05 虚拟存储器

5.1 虚拟存储器概述

为什么需要虚拟存储器?

> *有的作业很大，作业不能全部被装入内存，使该作业无法运行；*
> *大量作业要求运行时，但由于内存容量不足以容纳所有这些作业，使该作业无法运行。*

5.1.1 基本原理

1.常规存储器管理方式的特征

- 一次性：作业必须全部装入内存才能运行。
- 驻留性：作业中有部分只运行一次，始终占用内存资源

2.局部性原理

> *1968年，P.Denning指出：程序在执行时将呈现出局部性规律，即在一较短的时间内，程序的执行仅局限于某个部分，相应地，它所访问的存储空间也局限于某个区域。*
> *他还提出了其他论点。*

局限性又表现在：

1. 时间局限性：在程序中存在着大量循环操作。
2. 空间局限性：程序的顺序执行

3.虚拟存储器的基本工作情况

应用程序在运行之前仅须将那些当前要运行的「少数页面或段」先装入内存便可运行，其余部分暂留在盘上。如果页面或段未调入内存，则发出 `缺页（段）中断请求` ；如果页段已满，则OS发出 `页（段）置换` 。

5.1.2 虚拟存储器的定义和特征

1.定义

> *指具有 `请求调入功能` 和 `置换功能` ，能从逻辑上对内存容量加以扩充的存储器系统。*
> *容量为内外存之和；速度接近内存；每位成本接近外村。*

2.特征

与传统的存储器管理方式部分比较：

传统的存储器管理	虚拟存储器管理
一次性	多次性
常驻性	对换性
–	虚拟性

> *多次性和对换性建立在离散分配的基础上。*

5.1.3 虚拟存储器的实现方法

> *都要有硬件和软件的支持。*

- 请求分页系统
  - 请求分页的页表机制
  - 缺页中断机构
  - 地址变换机构
- 请求分段系统
  - 请求分段的段表机制
  - 缺页中断机构
  - 地址变换机构

5.2 请求分页存储管理方式

5.2.1 硬件支持

1.请求页表机制

段名	段长	段基址	存取方式	访问字段 A	修改位 M	存在位 P	增补位	外存始址

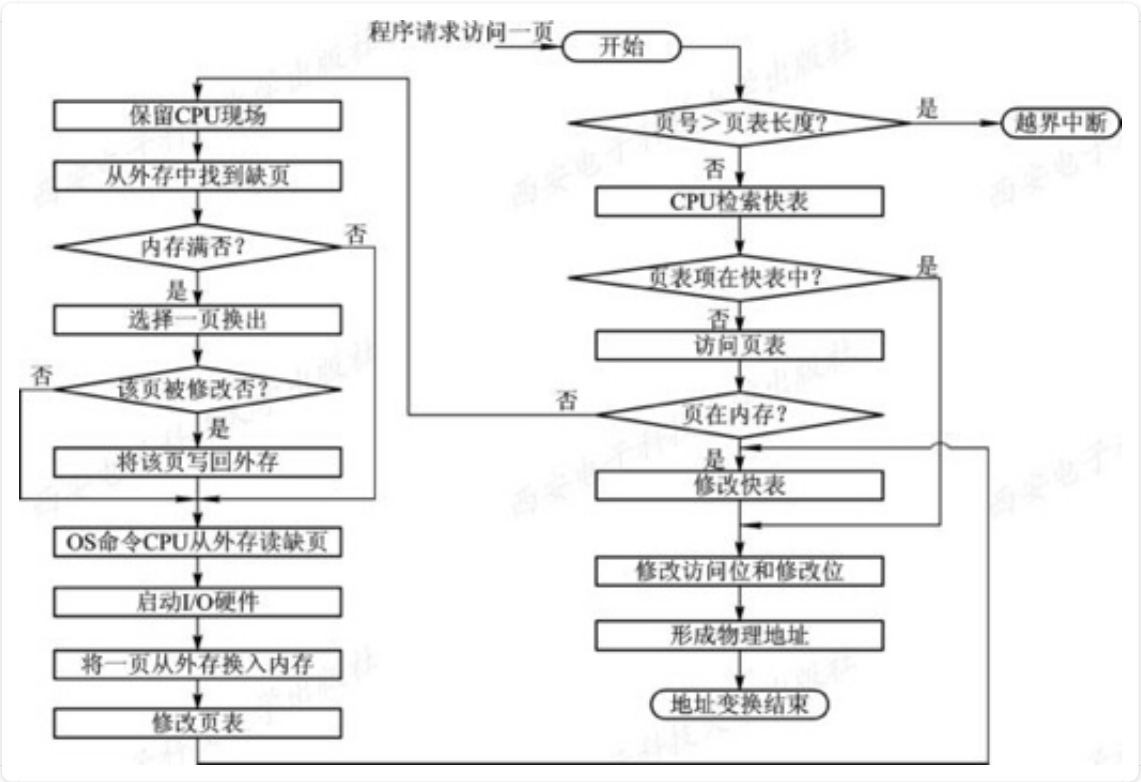
- 状态位P：该页是否已经调入内存。
- 访问字段A：该页在一段时间内访问的次数，或最近多久未被访问。
- 修改位M：该页调入内存后是否被修改过。
- 外存地址：该页在外存上的地址，通常是物理块号。

2.缺页中断机构

和一般中断的区别：

- 在指令执行期间产生和处理中断信号（CPU是在指令执行完才检查中断）
- 一条指令在执行期间可能产生多次缺页中断

3.地址变换机构



5.2.2 内存分配

1.最小物理块数的确定

> *指能保证进程正常运行所需的最小物理块数。*
> *与计算机的硬件结构有关。*

2.内存分配策略

> *在请求分页系统中，可采取两种内存分配策略，即固定和可变分配策略。在进行置换时，也可采取两种策略，即全局置换和局部置换。于是可组合出以下三种适用的策略。*

1. 固定分配局部置换(Fixed Allocation，Local Replacement)
2. 可变分配全局置换(Variable Allocation，Global Replacement)
3. 可变分配局部置换(Variable Allocation，Local Replacement)

3.物理块分配算法

- 「平均分配算法」。即将系统中所有可供分配的物理块平均分配给各个进程。
- 「按比例分配算法」。即根据进程的大小按比例分配物理块。如果系统中共有n个进程，每个进程的页面数为Si，则系统中各进程页面数的总和为：

  $$S = \sum_{i=1}^{n} S_i$$

  又假定系统中可用的物理块总数为m，则每个进程所能分到的物理块数为bi可由下式计算：

  $$b_i = \frac{S_i}{S} \times m$$

  这里，bi应该取整，它必须大于最小物理块数。
- 「考虑优先权的分配算法」。把内存中可供分配的所有物理块分成两部分：一部分按比例地分配给各进程；另一部分则根据各进程的优先权进行分配，为高优先进程适当地增加其相应份额。在有的系统中，如重要的实时控制系统，则可能是完全按优先权为各进程分配其物理块的。

5.2.3 页面调入策略

1.WHEN

- 预调页策略
- 请求调页策略

2.WHERE

- 「系统拥有足够的对换区空间」。这时可以全部从对换区调入所需页面，以提高调页速度。
- 「系统缺少足够的对换区空间」。这时凡是不会被修改的文件，都直接从文件区调入；而当换出这些页面时，由于它们未被修改，则不必再将它们重写到磁盘(换出)，以后再调入时，仍从文件区直接调入。但对于那些可能被修改的部分，在将它们换出时便须调到对换区，以后需要时再从对换区调入。
- 「UNIX方式」。

3.HOW

每当程序所要访问的页面未在内存时(存在位为“0”)，便向CPU发出一缺页中断，中断处理程序首先保留CPU环境，分析中断原因后转入缺页中断处理程序。

4.缺页率

假设一个进程的逻辑空间为n页，系统为其分配的内存物理块数为m(m≤n)。如果在进程的运行过程中，访问页面成功(即所访问页面在内存中)的次数为S，访问页面失败(即所访问页面不在内存中，需要从外存调入)的次数为F，则该进程总的页面访问次数为A = S + F，那么该进程在其运行过程中的缺页率即为

事实上，在缺页中断处理时，当由于空间不足，需要置换部分页面到外存时，选择被置换页面还需要考虑到置换的代价，如页面是否被修改过。没有修改过的页面可以直接放弃，而修改过的页面则必须进行保存，所以处理这两种情况时的时间也是不同的。假设被置换的页面被修改的概率是β，其缺页中断处理时间为ta，被置换页面没有被修改的缺页中断时间为tb，那么，缺页中断处理时间的计算公式为 $t=\beta\times ta+(1—\beta)\times tb$

5.3 页面置换算法(Page-Replacement

Algorithms)

> *在进程运行过程中，若其所要访问的页面不在内存，而需把它们调入内存，但内存已无空闲空间时，为了保证该进程能正常运行，系统必须从内存中调出一页程序或数据送到磁盘的对换区中。但应将哪个页面调出，须根据一定的算法来确定。把选择换出页面的算法称为页面置换算法(Page-Replacement Algorithms)。置换算法的好坏将直接影响到系统的性能。*

5.3.1 最佳置换算法和先进先出置换算法

1.最佳(Optimal)置换算法

> *最佳置换算法是由Belady于1966年提出的一种理论上的算法。*
> *其所选择的被淘汰页面将是以后永不使用的，或许是在最长(未来)时间内不再被访问的页面。采用最佳置换算法通常可保证获得最低的缺页率。但由于目前还无法预知，一个进程在内存的若干个页面中，哪一个页面是未来最长时间内不再被访问的，因而该算法是无法实现的，但可以利用该算法去评价其它算法。*

2.先进先出(FIFO)页面置换算法

> *FIFO算法是最早出现的置换算法。*
> *该算法总是淘汰最先进入内存的页面。*

该算法实现简单，只需把一个进程已调入内存的页面按先后次序链接成一个队列，并设置一个指针，称为替换指针，使它总是指向最老的页面。但该算法与进程实际运行的规律不相适应，因为在进程中，有些页面经常被访问，比如，含有全局变量、常用函数、例程等的页面，FIFO算法并不能保证这些页面不被淘汰。

5.3.2 最近最久未使用和最少使用置换算法

1. LRU(Least Recently Used)置换算法

> *最近最久未使用(LRU)的页面置换算法是根据页面调入内存后的使用情况做出决策的。利用「最近的过去」和「最近的将来」的近*

> *似。*

LRU置换算法的硬件支持

寄存器

为了记录某进程在内存中各页的使用情况，须为每个在内存中的页面配置一个移位寄存器，可表示为：

R = Rn−1Rn−2Rn−3 … R2R1R0
当进程访问某物理块时，要将相应寄存器的Rn−1位置成1。此时，定时信号将每隔一定时间(例如100 ms)将寄存器右移一位。如果我们把n位寄存器的数看作是一个整数，那么，具有最小数值的寄存器所对应的页面，就是最近最久未使用的页面。

栈

可利用一个特殊的栈保存当前使用的各个页面的页面号。每当进程访问某页面时，便将该页面的页面号从栈中移出，将它压入栈顶。因此，栈顶始终是最新被访问页面的编号，而栈底则是最近最久未使用页面的页面号。

3.最少使用(Least Frequently Used，LFU)置换算法

> *在采用LFU算法时，应为在内存中的每个页面设置一个移位寄存器，用来记录该页面被访问的频率。该置换算法选择在最近时期使用最少的页面作为淘汰页。*

5.3.3 Clock置换算法

> *Clock是LRU的近似算法*

1.简单的Clock置换算法

当利用简单Clock算法时，只需为每页设置一位访问位，再将内存中的所有页面都通过链接指针链接成一个循环队列。

图5−8　简单Clock置换算法的流程和示例

2.改进型Clock置换算法

在将一个页面换出时，如果该页已被修改过，便须将该页重新写回到磁盘上；但如果该页未被修改过，则不必将它拷回磁盘。换而言之，对于修改过的页面，在换出时所付出的开销比未修改过的页面大，或者说，置换代价大。在改进型Clock算法中，除须考虑页面的使用情况外，还须再增加一个因素——置换代价。

5.3.4 页面缓冲算法(Page Buffering Algorithm，PBA)

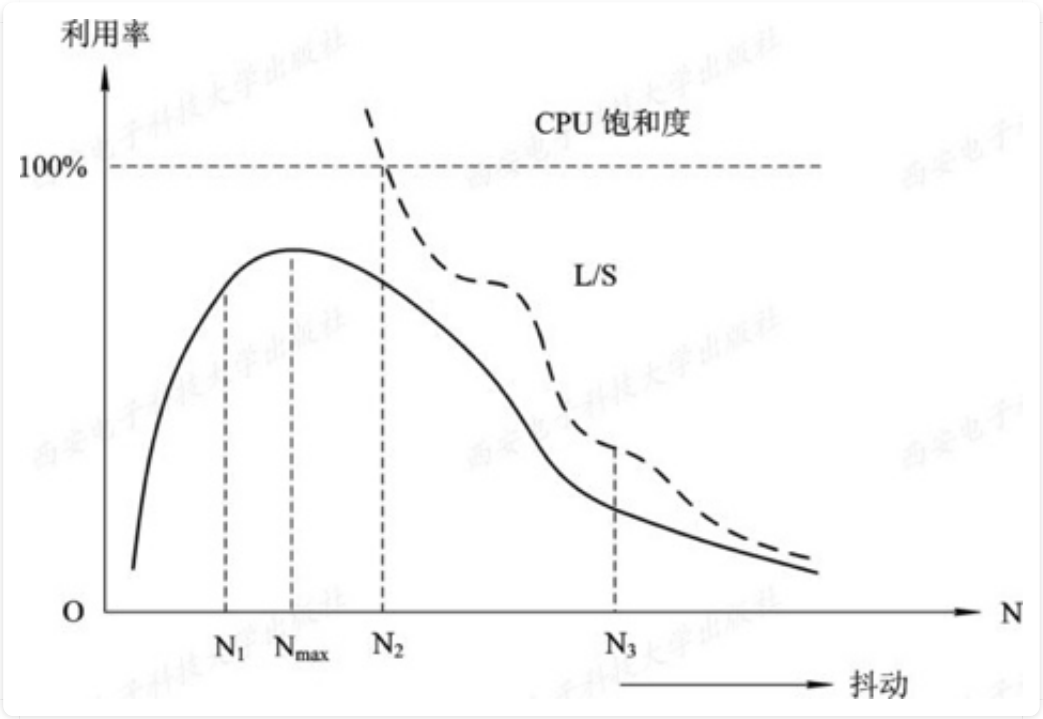
1.影响页面换进换出效率的若干因素

- 页面置换算法
- 写回磁盘的频率
- 读入内存的频率

2.页面缓冲算法PBA

PBA算法的主要特点是：

- 显著地降低了页面换进、换出的频率，使磁盘I/O的操作次数大为减少，因而减少了页面换进、换出的开销；
- 正是由于换入换出的开销大幅度减小，才能使其采用一种较简单的置换策略，如先进先出(FIFO)算法，它不需要特殊硬件的支持，实现起来非常简单。 1) 空闲页面链表 2) 修改页面链表 ## 5.3.5 访问内存的有效时间 与基本分页存储管理方式不同，在请求分页管理方式中，内存有效访问时间不仅要考虑访问页表和访问实际物理地址数据的时间，还必须要考虑到缺页中断的处理时间。 # 5.4 “抖动”与工作集 ## 5.4.1 多道程序度与“抖动” ### 1.多道程序度与处理机的利用率 处理机随进程数的实际利用率



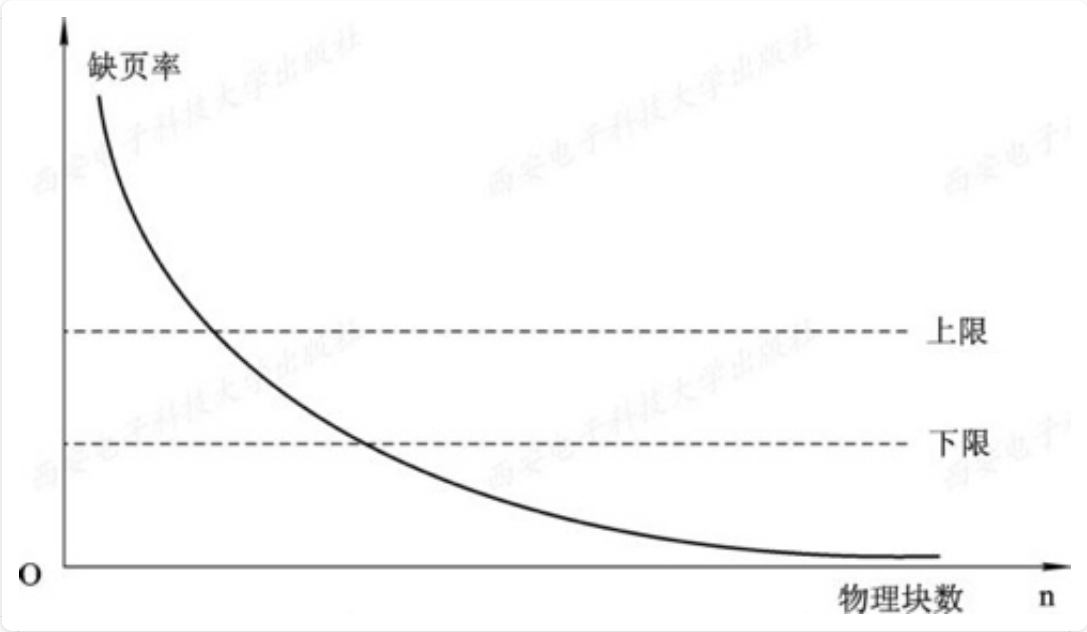
2.产生“抖动”的原因

同时在系统中运行的进程太多，由此分配给每一个进程的「物理块太少」，致使每个进程在运行时，「频繁出现缺页」，必须请求系统将所缺之页调入内存。这会使得在系统中排队等待页面调进/调出的进程数目增加。显然，对磁盘的有效访问时间也随之急剧增加，造成每个进程的大部分时间都用于页面的换进/换出，而几乎不能再去做任何有效的工作，从而导致发生处理机的利用率急剧下降并趋于0的情况。

5.4.2 工作集

1.工作集的基本概念

进程发生缺页率的时间间隔与进程所获得的物理块数有关。



物理块数过多或者过少都会对缺页率有较大影响。

2.工作集的定义

> *是指在某段时间间隔Δ里，进程实际所要访问页面的集合。*
> *Denning指出，虽然程序只需要少量的几页在内存便可运行，但为了较少地产生缺页，应将程序的全部工作集装入内存中。然而我们无法事先预知程序在不同时刻将访问哪些页面，故仍只有像置换算法那样，用程序的过去某段时间内的行为作为程序在将来某段时间内行为的近似。*

5.4.3 “抖动”的预防方法

> *几乎都是采用调节多道程序度来控制“抖动”发生。*

1.采取局部置换策略

在页面分配和置换策略中，如果采取的是可变分配方式，则为了预防发生“抖动”，可采取局部置换策略。

2.把工作集算法融入到处理机调度中

当调度程序发现处理机利用率低下时，它将试图从外存调入一个新作业进入内存，来改善处理机的利用率。

3.利用“L=S”准则调节缺页率

Denning于1980年提出了“L=S”的准则来调节多道程序度，其中L是缺页之间的平均时间，S是平均缺页服务时间，即用于置换一个页面所需的时间。如果是L远比S大，说明很少发生缺页，磁盘的能力尚未得到充分的利用；反之，如果是L比S小，则说明频繁发生缺页，缺页的速度已超过磁盘的处理能力。只有当L与S接近时，磁盘和处理机都可达到它们的最大利用率。理论和实践都已证明，利用“L=S”准则，对于调节缺页率是十分有效的。

4.选择暂停的进程

当多道程序度偏高时，已影响到处理机的利用率，为了防止发生“抖动”，系统必须减少多道程序的数目。

5.5 请求分段存储管理方式

5.5.1 硬件支持

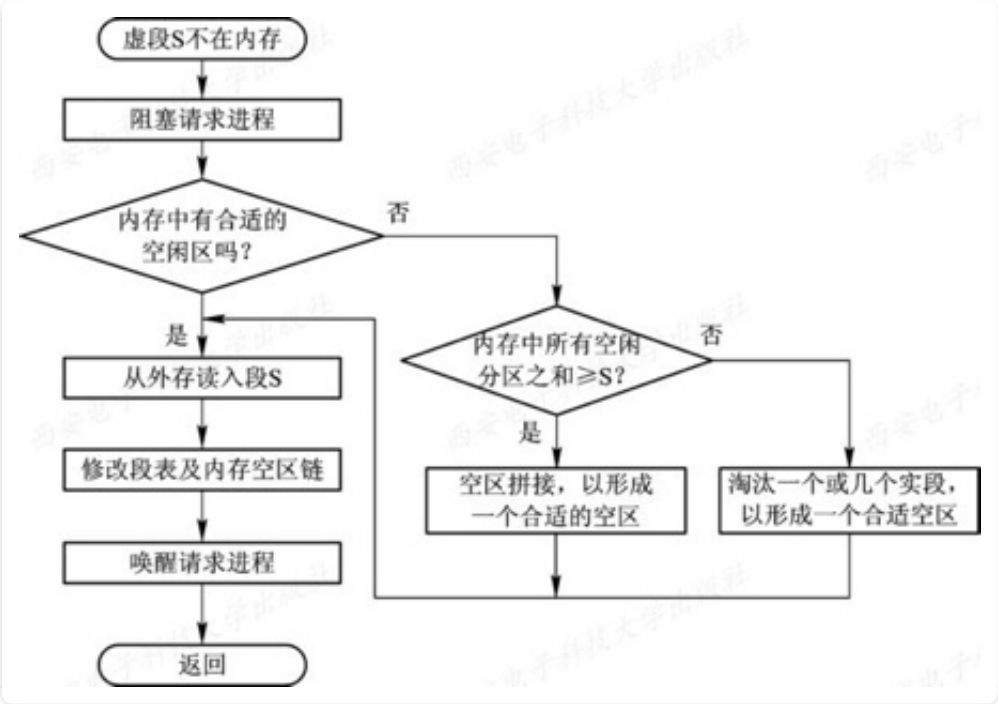
1.请求段表机制

具有请求分页机制中有的访问字段A、修改位M、存在位P和外存始址四个字段外，还增加了「存取方式」字段和「增补位」。

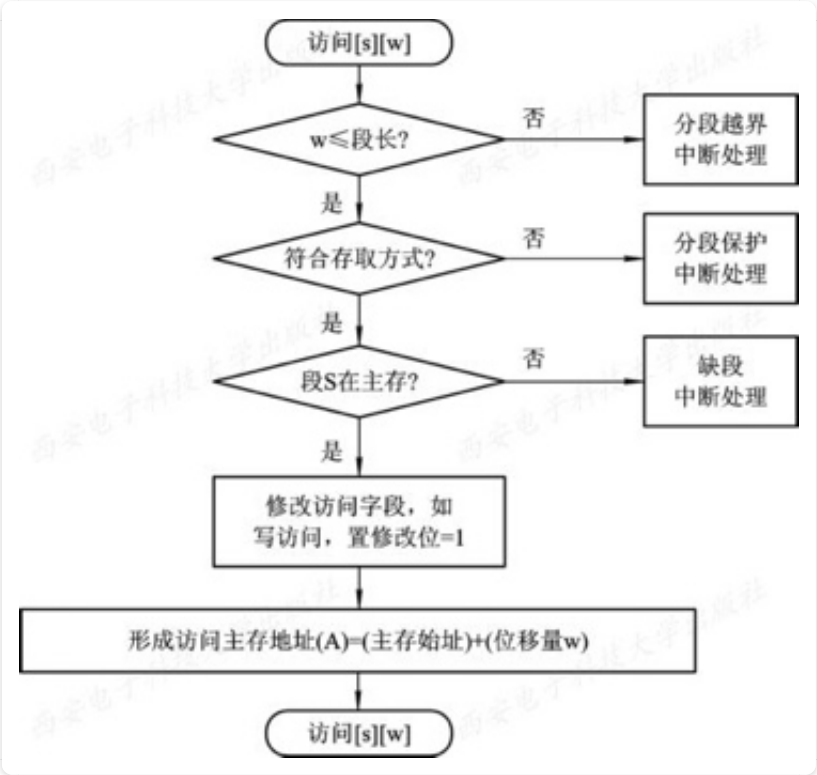
段名	段长	段基址	存取方式	访问字段 A	修改位 M	存在位 P	增补位	外存始址

- 存取方式：两位字段，只执行、只读、可读、可写。
- 增补位：本段在运行过程中是否做过动态增长。

2.缺段中断机构



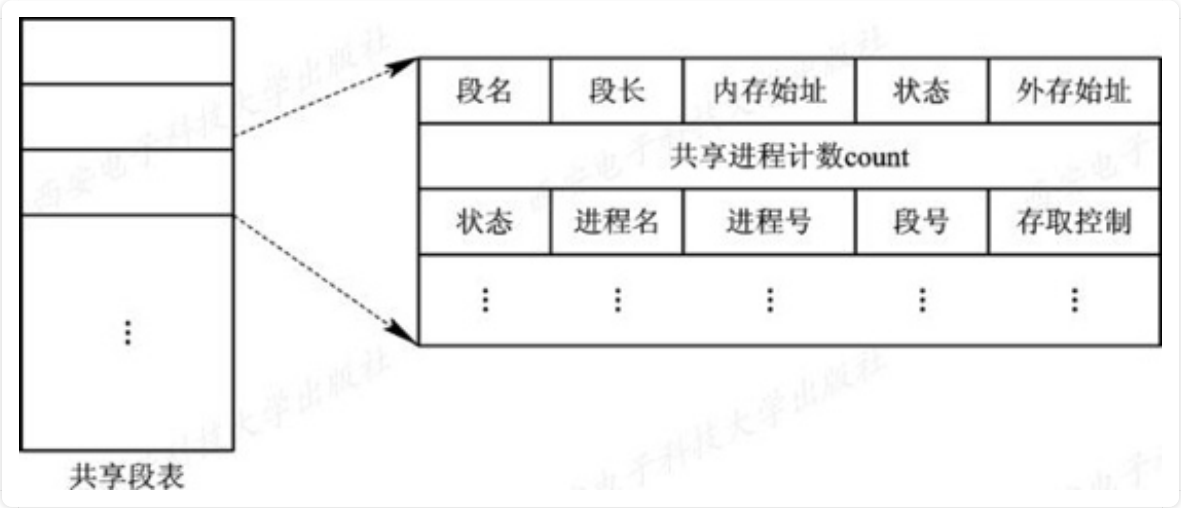
3.地址变换机构



5.5.2 分段的共享与保护

1.分段共享

（1）共享段表



- 共享进程计数count
- 存取控制字段
- 段号

（2）共享段的分配与回收

共享段的分配共享段的回收

2.分段保护

越界检查存取控制检查环保护机构