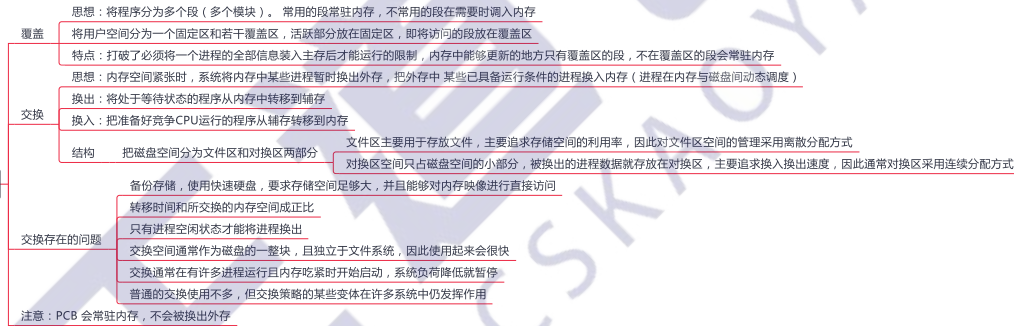


### 3.1 内存管理概念（上）

#### 内存管理的基本原理和要求



#### 覆盖与交换



#### 连续分配管理方式



### 3.1 内存管理概念（中）

#### 非连续分配管理方式

允许一个程序分散的装入不相邻的内存分区

设计思想

将主存空间划分为大小相等且固定的块，块相对较小，作为主存的基本单位，进程以块为单位尽心空间申请。  
分页存储与固定分区技术很像，但是其分页相对于分区又很小，分页管理不会产生外部碎片，产生的内部碎片也非常的小

分页存储的基本概念

页面和页面大小  
进程中的块 = 页  
内存中的块 = 页框（页帧）  
进程申请主存空间，为每个页面分配主存中可用页框，即页与页框一一对应

页面大小要适中

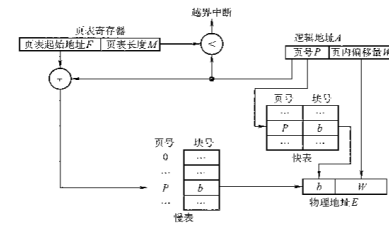
页面太小：进程页面数过多，页表过程，增加内存占用，降低硬件地址转换效率

页面太大：页内碎片过多，降低内存利用率

地址结构  
页号（有多少页的编号）+ 页内偏移（页内存了多少东西）

页表  
为了便于在内存中找到进程的每个页面对应的物理块，系统为每个进程建立一张页表，记录页面在内存中对应的物理块号，页表一般放在内存中

页表项：页号 + 物理内存中的块号（不要与地址结构搞混）  
页表项的物理内存块号 + 地址结构中的页内偏移 = 物理地址



基本分页存储管理方式

基本地址变换机构

计算方式  
页号  $P = A/L$ ，页内偏移量  $W = A \% L$   
比较页号P和页表长度M，若  $P > M$  产生越界中断  
页表中页号P对应的页表项地址 = 页表起始地址F + 页号P \* 页表项长度  
取出该页表项内容b  
计算  $E = b * L + W$  使用E去访问内存

页表项大小的设计应当尽量一页正好能装下所有的页表项

地址变换过程必须足够快，否则访存速率会降低

分页管理存在的问题  
页表不能太大，否则会降低内存利用率

组成  
设置一个页表寄存器（PTR），存放页表在内存中的起始地址F和页表长度M  
页表的起始地址和页表长度放在进程控制块（PCB）中

可优化方向：如果页表放在内存中，取地址访问一次内存，按照地址取出数据访问一次内存，共需要两次访问内存

优化：地址变换机构中增加一个具有并行查找能力的高速缓冲寄存器（快表），又称为相联存储器（TLB）  
相联存储器既可以按照地址查找也可以按照内容查找

基本地址变换机构  
两次访存

访问一个逻辑地址的访存次数  
具有快表的地址变换机构  
快表命中，只需一次访存  
快表未命中，需要两次访存

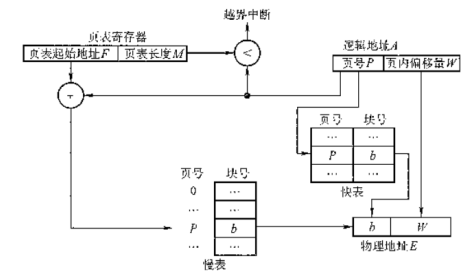
具有快表的地址变换机构

变换过程  
CPU给出逻辑地址后，先查询快表中是否命中

若快表命中，直接从快表中该页对应的页框号，与页内偏移量拼接成物理地址

若快表未命中，再按照正常方式从页表中查询相应页表项，并将该页表项存入快表中（按照一定策略）

查找图示



两级页表

如果页数过多，就会导致页表也过多，那么我们可以考虑设置一个用来储存页表的页表（套娃）

逻辑地址空间格式 = 一级页号 + 二级页号 + 页内偏移

设计多级页表的时候，最后一定要保证顶级页表一定只有一个

建立多级页表的目的在于建立索引，不必浪费主存空间去储存无用的页表项，也不用盲目式的查询页表项

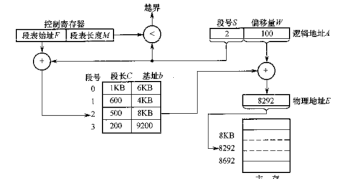
### 3.1 内存管理概念（下）

#### 基本分段存储管理方式

- 出发点
  - 分页是从计算机角度考虑设计的，目的是为了内存的利用率，提高计算机性能，分页通过硬件机制实现，对用户完全透明
  - 分段是从用户和程序员的角度提出，满足方便编程，信息保护和共享，动态增长及动态链接等多方面的需要
- 分段
  - 按照用户进程中的自然段划分逻辑空间
  - 地址结构 = 段号S + 段内偏移量W
- 段表
  - 页式系统中，页号和页内偏移对用户透明      段式系统中 段号和段内偏移量必须由用户显示的提供
  - 每个进程都有一张逻辑空间与内存空间映射的段表，这个段表项对应进程的一段，段表项记录该段在内存中的起始地址和长度
  - 段表内容 = 段号 + 段长 + 本段在内存中的地址

- 地址变换机构
  - 逻辑地址A中取出段号S和段内偏移量W
  - 比较段号S和段表长度M，若S>=M,则产生越界中断，否则继续执行
  - 段号S对应的段表项地址 = 段表起始地址F+段号S\*段表项长度，从该段表项中取出段长C，比较段内偏移量与C的大小判断是否出现越界
  - 取出段表项中该段的起始地址b,计算E = b+W，用得到的物理地址E去访问内存
- 段的共享与保护
  - 共享：两个作业的段表中响应表项指向被共享段的同一个物理副本来实现的      纯代码或者可重入代码以及不可修改的数据可以被共享
  - 保护机制
    - 存取控制保护
    - 地址越界保护

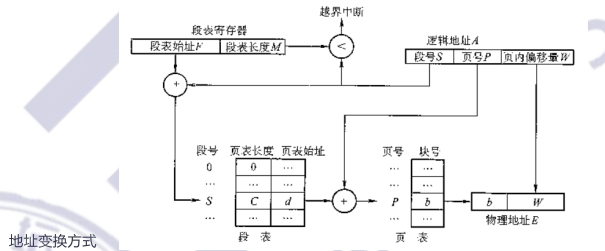
变换过程



页式存储有效的提高内存利用率，分段存储能反映程序的逻辑结构并有利于段的分享，将这两种方式结合一下      这种二者结合的方法经常在计算机理论中遇到

#### 段页式管理方式

- 思想
  - 作业的地址空间首先被分成若干逻辑段，每段有自己的段号
  - 每个段分成若干大小固定的页
  - 对内存空间的管理仍然和分页存储管理一样
- 地址结构      段号S+页号P+页内偏移量W
- 为了实现地址变换，系统为每个进程建立了一张段表，每个分段有一个页表      一个进程中，段表只能有一个，页表可以有多个

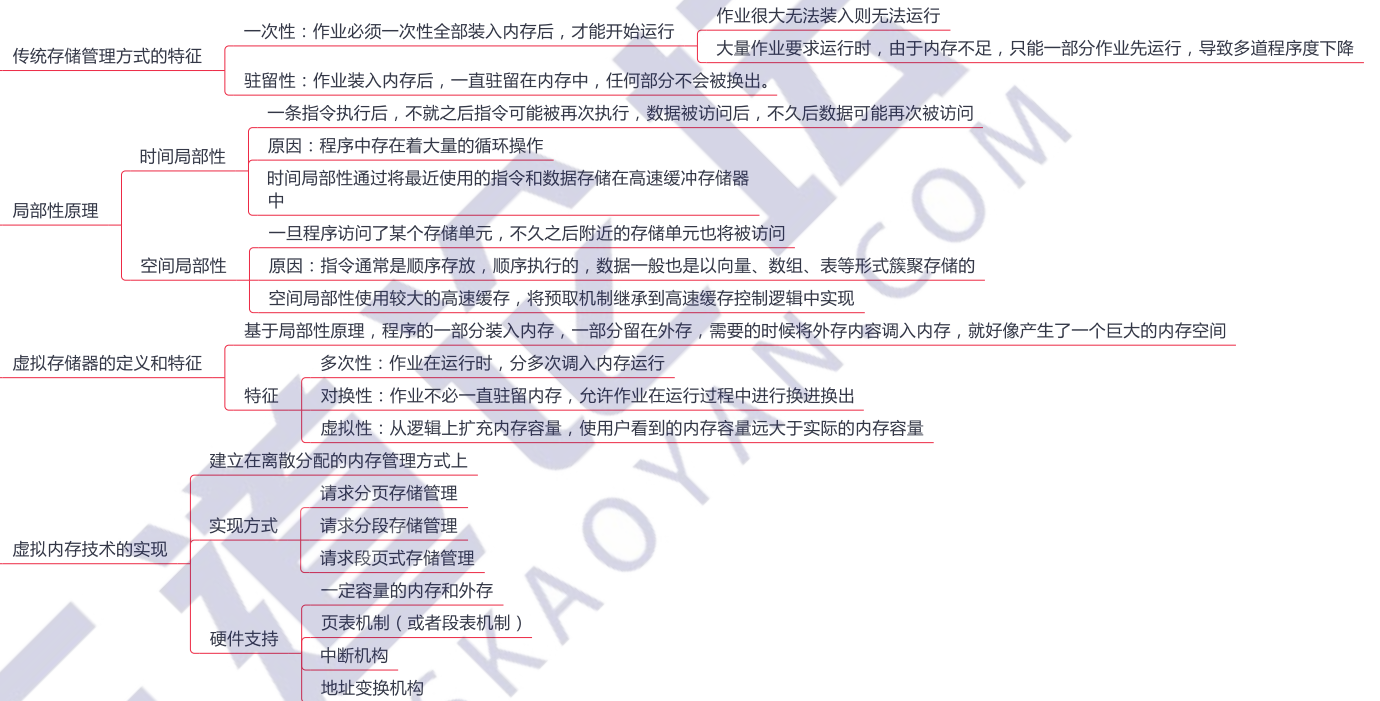


#### 补充

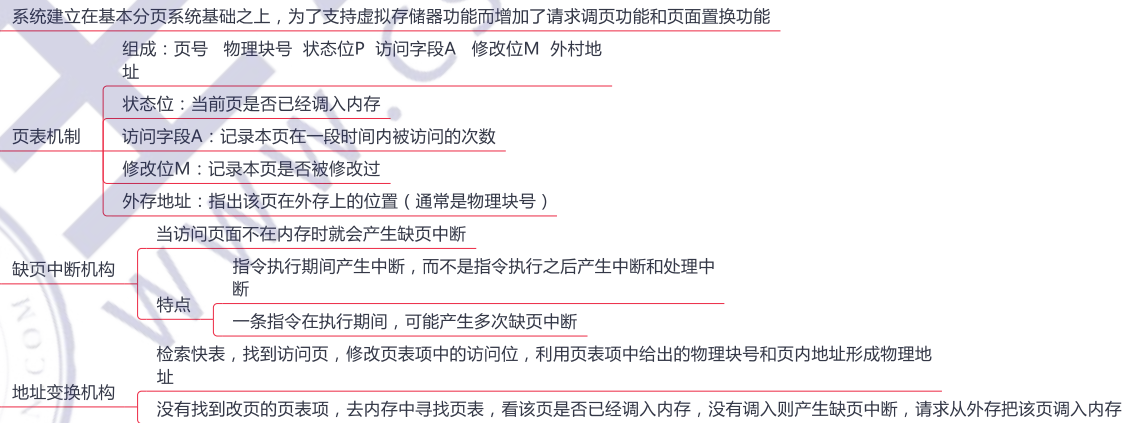
- 不能被修改的代码称为纯代码或可重入代码（不属于临界资源）
- 分段与分页的区别
  - 分页对用户不可见，分段对用户可见
  - 分页的地址空间是一维的，分段的地址空间是二维的
  - 分页（单级页表）、分段访问一个逻辑地址都需要两次访存，分段存储中也可以引入快表机构
  - 分段更容易实现信息的共享和保护（纯代码可重入代码可以共享）
- 分段与分页优缺点
  - 分页管理
    - 优点：内存空间利用率高，不会产生外部碎片，只有少量的页内碎片
    - 缺点：不方便按照逻辑模块实现信息的共享和保护
  - 分段管理
    - 优点：很方便按照逻辑模块实现信息的共享和保护
    - 缺点：如果段长过大，为其分配很大的连续空间会很不方便

## 3.2 虚拟内存管理（上）

### 虚拟内存的基本概念



### 请求分页管理方式



### 易混知识点

- 虚拟内存的最大容量是由计算机的地址结构（CPU寻址范围）确定的
- 虚拟内存的实际容量 =  $\min(\text{内存和外存容量之和}, \text{CPU寻址范围})$



## 3.2虚拟内存管理（下）

### 页面置换算法

- 最佳置换算法（OPT）
  - 选择永不使用或者最长时间内不再访问的页面进行淘汰，但是现实中是无法预知的
  - 优点：缺页率最小，性能最好
- 先进先出页面置换算法（FIFO）
  - 优先淘汰最早进入的页面
  - 优点：实现简单
  - 缺点：与进程的实际运行规律不匹配
  - Belady异常：增大分配的物理块数但是故障数不减反增 只有先进先出算法会出现
- 最近最久未使用（LRU）置换算法
  - 选择最近最长时间没有被访问的页面进行淘汰，每个页面设置一个访问字段，用来标识上次被访问到现在经历的时间
  - 优点：性能好
  - 缺点：实现复杂 需要寄存器和栈的硬件支持 LRU是堆栈类算法
- 时钟(CLOCK)置换算法
  - 像一个时钟一样转圈，每个页面设置一个使用位（访问位），遇到没有被使用的就会将页面换出，然后将使用位置0，如果遇到使用的就会将使用位置零，然后扫描下一个
  - 优点：性能接近于最佳置换算法
  - 缺点：实现复杂 开销大
- 改进型CLOCK算法
  - 使用位（访问位）的基础上增加修改位
  - 扫描缓冲区，选择第一个使用位和修改位都为0的页面换出
  - 扫描过程
    - 第一步失败后，查找使用位为0，修改位为1的进行替换，对于每个跳过的帧，将使用位置为0
    - 第二步失败后，指针回到初始地点且使用位（访问位）均为0，重复第一步
  - 优点：相对于未改进型，节省了时间

### 页面分配策略

- 驻留集：给一个进程的分配的物理页框的集合就是这个进程的驻留集
- 考虑因素
  - 分配给一个进程的存储量越小，任何时候驻留在主存中的进程数就越多，可以提高处理机的时间利用率
  - 一个进程在主存中的页数过少，页错误率就会相对较高
  - 页数过多，对进程的错误率也不会产生过多的影响
- 分配策略
  - 固定分配局部置换
    - 每个进程分配固定物理块数，缺页的时候就进行换页
    - 难以确定每个进程应该分配的物理块数
    - 太多导致资源利用率下降 太少导致频繁缺页中断
    - 进程分配一定物理块，系统自身保留一定空闲物理块，如果进程缺页，就对该进程分配新的物理块
  - 可变分配全局置换
    - 优点：最容易实现，动态调整物理块分配
    - 缺点：如果盲目分配物理块，就会导致多道程序并发能力下降
    - 根据进程的缺页情况，对物理块进行动态分配，如果频繁缺页，就对其多分配物理块，如果缺页率特别低，就减少其物理块
  - 可变分配局部置换
    - 优点：保持了系统的多道程序并发能力
    - 缺点：增大了开销，实现复杂
- 调入页面的时机
  - 预调页策略
    - 将预计不久被访问的页面调入，成功率约为50%
  - 请求调页策略
    - 当进程提出缺页的时候，再按照一定策略进行调页
    - 特点：一次调入一页，调入/调出页面数多时会花费过多的I/O开销
- 从何处调入页
  - 拥有足够的对换空间 可以全部从对换区调入所需页面，提高调页速度
  - 缺少足够的对换区空间 不会被修改的文件从文件区调入，可能被修改的部分换入对换区，以后再从对换区调入 原理：读速度比写速度快
  - UNIX方式 进程相关文件访问文件区，没有运行的页面从文件区调入，曾经运行过但又被换出的页面放在对换区

一般情况下两种策略同时使用

### 抖动

- 原因
  - 刚换出的页面又要换入内存
  - 分配的物理页帧数不足（主要原因）
  - 置换算法不当

### 工作集

- 某段时间内，进程要访问的页面集合。
- 原理
  - 操作系统跟走每个进程的工作集，并为进程分配大于其工作集的物理块
  - 落入工作集的页面需要调入驻留集中，落在工作集外面的页面可以从驻留集中换出
  - 若还有空闲物理块，可以再调入一个进程到内存以增加多道程序数。
  - 若所有进程的工作集之和超过了可用物理块的总数，操作系统就会暂停一个进程，并将其页面调出并将其物理块分配给其他进程