

操作系统复习

第1章 计算机系统概述

- 1 指令执行的基本指令周期
- 2 中断分类与中断处理过程
 - 2.1 中断的定义
 - 2.2 中断分类
 - 2.3 中断的意义
 - 2.4 无中断
 - 2.5 有中断
 - 2.6 中断和指令周期
 - 2.7 中断处理的过程
- 3 处理多中断的两种方法
 - 3.1 顺序中断处理（禁止中断）
 - 3.2 嵌套中断处理

4 存储器

- 4.1 存储器层次
 - 层次结构的特点
- 4.2 二级存储器（Cache - 内存）下计算内存的平均存取时间

5 高速缓存

动机

6 程序的局部性原理

第1章 计算机系统概述作业

- 1 什么是中断？
- 2 多中断的处理方式是什么？
- 3 什么是高速缓存？
- 4 习题

第2章 操作系统概述

1 操作系统的发展过程及衍生出来的操作系统类型（多道批处理系统，分时系统）

- 1.1 操作系统的演化
 - 1.1.1 串行处理时期（20世纪40年代到50年代中期）
 - 1.1.2 简单批处理系统
 - 1.1.3 多道批处理系统——多道程序设计（多任务处理）
- 1.2 分时系统

- 1.2.1 为什么要有分时系统？
- 1.2.2 分时系统的定义
- 1.2.3 分时和多道程序设计引发的新问题

2 与单道串行处理相比，多道程序设计如何提高资源利用率

- 3 多个作业并发执行时资源利用率的计算
 - 3.1 单道运行时间关系图
 - 3.2 多道运行时间关系图（不抢占）

第2章 操作系统概述作业

- 1 操作系统设计的三个目标是什么？
- 2 什么是操作系统的内核？
- 3 什么是多道程序设计？
- 4 习题

第3章 进程描述和控制

1 五状态进程模型，包含两个挂起态的模型，状态转换

- 1.1 五状态进程模型
- 1.2 被挂起的进程

2 进程映像，进程控制块 PCB（进程属性的集合）

- 2.1 进程映像
- 2.2 进程控制块PCB
 - 2.2.1 进程标识信息
 - 2.2.2 进程状态信息
 - 2.2.3 进程控制信息

3 进程的创建与终止

3.1 创建进程的步骤

3.2 进程终止的原因

4 进程切换

4.1 切换时机

4.2 模式切换

4.3 完整的进程切换步骤

5 执行模式的切换：用户态和系统态

5.1 无进程的内核

5.2 在用户进程中执行

5.3 基于进程的操作系统

6 UNIX 中，父进程通过系统调用 fork() 创建子进程

第3章 进程描述和控制作业

1 对于五状态进程模型，请简单定义每个状态。

2 操作系统创建一个新进程所执行的步骤是什么？

3 模式切换和进程切换有什么区别？

4 习题

第4章 线程

1 进程和线程区别（资源分配单位，调度运行单位）

1.1 进程的两个特点

1.2 线程和进程的区别

2 线程的优点，线程的三种状态（运行，就绪，阻塞）

2.1 线程的优点

2.2 线程的功能

2.2.1 线程状态

2.2.2 一个线程阻塞是否会导致整个进程阻塞？

3 用户级线程和内核级线程的特点

3.1 用户级线程

3.1.1 优点

3.1.2 缺点

3.2 内核级线程

3.2.1 优点

3.2.2 缺点

第四章 线程作业

1 哪些资源通常被一个进程中的所有线程共享？

2 列出用户级线程相对于内核线程的三个优点。

3 列出用户级线程相对于内核线程的两个缺点。

第5章：并发：互斥和同步

1 互斥的概念

2 临界资源与临界区

3 信号量含义，semWait、semSignal 含义

4 信号量原语定义（图 5.3）

5 用信号量实现互斥与同步

6 有限缓冲的生产者/消费者问题（图 5.13）

6.1 信号量问题的补充练习一

6.2 信号量问题的补充练习二

7 进程间通过“消息传递”交换信息：无阻塞 send 和阻塞 receive

第5章：并发：互斥和同步作业

1 写出信号量定义，semWait 和 semSignal 原语，以及用信号量实现互斥的伪代码

习题1

习题2

习题3

第6章：并发：死锁和饥饿

1 死锁原因：竞争资源、进程推进顺序不当

2 资源分配图（若死锁，则资源分配图必有环路，但有环路时不一定死锁）

3 死锁的四个必要条件

4 三种处理方法：预防，避免，检测和恢复

4.1 死锁预防

4.1.1 互斥

4.1.2 占有且等待

4.1.3 不可抢占

4.1.4 循环等待

4.2 死锁避免

4.2.1 优点

4.2.2 缺点

4.3 死锁检测

4.3.1 检测时机

4.3.2 恢复

5 银行家算法：要求能够判断现在是否安全，某进程请求资源是否能够满足

5.1 银行家算法举例1

5.2 银行家算法举例2

6 用信号量解决不死锁的哲学家就餐问题

第6章：并发：死锁和饥饿作业

1 产生死锁的四个条件是什么？

2 死锁避免、检测和预防之间的区别是什么？

3 习题1

4 习题2

5 习题3

第7章：内存管理

1 固定分区，动态分区分配策略 - 首次适配、下次适配、最佳适配

1.1 固定分区——分区大小

1.1.1 大小相等的分区

1.1.2 大小不等的分区

1.2 固定分区——放置算法

1.2.1 大小相等的分区

1.2.2 大小不等的分区

1.3 固定分区方案的缺陷

1.4 动态分区

1.4.1 缺陷

1.4.2 外部碎片解决方法

1.5 动态分区——放置算法

2 内部碎片，外部碎片

3 伙伴系统的分配与回收

4 重定位：将逻辑地址转换为物理地址

5 存储保护与越界：基址+界限寄存器

方法

存储键保护

界限寄存器（下页图）

6 分页：基本原理，逻辑地址结构，页和页框，页表，地址转换

6.1 基本原理

6.2 分页与固定分区的区别

6.3 逻辑地址结构

6.4 页表共享

7 分段：基本原理，逻辑地址结构，段表，地址转换

7.1 基本原理

7.2 分段与动态分区的区别

7.3 逻辑地址到物理地址转换

第7章：内存管理作业

1 为什么需要重定位进程的能力？

2 页和页框之间有什么区别？

3 习题1

4 习题2

5 习题3

6 习题4

第8章：虚拟内存

1 虚拟地址概念，实地址概念

- 2 虚拟分页：基本原理，虚实地址转换
- 3 缺页中断处理过程
- 4 转换检测缓冲区 TLB（快表）。根据内存访问时间、TLB 访问时间和 TLB 命中率，求将逻辑地址转换成物理地址并访问内存数据所需的有效访问时间（见作业）
 - 4.1 转换检测缓冲区 (translation look-aside buffer, TLB)
 - 4.2 采用快表的地址转换
- 5 虚拟分段和虚拟段页式的基本原理
 - 5.1 虚拟分段的基本原理
 - 5.2 虚拟段页式的基本原理
- 6 虚拟分页的置换算法：最佳置换 OPT、LRU、先进先出 FIFO
 - 6.1 最佳置换 OPT
 - 6.2 最近最少使用 LRU
 - 6.3 先进先出 FIFO
- 7 置换过程及缺页次数的计算（注：计算页框填满之前和之后发生的总缺页次数即可）
- 8 抖动

第8章：虚拟内存作业

- 1 解释什么是抖动。
- 2 转换检测缓冲区的目的是什么？
- 3 驻留集和工作集有什么区别？
- 4 习题1
- 5 习题2

第9章：单处理器调度

- 1 处理器调度的类型 - 长程，中程，短程
 - 1.1 长程调度
 - 1.1.1 何时调度？
 - 1.1.2 调度哪个？
 - 1.2 中程调度
 - 1.3 短程调度
- 2 调度准则与指标
- 3 非抢占式调度、抢占式调度
 - 3.1 非抢占式调度
 - 3.2 抢占式调度
- 4 调度算法：先来先服务(FCFS)、轮转RR、最短进程优先(SPN)、最高响应比优先(HRRN)。计算“周转时间”、“归一化周转时间(带权周转时间 Tr/Ts)”及所有作业的平均值
 - 4.1 先来先服务 FCFS
 - 4.2 轮转 RR q=1
 - 4.3 轮转 RR q=4
 - 4.4 最短进程优先 SPN
 - 4.5 最高响应比优先 HRRN

第9章：单处理器调度作业

- 1 简要描述三种类型的处理器调度。
- 2 抢占式和非抢占式调度有什么区别？

第10章：多处理器、多核和实时调度

- 1 多处理器系统中，采用简单的 FCFS 或“静态优先级+FCFS”调度算法就足够了
- 2 实时任务分类：硬、软，周期性、非周期性

第10章：多处理器、多核和实时调度作业

- 1 习题1

第11章：I/O管理和磁盘调度

- 1 程序控制 I/O：CPU 忙等 I/O 结束，CPU 与设备串行工作
- 2 中断驱动 I/O：各种设备通用，中断次数多
- 3 直接存储器访问 DMA 原理与 I/O 过程
- 4 缓冲 buffer 的主要作用：缓和 CPU 与 I/O 设备间速度不匹配矛盾，提高并行性
- 5 磁盘访问时间：寻道时间，旋转延迟时间，传输时间
- 6 磁盘调度算法：先进先出，最短服务时间优先算法(SSTF)，电梯。计算平均寻道长度
 - 6.1 先进先出 FIFO
 - 6.2 最短服务时间优先算法(SSTF)
 - 6.3 电梯SCAN
 - 6.4 C-SCAN调度

7 RAID 的核心技术：条带化， 并行访问， 块交叉校验， 镜像。 RAID 0, RAID 1

7.1 条带化

7.2 镜像

7.3 错误校正码

7.4 奇偶校验位

7.5 块交叉校验

第11章：I/O管理和磁盘调度作业

1 列出并简单定义执行I/O的三种技术。

第12章：文件管理

1 树型目录，文件共享

1.1 树型目录

1.2 文件共享

1.2.1 访问权限

1.2.2 同时访问

2 三种文件分配方法：连续分配，链接分配，索引分配

2.1 连续分配

2.2 链接分配（指针）

2.3 索引分配（索引表）

3 索引分配对文件尺寸的影响

4 磁盘空闲空间管理：位图

5 UNIX 中的文件控制块：索引节点 i-node

第12章：文件管理作业

1 列出并简单定义三种文件分配方法

2 习题

错题整理1

错题整理2

操作系统常用缩写总结

一、计算机系统概述

二、操作系统概述

三、进程描述和控制

四、线程、对称多处理和微内核

五、并发性：互斥和同步

六、并发性：死锁和饥饿

七、内存管理

八、虚拟内存

九、单处理器调度

十、多处理器和实时调度

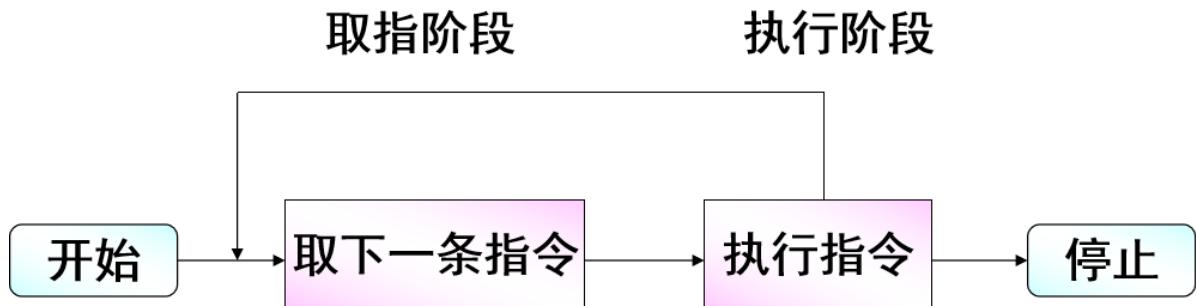
十一、I/O管理和磁盘调度

十二、文件管理

操作系统复习

第1章 计算机系统概述

1 指令执行的基本指令周期



基本指令周期

2 中断分类与中断处理过程

2.1 中断的定义

中断是一种机制，即允许其它模块（I/O、存储器）在处理器正常处理过程中打断其工作。

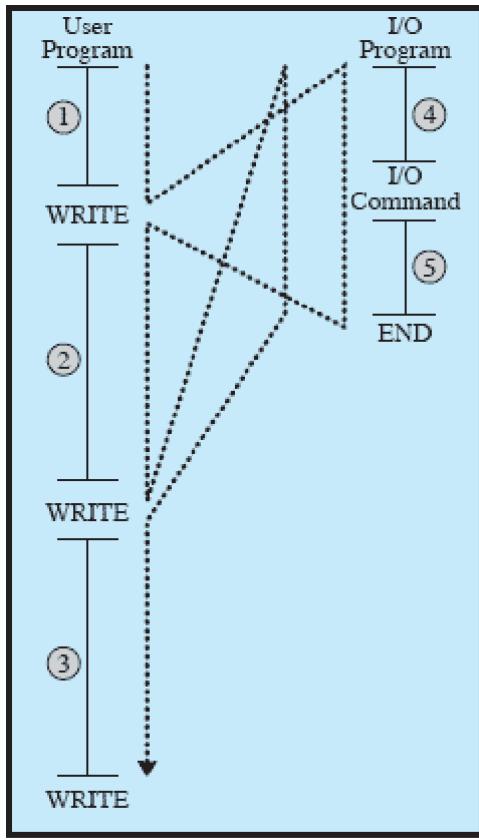
2.2 中断分类

- 程序中断
- 时钟中断
- I/O中断
- 硬件失效中断

2.3 中断的意义

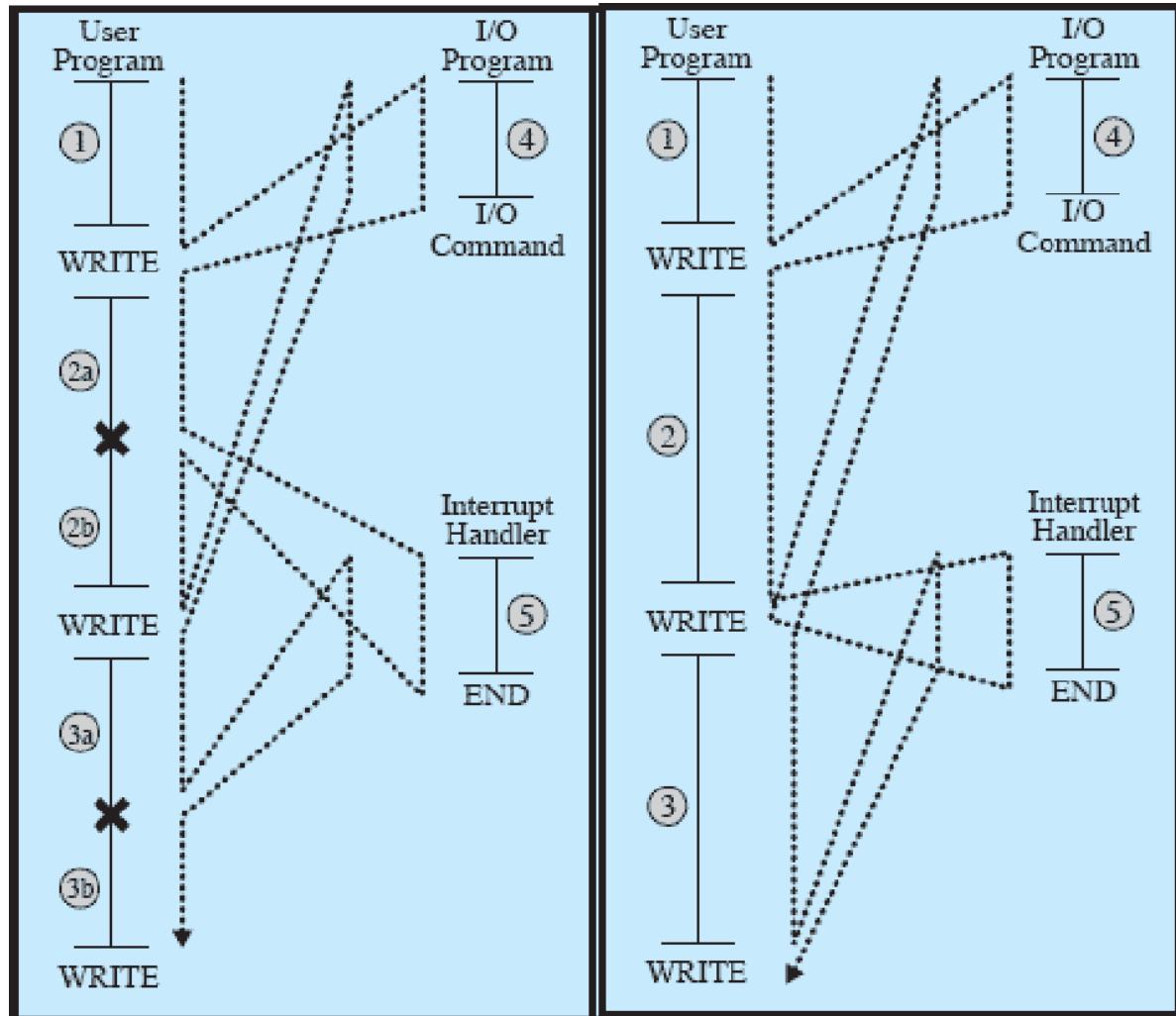
- 中断是提高处理器效率的一种手段。
- 利用中断功能，处理器可以在I/O操作的执行过程中执行其他指令。
- I/O操作和用户程序中指令的执行是并发的。

2.4 无中断



(a) No interrupts

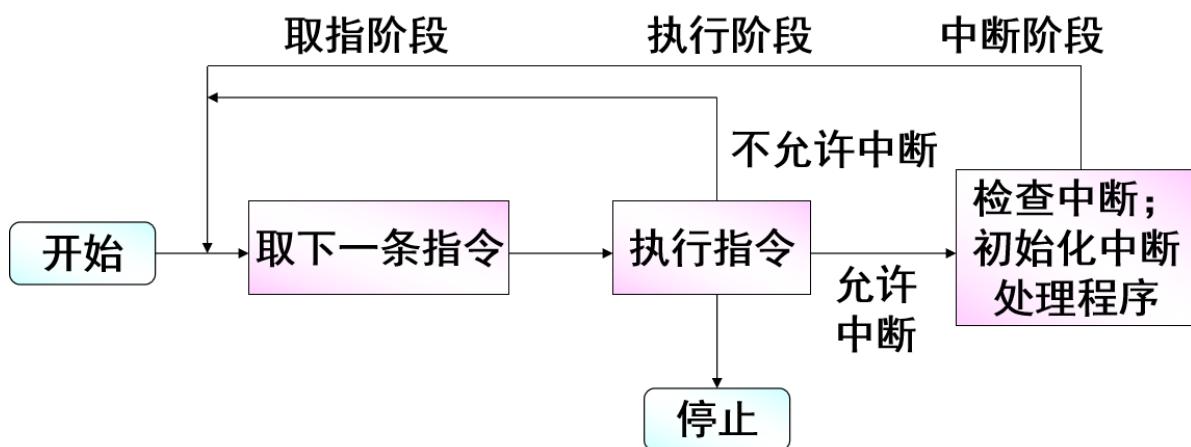
2.5 有中断



(b) Interrupts; short I/O wait

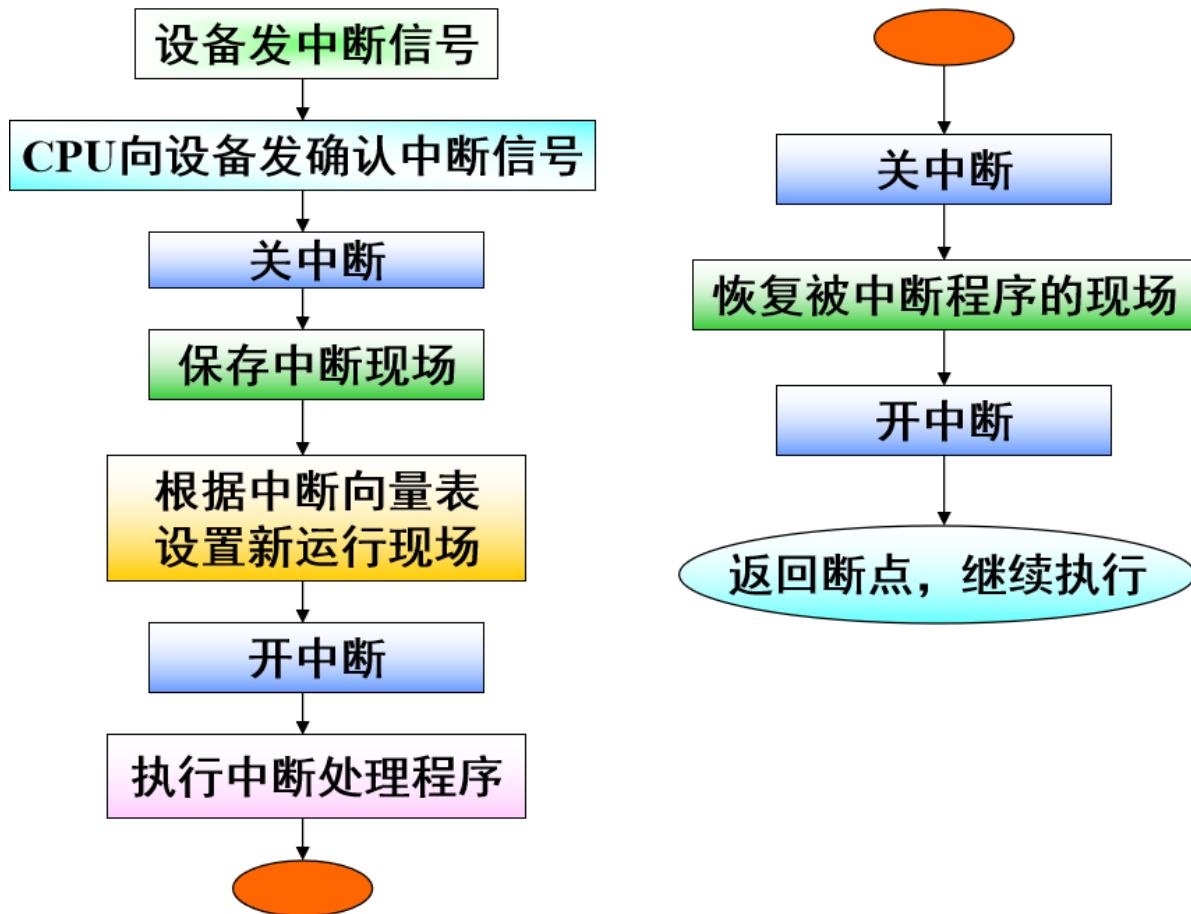
(c) Interrupts; long I/O wait

2.6 中断和指令周期



中断和指令周期

2.7 中断处理的过程

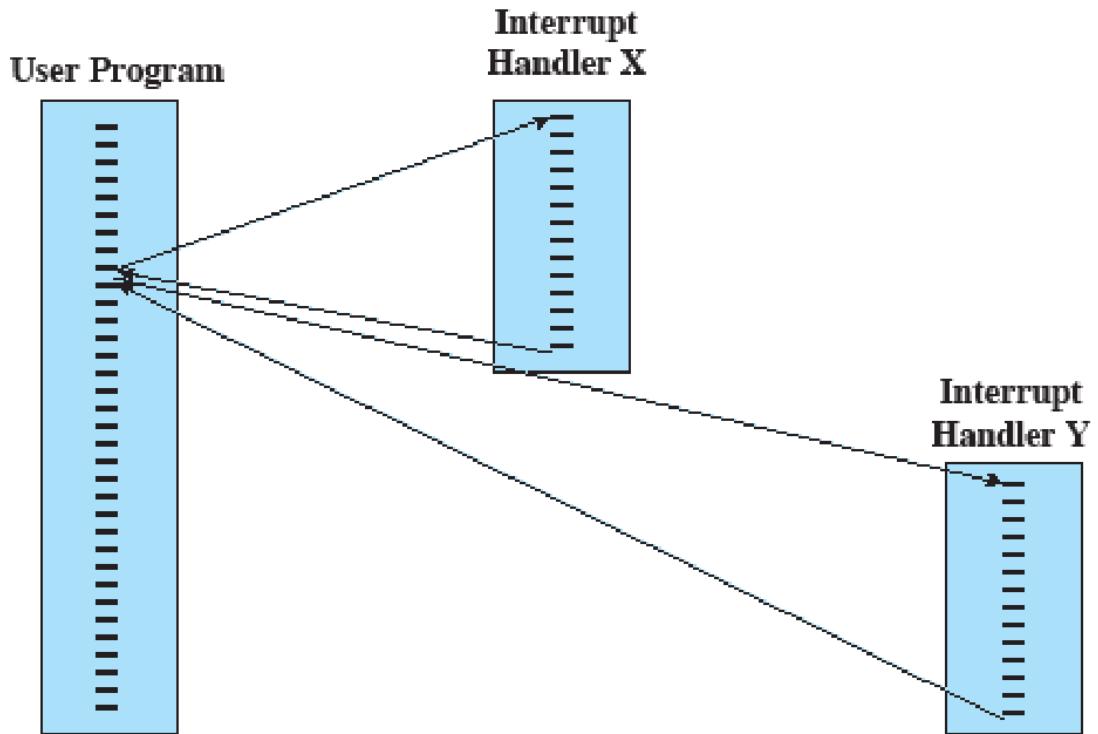


3 处理多中断的两种方法

3.1 顺序中断处理 (禁止中断)

当正在处理一个中断时，禁止中断（对任何新的中断请求信号不予理睬，处理完这个再处理下个）

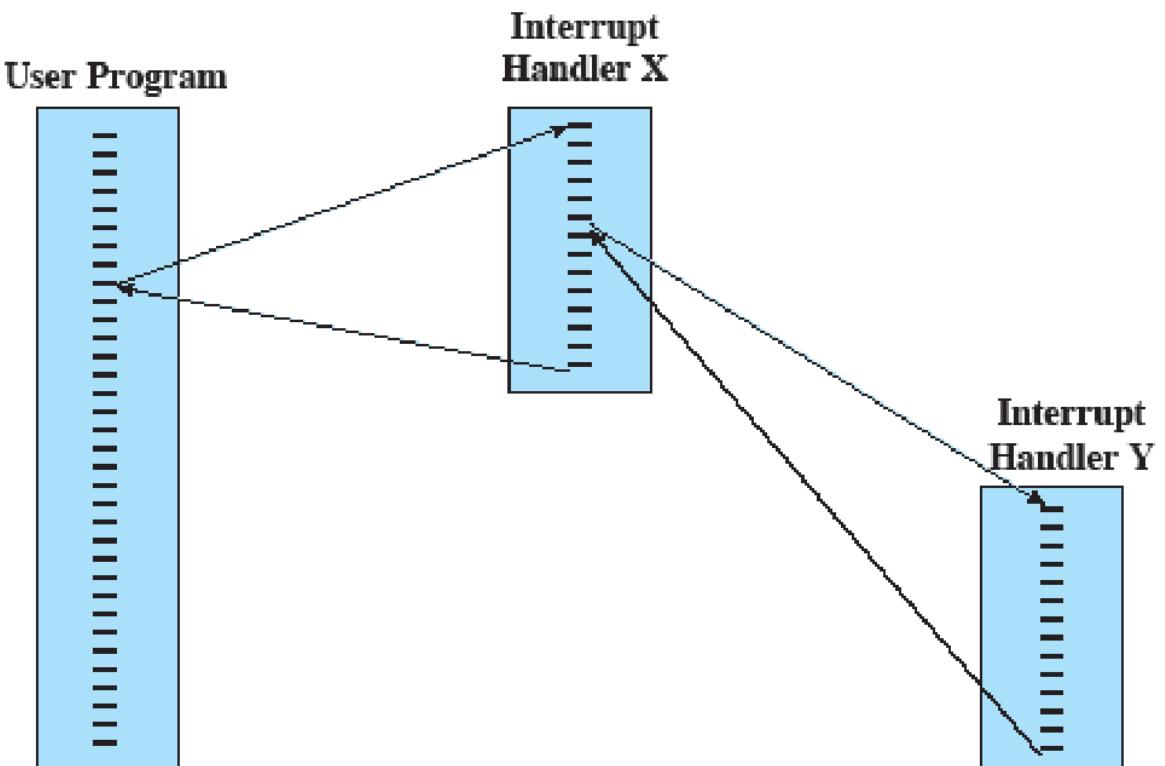
缺点：没有考虑相对优先级和时间限制的要求



(a) Sequential interrupt processing

3.2 嵌套中断处理

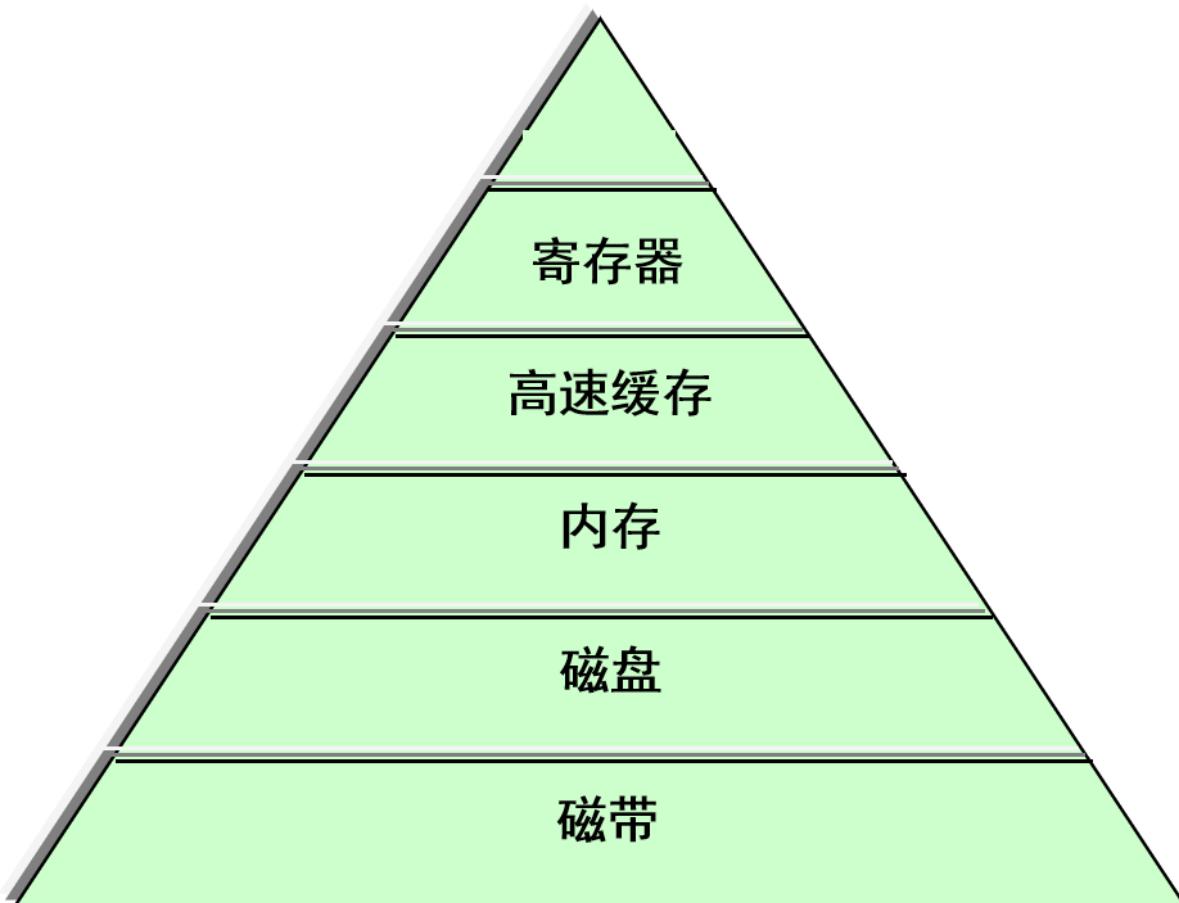
定义中断优先级，允许高优先级的中断打断低优先级的中断处理程序的运行。



(b) Nested interrupt processing

4 存储器

4.1 存储器层次



层次结构的特点

- 由上至下：每“位”的价格递减、容量递增、存取时间递增、处理器访问存储器的频率递减
- 容量较大、价格较便宜的慢速存储器，是容量较小、价格较贵的快速存储器的后备
- 存储器层次结构能够成功的关键：低层访问频率递减

4.2 二级存储器（Cache - 内存）下计算内存的平均存取时间

假定有一个二级存储器（内存+高速缓存），内存存取时间为1us，高速缓存存取时间为0.1us，且高速缓存的命中率为95%，则访问

一个字节的平均存取时间为：

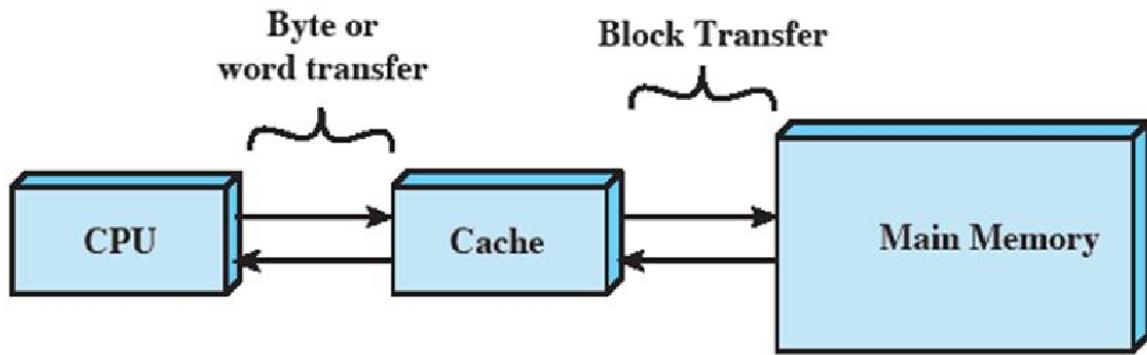
$$0.95 \times 0.1 + 0.05 \times (0.1 + 1) = 0.15 \text{ (us)}$$

5 高速缓存

为加快内存访问速度，CPU首先访问Cache，不命中时再访问内存且复制进Cache

动机

- 指令执行期间，处理器需要多次访问内存；
- 处理器和内存的速度不匹配，处理器速度的提高一直快于内存访问速度的提高——处理器执行指令的速度受限；
- 利用局部性原理，在处理器和内存之间提供一个容量小而速度快的存储器——高速缓存。



6 程序的局部性原理

指程序在执行过程中的一个较短时间内，所执行的指令地址或操作数地址分别局限于一定的存储区域中。

又可细分时间局部性和空间局部性。

时间局部性：最近访问过的程序代码和数据很快又被访问。

空间局部性：某存储单元被使用之后，其相邻的存储单元也很快被使用。

第1章 计算机系统概述作业

1 什么是中断？

中断是一种机制，允许其它模块（I/O、存储器）在处理器（CPU）正常处理过程中打断其工作。

2 多中断的处理方式是什么？

- 顺序中断处理：当正在处理一个中断时，禁止中断。
- 嵌套中断处理：允许高优先级的中断打断低优先级的中断处理程序的运行。

3 什么是高速缓存？

高速缓存是处理器和内存之间的一个容量小而速度快的存储器。利用局部性原理，解决处理器和内存速度不匹配的问题。

4 习题

一台计算机包括高速缓存、内存和一个用做虚拟存储器的磁盘。如果要存取的字在高速缓存中，则存取需要20ns；如果该字在内存中而不

在高速缓存中，则把它载入高速缓存需要60ns（包括初始检查高速缓存的时间），然后再重新开始存取；如果该字不在内存中，则需要

12ms从磁盘中取出该字，复制到高速缓存中还需要60ns，然后再重新开始存取。高速缓存的命中率为0.9，内存的命中率为0.6，则该系

统中存取一个字的平均存取时间是多少（单位：ns）？

	概率	所需时间 (ns)
在高速缓存	0. 9	20
不在高速缓存但在内存	$0.1 \times 0.6 = 0.06$	$60 + 20 = 80$
不在高速缓存和内存	$0.1 \times 0.4 = 0.04$	$12\text{ms} + 60 + 20 = 12,000,080$

平均存取时间为： $0.9 \times 20 + 0.06 \times 80 + 0.04 \times 12000080 = 480026 \text{ ns}$

常见时间单位换算：

1秒(s) = 1000 毫秒(ms)

1秒(s) = 1,000,000 微秒(μs)

1秒(s) = 1,000,000,000 纳秒(ns)

第2章 操作系统概述

1 操作系统的发展过程及衍生出来的操作系统类型（多道批处理系统，分时系统）

1.1 操作系统的演化

1.1.1 串行处理时期（20世纪40年代到50年代中期）

没有操作系统，用户必须顺序访问计算机。存在两个主要问题：

- 调度
- 准备时间

1.1.2 简单批处理系统

- 第一个操作系统（第一个批处理操作系统）：20世纪50年代中期，General Motors开发，用于 IBM701
- 中心思想：监控程序

用户作业——计算机操作员——将作业组织成批——输入——监控程序

每个程序处理完后返回到监控程序，同时，监控程序自动加载下一个程序。

- 作业控制语言 (JCL)：为监控程序提供指令。每个作业中的指令以JCL的基本形式给出。
- 涉及到的硬件功能

内存保护

定时器

特权指令

中断

- 用户态——用户程序
- 内核态——监控程序

1.1.3 多道批处理系统——多道程序设计（多任务处理）

- 内存同时保存多个程序，当一个作业需要等待I/O时，处理器可以切换到另一个不需要等待I/O的作业。
- 提高CPU的利用率。
- 需要中断技术、内存管理、进程调度等方面的支持。

1.2 分时系统

1.2.1 为什么要有分时系统？

批处理用户不能干预自己程序的运行，无法得知程序的运行情况，不利于程序调试和排错。

1.2.2 分时系统的定义

- 允许多个联机用户同时使用一个计算机系统进行交互式计算。
- 时钟中断，时间片技术。

1.2.3 分时和多道程序设计引发的新问题

- 作业的相互干扰
- 文件系统的保护
- 处理资源的竞争

2 与单道串行处理相比，多道程序设计如何提高资源利用率

由于任何一道作业的运行总是交替地串行使用CPU，外设等资源，即使使用一段时间的CPU，然后使用一段时期的I/O设备，

由于使用多道程序设计技术，加之对多道程序实施合理的运行调度，则可以实现CPU和I/O设备的高度并行，

可以大大提高CPU与外设的利用率。

3 多个作业并发执行时资源利用率的计算

若主存中有3道程序A、B、C，它们按A、B、C优先次序运行，各程序的计算轨迹为：

A：计算（20）、I/O（30）、计算（10）

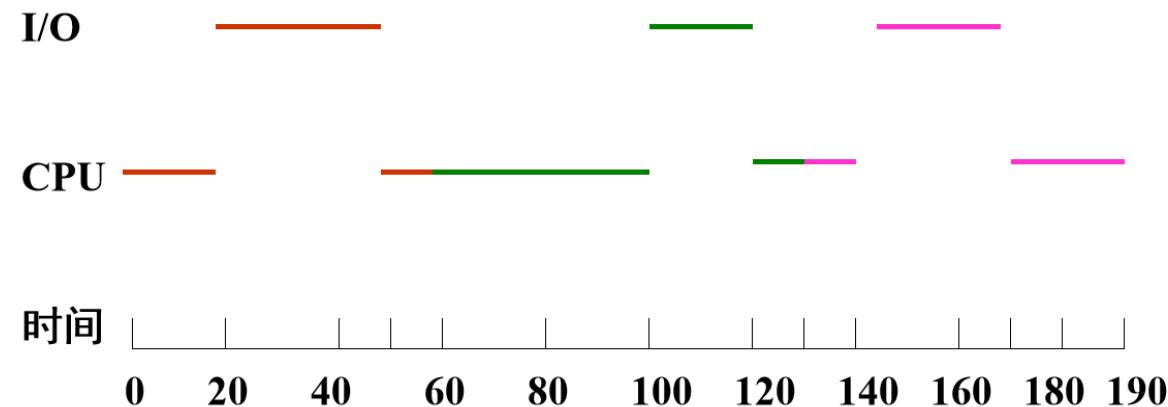
B：计算（40）、I/O（20）、计算（10）

C：计算（10）、I/O（30）、计算（20）

如果三道程序都使用相同设备进行I/O（即程序用串行方式使用设备，调度开销忽略不计）。

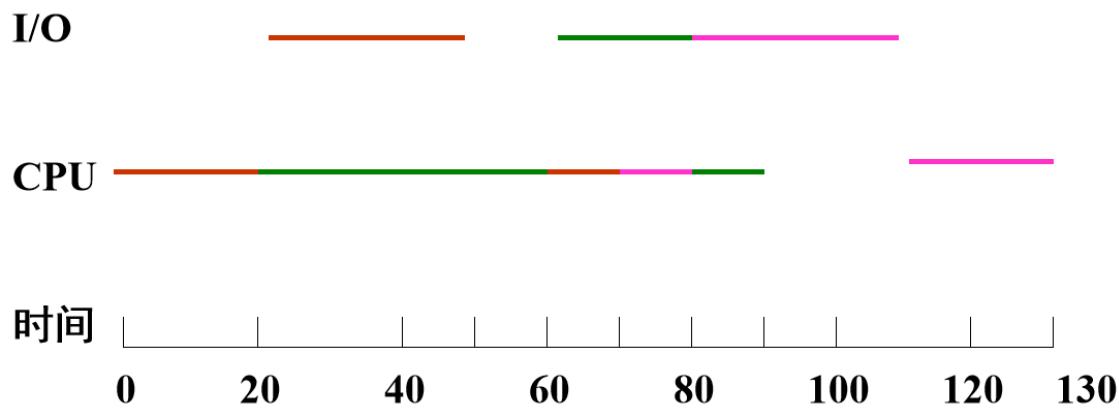
试分别画出单道和多道运行的时间关系图。两种情况下，CPU的平均利用率各为多少？

3.1 单道运行时间关系图



单道CPU利用率为 $(190 - 80) / 190 = 57.9\%$

3.2 多道运行时间关系图 (不抢占)



多道CPU利用率为 $(140 - 30) \div 140 = 78.6\%$

第2章 操作系统概述作业

1 操作系统设计的三个目标是什么？

- 方便：使计算机更易于使用。
- 有效：允许以更有效的方式使用计算机系统资源。
- 扩展能力：允许在不妨碍服务的前提下有效地开发、测试和引进新的系统功能。

2 什么是操作系统的内核？

内核是操作系统的一部分，包含操作系统中最重要的软件功能。内核常驻内存，运行于特权模式下，能够响应进程的调用和设备的中断。

3 什么是多道程序设计？

多道程序设计是现代操作系统的主要方案，允许多道程序同时在内存空间，使得单个CPU可以交替执行多个程序。

4 习题

在单CPU和两台I/O设备 (I1和I2) 的多道程序设计环境下，同时投入3个作业运行。其执行轨迹如下：

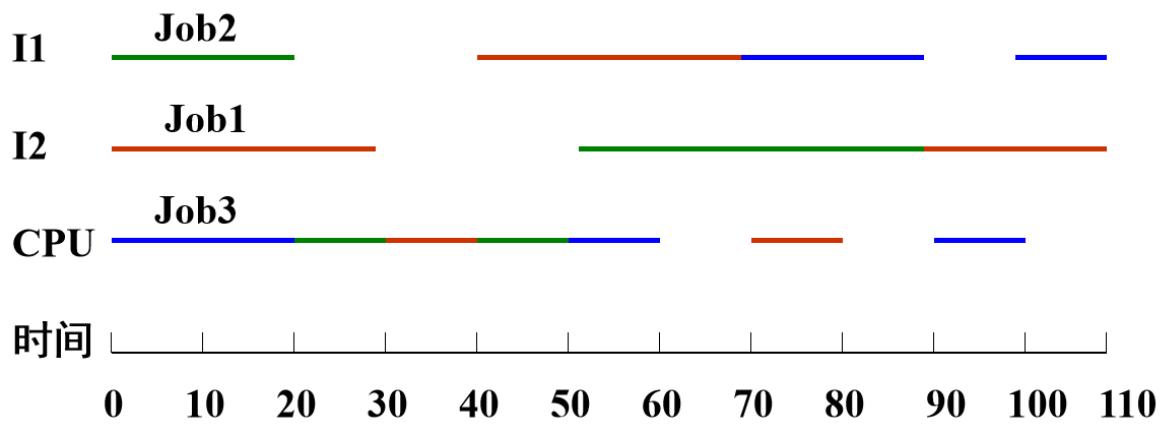
Job1: I2(30ms), CPU(10ms), I1(30ms), CPU(10ms), I2(20ms)

Job2: I1(20ms), CPU(20ms), I2(40ms)

Job3: CPU(30ms), I1(20ms), CPU(10ms), I1(10ms)

设CPU, I1和I2都能并行工作，作业优先级从高到低依次为 Job1, Job2, Job3，优先级高的作业可以抢占优先级低的作业的CPU，但不可抢占I1和I2。

求：(1) 从作业投入到完成，CPU的利用率。(2) I1和I2的设备利用率。



CPU利用率: $(110 - 30) \div 110 = 72.7\%$

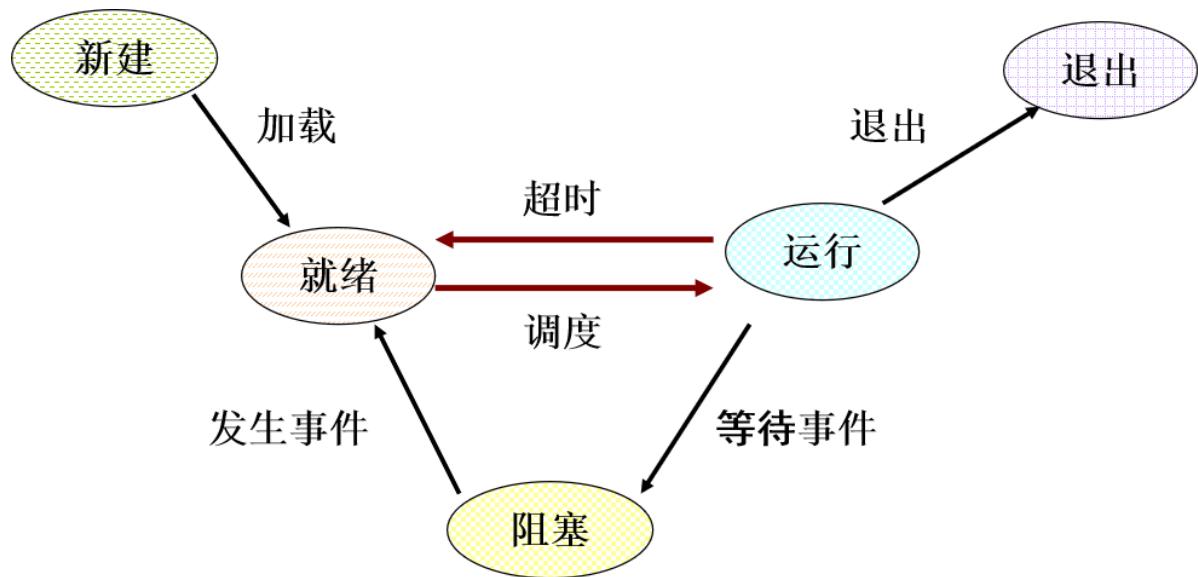
I1利用率: $(110 - 30) \div 110 = 72.7\%$

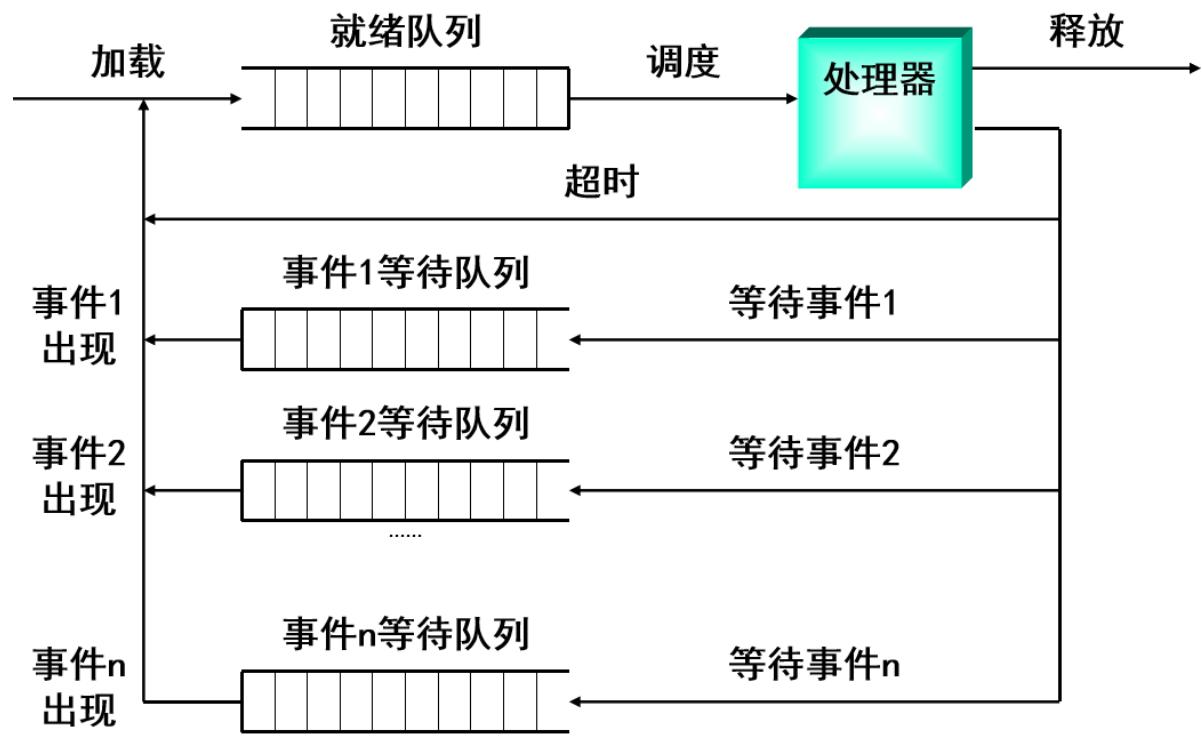
I2利用率: $(110 - 20) \div 110 = 81.8\%$

第3章 进程描述和控制

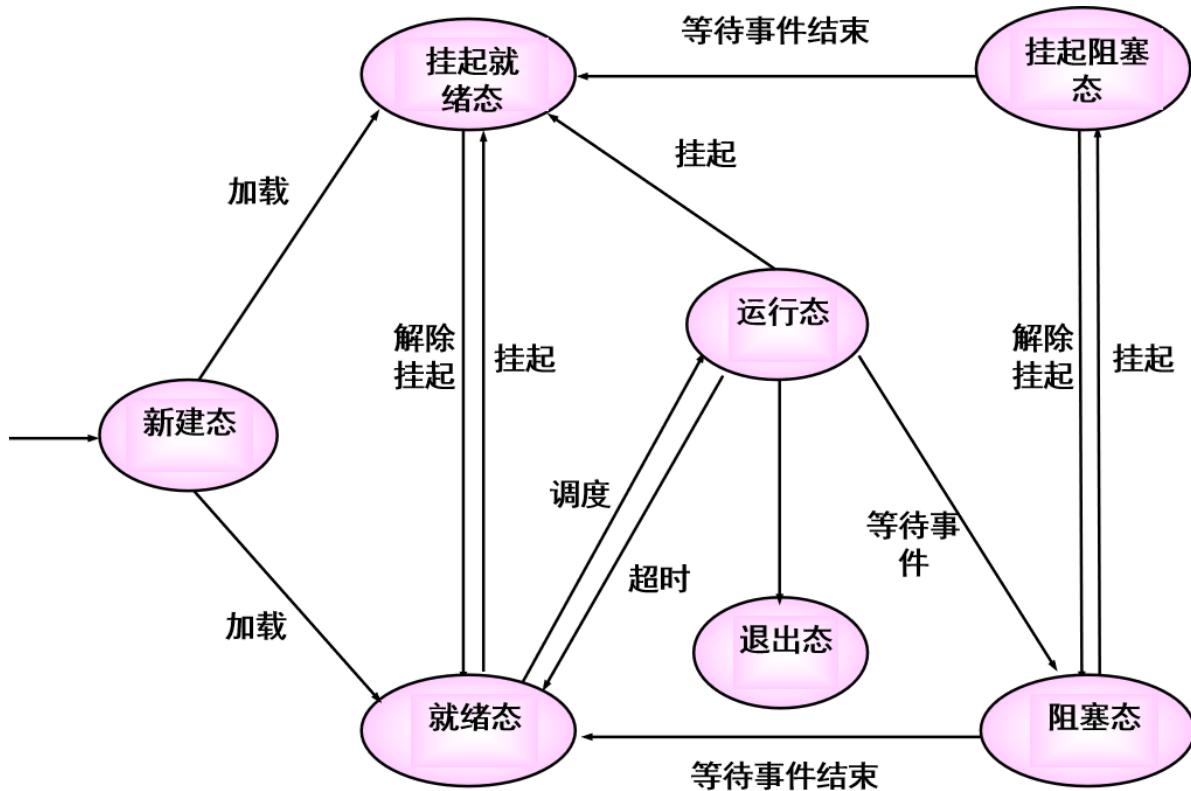
1 五状态进程模型，包含两个挂起态的模型，状态转换

1.1 五状态进程模型





1.2 被挂起的进程



2 进程映像，进程控制块 PCB (进程属性的集合)

2.1 进程映像

项目	说明
用户数据	用户空间中的可修改部分，包括程序数据、用户栈区域和可修改的程序
用户程序	将被执行的程序
系统栈	每个进程有一个或多个系统栈，用于保存参数、过程调用地址和系统调用地址
进程控制块	操作系统控制进程所需要的数据

2.2 进程控制块PCB



2.2.1 进程标识信息

进程ID、父进程ID、用户ID

2.2.2 进程状态信息

用户可见寄存器、控制和状态寄存器、栈指针

2.2.3 进程控制信息

调度和状态信息、数据结构、进程通信、进程特权、存储管理、资源的所有权和使用情况

3 进程的创建与终止

3.1 创建进程的步骤

- 分配进程标识符
- 分配空间
- 初始化进程控制块
- 设置正确的连接
- 创建或扩充其它数据结构

3.2 进程终止的原因

- 正常完成
- 各种错误和故障
- 操作员或操作系统干涉
- 父进程终止
- 父进程请求终止子进程

4 进程切换

进程切换是让**处于运行态**的进程中断运行，让出处理器，让操作系统指定的新进程运行。被中断进程的上下文环境需要保存。

4.1 切换时机

- 中断
 - 时钟中断
 - I/O中断
 - 内存失效
- 陷阱
- 系统调用

4.2 模式切换

定义：

与用户程序相关联的处理器执行模式（用户模式）和与操作系统相关联的处理器模式（内核模式）之间的切换，

进程切换必须在操作系统的内核模式下进行。

当中断发生时，处理器需要做如下工作：

把程序计数器置成中断处理程序的开始地址；
暂时中断正在执行的用户进程，把进程从用户态切换到内核态，去执行操作系统例行程序以获得服务。

保存的进程上下文环境包括：

所有中断处理可能改变的信息；
恢复被中断程序时所需要的信息。

模式切换可以不改变正处于运行态的进程状态，保存和恢复上下文环境开销小；

进程切换涉及进程状态的变化，开销较大。

4.3 完整的进程切换步骤

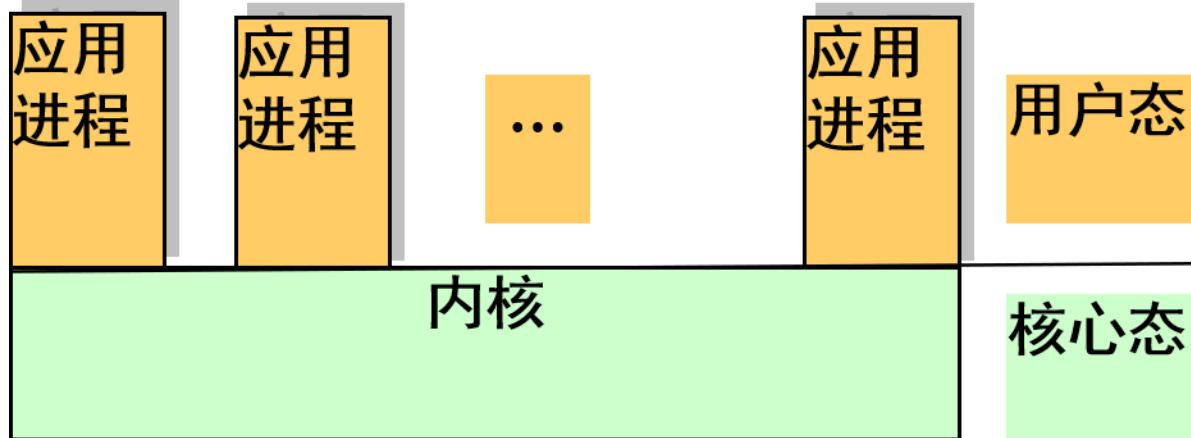
- 保存处理器上下文环境；
- 更新当前处于运行态进程的进程控制块；
- 将进程的进程控制块移到相应的队列；
- 选择另一个进程运行；
- 更新所选择进程的进程控制块；
- 更新内存管理的数据结构；
- 恢复处理器在被选择的进程最近一次切换出运行状态时的上下文环境。

5 执行模式的切换：用户态和系统态

操作系统也是由处理器执行的一个程序，那么，操作系统是一个进程吗？如果是，如何控制？

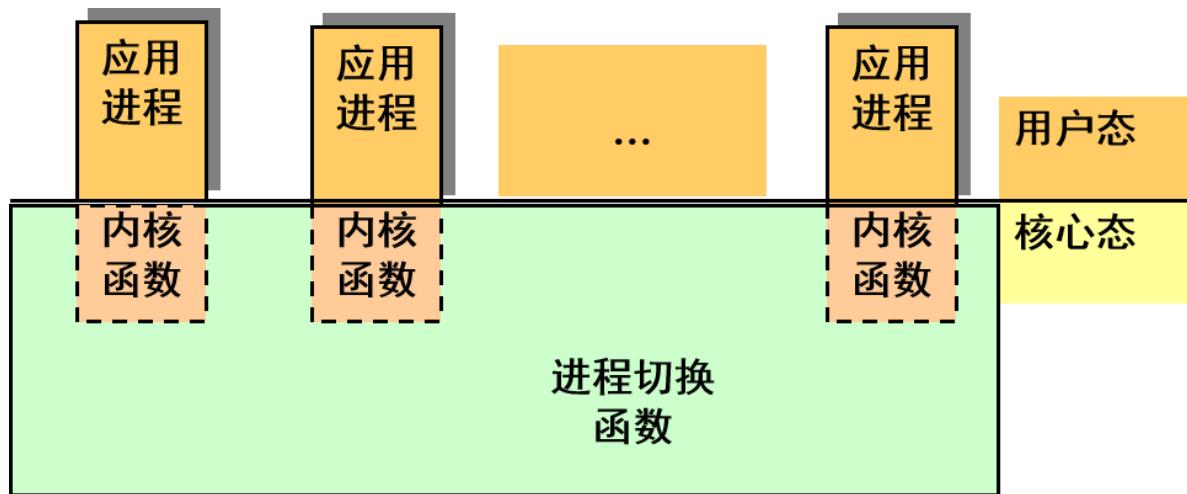
5.1 无进程的内核

进程的概念仅适用于用户程序，操作系统代码作为一个在特权模式下工作的独立实体被执行。



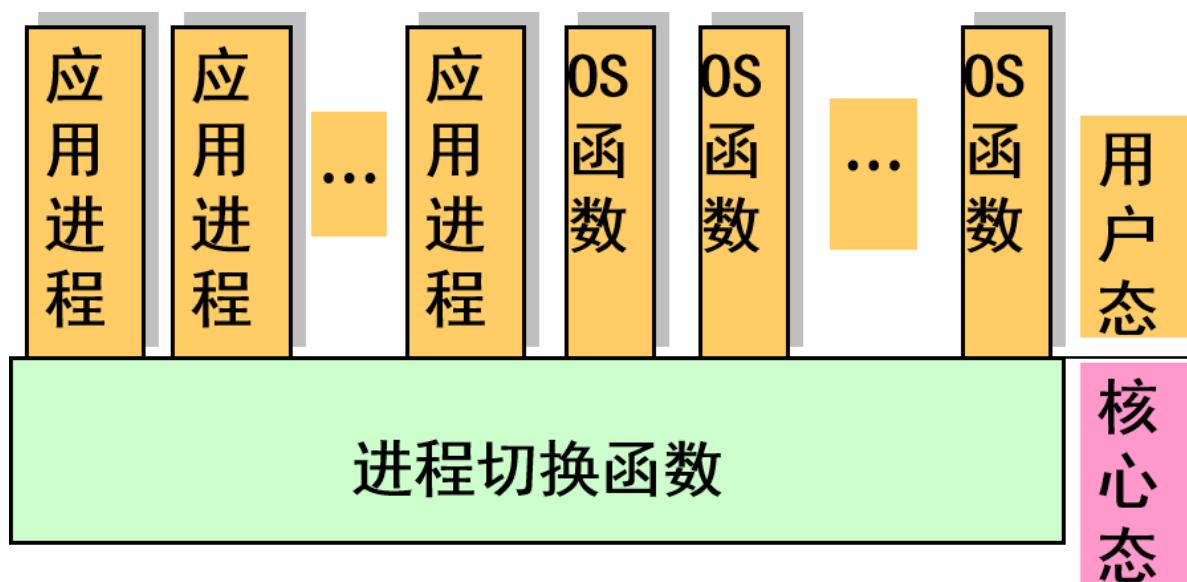
5.2 在用户进程中执行

操作系统是用户调用的一组例程，在用户进程中环境中执行，用于实现各种功能。



5.3 基于进程的操作系统

把操作系统作为一组系统进程来实现，主要的内核函数被组织成独立的进程。



6 UNIX 中，父进程通过系统调用 fork() 创建子进程

fork()有两个返回值：向父进程返回子进程的 PID，向子进程返回 0。

UNIX中，父进程通过系统调用fork()创建子进程，之后可能进行下面三种操作之一：

- 在父进程中继续执行。控制返回用户态下父进程进行fork调用处。
- 处理器控制权交给子进程。子进程开始执行代码，执行点与父进程相同。
- 控制转交给另一个进程。父进程和子进程都置于就绪态。

从fork中返回时，测试返回参数：

- 若值为0，则是子进程，可以转移到相应的用户程序中继续执行；
- 若值不为0（子进程的PID），则是父进程，继续执行主程序。

第3章 进程描述和控制作业

1 对于五状态进程模型，请简单定义每个状态。

- 新建态：刚刚创建的进程。
- 就绪态：进程做好了准备，只要有机会就可执行。
- 运行态：正在执行。
- 阻塞态：进程在某些事件发生前不能执行。
- 退出态：操作系统从可执行进程组中释放出的进程，自身停止或者因为某种原因被取消。

2 操作系统创建一个新进程所执行的步骤是什么？

- 给新进程分配一个唯一的进程标识符
- 给进程分配空间
- 初始化进程控制块
- 设置正确的链接
- 创建或扩充其它数据结构

3 模式切换和进程切换有什么区别？

- 发生模式切换可以不改变正处于运行态的进程状态，这种情况下，保存上下文环境和以后恢复上下文环境只需要很少的开销。
- 发生进程切换时，正在运行的进程被转换到另一个状态（就绪或阻塞等），操作系统必须使其环境产生实质性的变化，比模式切换需要做更多的工作。

4 习题

假设在时间5时，系统资源只有处理器和内存被使用。考虑如下事件：

时间5：P1执行对磁盘单元3的读操作。

时间15：P5的时间片结束。

时间18：P7执行对磁盘单元3的写操作。

时间20：P3执行对磁盘单元2的读操作。

时间22：—————P8就绪；P7写磁盘单元3操作，阻塞；P1读磁盘单元3操作，阻塞；P3读磁盘单元2操作，

时间22：—————阻塞；P5运行

时间24：P5执行对磁盘单元3的写操作。

时间28：P5被换出。——P5挂起阻塞/就绪

时间33：P3读磁盘单元2操作完成，产生中断。——P3读磁盘单元2操作完成之前被阻塞

时间36：P1读磁盘单元3操作完成，产生中断。——P1读磁盘单元3操作完成之前被阻塞

时间37：—————P8运行；P7写磁盘单元3操作，阻塞；P5写磁盘单元3，挂起阻塞；P3就绪；P1就绪

时间38：P8结束。——P8退出

时间40：P5写磁盘单元3操作完成，产生中断。——P5写磁盘单元3操作完成之前被阻塞

时间44：P5被调入。——P5就绪

时间47：—————P5就绪/运行；P7写磁盘单元3操作，阻塞；P8退出；P3就绪/运行；P1就绪/运行

时间48：P7写磁盘单元3操作完成，产生中断。——P7写磁盘单元3操作完成之前被阻塞

请分别写出时间22、37和47时每个进程的状态。如果一个进程在阻塞态，写出其等待的事件。

进程 时间	P1	P3	P5	P7	P8
22	阻塞-读 磁盘单 元3	阻塞-读 磁盘单 元2	运行	阻塞-写 磁盘单 元3	就绪
37	就绪	就绪	挂起阻 塞-写磁 盘单元3	阻塞-写 磁盘单 元3	运行
47	就绪或 运行	就绪或 运行	就绪或 运行	阻塞-写 磁盘单 元3	退出

时间47：P1、P3、P5两个就绪一个运行

第4章 线程

1 进程和线程区别（资源分配单位，调度运行单位）

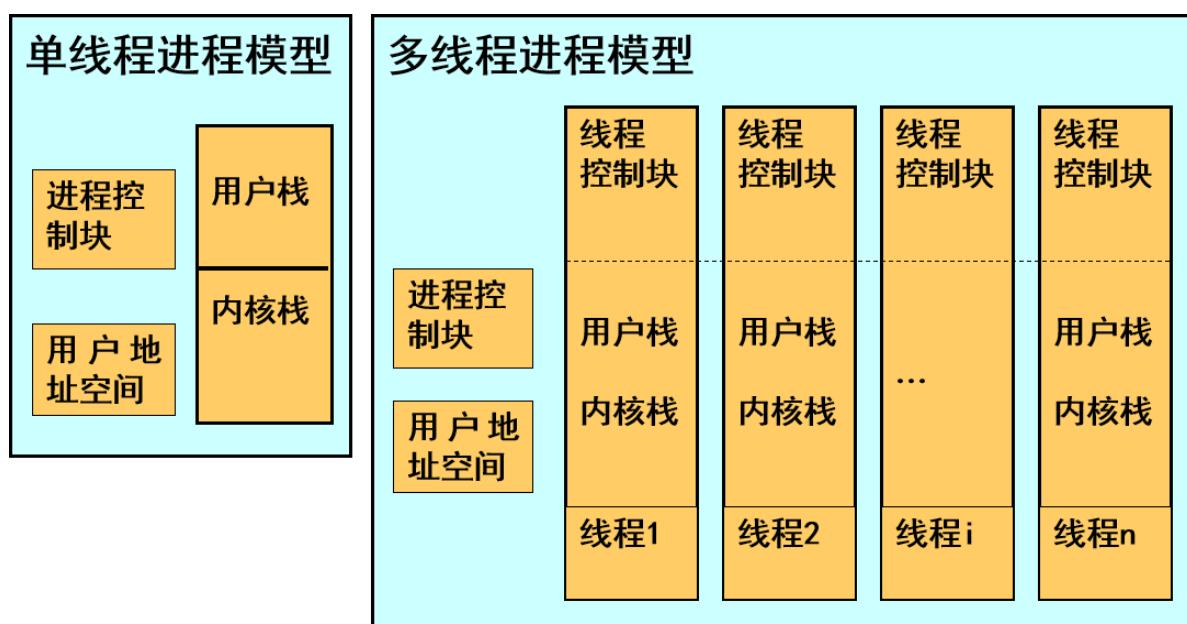
1.1 进程的两个特点

- 资源所有权：进程拥有对资源的控制权或所有权。
- 调度/执行：进程是一个可被操作系统调度和分派的单位。

进程的两个特点是独立的，操作系统可以独立地对其进行处理。

1.2 线程和进程的区别

- 线程（轻量级进程）：分派（调度运行）的单位。
- 进程（任务）：拥有资源所有权的单位。



2 线程的优点，线程的三种状态（运行，就绪，阻塞）

2.1 线程的优点

- 在一个已有进程中创建一个新线程比创建一个全新进程所需的时间少很多。
- 终止一个线程比终止一个进程花费的时间少。
- 同一进程内线程间切换比进程间切换花费的时间少。
- 线程提高了不同的执行程序间通信的效率。

2.2 线程的功能

2.2.1 线程状态

线程的关键状态：

- 就绪
- 运行
- 阻塞

挂起对线程没有意义。

2.2.2 一个线程阻塞是否会导致整个进程阻塞？

- 线程系统调用阻塞时，在多对1用户级线程模型下，会导致所属进程阻塞。
- 在1对1或多对多模型下，不会导致该问题的发生。
- 如果是单进程单线程的话，不管哪个模型，都会阻塞的。

3 用户级线程和内核级线程的特点

3.1 用户级线程

有关线程管理的所有工作都由应用程序完成，内核意识不到线程的存在。

3.1.1 优点

- 线程切换不需要内核态特权；
- 调度可以是应用程序相关的；
- 用户级线程可以在任何操作系统中运行，不需要对底层内核进行修改以支持用户级线程。

3.1.2 缺点

- 当执行一个系统调用时，会阻塞进程中所有线程；
- 无法利用多处理技术。

3.2 内核级线程

有关线程管理的工作由内核完成，应用程序只有一个到内核线程设施的应用程序编程接口。

3.2.1 优点

- 内核可同时把同一进程中的多个线程调度到多个处理器中；
- 若进程中的一个线程被阻塞，内核可以调度同一进程中的另一个线程；
- 内核例程本身也可以使用多线程。

3.2.2 缺点

- 在把控制从一个线程传送到同一个进程内的另一个线程时，需要内核的状态切换。

第四章 线程作业

1 哪些资源通常被一个进程中的所有线程共享？

- 进程控制块
- 用户地址空间

2 列出用户级线程相对于内核线程的三个优点。

- 线程切换不需要内核态特权，节省了两次状态转换的开销
- 调度可以是应用程序相关的
- 用户级线程可以在任何操作系统中运行，不需要对底层内核进行修改以支持用户级线程

3 列出用户级线程相对于内核线程的两个缺点。

- 用户级线程执行一个系统调用时，进程中的所有线程都会被阻塞
- 不能利用多处理技术

第5章：并发：互斥和同步

并发：单处理器多道程序设计系统中，进程交替执行。

并行：多处理器系统中，不仅可以交替执行进程，还可以重叠执行进程。

原子操作：保证指令序列要么作为一个组来执行，要么都不执行。

死锁：两个或两个以上的进程因其中的每个进程都在等待其他进程做完某些事情而不能继续执行。

活锁：两个或两个以上进程为了响应其他进程中的变化而持续改变自己的状态但不做有用的工作。

竞争条件：多个线程或进程在读写一个共享数据时，结果依赖于它们执行的相对时间。

饥饿：一个可运行的进程被调度程序无限期地忽略，不能被调度执行的情形。

1 互斥的概念

当一个进程在临界区访问共享资源时，其他进程不能进入该临界区访问任何共享资源。

2 临界资源与临界区

临界区：一段代码，在这段代码中进程将访问共享资源（临界资源）。

当一个进程已经在这段代码中运行时，另外一个进程就不能在这段代码中执行。

3 信号量含义，semWait、semSignal 含义

信号量是一个与队列有关的整型变量。可以初始化成非负数。

semWait操作使信号量减1。若值为负数，则执行semWait的进程阻塞，否则继续执行；

semSignal操作使信号量加1。若值小于或等于0，则被semWait操作阻塞的进程被解除阻塞。

4 信号量原语定义 (图 5.3)

```
struct semaphore {
    int count;
    queueType queue;
};

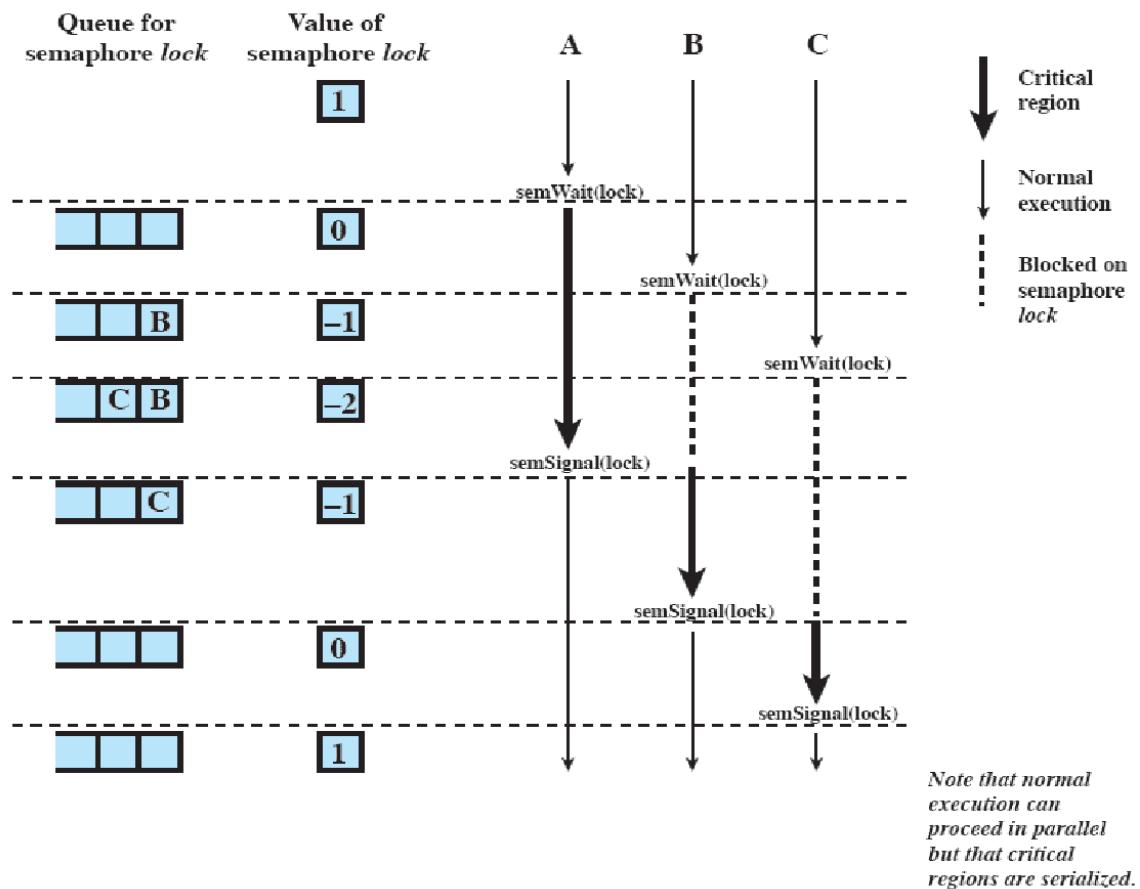
void semWait(semaphore s)
{
    s.count--;
    if (s.count < 0) {      把当前进程插入队列
        /* place this process in s.queue */;
        /* block this process */;
    }                      阻塞当前进程
}

void semSignal(semaphore s)
{
    s.count++;
    if (s.count <= 0) {      把进程P从队列中移除
        /* remove a process P from s.queue */;
        /* place process P on ready list */;
    }                      把进程P插入就绪队列
}
```

5 用信号量实现互斥与同步

```
/* program mutualexclusion */
const int n = /* number of processes */;
semaphore s = 1;                                进程数

void P(int i)
{
    while (true) {
        semWait(s);      临界区
        /* critical section */;
        semSignal(s);
        /* remainder */;
    }                      剩余部分
}
void main()
{
    parbegin (P(1), P(2), . . . , P(n));
}
```



6 有限缓冲的生产者/消费者问题 (图 5.13)

问题描述：

有一个或多个生产者生产某种类型的数据，并放置在缓冲区中；

有一个消费者从缓冲区中取数据，每次取一项；

系统保证避免对缓冲区的重复操作，即任何时候只有一个主体（生产者或消费者）可以访问缓冲区。

缓存已满时，生产者不能继续添加数据；

缓存已空时，消费者不能继续移走数据。

```
semaphore s=1,empty=n,full=0; //empty空位置初始化为n, full产品个数初始化为0
```

```
void producer(){           //生产者
    while (true) {
        produce();
        semwait(empty);    //empty空位置--
        semwait(s);
        append();
        semsignal(s);
        semsignal(full);   //full产品个数++
    }
}
```

```
void consumer(){
    while (true) {
        semwait(full);     //full产品个数--
        semwait(s);
        take();
    }
}
```

```

    semSignal(s);
    semSignal(empty); //empty空位置++
    consume();
}
}

```

6.1 信号量问题的补充练习一

桌子上有一个盘子，可以存放一个水果。父亲总是放苹果到盘子中，而母亲总是放香蕉到盘子中；儿子专等吃盘中的香蕉，而女儿专等吃盘中的苹果。

分析：

生产者 - 消费者问题的一种变形，生产者、消费者以及放入缓冲区的产品都有两类，但每类消费者只消费其中固定的一种产品。

数据结构：semaphore dish, apple, banana；

dish：表示盘子是否为空，初值为1

apple：表示盘中是否有苹果，初值为0

banana：表示盘中是否有香蕉，初值为0

```

process father(){
    semwait(dish);
    将苹果放到盘中;
    semSignal(apple);
}

process mother(){
    semwait(dish);
    将香蕉放到盘中;
    semSignal(banana);
}

process son(){
    semwait(banana);
    从盘中取出香蕉;
    semSignal(dish);
}

process daughter(){
    semwait(apple);
    从盘中取出苹果;
    semSignal(dish);
}

```

6.2 信号量问题的补充练习二

在一个盒子里，混装了数量相等的黑白围棋子。

现在用自动分拣系统把黑子、白子分开，设分拣系统有两个进程P1和P2，其中P1拣白子， P2拣黑子。

规定每个进程每次拣一子，当一个进程在拣时，不允许另一个进程去拣；当一个进程拣了一子时，必须让另一个进程去拣。

试用信号量协调两个进程的并发执行。

```

process P1(){
    while(true){
        semWait(S1);
        捣白子;
        semSignal(S2);
    }
}

process P2(){
    while(true){
        semWait(S2);
        捣黑子;
        semSignal(S1);
    }
}

```

7 进程间通过“消息传递”交换信息：无阻塞 send 和阻塞 receive

消息传递：合作进程之间进行信息交换。

消息传递原语：

```

send (destination, message)
receive (source, message)

```

- 阻塞send，阻塞receive：发送者和接收者都被阻塞，直到完成信息的投递。
- 无阻塞send，阻塞receive：接收者阻塞，直到请求的信息到达。
- 无阻塞send，无阻塞receive：不要求任何一方等待。

第5章：并发：互斥和同步作业

1 写出信号量定义，semWait和semSignal原语，以及用信号量实现互斥的伪代码

```

struct semaphore{
    int count;
    queueType queue;
}

void semWait(semaphore s){
    s.count--;
    while(s.count<0){
        s.queue.set(p);
        //阻塞p
    }
}

void semSignal(semaphore s){
    s++;
    while(s<=0){
        s.queue.remove(p);
        //就绪p
    }
}

```

```

const int n=number of p;
semaphore s=1;
void P(int i){
    while(true){
        semWait(s);
        //临界区
        semSignal(s);
        //剩余区域
    }
}
void main{
    parbegin(P(1),P(2),...,P(n));
}

```

```

struct semaphore {
    int count;
    queueType queue;
};
void semWait(semaphore s)
{
    s.count--;
    if (s.count < 0) {
        /* place this process in s.queue */;
        /* block this process */;
    }
}
void semSignal(semaphore s)
{
    s.count++;
    if (s.count <= 0) {
        /* remove a process P from s.queue */;
        /* place process P on ready list */;
    }
}

```

Figure 5.3 A Definition of Semaphore Primitives

```

/* program mutual exclusion */
const int n = /* number of processes */;
semaphore s = 1;
void P(int i)
{
    while (true) {
        semWait(s);
        /* critical section */;
        semSignal(s);
        /* remainder */;
    }
}
void main()
{
    parbegin (P(1), P(2), . . . , P(n));
}

```

Figure 5.6 Mutual Exclusion Using Semaphores

习题1

假设一个阅览室有100个座位，没有座位时读者在阅览室外等待；

每个读者进入阅览室时都必须在阅览室门口的一个登记本上登记座位号和姓名，然后阅览，离开阅览室时要去掉登记项。

每次只允许一个人登记或去掉登记。用信号量操作描述读者的行为。

```
int s=1,empty=100;
void reader(){
    semWait(empty);

    semWait(s);
    //在阅览室门口的一个登记本上登记座位号和姓名
    semSignal(s);

    //阅读

    semWait(s);
    //离开阅览室时要去掉登记项
    semSignal(s);

    semSignal(empty);
}
```

分析：实际上是一个非常简单的同步-互斥问题，不要想复杂了

数据结构：

```
struct{char name[10];
       int number;
     }A[100];

semaphore mutex, seatcount;

mutex: 用来互斥，初值为1
seatcount: 对空座位计数，初值为100

for(int i=0;i<100;i++)
{A[i].number=i; A[i].name=null;}
```

```

process reader i (char readernam e[])
{
    semWait (seatcount) ;
    semWait (mutex) ;
    for (int i=0; i<100; i++)
        if (A[i].name==null) A[i].name=readernam e; //跳出
    reader get the seat number=i ;
    semSignal (mutex) ;
    进入阅览室，在座位号i处坐下读书;
    semWait (mutex) ;
    A[i].name=null ;
    semSignal (mutex) ;
    semSignal (seatcount) ;
    离开阅览室; }

```

习题2

设公共汽车上，司机和售票员活动如下：

- 1) 司机：启动汽车，正常行车，到站停车；
- 2) 售票员：关车门，售票，开门上下客。

用信号量操作描述司机和售票员的同步。

```

int s1=0,s2=0; //s1表示是否允许司机启动汽车，其初值为0；s2表示是否允许售票员开门，其初值为0

void driver(){
    semWait(s1); //s1--,s1=-1,driver进入阻塞队列，不允许启动汽车
    //启动汽车
    //正常行车
    //到站停车
    semSignal(s2); //s2++,s2=1,saler进入就绪队列，不允许售票
}

void saler(){
    //关车门
    semSignal(s1); //s1++.s1=0, driver进入就绪队列，启动汽车
    //售票
    semWait(s2); //s2--,s2=0,saler进入阻塞队列，不允许售票
    //开门上下客
}

```

数据结构：

semaphore s1=0; 表示是否允许司机启动汽车
semaphore s2=0; 表示是否允许售票员开门

<pre> driver: while(true) { semWait(s1); 启动车辆; 正常行车; 到站停车; semSignal(s2); } </pre>	<pre> busman: while(true) { 关车门; semSignal(s1); 售票; semWait(s2); 开车门; 上下乘客; } </pre>
---	---

习题3

独木桥问题：东、西向汽车过独木桥。

桥上无车时允许一方汽车过桥，待全部过完后才允许另一方汽车过桥。

用信号量操作写出同步算法。(提示：参考读者优先的解法)

数据结构：

```

semaphore wait, mutex1, mutex2;
mutex1=mutex2=wait=1;
int count1, count2;
count1=count2=0;

```

```

process P 东() {
    semWait(mutex1);
    count1++;
    if(count1==1)    semWait(wait);
    semSignal(mutex1);
    过独木桥;
    semWait(mutex1);
    count1--;
    if(count1==0)    semSignal(wait);
    semSignal(mutex1);
}

```

```

process P 西() {
    semWait(mutex2);
    count2++;
    if(count2==1)    semWait(wait);
    semSignal(mutex2);
    过独木桥;
    semWait(mutex2);
    count2--;
    if(count2==0)    semSignal(wait);
    semSignal(mutex2);
}

```

第6章：并发：死锁和饥饿

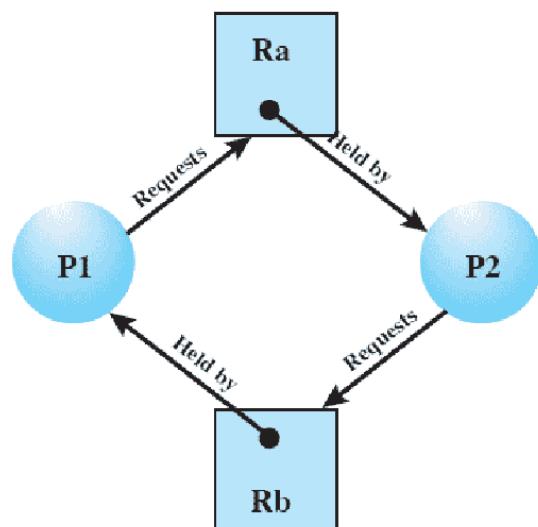
1 死锁原因：竞争资源、进程推进顺序不当

死锁：一组相互竞争系统资源或进行通信的进程间的永久阻塞。

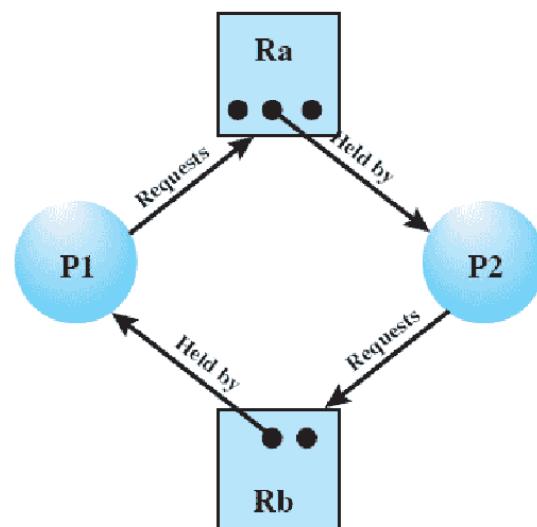
死锁问题没有一种有效的通用解决方案。

所有死锁都涉及两个或多个进程之间对资源需求的冲突。

2 资源分配图 (若死锁，则资源分配图中必有环路，但有环路时不一定死锁)



(c) Circular wait



(d) No deadlock

3 死锁的四个必要条件

- 互斥：一次只有一个进程可以使用一个资源。
- 占有且等待：当一个进程等待其他进程时，继续占有已经分配的资源。
- 不可抢占：不能强行抢占进程已占有的资源。
- 循环等待：存在一个封闭的进程链，使得每个进程至少占有此链中下一个进程所需要的一个资源。

4 三种处理方法：预防，避免，检测和恢复

4.1 死锁预防

间接的死锁预防方法：预防前三个条件

直接的死锁预防方法：预防第四个条件

4.1.1 互斥

该条件不可能被禁止

4.1.2 占有且等待

可要求进程一次性地请求所有需要的资源，并且阻塞进程直到所有请求都同时满足。

存在的问题：

- 一个进程可能被阻塞很长时间，已等待满足其所有的资源请求。
- 分配给一个进程的资源可能有相当长的一段时间不会被使用。
- 一个进程可能事先并不知道它所需要的全部资源。

4.1.3 不可抢占

- 如果占有某些资源的进程进一步申请资源时被拒绝，则该进程必须释放它最初占有的资源。
- 如果进程A请求当前被进程B占有的一個资源，则操作系统可以抢占进程B，要求它释放资源。

4.1.4 循环等待

定义资源类型的线性顺序。

如果一个进程已经分配到了R类型的资源，那么它接下来请求的资源只能是那些排在R类型之后的资源类型。

存在的问题：

- 会导致进程执行速度变慢；
- 可能在没有必要的情况下拒绝资源访问。

4.2 死锁避免

- 如果一个进程的请求会导致死锁，则不启动此进程
- 如果一个进程增加的资源请求会导致死锁，则不允许此分配

4.2.1 优点

- 不需要死锁预防中的抢占和回滚进程
- 比死锁预防的限制少

4.2.2 缺点

- 必须事先声明每个进程请求的最大资源。
- 进程必须是无关的，其执行的顺序必须没有任何同步要求的限制。
- 分配的资源数目必须是固定的。
- 在占有资源时，进程不能退出

4.3 死锁检测

- 死锁检测策略不限制资源访问或约束进程行为。
- 系统周期性地执行检测算法，检测循环等待条件是否成立。

4.3.1 检测时机

- 每个资源请求发生时
- 隔一段时间
- 每次资源请求时检测死锁

优点：可以尽早地检测死锁情况

缺点：频繁的检查会耗费相当多的处理器时间

4.3.2 恢复

- 取消所有死锁进程。
- 把每个死锁进程回滚到某些检查点，并重新启动所有进程。
- 连续取消死锁进程直到不再存在死锁。选择取消进程的顺序基于某种最小代价原则。每次取消后，必须重新调用检测算法，以测试是否仍存在死锁。
- 连续抢占资源直到不再存在死锁。同3。

5 银行家算法：要求能够判断现在是否安全，某进程请求资源是否能够满足

	A	B	C
P1	3	2	2
P2	6	1	3
P3	3	1	4
p4	4	2	2

Claim
(贷款方总需要的钱)

	A	B	C
P1	1	0	0
P2	6	1	2
P3	2	1	1
p4	0	0	2

Allocation
(银行已经分配给贷款方的钱)

	A	B	C
P1	2	2	2
P2	0	0	1
P3	1	0	3
p4	4	2	0

Need
(贷款方还需要的钱)

9	3	6
0	1	1

Resource
(银行可分配给贷款方的钱)

0	1	1

Available
(借出后还剩下的钱)

借款顺序

5.1 银行家算法举例1

说明：

5个进程P0...P4，
3种资源类型A、B、C，且实例个数分别为10、5、7
T0时刻状态如图所示

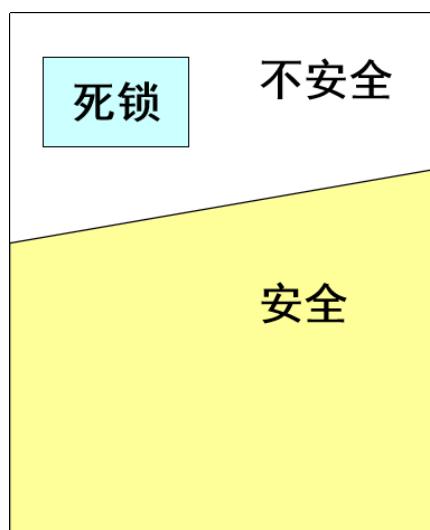
	<u>Allocation</u>			<u>Claim</u>			<u>Available</u>		
	A	B	C	A	B	C	A	B	C
P ₀	0	1	0	7	5	3	3	3	2
P ₁	2	0	0	3	2	2			
P ₂	3	0	2	9	0	2			
P ₃	2	1	1	2	2	2			
P ₄	0	0	2	4	3	3			

need:

	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

	Work			Need			Allocation			Work+allocation			finish
	A	B	C	A	B	C	A	B	C	A	B	C	
P1	3	3	2	1	2	2	2	0	0	5	3	2	True
P3	5	3	2	0	1	1	2	1	1	7	4	3	True
P4	7	4	3	4	3	1	0	0	2	7	4	5	True
P2	7	4	5	6	0	0	3	0	2	10	4	7	True
P0	10	4	7	7	4	3	0	1	0	10	5	7	True

T0时刻是安全状态，存在安全序列<P1, P3, P4, P2, P0>



结论：

- 安全状态不是死锁状态
- 死锁状态是不安全状态
- 不是所有不安全状态都是死锁状态

5.2 银行家算法举例2

说明：

5个进程P1...P5，
3种资源类型A、B、C，且实例个数分别为17、5、20
T0时刻状态如图所示

	<u>Allocation</u>			<u>Claim</u>			<u>Available</u>		
	A	B	C	A	B	C	A	B	C
P ₁	2	1	2	5	5	9	2	3	3
P ₂	4	0	2	5	3	6			
P ₃	4	0	5	4	0	11			
P ₄	2	0	4	4	2	5			
P ₅	3	1	4	4	2	4			

	Work			Need			Allocation	Work+allocation			finish		
	A	B	C	A	B	C	A	B	C				
P ₅	2	3	3	1	1	0	3	1	4	5	4	7	True
P ₁	5	4	7	3	4	7	2	1	2	7	5	9	True
P ₂	7	5	9	1	3	4	4	0	2	11	5	11	True
P ₃	11	5	11	0	0	6	4	0	5	15	5	16	True
P ₄	15	5	16	2	2	1	2	0	4	17	5	20	True

T0时刻是安全的，存在安全序列<P5, P1, P2, P3, P4>

6 用信号量解决不死锁的哲学家就餐问题

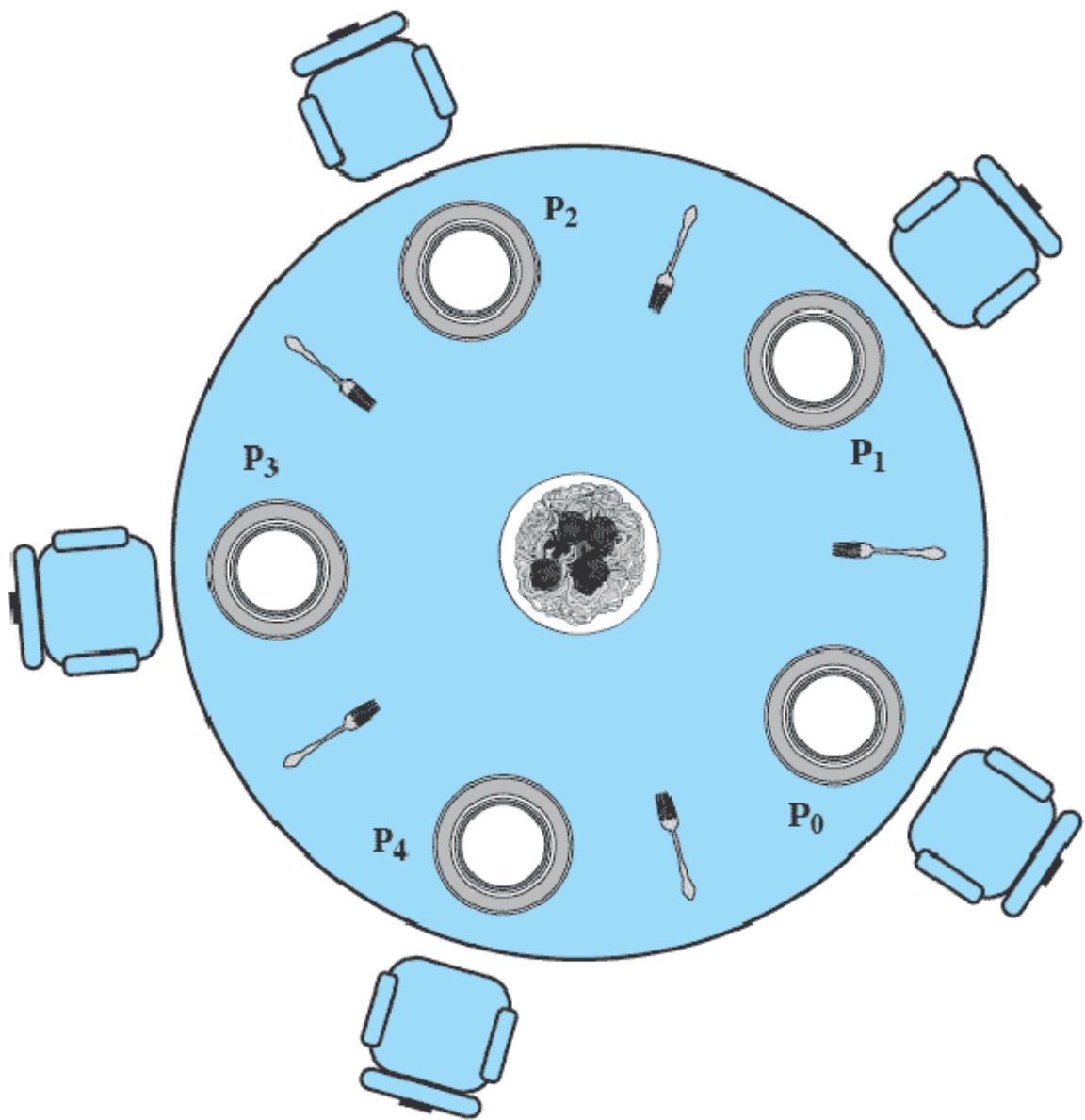
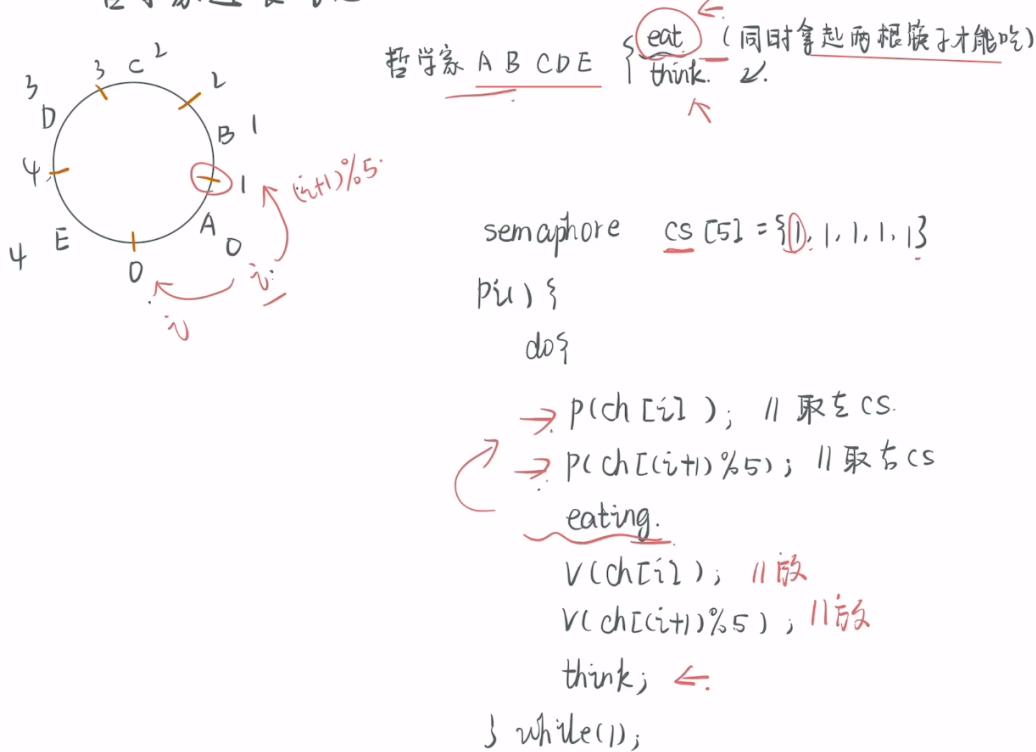


Figure 6.11 Dining Arrangement for Philosophers

```
/* program      diningphilosophers */
semaphore fork [5] = {1};
int i;
void philosopher (int i)
{
    while (true) {
        think();
        wait (fork[i]);
        wait (fork [(i+1) mod 5]);
        eat();
        signal(fork [(i+1) mod 5]);
        signal(fork[i]);
    }
}
```

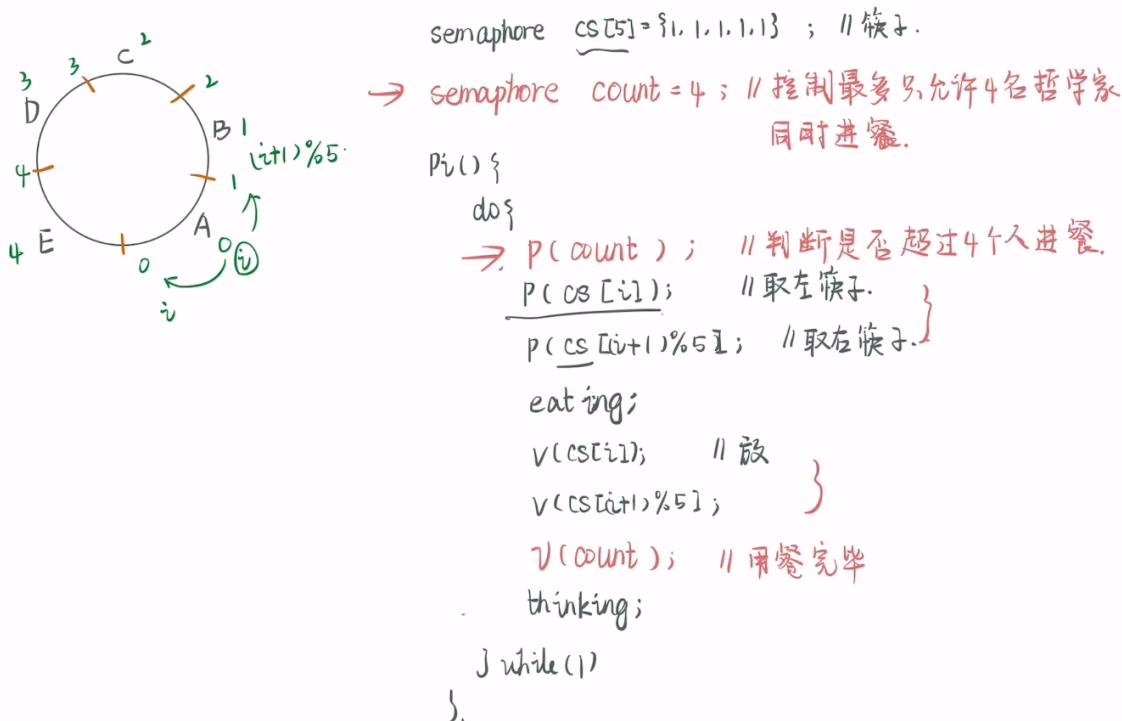
```
/* program  diningphilosophers */
semaphore fork[5] = {1};
semaphore room = {4};
int i;
void philosopher (int i)
{
    while (true) {
        think();
        wait (room);
        wait (fork[i]);
        wait (fork [(i+1) mod 5]);
        eat();
        signal (fork [(i+1) mod 5]);
        signal (fork[i]);
        signal (room);
    }
}
```

哲学家进餐问题



如果五名哲学家同时落座同时拿起左叉子，就会导致死锁

为了避免死锁，只允许最多四名哲学家同时进餐



第6章：并发：死锁和饥饿作业

1 产生死锁的四个条件是什么？

互斥、占有且等待、不可抢占、循环等待

2 死锁避免、检测和预防之间的区别是什么？

死锁避免通过限制进程启动或资源分配，预防通过运用某种策略来消除产生死锁的四个条件之一，来保证不让死锁状态出现；

而检测允许死锁出现，定期检测死锁的存在并从死锁中恢复出来。

3 习题1

说明：

6个进程:P0...P5

4种资源:A (15) 、 B (6) 、 C (9) 、 D (10)

T0时刻状态如图所示

	Allocation				Claim				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P ₀	2	0	2	1	9	5	5	5	6	3	5	4
P ₁	0	1	1	1	2	2	3	3				
P ₂	4	1	0	2	7	5	4	4				
P ₃	1	0	0	1	3	3	3	2				
P ₄	1	1	0	0	5	2	2	1				
P ₅	1	0	1	1	4	4	4	4				

问题：

1.验证可用资源向量的正确性。

2.计算需求矩阵。

3.指出一个安全的进程序列来证明当前状态的安全性。同时指出每个进程结束时可用资源向量的变化情况。

	work				Allocation				W+A				finish
	A	B	C	D	A	B	C	D	A	B	C	D	
P ₁	6	3	5	4	0	1	1	1	6	4	6	5	T
P ₂	6	4	6	5	4	1	0	2	10	5	6	7	T
P ₃	10	5	6	7	1	0	0	1	11	5	6	8	T
P ₄	11	5	6	8	1	1	0	0	12	6	6	8	T
P ₅	12	6	6	8	1	0	1	1	13	6	7	9	T
P ₀	13	6	7	9	2	0	2	1	15	6	9	10	T

系统处于安全状态，存在安全序列<P1,P2,P3,P4,P5,P0>

4.假设P5请求资源 (3, 2, 3,3) , 该请求应该被允许吗? 请说明理由。

	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	2	0	2	1	7	5	3	4	3	1	2	1
P1	0	1	1	1	2	1	2	2				
P2	4	1	0	2	3	4	4	2				
P3	1	0	0	1	2	3	3	1				
P4	1	1	0	0	4	1	2	1				
P5	4	2	4	4	0	2	0	0				

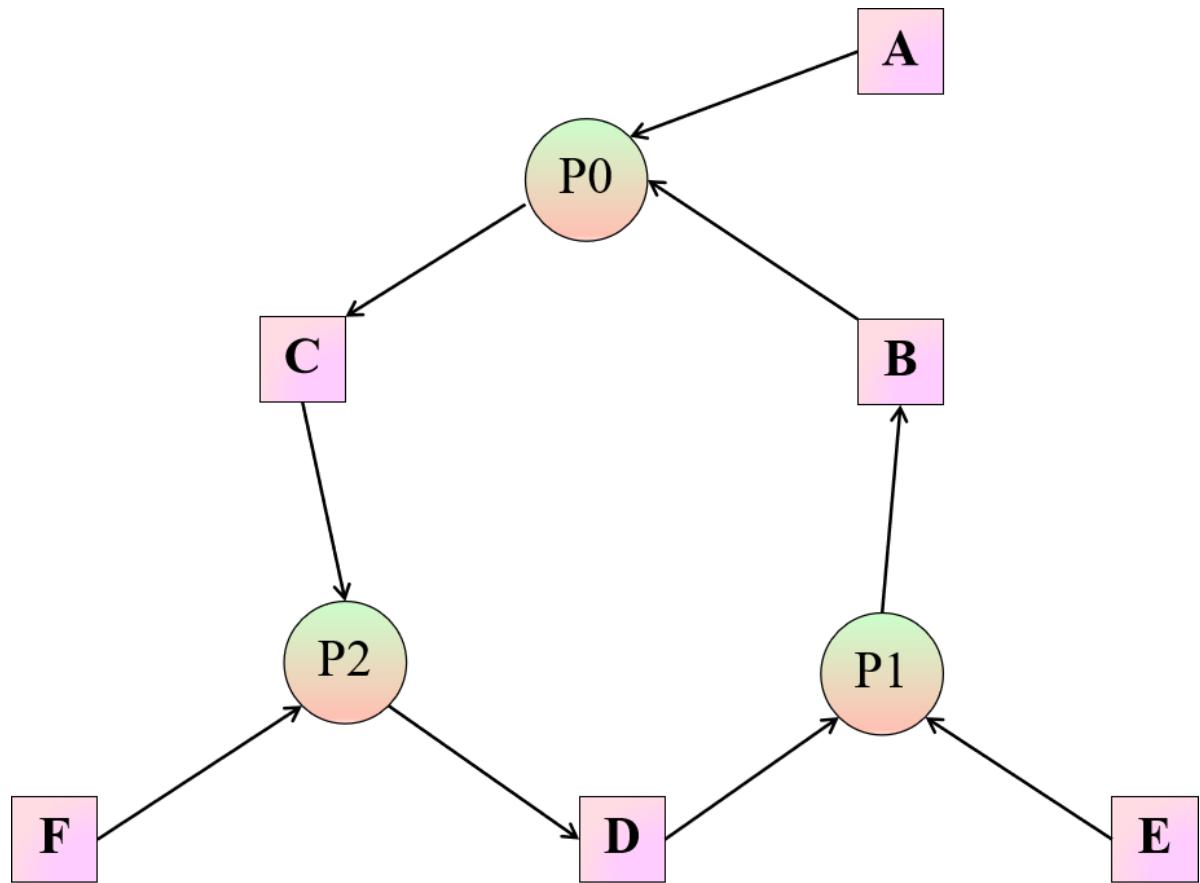
执行安全性算法， Available(3, 1, 2,1)不能满足任何进程，进入不安全状态，恢复旧数据结构， P5等待。

4 习题2

如下的代码涉及3个进程竞争6种资源 (A~F) 。

- a. 使用资源分配图来指出这种实现中可能存在的死锁。
- b. 改变某些请求的顺序来预防死锁。注意不能跨函数移动请求，只能在函数内部调整请求的顺序。使用资源分配图来证明你的答案。

<pre>Void P0() { while(true) { get(A); get(B); get(C); //critical region; //use A, B, C release(A); release(B); release(C); } }</pre>	<pre>Void P1() { while(true) { get(D); get(E); get(B); //critical region; //use D, E, B release(D); release(E); release(B); } }</pre>	<pre>Void P2() { while(true) { get(C); get(F); get(D); //critical region; //use C, F, D release(C); release(F); release(D); } }</pre>
---	---	---



```

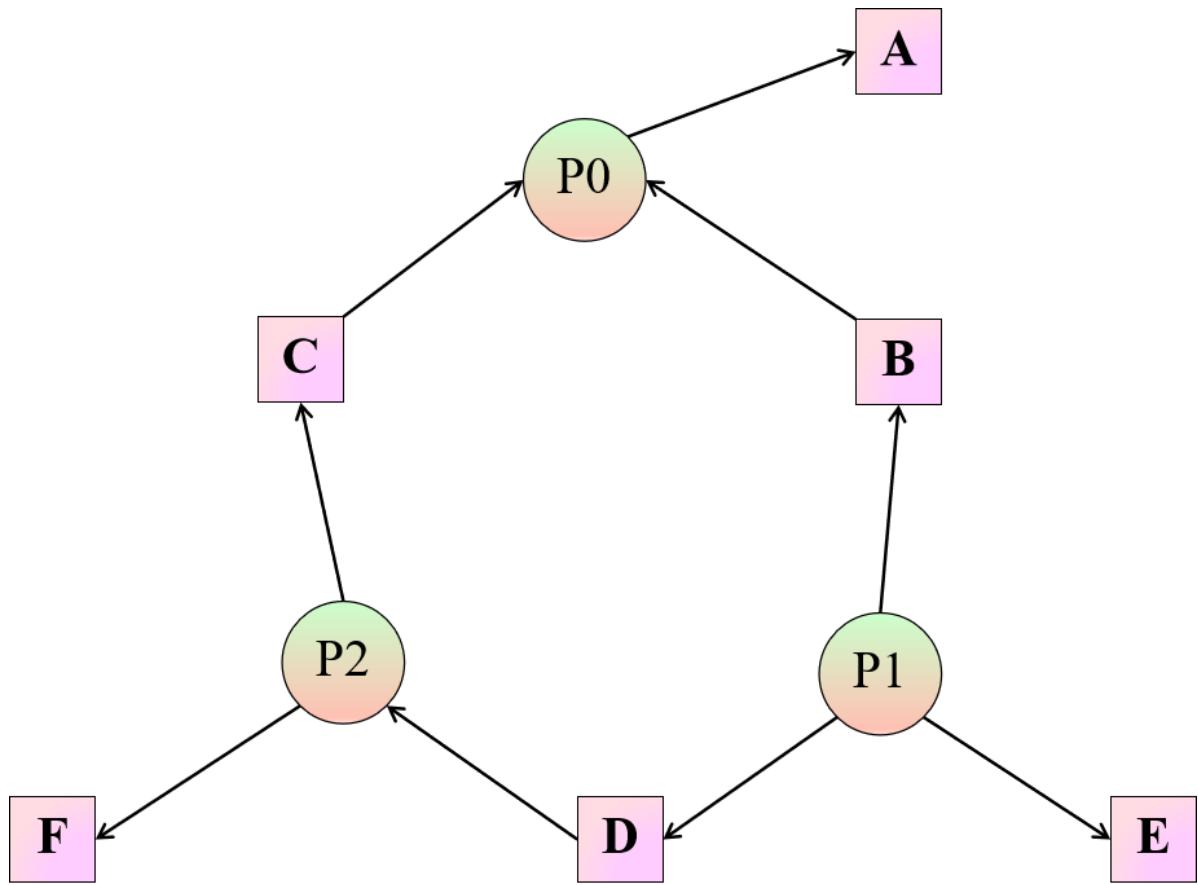
Void P0()
{
    while(true) {
        get(B);
        get(C);
        get(A);
        //critical region;
        //use A, B, C
        release(B);
        release(C);
        release(A);
    }
}
  
```

```

Void P1()
{
    while(true) {
        get(B);
        get(D);
        get(E);
        //critical region;
        //use D, E, B
        release(B);
        release(D);
        release(E);
    }
}
  
```

```

Void P2()
{
    while(true) {
        get(D);
        get(C);
        get(F);
        //critical region;
        //use C, F, D
        release(D);
        release(C);
        release(F);
    }
}
  
```



5 习题 3

考虑一个有4个进程和1种资源的系统。当前的资源请求和分配矩阵如下：

$$C = (3 \ 2 \ 9 \ 7), A = (1 \ 1 \ 3 \ 2)$$

最少需要多少单位的资源才能保证当前的状态是安全的？

当前状态

	Allocation	Need	Available
P0	1	2	?
P1	1	1	
P2	3	6	
P3	2	5	

根据当前状态推算Available的值，进一步推算资源总量

	work	Allocation	W+A	finish
P1	x	1	x+1	T
P0	x+1	1	x+2	T
P3	x+2	2	x+4	T
P2	x+4	3	x+7	T

$$\begin{aligned}
 x &\geq 1 \\
 x+1 &\geq 2 \\
 x+2 &\geq 5 \\
 x+4 &\geq 6 \\
 \text{可推算出 } x &\geq 3
 \end{aligned}$$

由此可知Available的值最小为3，则资源总量最小为10

第7章：内存管理

1 固定分区，动态分区分配策略 - 首次适配、下次适配、最佳适配

1.1 固定分区——分区大小

1.1.1 大小相等的分区

程序可能太大而不能放到一个分区中

内存的利用率非常低，会有内部碎片

1.1.2 大小不等的分区

可缓解上述问题，但不能完全解决

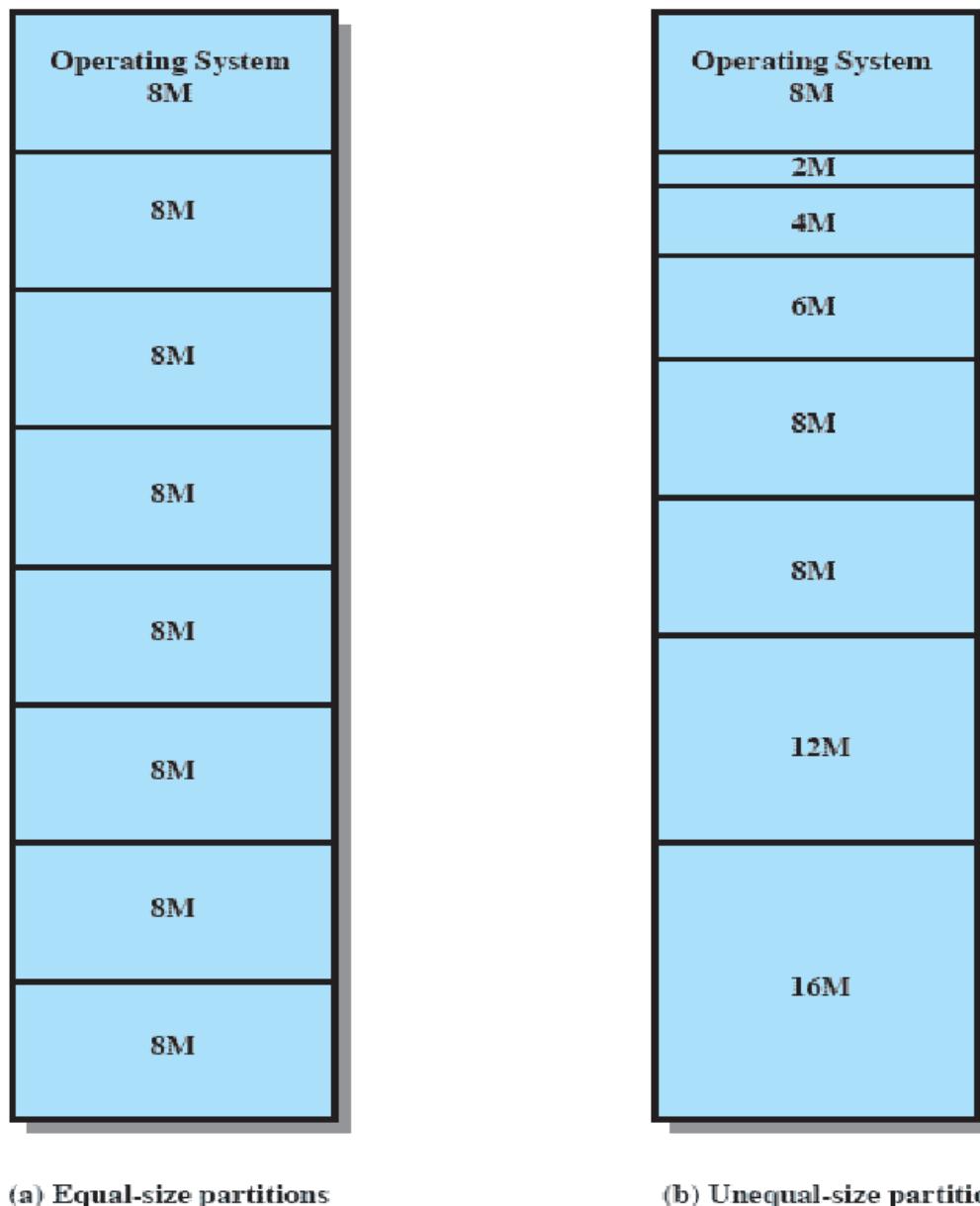


Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory

1.2 固定分区——放置算法

1.2.1 大小相等的分区

- 只要存在可用的分区，进程就可以装入分区。
- 若所有分区都被处于不可运行状态的进程所占据，则选择其中一个进程换出，为新进程让出空间。

1.2.2 大小不等的分区

- 把每个进程分配到能够容纳它的最小分区。
- 每个分区维护一个调度队列，用于保存从这个分区换出的进程。
- 为所有进程只提供一个队列。

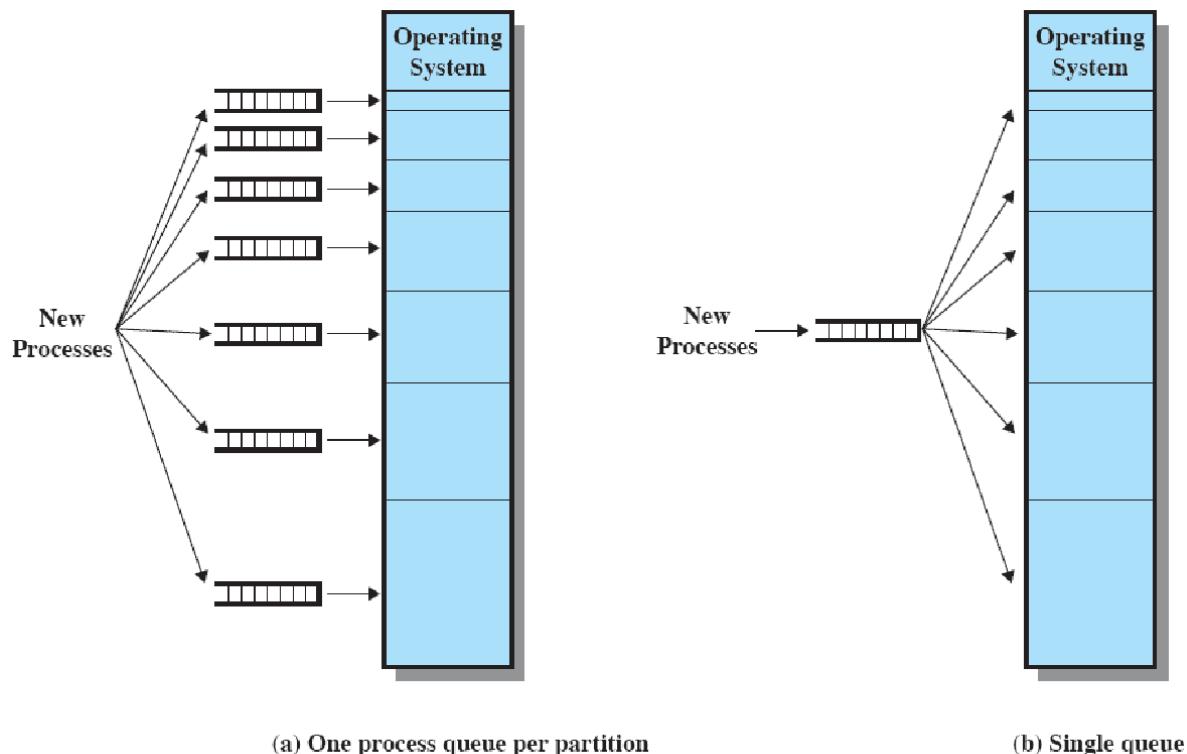


Figure 7.3 Memory Assignment for Fixed Partitioning

1.3 固定分区方案的缺陷

- 分区的数目在系统生成阶段已经确定，限制了系统中活动进程的数目。
- 分区大小在系统生成阶段事先设置，小作业不能有效地利用分区空间。

1.4 动态分区

分区长度和数目是可变的，当进程被装入内存时，系统会给它分配一块和它所需容量完全相等的内存空间。

1.4.1 缺陷

- 外部碎片

1.4.2 外部碎片解决方法

- 压缩：费时且浪费处理器时间

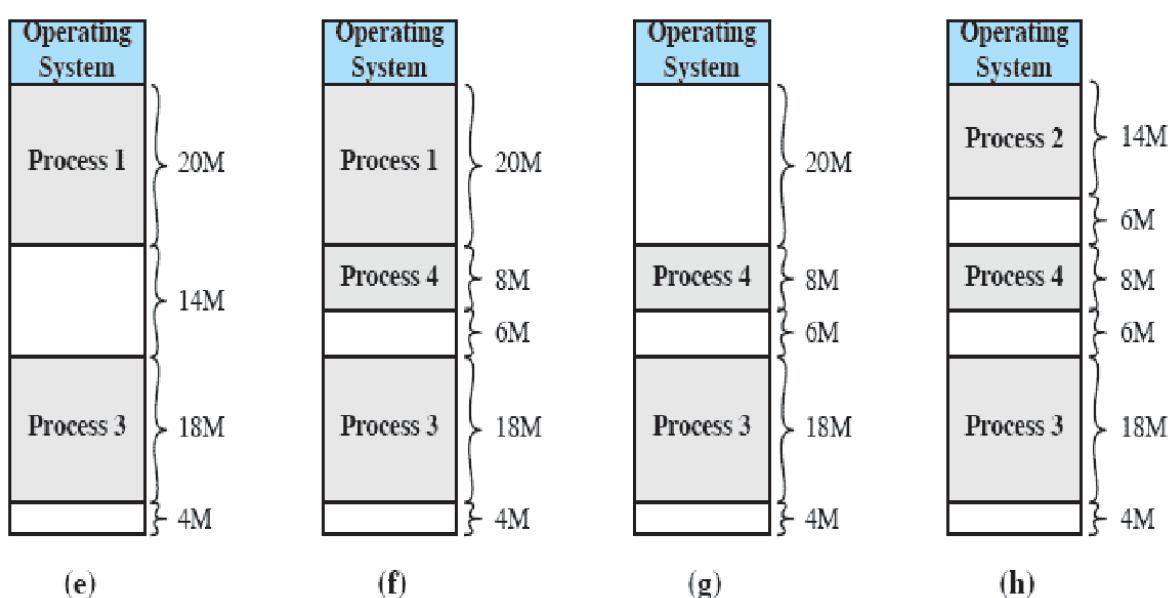
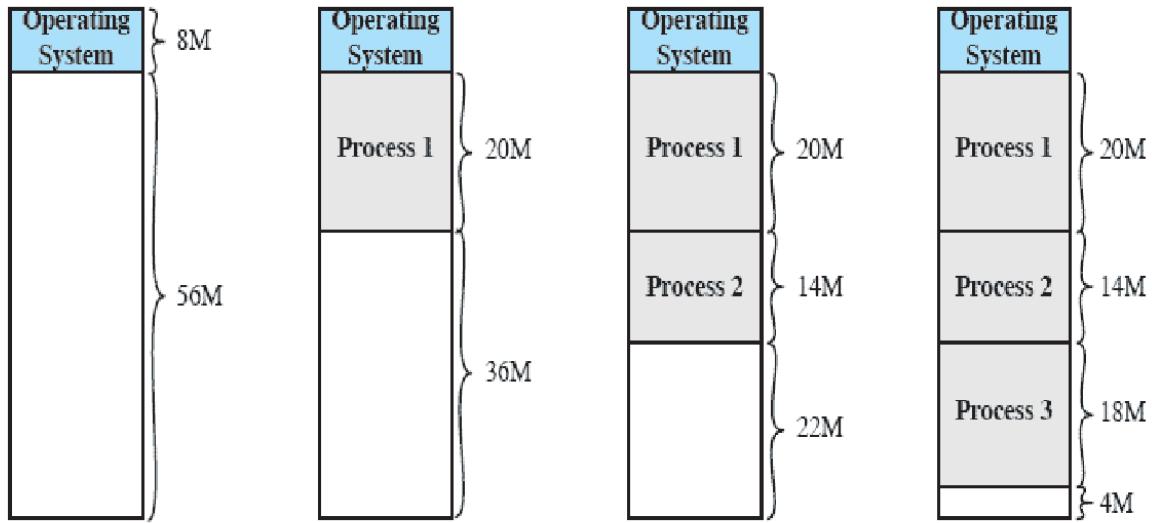


Figure 7.4 The Effect of Dynamic Partitioning

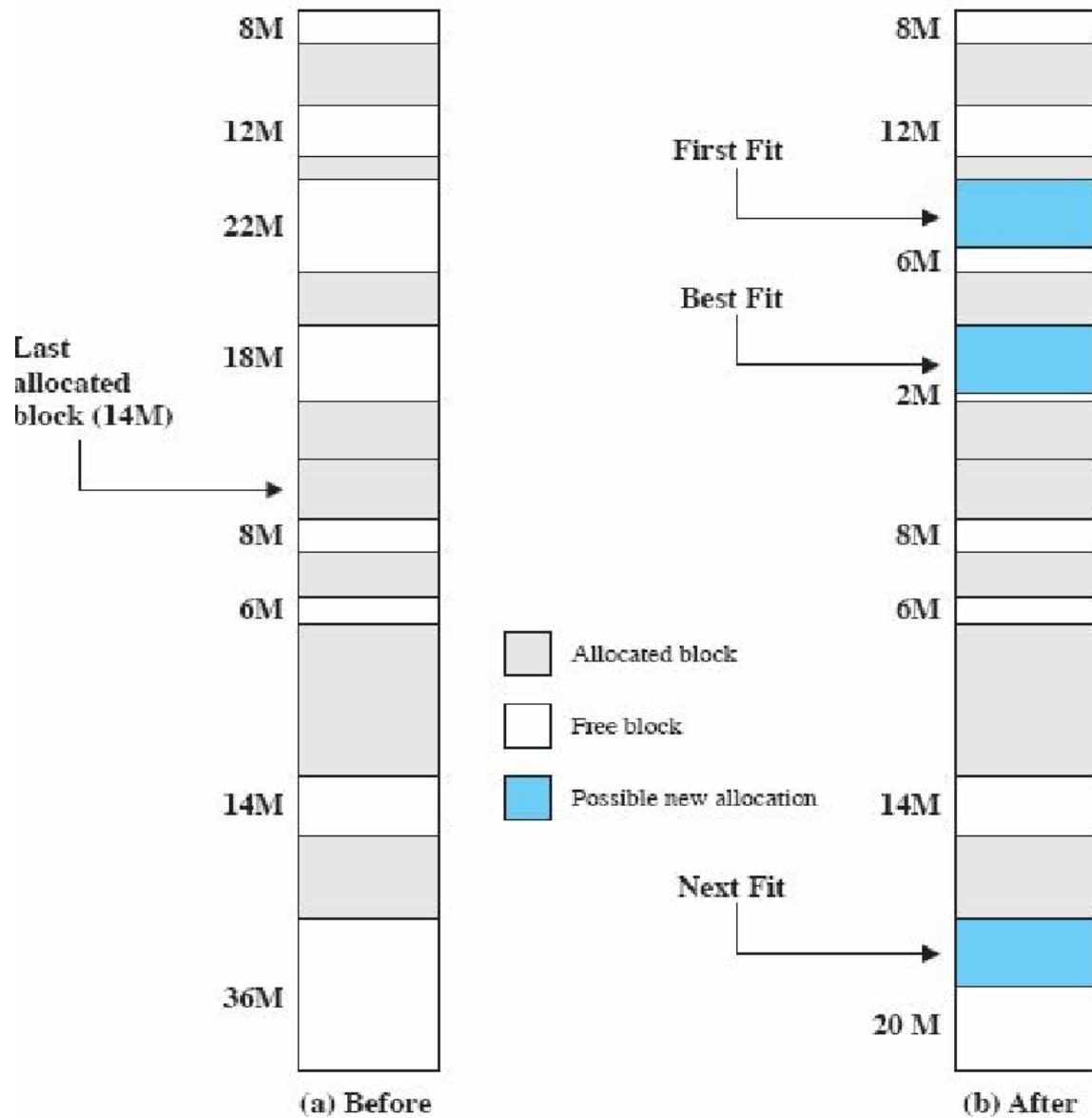
1.5 动态分区——放置算法

首次适配 (First Fit) : 从开始扫描内存, 选择大小足够的第一个可用块;

下次适配 (Next Fit) : 首次适配的变种, 每次分配时从未分配区的上次扫描结束处顺序查找, 选择下一个大小足够的可用块。

最佳适配 (Best Fit) : 选择与要求的大小最接近的块。

最差适配 (Worst Fit) : 选择符合要求大小的最大容量的块。



2 内部碎片，外部碎片

内部碎片：由于装入的数据块小于分区大小，因而导致分区内部存在空间浪费，这种现象称为**内部碎片**。

外部碎片：动态分区方法最终在内存中形成了许多小空洞。随着时间推移，内存中形成了越来越多的碎片，内存的利用率随之下降，这种现象称为**外部碎片**。

3 伙伴系统的分配与回收

1 Mbyte block	1 M					
Request 100 K	A = 128K	128K	256K		512K	
Request 240 K	A = 128K	128K	B = 256K		512K	
Request 64 K	A = 128K	C = 64K	64K	B = 256K		512K
Request 256 K	A = 128K	C = 64K	64K	B = 256K	D = 256K	256K
Release B	A = 128K	C = 64K	64K	256K	D = 256K	256K
Release A	128K	C = 64K	64K	256K	D = 256K	256K
Request 75 K	E = 128K	C = 64K	64K	256K	D = 256K	256K
Release C	E = 128K	128K		256K	D = 256K	256K
Release E			512K		D = 256K	256K
Release D				1M		

4 重定位：将逻辑地址转换为物理地址

逻辑地址（虚地址）：CPU所生成的地址

物理地址（实地址）：内存单元所看到的地址

重定位（地址转换）：把逻辑地址转换为物理地址

静态重定位

地址转换工作在进程执行前一次完成；

无须硬件支持，易于实现，但不允许程序在执行过程中移动位置。

动态重定位

地址转换推迟到最后的可能时刻，即进程执行时才完成；

允许程序在主存中移动、便于主存共享、主存利用率高。

5 存储保护与越界：基址+界限寄存器

保护操作系统不受用户进程所影响，保护用户进程不受其他用户进程所影响

方法

存储键保护

系统将主存划分成大小相等的若干存储块，并给每个存储块都分配一个单独的保护键（锁）；

在程序状态字PSW中设置有保护键字段，对不同的作业赋予不同的代码（钥匙）；钥匙和锁相配才允许访问。

界限寄存器（下页图）

上、下界防护：硬件为分给用户作业的连续的主存空间设置一对上、下界，分别指向该存储空间的上、下界。

基址、限长防护：基址寄存器存放当前正执行者的程序地址空间所占分区的始址，限长寄存器存放该地址空间的长度。

下限寄存器	2000
-------	------

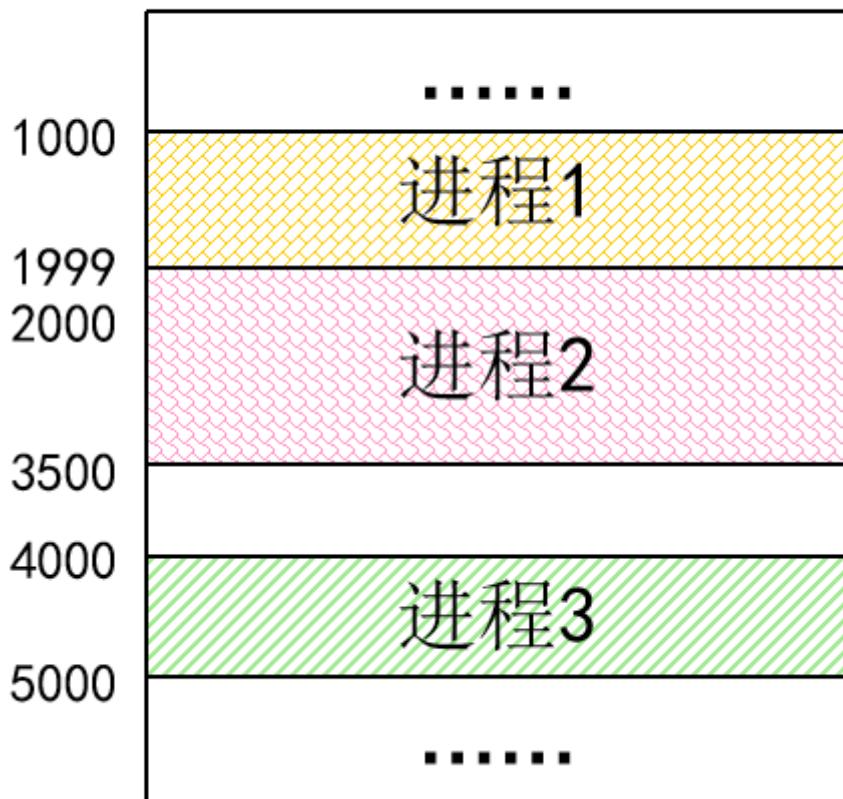
上限寄存器	3500
-------	------

基址寄存器	2000
-------	------

限长寄存器	1500
-------	------

进程id	下限+上限寄存器	基址+限长寄存器
1	1000+1999	1000+999
2	2000+3500	2000+1500
3	4000+5000	4000+1000

内存映像



6 分页：基本原理，逻辑地址结构，页和页框，页表，地址转换

慢表 (Page)：页表、段表存放在主存中，收到虚拟地址后要先访问主存，查询页表、段表，进行虚实地址转换。

快表 (TLB)：提高变换速度→用高速缓冲存储器存放常用的页表项。

页表是一张存放在**主存(即内存)**中的虚页号和实页号的对照表，记录着程序的虚页调入主存时被安排在主存中的位置，且页表一般长久的保存在内存中。

由于页表存放在主存中，因此程序每次访存至少需要两次：一次访存获取物理地址，第二次访存才获得数据。

6.1 基本原理

内存被划分成大小固定相等的块（页框），且块相对比较小，每个进程也被分成同样大小的块（页）。

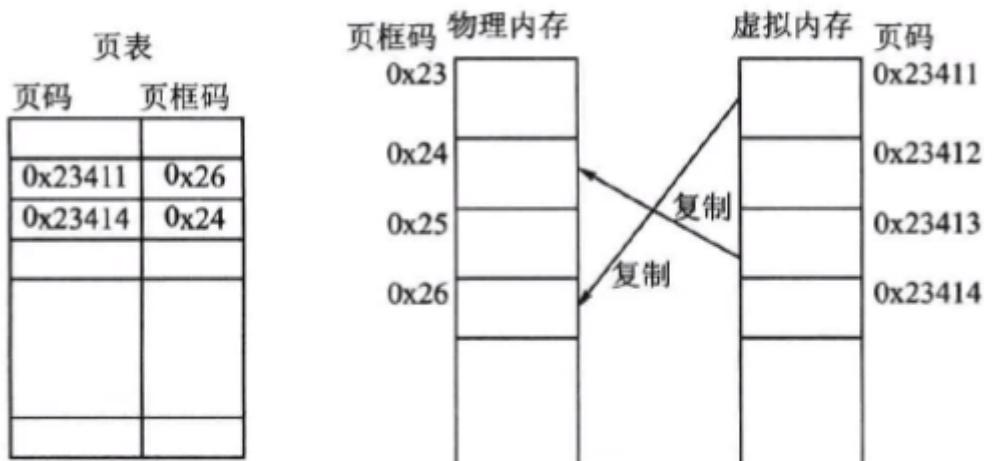
进程中称为页的块可以指定到内存中称为页框的可用块。

6.2 分页与固定分区的区别

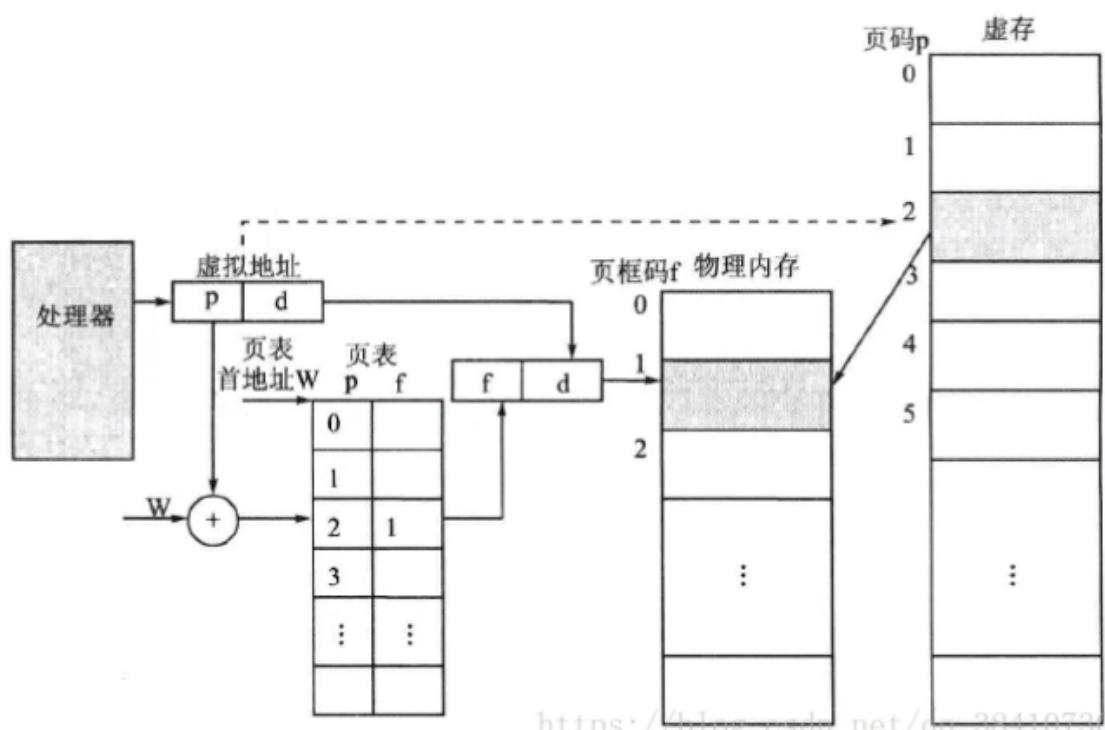
- 分页技术的分区相当小
- 一个程序可以占据多个分区
- 一个程序占据的多个分区不需要是连续的

6.3 逻辑地址结构

- 逻辑地址（页号，偏移量）
- 物理地址（页框号，偏移量）



https://blog.csdn.net/qq_38410730



https://blog.csdn.net/qq_38410730



逻辑内存

页号 页框号

页号	页框号
0	1
1	4
2	3
3	7

页表

页框号

0	
1	页0
2	
3	
4	页2
5	
6	页1
7	
	页3

物理内存

例：

- 说明：页大小为4B，页表如图所示，将逻辑地址0、3、4、13转换为相应物理地址
- 答案：20、23、24、9

0	5
1	6
2	1
3	2

页表

$$0/4=0 \dots 0$$

$$5 \times 4 + 0 = 20$$

$$3/4=0 \dots 3$$

$$5 \times 4 + 3 = 23$$

$$4/4=1 \dots 0$$

$$6 \times 4 + 0 = 24$$

$$13/4=3 \dots 1$$

$$2 \times 4 + 1 = 9$$

练习：逻辑地址到物理地址的转换

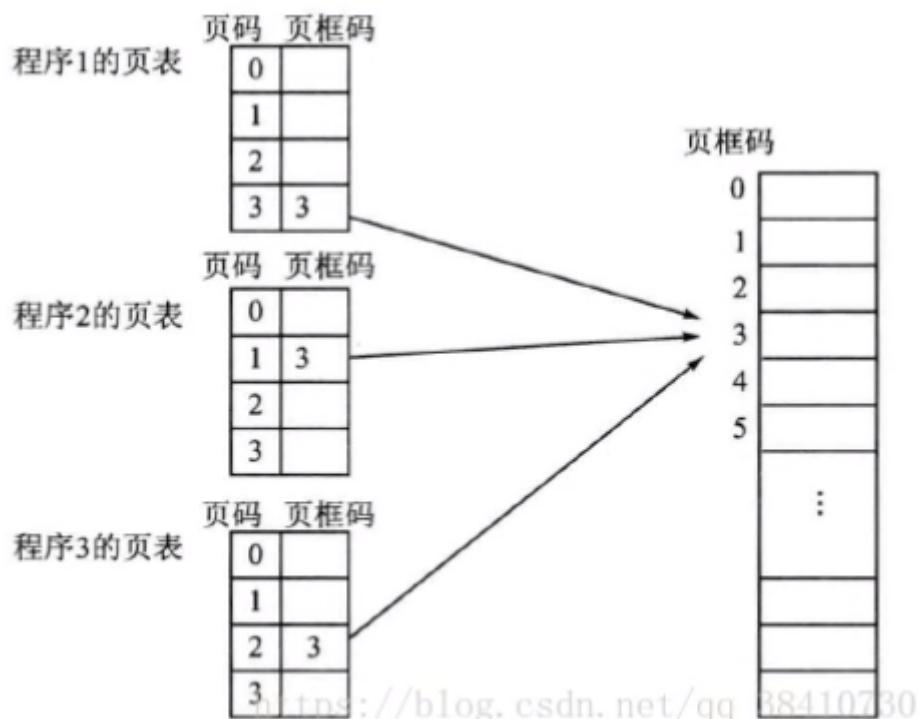
- 说明：页大小为1024B，页表如图所示，将逻辑地址1011、2148、3000、4000、5012转换为相应物理地址
- 答案：3059、1124、1976、7072、逻辑地址非法

$1011/1024=0\dots1011$	$2 \times 1024 + 1011 = 3059$
$2148/1024=2\dots100$	$1 \times 1024 + 100 = 1124$
$3000/1024=2\dots952$	$1 \times 1024 + 952 = 1976$
$4000/1024=3\dots928$	$6 \times 1024 + 928 = 7072$
$5012/1024=4\dots916$	页号4不存在

0	2
1	3
2	1
3	6

页表

6.4 页表共享



7 分段：基本原理，逻辑地址结构，段表，地址转换

7.1 基本原理

- 把程序和其相关的数据划分到几个段中。
- 段有一个最大长度限制，但不要求所有程序的所有段的长度都相等。

7.2 分段与动态分区的区别

- 一个程序可以占据多个分区
- 一个程序占据的多个分区不需要是连续的
- 会产生外部碎片，但跟动态分区比，会很小

7.3 逻辑地址到物理地址转换

- 段表：将逻辑地址映射为物理地址
- 段基地址：包含该段在内存中的开始物理地址
- 段界限：指定该段的长度
- 逻辑地址：段号s+段内偏移d
- 逻辑地址到物理地址的转换
 - 1) 段号与段表长度进行比较，若段号超过了段表长度，则越界（非法地址），否则转2)
 - 2) 根据段表始址和段号计算出该段对应段表项的位置，从中读出该段在内存的起始地址，检查段内地址是否超过该段的段长，若超过则越界（非法地址），否则转3)
 - 3) 将该段的起始地址与段内位移相加，从而得到要访问的物理地址

➤说明：段表如表1所示，将表2所示逻辑地址转换为相应物理地址

➤答案：见表3

段号	内存起始地址	段长
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

表1

段号	段内位移
2	88
4	100

表2

178	90+88
非法	100>96

表3

➤说明：段表如表1所示，将表2所示逻辑地址转换为相应物理地址

➤答案：见表3

段号	内存起始地址	段长
0	210	500
1	2350	20
2	100	90
3	1350	590
4	1958	95

表1

段号	段内位移	
0	430	210+430
1	10	2350+10
2	500	非法 500>90
3	400	1350+400
4	112	非法 112>95
5	32	非法 段号5不存在

表2

表3

第7章：内存管理作业

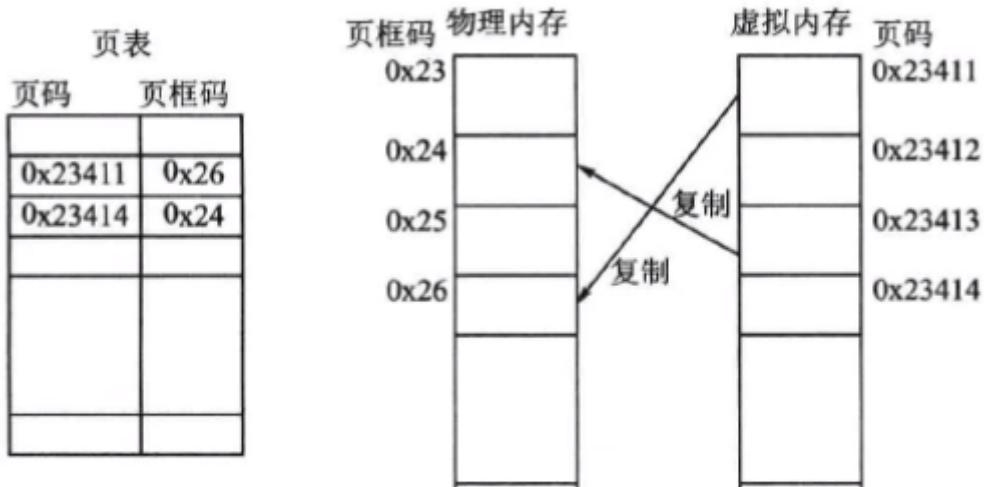
1 为什么需要重定位进程的能力？

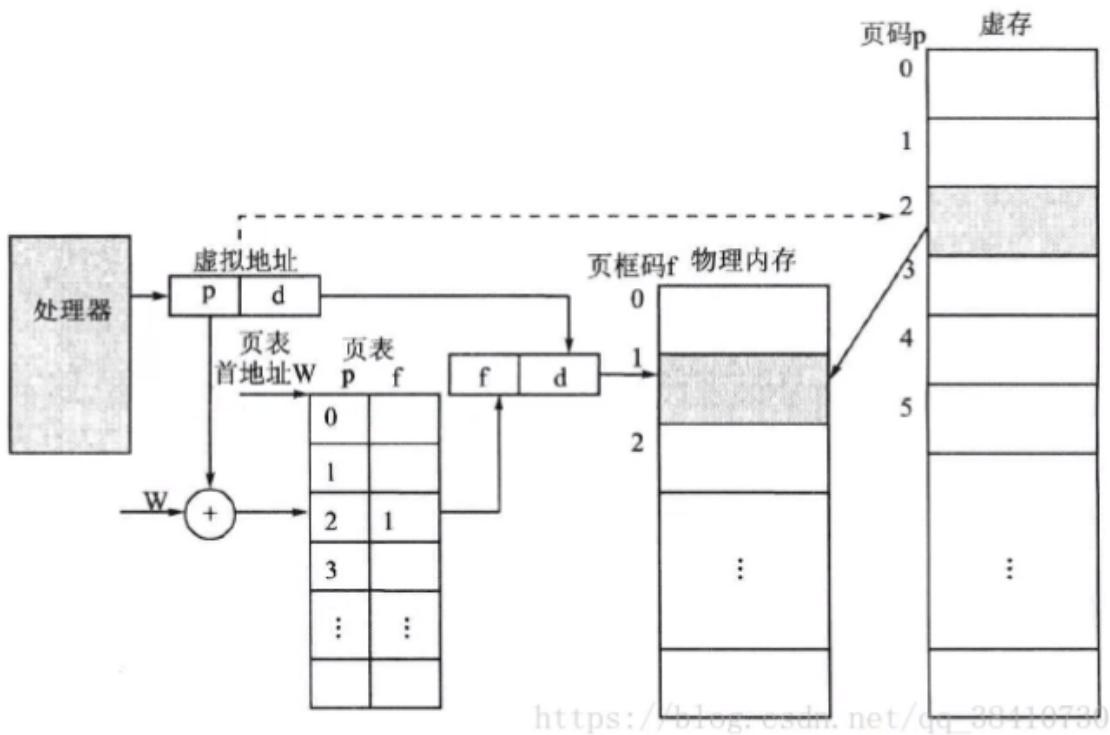
首先，程序员并不能事先知道在某个程序执行期间会有其他哪些程序驻留在内存中。

其次，为了提高处理器的利用率，允许进程换入或换出内存，当被换出内存的进程下一次被换入时，没必要也很难做到放在换出前的位置。

因此，需要重定位技术。

2 页和页框之间有什么区别？





内存被划分成大小固定相等的小块，称为页框，每个进程被分成同样大小的小块，称为页。

因此，页是逻辑上的划分，而页框是物理内存的划分，一页匹配一个页框。

3 习题1

假设使用动态分区，下图是经过数次放置和换出操作后的内存格局。

内存地址从左到右增长；灰色区域是分配给进程的内存块；白色区域是可用内存块。

最后一个放置的进程大小为2MB，用X标记。此后仅换出了一个进程。



a.换出的最大进程大小是多少？

1M

b.创建分区并分配给X之前，空闲块的大小是多少？

7M

c.下一个内存需求大小为3MB。在使用最佳适配/首次适配/下次适配/最差适配的情况下，分别在图上标出分配的内存区域。

最佳-3M, 首次-4M (左), 下次-5M, 最差-8M

4 习题2

一个1MB的内存块使用伙伴系统来分配内存。请画出类似图7.6的图来表示如下序列的结果：

A:请求70; B:请求35; C:请求80; 释放A; D:请求60; 释放B; 释放D; 释放C

	1MB						
请求70	A	128KB	256KB		512KB		
请求35	A	B	64KB	256KB		512KB	
请求80	A	B	64KB	C	128KB	512KB	
释放A	128KB	B	64KB	C	128KB	512KB	
请求60	D	64KB	B	64KB	C	128KB	512KB
释放B	D	64KB	128KB	C	128KB	512KB	
释放D	256KB		C	128KB		512KB	
释放C				1MB			

5 习题3

在一个简单分段系统中，包含如下段表：

段号	起始地址	长度（字节）
0	660	248
1	1752	422
2	222	198
3	996	604

对下面的每一个逻辑地址，确定其对应的物理地址或者说明段错误是否会发生。

- a. 0, 198 b. 2, 156 c. 1, 530 d. 3, 444
- e. 0, 222

a.0,198 $660+198=858$

b.2,156 $222+156=378$

c.1,530 段错误， $530>422$

d.3,444 $996+444=1440$

e.0,222 $660+222=882$

6 习题4

- 页式存储管理系统中，某进程页表如下。已知页面大小为1024字节，问逻辑地址600, 2700, 4000所对应的物理地址各是多少？
 - $600/1024=0\cdots600$
 - $7*1024+600=7768$
 - $2700/1024=2\cdots652$
 - $5*1024+652=5772$
 - $4000/1024=3\cdots928$ 非法地址

页号	页框号
0	7
1	3
2	5

第8章：虚拟内存

实存储器（实存）：内存

虚存储器（虚存）：磁盘

程序的局部性原理

指程序在执行过程中的一个较短时间内，所执行的指令地址或操作数地址分别局限于一定的存储区域中。又可细分时间局部性和空间局部性。

时间局部性：最近访问过的程序代码和数据很快又被访问。

空间局部性：某存储单元被使用之后，其相邻的存储单元也很快被使用。

虚拟内存（virtual memory）：允许进程的执行不必完全在内存中，程序可以比物理内存大。

在许多情况下不需要将整个程序放到内存中：

- 处理异常错误条件的代码（几乎不执行）
- 数组、链表和表通常分配了比实际所需更多的内存
- 程序的某些选项或特点可能很少使用

能够执行只有部分在内存中的程序的好处：

- 程序不再受现有的物理内存空间限制
- 更多程序可同时执行，CPU利用率相应增加
- 用户程序会运行的更快

虚拟存储管理实现技术：

- 使用分页实现虚存
- 使用分段实现虚存
- 使用段页式实现虚存

1 虚拟地址概念，实地址概念

虚拟地址：在虚拟内存中分配给某一位置的地址，它使得该位置可被访问，就好像是主内的一部分那样

实地址：内存中存储位置的地址

2 虚拟分页：基本原理，虚实地址转换

分页式虚存不把作业信息(程序和数据)全部装入主存，仅装入立即使用的页面，在执行过程中访问到不在主存的页面时，产生缺页中断，再从磁盘动态地装入。

怎样才能发现页面不在主存中呢？怎样处理这种情况呢？

采用的办法是：扩充页表的内容，增加驻留标志位和页面辅存的地址等信息。

3 缺页中断处理过程

当内存中所有页框都被占据，并且需要读取一个新页以处理一次缺页中断时，置换策略决定当前在内存中的哪个页将被置换。

目标：移出最近最不可能访问的页。

方法：基于过去的行为预测将来行为。

基本算法：

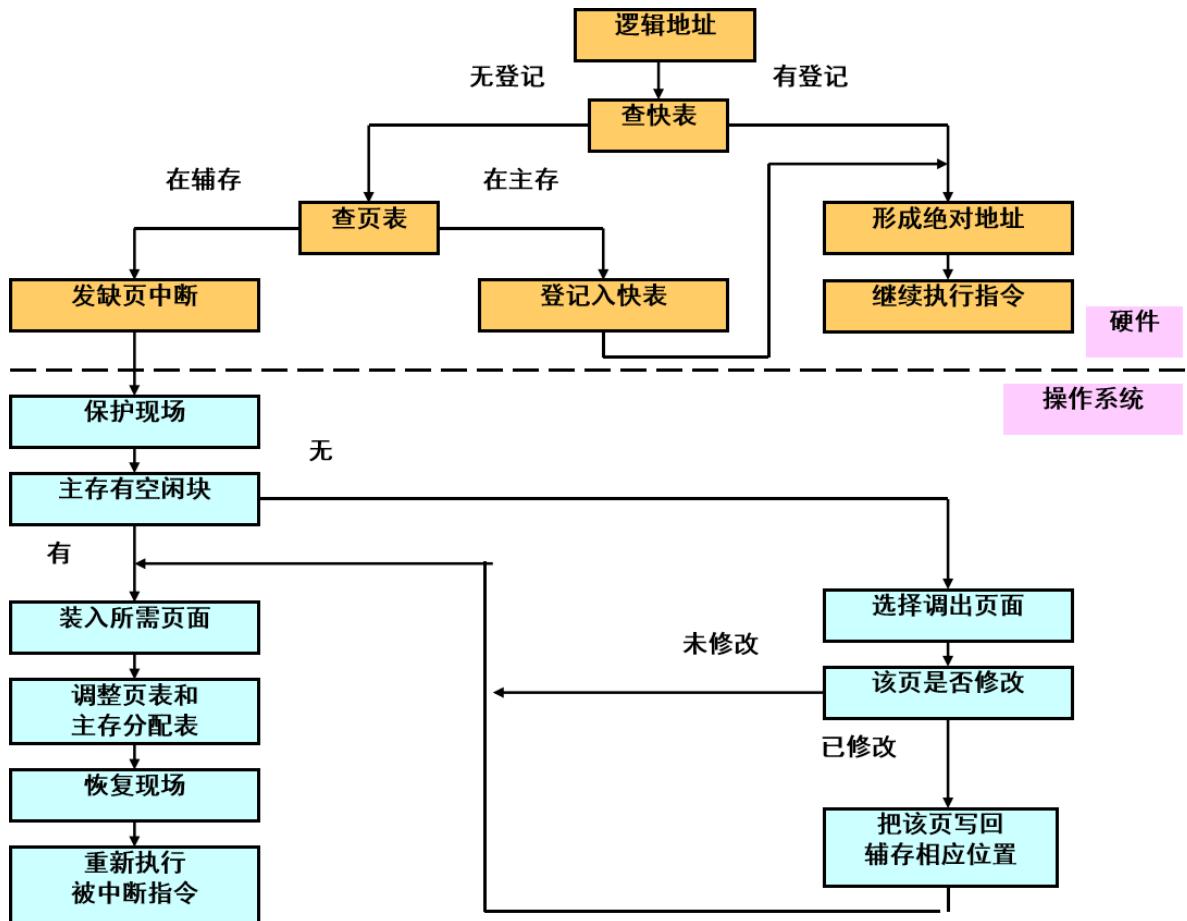
- 最佳OPT
- 最近最少使用LRU
- 先进先出FIFO
- 时钟Clock

4 转换检测缓冲区 TLB (快表)。根据内存访问时间、TLB 访问时间和 TLB 命中率，求将逻辑地址转换成物理地址并访问内存数据所需的有效访问时间 (见作业)

4.1 转换检测缓冲区 (translation look-aside buffer, TLB)

TLB与页表一起工作：

- 快表项包含页号及对应的页框号，当把页号交给快表后，它通过并行匹配同时对所有快表项进行比较。
- 如果找到，立即输出页框号，并形成物理地址；
- 如果找不到，再查找主存中的页表以形成物理地址，同时将页号和页框号登记到快表中；
- 当快表已满且要登记新页时，系统需要淘汰旧的快表项。



4.2 采用快表的地址转换

假定访问主存时间为100毫微秒，访问快表时间为20毫微秒，相联存储器为32个单元时快表命中率可达90%，按逻

辑地址存取的平均时间为：

$$(100 + 20) \times 90\% + (100+100+20) \times (1-90\%) = 130$$

比两次访问主存的时间 $100 \times 2 = 200$ 下降了三成多。

5 虚拟分段和虚拟段页式的基本原理

5.1 虚拟分段的基本原理

分段式虚拟存储系统把作业的所有分段的副本都存放在辅助存储器中，当作业被调度投入运行时，首先把当前需要的一段或几段装入主存，在执行过程中访问到不在主存的段时再把它们装入。

段表也需要增加一些控制位。

5.2 虚拟段页式的基本原理

分页对程序员是透明的，消除了外部碎片，可以更有效地使用内存。此外，移入或移出的块是大小相等的，易于处理。

分段对程序员是可见的，具有处理不断增长的数据结构的能力以及支持共享和保护的能力。

把两者结合起来就是段页式存储管理。

内存划分成大小相等的页框。

用户的地址空间被程序员划分成许多段，每个段一次划分成许多固定大小的页，页的长度等于内存中的页框大小。

6 虚拟分页的置换算法：最佳置换 OPT、LRU、先进先出 FIFO

6.1 最佳置换 OPT

- （可用页框数量3）引用串如下：

- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0
1 7 0 1

引用串	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
帧1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	
帧2		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	
帧3			1	1	1	3	3	3	3	3	3	3	1	1	1	1	1	1	1	
置换	F	F	F	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R		

$$\text{缺页率 } p=9/20=45\%$$

6.2 最近最少使用 LRU

- 思想：置换过去最长时间没有使用的页
- 例（可用页框数量3）引用串如下：

- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0
1

引用串	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
帧1	7	7	7	2	2	2	2	4	4	4	0	0	1	1	1	1	1	1	1	
帧2		0	0	0	0	0	0	0	3	3	3	3	3	0	0	0	0	0	0	
帧3			1	1	1	3	3	3	2	2	2	2	2	2	2	2	7	7	7	
置换	F	F	F	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R		

$$\text{缺页率 } p=12/20=60\%$$

6.3 先进先出 FIFO

- Belady异常：页错误率随着所分配的帧数增加而增加的现象，例：

	1	2	3	4	1	2	5	1	2	3	4	5
帧1	1	1	1	4	4	4	5	5	5	5	5	5
帧2		2	2	2	1	1	1	1	1	3	3	3
帧3			3	3	3	2	2	2	2	2	4	4
缺页:	F	F	F	R	R	R	R	R	R	R	R	R

可用帧数3,
9次缺页

	1	2	3	4	1	2	5	1	2	3	4	5
帧1	1	1	1	1	1	1	5	5	5	5	4	4
帧2		2	2	2	2	2	2	1	1	1	1	5
帧3			3	3	3	3	3	3	3	2	2	2
帧4				4	4	4	4	4	4	3	3	3
缺页:	F	F	F	F			R	R	R	R	R	R

可用帧数4,
10次缺页

7 置换过程及缺页次数的计算（注：计算页框填满之前和之后发生的总缺页次数即可）

已知页面走向为1、2、1、3、1、2、4、2、1、3、4，且开始执行时内存中没有页面。

若只给该作业分配2个物理块。

- 1.当采用FIFO、OPT、LRU页置换算法时缺页率为多少？
- 2.假定现有一种置换算法，该算法淘汰页面的策略为当需要淘汰页面时，就把刚使用过的页面作为淘汰对象，试问就相同的页面走向，其缺页率又为多少？
- 3.假如分配3个物理块呢？

➤ 2个物理块

1	2	1	3	1	2	4	2	1	3	4
1	1		3	3	2	2		1	1	4
	2		2	1	1	4		4	3	3
<hr/>										
1	1		1		1	4		4	4	
	2		3		2	2		1	3	
<hr/>										
1	1		1		1	4		1	1	4
	2		3		2	2		2	3	3
<hr/>										
1	1		3	1		1	1		3	4
	2		2	2		4	2		2	2

FIFO $9/11=81.8\%$

OPT $7/11=63.6\%$

LRU $8/11=72.7\%$

NEW $8/11=72.7\%$

30

➤ 3个物理块

1	2	1	3	1	2	4	2	1	3	4
1	1		1			4		4		
	2		2			2		1		
			3			3		3		
1	1		1			1			3	
	2		2			2			2	
			3			4			4	
1	1		1			1			1	1
	2		2			2			2	4
			3			4			3	3
1	1		1			1	1			1
	2		2			4	2			2
			3			3	3			4

FIFO $5/11=45.5\%$

OPT $5/11=45.5\%$

LRU $6/11=54.5\%$

NEW $6/11=54.5\%$

31

8 抖动

抖动在分页存储管理系统中，内存中只存放了那些经常使用的页面，而其它页面则存放在外存中，当进程运行需要的内容不在内存时，便启动磁盘读操作将所需内容调入内存，若内存中没有空闲物理块，还需要将内存中的某页面置换出去。

也就是说，系统需要不断地在内外存之间交换信息。

若在系统运行过程中，刚被淘汰出内存的页面，过后不久又要访问它，需要再次将其调入。而该页面调入内存后

不久又再次被淘汰出内存，然后又要访问它。如此反复，使得系统把大部分时间用在了页面的调入/换出上，而几乎不能完成任何有效的工作，这种现象称为抖动。

第8章：虚拟内存作业

1 解释什么是抖动。

抖动是虚存管理方案中可能出现的一种现象，处理器花费在交换上的时间多于执行指令的时间。

2 转换检测缓冲区的目的是什么？

TLB是一个缓冲，包含最近一段时间频繁用到的页表项，从而能够减少数据访问需要的时间。

3 驻留集和工作集有什么区别？

驻留集是指进程当前在内存里的页数，而工作集是指进程最近被引用过的页数。

4 习题1

设当前运行进程的页表如下。所有数字都是十进制的，所有计数从0开始，所有地址都是内存字节地址。页大小为1024B。

a. 详细描述虚拟地址是如何转换为物理地址的。

略

b. 如果下列虚拟地址能转换为物理地址，物理地址将是多少？（不处理可能出现的缺页中断）

1052, 2221, 5499

- $1052/1024 = 1\dots28 \quad 7 \times 1024 + 28 = 7196$
- $2221/1024 = 2\dots173 \quad \text{不在内存}$
- $5499/1024 = 5\dots379 \quad 0 \times 1024 + 379 = 379$

虚页号	有效位	引用位	修改位	页框号
0	1	1	0	4
1	1	1	1	7
2	0	0	0	-
3	1	0	0	2
4	0	0	0	-
5	1	0	1	0

5 习题2

考虑如下的页访问序列

: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 请画图说明页框的分配情况（三个物理块）：

- FIFO
- LRU
- 最佳（假设后续页面访问序列 1, 2, 0, 1, 7, 0, 1）
- 对每种情况计算缺页中断次数和缺页率。只计算页框初始化后发生的缺页中断。

后续访问序列：1,2,0,1,7,0,1

7	0	1	2	0	3	0	4	2	3	0	3	2
7	7	7	2		2	2	4	4	4	0		
	0	0	0		3	3	3	2	2	2		
		1	1		1	0	0	0	3	3		
7	7	7	2		2		4	4	4	0		
	0	0	0		0		0	0	3	3		
		1	1		3		3	2	2	2		
7	7	7	2		2		2			2		
	0	0	0		0		4			0		
		1	1		3		3			3		

FIFO
7/13=53.8%

LRU
6/13=46.2%

OPT
4/13=30.8%

考虑一个使用单级页表的分页系统。假设所需的页表总在内存中。

- a. 如果一次物理内存访问耗时200ns，那么一次逻辑内存访问耗时多少？

$$200 \times 2 = 400$$

- b. 现在添加一个MMU，对每次命中或缺页MMU造成20ns开销。假设85%的内存访问都命中MMU TLB。有效访问时间(EMAT)是多少？

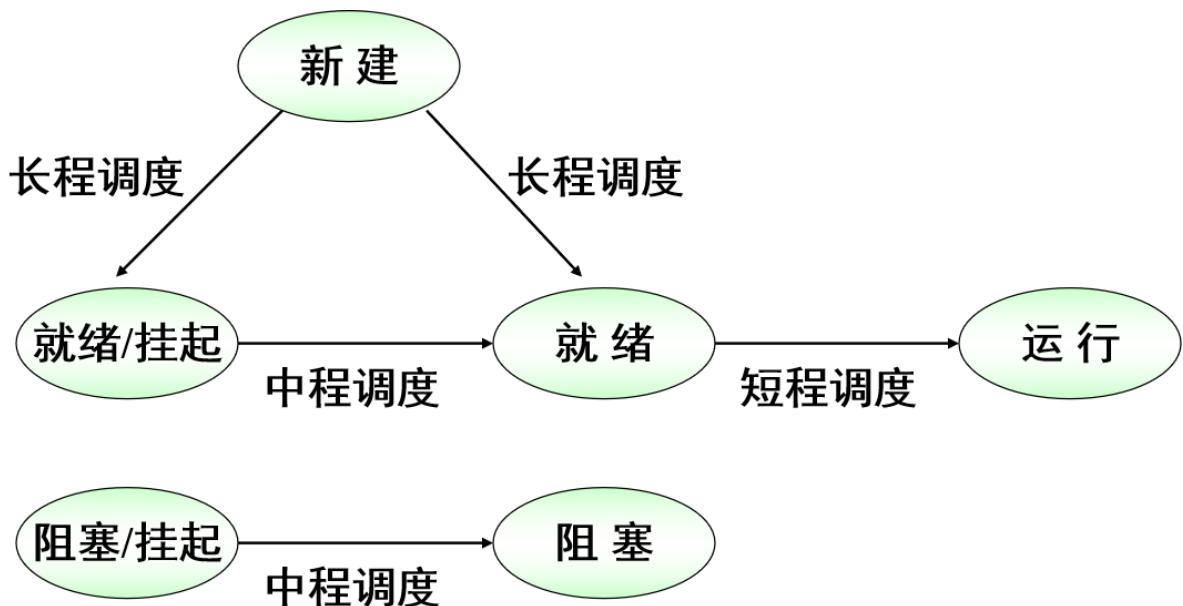
$$(20+200) * 85\% + (20+200+200) * 15\% = 250$$

- c. 解释TLB命中率是如何影响EMAT的？

TLB的命中率越大，有效访问时间越小。

第9章：单处理器调度

1 处理器调度的类型 - 长程，中程，短程



1.1 长程调度

1.1.1 何时调度？

- 有作业终止时
- 处理器的空余时间片超过了一定的阈值

1.1.2 调度哪个？

- 先来先服务
- 优先级
- 实时性
- I/O需求

1.2 中程调度

- 换入：取决于管理系统并发度的要求；
- 换出：进程的存储需求。

1.3 短程调度

分派程序，精确地决定下一次执行哪一个进程。

长程调度程序执行频率较低；

中程调度程序执行频率稍高；

短程调度程序执行频率最高。

2 调度准则与指标

1、面向用户，与性能相关

- 周转时间
 - 从提交到完成之间的时间间隔
- 响应时间
 - 从提交到开始接收响应之间的时间间隔
- 最后期限
 - 进程完成的最后期限

2、面向用户，与性能无关

- 可预测性
 - 希望提供给用户的服务能够随着时间的流逝展现给用户一贯相同的特性，而与系统执行的其他工作无关。

- 吞吐量
 - 单位时间内完成的进程数目
- 处理器利用率
 - 处理器处于忙的状态的时间百分比

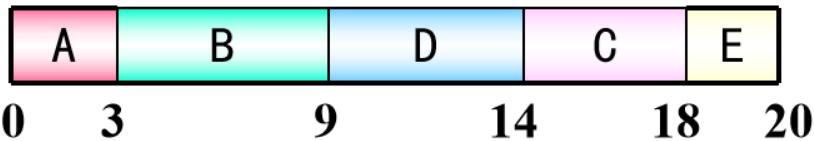
4、面向系统，与性能无关

- 公平性
 - 进程被平等对待
- 强制优先级
 - 进程被指定优先级，调度策略优先选择高优先级进程。
- 平衡资源
 - 保持系统中所有资源处于繁忙状态，较少适用紧缺资源的进程应该受到照顾。

3 非抢占式调度、抢占式调度

3.1 非抢占式调度

进程	到达时间	服务时间	优先级
A	0	3	3
B	2	6	2
C	4	4	4
D	6	5	1
E	8	2	5

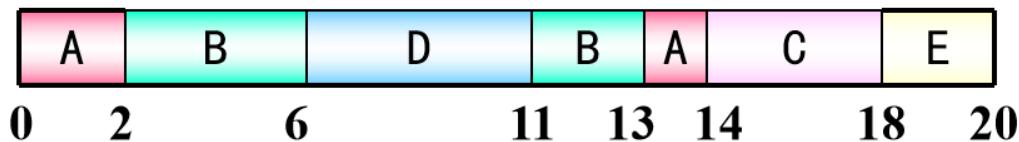


$$\text{平均周转时间: } (3+7+14+8+12)/5=8.8$$

$$\text{平均归一化周转时间: } (1+1.17+3.5+1.6+6)/5=2.65$$

3.2 抢占式调度

进程	到达时间	服务时间	优先级
A	0	3	3
B	2	6	2
C	4	4	4
D	6	5	1
E	8	2	5



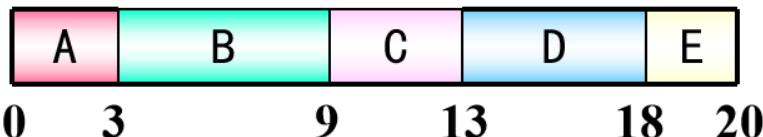
$$\text{平均周转时间: } (14+11+14+5+12)/5=11.2$$

$$\text{平均归一化周转时间: } (4.67+1.83+3.5+1+6)/5=3.4$$

4 调度算法：先来先服务(FCFS)、轮转RR、最短进程优先(SPN)、最高响应比优先(HRRN)。计算“周转时间”、“归一化周转时间(带权周转时间 Tr/Ts)”及所有作业的平均值

4.1 先来先服务 FCFS

进 程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



$$\text{平均周转时间: } (3+7+9+12+12)/5=8.6$$

$$\text{平均归一化周转时间: } (1+1.17+2.25+2.4+6)/5=2.56$$

4.2 轮转 RR q=1

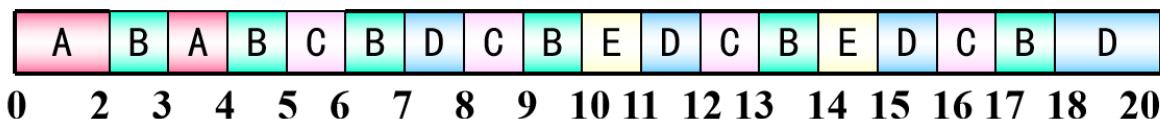
在分时系统中都采用时间片轮转算法进行进程调度。

时间片是指一个较小的时间间隔，通常为10-100毫秒。

在简单的轮转算法中，系统将**所有的就绪进程按先来先服务（即FIFO）规则排成一个队列**，将CPU分配给队首进程，且规定每个进程最多允许运行一个时间片；若时间片使用完进程还没有结束，则被加入就绪FIFO队列队尾，并把CPU交给下一个进程。

近几年，ulob时间片轮转算法只用于进程调度，它属于抢占调度方式。

进 程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

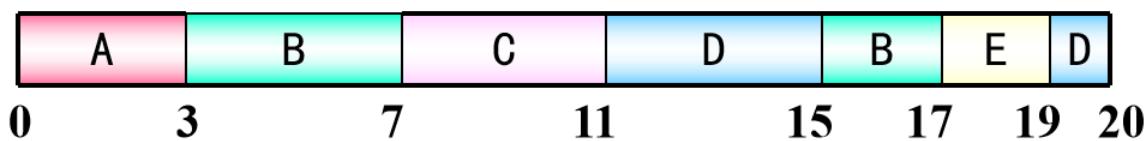


$$\text{平均周转时间: } (4+16+13+14+7)/5=10.8$$

$$\text{平均归一化周转时间: } (1.33+2.67+3.25+2.8+3.5)/5=2.71$$

4.3 轮转 RR q=4

进 程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

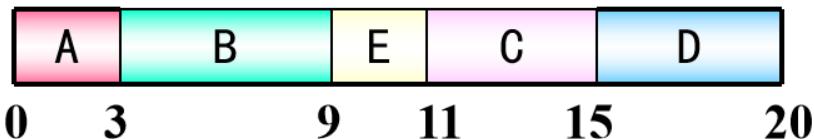


$$\text{平均周转时间: } (3+15+7+14+11)/5=10$$

$$\text{平均归一化周转时间: } (1+2.5+1.75+2.8+5.5)/5=2.71$$

4.4 最短进程优先 SPN

进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



$$\text{平均周转时间: } (3+7+11+14+3)/5=7.6$$

$$\text{平均归一化周转时间: } (1+1.17+2.75+2.8+1.5)/5=1.84$$

4.5 最高响应比优先 HRRN

$$R=(w+s)/s$$

R: 响应比

w: 等待处理器的时间

s: 预计的服务时间

$$\text{响应比} = (\text{作业处理时间} + \text{作业等待时间}) / \text{作业处理时间}$$

进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

B运行完，计算CDE的响应比：

$$R_C = (5+4)/4 = 2.25$$

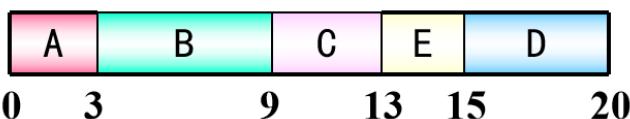
$$R_D = (3+5)/5 = 1.6$$

$$R_E = (1+2)/2 = 1.5$$

C运行完，计算DE的响应比：

$$R_D = (7+5)/5 = 2.4$$

$$R_E = (5+2)/2 = 3.5$$



$$\text{平均周转时间: } (3+7+9+14+7)/5=8$$

$$\text{平均归一化周转时间: } (1+1.17+2.25+2.8+3.5)/5=2.14$$

第9章：单处理器调度作业

1 简要描述三种类型的处理器调度。

- 长程调度决定是否把进程添加到当前活跃的进程集合中；
- 中程调度用于内外存的交换；
- 短程调度真正决定处理器下次要执行的就绪进程。

2 抢占式和非抢占式调度有什么区别？

- 抢占：当前正在运行的进程可能被中断，并转移到就绪状态；
- 非抢占：一旦进程处于运行状态，除非阻塞，会一直运行到终止状态。

第10章：多处理器、多核和实时调度

1 多处理器系统中，采用简单的 FCFS 或“静态优先级+FCFS”调度算法就足够了

- 多处理器情况下，调度原则的选择没有在单处理器中显得重要。
- 多处理系统中一般使用简单的FCFS或者在静态优先级方案中使用FCFS。

2 实时任务分类：硬、软，周期性、非周期性

实时任务的分类：

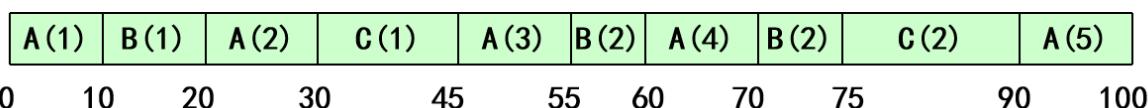
- 硬实时任务
- 软实时任务
- 周期性任务
- 非周期性任务

第10章：多处理器、多核和实时调度作业

1 习题1

考虑下表所示3个周期性任务，使用“最早完成最后期限调度”给出这组任务的调度图。

进程	到达时间	执行时间	完成最后期限
A(1)	0	10	20
A(2)	20	10	40
...
B(1)	0	10	50
B(2)	50	10	100
...
C(1)	0	15	50
C(2)	50	15	100
...



第11章：I/O管理和磁盘调度

执行I/O的三种技术：

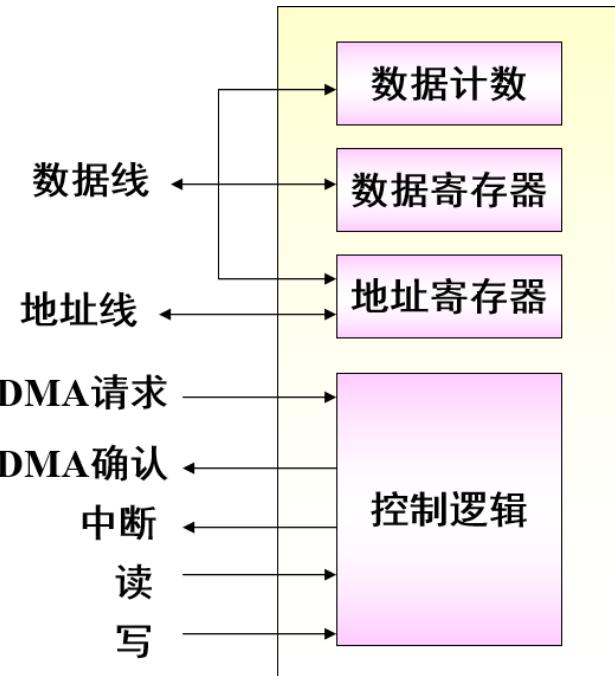
- 程序控制I/O
- 中断驱动I/O
- 直接存储器访问 (DMA)

1 程序控制 I/O：CPU 忙等 I/O 结束，CPU 与设备串行工作

2 中断驱动 I/O：各种设备通用，中断次数多

3 直接存储器访问 DMA 原理与 I/O 过程

DMA单元能够模拟处理器，像处理器一样获得系统总线的控制权，利用系统总线与存储器进行双向数据传送。



DMA技术工作流程

- 当处理器想读或写一块数据时，通过向DMA模块发送以下信息来给DMA模块发出一条命令：
 - 请求读或写操作的信号，通过读写控制线发送。
 - 相关的I/O设备地址，通过数据线发送。
 - 从存储器中读或往存储器中写的起始地址，在数据线上传送，并由DMA模块保存在其地址寄存器中。
 - 读或写的字数，通过数据线传送，并由DMA模块保存在其数据计数寄存器中。
- 处理器继续执行其工作；
- DMA模块直接从存储器中或往存储器中传送整块数据，一次传送一个字；
- 传送结束后，DMA模块给处理器发送一个中断信号。

6

4 缓冲 buffer 的主要作用：缓和 CPU 与 I/O 设备间速度不匹配矛盾，提高并行性

引入缓冲的目的：

- 改善中央处理器与外围设备之间速度不配的矛盾
- 提高CPU和I/O设备的并行性

例：某个用户进程需要从磁盘中读入多个数据块，对磁盘单元执行一个I/O命令，并等待（忙等或进程挂起）数据传送完毕。

存在的问题：

- 程序被挂起，等待相对比较慢的I/O完成。

- 干扰了操作系统的交换决策。

5 磁盘访问时间：寻道时间，旋转延迟时间，传输时间

6 磁盘调度算法：先进先出，最短服务时间优先算法(SSTF)，电梯。 计算平均寻道长度

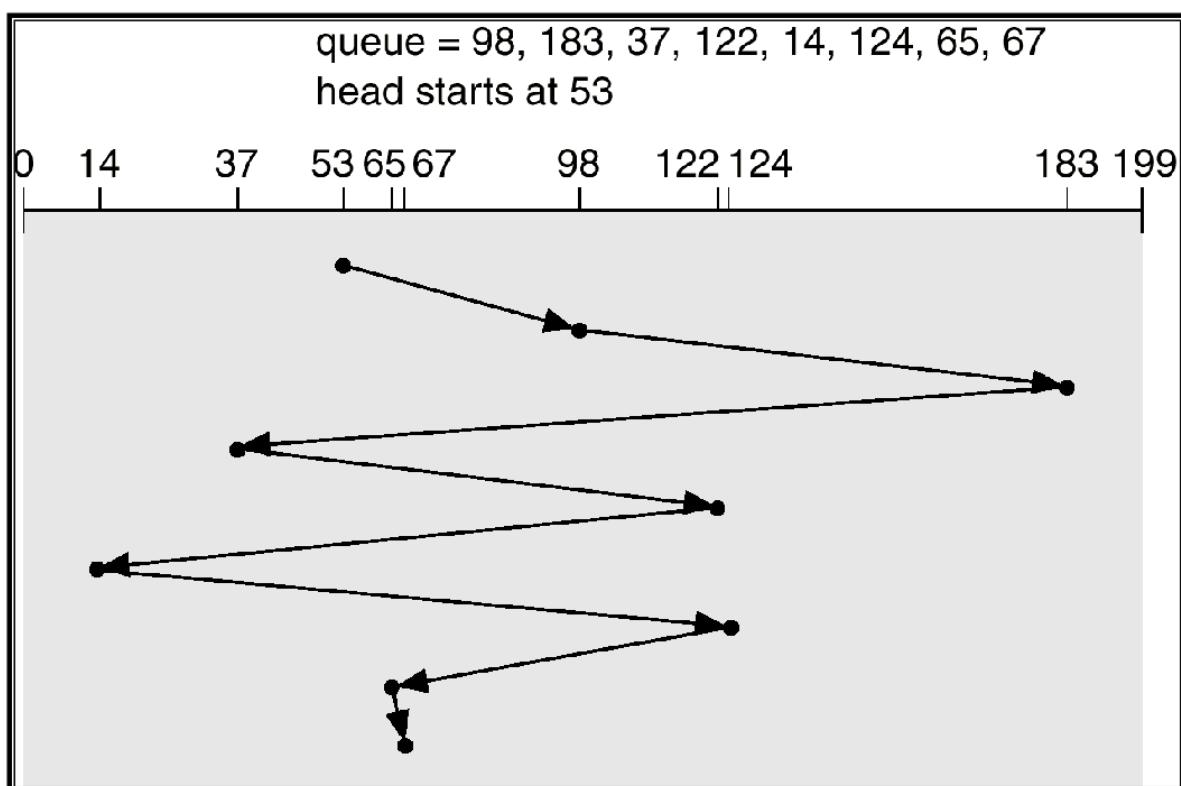
有一个磁盘队列，其I/O请求顺序如下：

98, 183, 37, 122, 14, 124, 65, 67

磁头开始位于53

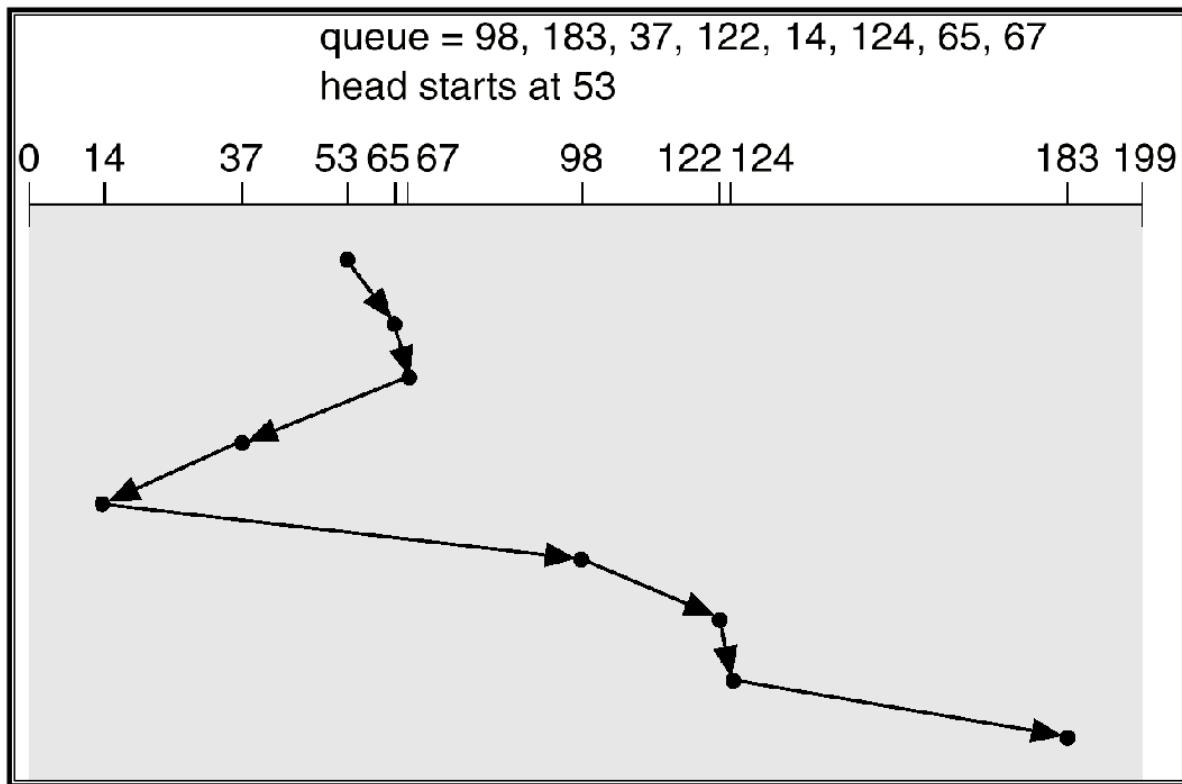
6.1 先进先出 FIFO

FIFO：平均寻道长度为 $640/8=80$



6.2 最短服务时间优先算法(SSTF)

SSTF：平均寻道长度为 $236/8=29.5$



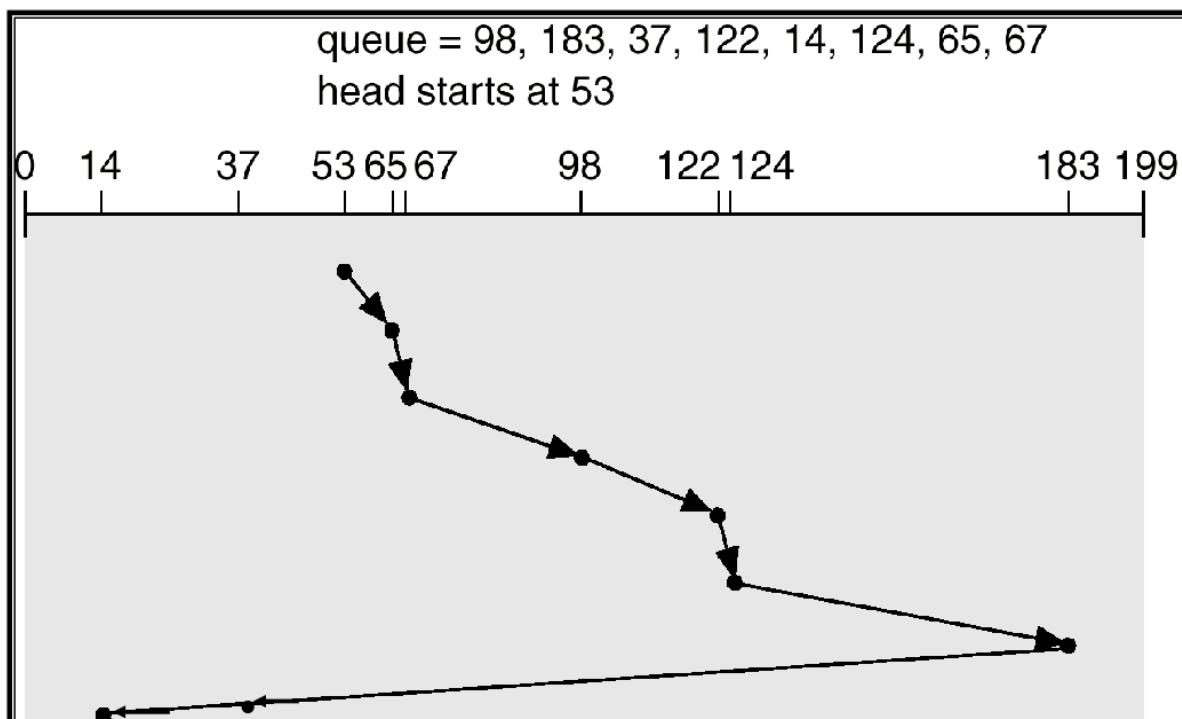
6.3 电梯SCAN

有一个磁盘队列，其I/O请求顺序如下：

98, 183, 37, 122, 14, 124, 65, 67

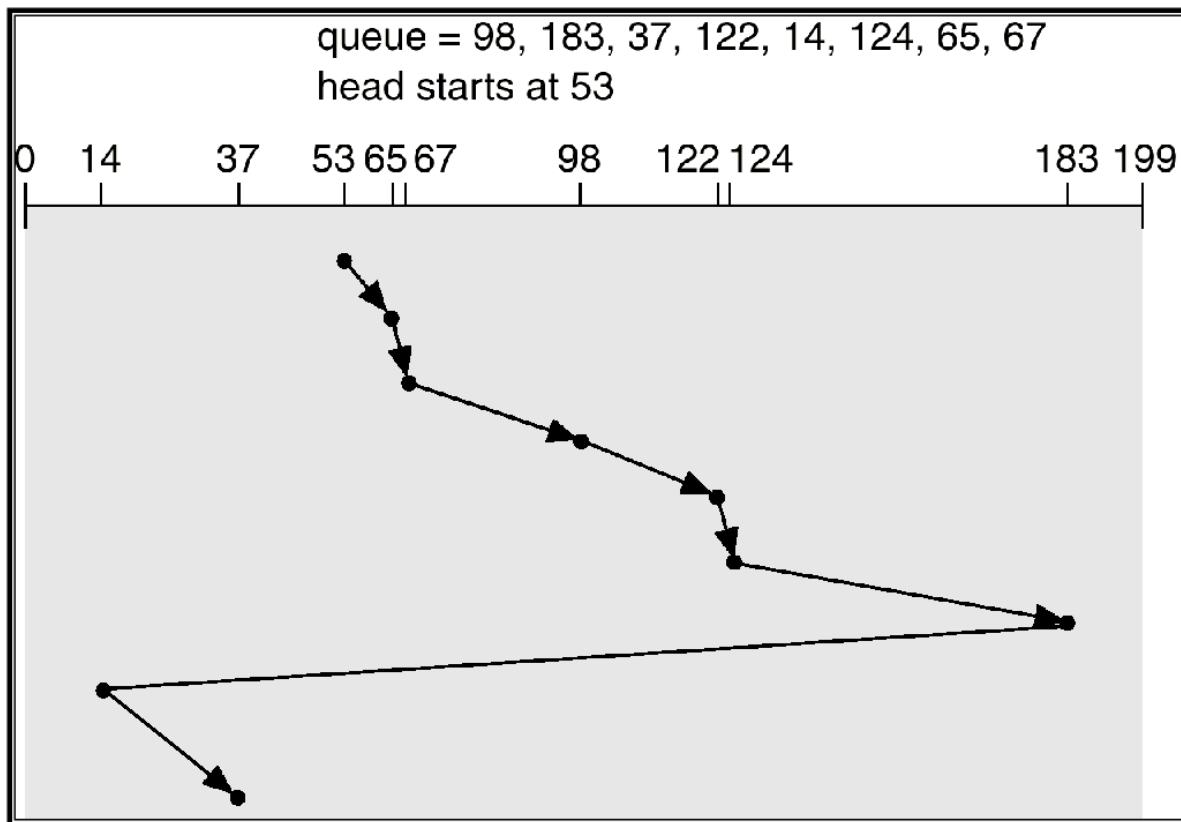
磁头开始位于53，向大方向移动

SCAN：平均寻道长度为 $299/8=37.375$



6.4 C-SCAN调度

C-SCAN : 平均寻道长度为 $322/8=40.25$



7 RAID 的核心技术：条带化，平行访问，块交叉校验，镜像。RAID 0，RAID 1

独立磁盘冗余阵列 (Redundant Array of Independent Disk)

RAID方案包括了7个级别，从0到6。

RAID分级如下所示：

- RAID0：无冗余和无校验的磁盘阵列。
- RAID1：镜像磁盘阵列。
- RAID2：采用纠错的海明码的磁盘阵列。
- RAID3：位交叉奇偶校验的磁盘阵列。
- RAID4：块交叉奇偶校验的磁盘阵列。
- RAID5：无独立校验的奇偶校验磁盘阵列。

不同级别表明了不同的设计体系结构，有三个共同特性：

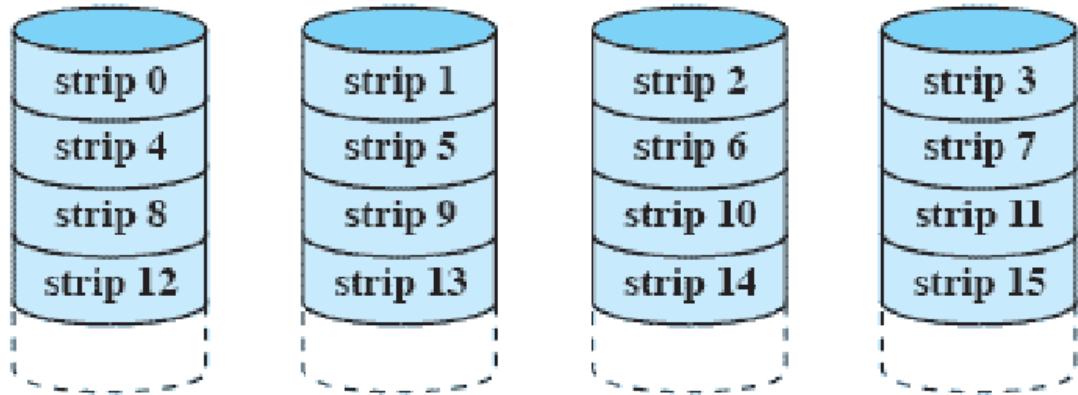
- RAID是一组物理磁盘驱动器，操作系统把它视为一个单个的逻辑驱动器。
- 数据分布在物理驱动器阵列中--条带化。
- 使用冗余的磁盘容量保存奇偶检验信息，从而保证当一个磁盘失效时，数据具有可恢复性。

7.1 条带化

一个条带可以是一个物理块、扇区或别的某种单元。条带被循环映射到连续的阵列成员中。

一组逻辑上连续的条带，如果恰好一个条带映射到一个阵列成员上，则称为一条条带。

优点：如果一个I/O请求由多个逻辑上连续的条带组成，该请求可以并行处理，从而减少I/O传输时间。

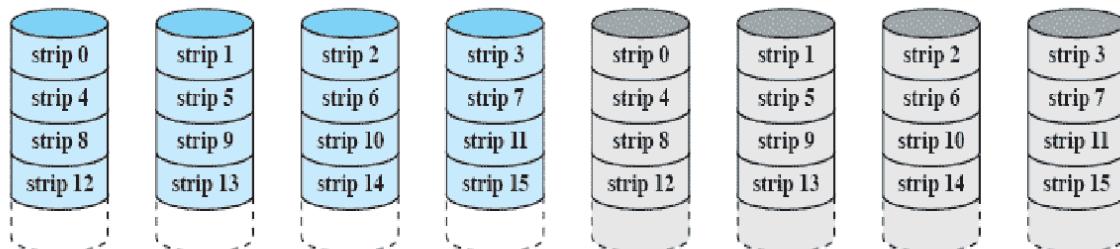


(a) RAID 0 (non-redundant)

7.2 镜像

每个逻辑条带映射到两个单独的物理磁盘上，使得阵列中的每个磁盘都有一个包含相同数据的镜像磁盘。

可靠性较好，但成本较高。

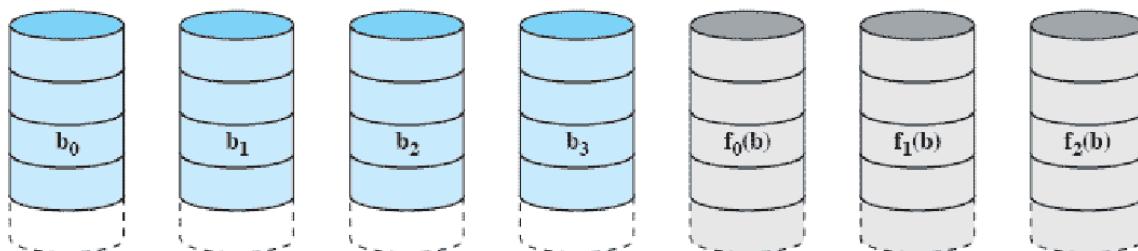


(b) RAID 1 (mirrored)

7.3 错误校正码

每个数据磁盘中的相应位都计算一个错误校正码，这个码位保存在多个奇偶检验磁盘中相应的位中。

成本仍然较高。

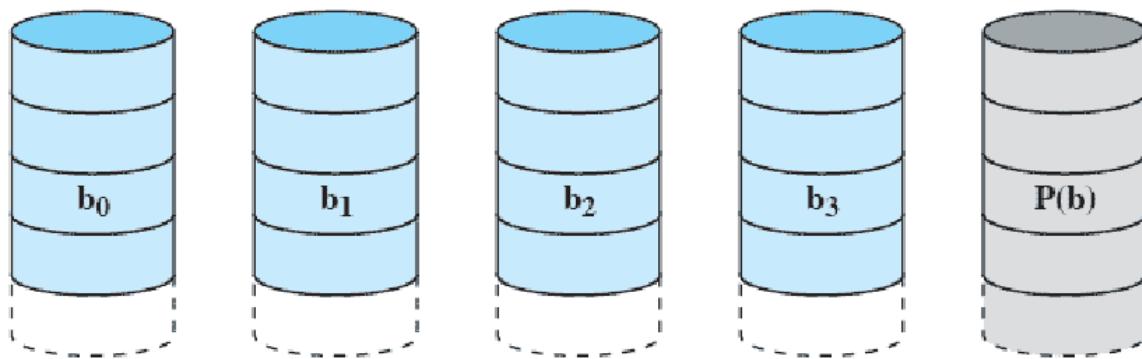


(c) RAID 2 (redundancy through Hamming code)

7.4 奇偶校验位

为所有数据磁盘中同一位置的位的集合计算一个简单的奇偶校验位，而不是错误校正码。

以较低的成本保证较好的可靠性。



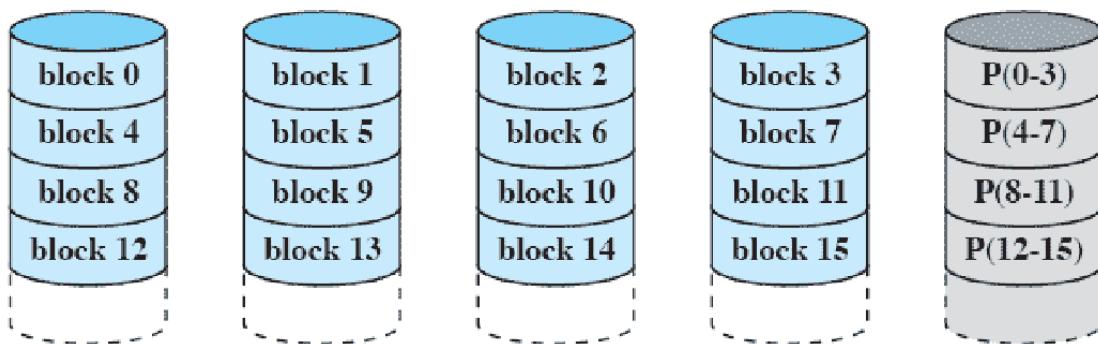
(d) RAID 3 (bit-interleaved parity)

7.5 块交叉校验

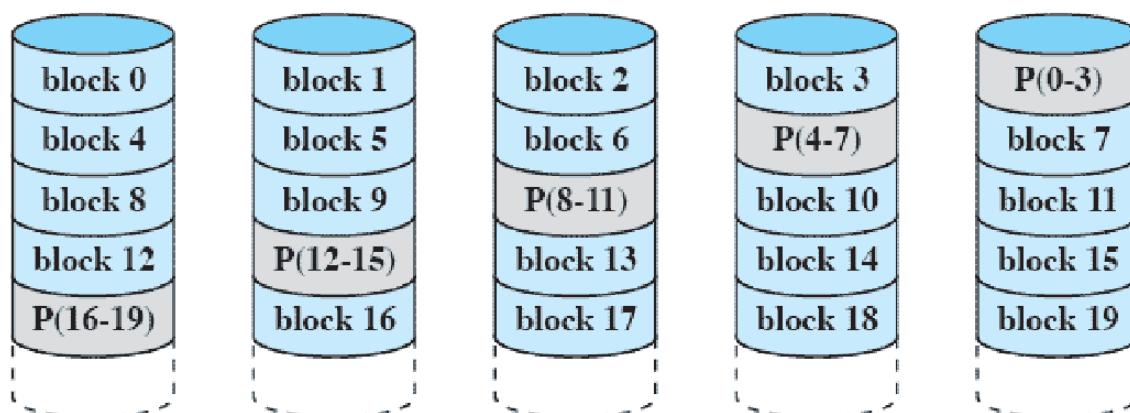
RAID4 与 RAID3 的原理大致相同，区别在于条带化的方式不同。

RAID4按照块的方式来组织数据，写操作只涉及当前数据盘和校验盘两个盘，多个 I/O 请求可以同时得到处理，提高了系统性能。

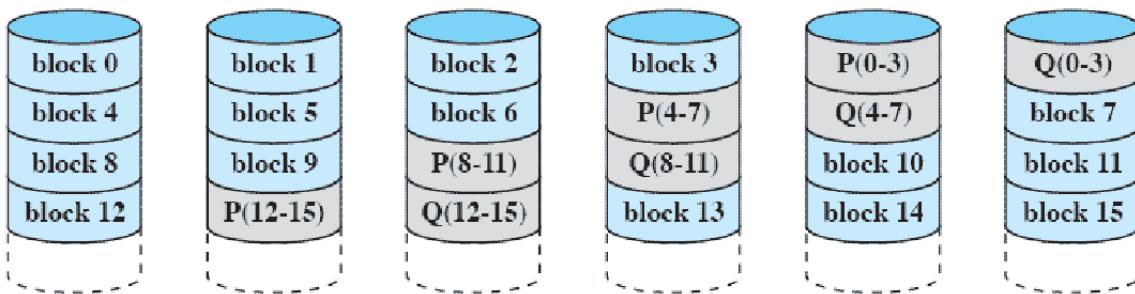
RAID4 按块存储可以保证单块的完整性，可以避免受到其他磁盘上同条带产生的不利影响。



(e) RAID 4 (block-level parity)



(f) RAID 5 (block-level distributed parity)



(g) RAID 6 (dual redundancy)

第11章：I/O管理和磁盘调度作业

1 列出并简单定义执行I/O的三种技术。

- 程序控制I/O：处理器代表进程向I/O模块发送一个I/O命令；然后进入忙等待，直到I/O操作完成。
- 中断驱动I/O：处理器代表进程向I/O模块发送一个I/O命令。若该I/O指令是非阻塞的，处理器继续执行发出I/O命令的进程的后续指令；若该I/O指令是阻塞的，处理器将当前进程设置为阻塞态并调度其它进程。
- 直接存储器访问（DMA）：DMA模块控制内存和I/O模块之间的数据交换。当需要传送一块数据时，处理器只需向DMA模块发请求，并且整个数据块传送结束后，才被中断。

第12章：文件管理

1 树型目录，文件共享

1.1 树型目录

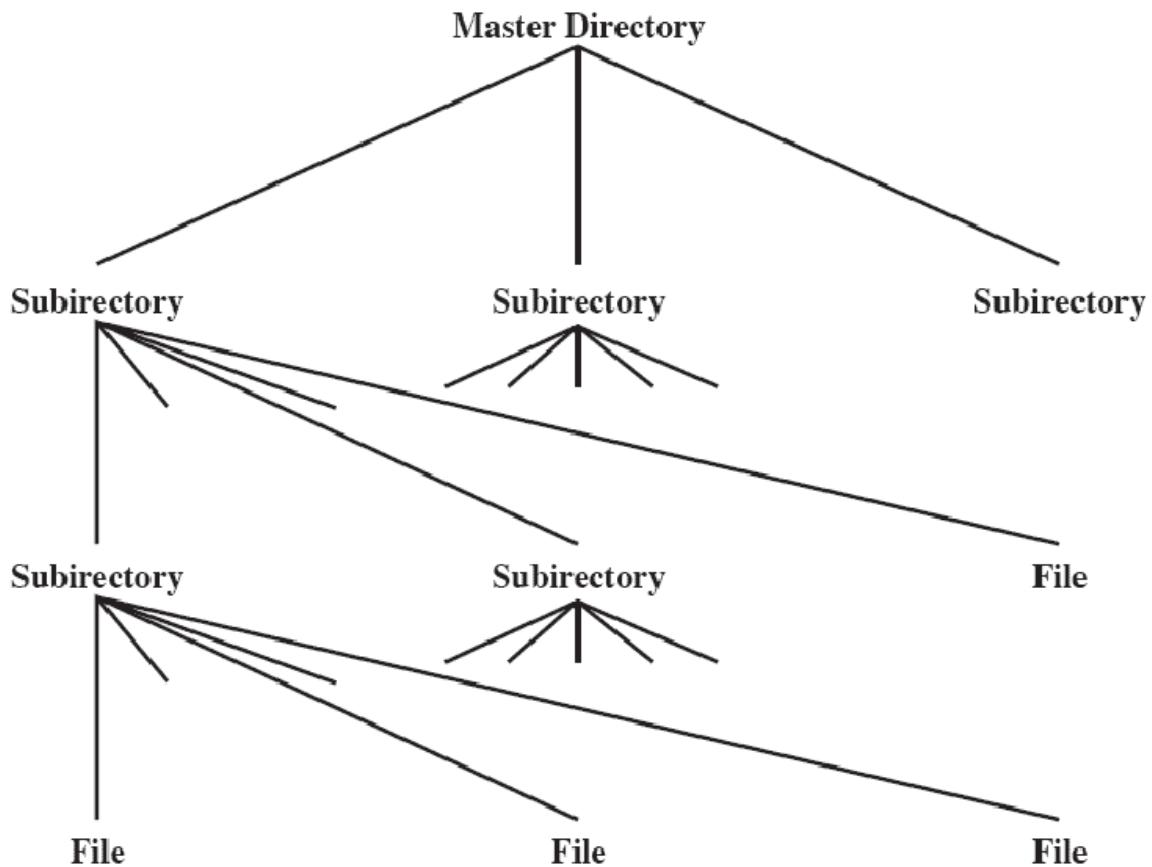


Figure 12.4 Tree-Structured Directory

1.2 文件共享

1.2.1 访问权限

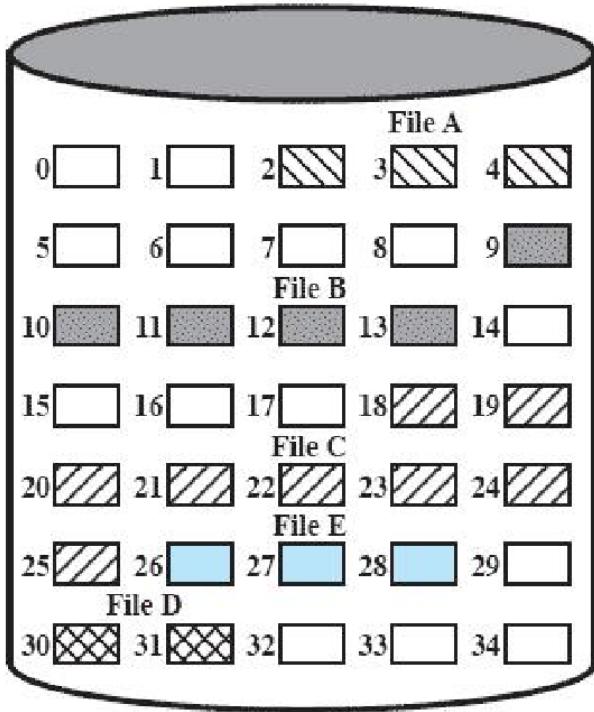
- 无
- 知道
- 执行
- 读
- 追加
- 更新
- 改变保护
- 删除

1.2.2 同时访问

- 加锁
- 互斥和死锁问题

2 三种文件分配方法：连续分配，链接分配，索引分配

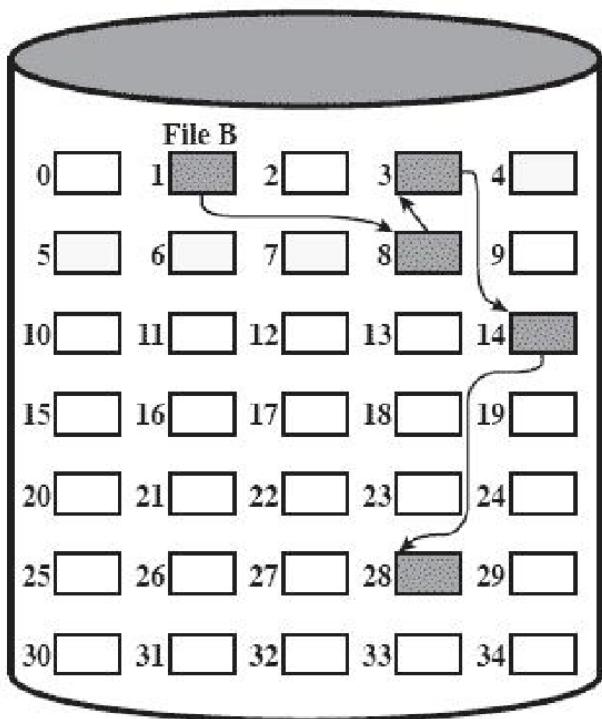
2.1 连续分配



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

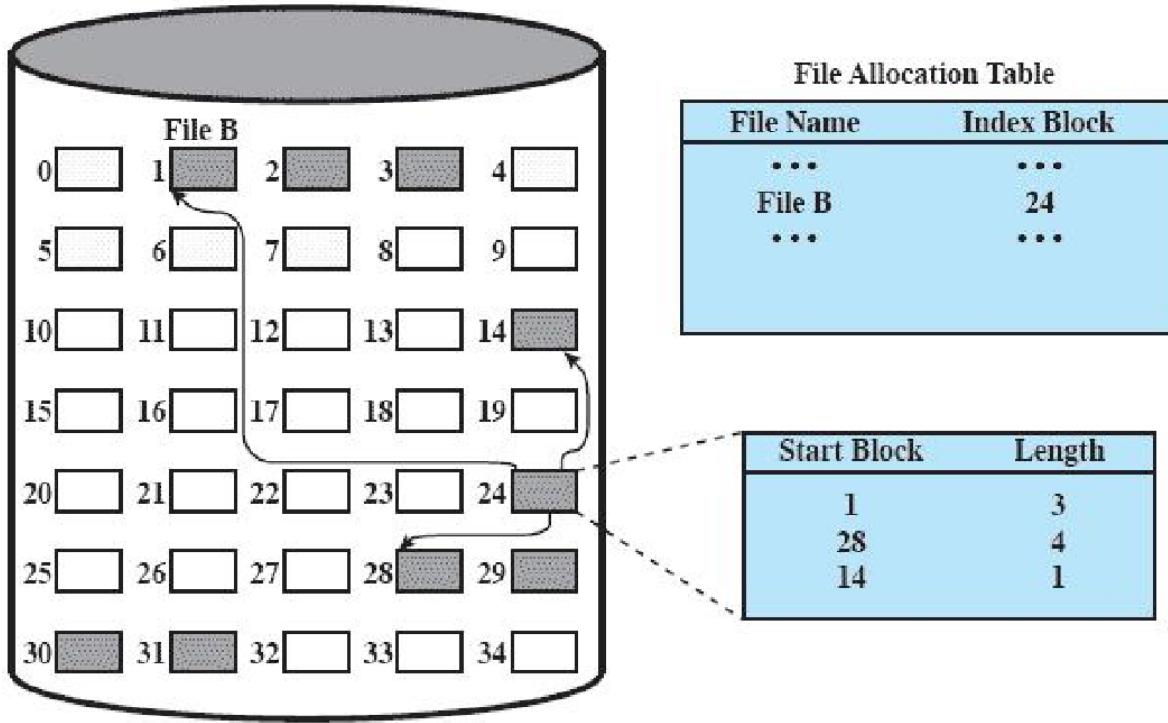
2.2 链接分配 (指针)



File Allocation Table

File Name	Start Block	Length
...
File B	1	5
...

7.3 索引分配 (索引表)



3 索引分配对文件尺寸的影响

例：如果块长 4KB（即索引块和数据块均长 4KB），每个指针 4B，则采用基于单个盘块的索引分配时，允许的文件最大尺寸是多少？

一个索引块可保存 $4\text{KB}/4\text{B} = 1\text{K}$ 个指针，每个指针指向一个数据块，文件最大尺寸为 $4\text{KB} \times 1\text{K} = 4\text{MB}$

4 磁盘空闲空间管理：位图

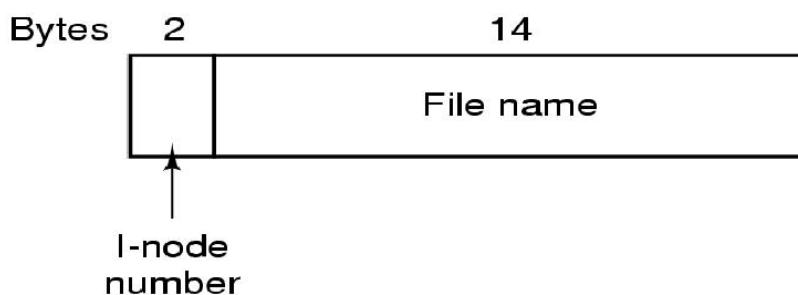
位图就是利用二进制的每一位来表示磁盘中一个块的使用情况，磁盘上所有的块都有一个二进制位与之对应。当值为0时，表示块空闲，值为1时，块已分配，形式如下：111111100111111110.....

Linux操作系统中就采用了位图的方式来管理空闲空间，不仅用于数据空闲块的管理，还用于inode空闲块的管理，因为inode也是存储在磁盘的。

空闲空间的管理

- 磁盘分配表
- 常用的空闲空间管理技术
 - 位表
 - 链接空闲区
 - 索引
 - 空闲块列表
- 卷
 - 一组在辅助存储上可寻址的扇区的集合，操作系统或应用程序用卷来进行数据存储。一个卷中的扇区在物理存储设备上不需要是连续的，只需要对操作系统或应用程序来讲是连续的。一个卷可能是更小的卷合并或组合后的结果。

5 UNIX 中的文件控制块：索引节点 i-node



目录文件的内容

文件名 14B	i-node # 2B
.....
mytest.c	56
.....

磁盘上的i-node表

i-node #	文件属性
56	存取权限，大小，文件主，建立/修改日期，磁盘地址等
.....

64B

例：设物理块大小为512B，某目录下有128个文件。

原来的FCB（文件控制块）占64B，则每物理块能容纳 $512/64=8$ 个FCB，则该目录文件需占 $128/8 = 16$ 块，查找一个文件的平均访盘次数为： $(1+16)/2 = 8.5$ 次。

采用i-node后：文件名部分有16B，i-node部分有64B，每物理块能容纳 $512/16=32$ 个文件名部分或 $512/64=8$ 个i-node，则该目录的文件名部分需占 $128/32 = 4$ 块，i-node部分需占 $128/8=16$ 块。查找一个文件的平均访盘次数为： $(1+4)/2 + 1 = 3.5$ 次。

第12章：文件管理作业

1 列出并简单定义三种文件分配方法

- 连续分配：在创建文件时，给文件分配一组连续的块。
- 链接分配：基于单个块，链中的每一块都包含指向下一块的指针。
- 索引分配：每个文件在文件分配表中有一个一级索引，分配给该文件的每个分区的索引中都有一个表项。

2 习题

当数据

很少修改并且以随机顺序频繁地访问时；

——索引

频繁地修改并且相对频繁地访问文件整体时；

——索引顺序

频繁地修改并以随机顺序频繁地访问时。

——散列或索引

从访问速度、存储空间的使用和易于更新（添加/删除/修改）这几方面考虑，为了达到最大效率，你将选择哪种文件组织？

错题整理1

1. 实时操作系统能及时处理由过程控制反馈的数据并响应。

2. 当 CPU 处于系统态（内核态）时，它可以执行计算机系统的**所有指令**。

3. 在“基址 B+限长 L”内存保护方案中，合法的逻辑地址 A 应该满足 $0 \leq A < L$ 。

4. 死锁预防的方法：

方法1：所有的进程在开始运行之前，必须**一次性地申请其在整个运行过程中所需要的全部资源**。

方法2：该方法是对第一种方法的改进，允许**进程只获得运行初期需要的资源，便开始运行，在运行过程中逐步释放掉分配到的已经使用完毕的资源，然后再去请求新的资源**。这样的话，资源的利用率会得到提高，也会减少进程的饥饿问题。

方法3：将系统中的所有资源顺序编号，**将紧缺的，稀少的采用较大的编号**，在申请资源时必须按照编号的顺序进行（**从小往大申请**），一个进程只有获得较小编号的进程才能申请较大编号的进程。

5. 死锁的避免：银行家算法

6. 进程在执行中发生缺页中断，经操作系统处理后，进程应执行**被中断的那一条指令**。

7. 在段式存储管理中，若逻辑地址的段内地址大于段表中该段的段长，则发生**越界中断**。

8. 在多核系统中，一般采用**FCFS**和**静态优先级**进程调度算法。

9. 将逻辑地址转换为内存物理地址的过程称为**地址映射(重定位)**。

10. 处理器工作状态分为两种模式。当 fork() 执行时 CPU 处于**系统态（内核态）**。

11. 为实现 CPU 和 I/O 设备的并行工作，操作系统引入了**中断硬件机制**。

12. 从**文件管理角度看**，文件由**文件控制块**和**文件体**两部分组成

13. 虚拟内存之所以有效，是因为程序运行时的**局部性原理**。

14.

PSW: Program Status Word 程序状态字

FCFS: First Come First Serve 先来先服务

PCB: Process Control Block 进程控制块

DMA: Direct Memory Access 直接存储器存取

MMU: Memory Management Unit 内存管理单元

15. 在分页虚拟存储管理系统中，什么情况下发生缺页中断？简述缺页中断的处理过程。

当 CPU 发出访问的逻辑地址的所在页还未调入内存时，发生缺页中断。

缺页中断的处理过程大致如下：

首先判断内存中是否有空闲？

如果没有则按照置换算法选择一个内存页淘汰，如果该页被修改过还需先写回磁盘，这样得到一个空闲帧。

然后按照页表所指明的该页磁盘地址把此页调入空闲帧，修改页表，重新执行刚才那条指令。

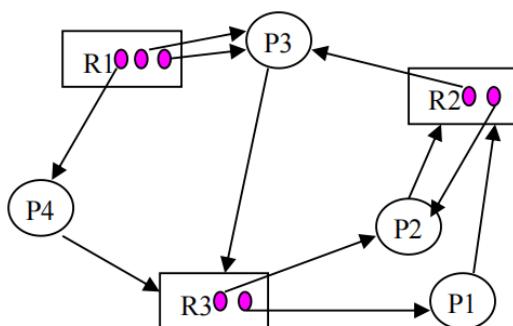
16. 简述可变分区存储管理中常用的 FF、BF、WF 分配算法的原理。

最先适应法(First Fit): 空闲区链表按起址递增顺序排列。分配时从链首开始查找，从第一个满足要求的空闲区中划分出作业需要的大小并分配，其余的部分作为一个新空闲区。

最佳适应法(Best Fit): 空闲区链表按区大小增顺序排列。分配时从链首开始查找第一个满足要求的空闲区就是满足要求的最小空闲区。

最坏适应法(Worst Fit): 空闲区链表按分区大小递减顺序排列。分配时从链首开始查找第一个空闲区不能满足要求时分配失败，否则从第一个空闲区中切出需要的大小分配。

8、系统资源分配图如下，请问现在是否已处于死锁状态，如果是，撤消哪个进程可以使系统代价最小地从死锁中恢复。



评分标准：每个问题 3 分。

答：已处于死锁状态。撤消 P1 代价最小，因为剥夺的资源最少。

11、若检测到 CPU 和磁盘利用率如下，请问现在可能发生了什么情况，应采取什么措施？

- 1) CPU 10%，磁盘 94%。
- 2) CPU 55%，磁盘 3%。

答：评分标准：每个 3 分。

- 1) CPU 10%，磁盘 94%：此时系统可能已经出现抖动，可暂停部分运行进程；**
- 2) CPU 55%，磁盘 3%：此时系统运行正常，磁盘利用率稍低，可增加进程数以提供资源利用率。**

错题整理2

- 1.采用用户级多线程策略时，操作系统内核调度的对象是**进程**。
- 2.**最短进程优先算法**是不可抢占式算法。
- 3.临界区是指**一段程序**。
- 4.磁盘空间碎片少且便于随机存取文件数据的文件分配方法是**索引分配**。
- 5.内存动态分区放置算法中，通常来说**最佳适配**性能最差。

下列选择中，不能使系统从死锁中恢复的措施是_____B_____。

- A. 杀死进程 B. 挂起进程 C. 抢占资源 D. 重启进程

- 6.动态分区分配算法会将内存分割成许多不连续的小分区，称之为**外部碎片**。
- 7.就计算密集型进程和IO密集型进程而言，**IO密集型进程**的CPU调度优先级更高。
- 8.多道批处理系统注重于**提高资源利用率**，而分时系统注重于**减少用户程序的响应时间**。
- 9.通常在PC机中，对磁盘的I/O控制采用**DMA或直接内存存取**的方式；对键盘的I/O控制采用**中断或中断驱动的I/O方式**。
- 10.

OPT: Optimal 最佳置换算法

RR: Round Robin 轮转

i-node: index node 索引节点

FF: First Fit 首次适配

FAT: File Allocation Table 文件分配表

PID: Process ID 进程标识符

11.当一个中断正在处理时，又发生新的中断，简述此时的两种处理方式。

处理多中断有两种方法。第一种方法是当正在处理一个中断时，禁止中断，顺序处理所发生的各个中断。

第二种方法是中断嵌套，定义中断优先级，允许高优先级中断打断低优先级中断的处理过程。

12.描述虚拟分页内存管理技术的基本原理及其优点。

内存被划分成许多大小相等的页框，进程被划分为许多大小与页框相等的页；

进程的页在需要访问时，装入内存中不一定连续的某些页框中。

优点：没有外部碎片，多道程序度更高，虚拟地址空间巨大。

13.简述死锁的四个必要条件。

互斥：涉及的是需互斥访问的临界资源。

占有且等待：进程申请资源而阻塞时，继续占有（不释放）已分配的资源。

不可抢占：进程已占用的资源未使用完前不可强行剥夺，只能由进程自愿释放。

循环等待：进程间形成一个等待资源的循环链。

14.简述进程的抢占式调度和非抢占式调度的区别。

非抢占：在这种情况下，一旦进程处于运行态，它就不断执行直到终止，或者为等待 I/O 或请求某些操作系统服务而阻塞自己。

抢占：当前正在运行的进程可能被操作系统中断，并转移到就绪态。

是否抢占当前进程的决策可能发生在新进程到达时，或者在一个中断发生后把一个被阻塞的进程置为就绪态

时，或者基于周期性的时间中断。

15.不死锁的哲学家进餐问题。

五支叉子放在五位哲学家之间，进餐前哲学家必须先拿起左边叉子，再拿起右边叉子，吃完后放下两支叉子。为防

止死锁，只允许四位哲学家同时拿起一支或两支叉子。用信号量方法实现此不死锁的哲学家进餐问题。

```
semaphore fork[5] = {1}; //每把叉子就是一个临界资源。
semaphore room = 4; //至多允许四人同时拿起叉子
int i;
void philosopher (int i)
{
    while (true) {
        think();

        wait (room);

        wait (fork[i]);
        wait (fork [(i+1) mod 5]);
        eat();
        signal (fork [(i+1) mod 5]);
        signal (fork[i]);

        signal (room);
    }
}
```

操作系统常用缩写总结

一、计算机系统概述

- MAR: memory address register 内存地址寄存器 (IO AR, IO BR同理)
- MBR: Memory buffer register 内存缓冲寄存器
- PC: program counter 程序计数器
- IR: instruction register 指令寄存器
- PSW: program status word 程序状态字
- SMP:对称多处理器
- RAM: random access memory 随机存储器
- ISR: Interrupt Service Routines 中断服务例程
- MMU: memory manage unit 存储管理部件

二、操作系统概述

- ISA: instruction system architecture 指令系统体系结构
- ABI: application binary interface 应用程序二进制接口
- API: application programming interface 应用程序编程接口
- DMA: direct memory access 直接内存访问

三、进程描述和控制

- PCB: process control block 进程控制块
- CPL: current privilege level 当前特权特权
- PSR: process status register 处理器状态寄存器
- IRT: interrupt return 中断返回(中断服务程序的最后一条指令,ret)

四、线程、对称多处理和微内核

- LWP: light weight process 轻量级进程，也称线程
- TCB: thread control block 线程控制块
- ULT: user-level thread 用户级线程
- KLT: kernel-level thread 内核级线程

五、并发性：互斥和同步

六、并发性：死锁和饥饿

七、内存管理

- DLL: dynamic link library 动态链接库

八、虚拟内存

- PTE: page table entry 页表项
- TLB: translation lookaside buffer 转换检测缓冲区
- OPT: optimal 最佳 (置换策略相关)
- LRU: least recently used 最近最少使用
- FIFO: first in first out 先进先出
- Clock: 时钟

九、单处理器调度

- FCFS: first come first service 先来先服务 (调度策略相关)
- RR: Round-Robind 轮转算法
- SPN: shortest process next 最短进程优先
- : shortest remaining time 最短剩余时间
- HRRN: Highest Response Ratio Next 最高响应比优先
- VRR: virtual round robin 虚拟轮转法

十、多处理器和实时调度

十一、I/O管理和磁盘调度

- SSTF: shortest seek time first 最短服务时间优先
- PRI: priority 进程优先级
- RAID: redundant array of inexpensive disk 廉价磁盘冗余阵列 (I: 也有independent独立的意思)

十二、文件管理

- FAT: file allocation table 文件分配表
- DAT: disk allocation table 磁盘分配表
- VFS: virtual file system 虚拟文件系统
- NTFS: New Technology File System 新文件系统