# 第五章 虚拟存储器

1. **5.1 虚拟存储器概述**

从逻辑上 进行内存的扩充

* 1. **5.1.1 常规存储管理方式的特征和局部性原理**
     1. **1. 常规存储器管理方式的 特征**
        1. **一次性**：必须一次性转入。大作业不能在小内存运行
        2. **驻留性**：不运行完，不取出
     2. **2. 局部性原理**
        1. **在一较短的时间内，程序的执行仅限于某个部分：它所访问的存储空间也局限于某个区域。**
        2. 论点：
           1. 1）多顺序执行
           2. 2）过程调用深度有限
           3. 3）多循环结构
           4. 4）多对数据结构的处理
        3. **局限性**
           1. **1）时间局限性** （循环操作）

如果程序的某条指令被执行，则不久以后该指令可能再次执行

如果某数据被访问过，则不就以后该数据可能再次被访问

* + - * 1. **2）空间局限性** （顺序执行）

一旦程序访问了某个存储单元，不久之后，其附近的存储单元也将被访问

* + 1. **3. 基本工作原理**
       1. 仅将当前需要的段 装入内存
       2. 缺页时，发出中断请求，进行调页；若内存满，则进行置换，再调入
  1. **5.1.2 定义和特征**
     1. **1. 虚拟存储器的定义**
        1. **具有调入和置换功能，能从逻辑上对内存容量加以扩充的一种存储系统。**
        2. 逻辑容量 => CPU的长度地址有关 ( **本质** ) ；内存 + 外存
        3. 运行速度 == 内存速度
        4. 每位成本 == 外存成本
     2. **2. 虚拟存储器的特征**
        1. **多次性**（最重要）：可以多次调入 [ 独有的特性]
        2. 对换性：可以运行过程中进程换进、换出
        3. 虚拟性：逻辑扩充内存容量
        4. **虚拟性 以多次性和对换性为基础**
  2. **5.1.3 实现方法**

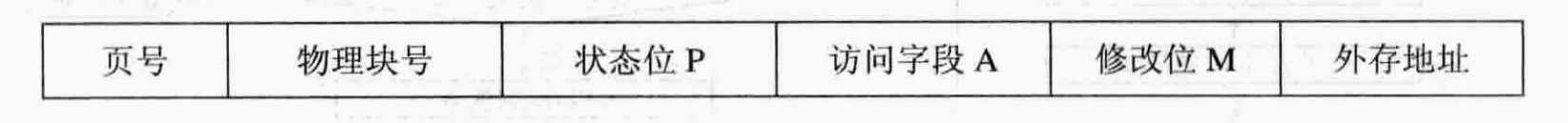
**均为 离散分配存储管理方式**

* + 1. **1. 请求分页系统**
       1. 页式虚拟存储系统 = 基本分页系统 + 请求调页功能 + 页面置换功能
       2. 硬/软件支持
          1. 硬件：请求分页的页表机制、缺页中断机构、地址变换机构
          2. 软件：请求调页、页面置换的软件
    2. **2. 请求分段系统**
       1. 段式虚拟存储系统 = 基本分段系统 + 请求调段功能 + 分段置换功能
       2. 硬软件支持
          1. 硬件：请求分段的段表机制、缺页中断机构、地址变化机构
          2. 软件：请求调段、段置换的软件

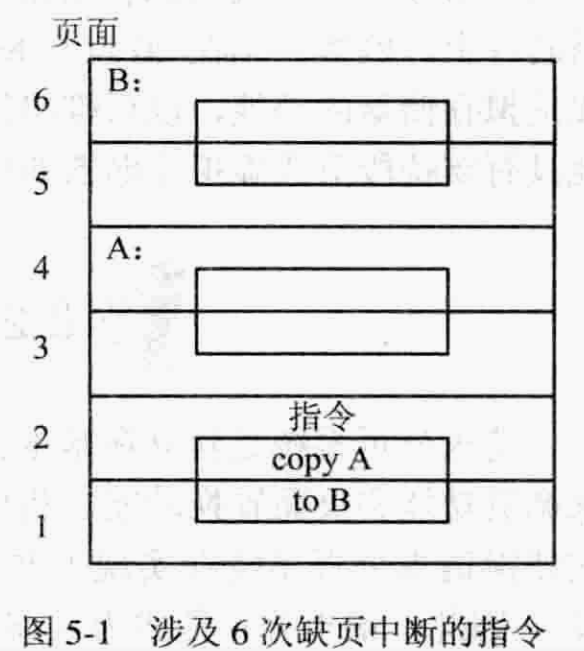
1. **5.2 请求分页存储管理方式**

相对于分页系统：多了支持页面调度和页面置换

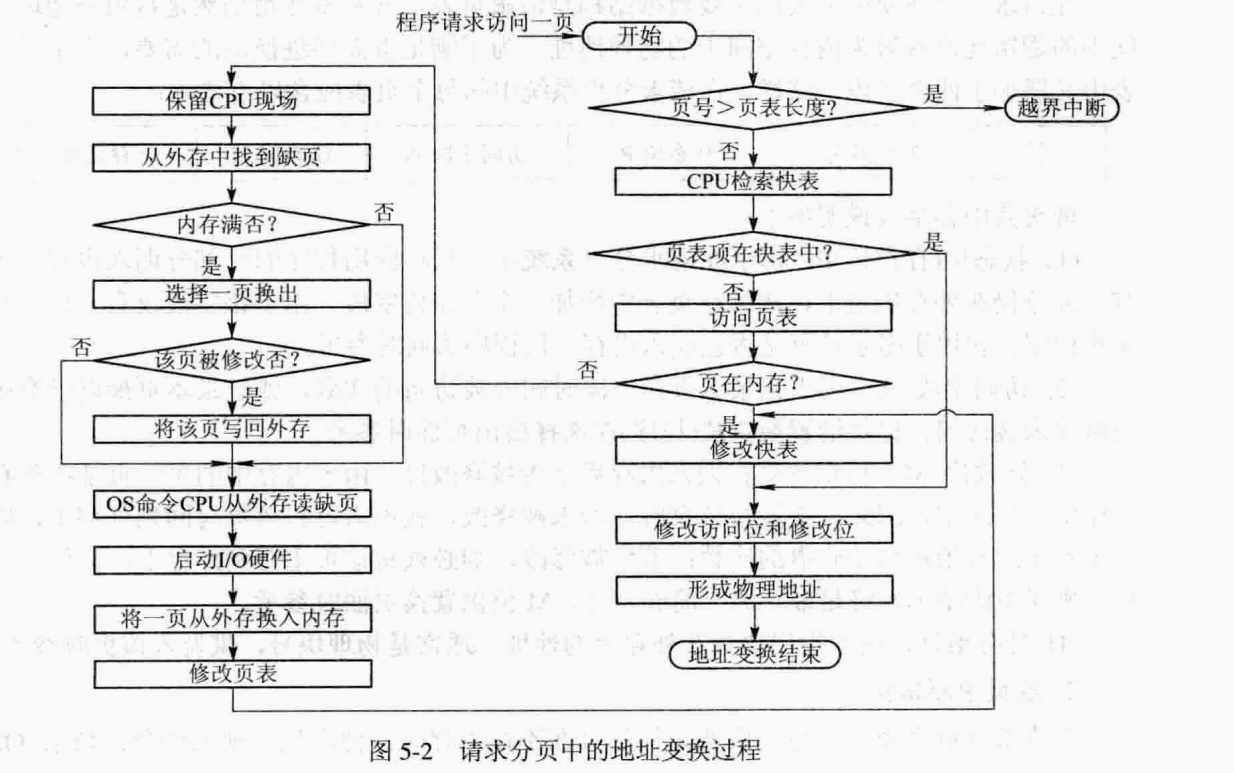
* 1. **5.2.1 请求分页中的硬件支持**
     1. **1. 请求页表机制**
        1. 功能：逻辑地址 == 》 物理地址
        2. 页表项：



* + - * 1. **状态位 P ：** 是否调入内存。1bit
        2. 访问字段A：访问次数。
        3. **修改位 M：**调入内存后，是否被修改。*判断写回时，是否需要更新外存的内容。*
        4. 外存地址：物理块号（也需要转换--> 磁盘扇区）
    1. **2. 缺页中断机构**
       1. 在**指令执行期间**产生和处理中断信号。
       2. 一条指令在执行期间**可能发生多次缺页中断**



* + - * 1. 页的划分是静态划分。因此一条指令可以在多页中，若都缺页，则会都需要中断调取。
    1. **3. 地址变化机构**
       1. 流程图



* + - 1. 检索快表。找得到，调用
      2. 快表找不到，去页表找。
         1. 状态位P ： 在内存。改页调入快表。（置换/调入）
         2. 状态位P：不在内存。产生中断，去磁盘找。放入内存，放入快表。
  1. **5.2.2 请求分页中的内存分配**
     1. **1. 最小物理块数的确定**

分页，一个进程分的页太多，影响并行度；太少，造成总缺页。

* + - 1. 最小物理块数：能保证进程正常运行的最小物理块数。
      2. 决定因素：指令格式、功能、寻址方式
      3. 直接寻址，最少物理块 >= 2
      4. 间接寻址，最少物理块 >= 3
    1. **2. 内存分配策略**

分配策略（2种）：固定和可变； 置换策略（2种）：全局和局部。

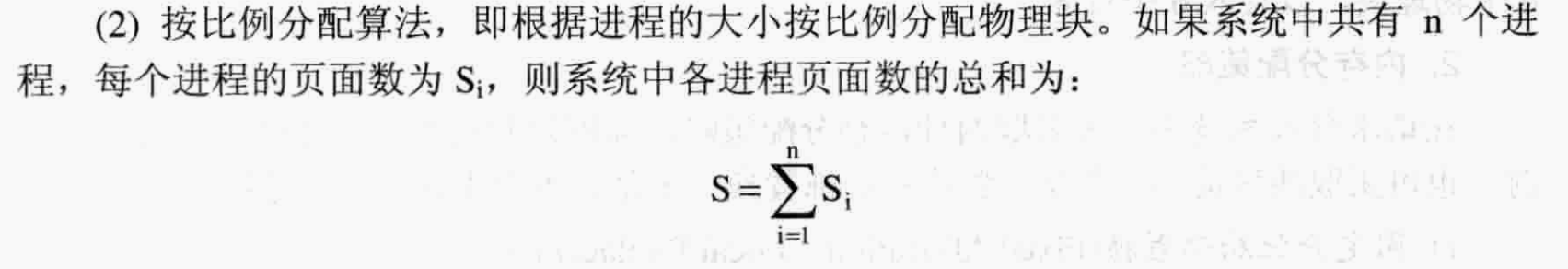
* + - 1. **1） 固定分配局部置换 （Fixed Allocation, Local Replacement ）**
         1. **固定分配：**每个进程分配一组**固定数目**的物理块，在进程运行期间不再改变。
         2. **局部置换：**若缺页，只从分配给该进程的n个页面中**选出一页换出，再选出一页调入**。

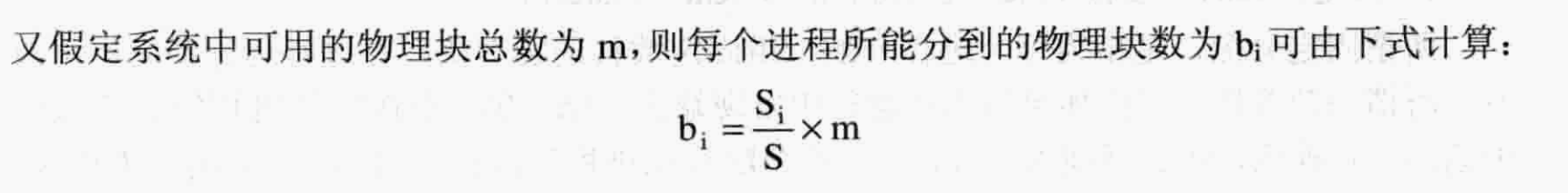
自己和自己换，给20块，就在20块内换入换出。

* + - * 1. 缺点（实现困难）

物理块分配：太少，缺页频繁；太多，内存驻留进程过多。

* + - 1. **2.）可变分配全局置换 （Variable Allocation, Global Replacement ）**
         1. **可变分配：**为每个进程分配一定数目的物理块，在进程运行期间，可根据情况做适当的增加或减少。
         2. **全局置换：**若缺页，将OS所保留的空闲物理块取出一块分配给该进程/ 所有进程的全部物理块为标的，选择一块换出，再调入缺页。
      2. **3）可变分配局部置换 （Variable Allocation, Local Replacement ）**
         1. 发生缺页，只是在进程的页面内进行换入换出
         2. **对于频繁中断的进程分配附加物理块。**
    1. **3. 物理块分配算法**
       1. **1）平均分配算法**
          1. 100个物理块，5个进程运行，每个进程20个物理块
       2. **2）按比例分配算法**





* + - * 1. **可分配到的物理块数 = （该进程页面数 / 所有进程页面数总和） × 物理块总数**
      1. **3）考虑优先权的分配算法**
         1. 重要的：分配多内存
  1. **5.2.3 页面调入策略**
     1. **1. 何时调入页面**
        1. **1. 预调页策略 （预取）**
           1. 将预测未来会被访问的页面调入内存。
           2. **调入页面可能会被访问。**
        2. **2. 请求调页策略 （置换）**
           1. 发现内存缺页，立即请求调页。
           2. **调入页面一定会被访问。**
     2. **2. 从何处调入页面**

磁盘 = 对换区（连续空间） + 文件区（离散）

* + - 1. 1）**对换区足够大**，直接在**对换区调入调出**

进程运行前，把文件全部拷贝到对换区

* + - 1. 2）对换区不够大，**不被修改，从文件区调换**；**可能被修改**，调出时，先放入对换区，再从**对换区调**。
      2. 3）UNIX方式。未运行，从文件区调换；运行过，又被换出，从对换区调。
    1. **3. 页面调入过程**
       1. 1）缺页（存在位 P=0），CPU发出中断请求。
       2. 2）中断处理程序：保留CPU环境、分析中断原因 =》缺页中断程序
       3. 3）缺页中断程序：查页表 =》物理块
          1. 内存未满，启动I/O，调入缺页，修改页表
          2. 内存满，置换页面，调入缺页，修改页表，存在位=1

未被修改，无需写回磁盘

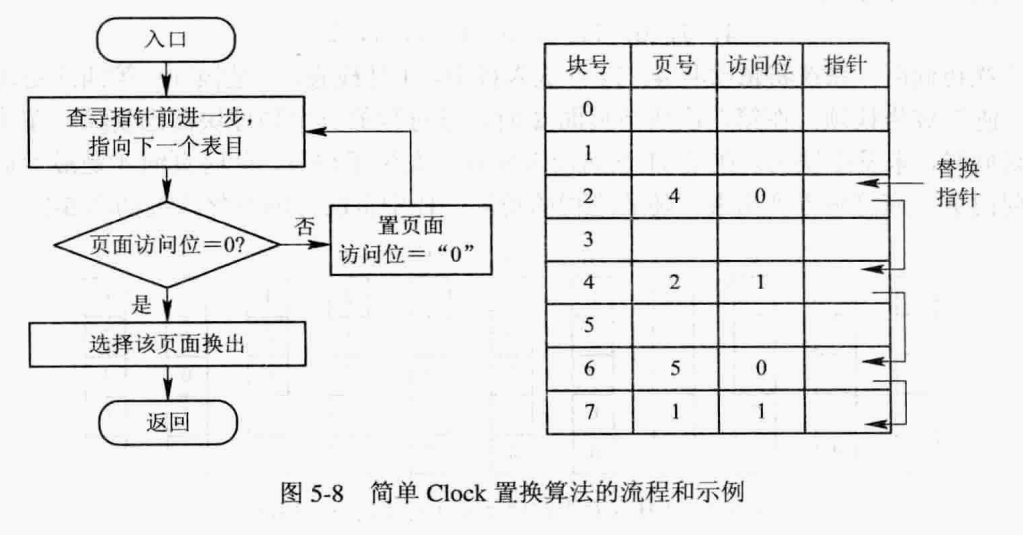
被修改，需写回磁盘。

* + - 1. 4）更新快表
      2. 5）形成物理地址，访问数据
    1. **4. 缺页率**
       1. **缺页率 = 页面访问失败次数 / 总次数**
       2. **命中率 = 命中次数 / 总次数**
       3. 影响因素
          1. 页面大小：页面划分大，缺页率低
          2. 进程分配内存块的数目：分配块数多，缺页率低
          3. 页面置换算法：算法优劣决定
          4. 程序固有特性：程序局部化越高，缺页率低
       4. **缺页中断处理时间**
          1. t = 页面被修改概率 × 页面修改缺页中断处理时间+ （1 - 页面被修改概率） × 页面未修改缺页中断处理时间

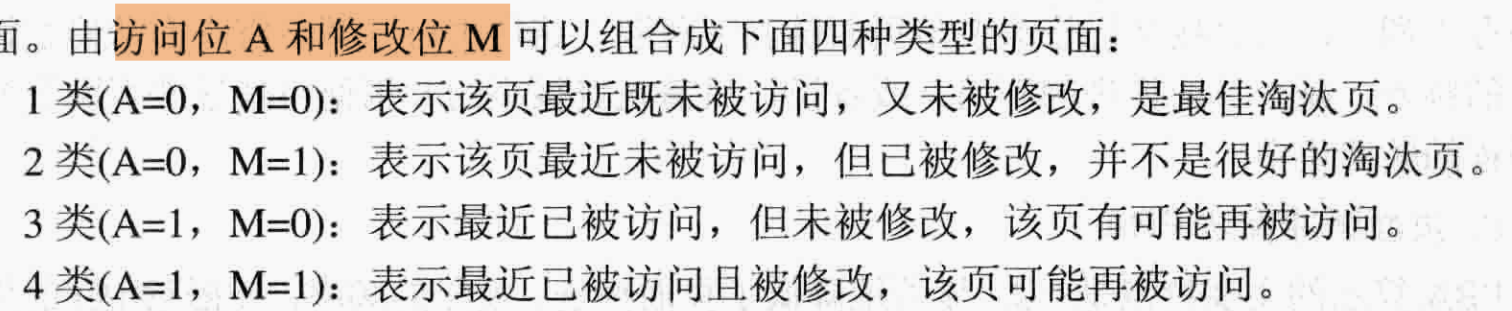
1. **5.3 页面置换算法**

**抖动：刚调出，又很快被调入**

* 1. **5.3.1 最近置换算法和先进先出置换算法**
     1. **1. 最佳（Optimal）置换算法 OPT**
        1. 算法：选择以后**最长时间不会被使用**的页面淘汰出去
        2. 特点：
           1. 理论上，性能最佳
           2. 实际上，无法实现
           3. 作为其他算法的评价指标
     2. **2. 先进先出 FIFO**
        1. 算法：总淘汰**最先进入内存**的页面
        2. 实现：一个始终指向最早页面的指针
  2. **5.3.2 最近最久未使用和最少使用置换算法**
     1. **3. 最近最久未使用置换算法 LRU（Least Recently Used）**
        1. 算法：选择**最长没有被使用**的进程调出
        2. 实现：每页新增一个访问字段，记录上次访问到现在的时间 t。t最大则淘汰。
        3. 硬件支持：
           1. 寄存器、栈
        4. **最接近最优，但硬件支持多**
     2. **4. 最少使用（Least Frequently Used， LFU）置换算法**
        1. 算法：选择在**最近时期使用最少**的页面作为淘汰页
  3. **5.3.3 Clock置换算法**
     1. **1. 简单的Clock**
        1. 每页设置一个访问位。访问后，置1
        2. 内存中所有页链接成一个循环队列
        3. **算法描述**
           1. 循环检查页的使用情况
           2. 访问位 = 0，则淘汰它；访问位 = 1，置0，继续找
           3. \*又称“最近未使用”置换算法 （NRL = Not Recently Used）
        4. 流程图



* + 1. **2. 改进的Clock**
       1. 增加对于修改位的考虑。**最好替换未访问，未修改的页面**



* + - 1. **算法描述**
         1. 循环查找
         2. a. 第一次 查找 ( A = 0, M = 0 )，存在，则找到淘汰页。不改变A
         3. b. 第二次 查找(A = 0, M = 1 )，访问位A 全置0
         4. c. 重复a,b两步，一定能找到淘汰页
  1. **5.3.4 页面缓冲置换算法 (Page Buffering Algorithm, PBA)**
     1. **1. 影响页面换进换出效率的若干因素**
        1. 1）页面置换算法
        2. 2）写回磁盘的频率：攒一段修改页，然后一起写回
        3. 3）读入内存的频率：没有写回时，可以直接在已修改换出页面链表上访问
     2. **2. 页面缓冲算法 PBA**
        1. 特点
           1. 显著降低页面换进、换出频率，减少开销
           2. 可采用较简单的置换策略。（不需要硬件支持）
        2. 实现
           1. **1）空闲页面链表：**

掌握空闲物理块，分配给频繁缺页的进程。

当进程需要时，把页面装入第一个物理块。当要换出时，把改物理块挂到链表末尾。

* + - * 1. **2）修改页面链表：**

减少已修改页面换出次数

保存已修改且需要被换出的页面，等被换出的页面数目达到一定值时，再一起换出外存

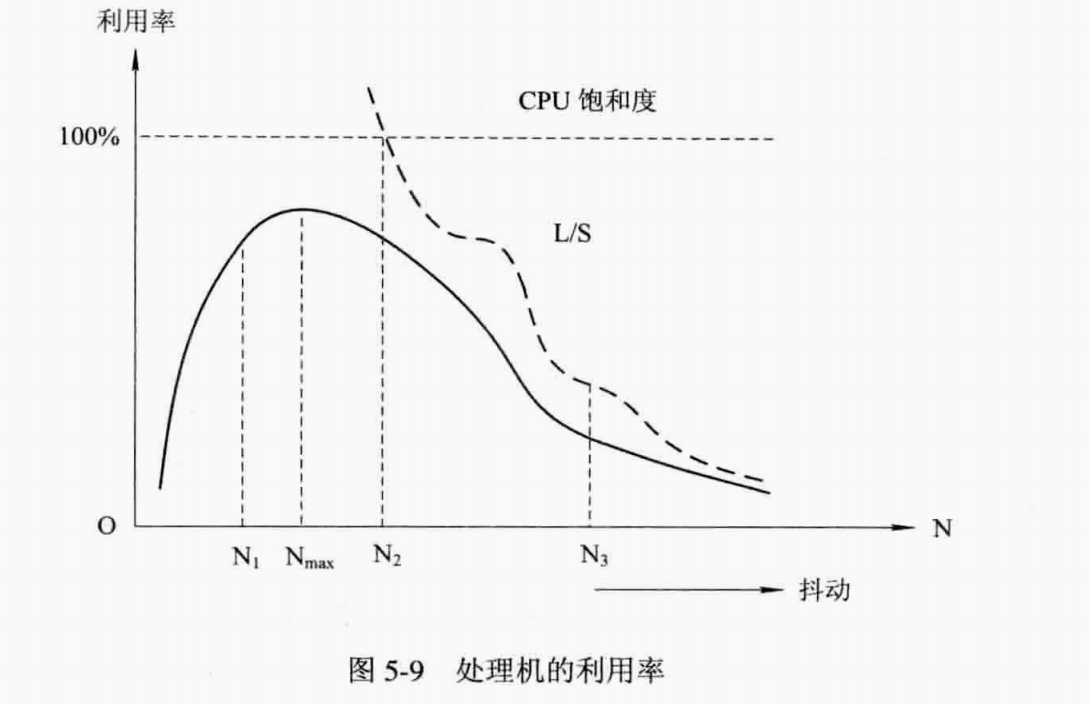
原因：因为磁盘中的扇区 = n个物理块。每次等到数据 累积到 磁盘的数据大小，再送出。（匹配磁盘的固定大小）

* 1. **5.4.5 访问内存的有效时间**

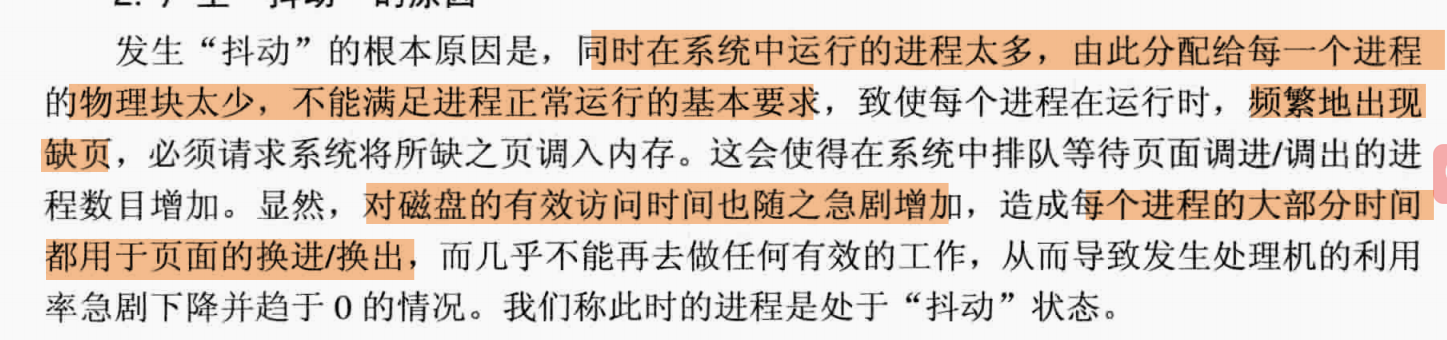
**查找快表时间： λ 访问实际物理地址所需时间： t （访问内存） ​缺页中断处理时间：ε 快表命中率：a 缺页率：f**

* + 1. 访问页在内存，且其对应页表项在快表中
       1. **EAT = λ + t**
       2. 查快表 + 访问内存
    2. 访问页在内存，且对应页不在快表中
       1. **EAT = λ + t + λ + t = 2( λ + t )**
       2. 查快表 + 查页表（内存）+ 更新快表 + 访问内存
    3. 访问页不在内存中
       1. **EAT = λ + t + ε + λ + t**
       2. 查快表 + 查页表 + 缺页中断 + 更新快表 + 访问内存
    4. 考虑命中率
       1. EAT = λ + a × t + ( 1 - a ) × [ t + ( 1 - f ) × ( t + λ) + f × ( ε + λ + t ) ]
    5. 不考虑命中率
       1. EAT = t + f × （ε + t ) + (1 - f) × t

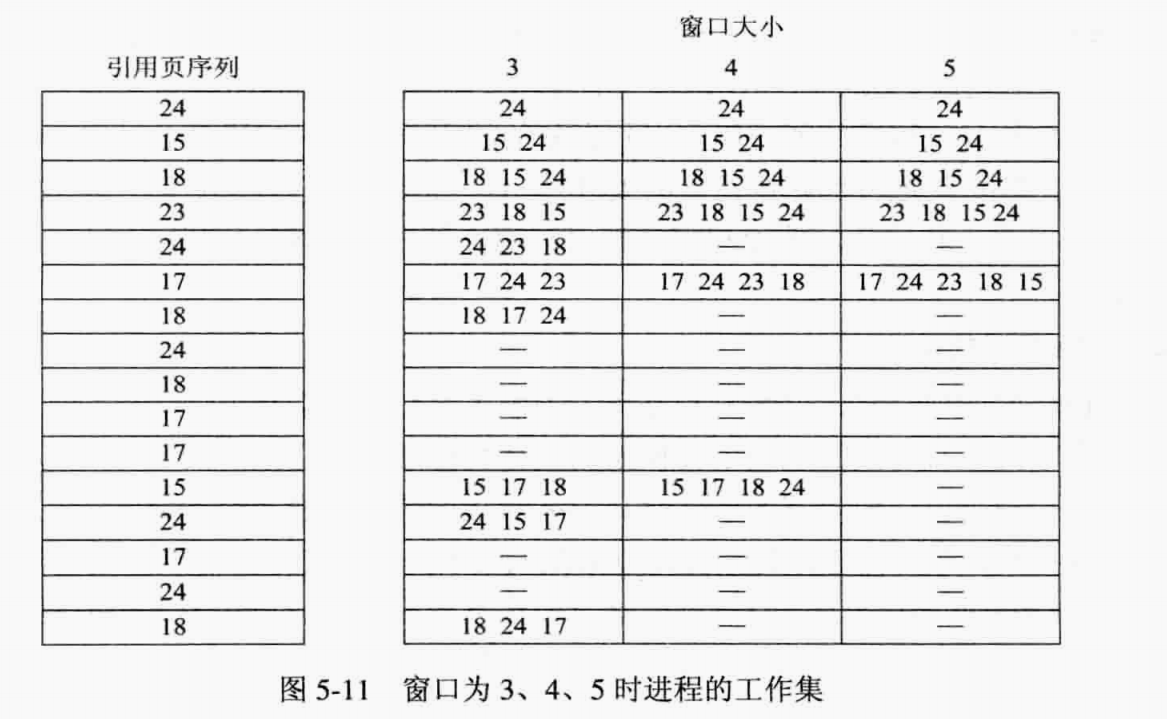
1. **5.4 抖动与工作集**
   1. **5.4.1 多道程序度与“抖动”**
      1. **1. 多道程序度与处理机的利用率**



* + 1. **2. 产生“抖动”的原因**



* 1. **5.4.2 工作集**
     1. 定义：在某段时间间隔△里，进程实际所要访问页面的集合。
     2. 窗口窗口尺寸：某进程在时间t的工作集记为w(t,△)，△就是窗口尺寸
     3. 图5-11

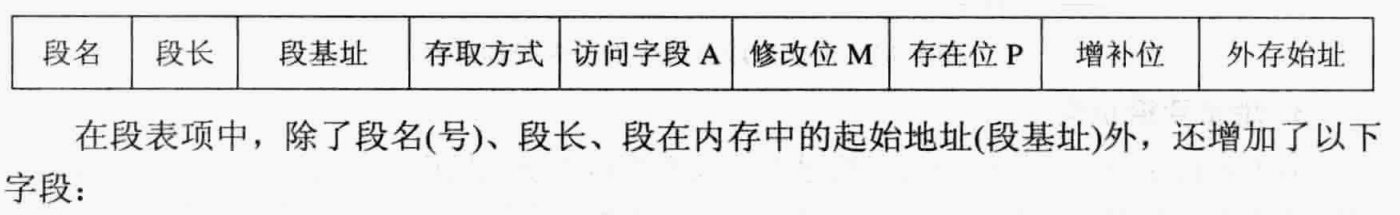


* 1. **5.4.3 “抖动”的预防方法**
     1. **1. 采用局部置换策略**
        1. 缺页时，仅在局部的内存块中，自我影响。不干扰其他进程
        2. 优点：简单
        3. 缺点：效果不是很好
     2. **2. 把工作集算法融入到处理机调度中**
        1. 调入前，先检查每个进程在内存的驻留页面是否足够多
           1. 足够多，从外存调入新的作业
           2. 页面不足，缺页率高的作业新增物理块，不再调入作业
     3. **3. 利用“L= S”准则调节缺页率**
        1. L ：缺页之间的平均时间
        2. S ： 平均缺页服务时间

**置换一个页面所需要的时间**

* + - 1. L > S ： 很少发生缺页。**磁盘没有充分利用。**
      2. L = S：磁盘和处理机都可达到最大利用率
      3. L < S：频繁缺页。**缺页速度超过磁盘处理能力**
    1. **4. 选择暂停的进程**
       1. 减少多道程序的数目
       2. 首先，选择暂停**优先级最低**的进程。
       3. 其次，选择暂停**优先级较低**的进程
       4. 进一步，暂停一个**并不十分重要，但却较大**的进程

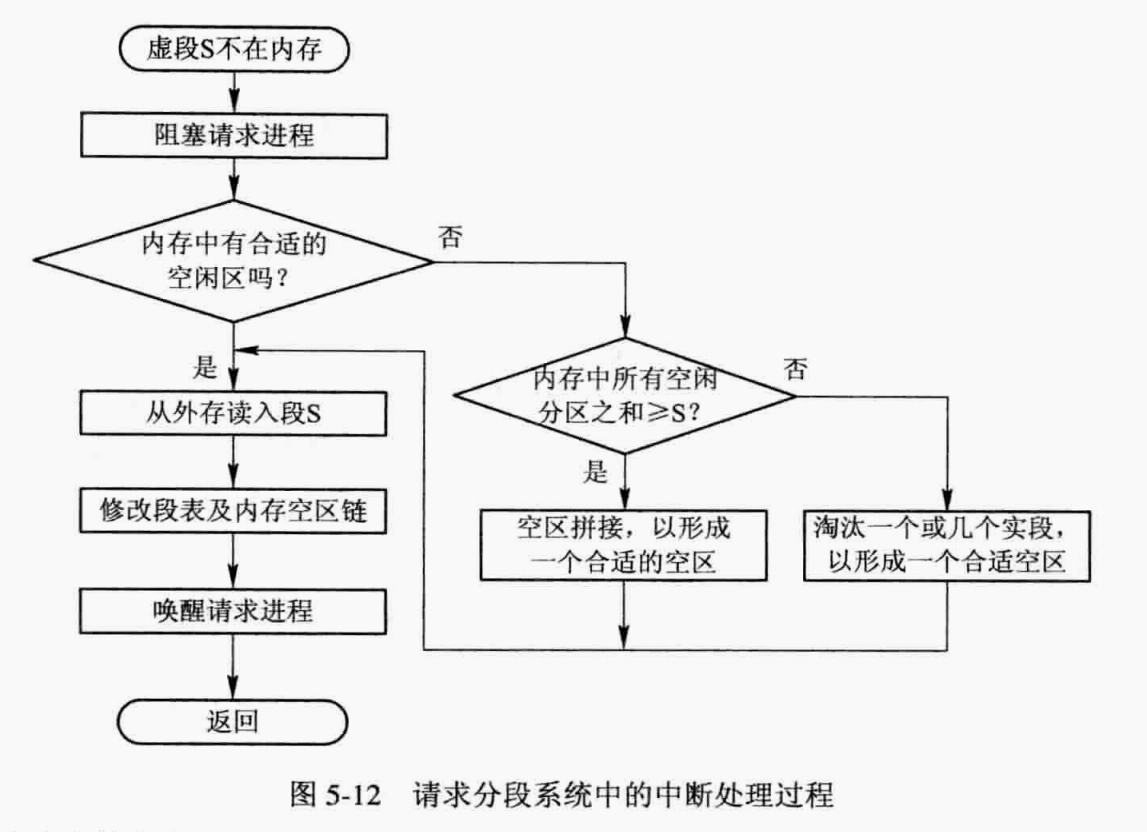
1. **5.5 请求分段存储管理方式**
   1. **5.5.1 请求分段中的硬件支持**
      1. **1. 请求段表机制**
         1. **段表项**



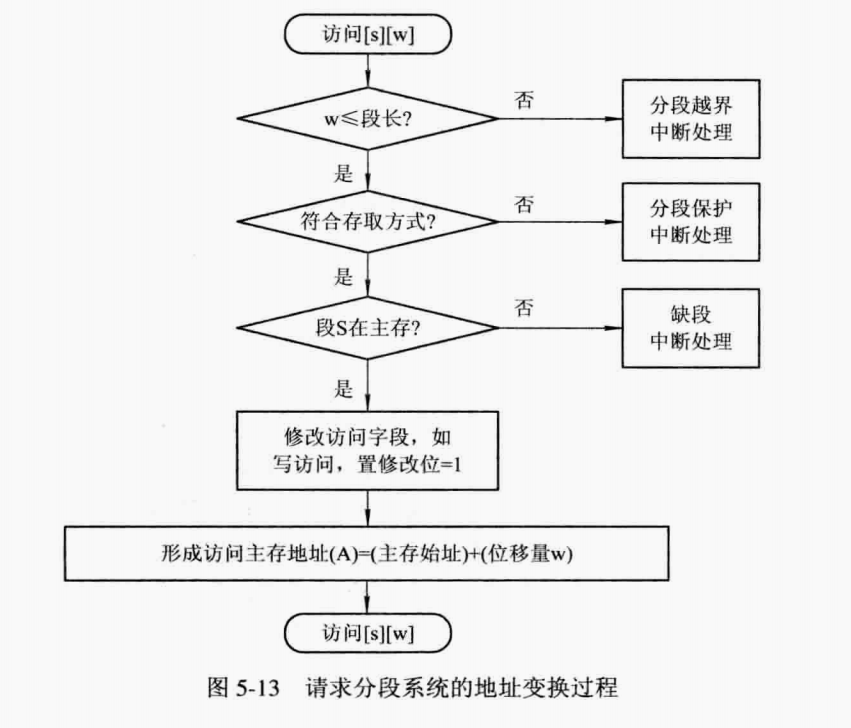
* + - * 1. 存取方式：只执行、只读、允许读写
        2. 访问字段 A：该段被访问的频繁程度
        3. 修改位 M：进入内存后是否被修改过
        4. 存在位 P：是否已调入内存
        5. **增补位：在运行过程中，是否做动态增长**
        6. 外存始址：在外存中的起始地址，起始盘块号
    1. **2. 缺段中断机构**

由于段的总大小较大，要考虑内存是否有足够的空间。没有，则需要看碎片之和是否满足，进行紧凑操作。

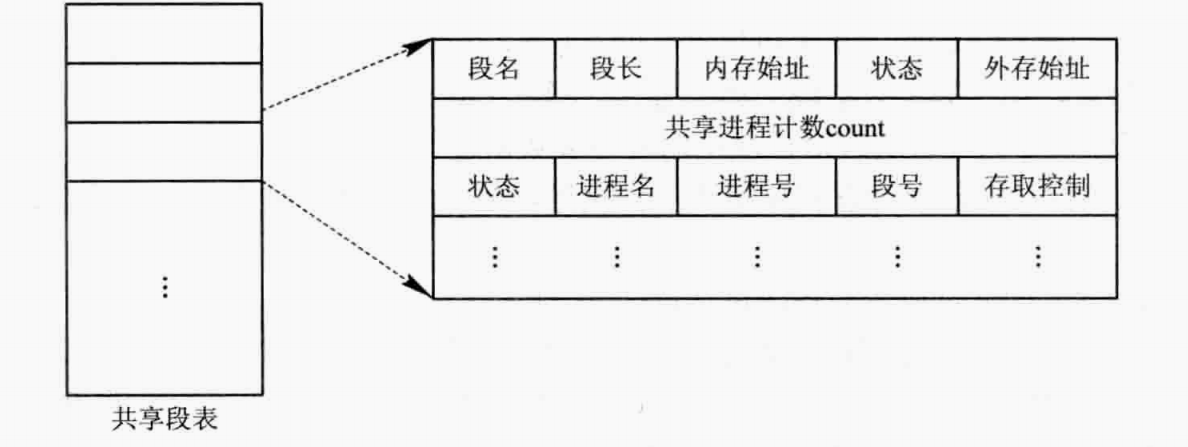
* + - 1. 图 5-12



* + 1. **3. 地址变化机构**
       1. 图5-13



* 1. **5.5.2 请求分段中的分段共享与保护（填空）**
     1. **1.共享段表**
        1. 图



* + - 1. 1）**共享进程计数count**：记录有多少进程正在共享该段。

**count = 0 才能释放，内存区**

* + - 1. 2）**存储控制字段**：读写权限
      2. 3）**段号**
    1. **2. 共享段的分配与回收**
       1. 1）分配
          1. 第一次调入，分配物理块，新增共享段表表项。
          2. 第二次调入，不分配物理块，新增表项，修改字段。
       2. 2）回收
          1. count -= 1, 若为0，则回收；否则，只是删除该表项
    2. **3. 分段保护**
       1. **1）越界检查**
          1. 地址变化机构：段表寄存器 = 段表始址 + 段表长度
          2. **中断：段号 ≥ 段表长度、段内地址 ≥ 段长**
       2. **2）存取控制检查**
          1. **“存取控制” 字段**
       3. **3） 环保护机制**
          1. 低编号的环具有高优先权
          2. 访问：相同 / 较低特权环（外环）中的数据
          3. 调用：相同 / 较高特权环（内环）中的服务

