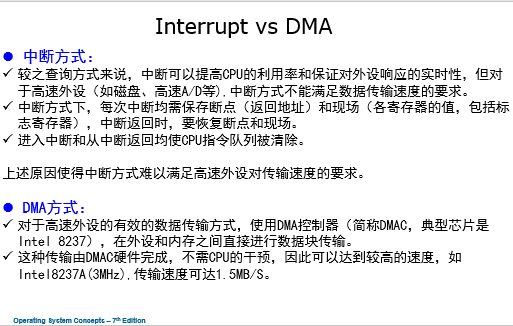
http://ishare.iask.sina.com.cn/f/23179635.html

第一章

* 操作系统作用：控制和协调各用户的应用程序对硬件的使用
* 操作系统的基本特征：并发、资源共享
* 中断：
  + 在计算机执行期间，系统内发生任何非寻常的或非预期的急需处理事件使得CPU暂时中断当前正在执行的程序，而转去执行相应的时间处理程序。待处理完毕后又返回原来被中断处继续执行或调度新的进程执行的过程。
  + 外部中断、内部中断
* 直接内存访问：
  + Direct Memory Access (DMA)
  + 允许外围设备和主内存之间直接传输它们的I/O数据，而不需要系统处理器的参与。
* 中断与DMA的区别



* 多CPU、多核：多CPU是指处理器核心是N个，其他例如缓存等配置也都是N份的；多核是指单个芯片上包含任意多个CPU的处理器
* 多道程序系统
  + 分时系统：在分时系统中，虽然CPU还是通过在作业之间的切换来执行多个作业，但是由于切换频率很高，用户可以在程序运行期间与之进行交互。分时操作系统允许许多用户同时共享计算机。由于分时系统的每个动作或命令都较短，因而每个用户只要少量CPU时间。随着系统从一个用户快速切换到另一个用户，每个用户会感到整个系统只为自己所用，尽管它事实上为许多用户所共享。每个用户在内存中至少有一个程序。装入到内存并执行的程序通常称为进程（process）。
* 操作系统双重模式
  + 用户模式
  + 内核模式

系统引导时，硬件开始处于内核模式。接着，装入操作系统，开始在用户模式下执行用户进程。一旦出现陷阱或中断，硬件会从用户模式切换到内核模式（即将模式设为0）。因此，只要操作系统获得了对计算机的控制，他就处于内核模式。系统在将控制交还给用户程序时会切换到用户模式（将模式设为1）。

第二章

* 对于用户来说 ⇒ 操作系统是个“黑盒子”
* 要打开这个盒子 ⇒ 先得找到“盒子的入口” ⇒ 系统接口（API）
* 接口: 连接两个设备并转换数据 ⇒ 系统接口连接用户和OS
* 学习转换 ⇒ 转换之前:用户如何使用计算机?
* 命令、GUI、Apps ⇒ 都是应用OS提供的函数接口编程序
* OS提供的函数 ⇒ 系统调用 ⇒ 系统调用有哪些?怎么做?
* 系统调用有哪些? ⇒ POSIX
* 操作系统提供给用户的界面：GUI, 图形用户界面；批处理；命令行
* 系统调用：提供了操作系统提供的有效服务界面；操作系统为用户提供的一些借口，这些接口提供了对系统硬件功能的操作。
  + 与库函数的区别：（1）库函数调用函数库中的一段程序，系统调用是操作系统的一个入口点；（2）库函数在用户地址空间执行，系统调用在内核地址空间执行；（3）库函数的运行时间属于“用户时间”，系统调用的运行时间属于“系统时间”；（4）库函数属于过程调用开销小，系统调用需要在用户空间和内核上下文环境间切换，开销较大。
* 系统调用类型：进程控制、文件管理、设备管理、信息维护和通信。
* 微内核VS宏内核

宏内核：国务院，组织机构很庞大，有好多人在管事

将内核从整体上作为一个大过程来实现，所有的内核服务都在一个地址空间运行，相互将内核从整体上作为一个大过程来实现，所有的内核服务都在一个地址空间运行，相互之间直接调用函数，高效简单。

有Unix，Linux

微内核：皇帝，所有权力集中在他手中，但办具体的事情要找别人

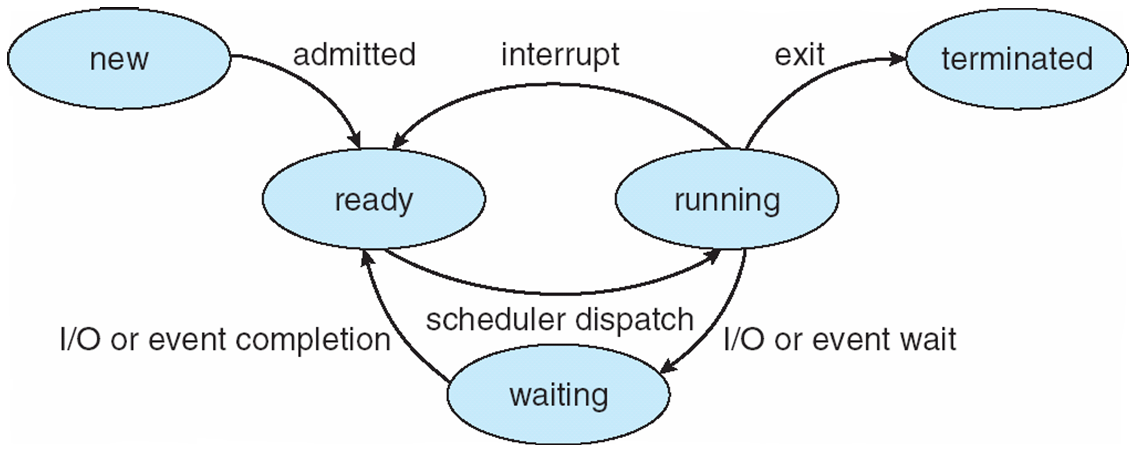
功能被划分为单独的过程，过程间通过消息传递进行通信，模块化程度高，一个服务失效不会影响另外一个服务。

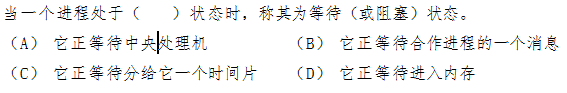
有Windows，minix，Mac

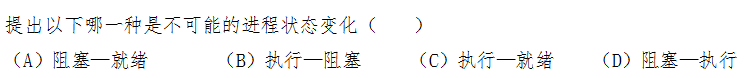
* 微内核：将所有非基本部分从内核中移走，并将它们实现为系统程序或用户程序。微内核通常包括最小的进程和内存管理以及通信功能。微内核的主要功能是使客户程序和运行在用户空间的各种服务进行通信。通信以消息传递形式提供。
* 模块化结构：面向对象编程技术生成模块化内核。这里，内核有一组核心部件，以及在启动或运行时对附加服务的动态链接。这种方法使用动态加载模块，并在现代的UNIX，如Solaris、Linux和Max OSX中很常见。

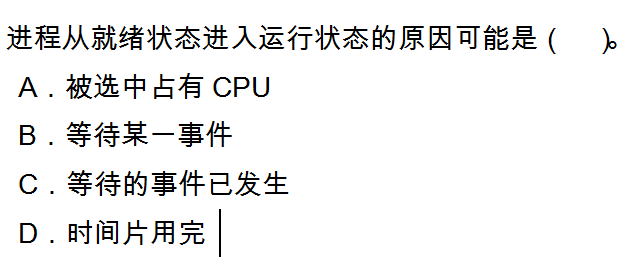
第三章

* **计算机解决问题** ⇒ **执行程序**
* **执行中的程序和静止程序存在很大区别** ⇒ **引出进程**
* **CPU太快** ⇒ **引出并发** ⇒ **进一步深化了进程**
* **进程走走停停** ⇒ **状态转化** ⇒ **现场切换** ⇒ **进程调度**
* **进程相互干扰** ⇒ **进程保护** ⇒ **地址空间** ⇒ **线程**
* **进程相互协作** ⇒ **进程同步** ⇒ **进程通信消息**
* **用户希望操纵进程** ⇒ **进程创建等**
* 进程定义
  + 运行中的程序
  + 资源分配的最小单位
  + CPU调度的一个单位
* 进程状态图



B

D



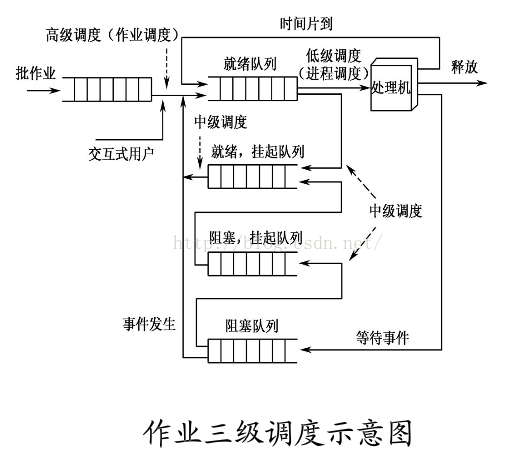
D

进程与线程

* 进程是具有独立功能的程序关于某个数据集合上的一次运行活动，是系统进行资源分配和调度的独立单位。
* 线程有时称轻量级进程，进程中的一个运行实体，是一个CPU调度单位。
* 进程和线程的不同之处可从以下四个方面比较：
  + (1) 调度：线程作为调度的基本单位，同进程中线程切换不引起进程，当不同进程的线程切换才引起进程切换；进程作为拥有资源的基本单位。
  + (2) 并发性：一个进程间的多个线程可并发。
  + (3) 拥有资源：线程仅拥有隶属进程的资源；进程是拥有资源的独立单位。
  + (4) 系统开销：进程大；线程小。
* 进程控制块 PCB（process control block）进程信息的仓库，这些信息在进程与进程之间是不同的
* 进程切换
* 长期调度、中期调度、短期调度
* (1)长期调度，又称为作业调度或高级调度，这种调度将已进入系统并处于后备状态的作业按算法选择一个或一批，为其建立进程，并进入主机，当该作业执行完毕时，还负责回收系统资源，在批处理系统中，需要有作业调度的过程，以便将它们分批地装入内存，在分时系统和实时系统中，通常不需要长期调度。它的频率比较低，主要用来控制内存进程的数量。（从缓冲期到内存）
* (2)中期调度，又称为交换调度。它的核心思想是能将进程从内存或从CPU竞争中移出，从而降低多道程序设计的程度，之后进程能被重新调入内存，并从中断处继续执行，这种交换的操作可以调整进程在内存中的存在数量和时机。其主要任务是按照给定的原则和策略，将处于外存交换区中的就绪状态或等待状态的进程调入内存，或把处于内存就绪状态或内存等待状态的进程交换到外存交换区。（从内存移到磁盘）
* (3)短期调度，又称为进程调度、低级调度或微观调度。这也是通常所说的调度，一般情况下使用最多的就是短期调度。它的主要任务是按照某种策略和算法将处理机分配给一个处于就绪状态的进程，分为抢占式和非抢占式。（就绪队列选中到CPU）
* 三级调度的区别在于频率不同，长期调度频率较低，短期调度频率较高。

进程调度

* 在许多进程或线程都准备使用CPU进行任务处理时，就会存在资源竞争和分配的问题。一般都会将进程或线程先放在一个缓冲池中，等待合适的时机调度程序从中选择一个进程或线程进行交给CPU进行处理。



* 进程创建
  + fork函数：十分重要
  + exec函数
* 进程通信
  + 两种模型

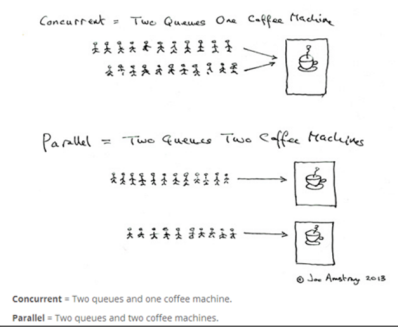
直接通信：需要通信的每个进程必须明确的命名通信的接受者和发送者。（每对进程之间只有一个线路）

间接通信：通过邮箱或端口发送和接收消息。（一个进程可能通过许多不同的邮箱与其他进程通信，但两个进程仅在其共享至少一个邮箱时可互相通信。）

* 并发与并行

并发：指多个进程实体，同存于内存中，能在一段时间内同时运行，并发性是进程的重要特征，同时也是操作系统的重要特征。引入进程的目的就是为了使程序能与其他进程的程序并发执行，以提高资源利用率。

并行：

（上面：并发；下面：并行）

第四章：线程

* **进程 = 地址空间 + 指令执行序列**
* **一个地址空间+多个指令执行序列** ⇒ **引出线程**
* **线程具有并发的优点，却比进程的代价低得多**
* **线程在同一地址空间中** ⇒ **线程库可以用户级实现**
* **用户级线程，核心级线程，两者都有**
* **各类线程的实现细节，其中上下文切换是核心**
* 线程定义（先有操作系统，再有线程）一种CPU利用的基本单元，它是形成多线程计算机的基础。
* 多线程优势：能充分使用多处理器体系结构，以便每个进程并行运行在不同的处理器上，不管有多少CPU，单线程进程只能运行在一个CPU上。在多CPU使用多线程加强了并发功能。
* 多线程模型

多（user thread）对一（kernal thread）模型

一对一模型

多对多模型

（一般线程数要多于进程）

第五章：CPU调度

* 并发能提高效率 ⇒ 并发的核心是进程能让出CPU
* 进程让出CPU ⇒ 下一个进程使用CPU ⇒ 这个选择就是调度
* 进程、线程(内核级、用户级)都能调度 ⇒ 任务调度
* 调度任务分类: 交互式，批处理
* 调度时机分类: 抢占式、非抢占式
* CPU调度算法: FCFS, SJF, Priority(批处理); RR(交互式)
* CPU调度算法: 多级队列, 多级反馈队列(混合)

— 掌握每种算法

— 各自特点

— 周转时间（结束时间-到达时间）、等待时间（开始时间-到达时间）、响应时间

* FCFS（先到先服务）公平，短作业等待时间长
* SJF（最短作业优先）SJF算法的平均等待时间最小
  + SRTF（最短剩余时间优先）
* Priority（按照优先级调度，只有优先级高的为空时才调度其他任务）
* RR（没学过）

第六章：进程同步

* **并发** ⇒ **多个进程同时存在** ⇒ **相互影响**
* **非原子操作共享变量** ⇒ **出现语义错误** ⇒ **竞争条件**
* **竞争条件** ⇒ **临界区** ⇒ **互斥** ⇒ **临界区进入方法**
* **复杂的peterson算法**⇒**强硬的关中断**⇒**硬件支持的TestAndSet**
* **都不适合用户实现** ⇒ **封装成锁**
* **一般的锁会盲等** ⇒ **引入睡眠** ⇒ **将锁一般化为信号量**
* **所有一切都是为了使用户更容易、使系统更好用(不出错)**
* 进程的关系：同步、互斥

进程互斥：是指某一资源同时只允许一个访问者对其进行访问，具有唯一性和排它性。但互斥无法限制访问者对资源的访问顺序，即访问是无序的。

进程同步：是指在互斥的基础上（大多数情况），通过其它

机制实现访问者对资源的有序访问。在大多数情况下，同步已经实现了互斥，特别是所有写入资源的情况必定是互斥的。少数情况是指可以允许多个访问者同时访问资源。

* 并发进程

在一个进程的工作没有完成之前，另一个进程就可以开始工作，这些进程就称为可同时执行的。或者称他们具有并发性，并且把可同时执行的进程称为并发进程。

并发进程的关系：进程互斥、进程同步

* 竞争条件：
  + 和时间有关的共享数据语义错误
  + 多个进程并发访问和操作同一数据且执行结果与访问发生的特定顺序有关
* 临界资源与临界区 critical section
  + 临界资源：一次仅允许一个进程使用的共享资源
  + 临界区：不论是硬件临界资源，还是软件临界资源，多个进程必须互斥地对它进行访问。临界区是每个进程中访问临界资源的那段程序。
  + 临界区访问原则：每次只允许一个进程进入临界区，进入后不允许其他进程进入。

解决临界区问题的三个条件

**1. Mutual Exclusion (互斥)**- If process ***Pi*** is executing in its critical section, then no other processes can be executing in their critical sections

**2. Progress (前进)** - If no process is executing in its critical section and some processes that wish to enter their critical sections, then only those processes that are not executing in their reminder sections can particulate in the decision on which will enter its critical section next, and this selection cannot be postponed indefinitely

**3. Bounded Waiting (有限等待)** - A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted

解决临界区问题的方法

* Software: Peterson’s Solution（适用于两个进程在临界区和剩余区间交替执行，该算法需要在两个进程之间共享数据项；只解决互斥问题，不解决同步问题）
* Hardware: Synchronization Hardware（硬件同步）
  + 原子指令（锁）
* Semaphores: PV 信号量

它们的本质就是：上锁 – 进入临界区 – 解锁

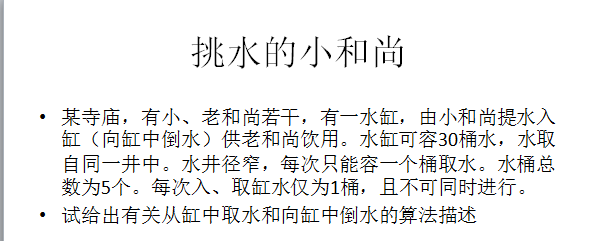
还记得我们银行家算法的那个实验吗？

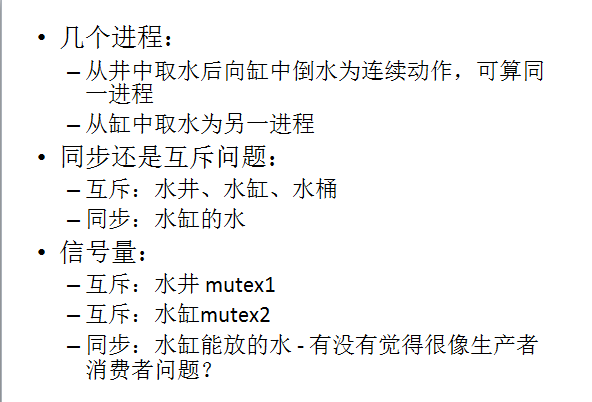
里面有一个mutex的互斥

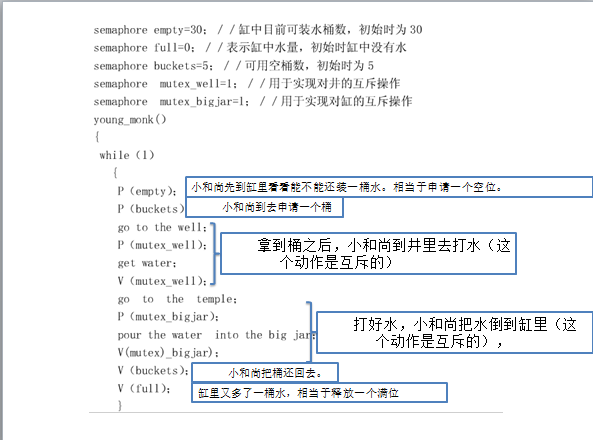
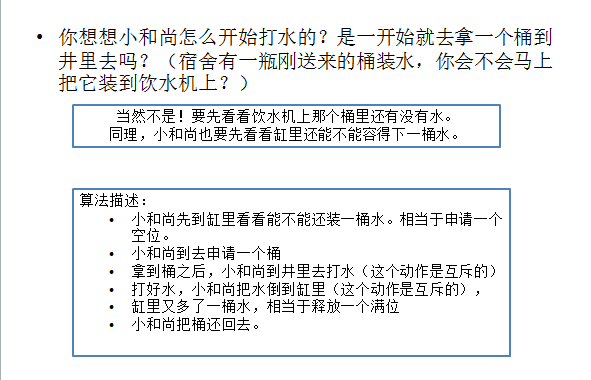
这个就是典型的多线程对同一变量操作的访问问题。

PV Semaphore

* **如何用PV原语解决问题？**
  + 观察一件案例，首先判断属于：进程同步？互斥？两者都有的混合问题？
    - 互斥：是指某一资源同时只允许一个访问者对其进行访问，具有唯一性和排它性。但互斥无法限制访问者对资源的访问顺序，即访问是无序的。
    - 同步：是指在互斥的基础上（大多数情况），通过其它机制实现访问者对资源的有序访问。在大多数情况下，同步已经实现了互斥，特别是所有写入资源的情况必定是互斥的。少数情况是指可以允许多个访问者同时访问资源。
  + 确定信号量，信号量可能有多个：有几个等待
    - 互斥：判断进程间是否互斥,关键是看进程间是否共享某一公有资源,一个公有资源与一个信号量相对应
    - 同步：进程同步时的信号量只与制约进程及被制约进程有关而不是与整组并发进程有关,所以称该信号量为私有信号量
  + 确定信号量的值
    - 可用资源实体数
  + 用PV原语实现同步或互斥



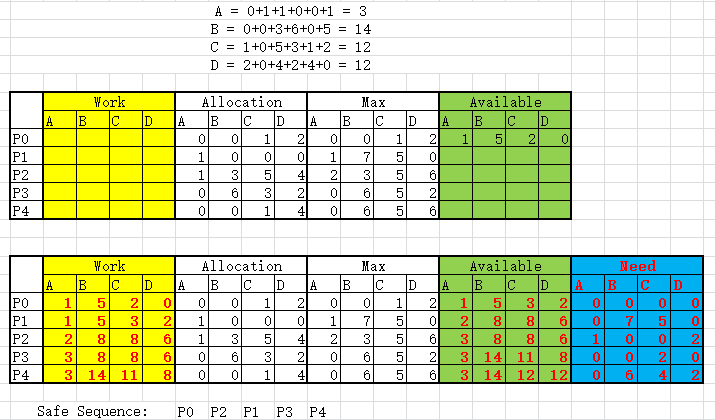
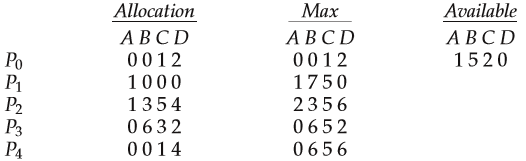




第七章：死锁

* **进程竞争资源** ⇒ **有可能形成循环竞争** ⇒ **死锁**
* **死锁需要处理** ⇒ **死锁分析** ⇒ **死锁的必要条件**
* **死锁处理** ⇒ **预防、避免、检测+恢复、忽略**
* **死锁预防: 破除必要条件** ⇒ **引入了不合理因素**
* **死锁避免: 用银行家算法找安全序列** ⇒ **效率太低**
* **死锁检测恢复: 银行家算法找死锁进程组并恢复** ⇒ **实现较难**
* **死锁忽略: 就好像没有死锁** ⇒ **现在用的最多**
* 什么是死锁？
  + 多个进程因循环等待资源而造成无法执行的现象
* 发生死锁的必要条件
  + **Mutual exclusion（互斥）:** only one process at a time can use a resource
  + **Hold and wait**(占有并等待)**:** a process holding at least one resource is waiting to acquire additional resources held by other processes占有资源，再去申请
  + **No preemption（不可抢占）:** a resource can be released only voluntarily by the process holding it, after that process has completed its task只能自愿放弃
  + **Circular wait** (循环等待) **:** there exists a set {*P*0, *P*1, …, *P*n} of waiting processes such that *P*0 is waiting for a resource that is held by *P*1, *P*1 is waiting for a resource that is held by *P*2, …, *Pn*–1 is waiting for a resource that is held by *P*n, and *P*n is waiting for a resource that is held by *P*0.在资源分配图里存在一个环路
* 资源分配图：有环、无环

银行家算法



Allocation=Allocation+request

Available=Available-request

Need=Need-request

初始：Work =available

Work= work+allocation

第八章：内存

* **内存的根本目的** ⇒ **把程序放在内存并让其执行**
* **程序执行需要重定位** ⇒ **编译、载入和运行三种定位时刻**
* **运行时重定位最成熟** ⇒ **从逻辑地址到物理地址的翻译**
* **内存如何管理** ⇒ **连续内存分配(分区)最直观**
* **程序由若干段组成** ⇒ **以段为单位的内存分区策略** ⇒ **分段**
* **分段对程序员自然，但会造成内存碎片**⇒ **分页** ⇒ **段页结合**
* 重定位bind
  + 什么是重定位：为执行程序而对其中出现的地址所做的修改（地址绑定）
  + 重定位的时机

编译时：如果编译时就知道进程将在内存中的驻留地址，那么就可以生成绝对代码。

加载时：如果在编译时并不知道进程将驻留在内存的什么地方，那么编译器就必须生成可重定位代码。

执行时：如果进程在执行时可以从一个内存移动另一个内存，那么绑定必须延迟到执行时才进行。

* 动态链接、静态链接

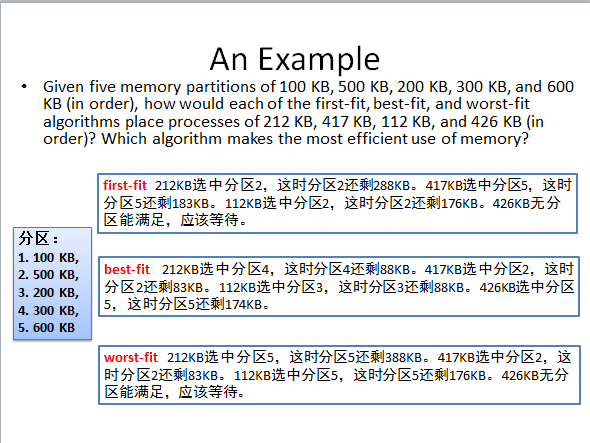
动态链接：将链接延迟到运行时，也可用于库更新

静态链接

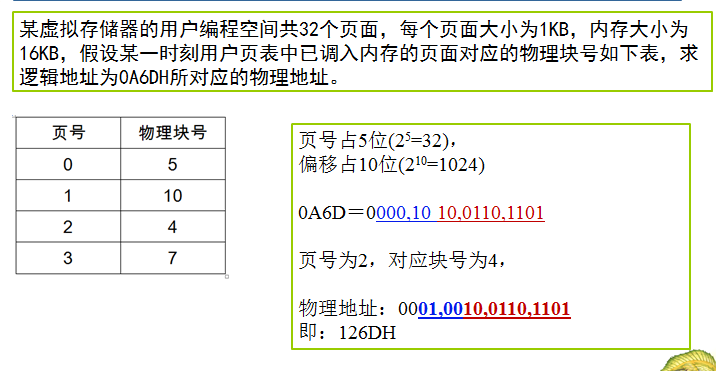
C:\Users\win8\Documents\Tencent Files\1569943279\Image\C2C\]FVURM6XMD9]6Q9`ORKF3YV.png

内存分配方案

* 最简单最直接：
  + 连续分配：动态内存分配
  + 连续分配：分段
  + 典型：首次适应、最佳适应、最差适应
  + 问题：碎片->紧缩



* 不连续分配：
  + 分页：
    - 逻辑地址转物理地址，反之也应该会计算
  + 段页结合
    - 方法



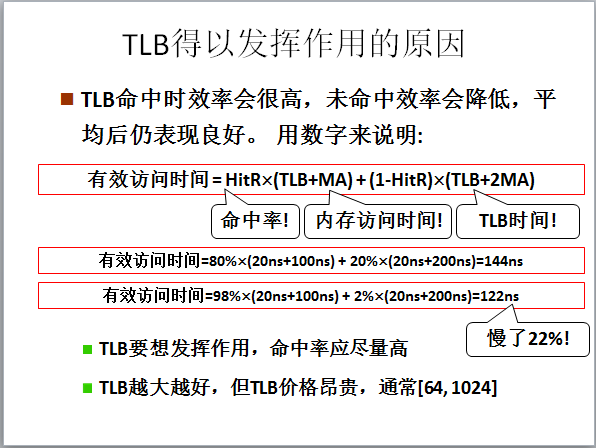
页表很大怎么办？

* Hierarchical Paging (层次页表/多级页表)
  + **地址翻译效率很低，要提高效率**
  + **TLB(Translation Look-aside Buffer)是一组相联快速内存：快表**

Hashed Page Tables (哈希页表)

Inverted Page Tables (反向页表)

快表访问



段页结合

优点：符合程序员习惯，并可高效利用内存

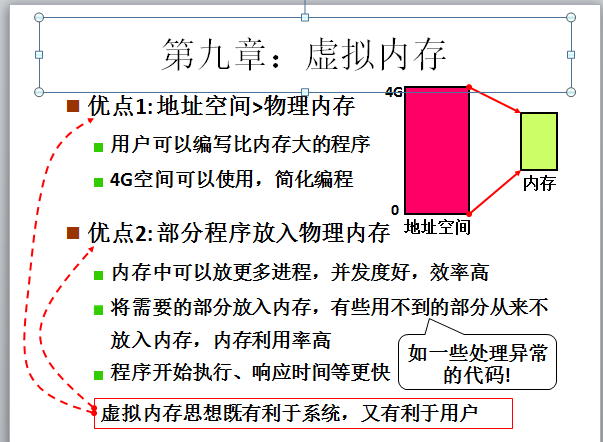
缺点：复杂，访问一次地址需要查表好多次

第九章：虚拟内存

* **内存的根本目的** ⇒ **把程序放在内存并让其执行**
* **只要将部分程序放进内存即可执行** ⇒ **内存利用率高**
* **可编写比内存大的程序** ⇒ **使用一个大地址空间(虚拟内存)**
* **部分程序在内存** ⇒ **其他部分在磁盘** ⇒ **需要的时候调入内存**
* **页表项存在P位** ⇒ **缺页产生中断** ⇒ **中断处理完成页面调入**
* **调入页面需要一个空闲页框** ⇒ **如果没有空闲页框** ⇒ **置换**
* **置换方法** ⇒ **FIFO**🡪**MIN**🡪**LRU**🡪**Clock**

优点：1.地址空间>物理内存；2.部分程序放入物理内存

虚拟内存思想既有利于系统，又有利于用户



如何选一个空闲页框? **页面淘汰(置换)**

FIFO，先进先淘汰

OPT，（往后数）离我最远的先淘汰

LRU，选最近一段时间内没有使用的