第五章 虚拟存储器

任立勇

2016年5月

目录

5.1 虚拟存储器概述

- 虚拟存储器的引入

前面所介绍的各种存储器管理方式，出现了下面这样两种情况：

（1）有的作业很大，其所要求的内存空间超过了内存总容量，作业不能全部被装入内存，致使该作业无法运行。

（2）有大量作业要求运行，但由于内存容量不足以容纳所有这些作业，只能将少数作业装入内存让它们先运行，而将其它大量的作业留在外存上等待。

5.1.1 常规存储管理方式的特征和局部性原理

1．常规存储器管理方式的特征

（1）一次性。

作业在运行前需一次性地全部装入内存，如果一次性地装入其全部程序，也是一种对内存空间的浪费。

（2）驻留性。

作业装入内存后，便一直驻留在内存中，直至作业运行结束。尽管运行中的进程会因I／0而长期等待，或程序模块在运行过一次后就不再运行了，但仍将继续占用宝贵的内存资源。

2.局部性原理

- *程序执行的局部性原理：*程序的执行总是呈现局部性。即，在一个较短的时间段内，程序的执行仅限于某个部分；相应的，它所访问的存储空间也局限于某个区域。

- 因此，只要保证进程执行所需的部分程序和数据驻留在内存，一段时间内进程都能顺利执行。

- 局限性又表现在下述两个方面：

(1) 时间局限性。

  – 如果程序中的某条指令一旦执行， 则不久以后该指令可能再次执行；如果某数据被访问过， 则不久以后该数据可能再次被访问。

  – 产生时间局限性的典型原因，是由于在程序中存在着大量的循环操作。

- (2) 空间局限性。
  - 一旦程序访问了某个存储单元，在不久之后，其附近的存储单元也将被访问;
  - 程序在一段时间内所访问的地址，可能集中在一定的范围之内，其典型情况便是程序的顺序执行。

实现虚拟存储的一般过程

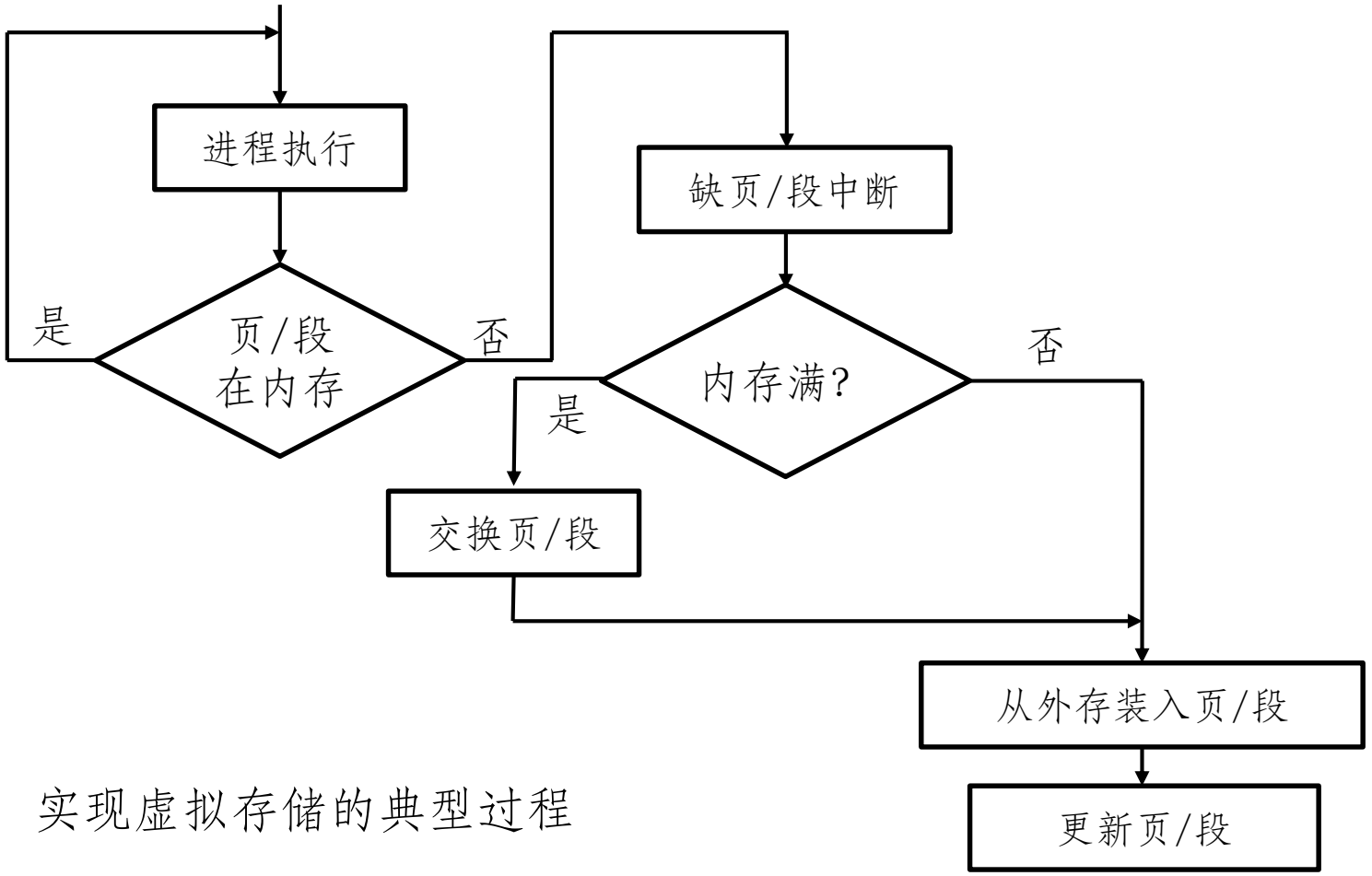
- 进程运行之前，仅需要将一部分页面或段装入内存，便可启动运行，其余部分暂时保留在磁盘上。
- 进程运行时，如果它所需要访问的页面（段）已经装入内存，则可以继续执行下去；
- 如果其所需要访问的页面（段）尚未装入内存，则发生缺页（段）中断，进程阻塞。
- 此时，系统将启动请求调页（段）功能，将进程所需的页（段）装入内存。

实现虚拟存储的一般过程

- 如果当前内存已满，无法装入新的页（段），则还需要利用页（段）置换功能，将内存中暂时不用的页（段）交换到磁盘上，以腾出足够的内存空间。

- 再将进程所需的页（段）装入内存，唤醒阻塞的进程，使之重新参与调度执行。

实现虚拟存储的典型过程



实现虚拟存储的典型过程

5.1.2 虚拟存储器的定义和特征

1．虚拟存储器定义

所谓虚拟存储器：是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一种存储器系统，其逻辑容量由内存容量和外存容量之和所决定，其运行速度接近于内存速度，而每位的成本却又接近于外存。

虚拟存储技术是一种性能非常优越的存储器管理技术，故被广泛地应用于大、 中、 小型机器和微型机中

虚拟存储器的特征

2. 虚拟存储器具有以下主要特征：

1．多次性

多次性是指一个作业被分成多次调入内存运行。

2．对换性

对换性是指作业的运行过程中进行换进、换出，换进和换出能有效地提高内存利用率。

3．虚拟性

虚拟性是指能够从逻辑上扩充内存容量，使用户所看到的内存容量远大于实际内存容量。

5.1.3 虚拟存储管理的实现方法

1、分页请求系统

- 这是在分页系统的基础上，增加了请求调页功能和页面置换功能所形成的页式虚拟存储系统。
- 允许只装入部分页面的程序（及数据），便启动运行。
- 以后，再通过调页功能及页面置换功能，陆续地把即将要运行的页面调入内存，同时把暂不运行的页面换出到外存上。
- 置换时以页面为单位。

5.1.3 虚拟存储管理的实现方法

•为了能实现请求调页和置换功能，系统必须提供必要的硬件支持和相应的软件:

（1）硬件支持。

①请求分页的页表机制上增加若干项，作为请求分页的数据结构;

②缺页中断机构:当要访问的页面尚未调入内存时，便产生缺页中断，请求调页;

③地址变换机构;

（2）实现请求分页的软件

①实现请求调页的软件

②实现页面置换的软件

5.1.3 虚拟存储管理的实现方法

2．请求分段系统

它允许只装入若干段的用户程序和数据，即可启动运行。以后再通过调段功能和段的置换功能，将暂不运行的段调出，同时调入即将运行的段。

5.1.3 虚拟存储管理的实现方法

- 为了实现请求分段，系统同样需要必要的硬件支持。一般需要下列支持：

（1）请求分段的段表机制，这是在纯分段的段表机制基础上增加若干项。
（2）缺段中断机构
（3）地址变换机构。

- 实现请求调段和段的置换功能也须得到相应的软件支持。

虚拟存储的典型问题 抖动（thrashing）

- 当进程要求装入新的页面或程序段时，如果当前没有足够的空闲空间，需要交换一些页面或段到外存。如果被交换出去的页面或段很快将被进程使用，则又需要将其换入内存。
- 如果系统花费大量的时间把程序和数据频繁地装入和移出内存而不是执行用户指令，那么，称系统出现了*抖动*。出现抖动现象时，系统显得非常繁忙，但是吞吐量很低，甚至产出为零。
- 根本原因：选择的页面或段不恰当。

5.2 请求分页存储管理方式

一、 请求分页的原理

请求分页系统是*建立在基本分页基础上*的，为了能支持虚拟存储器功能而增加了请求调页功能和页面置换功能。

二、 请求分页中的硬件支持

–具有一定容量的内存及外存的计算机系统

–页表机制

–缺页中断机构

–地址变换机构

5.2.1 请求分页中的硬件支持

(2) 页表机制

- 在请求分页系统中所需要的主要数据结构是页表。
- 在请求分页系统中的每个页表项如下所示：

页号	物理块号	状态位P	访问位A	修改位M	外存地址

- 状态位：也称存在位，标志该页是否驻留内存。
- 访问位：记录一段时间内该页被访问的情况，如一段时间内该页被访问的次数或者多长时间未被访问。
- 修改位：标记该页是否被修改过。注：为减少置换开销，通常选择未被修改过的页面置换。
- 外存地址：用于记录该页在外存上的存储地址。

5.2.1 请求分页中的硬件支持

(3) 缺页中断机构

- 在请求分页系统中，每当所要访问的页面不在内存时，便产生一缺页中断，请求OS将所缺之页调入内存。
- 地址转换时，检查页面的页表项中的存在位，如果为0，则产生一个缺页中断。

缺页中断处理过程

（1）操作系统接收到进程产生的缺页中断信号，启动中断处理例程，保留处理机现场；
（2）操作系统通知处理机从外存读取指定的页面；
（3）处理机激活I/O设备；
（4）检查内存有无足够的空闲空间装入该页面？若有，转（6），否则，执行（5）；
（5）利用页面置换算法，选择内存中的某个页面，换出内存；
（6）将指定页面从外存装入内存；
（7）更新该进程的页表；
（8）更新快表；
（9）计算物理地址。

5.2.1 请求分页中的硬件支持

- 缺页中断与一般中断的区别
  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产生多次中断

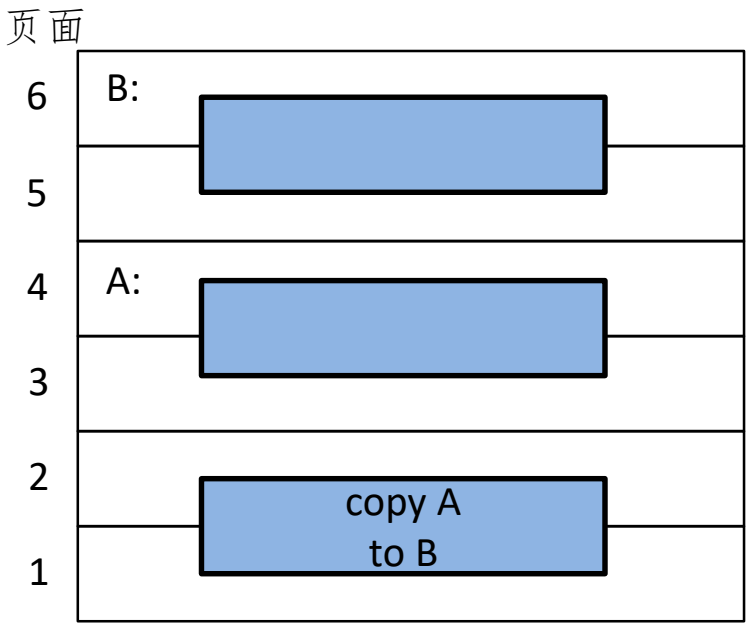
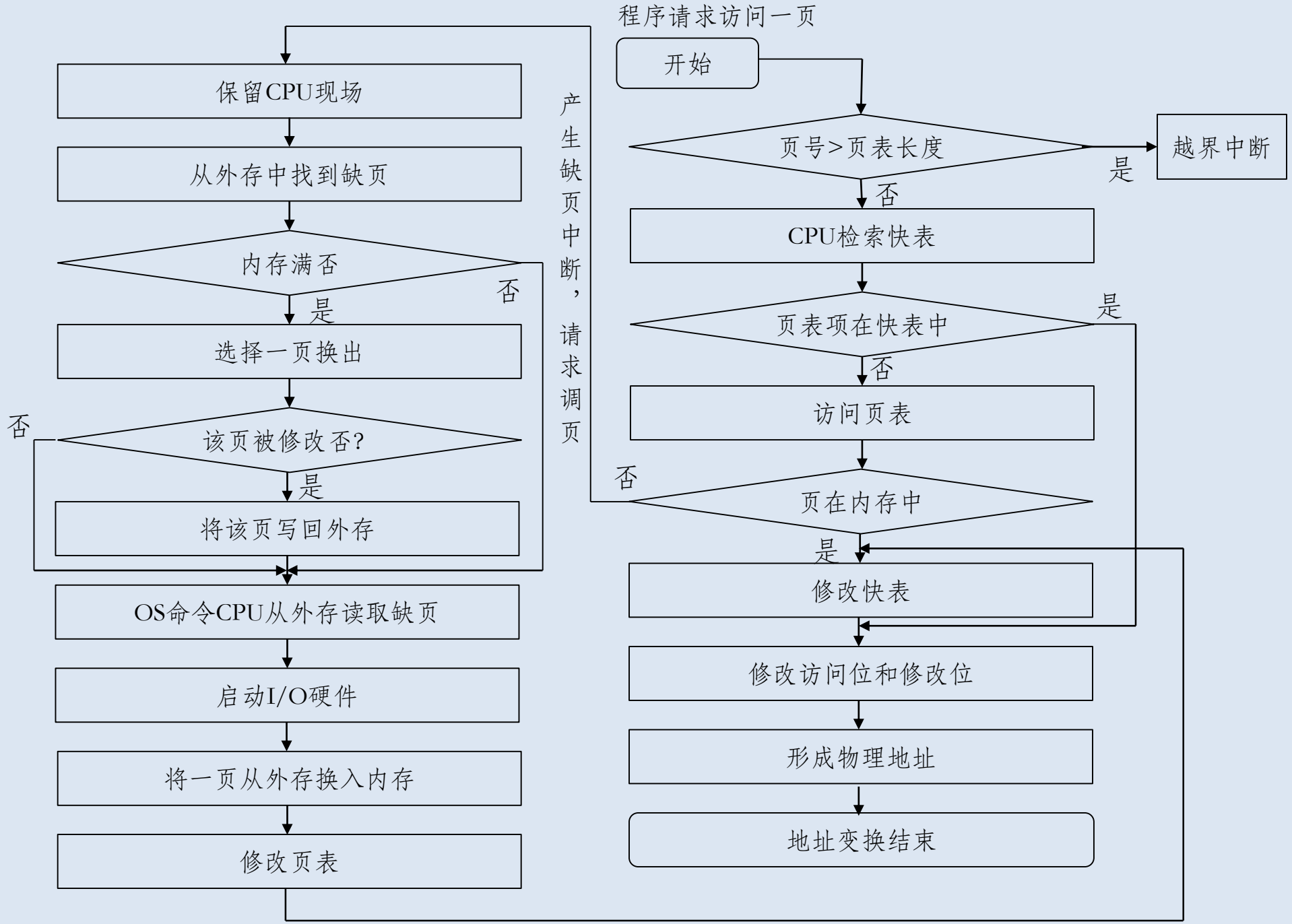


图5-1 涉及6次缺页中断的指令

5.2.1 请求分页中的硬件支持

(3) 地址变换机构

- 请求分页系统中的地址变换机构，是在分页系统地址变换机构的基础上，为实现虚拟存储器而增加了某些功能而形成的。


程序请求访问一页
开始
页号>页表长度
是
越界中断
否
CPU检索快表
页表项在快表中
是
否
访问页表
页在内存中
否
是
修改快表
修改访问位和修改位
形成物理地址
地址变换结束
产生缺页中断，请求调页
保留CPU现场
从外存中找到缺页
内存满否
否
是
选择一页换出
该页被修改否?
否
是
将该页写回外存
OS命令CPU从外存读取缺页
启动I/O硬件
将一页从外存换入内存
修改页表

5.2.2 请求分页中的内存分配

1. 最小物理块数的确定

- 这里所说的最小物理块数，是指能保证进程正常运行所需的最小物理块数。
- 对于某些简单的机器，若是单地址指令且采用直接寻址方式，则所需的最少物理块数为2:
  - 一块是用于存放指令的页面
  - 另一块是用于存放数据的页面。

  注明：如果该机器允许间接寻址，则至少要求有三个物理块。

5.2.2 请求分页中的内存分配

2. 物理块的分配策略

- 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

  2) 可变分配全局置换

  3) 可变分配局部置换

5.2.2 请求分页中的内存分配

1）固定分配局部置换

- 固定分配是指为每个进程分配一定数目的物理块，在整个运行期间都不再改变。
- 局部置换是如果进程在运行中发现缺页，则只能从该进程的*n*个页面中选出1页换出
- 实现这种策略的困难在于：应为每个进程分配多少个物理块难以确定。
  - 若太少，会频繁地出现缺页中断，降低了系统的吞吐量；
  - 若太多，又必然使内存中驻留的进程数目减少，进而可能造成CPU空闲或其它资源空闲的情况，而且在实现进程对换时，会花费更多的时间。

5.2.2 请求分页中的内存分配

2）可变分配全局置换（常用方式）

- 可变分配：在采用这种策略时，先为系统中的每个进程分配一定数目的物理块，可根据情况做适当增加或减少，而OS自身也保持一个空闲物理块队列。
- 全局置换：当某进程发现缺页时，由系统从空闲物理块队列中，取出一个物理块分配给该进程，并将欲调入的（缺）页装入其中。
- 这样，凡产生缺页（中断）的进程，都将获得新的物理块；
- 仅当空闲物理块队列中的物理块用完时， OS才能从内存中选择一页调出，该页可能是系统中任一进程的页，这样，自然又会使那个进程的物理块减少，进而使其缺页率增加。

5.2.2 请求分页中的内存分配

3）可变分配局部置换

- 为每个进程分配一定数目的物理块，但当某进程发现缺页时，只允许从该进程在内存的页面中选出一页换出，这样就不会影响其它进程的运行。
- 在进程运行过程中统计进程的缺页率，如果缺页率高，则为其增加一定的内存页，否则适当减少其内存的页面数。
- 当需要置换时只从本进程的内存页中选择，但此方式实现复杂，对进程的缺页情况的统计需要额外的开销。

5.2.2 请求分页中的内存分配

3．物理块分配算法

在采用固定分配策略时，如何将系统中可供分配的所有物理块分配给各个进程，可采用下述几种算法。

1）平均分配算法

将系统中所有可供分配的物理块，平均分配给各个进程。

2）按比例分配算法

根据进程的大小按比例分配物理块的算法。如果系统中共有n个进程，每个进程的页面数为S_i，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b_i，将有：

$b_i = \frac{s_i}{s} * m$ ，b_i应该取整，它必须大于最小物理块数。

3）考虑优先权的分配算法

通常采取的方法是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，适当地增加其相应份额后，分配给各进程。

5.2.3 页面调入策略

- 解决的问题：

1. 系统应当在*何时*把一个页面装入内存？
2. 从何处调入页面？
3. 页面调入过程？

4. 页面置换算法？

1.系统应当在*何时*把一个页面装入内存？

- 预调页 (Prepaging)
- 请求调页 (Demand Paging)

预调页方式

- 可采用一种以预测为基础的预调页策略，将那些预计在不久之后便会被访问的页面，预先调入内存。
- 处理过程：
  - 当进程创建时，预先为进程装入多个页面。
  - 缺页中断时，系统为进程装入指定的页面以及与之相邻的多个页面。
- 若局部性很差，预先装入的很多页面不会很快被引用，并会占用大量的内存空间，反而降低系统的效率。目前预调页的成功率仅约50%。

请求调页

- 仅当进程执行过程中，通过检查页表发现相应页面不在内存时，才装入该页面。
- 采用请求调页方式，一次装入请求的一个页面，磁盘I/O的启动频率较高，系统的开销较大。
- 当进程刚开始执行时，由于预先未装入进程的页面，故需要频繁地申请装入页面。执行一段时间以后，进程的缺页率将下降。

2.从何处调入页面？

- 在请求分页系统中的外存分为两部分：
  - 用于存放文件的文件区
  - 用于存放对换页面的对换区。
- 通常，对换区的磁盘I/O速度比文件区的高。这样，每当发生缺页请求时，系统应从何处将缺页调入内存，可分成如下三种情况。

2.从何处调入页面？

（1）系统拥有足够的对换区空间，

- 进程运行前需将全部有关文件从文件区拷贝到对换区。
- 这时可以全部从对换区调入所需页面，以提高调页的速度。

（2）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，由于它们未被修改而不必再将它们换出（直接丢弃），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。

（3）UNIX方式。

- 由于与进程有关的文件都放在文件区，应从文件区调入。
- 凡是未运行过的页面，都应从文件区调入。
- 而对于曾经运行过但又被换出的页面，由于是被放在对换区，因此在下次调入时，应从对换区调入。
- 允许页面共享

3.页面调入过程

①每当程序所要访问的页面未在内存时，便向CPU发出一缺页中断。

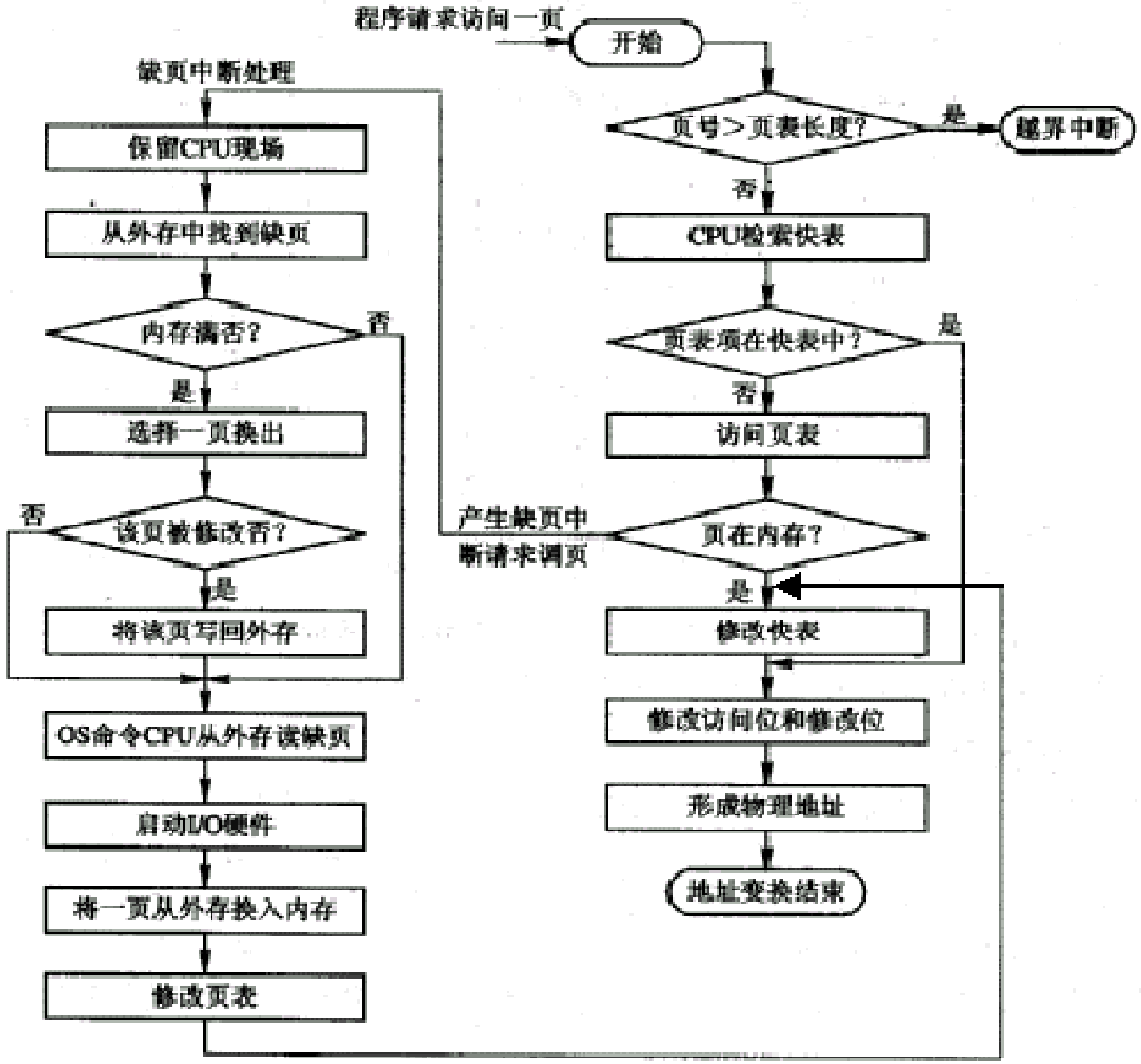
②中断处理程序首先保留CPU环境，分析中断原因后，转入缺页中断处理程序。

③如果内存已满，则须先按照某种置换算法从内存中选出一页准备换出；如果此页已被修改，则必须将它写回磁盘。

④然后再把所缺的页调入内存，并修改页表中的相应表项，置其存在位为“1”，并将此页表项写入快表中。

⑤形成所要访问数据的物理地址，再去访问内存数据。

- 整个页面的调入过程对用户是透明的。


程序请求访问一页
开始
页号>页表长度?
是
越界中断
否
CPU检索快表
页表项在快表中?
是
否
访问页表
页在内存?
产生缺页中
断请求调页
是
修改快表
修改访问位和修改位
形成物理地址
地址变换结束
缺页中断处理
保留CPU现场
从外存中找到缺页
内存满否?
否
是
选择一页换出
该页被修改否?
否
是
将该页写回外存
OS命令CPU从外存读缺页
启动I/O硬件
将一页从外存换入内存
修改页表

5.3 页面置换算法

- 定义：选择换出页面的算法
- 直接影响系统的性能。好的页面置换算法，应具有较低的页面更换频率。

1）最佳(优)置换算法

2）先进先出（FIFO）页面置换算法

3）最近最久未使用（LRU）置换算法

4）Clock置换算法

5）改进型Clock置换算法

6）其它置换算法

5.3.1 最佳页面置换算法

最佳(优)置换算法（Optimal)

- 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。
- 最佳置换算法是一种理想化的算法，它具有最好的性能，但实际上却难于实现。

举例:

- 假定系统为某进程分配了三个物理块， 并考虑有以下的页面访问序列：
- 7，0，1，2，0，3，0，4，2，3，0，3，2，1，2，0，1，7，0，1
- 进程运行时， 先将7，0，1三个页面装入内存。
- 以后，当进程要访问页面2时， 将会产生缺页中断。此时OS根据最佳置换算法，将选择页面7予以淘汰。（总共6次）

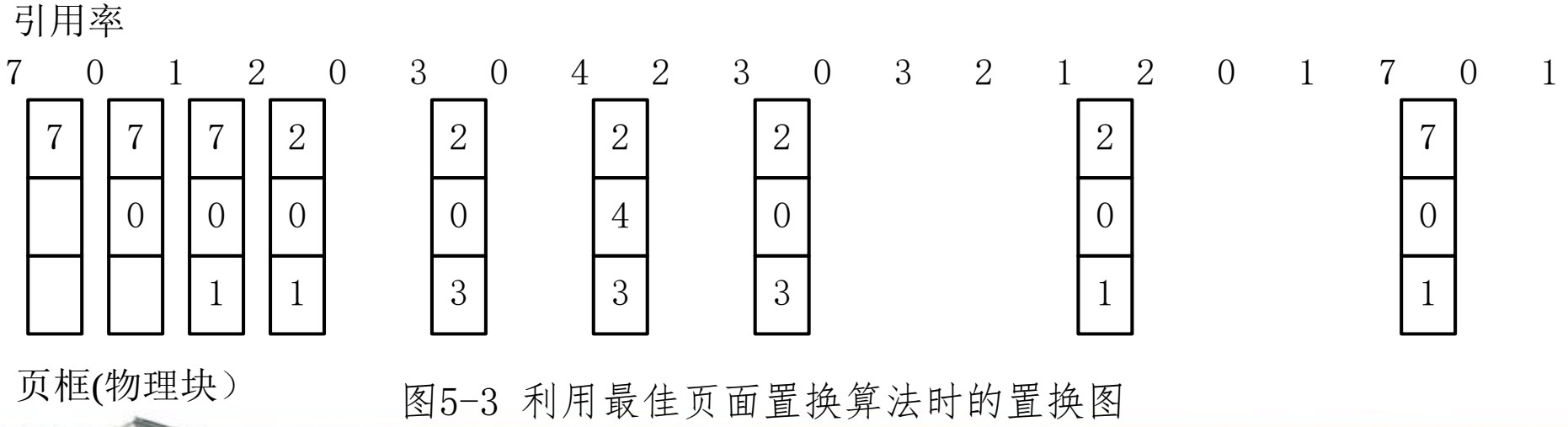


图5-3 利用最佳页面置换算法时的置换图

5.3.1 先进先出（FIFO）页面置换算法

- 该算法总是淘汰最先进入内存的页面，即选择在内存中驻留时间最久的页面予以淘汰 。（总共12次）

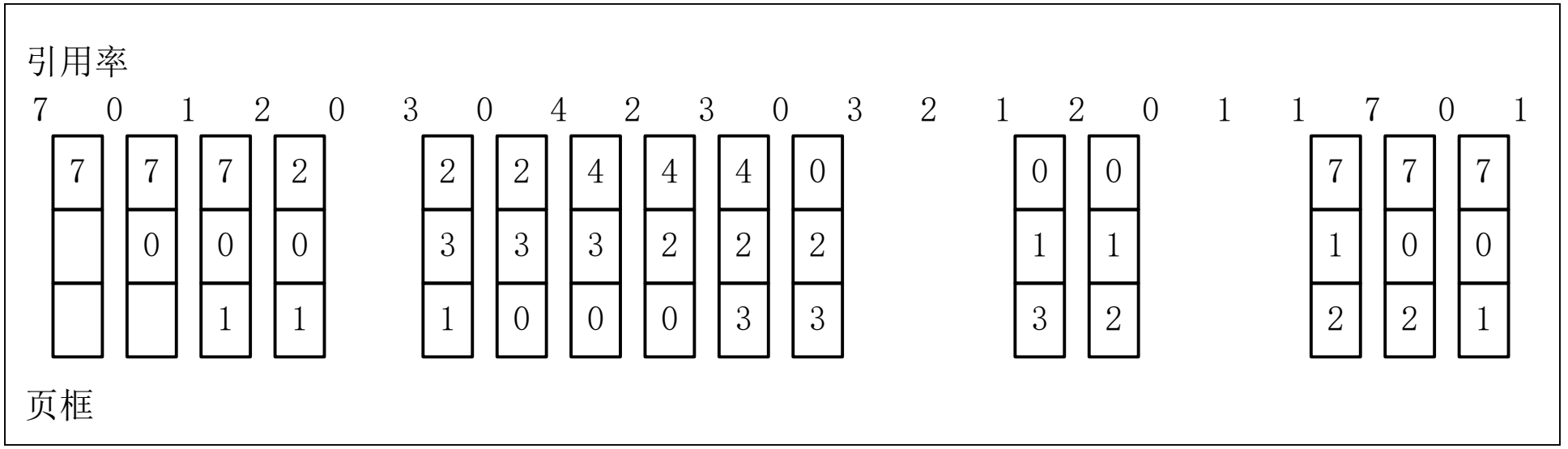


图5-4 利用FIFO置换算法时的置换图

5.3.2 最近最少使用（LRU）置换算法

- LRU置换算法是选择最近最少使用的页面予以淘汰。该算法赋予每个页面一个访问字段，用来记录一个页面自上次被访问以来所经历的时间t，当须淘汰一个页面时，选择现有页面中其t值最大的，即最近最久未使用的页面予以淘汰。

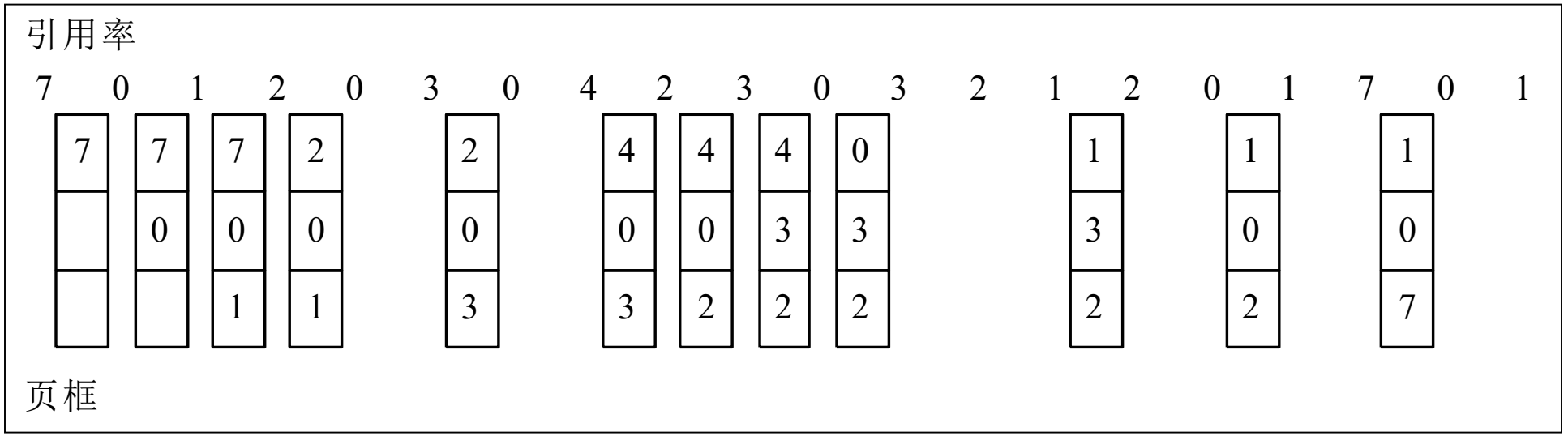


图5-5 利用LRU页面置换算法时的置换图

最近的过去推测最近的将来

LRU置换算法的硬件支持

- LRU置换算法虽然是一种比较好的算法。但要求系统有较多的支持硬件，
  - 为了了解一个进程在内存中的各个页面各有多长时间未被进程访问，
  - 以及如何快速地知道哪一页是最近最久未使用的页面，
- 须有以下两类硬件之一的支持：
  - 1）寄存器
  - 2）栈

l）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：

$$R=R_{n-1}R_{n-2}R_{n-3}\cdots R_2R_1R_O$$

当进程访问某物理块时，要将相应寄存器的最高位R_{n-1}位置成1。系统每隔一定时间（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。

实页 \ R	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
1	0	1	0	1	0	0	1	0
2	1	0	1	0	1	1	0	0
3	0	0	0	0	0	1	0	0
4	0	1	1	0	1	0	1	1
5	1	1	0	1	0	1	1	0
6	0	0	1	0	1	0	1	1
7	0	0	0	0	0	1	1	1
8	0	1	1	0	1	1	0	1

图5-6 某进程具有8个页面时的LRU访问情况

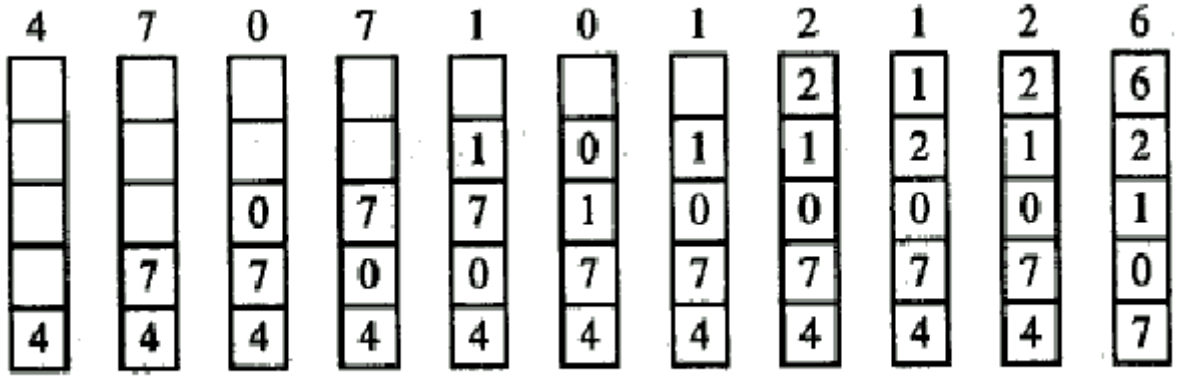
2）栈

- 可利用一个特殊的栈来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。
- 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。

举例

- 假定现有一进程所访问的页面的页面号序列为：4，7，0，7，1，0，1，2，1，2，6

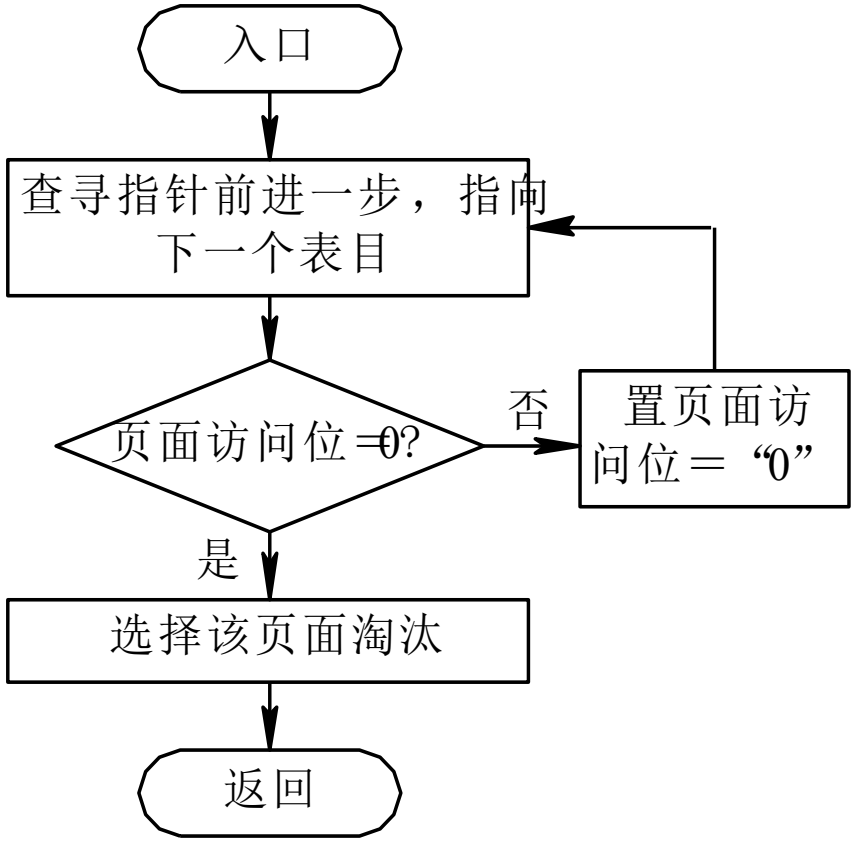
随着进程的访问，栈中页面号的变化情况 ·



5.3.3 Clock置换算法

1．简单的Clock置换算法

- 当采用简单c1ock算法时，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。
- 当某页被访问时，其访问位被置1。
- 置换程序从上次停止位置开始检查页面的访问位。
  - 如果是0，就选择该页换出；
  - 若为1，则重新将它置0，暂不换出，而给该页第二次驻留内存的机会。
- 由于该算法是循环地检查各页面的使用情况，故称为c1ock算法。
- 置换时是将未使用过的页面换出去，故又把该算法称为最近未用算法NRU。



块号	页号	访问位	指针
0			
1			
2	4	0	
3			
4	2	1	
5			
6	5	0	
7	1	1	

替换指针

图5-8 简单Clock置换算法的流程图和示例

改进型Clock置换算法

- 系统把一个页面移出内存时，如果该页面驻留内存期间没有被修改过，那么不必把它写回辅存，否则系统必须把它写回辅存。
- 换出未修改过的页面比换出被修改过的页面开销小。
- 显然，我们可以依据上述结论改进CLOCK算法。改进后的CLOCK算法将在置换范围内*首选*在最近没有被使用过、在驻留内存期间没有被修改过的页面作为被置换页面。

改进型Clock置换算法

- 由访问位A和修改位M可以组合成下面四种类型的页面：
  1类（A=0，M=0：表示该页最近既未被访问，又未被修改，是最佳淘汰页。
  2类（A=0，M=1）：表示该页最近未被访问，但已被修改，并不是很好的淘汰页。
  3类（A=1，M=0）：最近已被访问，但未被修改：该页有可能再被访问。
  4类（A=1，M=1）：最近已被访问且被修改，该页可能再被访问。

改进型Clock置换算法

- 执行过程可分成以下三步：
  （1）从指针所指示的当前位置开始，扫描循环队列，寻找A=0且M=0的第一类页面，将所遇到的第一个页面作为所选中的淘汰页。在第一次扫描期间不改变访问位A。

  （2）如果第一步失败，即查找一周后未遇到第一类页面，则开始第二轮扫描，寻找A=0且M＝1的第二类页面，将所遇到的第一个这类页面作为淘汰页。在第二轮扫描期间，将所有扫描过的页面的访问位都置0。

  （3）如果第二步也失败，亦即未找到第二类页面，则将指针返回到开始的位置，并将所有的访问位复0。然后重复第一步，如果仍失败，必要时再重复第二步，此时就一定能找到被淘汰的页 .

5.3.4 页面缓冲算法

1、影响页面换进换出效率的若干因素

（1）页面置换算法

（2）写回磁盘的频率。可考虑建立一个（已修改）换出页面的链表，仅当被换出页面数目达到一定值时，在将它们一起写回磁盘，从而减少磁盘I/O次数

（3）读入内存频率。如果进程访问上述链表中的某一页面，则无需执行磁盘读入，而直接从链表中获取。

5.3.4 页面缓冲算法

2、页面缓冲算法（PBA: Page Buffering Algorithm)

- 显著降低页面换进换出频率。
- 即可采用类似LRU和Clock算法改善分页系统的性能，又可采用类似FIFO采用简单的置换策略
- VAX/VMS操作系统

① 空闲页面链表

② 修改页面链表

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

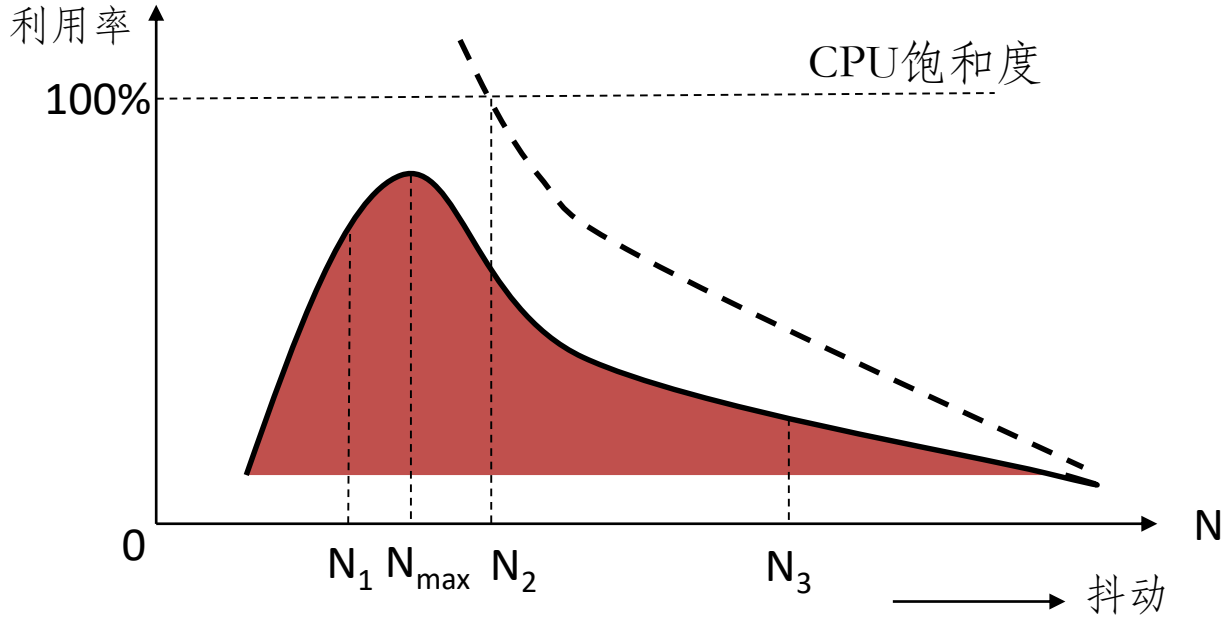


图5-9 处理机的利用率

5.4 “抖动”与工作集

2. 产生“抖动”的原因

- 发生抖动的根本原因是，同时在系统中运行的进程太多，由此分配给每个进程的物理块太少，不能满足进程正常运行的基本要求，致使每个进程在运行时，频繁地出现缺页，造成每个进程的大部分时间都用于页面的换进换出，从而导致处理机的利用率急剧下降并趋于0.

5.4.2 工作集

- 进程发生缺页率与进程所获得的物理块数有关
- “局部性原理”表明，程序在一段时间内仅局限较少的页面，在另一段时间内，又可能仅局限于对另一些较少的页面进行访问。

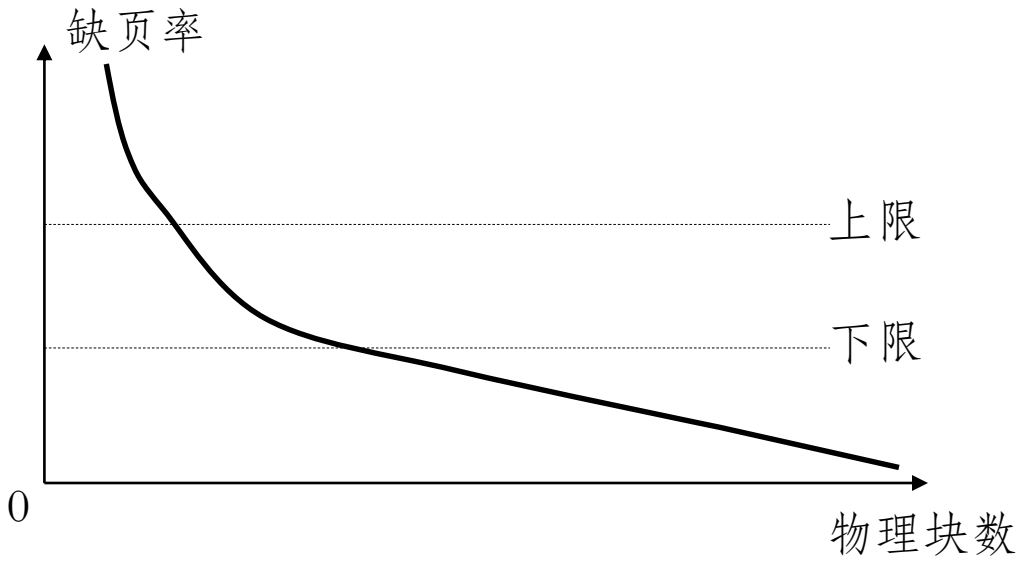


图5-10 缺页率与物理块数之间的关系

5.4.2 工作集

- 所谓工作集，指在某段时间间隔内△，进程实际所要访问页面的集合。
- 为较少产生缺页，应将程序的全部工作集装入内存。进程在时间t的工作集记为$w(t,\Delta)$，其中△称为工作集的窗口尺寸。
- 由此可将工作集定义为，进程在时间间隔（$t-\Delta,t$）中引用页面的集合

5.4.3 “抖动”的预防方法

1. 采取局部置换策略。将“抖动”造成的影响限制在较小的范围
2. 把工作集算法融入到处理机调度中。在调度新作业之前，检查当前驻留进程的页面是否足够多，判断是否会因为调入新作业导致缺页率增加
3. 利用“L=S”准则调节缺页率。其中L是缺页之间的平均时间，S是平均缺页服务时间，即置换一个页面所需的时间。
4. 选择暂停的进程。选择暂停或者优先级低的进程，将其页面换出内存，降低多道程序度，从而预防“抖动”。

5.5 请求分段存储管理方式

- 在请求分段系统中，程序运行之前，只需先调入若干个分段（不必调入所有的分段），便可启动运行。当所访问的段不在内存中时，可请求OS将所缺的段调入内存。
- 所需硬件支持
  - 段表
  - 缺段中断机构
  - 地址变换机构

5.5.1 请求分段中的硬件支持

1．段表机制

在请求分段式管理中所需的主要数据结构是段表，段表项有新的扩展。

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

增补位：表示本段在运行过程中是否做过动态增长

2. 缺段中断机构

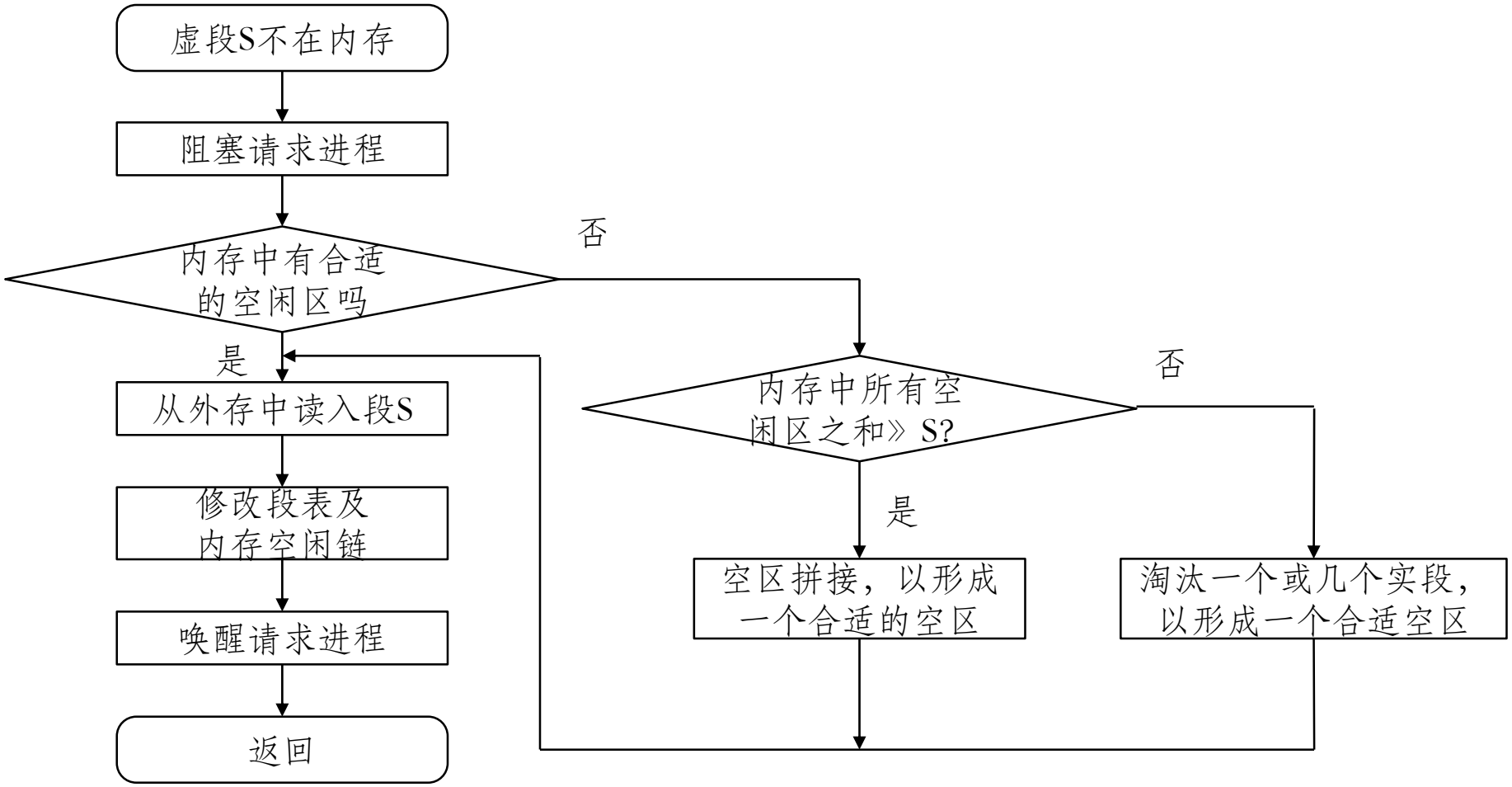


图5-12 请求分段系统中的中断处理过程

3. 地址变换机构

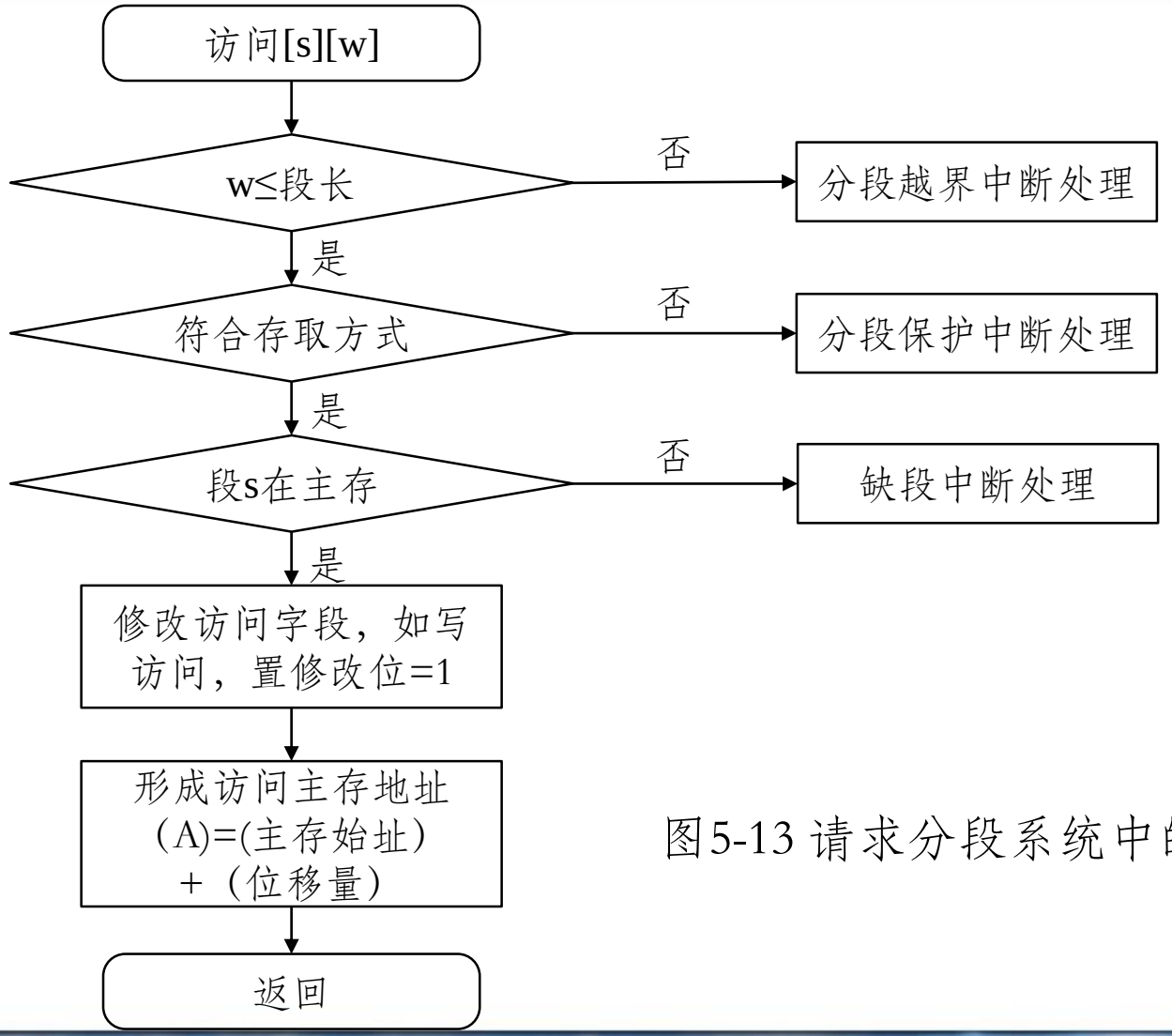


图5-13 请求分段系统中的地址变换过程

5.5.2 分段的共享与保护

1．共享段表

为了实现分段共享，可在系统中配置一张共享段表，所有各共享段都在共享段表中占有一表项。

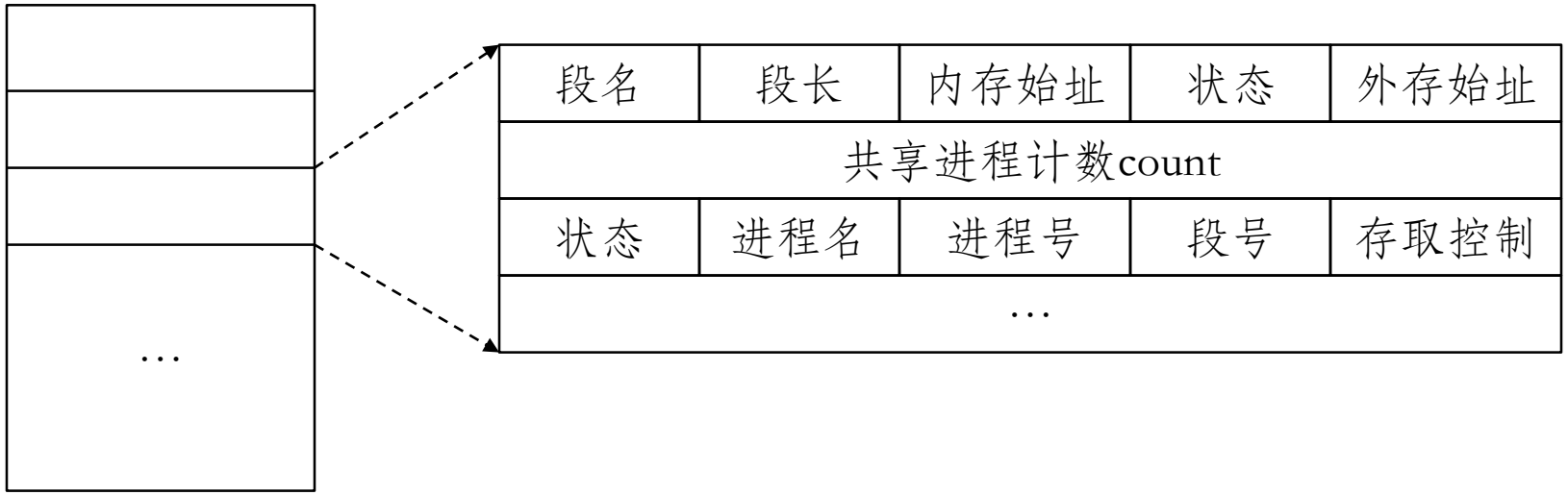


图5-14 共享段表项

5.5.2 分段的共享与保护

2．共享段的分配与回收

1）共享段的分配：

① 在为共享段分配内存时，对第一个请求使用该共享段的进程，由系统为该共享段分配一物理区，再把共享段调入该区，

② 同时将该区的始址填入请求进程的段表的相应项中，

③ 还须在共享段表中增加一表项，填写有关数据，把count置为1；

④ 之后，当又有其它进程需要调用该共享段时，只需在调用进程的段表中增加一表项，填写该共享段的物理地址；

⑤ 在共享段的段表中，填上调用进程的进程名、存取控制等，再执行count：＝count＋1操作，以表明有两个进程共享该段。

5.5.2 分段的共享与保护

2）共享段的回收

1. 当共享此段的某进程不再需要该段时，应将该段释放，包括
   ① 撤出在该进程段表中共享段所对应的表项，
   ② 取消调用者进程在共享段表中的有关记录。
   ③ 以及执行count：=count-1操作。
2. 若count结果为0，表明此时已没有进程使用该段，则须
   ① 由系统回收该共享段的物理内存，
   ② 以及取消在共享段表中该段所对应的表项；

5.5.2 分段的共享与保护

3．分段保护

分段系统中由于每个分段在逻辑上是独立的，因而比较容易实现信息保护。

常采用以下几种措施，来确保信息的安全：

1）越界检查：寄存器中放有段表长度信息，将逻辑地址空间的段号与段表长度进行比较，如果

① 段号等于或大于段表长度，将发出地址越界中断信号；

② 段内地址等于或大于段长也将产生地址越界中断信号；

5.5.2 分段的共享与保护

2）存取控制检查

段表的每个表项中设置“存取控制”字段

（1）只读 （2）只执行 （3）读 / 写

3）环保护机构

在该机制中规定：低编号的环具有高优先权。 OS核心处于0环内；某些重要的实用程序和操作系统服务，占居中间环；而一般的应用程序，则被安排在外环上

在环系统中，程序的访问和调用应遵循以下规则：

① 一个程序可以访问驻留在相同环或较低特权环中的数据。

② 一个程序可以调用驻留在相同环或较高特权环中的服务。

第5章 作业

1. 虚拟存储器有哪些特征？其中最本质的特征是什么？
2. 实现虚拟存储器需要哪几个关键技术？
3. 试比较缺页中断机构与一般的中断有何明显的区别
4. 说明请求分页系统中的地址变换过程
5. 在请求分页系统中，常采用哪几种页面置换算法
6. 影响页面换进换出的若干因素是什么？
7. 在请求分页系统中，产生抖动的原因是什么？有什么方法可防止
8. 在请求分段中的缺页中断处理过程