# 第五章　虚拟存储器

5.1　虚拟存储器概述
5.2　请求分页存储管理方式
5.3　页面置换算法
5.4　请求分段存储管理方式

韩明峰　　QQ 2022446359

# 5.1　虚拟存储器概述

问题的提出：

（1）一个程序要求的存储容量超过整个内存空间；

（2）有大量的作业需要装入内存运行而内存空间不足。

解决方案：

（1）从物理上增加内存容量，但这会增加系统成本，并且增加是有限的；

（2）从逻辑上增加内存容量，这正是虚拟存储技术所要解决的主要问题。

## 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：作业在运行前一次性地全部装入内存。

(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。

问题：一次性及驻留性在程序运行时是否是必须的？

## 2. 局部性原理

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。

(1)程序执行时，除了少部分的转移和过程调用指令外，在大多数情况下仍是顺序执行的。

(2)过程调用将会使程序的执行轨迹由一部分区域转至另一部分区域，过程调用的深度在大多数情况下不超过5。

(3)程序中存在许多循环结构，它们由少数指令构成，但会多次执行。

(4)为数组、链表等数据结构所分配的内存，通常多于实际需要值。

## 5.1.2 虚拟存储器的定义和特征

### 1. 虚拟存储器的定义

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一种存储器系统。

### 2. 虚拟存储器的特征

(1)多次性：分多次调入内存。

(2)对换性。

(3)虚拟性：用户看到的内存容量远大于实际的内存容量。

虚拟性是以多次性和对换性为基础的；

而多次性和对换性又必须建立在离散分配的基础上。

## 5.1.3　虚拟存储器的实现方法

### 1. 请求分页系统

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，置换是以页为单位进行的。

## 2. 请求分段系统

是在分段系统的基础上，增加了请求调段及分段置换功能后形成的段式虚拟存储系统，置换是以段为单位进行的。

# 5.2 请求分页存储管理方式

## 5.2.1 请求分页的处理流程

### 1. 请求页表机制

在请求分页系统中的每个页表应含以下诸项：

| 页号 | 物理块号 | 存在位 | 访问字段 | 修改位 | 外存地址 |
|---|---|---|---|---|---|

(1)存在位：指示该页是否已调入内存。

(2)访问字段：用于记录本页被访问的频繁程度。

(3)修改位：表示该页在调入内存后是否被修改过。若已被修改，置换该页时必须将该页重写到外存。

(4)外存地址：用于指出该页在外存上的地址。

## 2. 缺页中断

缺页中断是一种特殊的中断，它与一般的中断相比，有着明显的区别，主要表现如下：

(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缺页中断。

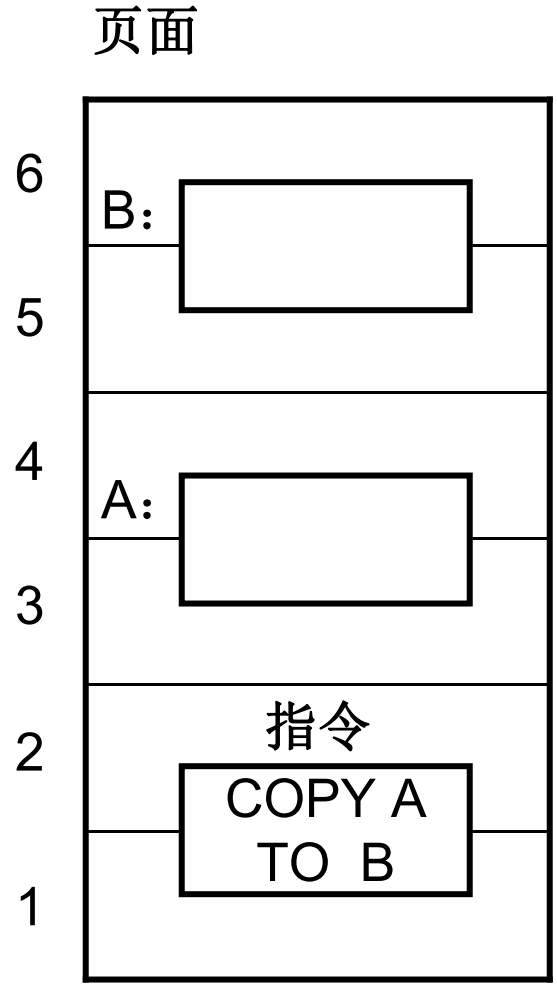
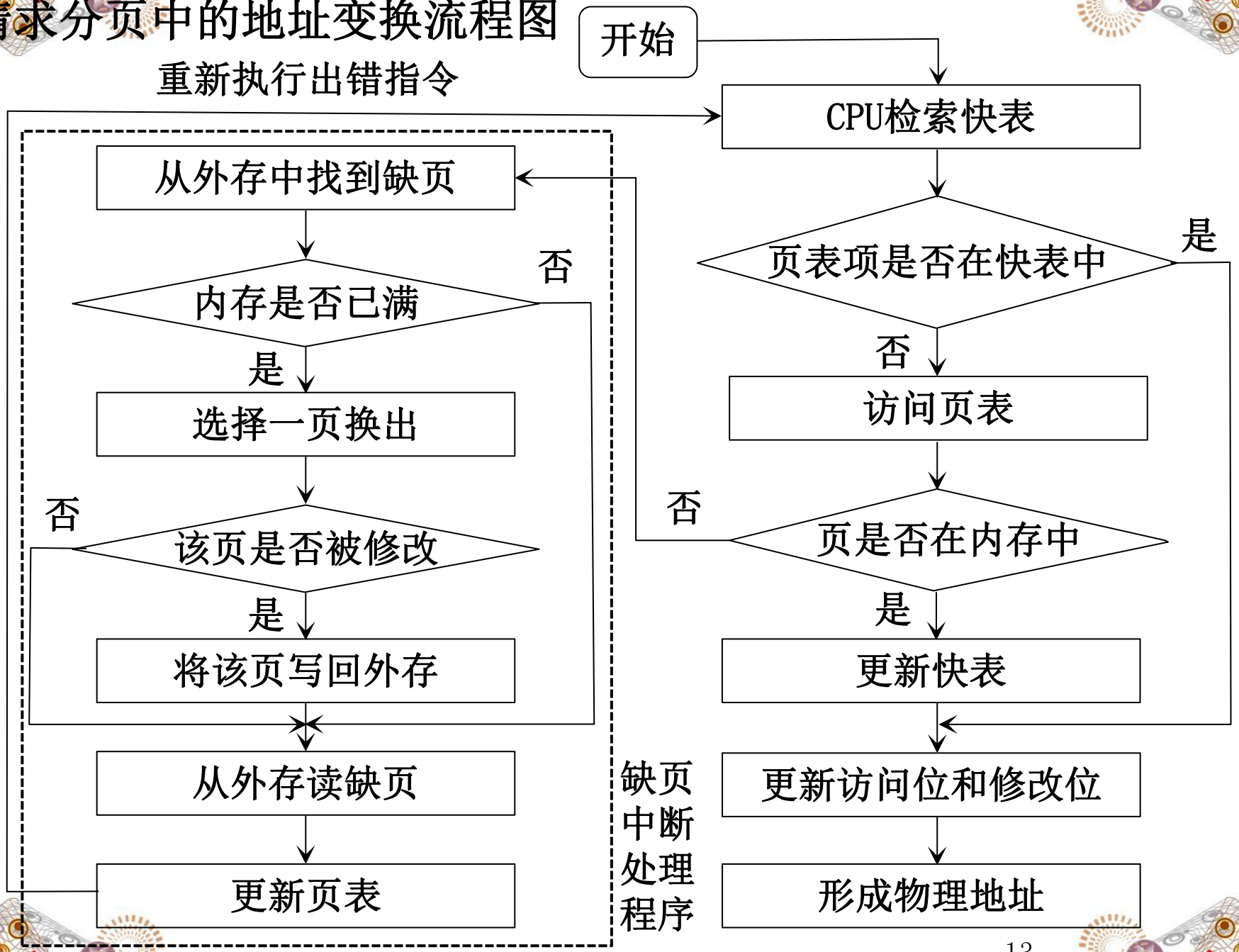



图5-1 涉及6次缺页中断的指令

### 3. 地址变换流程

请求分页系统中的地址变换流程，是在分页系统地址变换流程的基础上，再为实现虚拟存储器而增加了某些功能而形成的，如产生和处理缺页中断，以及从内存中换出一页的功能等。

# 请求分页中的地址变换流程图



## 5.2.2　请求分页中的内存分配

### 1. 最小物理块数的确定

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。当系统为进程分配的物理块数少于此值时，进程将无法运行。

## 2. 内存分配策略

| 两种内存分配策略 | 两种页面置换策略 |
| --- | --- |
| 固定分配 | 全局置换 |
| 可变分配 | 局部置换 |

(1)固定分配局部置换：为每个进程分配一定数目的物理块，在整个运行期间不再改变；如果进程在运行中发现缺页，只能从该进程在内存中的n个页面中选出一页换出，然后再调入一页。

困难：应为每个进程分配多少个物理块难以确定。

(2)可变分配全局置换：凡是产生缺页的进程，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才从内存中选择一页调出，该页可能是系统中任一进程的页。

(3)可变分配局部置换：在固定分配局部置换的基础上，如果进程在运行中频繁发生缺页中断，则系统再为进程分配若干物理块；如果进程在运行中缺页率特别低，则适当减少分配给该进程的物理块。

### 3. 固定分配策略的物理块分配算法

(1)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。

(2)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$S_i$，系统中各进程页面数的总和为**S**，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$b_i$可由下式计算：

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

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

(3)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，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

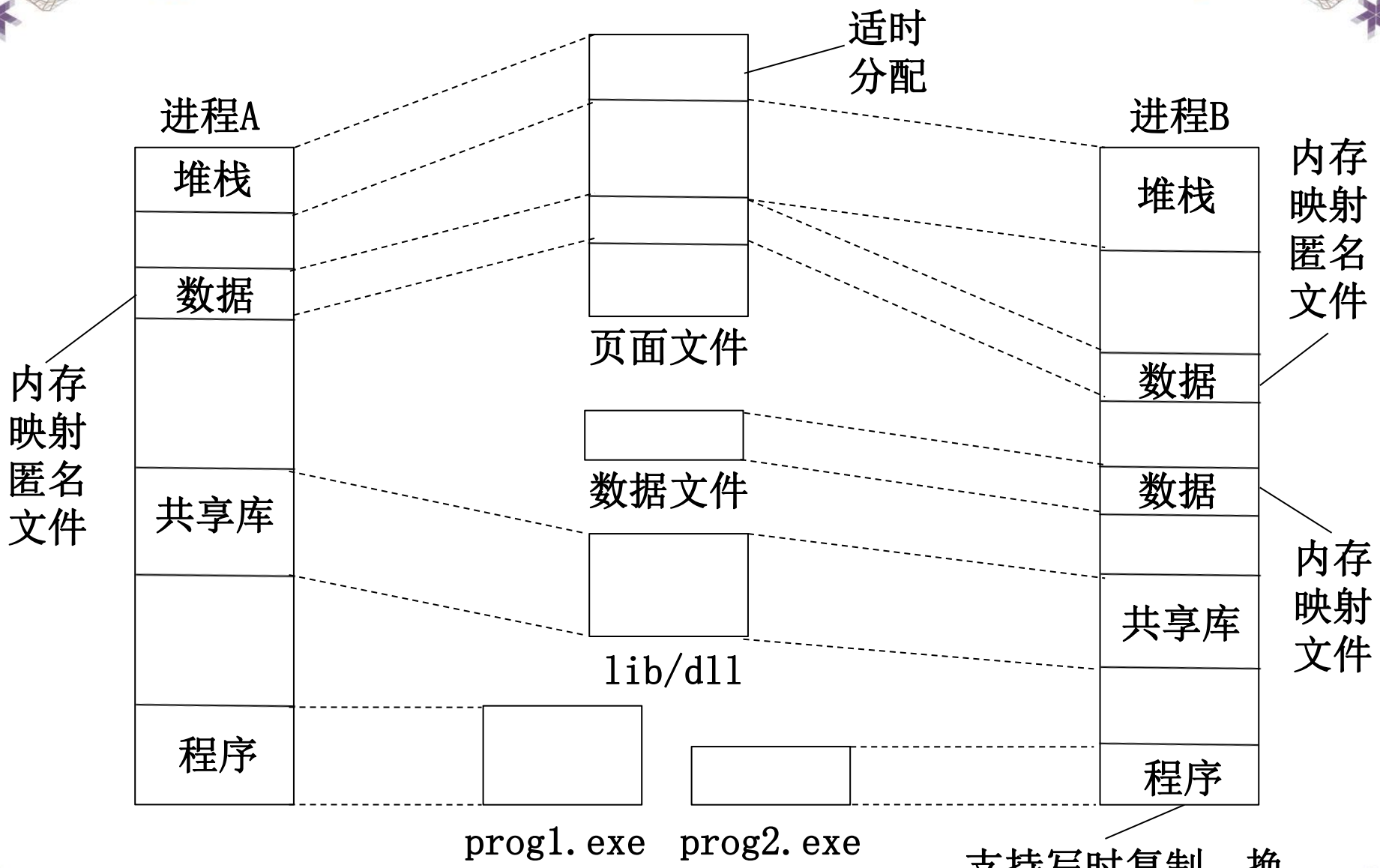
### 1. 何时调入页面

(1)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。 用于首次将进程调入内存，以及将工作集中所有页调入内存。

(2)请求调页策略。当程序在运行中需要访问某部分程序和数据时，若发现其所在的页面不在内存，便立即提出请求，由OS将其所需要的页面调入内存。

## 2. 从何处调入页面

以Windows操作系统为例来说明逻辑页面与磁盘文件之间的映射关系。




进程的虚拟地址空间与磁盘文件的映射关系

### 3. 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：

$$f = \frac{F}{A}$$

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$t_a$，被置换页面没有被修改的缺页中断时间为$t_b$，那么，缺页中断处理时间的计算公式为：

$$t=\beta\times t_a+(1-\beta)\times t_b$$

思考：此公式有没有忽略了某些情况？

(1)共享已在内存中的页面。

(2)有空闲内存，不需页面置换。

# 5.3 页面置换算法

## 5.3.1 最佳置换算法

由Belady于1966年提出的一种理论上的算法。

(1)算法：淘汰永不使用的或是在最长时间内不再被访问的页（页面被访问的次序越靠后越可能被淘汰）。

(2) 通常可保证获得最低的缺页率。

(3)无实现价值，作为其它算法的衡量标准。

【例5.1】设页面请求次序：

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

驻留集为3，假定最初存储块为空，采用OPT置换算法。请问：

（1）发生了几次缺页中断？

（2）缺页率多少？

解：

页面请求次序

| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 7 | 2 | | 2 | | 2 | | | 2 | | | 2 | | | | 7 | | |
| | 0 | 0 | 0 | | 0 | | 4 | | | 0 | | | 0 | | | | 0 | | |
| | | 1 | 1 | | 3 | | 3 | | | 3 | | | 1 | | | | 1 | | |
| | | | 7 | | 1 | | 0 | | | 4 | | | 3 | | | | 2 | | |

被换出的页面

(1)从上面的演示知，利用OPT，发生了6次页面置换，发生了9次缺页中断；

(2) 缺页率=缺页次数/访问次数=9/20=0.45

## 5.3.2 先进先出置换算法（最早出现的置换算法）

依据：最早调入主存的页，其不再被使用的可能性比最近调入主存的页要大。

算法：淘汰最先进入主存的页。

实现：已调入内存的页面，按先后次序链接成一个队列。

缺点：有些页面经常被访问，但FIFO算法并不能保证这些页面不被淘汰。

【例5.2】设页面请求次序：

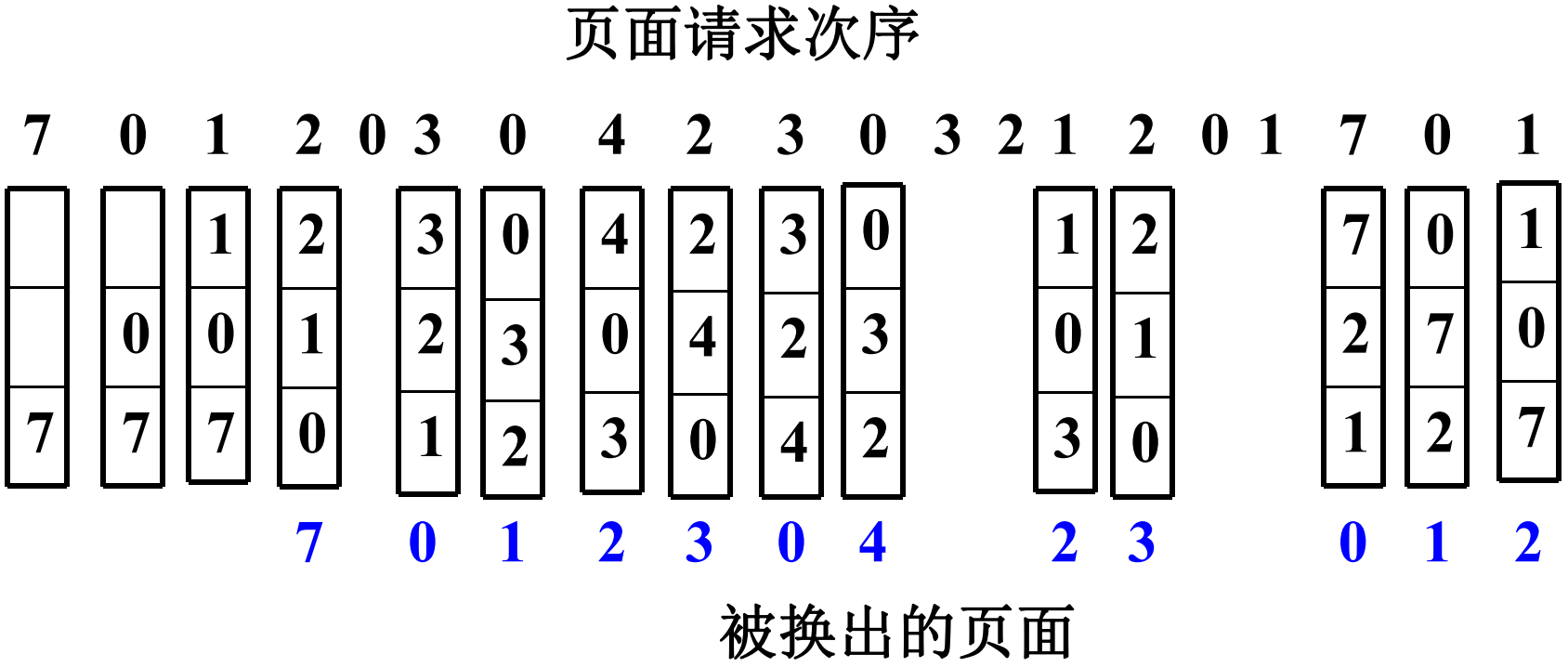
7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

驻留集为3，假定最初存储块为空，采用FIFO置换算法。请问：

（1）发生了几次缺页中断？

（2）缺页率多少？

解：




（1）从上面的演示知，利用FIFO，发生了12次页面置换，发生了15次缺页中断。

（2）缺页率=缺页次数/访问次数=15/20=0.75 。

## 5.3.3 最近最久未使用算法

### 1. LRU(Least Recently Used)置换算法的描述

依据：它认为过去一段时间里不曾被访问过的页面，在最近的将来可能也不会再被访问（用过去预测未来）。

算法：当需要置换一页面时，选择在最近一段时间内最久不用的页面予以淘汰。

实现：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，选择t值最大的淘汰。

【例5.3】设页面请求次序 ：

7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

驻留集为3，假定最初存储块为空，采用LRU置换算法。请问：

（1）发生了几次缺页中断？

（2）缺页率多少？

解：

页面请求次序

| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
| | 0 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 7 |
| 7 | 7 | 7 | 0 | 1 | 2 | 2 | 3 | 0 | 4 | 2 | 2 | 0 | 3 | 3 | 1 | 2 | 0 | 1 | 0 |
| | | | 7 | | 1 | | 2 | 3 | 0 | 4 | | | 0 | | 3 | | 2 | | |

被换出的页面

（1）从上面的演示知，利用LRU，发生了9次页面置换，发生了12次缺页中断。

（2）缺页率=缺页次数/访问次数=12/20=0.60

## 2. LRU置换算法的硬件支持

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

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

当进程访问某物理块时，要将相应寄存器的$R_{n-1}$置成1；定时信号每隔一定时间将寄存器右移一位。

如果把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。

| 页面 \ 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**

## 5.3.4　访问内存的有效时间

（1）被访问的页在内存中，且其对应的页表项在快表中。

EAT1=查找快表时间+访问物理地址时间

（2）被访问的页在内存中，且其对应的页表项不在快表中。

EAT2=查找快表时间+查找页表时间+修改快表时间+访问物理地址时间

## 5.3.4　访问内存的有效时间

（3）被访问的页不在内存中。

EAT3=查找快表时间+查找页表时间+处理缺页中断时间(假设含修改页表、修改快表时间)+查找快表时间+访问物理地址时间。

其中处理缺页中断时间随页面是否被修改而不同。

（4）a表示命中率，f表示缺页率。

EAT=a.EAT1+(1-a-f)EAT2+f.EAT3

【例5.4】页面大小为4KB，一次内存的访问时间是100ns，一次快表（TLB）的访问时间是10ns，处理一次缺页的平均时间为$10^8$ns（已含更新TLB和页表的时间），进程的驻留集大小固定为2，采用最近最久未使用置换算法（LRU）和局部淘汰策略。假设：

（1）TLB初始为空；

（2）地址转换时先访问TLB，若TLB未命中，再访问页表（忽略访问页表之后的TLB更新时间）；

（3）存在位为0表示页面不在内存，产生缺页中断，缺页中断处理后，返回到产生缺页中断的指令处重新执行。

页表如下：

| 页号 | 物理块号 | 存在位 |
| --- | --- | --- |
| 0 | 101H | 1 |
| 1 | -- | 0 |
| 2 | 254H | 1 |

设有虚地址访问序列2362H、1565H和25A5H，请问：

（1）依次访问上述三个虚地址，各需多少时间？给出计算过程。

（2）基于上述访问序列，虚地址1565H的物理地址是多少？

解：

（1）页面大小为4KB，即$2^{12}$，则得到页内位移占虚地址的低12位，页号占剩余高位。可得虚地址的页号P如下：

1）2362H：

① P=2，访问快表10ns；

② 因快表初始为空，访问页表100ns得到物理块号（更新快表）；

③ 合成物理地址后访问主存100ns，

共计：10ns+100ns+100ns=210ns。

2）1565H：

① P=1，访问快表10ns落空；

② 访问页表100ns落空；

③ 进行缺页中断处理$10^8$ns（更新页表和快表）；

④ 重新执行出错指令，访问快表10ns；

⑤ 合成物理地址后访问主存100ns，

共计：10ns+100ns+$10^8$ns+10ns+100ns。

3）25A5H：

① P=2，访问快表，因第一次访问已将该页号放入快表，因此花费10ns便可合成物理地址；

② 访问主存100ns，

共计：10ns+100ns=110ns。

（2）当访问虚地址1565H时，

1)产生缺页中断；

2)合法驻留集为2，必须从页表中淘汰一个页面。2号页面刚被访问，根据题目的置换算法，应淘汰0号页面，因此1565H的对应物理块号为101H；

3)由此可得1565H的物理地址为101565H。

思考：

（1）以上计算包含了几种访问内存时间计算的情况？

（2）本题假设TLB初始为空，这可能发生在什么情况下？

# 5.4　请求分段存储管理方式

## 5.4.1　请求分段的处理流程

### 1. 请求段表机制

在请求分段系统中所需要的主要数据结构是段表。由于在应用程序的许多段中，只有一部分段装入内存，其余的一些段仍留在外存上，故需在段表中增加若干项，以供程序在调进、调出时参考。

请求分段系统中的段表项：

| 段名 | 段长 | 段的基址 | 存取方式 | 访问字段 | 修改位 | 存在位 | 增补位 | 外存始址 |
|---|---|---|---|---|---|---|---|---|

(1)存取方式：用于标识本分段的存取属性。

(2)访问字段：用于记录本段被访问的频繁程度。

(3)修改位：表示该段在调入内存后是否被修改过。

(4)存在位：指示该段是否已调入内存。

(5)增补位：用于表示该段在运行中是否做过动态增长。

(6)外存地址：用于指出该段在外存上的起始地址(盘块号)。

## 2. 缺段中断

缺段中断同样需要在一条指令的执行期间，产生和处理中断，以及在一条指令执行期间，可能产生多次缺段中断。

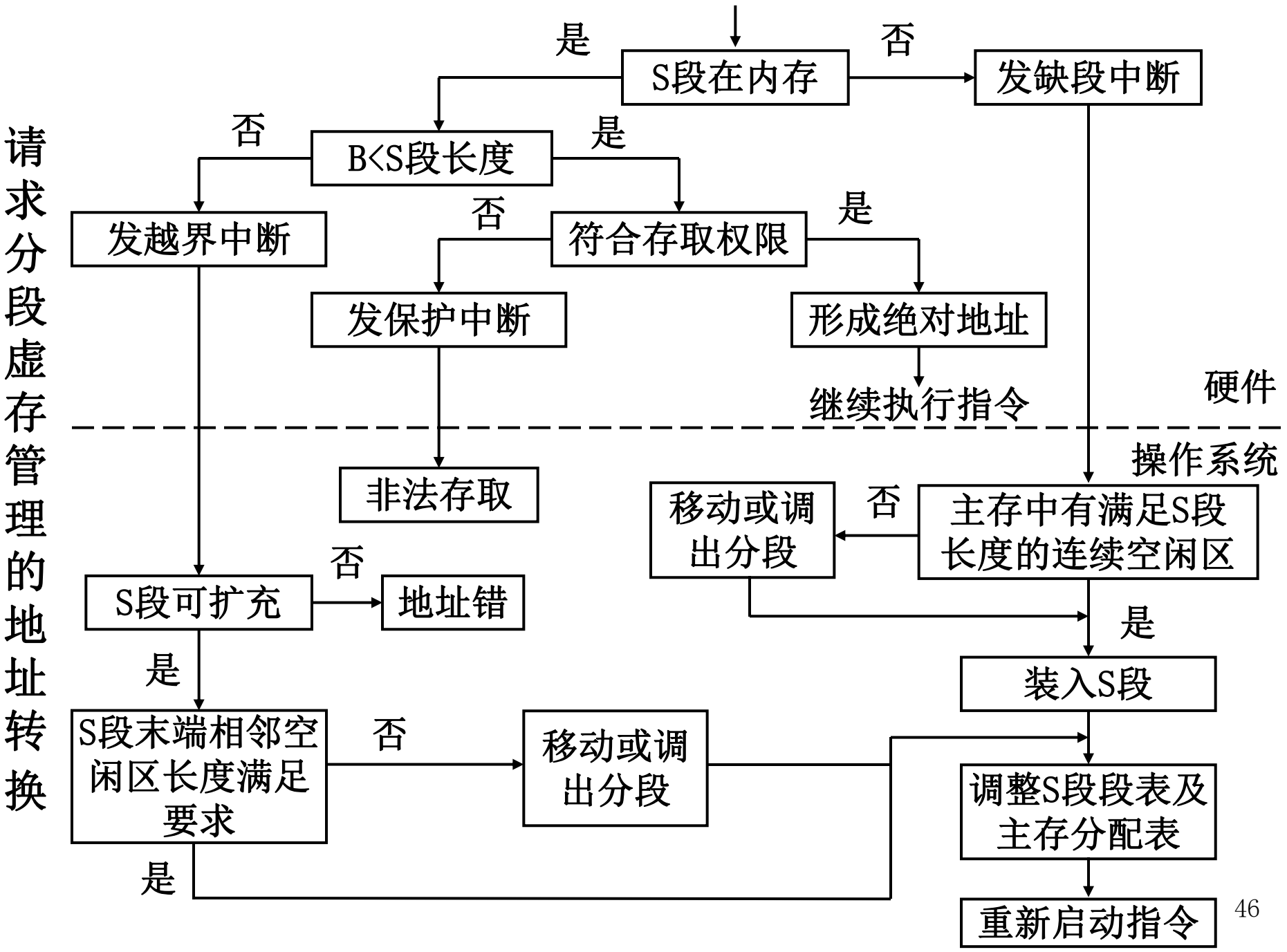
但不会出现一条指令被分割在两个分段中或一组信息被分割在两个分段中的情况。

### 3. 地址变换流程

请求分段系统中的地址变换流程是在分段系统地址变换流程的基础上形成的。

因为被访问的段并非全在内存，所以在地址变换时，若发现所要访问的段不在内存，必须先将所缺的段调入内存，并修改段表，然后才能再利用段表进行地址变换。

为此，在地址变换流程中又增加了某些功能，如缺段中断的请求及处理等。


请求分段虚存管理的地址转换
是
否
S段在内存
发缺段中断
否
是
B<S段长度
发越界中断
否
是
符合存取权限
发保护中断
形成绝对地址
继续执行指令
硬件
操作系统
非法存取
移动或调出分段
否
主存中有满足S段长度的连续空闲区
否
S段可扩充
地址错
是
是
装入S段
S段末端相邻空闲区长度满足要求
否
移动或调出分段
是
调整S段段表及主存分配表
重新启动指令

## 5.4.2 分段的共享与保护

### 1. 共享段表

为了实现共享，可在内存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。

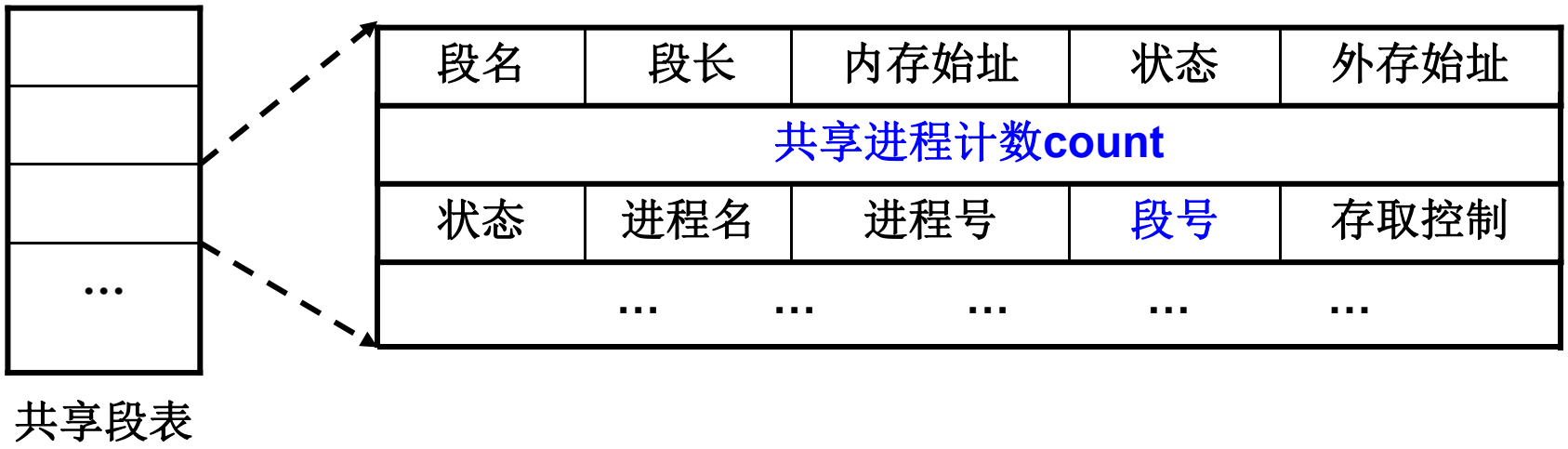



图5-14 共享段表项

## 2. 共享段的分配与回收

(1)共享段的分配

对第一个请求使用该共享段的进程：

1)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，填写调用进程的进程名、段号和存取控制等，把count置为1。

当又有其他进程需要调用该共享段时：

1)无需再为该段分配内存；

2)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；

3)在共享段的段表中，填上调用进程的进程名、段号和存取控制等，再执行**count=count+1**操作。

(2)共享段的回收

1)撤消在该进程段表中共享段所对应的表项；

2)执行count=count-1操作；

3)如果结果为0，则需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；

否则只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

段表寄存器存放了段表长度，段表中存放了每个段的段长。在进行存储访问时，将段号与段表长度比较，段内地址与段长比较。

(2)存取控制检查

段表中的每个表项都设置了“存取控制”字段，用于规定该段的访问方式。

只读　　只执行　　读/写