第 7 章

内存管理

什么是内存?

内存(Memory)也被称为内存储器或主存储器，其作用是用于暂时存放CPU中的运算数据，以及与硬盘等外部存储器交换的数据。

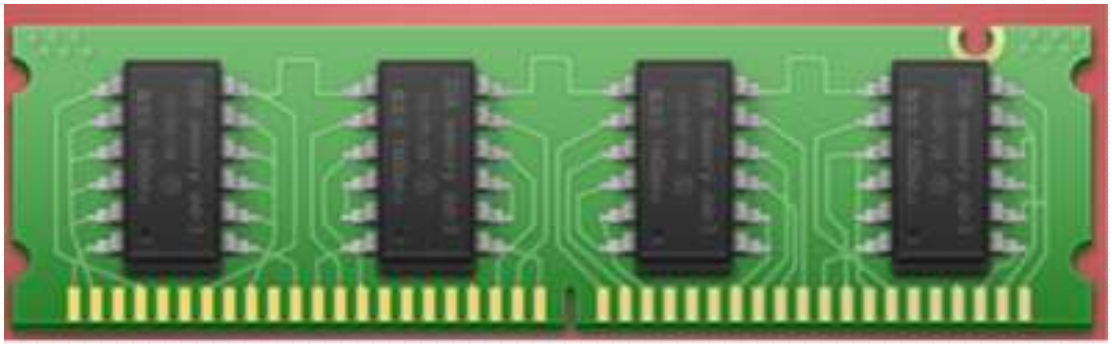


图 7-1 内存条

内存以字节为存储单位，内存地址空间（也称为物理地址空间）是指对内存编码的范围。所谓编码就是对每一个物理存储单元（一个字节）分配一个号码，通常叫作“物理地址”或“内存地址”。分配一个号码给一个存储单元的目的是为了便于找到它，完成数据的读写，这就是所谓的“寻址”

指令运行的原理

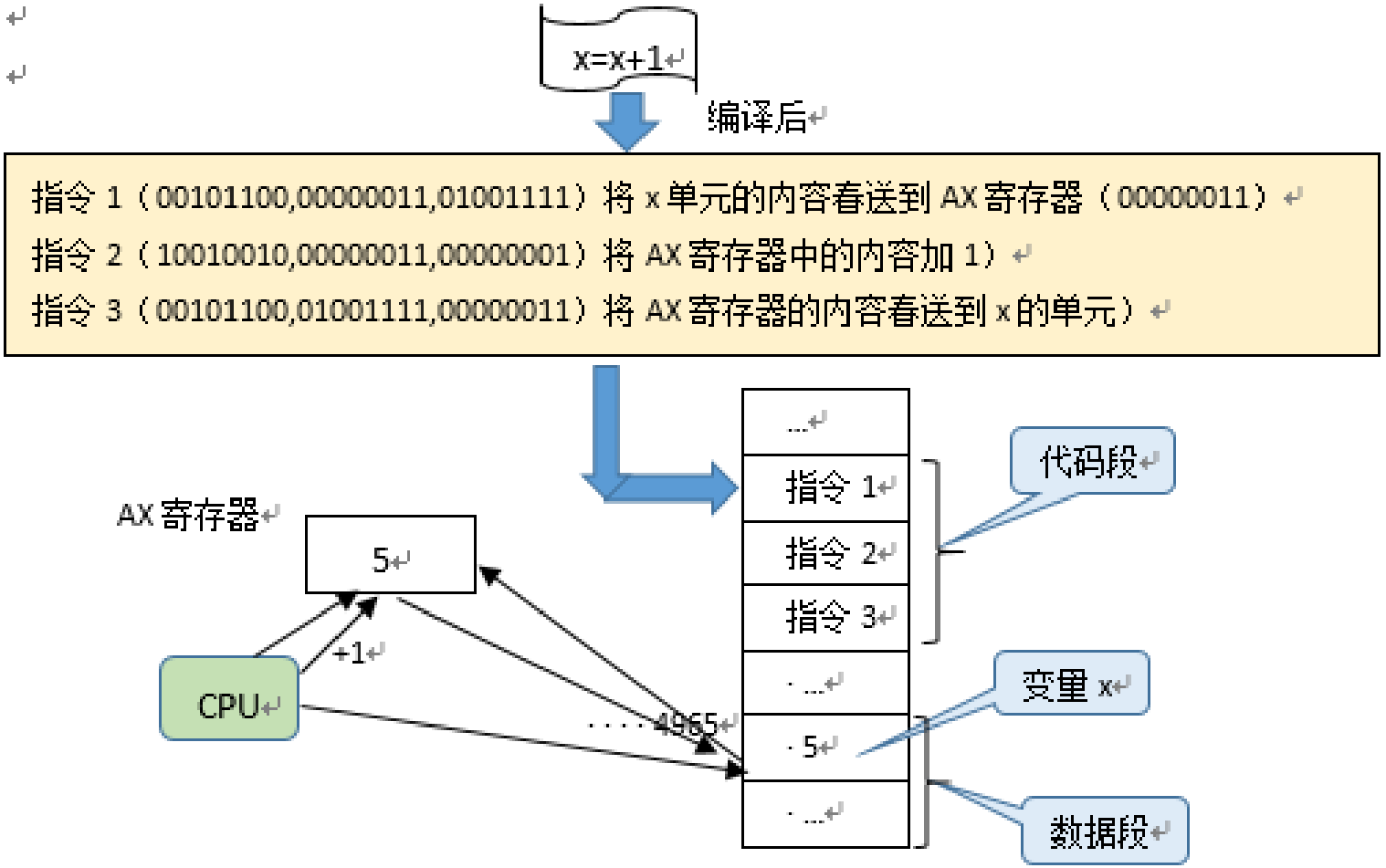


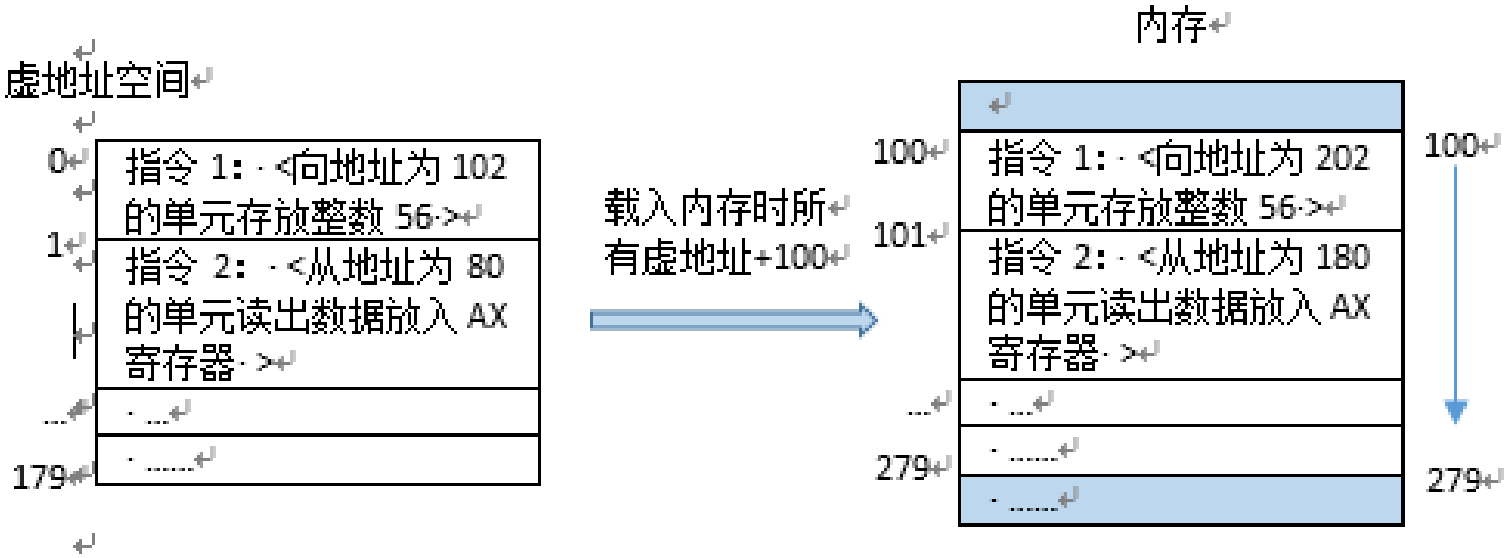
图 7-2 指令运行原理图

地址重定位

地址重定位可以分为动态地址重定位与静态地址重定位两种方式。

1）静态地址重定位是在程序执行之前，由装配程序完成的地址映射工作，如图7-3所示。

静态地址重定位的优点是不需要硬件支持，但是缺点是必须占有连续的内存空间，这就难以做到数据和程序的共享。



地址重定位（续）

2）**动态重定位：**程序在内存中如果发生移动，就需要采用动态的重定位方式。编译、链接后的装入模块的地址都是从“0”开始的。这种方式需要一个重定位寄存器的支持。

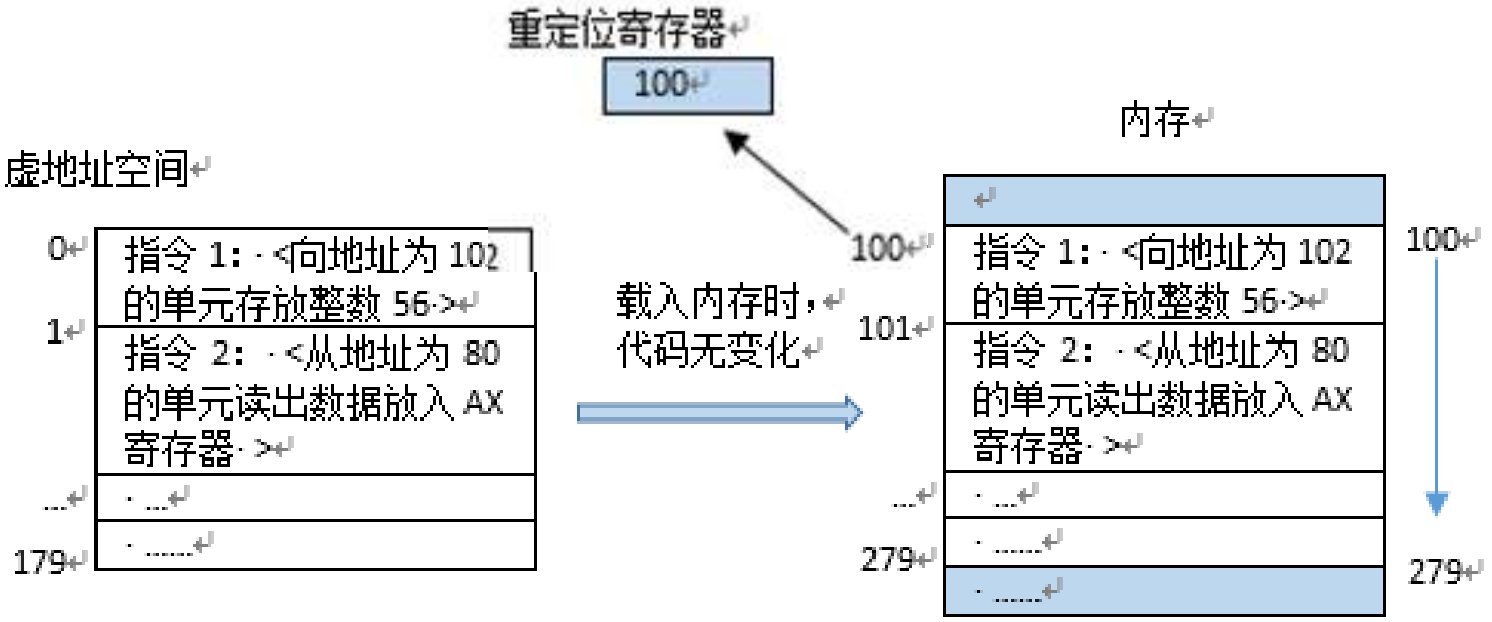


图 7-4 动态地址重定位

程序链接

1）**静态链接：**在程序运行之前，先将各目标模块及它们所需的库函数链接成一个完整的可执行程序，以后不再拆开，图7-5展示了静态链接的实质。

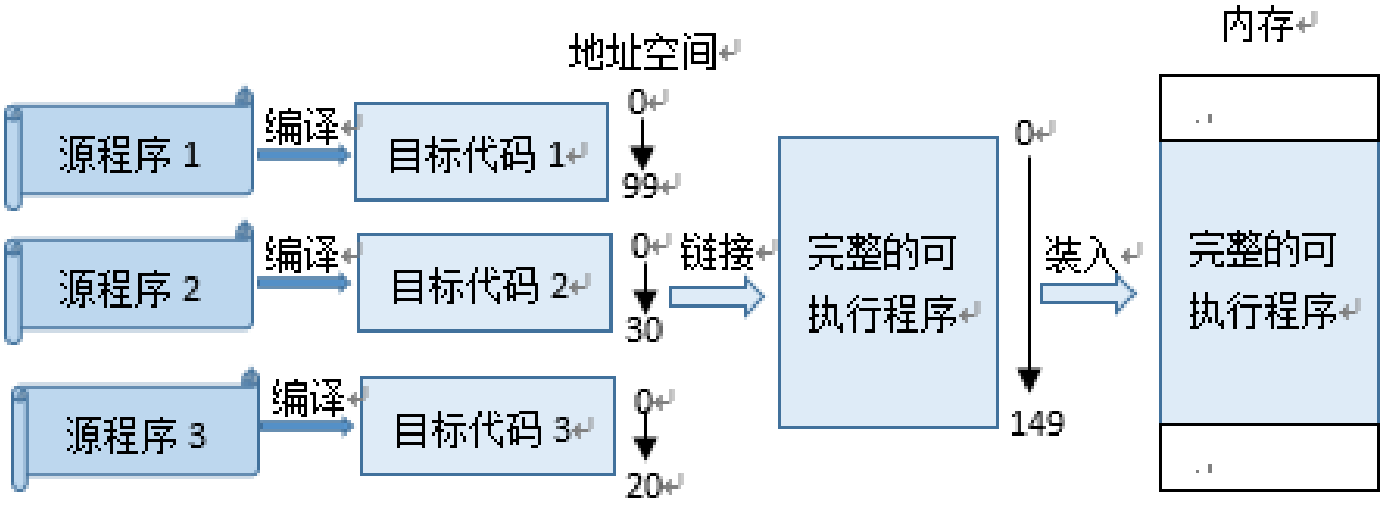


图 7-5 程序静态链接示意图

程序链接（续）

2）**装入时动态链接：**将源程序编译后所得到的一组目标模块，在装入内存时，采用边装入边链接的链接方式。

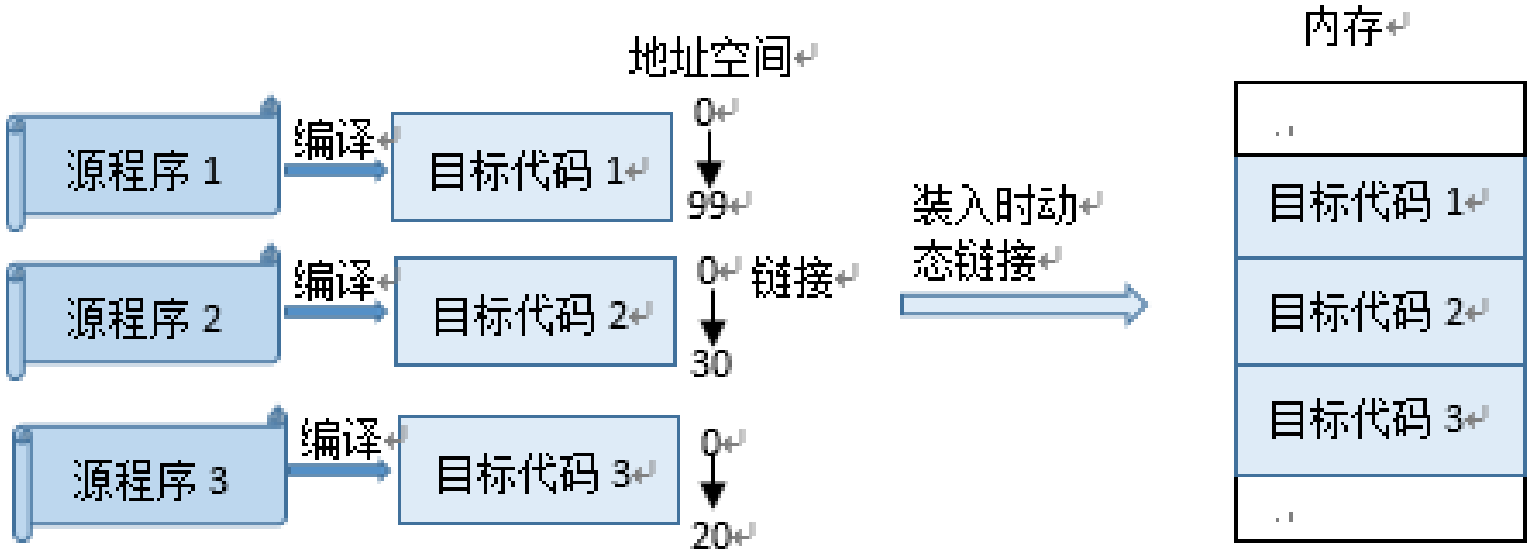


图 7-6 装入时动态链接示意图

程序链接（续）

3）运行时动态链接：对某些目标模块的链接，是在程序执行中需要该目标模块时，才对它进行的链接。其优点是便于修改和更新，便于实现对目标模块的共享。通常被链接的共享代码称为动态链接库(DLL)或共享库(shared library)。

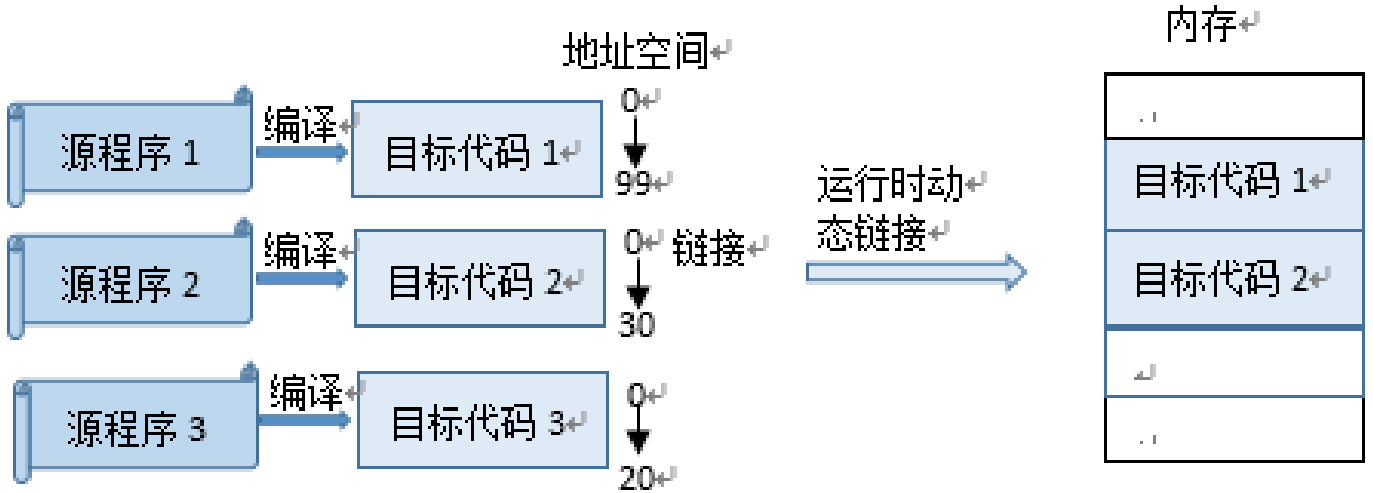


图 7-7 运行时动态链接示意图

内存覆盖

覆盖的基本思想是： 可以把程序划分为若干个功能上相对独立的程序段，如果内存空间有限，无法容纳程序的全部代码，此时可以把部分程序段先调入内存，首先将那些即将要访问的段放入覆盖区，其他段放在外存中，在需要使用这些外存中的段时，系统再将其调入覆盖区，替换覆盖区中原有的段

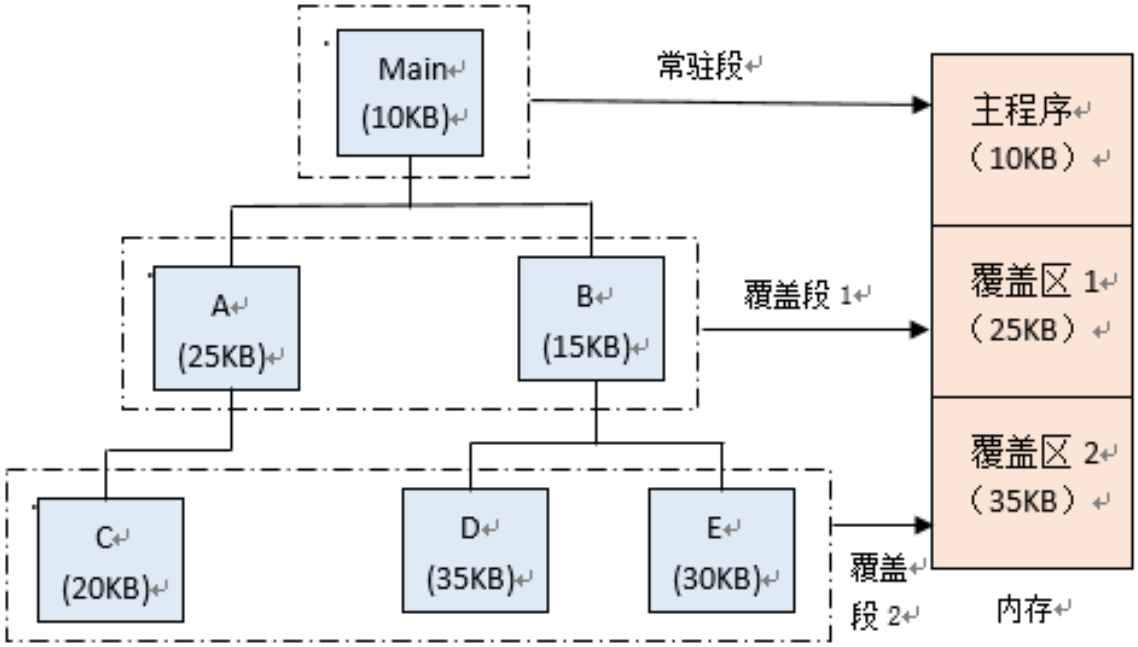


图 7-8 覆盖技术示意图

内存交换

交换的基本思想是：把处于等待状态（或在CPU调度原则下被剥夺运行权利）的程序从内存移到辅存，把内存空间腾出来，这一过程叫做“换出”；把准备好竞争CPU运行的程序从辅存移到内存，这一过程又称为“换入”，内存交换的示意图可见图7-9。

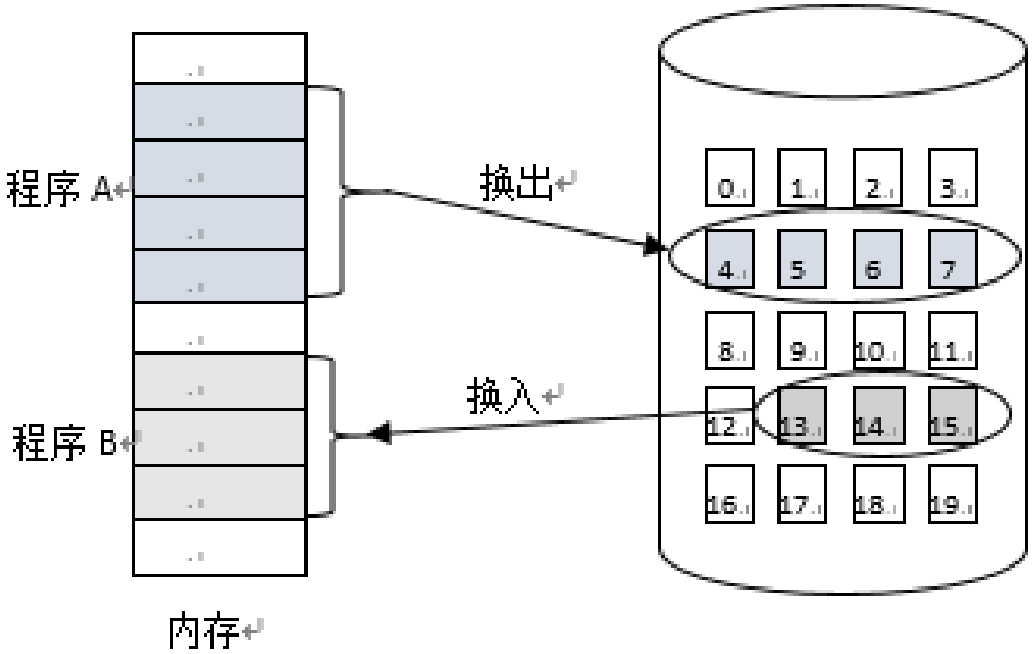
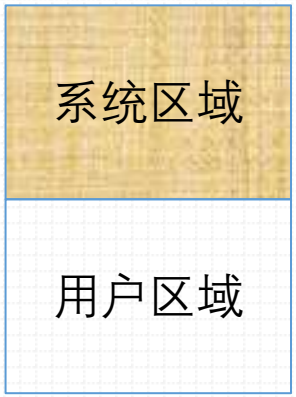


图 7-9 内存交换示意图

单一连续分配方案

采用这种管理方案时，内存被分成两个区域，一个是系统区域，仅供操作系统使用，可以驻留在内存的低地址部分,也可以驻留在高地址部分（通常设置在内存的低端）；另一个是用户区，它是除系统区以外的全部内存区域，这部分区域是提供给用户程序使用的区域，任何时刻主存储器中最多只有一个进程。所以，单一连续区存储管理只适用于单道程序系统。显然，单一连续分配方式早已经不能满足现代操作系统的需求，如果要支持多道程序系统，则必须要对该管理方法做出相应的改进，分区存储管理就是改进之后的连续内存分配方法。



固定大小的分区方案

表 7-1· 固定大小分区表

分区号	大小（K）	起始地址	分配状态
1	30	10	空闲
2	30	40	空闲
3	30	70	空闲
…	…	…	…

图 7-10· · 固定大小分区示意图

大小不等的分区方案

表 7-2 大小不等分区表

分区号	大小（K）	起始地址	分配状态
1	10	10	空闲
2	30	20	已分配
3	50	50	空闲
…	…	…	…

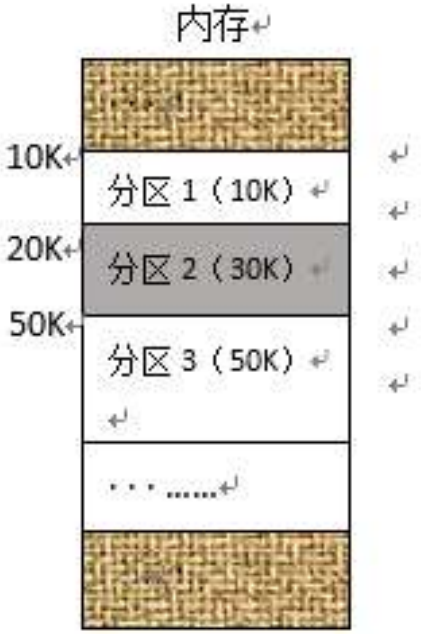


图 7-11 大小不等的分区示意图

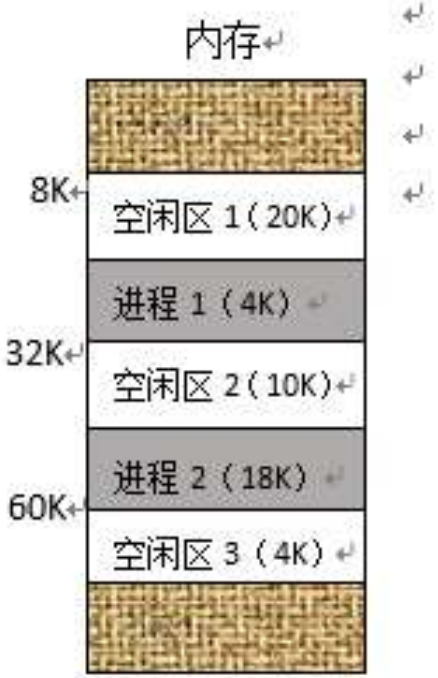
固定分区管理方式存在两个问题：

（1）一是程序可能太大而放不进任何一个分区中。

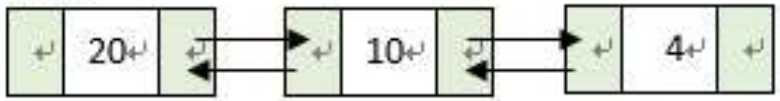
（2）会产生内部碎片。

动态分区的分配方案

动态分区分配又称为可变分区分配，是一种动态划分内存的分区方法。这种分区方法不预先将内存划分，而是在程序装入内存时，根据所需的大小动态地建立分区，并使分区的大小正好适合进程的需要。



空闲区链表



空闲分区表

空闲区号	大小（K）	起始地址	分配状态
1	20	8	空闲
2	10	32	空闲
3	4	60	空闲

图 7-12 动态分区方案中的数据结构

1、首次适应算法

若有一个进程申请分配 40 KB 大小的空间

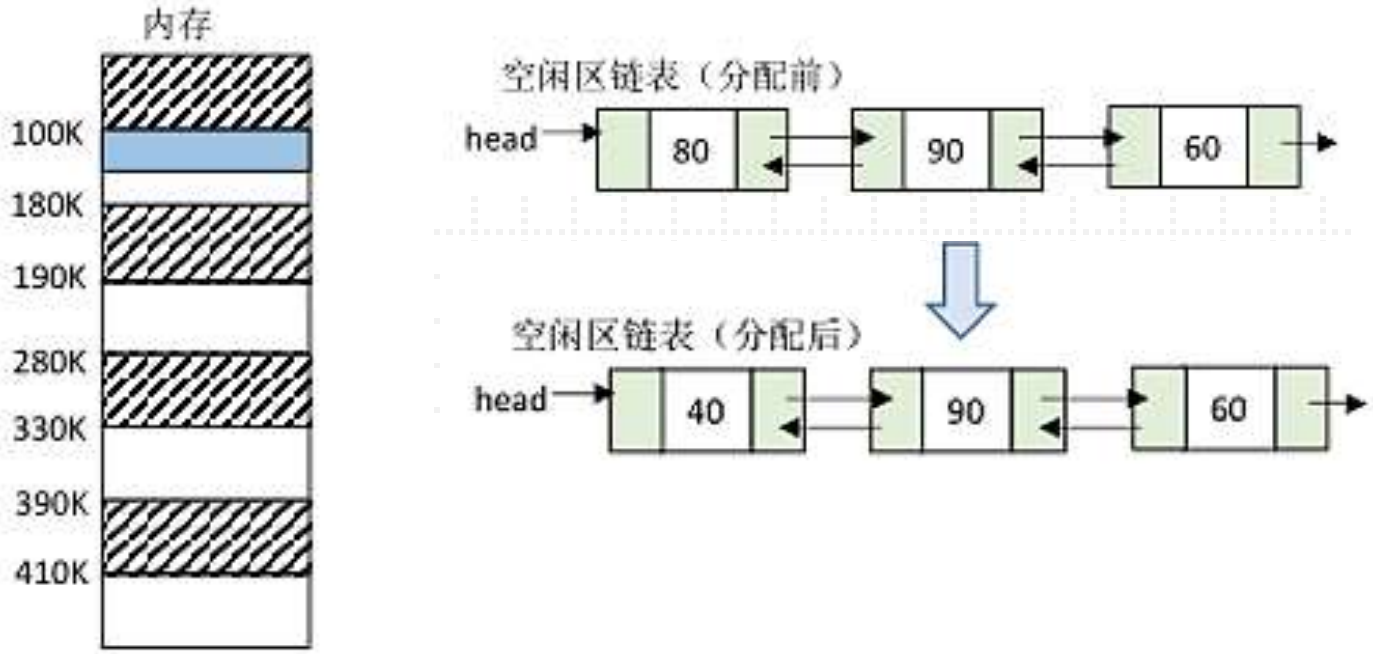


图 7-13　首次适应分配算法

2、最佳适应算法

若有一个进程申请分配 40 KB 大小的空间

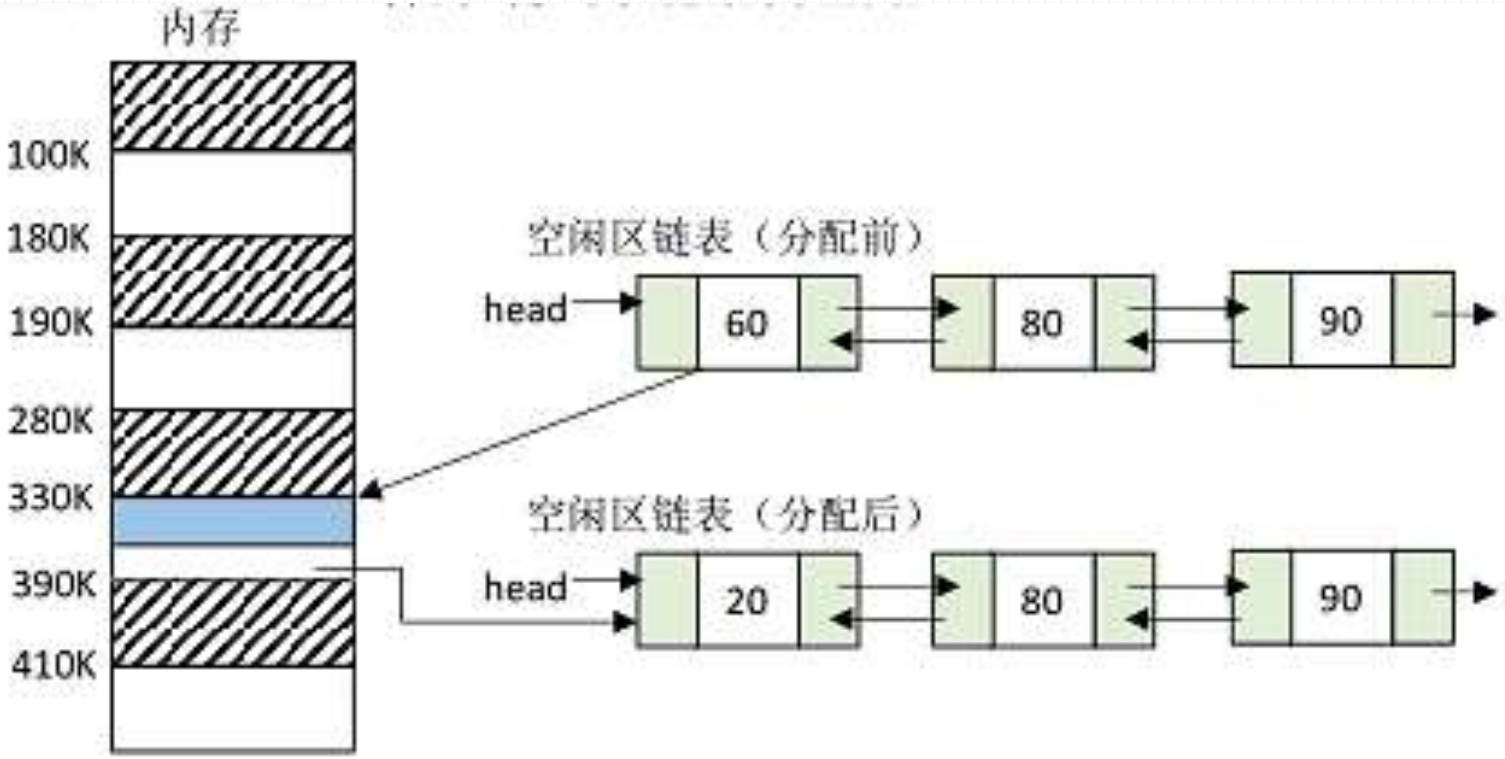


图 7-14　最佳适应分配算法

2、最坏适应算法

若有一个进程申请分配 40 KB 大小的空间

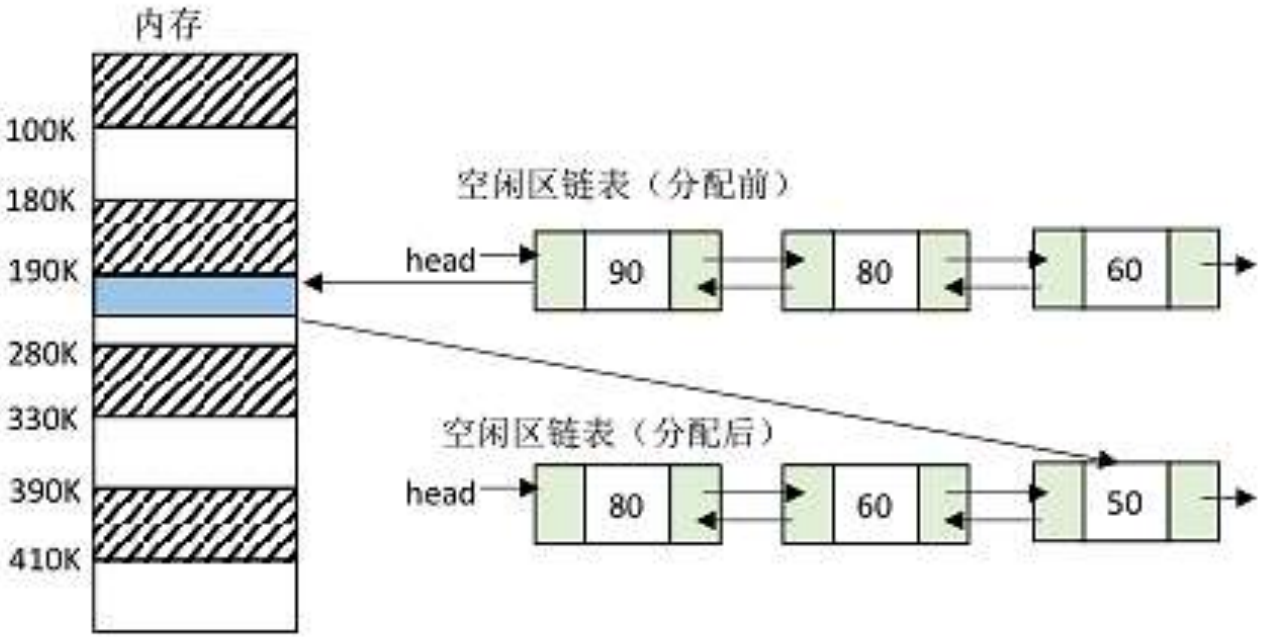


图 7-15　最坏适应分配算法

动态分区的分配方案（续）

动态分区的分配方案中没有内部碎片，但是会产生外部碎片。外部碎片问题的解决方法可以考虑这样一些措施：

1）规定分割空闲区的下限值。分割空闲区时，若剩余部分小于下限值，则此空闲区不进行分割，而是全部分配给用户进程。

2）内存拼接技术。将所有空闲区集中构成一个大的空闲区。拼接的时机一般在释放区回收的时候或者系统找不到足够大的空闲区的时候；或者定期进行拼接，但是拼接操作会消耗大量的系统资源。

3）解除程序占用连续内存才能运行的限制（分页和分段就是借鉴了这种思想）把程序分拆为多个部分装入到不同分区，充分利用碎片。

分页存储管理的基本思想

分页存储管理将进程的逻辑地址空间分成若干个大小相等的页面（虚页），并为各页加以编号。相应地，也把内存的物理地址空间分成若干个页帧（页框），同样也为他们加以编号。在为进程分配内存空间时，将进程中的若干个虚页分别装入到多个不相邻的页帧中。

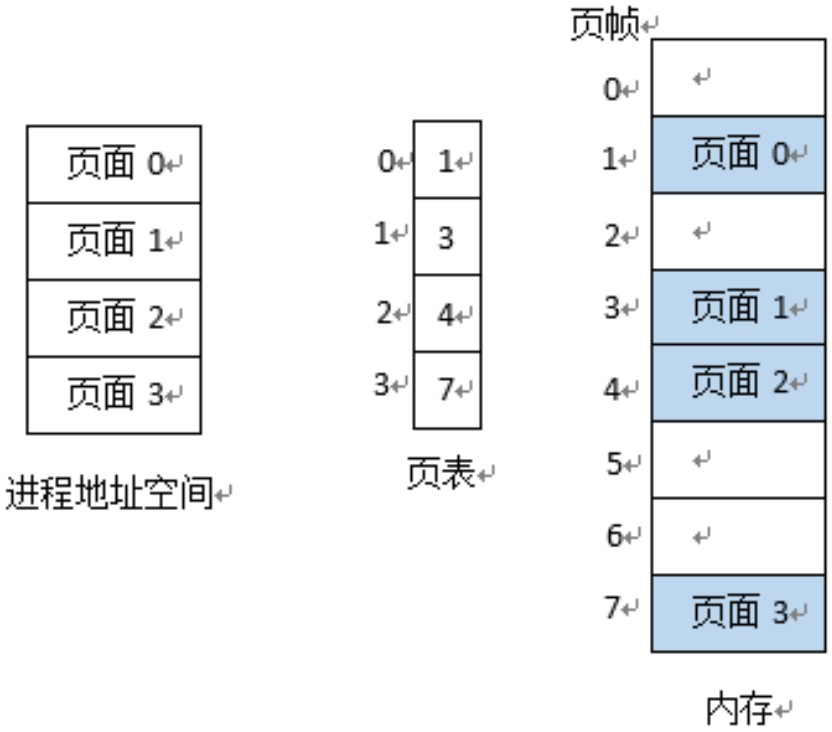


图 7-16 · 页式存储管理示意图

分页存储管理的基本思想（续）

分页系统中的地址结构如下：

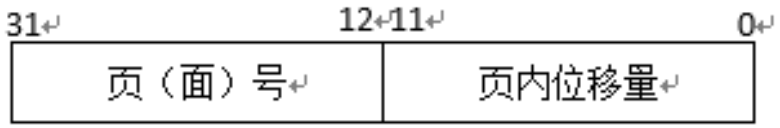


图 7-17 页式系统中的虚地址结构

假设某系统物理内存大小为4GB，页面大小为4KB，则每个页表项至少应该为多少字节?

4GB= 2^{32} B，**4KB= 2^{12} B**，因此**4GB**的内存总共会被分为**2^{32}/ 2^{12}= 2^{20}**个内存块，内存页帧号的范围应该是**0~2^{20} -1**，因此至少要**20**个二进制位才能表示这么多的内存页帧号，故每个页表项至少要**3**个字节才够(每个字节**8**个二进制位，**3**个字节共**24**个二进制位)

地址变换机构

1. 基本的地址变换机构

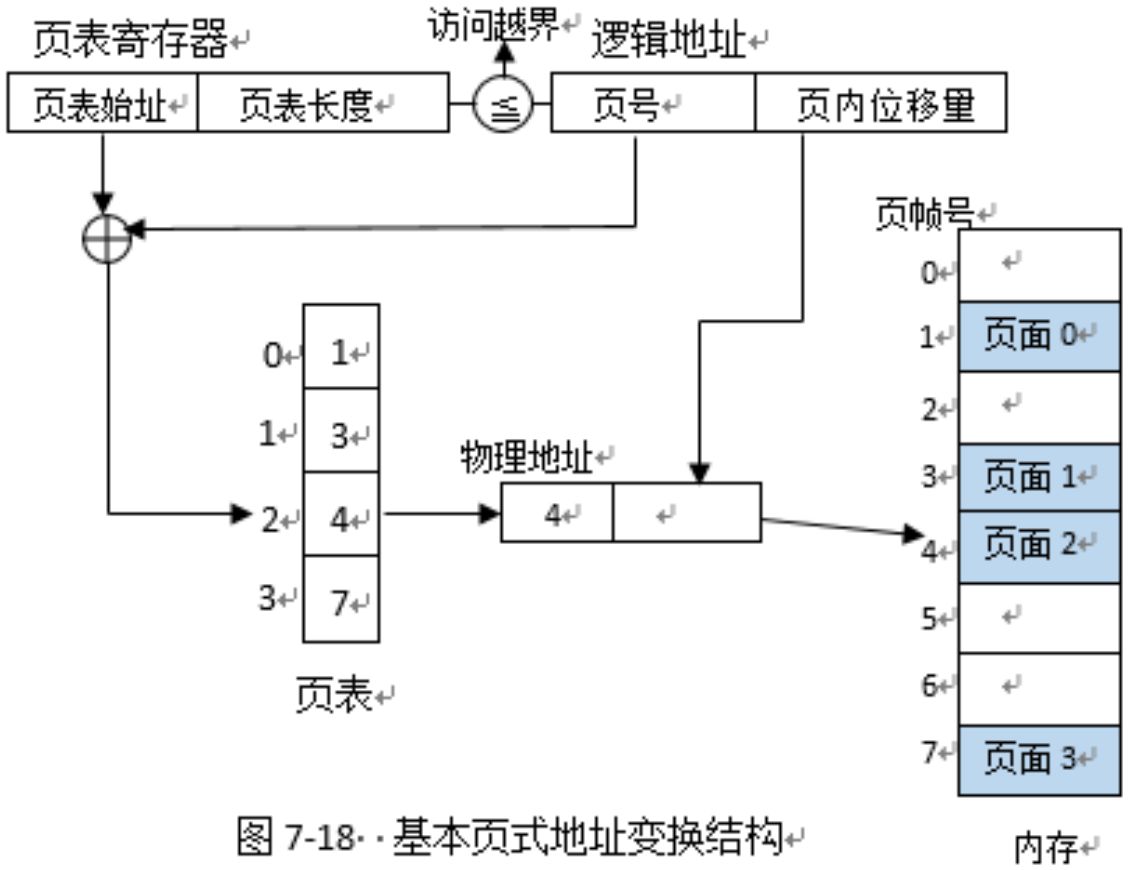


图 7-18 基本页式地址变换结构

CPU执行指令时，每次存取一个数据时，都要两次访问内存

某虚拟存储器的用户空间共有 32 个页面，每页 1KB，主存 16KB。假定某时刻系统为用户的第 0、1、2、3 页分别分配的物理块号分别为 5、10、4、7，那么虚地址 0A5CH 对应的物理地址是 （　　）。

0	5
1	10
2	4
3	7

某虚拟存储器的用户空间共有 32 个页面，每页 1KB，主存 16KB。假定某时刻系统为用户的第 0、1、2、3 页分别分配的物理块号分别为 5、10、4、7，那么虚地址 0A5CH 对应的物理地址是 （　　）。

0	5
1	10
2	4
3	7

虚地址 0A5CH【H 表示十六进制，该虚地址为十六进制的 0A5C】，可知该逻辑地址的二进制形式为 :0000 1010 0101 1100

每页大小为1K,故页内偏移占用10位，那么，前面 6 位为页号，因此虚地址

0000 10【页号】10 0101 1100 【页内偏移】，页号为 2

综上，虚地址 0A5CH 对应的物理块号为 4，块内的偏移地址为：

10 0101 1100

物理地址直接拼接物理块号与页内偏移量：

0001 0010 0101 1100（翻译为十六进制为 125CH）

地址变换机构

2. 具有快表的地址变化机构

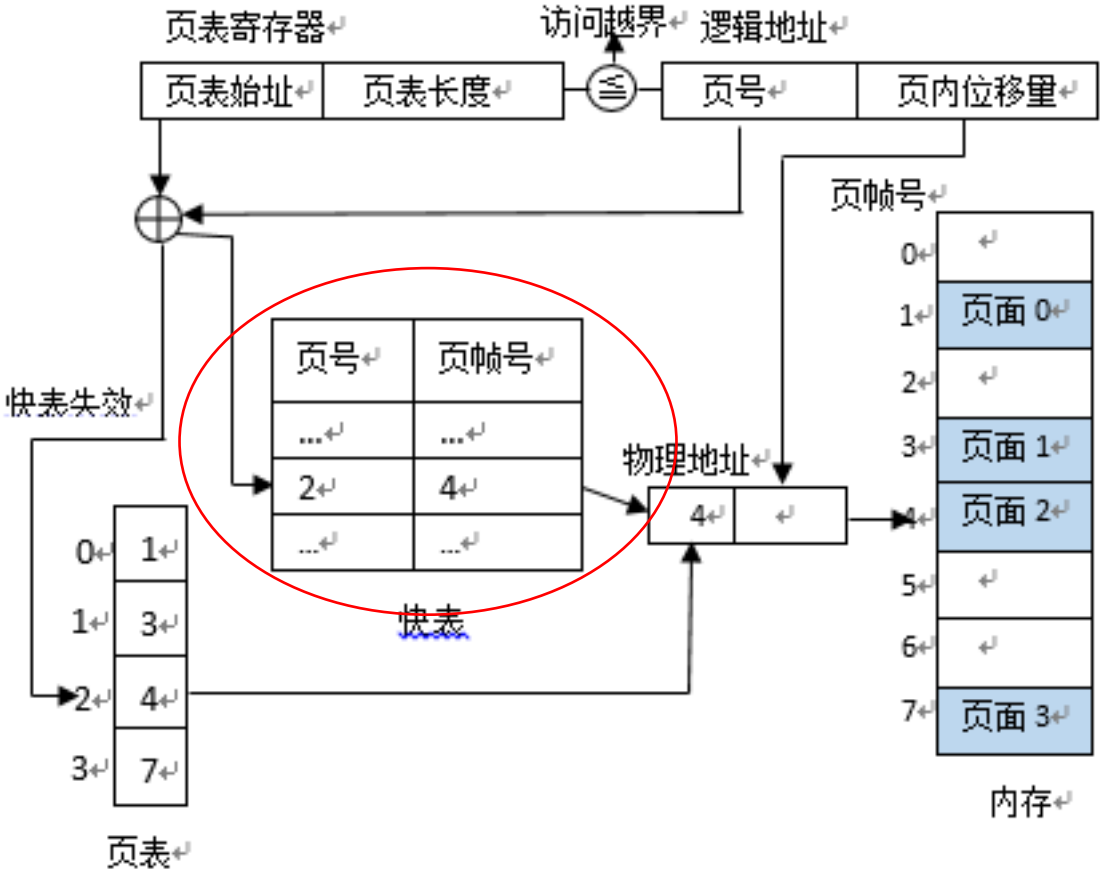


图 7-19 · 具有快表的地址变换结构

地址变换机构

2. 具有快表的地址变化机构

有效访问时间(EAT)可定义为：从进程给出特定的虚地址的访问请求，经过地址变换，到内存中找到对应的实际物理地址单元并取出数据，所需花费的总时间。

假定访问一次内存的时间为t，λ为查找快表所需要的时间，a为快表的命中率，则：

1）基本分页存储管理方式中，有效访问：EAT=2t

2）引入快表的分页存储管理方式中，通过查询快表可直接得到虚页所对应的页帧号，由此拼接形成实际物理地址，减少了一次访问内存，缩短了进程访问内存的有限时间。：

$EAT=(\lambda+t)\ a+(2t+\lambda)(1-a)=2t+\lambda-t\times a$

地址变换机构

2. 具有快表的地址变化机构

例：在一个引入了快表的分页存储系统中，页表放在内存中，多数活动页表项都可以存在快表中。假定一次内存访问时间是100ns，查找快表的时间为20ns，若快表的命中率是85%，则访问内存的有效时间为多少？若快表的命中率为50%，那么访问内存的有效时间为多少？

解：当快表的命中率为85%时，有效存取时间为：

EAT=0.85×（20+100）＋(200+20)×(1－0.85)＝135ns

当快表的命中率为50%时，有效存取时间为：

EAT=0. 5×（20+100）＋(200+20)×(1－0.5) ＋ 100＝170ns

两级或多级页表

由于现在计算机可以拥有很大的逻辑地址空间，因此页表变得非常大，占用很多的内存空间，而且页表需要连续存放在内存中，这显然是不现实的。怎么办呢？我们可以采用采用两级（或多级）页表的方式。

图 7-20 二级页表的逻辑地址

两级或多级页表

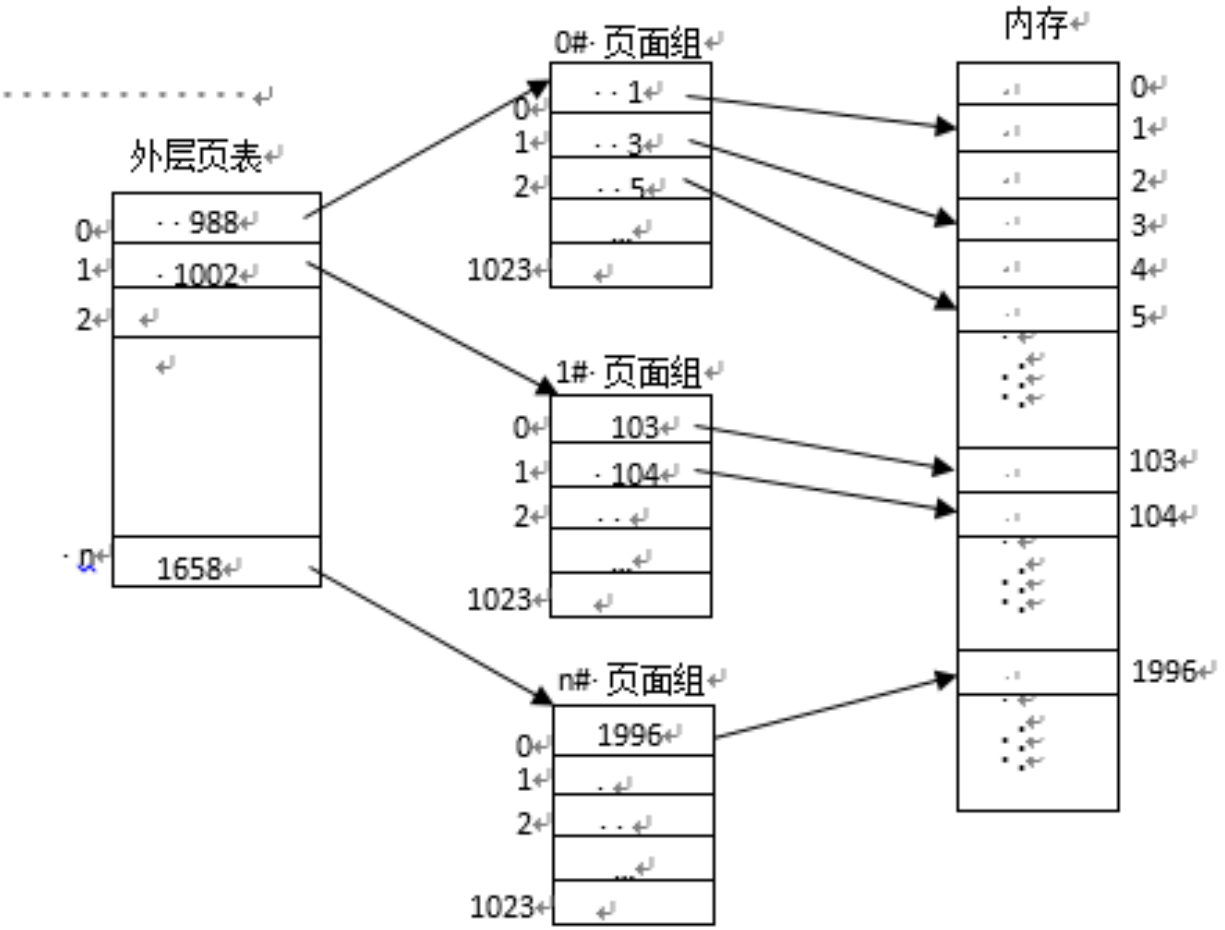
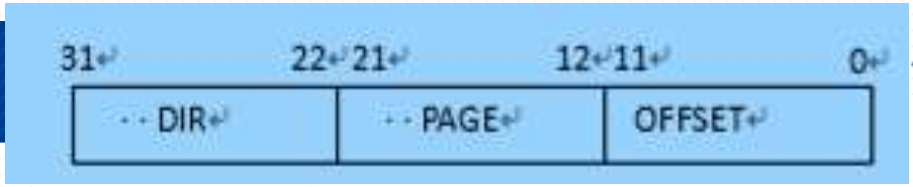
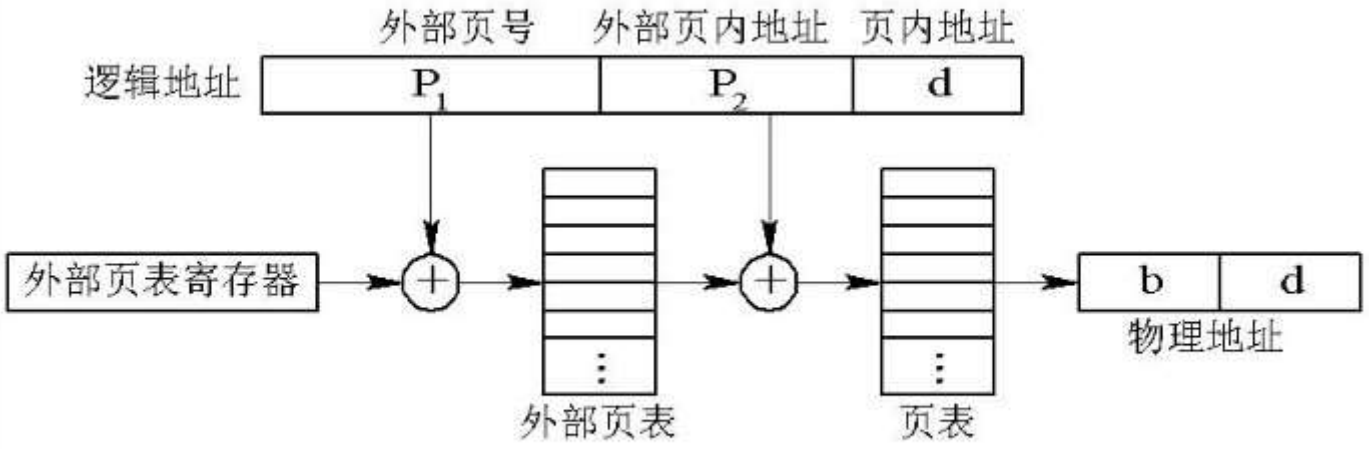


图 7-21 二级页表结构

两级或多级页表

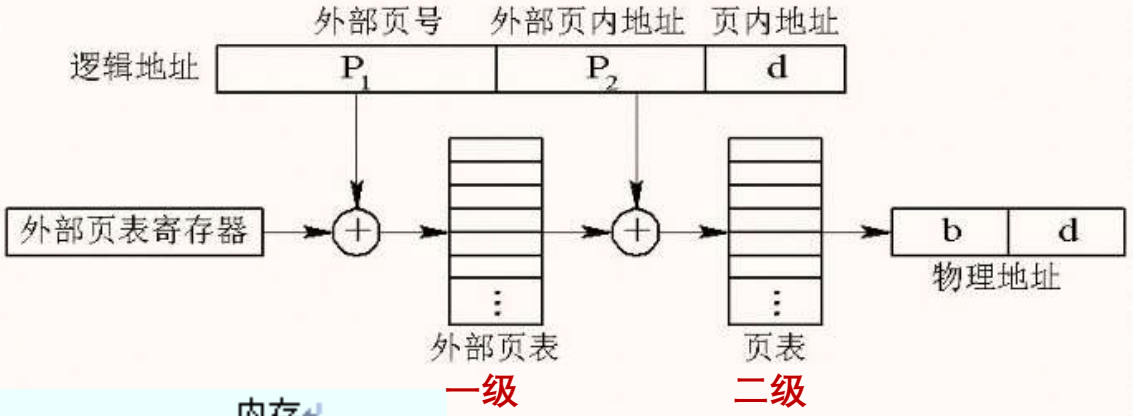


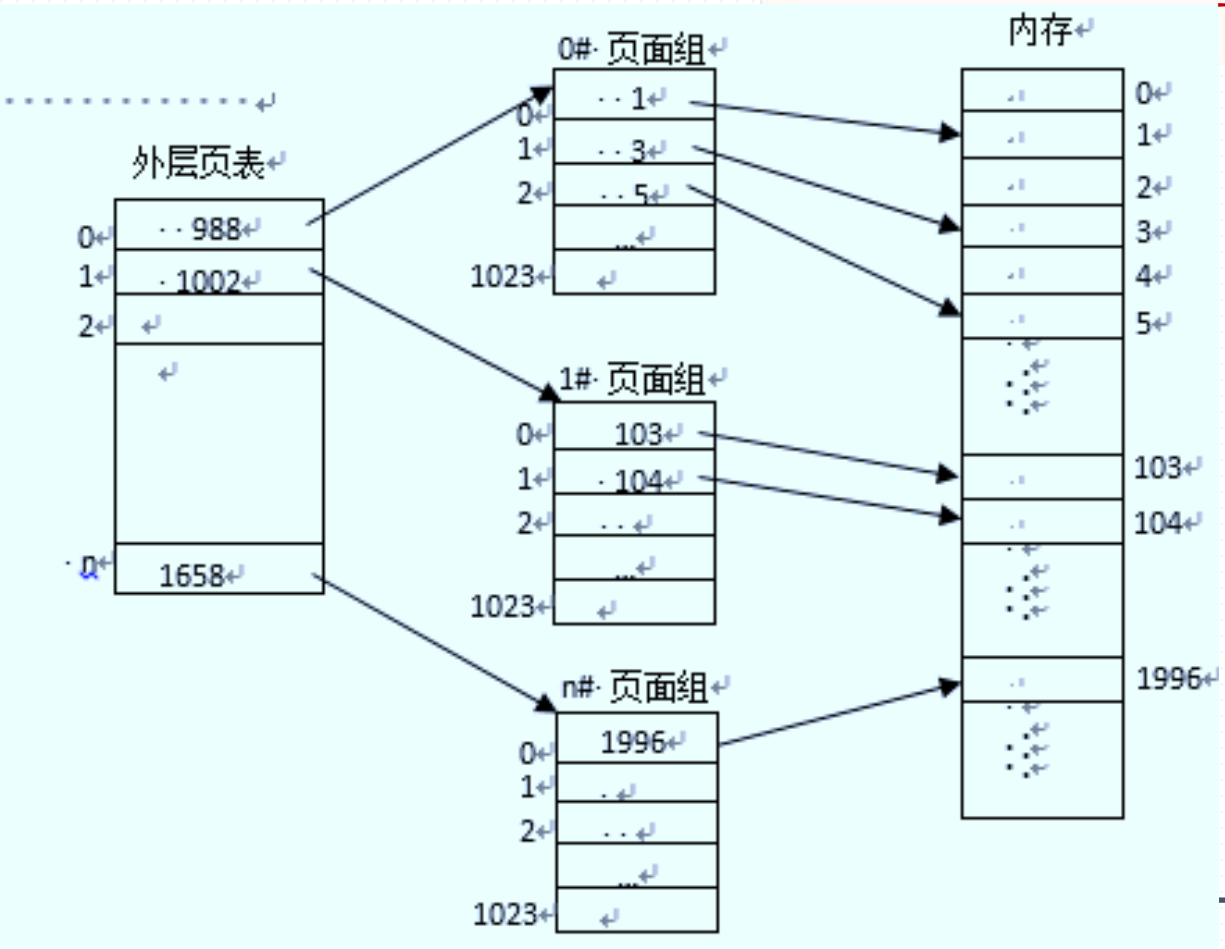
在地址变换机构中需要增设一个外层页表寄存器，用于存放外层页表的始址。



7.4 分页存储管理

两级或多级页表





段式存储管理的基本思想

在段式存储管理中，进程的地址空间由若干个逻辑分段组成，每一个分段是一组逻辑意义完整的信息集合，并且有自己名字（段名），每一段都是以0开始的连续的一维地址空间，整个进程的地址空间是二维的，由段号与段内位移量（段内地址）组成。

段号	段内位移量

段式存储管理的基本思想（续）

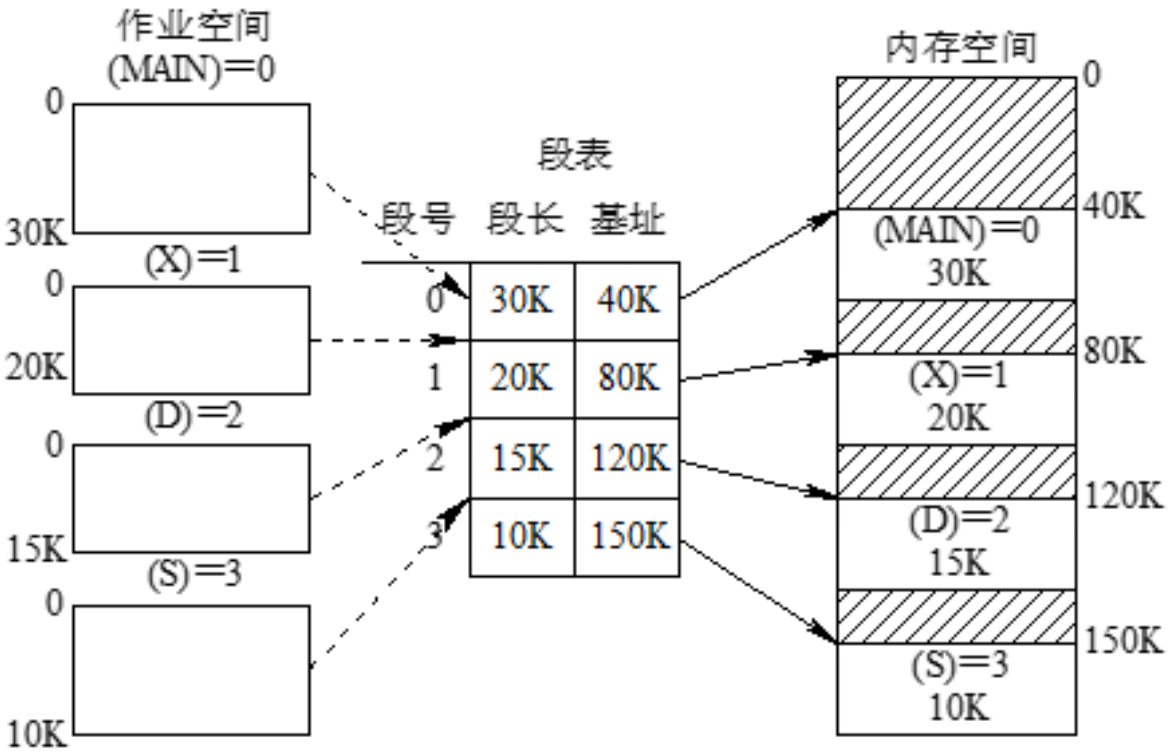
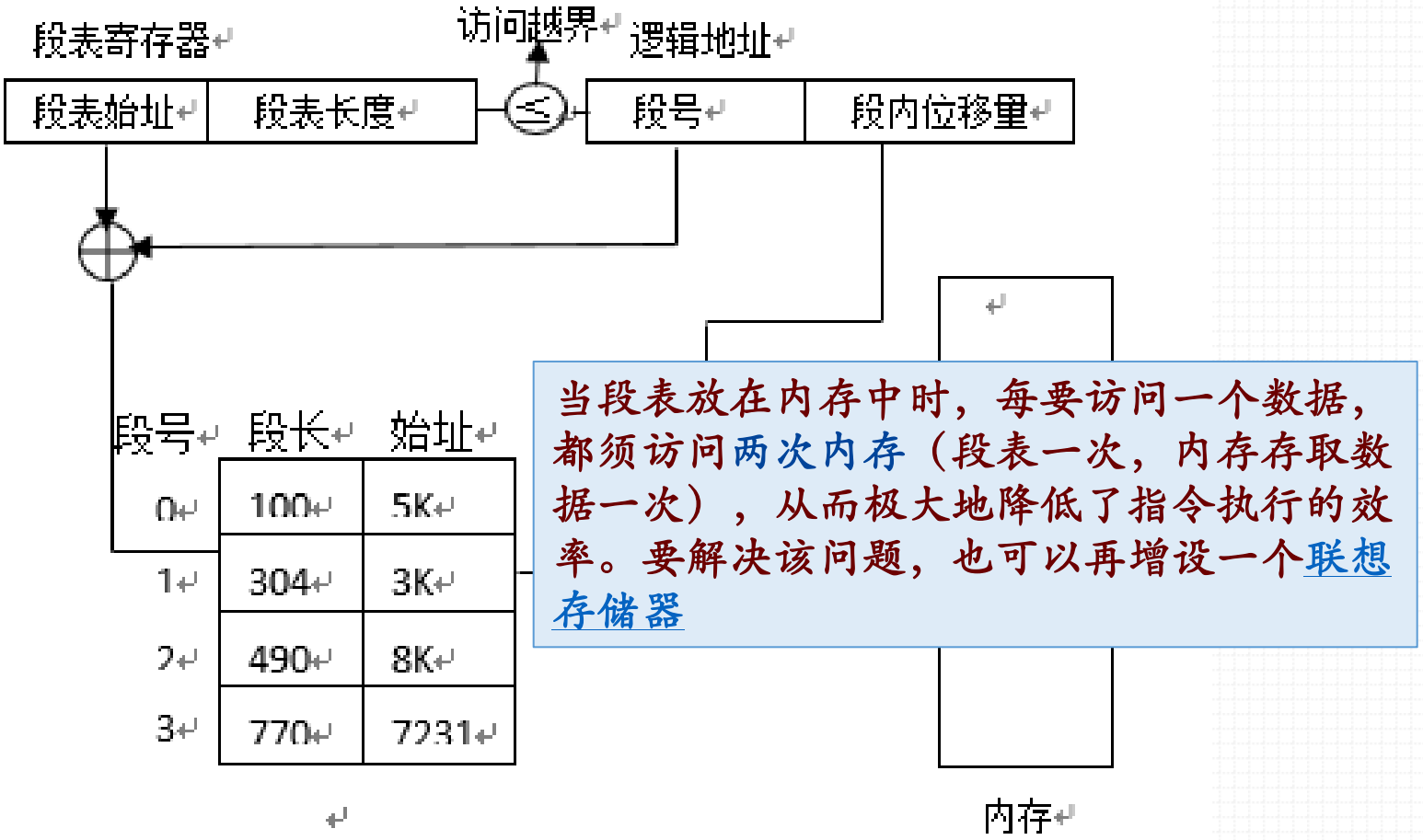


图 7-22 段式存储管理示意图

段式存储管理的地址变换



当段表放在内存中时，每要访问一个数据，都须访问两次内存（段表一次，内存存取数据一次），从而极大地降低了指令执行的效率。要解决该问题，也可以再增设一个联想存储器

图 7-23 段式存储管理地址变换机构

共享段表

为了实现分段共享，可在系统中配置一张共享段表，所有需要共享的段都在共享段表中占有一表项。

表 7-4 共享段表

共享段名	段长	起始地址	共享计数 count	进程名	段号	存取控制	...

记录共享该段的进程数目

不同的进程可以使用不同的段号去共享该段

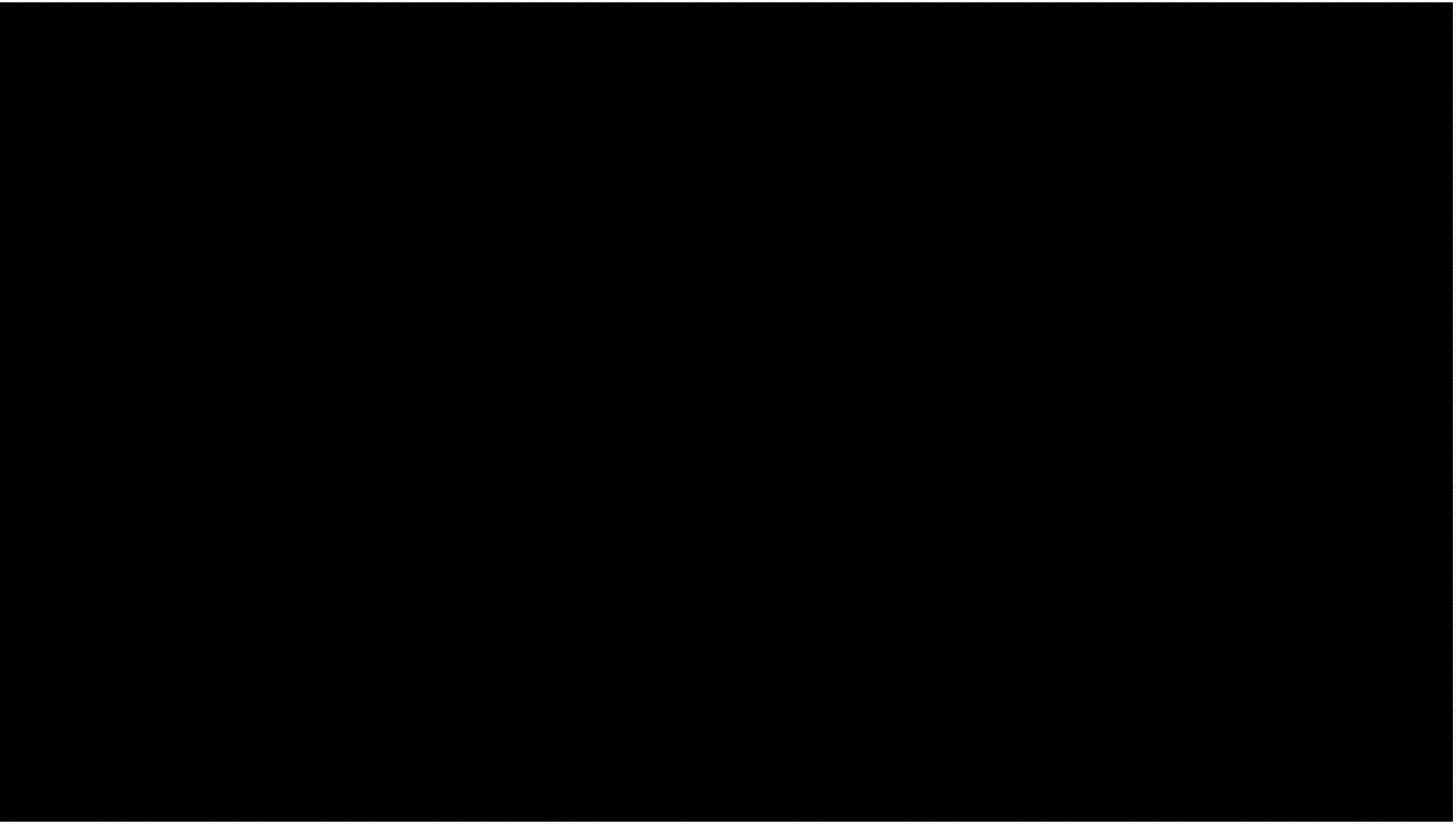


表7-5 分页式存储管理与分段式存储管理对比表

	分页	分段
管理思想	页是信息的物理单位，是为了管理主存的方便而划分的，页的大小固定不变，实现了程序的非连续存放	段是信息的逻辑单位，它是根据用户的需要划分的，段的大小是不固定的，它由其完成的功能决定，因此段对用户是可见的；
虚地址	页式向用户提供的是一维地址空间，其页号和页内偏移是机器硬件的功能。	段式向用户提供的是二维地址空间
共享与存储访问控制	可以实现页面共享，但使用受到诸多限制，访问控制困难	便于共享逻辑完整的信息，易于实现存取访问权限控制
动态链接	不支持动态链接	支持动态链接

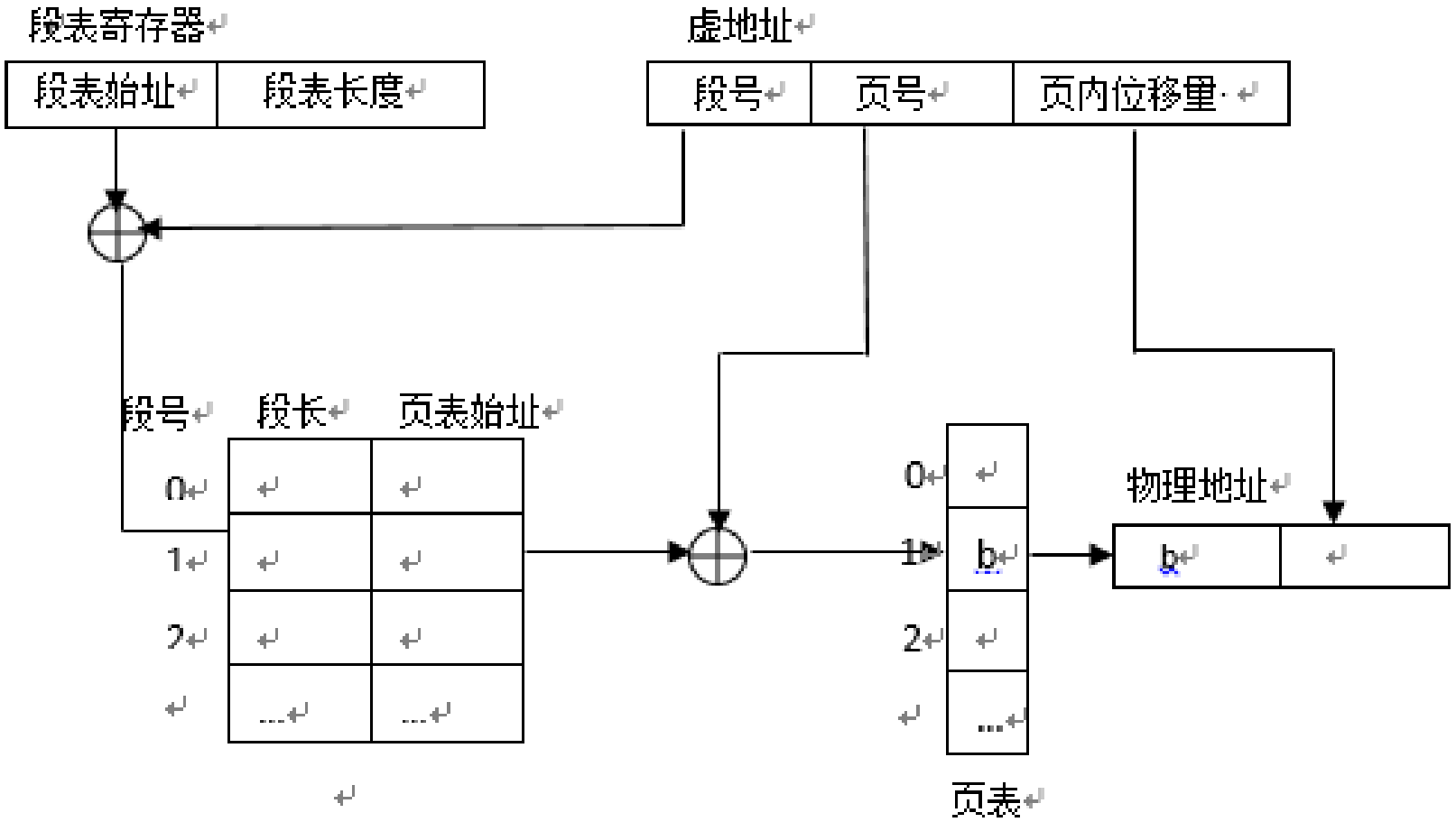
段页式管理的基本思想：

一个进程有一个自己的二维地址空间。一个进程中所包含的独立逻辑功能的程序或数据仍被划分为段，并有各自的段号S，对于S中的程序或数据，则按照一定的大小将其划分为不同的页。

段页式管理系统的进程虚拟地址由三部分组成：
段号s、页号p和页内位移量d。
程序员可见的仍是段号s和段内位移量w。
p和d是由地址变换机构把w的高几位解释成页号p，以及把剩下的低位解释为页内地址d而得到的。

段号	页号	页内位移量

段页式管理的地址变换机构



防止地址越界

可采用界限寄存器的方式，由操作系统给定进程的上、下界寄存器内容，从而限定每个用户进程的内存范围，禁止进程的越界访问。

防止操作越权

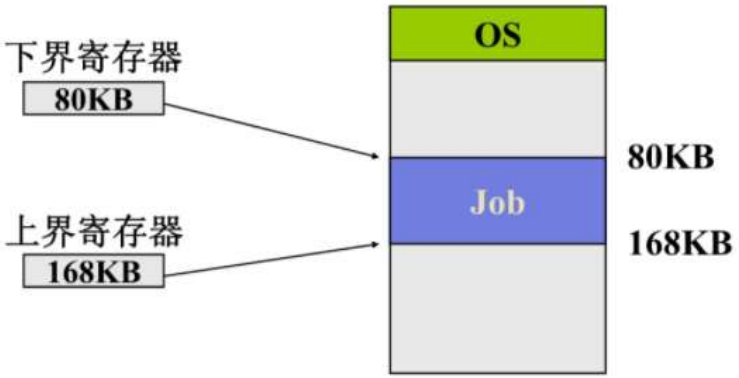
对属于自己区域的信息，可读可写；对公共区域中允许共享的信息或获得授权许可的信息，可读而不可修改；对未获授权使用的信息，不可读、不可写。

可以通过设置内存区域的访问控制字段来进行管控。

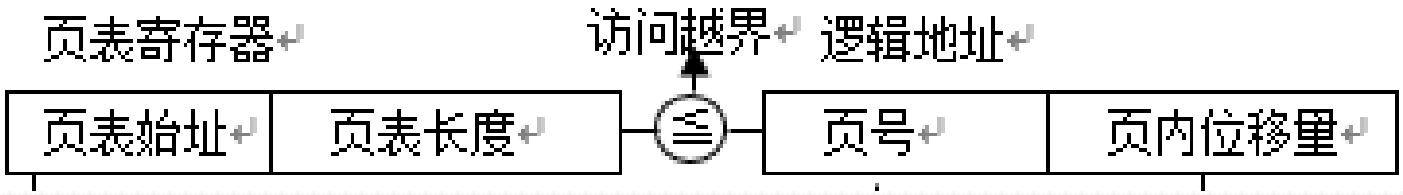
	防止地址越界	防止操作越权
连续存储管理	使用上、下界寄存器	
分页管理	基址-限长存储保护	存取控制字段
分段管理	基址-限长存储保护	存取控制字段

	防止地址越界	防止操作越权
连续存储管理	使用上、下界寄存器	
分页管理	基址-限长存储保护	存取控制字段
分段管理	基址-限长存储保护	存取控制字段

- 处理机中设置上界和下界寄存器,处理机对每一条访存指令中的地址与界限寄存器比较,若不在上/下界间则产生越界中断,禁止访问。
- 也可用基址/限长寄存器方式实现



	防止地址越界	防止操作越权
连续存储管理	使用上、下界寄存器	
分页管理	基址-限长存储保护	存取控制字段
分段管理	基址-限长存储保护	存取控制字段

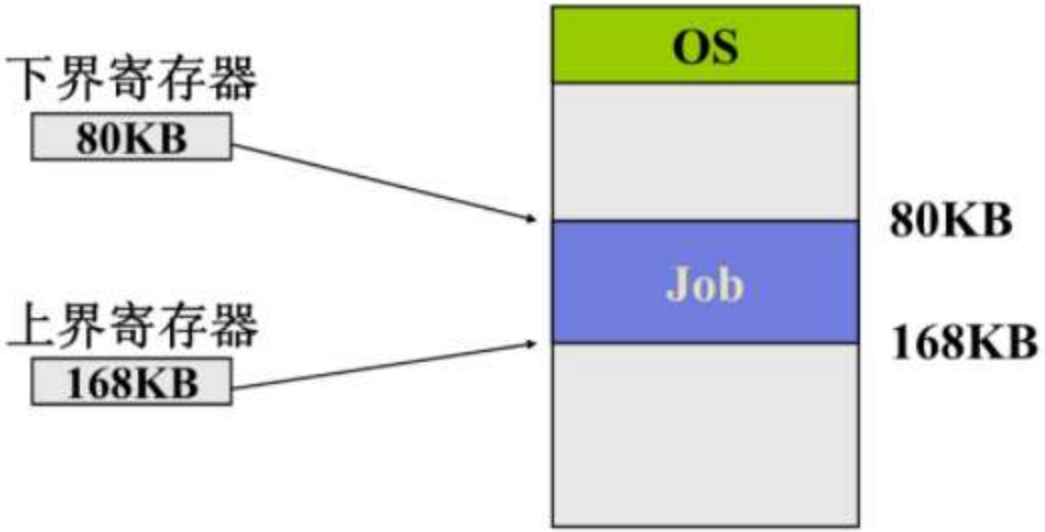


即使在简单的分页系统中，也常在页表的表项中设置一存取控制字段，用于对该存储块中的内容加以保护。

	防止地址越界	防止操作越权
连续存储管理	使用上、下界寄存器	
分页管理	基址-限长存储保护	存取控制字段
分段管理	基址-限长存储保护	存取控制字段

存储保护一般以硬件保护机制为主，软件为辅，因为完全用软件实现系统开销太大，速度成倍降低。当发生越界或非法操作时，硬件产生中断，进入操作系统处理。

- 处理机中设置上界和下界寄存器,处理机对每一条访存指令中的地址与界限寄存器比较,若不在上/下界间则产生越界中断,禁止访问。
- 也可用基址/限长寄存器方式实现



所谓虚拟内存技术，又称为虚拟存储技术，就是把内存与外存有机的结合起来使用，从而得到一个容量很大的“内存”，该技术可以让系统看上去具有比实际物理内存大得多的内存空间并为实现多道程序的执行创造了条件。

当操作系统支持虚拟存储技术的时候，进程只需要将部分代码载入内存即可使得程序在内存中运行。当下一条需要执行的指令不在内存时，则需要将新的程序段调入内存，将旧的程序段置换出去。在计算机技术中将内存中的程序段复制回外存的做法叫做“换出”，而将外存中程序段映射到内存的做法称为“换入”。经过不断有目的的换入和换出，处理器就可以运行一个大于实际物理内存的应用程序了。或者说，处理器似乎拥有了一个大于实际物理内存的内存空间。

局部性原理

局部性原理是指程序在执行过程中的一个较短时间内，所执行的指令地址或操作数地址分别局限于一定的内存区域中。

- 程序可能包含若干循环。一般情况下循环是由相对较少的指令组成，在循环过程中，计算分页系统　存储键保护被限制在程序中相邻部分。
- 很少出现连续的过程调用，相反，程序中过程调用的深度限制在小范围内，一段时间内，指令引用被局限在很少几个过程中。
- 对于连续访问数组之类的数据结构，在短时间内往往是对内存中相邻位置的数据的操作。
- 程序中有些部分是彼此互斥的，不是每次运行时都会调用，如出错处理代码等。

因此，进程部分代码载入内存是可行的

请求分页储存管理

在进程开始运行之前，仅装入当前要执行的部分页面即可运行；在执行过程中，如果所要访问的页面已调入内存，则进行地址转换，得到欲访问的内存物理地址，如果不在内存中，则产生一个“缺页中断”。如果此时内存能容纳新页，则启动磁盘I/O将其调入内存，如果内存已满，则通过页面置换功能将当前所需的页面调入。

页号	页帧号	状态位	访问位	修改位	辅存地址

图 7-25 扩展后的页表

请求分页储存管理

缺页中断机构

地址变换

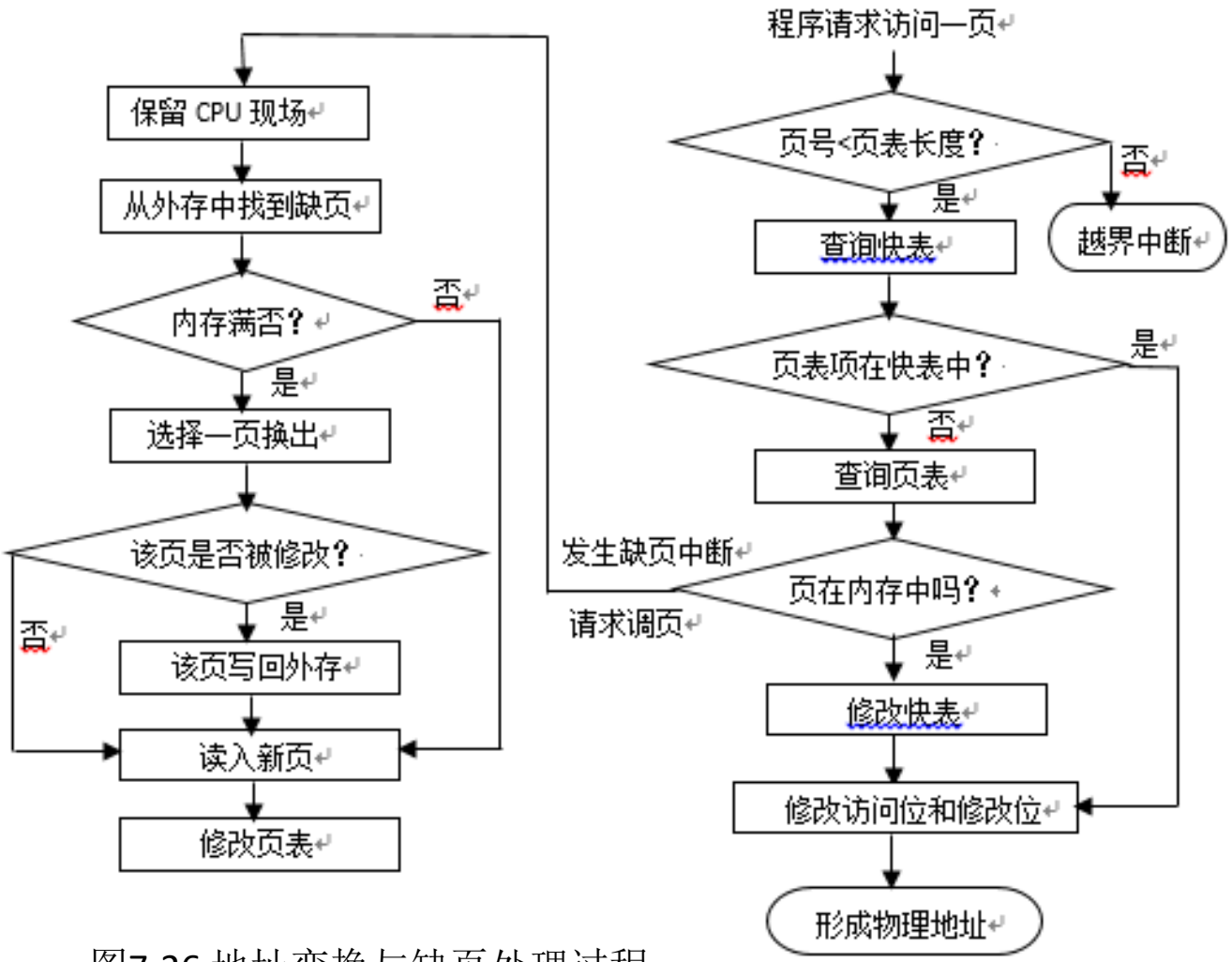


图7-26 地址变换与缺页处理过程

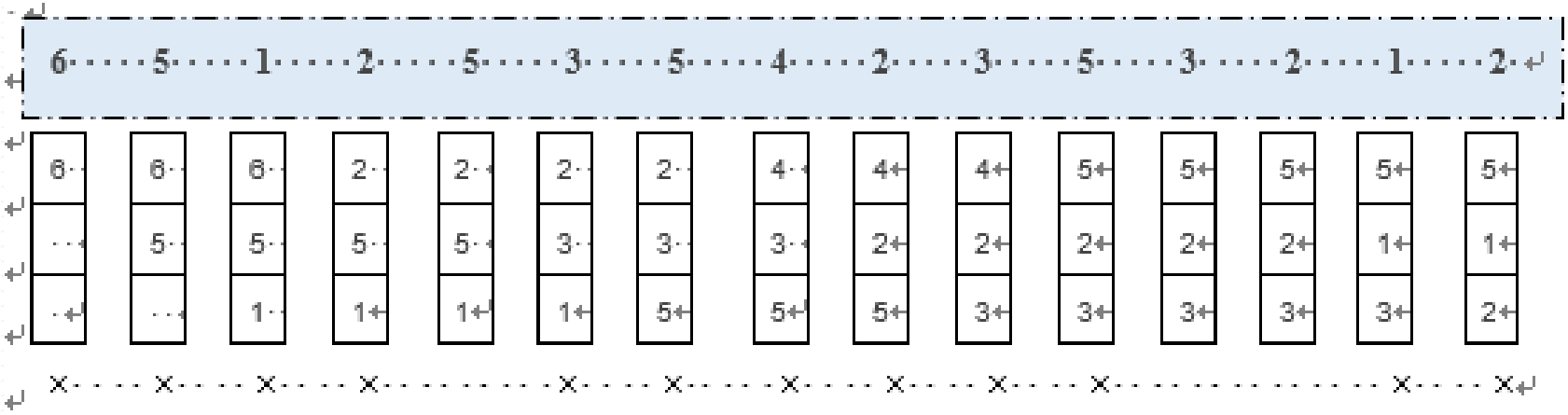
页面置换算法

1. 先进先出页面置换算法（FIFO）

总是选择在主存中停留时间最长（即最老）的页面进行置换，即先进入内存的页，先被换出内存。

场景：假设某进程被分配了3个页帧，程序页面访问序列为：

6, 5, 1, 2, 5, 3, 5, 4, 2, 3, 5, 3, 2, 1, 2

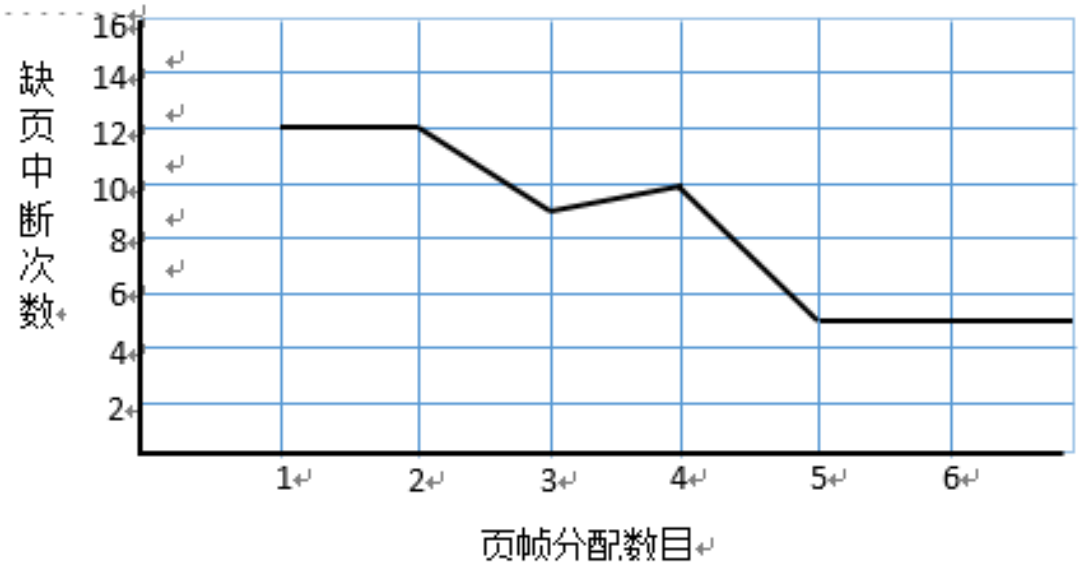


页面置换算法（续）

1. 先进先出页面置换算法（FIFO）---Belady现象

假设页面的引用序列如下：

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



Belady现象是FIFO算法特有的！

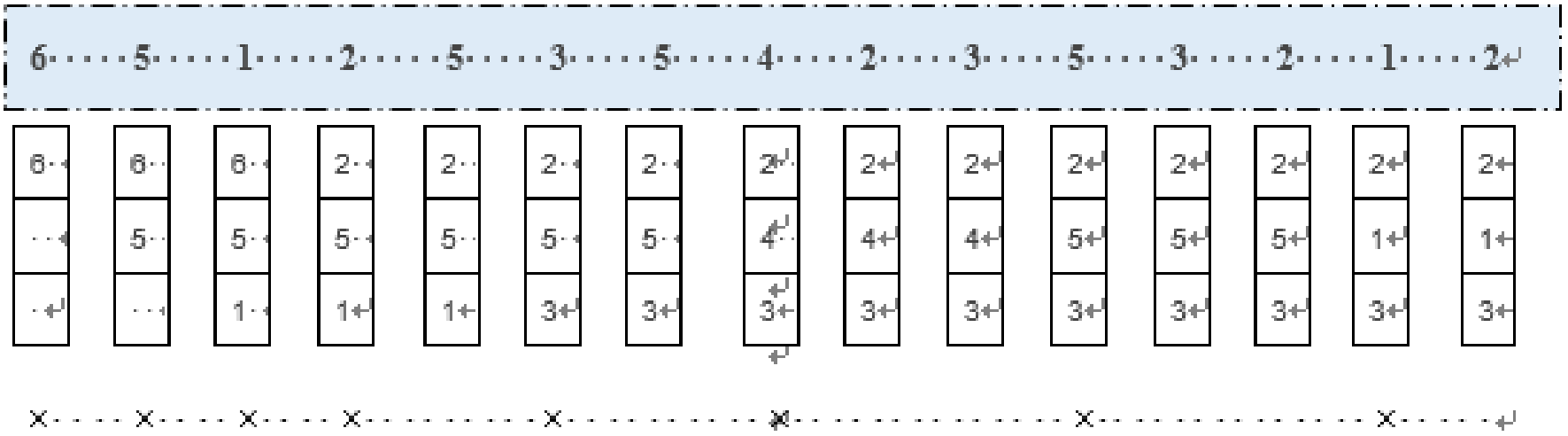
页面置换算法（续）

2. 最优页面置换（OPT）算法

最优置换算法（OPT）是指，其所选择的被淘汰页面，将是以后永不使用的，或许是在最长(未来)时间内不再被访问的页面

场景：假设某进程被分配了3个页帧，程序页面访问序列为：

6, 5, 1, 2, 5, 3, 5, 4, 2, 3, 5, 3, 2, 1, 2



页面置换算法（续）

3. LRU 页面置换算法

如果以最近的过去作为不久将来的近似，那么就可以把过去最长一段时间里不曾被使用的页面置换掉。它的实质是，当需要置换一页时，选择在最近一段时间里最久没有使用过的页面予以置换。

LRU算法需要实际硬件的支持。其问题是怎么确定最后使用时间的顺序，对此有两种可行的办法。

（1）第一种方法是使用计数器：在最简单的情况下，为每个页表条目关联一个使用时间域，并为 CPU 添加一个逻辑时钟或计数器。

（2）实现 LRU 置换的另一种方法是采用堆栈来实现的。

页面置换算法（续）

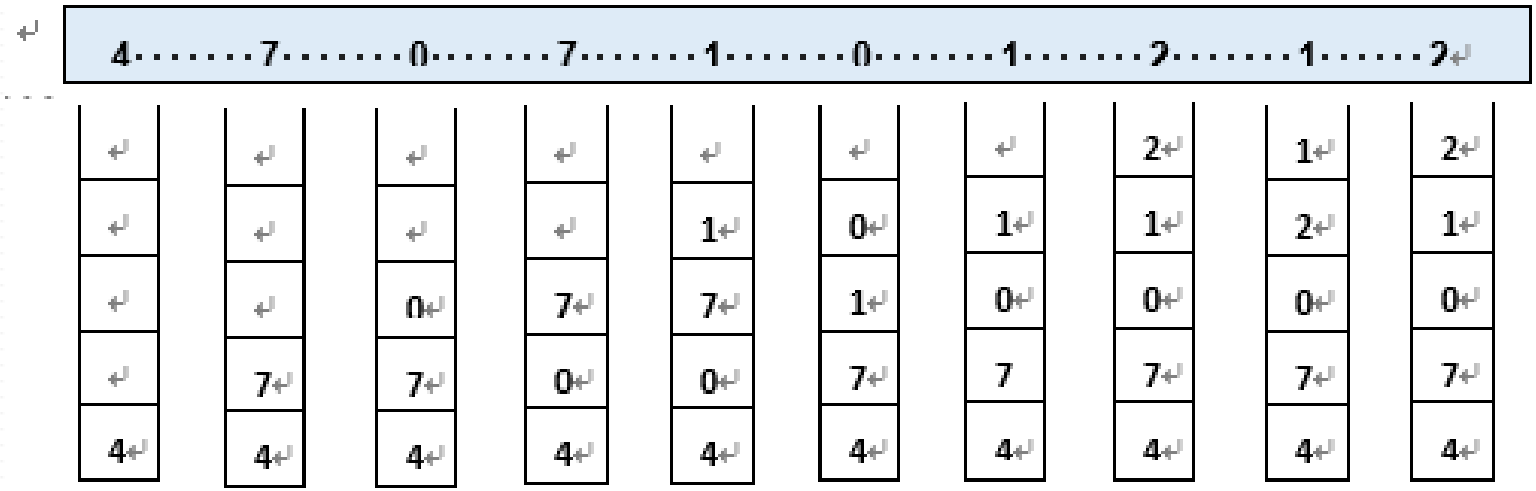
3. LRU 页面置换算法

采用堆栈来实现LRU

每当页面被引用时，它就从堆栈中移除并放在顶部。这样，最近使用的页面总是在堆栈的顶部，最近最少使用的页面总是在底部。

4	7	0	7	1	0	1	2	1	2

页码堆栈的变化过程为：



页面置换算法（续）

3. LRU 页面置换算法

场景：假设某进程被分配了3个页帧，程序页面访问序列为：

6, 5, 1, 2, 5, 3, 5, 4, 2, 3, 5, 3, 2, 1, 2

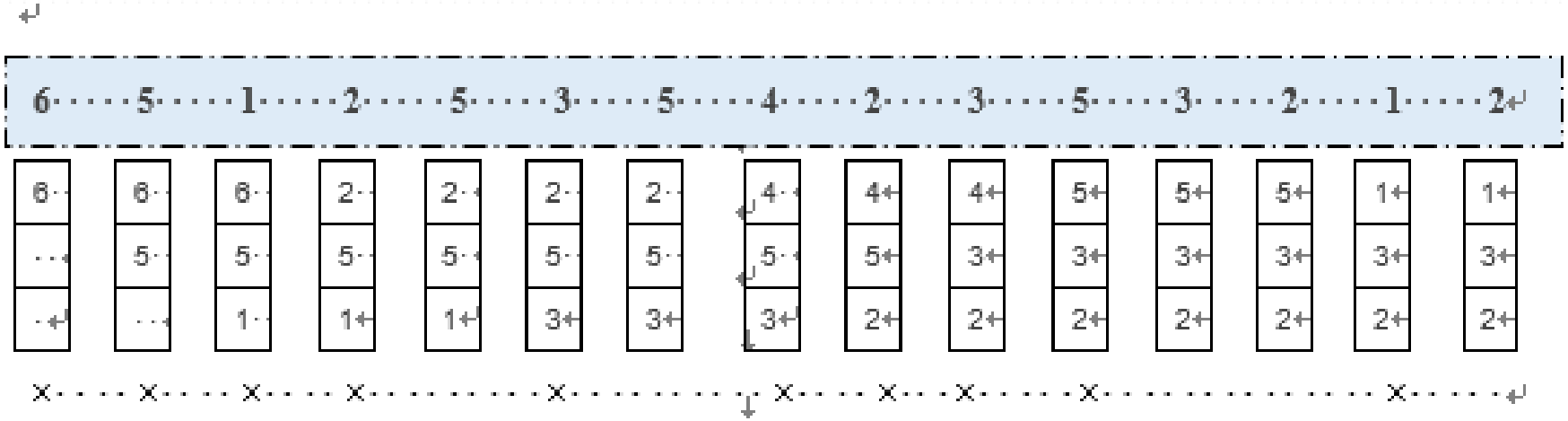


图 7-31 LRU 页面置换算法

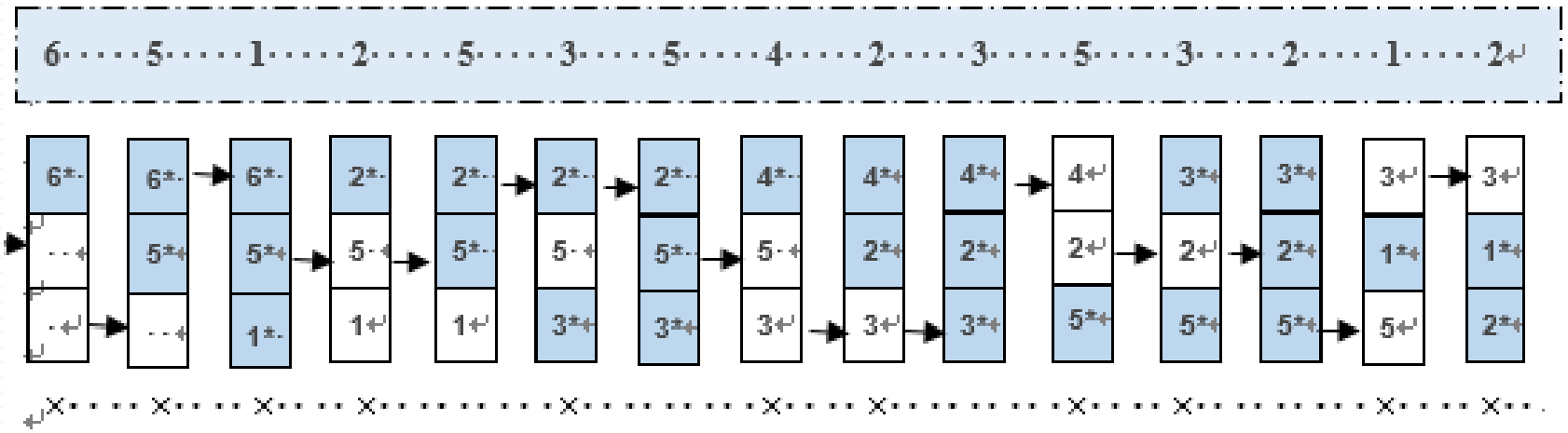
页面置换算法（续）

4. 近似LRU页面置换-二次机会算法

当选择置换页面时，依然和FIFO一样，选择最早置入内存的页面。但是二次机会法还设置了一个访问状态位。所以还要检查页面的的访问位。如果是0，就淘汰这页；如果访问位是1，就给它第二次机会，并选择下一个FIFO页面

场景：假设某进程被分配了3个页帧，程序页面访问序列为：

6, 5, 1, 2, 5, 3, 5, 4, 2, 3, 5, 3, 2, 1, 2



页面缓冲算法

①空闲页面链表

实际上该链表是一个空闲页帧链表，用于分配给频繁发生缺页的进程，以降低该进程的缺页率。当这样的进程需要读入一个页面时，便可利用空闲页帧链表中的第一个页帧来装入该页。当有一个未被修改的页要换出时，实际上并不将它换出到外存，而是把它们所在的页帧挂在空闲链表的末尾（页面在内存中并不做物理上的移动，而是将页表中的表项移到该链表表尾）。

②修改页面链表

它是由已修改的页面所形成的链表。设置该链表的目的是为了减少已修改页面换出的次数。当进程需要将一个已修改的页面换出时，系统并不立即把它换出到外存上，而是将它所在的物理块挂在修改页面链表的末尾。这样做的目的是降低将已修该页面写回磁盘的频率，降低将磁盘内容读入内存的频率。

页帧分配算法

页帧的分配算法通常需要考虑驻留在内存中进程的情况，常见的分配算法包括平均分配法与按比例分配法。

◆ 平均分配

即最大帧数除以进程总数得到一个平均的值，这个值就是操作系统为每个进程分配的页帧数目。单纯的平均分配没有考虑到不同进程可能需要不同大小的内存空间。

◆ 按进程大小比例分配

按照进程地址空间的大小，按比例给进程分配页帧数。设进程p_i的需地址空间的页面数为s_i，且定义：

$$S = \sum s_i$$

如果空闲页帧总数为m，那么进程p_i可分配到a_i个页帧，这里a_i近似为：

$a_i=(s_i/S)*m$，当然a_i为满足条件的整数。

页帧分配算法

考虑一下这样的系统，每个页帧大小为1K，目前内存一共有60个空闲页帧，现在有2个进程，一个进程的地址空间为100个页面，另一个有20个页面。

- **按照平均分配的方案**

  每个进程可以分配到30个页帧大小的内存空间

- **按进程大小比例分配**

  较大进程可以分配的页帧数目为：60* 100/(100+20) =50;

  较小的进程可分配的页帧数目为：10。

  这样两个进程可以在合理大小的内存空间中执行。

页帧分配策略

局部分配的策略

页面的置换都是发生在同一个进程已分配内存空间中。局部算法可以有效地为每个进程分配固定的页帧

全局分配的策略

全局分配策略允许进程抢占其他进程分配的页帧。全局算法在可运行进程之间动态地分配页框，因此分配给各个进程的页框数是随时间变化的。

常驻集

进程在运行时，当前时刻实际驻留在内存当中的页面集合。

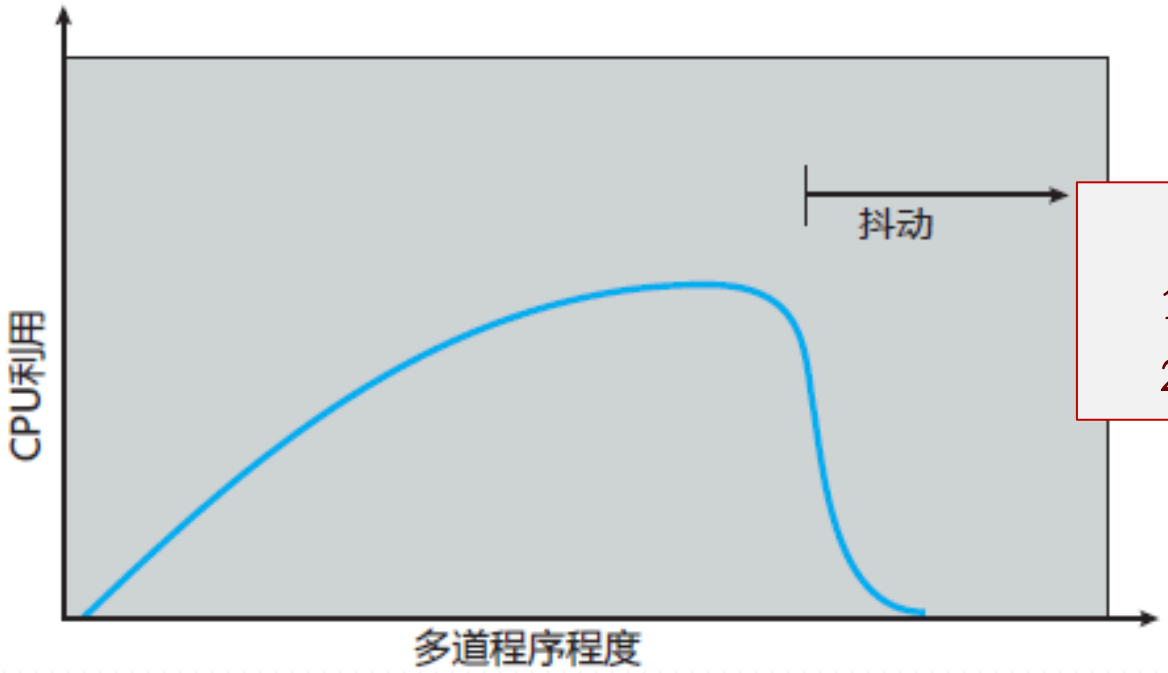
工作集

进程在过去一段时间访问的页面的集合

全局算法在通常情况下工作得比局部算法好，当工作集的大小随进程运行时间发生变化时这种现象更加明显。若使用局部算法，即使有大量的空闲页框存在，工作集的增长也会导致颠簸。如果工作集缩小了，局部算法又会浪费内存。在使用全局算法时，系统必须不停地确定应该给每个进程分配多少页框。

系统抖动

随着进程的增加，CPU的利用率也会增加，但是如果同一时间进程过多，每个进程占用的帧就相应变少了，就可能出现进程执行时需要经常性地发生缺页中断、CPU利用率又降低了的现象，而这时，操作系统还以为是进程数量太少导致的，还继续加入进程，导致每个进程占用的帧更少、CPU利用率更低的恶性循环，这种现象称为或系统抖动



系统抖动的原因：

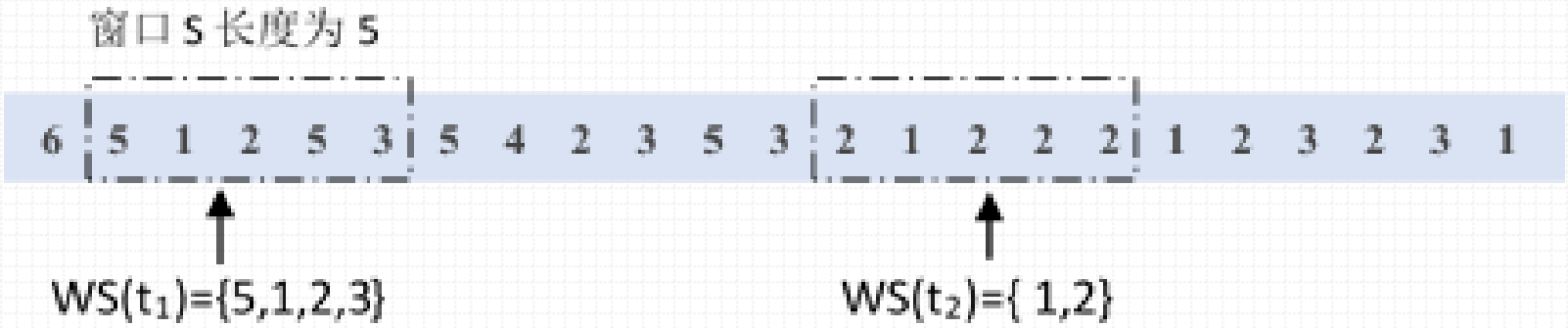
1）分配的页帧数量太小

2）置换算法选择不当

解决系统抖动的方法

1）工作集策略

工作集合策略是通过计算每一个进程的工作集近似得到进程需要的页帧数，如果这个总数大于内存的物理帧数，则说明系统颠簸了，需要减少进程。



解决系统抖动的方法

2）缺页率策略

缺页频率是另一种更为直接的防止抖动的方法。因此系统如果随时能够检测到系统中的缺页错误的情况，就可以动态地调整为进程分配的页帧数目。

我们可以设置所需缺页率的上下限。如果实际缺页率超过上限，则可为进程再分配更多的页帧；如果实际缺页率低于下限，则可从进程中移走页帧。因此，可以靠直接监测缺页率来防止抖动。

实际上，抖动及其导致的交换对性能的负面影响很大。目前处理这一问题的最佳实践是，在可能的情况下提供足够物理内存以避免抖动和交换。

10、有一个操作系统采用段式存储管理方案，用户区内存为512K，分配时截取空闲块的前半部分(小地址部分)。初始时内存全部空闲。系统执行如下申请、释放操作序列。

申请300K，申请100K，释放300K，申请150K，申请50K，申请90K

(1) 若采用首先适应算法，空闲块表中有哪些空块(指出大小，地址)；

答：操作系统采用段式存储。执行申请释放序列后，结果如下：

a、如果采用首先适应算法，空闲块表中的空块有

地址	大小
290k	10k
400k	112k

b、如果采用最佳适应算法，空闲块表中的空块有

地址	大小
240k	60k
450k	62k

c、若继续申请80k

如果之前采用首先适应算法，则直接分配起始地址为400k的连续80k空间

如果之前采用最佳适应算法，则需要首先采用拼接技术对空闲空间进行合并，然后在合并后的空闲空间中分配连续80k空间。

在上述情况中采用最佳适应算法却导致后来的内存直接分配失败而不得不进行内存空间整理。这说明最佳适应算法并不是所有时候都能够保持大块连续的空闲空间。

思考题2

(16分)请求分页管理系统中，假设某进程的页表内容见表：

页号	页框（Page Frame）号	有效位
0	101H	1
1		0
2	254H	1

页面大小为4KB，一次内存访问的访问时间为100ns，一次快表（TLB）的访问时间为10ns，处理一次缺页的平均时间为**800ns**（已包含更新TLB和页表的时间），进程的驻留集大小固定为2,采用最近最少使用置换算法（LRU）和局部淘汰策略。有如下假设：

- TLB初始为空；
- 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）；
- 有效位为0时表示页面不在内存中，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。设有虚地址访问序列2362H、1565H、25A5H，请问：

1）依次访问上述三个虚地址，各需要多少时间？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（使用16进制或二进制表示物理地址）？请说明理由。

页号	页框（Page Frame）号	有效位
0	101H	1
1		0
2	254H	1

答案：

1）访问虚地址2362H：**0010，001101101100** 其中低**12**为页内位移量；**0010**为虚页号，**2**号虚页在内存中，访问时间为：**2*100+10=210ns**

访问虚地址：1565H：虚页号为**1**，该页不在内存，需要发生缺页中断，故访问时间为： **200+10+800=1010ns**

访问虚地址：25A5H:虚页号为**2**，该页在内存，而且快表更新，故访问时间为：

100+10=110ns

2）虚地址**1565H**：虚页号为**1**，页内位移量为**565H**

目前**1**号虚页不在内存，需要进行页面置换，采用**LRU**置换策略，选择**0**号虚页置换，于是虚地址**1565H**的物理地址为：**101565H**

9、对一个将页表放在分页系统的内存中：

（1）如果访问内存需要0.2μs，有效访问时间为多少？

（2）如果增加一个快表，且假定在快表中找到页表项的几率高达90%，则有效访问时间又是多少（假定查找快表需花的时间为0）？

分页系统要访问两次，第一次要访问页表，将页号换成页地址，并与偏移量相加，得出实际地址，第二次要访问实际的地址的，所以所用时间是0.4μs，如果有快表，命中率为90%，则访问时间为
0.2*90%+0.4*10%=0.18+0.04=0.22μs
由于CPU以及快表本身耗用的时间没有给出，所以假定这些时间可以忽略不计

1、有一个虚拟存储系统。分配给某进程3页内存，开始时内存为空，页面访问序列如下：

6，5，4，3，2，1，5，4，3，6，5，4，3，2，1，6，5

（1）若采用先进先出页面置换算法(FIFO)，缺页次数为多少？

（2）若采用最近最少使用页面置换算法(LRU)，缺页次数为多少？

（3）若采用最佳页面置换算法算法呢？

答:

(1)：17次

(2)：17次

(3)：11次

2、假设当前在处理机上执行的进程的页表如下，所有数字都是十进制，页的大小为1024B。对于给定的以下虚拟地址，其物理地址是多少？

a) 1052
b) 2221
c) 5499

页号	存储块号
0	4
1	7
2	—
3	2
4	—
5	0

1052：

int(1052/1024)=1，1052%1024=28，

页号为1，查页表得到存储块号为7。

7*1024+28=7196

2221：

int(2221/1024)=1，2221%1024=173，

页号为2，查页表知该页没有调入内存，产生缺页中断。、

5499：

int(5499/1024)=5，5499%1024=379，

页号为5，查页表得到存储块号为0。

0*1024+379=379

作业

3、一个进程分配有4个页面，如下表（下面的数字均为十进制，每项数据都是从0开始计数的）。访问页号为4的页，发生缺页时，分别采用下列页面置换算法时，将置换哪一页，并解释原因。

（1）OPT（最佳）置换算法

（2）FIFO（先进先出）置换算法

（3）LRU（最近最少使用）算法

页号	存储块号	加载时间	访问时间	访问位	修改位
0	2	26	162	1	0
1	1	130	160	0	0
2	0	60	161	0	1
3	3	20	163	1	1

解答：

- OPT置换算法：题目给出条件不足，不能推断出置换哪一页
- FIFO置换算法：置换第3页，因为它的加载时间最早。
- LRU算法：置换第1页，因为它的访问时间最早。

本章结束，谢谢！