第五章　虚拟存储器

BACK

5.1 虚拟存储器的基本概念

- 前面介绍的各种存储管理方式，大都需要将作业全部装入内存后方可运行。 于是，出现了下面两种情况：
  - 当作业要求的内存空间超过内存总容量时，无法装入内存运行；
  - 有大量作业要求运行，但内存容量不足以容纳所有这些作业，只能将少量作业装入内存运行。
- 解决内存容量不够问题的办法：
  - 从物理上扩充内存；
  - 从逻辑上扩充内存——虚拟存储器。

5.1.1 虚拟存储器的引入

1. 传统存储器管理方式的特征

- 一次性
- 驻留性

不用或暂不用的程序（数据）占据了大量内存空间，使得需要运行的作业无法装入运行。

2. 局部性原理

- 程序在执行时将呈现局部性规律，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。
  - 时间局部性——某指令一旦执行，则不久后该指令可能再次被执行（数据亦然）。**循环**
  - 空间局部性——程序一旦访问了某个存储单元，不久后，附近的存储单元也将被访问。**顺序执行**

5.1.2 虚拟存储器的定义和特征

- 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系统。 虚拟存储器基本原理如下：

部分装入

- 应用程序在运行前，没有必要全部装入(局部性原理)，仅将那些当前要运行的页面或段先装入内存便可以运行，其余部分暂留在磁盘上。

请求调页

- 程序在运行时，如果所要访问的页（段）已调入内存，便可继续执行下去；否则，应利用OS所提供的请求调页（段）功能，将它们调入内存，以便继续运行。

页面置换

- 如果此时内存已满，则需利用页（段）置换功能，将内存中暂不用的页（段）调到磁盘上，再将访问的页（段）调入内存，使程序继续运行下去。

5.1.2 虚拟存储器的定义和特征

- **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系**

虚拟存储器的逻辑容量由内存容量和外存容量之和所决定。

运行速度接近于内存，每位的成本接近于外存。

2. 特征

- **多次性**：一个作业被分成多次调入内存运行(部分装入)；
- **对换性**：作业在运行过程中进行换进、换出，不需要常驻内存；
- **虚拟性**：从逻辑上扩充内存，使用户看到的内存容量大于实际内存容量。这是虚拟存储器所表现的最重要的特征，也是实现虚拟存储器的最重要的目标。

虚拟性是以多次性和对换性为基础的，仅当系统允许将作业分多次调入内存，并能将内存中暂时不运行的程序和数据换至磁盘上时，才有可能实现虚拟存储器。

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. 分页请求系统

- **在分页系统基础上，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。**
- 必须提供硬件支持和相应的软件：
  - **硬件支持**
    - 请求分页的页表机制：在纯分页的页表机制上增加若干项而形成，作为请求分页的数据结构。
    - 缺页中断机构：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，请求OS将所缺的页调入内存。
    - 地址变换机构
  - **实现请求分页的软件**——请求调页、实现页面置换的软件

2. 请求分段系统

- **在分段系统基础上，增加了请求调段功能和分段置换功能所形成的段式虚拟存储系统。**
- **硬件支持**
  - 请求分段的段表机制
  - 缺段中断机构
  - 地址变换机构
- **实现请求分段的软件**——请求调段、实现段置换的软件

5.2 请求分页存储管理方式

5.2.1 请求分页中的硬件支持

1．页表机制

每个页表表项如下所示：

页号	物理块号	状态位 P	访问字段 A	修改位 M	外存地址

- 状态位P：指示该页是否已调入内存；只有一位，也称为位字
- 访问字段A：记录本页在一段时间内被访问的次数，或记录本页最近已有多长时间未被访问，提供给置换算法在选择换出页面时参考
- 修改位M：表示该页被调入内存后是否被修改过
- 外存地址：指出该页在外存的地址，通常是磁盘物理块号。

2．缺页中断机构

- 在指令执行期间产生和处理——通常情况下，CPU都是在一条指令执行完后才检查是否有中断请求到达。
- 一条指令执行期间可能要产生多次：硬件机构应能保存多次中断时的状态，并保证最后能返回到中断前产生缺页中断的指令处继续执行。

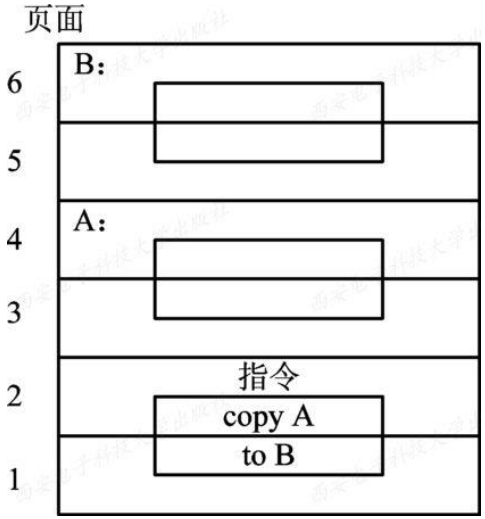


图5-1 涉及6次缺页中断的指令

3．地址变换机构

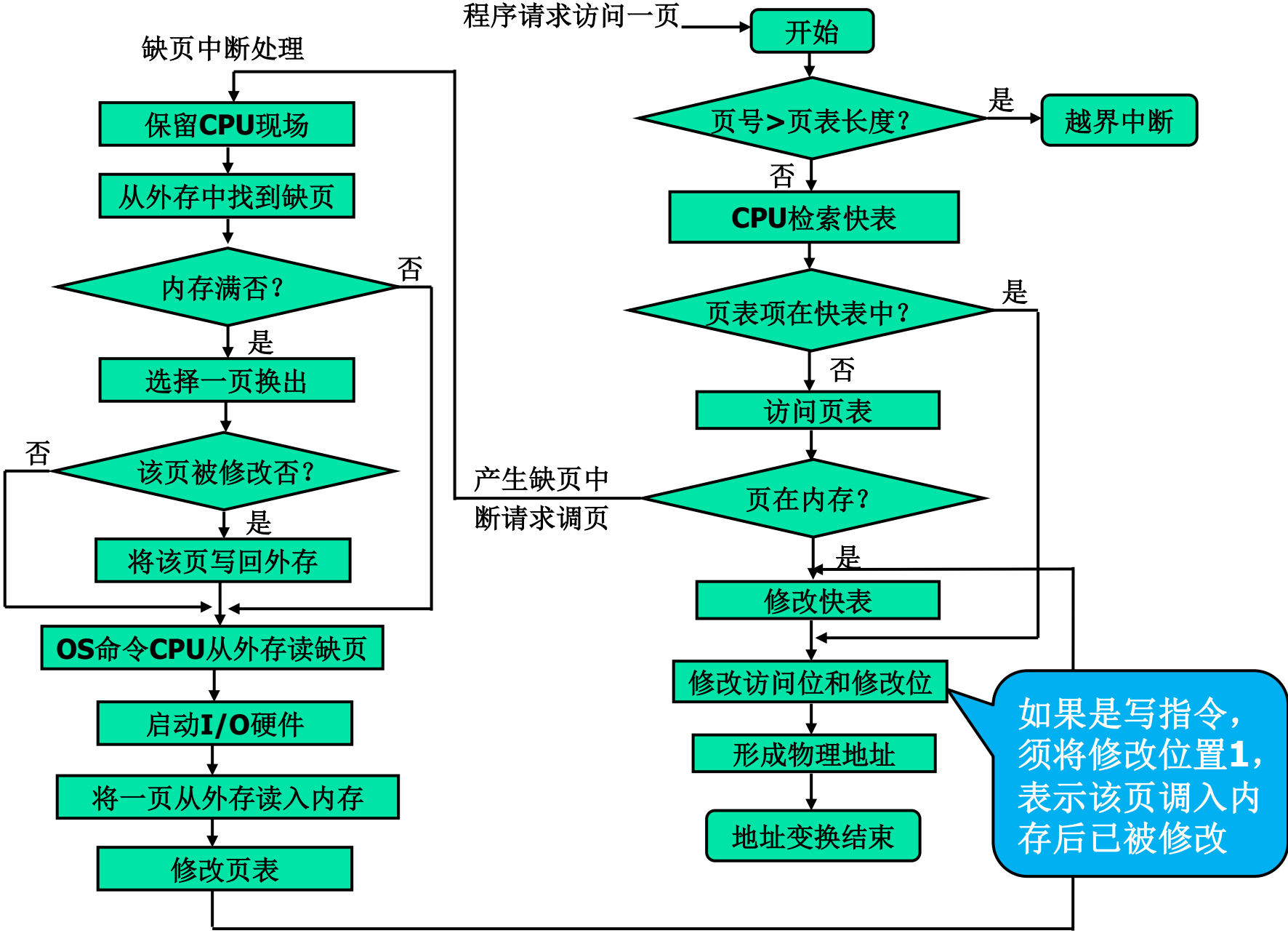


图5-2　请求分页中的地址变换过程

5.2.2 内存分配策略和分配算法

- 为进程分配内存时，涉及到3个问题：
  - **最小物理块数的确定**
    - 取决于指令的格式、功能和寻
  - **物理块的分配策略**
    - 固定分配局部置换
    - 可变分配全局置换——用于若干OS中
    - 可变分配局部置换
  - **物理块分配算法**
    - 平均分配算法——显
    - 按比例分配算法——
    - 考虑优先权的分配算法——合理

最小物理块数：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

固定分配，为每个进程分配一组固定数目的物

用于

若

的增

全局

OS所

进程

选择

配给

缺点：增加缺页率

当某进程发现缺页时，只允许该进程在内存的页面中选择一页换出，这样不会影响其他进程的运行。

如果进程频繁缺页，系统须再为该进程分配若干附加的物理块，直至该进程的缺页率减少到适当程度；

反之，若进程在运行期间缺页率特别低，可适当减少分配的物理块数，但不要引起缺页率的明显增加。

5.2.3 页面调入策略

- 何时调入页面
  - 预调页策略——若预测较准确，则非常有吸引力
  - 请求调页策略——每次仅调入一页，系统开销较大，增加了磁盘I/O的启动频率。
- 从何处调入页面
  - 系统有足够的对换区——全部从对换区调入
  - 系统缺少对换区——全部从文件区调入
  - UNIX方式——凡未运行的页面，从文件区调入；曾经运行过又被换出的页面，从对换区调入。共享页面如已被其他进程调入内存，则无须再调入
- 缺页率
  - f = F/A = F/（S+F）

    S：访问页面成功的次数；F：访问页面失败的次数

5.3 页面置换算法

好的页面置换算法，应具有较低的页面更换频率

不适当的算法可能导致进程发生“抖动”（Trashing），即刚被换出的页很快又要被访问，需要将它重新调入，此时需要再选一页调出；而此刚被调出的页很快又要被访问，又需要将它调入。

5.3 页面置换算法

好的页面置换算法，应具有较低的页面更换频率

5.3.1 最佳置换算法和先进先出置换算法

1. 最佳置换算法（OPT）

- 选择以后永不使用的或者是未来最长时间内不再使用的页面淘汰

目前人们还无法预知，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，因此该算法只是一种理论上的算法，是无法实现的。

可以利用该算法去评价其他算法。

5.3 页面置换算法

好的页面置换算法，应具有较低的页面更换频率

5.3.1 最佳置换算法和先进先出置换算法

1. 最佳置换算法（OPT）

- 选择以后永不使用的或者是未来最长时间内不再使用的页面淘汰

【例5-1】假定系统为某进程分配了3个物理块，并考虑以下的页面引用串：

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

引用串	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
物理块	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
被置换的页				7		1		0			4			3				2		

图5-3 利用最佳页面置换算法的置换图

发生6次页面置换

2．先进先出（FIFO）页面置换算法

选择在内存中驻留时间最长的页面淘汰。
设置一个指针，指向最老的页面。

仍以例5-1为例。

【例5-1】假定系统为某进程分配了3个物理块，并考虑以下的页面引用串：
7，0，1，2，0，3，0，4，2，3，0，3，2，1，2，0，1，7，0，1

引用串	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
物理块	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
		0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
			1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
被置换的页				7			0	1	2	3	0	4			2	3			0	1	2

图5-4 利用FIFO页面置换算法时的置换图

进行了12次页面置换

2．先进先出（FIFO）页面置换算法

选择在内存中驻留时间最长的页面淘汰。
设置一个指针，指向最老的页面。

仍以例5-1为例。

【例5-1】假定系统为某进程分配了3个物理块，并考虑以下的页面引用串：
7，0，1，2，0，3，0，4，2，3，0，3，2，1，2，0，1，7，0，1

FIFO的性能较差，因为它所依据的条件是各个页面调入内存的时间，而页面调入内存的时间并不能反映页面的使用情况。

进行了12次页面置换

5.3.2 最近最久未使用（LRU）置换算法

选择最近最久未使用的页面进行淘汰。

LRU页面置换算法的演算过程可以借助“栈”来完成：栈顶始终是最新的，栈底是最近最久未使用的页面号。

仍以例5-1为例。

引用串	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
物理块	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
被置换的页				7		1		2	3	0	4			0		3		2		

图4-35 LRU页面置换算法的置换图

发生9次页面置换

2. 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

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

当进程访问某物理块时，要将相应寄存器的R_{n-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。

R / 实 页	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。假定现有一进程，它分有五个物理块，所访问的页面的页面号序列为：

4，7，0，7，1，0，1，2，1，2，6

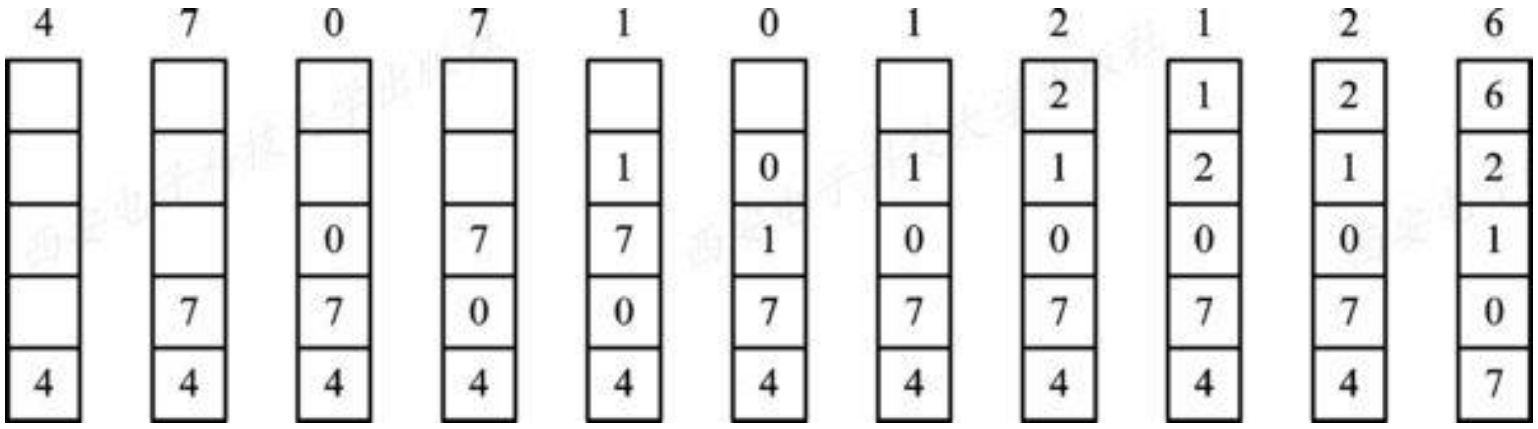


图5-7　用栈保存当前使用页面时栈的变化情况

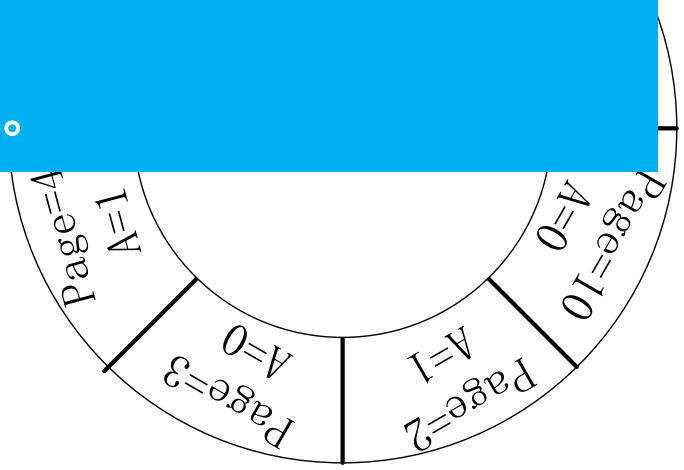
5.3.3 Clock置换算法

（1）简单Clock置换算法(最近未用算法)

最近最久未使用算法（LRU）是一种较好的算法，但需要较多的硬件支持，实现所需要的成本较高。

实际应用中，大多采用LRU的近似算法。

- 置换页面时，从指针位置开始，查看指针所指页面的访问位是否为1，如果是1，则将该页面的访问位修改为0。
- 否则，挑选该页面换出内存，并将新的页面换入内存，置新页面访问位为1；最后让指针指向下一个页面。
- 第1圈扫描结束后，若没有找到淘汰的页面，则进行第2圈扫描，此时，必定能够找到淘汰的页面。



Clock置换算法举例

仍以例5-1为例：

假定系统为某进程分配了3个物理块，并考虑以下的页面引用串：7，0，1，2，0，3，0，4，2，3，0，3，2，1，2，0，1，7，0，1。采用Clock置换算法，计算其页面置换次数。

引用串	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
物理块	7*	7*	7*	2*	2*	2*	2*	4*	4*	4*	4	3*	3*	3	3	0*	0*	0	0*	0*
		0*	0*	0	0*	0	0*	0	2*	2*	2	2	2*	1*	1*	1*	1*	7*	7*	7*
			1*	1	1	3*	3*	3	3	3*	0*	0*	0*	0	2*	2*	2*	2	2	1*
被置换的页				7		1		2	0		3	4		2	0	3		1		2

Clock页面置换算法的置换图

发生11次页面置换

蓝色为指针位置，*为访问标志

改进型Clock置换算法(UNIX SVR4采用此算法)

淘汰被修改过的页面时，需将其写回磁盘(置换代价高)，因此应淘汰既未被访问又未被修改的页面。为此，每个页面除了有访问位A外，还增加一个修改位M。由访问位A与修改位M可以组成下面4种类型的页面：

- 1类(A=0,M=0)，是最佳淘汰页；
- 2类(A=0,M=1)；
- 3类(A=1,M=0)；
- 4类(A=1,M=1)，最近被访问且被修改过的页，最不应该淘汰。

(1)从指针当前位置开始，扫描循环队列，寻找A=0且M=0的第1类页面，将所遇到的第一个页面淘汰。
(2)若第1步查找一周后未遇到第1类页面，则寻找A=0且M=1的第2类页面，将所遇到的第一个页面淘汰。第2轮扫描中将所有扫描过的页面的访问位A清0。
(3)若第2轮扫描失败，则返回(1)，若仍失败，再重复第(2)步，此时就一定能找到被淘汰的页。

淘汰原3类页　　淘汰原4类页

课后作业

1. 有一个页式虚存系统，某进程占用3个内存块，开始时内存为空，执行如下访问页号序列：

0，1，2，3，4，1，2，0，5，1，0，1，2，3，2，4，5

(1) 采用先进先出（FIFO）置换算法，缺页次数是多少？

(2) 采用LRU置换算法，缺页次数是多少？

(3) 若用最优（OPT）算法呢？

2. 在一个请求分页系统中，采用LRU页面置换算法时，假如一个作业的页面走向为1、3、2、1、1、3、5、1、3、2、1、5，当分配给该作业的物理块数M分别为3和4时，试计算在访问过程中所发生的缺页次数和缺页率，并比较所得结果。

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，只需调入若干分段（不必调入所有分段），便可启动运行。
- 当所访问的段不在内存时，可请求OS将所缺的段调入内存。
- 像请求分页系统一样，为实现请求分段存储管理功能，同样需要一定的硬件支持和相应的软件。

5.5.1 请求分段中的硬件支持

1. 段表机制

- 请求分段式管理中，所需的最主要数据结构是**段表**。
- 由于应用程序的段，只有一部分装入内存，故需在段表中增加若干项，以供程序在调进、调出时参考。

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

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

增加了以下诸项：

存取方式：用于标识本分段的存取属性是执行、只读、读/写

访问字段**A**：用于记录该分段被访问的频繁程度

修改位**M**：用于表示该段进入内存后是否被修改，供分段置换时参考

存在位**P**：指示本段是否已调入内存，供程序访问时参考

增补位：用于表示该段进入内存后是否做过动态增长

外存始址：指示本段在外存中的起始地址，即起始盘块号

2. 缺段中断机构

- 在请求分段系统中，每当发现运行进程所要访问的段不在内存时，便由缺段中断机构产生一缺段中断信号，进入OS后由缺段中断处理程序将所需的段调入内存。
- 同样需要在一条指令的执行期间，产生和处理中断（一条指令执行期间可能产生多次缺段中断）
- 缺段中断的处理比缺页中断复杂（段不是定长）

缺段中断的处理过程如图5-12所示。

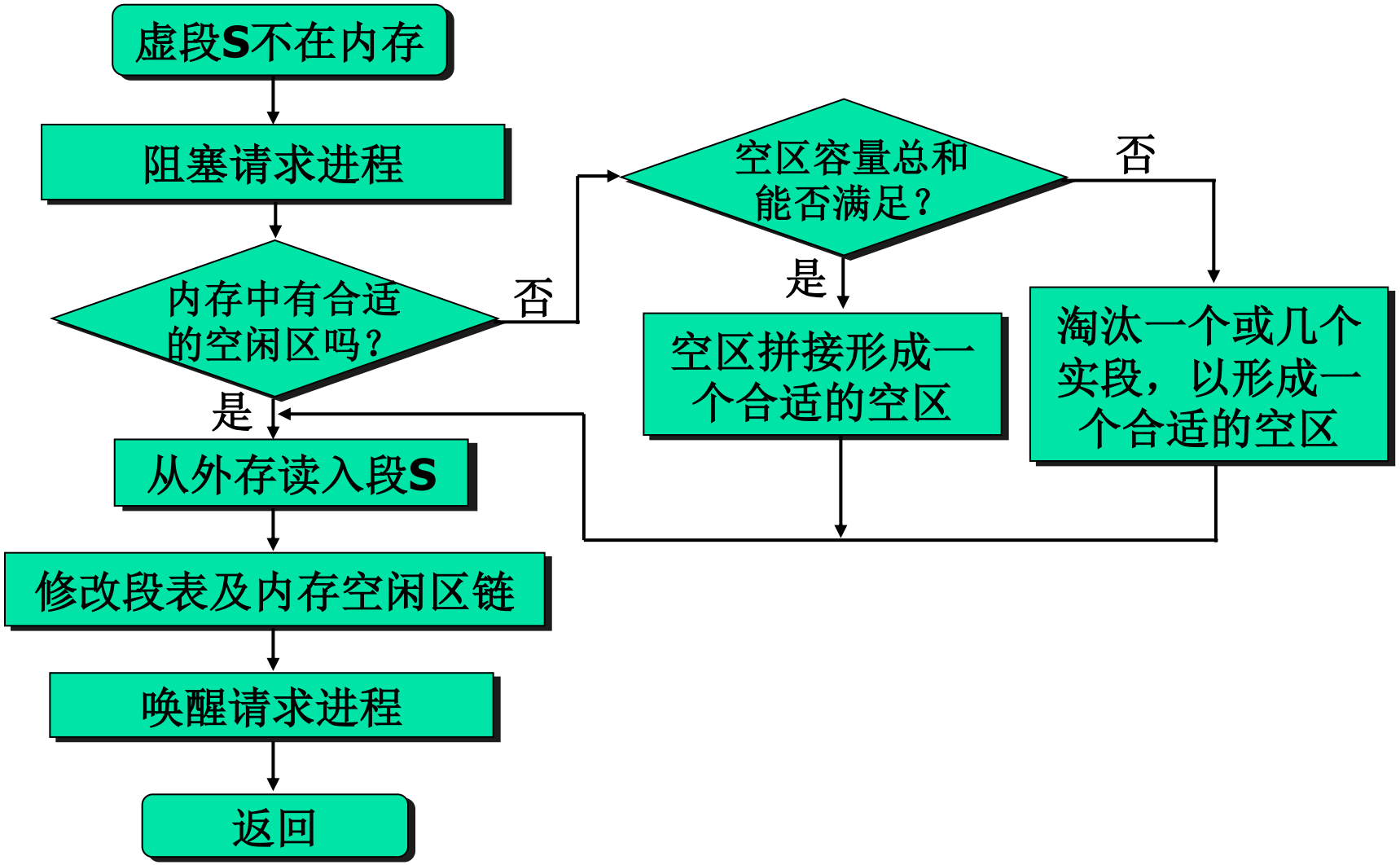


图5-12 请求分段系统中的中断处理过程

3. 地址变换机构

请求分段系统的地址变换机构，是在分段系统的地址变换机构的基础上形成的。

因为在地址变换时，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，然后才能利用段表进行地址变换。因此在地址变换机构中增加了缺段中断的请求和处理等功能。

请求分段系统的地址变换过程如图5-13所示。

S： 段号

W： 位移量

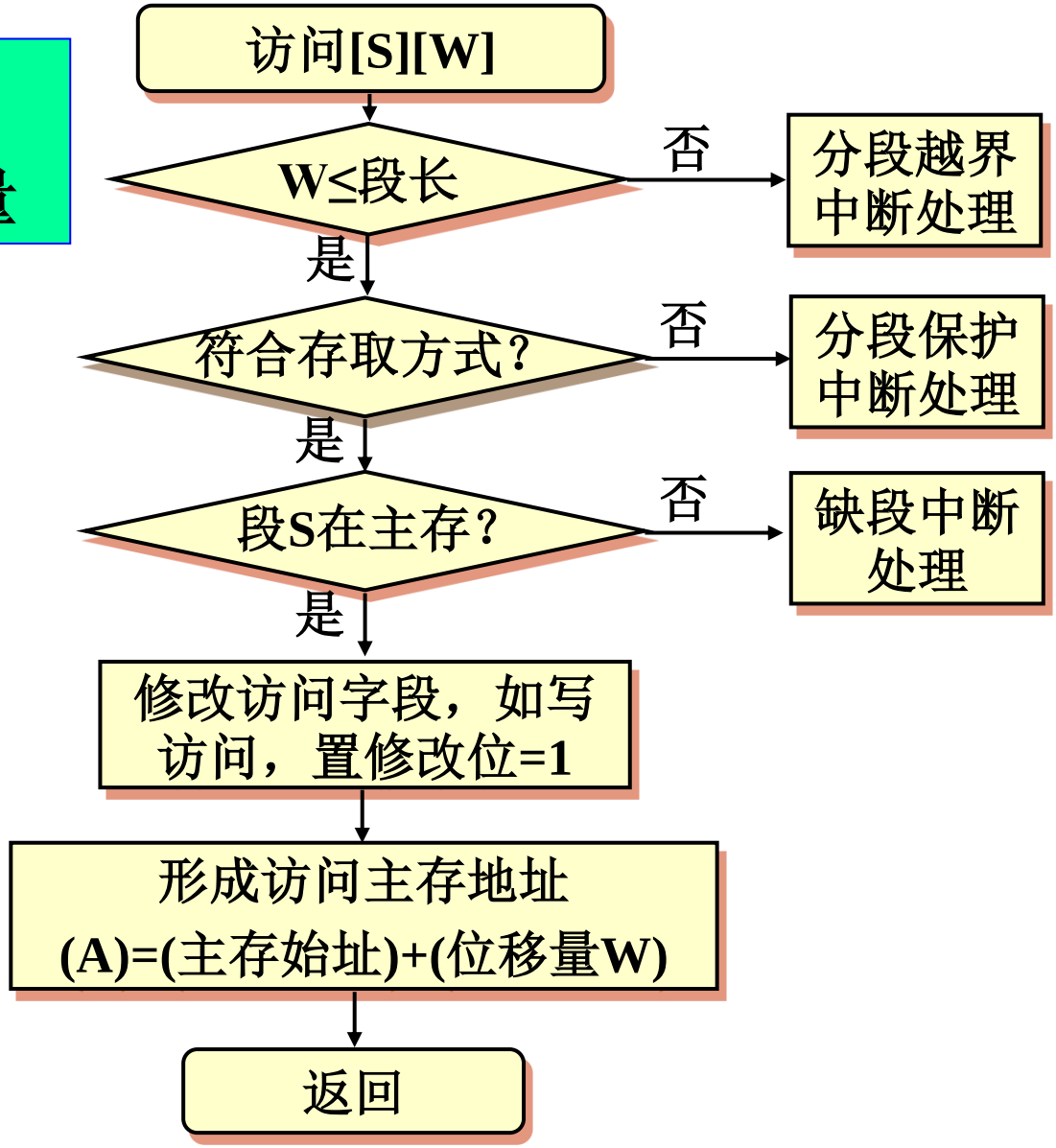


图5-13 请求分段系统的地址变换过程