虚拟内存管理

3.2.1 虚拟内存的基本概念

1. 传统存储管理方式的特征

3.1 节讨论的各种内存管理策略都是为了同时将多个进程保存在内存中，以便允许进行多道程序设计。它们都具有以下两个共同的特征：

1）一次性。作业必须一次性全部装入内存后，才能开始运行。这会导致两种情况：①当作业很大而不能全部被装入内存时，将使该作业无法运行；②当大量作业要求运行时，由于内存不足以容纳所有作业，只能使少数作业先运行，导致多道程序度的下降。

2）驻留性。作业被装入内存后，就一直驻留在内存中，其任何部分都不会被换出，直至作业运行结束。运行中的进程会因等待 I/O 而被阻塞，可能处于长期等待状态。

由以上分析可知，许多在程序运行中不用或暂时不用的程序（数据）占据了大量的内存空间，而一些需要运行的作业又无法装入运行，显然浪费了宝贵的内存资源。

2．局部性原理

要真正理解虚拟内存技术的思想，首先须了解著名的局部性原理。从广义上讲，快表、页高速缓存及虚拟内存技术都属于高速缓存技术，这个技术所依赖的原理就是局部性原理。局部性原理既适用于程序结构，又适用于数据结构（更远地讲，Dijkstra 关于“goto 语句有害”的著名论文也出于对程序局部性原理的深刻认识和理解）。

局部性原理表现在以下两个方面：

1）时间局部性。程序中的某条指令一旦执行，不久后该指令可能再次执行；某数据被访问过，不久后该数据可能再次被访问。产生的原因是程序中存在着大量的循环操作。

2）空间局部性。一旦程序访问了某个存储单元，在不久后，其附近的存储单元也将被访问，即程序在一段时间内所访问的地址，可能集中在一定的范围之内，因为指令通常是顺序存放、顺序执行的，数据也一般是以向量、数组、表等形式簇聚存储的。

时间局部性通过将近来使用的指令和数据保存到高速缓存中，并使用高速缓存的层次结构实现。空间局部性通常使用较大的高速缓存，并将预取机制集成到高速缓存控制逻辑中实现。虚拟内存技术实际上建立了“内存-外存”的两级存储器结构，利用局部性原理实现高速缓存。

3．虚拟存储器的定义和特征

基于局部性原理，在程序装入时，仅须将程序当前要运行的少数页面或段先装入内存，而将其余部分暂留在外存，便可启动程序执行。在程序执行过程中，当所访问的信息不在内存时，由操作系统将所需要的部分调入内存，然后继续执行程序。另一方面，操作系统将内存中暂时不使用的内容换出到外存上，从而腾出空间存放将要调入内存的信息。这样，系统好像为用户提供了一个比实际内存容量大得多的存储器，称为虚拟存储器。

之所以将其称为虚拟存储器，是因为这种存储器实际上并不存在，只是由于系统提供了部分装入、请求调入和置换功能后（对用户透明），给用户的感觉是好像存在一个比实际物理内存大得多的存储器。但容量大只是一种错觉，是虚的。虚拟存储器有以下三个主要特征：

1）多次性。是指无须在作业运行时一次性地全部装入内存，而允许被分成多次调入内存运行，即只需将当前要运行的那部分程序和数据装入内存即可开始运行。以后每当要运行到尚未调入的那部分程序时，再将它调入。多次性是虚拟存储器最重要的特征。

2）对换性。是指无须在作业运行时一直常驻内存，在进程运行期间，允许将那些暂不使用的程序和数据从内存调至外存的对换区（换出），待以后需要时再将它们从外存调至内存（换进）。正是由于对换性，才使得虚拟存储器得以正常运行。

3）虚拟性。是指从逻辑上扩充内存的容量，使用户所看到的内存容量远大于实际的内存容量。这是虚拟存储器所表现出的最重要特征，也是实现虚拟存储器的最重要目标。

3.2.2 请求分页管理方式

请求分页系统建立在基本分页系统基础之上，为了支持虚拟存储器功能而增加了请求调页功能和页面置换功能。请求分页是目前最常用的一种实现虚拟存储器的方法。

在请求分页系统中，只要求将当前需要的一部分页面装入内存，便可以启动作业运行。在作业执行过程中，当所要访问的页面不在内存中时，再通过调页功能将其调入，同时还可通过置换功能将暂时不用的页面换出到外存上，以便腾出内存空间。请求分页的基本概念/思想

为了实现请求分页，系统必须提供一定的硬件支持。除了需要一定容量的内存及外存的计算机系统，还需要有页表机制、缺页中断机构和地址变换机构。请求分页所需的硬件支持

页表机制

1．驻留集大小

对于分页式的虚拟内存，在进程准备执行时，不需要也不可能把一个进程的所有页都读入主存。因此，操作系统必须决定读取多少页，即决定给特定的进程分配几个页框。给一个进程分配的物理页框的集合就是这个进程的驻留集。需要考虑以下几点：

1）分配给一个进程的页框越少，驻留在主存中的进程就越多，从而可提高 CPU 的利用率。

2）若一个进程在主存中的页面过少，则尽管有局部性原理，缺页率仍相对较高。

3）若分配的页框过多，则由于局部性原理，对该进程的缺页率没有太明显的影响。

页号	物理块号	状态位 P	访问字段 A	修改位 M	外存地址

图 3.20　请求分页系统中的页表项

增加的 4 个字段说明如下：

- 状态位 P。用于指示该页是否已调入内存，供程序访问时参考。
- 访问字段 A。用于记录本页在一段时间内被访问的次数，或记录本页最近已有多长时间未被访问，供置换算法换出页面时参考。
- 修改位 M。标识该页在调入内存后是否被修改过，以确定页面置换时是否写回外存。
- 外存地址。用于指出该页在外存上的地址，通常是物理块号，供调入该页时参考。

2．缺页中断机构

在请求分页系统中，每当所要访问的页面不在内存中时，便产生一个缺页中断，请求操作系统将所缺的页调入内存。此时应将缺页的进程阻塞（调页完成唤醒），若内存中有空闲块，则分配一个块，将要调入的页装入该块，并修改页表中的相应页表项，若此时内存中没有空闲块，则要淘汰某页（若被淘汰页在内存期间被修改过，则要将其写回外存）。

缺页中断作为中断，同样要经历诸如保护 CPU 环境、分析中断原因、转入缺页中断处理程序、恢复 CPU 环境等几个步骤。但与一般的中断相比，它有以下两个明显的区别：

- 在指令执行期间而非一条指令执行完后产生和处理中断信号，属于内部异常。
- 一条指令在执行期间，可能产生多次缺页中断。

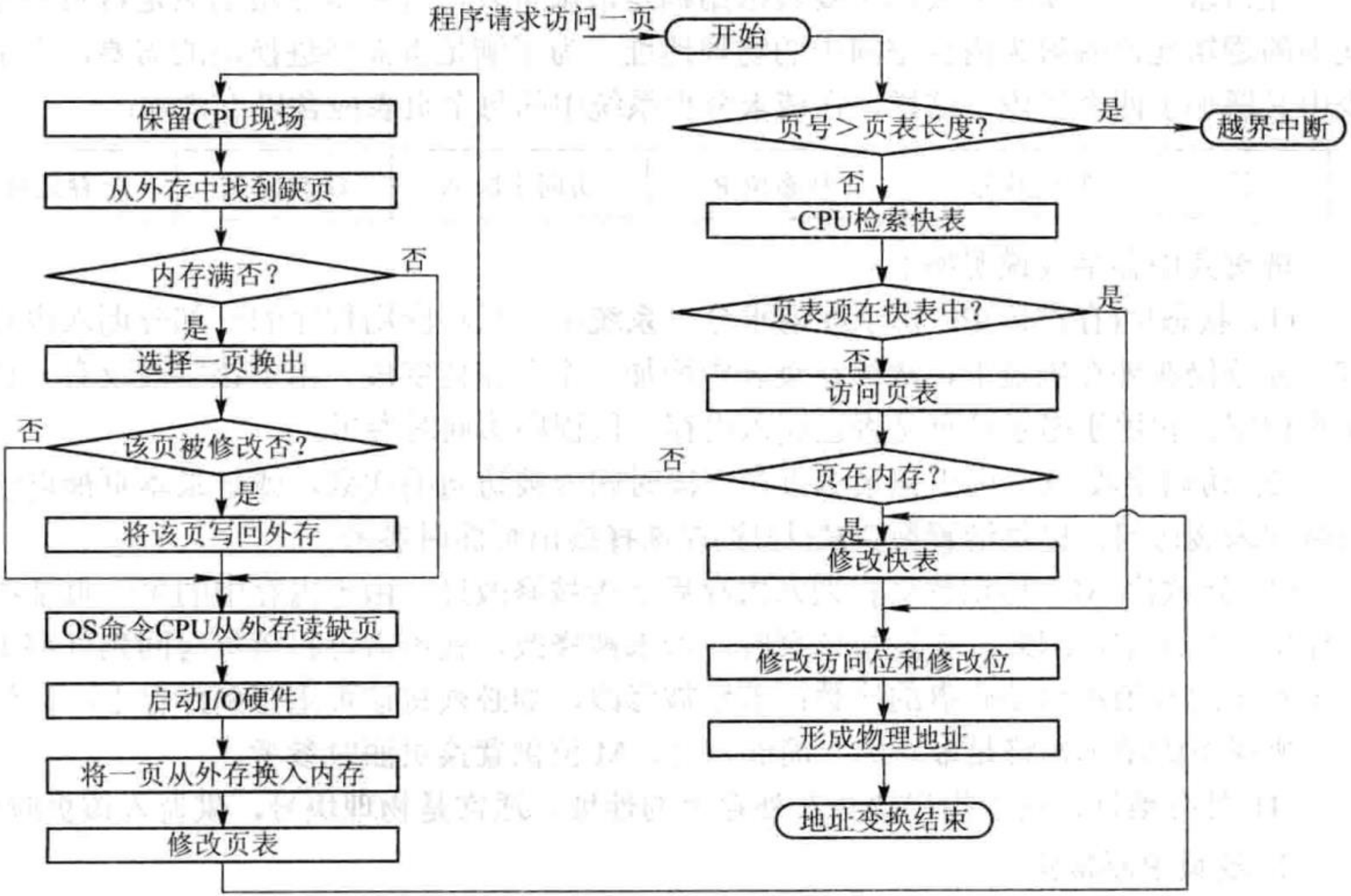
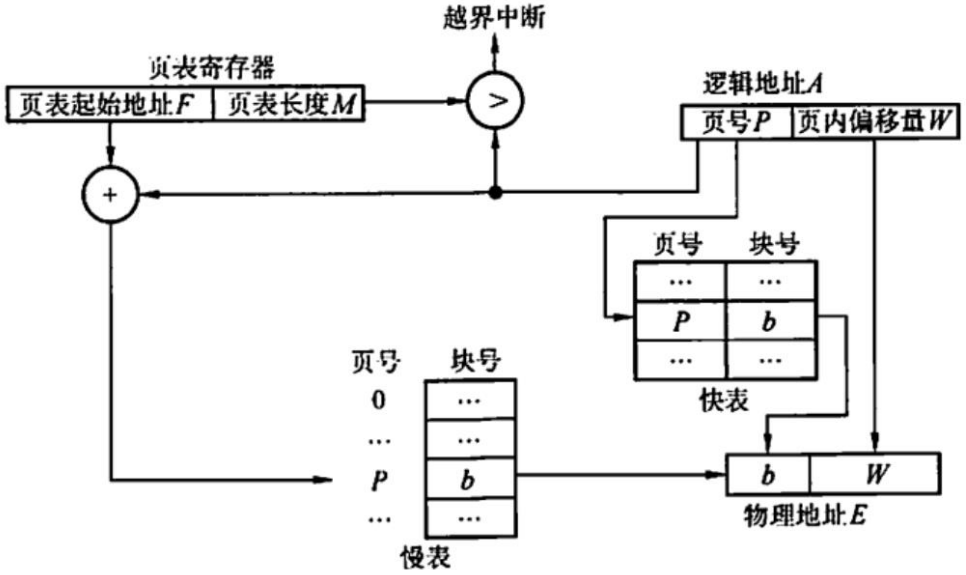


图 5-2　请求分页中的地址变换过程

见图示

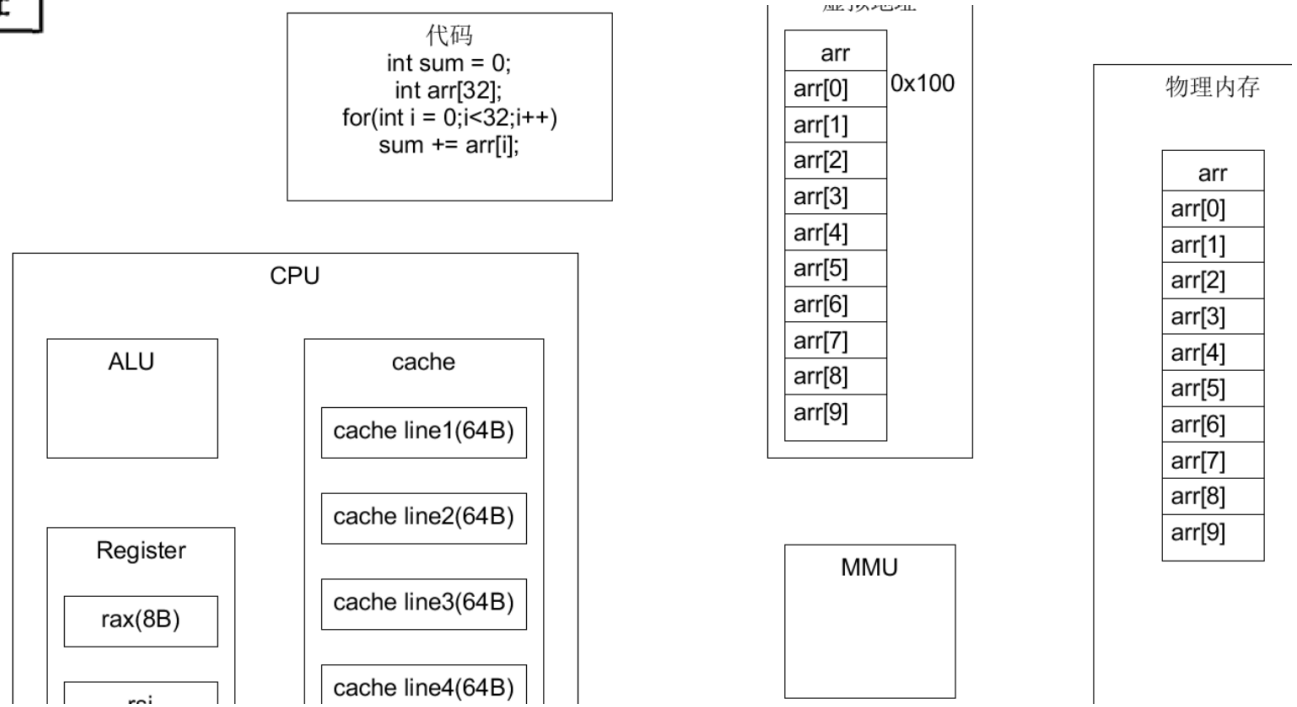
程序局部性原理的应用

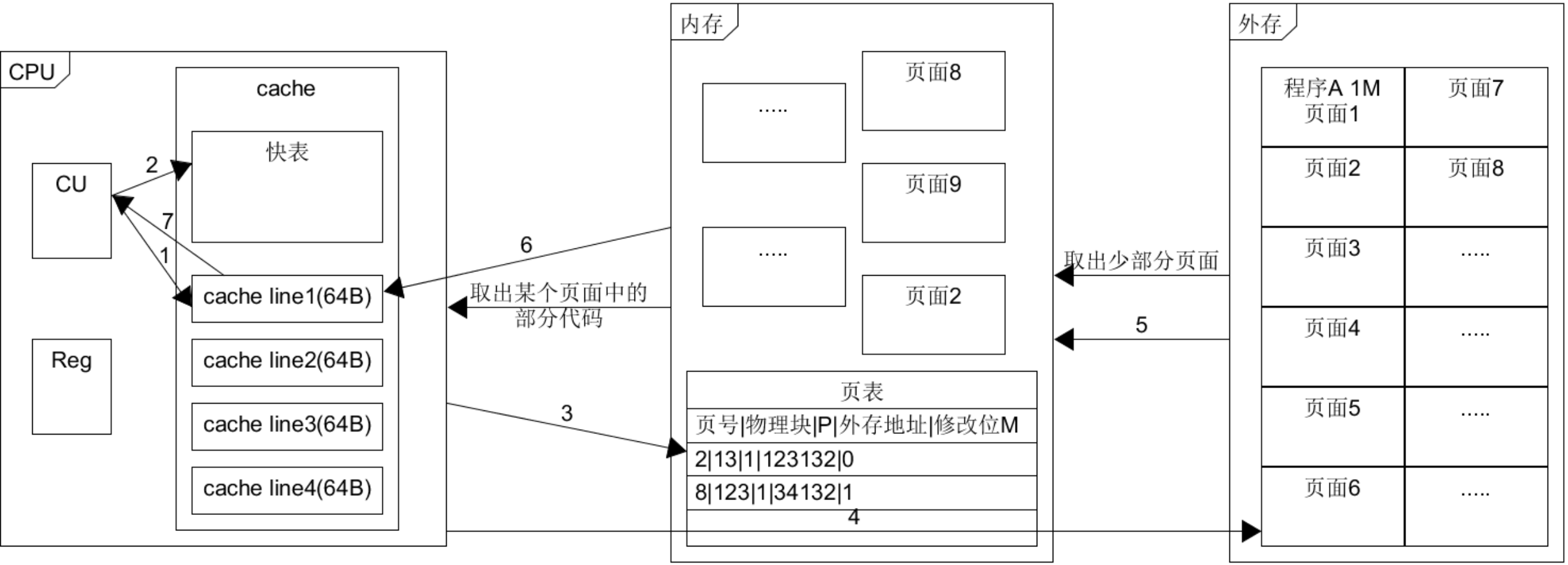
- 高速缓存技术
- 虚拟内存技术



页号	物理块号	状态位 P	访问字段 A	修改位 M	外存地址

图 3.20　请求分页系统中的页表项




CPU
cache
快表
CU
Reg
2
7
1
cache line1(64B)
cache line2(64B)
cache line3(64B)
cache line4(64B)
内存
.....
页面8
页面9
.....
页面2
6
取出某个页面中的
部分代码
3
页表
页号|物理块|P|外存地址|修改位M
2|13|1|123132|0
8|123|1|34132|1
4
取出少部分页面
5
外存
程序A 1M
页面1
页面7
页面2
页面8
页面3
.....
页面4
.....
页面5
.....
页面6
.....

请求分页内存分配策略

2．内存分配策略

在请求分页系统中，可采取两种内存分配策略，即固定和可变分配策略。在进行置换时，也可采取两种策略，即全局置换和局部置换。于是可组合出下面三种适用的策略。

（1）固定分配局部置换

为每个进程分配一定数目的物理块，在进程运行期间都不改变。所谓局部置换，是指如果进程在运行中发生缺页，则只能从分配给该进程在内存的页面中选出一页换出，然后再调入一页，以保证分配给该进程的内存空间不变。实现这种策略时，难以确定应为每个进程分配的物理块数目：太少会频繁出现缺页中断，太多又会降低 CPU 和其他资源的利用率。

（2）可变分配全局置换

先为每个进程分配一定数目的物理块，在进程运行期间可根据情况适当地增加或减少。所谓全局置换，是指如果进程在运行中发生缺页，系统从空闲物理块队列中取出一块分配给该进程，并将所缺页调入。这种方法比固定分配局部置换更加灵活，可以动态增加进程的物理块，但也存在弊端，如它会盲目地给进程增加物理块，从而导致系统多道程序的并发能力下降。

（3）可变分配局部置换

页面置换算法

- OPT（往后看）

最佳置换算法选择的被淘汰页面是以后永不使用的页面，或是在最长时间内不再被访问的页

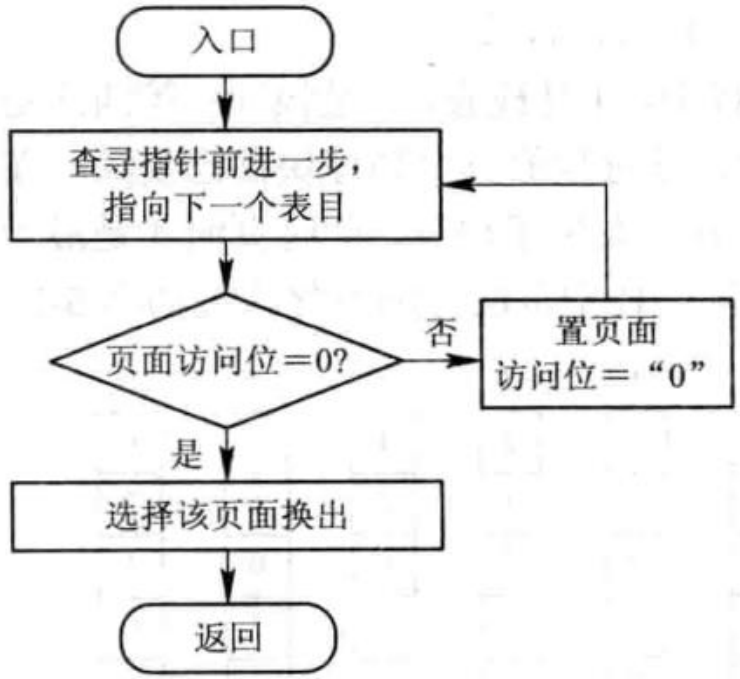
- FIFO
- LRU （往前看）（146. LRU 缓存 - 力扣（LeetCode））

选择最近最长时间未访问过的页面予以淘汰，它认为过去一段时间内未访问过的页面，

- CLOCK(见课本)
- LFU
- 页面缓冲算法PBA

1. 简单的 Clock 置换算法

当利用简单 Clock 算法时，只需为每页设置一位访问位，再将内存中的所有页面都通过链接指针链接成一个循环队列。当某页被访问时，其访问位被置 1。置换算法在选择一页淘汰时，只需检查页的访问位。如果是 0，就选择该页换出；若为 1，则重新将它置 0，暂不换出，给予该页第二次驻留内存的机会，再按照 FIFO 算法检查下一个页面。当检查到队列中的最后一个页面时，若其访问位仍为 1，则再返回到队首去检查第一个页面。图 5-8 示出了该算法的流程和示例。由于该算法是循环地检查各页面的使用情况，故称为 Clock 算法。但因该算法只有一位访问位，只能用它表示该页是否已经使用过，而置换时是将未使用过的页面换出去，故又把该算法称为最近未用算法或 NRU(Not Recently Used)算法。



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

图 5-8　简单 Clock 置换算法的流程和示例

（2）改进型 CLOCK 置换算法

将一个页面换出时，若该页已被修改过，则要将该页写回磁盘，若该页未被修改过，则不必将它写回磁盘。可见，对于修改过的页面，替换代价更大。在改进型 CLOCK 算法中，除考虑页面使用情况外，还增加了置换代价——修改位。在选择页面换出时，优先考虑既未使用过又未修改过的页面。由访问位 A 和修改位 M 可以组合成下面四种类型的页面：

- 1 类 $A=0, M=0$：最近未被访问且未被修改，是最佳淘汰页。
- 2 类 $A=0, M=1$：最近未被访问，但已被修改，不是很好的淘汰页。
- 3 类 $A=1, M=0$：最近已被访问，但未被修改，可能再被访问。
- 4 类 $A=1, M=1$：最近已被访问且已被修改，可能再被访问。

内存中的每页必定都是这四类页面之一。在进行页面置换时，可采用与简单 CLOCK 算法类似的算法，差别在于该算法要同时检查访问位和修改位。算法执行过程如下：

1）从指针的当前位置开始，扫描循环队列，寻找 $A=0$ 且 $M=0$ 的 1 类页面，将遇到的第一个 1 类页面作为选中的淘汰页。在第一次扫描期间不改变访问位 A。

2）若第 1）步失败，则进行第二轮扫描，寻找 $A=0$ 且 $M=1$ 的 2 类页面。将遇到的第一个 2 类页面作为淘汰页。在第二轮扫描期间，将所有扫描过的页面的访问位都置为 0。

3）若第 2）步也失败，则将指针返回到开始的位置，并将所有帧的访问位置为 0。重复第 1）步，并且若有必要，重复第 2）步，此时一定能找到被淘汰的页。

改进型 CLOCK 算法优于简单 CLOCK 算法的地方在于，可减少磁盘的 I/O 操作次数。但为了找到一个可置换的页，可能要经过几轮扫描，即实现算法本身的开销将有所增加。

操作系统中的页面置换算法都有一个原则，即尽可能保留访问过的页面，而淘汰未访问的页面。简单的 CLOCK 算法只考虑页面是否被访问过；改进型 CLOCK 算法对这两类页面做了细分，分为修改过的页面和未修改的页面。因此，若有未使用过的页面，则当然优先将其中未修改过的页面换出。若全部页面都用过，还是优先将其中未修改过的页面换出。

1. 请求段表机制

在请求分段式管理中所需的主要数据结构是请求段表。在该表中除了具有请求分页机制中有的访问字段 A、修改位 M、存在位 P 和外存始址四个字段外，还增加了存取方式字段和增补位。这些字段供程序在调进、调出时参考。下面给出请求分段的段表项。

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

在段表项中，除了段名(号)、段长、段在内存中的起始地址(段基址)外，还增加了以下字段：

(1) 存取方式。由于应用程序中的段是信息的逻辑单位，可根据该信息的属性对它实施保护，故在段表中增加存取方式字段，如果该字段为两位，则存取属性是只执行、只读和允许读/写。

(2) 访问字段 A。其含义与请求分页的相应字段相同，用于记录该段被访问的频繁程度。提供给置换算法选择换出页面时参考。

(3) 修改位 M。该字段用于表示该页在进入内存后是否已被修改过，供置换页面时参考。

(4) 存在位 P。该字段用于指示本段是否已调入内存，供程序访问时参考。

(5) 增补位。这是请求分段式管理中所特有的字段，用于表示本段在运行过程中是否做过动态增长。

(6) 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

在请求分段系统中采用的是请求调段策略。每当发现运行进程所要访问的段尚未调入内存时，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。与缺页中断机构类似，缺段中断机构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。但由于分段是信息的逻辑单位，因而不可能出现一条指令被分割在两个分段中，和一组信息被分割在两个分段中的情况。缺段中断的处理过程如图 5-12 所示。由于段不是定长的，这使对缺段中断的处理要比对缺页中断的处理复杂。

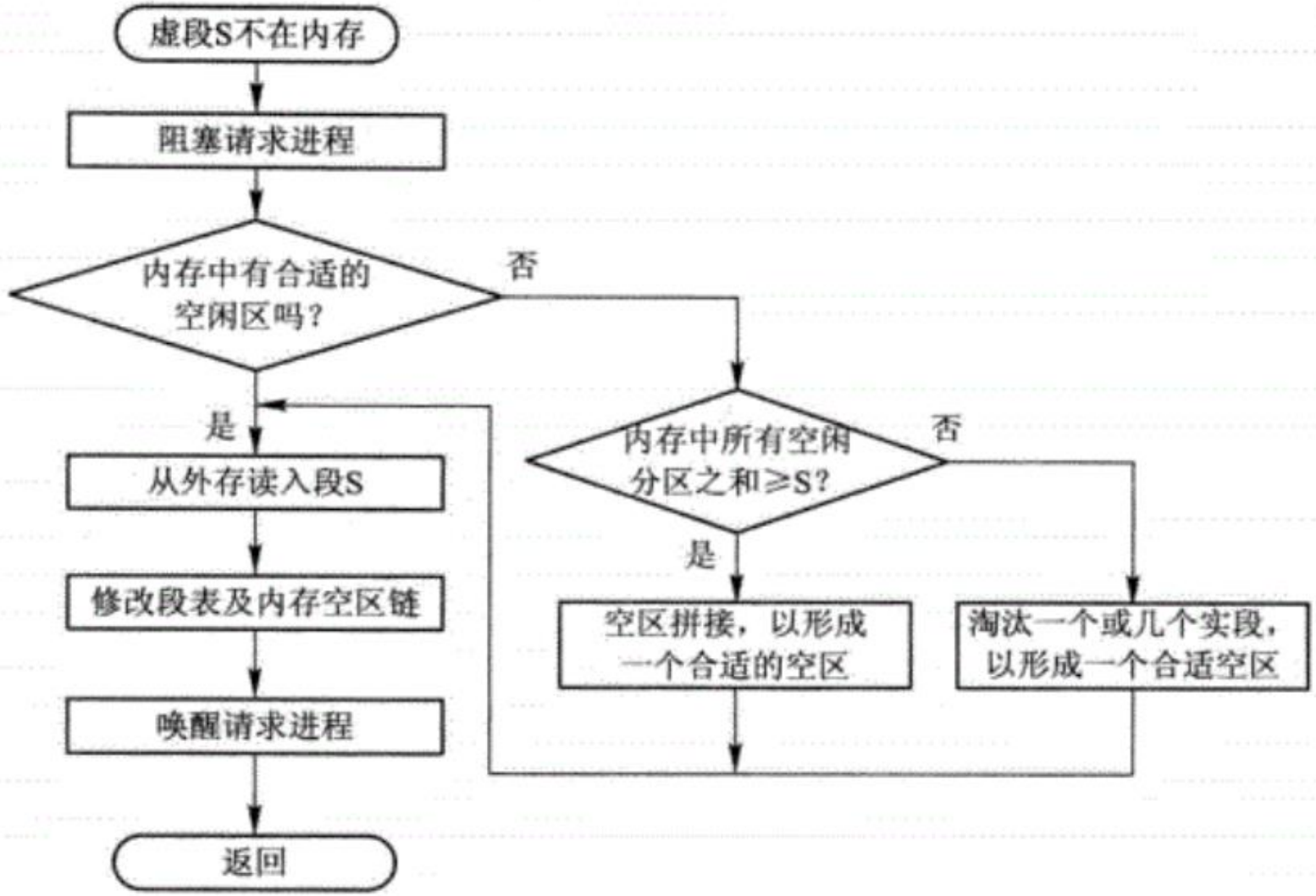


图 5-12　请求分段系统中的中断处理过程

3. 地址变换机构

请求分段系统中的地址变换机构是在分段系统地址变换机构的基础上形成的。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，在地址变换机构中又增加了某些功能，如缺段中断的请求及处理等。图 5-13 示出了请求分段系统的地址变换过程。

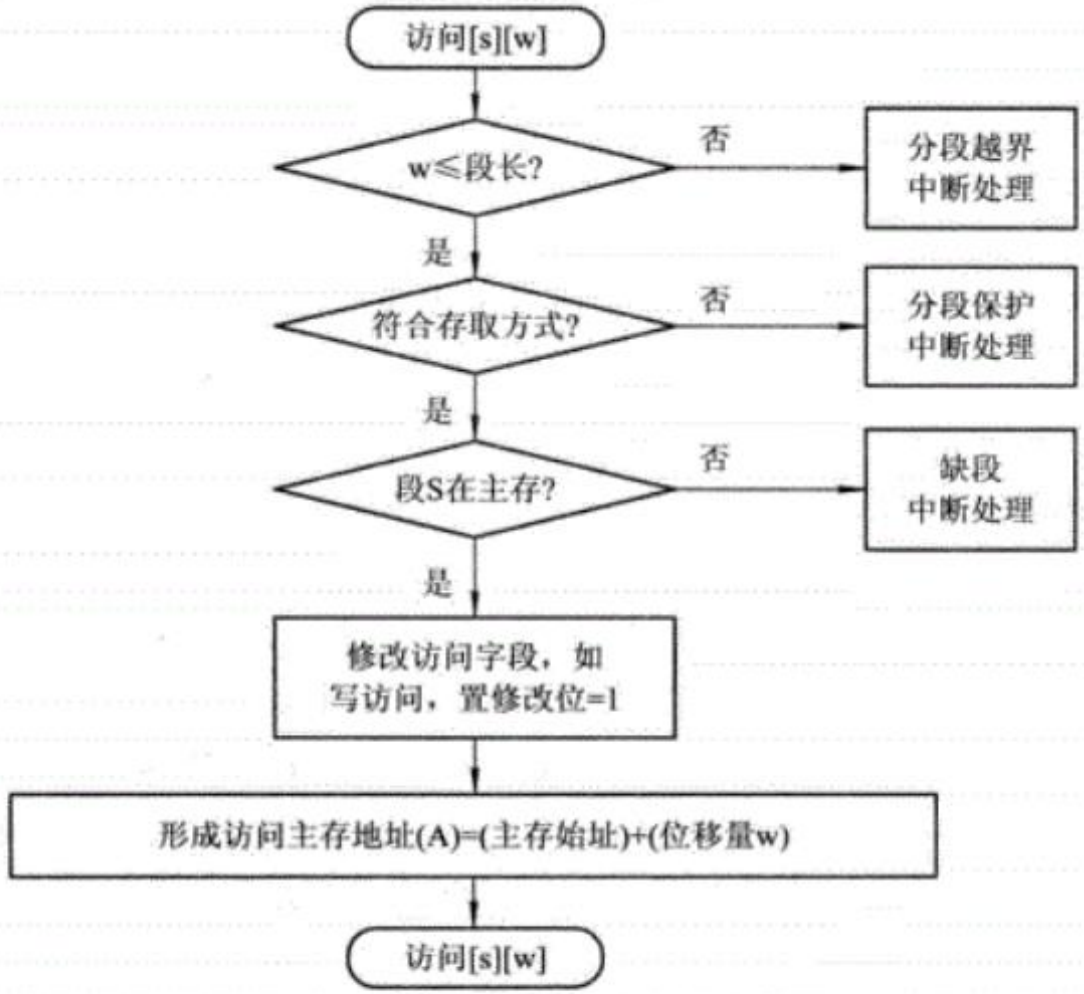


图 5-13　请求分段系统的地址变换过程

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

在段表项中，除了段名(号)、段长、段在内存中的起始地址(段基址)外，还增加了以下

3. 解释请求分页系统中的页表项的每个字段

页号	物理块号	状态位 P	访问字段 A	修改位 M	外存地址

真题

- 2012.07 时钟页面置换算法的基本思想与特点
- 2012.08 请求分页存储管理技术的基本思想与应用
- 2013.03 请求分页存储管理的基本概念和分配策略
- 2014.10 画LRU页面置换算法的置换图
- 2017.10 页式虚拟存储管理计算
- 2018.17 请求页式和静态页式有什么区别
- 2019.10 缺页中断次数计算
- 2019.19 页面置换算法计算
- 2020.03 解释请求分页页表项的字段

练习

- 1.什么是程序运行时的时间局部性和空间局部性?
- 2.虚拟存储器有哪些特征？其中最本质的特征是什么?
- 3.在请求分页系统中，页表应包括哪些数据项？每项的作用是什么?
- 4.试说明请求分页系统中的地址变换过程。
- 5.何谓固定分配局部置换和可变分配全局置换的内存分配策略?
- 6.在请求分页系统中，常采用哪几种页面置换算法?
- 7.试说明改进型Clock置换算法的基本原理。
- 8.什么是抖动？产生抖动的原因是什么?
- 9.在请求段表机制中，应设置哪些段表项?

- 1.时间局限性：如果程序中的某条指令被执行，则不久之后该指令可能再次执行；如果某数据被访问过，则不久以后该数据可能再次被访问。产生时间局限性的典型原因是在程序中存在着大量的循环操作。空间局限性：一旦程序访问了某个存储单元，在不久之后，其附近的存储单元也将被访问，即程序在一段时间内所访问的地址可能集中在一定的范围之内，其典型情况便是程序的顺序执行。
- 2.多次性、对换性、虚拟性。虚拟性
- 3. （1）状态位（存在位）P：它用于指示该页是否已调入内存，供程序访问时参考。（2）访问字段A：用于记录本页在一段时间内被访问的次数，或记录本页最近已有多长时间未被访问，提供给置换算法（程序）在选择换出页面时参考。（3）修改位M：标识该页再调入内存后是否被修改过。（4）外存地址：用于指出该页在外存上的地址，通常是物理块号，供调入该页时参考。

- 4.在进行地址变换时，首先检索快表，试图从中找出所要访问的页。若找到，便修改页表项中的访问位A，供置换算法选换出页面时参考。对于写指令，还需将修改位M置成“1”，表示该页再调入内存后已被修改。否则，去内存中查询页表，如果发生缺页中断则通过OS的调页程序将页面调入内存。最后利用页表项中给出的物理块号和页内地址形成物理地址。
- 5.（1）固定分配局部置换。固定分配为每个进程分配一组固定数目的物理块，在进程运行期间不再改变。局部置换是指如果进程在运行中发现缺页，则只能从分配给该进程的n个页面中选出一页换出，然后再调入一页，以保证分配给该进程的内存空间不变。

（2）可变分配全局置换。可变分配是指先为每个进程分配一定数目的物理块，在进程运行期间，可根据情况做适当的增加或减少。全局置换是指如果进程在运行中发现缺页，则将OS所保留的空闲物理块取出一块分配给该进程，或者以所有进程的全部物理快为标的，选择一块换出，然后将所缺之页调入。

- 6.采用的页面置换算法有：最佳置换算法和先进先出置换算法，最近最久未使用（LRU）置换算法，Clock置换算法，最少使用置换算法，页面缓冲算法等。
- 7.因为修改过的页面在换出时付出的开销比未被修改过的页面大，在改进型Clock算法中，既考虑页面的使用情况，还要增加置换代价的因素；在选择页面作为淘汰页面时，把同时满足未使用过和未被修改过作为首选淘汰页面。
- 8.抖动就是指当内存中已无空闲空间而又发生缺页中断时，需要从内存中调出一页程序或数据送磁盘的对换区中，而接下来刚被换出的页很快被访问，需重新调入。如此频繁更换页面，使得系统把大部分时间用在了页面的调进换出上，而几乎不能完成任何有效的的工作，我们称这种现象为“抖动”。

产生抖动的原因是系统中 同时运行的进程太多，由此分配给每个进程的物理块太少，不能满足进程正常运行的基本要求，致使每个进程运行时频繁缺页。

- 9.段名、段长、段基址、存取方式、访问字段、修改位、存在位、增补位、外存始地址。