

第5章 虚拟存储器 (3学时)



主讲教师：张春元

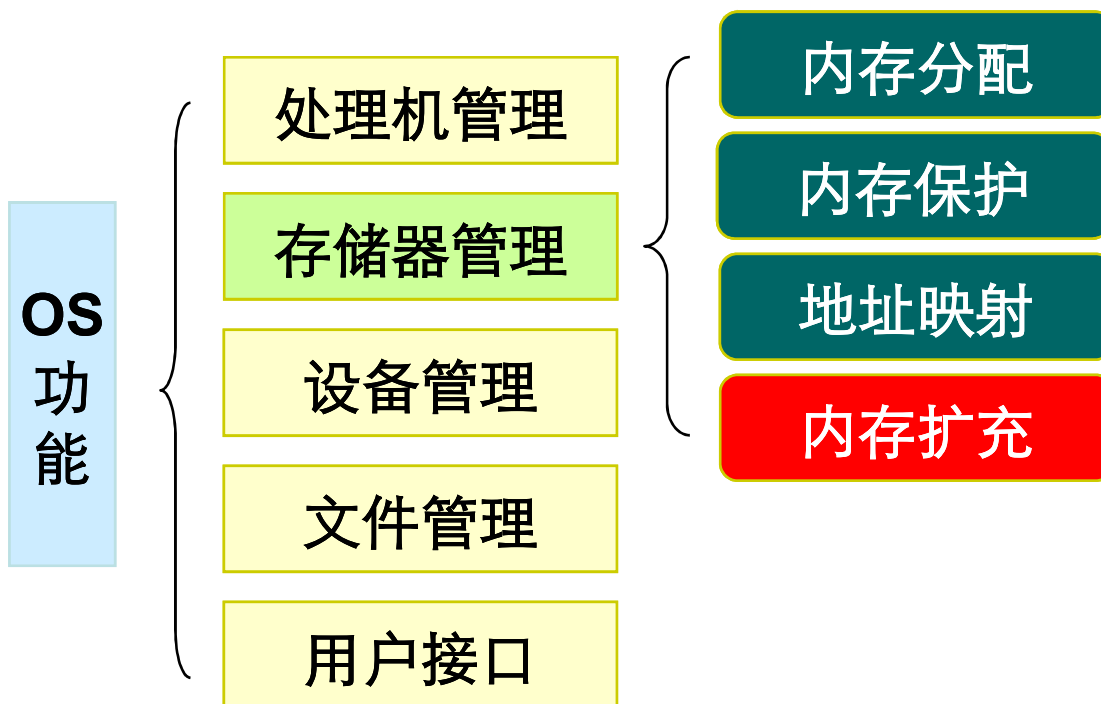
联系电话：13876004640

课程邮箱：haidaos@126.com

邮箱密码：[zhangchunyuan](#)



本章内容所处位置





本章主要内容

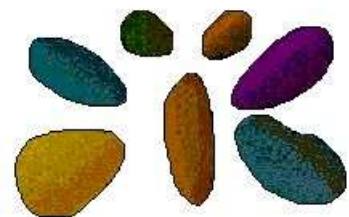
- ❖ 5.1 虚拟存储器概述
- ❖ 5.2 请求分页存储管理方式
- ❖ 5.3 页面置换算法
- ❖ 5.4 抖动与工作集
- ❖ 5.5 请求分段存储管理方式





5.1 虚拟存储器概述

- ❖ 5.1.1 常规存储管理方式的特征和局部性原理
- ❖ 5.1.2 虚拟存储器的定义和特征
- ❖ 5.1.3 虚拟存储器的实现方法





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

- 要求作业全部装入内存才能运行

* 2> 驻留性

- 作业装入内存后便一直驻留内存，直至运行结束

❖ 2、局部性原理（虚拟存储器的理论基石）

- * Denning P指出：程序在执行时将呈现出局部性规律，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。

- 1> 时间局限性：循环执行造成

- 2> 空间局限性：顺序执行造成



5.1.2 虚拟存储器的定义与特征

❖ 1、虚拟存储器的定义

- * 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扩充的一种存储系统。
- * 虚拟存储器的逻辑容量由内存容量和外存容量之和决定，其运行速度接近于内存速度，成本接近于外存。
- * 实质：以时间换空间，但时间牺牲不大。



5.1.2 虚拟存储器的定义与特征

❖ 2、虚拟存储器的特征

* 多次性 (虚拟存储器最重要的特征)

- 一个作业被分成多次调入内存运行

* 对换性

- 允许在作业的运行过程中进行换进、换出

* 虚拟性

- 能够从逻辑上扩充内存容量，用户可使用的内存容量远大于实际内存容量。



5.1.3 虚拟存储器的实现方法

虚拟存储器的实现也是建立在离散分配的存储管理方式基础上的，主要有如下两种实现方式：

❖ 1> 请求分页系统

- * **定义：**请求分页系统是在纯分页系统基础上，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。
- * **基本原理：**请求分页系统允许程序只装入少数页面便可启动运行，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。
- * **硬件要求：**页表机制；缺页中断机构；地址变换机构。
- * **软件要求：**页面请调、置换软件等



5.1.3 虚拟存储器的实现方法

❖ 2> 请求分段系统

- * **定义：**请求分段系统是在纯分段系统基础上，增加了请求调段功能和段的置换功能所形成的段式虚拟存储系统。
- * **基本原理：**请求分段系统允许程序只装入少数段便可启动运行，以后，再通过调段功能及段的置换功能，陆续地把即将要运行的段调入内存，同时把暂不运行的段换出到外存上。置换时以段为单位进行。
- * **硬件要求：**段表机制；缺段中断机构；地址变换机构。
- * **软件要求：**段的请调、置换软件等



本章主要内容

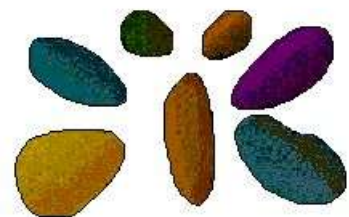
- ❖ 5.1 虚拟存储器概述
- ❖ 5.2 请求分页存储管理方式
- ❖ 5.3 页面置换算法
- ❖ 5.4 抖动与工作集
- ❖ 5.5 请求分段存储管理方式





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
- ❖ 5.2.2 内存分配策略和分配算法
- ❖ 5.2.3 调页策略





5.2.1 请求分页中的硬件支持

❖ 1、页表机制

- * 为了能提供请求调页和页面置换功能，相对于纯分页系统，请求分页系统的页表数据结构增加了四个字段：

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 — 用于指示该页是否已调入内存
- 访问字段A — 用于记录本页在一段时间内被访问的次数，或记录本页在最近多长时间未被访问
- 修改位M — 该页在调入内存后是否被修改过
- 外存地址 — 该页在外存上的地址，通常是物理块号



5.2.1 请求分页中的硬件支持

❖ 2、缺页中断机构

* 缺页中断

- 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将所缺页面调入内存。

* 缺页中断的特点（缺页中断和一般中断的区别）

- 1> 缺页中断在指令执行期间产生和处理，而一般中断要等到一条指令执行完才处理。
- 2> 一条指令在执行期间，可能产生多次缺页中断，系统中的硬件机构应能**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。



5.2.1 请求分页中的硬件支持

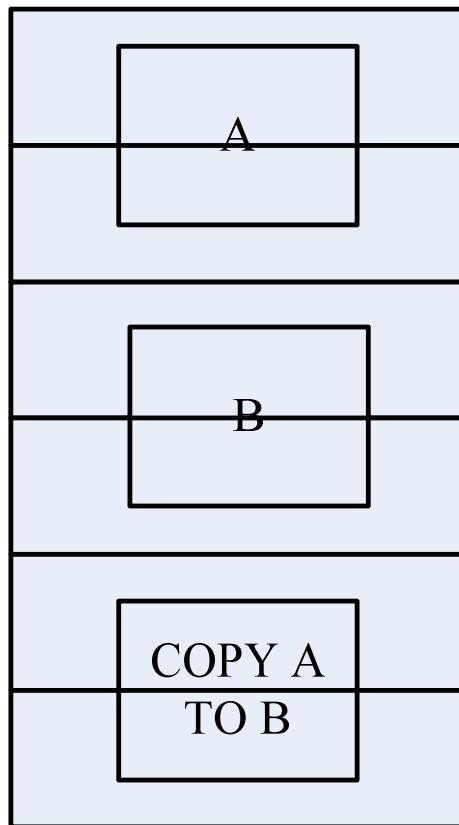
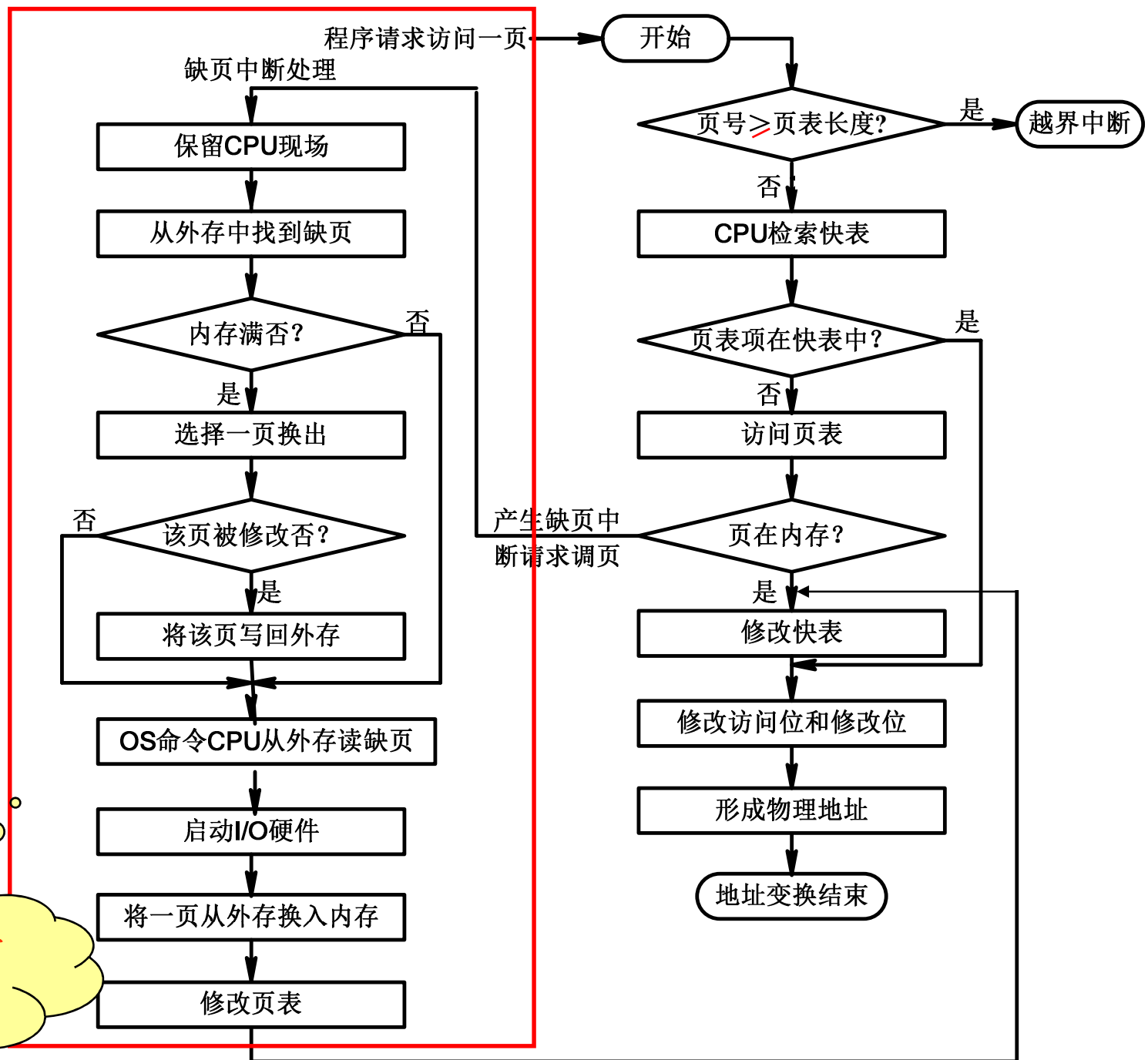


图5-1 涉及6次缺页中断的指令示例



地址变换机构

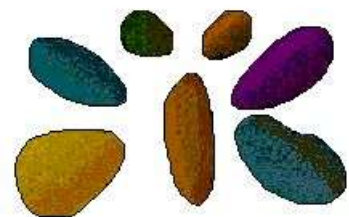
请求调页和页面置换处理





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
- ❖ 5.2.2 内存分配策略和分配算法
- ❖ 5.2.3 调页策略





5.2.2 内存分配策略和分配算法

❖ 1、最小物理块数确定

- * 最小物理块数是指能保证进程正常运行所需的最少物理块数，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。
- * 进程应获得的最少物理块数与计算机的硬件结构有关，取决于指令的格式、功能和寻址方式。
 - 直接寻址方式，最少需2个物理块
 - 指令、数据至少各需1个物理块
 - 例：MOV AX, [1234H]
 - 间接寻址方式，最少需3个物理块
 - 例：MOV AX, [BX]



5.2.2 内存分配策略和分配算法

❖ 2、物理块分配策略

* 分配策略

- **1> 固定分配策略：**分配给进程的物理块数是固定的，并在最初装入时（即进程创建时）确定块数。当进程执行过程中出现缺页时，只能从分给该进程的物理块中进行页面置换。
- **2> 可变分配策略：**允许分给进程的物理块数随进程的活动而改变。如果一个进程在运行过程中持续缺页率太高，这就表明该进程的局部化行为不好，需要给它分配另外的物理块，以减少它的缺页率。如果一个进程的缺页率特别低，就可以减少分配的物理块，但不要显著增加缺页率。



5.2.2 内存分配策略和分配算法

* 置换策略

- 1> 局部置换策略：每个进程只能从分给它的一组物理块中选择置换块。
- 2> 全局置换策略：允许一个进程从全体物理块（包括分配给别的进程的块）的集合中选取置换块，尽管该块当前已分给其他进程，但还是能强行剥夺。

* 常用分配与置换组合策略

- 1> 固定分配局部置换策略
 - 缺点：难以确定固定分配的物理块数（太少：置换率高；太多：浪费）
- 2> 可变分配全局置换策略
- 3> 可变分配局部置换策略
 - 根据进程的缺页率对所分配的物理块数调整，进程之间相互不会影响。



5.2.2 内存分配策略和分配算法

❖ 3、物理块分配算法

* 1> 平均分配算法

- **算法思想：**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

- **算法思想：**根据进程的大小按比例分配物理块。

$$S = \sum_{i=1}^n S_i \quad b_i = \frac{S_i}{S} \times m$$

其中： S 为各进程总页面数； b_i 为每个进程应分得的物理块数， b_i 应为整数且大于最小物理块数； m 为可用的总物理块数。



5.2.2 内存分配策略和分配算法

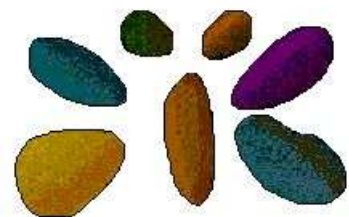
* 3> 考虑优先权的分配算法

- **算法思想**：对重要的、紧迫的进程为它分配较多的内存空间。**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 请求分页存储管理方式

- ❖ 5.2.1 请求分页中的硬件支持
- ❖ 5.2.2 内存分配策略和分配算法
- ❖ 5.2.3 调页策略





5.2.3 调页策略

❖ 1、调入页面的时机

* 1> 预调页策略

* **基本思想**：将那些预计在不久之后便会被访问的页面预先调入内存。

* **缺点**：成功率约50%，适于进程首次调入时使用

* 2> 请求调页策略

* **基本思想**：根据需要将需要访问的缺页由OS将其调入内存。

* **缺点**：每次仅调入一页，系统开销大

* **注**：目前的虚拟存储系统大多采用此种策略。



5.2.3 调页策略

❖ 2、确定从何处调入页面

- * **对换区**：采用连续分配方式，速度快，一般修改过、运行过的页面被换出时应放入对换区，需要时再从对换区换入。
- * **文件区**：采用离散分配方式，速度稍慢，如果对换区空间不够用，则将不会被修改的页面、未运行过的页面放在文件区。
- * **对共享页面**，应判断其是否在内存区，如在则无需调入。

❖ 3、页面调入过程

- * 缺页中断 + 页面置换



5.2.3 调页策略

❖ 4、缺页率

* 缺页率：

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

其中： F 为缺页次数， A 为页面总访问数

* 缺页中断处理时间：

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

其中： β 为缺页时被置换出的页面的修改概率，
其缺页中断处理时间为 t_a ； t_b 为缺页时被置换出的
页面的没有修改的中断处理时间



5.2.3 调页策略

* 缺页率影响因素

- (1) 页面大小
 - 页面越大，缺页率越低
- (2) 进程所分配物理块的数目
 - 一般而言，分配的物理块越多，缺页率越低
- (3) 页面置换算法
- (4) 程序固有特性
 - 程序的局部化程度越高，缺页率越低



本章主要内容

- ❖ 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 抖动与工作集
- ❖ 5.5 请求分段存储管理方式





5.3 页面置换算法

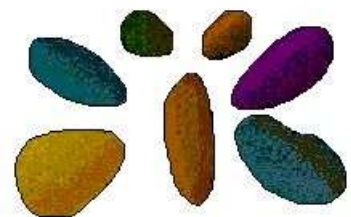
❖ 目的：减少对换量，提高系统性能

- * 1、最佳置换算法
- * 2、先进先出（FIFO）置换算法
- * 3、最近最久未使用（LRU）置换算法
- * 4、最少使用（LFU）置换算法
- * 5、Clock置换算法
- * 6、页面缓冲（PBA）置换算法



5.3 页面置换算法

- ❖ 5.3.1 最佳置换算法和先进先出置换算法
- ❖ 5.3.2 最近最久未使用和最少使用置换算法
- ❖ 5.3.3 Clock置换算法
- ❖ 5.3.4 页面缓冲置换算法
- ❖ 5.3.5 访问内存的有效时间





5.3.1 最佳置换算法和先进先出置换算法

❖ 1、最佳 (OPT) 置换算法

* 算法思想

- 选择以后永不使用的或是在(未来)最长时间内不再被访问的页面作为淘汰页面置换出去。

* 算法特点

- 1> 理想化算法，具有最好的性能，可使缺页率最低。
- 2> 难以确定理想的淘汰页面，因此算法难以实现。



5.3.1 最佳置换算法和先进先出置换算法

❖ 例：假定系统为某进程分配了三个物理块，进程运行将进行页号引用：7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1，采用最佳置换算法如何置换？

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		

缺页次数：9次(包括前3个)，缺页率：9/20=45%

置换次数：6次



5.3.1 最佳置换算法和先进先出置换算法

❖ 2、先进先出（FIFO）置换算法

* 算法思想

- 总是淘汰最先进入内存的页面，即选择在内存驻留时间最长的页面予以淘汰。

* 算法实现

- 将进程在内存中页面按先后次序链接成一个队列，并设置一个指针，称为替换指针，使它总是指向最老的页面。

* 算法特点

- 简单、易实现；貌似公平，实际上不公平，不切实际，有些经常被访问的页面可能先被淘汰，因此性能较差。



5.3.1 最佳置换算法和先进先出

是否页框数增加就一定减少缺页数呢?

❖ 例：假定系统为某进程分配了三个物理块 进程运行
将进行页号引用：7, 0, 1, 2, 0, 3, 0, 4, 2, 3,
0, 3, 2, 1, 2, 0, 1, 7, 0, 1, 采用FIFO置换算
法如何置换?

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

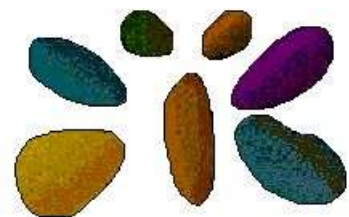
缺页次数：15次(包括前3个)，缺页率：15/20=75%

置换次数：12次



5.3 页面置换算法

- ❖ 5.3.1 最佳置换算法和先进先出置换算法
- ❖ 5.3.2 最近最久未使用和最少使用置换算法
- ❖ 5.3.3 Clock置换算法
- ❖ 5.3.4 页面缓冲置换算法
- ❖ 5.3.5 访问内存的有效时间





5.3.2 最近最久未使用和最少使用置换算法

❖ 1、最近最久未使用（LRU）置换算法的描述

* 算法思想

- 选择最近一段时间最久未使用的页面予以淘汰。

* 算法实现

- 采用寄存器或栈记录各页面自上次被访问以来所经历的时间 t ，每次从中选择 t 值最大的一个页面进行淘汰。

* 算法特点

- 考虑了程序设计的局部性原理，有一定合理性，但页面的过去和未来走向无必然联系；需要跟踪记录每一页的使用情况，系统开销较大。



5.3.2 最近最久未使用和最少使用置换算法

❖ **例：**假定系统为某进程分配了三个物理块，进程运行将进行页号引用：7，0，1，2，0，3，0，4，2，3，0，3，2，1，2，0，1，7，0，1，采用LRU置换算法如何置换？

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

缺页次数：12次(包括前3个)，缺页率：12/20=60%

置换次数：9次



5.3.2 最近最久未使用和最少使用置换算法

* LRU置换算法的硬件支持

■ 1> 寄存器

- 寄存器的表示：为记录某进程各页在最近一段时间内的使用情况，为内存中各页面配置一个移位寄存器，n位寄存器可表示为：

$$R = R_{n-1} R_{n-2} R_{n-3} \dots R_2 R_1 R_0$$

- 寄存器的用法：某页面如被访问，则将其寄存器的 R_{n-1} (即最高位)置1；定时将寄存器右移1位；如要置换页面，选择R值最小的页面进行淘汰



图 两个页面的8位寄存器移位情况

t0

	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

t1

	R ₇	R ₆	R ₅	R ₄	R ₃	R ₂	R ₁	R ₀	
1	0	0	1	0	1	0	0	1	0
2	0	1	0	1	0	1	1	0	0

访问页面1

	R ₇	R ₆	R ₅	R ₄	R ₃	R ₂	R ₁	R ₀
1	1	0	1	0	1	0	0	1
2	0	1	0	1	0	1	1	0

t2

	R ₇	R ₆	R ₅	R ₄	R ₃	R ₂	R ₁	R ₀	
1	0	1	0	1	0	1	0	0	1
2	0	0	1	0	1	0	1	1	0



图 5-6 某进程具有8个页面时的LRU访问情况

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

最近被访问

页面3的R值最小，应首先被淘汰



5.3.2 最近最久未使用和最少使用置换算法

* LRU置换算法的硬件支持（续）

- 2> 栈
- 当进程访问某页时，将其移出压入“栈顶”，如要置换页面，将“栈底”页面换出。
- 注意：此处的栈比较特殊，一般的栈是先进后出，队列才是先进先出。



5.3.2 最近最久未使用和最少使用置换算法

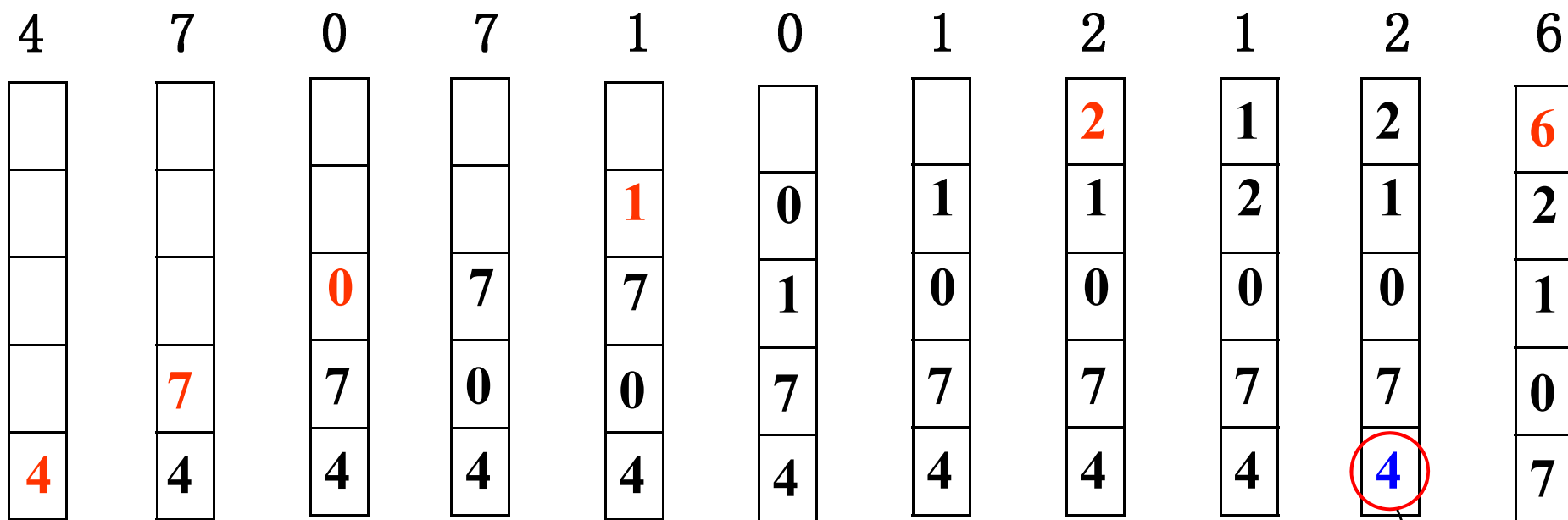


图5-7 用栈保存当前使用页面时栈的变化情况

出栈



5.3.2 最近最久未使用和最少使用置换算法

❖ 2、最少使用（LFU）置换算法

* 算法思想

- 与LRU类似，设置一个移位寄存器记录页面访问情况，如要置换页面，选择 $\sum R_i$ 值最小的页面进行淘汰。

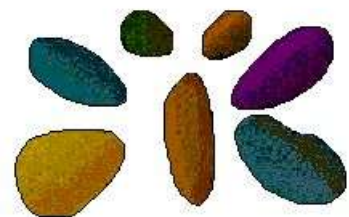
* 算法缺陷

- LFU算法并不能真正反映出页面的使用情况，因为在每一时间间隔内，只是用寄存器的一位来记录页的使用情况，因此访问一次和访问10000次是等效的。



5.3 页面置换算法

- ❖ 5.3.1 最佳置换算法和先进先出置换算法
- ❖ 5.3.2 最近最久未使用和最少使用置换算法
- ❖ 5.3.3 Clock置换算法
- ❖ 5.3.4 页面缓冲置换算法
- ❖ 5.3.5 访问内存的有效时间





5.3.3 Clock置换算法

❖ 1、简单型Clock置换算法（最近未用算法NRU）

* 算法思想

- 为每页 设置一位访问位，再将内存中所有页面通过链接指针链成一个 循环队列。当某页被访问时，其访问位被置1，表示该页最近使用过。置换算法在选择一页淘汰时，只需 循环检查各页的访问位：如果为1，则将该访问位置0，暂不换出；如果为0，则将该页换出，算法终止。
- 简单型Clock算法 每次选择的淘汰页面均是最近未使用的页面，因此该算法也称最近未用算法。

5.3.3 Clock置换算法

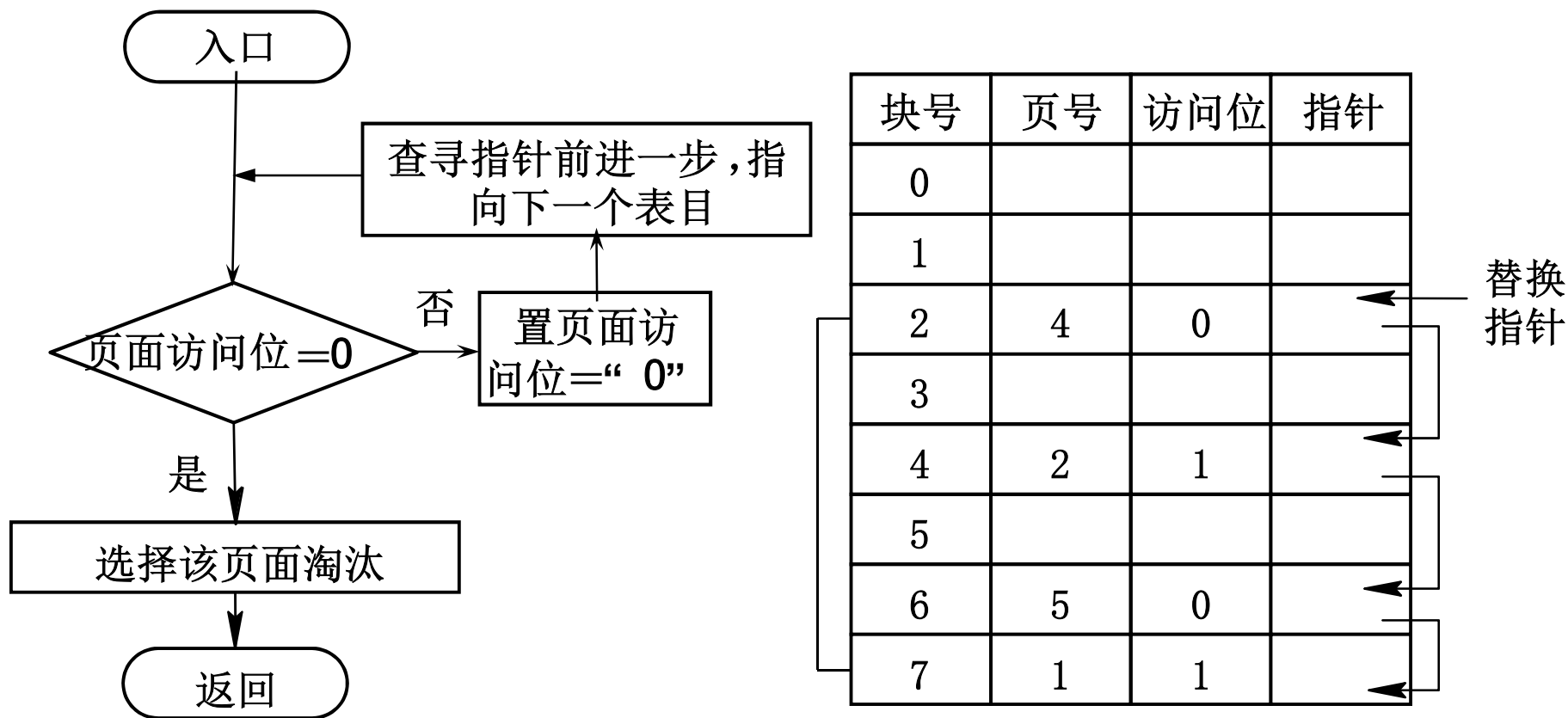
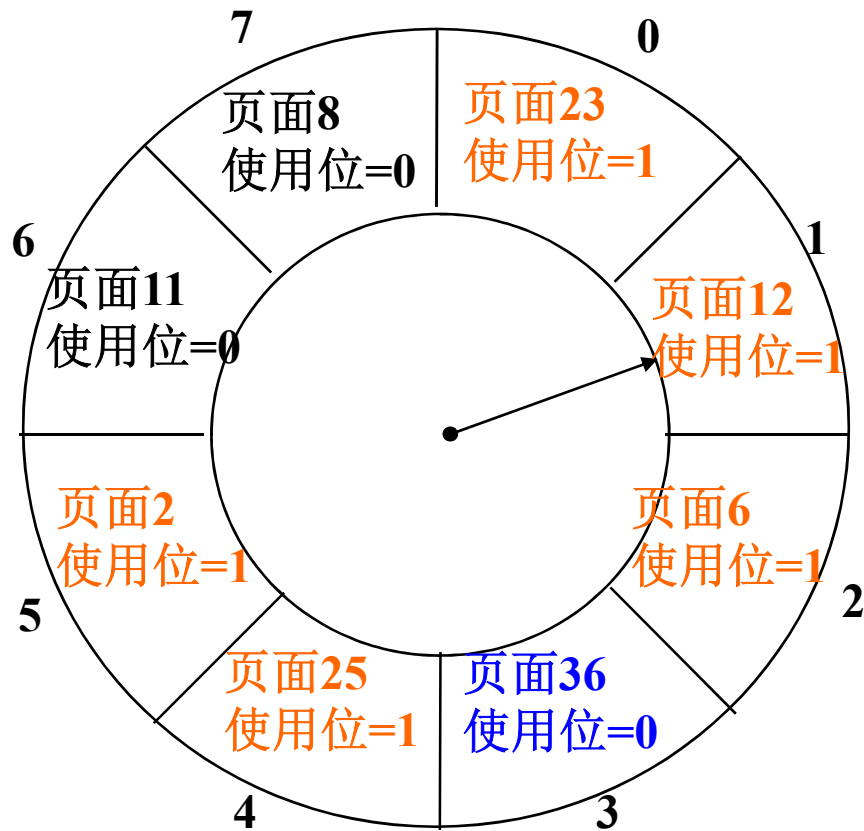


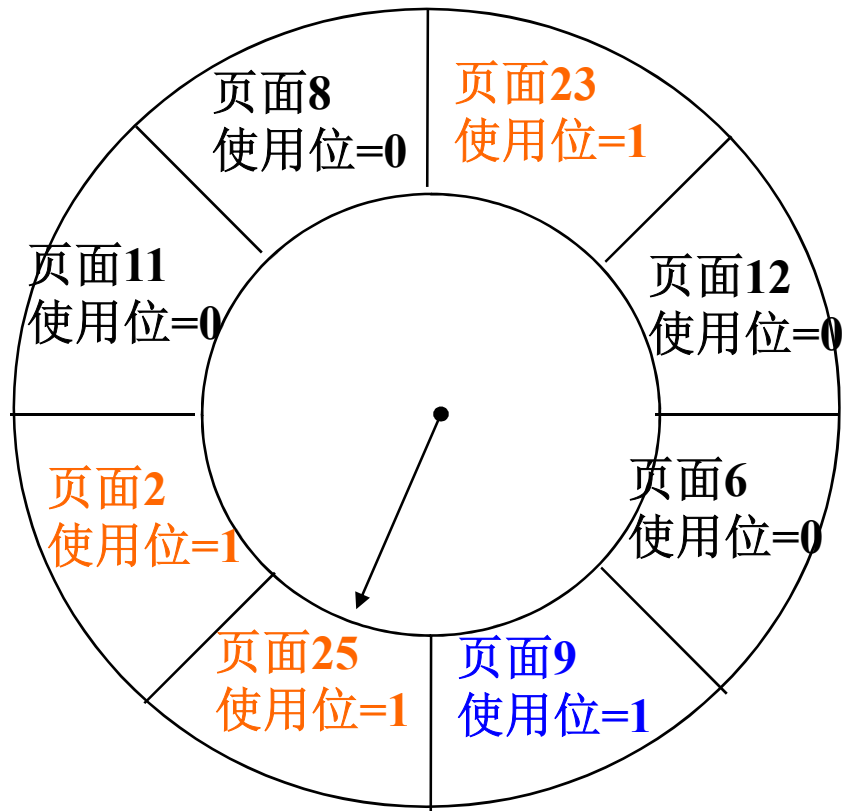
图 4-31 简单型Clock置换算法的流程和示例



5.3.3 Clock置换算法



(a) 页面置换前状态



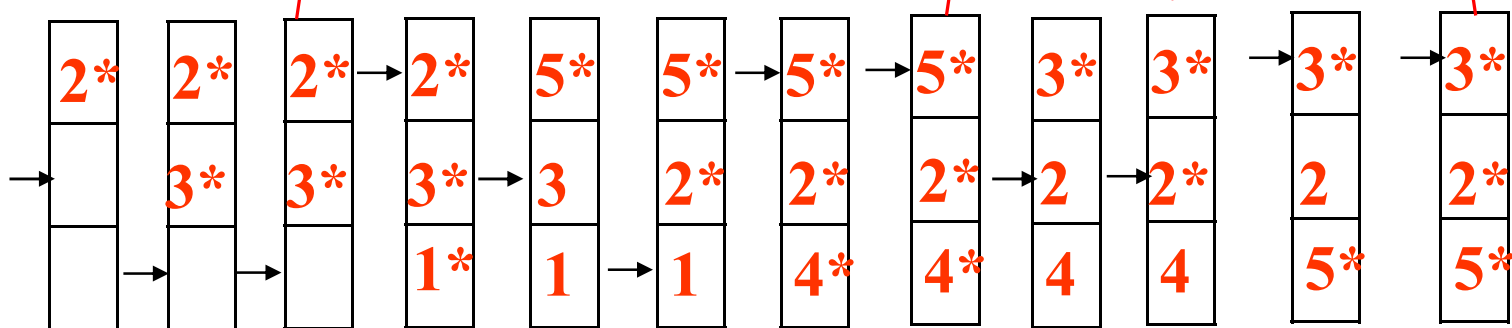
(b) 页面置换后状态

图 简单型Clock置换算法示例

注：如果需要访问的页面已在块中，原指针应保持不动，但这个页面的访问位应修改成1（操作系统精髓与设计原理第6版 william stallings 机械工业出版社 P247）

❖ **例：**假定系统为某进程分配了三个物理块，进程运行将进行页号引用：2，3，2，1，5，2，4，5，3，2，5，2，采用Clock置换算法如何置换？

2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2



缺页

Y

Y

Y

Y

Y

Y

Y

Y

缺页次数：8次(包括前3个)，缺页率：8/12=66.7%

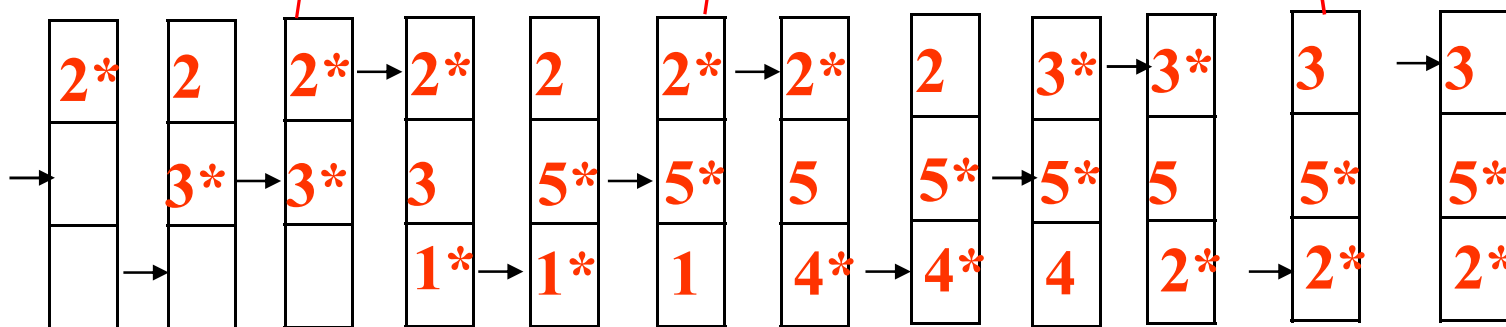
置换次数：5次

注：页框中的*表示访问位为1，否则为0

另一种说法：如果需要访问的页面已在块中，指针照样移动，这个页面的访问位应修改成1（易于实现）。

❖ 例：假定系统为某进程分配了三个物理块，进程运行将进行页号引用：2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2，采用Clock置换算法如何置换？

2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2



缺页 Y Y Y Y Y Y Y

缺页次数：7次(包括前3个)，缺页率：7/12=58%

置换次数：5次

注：页框中的*表示访问位为1，否则为0



5.3.3 Clock置换算法

没修改过的
页置换出后不用
重写回外存

❖ 2、改进型Clock置换算法

* 算法思想

- 改进型Clock算法每次选择的淘汰页面除了**最近未被使用过**，最好还**未被修改过**（使置换代价尽可能小）。
- 每页设置一个访问位A和一个修改位M

第一类页面	A=0	M=0	最佳淘汰页面
第二类页面	A=0	M=1	次佳淘汰页面
第三类页面	A=1	M=0	该页可能再被访问
第四类页面	A=1	M=1	该页可能再被访问



5.3.3 Clock置换算法

* 算法步骤

- **第1步**（第1轮扫描）：寻找第一类页面，将所遇到的第1个第一类页面作为淘汰页面，如果找不到，则转入第2步。
- **第2步**（第2轮扫描）：寻找第二类页面，将所遇到的第1个第二类页面作为淘汰页面，如果找不到，则转入第3步。**在扫描期间，将所有扫描过的页面访问位置0。**
- **第3步**：将指针返回到开始的位置，转第1步。

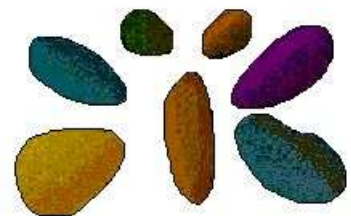
* 算法特点

- 减少了磁盘I/O次数；但可能要经过多轮扫描



5.3 页面置换算法

- ❖ 5.3.1 最佳置换算法和先进先出置换算法
- ❖ 5.3.2 最近最久未使用和最少使用置换算法
- ❖ 5.3.3 Clock置换算法
- ❖ 5.3.4 页面缓冲置换算法
- ❖ 5.3.5 访问内存的有效时间





5.3.4 页面缓冲置换算法

❖ 1、影响页面换进换出效率的若干因素

* (1) 页面置换算法（影响页面换进换出效率最重要的因素）

- 好的页面置换算法应具有较低的缺页率，可有效减少页面换进换出的开销

* (2) 写出磁盘的频率

- 应尽可能集中写出，以减少写出磁盘的频率
- 在内存中建立一个已修改拟换出页面链表，用来暂存拟换出页面（修改过的），集中到一定数量再写出

* (3) 读入内存的频率

- 如果要访问的页面在已修改换出页面链表中，可直接从中取回，而不用从磁盘读入。

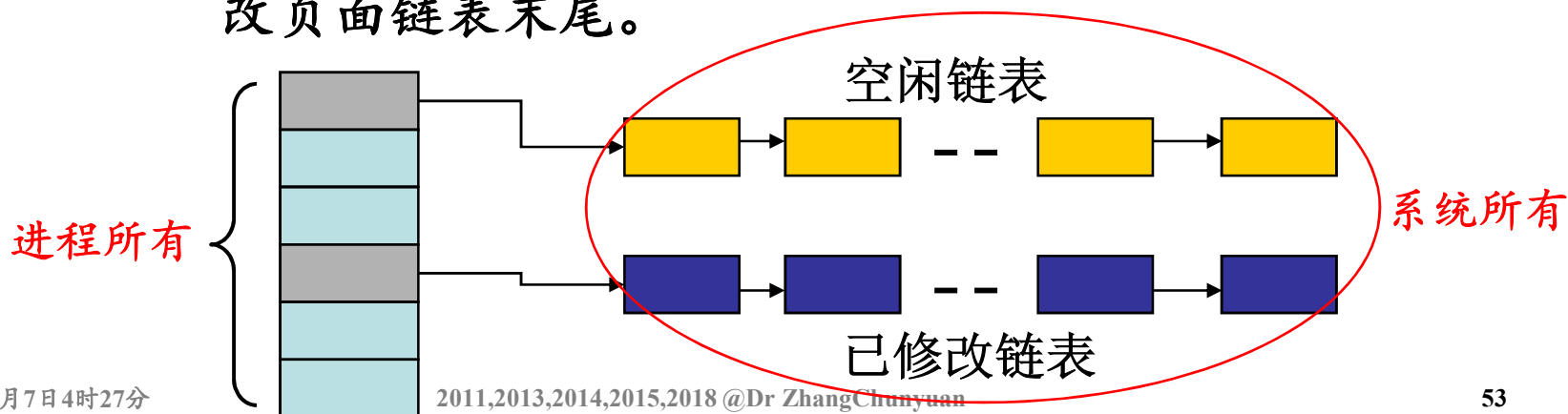


5.3.4 页面缓冲置换算法

❖ 2、页面缓冲（PBA）置换算法

* 算法思想

- PBA采用可变分配和局部置换方式，置换算法采用FIFO。
- 该算法在内存中设置1个空闲物理块链表和1个已修改页面链表，置换时如果被淘汰页面未被修改，就将其直接挂在空闲链表末尾，并从空闲链表表首取出一个空闲块用来装入缺页；否则，将其挂在已修改页面链表末尾。





5.3.4 页面缓冲置换算法

- 当已修改页面链表中的页面达到一定数量时，例如64个页面，再将它们一起写回磁盘，从而减少了磁盘I/O的操作次数。
- 当有位于两个链表中的页面又要访问时，可以无需访问外存，直接从这两个链表中恢复，减少开销，从而起到缓冲的作用。

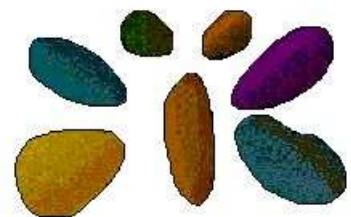
* 算法特点

- (1) 显著地降低了页面换进换出的频率。
- (2) 可以采用较简单的置换策略，如FIFIO，不需要特殊硬件的支持，实现简单。



5.3 页面置换算法

- ❖ 5.3.1 最佳置换算法和先进先出置换算法
- ❖ 5.3.2 最近最久未使用和最少使用置换算法
- ❖ 5.3.3 Clock置换算法
- ❖ 5.3.4 页面缓冲置换算法
- ❖ 5.3.5 访问内存的有效时间





5.3.5 访问内存的有效时间

- ❖ 1、被访问页在内存中且其对应的页表项在快表中
 - * 无缺页中断
 - * $EAT = \text{查找快表的时间} + \text{访问1次内存的时间}$
- ❖ 2、被访问页在内存中且其对应的页表项不在快表中
 - * 无缺页中断
 - * 快表需访问两次：查快表、修改快表
 - * $EAT = 2 * (\text{查找快表的时间} + \text{访问1次内存的时间})$
- ❖ 3、被访问页不在内存中
 - * $EAT = 2 * (\text{查找快表的时间} + \text{访问1次内存的时间})$
+ 中断处理时间



5.3.5 访问内存的有效时间

- ❖ **【补充示例1】** 现有一请求调页系统，页表保存在寄存器中。若有一个被替换的页未被修改过，则处理一个缺页中断需要8ms；若被替换的页已被修改过，则处理一个缺页中断需要20ms；内存存取时间为1 μ s，访问页表的时间可忽略不计。假定70%被替换的页被修改过，为保证平均有效存取时间不超过2 μ s，可接受的最大**缺页率**是多少？

注：（**缺页率** = 缺页次数/总的页面访问次数）



5.3.5 访问内存的有效时间

❖ **解：**如果用 p 表示缺页率，则有效存取时间不超过 $2\mu s$ 可表示为：

$$(1-p) \times 1\mu s + p \times (0.7 \times 20ms + 0.3 \times 8ms + 1\mu s) \leq 2\mu s$$

因此可计算出：

$$p \leq 1/16400 \approx 0.00006$$

页面在内存中

页面不在内存中（缺页）

被置换的页没有修改过

被置换的页修改过



本章主要内容

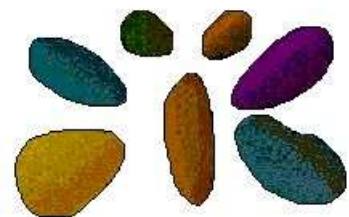
- ❖ 5.1 虚拟存储器概述
- ❖ 5.2 请求分页存储管理方式
- ❖ 5.3 页面置换算法
- ❖ 5.4 抖动与工作集
- ❖ 5.5 请求分段存储管理方式





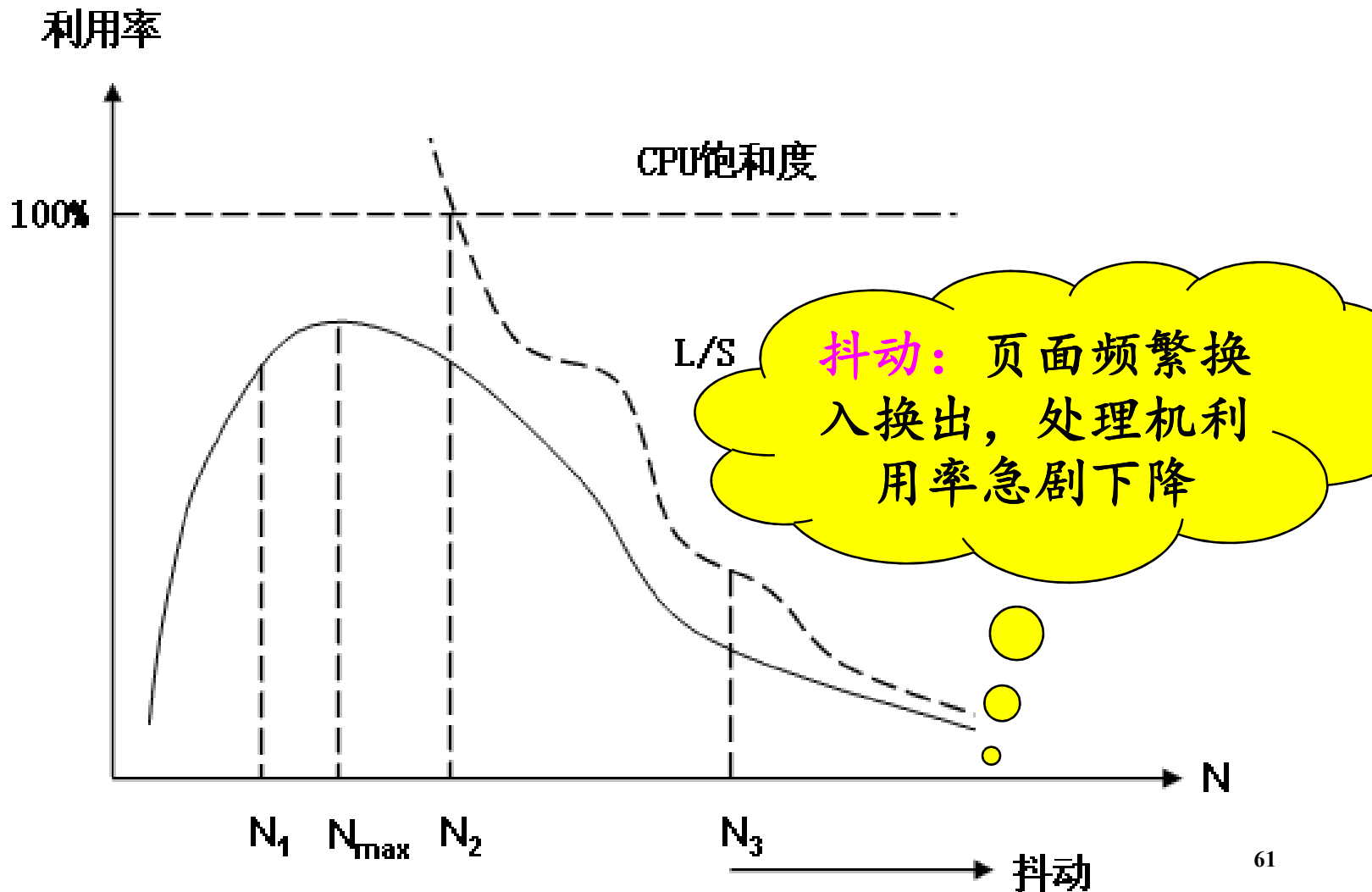
5.4 抖动与工作集

- ❖ 5.4.1 多道程序度与抖动
- ❖ 5.4.2 工作集
- ❖ 5.4.3 抖动的预防方法



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





5.4.1 多道程序度与抖动

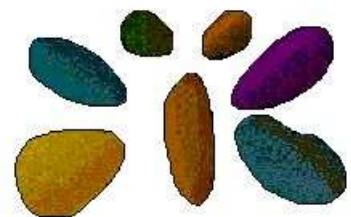
❖ 2、产生抖动的原因

- * 进程太多，分配给每个进程的物理块太少，从而缺页率太高
- * 页面淘汰算法不合理



5.4 抖动与工作集

- ❖ 5.4.1 多道程序度与抖动
- ❖ 5.4.2 工作集
- ❖ 5.4.3 抖动的预防方法

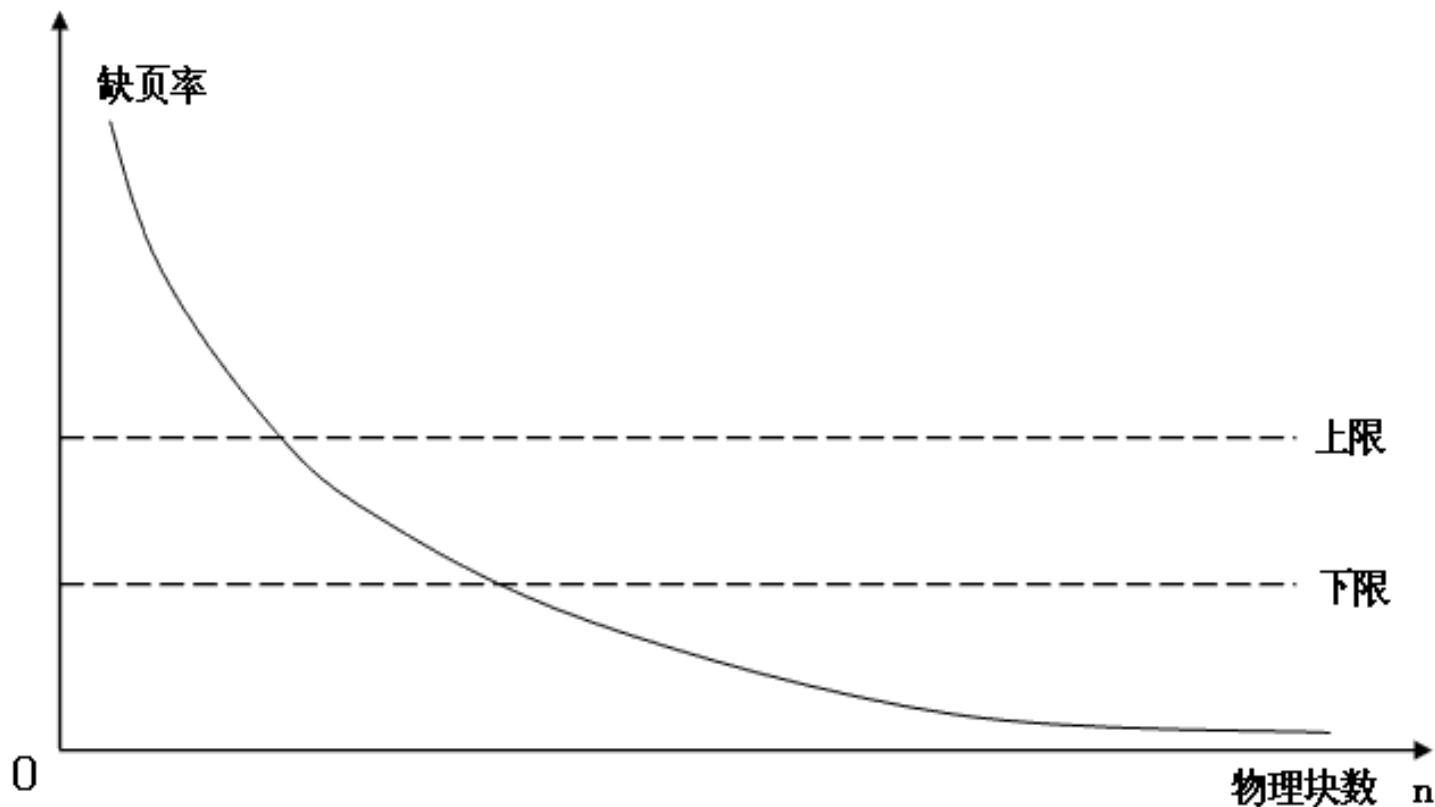




5.4.2 工作集

❖ 1、缺页率与物理块数之间的关系

* 缺页率与进程所获得的物理块数有关。





5.4.2 工作集

❖ 2、工作集的定义

- * **工作集**：进程在时间间隔 $(t-\Delta, t)$ 中引用页面的集合，记作 $w(t, \Delta)$ ，其中 Δ 称为工作集的**窗口尺寸**。
- * 工作集 $w(t, \Delta)$ 是二元函数，即在不同时间 t 的工作集大小不同，所含的页面数也不同；与窗口尺寸 Δ 有关，**工作集大小是窗口尺寸 Δ 的非降函数**：

$$w(t, \Delta) \subseteq w(t, \Delta+1)$$



5.4.2 工作集

窗口大小

引用页序列

24
15
18
23
24
17
18
24
18
17
17
15
24
17
24
18

3

24
15 24
18 15 24
23 18 15
24 23 18
17 24 23
18 17 24
.
.
.
.
15 17 18
24 15 17
.
.
18 24 17

4

24
15 24
18 15 24
23 18 15 24
.
17 24 23 18
.
.
.
.
15 17 18 24
.
.
.
.

5

24
15 24
18 15 24
23 18 15 24
.
17 24 23 18 15
.
.
.
.
.
.
.
.
.



5.4.2 工作集

❖ 3、抖动的预防方法

* (1) 采取可变分配局部置换策略

- 即使某进程发生抖动，也不会对别的进程产生影响，但该进程会长期处于磁盘I/O等待队列中，从而会延长别的进程缺页中断处理时间。

* (2) 把工作集算法融入到处理机调度中

- 在调度中融入了工作集算法，则在调度程序准备从外存调入新作业之前，必须先检查每个进程在内存的驻留页面是否足够多，而不是单纯依据CPU利用率来调入新作业。



5.4.2 工作集

* (3) 利用 “ $L=S$ ” 准则调节缺页率

- L 是缺页之间的平均时间。
- S 是平均缺页服务时间，即用于置换一个页面所需的时间。
- $L > S$ 表明较少缺页； $L < S$ 表明缺页频繁； $L = S$ 表明磁盘和处理机可达到它们的最大利用率。

* (4) 选择暂停的进程

- 当多道程序度偏高时，已影响到处理机的利用率，为了防止发生“抖动”，系统必须减少多道程序的数目，一般选取优先级低的进程暂停。



本章主要内容

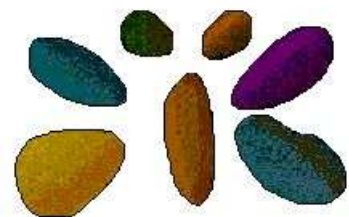
- ❖ 5.1 虚拟存储器概述
- ❖ 5.2 请求分页存储管理方式
- ❖ 5.3 页面置换算法
- ❖ 5.4 抖动与工作集
- ❖ 5.5 请求分段存储管理方式





5.5 请求分段存储管理方式

- ❖ 5.5.1 请求分段存储管理方式
- ❖ 5.5.2 分段的共享与保护





5.5.1 请求分段中的硬件支持

❖ 1、段表机制

- * 为了能提供请求调段和段的置换功能，相对于纯分段系统，请求分段系统的段表数据结构增加了六个字段：

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

- **存取方式** —标识本段存取属性（只执行、只读或可读写）
- **访问字段A** —记录本段在一段时间内被访问的次数，或记录本段在最近多长时间未被访问
- **修改位M** —表示本段在调入内存后是否被修改过
- **存在位P** —用于指示本段是否已调入内存
- **增补位** —本段在运行过程中是否做过动态增长
- **外存始址** —本段在外存上的起始地址



5.5.1 请求分段中的硬件支持

❖ 2、缺段中断机构

* 缺段中断

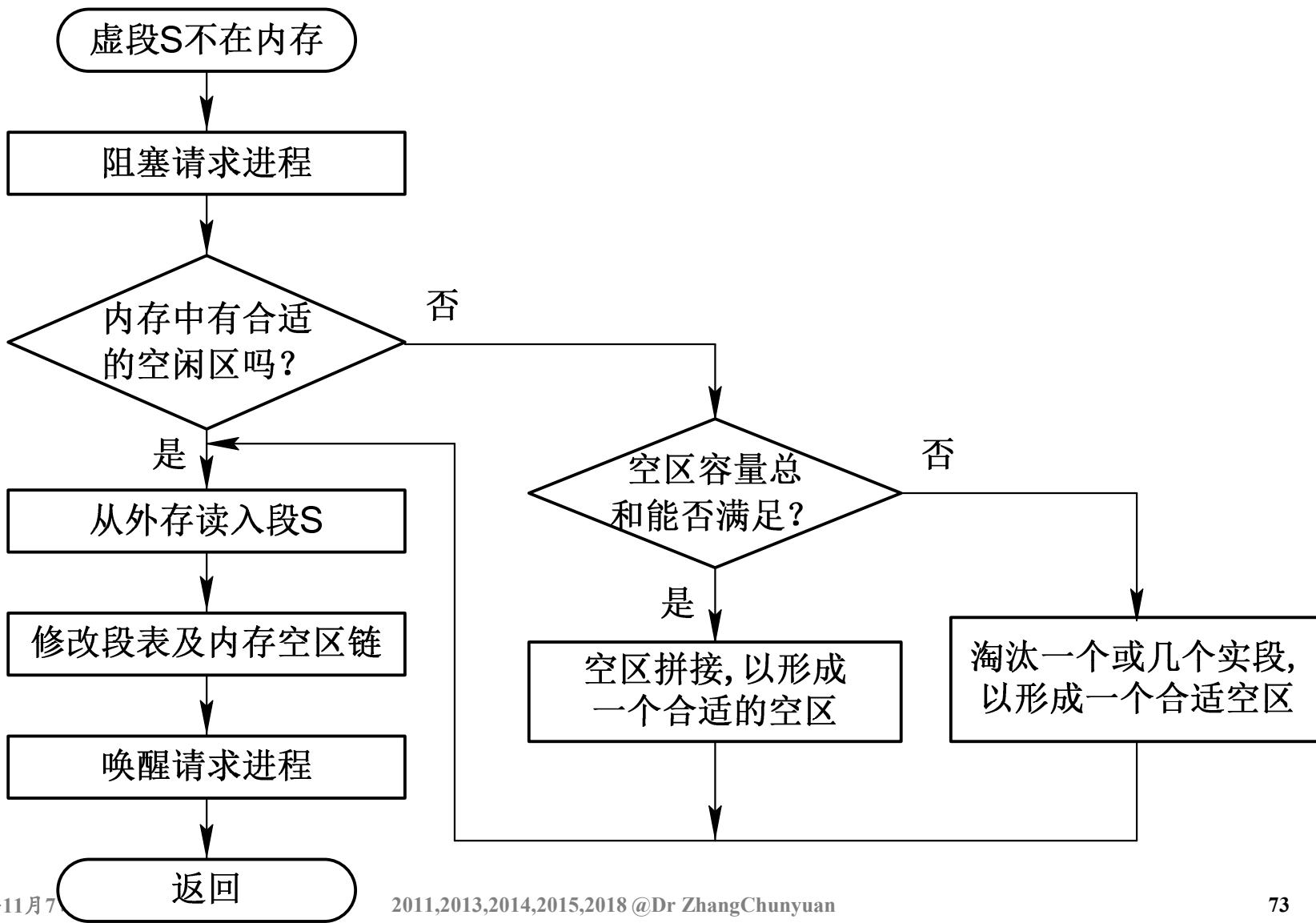
- 在请求分段系统中,每当所要访问的段不在内存时,便产生一缺段中断,请求OS将所缺的段调入内存。

* 缺段中断的特点

- 1> 与缺页中断类似: 在指令执行期间产生和处理中断信号, 且一条指令在执行期间可能产生多次缺段中断。
- 2> 与缺页中断不同: 不可能出现一条指令或一个逻辑单位信息被分割在两个分段之中; 另各段长度不一, 缺段中断的处理比缺页中断复杂。

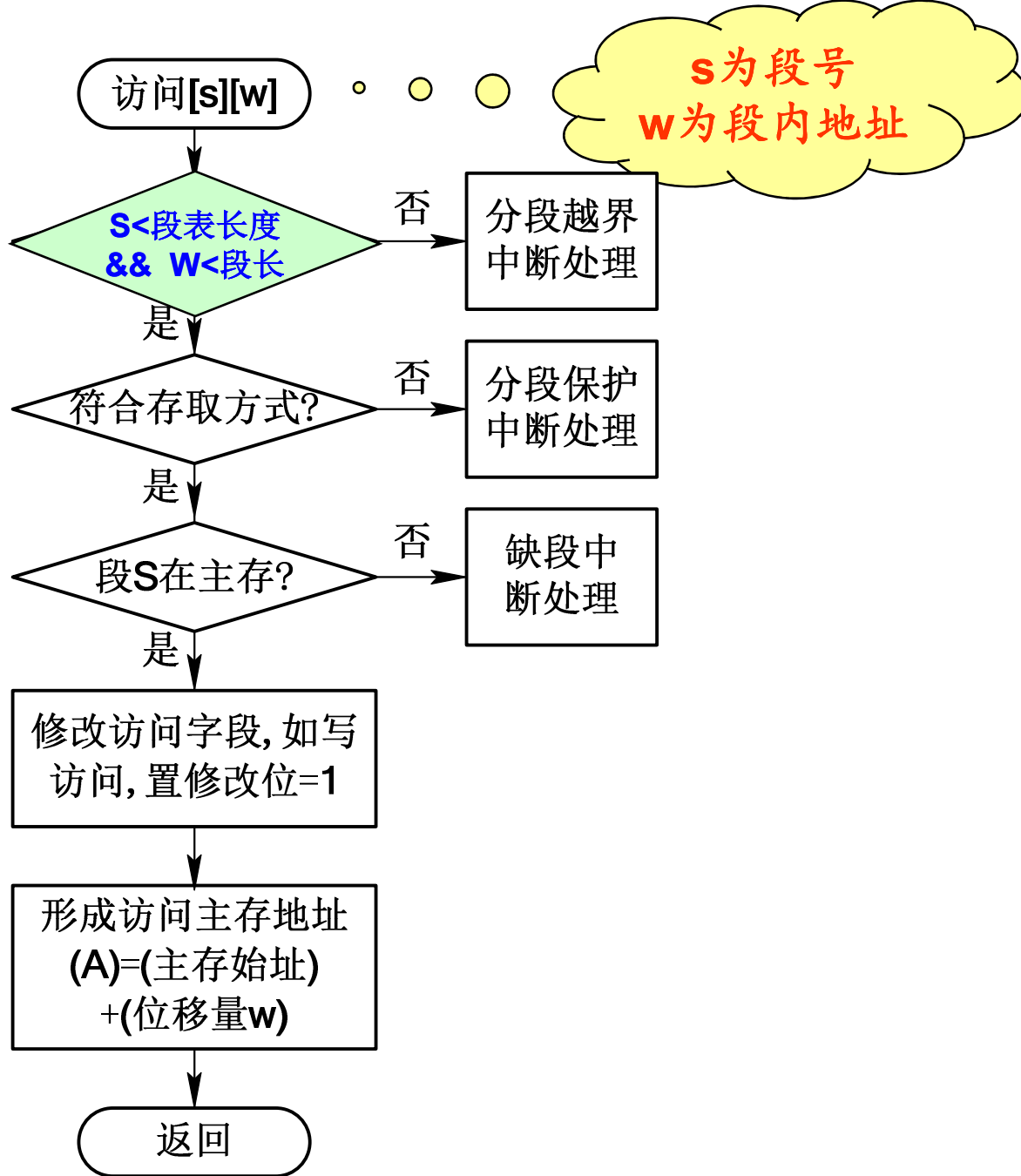


图5-12 请求分段系统中的中断处理过程





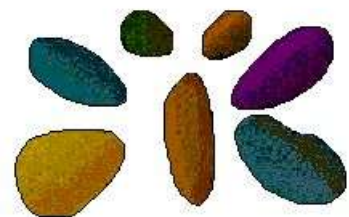
地址变换机构





5.5 请求分段存储管理方式

- ❖ 5.5.1 请求分段存储管理方式
- ❖ 5.5.2 分段的共享与保护





5.5.2 分段的共享与保护

❖ 1、共享段表（用来管理系统中各共享段）

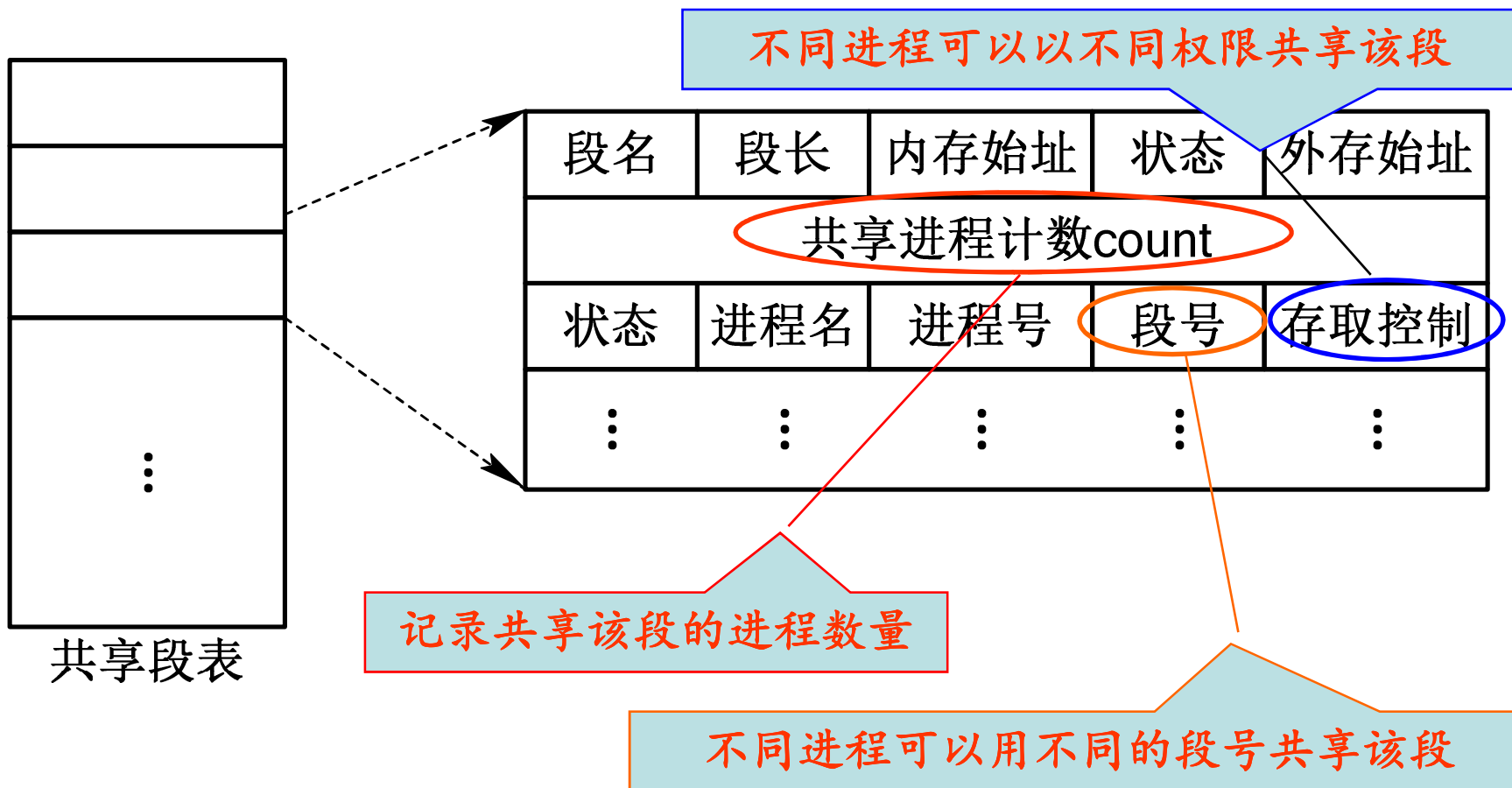


图5-14 共享段表项



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

- 共享段第一次被某进程访问：a> 分配内存，在共享段表中增加表项，置该表项的count=1；b> 修改该访问进程的段表。
- 共享段非第一次被某进程访问：a> 无需分配内存，只需修改共享段表相应表项，置表项的count++；b> 修改该访问进程的段表。

* 2> 共享段的回收

- 若count==0，释放该段所占内存，撤消共享段表相应表项；若count>0，count--，修改共享段表相应表项。



5.5.2 分段的共享与保护

❖ 3、分段保护（分段信息安全保护）

* 1> 越界检查

- 段号越界检查：段号 $<$ 段表长度
- 段内偏移越界检查：段内地址 $<$ 段长

* 2> 存取控制检查

- 只执行：进程只能调用该段，不能读/写
- 只读：进程只能读取该段中的程序和数据
- 读/写：进程可读/写该段中的程序和数据

* 3> 环保护机构

- 内环的权限高于外环
- 内环可访问相同环和外环数据
- 外环可请求相同环和内环服务



本章小结

❖ 虚拟存储器理论*

- * 局部性原理 **

- * 虚拟存储器定义 **

❖ 请求分页存储管理方式*

- * 请求页表机制、缺页中断机构、地址变换机构 **

❖ 六种页面置换算法**

- * OPT、FIFO、LRU、CLOCK **

❖ 抖动与工作集*

- * 抖动的原因 **、预防方法*

❖ 请求分段存储管理*

- * 请求段表机制、缺段中断机构、地址变换机构 *

- * 共享段表 **



**掌握
*理解



本章小结

❖ 重要概念

- * 虚拟存储器、请求分页存储管理、请求分段存储管理、缺页率、缺页中断、缺段中断、共享段表、可重入码、工作集、抖动等



本章作业

❖ 要求:

- * 一定要做在作业本上

❖ 作业内容:

- * 操作系统第4、5章网络在线测试
- * 课后习题P177 第13题（另增加一问：如果采用OPT、LRU算法置换，其缺页次数和缺页率又为多少？）
- * 要考研的同学：本章示例1、Clock算法示例自己做一遍，不强求。



本章课程结束！ 谢谢大家！