   答：上面我们就提到：“408 冲刺背诵手册的目的就是把那些需要特别记忆而有容易记错的内容提取出来，便于让大家加深对这些知识点的印象。”，所以这本手册只是整理了特别需要记忆的内容，**帮你把整个框架搭建起来**，而那些非记忆和次要的内容，我不会放在这本手册上。如果所有东西都放在这个手册上，那篇幅就太大了，恐怕几百页都不够，不如去看教材，那是最全的。这也是为什么我不整理 408 笔记的原因，王道书（或者其他辅导书）已经给我们整理的够好了，我们只需要在听课的时候

[1]封面图片来源：https://wallhaven.cc/

把书上没有的东西添上去就行了。（关于学习方法，我就不过多讨论了，有兴趣的可以去看我做的 25 考研 408 复习攻略。）

2. **为什么会有水印?**

答：很多同学在去年都问了这个问题，我的回答是“防止盗用”，仅此而已。当然，这些水印绝对不会影响大家的阅读，而且今年的水印我会专门设计，看起来更协调更“不像”水印一些！

关于**408 冲刺手册的发布时间**：由于 408 一共四个考试科目，所以我会分四次发布。

- **《数据结构冲刺背诵手册》**（每年 4 月发布）
- **《计算机组成原理冲刺背诵手册》**（每年 5 月发布）
- **《操作系统冲刺背诵手册》**（每年 6 月发布）
- **《计算机网络冲刺背诵手册》**（每年 7 月发布）

此外，对于电子版的 408 冲刺背诵手册，我会紧随大纲的改动进行更改、添加内容，如有改动，大家只需要重新下载即可！

微信扫描下方二维码，关注公众号：研小布，可获取 408 冲刺背诵手册的最新发布消息！如果有任何建议或者改进措施，也可以通过公众号告诉我！

最后，谢谢大家的支持，正是有了你们的支持，我才有继续做下去的动力！祝考研顺利！

目录[2]

第 1 章　系统概述 1
1.1　四个重要问题★ 1
1.2　OS Boot 1
1.2.1　引导流程★✪ 1
1.3　程序运行的内存映像 2
1.3.1　地址空间结构 2
1.3.2　内存映像几大要素总结★ 2
1.4　时钟中断 3
1.4.1　概念解释 3
1.4.2　时钟中断会做的事情 (三个修改)✪ 3
1.4.3　置时钟指令 3
1.5　中断类 3
1.5.1　中断处理和子程序调用的区别✪ 3
1.5.2　“送中断向量”和“初始化中断向量表”的区别✪ 3
1.5.3　执行系统调用的过程★ 4

第 2 章　进程与线程 5
2.1　十一个重要问题★ 5
2.2　各种关系总结 5
2.2.1　临界区与临界资源之间的关系 5
2.2.2　临界资源与共享资源的区别✪ 6
2.2.3　并发进程之间的关系 6
2.2.4　进程与线程的关系★ 6
2.2.5　父子进程的关系 6
2.2.6　进程与作业之间的区别✪ 6
2.3　进程通信✪ 7
2.3.1　共享存储 7
2.3.2　消息传递 7
2.3.3　管道通信 (Pipe) 7
2.4　进程调度 8
2.4.1　调度性能指标★▲ 8
2.4.2　三级调度★✪ 9
2.4.3　进程切换的代价 9
2.4.4　各种调度算法的特性★✪▲ 9
2.5　同步与互斥 10
2.5.1　软件实现临界区互斥✪ 10
2.5.2　硬件实现临界区互斥✪ 10
2.5.3　前驱图✪ 12
2.5.4　管程✪ 12
2.6　死锁 13
2.6.1　死锁的预防★✪ 13
2.6.2　死锁避免✪▲ 13
2.6.3　死锁检测和解除✪▲ 13
2.7　PV 信号量✪ 14
2.7.1　整形信号量 14
2.7.2　记录型信号量 14
2.8　PV-读者写者问题✪▲ 15
2.8.1　读优先 15
2.8.2　平衡读写 16
2.8.3　写优先 17
2.9　PV-哲学家进餐问题✪▲ 18

第 3 章　内存管理 20
3.1　四个重要问题★ 20
3.2　重定位所处过程总结 20
3.3　连续分区管理★✪ 20

[2]★：表示优先记忆内容；✪：表示重要考点；▲：表示重要计算

3.4 虚拟内存管理 21
3.4.1 基本概念 21
3.4.2 内存分配/置换策略★✪ ... 21
3.4.3 磁盘文件区与对换区 22
3.5 CLOCK 置换算法 22
3.5.1 简单 CLOCK 22
3.5.2 改进型 CLOCK ★✪▲ 22
3.6 抖动和工作集 22
3.6.1 抖动/颠簸✪ 22
3.6.2 工作集与驻留集▲ 23
3.7 内存映射文件 23
第 4 章 文件管理 25
4.1 三个重要问题★ 25
4.2 文件共享 25
4.2.1 硬链接 25
4.2.2 软链接★✪ 25
4.2.3 open 与 read 系统调用 25
4.3 FAT 文件分配表 26
4.3.1 基本结构 26
4.3.2 FAT 长度计算✪▲ 26
4.4 不同盘块组织方式的优缺点 ... 27
4.5 文件数量的计算▲ 27
第 5 章 IO 管理 29
5.1 十一个重要问题★ 29
5.2 IO 设备的分类 30
5.3 磁盘管理 30
5.3.1 磁盘的地址结构 30
5.3.2 簇号与磁盘地址的转换✪▲ 30
5.3.3 磁盘的初始化工作★ 31
5.3.4 磁盘调度算法▲ 31
5.4 固态硬盘 SSD 32
5.4.1 原理 32
5.4.2 组成 32
5.4.3 读写特性 33
5.4.4 磨损均衡技术（Wear Leveling） 33
5.5 缓冲区✪▲ 33
5.5.1 单缓冲 33
5.5.2 双缓冲 33
5.5.3 缓冲池 34
5.5.4 磁盘高速缓存 34
5.5.5 高速缓存与缓冲区对比 34
5.6 IO 接口★ 35
5.7 SPOOLing 技术✪ 35
5.7.1 组成★ 35
5.7.2 组件管理程序 35
5.7.3 特点 35
5.7.4 要求 35

请添加学姐微信：kaoyan1200

第 1 章 系统概述

1.1 四个重要问题★

1. 系统调用的目的?
   - 直接目的是请求系统服务，在该过程中还会间接申请系统资源
2. 特权指令/广义指令：用户态下调用，核心态下执行
3. 常见特权指令
   - 对 IO 设备操作相关的指令
   - 有关访问程序状态的指令
   - 存储特殊寄存器相关的指令
4. OS 与用户通信接口通常包含哪些?
   - shell
   - 命令解释器
   - 广义指令

1.2 OS Boot

1.2.1 引导流程★✪

1. 激活 CPU：读取 BIOS 中的 boot 程序，执行 BIOS 的第一条指令
   - CPU 加电，CS:IP 指向 FFFF0H
   - 执行 JMP 跳转到 BIOS
   - 登记 BIOS 中断程序入口地址
2. 硬件自检
3. 加载 Boot Device
4. 加载 MBR：MBR 的作用是告诉 CPU 应该去哪个主分区寻找 OS
5. 扫描硬盘分区表，加载活动分区
6. 加载 PBR：活动分区的第一个扇区称为 PBR，目的是寻找并启动该分区下的启动管理器
7. 加载启动管理器：启动管理器就是活动分区中负责引导整个 OS 启动的程序
8. 加载 OS

1.3 程序运行的内存映像

1.3.1 地址空间结构

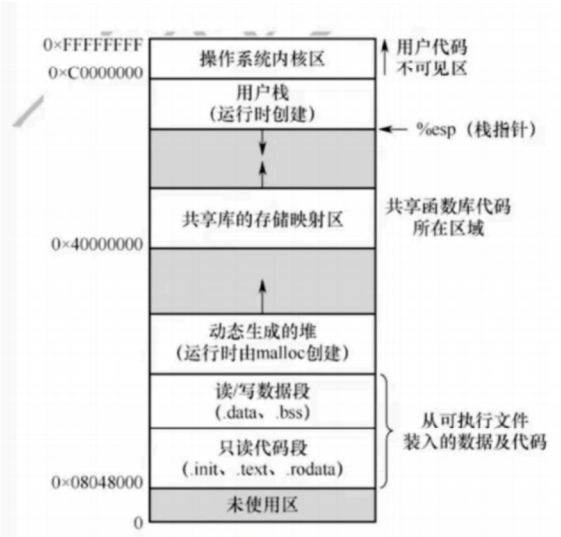
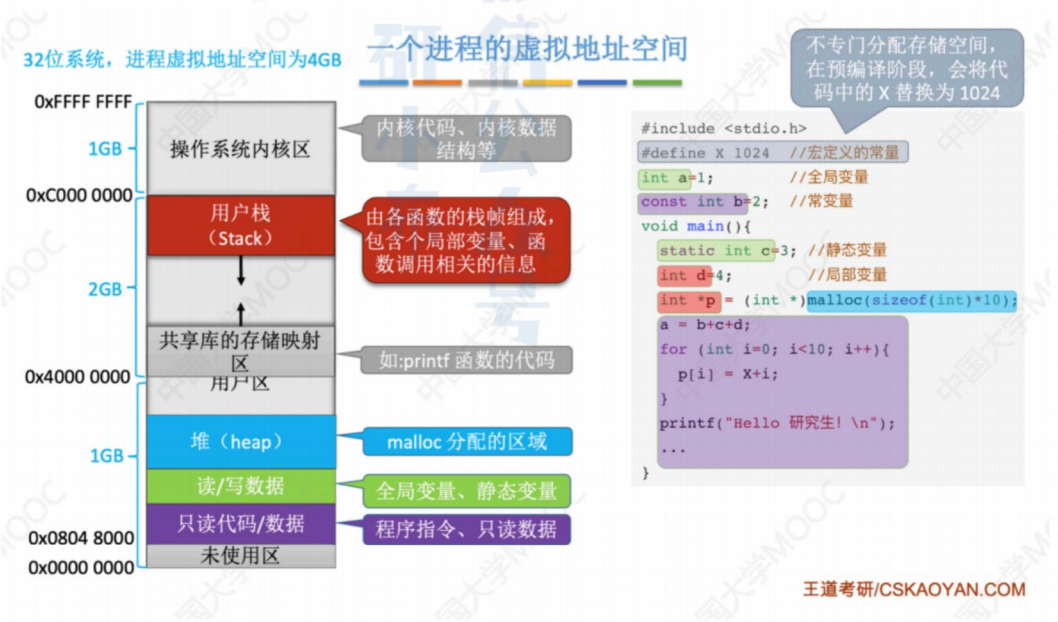


图 1.1 内存中的一个进程



1.3.2 内存映像几大要素总结★

- 代码段（read only）：存放二进制代码（可被多个进程共享）、const 常量
- 数据段：静态变量、全局变量
- 代码段与数据段合称为：正文段
- 堆：动态分配的变量
- 栈：实现函数调用，临时/局部变量（包括未赋值）
- 系统内核区：PCB（普通进程无法访问）

- 共享存储区：动态加载的共享库（dll 等）

1.4 时钟中断

1.4.1 概念解释

时钟中断：指的就是时间片到

- 系统每隔一段时间向 CPU 发送一个“时钟中断”，从而实现分时
- 当 CPU 收到时钟中断信号后，转而执行“时钟中断服务程序”

1.4.2 时钟中断会做的事情 (三个修改)✪

- 修改内核中时钟变量的值
- 修改当前进程在时间片内的剩余执行时间
- 修改当前进程占用 CPU 时间（总占用时间）

1.4.3 置时钟指令

修改时钟内容，核心态下运行

1.5 中断类

1.5.1 中断处理和子程序调用的区别 ✪

- 中断处理：
  - 保存 PC
  - 保存 PSW：其中一个作用是标记 CPU 状态（目态/管态）因为执行完中断需要返回中断前的状态，所以需要保存 PSW（eg 嵌套中断）
- 子程序调用：
  - 仅仅保存程序断点（PC）

1.5.2 “送中断向量”和“初始化中断向量表”的区别 ✪

- 中断向量表：内存中专门用于存放所有类型的中断向量的区域，一般在开机的时候由 OS 初始化
- 送中断向量：指的是中断隐指令根据中断类型选择对应的中断向量送给 PC，让 PC 接下来能够执行对应的中断服务程序。

1.5.3 执行系统调用的过程 ★

1. 传递系统调用参数
2. 执行陷入（trap）指令
3. 执行相应服务程序
4. 返回用户态

章节手记

第 2 章 进程与线程

2.1 十一个重要问题 ★

1. 创建一个新进程需要做的事：
   - 申请一个空白 PCB
   - 初始化 PCB 必要信息：进程标志信息、进程优先级、处理机状态信息等
2. 进程的封闭性：进程封闭性指的是，进程执行结果只和本身有关（无论进程以什么速度或顺序执行，都不会影响结果）
3. 多线程模型：UTL、KTL（记忆：英国 UK）
4. UNIX 属于什么操作系统?
   - 分时操作系统
5. 无中断，不并发
   - 没有中断，就没有多道程序设计，中断是非常非常重要的技术
6. pip 文件存放位置?
   - 主存（所以管道大小与磁盘大小无关）
7. OS 通过什么控制进程?
   - 操作系统是通过 PCB 来对进程进行控制和管理的
8. 分时操作系统的时间片可调吗?
   - 不可调整
9. 中断结束后是否有可能引发调度?
   - 有可能。因为中断结束后 OS 同样会检测时间片轮，如果刚好轮到自己，那就会直接调度上处理机运行
10. 降低进程优先级的合时机：不应在进程运行时降低，应当在时间片到后降低优先级!
11. 实现互斥必须遵循的原则：
    - 空闲让进：临界区空闲时允许立即进入
    - 忙则等待：当临界区中已有进程时，其他进程需要等待
    - 有限等待：请求访问临界区的进程，需保证在有限时间内进入
    - 让权等待：当进程无法进入临界区时，应立即释放处理器（非必须，别忘了，所有硬件互斥方法都没法实现让权等待）

2.2 各种关系总结

2.2.1 临界区与临界资源之间的关系

- 临界区：访问临界资源的代码段

- 临界资源：① 一个时间段内允许多个进程使用；② 一个时刻内仅允许一个进程使用的资源

2.2.2 临界资源与共享资源的区别✪

- 临界资源：一次只供一个进程使用（eg. 缓冲区、队列、打印机）
- 共享资源：一次可以供多个进程使用（eg. 共享区代码、磁盘）

小布提示

它们之间的区别就在于一段时间内能否被多个进程访问。

2.2.3 并发进程之间的关系

- 没有必然的关系，只有时间上的偶然重合（异步性）
- 所以他们之间可能是相关的，也可能是无关的

2.2.4 进程与线程的关系★

	进程	线程
基本单位	永远是“**资源**分配”的基本单位	引入线程后，线程作为“**CPU调度**”的基本单位，
资源	拥有资源，同下	没有独立资源，可访问**父进程**的资源
独立性	拥有**独立的**地址空间	共享**所属进程**资源

小布提示

虽然线程从属于进程，但也有自己的 PCB

2.2.5 父子进程的关系

- 父子进程可并发执行
- 子进程独享一块虚拟地址空间（补充：父子进程可以共享一部分资源，但不能共享虚拟地址空间）
- 父子进程拥有不同的 PCB

2.2.6 进程与作业之间的区别✪

- 作业：以用户的角度出发，由用户提交，以用户任务为基本单位
- 进程：以 OS 角度出发，由 OS 生成，是 OS 资源分配和独立运行的基本单位

2.3 进程通信 ✪

2.3.1 共享存储

在通信的进程间存在一块可直接访问的共享空间，通过对这片空间 R/W 进行信息交换。

特点与实现方式：

- 需使用互斥工具
- 低级方式：基于数据结构（包括文件）
- 高级方式：基于存储区

2.3.2 消息传递

数据交换以格式化的 Message 为单位，需要使用 OS 提供的“发送信息”和“接收信息”两个原语进行通信。

实现方式：

- 直接通信方式：发送方直接给接收方发送消息，并将它挂到进程的接收消息缓冲队列上，接收方从队列中取得消息
- 间接通信方式：两个进程之间通过“信箱”来中转信息

2.3.3 管道通信 (Pipe)

- 管道：连接两个进程之间的一个共享文件（又名 pipe 文件），数据在管道中以字符流的形式传输。
- 特性：
  - 只存在于内存中（磁盘速度太慢了，也达不到进程通信的要求）
  - 半双工（若要实现全双工，需要两条管道）
  - 只要管道没满，就能继续写；只要管道没空，就能继续读
  - 破坏性读出：管道读取数据是一次性操作，数据一旦被读取，立马释放空间存入新的数据
- 管道基本功能：
  - 确定对方的存在
  - 同步
  - 互斥
- 读写者数量要求：同一时间，管道可以拥有多个写进程，但只能有一个读进程。

小布提示

因为管道的本质就是缓冲区，是破坏性读出，一旦多个进程一起读，会把别的进程数据给破坏了。而写操作并不会破坏管道内原有的数据，所以允许多个进程同时写。

2.4 进程调度

2.4.1 调度性能指标 ★▲

- 周转时间 = 作业完成时间 − 提交时间
- 带权周转时间 = $\frac{\text{周转时间}}{\text{实际运行时间}}$
- 平均带权周转时间 = 带权周转时间的加权平均
- 等待时间：进程被创建后未上机的时间
- 平均等待时间：等待时间的加权平均
- 响应时间：从用户提交请求到系统产生响应所需的时间

小布提示

进程切换和调度的时间，表示系统在每次上处理机运行前所需要耗费的时间

例题 2.1 【2018 年 T24】

24. 某系统采用基于优先权的非抢占式进程调度策略，完成一次进程调度和进程切换的系统时间开销为 1μs。在 T 时刻就绪队列中有 3 个进程 P_1、P_2 和 P_3，其在就绪队列中的等待时间、需要的 CPU 时间和优先权如下表所示。

进程	等待时间	需要的 CPU 时间	优先权
P_1	30μs	12μs	10
P_2	15μs	24μs	30
P_3	18μs	36μs	20

若优先权值大的进程优先获得 CPU，从 T 时刻起系统开始进程调度，系统的平均周转时间为（　）。

A. 54μs　　B. 73μs　　C. 74μs　　D. 75μs

解析 注意题中说：完成一次进程调度和调度和切换的时间为 $1\mu s$，表示进程每次上机执行前都要花费 $1\mu s$ 的时间。

如图所示：

经过计算，平均周转时间 $= \frac{(25+15)+(62+18)+(75+13)}{3} = 75$

2.4.2 三级调度 ★✪

- 高级/作业调度
- 中级/内存调度
- 低级/进程调度：频率最快、最基本、不可或缺

2.4.3 进程切换的代价

- 重新保存/恢复 PTR
- TLB 全失效
- Cache 全失效
- 运行初期缺页率较高

进程的内容（被该进程的各线程共享）	每个线程独有的内容
地址空间 全局变量 打开文件	程序计数器PC 寄存器 堆栈 状态

红字全是进程切换时需要保存/恢复的内容

2.4.4 各种调度算法的特性 ★✪▲

- FCFS：不利于短作业
- SJF：最短平均周转时间，长作业可能饥饿
- 高响应比优先：响应比 $=\frac{\text{等待时间+要求服务时间}}{\text{要求服务时间}}$(倒三角)，满足短作业优先（总体兼顾），且不会发生饥饿

小布提示

I/O 密集型作业 = 长作业

CPU 密集型 = 短作业 ♣

例题 2.2 王道课后习题：

03.（ ）有利于 CPU 繁忙型的作业，而不利于 I/O 繁忙型的作业。

A. 时间片轮转调度算法 **B. 先来先服务调度算法**

C. 短作业（进程）优先算法 **D. 优先权调度算法**

解析 I/O 繁忙型 = 短作业（这里区分一下 I/O 繁忙型和上面的 I/O 密集型，I/O 密集更倾向于一段时间内主要在做 I/O，而 I/O 繁忙更倾向于一段时间内做很多次 I/O，需要多次用到 CPU），而 FCFS 不利于短作业，选 B.

2.5 同步与互斥

2.5.1 软件实现临界区互斥✪

算法	优点	缺点
单标志法		违背空闲让进
双标志先检查	空闲让进	违背忙则等待
双标志后检查		可能导致饥饿
Peterson's Algorithm	不会导致饥饿	无法实现让权等待

2.5.2 硬件实现临界区互斥✪

小布提示

首先牢记：所有硬件方法均不能实现让权等待！

例题 2.3 【2018 年 T32】

32. 下列同步机制中，可以实现让权等待的是_____。

A. Peterson 方法　　B. swap 指令　　C. 信号量方法　　D. TestAndSet 指令

解析 由上面的的结论，可以直接排除 BD

皮特森算法也无法满足让权等待

所以答案选 C

- 中断屏蔽法：（又称低级方法/元方法）

```
……
关中断；
临界区；
开中断；
……
```

缺点：

- 限制了处理机的交替执行程序能力，会导致处理效率降低。
- 将关中断权限交给用户是一种很危险的做法

- TSL（Test and Set Lock）：

```
bool TSL(bool &lock){
    bool old;
    old=lock;
```

```
    lock=true;
    return old;
}

do{
    ......
    while(TSL(&lock));  //lock=true表示资源正在被占用
    临界区;
    lock=false;         //解锁资源
    ......
}
```

- TSL 做的事情其实就是表明自己想用资源的意愿，并且返回当前资源的状态（是否被人使用）
- 当别人用完资源后（lock=false），TestAndSet 会立马检测到资源空闲，于是将 lock 重新设为 true（锁住资源），并向上层报告资源当前可供自己使用

总结

重要考点总结：

- 使用 TSL 实现互斥的进程，不会进入阻塞态，因为 TSL 实现阻塞的本质是让进程不断执行空的 while 循环。
- 进入“等待”状态的进程，不会主动放弃 CPU，因为上一条已经说了，进程不会进入阻塞态，自然也就无法主动放弃 CPU。
- TSL 不满足”让权等待“，因为让全等待指的是：进程无法访问临界区时，主动放弃 CPU，显然他不满足。

例题 2.4 【2016 年 T27】

27. 使用 TSL（Test and Set Lock）指令实现进程互斥的伪代码如下所示。

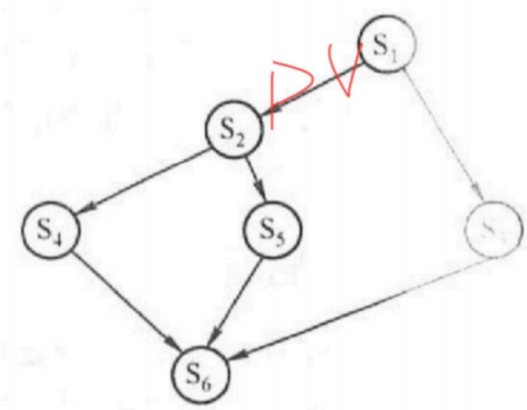
```
do{
    ...
    while(TSL(&lock)) ;
    critical section;
    lock=FALSE;
    ...
} while(TRUE);
```

下列与该实现机制相关的叙述中，正确的是_____。

A. 退出临界区的进程负责唤醒阻塞态进程
B. 等待进入临界区的进程不会主动放弃 CPU
C. 上述伪代码满足“让权等待”的同步准则
D. while(TSL(&lock))语句应在关中断状态下执行

解析 答案：B

2.5.3 前驱图✪



- 关键：前 V 后 P
- 注意：用作前驱关系的信号量初值为 0

2.5.4 管程 ✪

- 背景：管程是在程序设计语言中被引入的，是一种高级的同步机制
- 特点：
  - 管程把对共享资源的操作封装起来
  - 每次仅允许一个进程进入管程
- 条件变量：
  - `x.wait`：x 条件（请注意这里的条件并不像 PV 信号量那样是某个具体值）不满足时，将自己插入 x 条件的等待队列，并释放管程，允许其他进程使用该管程（让其他人先用）
  - `x.signal`：x 条件满足时，唤醒一个因 x 条件而阻塞的进程
- 易错点：
  - 管程的 wait() 和 signal() 操作不会对条件变量进行检查，这个条件变量仅用于实现管程内部队列的排队功能（并不像信号量那样代表实际的资源数量）
- 与 PV 操作相比：

	管程	PV
主要功能	进程的阻塞/唤醒，**但仅限于"排队等待"**	进程的阻塞/唤醒
剩余资源的表示	用**共享数据结构**记录	**信号量的值**反应了资源**剩余数**
灵活性	管程是**被**进程**调用的**，**无法创建和撤销**	自由编写

2.6 死锁

2.6.1 死锁的预防★ ✪

- 打破循环等待：给系统资源编号（'循环编号'）
- 打破请求并保持：预先静态分配（'保持静态'）
- 打破不剥夺：实现较为复杂
- 打破互斥：一般无法打破

2.6.2 死锁避免 ✪ ▲

- 银行家算法：
  - Available：剩余可用
  - Max：最大需求
  - Allocation：目前已分配
  - Need：剩余需求

小布提示

银行家算法无法检测出是否处于死锁状态 ♣

- 系统安全状态：进入不安全状态，未必会发生死锁；但是发生死锁，一定是在不安全状态。

2.6.3 死锁检测和解除 ✪ ▲

- 资源分配图
  - 如果能够消除所有的边，说明资源分配图可完全简化
  - 如果不能消除所有的边，则说明发生死锁
  - 孤点：表示没有任何一条边与该点相连
  - 包含：
    - 大圆圈：进程
    - 框中的一个圆：一类资源中的一个资源
    - 资源-> 进程的边：分配边
    - 进程-> 资源的边：请求边

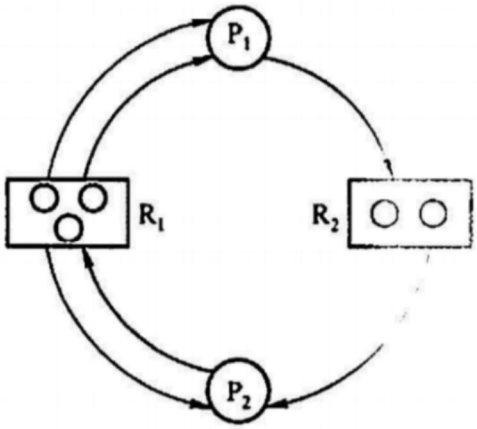


图 2.15 资源分配示

- 死锁定理：就是通过优化资源分配图来检测死锁
- 死锁解除（记忆撤销剥夺与回退）：
  - 撤销进程法
  - 资源剥夺法
  - 进程回退法

2.7 PV 信号量 ✪

2.7.1 整形信号量

```
wait(S){
    while(S<=0);
    S--;
}
signal(S){
    S++;
}
```

2.7.2 记录型信号量

```
wait(semaphore S){
    S.value--;
    if (S.value < 0){
        // 添加进程到S.L
        block(S.L); // 自我阻塞（达到让权等待）
    }
```

```
}
```

```
signal(semaphore S){
    S.value++;
    if (S.value <= 0){
        // 从S.L中移除该进程P
        wakeup(P); // 唤醒该进程
    }
}
```

2.8 PV-读者写者问题 ✪ ▲

小布提示

读写者问题中，通常写者直接是互斥访问的，而读者之间是不互斥访问的。想要实现这个核心逻辑就是为读者分配一个 count 计数器，并利用 if 判断确保后来的读者跳过 P() 互斥操作。

2.8.1 读优先

```
//读者优先
Semaphore lock=1; //读者与写者之间的互斥锁
Semaphore m=1;    //count的互斥锁
int count=0;      //读者计数器
Reader(){
    P(m);
    if(count==0) P(lock); //第一个读者负责上锁
    count++;
    V(m);
    //reading……
    P(m);
    count--;
    if(count==0) V(lock); //最后一个读者负责解锁
    V(m);
}

Writer(){
    P(lock);
```

```
    //writing.....
    V(lock);
}
```

2.8.2 平衡读写

小布提示

实际上要实现“平衡读写”，只需要在读优先的基础上加一个 queue 信号量来实现“排队”操作。queue 信号量的使用精髓在于：干任何事情前首先排队，在真正读写之前，通知队列中的下一个进程。

```
//读写平衡
Semaphore lock=1; //读者与写者之间的互斥锁
Semaphore queue=1; //相当于一个排队的队列作用（用于实现读写平衡）
Semaphore m=1;    //count的互斥锁
int count=0;      //读者计数器
Reader(){
    P(queue);             //先排队
    P(m);
    if(count==0) P(lock); //第一个读者负责上锁
    count++;
    V(m);
    P(queue);             //真正读之前通知队列中的下一个进程
    //reading.....
    P(m);
    count--;
    if(count==0) V(lock); //最后一个读者负责解锁
    V(m);
}

Writer(){
    P(queue); //排队
    P(lock);
    V(queue)  //真正写之前通知队列中的下一个进程
    //writing.....
    V(lock);
}
```

2.8.3 写优先

写优先的实现相对复杂，它的实现原理如下：

- 先将写者改造为读者（加上自己单独的 count 计数器），此时写者之间也不互斥
- 因此要在写操作的前后加上一对 PV 恢复写互斥
- 接下来就是通过读写者之间互相给对方上锁，来实现读写者之间的互斥。
- 若此时再给读者套一对”检测 PV“，则读者的优先级会更低，从而实现写优先

小布提示

”检测 PV” 的精髓在于：真正读写之前要先解锁（准确说是给同伴解锁），否则会影响到原有的同步顺序

```
//写优先
Semaphore Wlock=1; //用于给写者上锁
Semaphore Rlock=1; //用于给读者上锁
Semaphore Wm=1;    //Wcount的互斥锁
Semaphore Rm=1;    //Rcount的互斥锁
int Wcount=0;      //写者计数器
int Rcount=0;      //读者计数器
Reader(){
    P(Rlock);      //”检测PV“：此时如果发现已经被写者上锁，则会卡住（实现了写优先）
    P(Rm);
    if(Rcount==0) P(Wlock); //第一个读者给写者上锁
    Rcount++;
    V(Rm);
    V(Rlock);      //若是自己上的锁，则在读之前要先给其他同伴解锁（实现了读者不互斥）
    //reading……
    P(Rm);
    Rcount--;
    if(Rcount==0) V(Wlock); //最后一个读者给写者解锁
    V(Rm);
}

Writer(){
    P(Wm);
    if(Wcount==0) P(Rlock); //第一个写者给读者上锁（写优先的关键）
```

```
    Wcount++;
    V(Wm);

    P(Wlock);  //这对Wlock是用于实现写者之间的互斥（很容易遗忘）
    //writing.....
    V(Wlock);

    P(Wm);
    Wcount--;
    if(Wcount==0) V(Rlock);  //最后一个写着负责为读者解锁
    V(Wm);
}
```

2.9 PV-哲学家进餐问题✪▲

总结

实现精髓在于：

- 用 int 类型定义所有资源
- 一次性索取/归还所有资源


```
semaphore Lock = 1; // 互斥锁
int 所有资源 = 资源数;
Process () {
    while(1) {
        P(Lock); // 在取资源前先套一个大锁，然后一口气拿走所有资源
        if(所有资源都足够) { //循环检测资源资源充足
            // 所有资源的int值减少，一口气取走所有资源
            V(Lock); // 取走所有资源后解锁
            break; // 并跳出用于检测资源的while循环
        }
          V(Lock); // 资源不够，解锁并再循环一次
    }

      //....
    // 此处做进程该做的事情（eg.哲学家干饭）

    P(Lock);
```

```
    // 所有资源int值增加，一口气归还所有资源
    V(Lock);
}
```

✍ 章节手记

第 3 章 内存管理

3.1 四个重要问题★

- 引入“段式存储管理”的目的（优点）?
  - 方便编程
  - 共享和保护
  - 动态链接和增长（增加实际内存容量）
- 进程与页表的关系?
  - 每个进程都有一张页表
  - 绝大多数进程的页表都是常驻内存的（不论是否运行）
- 影响请求分页系统有效（平均）访存时间的是?
  - 缺页率
  - 磁盘读写时间
  - 内存访问时间
  - 执行缺页处理程序的 CPU 时间
- 发生缺页时，操作系统是不会检查是否越界的
  - 因为“缺页”只有可能在查询页表的时候发现，此时我们使用的已经是转换完成的物理地址了，所以一定不会越界。
  - “越界检查”这个步骤是在 $VA \rightarrow PA$ 的过程中就已经完成了。

3.2 重定位所处过程总结

- 绝对装入：链接、编译阶段
- 静态重定位（可重定位装入）：装入阶段
- 动态重定位（动态装入）：运行阶段

总结

地址变换阶段总结：

- 形成逻辑地址：链接阶段
- 逻辑地址-> 物理地址：运行阶段

3.3 连续分区管理 ★✪

- 单一连续分配
  - 内存中永远只有一道程序
  - 仅适用于单用户、单任务 OS

- 固定分区分配
  - 一种最简单的多道程序管理方式
- 动态分区分配

算法	首次适应	最佳适应	最坏适应	邻近适应
特点	每次从头找	从小往大找	从大往小找	继续向下找
优点	综合性能ok，开销小	会有更多大分区被保留	内部碎片少	开销小
缺点		产生内部碎片，开销大	大分区很快被用完	高地址大分区很快用完

3.4 虚拟内存管理

3.4.1 基本概念

- 虚拟存储器的特征：
  - 多次性：作业分多次调入内存
  - 对换性：作业无需常驻内存，允许多次换入换出
  - 虚拟性：逻辑上扩充主存【最主要目标】
- 实现方式：
  - 请求分页管理
  - 请求分段管理
  - 请求段页式管理

小布提示

注意：无论以上哪种实现方式，都需要硬件支持！

- 所需硬件：
  - 一定量的内外存
  - 页/段表机制
  - (缺页) 中断机构
  - 地址变换机构

3.4.2 内存分配/置换策略 ★ ✪

- 固定分配：分配给一个进程的页框数是定死的
- 可变分配：分配给一个进程的页框数可变
- 全局置换：缺页时，从主存分配更多页框
- 局部置换：缺页时，从驻留集内部替换

由上述各 2 种不同的置换和分配策略的规则组合，产生了 3 种不同的内存分配策略：

- 固定分配局部置换
- 可变分配局部置换
- 可变分配全局置换

3.4.3 磁盘文件区与对换区

	文件区	对换区
存放内容:	文件数据	对换页面
存储结构:	**离散**分配（文件）	**连续**分配（字符流）
读写速度:	低速	高速

3.5 CLOCK 置换算法

3.5.1 简单 CLOCK

- 首轮：找访问位 =0 的，并置 0
- 后续：同首轮

小布提示

当前指针指向的页面在第一轮也会去找 ♣

3.5.2 改进型 CLOCK ★ ✪ ▲

在简单 CLOCK 的基础上增加为 < 访问，修改 >（先访问再修改）。

- 首轮：找 (0,0)
- 二轮：找 (0,1) 且将访问位置 0
- 三轮：找 (0,0)
- 四轮：找 (0,1)，经过前三轮，这一轮必能找到

总结

该算法的淘汰页次序为：(0,0), (0,1), (1,0), (1,1) ♡

3.6 抖动和工作集

3.6.1 抖动/颠簸 ✪

刚换入的页面又要换出，刚刚换出的页面又要调入。

小布提示

抖动会导致大量时间用于 IO，导致排队进程急剧增加，恶性循环，从而导致系统性能接近于 0

3.6.2 工作集与驻留集 ▲

- 驻留集：预先给进程分配的物理块
- 工作集：进程实际访问页面的集合

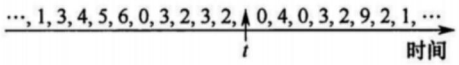
小布提示

工作集窗口包含重复!!!，但工作集不包含重复。

通常工作集 ⩽ 驻留集。

例题 3.1 王道课后习题：

29. 某进程访问页面的序列如下所示。



若工作集的窗口大小为 6，则在 t 时刻的工作集为（ ）。

A. {6, 0, 3, 2}　　B. {2, 3, 0, 4}

C. {0, 4, 3, 2, 9}　　D. {4, 5, 6, 0, 3, 2}

解析 解题核心：要知道工作集窗口是可以包含重复的，但工作集是不包含重复的

本题工作集窗口 $= 6$，从 t 往前数 6 个元素：$\{6, 0, 3, 2, 3, 2\}$

去除重复后得到工作集：$\{6, 0, 3, 2\}$

3.7 内存映射文件

- 该技术能够将磁盘文件的内容映射到虚拟内存的某个区域
- 使得用户对该区域的读写就仿佛在读写主存一样，<u> 大大减少了 IO 次数 </u>，便于进程之间共享数据。
- 只有当 close 调用关闭文件时，才会将页面页面写回磁盘，并解除映射，（当然 OS 也有可能会定期检查是否需要写回磁盘）

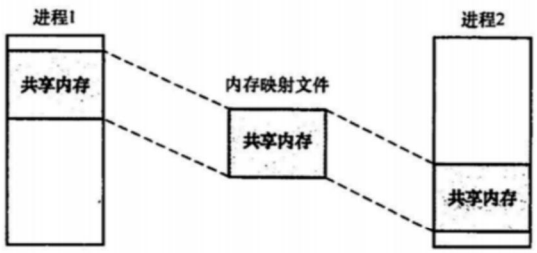


图 3.28　采用内存映射 I/O 的共享内存

章节手记

第 4 章 文件管理

4.1 三个重要问题 ★

- 可用于文件系统管理空闲磁盘块的数据结构有哪些?
  - 位图
  - 空闲磁盘块链
  - FAT 文件分配表（主要功能用于记录文件分配信息，兼职管理空闲盘块）
- 文件访问控制信息（ACL）的合理存储位置?
  - FCB
- 限制文件访问的因素
  - 由用户权限和文件属性共同限制

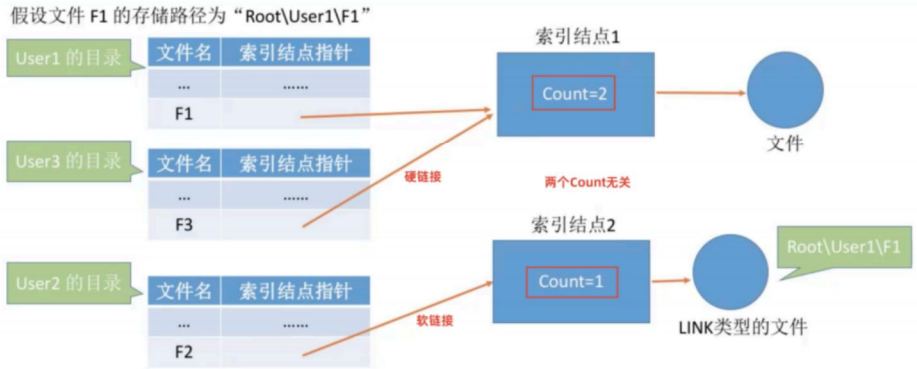
4.2 文件共享

4.2.1 硬链接

共享该文件的所有用户都直接指向真实文件的索引节点。

4.2.2 软链接★ ✪

只有文件主能够指向真实文件的索引结点。而其他共享用户都是自己先创建一个新的索引结点，该索引结点指向 LINK 文件，新索引结点的 Count 数与真实文件索引结点 Count 无关!!。



4.2.3 open 与 read 系统调用

- 先 open 后 read
- open(文件名，路径)：return 文件描述符
- read(文件描述符 fd，缓冲区首址 buf，读取字节数)

4.3 FAT 文件分配表

4.3.1 基本结构

FAT（File Allocation Table），是一种在“链式分配盘块”方式中用到的一种数据结构，它的本质类似于一个静态链表，数据结构由 < 物理块号，下一块 > 构成，当“下一块”为-1 通常表示为文件占用的最后一块盘块。

物理块号	下一块
0	1
1	-1
2	5
3	-1
4	23
5	0
……	
22	
23	3

FAT（文件分配表）

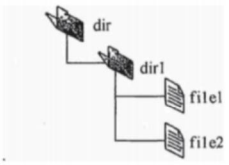
4.3.2 FAT 长度计算 ✪▲

本质就是计算 FAT 表项的数量，而 FAT 表项的数量又由 < 物理块号，下一块 > 能表示的最大范围决定，因此，核心就是知道这两个字段的位数。

例题 4.1 【2016 T47】以第 2 小问为例：

47. 某磁盘文件系统使用链接分配方式组织文件，簇大小为 4KB。目录文件的每个目录项包括文件名和文件的第一个簇号，其他簇号存放在文件分配表 FAT 中。

（1）假定目录树如下图所示，各文件占用的簇号及顺序如下表所示，其中 dir、dir1 是目录，file1、file2 是用户文件。请给出所有目录文件的内容。



文件名	簇号
dir	1
dir1	48
file1	100、106、108
file2	200、201、202

（2）若 FAT 的每个表项仅存放簇号，占 2 字节，则 FAT 的最大长度为多少字节？该文件系统支持的文件长度最大是多少？

（3）系统通过目录文件和 FAT 实现对文件的按名存取，说明 file1 的 106、108 两个簇号分别存放在 FAT 的哪个表项中。

（4）假设仅 FAT 和 dir 目录文件已读入内存，若需将文件 dir/ dir1/ file1的第 5000 个字节读入内存，则要访问哪几个簇？

解析 每个表项仅存簇号这一个字段，因此只需计算簇号的位数就知道能表示的最大范围

故 FAT 的最大长度 = 最大能表示的簇数 × 单个表项长度 $= 2^{16} * 2B = 128KB$

4.4 不同盘块组织方式的优缺点

组织方式	优点	缺点
连续分配	随机读写速度**最快**，空间利用率最高	文件的增删改不方便
链接分配	增删改方便	不支持随机读写，且指针占用额外空间
索引分配	支持随机读写（速度没连续分配快），且增删改也方便	索引结点占用额外空间

小布提示

若题目中提到了“文件不可修改/无需修改”等，这是一个强提示，通常都是可以选用连续分配方式的。

4.5 文件数量的计算 ▲

- 制约文件数量的因素有两个：
  - 空闲盘块数：空闲盘块越多，能存放的文件越多（千万注意一个文件不管多小，至少占用 1 个盘块）
  - 索引节点数：一个文件对应一个索引结点
  - 所以如果告诉你文件大小让你计算最多能存的文件数，用下面的步骤：
    1. 计算出空闲盘块能存放的文件数量 n_1
    2. 计算出系统索引结点数 n_2

3. 最大文件数量 $= \min\{n_1, n_2\}$

✍ 章节手记

第 5 章 IO 管理

5.1 十一个重要问题★

1. 不会产生磁臂黏着的算法：FCFS
2. IO 软件结构层次：用户、无关、驱动、中断

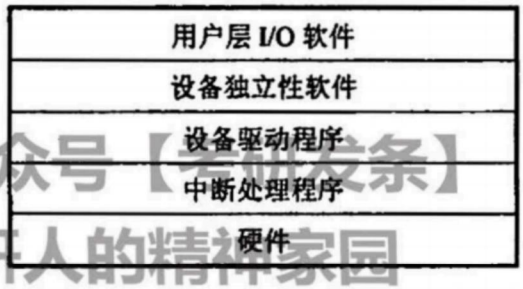


图 5.5 I/O 层次结构

3. IO 层次

> **小布提示**
>
> 牢记：中断处理程序与硬件直接打交道 ♣

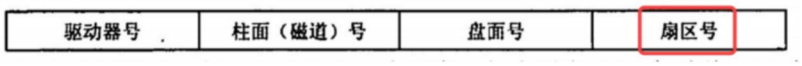
4. 通道，设备控制器和设备之间的关系？
   - 通道控制设备控制器，设备控制器控制设备工作
5. 设备分配时应考虑的问题：
   - 设备固有属性：独占设备/共享设备/虚拟设备
   - 设备分配算法
   - 设备安全性：安全分配方式/不安全分配方式
6. 在内存中设置磁盘缓冲区的目的：减少 IO 次数
7. 管理缓冲区中首要考虑的问题是？
   - 进程与缓冲区之间的同步。（因为不管缓冲区大小、数量如何，你都得遵循缓冲区未空可读，未满可写的同步原则）
8. 缓冲技术的缓冲池所在位置：主存
9. 每台设备在系统中都有一个独立的编号，称为？
   - 绝对号
10. 光盘读写特性：可随机读，但不可随机写
11. 初始化中断向量表是由 OS 完成的吗？
    - 是的，这一步操作仅仅只是初始化存放中断向量的那张表而已，并非中断服务寻址（硬件完成）

5.2 IO 设备的分类

- 块设备：以“块”为交换单位，高速、可寻址、支持随机读写
- 字符设备：以“字符”为交换单位、低速、不可寻址、不支持随机读写

5.3 磁盘管理

5.3.1 磁盘的地址结构



5.3.2 簇号与磁盘地址的转换 ✪▲

- 柱面 (磁道) 号 = 簇号/单磁道簇数/单柱面磁道数
- 盘面号 = 簇号/单磁道簇数 % 单柱面磁道数
- 扇区号 = 簇号 × 单簇包含扇区数 % 单柱面扇区数

例题 5.1 【2019 年 T44】

44.（7 分）某计算机系统中的磁盘有 300 个柱面，每个柱面有 10 个磁道，每个磁道有 200 个扇区，扇区大小为 512B。文件系统的每个簇包含 2 个扇区。请回答下列问题：

（1）磁盘的容量是多少？

（2）假设磁头在 85 号柱面上，此时有 4 个磁盘访问请求，簇号分别为 100 260、60 005、101 660 和 110 560。若采用最短寻道时间优先（SSTF）调度算法，则系统访问簇的先后次序是什么？

（3）第 100 530 簇在磁盘上的物理地址是什么？将簇号转换成磁盘物理地址的过程是由 I/O 系统的什么程序完成的？

解析 (1):

磁盘容量 = 总扇区数 * 单位扇区大小 $= 300 * 10 * 200 * 512B = 3 * 10^5 KB$

(2):

分别利用公式计算 4 组簇号的磁道号:

100260 : 磁道号 $= 100260/100/10 = 100$

60005 : 磁道号 $= 60005/100/10 = 60$

101660 : 磁道号 $= 101660/100/10 = 101$

110560 : 磁道号 $= 110560/100/10 = 110$

由于当前所在磁道为 85，根据 SSTF 算法，磁道访问顺序为:

100260、101660、110560、60005

(3):

磁道号 $= 100530/100/10 = 100$

盘面号 $= 100530/100\%10 = 5$

簇号 $= 100530 * 2\%(100 * 2) = 60$

由驱动程序完成

5.3.3 磁盘的初始化工作 ★

- 低级格式化（物理）：
  - 划分扇区
  - 确定扇区所需数据结构：包括扇区校验码等（低级格式化通常采用特殊的数据结构来填充扇区）
- 磁盘分区：
  - 将磁盘的若干柱面划分为一个分区
- 高级格式化（逻辑）：
  - 建立文件系统：包括根目录等
  - 对空闲盘块信息初始化（位示图、空闲分区表等）

例题 5.2 【2017 年 T29】

29. 下列选项中，磁盘逻辑格式化程序所做的工作是_____。

I. 对磁盘进行分区

II. 建立文件系统的根目录

III. 确定磁盘扇区校验码所占位数

IV. 对保存空闲磁盘块信息的数据结构进行初始化

A. 仅 II　　B. 仅 II、IV　　C. 仅 III、IV　　D. 仅 I、II、IV

解析 答案：B

I 属于分区，分区和格式化是两件不同的事情

III 是低级格式化干的事情

5.3.4 磁盘调度算法 ▲

小布提示

首先注意：408 中的 SCAN=LOOK；C-SCAN=C-LOOK

- SCAN/LOOK：来回都干活
- C-SCAN/C-LOOK：只在一个方向干活

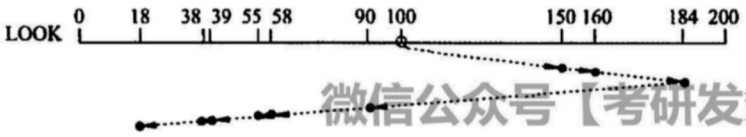


图 5.20　LOOK 磁盘调度算法

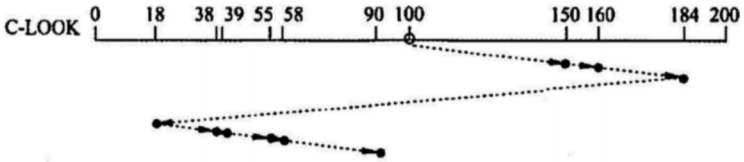


图 5.21　C-LOOK 磁盘调度算法

5.4 固态硬盘 SSD

5.4.1 原理

基于闪存 FlashMemory 技术属于电可擦除 ROM(EEPROM)

5.4.2 组成

- 闪存翻译层：负责翻译逻辑块号，找到对应的页（Page）
- 存储介质：多个闪存芯片（Flash Chip），每个 Chip 包含多个块（Block），每个 Block 包含多个 Page

小布提示

注意：在传统机械硬盘中，读写以“块”为基本单位，而在 SSD 中的“页”就相当于 HDD 中的块。

5.4.3 读写特性

- 以 Page 为单位读取
- 以 Block 为单位擦除
- 读快写慢，如果要写入的页已经有数据，则需要先将块内其他复制到新的块中，再写入该新块

5.4.4 磨损均衡技术（Wear Leveling）

总结

思想：把“擦除”平均分布在各个块中，以提升使用寿命。

- 动态磨损均衡：写入数据时触发，优先选择擦除次数少的块
- 静态磨损均衡：SSD 在后台自己检测，判别。把读多写少的数据放到老旧的块中。让新块承担写任务。

5.5 缓冲区✪▲

5.5.1 单缓冲

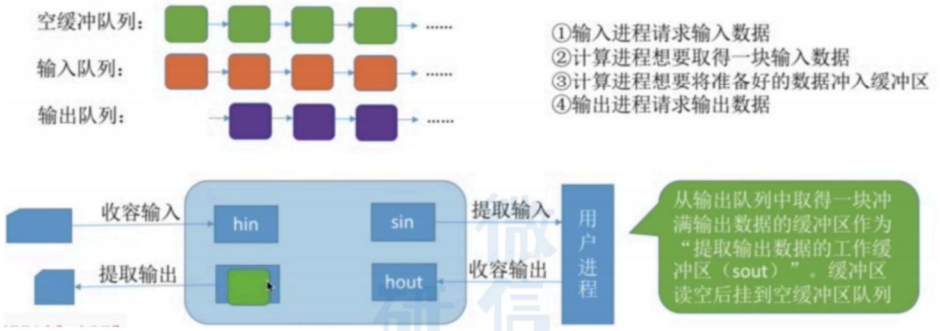
处理平均时间：$\max(C, T) + M$

5.5.2 双缓冲

处理平均时间：$\max(T, C + M)$

5.5.3 缓冲池

- 三种队列
  - 空缓冲队列
  - 输入队列
  - 输出队列
- 四种缓冲区
  - hin：收容输入
  - hout：收容输出
  - sin：提取输入
  - sout：提取输出



5.5.4 磁盘高速缓存

- 工作原理：从主存中划出一片空间来存储磁盘中读出的块信息（所以从逻辑上来看磁盘高速缓存是磁盘，但是物理上来看是驻留在主存当中的盘块）
- 高速缓存在内存中的两种形式：
  - 在内存中单独开辟空间
  - 把未利用的空间作为缓冲池

5.5.5 高速缓存与缓冲区对比

表 5.1　高速缓存和缓冲区的对比

		高速缓存	缓冲区
相同点		都介于高速设备和低速设备之间	
区别	存放数据	存放的是低速设备上的某些数据的复制数据，即高速缓存上有的，低速设备上面必然有	存放的是低速设备传递给高速设备的数据（或相反），而这些数据在低速设备（或高速设备）上却不一定有备份，这些数据再从缓冲区传送到高速设备（或低速设备）
	目的	高速缓存存放的是高速设备经常要访问的数据，若高速设备要访问的数据不在高速缓存中，则高速设备就需要访问低速设备	高速设备和低速设备的通信都要经过缓冲区，高速设备永远不会直接去访问低速设备

5.6 IO 接口 ★

- 状态端口：用来存放 IO 设备状态
- 控制端口：用来存放控制命令

在一次完整的读写过程中，由于 CPU 会先查询状态，（当设备空闲时）才会向 IO 接口发出控制命令，且在本次操作结束前都不会再使用到设备的状态信息，所以两个端口的使用时间互不冲突，可以共用一个寄存器。

小布提示

状态端口和控制端口可以共用一个寄存器。

5.7 SPOOLing 技术 ✪

5.7.1 组成 ★

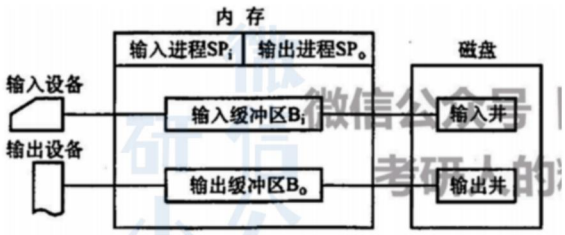


图 5.12　SPOOLing 系统的组成

5.7.2 组件管理程序

- 预输入程序：管理输入缓冲区
- 缓输出程序：管理输出缓冲区
- 井管理程序：管理输入/输出井

5.7.3 特点

- 提高 IO 速度
- 将独占设备改成了共享设备
- 实现了虚拟设备的功能（每个用户都认为自己独占了设备）
- 由 OS 控制设备与 I/O 井之间的数据传送（因为 SPOOLing 的内部实现细节是对用户透明的，也正是如此，才会让用户觉得自己在独占设备）

5.7.4 要求

- 需要外存支持

- 需要多道程序设计支持

章节手记