第三章 内存管理

目　录

CONTENTS

3

1.计算机系统的组成

- 计算机的存储系统主要包括内存储器和外存储器。
- 内存储器（Memory），是处理器能直接寻址的存储空间，由半导体器件制成，用来存放处理器执行时所需要的程序和数据，以及与硬盘等外部存储器交换的数据，程序和数据只有在内存中才能被处理器直接访问。
- 外存储器也叫辅助存储器，用来存放需要长期保存的数据，外存储器的管理属于文件系统的范畴。

3

- 内存储器分两部分：
- 一部分是系统区，用来存放操作系统以及一些标准子程序、例行程序等，这些是长驻内存的部分，系统区用户不能使用；
- 另一部分称为用户区，分配给用户使用，用来存放用户的程序和数据等。内存管理的主要工作就是对内存储器中的用户区进行管理。

3.1.1 计算机存储系统的结构

按照计算机的体系结构，计算机存储系统可以划分为3个层次，分别是：处理器寄存器和高速缓存、内存储器、外存储器，如图3.1 所示。越往上，存储介质的访问速度越快，价格也越高；越往下，存储介质的存储空间越大。

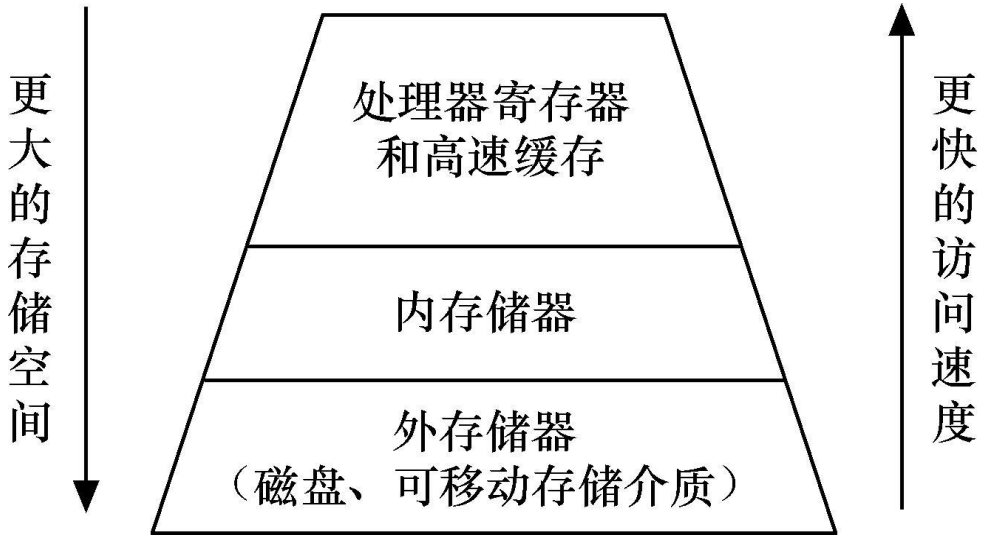


图3.1 计算机系统的结构与使用关系

3.1.1 计算机存储系统的结构

1．处理器寄存器和高速缓存

处理器寄存器主要包括通用寄存器、指令寄存器、地址寄存器和数据缓冲寄存器等一系列寄存器，用于存储理器中与控制流和数据流相关的信息。它容量小处，速度快，一般以字（word）为单位。一个计算机系统一般包括几十个甚至上百个寄存器。

3.1.1 计算机存储系统的结构

- 高速缓存（Cache）是为了解决处理器与内存之间速度不匹配而引入的。其存储容量比处理器寄存器大，访问速度比寄存器慢，但远比内存快。当处理器要读取数据时，首先访问高速缓存，如果所要访问的数据已经在高速缓存中，则直接从高速缓存中读取信息；如果要访问的数据不在高速缓存中，那就需要从内存中读取信息。随着硬件技术的发展，现在已经将高速缓存封装在处理器芯片中，所以常将高速缓存与处理器寄存器归到一个层次。

3.1.1 计算机存储系统的结构

2．内存储器

内存储器也称为内存，属于主机范畴。内存中存储处理器执行时所需要的代码和数据。内存的空间远大于高速缓存，但内存中的数据断电即消失，无法永久储存。一个计算机系统中所配备的内存容量是衡量计算机性能的一个重要的指标，计算机最大内存容量受到计算机系统结构的限制。

3.1.1 计算机存储系统的结构

3．外存储器

外存储器是计算机系统中最大规模的存储器，用来存储各种数据和软件。外存储器容量巨大并能够永久存储信息，断电后数据不会丢失，外存储器的价格低但是访问速度慢。外存储器包括各种磁盘、磁带、光盘以及其他移动存储设备。磁盘中的硬盘是计算机系统中大量联机信息的保存者，硬盘常常作为内存的补充，用来实现虚拟存储系统。

3.1.1 计算机存储系统的结构

例如，某台计算机的存储系统可以按层次配置如下：CPU中的寄存器 100个字，存取周期10ns；高速缓存2MB，存取周期 15ns；主存储器2GB，存取周期60ns；磁盘容量500GB，存取周期毫秒级。这台计算机足够胜任日常工作，其多层次的存储体系有很高的性能/价格比。

3.1.2 地址的表示与地址转换

1．逻辑地址空间

高级语言编写的源程序通过编译或汇编后得到目标程序。目标程序使用的地址称为逻辑地址，也叫相对地址，一个用户作业的目标程序的逻辑地址集合称为该作业的逻辑地址空间。作业的逻辑地址空间可以是一维的，这时逻辑地址限制在从0开始顺序排列的地址空间内；也可以是二维的，这时整个用户作业被分为若干段，每段有不同的段号，段内地址从0开始。

3.1.2 地址的表示与地址转换

2．物理地址空间

当程序运行时，它将被装入内存中的某段空间内，此时程序和数据的实际地址不可能同原来的逻辑地址一致，程序在物理内存中的实际存储单元称为物理地址，也叫绝对地址，物理地址的总体就构成了用户程序实际运行的物理地址空间。不同程序的物理地址空间绝对不能冲突。

在以下章节中，如无特别说明，地址编址和地址长度的单位都是字节（Byte）。

3.1.2 地址的表示与地址转换

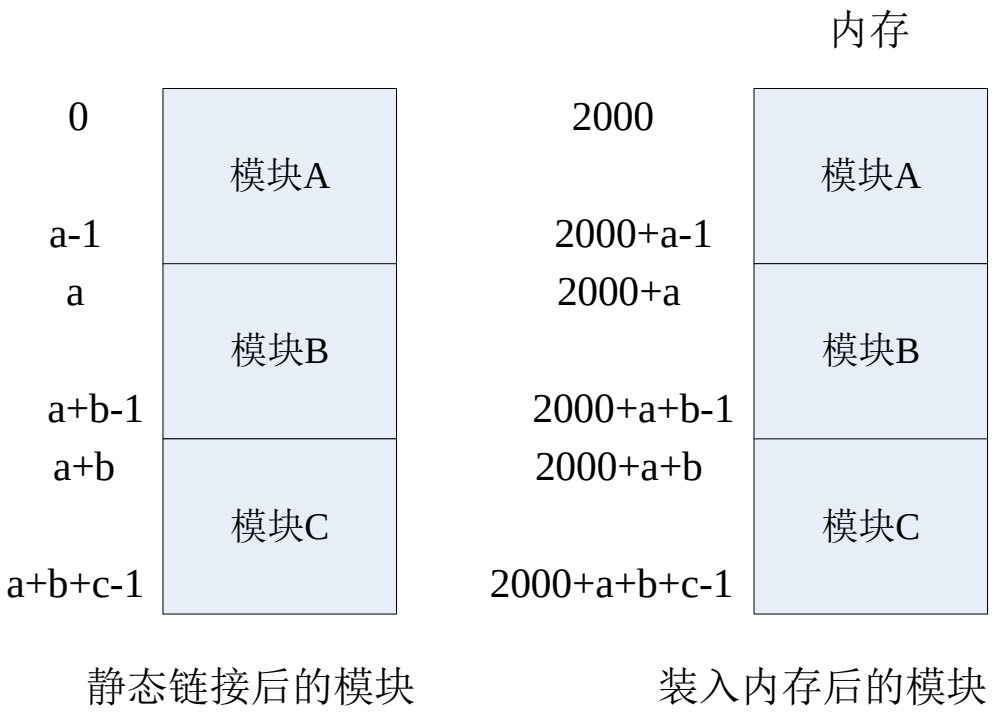
3. 地址转换

只有把程序和数据的逻辑地址转换为物理地址，程序才能正确运行，该过程称为地址转换或地址重定位。地址转换有静态重定位和动态重定位两种方式。

静态重定位：这种方式是在用户作业装入内存时由装入程序(装配程序)实现从逻辑地址到物理地址的转换，地址转换在作业执行前一次完成。

3. 1. 2 地址的表示与地址转换

- 图中有3个待装入内存的目标模块，其逻辑地址从0开始，到（a +b + c −1）结束。在装入内存时，如果只能从内存地址2000开始容纳该作业，则在静态重定位装入方式下该程序装入内存时的物理地址为从2000开始，到（2000 + a + b + c−1）结束，如图3.2所示。



3.1.2 地址的表示与地址转换

- 静态重定位方式的优点是实现简单，从逻辑地址到物理地址变换不需要专门的硬件便能完成；缺点是必须为程序分配一段连续的存储空间，并且程序在执行过程中不能在内存中移动。

3.1.2 地址的表示与地址转换

动态重定位：

程序执行过程中，CPU在访问程序和数据之前才实现地址转换，称为动态重定位。动态重定位必须借助于硬件地址转换机构来实现，硬件系统中设置了一个定位寄存器，当操作系统为某程序分配了一块内存区域后，装入程序把程序装入到所分配的区域中，然后把该内存区域的起始地址置入定位寄存器中。在程序执行过程中需要进行地址转换时，只需将逻辑地址与定位寄存器中的值相加就可得到物理地址。这种地址转换方式是在指令过程中进行的，所以称动态重定位。

3.1.2 地址的表示与地址转换

- 采用动态重定位可实现程序在内存中的移动。在程序执行过程中，若把程序移到一块新的内存区域后，只要改变定位寄存器中的内容，该程序仍可正确执行，但采用静态定位时，程序执行过程中是不能移动的。
- 动态重定位的优点是内存的使用更加灵活，容易实现内存的动态扩充和共享；缺点是实现过程中需要附加硬件支持，内存的管理也更加复杂。

3. 1. 3 内存管理的功能

1. 内存的分配和回收：操作系统根据用户程序的请求，在内存中按照一定算法把找到一块空闲，将其分配给申请者；并负责把释放的内存空间收回，使之变为空闲区。

2. 提高内存的利用率：通过多道程序共享内存，提高内存资源的利用率。

3. 通过虚拟存储技术“扩充”内存容量：使用户程序在比实际内存容量大的情况下，也能在内存中运行。

4. 内存信息保护：保证各个用户程序或进程在各自规定的存储区域内操作，不破坏操作系统区的信息，并且互不干扰。

3.1.4 覆盖与交换技术

1．覆盖技术

一个程序通常由若干个功能上独立的程序段组成，在运行时，并不是所有的程序段都同时进入内存执行。这样，我们就可以按照程序自身的逻辑结构，让不同时执行的程序段先后共享同一块内存区域，这就是覆盖技术。

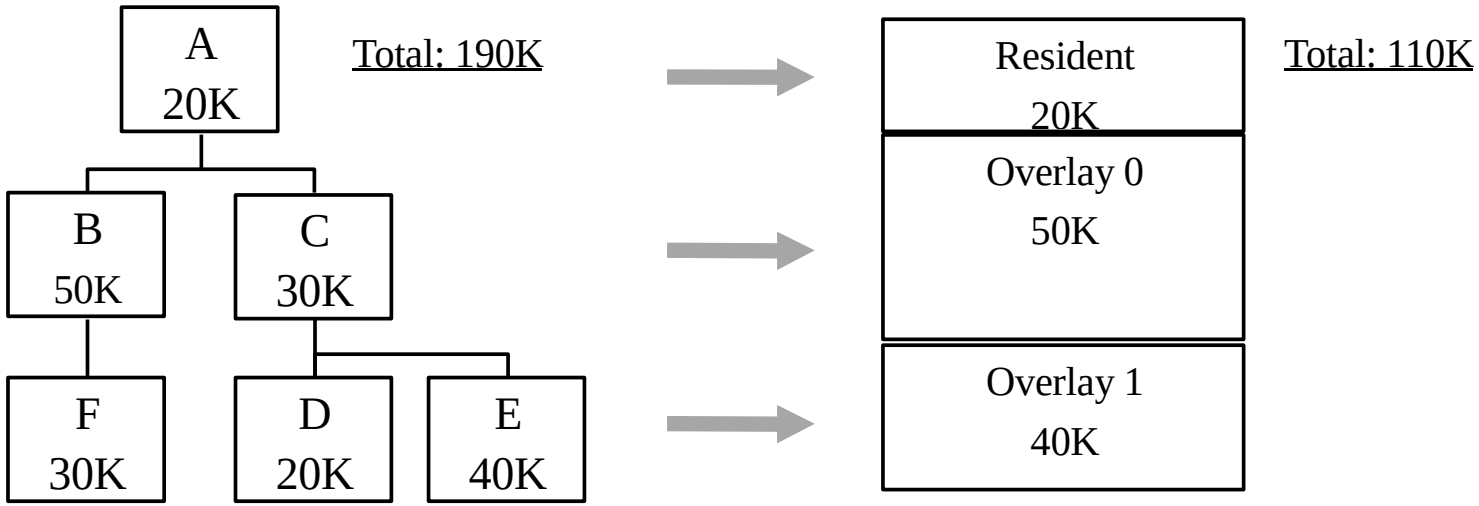
覆盖技术先将程序必需的部分代码和数据调入内存，其余部分先放在外存中，当要访问的程序或数据不在内存时，由操作系统负责将其从外存中调入，这就解决了在较小的内存空间中运行较大程序的问题。

3. 1. 4 覆盖与交换技术

覆盖技术首先将大的用户程序划分为一个个相对独立的程序单位，将程序执行时不需要同时装入内存的程序单位组成一个个覆盖段，每个覆盖段的长度不能超过已有内存空间大小。各个覆盖段分先后顺序进入到所分配的内存空间中，后进入内存的覆盖段将先进入的段覆盖。

3.1.4 覆盖与交换技术

例如：某程序由A、B、C、D、E、F等六个程序段组成，它们之间的调用关系如图3.3左图所示。其中，程序段A只调用B和C，程序段B只调用F，而程序段C只调用D和E。由于B和C之间没有相互调用，所以它们可以共享同一覆盖区。覆盖区的大小以能装入所有共享的程序段为准。本例中，与B、C对应的覆盖区的大小为50K。类似地，D、E、F也可以共享一大小为40K的覆盖区，如下图所示。



3. 1. 4 覆盖与交换技术

虽然该进程所需要的总的内存空间为20K+50K+30K+30K+20K+40K=190K，但是在采用了覆盖技术之后，只需要110K的内存就够了。

在程序执行全过程都要使用的部分不能对其进行覆盖，真正能够被覆盖的是分阶段使用的程序部分。目前这一技术仅应用于小型系统中的系统程序的内存管理上，例如，MS-DOS的启动过程中，就多次使用了覆盖技术。

3.1.4 覆盖与交换技术

覆盖技术的缺点：

1）覆盖技术对用户不透明，用户在编程时必须划分程序模块和确定程序模块之间的覆盖关系，增加了编程复杂度。

2）从外存装入覆盖文件，是以时间的延长来换取空间的节省。

3. 1. 4 覆盖与交换技术

为了释放部分内存空间，由操作系统根据需要，将某些暂时不运行的进程或程序段从内存移到外存的交换区中；当内存空间富余时再给被移出的进程或程序段重新分配内存，让其进入内存，这就是交换技术，又称为“对换”或“滚进/滚出(roll-in/roll-out)。

交换技术能够提高系统的性能和多道度，从内存到外存的交换为换出，从外存到内存的交换为换入。通过不断的换入、换出，使得用户看来好像内存扩大了，从而实现了内存扩充的目的。

3 3.1.4 覆盖与交换技术

根据每次交换的单位不同，交换技术在实现中有种情况：

- 1）以整个进程为单位的交换：每次换入或换出的是一个进程，此策略多用于早期的分时系统中，以实现在小型机上的分时运行。
- 2）以进程的一部分为单位的交换：在现代操作系统中，借助于页式或段式内存管理，先将进程的内存空间划分为若干页面或段，然后以页面或段为基本单位进行交换。在操作系统中，常见的有页面置换（在页式存储管理中介绍）和段置换（在段式存储管理中介绍），这也是虚拟存储技术的基础。

3.1.4 覆盖与交换技术

3. 覆盖技术和交换技术的比较

1） 与覆盖技术相比，交换技术不要求用户给出程序段之间的逻辑覆盖结构。

2） 交换发生在进程或作业之间，而覆盖发生在同一进程或作业内。

3） 覆盖只能覆盖那些与覆盖段无关的程序段。

3.2 分区内存管理

- 3.2.1 单一连续内存管理
- 3.2.2 固定分区内存管理
- 3.2.3 可变分区内存管理

3.2.1 单一连续内存管理

1. 基本思想

- 单一连续内存管理适用于单用户单任务操作系统，是最简单的内存管理方式。单一连续内存管理将内存空间分为系统区和用户区，系统区存放操作系统常驻内存的代码和数据，用户区全部分配给一个用户作业使用。在这种方式下，在任一时刻主存储器最多只有一道程序，各个作业只能按次序一个一个地装入主存储器运行。
- 通常，系统区位于内存底部的低地址部分，用户区位于内存顶部的高地址部分。

3.2.1 单一连续内存管理

2. 内存的分配与回收

- 由于内存中的用户区只能装入一个程序运行，所以该程序被装入内存时，就从内存用户区的基地址开始，连续存放。在运行过程中，该程序独占内存，直到退出，操作系统收回内存再分配给下一个程序使用。

3.2.1 单一连续内存管理

3. 地址转换与内存保护

- 单一连续内存管理多采用静态重定位来进行地址转换。操作系统设置一个界限寄存器用来设置内存中系统区和用户区的地址界限；通过装入程序把目标模块装入到从界限地址开始的区域，如图3.4所示。

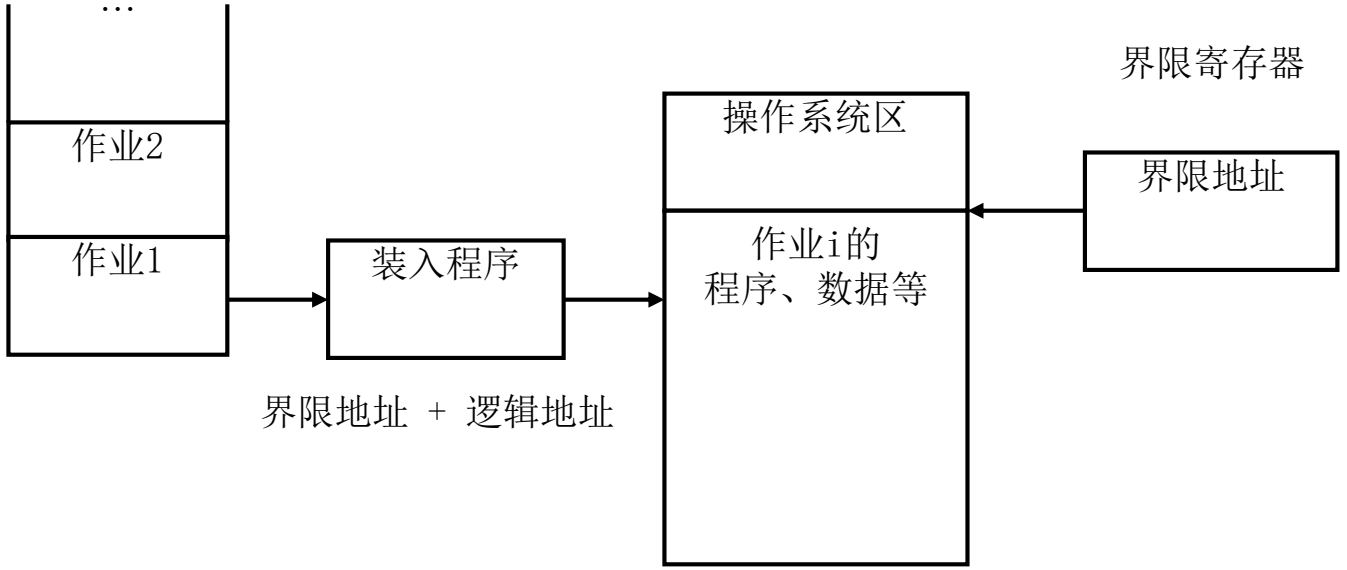


图3.4 采用静态重定位的单一连续内存管理

3.2.1 单一连续内存管理

- 内存保护由装入程序来执行，装入时由装入程序检查物理地址是否超过界限地址，超过则可以装入；否则产生地址错误，不能装入。这样，用户的程序总是被装入到合法的用户区域内，而不会进入系统区。
- 采用静态重定位的优点是实现简单，无需硬件地址变换机构支持。缺点是作业只能分配到一个连续存储区域中，程序执行期间不能在内存中移动，无法实现程序共享。

3.2.1 单一连续内存管理

- 单一连续内存管理也可以采用动态重定位方式来转换地址。系统设置一个定位寄存器，它既用来指出内存中的系统区和用户区的地址界限，又作为用户区的基地址；装入程序把程序装入到从界限地址开始的区域，但不同时进行地址转换；而是在程序执行过程中动态地将逻辑地址与定位寄存器中的值相加就可得到绝对地址，如图3.5所示。

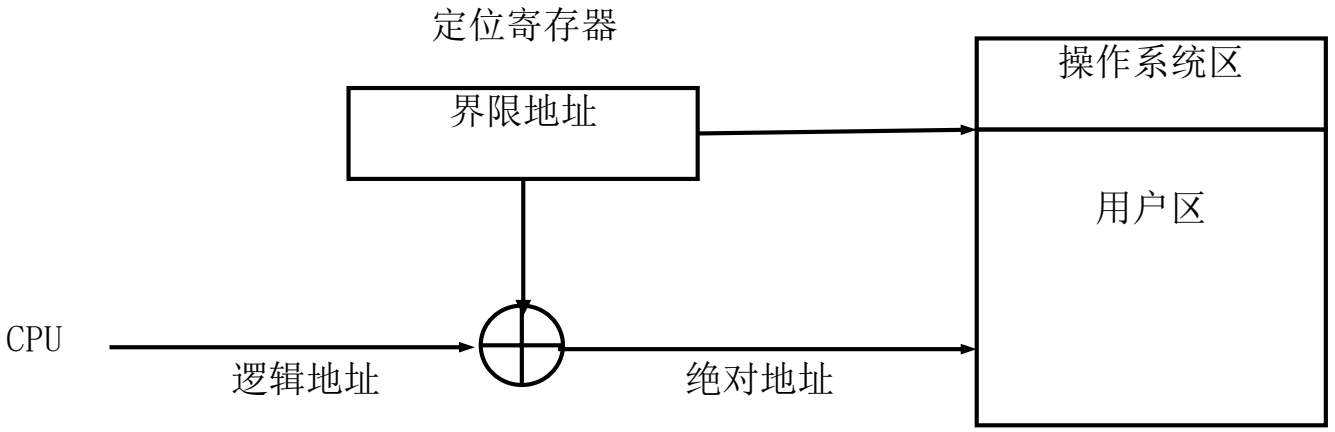


图3.5 采用动态重定位的单一连续内存管理

3.2.1 单一连续内存管理

单一连续内存管理非常简单，系统开销小，其主要缺点如下：

- 1) 内存利用率低。用户程序所需空间一般均小于内存用户区空间，剩余的内存空间也不能被其它用户使用。
- 2) CPU利用率低。当运行中的程序进行I/O操作时，CPU会处于空闲等待状态。
- 3) 外设利用率低。用户控制所有资源，有些资源在运行期间可能并不使用，也不能为其它用户使用。
- 4）不能进行内存扩充。当内存容量小于某一程序所需要的内存空间时，该程序便无法运行。

3. 2. 2 固定分区内存管理

- **1. 基本思想**

固定分区内存管理是预先把可分配的内存空间分割成若干个大小固定的连续区域，每个区域的大小可以相同，也可以不同，每个区域称为一个分区。每个分区可以装入且只能装入一个用户作业。这样，分区后的内存中就可以装入多道程序，从而支持多道程序并发设计。如图3.6所示。

操作系统区（8KB）
用户分区（8KB）
用户分区（16KB）
用户分区（16KB）
用户分区（16KB）
用户分区（32KB）
用户分区（32KB）

3. 2. 2 固定分区内存管理

- **分区划分的方法有分区大小相等和分区大小不等两种方式。**

- **分区大小相等**

  所有分区的大小都相等，这种方式适合计算机工业控制系统。因为在计算机工业控制系统中，所有控制对象都具有相同的条件，完成相同的控制任务和控制指标。

  该方式的缺点是：因为分区大小都一样，所以较小的进程装在分区里会浪费内存，而较大的进程则无法装入内存运行。

3.2.2 固定分区内存管理

- **分区大小不等**

把可分配的内存空间分割为大小不等的多个分区，大的分区可以分配给大的进程，小的分区可以分配给小的进程。与分区大小相等分配方式比较，分区大小不等的分配方式使得内存的分配更加灵活，内存的浪费更少。

3.2.2 固定分区内存管理

3. 固定分区的内存分配

为了说明各分区的分配和使用情况，系统设置一张“内存分配表”，该表如图3.7所

分区号	起始地址	长度	占用标致
1	5KB	5KB	0
2	10KB	10KB	0
3	20KB	10KB	Jobl
4	30KB	20KB	0
5	50KB	20KB	Job2
6	70KB	30KB	0

图3.7 固定分区内存管理的内存分配表

3. 2. 2 固定分区内存管理

- 内存分配表指出了各分区的起始地址和分区的长度，占用标志位指示该分区是否被使用，当占用的标志位为“0”时，表示该分区尚未被占用。只能将那些占用标志为“0”的分区分配给用户作业使用，当某一分区被分配给一个作业后，就在占用标志栏填上占用该分区的作业名，如在图3.7中，第3和第5分区分别被作业Job1和Job2占用，而其余分区为空闲。

3 3.2.2 固定分区内存管理

- 静态重定位：装入程序在进行地址转换时检查其绝对地址是否在指定的分区中，若是，则可把程序装入，否则不能装入。固定分区方式的内存回收很简单，只需将内存分配表中相应分区的占用标志位置成“0”即可。
- 动态重定位：如图3.8所示，计算机系统设置了一对地址寄存器——上限/下限寄存器；当一个进程占有CPU执行时，操作系统就从内存分配表中取出相应的地址放进上限/下限寄存器；硬件的地址转换机构根据下限寄存器中保存的基地址B1与逻辑地址相加就得到绝对地址；硬件的地址转换机构同时把绝对地址和上限/下限寄存器中保存的地址进行比较，就可以实现存储保护。

3. 2. 2 固定分区内存管理

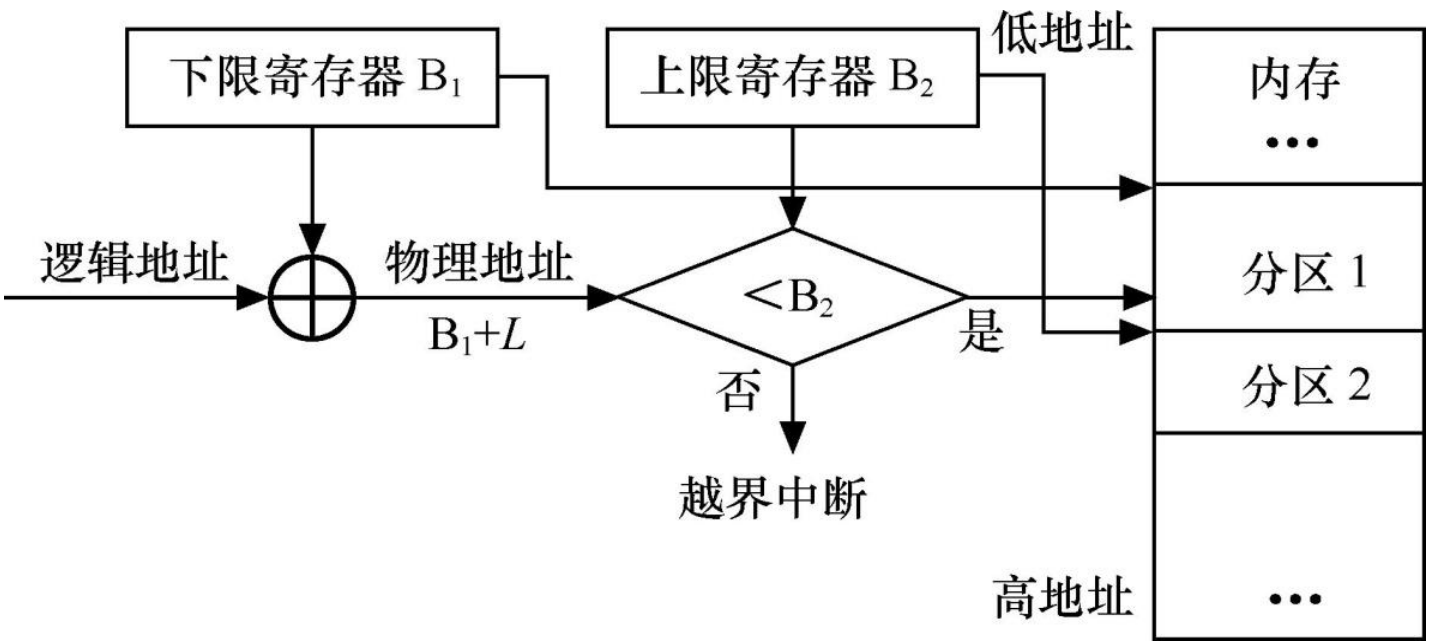


图3.8 固定分区存储管理的地址转换和存储保护

3.2.2 固定分区内存管理

•5. 固定分区分配的优点和缺点

固定分区的划分在操作系统初始化时完成。在系统启动时，系统管理员根据系统要运行的作业的需要来划分分区。当用户作业进入分区时，按照用户作业的大小从分区表中选择适当的空闲分区。与单一连续分配方式比较，固定分区分配方式使得系统资源的利用率和吞吐量有一定的提高。

3. 2. 2 固定分区内存管理

- **缺点:**

1）内存空间的利用率不高：例如图3.7中若Job1和Job2两个作业可能实际只需要8 K和16 K的内存，但它们却占用了10 K和20 K的两个分区，分别浪费了2K和4K的内存空间。

2）由于每个分区大小固定，这样就限制了可容纳的程序的大小。在装入一个程序时，若找不到足够大的分区，则无法装入。

3. 2. 3 可变分区内存管理

- **1. 基本思想**
- 事先不确定分区的大小，也不确定分区的数目。当某一用户作业申请内存时，检查内存中是否有一块能满足该作业的连续存储空间，若有就把这一空间划出一块区域给该用户使用，这种方式就称作可变分区内存管理。
- 分区的大小是按作业的实际需要量来定的，分区的个数也是随机的，所以可变分区内存分配可以克服固定分区方式中的内存的浪费现象。
- 可变分区克服了固定分区内存利用率低的问题，更适合多道程序环境。

3.2.3 可变分区内存管理

- 采用可变分区方式的内存分配示例如图3.9。

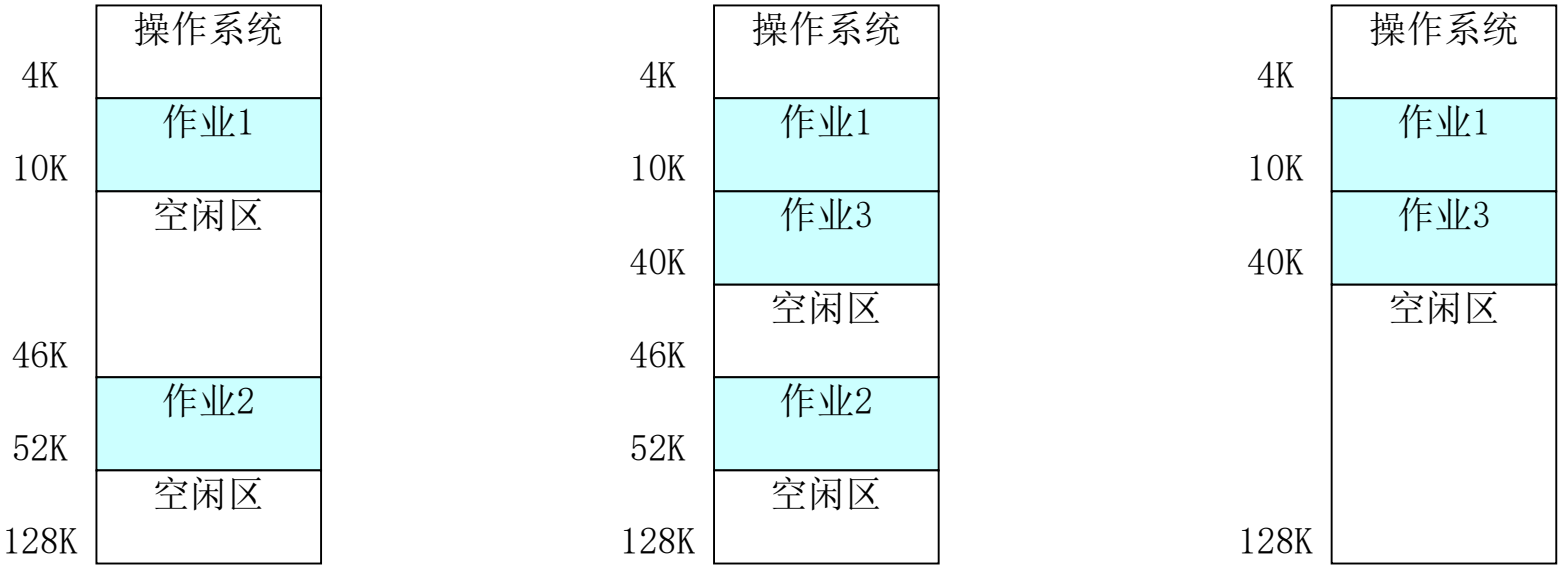


图3.9 可变分区内存的分配示例

3. 2. 3 可变分区内存管理

- 随着作业的装入、撤离，内存空间被分成许多个分区，有的分区被占用，有的分区空闲。当一个新的作业要求装入时，必须找一个足够大的空闲区，能容纳该作业，如果找到的空闲区大于作业需要量，则作业装入后又把原来的空闲区分成两部分，一部分被作业占用；另一部分又成为一个较小的空闲区。当一作业运行结束撤离时，它归还的区域如果与其它空闲区相邻，则可合成一个较大的空闲区，以利于大作业的装入。

3. 2. 3 可变分区内存管理

- 对分区信息进行描述的数据结构，一张是已分配区的情况表，另一张是未分配区的情况表，如图3.10。

分区号	起始地址	长度	标志
1	4KB	6KB	Job1
2	46KB	6KB	Job2

(a) 已分配区情况表

分区号	起始地址	长度	标志
1	10KB	36KB	未分配
2	52KB	76KB	未分配

(b) 未分配区情况表

图3.10 可变分区存储管理的内存分配表

3. 2. 3 可变分区内存管理

- 图3.10的两张表的内容是根据图3.9最左边的情况生成的。当要装入长度为30K的作业3时，首先从未分配区情况表中寻找一个能够容纳它的空闲区，长度为36K的空闲区就适合此作业；然后将该区分成两部分，一部分30K，用来装入作业 3 ，成为已分配区；另一部分为6K，仍是空闲区。这时，应从已分配区情况表中找一个空栏目登记作业 3 所占用的区的起址、长度，同时修改未分配区情况表中空闲区的长度和起址。当作业撤离时则已分配区情况表中的相应状态应改成“空”，而将收回的分区登记到未分配情况表中，若有相邻空闲区则将其连成一片后登记。

3. 2. 3 可变分区内存管理

- 操作系统也可以通过链表方式来管理空闲分区，将所有的空闲分区通过前向和后向指针串在一起组成双向空闲分区链，如图3.11所示

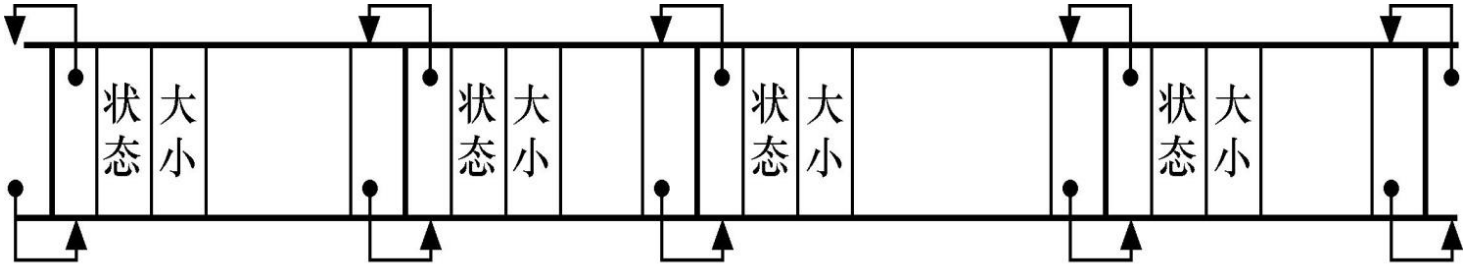


图3.11 空闲区链表

3.2.3 可变分区内存管理

- 空闲区链表管理比空闲区表格管理较为复杂但其优点是链表自身不占用存储单元。为了方便检索空闲分区链，每个空闲分区的首部还设置有分区状态和分区大小标志，这样，系统查找分区链时可直接得知分区的大小和分配情况，省去了查询空闲分区表的时间。
- 不论是空闲区链表管理还是空闲区表格管理，链和表中的空闲区都可按一定规则排列，例如，按空闲区从大到小排列或从小到大排列，以方便空闲区的查找和回收。

3.2.3 可变分区内存管理

3. 可变分区的内存分配算法

1）最先适应分配算法。每次分配时，总是从头顺序查找未分配区表或空闲区链表，将找到的第一个能满足长度要求的空闲区分配给用户作业使用。从该空闲分区中分割一部分给作业，另一部分仍作为空闲分区；如果空闲分区全部查找完也不能满足该作业要求，则系统不能为该作业分配内存。

- 首先利用内存中的低地址空闲分区，保留了大的高地址空闲分区，以便能容纳大的用户作业。
- 缺点：每次都是从未分配分区表或空闲区链表的开始查找空闲分区，低地址段的空闲分区被不断分割，形成了许多小的、难以利用的空闲分区，称为“碎片”；同时每次都从开始查找，花费时间较长。

3.2.3 可变分区内存管理

2）循环首次适应分配算法。循环首次适应法是对最先适应法的改进。为作业分配内存时，系统不是从空闲分区表的开始处开始查找，而是从上次为作业分配分区后的位置开始查找，找到第一个满足作业大小的空闲分区，分配并分割该空闲分区。

- 该分区分配算法克服了首次适应算法的缺点，使得空闲分区的分布更加均匀，查找空闲分区所需要的时间更短。但是，小分区或“碎片”问题仍然不能解决。

3.2.3 可变分区内存管理

3）最优适应分配算法：从空闲区中挑选一个能满足作业要求的最小分区，这样可以避免分割一个更大的区域，使大作业比较容易装入。

- 可把空闲区按长度以递增顺利排列，查找时总是从最小的一个区开始，直到找到一个满足要求的区为止。
- 收回一个分区时也必须对空闲区链重新排列。
- 最优适应分配算法找出的分区一般都是无法正好满足作业的内存要求，分割后剩下的空闲区很小，无法再次使用，成为“碎片”。另外，这些小的空闲区占据了空闲区表的开始部分，增加了查找空闲区表或空闲区链的时间开销。

3.2.3 可变分区内存管理

4）最坏适应分配算法：从空闲区中挑选一个最大的区给作业使用，这样可使分割后剩下的空闲区仍然比较大，仍然能满足以后的作业装入要求，也减少了内存中“碎片”的大小和个数。

- 可把空闲区按长度以递减顺序排列，查找时只要看第一个分区能否满足作业要求，若能满足，则分配给该作业使用。
- 最坏适应分配算法的查找效率很高，对中、小作业有利。
- 最坏适应分配算法缺点：随着系统的运行，大空闲区会不断减少，这样，大的作业可能会无法装入内存。

3. 2. 3 可变分区内存管理

5）快速适应算法：把不同长度的空闲区归类，为每种长度的空闲区设立单独的空闲区链表。这样，系统中存在多个空闲分区链。

- 例如，有一个N项的空闲分区表，该表第一项是指向长度为2KB的空闲区链表表头的指针，第二项是指向长度为4KB的空闲区链表表头的指针，第三项是指向长度为8KB的空闲区链表表头的指针，依此类推。
- 为作业分配内存时，根据作业大小查找空闲分区表，找到能够容纳它的最小的空闲分区链表的起始指针，然后再从相应的空闲分区链中取第一个空闲分区分配给该作业即可。
- 优点是查找空闲分区迅速，找到的空闲分区是能容纳它的最小空闲区，这样能够保留大的空闲分区。
- 缺点是回收分区较困难，算法复杂，系统开销大。

3.2.3 可变分区内存管理

- **对比分析**
- 从搜索空闲区速度及内存利用率来看，最先适应分配算法、循环首次适应分配算法和最优适应算法比最坏适应算法性能好。如果空闲区按从小到大排列，则最先适应分配算法等于最优适应分配算法。反之，如果空闲区按从大到小排列，则最先适用分配算法等于最坏适应分配算法。空闲区按从小到大排列时，最先适应分配算法能尽可能使用低地址区，从而，在高地址空间有较多较大的空闲区来容纳大的作业。循环首次适应分配算法会使存储器空间得到均衡使用。最优适应分配算法的内存利用率最好，因为，它把刚好或最接近申请要求的空闲区分给作业；但是它可能会导致空闲区分割下来的部分很小。

3.2.3 可变分区内存管理

4．地址转换与内存保护

- 可变分区内存管理是采用动态重定位方式来装入作业的，其地址转换由硬件机构完成。硬件设置了两个专门的寄存器：基址寄存器和限长寄存器。
- 基址寄存器存放分配给作业使用的分区的起始地址
- 限长寄存器存放该分区的存储空间的长度。

3. 2. 3 可变分区内存管理

当用户作业占有CPU运行时，操作系统把分配给该作业的分区的起始地址和长度送入基址寄存器和限长寄存器，启动作业执行时由硬件根据基址寄存器进行地址转换得到绝对地址，地址转换如图3.12所示。

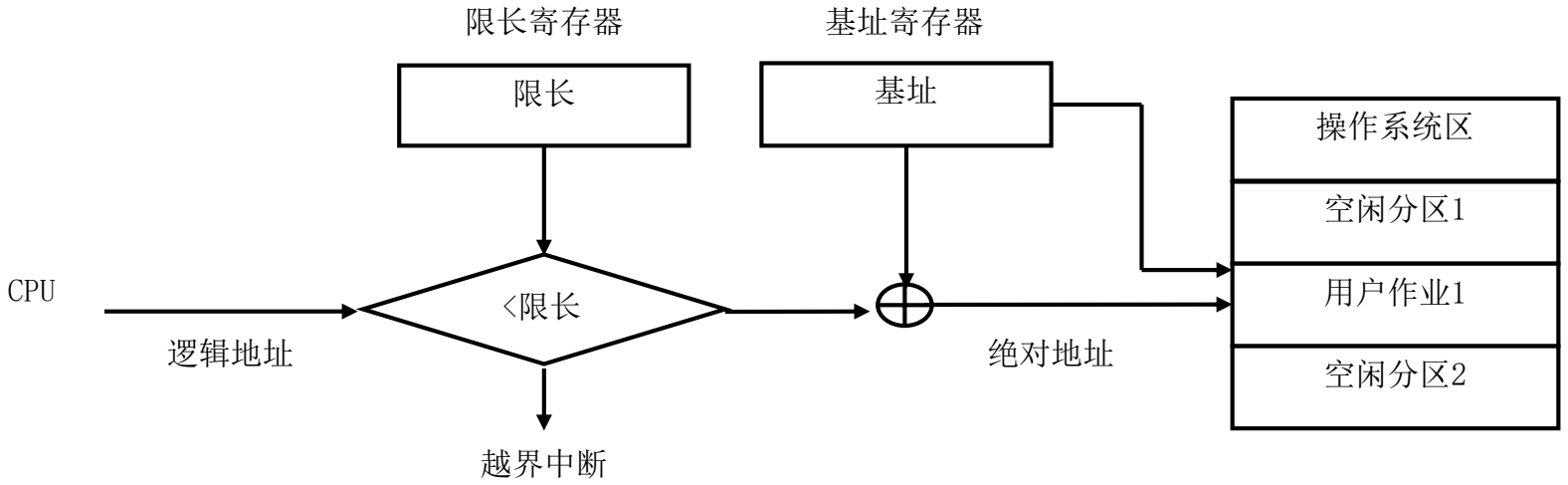


图3.12 可变分区存储管理的地址转换和存储保护

3. 2. 3 可变分区内存管理

- 当逻辑地址小于限长值时，则由逻辑地址加基址寄存器的值就可得到绝对地址；当逻辑地址大于限长值时，就表示作业欲访问的内存地址超出了所分得的内存区域，禁止访问该地址，起到了存储保护的目的。
- 在多道程序系统中，只需要一对基址/限长寄存器就足够了。因为当作业在执行过程中出现等待时，操作系统把基址/限长寄存器的内容随同该作业的其它信息，如PSW，通用寄存器的内容等一起保存起来。当作业被选中执行时，则把选中作业的基址/限长值再送入基址/限长寄存器。
- 世界上最早的巨型机 CDC6600 便采用这一方案。

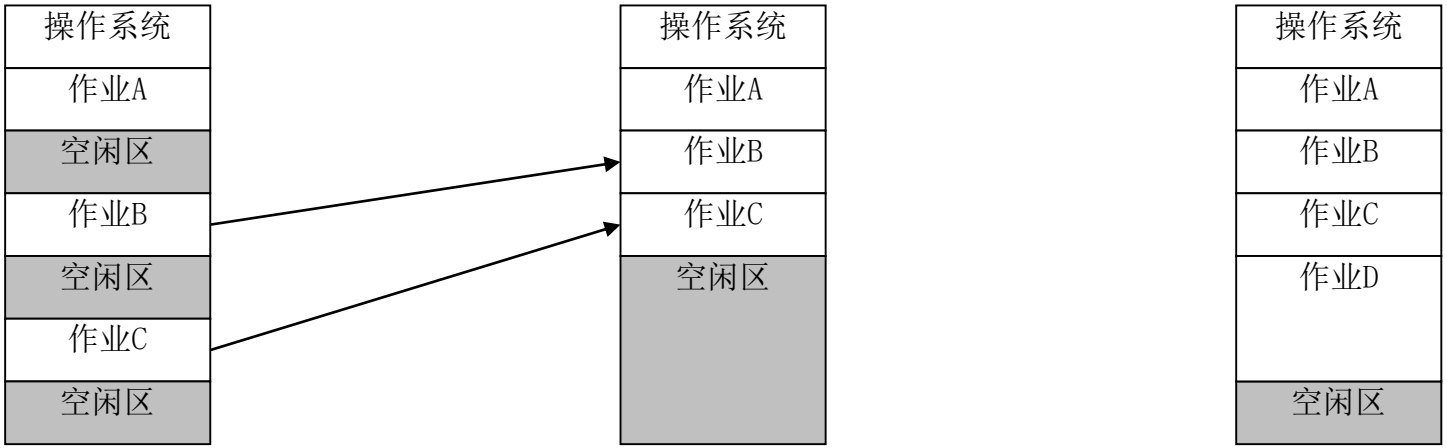
3

施乐Alto微电脑

1973年施乐公司Xerox推出了Alto，它首次将计算机所有元素都结合在一起。Alto具有图形界面操作系统，使用3键鼠标、位运算显示器、图形窗口和以太网络连接。Alto能与另一台Alto计算机和激光打印机连成网络。

3.2.3 可变分区内存管理

当系统运行一段时间后，内存被多次分配和回收，会产生许多不连续的空闲空间。有可能出现这样的现象：内存中每一块空闲空间都不能满足某一作业的内存请求，而所有空闲空间的总和却能满足该作业。这时可采用紧凑技术把内存中的作业改变存放区域，使分散的空闲区能够汇聚成一个大的空闲区，从而有利于大作业的装入，紧凑技术的示例如图3.13。



(a) 原内存分配情况　　b) 紧凑后内存中的作业　　(c) 装入新作业

3. 2. 3 可变分区内存管理

- 紧凑技术可以汇聚内存中的空闲区，但也增加了系统的开销，而且不是任何时候都能对一道程序进行移动的。比如：当外围设备正在与内存进行信息交换时，会按照已经确定的内存绝对地址完成信息传输的，所以此时不能移动。

3 3.3 页式内存管理

- 非连续的内存分配方式，也叫离散分配方式，其基本出发点是打破程序装入的整体性和存储分配的连续性，首先将用户进程的逻辑地址空间划分成多个子部分，以子部分为单位装入物理内存，这些子部分可以分布在若干非连续的内存块上，实现了离散储存，以充分利用内存。
- 内存非连续分配方式主要包括页式存储管理和段式存储管理两种方式。

3. 3. 1 页式存储管理的基本原理

- **页：**将用户进程的逻辑地址空间划分为大小相等的区，每一个区称为一页或一个页面，并对各页从0开始编号，如第0页、第1页等。
- **物理块：**将物理内存也划分成与页大小相等的区，每一个区称为一个物理块(block)，或称为块、页框，也同样对它们加以编号，如0号块、1号块等。
- 物理块的尺寸大小通常会取2的幂次。物理块的大小由计算机硬件决定，页的大小由物理块的大小决定。如Intel 80386系列处理器系统和Motorola 68030处理器系统的块的大小为4096B，IBM AS/400的块的大小为512B。

3.3.1 页式存储管理的基本原理

内存分配的基本单位是页，当装入一个用户程序时，按页为单位，每一页装入一个物理块中，一个用户进程装入到内存中时各个物理块之间不需要连续。

进程的最后一页经常装不满一块，所以会在最后一块内形成不可利用的碎片，称之为“页内碎片”。而其他页在装入内存时，都能填满所在的物理块。

逻辑地址形式：进程的逻辑地址可以用页号和页内偏移表示，页的大小与物理块的大小相等。逻辑地址格式如下：

3. 3. 1 页式存储管理的基本原理

- 例如，现在流行的32位操作系统其逻辑地址是32位，采用页式内存管理，如果每页大小4096B，那么页内偏移要占用其逻辑地址的低12位，从0位开始到11位结束。逻辑地址剩余的高20位用来表示页号，从12位开始到31位结束，这样最多允许有2^{20}（1M）个页面。页面的编号从0开始，分别为0，1，2，3 …，$2^{20}-1$，如图3.14所示。

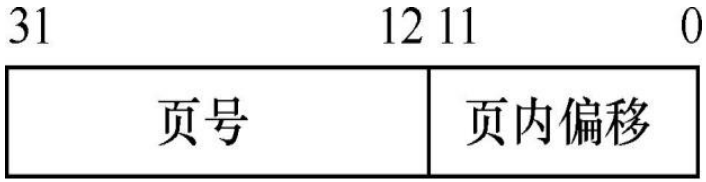


图3.14 页式存储的逻辑地址

3. 3. 1 页式存储管理的基本原理

- 如果进程的逻辑地址是A，页面大小是L，则页号P和页内偏移d为：

$$P = INT[A/L], \quad d = [A] MOD\ L$$

- 其中INT表示求整数，MOD表示求余数。

3.3.1 页式存储管理的基本原理

- 例如：某计算机系统每页大小为1KB（1024），计算逻辑地址2345（十进制）的页号和页内偏移：

L = 1024， A = 2345

P = int[2345/1024] = 2

d = [2345]mod 1024 = 297

这就表示逻辑地址2345处于2号页面，页内偏移为297。

3.3.1 页式存储管理的基本原理

还可以通过计算地址位得到页号和页内偏移量，2345所对应的2进制数为：1001，0010，1001；

页的大小为1024，即页内偏移占用低10位地址：01，0010，1001，相当于十进制的297，即页内偏移量为297；

页号占用剩余22位高地址段，即页号为10，相当于十进制的2。

所以，逻辑地址2345对应于2号页面，页内偏移为297。

3.3.2 页式存储管理的内存的分配与回收

页式存储管理在进行内存分配时，以物理块为单位进行分配，一个作业有多少页，在装入内存时就必须给它分配多少个物理块。但是，和分区式内存管理不同的是分配给作业的物理块可以是不连续的。

在进行内存分配时，首先，操作系统为进入内存的每个用户作业建立一张页表，页表用来指出逻辑地址中的页号与内存中物理块号的对应关系。

同时，在页式存储管理系统中还存在一张作业表，作业表中的每个登记项登记了每个作业的页表始址和长度。作业表和页表的一般格式如图3.15所示。

3.3.2 页式存储管理的内存的分配与回收

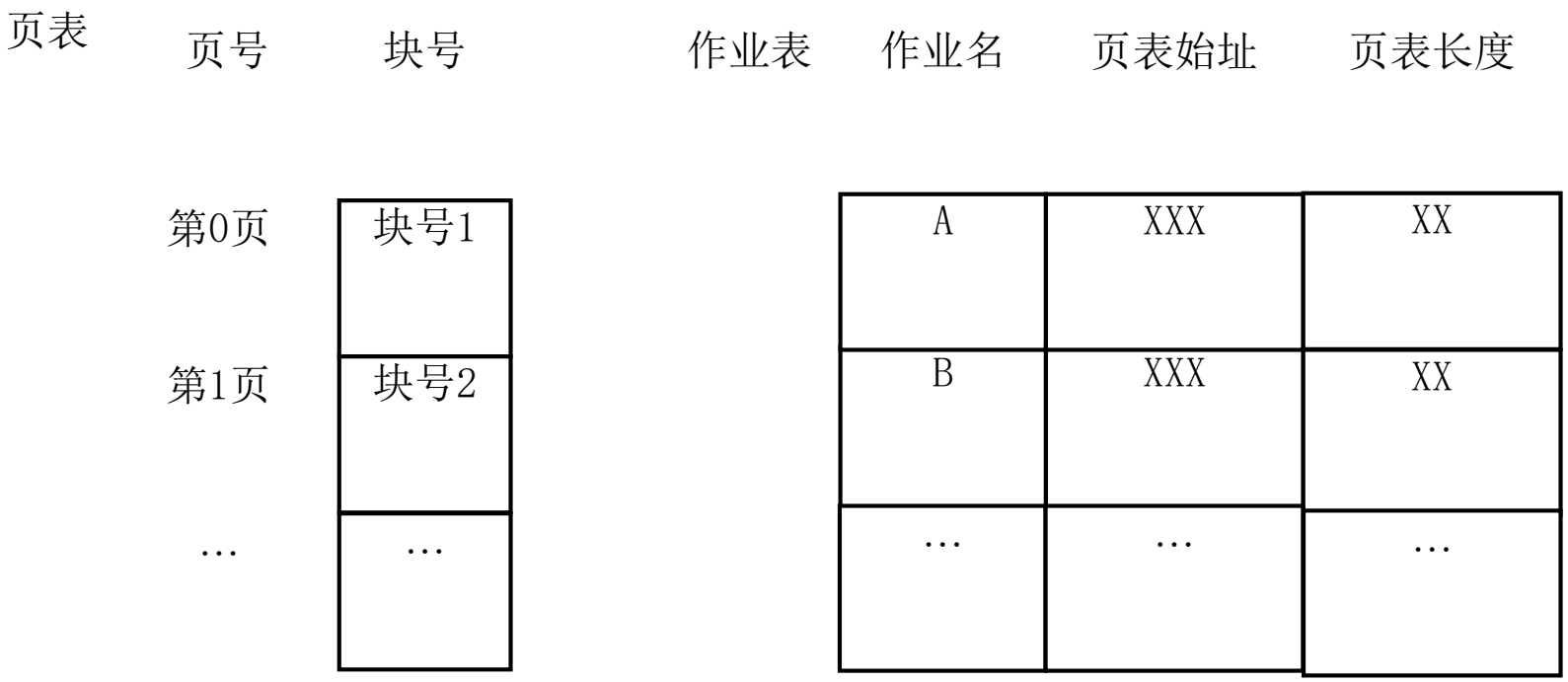


图3.15 页表和作业表的一般格式

3. 3. 2 页式存储管理的内存的分配与回收

- 某进程的页表如图3.16所示，页面大小为1KB，则逻辑地址2345在第2号页面，页内偏移为297。
- 查找页表，得到该页在内存中的块号为4，块号乘块长为4096，该逻辑地址在内存中的物理地址为4096加上块内偏移，页内偏移即等于块内偏移，为297。所以，物理地址为4096加297，即4393。

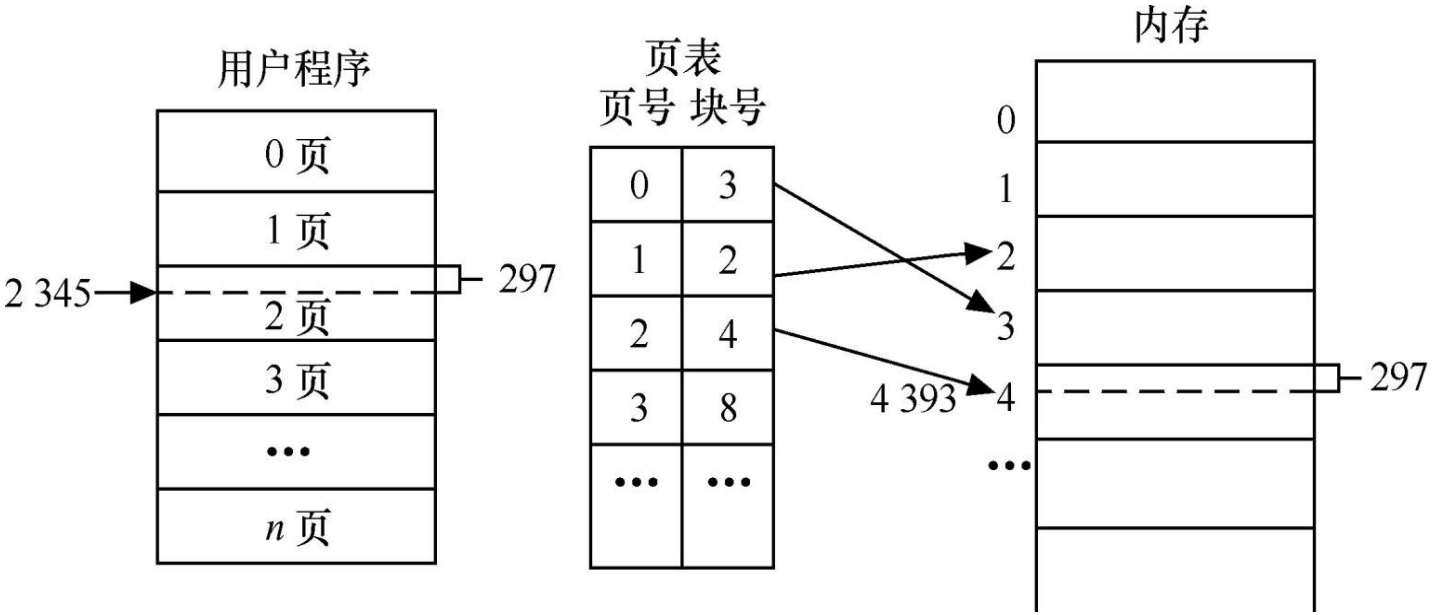


图3.16 进程的页表

3.3.2 页式存储管理的内存的分配与回收

- 页表的表项中除了有页号和块号外，还有存取控制字段，用于实现对内存物理块的保护。页表的长度由用户进程的长度决定，每个在内存中的用户进程都会建立一张页表。如果进程不处于运行状态，页表的起始地址和长度存放在进程的PCB中。只有某一进程被调度运行时，系统才会从运行进程的PCB中将页表起始地址和长度装入页表寄存器。
- 所以，一个处理器只需要一个页表寄存器。

3.3.3 页式存储管理的地址转换

- 在页式存储管理中，进程的逻辑地址到物理地址的转换需要硬件来完成，该硬件为地址转换机构。

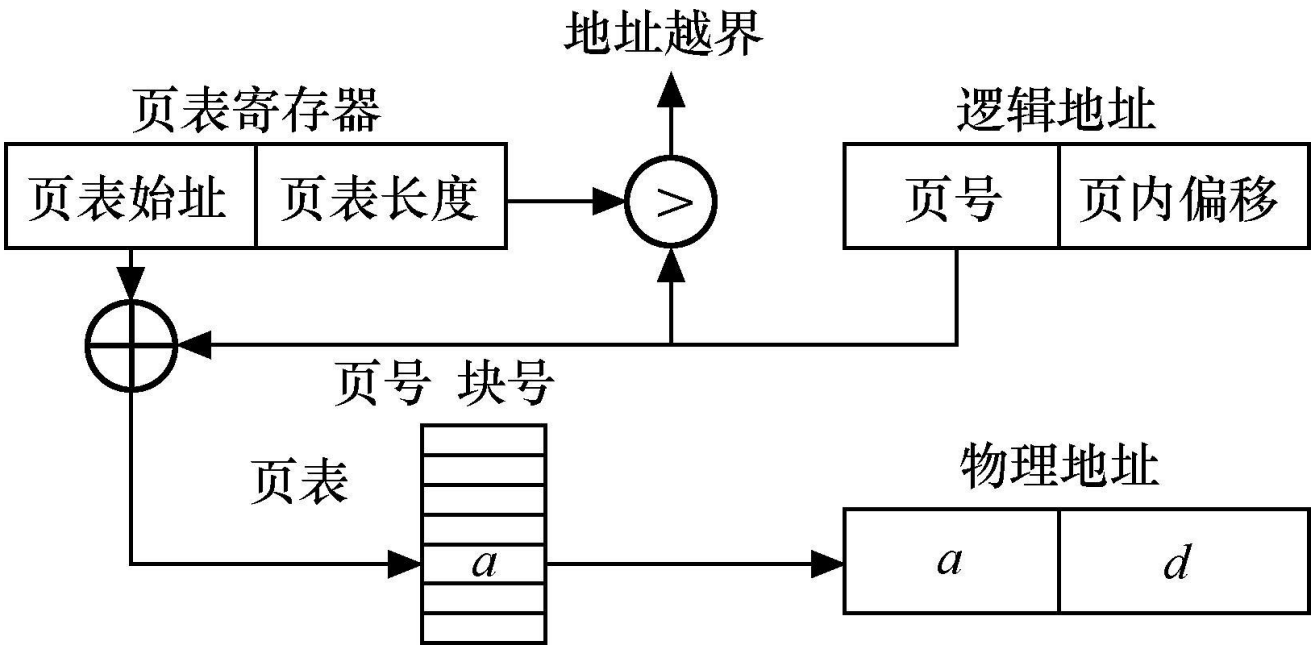


图3.17 页式存储管理的地址转换机构

3 3.3.3 页式存储管理的地址转换

- 当处理器要访问某逻辑地址时，地址转换机构自动从逻辑地址的低地址部分得到页内偏移，从高地址部分得到页号。将页号与页表寄存器中的页表长度比较，如果页号大于或等于页表长度，表示该页在页表中没有相应项，本次所访问的地址已经超越进程的地址空间，则产生地址越界中断；
- 否则，从页表寄存器得到页表在内存中的起始地址。用页号作为索引查找页表，得到页表项，从而可以查到该页在内存中的物理块号。
- 最后，将页内偏移装入物理地址寄存器的低位字段中，将物理块号装入物理地址寄存器的高位字段中，此时物理地址寄存器中的内容就是物理地址，如图3.17所示。

3.3.4 快表

- 由地址转换过程可见，处理器每运行一条指令总是先根据指令的逻辑地址，通过访问内存中的页表才能得到物理地址，根据物理地址再去访问内存才能得到需要的指令，即处理器需要两次访问内存。同样，处理器要访问一个数据也需要两次访问内存。
- 为了提高程序的运行速度，可以将最近访问过的页的页表项信息存放在高速缓存中，高速缓存也称为“联想存储器”，其中的页表称为“快表”。

3.3.4 快表

- 计算机系统中通常都设置一个专用的高速缓冲存储器，用来存放页表的一部分，称为相联存储器（associative memory），存放在相联存储器中的页表称快表。
- 相联存储器的访问速度比内存快，但造价高，故一般都是小容量的，例如8~16个单元。高速缓冲存储器由半导体实现，其工作周期与CPU的工作周期接近，所以访问快表的速度远快于访问内存中页表的速度。

3.3.4 快表

- 根据程序执行的局部性的特点，即在一定时间内总是反复地访问某些页，若把这些页登记在快表中，无疑地将大大加快指令或数据的访问速度。快表的格式如下.
- 快表里登记了已在相联存储器中的页及其对应的内存的块号。

页号	块号
……	……
页号	块号

3. 3. 4 快表

- 借助于快表，物理地址形成的过程是：

1. 按逻辑地址中的页号查快表，若该页已登记在快表中，则由块号和块内偏移形成绝对地址；
2. 若快表中查不到页号，则只能再查内存中的页表来形成绝对地址，同时将该页登记到快表中。
3. 当快表填满后，又要在快表中登记新页时，则需在快表中按一定策略淘汰一个旧的登记项，最简单的策略是“先进先出”，即总是淘汰最先登记的那一页。

3.3.4 快表

- 采用快表可以使地址转换时间大大下降，例如：假定访问内存的时间为200纳秒，访问快表的时间为40纳秒，相联存储器为16个单元时查快表的命中率可达90%，于是按逻辑地址进行存取的平均时间为：
- （200＋40）×90%＋（200＋200+40）×10%＝260纳秒
- 比两次访问内存的时间200×2＝400纳秒下降了36%。

3.3.4 快表

- 整个系统只有一个相联存储器，只有占用CPU的进程才能占有相联存储器。
- 多道程序中，当某道程序让出处理器时，应同时让出相联存储器。由于快表是动态变化的，所以让出相联存储器时应把快表保护好以便再执行时使用。当一道程序占用处理器时，除设置页表寄存器外还应将它的快表送入相联存储器中。

3.3.5 页的共享和保护

- 页式存储管理有利于实现多个作业共享程序和数据。在多道程序系统中，编译程序、编辑程序、解释程序、公共子程序、公用数据等都是可共享的，这些共享的信息在内存中只要保留一个副本。

- 在实现共享时，必须区分数据的共享和程序的共享。
- 实现数据共享时，可允许不同的作业对共享的数据页使用不同的页号，只要让各自页表中的有关表目指向共享的数据物理块。
- 实现程度共享时，由于页式存储的逻辑地址空间是连续的，所以程序运行前它们的页号应该是确定的。

3.3.5 页的共享和保护

- 现假定有一个共享的编辑程序，其中含有转移指令，转移指令中的转移地址必须指出页号和单元号，如果是转向本页，则页号应与本页的页号相同。现在如果有两个作业共享这个编辑程序，假定一个作业定义它的页号为3，另一作业定义它的页号为5，然而在内存中只有一个编辑程序，于是转移地址中的页号不能按作业的要求随机的改成3或5，因此对共享的程序必须规定一个统一的页号。当共享程序的作业数增多时，要规定一个统一的页号是较困难的。

3.3.5 页的共享和保护

- 实现信息共享必须解决信息的保护问题。常用的办法是在页表中增加一些标志位，用来指出该页的信息是：读/写；只读；只可执行；不可访问等，在指令执行时进行核对。例如，要想向只读块写入信息则指令停止，产生中断。

3. 3. 6 多级页表

- 为了快速查找页表页在内存中的物理块号，为这些页表页再设计一个地址索引表，即页目录表。页目录表的表项中包含每个页表页所在的内存物理块号和相关信息。系统将页表分为了两级：一级为页目录表，二级为页表页。
- 页表页中的每个表项记录了每个页的页号和对应的物理块号，如图3.18所示。

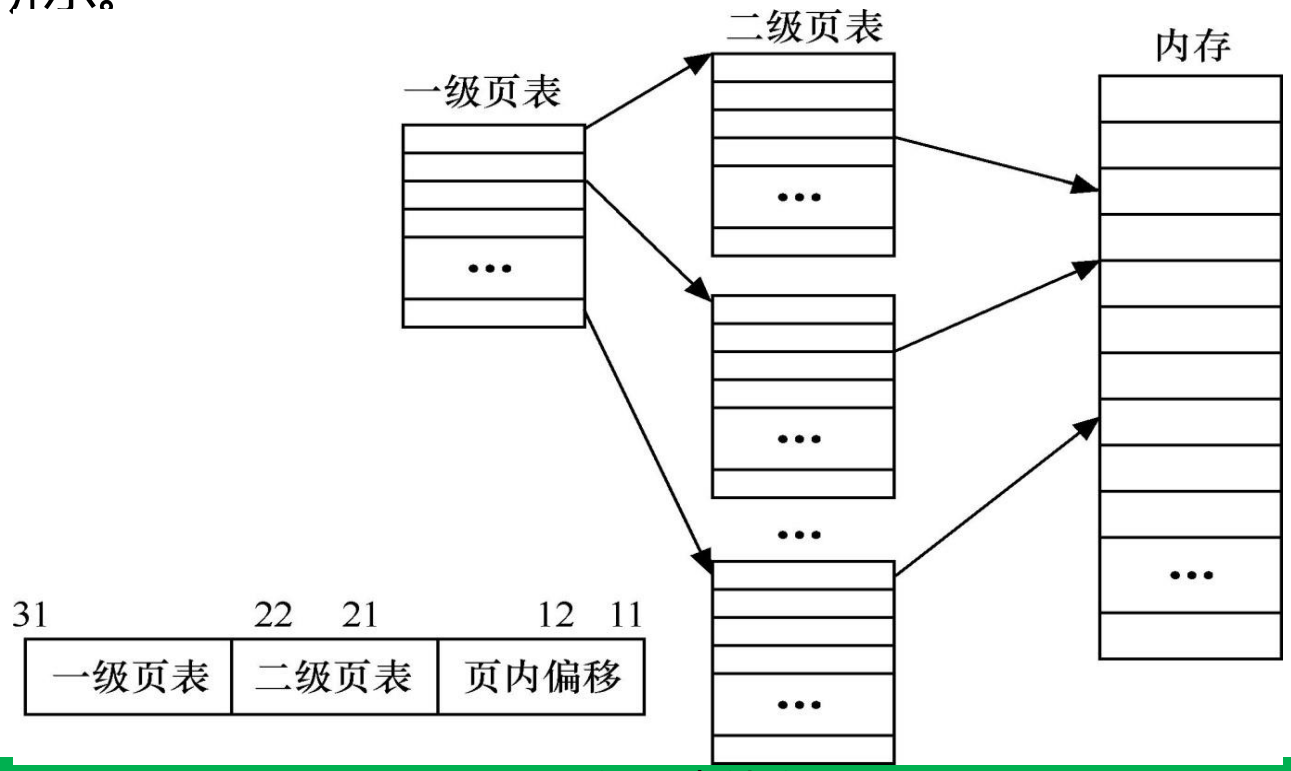


图3.18 二级页表结构

3.3.6 多级页表

二级页表的逻辑地址被划分为三部分：

页目录、页表页、页内偏移

首先由页目录表寄存器提供当前运行进程的页目录表（一级页表）在内存中的起始地址；

由页目录表（一级页表）在内存中的起始地址加上页目录号（即需要查找的页表某页在页目录中的编号），得到页表某页的物理块号；

通过页表某页的物理块号得到该页表页（二级页表中的一页），由页表页号（某页在页表页中的编号）查询该页表页（二级页表中的一页）项，得到对应的物理块号；

3.3.6 多级页表

最后将该物理块号加上页内偏移，最终得到物理地址。二级页表的地址转换过程如图3.19所示。

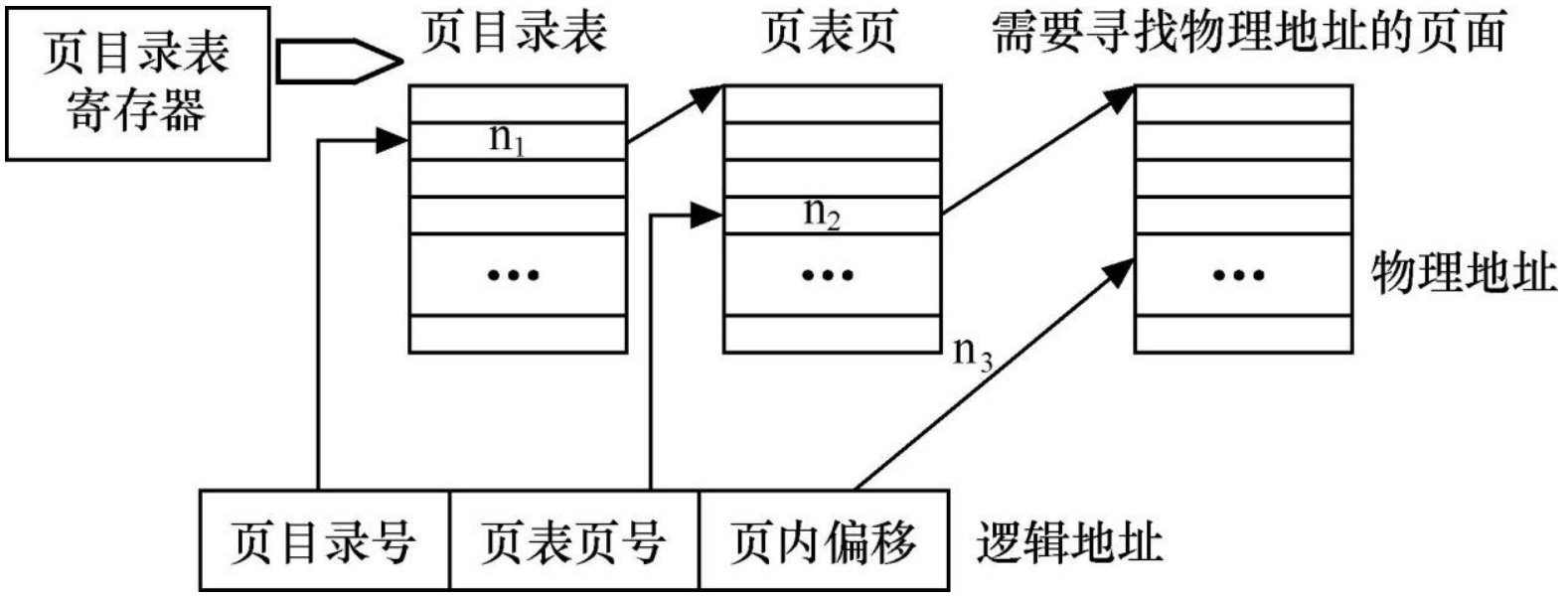


图3.19 二级页表地址转换过程

3.3.6 多级页表

- 二级页表地址变换获取内存信息需要三次访问内存：第一次访问的是页目录表，第二次访问的是页表页，第三次访问通过物理地址获取内存信息。为了节约时间，系统也可以采用快表获取内存信息。

- 当逻辑地址的位数更多时，系统会采用三级、四级，甚至更多级的页表。级别更多，灵活性越大，但是管理也更复杂。

3. 3. 6 多级页表

- 例如：SUN公司的Solaris操作系统基于SPACE处理器，采用了三级页表，如图3.20所示。32位逻辑地址分为四部分：高8位部分作为一级页表，之后的6位作为二级页表，再向后6位作为三级页表，最后12位作为页内地址，页面大小为4KB。

3.3.6 多级页表

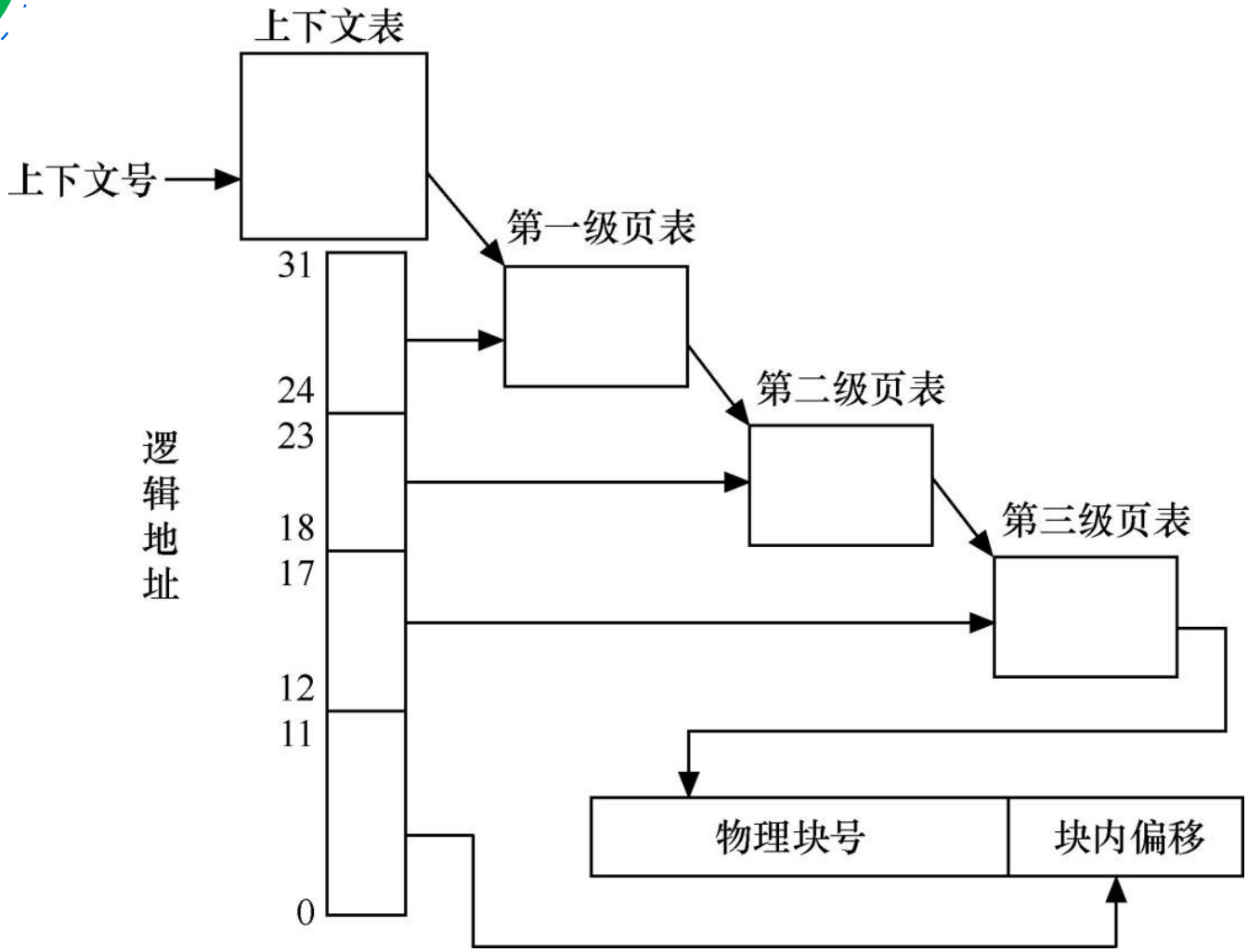


图3.20 Solaris的三级页表结构

3.3.6 多级页表

- 在地址变换之前，操作系统会给每个新进程分配一个上下文号，进程保留自己的上下文号直到结束。系统硬件支持高达4096个上下文号。地址变换过程是将上下文号与逻辑地址一起输入给地址变换机构。
- 上下文号作为上下文表的索引得到进程的第一级页表起始地址；逻辑地址中的高8位作为索引查找一级页表，得到二级页表的起始地址；
- 逻辑地址中的高8位之后的6位作为索引查找二级页表，得到三级页表的起始地址，逻辑地址中的再后面6位作为索引得到三级页表中的物理块号；
- 最后将逻辑地址中的低12位作为块内偏移，得到物理地址。

3

3.4 段式存储管理

3.4.1 段式存储管理的基本原理

1．逻辑地址空间分段

前面所讨论的内存管理方法都是假设用户程序从“0”开始编址，逻辑地址空间都是一维的。

事实上，一个程序往往由一个程序段、若干子程序数组段和工作区段所组成，每个段都从“0”开始编址，段内地址是连续的。

因此，可以按照程序的逻辑段结构，将一个程序按段为单位来分配内存，一段占用一块连续的内存地址空间，段与段之间不需要连续。

3.4.1 段式存储管理的基本原理

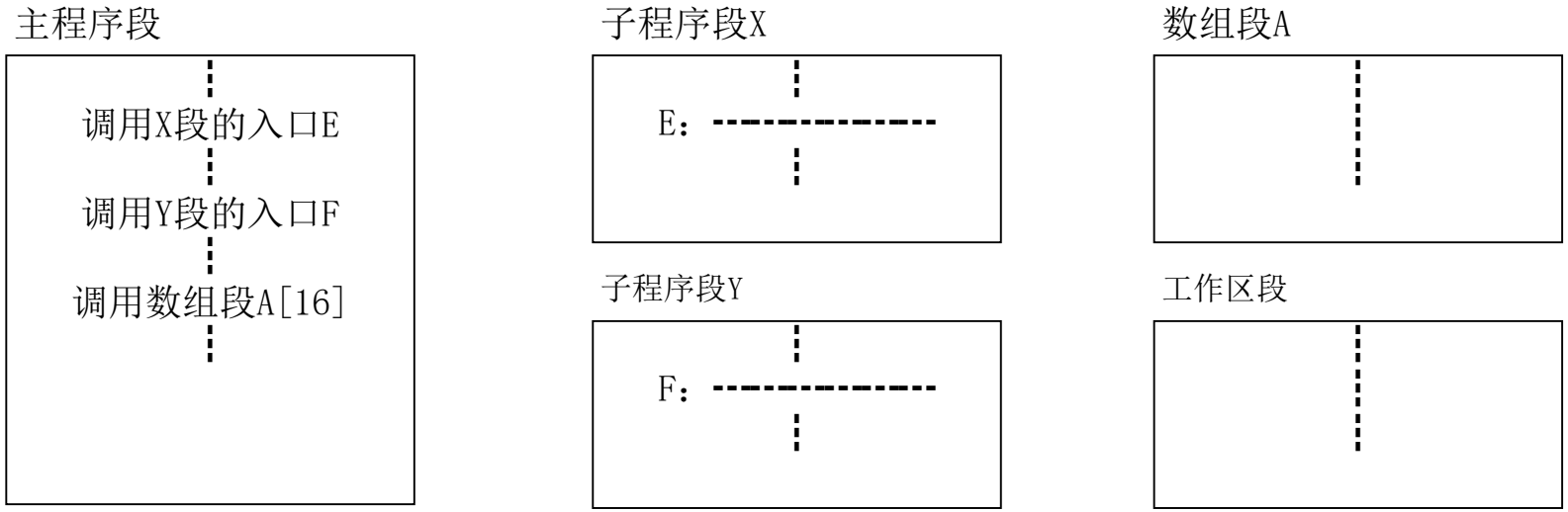
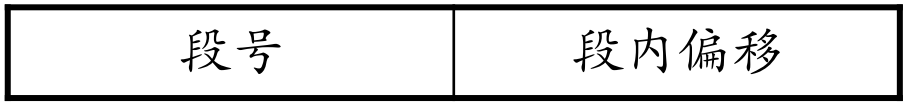


图 3.21　程序的分段结构

3. 4. 1 段式存储管理的基本原理

- 分段式存储管理是以段为单位进行内存分配，逻辑地址空间是一个二维空间，分为段号和段内偏移两部分，如下所示。



- 页式存储管理中，逻辑地址分页用户不可见，连续的逻辑地址空间根据内存物理块的大小自动分页。
- 段式存储管理中，由用户来决定逻辑地址如何分段。用户在编程时，每个段的最大长度受到地址结构的限制，每个程序中允许的最多段数也受限制。
- 例如，PDP-11/45的段址结构为：段号占3位，段内地址占13位，也就是一个作业最多可分8段，每段的长度最大8K字节。

3.4.1 段式存储管理的基本原理

- 分段存储管理中操作系统为每个作业创建一张段表，每个段在段表中占有一项。段表中有段号、段长、段在内存的起始地址和存取控制字段等信息。
- 段式存储管理系统还包括一张作业表，每个作业在作业表中有一个登记项。作业表和段表的一般格式如图3.22：

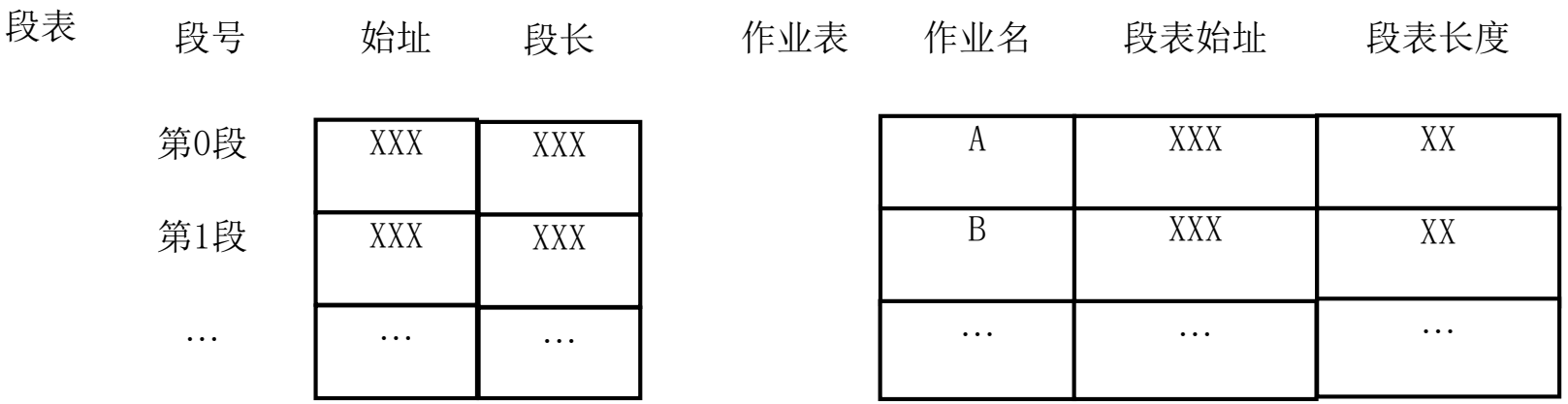


图3.22 段表和作业表的一般格式

3.4.1 段式存储管理的基本原理

- 在段式存储管理中，对内存的分配类似于可变分区内存分配方式，因此其内存分配算法可以参照可变分区内存分配算法来设计，比如最先适应、最优适应或最差适应法等，此处不再赘述。

3.4.2 段式存储管理的地址转换和内存保护

- 段地址转换借助于段表完成。段表寄存器用来存放当前占用处理器的作业的段表始址和长度，查询段表得到每段在内存的起始地址，将段的起始地址加上段内偏移则得到物理地址，段式存储管理的地址转换和存储保护流程图3.23。

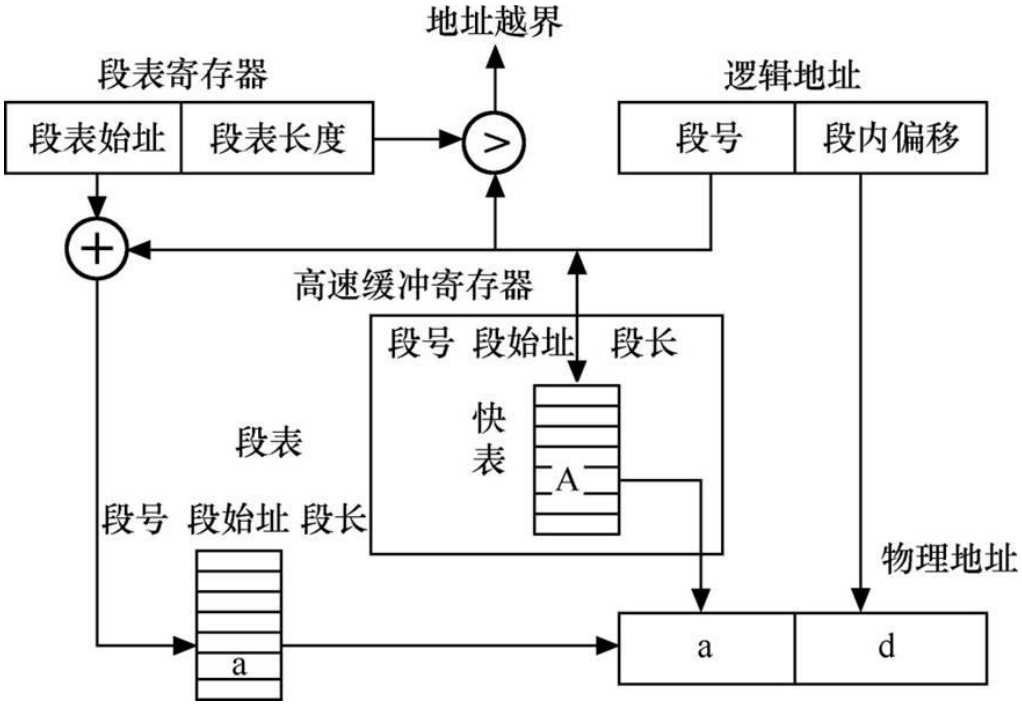


图3.23 段式存储管理的地址转换和存储保护

3. 4. 2 段式存储管理的地址转换和内存保护

段地址转换过程如下：

（1）将CPU给出的逻辑地址由地址转换机构分为段号和段内地址（即段内偏移）；

（2）查询段表寄存器得到段表在内存的起始地址；

（3）将段号带入查询段表，得到该段在内存的起始地址和段长；

（4）将段内地址与段长比较，如果段内地址大于段长，则发出越界中断；否则合法。段在内存的起始地址与段内地址相加，就为该逻辑地址对应的内存物理地址。

同样，借助于联想寄存器，在快表中保存最近使用过的段表项，可以更快地实现地址转换。

3. 4. 2 段式存储管理的地址转换和内存保护

- 利用段表寄存器中的段表长度与逻辑地址中的段号比较，若段号超过段表长则产生越界中断，再利用段表项中的段长与逻辑地址中的段内偏移比较，检查是否产生越界中断。

- 对段的越权保护可通过在段表中增加相应的存取控制权限字段来实现。权限字段显示对段的读、写是否允许，用它来检查对该段内地址的访问操作是否合法。只有当每次操作都在合法的权限范围内，才能正常完成访问操作，否则出错。

3.4.3 段的共享

- 在段式存储管理中，所谓段的共享，事实上就是共享分区，为此计算机系统要提供多对基址/限长寄存器。这样，几个作业所共享的例行程序就可放在一个公共的分区中，只要让各道的共享部分有相同的基址/限长值。
- 由于段号仅仅用于段之间的相互访问，段内程序的执行和访问只使用段内地址，因此不会出现页共享时出现的问题，对数据段和代码段的共享都不要求段号相同。当然对共享区的信息必须进行保护，如规定只能读出不能写入，欲想往该区域写入信息时将遭到拒绝并产生中断。

3.4.4 分段和分页的比较

- 段是信息的逻辑单位，由源程序的逻辑结构所决定，用户可见，段长可根据用户需要来规定，段起始地址可以从任何地址开始。在分段方式中，源程序(段号，段内偏移)经连结装配后仍保持二维结构。
- 页是信息的物理单位，与源程序的逻辑结构无关，用户不可见，页长由系统确定，页面只能以页大小的整倍数地址开始。在分页方式中，源程序(页号，页内偏移)经连结装配后变成了一维结构。

3.4.5 段页式存储管理

- 页式存储基于存储器的物理结构，存储利用率高，便于管理，但难以实现存储共享、保护和动态扩充；段式存储基于应用程序的结构，有利于模块化程序设计，便于段的扩充、动态链接、共享和保护，但往往会形成段之间的碎片，浪费存储空间。
- 可以把两者结合起来，在分段式存储管理的基础上实现分页式存储管理，这就是段页式存储管理，是目前应用最多的一种存储管理方式。

3. 4. 5 段页式存储管理

段页式存储管理的基本原理：

1．程序根据自身的逻辑结构划分成若干段，这是段页式存储管理的段式特征。

2．内存的物理地址空间划分成大小相等的物理块，这是段页式存储管理的页式特征。

3．将每一段的线性地址空间划分成与物理块大小相等的页面，于是形成了段页式存储管理。

4．逻辑地址分3个部分：段号、段内页号和页内位移，其形式为：

段号（s）	段内页号（p）	页内位移（d）

对于用户来说，虚拟地址应该由段号 s 和段内位移 d′ 组成，用户看不到如何分页。而是由操作系统自动把 d′ 解释成两部分：段内页号 p 和页内位移 d，也就是说，d′ =p×块长+d。

3.4.5 段页式存储管理

5．段页式存储管理的数据结构包括段表和页表。

段表中包括段号、该段页表的起始地址、页表长度等信息，页表中包括页号、对应的物理块号等信息。段表、页表和内存的关系如图3.24所示

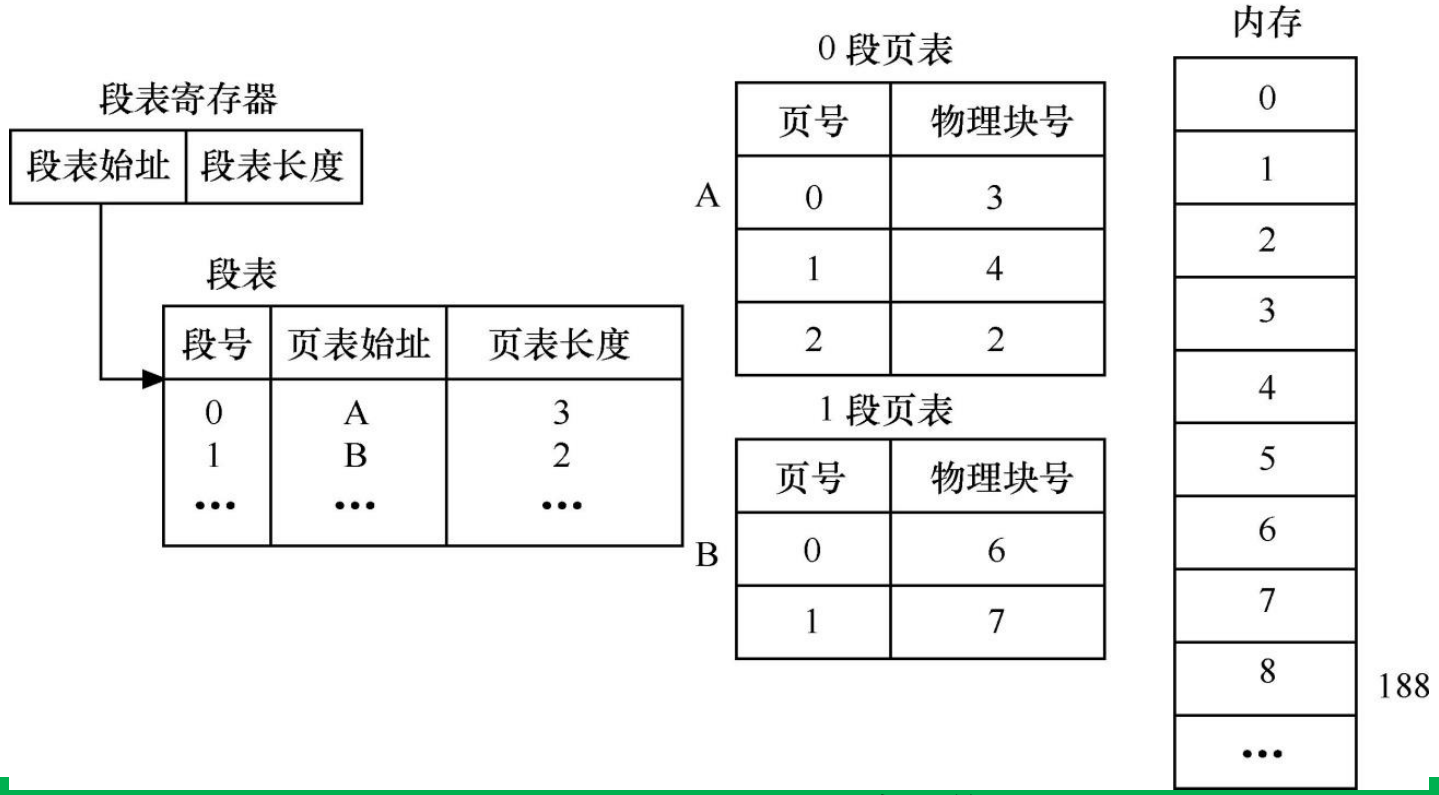


图3.24 段表、页表和内存的关系

3.4.5 段页式存储管理

6．动态地址转换

段页式存储管理的动态地址转换机构由段表、页表和快表构成。其动态地址转换过程如下：

操作系统把正在运行的作业的段表起始地址装入段表寄存器中，操作系统从逻辑地址中取出段号和段内页号，用段号作为索引查询快表中的段表，如果从快表得到页表起始地址和页表长度，再查询快表中的页表，从而得到所在页面对应的内存物理块号，将该物理块号与页内位移拼接，即为所需要的物理地址。如果不能从快表得到段和页的信息，则需要用段号作为索引查询内存中的段表，得到页表长度和页表的起始地址。再以页号作为索引查询页表，得到该页所对应的物理块号，同时将段表中的相应段信息和页表中的相应信息写入快表，并将物理块号与页内位移拼接，就可以得到物理地址。

3. 4. 5 段页式存储管理

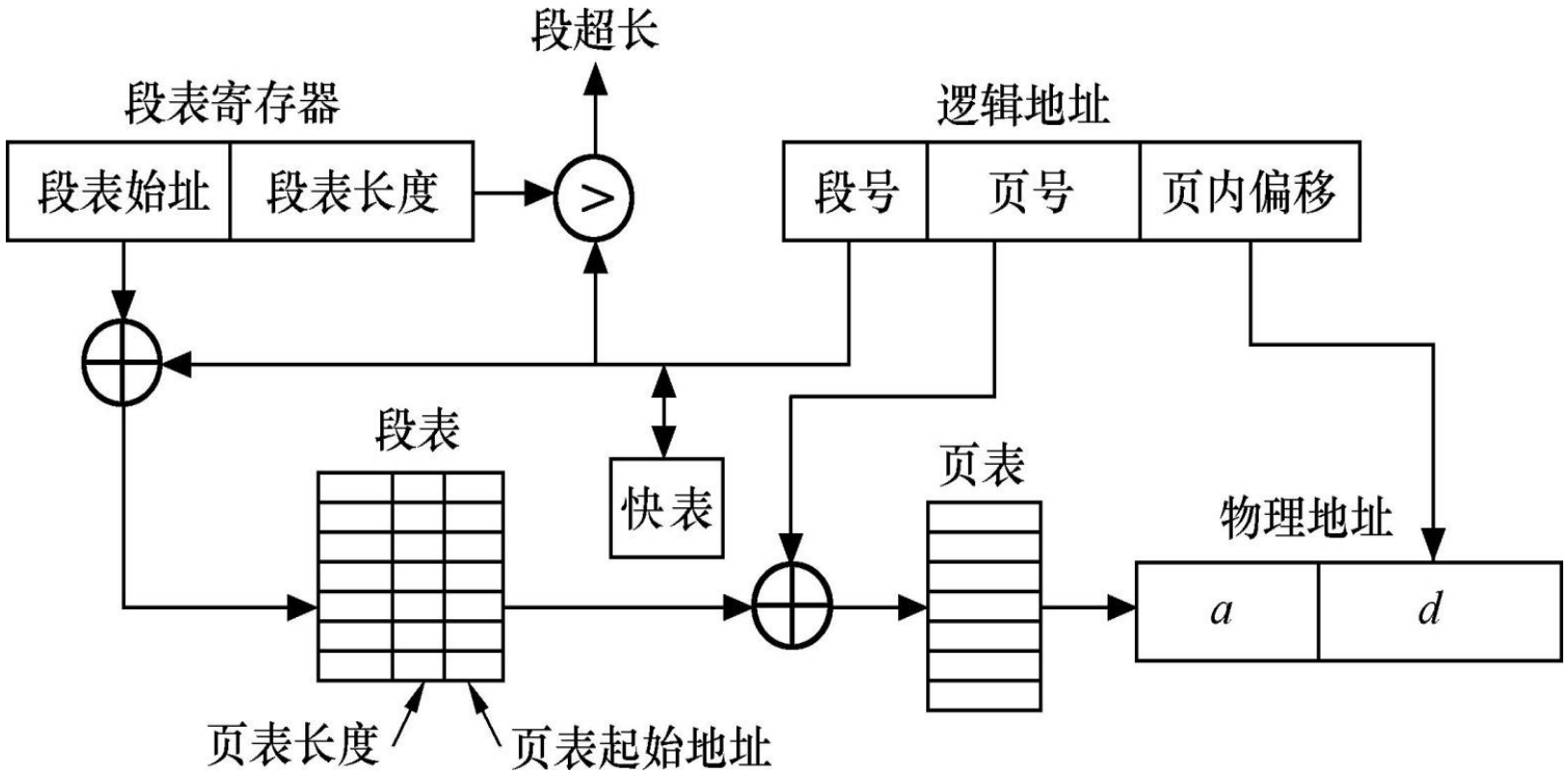


图 3.25 段页式存储管理地址转换和存储保护

3.5 虚拟存储技术

3.5.1 虚拟存储技术的提出

- 由于受到计算机体系结构和成本的限制，计算机的内存容量总是有限的。在传统的存储管理中，如果一个作业要运行，必须将该作业的全部信息都装入内存，并在整个作业运行结束后，才能释放内存。如果一个作业的信息大于内存容量，则无法装入内存，也无法运行；如果系统有大量的作业申请进入内存，则系统只能接纳相当有限的作业，系统的多道度和性能都难以得到提高。

3.5.1 虚拟存储技术的提出

- 但绝大部分的作业在执行时实际上不是同时使用这些信息的，作业的某部分信息，比如异常处理，可能从来不会使用；也可能运行完一次后，再也不会使用。既然作业的全部信息是分阶段需要，则可以分阶段将作业信息调入内存，而不需要一次将作业的全部信息调入内存。于是，提出了这样的问题：能否将作业不执行的部分暂时存放在外存，待到进程需要时，再将其从外存调入内存。将外存作为内存的补充，从逻辑上扩充内存，这就是虚拟存储技术。

3.5.2 程序的局部性原理

（1）顺序性

程序在运行时除了少部分的分支和过程调用指令外，大部分都是顺序执行。

（2）局限性

程序在运行时，如果有若干个过程调用，程序执行的轨迹会转移至调用区域，但过程调用一般由相对较少的指令组成。经过对大量的实例研究而发现：在大多数情况下，过程调用的深度不会很深，一般在5级以下。当程序在某个局部范围内运行时，系统可以只将相关的局部信息放入内存，其它不相关或暂时不相关的信息可以放在外存。

3.5.2 程序的局部性原理

（3）多次性

程序中的循环结构通常只由少数指令构成。这些集中在一起的少数指令被多次执行，在内存中可以只放入一个版本。

（4）独立性

程序中可能存在彼此互斥或相互独立的部分，每次运行时总有部分程序不被使用，没有必要将不被使用的部分放入内存。

总之，程序的局部性原理说明：程序的一次性装入内存与全部驻留内存都是不必要的。

3.5.3 虚拟存储技术的基本思想

- 虚拟存储技术的思想：将外存作为内存的扩充，作业运行不需要将作业的全部信息放入内存，将暂时不运行的作业信息放在外存，通过内存与外存之间的对换，使系统逐步将作业信息放入内存，最终达到能够运行整个作业，从逻辑上扩充内存的目的。

3.5.3 虚拟存储技术的基本思想

虚拟存储器定义：虚拟存储器是指具有请求调入功能和置换功能，能够从逻辑上对内存空间进行扩展，允许用户的逻辑地址空间大于物理内存地址空间的存储器系统。虚拟存储器的组织形式如图3.26所示。

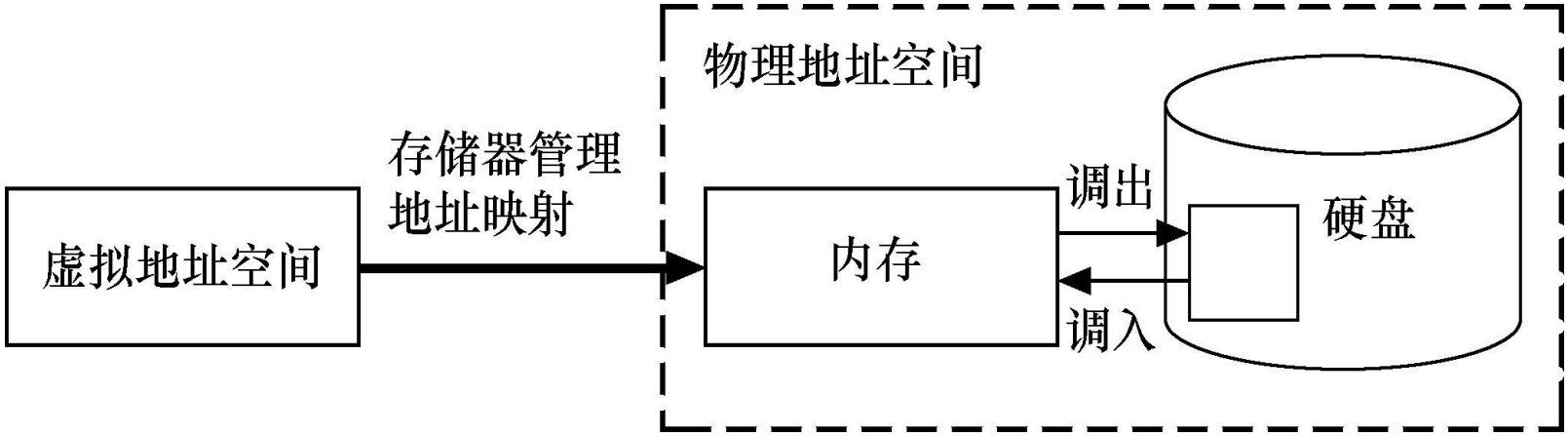


图3.26 虚拟存储器的组织形式

3.5.3 虚拟存储技术的基本思想

- 虚拟存储器的容量由计算机的地址结构和辅助存储器的容量决定，与实际的主存储器的容量无关。如果计算机系统的地址为32位，则可寻址的范围为0～4GB；如果计算机系统的地址为20位，则可寻址的范围为0～1MB。计算机系统的可寻址范围为虚拟存储器的最大范围。虚拟存储器的实现对用户来说是感觉不到的，他们总以为有足够的内存空间可容纳他的作业。
- 虚拟存储技术的实现基础是内存的分页或分段管理，采用的是进程的分页或分段在内存与外存之间对换。

3. 5. 3 虚拟存储技术的基本思想

- 虚拟存储技术允许进程的逻辑地址空间比物理内存空间大，即小空间能够运行大程序，打破了程序运行受内存空间的约束，使操作系统不但能够接纳更大的作业，而且还能接纳更多的作业，提高了系统的多道度和性能。

3.6 请求分页虚拟存储管理

3. 6. 1 请求分页虚拟存储管理的基本原理

请求分页虚拟存储管理是在页式存储管理的基础上增加了请求调页和页面置换功能，其基本原理如下：

1）首先，物理的内存空间被划分为等长的物理块，并对块编号。同时，用户程序也进行分页，这些与页式存储管理相同。

2）在用户程序开始执行前，不将该程序的所有页都一次性装入内存，而是先放在外存。当程序被调度投入运行时，系统先将少数页装入内存，随着程序运行，如果所访问的页在内存中，则对其管理与分页存储管理情况相同。

3.6.1 请求分页虚拟存储管理的基本原理

3）若发现所要访问的数据或指令不在内存中，就会产生缺页中断，到外存寻找包含所需数据或指令的页，并将其装入到内存的空闲块中。

4）在装入一页的过程中，若发现内存无空闲块或分配给该进程的物理块已用完，则需要通过页面置换功能从已在内存的页中挑选一个将其淘汰，释放所占用的物理块后将新的页面装入该块，进程继续运行。

5）被淘汰的页面如果刚才被修改过，则还需要将其回写到外存，以保留其最新内容。

3. 6. 1 请求分页虚拟存储管理的基本原理

请求分页虚拟存储管理与页式存储管理区别是增加了请求调页和页面置换等功能，需解决如下问题：

1）需要提供一个全新的页表机制来记录任一页是在内存或在外存的位置、是否被修改过等信息。

2）在请求分页虚拟存储管理方式下，内存中允许装入多个进程，每个进程占用一部分物理块，问题在于：为每个进程分配多少个物理块才合适？采用何种分配方式才合理？

3）进程运行过程中发现所要访问的数据或指令不在内存时，便会产生缺页中断，到外存寻找该页，此时，缺页中断如何处理？

4）一个页面或者是一开始就装入内存，或者是在运行中被动态的装入内存，如何进行地址重定位？

5）在动态装入一个页面时，若发现内存当前无空闲块或分配给该进程的物理块已经用完，则需要从已在内存的页面中选出一个将其淘汰。问题在于：在什么范围内选择要淘汰的页面？淘汰哪个页面？这就需要合适的页面置换算法。

3.6.2 请求分页虚拟存储管理的硬件支持

请求分页虚拟存储的硬件支持包括：请求分页的页表机制、缺页中断机构和地址转换机构。

1．请求分页的页表机制

在虚拟存储管理中，页表除了要完成从逻辑地址到物理地址的转换外，还需要提供页面置换的相关信息。因此，页表中除了有页号和物理块号等信息外，还增加了页的状态位、外存地址、修改位、访问字段等信息，如图3.27所示。

页号	物理块号	状态位	外存地址	修改位	访问字段

图3.27 页式虚拟存储管理的页表

3.6.2 请求分页虚拟存储管理的硬件支持

状态位： 用于标志一页是否已装入内存。如果该页已在内存，则该页所对应的状态位置“1”；否则置“0”。

外存地址： 页在外存中的地址。当需要将某页调入内存时，需要查询页表中的外存地址项得到该页在外存的地址，在外存找到该页。

修改位： 页在内存中是否被修改过的标志，用来确定如果该页被换出内存时，是否需要再回写入外存。如果该页在内存里没有被修改过，那么该页中的内容和在外存中的内容是一致的，则被换出内存时不需要再写入外存，节约了写入外存的时间。如果该页在内存中已经被修改过了，被换出内存时需要再写入外存。

3.6.2 请求分页虚拟存储管理的硬件支持

- ***访问字段：***标志页在内存时是否被访问过。用于进行页面置换时考虑是否将该页换出内存。如果该页被访问过，在进行页面置换时，系统会考虑该页可能以后会被再次访问而不将其换出。

3.6.2 请求分页虚拟存储管理的硬件支持

2．缺页中断机构

- 在进程运行过程中，当发现所访问的页不在内存时，缺页中断机构便产生一个缺页（Page fault）中断信号，操作系统接到此信号后，就运行缺页中断处理程序，将所需要的页调入内存。缺页中断与一般中断类似，都需要经历保护CPU环境、分析中断原因、转入中断程序处理、中断处理后恢复CPU环境等步骤。但缺页中断与一般中断也有不同：

3.6.2 请求分页虚拟存储管理的硬件支持

1) CPU检测中断的时间不同。对一般的中断信号，CPU是在一条指令执行完后检测其是否存在，检测时间以一个指令周期为间隔。而对缺页中断信号，CPU在一条指令执行期间，只要有中断信息就可检测，不需要等待一个指令周期。因此，CPU检测缺页中断更及时。

2) CPU可以多次处理。如果在一个指令周期中多次检测到缺页中断，CPU都会及时处理。

3. 6. 2 请求分页虚拟存储管理的硬件支持

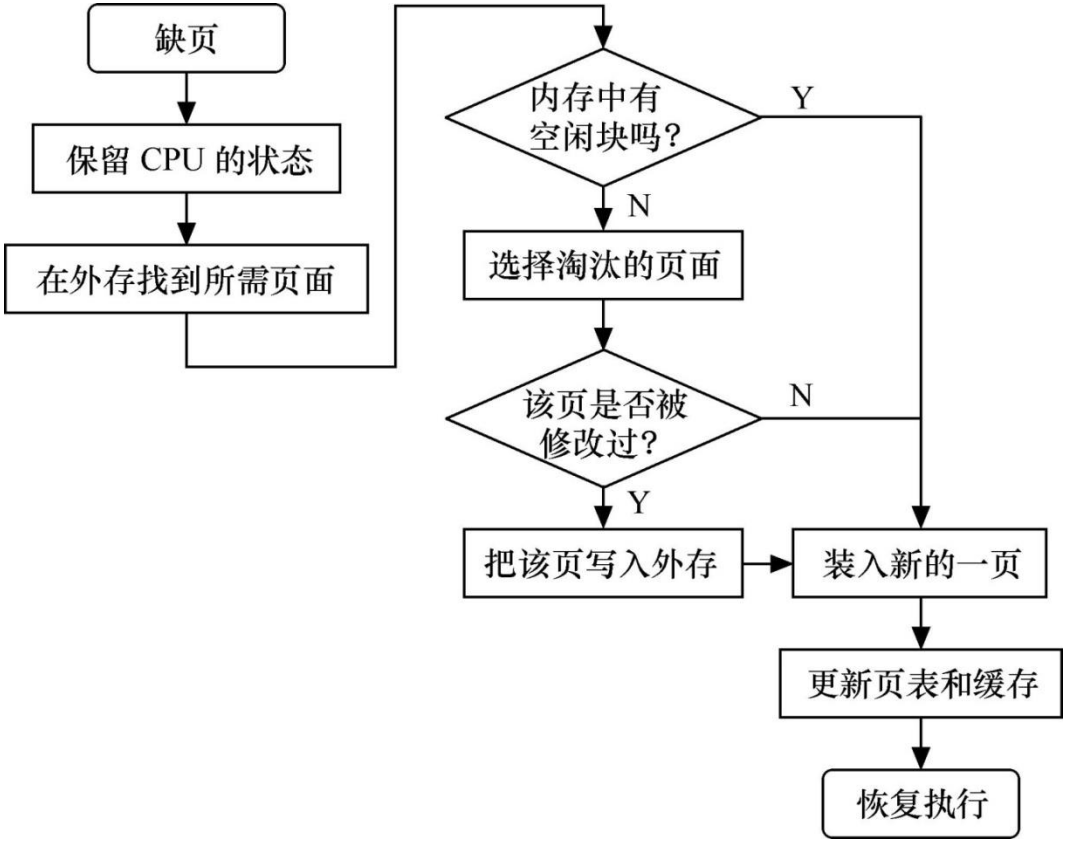


图3.28缺页中断的处理过程

3. 6. 2 请求分页虚拟存储管理的硬件支持

- 缺页中断处理流程：先查看内存是否有空闲块，若有则按该页在外存中的地址将该页找出并装入内存，在页表中填上它占用的块号且修改标志位。
- 若内存已没有空闲块，则必须先淘汰已在内存中的某一页，再将所需的页装入，对页表和内存分配表作相应的修改。
- 淘汰某页时，要查看该页的修改位来判断该页是否修改过，若该页在执行过程中没有被修改过，那么不必重新写回到存储器中，而已修改过的页调出时必须再将该页写回到外存中。

3.6.2 请求分页虚拟存储管理的硬件支持

- ***3. 地址转换机构***
- 请求分页虚拟存储技术是在程序执行过程中逐步将程序页面调入内存的，所以从逻辑地址到物理地址的转换是在程序运行过程中完成的，是动态重定位装入，地址变换机构如图3.29所示。

3. 6. 2 请求分页虚拟存储管理的硬件支持

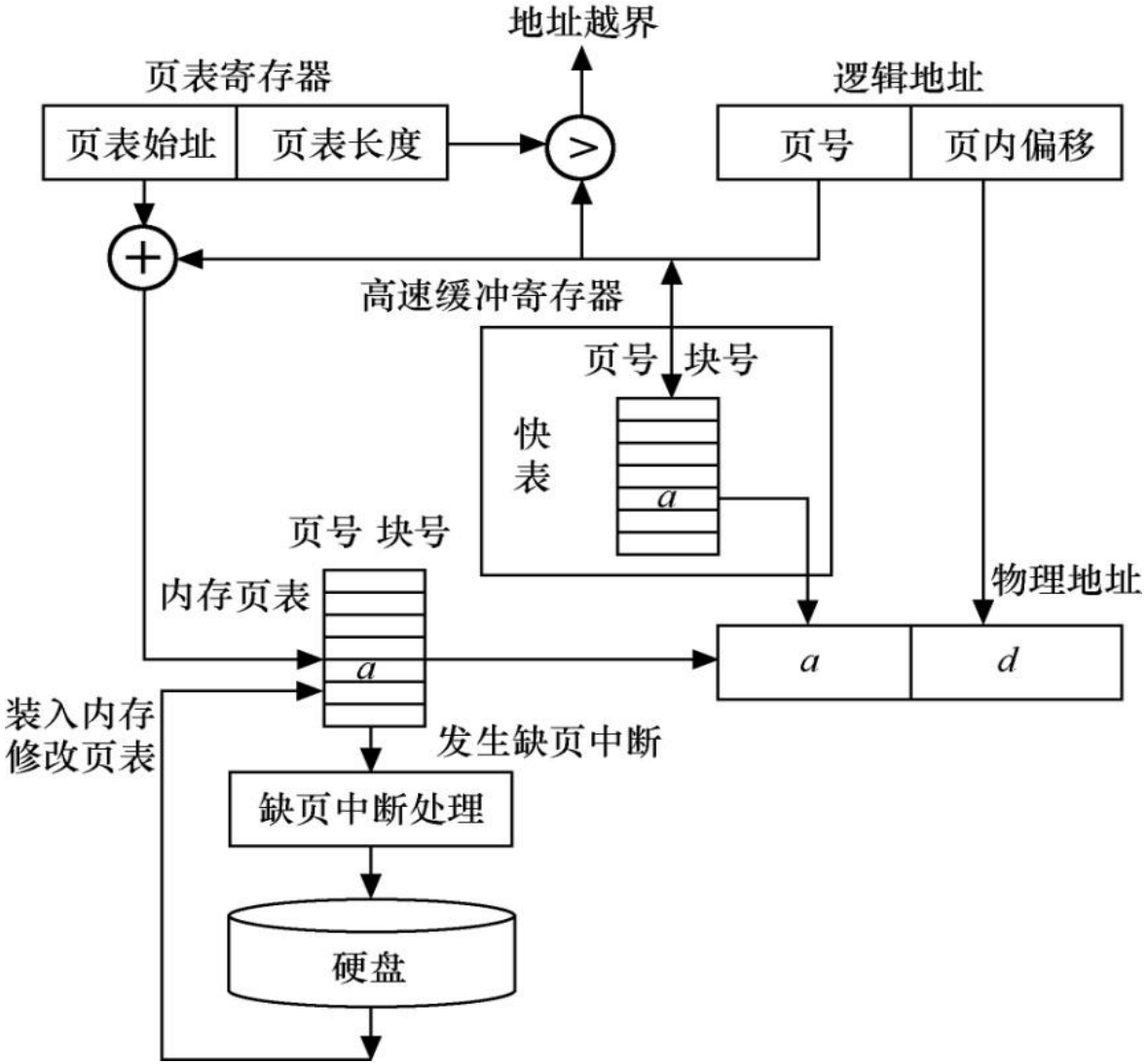


图3.29 请求分页虚拟存储的地址转换机构

3. 6. 2 请求分页虚拟存储管理的硬件支持

- 当进程被调度时，操作系统将进程PCB中的页表起始地址和长度装入页表寄存器中。
- CPU从逻辑地址中取得页号，根据页号查询快表，如果快表中有该页的内存块号，则将内存块号与页内偏移一起作为该页在内存的物理地址；如果快表中没有该页的内存块号，则CPU从页表寄存器得到页表在内存中的起始地址，查询页表，得到该页的内存块号，则将该块号与页内偏移一起作为该页在内存中的物理地址，同时将该页的页号和内存块号写入快表，以备下次查询时使用。如果查询页表而没有得到该页的内存块号，即表示该页不在内存，产生一个缺页中断信号，请求操作系统将该页调入内存。

3. 6. 2 请求分页虚拟存储管理的硬件支持

- 在调入一页的过程中，如果进程空间没有空余的物理块，则系统需要调出一页后再将该新页调入内存。同时系统修改页表，从页表中去除调出的页面，加上调入的页面。当这些工作完成后，处理器返回用户进程，继续执行被中断的指令。

3. 6. 3 页面分配策略与页面调度算法

请求分页虚拟存储管理支持多个进程同时装入内存，操作系统为各个进程分别分配部分物理块，并将各自的部分页调入内存，在实施中涉及到以下三个策略：

- 页面分配策略

  分配策略决定系统应该给一个进程分配多少物理块，进程才能运行。

- 页面调入策略

页面调入策略决定如何将进程所需要的页面调入到内存。

- 页面置换策略

页面置换策略决定内存中的哪些页面被换出内存。

3. 6. 3. 1 页面分配策略

- 页面分配负责为多个进程分配相应的物理块，分配时需要综合考虑系统的并发性、吞吐量和缺页中断率等因素，通常分为固定分配和可变分配两种不同的方式。

1. 固定分配方式

为每个进程分配固定数量的物理块，其数量在进程创建时，由进程的类型（交互性、批处理或应用性）或根据用户的要求确定，在进程的整个运行期间都不再改变。

具体的分配方式包括按进程平均分配法、进程按比例分配法和进程优先权分配法等。

3.6.3.1 页面分配策略

1）进程平均分配法

将内存中所有可分配的物理块平均分配给进入系统的各个进程。如果有m个内存物理块，n个进程，则每个进程分得的内存物理块数为m/n取整数（小数部分不计）。

该分配方法实现简单，但没考虑各个进程大小不一，常常会造成内存的浪费。

该分配方式另一个特点：分配给每个进程的物理块数会随着内存中进程数的多少而变化。

3. 6. 3. 1 页面分配策略

2）进程按比例分配法

其思想是：根据进程的大小，进程按照比例分配内存物理块数。如果系统中有m个内存物理块，n个并发进程，每个进程的页面数为Si。则系统中每个进程能够分得的内存物理块数bi为（bi取整）：

$$b_i = (mS_i)/(\sum_{i=1}^{n} S_i)$$

例: 如果内存能够提供62个内存物理块，并发进程有P1（10页）和P2（127页），则进程P1和P2分配到的物理块分别为:

$$b_1 = (62\times10)/(10+127) \approx 4, b_2 = (62\times127)/(10+127) \approx 57$$

3.6.3.1 页面分配策略

3）进程优先权分配法

高优先级的、时间要求紧迫的进程，操作系统给其分配较多的内存物理块，使其能够快速完成。在实际应用中，将按比例分配法和进程优先级结合起来考虑，系统把内存中可以分配的物理块分为两部分，一部分按照比例分配给各并发进程，另一部分根据进程的优先级进行适当增加。

3.6.3.1 页面分配策略

2. 可变分配方式

可变分配方式是指分配给进程的物理块数，在该进程的运行期间可以动态变化。它用来解决由于事先分配给进程的物理块太少而导致的频繁缺页问题。

具体实现时，先为每一进程分配必要数量的物理块，使之可以开始运行，系统中余下的空闲物理块组成一个空闲物理块队列，当某一进程在运行过程中发生缺页时，系统从空闲物理块队列中取出一个空闲块分配给该进程。只要该空闲队列中还有物理块，凡发生缺页的进程都可以才该队列中申请并获得物理块。

3. 6. 3. 1 页面分配策略

3. 进程的最小物理块数

最小物理块数是保证进程正常运行所需要的最小内存块数。进程需要的最小物理块数与计算机的硬件结构有关，取决于计算机的指令格式、功能和寻址方式。如果计算机采用单地址指令的直接寻址方式，则只需要用于存放指令的页面和存放数据的页面，最小物理块数为2；如果采用间接寻址方式，则至少需要3个物理块。对于功能较强大的计算机，指令长度可能会超过多个字节，指令本身需要跨过多个页面，则物理块的最小需要数会更大。

3.6.3.2 页面调入策略

页面调入策略有两种：

- **请求页（demand paging）调入**
- **预先页（prepaging）调入**

3.6.3.2 页面调入策略

1.请求页调入

请求页调入简称请调，是指在CPU需要访问进程某页面时，发现所访问的页面不在内存，CPU发出缺页中断信号，请求将该页调入内存。操作系统接收到缺页中断请求后，为之分配物理块并从外存将该页调入内存。

每个进程在刚开始执行时，所需的页面很多，会产生多次缺页中断，页面被逐一调入内存。根据程序的局部性原理，当进程运行一段时间后，所需要的页面会逐步减少，缺页中断次数会逐渐下降，最后趋向于很低的水平，进程运行进入相对平稳阶段。

请求页调入策略的优点是只有在需要时才将页面调入内存，节省了内存空间。

3.6.3.2 页面调入策略

请求页调入策略的缺点：

1）在进程初次执行时，开始阶段会有大量的页调入内存，这时的进程切换开销很大。

2）发生缺页中断时操作系统会调入页面，而每次又仅调入一个页面，启动磁盘作I/O的频率很高。由于每次启动磁盘时会产生一个时间延迟，因此，会造成系统用于I/O的时间增长，系统效率降低。

3）对于执行顺序跳跃性大的程序，缺页情况变化大，难以趋向稳定的水平，从而引起系统不稳定。

3.6.3.2 页面调入策略

2. 预先页调入

预先页调入简称预调，是指由操作系统根据某种算法，预先估计进程可能要访问的页面，并在处理器需要访问页面之前先将页面预先调入内存。

该调入策略的优点是一次可将多个页面调入内存，减少了缺页中断的次数和I/O操作次数，系统付出的开销减少。如果预先动态估计准确率高，该调入策略会大大提高系统效率。

3.6.3.2 页面调入策略

该调入策略的缺点：

1）如果预先动态估计准确率较低，调入的页面不被使用的可能性大，系统效率较低。

2）如果程序员不能预先提供所需程序部分的信息，则该调度策略难以实施。

在实际应用中，页面调入会将请求页调入和预先页调入结合起来。在进程刚开始执行时或每次缺页中断时，采用预先页调入。在进程运行稳定后，如果发现缺页，系统可采用请求页调入。

3.6.3.3 页面置换策略

当需要从外存调入页到内存时，如果当前内存没有空闲物理块，则操作系统需要将某些页置换出内存，再将新的页面换入内存。选择被置换出的页有两种策略：全局置换和局部置换

1. 全局置换

全局置换是指操作系统从所有当前位于内存的页面中选择一个页面淘汰，释放出对应的物理块，而不是仅从需要该页的进程的物理块换出。这种置换方法会影响大多数进程的运行，是一种动态方法。

3.6.3.3 页面置换策略

2. 局部置换

局部置换是指当某进程有页面需要换入到内存时，只能从该进程目前已在内存的页面中选择一页淘汰，该置换方法对其它进程没有影响。

- 局部置换与全局置换比较有明显的缺点：如果进程在执行期间所需要的内存物理块数发生变化，页面置换发生频繁，即使系统有空闲的物理块，也不可能增加给该进程；另外，如果系统给某进程分配的物理块数太多，系统不会收回，最终会造成内存空间浪费。

3.6.3.4 页面置换与页面分配的关联

1. 固定分配局部置换

为每个进程分配固定数量的物理块，在进程的整个运行期间都不再改变。当一个进程运行中发生缺页中断时，操作系统只从该进程在内存中的页面中选择一页淘汰。

该策略不足在于：应为每个进程分配多少物理块数难以确定。如果分配给进程的物理块太少，缺页中断率高，进而导致整个多道程序系统运行缓慢；给多了，会使内存中能同时执行的进程数减少，进而造成处理器空闲和其他设备空闲，浪费资源。

3.6.3.4 页面置换与页面分配的关联

2. 可变分配全局置换

先为每一进程分配必要数量的物理块，使之可以开始运行，系统中余下的空闲物理块组成一个空闲物理块队列，当某一进程在运行中发生缺页时，系统从空闲物理块队列中取出一个空闲块分配给该进程。直到系统拥有的空闲物理块耗尽，才会从内存中选择一页淘汰，该页可以是内存中任一进程的页面。

该策略易于实现，且可以有效地减少缺页中断率，是采用得较多的一种分配和置换策略。

3.6.3.4 页面置换与页面分配的关联

3. 可变分配局部置换

新进程装入内存时，根据应用类型、程序要求，先分配给一定数目物理块。当产生缺页中断时，系统只能从产生缺页中断的进程的页面中选一个页面淘汰，不能影响其他进程的运行。操作系统要不时重新评价进程的物理块分配情况，增加或减少分配给进程的物理块以改善系统总的性能。

3.6.4 页面置换算法

请求分页虚拟存储管理规定，当需要从外存调入一个新的页面时，如果此时物理内存无空闲块，系统必须按照一定的算法选择内存中的一些页面调出，并将所需的页面调入内存，这个过程叫页面置换。页面置换算法决定从内存中置换出哪一个页面。

衡量页面置换算法的重要的指标是缺页率。

3. 6. 4 页面置换算法

- 一个进程或一个作业在运行中成功的页面访问次数为S，即所访问的页面在内存中；
- 不成功的页面访问次数为F，即访问的页面不在内存，需要缺页中断并调入内存；
- 需要访问的页面的总次数为A = S + F。
- 则缺页率f为：f = F / A

3.6.4 页面置换算法

影响缺页率的因素如下：

1）进程分得的内存物理块数越多，缺页率越低。

2）划分的页面越大，缺页率越低。

3）如果程序局部性好，则缺页率低。

4）如果选取的置换算法优，则缺页率低。

在进程的内存物理块数和页面大小不能改变的情况下，要减少缺页率，就要考虑选择合适的页面置换算法。

3.6.4 页面置换算法

1. 先进先出（FIFO）页面置换算法

先进先出（Fist In First Out）算法的基本思想：总是选择最先进入内存的页面或驻留时间最长的页面先淘汰。该算法最早出现，易于实现。

实现：可以将所有页面按照进入内存的次序排成一个队列，设置一个替换指针指向队头的一页。当需要进行页面淘汰时，替换指针指向的即为当前最先进入内存的页面，该页被淘汰，然后修改指针指向淘汰页后一个页面即可，调入的新的页面排入队尾。

3. 6. 4 页面置换算法

- 假如：某进程的页面访问序列为：6 0 1 2 0 3 0 5 2 3 0 3 2 1 2 0 1 1 6 0 1，操作系统为该进程分配了三个内存物理块。FIFO页面置换过程如图3.30所示。

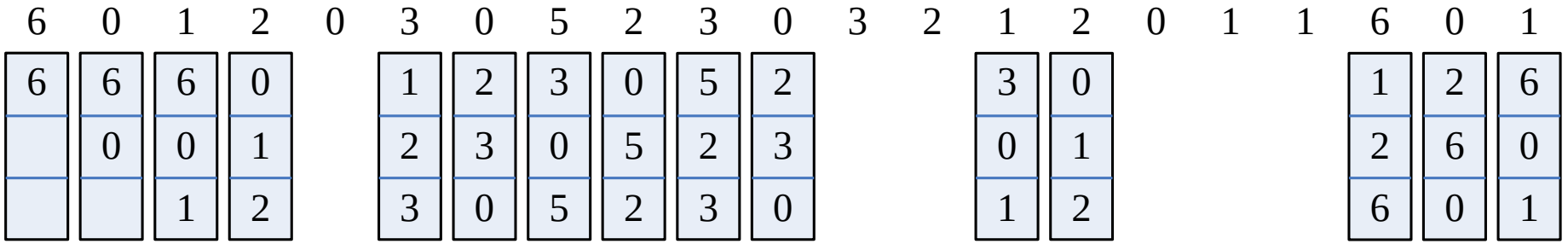


图3.30 FIFO页面置换算法

3. 6. 4 页面置换算法

- 按照FIFO页面置换算法，缺页12次（最先进入的3个页面是正常调入，不算缺页调入），缺页率为12/21。
- 先进先出页面置换算法开销低、容易编程实现，适合于线性顺序特性好的程序。但是该算法没有考虑到页面的访问频率，很可能刚被换出的页面马上又要被访问，使得缺页率偏高。

3. 6. 4 页面置换算法

- 为了改善FIFO算法，减少缺页率，科学家尝试在进程发生缺页时给进程增加物理块。在实验中，Belady发现了一个奇怪的现象，该现象也被称为Belady异常现象。即：当页数在一定范围内时，缺页率反而随分配给进程的物理块数的增加而增加，如图3.31所示。当内存物理块数从4增加到6时，缺页率增加了，该现象说明，如果要改善系统性能，不能只靠给进程增加内存物理块。

3.6.4 页面置换算法

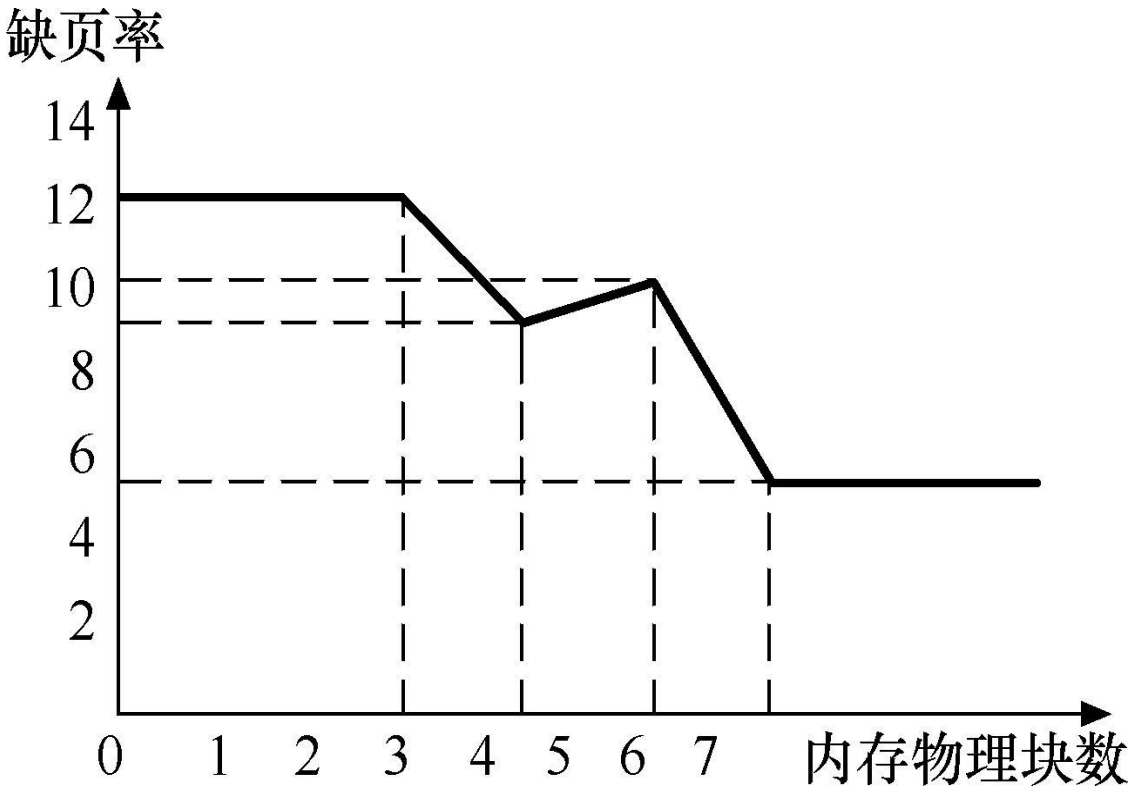


图3.31 Belady异常现象

3. 6. 4 页面置换算法

2.最佳（OPT）页面置换算法

- 最佳（optimal）页面置换算法由Belady在1966年提出，基本思想：在选择页面置换时应该考虑该页面将来使用的情况，将来最长时间不用的页面被淘汰。在进程采用固定页面分配的情况下，最佳页面置换算法具有最低的缺页率。

3.6.4 页面置换算法

- 假如：某进程的页面访问序列为：6 0 1 2 0 3 0 5 2 3 0 3 2 1 2 0 1 1 6 0 1，操作系统为该进程分配了三个内存物理块。OPT页面置换过程如图3.32所示。

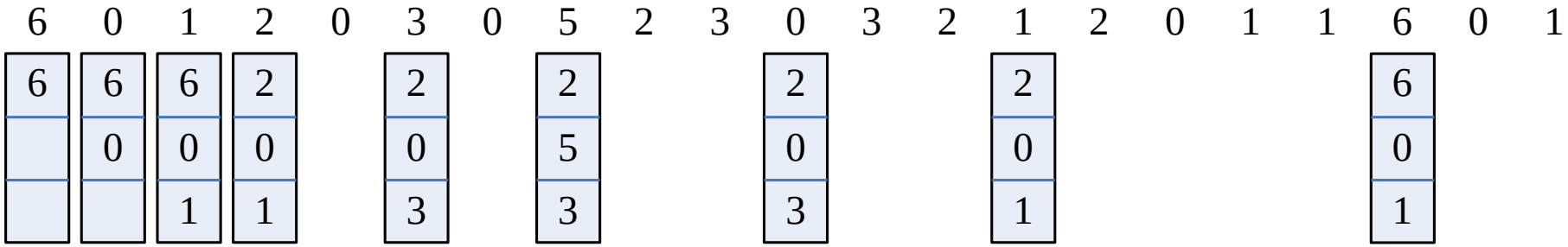


图3.32 OPT页面置换算法

3. 6. 4 页面置换算法

- 按照最佳页面置换算法，缺页6次，缺页率为6/21。
- 最佳页面置换算法的难点在于很难准确预知进程在内存的页面哪些会在未来最长时间不会被访问。因此，最佳页面置换算法只是一种理想化的页面调度算法，很难实现。
- 该算法可以作为评判其它的置换算法的准则。

3. 6. 4 页面置换算法

- **3.6.4 最近最久未使用页面置换算法**
- 实现方法：系统须维护一个页面淘汰队列，该队列中存放当前在内存中的页号，每当访问一页时就调整一次，使队尾总指向最近访问的页，而队列头部就是最近最少用的页，发生缺页中断时总淘汰队列头所指示的页；而执行一次页面访问后，需要从队列中把该页调整到队列尾。

3. 6. 4 页面置换算法

- 假如：某进程的页面访问序列为：6 0 1 2 0 3 0 5 2 3 0 3 2 1 2 0 1 1 6 0 1，操作系统为该进程分配了三个内存物理块。LRU页面置换过程如图3.33所示。

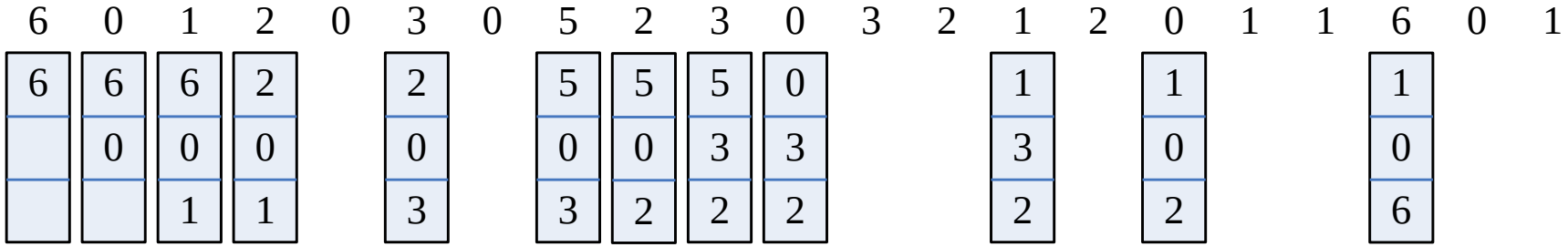


图3.33 LRU页面置换算法

3. 6. 4 页面置换算法

- 按照LRU页面置换算法，缺页9次，缺页率为9/21。
- LRU算法能够合理地预测程序运行状态，具有很好的置换性能，被公认为是一种性能好且可以实现的页面置换算法，但是LRU算法在实现起来比较复杂。

3. 6. 4 页面置换算法

4. 时钟（clock）置换算法

- 时钟置换算法的基本思想是：
- 1）将内存中所有的页面组织成一个循环队列，形成一个类似于时钟表面的环形表，循环队列指针类似于钟的指针，用来指向可能被淘汰的页面，指针开始时指向最先进入内存的页面，如图3.34所示。
- 2）时钟置换算法需要在页表中为每一页增加一个访问位R。当页面首次装入内存时，R的初值设置为“0”。当某个页面被访问过后，R的值被设置为“1”。

3. 6. 4 页面置换算法

3）选择淘汰页面的方法是从指针当前指向的页面位置开始扫描时钟环，如果某个页面页表中的R为“1”，表明该页被访问过，将R清“0”，并跳过该页；如果某个页面页表中的R为“0”，表明该页没有被访问过，该页被淘汰，指针推进一步；如果所有的页面都被访问过，指针绕环一圈，将所有页面的R清“0”，指针回到起始位置，选择该页淘汰，指针推进一步。

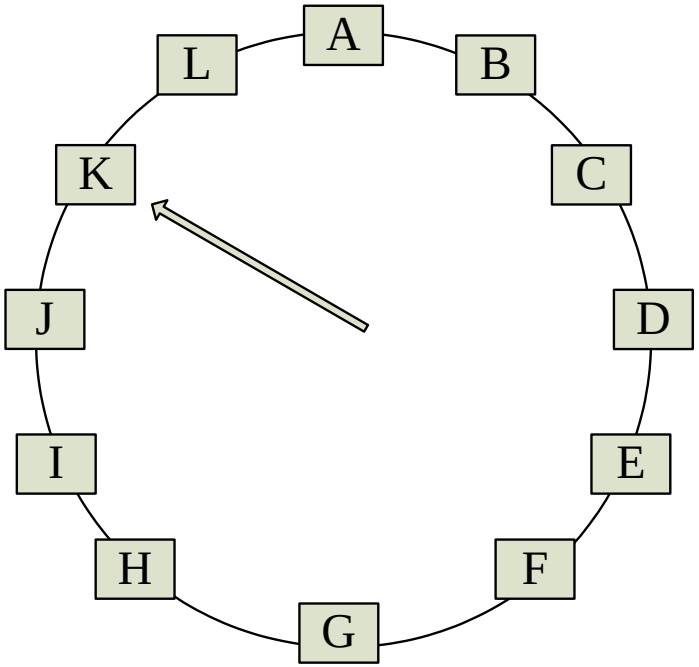


图3.34 时钟置换算法

3. 6. 4 页面置换算法

- 为了提高置换效率，在页面置换时，如果被淘汰的页面没有被修改过，则不需写回外存。这样，将页表中的访问位R和页表中的修改位M配合，产生改进的时钟置换算法。

3. 6. 4 页面置换算法

访问位R和修改位M的组合有下面四种情况：

1．最近没有被访问（R = 0），没有被修改（M = 0）

从指针当前位置开始，选择第一个R = 0，M = 0的页面淘汰。

2．最近没有被访问（R = 0），但是被修改过（M = 1）

如果没有R = 0，M = 0的页面，则从开始位置重新开始查找第一个R = 0，M = 1的页面并淘汰之。

3．最近被访问（R = 1），没有被修改（M = 0）

如果没有R = 0，M = 1的页面，表示所有的页面R = 1。在再次回到开始位置时，所有页面的访问位R都被清“0”。从开始位置查找第一个M = 0的页面并淘汰之，此时不用将淘汰页面“写”回外存。

4．最近被访问（R = 1），被修改（M = 1）；

当前面三种情况都不存在时才考虑R = 1，M = 1的情况。

3. 6. 4 页面置换算法

- 该策略的主要优点是没有被修改过的页面会被淘汰，但不必写回外存，节省了I/O时间。但是查找淘汰页面可能需要多次扫描时钟，增加了算法的开销。
- Macintosh操作系统就采用了这种既考虑访问位，又考虑修改位的改进的时钟页面置换算法。

3.6.5 影响请求页式存储管理性能的因素

1．分配给进程的内存块数与缺页率的关系

- 一般说来，分给进程的物理块数越多，缺页率越小。例如，若某进程逻辑地址共需30个页面，取极端情况，为其分配30个物理块。则所有页面全部进入内存，缺页率自然为0。不过此时请求页式管理已变成了页式管理。如果取另一个极端，即只分给进程一个物理块，只能容纳下一页，这种情况下毫无疑问会频繁地发生缺页中断，缺页率最大。
- 试验结果表明，对每个进程来说，为其分配进程空间页面数约一半的物理块时，请求页式的效果最好。

3.6.5 影响请求页式存储管理性能的因素

2．页面大小对系统性能的影响

1）从页表大小考虑。

如果页面较小，页数就要增加，页表也随之扩大，为了控制页表所占的内存空间，应选择较大的页面尺寸。

2）从内存利用率考虑。

内存以块为单位，一般情况下进程的最后一个页面总是装不满一个物理块，会产生内部碎片，为了减少内部碎片，应选择小的页面尺寸。

3.6.5 影响请求页式存储管理性能的因素

3）从读写一个页面所需的时间考虑。

作业存放在辅助存储器上，从磁盘读入一个页面的时间包括等待时间（移臂时间+旋转时间）和传输时间，通常等待时间远大于传输时间。显然，加大页面的尺寸，有利于提高 I/O 的效率。

3. 6. 5 影响请求页式存储管理性能的因素

- 综合考虑以上几点，现代操作系统中，页面大小大多选择在 512 B到 4KB之间。
- 如：Atlas为512B、IBM370系列机为2048B或4096B、VAX为512 B、IBM AS/400为512B、Intel x86为4096B、MIPS R4000 提供从4096 B至16MB节共7种页面长度。
- 页面长度是由 CPU 中的MMU 规定的，操作系统通过特定寄存器的指示位来指定当前选用的页面长度。

3. 6. 5 影响请求页式存储管理性能的因素

3. 缺页率对系统性能的影响

用p表示缺页率，如果p = 0，则不缺页；如果p = 1，则始终缺页。

抖动：由于缺页而引起的一种系统现象，即处理器频繁地处理页面的换出和调入，使得处理器实际处理程序的能力大大减小。“抖动”现象常在缺页率非常高时发生。

用st表示缺页处理时间。缺页处理时间包括从外存取相关页面并将其放入内存的时间。

用ma表示对内存一个页面的访问时间。

3. 6. 5 影响请求页式存储管理性能的因素

用vt表示有效访问时间。

在非缺页的情况下，vt = ma

在缺页率为p的情况下，vt= (1 – p) × ma + p × st

在任何情况下，缺页处理时间由下面三个主要部分构成：

（1）缺页中断服务时间；

（2）读页面时间；

（3）恢复进程时间。

3. 6. 5 影响请求页式存储管理性能的因素

- 缺页中断服务和恢复进程所花费的时间位于1ms-100ms之间，设备寻道时间为15ms，磁盘延迟时间为8 ms，传输时间为1ms，则包括硬件和软件在内的整个缺页处理时间st最少为25ms。
- 内存访问时间ma为100ns。
- 则有效访问时间vt为：

$$vt = (1 - p) \times ma + p \times st$$
$$= (1 - p) \times 100 + p \times 25\,000\,000$$
$$= 100 + 24\,999\,900 \times p(ns)$$

可见，有效访问时间直接正比于缺页率。

3. 6. 5 影响请求页式存储管理性能的因素

如果缺页率为1/1 000，则在此概率下有效访问时间约为25us，而没有缺页时的页面访问时间仅为0.1us，可见缺页造成有效访问时间增加很多。

在实际应用中，缺页不只使得缺页的进程运行减慢，还会影响到其他进程的运行。如果一个进程队列阻塞等待某个设备，而该设备正用于一个缺页的进程，则等待设备的进程会等待更长的时间才能得到请求的设备。

可见，缺页不只使得缺页进程本身的运行减慢，还会使得整个系统的运行效率降低，系统性能下降。

3.7 请求分段虚拟存储管理

3.7.1 请求分段虚拟存储管理的基本原理

- 请求分段的基本思想：将用户程序的所有段首先放在辅助存储器中，当用户程序被调度投入运行时，首先把当前需要的一段或几段装入内存，在执行过程中访问到不在内存的段时再把它们从外存装入。
- 请求分段的段表包括：段号、段长、存取权限、在内存中的起始地址、在外存中的起始地址、是否在内存、修改标志、共享标志和扩充位等。

3.7.1 请求分段虚拟存储管理的基本原理

- 在程序执行中需要访问某段时，查段表，若该段在内存，则按段式存储管理进行地址转换得到绝对地址。
- 若该段不在内存中，则硬件发出一个缺段中断。操作系统处理这个中断时，先查找内存分配表，找出一个足够大的连续区域容纳该分段。如果找不到足够大的连续区域则检查空闲区的总和，若空闲区总和能满足该分段要求，那么进行适当移动后，再将该段装入内存。若空闲区总和不能满足要求，则可调出一个或几个分段在辅助存储器上，再将该分段装入内存。

3.7.1 请求分段虚拟存储管理的基本原理

- 在执行过程中，有些数据段的大小会随输入数据多少而变化。这就需要在该分段尾部添加新信息，但添加后的段的总长度不应超过硬件允许的每段最大长度。对于这种变化的数据段，当要往其中添加新数据时，由于欲访问的地址超出原有的段长，硬件首先会产生一个越界中断。操作系统处理这个中断时，先去判断该段的“扩充位”标志，如可以扩充，则允许增加段的长度，如该段不允许扩充，那么这个越界中断就表示程序出错。

3. 7. 1 请求分段虚拟存储管理的基本原理

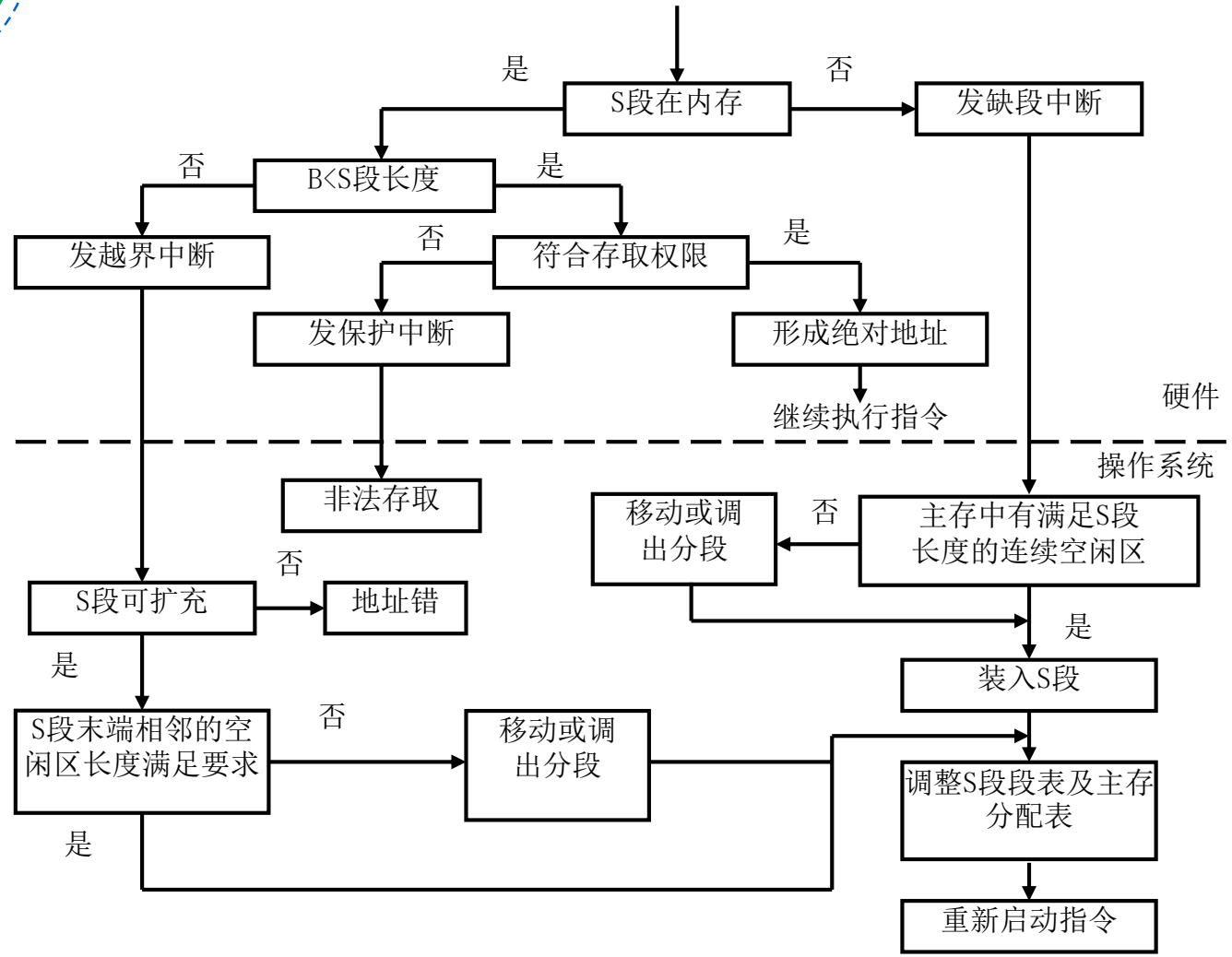


图3.35 分段式存储管理的地址转换和存储保护

3.7.2 请求分段虚拟存储管理的段的共享和保护

- 请求分段虚拟存储管理为了实现段的共享，除了原有的进程段表外，还要在系统中建立一张段共享表，每个共享分段占一个表项，每个表项包含两部分内容：
- 第一部分包含共享段名、段长、内存起址、状态位（如在不在内存）、辅存地址、共享进程个数计数器。
- 第二部分包含共享该段的所有进程名、状态、段号、存取控制位（通常为只读）。

3.7.2 请求分段虚拟存储管理的段的共享和保护

- 当出现第一个要使用某个共享段的进程时，操作系统为此共享段分配一块内存，再将共享段装入该区。同时将共享段在内存的起始地址填入共享段表中对应项的内存始址处，共享进程个数计数器加1，修改状态位为1（在内存），填写使用该共享段的进程的有关信息（进程名、使用共享段的段号、存取控制等）。而进程段表中共享段的表项指向内存共享段表地址。此后，当又有进程要使用该共享段时，仅需直接填写共享段表和进程段表，以及把共享进程个数计数器加1就可以了。

3.7.2 请求分段虚拟存储管理的段的共享和保护

- 当进程不再使用共享段时，应释放该共享段，除了在共享段表中删去进程占用项外，还要把共享进程个数计数器减1。共享进程个数计数器为0时，说明已没有进程使用此共享段了，系统需要回收该共享段的物理内存，并把占用表项也取消。
- 这样做的优点：不同进程可以用不同段号使用同一个共享段；由于进程段表中共享段的表项指向内存共享段表地址，所以，每当共享段被移动、调出或再装入时，只要修改共享段表的项目，不必要修改共享该段的每个进程的段表。

3.7.2 请求分段虚拟存储管理的段的共享和保护

- 由于每个分段在逻辑上是独立的，实现存储保护很方便：

一是越界检查。在段表寄存器中存放了段长信息，在进程段表中存放了每个段的段长。在存储访问时，首先，把指令逻辑地址中的段号与段表长度进行比较，如果段号等于或大于段表长度，将发出地址越界中断；其次，还需检查段内地址是否大于段长，若是的话将产生地址越界中断，从而确保每个进程只在自己的地址空间内运行。

二是存取控制检查。在段表的每个表项中，均设有存取控制字段，用于规定此段的访问方式，通常设置的访问方式有：只读、读写、只执行等。

3. 7. 3 请求段页式虚拟存储管理

- 请求段页式虚拟存储管理是在段页式存储管理的基础上增加了用以实现虚拟存储的缺页中断机制、缺段中断机制来实现的。
- 与传统的段页式存储管理一样，用户的逻辑地址空间被划分为段号、段内页号与页内偏移。但是，请求段页式管理并没有将一个作业的所有段在作业运行前全部装入内存，只是部分段装入内存，因此，还需要有作业表来记载进入内存的作业段的情况。

3.7.3 请求段页式虚拟存储管理

- 作业表中登记了进入系统中的所有作业及该作业的段表的起始地址等信息，段表中至少包括该段是否在内存、该段页表的起始地址等信息，页表中包括该页是否在内存、对应的物理块号等信息。
- 请求段页式虚拟存储管理的动态地址转换机构由段表、页表和快表构成。在地址转换过程中如果所访问的段内页在内存中，则对其处理与段页式存储管理的情况相同。如果不能从内存段表中查询出所需要的段，则表示该段不在内存，系统发出请求调段中断信号。如果段表中存在该段信息，但是没有所在的页面信息，则表示该页不在内存，系统发出请求调页中断信号。

3.7.3 请求段页式虚拟存储管理

- 同样，如果发出请求调段或请求调页后，如果没有内存空间，则需要在内存和外存之间进行置换。置换算法思想与页面置换算法思想类似。

3 3.8 Windows 7内存管理技术

- Windows 7与Windows 2000/XP一样主要运行在与Intel Pentium CPU兼容的硬件平台。Intel Pentium CPU提供三种工作模式：实地址模式（real mode）、虚地址模式（又称为保护模式，protection mode）和虚拟模式（virtual mode）。
- 实地址模式采用段式存储器管理或单一连续存储器管理，不启用分页机制，只能寻址1MB地址空间。DOS操作系统采用这种模式。

3 3.8 Windows 7内存管理技术

- 虚地址模式采用三种内存管理方式：段式虚拟存储器管理、页式虚拟存储器管理和段页式虚拟存储器管理。Linux和Windows操作系统采用这种模式。
- 虚拟模式是在保护方式下的实地址模式的仿真，允许多个8086应用程序在386以上CPU中运行。

3.8.1 基于分页管理的Windows 2000/XP/7

- Window 2000/XP采用请求页式虚拟存储器管理，提供32位的虚拟地址，为每一个进程提供一个受保护的4G虚拟地址空间。虚拟地址空间布局为低2G的地址空间为用户程序区，高2G的地址空间为操作系统区，如图3.36所示。

图3.36 Windows系统虚拟存储器地址布局

3.8.1 基于分页管理的Windows 2000/XP/7

- 系统区又分为固定页面区、页交换区和操作系统驻留区。固定页面区中存放关键的系统代码，页面不可与外存对换；页交换区存放非常驻系统代码和数据，可以与外存进行页面对换；操作系统驻留区存放操作系统内核、执行体和引导驱动程序以及硬件抽象代码层，非常重要永不失效，为了加快运行速度，这一区的寻址由硬件直接映射。
- 另外，在操作系统引导时，也可以选择另一种地址分配方式：3GB用户程序区和1GB操作系统区。这种情况主要用于运行大的用户程序。

3.8.1 基于分页管理的Windows 2000/XP/7

- **1．页表**
- 在Windows 2000/XP系统中的页表如图3.37所示。

页框号	保留	全程符	修改	访问	禁用	缓存	通写	所有者	写	有效

图3.37 Windows 2000/XP系统中的页表

3.8.1 基于分页管理的Windows 2000/XP/7

2．逻辑地址到物理地址的变换

在Windows 2000/XP系统中的32位逻辑地址被划分为：页表目录索引、页表页索引和页面。其中页表目录索引占10位，页表页索引占10位，页面占12位。

页表机制为2级页表，页面大小为4K。分页管理中采用了二级页表结构实现进程的逻辑地址到物理地址的变换，如图3.38所示。

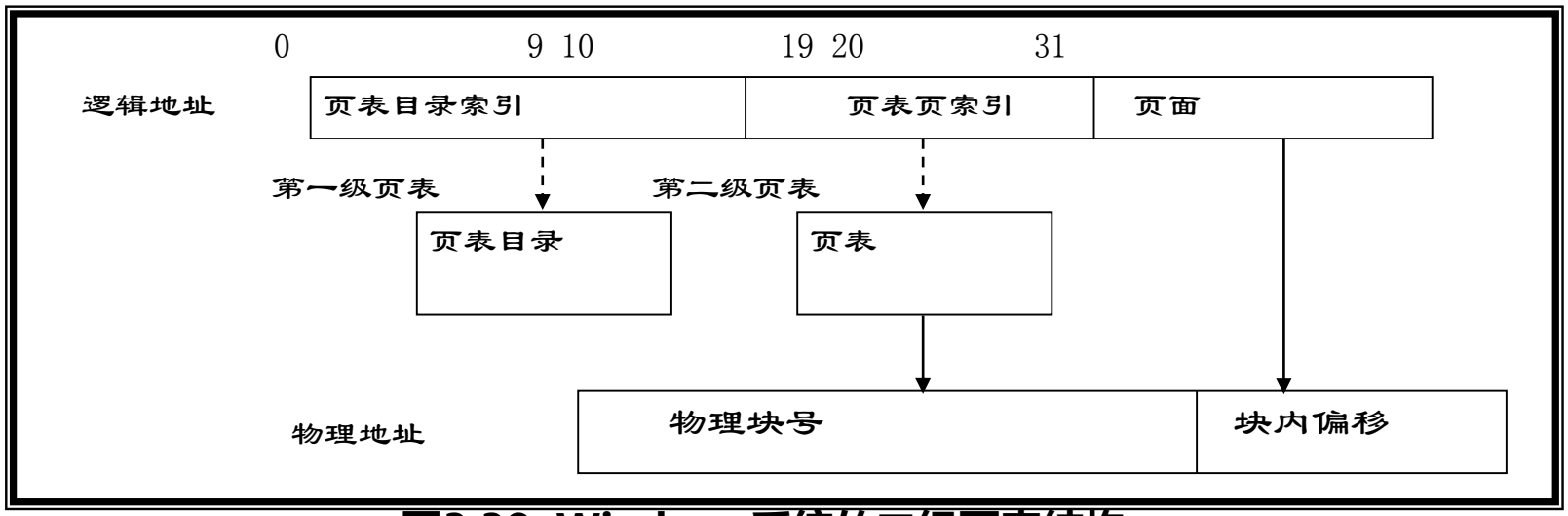


图3.38 Windows系统的二级页表结构

3.8.1 基于分页管理的Windows 2000/XP/7

- 页目录用来指向进程页表地址。每个进程都拥有自己的页目录。
- 在地址变换时，操作系统从运行进程的进程控制块中得到进程页目录的起始地址，并将该地址放入页目录寄存器中。通过页目录寄存器寻址到页目录。

3.8.1 基于分页管理的Windows 2000/XP/7

- 页目录的目录项中包含有进程所有页表的位置和状态，即通过页目录得到页表。由于页表占用10位地址，故最多有1024张页表。每张页表最长可以达到1024个页表项，指向1024个页面。用户进程最多可以占用512个页表项。
- 页表中包括内存物理块号、修改位、访问位、有效位、禁用缓冲标志和进程的拥有者标志符等信息。因此，通过查询页表得到逻辑地址中的页面号对应的物理块号，最后物理块号与页内偏移一起构成物理地址。

3.8.2 Windows 2000/XP/7系统的虚拟存储管理实现

1．请求调页

当发生缺页时，系统首先检查所需页是否在后备链表或修改链表中。如在，则将其移出，放入进程的工作集中，不再分配新的物理块；若不在，如果需要一个零初始化页，则内存管理程序在零页链表中取出第一页。如果零页链表为空，则从空闲链表中取出一页并对它进行初始化。如需要的不是零初始化页，就从空闲页表中取出第一页。如果空闲链表为空，就从零初始化页中取出一页。如果以上任一情况中零页链表和空闲表均为空，那么使用后备链表。

2．页面淘汰算法与工作集

- Windows 2000/XP系统采用请求页调入和预调入两种调页方式。当一个线程发生缺页时，内存管理器引发中断的页面及后继的少量页面一起装入内存。为了防止进程损失太多内存，系统采用局部FIFO算法淘汰页面。
- Windows 2000/XP的进程工作集为进程当前在内存中的页面集合。当创建一个进程时，系统为其指定最小工作集和最大工作集。开始时，所有进程缺省工作和的最小和最大工作集是相同的。系统初始化时，会计算一个进程的最小和最大工作集值，当物理内存大于32MB时，进程缺省最小工作集为50页，最大工作集为345页。在进程执行过程中，内存管理器会对进程工作集大小进行自动调整。

3.8.2 Windows 2000/XP/7系统的虚拟存储管理实现

- 当一个进程的工作集降到最小后，如果该进程再发生缺页中断，并且内存并不满，系统会自动增加该进程的工作集尺寸。
- 当一个进程的工作集升到最大后，如果没有足够的内存可用，该进程每发生一次缺页中断，系统都要从该进程工作集中淘汰一页，再调入所请求的页面。如果有足够内存可用，系统允许一个进程的工作集超过它的最大工作集尺寸。

3.8.2 Windows 2000/XP/7系统的虚拟存储管理实现

- 当物理内存剩余不多时，系统将检查内存中的每个进程，查看其当前工作集是否大于其最大工作集，如果是，则淘汰该进程工作集中的一些页，直到空闲内存数量足够或每个进程都达到最小工作集。
- 为了测试和调整进程当前工作集的合适尺寸，系统会定时从进程中淘汰一个有效页，观察其是否会发生缺页中断。如果进程没有发生缺页中断，则该进程工作集减1，并回收该页框到空闲链表中。
- 所以，Windows 2000/XP的虚拟内存管理总是为每个进程提供可能好的性能，而无需用户或系统管理员的干预。

3．盘交换区

- Windows 2000/XP可以支持多达16个盘交换文件(页面调度文件)。

3.8.3 Windows 2000/XP/7的内存空间分配

1．Windows 2000/XP系统用户空间分配

Windows 2000/XP系统中有三种应用程序内存管理方法。

（1）**虚页内存分配**：适合管理大型对象数据或动态结构数组。

在虚页内存分配中，内存管理器用一组虚地址描述符（virtual address descriptor）来描述每个进程的虚地址空间状态，包括地址范围、该地址范围是共享还是私有、子进程能否继承该地址范围、地址范围内应用于页面的保护限制等。为了快速查找虚地址，虚地址描述符构造成一棵二叉树，如图3.39所示。

3.8.3 Windows 2000/XP/7的内存空间分配

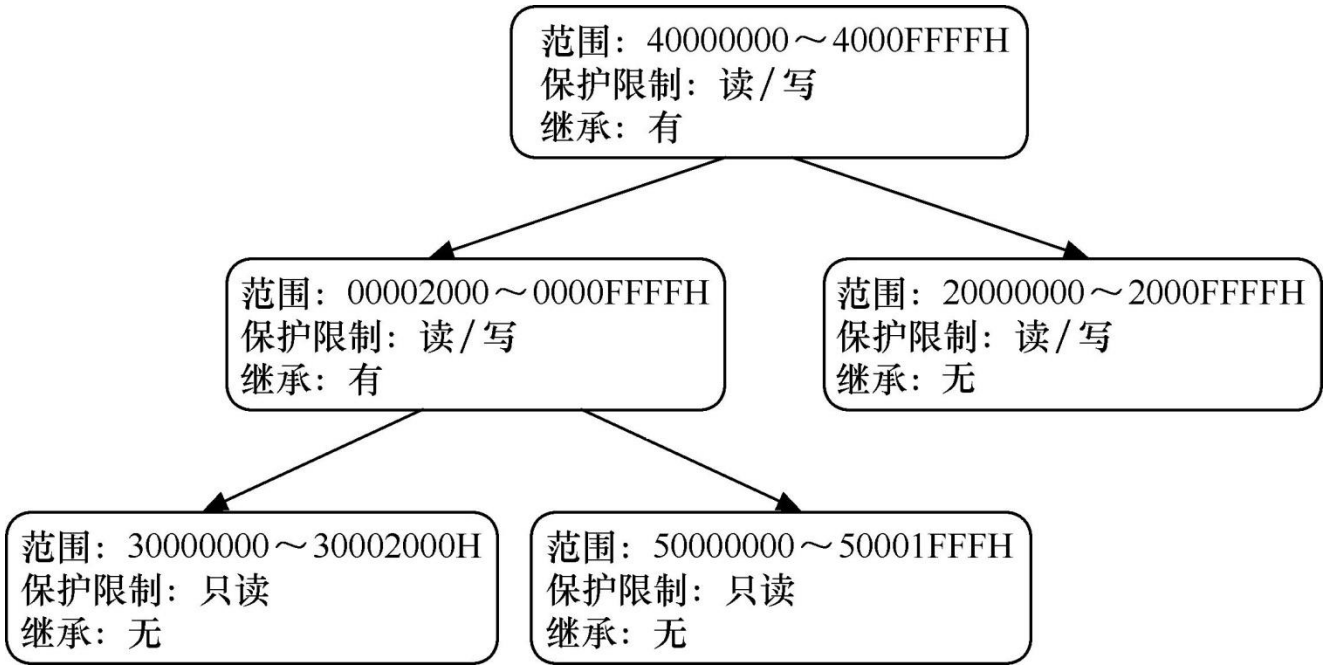


图3.39 Windows系统虚地址描述二叉树

3.8.3 Windows 2000/XP/7的内存空间分配

当进程保留地址空间或映射一个内存区域时，内存管理器创建一个虚地址描述符来保存分配请求提供的信息。当线程首次访问一个地址时，内存管理器必须为包含此地址的页面创建一个页表项，并将虚地址描述符中相应的虚地址空间状态写入页表项中。如果该地址没有落在虚地址描述符所覆盖的地址范围，或所在的地址范围仅被保留而没有被提交，内存管理器则会知道该线程试图使用之前没有分配的内存，将产生一次访问违规。

3.8.3 Windows 2000/XP/7的内存空间分配

- (2) **内存映射文件**：适合管理大型数据流文件及实现多个进程之间的数据共享。
- 内存映射文件表示可以被两个进程或多个进程共享的内存块。一个进程中的一个线程可以创建并命名一个内存映射文件，其他的进程能够打开这个内存映射文件的句柄。一旦内存映射文件的句柄被打开，一个线程能把该内存映射文件映射到自己或另一个进程的虚地址空间中。系统利用内存映射文件将可执行映像和动态链接库装入内存，高速缓存管理器利用内存映射文件访问高速缓存文件中的数据，内存映射文件将一个文件映射到进程地址空间，然后，可以像访问内存中的一个大数组一样访问该文件，而不是对文件进行读写。

3.8.3 Windows 2000/XP/7的内存空间分配

（3）**内存堆分配**：适合管理大量小对象数据。

为了让更多的进程能够访问更多的物理内存空间，Windows 2000/XP提供了一个视窗（address windowing extension）机制来给内存映射文件空间中的一部分空间保留虚地址，这部分空间是进程当前需要的。所映射的这一部分称为该内存映射文件的一个视窗。映射一个内存映射文件的视窗就是使该内存映射文件的一部分能在虚地址空间中可见。相反，删除映射一个内存映射文件的视窗就是把它从进程的虚地址空间中删去。进程可以通过对一个内存映射文件多次开设和删去视窗，达到在较小的虚地址空间中处理较大物理内存空间的目的。

3.8.3 Windows 2000/XP/7的内存空间分配

- 例如，一个有8G物理内存空间的高端Windows 2000/XP数据库服务器上，应用程序可以利用视窗来分配和使用将近8GB内存作为数据库缓存。视窗对于多于2GB或3GB物理内存的系统最有用，因为对于32位的进程，这是唯一的可直接使用多于2GB或3GB内存的方式。使用视窗函数来实现内存分配首先是分配将要使用的物理内存，创建一个虚地址空间作为视窗来映射内存映射文件，把内存映射文件映射到视窗。

3.8.3 Windows 2000/XP/7的内存空间分配

一个进程可以对一个内存映射文件开一个或多个视窗，不同进程可以对同一内存映射文件开设视窗。视窗机制使用在以下情况：当基于内存映射文件的区域对象远远大于虚地址空间时；或基于盘对换的区域远远大于交换空间时；或共享时。Windows 2000/XP提供了出色的内存共享机制，共享的进程通过创建“文件映射对象”作为共享的内存区域。当两个进程对同一文件映射对象建立视窗时，系统就发生了对文件映射对象的共享。不同的进程的视窗在区域对象空间中的位置可以相同也可以不同。

3.8.3 Windows 2000/XP/7的内存空间分配

2．Windows 2000/XP系统中应用程序内存管理方法

（1）虚页内存分配

在Windows 2000/XP中使用虚拟内存要分三个阶段：保留内存、提交内存和释放内存。

- 保留内存

用户为了向系统表明将要用较大的内存空间，可向虚拟内存管理申请保留一段连续的虚拟内存地址空间，并指明该段空间的大小。其它进程或本进程的其他线程不能使用这段地址空间。但是，目前这段空间并不马上使用，只在需要时才使用。

3.8.3 Windows 2000/XP/7的内存空间分配

- **提交内存**

  提交内存这里指提交物理页面。在已保留的区域中，物理页面在第一次实际访问该虚拟地址时才提交，也可以在保留内存的同时提交。

- **释放内存**

  当进程不再需要被提交的内存或保留的地址空间时，可以回收盘交换区或从进程虚拟地址空间中释放虚拟地址。

3.8.3 Windows 2000/XP/7的内存空间分配

（2）内存映射文件

内存映射文件主要用于：

- Windows执行体使用内存映射把可执行文件.exe和动态连接库.dll文件装入内存，节省应用程序启动所需要的时间。
- 进程使用内存映射文件来存取磁盘文件信息，从而可以减少文件输入/输出，且不必对文件进行缓存。
- 多个进程可使用内存映射文件来共享内存中的数据和代码。

3.8.3 Windows 2000/XP/7的内存空间分配

3）内存堆分配

堆是保留地址空间中一个或多个页组成的区域，内存堆分配是由堆管理器按照更小块划分和分配内存的技术。

3.8.4 Windows 2000/XP内存页面级保护机制

Windows 2000/XP提供了4种保护机制防止用户破坏其它进程或操作系统。

（1）区分内核态和用户态，只有核心态线程才能访问核心态下组件使用的数据结构和内存缓冲池，用户态线程不能访问核心态下组件使用的数据结构和内存缓冲池。

（2）系统通过虚拟地址映射机制保证每个进程有独立、私有的虚拟地址空间，禁止其他进程的线程访问。

3.8.4 Windows 2000/XP内存页面级保护机制

（3）以页面为单位的保护机制，页表中包含了页级保护标志，如只读、读写等，以决定用户态和核心态可访问的类型，实现访问监控。

（4）以对象为单位的保护机制，每个区域对象具有附加的标准存取控制，当一个进程试图打开它时，系统会检查存取控制，以确保该进程是否被授权访问对象。

3.8.5 Windows7内存管理的新特点

1．Superfetch技术

Windows7的内存管理采用了Superfetch技术，即“超级预读取”，这一技术是由WindowsXP系统中的Prefetch发展而来。

Windows XP操作系统对虚拟内存技术做了进一步改进，发展出了预取技术（Prefetch），预取技术的基本思路是：在载入某个程序之前，预先从硬盘上中载入一部分该程序运行所需的数据到物理内存中，这样便能加快程序的启动速度。

3.8.5 Windows7内存管理的新特点

- 在Widows XP中，使用预取技术的具体方法是：在系统和应用程序启动时，监视内存页面与交换文件以及硬盘上其它文件的数据交换状况，当发生数据交换时，Windows XP会纪录下每一个程序运行时经常需要读取的硬盘文件，并将读取的情况记录在\windows\Prefetch目录中的pf后缀名文件中。

3.8.5 Windows7内存管理的新特点

- 一旦建立了这些pf文件，在每次需要启动系统或相应程序的时候，Windows会首先中断当前准备载入的程序，而转去查找\Windows\Prefetch目录，看是否有当前载入程序的纪录，如果有纪录，则马上按照纪录的情况载入程序运行过程中可能会用到的所需文件到物理内存中。这项任务完成之后，Windows才继续载入被中断的程序。
- 经过这样的处理之后，在程序运行过程中，需要读取那些文件时，由于文件已经被“预取”到内存中，此时就不用再到硬盘上进行读取，因此减轻了程序载入过程中频繁交换内存页面与交换文件的现象，改善了内存不足时程序运行的响应速度。

3.8.5 Windows7内存管理的新特点

- 为了进一步优化预取操作的效率，Windows XP还会定期对pf文件进行分析处理，组织好程序文件载入的顺序，并将这些分析处理后的信息存放在\Windows\Prefetch目录中的Layout.ini文件中。同时还会通知磁盘碎片整理程序，在下次运行碎片整理时，按照Layout.ini文件记录的内容，将相关文件的位置整理在连续的硬盘区块中。
- 这项技术从根本上说仍然属于被动式的调度。换句话说，只有在程序主动发起载入请求时，Windows才会进行相关的调度操作。

3.8.5 Windows7内存管理的新特点

- 被动式调度的存在可能对系统性能造成一些影响。例如：在工作的午休时间运行杀毒软件。此时Windows XP系统会将工作程序所占用的内存页面写入硬盘交换文件中，并读取杀毒软件的文件载入内存。午休过后，杀毒软件已经运行完毕，但是在重新开始使用工作程序的时候，系统仍然需要经历杀毒软件和工作程序的硬盘交换文件与内存页面的交换过程，此时程序的响应速度明显降低。
- 如果系统能够进一步自动记录下这些经常性的操作行为所发生和结束的时间，当时运行的前台和后台软件等等详细情况，那么在内存有空闲空间的时候，就可以在预定的时机预先将一部分文件载入到内存中，这样就避免了上面例子中的情况。

3.8.5 Windows7内存管理的新特点

- Windows7中Superfetch技术解决了这一问题，Superfetch技术不但继承了Windows XP预取技术的全部优点，还进一步具备监视程序运行时状况，时间等详细情况的功能，可以根据用户的使用习惯，自动预先将存放在硬盘的交换文件转换到内存页面中去，使用户经常运行的程序启动时的速度得到进一步的加快。
- Superfetch技术的思想是：“过分空余的内存空间即是浪费”。如果一个操作系统总是保留着过多的空余物理内存耗费电能，却不能够利用这些多余的内存空间提高系统性能的话，为什么不更好地利用这些多余的内存空间呢？将这些多余的物理内存作为缓存使用，就是Superfetch技术的本质。

3.8.5 Windows7内存管理的新特点

- **2．ReadyBoost 技术**
- 当系统的内存不够大，所以无法全面的发挥Superfetch的功能时，Windows7的ReadyBoost技术可以帮助解决这个问题。ReadyBoost功能是利用闪存的容量作为Superfetch预加载页面的储存空间。由于采用了特殊的算法，所以并不会影响闪存的寿命。用于ReadyBoost的闪存最好为物理内存大小1~2.5倍。如果系统物理内存足够大的话，就没有必要使用ReadyBoost了。

3 小结

- 存管理和虚拟存储技术是操做系统最重要的功能之一。
- 内存管理分为连续管理和离散管理。连续管理分为单一连续管理和分区管理。离散管理分为页式存储管理、段式存储管理和段页式存储管理。
- 连续管理中的单一连续管理是最简单的内存管理方式，该方式只适合单道程序环境。分区存储管理适合多道程序环境。在分区存储管理中，可变分区分配算法包括首次适应法、循环首次适应法、最优适应法、最坏适应法和快速适应法。

3 小结

- 页式存储管理采用了对进程的逻辑地址空间分页，对内存的物理空间分块，页的大小等于块大小等基本思想，通过页表和地址转换机构实现逻辑地址到物理地址的变换，能够有效地利用内存空间。
- 段式存储管理的实现思想与分页存储管理相似。分段存储管理体现了程序设计思想，易于实现段的共享和保护。
- 段页式存储管理将分段与分页结合，发挥了分页和分段存储器管理的优势。
- 虚拟存储管理是为解决内存扩充问题而提出，程序运行不需要调入程序的全部信息到内存便可运行，使得计算机能够运行更大、更多的用户作业，提高了系统的吞吐量。

3 小结

- 虚拟存储管理包括：请求分页虚拟存储管理、请求分段虚拟存储管理和请求段页式虚拟存储管理。
- 为了实现虚拟存储管理思想，是将内存中的页面或分段与外存中的页面或分段进行置换。主要的页面置换算法有先进先出页面置换算法、最佳页面置换算法、最近最久未使用页面置换算法、时钟置换算法等。本章在讲述虚拟存储管理的基础上，从实际应用出发，以Windows操作系统为实例介绍了存储器管理思想的实现。