思考：

1.大作业（超过内存总容量）

2.大量作业（内存总容量不足以容纳所有作业）

怎么办？

第七章 虚拟存储管理

7.1 虚拟存储器的基本概念

7.1.1 引入背景

1. 常规存储器管理方式的特征

(1) 一次性。

(2) 驻留性。

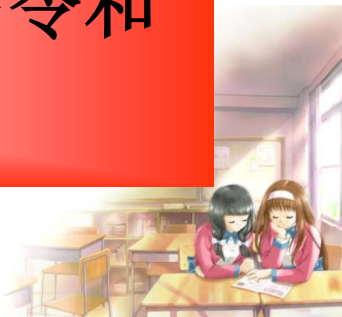
2. 局部性原理

局部性原理(principle of locality)：指程序在执行过程中的一个较短时期，所执行的指令地址和指令的操作数地址，分别局限于一定区域。

可以表现为：

时间局部性：一条指令的一次执行和下次执行，一个数据的一次访问和下次访问都集中在一个较短时期内；

空间局部性：程序在一段时间内所访问的指令和数据的地址，可能集中在一定的范围之内。



3. 虚拟存储的基本原理

- 在程序装入时，不必将其全部读入内存，只需将当前需要执行的部分页或段读入内存，就可让程序开始执行。
- 在程序执行过程中，如果需执行的指令或访问的数据尚未在内存（称为缺页或缺段），则由处理器通知操作系统将相应的页或段调入到内存，然后继续执行程序。
- 操作系统将内存中暂时不用的页或段调出保存在外存上，从而腾出空间存放将要调入的页或段。

总之，只需程序的一部分在内存就可执行。

7.1.2 虚拟存储器

1. 虚拟存储器的定义

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。

2. 虚拟存储器的特征

1.离散分配：物理内存分配的不连续。

2.部分装入：指在作业运行时没有必要将其全部装入，只需将当前要运行的部分代码和数据装入内存即可。部分装入是虚拟存储器最重要的特征。

3.多次交换：进程在执行过程中，允许将暂时不用的部分代码和数据从内存调到外存的交换区。待以后需要时，再从外存调到内存。

4.虚拟扩充：通过物理内存和外存相结合，提供大范围的虚拟地址空间，从而从逻辑上扩充内存容量。

3. 虚拟存储器的实现方法

（1）请求分页系统：在简单页式存储管理的基础上， 增加请求调页和页面置换功能。

1)硬件支持 。

① 请求分页的页表机制，它是在纯分页的页表机制上增加若干项而形成的；② 缺页中断机构，即每当用户程序要访问的页面尚未调入内存时，便产生一缺页中断，以请求OS将所缺的页调入内存；③ 地址变换机构，它同样是在纯分页地址变换机构的基础上发展形成的。

2) 实现请求分页的软件。

（2）请求分段系统：在简单段式存储管理的基础上，增加请求调段和段置换功能。

1)硬件支持 。

① 请求分段的段表机制，它是在纯分段的段表机制上增加若干项而形成的；② 缺段中断机构，即每当用户程序要访问的段尚未调入内存时，便产生一缺段中断，以请求OS将所缺的段调入内存；③ 地址变换机构， 它同样是在纯分段地址变换机构的基础上发展形成的。

2) 实现请求分段的软件。

（3）虚拟段页式存储系统

是虚拟页式和虚拟段式存储管理的结合。

- 存储管理的分配单位是：段，页
- 逻辑地址的组成：段号，页号，页内偏移地址。
- 地址变换：先查段表，再查该段的页表。缺段中断和缺页中断。

7.2 请求分页存储管理

7.2.1 工作原理

1. 页表机制

程序访问参考

置换页面参考

页号	物理块号	状态位P	访问字段A	修改位M	外存地址

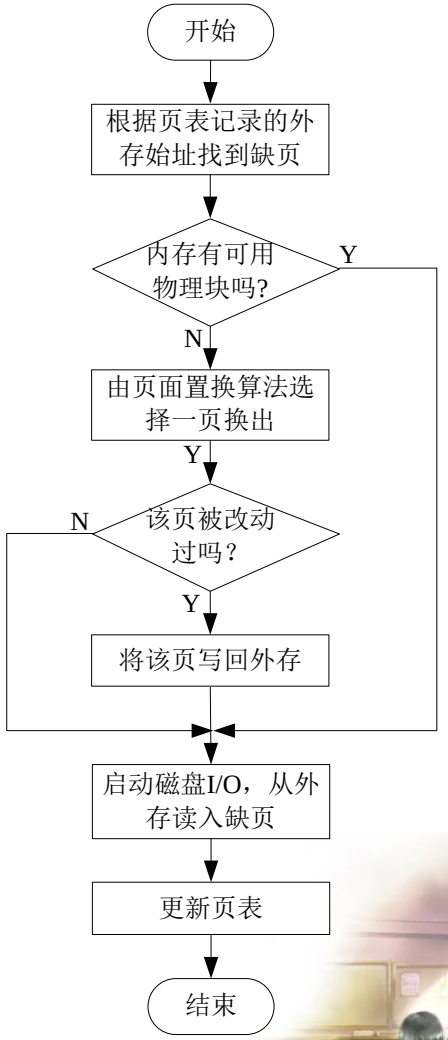
选择换出页面参考

调入该页时参考

7.2.1 工作原理

2. 设置缺页中断机构

- 当访问页表的状态位，发现页不在内存时，便产生一个缺页中断。缺页中断处理过程如下：
  - 根据页表中记录的缺页在外存的起始地址，到外存中找到相应的页。
  - 若内存有可用的物理块，则启动磁盘I/O 将该页调入内存；
  - 若内存中没有可用的物理块，则还需根据页面置换算法淘汰一些页，若淘汰的页曾做过改动，还需将此页重写回外存，最后将缺页调入内存指定的物理块。



3. 升级地址变换机构

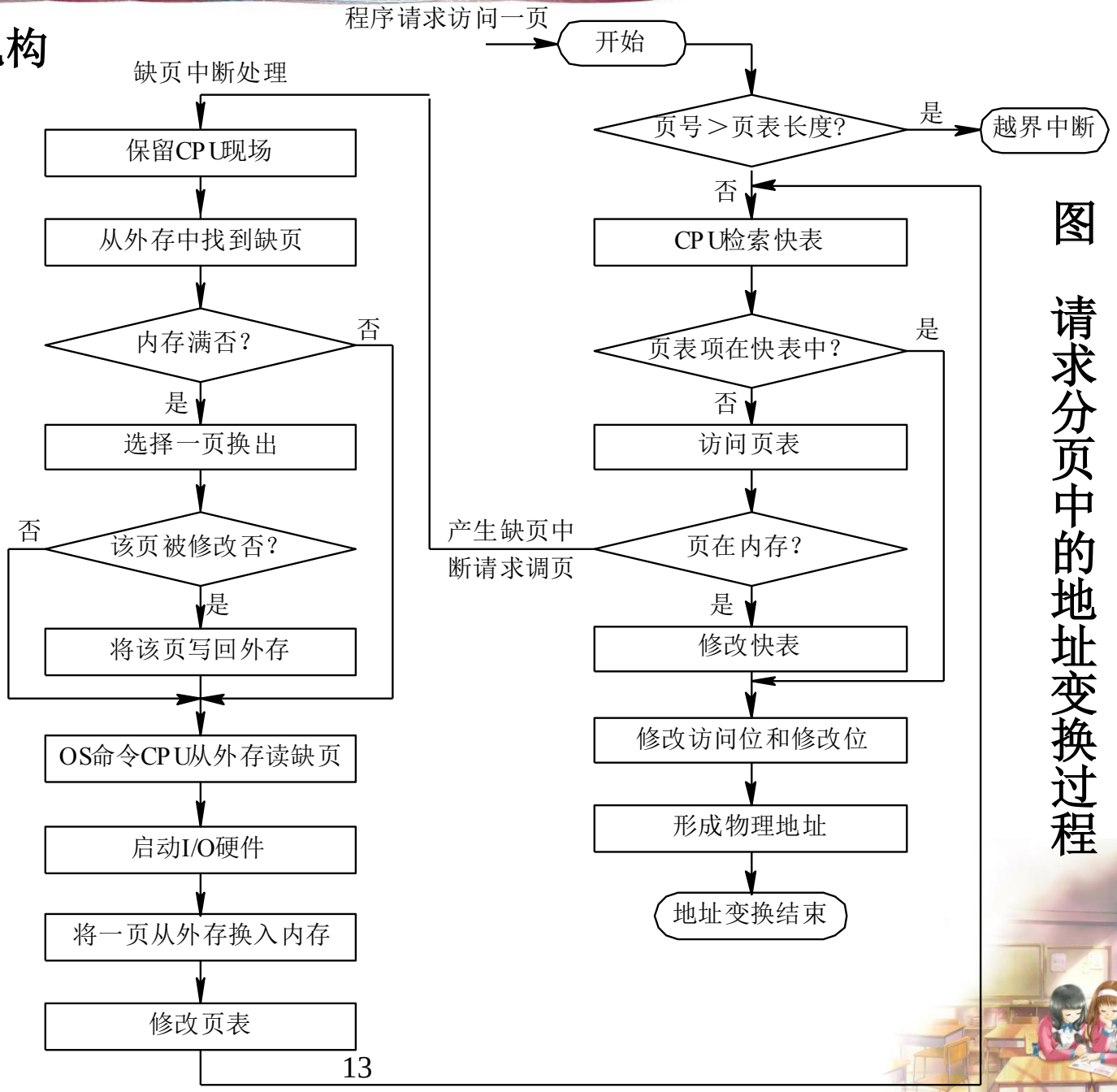


图 请求分页中的地址变换过程

7.2.2 驻留集管理

1. 驻留集大小

- 驻留集：给每个进程分配的物理块的集合。
- 驻留集的大小与缺页率有一定关系：
  - 驻留集越小，驻留内存的进程数就会越多，但相应的缺页率会相对较高。
  - 驻留集不断增加，其缺页率会逐渐降低，但根据局部性原理，驻留集超过一定的大小后（曲线的拐点附近），进程的缺页率并没有明显改善。

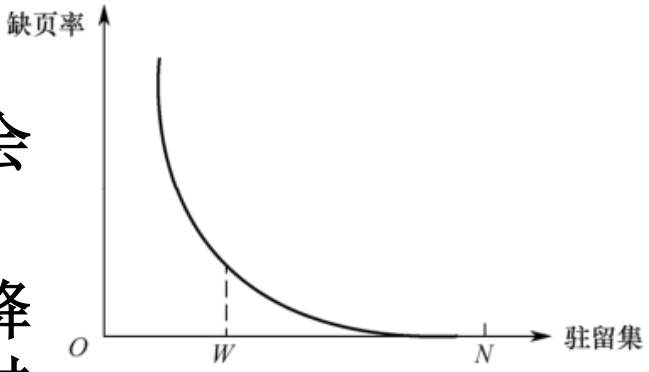


图 7-3 驻留集大小与缺页率的关系

♦ 通常为进程分配的物理块数应该取曲线的拐点附近或稍大些。

7.2.2 驻留集管理

2. 驻留集管理策略

- 给每个进程分配的物理块的数目是否固定？

  ——驻留集分配策略

- 置换范围是否局限于该进程内部？

  ——置换策略

7.2.2 驻留集管理

（1）驻留集分配策略

- 固定分配策略：为一个进程分配固定数目的内存物理块。
  - 这个数目在进程创建时确定。
  - 一旦发生缺页，只能替换该进程在内存中的某页，以保证进程占有的物理块的数目一直保持不变。
- 可变分配策略：分配给一个进程的物理块数可以变化。
  - 若缺页率高，则增加分配；若低，则减少分配。
  - 系统管理开销大。

7.2.2 驻留集管理

（2）置换策略

- 全局置换策略：发生缺页需要页面置换时，可以在驻留内存的所有页中进行选择，不管它属于哪一个进程。
- 局部置换策略：发生缺页需要页面置换时，只能从发生缺页的进程在内存的页中进行选择。

内存中加载的重要内容，如操作系统内核及用到的重要数据结构等，需要通过“锁定”排除于置换范围之外。锁定可以通过在页表和物理块表中为每个表项增加一个Lock位来实现。

7.2.2 驻留集管理

- 驻留集若采用固定分配策略，则只能组合局部置换，驻留集管理策略存在下面3 种：

表 7-2 驻留集管理策略

管理策略	局部置换策略	全局置换策略
固定分配策略	一个进程占有的物理块数是固定的，页面置换时只能置换该进程在内存的页	不可能
可变分配策略	分配给进程的物理块数是可以变化的，页面置换时只能置换该进程在内存的页	分配给进程的物理块数可以变化，页面置换时可以从内存未锁定的所有物理块中选择

  - 固定分配局部置换策略
  - 可变分配全局置换策略
  - 可变分配局部置换策略

7.2.3 调页策略

调页策略用于确定何时将进程所需的页调入内存。在虚拟页式管理中有两种常用策略。

- 请求调页(demand paging)：只调入发生缺页时所需的页面。
  - 优点：容易实现。
  - 缺点：对外存I/O次数多，开销较大。
- 预调页(prepaging)：在发生缺页需要调入某页时，一次调入该页以及相邻的几个页。
  - 优点：提高调页的I/O效率。
  - 缺点：基于预测，若调入的页在以后很少被访问，则效率低。常用于程序装入时的调页。

缺页率

在进程的运行过程中，访问页面成功的次数为S，访问页面失败的次数为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：

$$f=F/A$$

影响缺页率的因素：

(1)页面大小。

(2)进程所分配物理块的数目。

(3)页面置换算法。

(4)程序固有特性。

7.2.4 页面置换算法

- 功能：需要调入页面时，选择内存中哪个物理页面被置换。称为replacement policy。
- 目标：把未来不再使用的或在较长时间内不会再访问的页面调出，通常只能在局部性原理指导下依据过去的统计数据进行预测；

常用于评价其他算法

1. 最佳页面(Optimal)置换算法

由Belady于1966年提出的一种理论上的算法。其所选择被淘汰页面，将是以后永不使用的，或在最长(未来)时间内不再被访问的页面。通常可保证获得最低的缺页率。但这是一种理想情况，是实际执行中无法预知的，因而不能实现。

1．最佳页面置换算法

- 页面访问串/页面访问流：进程执行时对页面的访问序列。
- 假设系统采用固定分配局部置换策略，系统为进程分配了3 个物理块，若该进程执行时的页面访问串为“5,0,1,2,0,4,0,7,2,4,0,4,2,1,2,0,1,5,0,1”，采用OPT 页面置换算法的置换结果如图所示。

5	0	1	2	0	4	0	7	2	4	0	4	2	1	2	0	1	5	0	1
5	5	5	2		2		2			2			2				5		
	0	0	0		0		7			0			0				0		
		1	1		4		4			4			1				1		

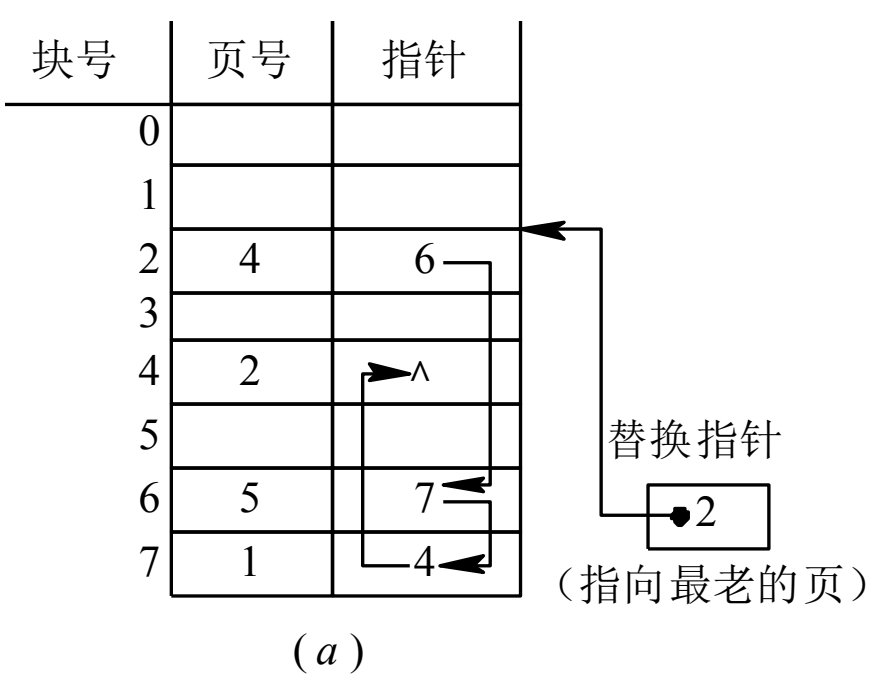
图 7-4 OPT 页面置换算法的置换结果

- 采用请求调页策略和OPT 页面置换算法，共发生了9 次缺页，其中6 次页面置换，进程访问的总页面数为20，缺页率为45%。

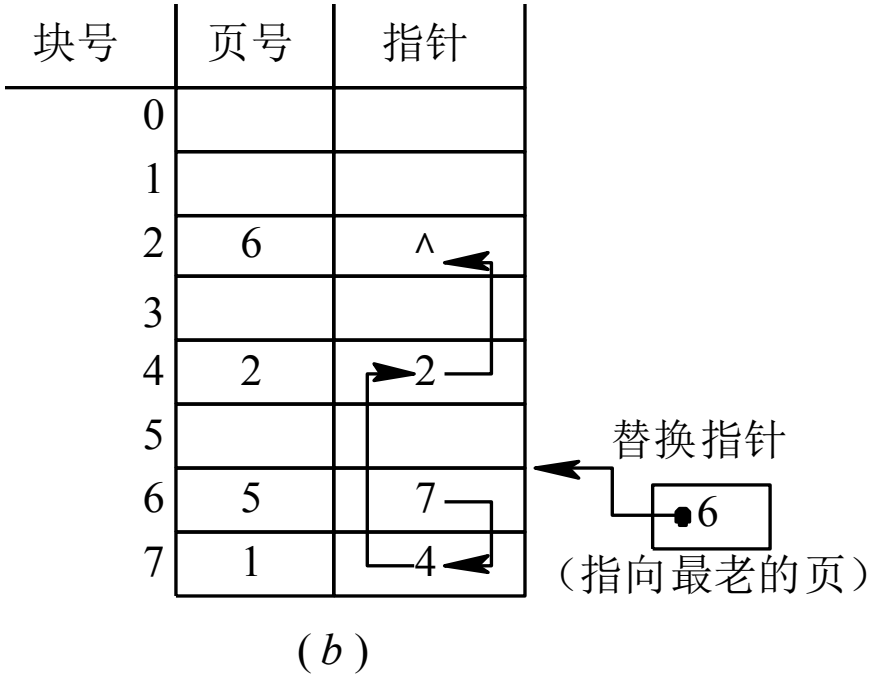
2. 先进先出(FIFO)页面置换算法

基本思想是，总是先淘汰那些驻留在内存时间最长的页面，即先进入内存的页面先被淘汰。

这种算法实现起来比较简单，但性能较差。调入内存的页面按调入先后次序链接成一个队列，设置一个替换指针，指向其中最早进入的页。


块号
页号
指针
0
1
2
4
6
3
4
2
^
5
6
5
7
7
1
4
替换指针
2
（指向最老的页）

（*a*）


块号
页号
指针
0
1
2
6
^
3
4
2
2
5
6
5
7
7
1
4
替换指针
6
（指向最老的页）

（*b*）

2．先进先出置换算法

- 对于图7-4 的例子，采用FIFO 页面置换算法的置换结果如图所示。

5	0	1	2	0	4	0	7	2	4	0	4	2	1	2	0	1	5	0	1
5	5	5	2		2	2	7	7	7	0			0	0			5	5	5
	0	0	0		4	4	4	2	2	2			1	1			1	0	0
		1	1		1	0	0	0	4	4			4	2			2	2	1

图 7-5　FIFO 页面置换算法的置换结果

- 采用请求调页策略和FIFO 页面置换算法时，共发生了15 次缺页中断，进行了12 次页面置换，缺页率为75%。

- 当发生缺页中断，且内存无空闲物理块时，需要通过置换算法选择换出某些页面。当置换算法选择不当，会导致刚被换出的页很快又要被换入，因此缺页率会很高。而且频繁地换入换出，使得大部分时间都浪费在页面置换上。——“抖动”（“颠簸”）
- 原因：

  a.页面淘汰（置换）算法不合理；

  b.分配给进程的物理块数太少。

3.最近最久未使用(LRU) 置换算法

各页面将来的使用情况是无法预测的，但根据局部性原理，刚刚被访问的页面接下来被访问的可能性很大，因此可以用“最近的过去”作为“最近的将来”的合理近似。

该算法选择内存中最久未使用的页面被置换。这是局部性原理的合理近似。

3. 最近最久未使用置换算法

5	0	1	2	0	4	0	7	2	4	0	4	2	1	2	0	1	5	0	1
5	5	5	2		2		7	7	7	0			1		1		1		
	0	0	0		0		0	0	4	4			4		0		0		
		1	1		4		4	2	2	2			2		2		5		

图 7-6 LRU 页面置换算法的置换结果

- 对于同样的页面访问串，采用请求调页策略和LRU页面置换算法，共产生了12 次缺页，进行了9 次页面置换，缺页率为60%。
- 该算法的性能优于FIFO置换算法，接近于OPT置换算法。在实际应用中，如何确定页面的最后使用时间顺序是一个问题。

LRU置换算法的硬件支持

1) 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

$$R=R_{n-1}R_{n-2}R_{n-3}\ \ldots\ R_2R_1R_0$$

被访问时左边最高位置1，定期右移并且最高位补0，于是寄存器数值最小的是最久未使用页面。

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

把被访问的页面移到栈顶，于是栈底是最久未使用页面。

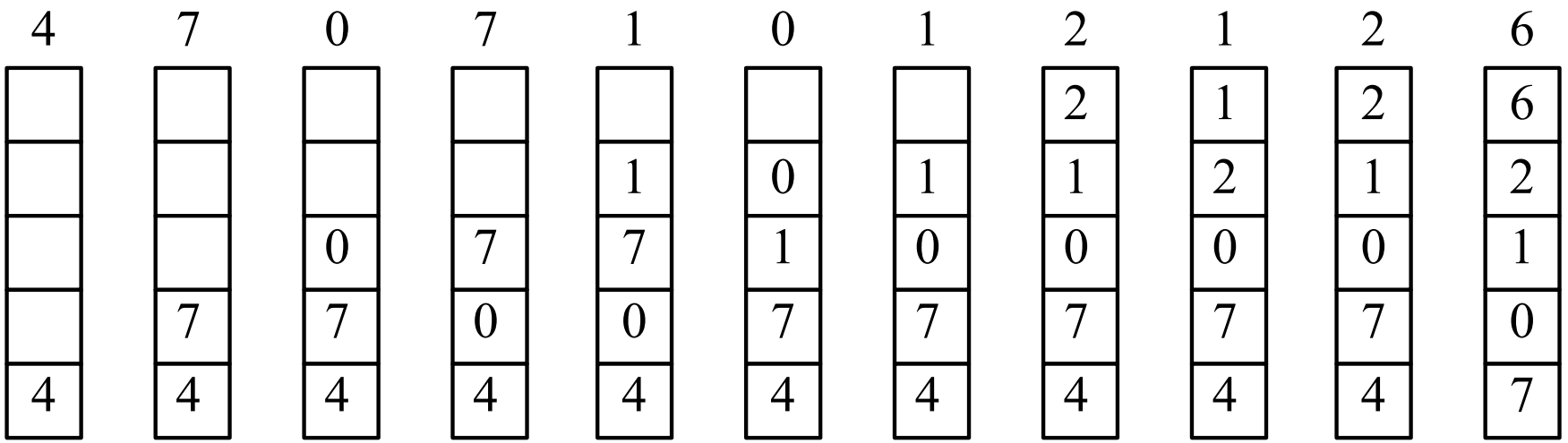
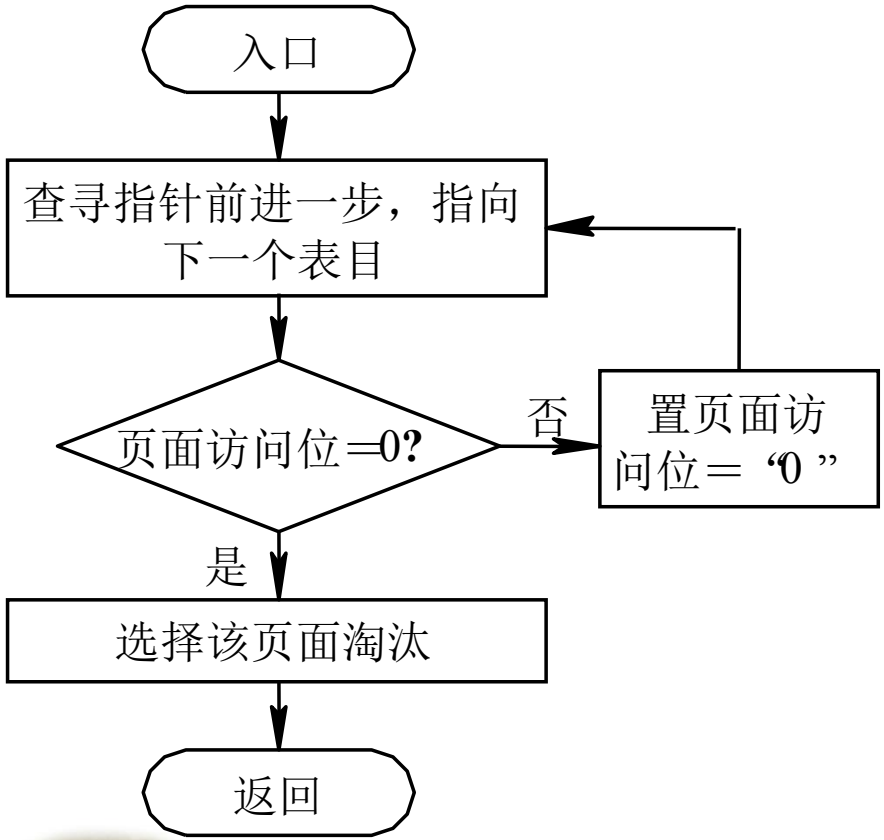


图 用栈保存当前使用页面时栈的变化情况

4. Clock置换算法

（1）简单的Clock置换算法(最近未使用算法或轮转算法)：

它是LRU(最近最久未使用算法)和FIFO的折衷。



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 简单Clock置换算法的流程和示例

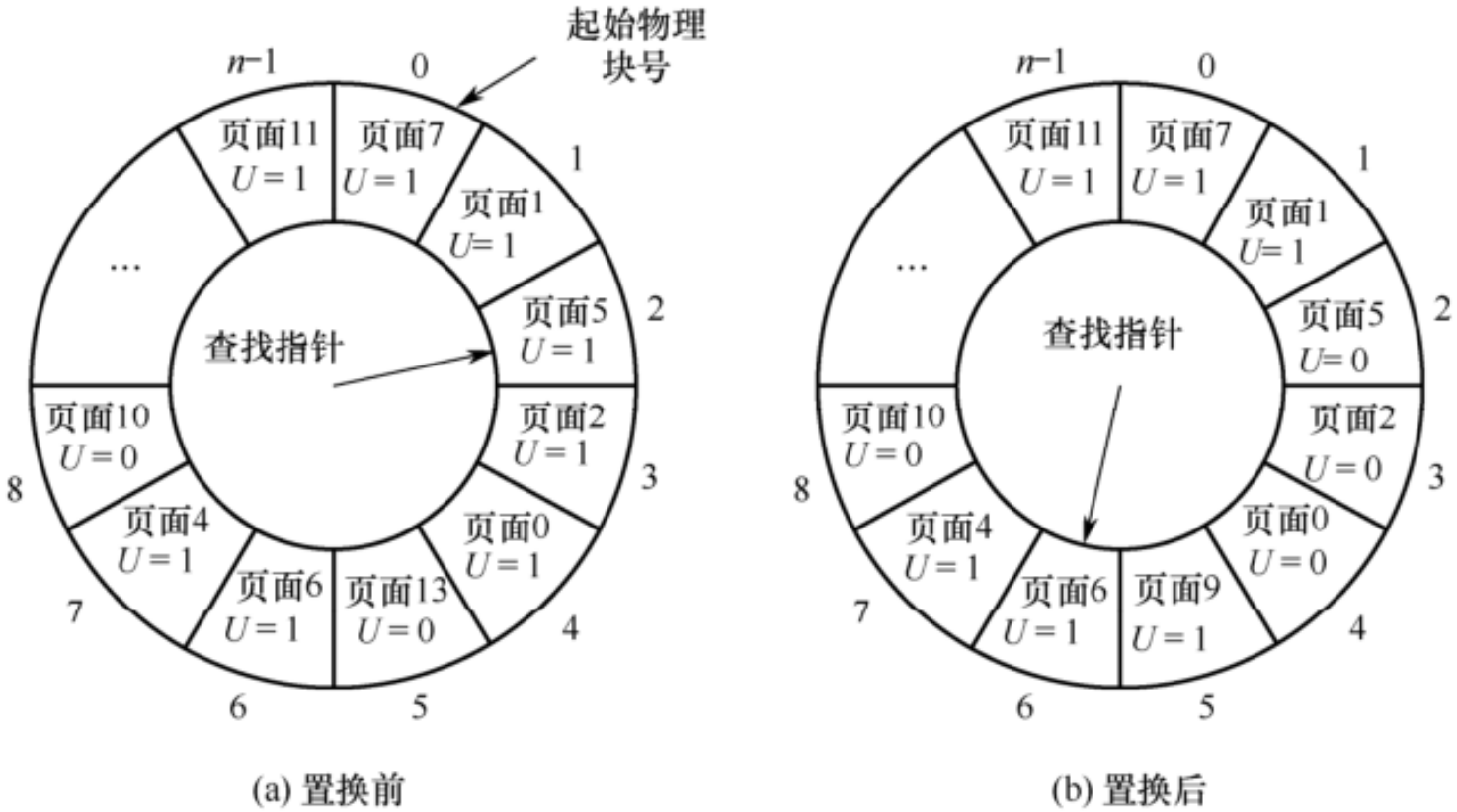


图 7-8 简单 Clock 置换算法示例

（2）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。

其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。

7.3 请求分段存储管理

7.3.1 工作原理

1. 段表机制

置换段时参考

是否做过动态增长

段名	段长	段的基址	存取方式	访问字段A	改动位M	状态位P	增补位	外存始址

选择换出段时参考

程序访问参考

调入该段时参考

在段表项中， 除了段名(号)、 段长、 段在内存中的起始地址外， 还增加了以下诸项：

(1) 存取方式。用于标识该段的存取属性，如只执行、只读、可读/写。

(2) 访问字段A。用于记录该段被访问的频繁程度，为页面置换算法淘汰段提供参考。

(3) 改动位M。用于表示该段在进入内存后是否被改动过，供页面置换算法使用。

(4) 状态位P。用于表示该段是否已调入内存。

(5) 增补位。用于表示该段在运行过程中是否做过动态增长。

(6) 外存始址。用于记录该段在外存的起始地址，即起始盘块号，供调入该段时使用。

2. 缺段中断机构

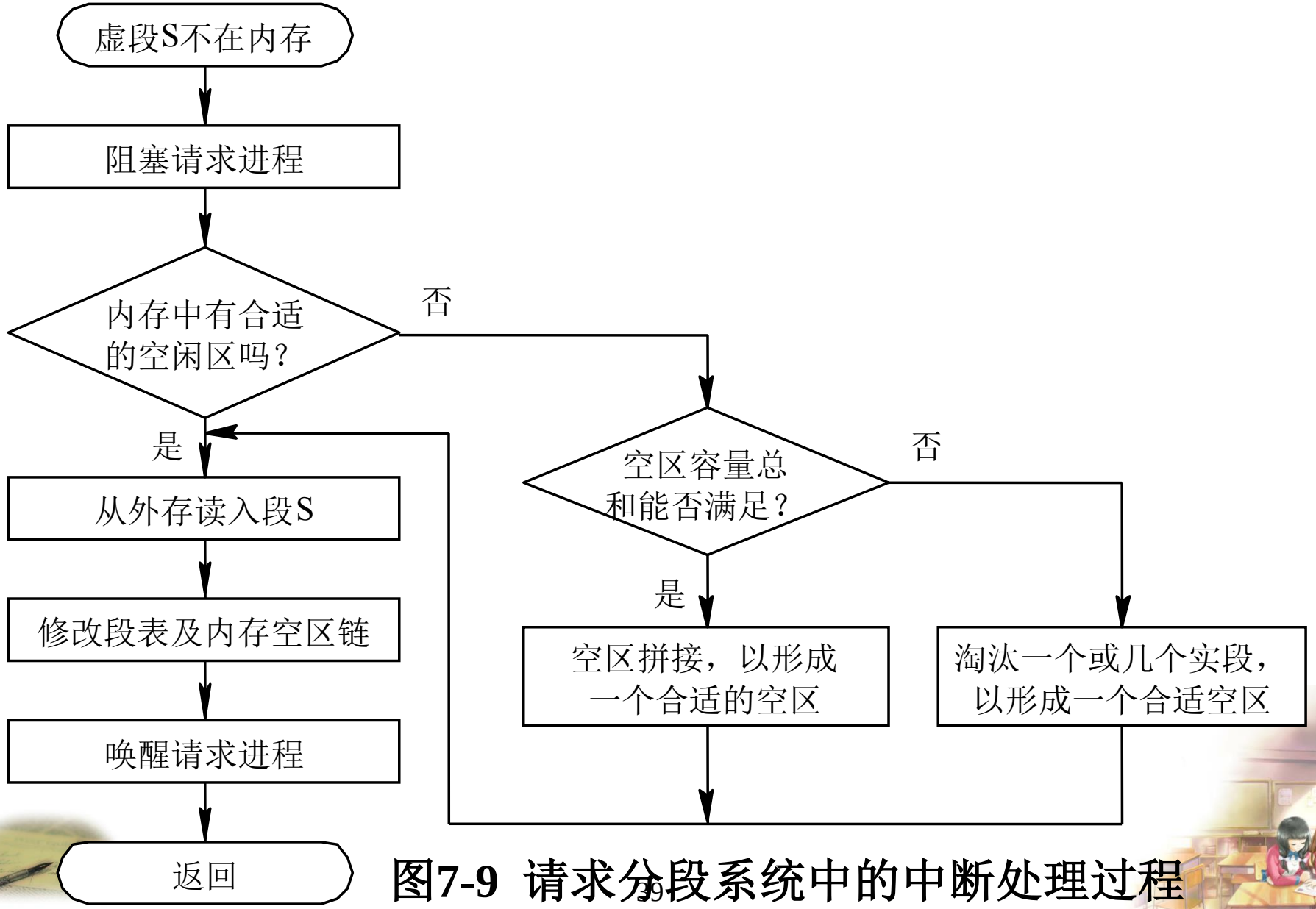


图7-9 请求分段系统中的中断处理过程

3. 地址变换机构

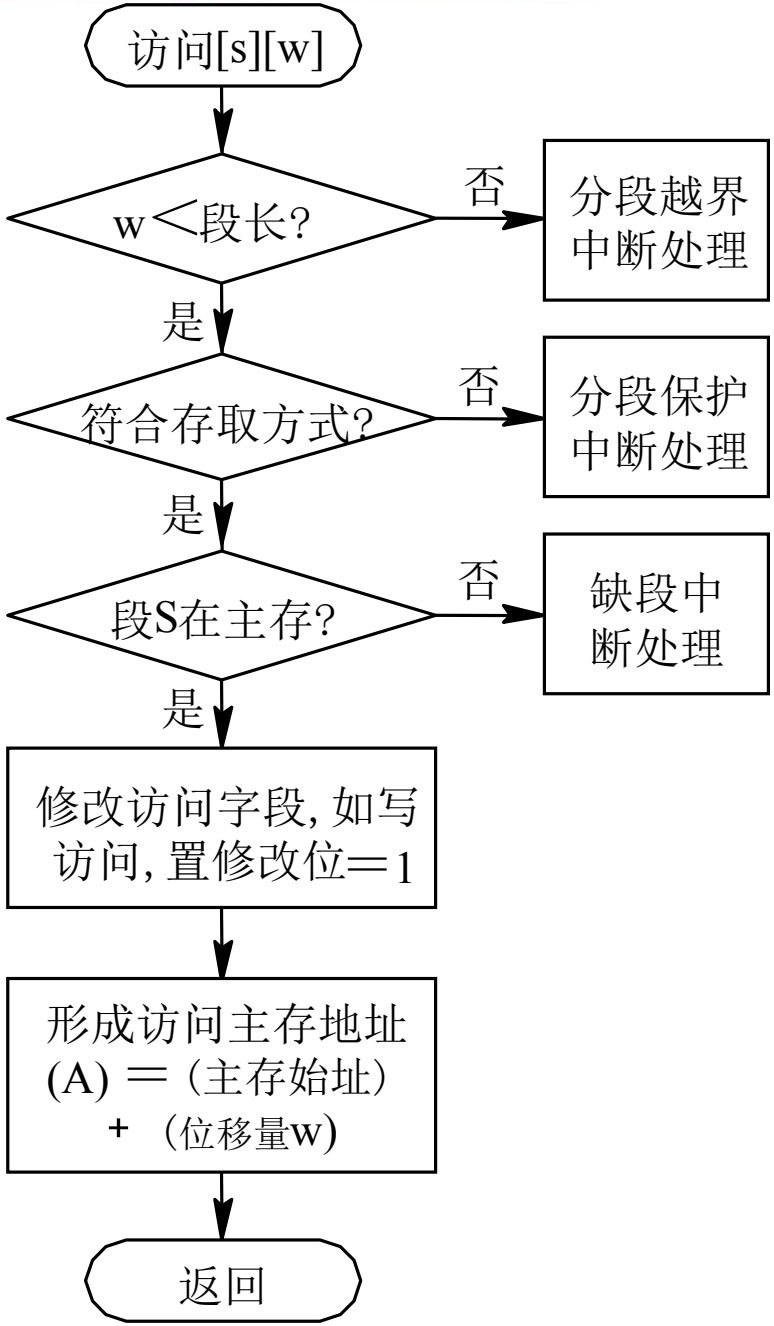


图 7-10 请求分段系统的地址变换过程

7.3.2 段的共享与保护

1. 段的共享

- 请求分段存储管理以段为单位进行信息共享。实现上，系统只需在每个进程的段表中为共享代码添加一个段表项，然后设置该段表项的段始址指向共享代码在内存的副本。

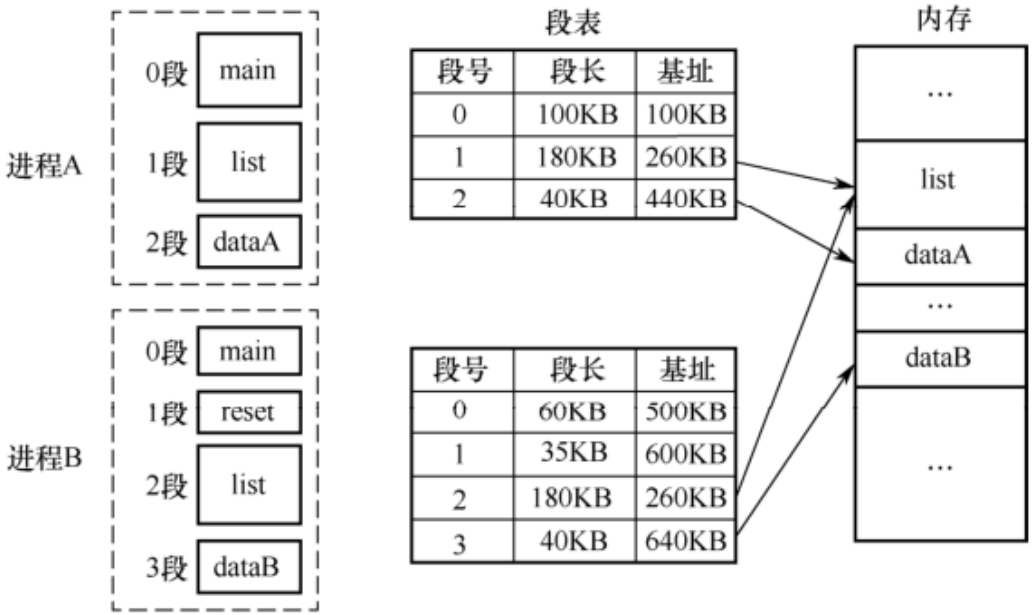


图 7-11 list 段的共享示意图

7.3.2 段的共享与保护

- 为了完成共享段的分配和回收，需要在系统中设置一张共享段表，所有共享段都在共享段表中占有一个表项。
- 共享段表的表项中记录了共享段的段名、段长、内存始址、状态、外存始址，还记录了共享进程计数count 及每个进程的情况，如进程名、进程号、该共享段在该进程中的段号等。

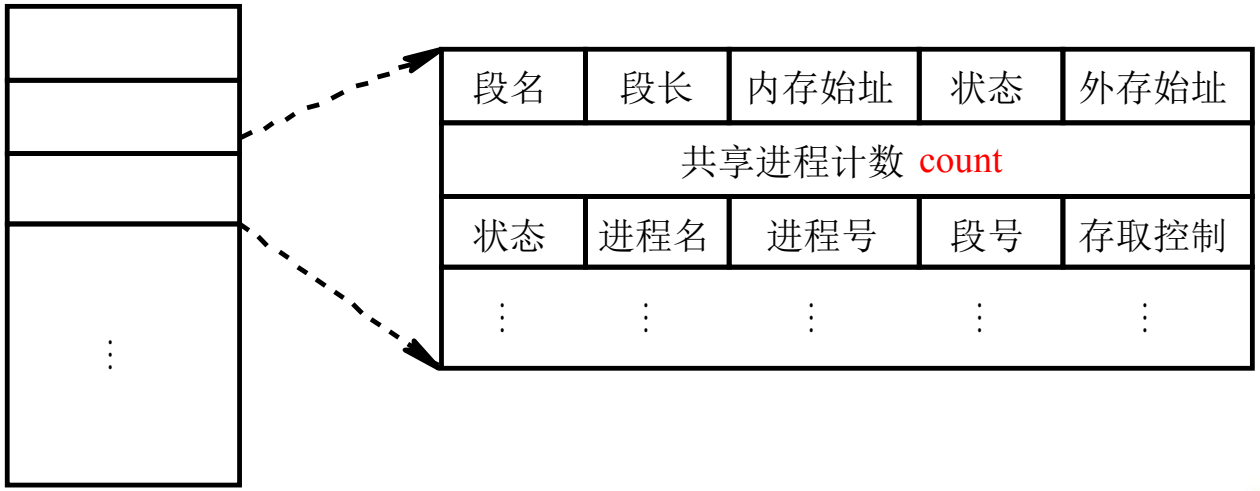


图 7-12 共享段表

2. 共享段的分配与回收

1) 共享段的分配

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，由系统为该段分配一物理区，再把其调入该区，同时将该区的始址填入请求进程的段表的相应项中，还须在共享段表中增加一表项，填写有关数据，**把count置为1**；之后当又有其它进程需调用该段时，由于该段已被调入内存，故无须再为该段分配内存，而只需在调用进程的段表中，增加一表项，填写该段的物理地址；在共享段表中，填上调用进程的进程名、存取控制等，再执行**count=count+1**，以表明有两个进程共享该段。

2) 共享段的回收

当共享此段的某进程不再需要该段时，应将该段释放，包括取消在该进程段表中共享段所对应的表项，以及执行**count=count-1**。若结果为**0**，则须由系统回收该共享段的物理内存，以及取消在共享段表中该段所对应的表项， 表明此时已没有进程使用该段；否则(减1结果不为0)， 则只是取消调用者进程在共享段表中的有关记录。

3. 分段保护

1) 越界检查：段表长度；每段段长

2) 存取控制检查

(1) 只读

(2) 只执行

(3) 读/写

小 结

虚拟存储器概念、特征，局部性原理；
请求分页，硬件支持，缺页中断与一般中断的区别；
页面分配与置换策略；
页面置换算法(注意比较)；
抖动的产生；
请求分段，硬件支持，缺段中断与地址变换；

第五章 典型题型分析与解答

1.在一个请求分页系统中，假如一个作业的页面走向为4，3，2，1，4，3，5，4，3，2，1，5，目前它还没有任何页装入内存，当分配给该作业的物理块数目M分别为3和4时，请分别计算采用OPT、LRU、FIFO页面淘汰算法时访问过程中所发生的缺页次数和缺页率，并比较所得的结果。

2.某虚拟存储器的用户空间共有32个页面，每页1K，主存16K。假定某时刻系统为用户的第0、1、2、3页分配的物理块号为5、10、4、7，而该用户作业的长度为6页，试将十六进制的虚拟地址0A5C、103C、1A5C转换成物理地址。

作业：在一个请求分页系统中，假如一个作业的页面走向为1，2，3，4，1，2，5，1，2，3，4，5，目前它还没有任何页装入内存，当分配给该作业的物理块数目M分别为3和4时，请计算采用FIFO页面淘汰算法时访问过程中所发生的缺页次数，看看会出现什么异常结果？