



经典教材《计算机操作系统》**最新版**

第6章 虚拟存储器

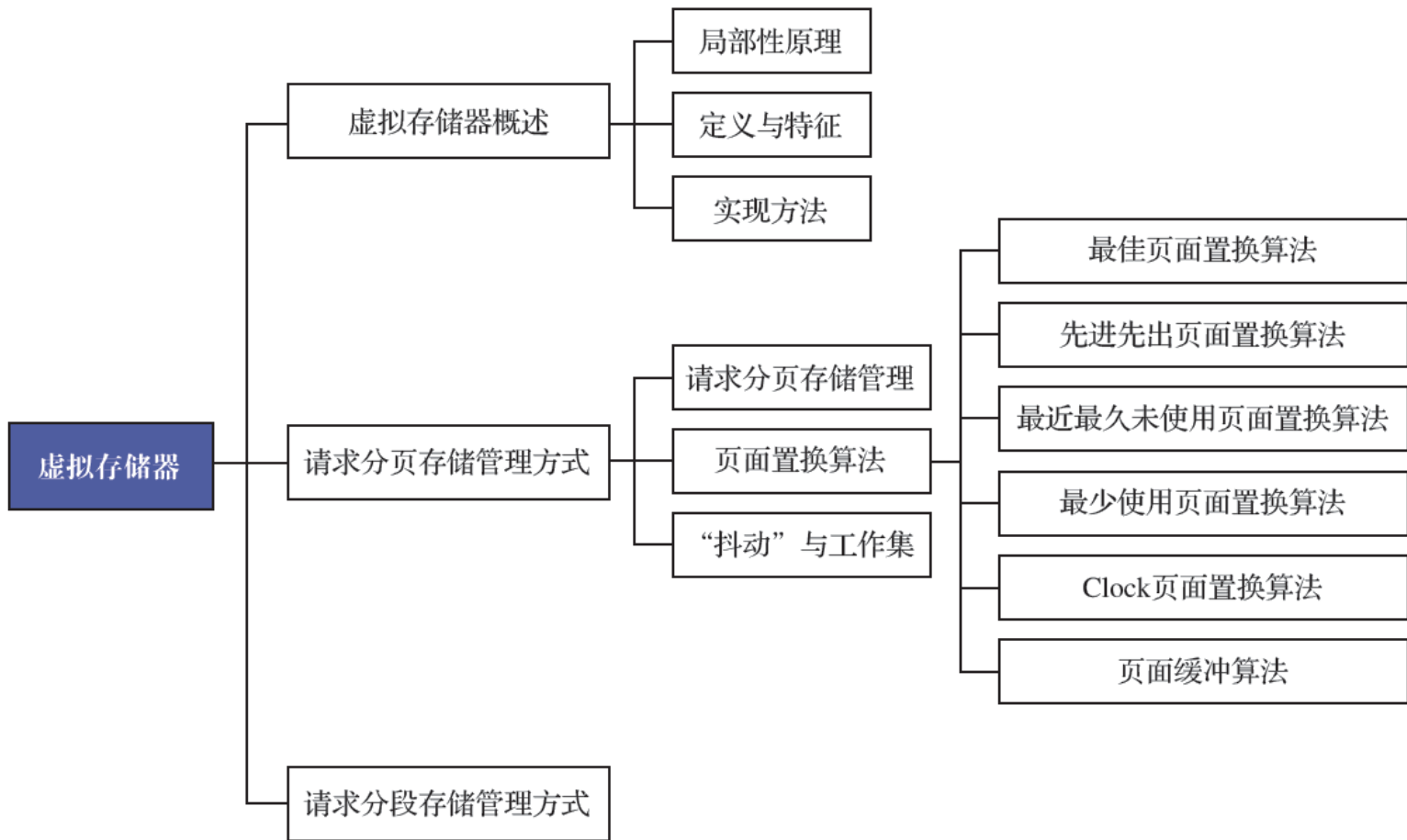
主讲教师：陆丽萍





第6章知识导图

第1章	操作系统引论
第2章	进程的描述与控制
第3章	处理机调度与死锁
第4章	进程同步
第5章	存储器管理
第6章	虚拟存储器
第7章	输入/输出系统
第8章	文件管理



若使用单级页表结构，一个页表有多大？

32位地址空间，页4K，页表项4B，

页表大小： $2^{32} / 4K * 4 = 4MB$

64位地址空间，页4K，页表项8B，

页表大小： $2^{64} / 4K * 8 = 33,554,432 \text{ GB}$



内容导航:



6.1 虚拟存储器概述



6.2 请求分页存储管理方式



6.3 页面置换算法



6.4 抖动与工作集



6.5 请求分段存储管理方式



6.6 虚拟存储器实现实例

第6章 虚拟存储器

前面所介绍的各种存储器管理方式，有一个共同特点：作业全部装入内存后方能运行。



问题：

- 大作业装不下
- 少量作业得以运行



解决办法：

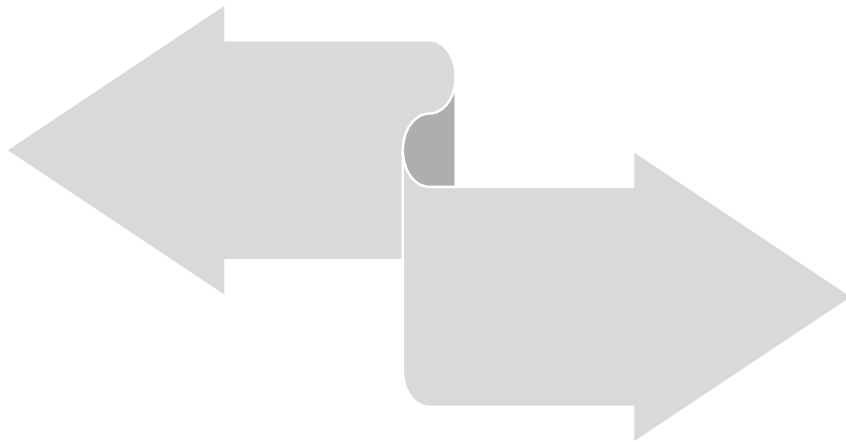
- 扩充内存
- 逻辑上扩充内存容量（虚拟存储器）



常规存储器管理方式的特征



一次性：作业被一次性全部装入内存



驻留性：作业一直驻留在内存

一次性和驻留性使许多在程序运行中不用或暂不用的程序（数据）占据了大量的内存空间，使得一些需要运行的作业无法装入运行。



1968年, P. denning 提出:

- 程序执行时, 除了少部分的转移和过程调用外, 在大多数情况下仍然是顺序执行的。
- 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, 过程调用的深度一般小于5。程序将会在一段时间内都局限在这些过程的范围内运行。
- 程序中存在许多循环结构, 多次执行。
- 对数据结构的处理局限于很小的范围。



表现:

- 时间局部性: 一条指令被执行了, 则在不久的将来它可能再被执行。大量的循环操作。
- 空间局部性: 若某一存储单元被使用, 则在一定时间内, 与该存储单元相邻的单元可能被使用。程序的顺序执行。



基于局部性原理，应用程序在运行之前，没有必要全部装入内存，仅须将那些当前要运行的部分页面或段先装入内存便可运行，其余部分暂留在盘上。



虚拟存储器：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，而成本接近于外存。



多次性：作业中的程序和数据允许被分成多次调入内存允许



对换性：作业运行时无须常驻内存



虚拟性：从逻辑上扩充了内存容量，使用户看到的内存容量远大于实际内存容量



请求分页系统

- 硬件支持：页表、缺页中断、地址变换机构。
- 软件支持：请求调页软件、页面置换软件。



请求分段系统

- 硬件支持：段表、缺段中断、地址变换机构。
- 软件支持：请求调段软件、段置换软件。









段页式虚拟存储器

- 增加请求调页和页面置换。
- Intel 80386 及以后。



内容导航:

-  6.1 虚拟存储器概述
-  **6.2 请求分页存储管理方式**
-  6.3 页面置换算法
-  6.4 抖动与工作集
-  6.5 请求分段存储管理方式
-  6.6 虚拟存储器实现实例

第6章 虚拟存储器

请求分页中的硬件支持



请求页表机制

- 状态位P：指示该页是否在内存
- 访问字段A：记录该页在一段时间内被访问的次数
- 修改位M：也称脏位，标志该页是否被修改过，确认是否需要回写到外存
- 外存地址：指示该页在外存中的地址（物理块号）

页号	物理块号	状态位P	访问字段A	修改位M	外存地址
----	------	------	-------	------	------



缺页中断机构

- 在指令执行期间产生和处理中断信号
- 一条指令在执行期间，可能产生多次缺页中断



地址变换机构

- 与分页内存管理方式类似



缺页中断机构

- 在指令执行期间产生和处理中断信号
- 一条指令在执行期间，可能产生多次缺页中断

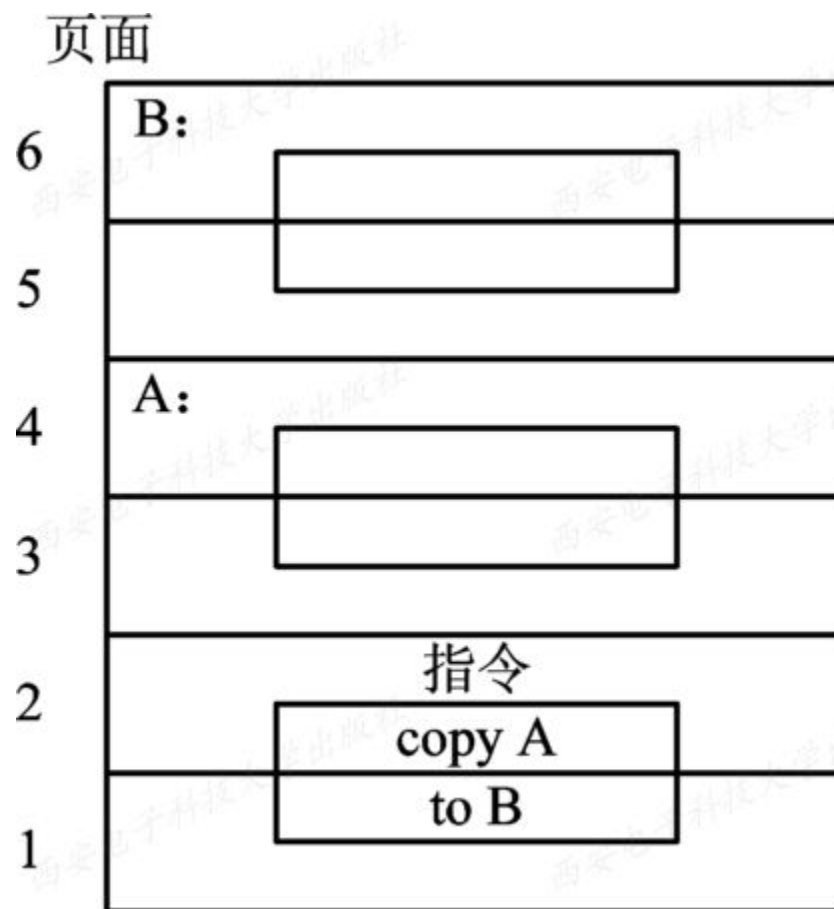
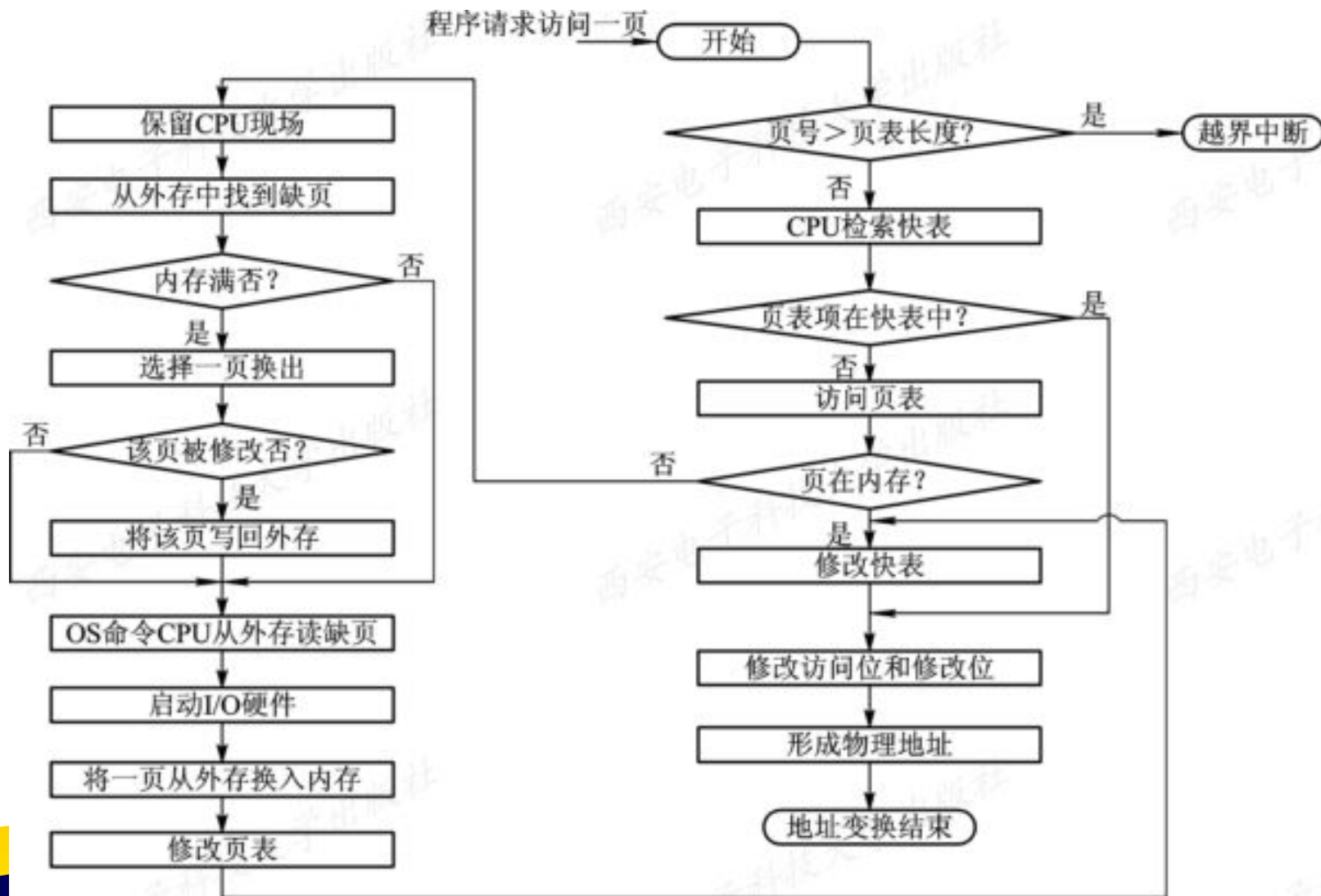


图6-2 涉及6次缺页中断的指令



地址变换机构

➤ 与分页内存管理方式类似





最小物理块数的确定:

- 保证进程正常运行所需的最小物理块数。取决于指令的格式、功能和寻址方式。



物理块的分配策略:

- 固定分配 局部置换;
- 可变分配 全局置换;
- 可变分配 局部置换。



物理块分配算法:

- 平均分配算法;
- 按比例分配算法;
- 考虑优先权的分配算法;

- 计算各进程页面总和 S

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

- 计算每个进程所能分到的物理块数 b_i

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



何时调入页面

- 预调页策略：预先调入一些页面到内存
- 请求调页策略：发现需要访问的页面不在内存时，调入内存



从何处调入页面（外存分为对换区和文件区）

- 如系统拥有足够的对换区空间，全部从对换区调入所需页面
- 如系统缺少足够的对换区空间，凡是不会被修改的文件，都直接从文件区调入；当换出这些页面时，由于未被修改而不必再将它们重写磁盘，以后再调入时，仍从文件区直接调入
- UNIX方式：未运行过的页面，从文件区调入；曾经运行过但又被换出的页面，从对换区调入



如何调入页面？



01

查找所需页在磁盘上的位置。

02

查找一内存空闲块：

- 如果有空闲块，就直接使用它；
- 如果没有空闲块，使用页面置换算法选择一个“牺牲”内存块；
- 将“牺牲”块的内容写到磁盘上，更新页表和物理块表。

03

将所需页读入（新）空闲块，更新页表。

04

重启用户进程。

缺页率



访问页面成功(在内存)的次数为S



访问页面失败(不在内存)的次数为F



总访问次数为 $A=S+F$



缺页率为 $f= F/A$



影响因素：页面大小、分配内存块的数目、页面置换算法、程序固有属性



缺页中断处理的时间（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 ，）

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



缺页中断处理时间的例子



存取内存的时间= 200 nanoseconds (ns)



平均缺页处理时间 = 8 milliseconds (ms)



$$t = (1 - p) \times 200\text{ns} + p \times 8\text{ms}$$

$$= (1 - p) \times 200\text{ns} + p \times 8,000,000\text{ns}$$

$$= 200\text{ns} + p \times 7,999,800\text{ns}$$









如果每1,000次访问中有一个缺页中断，那么：

$$t = 8.2 \text{ ms}$$

这是导致计算机速度放慢40倍的影响因子！



内容导航:

-  6.1 虚拟存储器概述
-  6.2 请求分页存储管理方式
-  6.3 **页面置换算法**
-  6.4 抖动与工作集
-  6.5 请求分段存储管理方式
-  6.6 虚拟存储器实现实例

第6章 虚拟存储器



页面置换：找到内存中没有使用的一些页，换出

- 算法：替换策略
- 性能：找出一个导致最小缺页数的算法



同一个页可能会被装入内存多次（可能造成“抖动”）



页面置换完善了逻辑内存和物理内存的划分——在一个较小的物理内存基础之上可以提供一个大的虚拟内存



页面置换算法

- 最佳置换算法 (OPT)
- 最少使用算法 (LFU)
- 先进先出置换算法 (FIFO)
- Clock置换算法
- 最近最久未使用置换算法 (LRU)
- 页面缓冲算法



需要一个最小的缺页率



通过运行一个内存访问的特殊序列（访问序列），计算这个序列的缺页次数



被置换的页将是之后最长时间不被使用的页



4 帧的例子

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

1
2
3
4

4

6 次缺页

5



怎样知道被置换的页是之后最长时间不被使用的页?



无法实现的算法, 可用来评价其他算法

reference string

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

page frames



先进先出置换算法



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



引用串：1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

➤ 3 个页：9次缺页

➤ 4 个页框：10次缺页

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

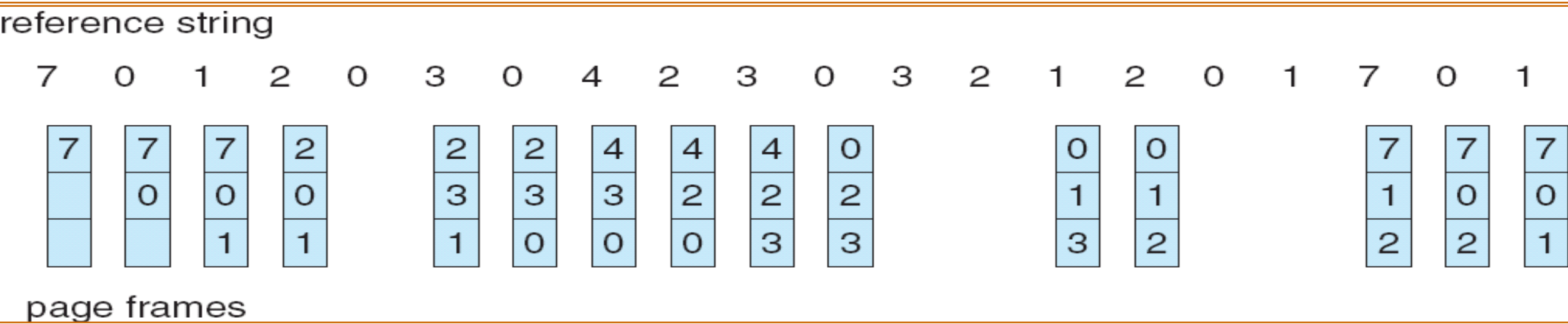
9 次缺页

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

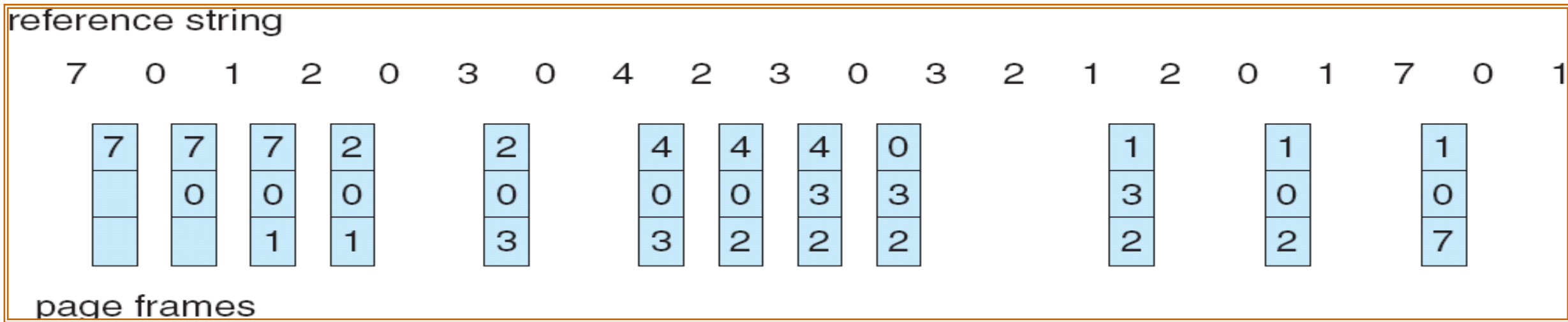
10 次缺页



先进先出置换算法例子



选择最近最久未使用的页面予以淘汰





寄存器：为内存中的每个页面设置一个移位寄存器

- 被访问的页面对应寄存器的 R_{n-1} 位置为1，定时右移
- 具有最小数值的寄存器所对应的页面为淘汰页

实 页 \ R	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



栈：保存当前使用的各个页面的页面号

- 被访问的页，移到栈顶
- 栈底是最近最久未使用页面的页面号

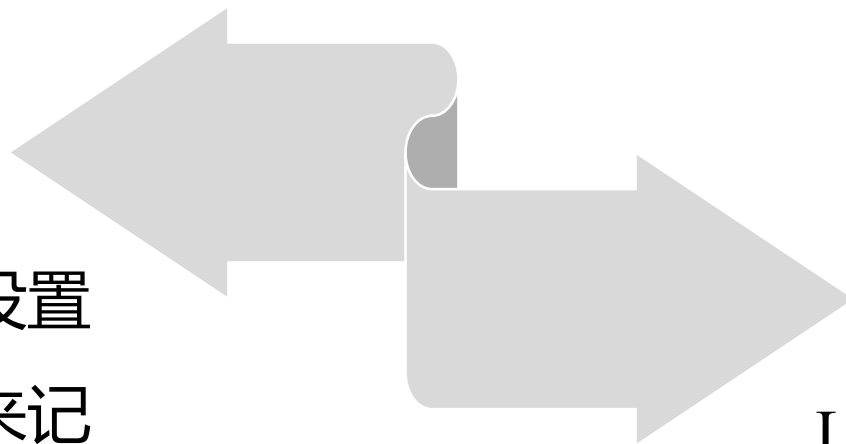
假定现有一进程，它分有五个物理块，所访问的页面的页面号序列为：

4, 7, 0, 7, 1, 0, 1, 2, 1, 2, 6

4	7	0	7	1	0	1	2	1	2	6
							2	1	2	6
				1	0	1	1	2	1	2
		0	7	7	1	0	0	0	0	1
	7	7	0	0	7	7	7	7	7	0
4	4	4	4	4	4	4	4	4	4	7



为内存中的每个页面设置
一个移位寄存器，用来记
录该页面的被访问频率

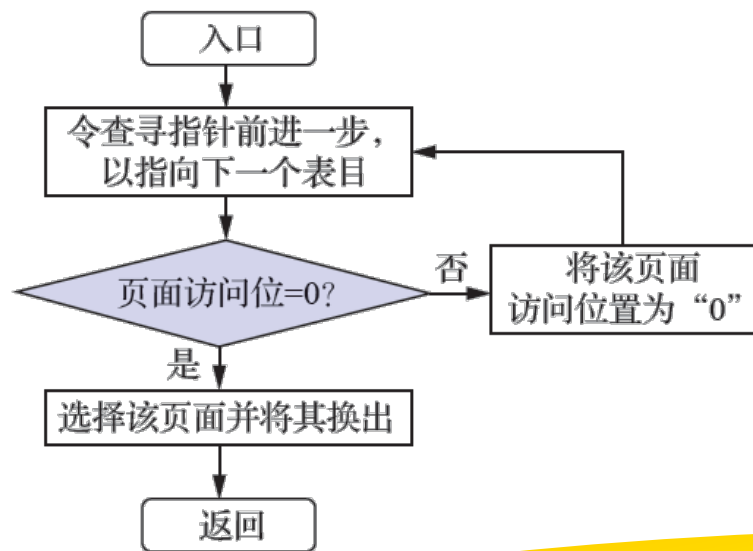


LFU 选择在最近时期使用最
少的页面作为淘汰页

OS LRU的近似算法，又称最近未用(NRU)或二次机会页面置换算法

OS 简单的Clock算法

- 每个页都与一个访问位相关联，初始值位0
- 当页被访问时置访问位为1
- 置换时选择访问位为0的页；若为1，重新置为0



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



除须考虑页面的使用情况外，还须增加置换代价



淘汰时，同时检查访问位A与修改位M

- **第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。
- **第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。
- **第3类** ($A=1, M=0$)：表示该页最近已被访问，但未被修改，该页有可能再被访问。
- **第4类** ($A=1, M=1$)：表示该页最近已被访问且被修改，该页有可能再被访问。



置换时，循环依次查找第1类、第2类页面，找到为止



影响效率的因素：

- 页面置换算法、写回磁盘的频率、读入内存的频率



目的：

- 显著降低页面换进、换出的频率，减少了开销
- 可采用较简单的置换策略，如不需要硬件支持



具体做法：

- 设置两个链表：
 - ①空闲页面链表：保存空闲物理块
 - ②修改页面链表：保存已修改且需要被换出的页面等被换出的页面数目达到一定值时，再一起换出外存，



访问页在内存，且其对应页表项在快表中

➤ $EAT = \lambda + t$ λ 为查找快表所需要的时间， t 为访问一次内存所需要的时间



访问页在内存，且其对应页表项不在快表中

➤ $EAT = \lambda + t + \lambda + t = 2(\lambda + t)$



访问页不在内存中

➤ 需进行缺页中断处理，有效时间可分为查找快表的时间、查找页表的时间、处理缺页的时间、更新快表的时间和访问实际物理地址的时间

➤ 假设缺页中断处理时间为 ε

$$EAT = \lambda + t + \varepsilon + \lambda + t = \varepsilon + 2(\lambda + t)$$

➤ f 为缺页率， φ 为缺页中断处理时间（只考虑缺页率，不考虑命中率）

$$EAT = t + f \times (\varphi + t) + (1 - f) \times t$$

已知某程序访问以下页面：0, 1, 4, 2, 0, 2, 6, 5, 1, 2, 3, 2, 1, 2, 6, 2, 1, 3, 6, 2。如果程序有3个页框可用且使用下列替换算法，求出现缺页的次数。

(1) FIFO替换算法 (2) LRU替换算法

(1) FIFO 算法总是淘汰最先进入内存页面，即选择在内存中驻留时间最长的页予以淘汰。算法如图所示：

	0	1	4	2	0	2	6	5	1	2	3	2	1	2	6	2	1	3	6	2
0	0	0	0	2	2		2	5	5	5	3				3		3			2
1		1	1	1	0		0	0	1	1	1				6		6			6
4			4	4	4		6	6	6	2	2				2		1			1

缺页率 $13/20=65\%$

已知某程序访问以下页面：0, 1, 4, 2, 0, 2, 6, 5, 1, 2, 3, 2, 1, 2, 6, 2, 1, 3, 6, 2。如果程序有3个页框可用且使用下列替换算法，求出现缺页的次数。

(1) FIFO替换算法 (2) LRU替换算法

(2) LRU 算法是最近最久未使用的页面予以淘汰。算法如图所示：

0	1	4	2	0	2	6	5	1	2	3	2	1	2	6	2	1	3	6	2
0	0	0	2	2		2	2	1	5	3				6			3	3	3
	1	1	1	0		0	5	5	1	1				1			1	1	2
		4	4	4		6	6	6	2	2				2			2	6	6

缺页率 $14/20=70\%$







某系统使用请求分页存储管理，如果页在内存中，满足一个内存请求需要200ns。如果页不在内存，如有空闲的页框或者没有修改的换出的页，则请求需要7ms。如果替换出的页已经被修改，则需要15ms，如果缺页率是5%，并且有60%为修改后要换出的页，问有效访问时间是多长？假设系统只运行一个进程且页交换时CPU空闲。

200ns内得到满足的访问占用全部访问的95%。5%的访问造成缺页，其中40%的需要7ms。因此， $5\% \times 40\% = 2\%$ 的访问需要7ms。类似地， $5\% \times 60\% = 3\%$ 的访问需要15ms。把所有的时间转换为us，结果如下：


$$\text{有效访问时间} = 0.95 \times 0.2 + 0.02 \times 7000 + 0.03 \times 15000。$$

$$\text{有效访问时间} = 590.19\mu\text{s}。$$




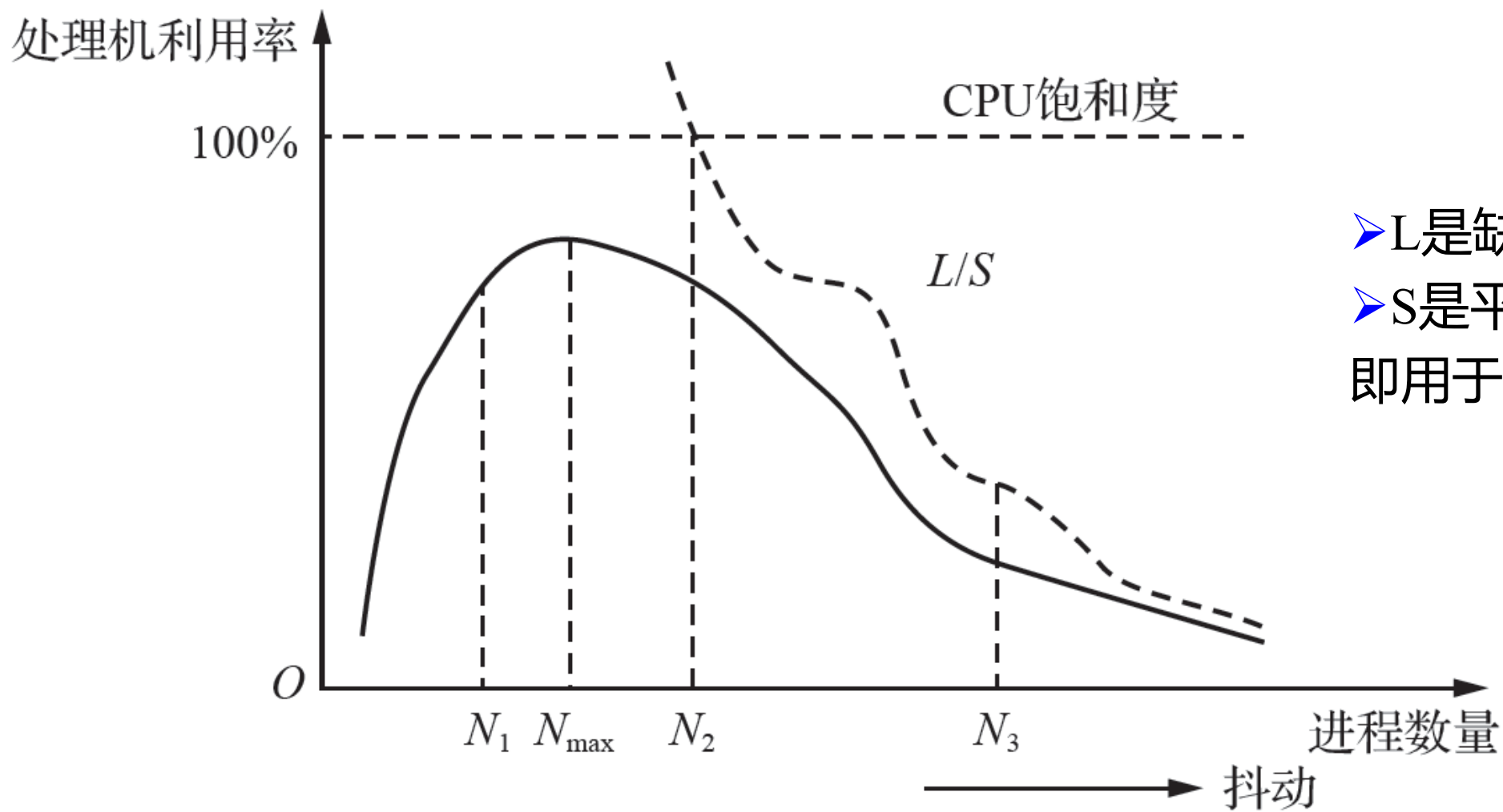
-  6.1 虚拟存储器概述
-  6.2 请求分页存储管理方式
-  6.3 页面置换算法
-  **6.4 抖动与工作集**
-  6.5 请求分段存储管理方式
-  6.6 虚拟存储器实现实例

第6章 虚拟存储器

 如果一个进程没有足够的页，那么缺页率将很高，这将导致：

- CPU利用率低下.
- 操作系统认为需要增加多道程序设计的道数
- 系统中将加入一个新的进程

 **抖动**(Thrashing)：一个进程的页面经常换入换出



- 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的时间



产生“抖动”的原因



根本原因：

- 同时在系统中运行的进程太多；
- 因此分配给每一个进程的物理块太少，不能满足进程运行的基本要求，致使进程在运行时，频繁缺页，必须请求系统将所缺页面调入内存。



抖动的发生与系统为进程分配物理块的多少有关。



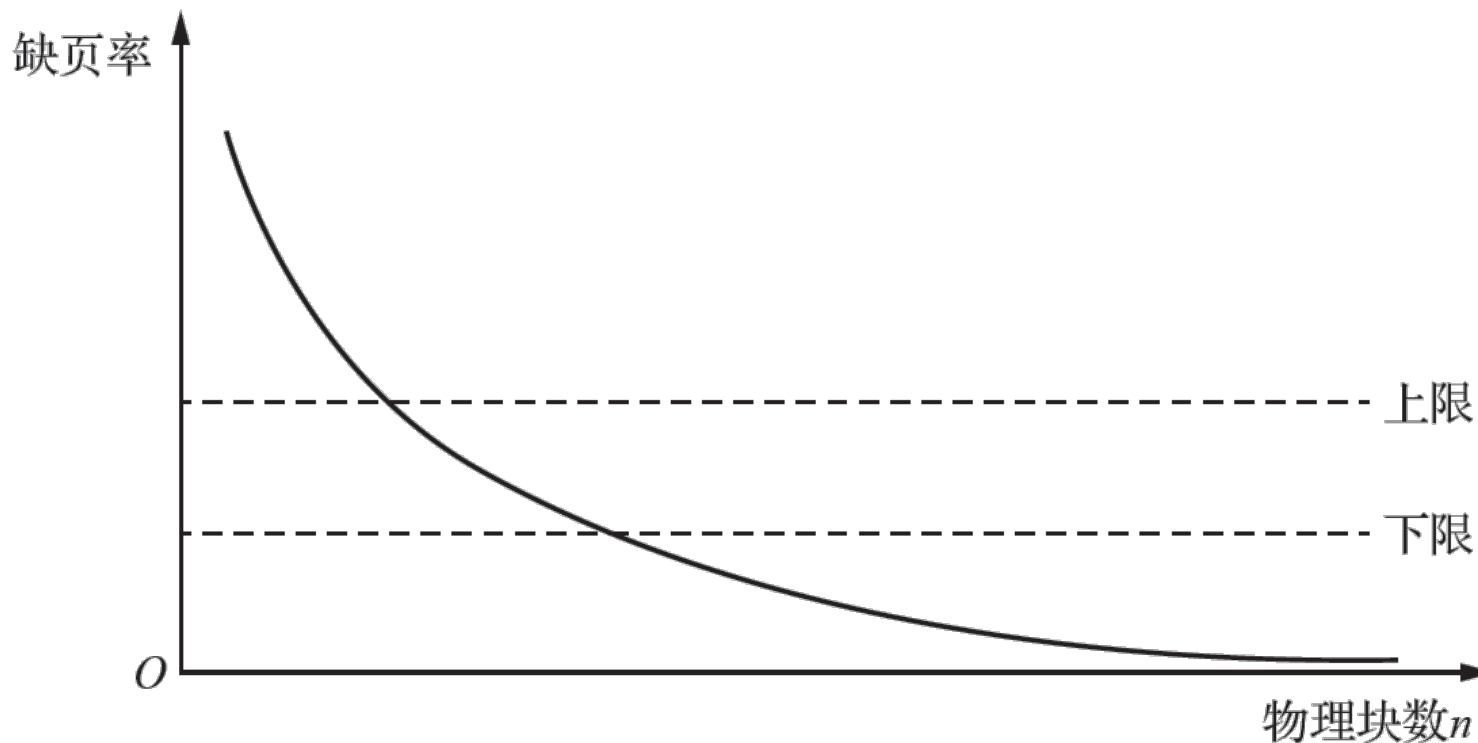
缺页率与物理块数之间的关系



进程发生缺页的时间间隔与所获得的物理块数有关。



根据程序运行的局部性原理，如果能够**预知某段时间内程序要访问的页面**，并将它们预先调入内存，将会大大降低缺页率。





所谓**工作集**，指在某段时间间隔 Δ 里进程实际要访问页面的集合。



把某进程在时间 t 的工作集记为 $w(t, \Delta)$,其中的变量 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；工作集与窗口尺寸 Δ 有关，是 Δ 的非降函数，即：

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



窗口大小

访问页面序列

	3	4	5
24	24	24	24
15	15 24	15 24	15 24
18	18 15 24	18 15 24	18 15 24
23	23 18 15	23 18 15 24	23 18 15 24
24	24 23 18	—	—
17	17 24 23	17 24 23 18	17 24 23 18 15
18	18 17 24	—	—
24	—	—	—
18	—	—	—
17	—	—	—
17	—	—	—
15	15 17 18	15 17 18 24	—
24	24 15 17	—	—
17	—	—	—
24	—	—	—
18	18 24 17	—	—

01

采取**局部置换策略**：只能在分配给自己的内存空间内进行置换；

02

把工作集算法融入到处理机调度中；

03

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的时间
- $L>S$ ，说明很少发生缺页

- $L<S$ ，说明频繁缺页
- $L=S$ ，磁盘和处理机都可达到最大利用率

04

选择暂停进程。

考虑一个请求调页系统，它采用全局置换策略和平均分配内存块的算法（即若有 m 个内存块和 n 个进程，则每个进程分得 m/n 个内存块）。如果在该系统测得如下的CPU和对换盘利用率，请问能否用增加多道程序的度数来增加CPU的利用率？为什么？

- (1) CPU的利用率为13%， 盘利用率为97%。
- (2) CPU的利用率为87%， 盘利用率为3%。
- (3) CPU的利用率为13%， 盘利用率为3%。

多道程序的度就是操作系统将多少个进程放入了内存,或者说有多少个进程被允许对CPU进行抢占运行。

如果在该系统测得如下的CPU和对换盘利用率，请问能否用增加多道程序的度数来增加CPU的利用率？为什么？

(1) CPU的利用率为13%，盘利用率为97%。

(2) CPU的利用率为87%，盘利用率为3%。

(3) CPU的利用率为13%，盘利用率为3%。







(1) 这种情况表示系统在频繁进行页面置换，CPU大部分时间被花在页面置换上。此时，增加多道程序会进一步增加缺页率，使系统性能进一步恶化。所以，不能用增加多道程序的度数来增加CPU的利用率，反而应减少内存中的作业道数。

(2) 这种情况下，CPU的利用率比较高，但盘的利用率比较低，这表示运行进程的缺页率很低，可以适当增加多道程序的度数来提高CPU的利用率。

(3) 在这种情况下，CPU的利用率较低，盘的利用率也非常低，表示内存中可运行的程序数不足，此时，应该增加多道程序的度数来提高CPU的利用率。



内容导航:

-  6.1 虚拟存储器概述
-  6.2 请求分页存储管理方式
-  6.3 页面置换算法
-  6.4 抖动与工作集
-  **6.5 请求分段存储管理方式**
-  6.6 虚拟存储器实现实例

第6章 虚拟存储器

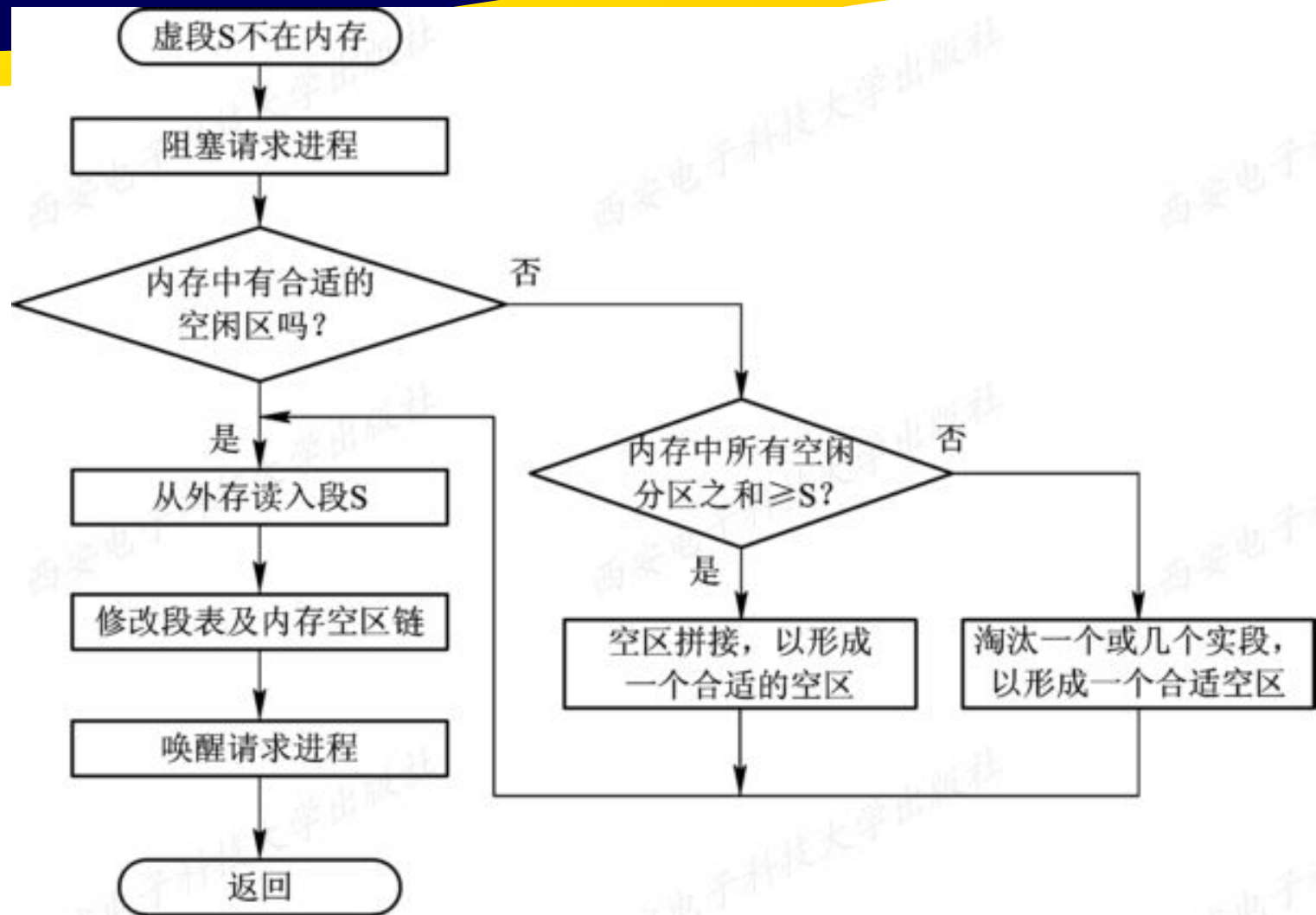
请求段表机制

- 存取方式：表示段存取属性为只执行、只读或允许读/写
- 访问字段A：记录该段在一段时间内被访问的次数
- 修改位M：标志该段调入内存后是否被修改过
- 存在位P：指示该段是否在内存
- 增补位：表示该段在运行过程中是否做过动态增长
- 外存始址：指示该段在外存中的起始地址（盘块号）

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

缺段中断机构

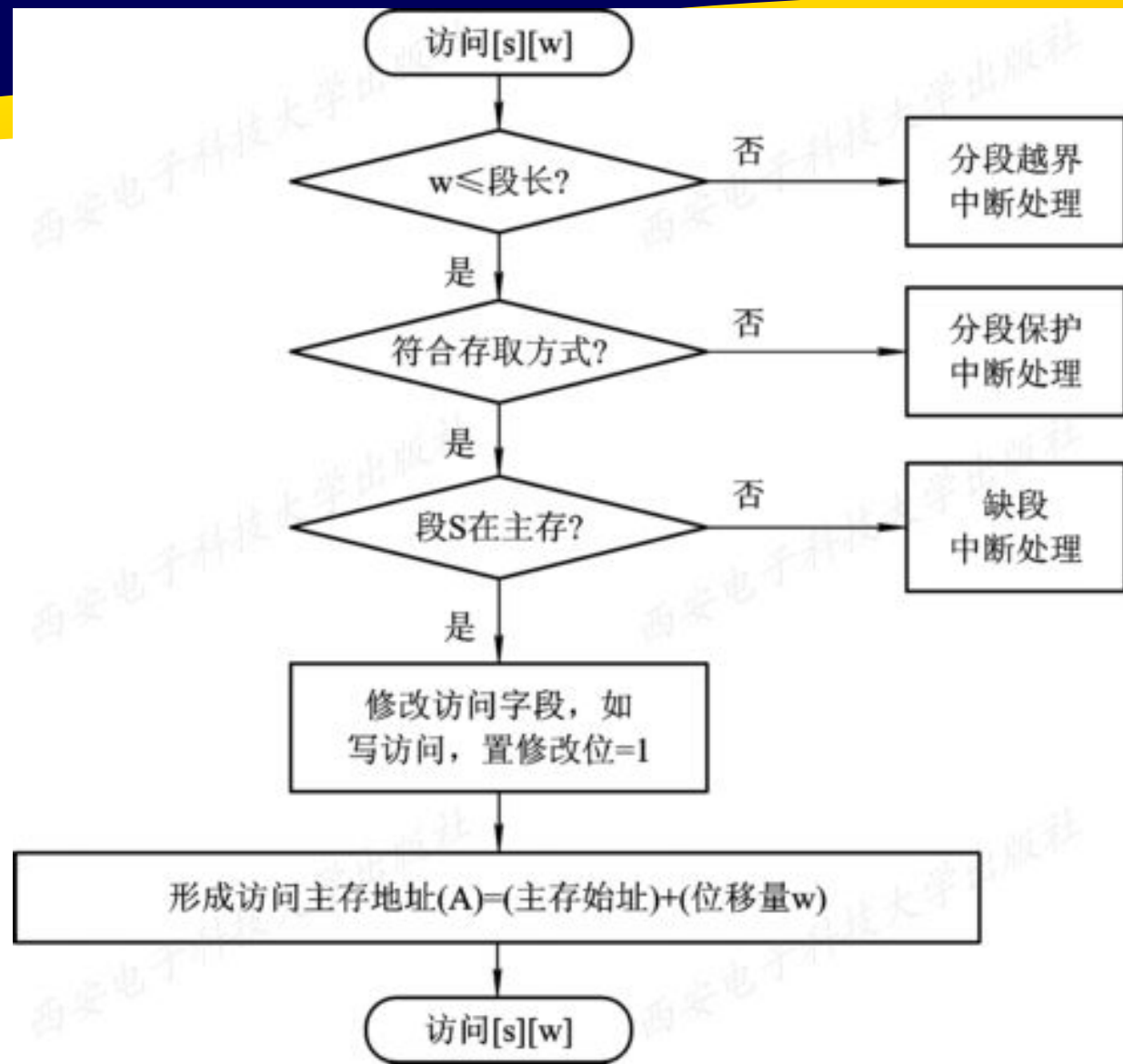
- 在指令执行期间产生和处理中断信号
- 一条指令在执行期间，可能产生多次缺段中断
- 由于段不是定长的，对缺段中断的处理要比对缺页中断的处理复杂



请求分段系统中的中断处理过程

地址变换机构

- 若段不在内存中，则必须先将所缺的段调入内存，并修改段表，然后利用段表进行地址变换。

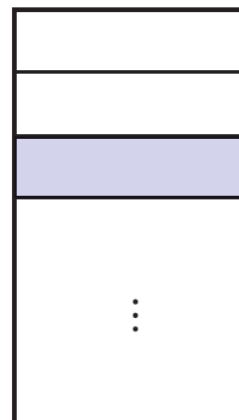


请求分段系统的地址变换过程



共享段表：保存所有的共享段

- 共享进程计数count
- 存取控制字段
- 段号



共享段表

段名	段长	内存始址	状态位	外存始址
共享进程记数count				
状态	进程名	进程号	段号	存取控制
⋮	⋮	⋮	⋮	⋮



共享段的分配

- 对首次请求使用共享段的用户，分配内存，调入共享段，修改该进程段表相应项，再为共享段表增加一项， $count=1$
- 对其他使用共享段的用户，修改该进程段表相应项，再为共享段表增加一项， $count=count+1$



共享段的回收

- 撤销在该进程段表中共享段所对应的表项，并执行 $count=count-1$ 操作
- 若为0，回收该共享段的内存，并取消共享段表中对应的表项
- 若不为0，只取消调用者进程在共享段表中的有关记录



越界检查：

- 由地址变换机构来完成；
- 比较段号与段表长度；段内地址与段表长度。



存取控制检查：以段为基本单位进行。

- 通过“存取控制”字段决定段的访问方式；
- 基于硬件实现。



环保护机构：

- 低编号的环具有高优先权；
- 一个程序可以访问驻留在相同环或较低特权环（外环）中的数据；
- 一个程序可以调用驻留在相同环或较高特权环（内环）中的服务。

设作业的虚拟地址为24位，其中高8位为段号，低16位为段内相对地址。试问：

(1) 一个作业最多可以有多少段？

(3) 每段的最大长度为多少字节？

(3) 某段式存储管理采用如下段表,试计算[0,430]、[1,50]、[2,30]、[3,70]的主存地址。其中方括号内的前一元素为段号，后一元素为段内地址。当无法进行地址变换时，应说明产生何种中断。

段号	段长	主存起始地址	是否在主存
0	600	2100	是
1	40	2800	是
2	100		否
3	80	4000	是

(1) 一个作业最多可以有 $2^8=256$ 个段。

(2) 每段的最大长度为 $2^{16}=64\text{KB}=65536$ 字节。

(3) 逻辑地址 (0, 430) 的主存地址为 $2100+430=2530$;

逻辑地址 (1, 50) 无法进行地址变换, 因为产生了越界中断;







逻辑地址 (2, 30) 无法进行地址变换, 因为产生了缺段中断;

逻辑地址 (3, 70) 的主存地址为 $4000+70=4070$ 。

段号	段长	主存起始地址	是否在主存
0	600	2100	是
1	40	2800	是
2	100		否
3	80	4000	是



内容导航:

-  6.1 虚拟存储器概述
-  6.2 请求分页存储管理方式
-  6.3 页面置换算法
-  6.4 抖动与工作集
-  6.5 请求分段存储管理方式
-  6.6 **虚拟存储器实现实例**

第6章 虚拟存储器

- OS 采用请求页面调度以及簇来实现虚拟存储器
- OS 使用簇在处理缺页中断时，不但会调入不在内存中的页（出错页），还会调入出错页周围的页
- OS 创建进程时，系统会为其分配工作集的最小值和最大值
 - 最小值：进程在内存中时所保证页面数的最小值
 - 若内存足够，可分配更多的页面，直到达到最大值
 - 通过维护空闲块链表（与一个阈值关联）来实现
 - 采用局部置换方式
- OS 置换算法与处理器类型有关
 - 如80x86系统，采用改进型Clock算法



实例2: Linux系统 (以32位为例)



虚拟存储器是大小为4GB的线性虚拟空间。



4GB的地址空间分为两个部分:

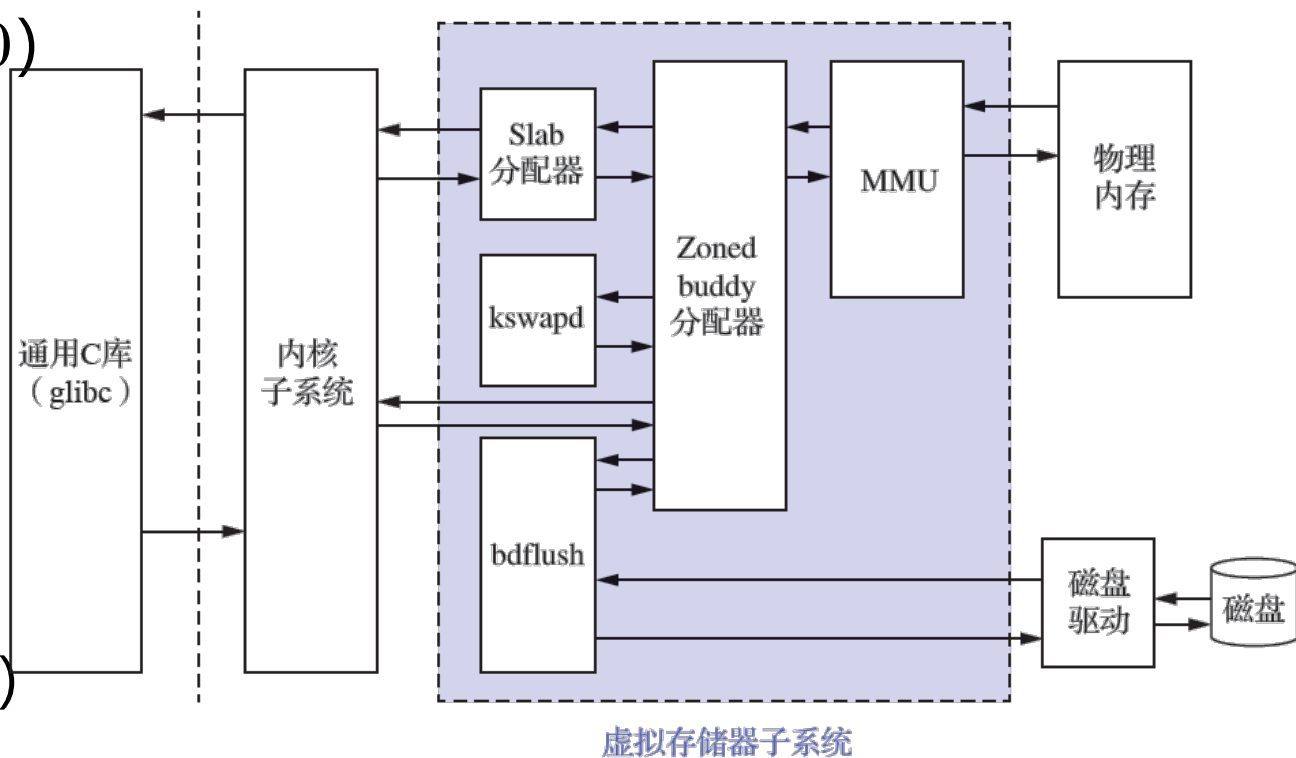


□ 用户空间占据0~3GB (0xC0000000)

- 由用户进程使用 (MMU)
- 使用请求页式存储管理

□ 内核空间占据3GB~4GB

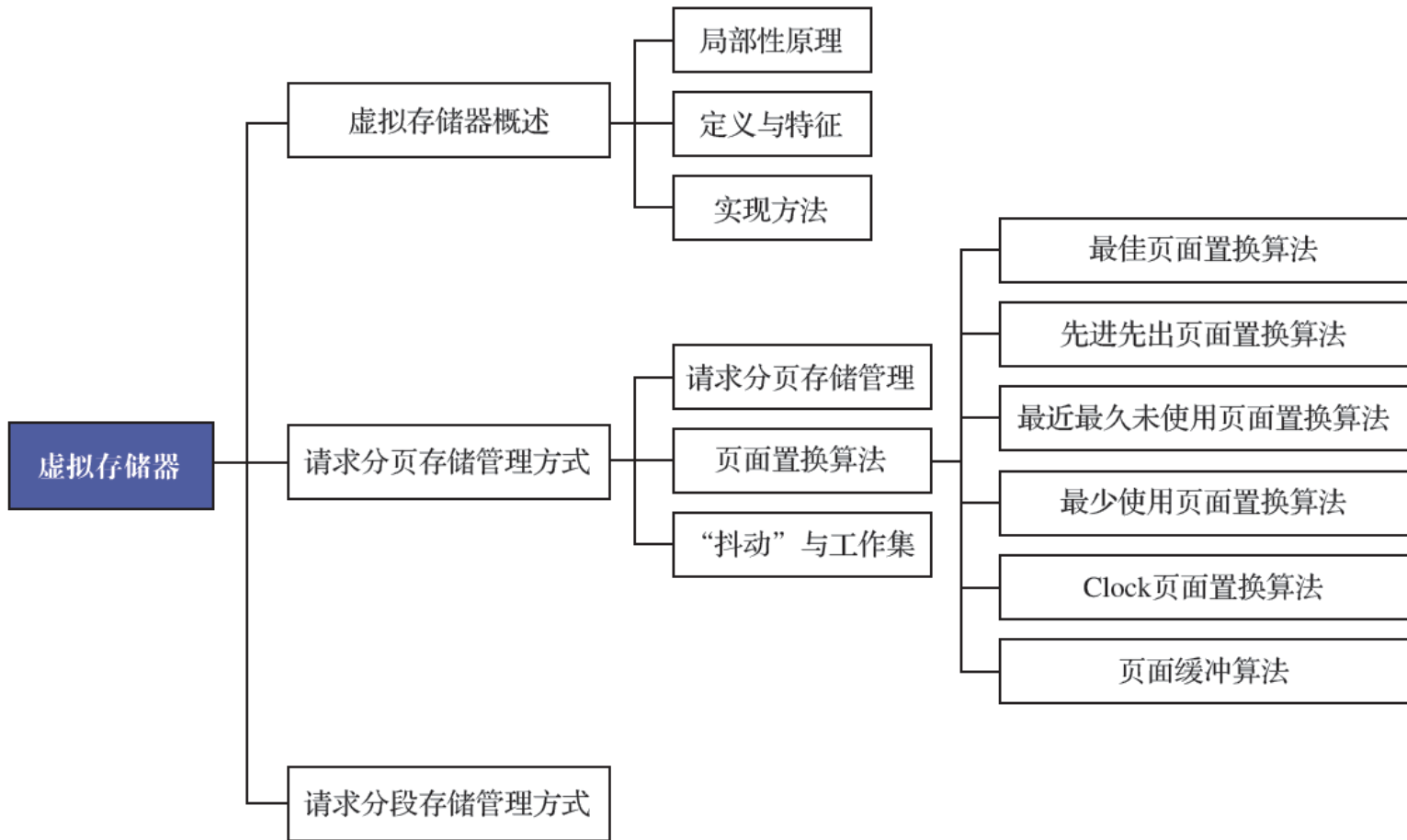
- 由内核负责
- 使用buddy和slab内存管理 (zoned buddy 分配器和slab分配器)





学而时习之（第6章总结）

第1章	操作系统引论
第2章	进程的描述与控制
第3章	处理机调度与死锁
第4章	进程同步
第5章	存储器管理
第6章	虚拟存储器
第7章	输入/输出系统
第8章	文件管理





第一次作业

简答题

1	2	3	4	5	6	7	8
9	10	11	12				

计算题

13	14	15	16	17	
----	----	----	----	----	--

综合应用题

18	19	20	21	
----	----	----	----	--

标黄色为本次作业



第二次作业

简答题

1	2	3	4	5	6	7	8
9	10	11	12				

计算题

13	14	15	16	17
----	----	----	----	----

综合应用题

18	19	20	21
----	----	----	----

标黄色为本次作业

积极推进操作系统产品化

坚决构建基于国产操作系统的产业氛围

国内Linux系统的发展与国际上Linux系统的市场占有情况密切相关。

随着大数据与云计算等前沿技术的快速发展，越来越多的互联网公司开始构建自主控制与维护的云计算平台。具有开源与跨平台等属性的Linux系统，搭配采用Arm64芯片的计算平台，成为了这些互联网公司的首选技术方案。与此同时，Linux服务器端解决方案通过互联网企业迅速应用到了大数据与云计算的市场环境中。

但是，互联网企业使用Linux服务器端时，并未因采用Linux系统而形成典型的操作系统销售市场，专业的Linux系统厂商在服务器市场中还未形成较大的市场影响力。目前，国内海量的应用软件都是基于Windows系统的，因为该系统用户学习成本低、熟练程度高；而针对Linux系统，存在用户熟练程度低、对专业技术支持团队的依赖程度高、使用和维护成本高等问题。

积极推进操作系统产品化

坚决构建基于国产操作系统的产业氛围

为了更好、更快地解决上述问题，亟须建立顺畅的产品服务情况与用户使用预期的沟通渠道，通过了解并满足用户针对操作系统在使用、维护等方面的多种需求，提升国产（基于Linux系统进行二次开发的）操作系统的整体性能。同时，亟须确定一个兼具稳定性和一致性的开发接口，以使开发Linux系统应用软件的代码可以跨平台落地，进而减少因操作系统不同而导致的应用软件重复开发与测试工作，最终形成基于较为成熟的国产操作系统的产业氛围。

那么，应该如何建立产品服务情况与用户使用预期的沟通渠道？又应该如何确定兼具稳定性和一致性的开发接口呢？