第 4 章　存储管理

存储管理是操作系统的重要组成部分，它负责管理计算机系统的存储器，存储器可分成内存储器（简称内存，又称主存）和辅助存储器（简称辅存，又称外存）两类。本章讨论内存储器的管理。

内存储器的存储空间一般分为两部分：一部分是系统区，存放操作系统以及标准子程序、例行程序等；另一部分是用户区，存放用户的程序和数据等。存储管理主要是对内存储器中的用户区域进行管理。

- 存储管理的目的和功能
- 覆盖和交换技术
- 虚拟存储技术
- 缓冲存储器
- 存储管理机制
- Windows Server 2008 的内存管理

4.1　存储管理的目的和功能

程序在内存中的位置只有在装入执行时才能确定，因而在编写程序时不能使用绝对地址，用户按逻辑地址编写程序，操作系统为程序分配一个内存空间时，逻辑地址和分配到的内存空间的绝对地址不一致，而处理机执行指令按绝对地址进行，所以操作系统必须把逻辑地址转换绝对地址，处理机才能得到指令的位置。把逻辑地址转换成绝对地址的过程称为地址转换（又称重定位）。重定位有两种：静态重定位、动态重定位。静态重定位是指作业在装入过程中进行的地址转换，需要由专门设计的重定位装入程序完成，不需要硬件地址变换结构，实现起来比较简单，但要求给作业一个连续的存储空间，在作业执行期间不能移动，难以实现内存共享和内存的扩充。动态重定位是指在作业执行过程中，当访问指令或数据时才进行的地址变换，需要硬件地址变换机构实现，实现存储管理的软件算法比较复杂。但动态重定位使内存的使用更加有效，便于实现动态链接，内存的共享和内存的扩充能够得以实现。

当内存中存在多个程序时，为了防止程序访问和写入错误，多个程序间应进行保护，并能用合理而有效的方法进行调度。

程序和数据只有存放在 CPU 能直接访问的内存中，这个程序才能够执行。尽管 RAM 芯片的集成度不断提高，价格不断下降，但内存容量仍是计算机系统中的主要“瓶颈”资源，为使用户编写程序时不受内存容量的限制，操作系统采用“扩充”技术使用户使用比实际内存容量大的存储空间。

总之，存储管理的目的是尽可能地方便用户和提高内存的效率。存储管理主要完成如下的四个功能。

（1）内存的分配和管理。程序执行时，向操作系统申请内存空间，由存储管理实施具体的分配。存储管理使用分配记录表表示内存的使用情况和状态，根据程序申请内存的大小按一定的策略分配给程序，并把使用情况记录在分配记录表中，从而完成内存的分配。如果不能满足申请内存，则申请程序将等待，直到有足够的内存空间进行分配。

存储管理接受系统和程序释放的存储区，或主动回收不再使用的存储区，并修改分配记录表，完成内存的回收。

（2）内存空间的共享。内存空间的共享是为了提高内存空间的利用效率，一方面使多道程序能动态地共享内存，另一方面使多道程序能共同使用某些程序段和数据。

（3）存储保护。存储保护是保证多道程序都在自己所属的存储区内操作，互不干扰。防止用户程序的错误破坏系统程序，也要防止多道程序之间的相互干扰和破坏。一般由硬件提高保护功能，软件配合实现。

（4）存储扩充。存储扩充所指的扩充不是内存储器硬件上的扩充，而是利用存储管理为用户提供一个比实际内存更大的存储空间。这样，用户编制程序时，不需考虑实际内存空间的容量。就好像使用一个足够容量的内存储器一样，大大方便用户的使用。

4.2 覆盖和交换技术

覆盖和交换技术都是实现“扩充”内存的存储管理技术，主要用来解决在较小的存储空间中运行大作业时遇到的矛盾。既可用于一个作业内部，也可实施于多个作业之间，以达到多道或分时运行的目的。它们通常和分区存储管理技术配合使用。

4.2.1 覆盖技术

当一个作业要求运行时，若它所需的存储空间得到满足，系统就一次把它装入内存，一直在内存中驻留，直到运行结束。若它在内存中一次装不下，就需解决作业在内存中装不下的问题，较早采用的一种方法是覆盖技术。

它是指一个作业的若干程序段，或几个作业的某些部分共享某一段存储空间。它不仅可用于一个作业内部的覆盖处理，也可用于多道程序系统中各作业之间，下面以一个作业的内部覆盖处理为例说明。

设一个作业由模块 A、B、C、D、E、F 组成，大小为 1900KB，其中 A 是主模块，其余为子模块，A 调用 B 或 C，但不会同时调用，B、C 之间也不互相调用，B 只调用 F，C 调用 D 和 E，D、E 和 F 也不会同时被调用，也不互相调用，其模块调用结构如图 4-1 所示。因此，模块 B 和 C 不用同时都在内存中，模块 D、E、F 也不必同时驻留在内存中。这样，可建立如下的覆盖结构：把整个作业分成两部分，模块 A 为常驻内存部分，大小为

200KB，其余为覆盖部分，大小为 900KB，覆盖部分又分成两个覆盖段，分别为 500KB 的覆盖段 0 和 400KB 的覆盖段 1，覆盖段 0 由模块 B 和 C 组成，覆盖段 1 由模块 D、E、F 组成，总共只需 1100KB 的内存。覆盖部分是以文件形式存储于辅助存储器上，在作业执行时，当需要访问某个覆盖部分，才把它调入内存。

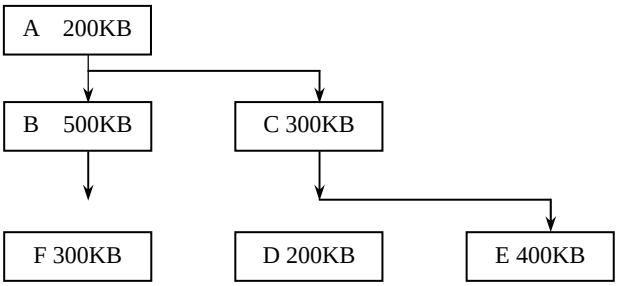

图 4-1 模块调用结构

当作业运行时，系统根据其覆盖结构，分配一段存储空间，其大小等于常驻部分和两个覆盖区之和。覆盖区和覆盖段是一一对应的。覆盖区长度由相应覆盖段中最大模块的长度确定。确定相应的存储空间后，把 A、B、F 首先装入内存中，当 A 运行到调用 C 时，才将 C 装入到覆盖区 0，自动将 B 覆盖，同时也将 D 装入到覆盖区 1，自动将 F 覆盖。当 C 运行到调用 E 语句时，再把 E 装入到覆盖区 1，覆盖掉 D。其存储分配和覆盖结构如图 4-2 所示。

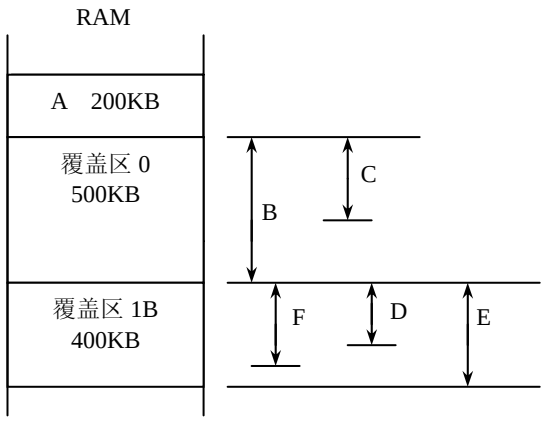

图 4-2 存储分配和覆盖结构

虽然可由操作系统完成自动覆盖，但要求作业各模块间有明确的调用结构，并要求用户向系统指明其覆盖结构，即对用户不透明，增加了用户的负担。目前主要将这一技术用于小型系统中的系统程序的内存管理上，主要因为系统程序的覆盖结构对系统软件设计者是已知的。在 MS-DOS 系统中，也采用了覆盖技术。

4.2.2 交换技术

交换技术主要是用于解决内存不足的问题，广泛用于小型分时系统的存储管理中，在分

时系统中，主机与多个终端用户相连，不可能将用户作业全部放入内存中，而是当调度程序允许某一进程运行时，把该进程调入内存运行一个时间片。当时间片用完或进程等待时，从内存中以文件的形式保存在辅助存储器中，让出 CPU 和内存空间，直到再次调用它时，才重新进入内存运行，又需要把程序和数据调入内存。这种把一个作业装入内存之后仍能把它交换出内存或再交换入内存的技术称为交换技术，也称滚入滚出。

交换技术可以保证合理的响应时间，并且可以利用辅存解决内存的不足，但却是以花费处理机的时间为代价的。交换技术中的关键是如何减少交换的信息量，这样才能提高系统的效率，对于分时系统来说，作业比较小，交换的信息量也少，系统的效率较高。对于大型作业，如果每次交换出整个作业，则交换的信息量太大，系统的效率将大大降低。

与覆盖技术相比，交换技术的交换过程对用户是透明的，但交换技术需要较多的软件支持。

交换技术的发展导致了虚拟存储技术的出现。

4.3 虚拟存储技术

虚拟存储技术首先由英国曼彻斯特大学提出，1961 年用该校的 Atras 计算机实现了这种技术。20 世纪 70 年代以来，该技术得到广泛使用，许多大型的计算机系统均采用了虚拟存储管理技术，如 PDP-11/45、IBM370 系列等计算机就是使用虚拟存储管理技术。在微型计算机系统中也广泛地使用虚拟存储管理技术，如采用分页技术的微型机有美国的 NS32032、日本的 V70，使用分段技术的有 Intel 80286，分段加分页的有 Intel 80386、Z-8000 等。

4.3.1 虚拟存储器的基本概念

虚拟存储器是指以透明方式提供给用户一个比实际内存大得多的作业地址空间。它不是一个实际的物理存储器，而是一个容量非常大的内存储器的逻辑模型。以处理机提供的逻辑地址访问虚拟存储器，用户可在非常大的地址空间中安排程序和数据，就仿佛拥有这么大的内存空间一样。

虚拟存储器提供比实际内存大得多的空间，作业运行时，就需要在虚拟地址和实际内存地址之间建立对应关系，也就是虚拟地址要转换到实际内存地址上，处理机才能访问内存中的指令和数据。这种转换是由动态地址映像机构来实现的。

当能够实现虚拟地址和实际地址转换后，还需确定用户作业所在的位置，以及内存和辅存中的程序和数据如何进行交换等。下面来看一下实现虚拟存储器的主要问题。

4.3.2 虚拟存储器建立的主要问题

把作业的地址空间和实际内存空间明确地区分开，从而有了虚拟存储器的概念。但在具体实现虚拟存储管理功能的时候，必须解决下面的三个主要问题。

1. 虚拟空间放在系统何处

一般使用大容量的辅存（磁盘、磁鼓）来存放用户的虚拟空间的全部数据。所以作业的虚拟地址空间不能无限增大，它受到两个条件的制约：一是指令中地址长度的限制，因为进

程访问的虚拟地址应限制在指令中地址长度所表示的范围内，即不能超过逻辑地址字长所表示的范围，而逻辑地址字长是由机器的硬件结构决定的；二是辅助存储器大小的限制，用户的虚拟空间不能超过辅存的容量，随着处理机功能的强大，制约最大虚拟空间的往往是辅存的容量。

2. 何时和如何进行内存、辅存的对换

在分页、分段、段页式实现的虚拟存储管理中，内存和辅存对换时机、策略稍有不同，但一般都是以软、硬件相结合的方式进行对换，而且对用户是透明的。如请求分页方式中，实现内存、辅存的对换是以页为单位，在作业运行过程中执行一条不在内存中页的指令而引发，并由缺页中断实现页的调入和调出。

3. 虚拟地址和实际地址的转换

作业的逻辑地址，在虚拟存储系统中被称为虚拟地址，而处理机可直接访问的内存空间地址被称为实际地址。作业的欲执行部分必须装入到实际地址空间中去，并建立虚拟地址和实际地址的映射关系。这就是虚拟地址和实际地址的转换。

虚拟地址和实际地址都是由处理机提供的。只不过虚拟地址是形式地址，通常由处理机的寻址方式和指令格式决定，由处理机内部的一些寄存器的某种组合来表达。因此，虚拟地址的字长通常与处理机内部数据总线的宽度有关。处理机提供的虚拟地址仅用于用户编程。实际地址则不然，它是处理机直接驱动的。因此，实际地址的字长通常由 CPU 芯片的地址引脚个数或计算机系统的地址总线宽度决定。处理机提供的实际地址用于完成实际存储单元的具体存取操作。如 Intel 80286 有 24 位地址线，其实际地址空间最大范围为 16MB。Intel 80286 采用分段存储管理技术，每个段的最大段长为 64KB，两个地址控制寄存器，两个 8K 的描述符表，其虚拟地址空间为 2*8K*64KB=1024MB。

4.4 缓冲存储器

处理机能够直接存取的是内存，处理机的速度比内存的速度快很多，处理机从内存中存取指令和数据时，就需要等待把内存中的数据存取出来。为了提高内存的存取数据速度，使它与处理机的速度相匹配，许多计算机系统都采用了缓冲存储器解决内存的速度与处理机的速度不匹配的矛盾。

缓冲存储器在处理机和内存之间形成这种三级存储结构，由于缓冲存储器的速度比较快，又称为高速缓冲存储器。其不参与存储器编址，对用户是透明的。

目前有许多计算机采用缓冲存储器，如 MC68020、Z8000、Intel 80386、Intel 80486、Intel 80586 等，缓冲存储器的大小从 2KB 到 256KB 不等。缓存技术不属于内存扩充技术。

4.4.1 缓冲存储器的结构

缓存读双字指令的速度为 120ns，内存的速度为 1100ns，处理机的速度为 150～300ns。缓存的速度比内存高一个数量级，同处理机的速度相匹配。这样，缓存在处理机和内存之间起到了缓冲的目的。

处理机、缓存和内存形成了三级存储结构，其关系如图 4-3 所示。

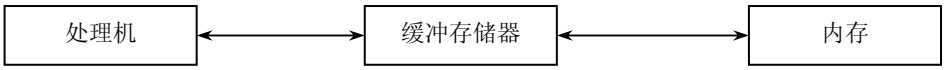

图 4-3　处理机、缓存和内存的关系

缓冲存储器由缓冲存储器、缓存目录和缓存控制器三部分构成。通常缓冲存储器和内存都分为若干块，我们以 4KB 的缓存为例，假设每块的大小为 64 字节，把缓存分为 2 个区，每区 2KB，每区有 32 个块，同一区中的各块用列号来表示，如图 4-4 所示。

区号	列号 0	1	2	……	30	31
0						
1						

图 4-4　4KB 缓存的区块形式

4KB 缓存的缓存目录有 32 个表目，对应缓存的 0～31 的列号，缓存目录的表目分为两个区，对应于缓存的两个区 0～1。这样缓存中的每一块都对应一个固定的表目。每个表目包括内存地址行号和状态位，另外，两区的每个表目对应一个标志位。表目的内存地址行号表示缓存块的内容所对应的内存地址所在的行。状态位有三种：有效位、修改位、故障位。当有效位为 1 时，表示该表目对应的缓存块中的数据已经是无效的；为 0 时，则对应是有效的。

当修改位为 1 时，表示该表目对应的缓存块中的数据已经被修改过；为 0 时，则没有修改过。

当故障位为 1 时，表示该表目对应的缓存块有故障，否则无故障。标志位表示最近访问了两区中的哪一区，以作为选择淘汰块使用。缓存目录的结构形式如图 4-5 所示。

区号	列号 0	1	2	……	30	31
0	内存地址行号					
	状态位					
1	内存地址行号					
	状态位					
	标志位					

图 4-5　4KB 缓存目录的结构形式

同样，内存和缓存一样也分成 64 字节的块，每行有 32 列，大小为 2KB，行号的大小由内存的大小来确定。其形式如图 4-6 所示。

行号	列号 0	1	2	……	30	31
0						
1						

图 4-6　内存的划分形式

这样，在编写内存地址的时候，就可以用行号、列号、每块的字节数的格式来表示。例如，80386 的编址形式为行号（17 位），列号（10 位），字节数（5 位）。

4.4.2 缓冲存储器的工作原理

由于处理机在进行读取和写入时，缓冲存储器的工作过程不同，分别描述如下。

（1）读取指令或数据。当处理机要读取指令或数据时，缓存控制器就自动查找缓存目录，以确定包含指令或数据的内存块是否在缓存中。它是根据指令或数据地址的列号部分查找缓冲目录表目的列号，然后把对应列号表目中分属区 0 和区 1 的内存地址行号与指令或数据地址的行号部分进行比较。如果匹配，并且状态位中的有效位为“0”，则把匹配表目所对应的缓存块中的内容直接送给处理机，且把标志位表示成对应块最近已被访问过。若两个区的对应表目中的内存地址行号都不与之匹配，则说明该块不在缓存。需要从内存中把该块内容送给处理机，同时该块内容也被送到缓存中的相应列的某块内。

（2）写入数据。当处理机要求写入数据到某内存单元时，也是首先由缓存控制器查找缓存目录，如包含此单元地址的内存块已经在缓存中，则处理机把在缓存中的该块内容加以写入，并把缓存目录中相应表目中的修改位置为“1”。这里是不是直接写入内存，有两种方法：惰性方法和立即存方法。惰性方法是指数据写入缓存时，不立即修改相应内存块中的内容，而是直到该缓存块中的内容被淘汰出缓存时，才把该缓存块写入内存相应块中。立即存方法是指对缓存和内存的相应块同时写入。若该块不在缓存中，则先把该块从内存读入缓存，然后再写入缓存中。

（3）通道读取指令或数据。当通道（I/O 处理机）从内存读取指令或数据时，则查找缓存目录，假若包含该指令或数据地址的块在缓存中，则从缓存中把该块送往通道。若不在缓存中，则从内存读出，但不把该块放入缓存中。

（4）通道写入数据。当通道向内存某单元写入数据时，该数据只写入内存中。但缓存控制部件同时查找缓存目录。如果不在缓存中，则不进行任何操作。若该内存单元所在的块在缓存中，则相应表目的状态位中的有效位被置成“1”，使该数据块无效。

缓冲存储器的设置，使 CPU 所需的指令与数据绝大多数来自缓存，而不通过总线去访问内存，避免了处理机的等待，大大提高了系统的吞吐率，是发挥处理机能力的有效方法。

4.5 存储管理机制

存储管理的功能有四个：内存的分配与管理，内存空间的共享，存储保护和存储扩充。各种计算机系统的结构、性能和实现目的不同，它们采用的操作系统一般也不同，各种操作系统之间最明显的区别之一，一般在于它们所采用的存储管理方案不同。操作系统发展到现在，出现了成百上千种，存储管理方案也出现了很多，大致把存储管理方案概括成四种：分区管理、分页管理、分段管理和段页式管理。下面介绍各个方案的基本思想和实现技术。

4.5.1 分区存储管理

分区管理是能满足多道程序的最简单的存储管理方案。它是将内存划分为若干个连续的区域，称为分区。每个分区存储一个程序，而且程序只能在它所在的分区中运行。

如果是单道批处理系统，对内存的用户程序区域不再划分，采用单一连续区存储管理方案，是将内存的一部分固定分配给操作系统，其余部分分配给一个作业使用。一些单用户单任务的微型计算机系统采用单一连续区存储管理方案。

在多道批处理系统中，为允许多个作业共享内存和处理机，可以采用分区存储管理方案。把内存划分成若干个分区，每个分区分配一道作业。

分区存储管理的划分方式有很多，可以归纳为两大类：固定式分区和可变式分区。

1. 固定式分区

固定式分区是指在处理作业之前，存储器被划分成若干个分区。关于分区的说明，可由系统操作员或由操作系统实现。除操作系统本身占用一个分区外，其余的每一个分区可分配一道作业。当划分好后，在系统运行期间不再重新分区。因此，又称为静态式分区。

根据欲处理作业的不同规模，一般将内存的区域划分成大小不同的区域。系统建立一个分区说明表，说明分区的大小、起始地址和是否已分配的标志。分区说明表如图 4-7 所示。

区号	大小	起始地址	标志
1	16K	20K	已使用
2	32K	36K	已使用
3	64K	68K	未分配

图 4-7　分区说明表

由于每一个分区的大小是固定的，故每个注册的作业必须说明所要求的最大容量。在调度作业时，由存储管理程序根据作业所需内存空间，在分区说明表中找出一个足够大的空闲分区分配给它，然后用重定位装入程序将此作业装入。若找不到，则通知作业调度模块，另外选择一个作业。图 4-8 表示某一时刻，作业 A，B 分别被分配到 1，2 两个分区，第三分区尚未分配，操作系统永久占据内存低地址区 20KB 的内存分配情况。一个作业结束时，系统又调用存储管理程序查找分区说明表，把所占分区的使用标志修改为未分配状态。

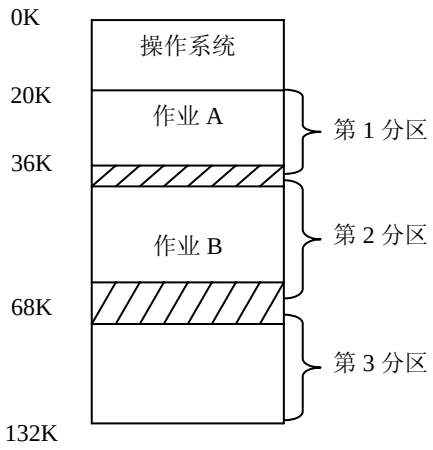

图 4-8　内存分配图

采用这种技术，虽使多个作业共驻内存，但一个作业的大小不可能正好等于某个分区的大小，于是在分配的分区中总有一部分被浪费。有时这种浪费还相当严重。尤其是，在如图 4-8 所示的内存使用情况下，第 2 分区的未分配部分和第 3 分区已是物理上的一个连续区域，可是若有一个大小为 65KB 的作业申请内存则将被系统拒绝。因为分区的大小是预先划分的，分区说明表中指出，只有第 3 分区是未分配的，它的大小是 64KB，小于 65KB，所以系统不能分配。

固定式分区实现技术简单，但内存利用率不高，适用于作业的大小及多少事先比较清楚的系统中。

2. 可变式分区

可变式分区指存储空间的划分是在装入作业时进行的，且分区大小正好适应作业的需要。分区的个数和大小不是事先划分，而是根据作业大小进行调整，因此又称动态式分区。这种存储管理技术是固定式分区的改进，可获得较大灵活性而提高内存利用率。

系统初启时，内存中除常驻的操作系统外，其余的是一个完整的大空闲区。随后，对调入的若干作业接连划分几个大小不等的分区分配给它们。但是，系统运行一段时间后，随着作业的撤除且相应分区的释放，原来一整块的存储区会形成空闲分区和已分配分区相间的局面。

可变分区在分配时，首先找到一个足以容纳该作业的空白分区，如果这个空闲区比所要求的大，则将它分成两部分：一部分成为已分配的分区，剩下一部分仍为空闲区。如果没有足够大的区域，将拒绝进行分配。

在回收撤除作业所占分区时，要检查回收的分区是否与空闲区邻接，若是则加以合并，使之成为一个连续的大空闲区。

实现可变式分区的方法很多，一般需要登记内存占用的情况，另外能实现内存分配和回收。记录内存分配情况的数据结构主要有两种，一种是表格形式，另一种是空闲区链形式。分配算法一般有两种：一种是最佳适应法，它从全部空闲区中找出能满足作业需求的容量最小的空闲区分配；此方法能使碎片（不连续的小空闲区）尽量小。另一种是最先适应法，它按序查找，把最先找到的满足需求的空闲区分配，以减少查找时间。

记录内存分配情况的表，一般是两张表，一张表说明已分配的分区，称为 P 表；另一张表说明空闲的分区，称为 F 表。两表的登记状况对应某一时刻内存分配情况。表项的数目要足够，一些暂时未使用的表项称为空表目。说明两个表的情况如图 4-9 所示。

对于空闲分区说明表来说，它的表项排序方法与所采用的算法相适应。采用最佳适应法时，空闲分区按由小到大排序。采用最先适应法时，空闲分区按起址由低到高排序。对于已分配分区说明表，则不需要排序。

采用空闲区链数据结构的可变式分区存储管理，其实现方法是把每个空闲区的起始若干个字节分为两部分：一部分是链指针，指向下一空闲区的起始地址；另一部分表明本空闲区的大小。系统用一固定单元作为链的头指针，指向第一个空闲区的起始地址。最后一个空闲区的链指针中放着链尾标志。这样使用链指针把所有空闲分区链接在一起，构成了一条空闲区链。还有一种实现方法是采用双向链数据结构，它分成三部分：向后链接指针、向前链接指针和本空闲区的大小。它形成一个循环链，分配和回收时，查找起来比较方便。

序号 P	大小	起址	状态
1	8K	20K	已分配
2	32K	28K	已分配
3	—	—	空表目
4	120K	92K	已分配
5	—	—	空表目
…	…	…	…

序号 F	大小	起址	状态
1	32K	60K	空闲
2	300K	212K	空闲
3	—	—	空表目
4	—	—	空表目
5	—	—	空表目
…	…	…	…

图 4-9　可变式分区说明表

可变式分区管理有零头问题。零头是不连续的小块空闲区。可能所有零头的总和大于作业的容量要求，但由于不连续无法进行分配。解决零头的方法是移动已分配的作业，使零散的小空闲区连成一个大的连续空闲区，进行分配。这种方法需要对作业重新定位，系统代价太高。

3. 分区的保护措施

为了防止一个作业有意或无意破坏操作系统或其他作业，分区管理的存储保护采用两种方法。一种是界地址法，另一种是保护键法。

界地址法是指操作系统设置一对上、下界寄存器，来限制当前运行作业的工作地址范围。当选中某个作业运行时，先将它的上、下界地址装入寄存器中，每一次访问存储器的地址时都要同这两个寄存器进行比较，若超出这个范围，便产生越界保护性中断。

保护键法是指操作系统给每个存储块都分配一个单独的保护键，它相当于一把锁。在程序状态字中设置有保护键字段，对不同的作业赋予不同的代码，相当于一把钥匙。当程序进行读、写访问时，程序状态字中的保护键字段内容必须与被访问存储块的保护键相一致，才可进行读写访问，否则不能进行读写。当作业运行时产生非法访问，系统将产生保护性中断。

4. 分区存储管理的优缺点。

分区管理的主要优点是：实现了内存的共享；在进行分区分配时，系统占用存储量较少，算法也比较简单；分区保护的方法比较简单，实现比较容易。

分区管理的主要缺点是：内存不能充分利用，存在严重的零头问题；不能实现对内存的“扩充”；要求一个作业执行之前必须全部装入内存；较难实现几个作业间共享某一程序段或数据段。

4.5.2 分页存储管理

在分区存储管理中，一个作业总要求占用一个连续的内存区，因而产生了零头问题。即使所有零头的总和超过一个作业要求的总量，但因为零头之间不连续，也无法分配给作业。如果可以允许一个作业存放在不连续的内存中而又能保证作业连续地得以运行，就可解决零头问题。这就是分页存储管理的由来，分页存储管理有两种：简单页式存储管理和请求页式存储管理。

1. 简单页式存储管理

简单页式管理是指系统把内存的存储空间等分成大小相同的存储块，称为块。同样，作业的地址空间也分成大小与块相同的部分，称为页。当一个有 m 个页的作业运行时，为该作业分配 m 个块，这些块可以是不连续的。

（1）简单页式管理的基本概念。

块：把内存划分成相同大小的存储块，称为块。对于一个特定的计算机系统来说，块的大小通常是固定不变的，并给各个块从零开始依次编号。块的大小一般是 1K～4K。

页：把一个作业的逻辑地址空间划分成若干个与块大小相同的部分，每部分称为页。并给各个页从零开始依次编号。这样，在分页系统中，每一个逻辑地址用一个数对(p,d)来表示，其中 p 是页面号，d 是在页面号中的相对地址，称为页内地址。

页表：为了解决作业的逻辑地址转换为实际物理地址的问题，在内存中为每个作业建立一个页面映像表，称为页表。页表包括两部分：页号和块号。页号是从 0 开始连续编号，表示作业各页的序号。块号指出该页在内存中的位置。

当作业调度程序调入作业时，为该作业建立进程，系统为进程建立页表。在撤消进程时清除其页表。除此之外，系统还设置一个页表地址寄存器指出当前运行的进程的页表起始地址和页表的长度。

进程的逻辑地址是用一个数对(p,d)来表示，那么这个数对在机器指令的地址场中又如何表示？通常将该地址分为两部分：一部分表示该地址所在页面的页号 p，另一部分表示页内地址 d。页号和页内地址所占位数，取决于页的大小。如 IBM370 的指令地址域长度为 24 位，页的大小为 2KB，则页内地址部分占 11 位。

机器指令的地址域中的地址由硬件机构自动转换成对应的页号和页内地址。

（2）直接映像的页地址转换。

当进程被调度到处理机上运行时，操作系统自动将该进程的页表起始地址装入页表地址寄存器中，当进程要访问某个逻辑地址时，则分页的地址映像硬件自动按页面大小将地址域从某位截成两部分：页号和页内地址(p,d)。这时以页号 p 在页表中查找，具体过程为：加法器将页表地址寄存器内容与页号相加，得到该页在页表中的表目地址，该表目中包含有该页对应的块号 q，然后地址映像硬件又自动将块号 q 与页内地址拼成了实际内存的绝对地址，实现了地址映像。如图 4-10 所示。

直接映像的分页系统对系统的性能有很大的影响，处理机执行指令的速度降低为原来的二分之一。主要因为 CPU 至少访问两次内存才能得到所要的数据。第一次查页表以找出对应的块号，第二次访问所需数据。为了提高查表的速度，可以采用更快的地址转换方法，即相关映像页地址转换法。

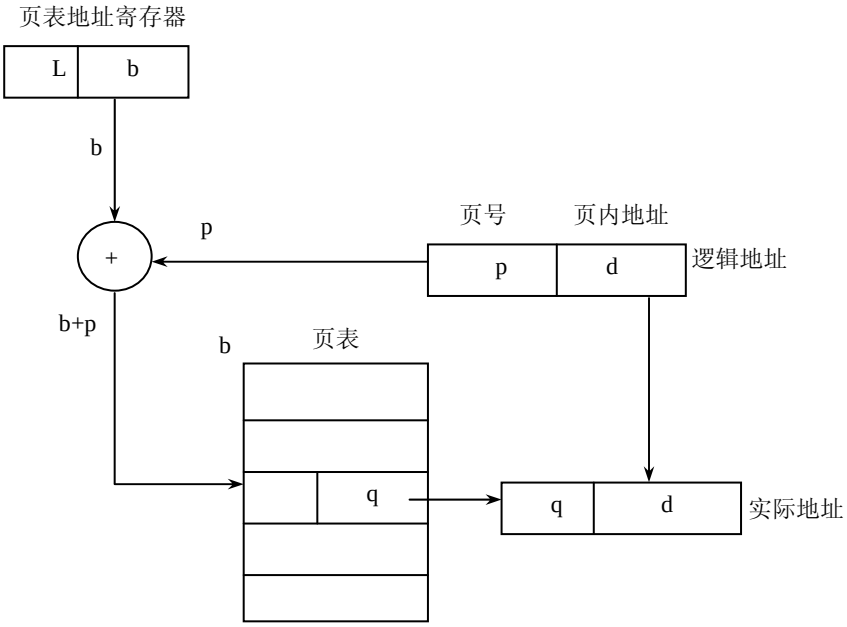

图 4-10　直接映像的地址转换

（3）相关映像页地址转换。

为了提高页地址转换速度，在地址变换机构中加入一定数量的高速半导体相关存储器。

半导体存储器的存取速度比一般存储器高一个数量级，而且具有并行查询能力。其中存放着正在运行进程中的最常用的部分页面的页号和它的相应块号。构成一张“快表”。其地址转换过程为：当运行进程要访问逻辑地址时，硬件机构将地址截成页号 p 和页内地址 d。地址转换机构首先以页号 p 和相关存储器中各表目同时进行比较，以确定该页是否在相关存储器中。若在其中，则相关存储器即送出相应的块号，并与页内地址一起拼成绝对地址，然后访问内存。若该页不在相关存储器中，则使用直接映像方法查找进程的页表，找出其块号 q 与页内地址拼成绝对地址，然后访问内存。同时将该页的页号及对应的块号一起送入相关存储器的空闲表目中。如无表目，一般将最先装入的表目淘汰，腾出表目位置。在地址转换过程中，相关映像和直接映像是同时进行的，当相关映像成功时，就自动停止直接映像工作。相关映像的地址转换过程如图 4-11 所示。

由于程序运行中有局部性的特点，即刚被访问过的单元在很短的时间内还将被访问（时间局部性）和刚被访问的单元的邻近的单元也将被访问（空间局部性）。所以使用一定数量的高速寄存器作相关存储器，从相关存储器中找到所需页号的概率将达到 90%左右。如果继续增加高速寄存器的数量，其概率还将提高。从这里看出，使用相关映像的地址转换方法将大大提高系统的性能。

（4）简单页式管理的分配方法。

为了实现分页存储管理，操作系统需建立和管理一些基本的数据库：页表、进程表和存储分配表。

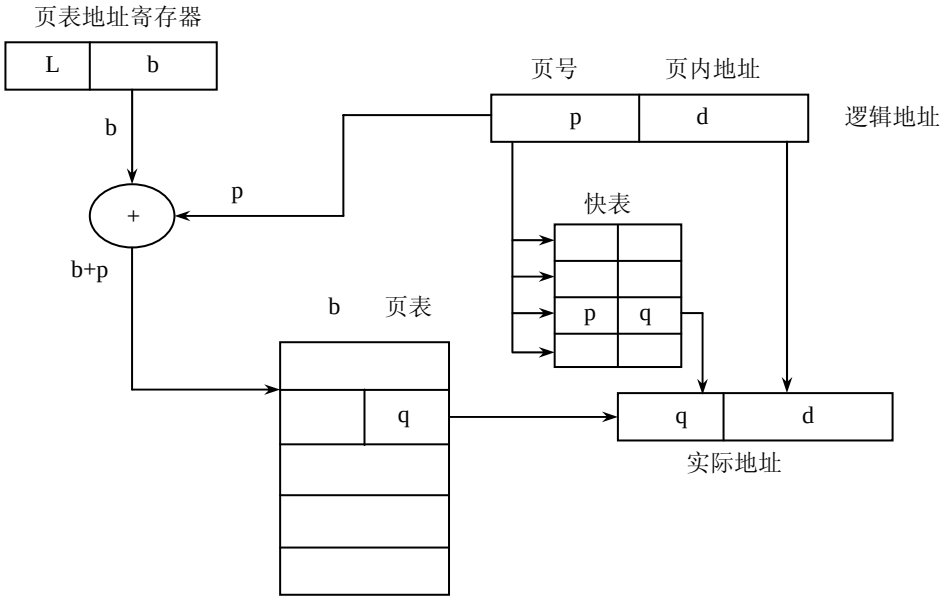

图 4-11　相关映像的地址转换

进程表是整个系统的进程控制表的集合，记录每个进程的运行情况。其中与存储管理有关的信息是页表起始地址和页表长度。当某作业被调度进入内存时，操作系统将其信息登入该进程的进程控制表，当进程运行时，操作系统的调度程序从此表中将页表起始地址和页表长度装入页表地址寄存器中。

存储分块表是整个系统的一个，用来记录内存每个存储块的状态，指出该块的状态是已分配还是空闲。

简单页式存储管理的分配和回收方法是：作业的逻辑空间和内存的地址空间分成页和块后，作业运行时，判断内存的块数是否满足作业的需要，而且是整个作业的需要，如果空闲块数不够，则拒绝装入，否则就分配内存给作业，以满足其运行需要。回收时，将该作业所占的存储块在存储分块表中记为空闲状态，同时把进程表中的该作业的表目改为空表目。简单页式管理的分配算法流程如图 4-12 所示。

简单页式存储管理虽然消除了分区间的零头问题，但一个作业的地址空间不一定是页的整数倍，最后一个块一般是不满的。这种最后一块内的空闲部分称为内零头。内零头的平均大小为半页左右。为减少内零头所造成的浪费，可把页的长度变小，但又增大了页表的长度，降低查找的速度。

简单页式管理与分区存储管理一样，作业的地址空间不能超过内存的实际地址空间。为了解决这个问题，引入了虚拟存储技术，发展出一种新的页式存储管理：请求页式存储管理。

2. 请求页式存储管理

请求页式存储管理的基本思想是：当运行一个作业时，并不要求把该作业的全部程序和数据都装入内存，可以只把目前要执行的几页调入内存的空闲块中，其余仍保存在辅存中，以后根据作业运行的需要再调入内存。

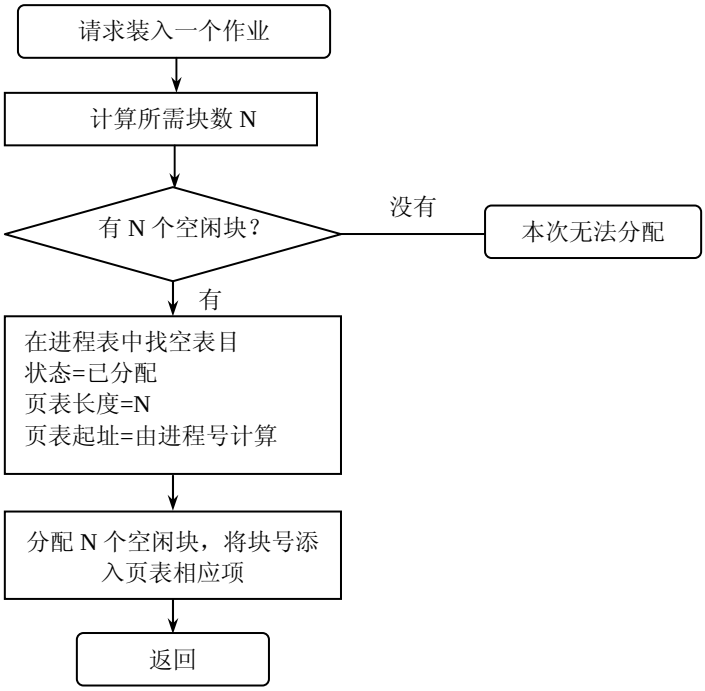

图 4-12　简单页式管理的分配流程

请求页式管理在作业执行时只装入部分页在内存中，那么在实际处理的时候，就会出现新的问题：作业的页表应登记哪些页已在内存中，哪些在辅存中，这需要扩充页表的项目；当需要从辅存中调入页时，进行缺页中断处理。另外，当欲调入页而内存中无空闲块时，如何进行处理，涉及淘汰算法方面的问题。

（1）页表项目的扩充。

在简单页式管理中，页表只包括页号和块号，在请求页式管理中，只有这两项是不够的，还需要表示出该页的状态，是否在内存中，是否被修改过，是否被访问过，如果不在内存中在辅存的地址是多少等信息。具体来说，除了页号和块号项外，请求页式管理中的页表包括如下几项：

状态项：一般有三种状态。“断开”状态，表示该页不在内存；“连接”状态，表示该页已分配块号，但正在进行数据的输入输出操作，此时，CPU 对该页不能进行访问，但通道可以对它访问；“可寻址”状态，表示该页已分配块号，CPU 可以访问它。

修改位：该位为 0 时，表示该页中的数据未被修改过。该位为 1 时，表示该页中的数据被修改过。

访问位：表示该页在最近期间是否被 CPU 访问过。该位为 0 时，表示该页未被访问过。为 1 时，表示该页最近被访问过。用于页面淘汰策略中。

辅存地址：指出该页在辅存中的地址，以便用来调入。

（2）请求页式管理的地址变换和缺页中断处理。

进程访问的是逻辑地址，进程通过处理机执行程序，处理机只能访问在内存中的物理地址，那么在请求页式管理中逻辑地址到物理地址的转换是由软硬件共同来完成的。由三部分

共同完成地址转换：分页地址变换硬件部分；中断处理硬件部分；缺页中断处理程序。其中缺页中断处理是软件。

其中分页地址变换硬件是主要的部分，它独立完成正常的地址变换工作。其余两部分只有在发生缺页情况下才协助分页地址变换硬件进行工作。

在程序执行时，请求页式管理的地址变换过程是这样的：

1）将逻辑地址由硬件自动截成页号 p 和页内地址 d。

2）硬件自动查找页表中该页状态位，看该页是否在内存中，如果在内存中，则继续进行第 3 步；如果不在内存中，则执行第 5 步。

3）硬件将页表中的该页所对应的块号与页内地址拼成物理地址。

4）硬件完成地址转换，准备转换下一个逻辑地址。返回第 1 步。

5）硬件使“缺页中断触发器”被触发，并由中断处理硬件继续工作。

6）当触发的缺页中断被接受后，中断处理硬件自动进行处理机状态的切换。即把处理机的状态保存起来，并把对 CPU 的控制权转给缺页中断处理程序。

7）缺页中断处理程序保存断点现场。

8）在存储分块表中判断是否有足够的空闲块，如果有，从辅存中调入所需的页面；如果没有，根据“淘汰”策略选择淘汰页面，调整页表，使其有空闲块进行分配，如果淘汰页面被修改过，则应写回辅存中相应分页。

9）根据分配情况，调整页表。

10）恢复现场，返回断点执行，即返回第 2 步执行。

3. 淘汰策略

在缺页中断处理时，欲调入的某页，但内存无空闲块，就需要把在内存中的某一页调出内存。应该调出哪一页？这就是淘汰问题。

如果淘汰的策略不合理，有可能刚被调出的页立即又要求被调入。内存和辅存之间这种频繁地来回调入调出页面的现象称为抖动。这是在确定淘汰策略时应避免的情况。

常见的淘汰策略主要有以下三种：

（1）先进先出法：指先调入内存的页面先淘汰，即总选择驻留内存时间最长的页面进行淘汰。因为最早调入的页面其不再使用的可能性比最近调入的要大。

（2）最近很久未用法：指最近很长时间未被访问过的页面被淘汰。因为如果某页刚被访问过，可能最近还被访问；相反，若某页最近很长时间未被访问，则最近被访问的可能性极小。

（3）最不经常使用法：指选择最不经常使用的页面淘汰。它实现起来比较简单，只需为每页设一个计数器。

4. 页的共享

在多道程序系统中，特别在分时系统中，数据共享是很重要的。在分页系统中，各个共享进程应能访问共享数据的页。共享的方法是使这些共享进程的逻辑空间中的页指向具有共享数据的相同的块号。这样，同一个块号就对应多个进程中的多个页号，从而，实现页的共享。

页面共享中必须处理好共享页中的信息保护问题。当一个进程正从共享页中读取数据时，要防止另一进程修改该页中的数据。在大多数执行共享的系统中，将程序分为过程区和数据区。不可被修改的过程称为纯过程，它是可以共享的。私用的可修改的数据和过程是不可被共享的。

4.5.3 分段存储管理

前面介绍的各种存储管理技术中，为作业的逻辑地址空间是一个一维的线性地址空间。而一个作业常常由大量的标准和非标准的程序模块和数据段组成。如果作业运行前，将其所需的各程序段和数据段连成一维的线性地址空间，这浪费机时，特别是在程序和数据共享时，不便于实现。因此编程人员希望把信息按其内容或函数关系分成段，每段有自己的名字，可以根据段名访问相应的数据和程序段。内存也按段进行分配。这样，就有了分段存储管理技术。

1. 分段存储管理的基本概念

（1）进程的逻辑地址空间：一个段可定义为一组逻辑信息，如子程序、数组或工作区。这样，每个进程的地址空间按照程序的自然逻辑关系划分成若干个段，每段都有自己的名字，且都是一段连续的地址空间。所以整个作业的逻辑地址空间是二维的。每个逻辑地址由两部分组成：段名和段内地址。如分段 A，其逻辑地址范围 0～3K。一个作业的分段情况如图 4-13 所示。分段经过编译或汇编后，通常为每一段分配一个编号，称为段号，每个段的地址空间编成连续的地址空间，于是经过编译后的逻辑地址由两部分组成：表示段名的唯一段号和段内相对地址。

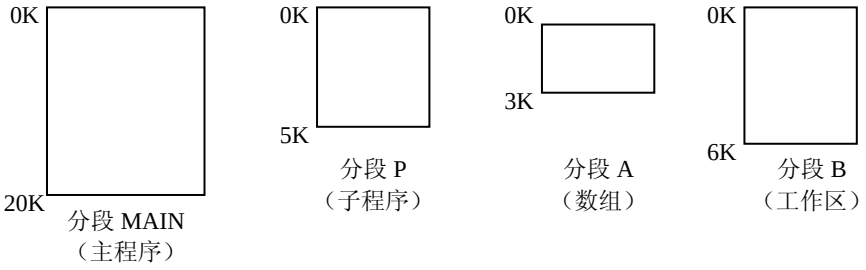

图 4-13　分段的地址空间

（2）程序的地址结构：每个逻辑地址用段号 s 和段内相对地址 w 来说明，可以表示成(s,w)，指令的地址域也用段号 s 和段内相对地址 w 来表示逻辑地址。一旦段号和段内相对地址的长度确定了，一个作业的地址空间允许的最多段数及段内长度也就确定了。假设某机器指令的地址部分为 24 位，规定其中 8 位表示段号，其中 16 位表示段内相对地址，这个地址结构可允许的最多的段数为 256 个段，最大的段长为 64KB。

（3）段表：当系统执行某个进程时，为其建立一个段映像表，简称段表。段表中包括以下内容：段号、段长（段的长度）、段在内存中的起始地址等。段表按段号从小到大的顺序排列，当作业调度程序调入该作业时为其进程建立段表，在撤消进程时，清除此进程的段表。

系统还使用一个段表地址寄存器表示段表在内存中的起始地址和段表的长度。它是在进程运行之前，操作系统从进程控制表中将这两项装入到段表寄存器中，从而保证地址的动态转换。

2. 分段管理中的地址转换

分段存储管理是指管理由若干个分段组成的作业，并且按分段进行存储分配。实现分段

管理的关键在于，如何把分段的二维地址空间中的一个作业，在线性的存储空间中正确运行。也就是如何把分段地址结构转换成一个线性的地址结构。和分页管理一样，采用动态重定位技术，通过与分页管理类似的地址变换机构来实现。当进程运行时，由系统将该进程的段表地址和段表长度装入段表地址寄存器中，当进程访问某逻辑地址(s,w)时，系统按如下步骤进行地址转换（如图 4-14 所示）：

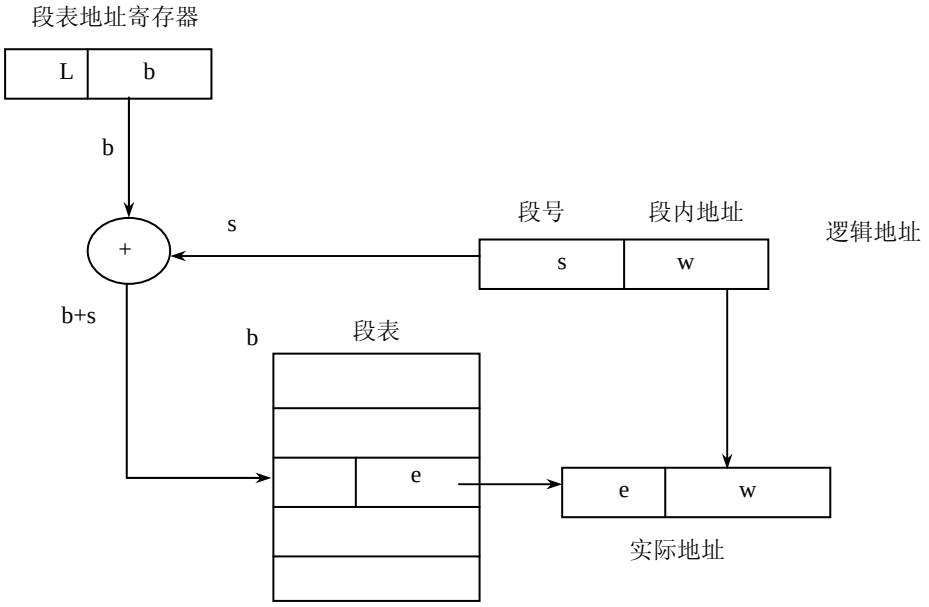

图 4-14　分段地址转换

（1）根据段表地址寄存器的内容找到段表地址。

（2）利用逻辑地址中的段号 s 在段表中找到该表目的入口地址，得到该段在内存中的起始地址 e。

（3）将段内相对地址 w 与该段的起始地址 e 相加，即得到访问单元在内存的物理地址，并进行访问。

以上三步是由硬件自动完成的。与分页中类似，CPU 执行指令的速度降低为原来的二分之一。为了提高地址转换速度可以采用高速相关存储器技术。

由于分页和分段管理的地址转换机构十分相似，所以常常把它们搞混。其实，它们在概念上是完全不同的，分页的作业地址空间是一个单一的线性地址空间，而分段的作业地址空间是二维的。页和块对应，是信息的物理单位，大小固定，分页活动是看不见的，仅仅用于对内存的管理。段是信息的逻辑单位，它是有意义的一组信息，其长度不定。分段对用户是可见的，可以在用户编程时确定，也可以在编译程序对源程序编译时根据信息的性质来划分。

3. 分段的虚拟存储管理

分段存储管理，也可按类似请求页式管理来处理，为用户提高一个比内存空间大得多的地址空间，即为作业提供一个虚拟存储器。同样，虚拟存储器的容量由计算机的地址结构确定。在一个系统中可以建立多个虚拟存储器，每个作业存于一个虚拟存储器中。虚拟存储器和内存物理地址之间的映射由各自的段表实现。

段式虚拟存储管理在作业执行时只装入部分段在内存中，在执行过程中访问到不在内存的段时再把它们装入。在段表中必须说明哪些段已在内存中，存放在什么位置，段长是多少，哪些在辅存中，位置在哪里，还需要设置该段是否被修改过，是否能移动，是否可被扩充等，这需要扩充段表的项目；当需要从辅存中调入段时，进行缺段中断处理。

（1）段表项目的扩充。

在前面介绍的段表中，只包括段号、段长和段在内存的起始地址，在段式虚拟存储管理中，只有这两项是不够的，需要把段表扩充为如下几项：

段号、段长和段在内存中的起始地址。

状态项：该位为 0 时，表示该段不在内存；该位为 1 时，表示该段在内存。

修改位：该位为 0 时，表示该段中的数据未被修改过。该位为 1 时，表示该段中的数据被修改过。

访问位：表示该段在最近期间是否被 CPU 访问过。该位为 0 时，表示该段未被访问过；该位为 1 时，表示该段最近被访问过，用于段的淘汰策略中。

存取位：表示该段是否可以进行读写，或者是否可以执行。

扩充位：表示该段是否可以进行扩充。

辅存地址：指出该段在辅存中的地址，以便用来调入。

（2）段式虚拟管理的地址转换。

在作业执行中访问某段时，由硬件的地址转换机构查段表，若该段不在内存，则硬件发出一个缺段中断，操作系统找出一个足够大的连续区域以容纳该分段。如果找不到一个足够大的连续区域，则检查空闲区的总和，如果空闲区总和能满足该分段要求，那么进行适当移动后，将该分段装入内存。若空闲区总和不能满足要求，则调出一个或几个分段到辅存中，再将该分段调入内存中。如果该段在内存中，则判断欲访问的地址是否超出段长，如果超出，则硬件产生一个越界中断，操作系统处理这个中断时，先判断该段的“扩充位”标志是否可以扩充，若可以扩充，则增加段的长度。如果该段不允许扩充，那么这个越界中断就表示程序出错。最后，还需要判断该段的“存取位”与访问的方式是否符合，如果不符合，则产生保护中断。若符合，则改变该段的“访问位”和“修改位”。然后，把该段在内存的起始地址和段内相对地址相加，得出应该访问的内存地址。段式虚拟存储管理的地址变换过程如图 4-15 所示。

4. 分段的动态连接

一个作业是由许多程序模块和数据段组成的，无论是分区存储管理还是分页存储管理技术，为了程序正确地运行，必须由连接装配程序把一个作业所调用的各个子程序和数据段连接成一个可运行的目标模块，即将其连接装配成一个一维的线性连续地址空间。在连接装配过程中，既要把所需的子程序从各个程序库中找出来，并且要重定位，把它们组成一个统一编址的相对地址空间。这个过程称为运行程序的“静态连接过程”。

这种连接装配过程既复杂又费时间，有时连接过程所花费的机时比该作业的执行时间还长。有时连接装配好的模块在程序运行过程中根本用不到，造成机时和内存空间的浪费。因此，最好能采用当运行到某段时，再连接该段的方法，这种方法称为“动态连接”方法。它是指在程序运行开始时，只将主程序段装配好并调入内存。其他各段的装配是在主程序段运行中逐步进行的。每当需要调用一个新段时，再将这个新段装配好，并与主程序段连接。在

分段存储管理中，由于逻辑地址空间是二维的，每段有自己的段名，因而实现动态连接比较容易。而在分页存储管理中动态连接是难以实现的。

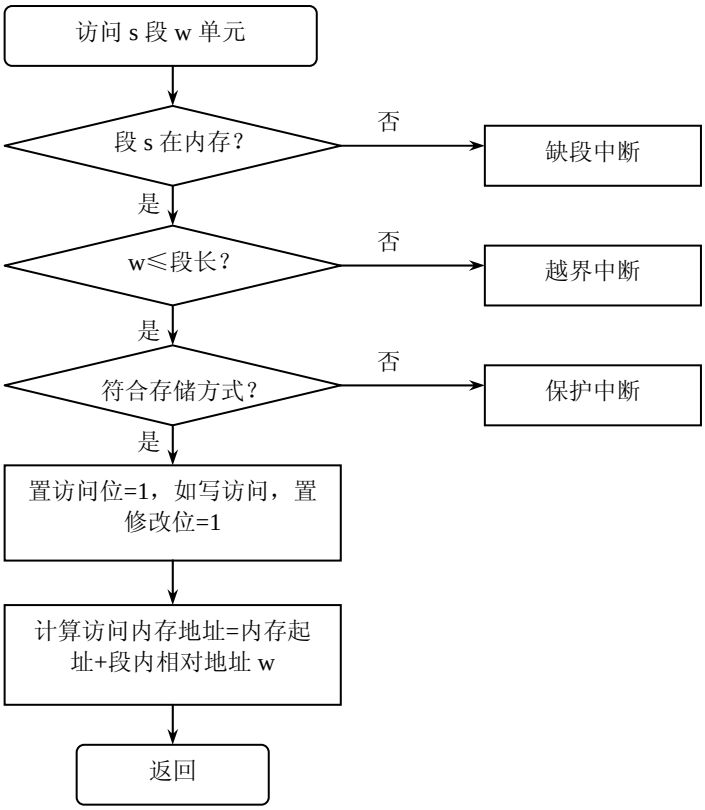

图 4-15　段式虚拟存储管理的地址变换过程

实现动态连接需要两个附加的硬件设施：间接编址和连接中断位。间接编址指指令中的地址部分不是存取数据的直接地址，而是间接地址，即存放这个直接地址的那个单元地址。那个包含直接地址的字称为间接字。连接中断位设在间接字中。直接编址和间接编址类似于机器指令的直接地址和间接地址，例如，直接地址 LOAD 1000 表示装入地址 1000 中的内容 500；间接编址 LOAD * 1000 表示以 1000 中的内容 500 为地址，装入地址 500 中的内容 30，如图 4-16 所示。

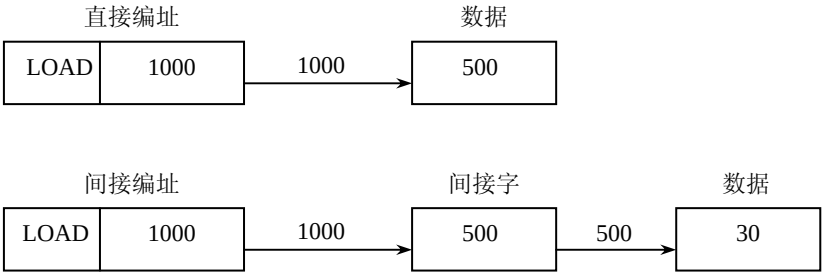

图 4-16　直接编址和间接编址

在实现动态连接时，把间接字的第 0 位作为连接中断位，以 L 来表示。当 L=1 时，表示需要进行连接，发出连接中断信号，转操作系统处理，进行连接工作。当 L=0 时，表示不需要连接。间接字的格式为：

L		直接地址

通过一个示例说明间接编址指令的执行过程。当一个分段（如 3 段）对另一分段（如 4 段）产生符号形式的调用（如 LOAD 1,[X]|[Y]），那么编译程序或汇编程序就产生一个间接编址的指令来代替，且将间接字中的连接中断位设为 1，直接地址指向代表符号调用的字符串（7"[X]|[Y]"）的所在位置。

分段 3 执行到 LOAD * 1,3|100 指令时将产生连接中断，操作系统对连接中断的处理过程如下（如图 4-17 所示）：

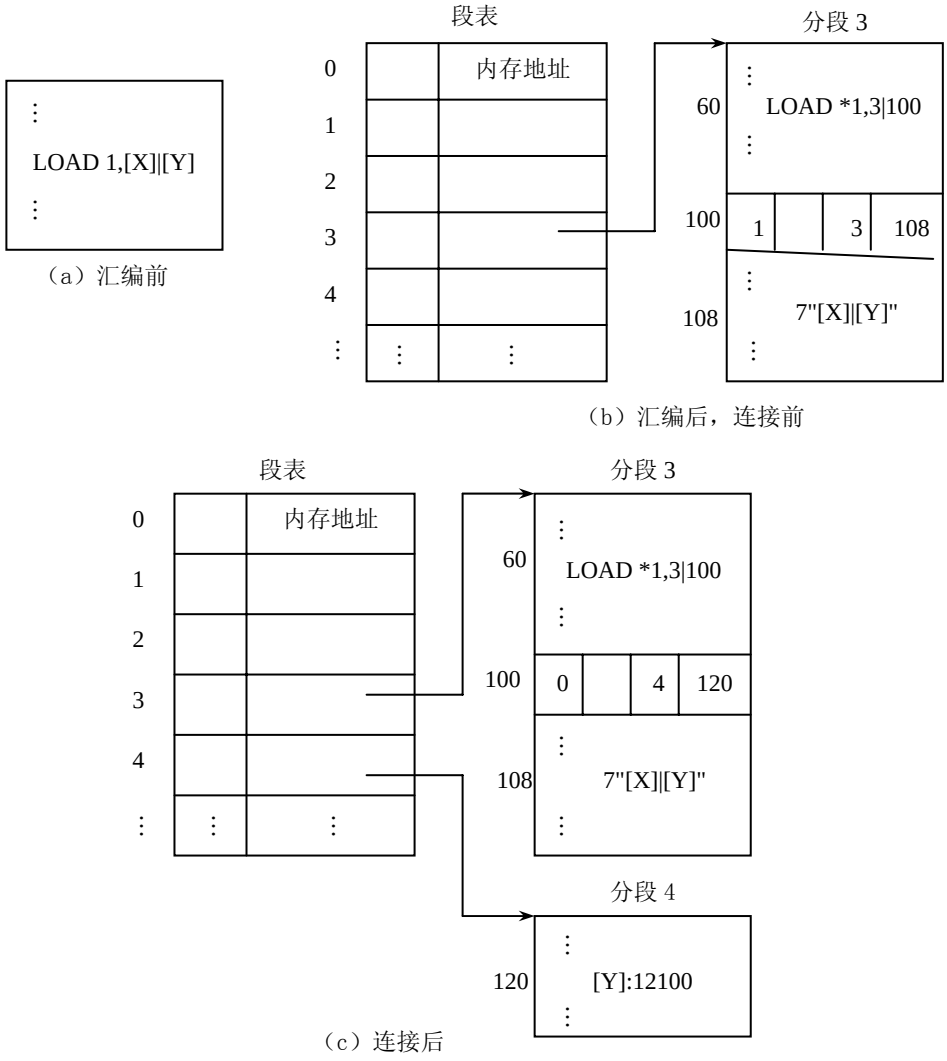

图 4-17　分段的动态连接

（1）从 3 段 100 单元取出间接字。

（2）取出间接字中的直接地址 3 段 108 单元。

（3）按直接地址取出要连接段的符号名[X]|[Y]，按定义给它分配段号（设[X]=4，[Y]=120）。

（4）查[4]段是否在内存，若不在则从辅存中把它调入内存，并登记段表。

（5）修改间接字，置连接中断位为 0，且使直接地址为连接的分段地址，即 4 段 120 单元。

（6）重新启动被中断的指令执行。

当指令重新执行时，由于 100 单元处的连接中断位为 0，不再引起中断，且直接地址就是要连接的地址 4 段 120 单元，于是可从 4 段 120 单元读出所需的数据装入到 1 号寄存器。

5. 分段的共享

段式虚拟存储管理利用分段的动态连接功能很容易实现分段的共享，由于不能事先知道什么时候调用共享段，且段号可在动态连接时分配，因此不可能事先规定共享段的段号，这样一个共享段对不同的调用者可以规定不同的段号。例如，对公共子程序 COS，在作业 1 调用时具有段号 6，而在作业 2 调用时可以用段号 2。

（1）共享段表：为了实现段的共享，需要设置一个共享段表来记录段的共享情况，共享段表一般包括段名、状态、在内存的起始地址、段长、调用该段的作业数、作业名和作业定义的段号等信息。

（2）分段共享的实现原理：当某个作业首次调用某个段产生一个连接中断后，进行动态连接，如果连接的是共享段，则在共享段表和该作业的段表中填写必要的项目。当共享段被调出内存后，应在共享该段的每个作业的段表中表示该段不在内存；当它被移动后，要修改共享段的每个作业的段表中所指示的地址；如果让作业段表的共享段的表目指向共享段表，就可减少修改表目的工作量，在移动、调出和再装入时，只要修改共享段表即可。显然，如果采用直接地址方式，共享段的移进移出所引起的共享段表和各作业的段表的调整是十分复杂的。因此利用分段的动态连接，比较容易实现段的共享。

6. 分段的保护

实现信息的共享必须解决共享信息的保护。在分段存储管理一般采取如下三种措施实现存储保护：

（1）在段表设置段长指示值，以表明段的长度。当存储访问时，段地址中的相对地址与段长相比较，如超过段长，便发出越界中断。这样，每个作业被限制在自己的地址空间中运行，于是这些段表便把各个作业互相分开了，因为一个作业的所有存储访问都是通过段表来变换的，所以它们只涉及自己的地址空间，不存在一个用户作业破坏另一个用户地址空间的危险。

（2）设置存取控制：在段表的每个表目中，除指明段长外，再增加“存取位”项。这种段的保护，对于非共享段来说，主要是用来指示程序设计的错误。而对于共享段来说，如果是过程，则必须禁止任何作业修改它，若是数据段，只允许各作业去读它，而不能去写。当试图对过程进行读写或对数据段进行写时，系统将发出段保护中断。

（3）采用存储保护键：由于 I/O 通道对存储器的访问不经过段表，因此，除段保护外，还采用存储键保护。就是系统为每道作业设置一个保护键，为某作业分配内存时，根据

它的保护键在相应的段表中建立键标志，程序执行时将程序状态字中的键与访问段的键进行核对，相符时才可访问该段。

4.5.4 段页式存储管理

分段存储管理不仅具有逻辑上的清晰性和完整性，而且便于扩充、动态连接和实现共享。但它有一个缺点就是每个分段必须占据内存的连续空间，即一个分段不能比内存实际容量大。而分页存储管理在管理存储空间时，可以给作业一个比内存实际地址大得多的虚拟地址空间。这样，兼用分段和分页两种方法，即采用段页式存储管理。段页式存储管理是用分段的方法来分配和管理虚拟存储器，用分页的方法来分配和管理内存。这样，一方面可以保持分段地址空间所带来的优点，如允许分段动态扩展，可实现分段的动态连接，分段的共享和实施段的保护等；另一方面，内存分区的拼接问题，辅助存储器的管理以及分段大小的限制等问题都可得到解决。

1. 段表和页表

段页式管理是将用户作业按逻辑意义分段，每个段再划分为大小相同的页，而内存空间则划分成与页大小相同的存储块，并以块为单位分配内存。这样可以允许一个分段在内存中不必连续存放，也不必一次全部装入。作业地址空间的逻辑地址分为段号 s、页号 p 和页内地址 d 三部分，如下所示：

用户编写自己的作业仍使用段名和段内相对地址，即逻辑地址仍反映的是二维地址空间。由系统的地址变换机构，根据页的大小把段内相对地址变换成页号和页内地址，这个过程对用户是透明的。

为实现动态地址变换，对于每个作业都设立一个段表和若干个页表，段表中一般指出每段的状态、页表的起始地址和页表的长度。页表的长度是由段长和页面大小决定的，当页面大小确定后则由段长决定。页表一般包括状态、页号和块号。状态指明该页是否在内存中。块号说明该页所对应在内存中的块。另外，还有一个段表地址控制寄存器来指明段表的起始地址和段表长度。段表和页表的关系如图 4-18 所示。

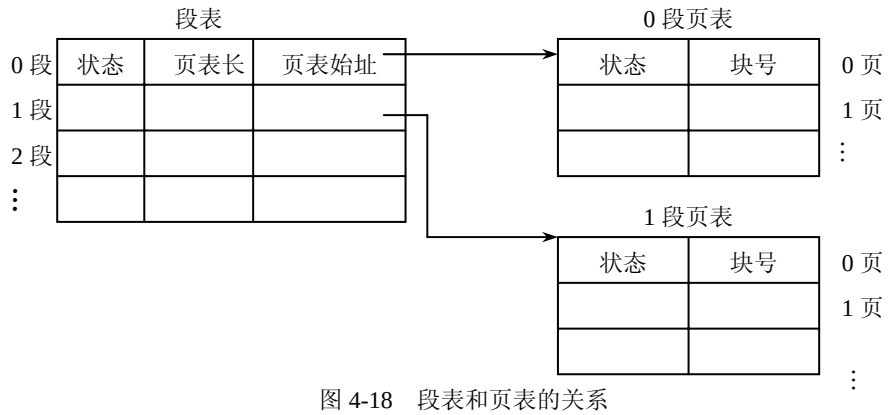

图 4-18　段表和页表的关系

2. 段页式存储管理的地址转换

当作业运行时，欲访问虚拟地址(s,p,d)，在没有相关存储器的情况下，地址转换过程如下：

（1）地址转换硬件将段表地址控制寄存器与指令地址域中的段号 s 相加，得到欲访问段 s 在该作业的段表中表目入口地址。

（2）从该表的表目中得到该段的页表起始地址，并将其与地址场中的页号 p 相加后得到欲访问页 p 在该段的页表中的表目入口地址。

（3）从该页表的表目中取出其对应的块号与指令地址场中的页内地址 d 拼成内存的实际地址。

在使用相关存储器的情况下，相关存储器的表目中应包括如下内容：段号、页号、块号、该段页表长以及存取控制信息等。每次地址转换时，同时进行直接映像和相关映像的地址转换工作。当相关映像地址转换成功即自动停止直接映像工作。相关映像的地址转换过程如下：

（1）以段号、页号为索引同时对相关存储器的各表目进行比较。

（2）如果匹配，则比较页表长度和存取控制信息以判断访问的合法性。若不合法，则停止地址转换工作，发出相应的中断信号。

（3）取出块号和页内地址 d 拼成内存的实际地址，然后对该地址进行访问。

（4）如果在相关存储器中查找不到该表目，则通过直接映像来找出块号，完成对内存的访问，同时将有关信息装入到相关存储器中。

段页式存储管理是分段存储管理和请求页式存储管理的结合，因此它具有两者的全部优点：提供虚拟存储器的功能；没有紧缩问题，也没有页外的碎片存在；便于处理变化的数据结构，分段可动态增长；容易实现共享；便于实现动态连接；提供更好的存取保护。

段页式存储管理的主要缺点是：增加了软件的复杂性和管理的时间；需要的硬件支持也增加了；系统存在着抖动的危险；和分页管理一样，存在页内碎片。

段页式存储管理对大、中型计算机来说是使用的最广泛最灵活的一种存储管理方法。

4.6 Windows Server 2008 的内存管理

Windows Server 2008 标准版的存储管理主要包括两个方面：一方面是转换或映射进程的虚拟地址空间到实际的物理内存，这样当线程读取或写入虚拟地址空间时，能保证访问正确的物理地址。另一方面当内存使用不足时，调度内存中的一些内容到磁盘上，并且在需要的时候再把它调入到物理内存中。

除了提供 64 位虚拟内存管理外，而且提供内存保护、共享内存、写时复制内存的功能，以及对应用程序使用大量的、稀疏的地址空间的支持。首先来研究 Windows Server 2008 内存管理器的结构和功能。

4.6.1 内存管理器的结构和功能

Windows Server 2008 的内存管理程序是 Windows Server 2008 执行体的主要组成部分之一，是 Windows Server 2008 的基本存储管理系统，位于 Ntoskrnl.exe 文件中。

1. 内存管理器的结构

内存管理器提供了一系列服务，来完成对内存的管理，内存管理器由以下组件构成：

（1）一组执行体系统服务程序，用于虚拟内存的分配、释放和管理。这些服务大多数都是以 Win32 API 或核心驱动程序接口形式出现。

（2）一个变换无效和访问错误陷阱处理程序，用于解决硬件检测到的内存管理异常事件，并代表进程将虚拟页面装入内存。

（3）运行在 6 个不同内核模式系统线程的描述表中的几个关键组件。

工作集管理器：优先级为 16，它每秒运行一次，当空闲内存低于某一界限时，便驱动所有的内存管理规则，例如工作集的维护和已修改页的写入。它与系统线程联合来执行实际的工作。

进程/堆栈交换程序：优先级为 23，用于执行进程和内核线程堆栈的换入和换出操作。在需要执行换入和换出操作时，通过内核中平衡设置管理器和在内核中的线程调度代码可以唤醒这个程序。

已修改页面写入器：优先级为 17，把修改列表中的无用的页写回到相应的页面调度文件中。当需要减少修改链表的大小时，将唤醒这个线程。

映射页面写入器：优先级为 17，把映射文件中无用的页写到磁盘。当需要减少修改链表的大小，或映射文件中某些页面在修改链表中超过了 5 分钟时，它将被唤醒。

废弃段线程：优先级为 18，负责系统高速缓存和页面文件的增加和减少。

零页线程：优先级为 0，使空闲列表内的页面清零，以便有足够的零页面满足将来的零页面需求。

2. 内存管理器的功能

内存管理器提供了一组系统服务来分配和释放虚拟内存、在进程间共享内存、映射文件到内存、刷新虚拟页到磁盘、恢复关于虚拟页面范围的信息、更改它们的保护和把它们锁入内存。

像其他的 Windows Server 2008 执行体服务一样，内存管理服务允许它们的调用程序提供进程句柄，以指示其虚拟内存将被操作的特殊进程。这样，调用程序就可以操作它自己的内存或操作另一个进程的内存。在一个进程中，可以创建另一个进程，并给自己操作子进程的虚拟内存的权利。因此父进程可以通过调用虚拟内存服务并以参数的形式传递句柄到子进程中来代表子进程分配、释放、读取和写入内存。子系统使用这个特性来管理它们的进程的内存，同时这个特性对于执行调试程序也很关键，因为调试程序必须能够读取和写入被调试进程的内存。

内存管理器提供的功能具体来说，包括如下五个方面：

（1）保留和提交虚拟内存。

Windows Server 2008 提供了一个可选择的两阶段方法来分配内存，应用程序可以首先保留地址空间，然后提交在那个地址空间的存储器到页面调度文件或映射一个映射文件的视口。保留内存是一种简单的方法，线程保留虚拟地址的一个范围以供将来使用。访问保留内存将导致一个访问侵犯，因为在内存中提交的内存没有包含数据。提交的内存是内存管理器使用的对应于磁盘存储器的内存，访问提交的内存，如果它没在物理内存中，页面将被调入到物理内存中，然后再进行访问。

进程地址空间中的页面可以是空闲、保留或提交的。可以回收或释放地址空间，回收和释放的差别在于，回收的内存仍然是保留的，但是释放的内存既不是提交的，也不是保留的，它是空闲的。

分两步保留和提交内存能够减少页面文件的使用。在 Windows Server 2008 下，保留内存是一种又快又占用资源少的操作，因为它不消耗任何页面文件空间或进程页面文件配额。所以需要被更新或构造的是相对较小的代表进程地址空间的数据结构。

保留，然后提交，对于需要大量和连续的内存缓冲区的应用程序是非常有用的。地址空间可以被保留，然后在需要的时候再提交，而不是为整个区域提交页面文件存储器。在创建线程时，就保留一个地址空间，大约 1MB，然后再提交两个页面，仅 4KB，一个用于初始页面，另一个作为保护页，并能够自动扩展。

（2）共享内存。

共享内存指对一个以上的进程使用同一段物理内存。例如，如果两个进程编译 C 程序，很明显编译器只被装入内存一次，这样当另外一个进程要调用它时，只需把第二个进程的虚拟地址映射到编译器所占据的物理页块就可以了。

在内存管理器中用于执行共享内存的基本单元称作“区域对象”。区域对象在内存管理器中用来把虚拟地址映射到磁盘页面上，不管它是在页面文件中还是在应用程序将要访问的其他文件中，它就像是在内存中。一个区域对象可以被一个进程或多个进程打开。区域对象可以被连接到在磁盘上已打开的文件或一串页面调度文件上。

区域对象可以引用比进程地址空间能够容纳的文件大得多的文件，这样，要访问区域对象，进程可以只映射它需要的区域对象的一部分，称为区域的一个视口，然后指定映射的范围。一个视口提供了进入这个共享的内存区的一个窗口，不同进程可以映射一个区域对象的不同视口。

用映射一个区域对象的一些视口可使进程访问很大的一些内存，否则该进程也许没有足够的虚拟地址空间来映射它们。例如我们可能使用一个很大的数据库，当使用该数据库时，就可以创建一个区域对象来包含整个数据库。当一个用户进程查询该数据库时，把该区域对象的视口映射到进程的虚拟地址空间，从中获得数据。查询完该视口，然后映射该对象的另一个视口。实际上 Windows Server 2008 通过区域对象和视口的方法每次打开一个窗口，从而可从数据库的每一部分获得数据，而不会超出进程虚拟地址空间。

内存管理器通过区域对象的不同视口的方法，使进程像在内存中访问一个大数组那样存取文件，这称为映像文件 I/O 活动。应用程序可使用映像文件方便地完成到文件的输入/输出。

（3）内存保护。

Windows Server 2008 为进程和操作系统本身提供内存保护，这样就没有用户进程在无意中破坏其他进程或操作系统的地址空间。通过以下四种主要的方法提供这种保护。

1）所有系统范围内的在内核模式下的组件使用的数据结构和内存交换区只能在内核模式下被访问，用户模式线程不能访问这些页面。如果它们试图这样做，硬件将产生错误，内存管理器会报告线程访问侵犯。

2）每一个进程都有一个独立的、专用的地址空间，禁止其他进程的任何线程访问，除非使用共享内存，在这种情况下只有共享页面能够被一个以上的进程查看。每次线程引

用一个地址，虚拟内存硬件和内存管理器共同工作，干预并变换虚拟地址到物理地址。通过控制虚拟地址如何变换，能够确保在一个进程内运行的线程不会错误地访问属于其他进程的页面。

3）除了提供虚拟地址到物理地址变换的隐含保护外，Windows Server 2008 所支持的所有处理机都提供了某些形式的硬件控制的内存保护，如读/写，只读等，这种保护的具体细节因处理机的不同而不同。例如，在进程地址空间内的代码页面被标记为只读，这样就禁止了用户线程对它们的修改。

4）共享内存区中的区域对象具有标准的访问控制列表（ACL），当进程试图打开共享内存的区域对象时会检查 ACL，这样就把对共享内存的访问限制在那些有适当权限的进程之内。当线程创建一个区域来包含一个映射文件时，安全性就会起作用。要创建该区域，线程必须至少对此文件具有读的访问权限，否则该操作将失败。

一旦线程成功为区域对象打开了句柄，它仍将服从前面所说的内存管理和基于硬件的页面保护。如果对页面级保护的更改不违反区域对象在 ACL 中的权限，线程就可以在区域内更改对虚拟页面的页面级保护。例如，内存管理器允许线程把只读区域的页面改变为具有写时复制访问权限，但不能具有读/写访问权限。允许写时复制权限是因为它不影响其他进程共享数据。

这四种主要的保护机制使 Windows Server 2008 成为坚固的、可靠的操作系统，使它不受应用程序错误的影响，并可以从应用程序错误中恢复。

（4）写时复制。

写时复制页面保护是内存管理器用来节约物理内存的优化技术。当进程映射包含读/写页面的区域对象的写时复制视口，不是在映射视口时同时制作进程专用副本，而是内存管理器延迟到页面被修改后才制作页面的副本。

在内存共享时，如果页面被标记为写时复制，那么任何一个进程中的线程写入该页面，就会产生内存管理错误。内存管理器观察到写操作是作用于写时复制页面，因此它不是以访问侵犯报告该错误，而是在物理内存中分配一个新的读/写页面，复制原始页面的内容到新页面，更新在这个进程中相应的页面映射信息来指向新的位置并取消异常，这样就产生错误的指令重新被执行。此时就可以成功地进行写操作，新复制的页面是进行写操作的进程专用的，并且对仍然共享写时复制页面的其他进程是不可见的。每个写到那个相同共享页面的进程都将得到它的专用副本。

写时复制被用于实现调试程序中的断点支持。例如，在默认状态下，代码页在开始时是只读的。然而程序员在调试程序时设置一个断点，调试程序就必须把断点指令添加到代码中。要完成这项工作，它首先要更改页面的保护为写时复制，然后更改指令。内存管理器立即为要在其线程中设置断点的进程创建代码页的专用副本，而其他的进程仍然继续使用未更改的页面。

写时复制是怠惰计算的一个例子，怠惰计算是内存管理器经常使用的。怠惰计算算法避免执行昂贵的操作，除非是在绝对需要时，否则如果不需要此项操作，在它上面就不会浪费任何时间。

（5）堆管理。

堆是一个或多个页面的区域，通过堆管理器提供一组函数可以把这个区域分开并分配成

较小的块。每个进程以一个默认进程堆开始，一般是 1MB，这个大小只是初始保留值，在需要的时候，它可以自动扩展。进程可以创建另外的专用堆。当进程不再需要专用堆的时候，可以恢复为虚拟地址空间。

为了从默认堆中分配内存，线程必须得到它的句柄。有了堆句柄以后，线程就可以从堆中分配和释放内存。堆管理器也为每个堆进行串行化分配和回收，这样多线程同时调用堆函数而不会破坏数据结构。

4.6.2 地址空间分布

虚拟存储管理实现了一个复杂而有效的虚拟存储系统，它为每个进程提供了一个很大的虚拟地址空间。为了做到这一点，Windows Server 2008 抛弃了所有基于 Intel 芯片的早期个人计算机都使用的分段模式所造成的每段 64K 的局限性，因为这个局限使得 32 位计算机难以发挥它的全部性能。同时 Windows Server 2008 的设计者认为，虚拟空间的线形模式比分段模式更与内存的实际结构相吻合，所以采用“请求分页的虚拟存储管理技术”，每个页面的大小是 4KB。

Windows Server 2008 的进程都有 4GB 大的虚拟地址空间，这 4GB 的地址空间被等分成两部分，虚拟地址空间中高地址的 2GB 空间保留给操作系统使用，而低地址的 2GB 空间为用户存储区，可被用户进程使用。地址空间的分布情况如图 4-19 所示。

图 4-19　虚拟地址空间分布

用户地址空间主要为应用程序代码、用户数据、每个线程的堆栈和动态连接库（DLL）代码等提供地址空间，范围是 00000000H～7FFFFFFFH，大小是 2GB，其中 00010000H～7FFEFFFFH 是进程地址空间，其余 192KB 的地址空间为线程环境块、进程环境块和地址保护区等使用。另外，系统提供系统变量来表示用户地址空间，也可以使用进程性能计数器获得进程的地址空间使用情况。

可以使用“性能监视器”检查不同进程的内存性能计数器，也可以使用其他的实用程序来查看进程的物理和虚拟内存使用情况。

系统地址空间为操作系统提供地址空间，范围是 80000000H～FFFFFFFFH，大小是2GB，其分为三部分，第一部分是 80000000H～C0000000H，为直接映射区，包括系统代码和系统映射视图等，系统代码中包含用来引导系统的操作系统映像、HAL 和设备驱动程序，系统映射视图包含用于映射 Win32 子系统的可加载内核部分 Win32k.sys，以及它使用的内核图形驱动程序。这一区域的寻址是由硬件直接变换的，而且这些页面常驻内存，其存取速度特别快，从其内容中看出，它存放的是 Windows Server 2008 内核中频繁使用、响应速度块的代码。第二部分是 C0000000H～C0800000H，为页交换区，包括进程的页表和页目录、超空间、进程工作集列表等。第三部分是 C0800000H～FFFFFFFFH，为固定页面区，其中一些地址空间是固定的，一些地址空间是在系统启动时，根据内存的大小和系统的运行情况而计算出来的，所以它们是变化的，固定页面区用以存放永不被换出内存的页面，这些页面中存放系统需常驻内存的代码。另外，可以用系统变量来表示系统地址空间。

4.6.3 地址变换

我们已经了解了 Windows Server 2008 是怎样构造 32 位虚拟地址空间的，下面我们就来看看它怎样把这些地址空间映射到实际的物理页面中。

我们知道，用户应用程序引用 32 位虚拟地址，CPU 利用内存管理器来创建和维护的数据结构把虚拟地址转换为物理地址。现在通过一个例子来说明映射方法，如图 4-20 所示，说明了将三个连续的虚拟页面映射到三个不连续的物理页面上。从图中可以看出，虚拟页面和物理内存之间是间接关系，虚拟地址不直接映射到物理地址，每个虚拟页面和系统空间中的页表项（PTE）有关系，页表项中包括虚拟地址被映射到的物理地址。

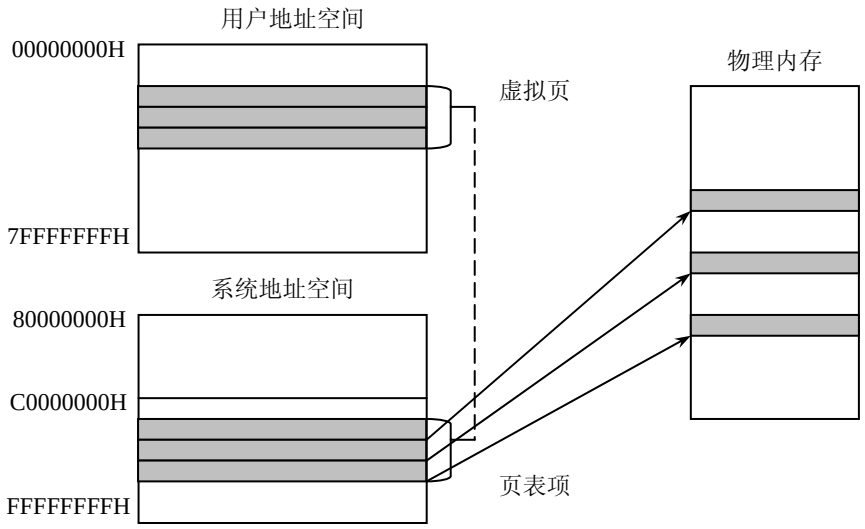

图 4-20　虚拟地址和物理地址的映射关系

1. 地址变换过程

Windows Server 2008 的地址变换机构和传统的页面地址变换机构是不同的，它是采用二级页表结构将虚拟地址变换位物理地址。32 位的虚拟地址解释为三个组成部分：页目录位

移、页表位移和字节位移。利用它们来实现从虚拟地址到物理地址的定位。在一般情况下，页目录位移为 10 位，页表位移为 10 位，字节位移为 12 位，字节位移 12 位正好和一个页面大小 4KB 相对应，其结构如图 4-21 所示。

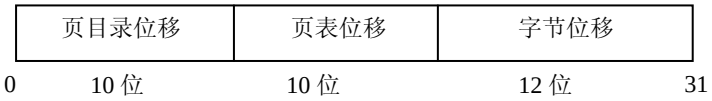

图 4-21　32 位虚拟地址的结构

在二级页表结构中，页目录用来定位页表，它是第一级表，每个进程一个页目录，每个页目录包含 1024 个表目，每个页目录表目指出其第二级页表所在的页块号（或称页表地址）。页表用来定位页表项，页表项中包含页块号，即页所对应的物理地址，它是第二级表，也包含 1024 个表目，每个表目是 4 个字节，恰好能表示一页的空间。虚拟地址转换为物理地址的机构如图 4-22 所示。

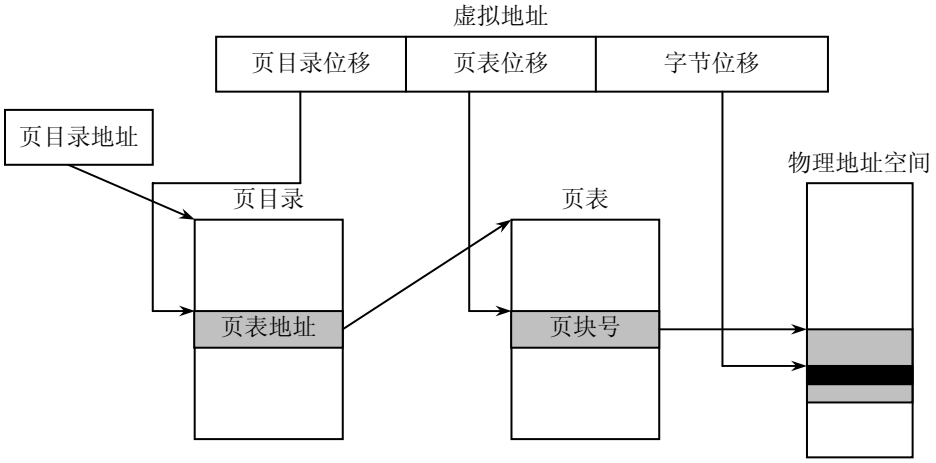

图 4-22　地址变换机构

虚拟地址转换为物理地址的基本步骤如下：

（1）内存管理器硬件定位当前进程的页目录。在进程运行时，一般是通过操作系统设置一个专用的 CPU 寄存器来通知硬件新进程的页目录地址。

（2）当确定页目录的地址后，页目录位移用来在页目录中定位指定的页目录项，页目录项描述了映射虚拟地址时需要的页表位置。

（3）页表位移用来在页表中定位页表项的位置，页表项中表示了要访问的虚拟页面的位置。

（4）在通过页表项来定位页面时，如果页面有效，则这个页面将对应物理内存中的物理页面号码。如果无效，则内存管理器故障处理程序将定位页面并试图使页面有效。如果不能使页面有效，故障处理程序将产生一个访问侵犯或错误检查。

（5）如果页表项定位的页面是有效的，字节位移就可以定位在物理页面内所需数据的地址，从而可以访问数据。

Windows Server 2008 为什么要采用二级页表结构呢？这时因为每个进程的虚拟地址空间太大了，它的 32 位地址使得一个进程有 4GB 的虚拟地址。每页大小为 4KB，每个进程的地

址空间可有 1M 个页面。也就是说每个进程的页表有 1M 个表目，每个表目占 4 个字节，一个进程的全部页表就可能要占 1024 个页面。而且每个进程都有独立的地址空间，这将是多么大的内存开销。所以为避免把所有内存都消耗在页表上，它的虚拟存储管理程序不把所有页表放在内存中，而是根据需要把这些页表换入、换出内存。

2. 页目录

每个进程都有一个单独的页目录，它是内存管理器创建的特殊页面，用于映射该进程的所有页表的位置。进程页目录的物理地址存储在核心进程块中，同样，它也被虚拟地映射到进程的虚拟地址中，这是因为运行在内核模式的代码几乎全部都能引用虚拟地址，而不是物理地址。

CPU 之所以能知道页目录页面的位置，是因为在 CPU 中有一个特殊的寄存器，它是由包含页目录物理地址的操作系统加载的。每次当产生线程时，这个寄存器通过在内核中的例程从核心进程块中加载。

页目录由页目录项构成，每一个页目录项的长度是 4 个字节，并且描述了用于其所属进程的所有可能的页表的状态和位置，因为页表是根据需要建立的，所以对于大多数进程来说，页目录只是指向页表的一个小集合。

3. 页表

在页面内用字节位移来定位一个字节之前，CPU 首先需要能够查找包含所要求数据字节的页面。要做到这一点，操作系统就要构造包含映射信息的另一个内存页面，此页面包含查找含有所需数据的页面所需要的映射信息。这个含有映射信息的页面称为页表。因为 Windows Server 2008 为每一个进程都提供一个专用的地址空间，每一进程都有其自己的进程页表集来映射其专用的地址空间，所以每个进程的映射情况都是不相同的。

页表是由一个页表项数组构成的。它有 1024 个表目，每个表目的长度为 4 个字节。有效的页表项（PTE）有两个主要的字段：包含数据的物理页面的页块号或在内存中的某个页面的物理地址的页块号以及一些描述页面状态和页面保护的标志。有效页表项的结构如图 4-23 所示。

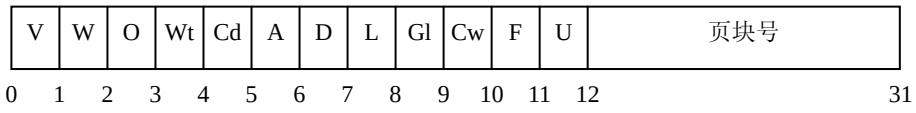

图 4-23　有效页表项的构成

第 0 位 V 称为“有效”位，它表示该表目是否有效，也就是该表目是否映射到了物理内存中的页面。

第 1 位 W 称为“写入”位，在单处理机系统中，表示页面是可以读写的还是只读的，在多处理机系统中，表示页面是否是可写的。

第 2 位 O 称为“所有者”位，表示用户模式下的代码是否可以访问该表目指向的页面，也就是说页面是否被限制在内核模式下访问。

第 3 位 Wt 称为“写通”位，表示在写入此页面时，禁止使用高速缓存，所以该页面发生变化时应立即被刷新到磁盘上。

第 4 位 Cd 称为“禁止”位，表示禁止使用对应于该页的高速缓存中的页面。

第 5 位 A 称为“访问”位，表示该页面已经被读取。

第 6 位 D 称为“重写”位，如果设置该位，页面就将被写入；如果清除该位，页面就处于只读状态，而不能写入，如果线程此时试图写入页面，将会出现内存管理异常。

第 7 位 L 称为保留位，在页表项无效时使用。

第 8 位 Gl 称为“共用”位，表示该表目应用于所有的进程。

第 9、10、11 位都是保留位，在页表项无效时使用。

第 12～31 位为该表目所对应的页块号。

在这 32 位中，“写入”位、“访问”位、“重写”位含有页面的保护和访问信息，如果页表项所代表的物理页面还未读出或写入，“访问”位将被清除；如果页面是初次读入或写入的，处理机就会设置该位。只有页面被初次写入，处理机才设置“重写”位。另外，对“写入”位清除时，页面处于只读状态；设置该位后，页面就是可读写的。

4. 变换查找缓冲区

我们知道每次地址变换都要进行两次查找，一是查找正确的页表是否在页目录中，另一个是查找正确的项是否在页表中，最后才从内存中存取数据。这样每从内存中访问一个数据，需要三次访问内存。这样采用二级页表结构会带来访问速度变慢的问题。实际上 Windows Server 2008 访问内存的速度并不慢。这是因为大多数的 CPU 都会高速缓存地址变换，重复访问同一个地址不需要重新变换。

在 Windows Server 2008 系统中，同样以联合数组的形式提供了一种称为变换查找缓冲区的高速缓存。变换查找缓冲区就是我们在请求页式存储管理中所说的“快表”，它的单元可以被同步地读出和目标值相比较。在使用变换查找缓冲区时，其中包含大多数最近已使用的页面从虚拟地址到物理页面的映射和应用于每一个页面的页面保护类型，具体地说，它的标志部分保存了虚拟地址，它的数据部分保存了一个物理页面的号码、保护字段、有效位和重写位等，这些用来指出与它的页表项相符的页面条件，另外，也可以设置共用位，那么就可以在多个进程中使用。访问变换查找缓冲区的形式如图 4-24 所示。

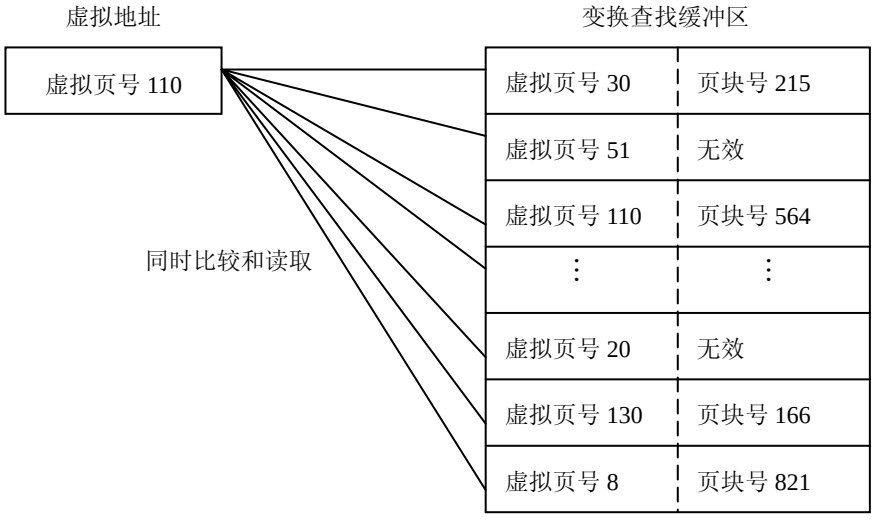

图 4-24　访问变换查找缓冲区

从图中可以看出，当访问虚拟地址 110 号时，与变换查找缓冲区中的每一项同时比较和读取，快速找到虚拟地址 110 号所对应的物理页块号 564，这样就实现了虚拟地址到物理地址的变换，接着根据变换的结果访问物理内存的数据单元就实现了对内存的访问，得到需要的数据。一般情况下，经常使用的虚拟地址很可能在变换查找缓冲区中，这就提供了非常快的虚拟地址到物理地址的变换和快速访问内存。如果虚拟地址不在变换查找缓冲区中，它可能仍然在内存中，这时需要使用二级页表机构来查找它，然后进行数据访问。如果虚拟页面被调出了内存或者如果内存管理器改变了页表项，导致变换查找缓冲区中的单元无效，在某个进程再次访问变换查找缓冲区时，将发生页面错误，那么内存管理器会把页面调入到内存中，然后在变换查找缓冲区中为该页面重新建立相应的单元。

5. 地址变换的实例

要弄清地址变换是如何工作的，以虚拟地址 00050001H 为例，看一下它怎么变换为相应的地址。

首先将 00050001H 变换为二进制，然后将它分隔成用于变换一个地址所需要的三个部分。在二进制状态下，00050001H 是 0000 0000 0000 0101 0000 0000 0000 0001。分隔成三个部分，其中页目录位移为 0H，页表位移为 50H，字节位移为 1H，如图 4-25 所示。

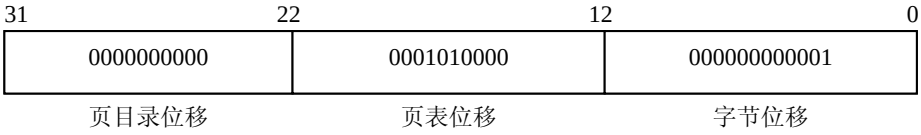

图 4-25　虚拟地址结构实例

当某个进程在运行时，如果要进行地址变换，CPU 要求有进程的页目录的物理地址，它存储在 CR3 寄存器中，可以通过检测 CR3 寄存器来显示该地址，假设 CR3 寄存器中的数据是 012F0000。

在这种情况下，页目录被存储在物理地址 012F0000H 中，由于页目录位移为 0，因此页目录项的物理地址是 012F0000H。

进程页目录在虚拟地址空间中是从 c0300000H 排列的，因为查看的是第一个页目录项，所以其虚拟地址是 c0300000H。

页表项在虚拟地址 c0000140H 中。可以使用页表位移乘以页表项的大小来计算出来页表项的虚拟地址：4×50H=140H。因为内存管理器从虚拟地址 c0000000H 开始建立页表，这样可以得到页表项的虚拟地址 c0000140H。

4.6.4　页面错误处理

我们已经知道，当页表项 PTE 有效时如何解决地址变换。如果清除了 PTE 的有效位，就表明所需的页面因某种原因使此进程不能进行访问，从而形成了无效的 PTE。下面描述无效 PTE 的类型以及如何解决对它们的引用。

对无效页面的引用导致页错误。内核陷阱处理程序调度这种错误到内存管理器错误处理程序中解决。这个例程运行在引起错误的线程的描述表中，并且负责解决这个错误或产生相应的异常报告。产生错误的原因很多，在表 4-1 中列出了主要的错误和处理结果。

表 4-1　访问错误的原因及结果

错误原因	结果
访问一个没有驻留在内存中但在磁盘上的页面文件或映射文件中的页面	分配一个物理页面，并将所需的页面从磁盘上读取到进程工作集中
访问一个在备用或更改列表中的页面	将页面变换到进程或系统工作集中
访问一个没有占用存储器的页面，如保留的地址空或未被分配的地址空间	访问侵犯
从用户模式访问一个只能在内核模式下被访问的页面	访问侵犯
写入一个只读的页面	访问侵犯
访问一个请求零页面	添加一个由零填充的页面到进程工作集中
写入一个保护页面	保护页侵犯
写入一个写时复制页面	制作进程专用页面的副本，并替代在进程或系统工作集中的原始页面
引用一个在系统空间中但不在进程页目录中的页面	从主系统目录结构复制页目录项
在多处理机系统中，写入一个有效但尚未写入的页面	在 PTE 中设置页面重写标志位

1. 无效的 PTE

无效的 PTE 在各种情况下是类似的，但不完全相同。下面来看几种情况下的无效的 PTE 结构。

如果所需的页面驻留在页面调度文件中，其 PTE 的结构如图 4-26 所示。

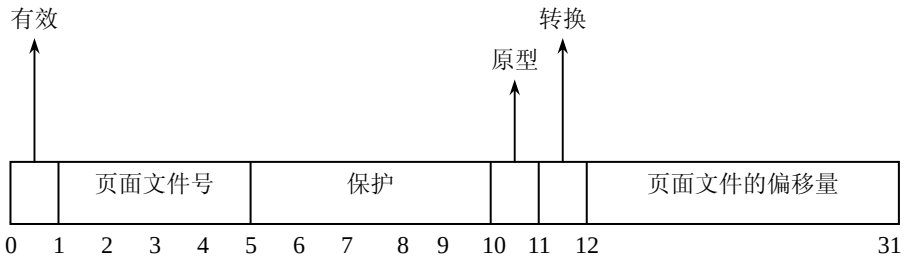

图 4-26　页文件中无效的 PTE

当需要零页面时，页面调度程序将查看零页面列表，如果列表是空的，页面调度程序就从备用列表中取出一个页面并且把它置零。此时 PTE 格式与上面所示的页文件的 PTE 相同，只是页面文件号和页面文件偏移量为零。

所需的页面在内存中的备用、更改或更改不写入列表中，从列表中删除此页并添加到工作集中，这时 PTE 的结构如图 4-27 所示。

当 PTE 的数目为零，或页表不存在，这意味着所引用的虚拟地址是被保留的，应该建立页表来表示新提交的地址空间。

2. 原型 PTE

如果一个页面被两个以上的进程所共享，内存管理器将建立原型 PTE 页表项来表示共享的页面。当首次创建区域对象时将创建原型 PTE。

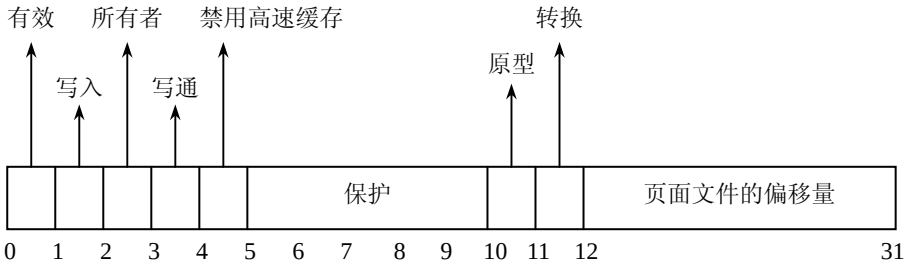

图 4-27　备用列表中的无效的 PTE

当进程首次引用一个映射到区域对象视口的页面时，内存管理器使用原型 PTE 的信息填入在进程页面中用于地址变换的实际的 PTE 中。

当共享的页面有效时，进程 PTE 和原型 PTE 都指向包含数据的物理页面。为了表示引用有效共享页面的进程 PTE 的数量，在页块数据库的每一项中增加了一个计数器。这样，内存管理器就可以决定何时一个共享页面不再被任何页表引用，这样就可以把它置为无效并移动到相应列表中或写到磁盘上。

当使一个页面无效时，用一个特殊的 PTE 写入进程页表中的 PTE，这个特殊的 PTE 指向描述此页面的原型 PTE，其结构如图 4-28 所示。

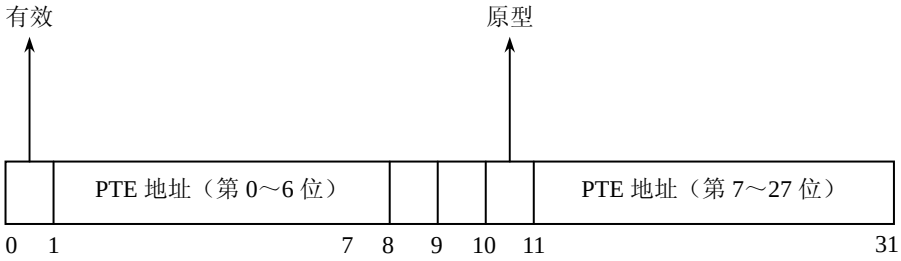

图 4-28　指向原型 PTE 的无效 PTE 的结构

这样，在以后访问此页面时，内存管理器就能够使用在这个 PTE 中的信息来定位原型 PTE，这个 PTE 描述被引用的页面。

虽然这些原型 PTE 项的格式和真实的 PTE 项相同，但是这些原型 PTE 不用于地址变换，它们是页表和页块数据库之间的一层，并且从不直接在页表中出现。

通过把一个共享页面的所有访问程序指向一个原型 PTE 来解决错误，内存管理器可以管理共享页面而不需要更新每个共享此页面进程的页表。例如，一个共享的代码或数据页面在某时会被调出并放入到磁盘上。当内存管理器从磁盘取回这个页面时，它只需更新原型 PTE 来指向页面的新的物理位置，每个共享此页面的进程的 PTE 都保持原样。

4.6.5　页面调度策略和工作集

在进程执行时，并不是把所有的页面都调入到物理内存中，开始只是调入一小部分，然后根据需要再进行调入，这样，必须制定一定的策略来决定哪些页面将被调入物理内存并保留多长时间。另外，对调入到物理内存的页面需要进行管理，这样就出现了“工作集”这个概念，它是用来描述驻留在物理内存中的虚拟页面的集合。它分为两种：进程工作集和系统工作集。

1. 页面调度策略

虚拟内存系统通常定义三种策略来规定如何或何时执行页面调度操作，分别是取页策略、置页策略和淘汰策略。

取页策略决定什么时候页面调度程序把一个页面从磁盘调入内存。有一种取页策略是把线程需要的页面在使用之前装入内存。另一种取页策略是“按需页面调度策略”，它是仅当页错误发生时才将所需的页面调入物理内存。在按需页面调度系统中，当进程的线程第一次开始执行时，进程会引发许多页错误，因为线程需要得到一些初始页面才能运行，一旦这些页面被调入内存，该进程的页面调度活动就会减少。

内存管理器使用按需页面调度算法及“集群”方法把页面调入内存。当线程收到错误时，内存管理器把出错页面和它附近的一些页面装入内存。这个策略是为了把线程引发的页面调度 I/O 的数量减到最小。因为有一些程序，特别是大的程序，它在任何给定的时间内都只在它们地址空间的小的区域内执行，因此加载几簇虚拟页面会减少读取磁盘的次数。使用这种集群方法提前取页使缺页情况大大减少。

当线程收到页错误时，内存管理系统也必须确定在物理内存中放置虚拟页面的地方。为此使用一组规则，即置页策略。一般是找到一个未分配的页块进行分配即可。

如果当页错误发生时物理内存满了，使用淘汰策略来决定哪一个虚拟页面必须从内存中删去来为新的页面腾出空间。常用的淘汰策略包括最近最少使用（LRU）和先进先出（FIFO）策略。最近最少使用算法要求虚拟内存系统跟踪内存中的页面何时被使用，当需要新的页块时，最长时间没有使用过的页面被调入到磁盘上，它的页块被释放以满足页错误。先进先出算法简单一些，它把在物理内存中驻留时间最长的页面删除，而不管它使用得多么频繁。

淘汰策略可以进一步按特性分为全局的和局部的。全局的淘汰策略允许淘汰任何页块，而不管那些页块是属于哪一个进程的。例如，全局的淘汰策略使用先进先出算法在内存中寻找驻留时间最长的页面并且释放它来满足页错误。局部淘汰策略把对最长页的搜索限定在引发页错误的进程所拥有的页面中。全局的淘汰策略使进程易受其他进程的侵犯，一个具有有害行为的应用程序能够通过在所有进程中引起过量的页面调度而破坏整个操作系统。

在多处理机系统上，实现局部 FIFO 淘汰策略的形式。在单处理机系统上，实现更接近于最近最少访问策略的方法。分配给每个进程一定数量可动态调节的页块，称为进程工作集。当有进程需要物理内存，而工作集达到它的界限或需要调整时，内存管理器将从工作集中删除页，直到它确认有足够的空闲页面为止。

2. 工作集

内存管理系统为每一个进程分配固定数量的页面，并可动态地调整这个数量。那么这个数量如何确定？又如何动态调整呢？这个数量就用每个进程的工作集来确定，并且根据内存的负荷和进程的缺页情况动态地调整其工作集。

每个进程都以同样的默认工作集的最大值和最小值开始，在系统初始化时根据物理内存的大小来计算工作集的最大值和最小值，物理内存的大小与工作集的最大值和最小值的关系如表 4-2 所示。

表 4-2　进程最小和最大工作集的默认值

内存大小	默认的最小工作集（页）	默认的最大工作集（页）
≤19MB	20	45
20～32MB	30	145
≥32MB	50	345

当页错误发生时，检查进程的工作集限制和系统上的空闲页块数。如果条件允许，内存管理器允许进程把它的工作集增加到最大值，如果有足够的空闲页面就可以超过这个最大值。如果进程达到了它的限制并且要求更多的页面时，内存管理器将为该进程产生的每个新的页错误删除该进程页面中的一页。

当物理内存变少时，内存管理器使用被称为“工作集自动调整”的技术来增加系统中有效空闲内存的数量。工作集自动调整程序将检查内存中的每个进程，将它的当前工作集大小与它的最小工作集数值比较。当进程正在使用的页面数大于它的最小值时，工作集自动调整程序将从进程工作集中删除一些页面，使这些页面能被其他的进程使用。如果空闲的内存还是太小，内存管理器就继续从进程的工作集中删除页面，直到它达到系统中空闲页面的最小值。如果由于进程在上次调整后引起了很多页错误，此进程将不被调整。由于这个原理，如果内存管理器调整的页面是正被使用的页面，它会停止调整直到下一个调整循环。

在单处理机系统中，内存管理器试图删除最近不被访问的页面。它是通过检查硬件页表项中的访问位来查看此页是否被访问。如果此位是清除的，此页将从工作集中删除。如果此位已被设置，内存管理器将清除它并继续检查工作集中的下一页。以这种方式，如果下一次工作集管理器检查这个页时已访问位是清除的，它就能知道这个页面从上次检查后未被访问。用于页面的这个扫描在工作集列表中一直进行删除操作，直到删除了所需的页面数或扫描回到了开始点。每次扫描时，将从它上次离开的地点重新开始。

在多处理机系统中，工作集管理器不检查访问位，因为要清除此位将导致其他处理机上的变换查找缓冲区的地址无效。这样，页面将从工作集中被删除而不管访问位的状态。

同进程工作集一样，可以由系统工作集来管理操作系统中可分页的代码和数据。在系统工作集中可驻留五种不同的页面：系统高速缓冲页面、页交换区、Ntoskrnl.exe 中可分页的代码、设备驱动程序中可分页的代码和系统映射的视口。

系统工作集大小的最大值和最小值是在系统初始化时，根据物理内存的大小和系统高速缓存的大小等计算的。仅根据物理内存的大小，系统工作集的最大值和最小值的大小如表 4-3 所示。

表 4-3　系统最小和最大工作集的大小

内存大小	最小工作集（页）	最大工作集（页）
≤19MB	388	500
20～32MB	688	1150
≥32MB	1188	2050

当系统根据物理内存的大小确定系统工作集的最大值和最小值的初始值后，检查系统工

作集的最大值是否大于系统高速缓存的虚拟大小，如果大于它，工作集就要缩小为系统高速缓冲的虚拟大小。

3. 平衡集管理器和交换程序

工作集的扩充和调整发生在称为平衡集管理器的系统线程描述表中。平衡集管理器是在系统初始化时创建的。它通过调用内存管理器中的工作集管理器来执行工作集的分析和调整。

平衡集管理器根据两种不同的事件对象进行对工作集的扩充和调整：一种是当一个设置为每秒激发一次的周期计时器到期后被设置为有信号态的事件；另一种是内部工作集管理器事件，当内存管理器确定需要调整工作集时，它就可以在不同的时候发出信号。例如，如果系统遇到高的页面出错率或空闲列表太小时，工作集管理器唤醒平衡集管理器，以便能够调用内存管理器开始调整工作集。如果内存很多时，工作集管理器通过把出错页调入内存的方法允许出错进程逐渐增加它的工作集大小，但是工作集仅根据需要增加。

如果平衡集管理器被其自身的 1 秒计时器到期而唤醒，就执行如下操作：

（1）平衡集管理器每唤醒四次就会为一个事件设置为有效，从而唤醒另一个称为交换程序的系统进程。

（2）检查后备链表，如果有必要，为了减少访问时间，就调整此链表。

（3）寻找当 CPU 处于等待状态而需要通过提高其优先级而执行的线程。

（4）调用内存管理器的工作集管理器。工作集管理根据其内部的计数器，来决定何时执行工作集的调整和如何快速调整。

交换程序由在内核中的调度代码唤醒，把线程的内核堆栈交换出内存，或者包含线程的进程把它的工作集交换出内存。交换程序寻找在一段时间内处于等待状态的线程，当找到时，交换程序就标记将要被交换到页面映射文件的线程内核堆栈，以便回收它的物理内存。所以如果线程已经等待了很长时间，它将会等待更长的时间。当进程中的最后一个线程把它的内核堆栈从内存中删除时，进程工作集就会全部交换出内存，这样就会导致长时间等待的进程具有零工作集大小的情况。

4.6.6 物理内存管理

1. 页面状态

虽然工作集描述了进程或系统所拥有的驻留页面，但实际描述物理内存中每个页面状态的是页块数据库。为了描述每个页面，将页面分成 8 种状态，如表 4-4 所示。

表 4-4 页面状态

状态	说明
有效	页面是工作集的一部分，并且一个可用的页表项指向它
过渡	页面的临时状态，工作集不拥有此页面且它不存在于任何页面调度列表中，当该页面的 I/O 正在进行时，该页面就处于这种状态
备用	以前属于工作集但已经被删除。从页面最后驻留内存以来，没有被修改过，该页面被标记为无效，并处于过渡状态。当进程缺页并需要该页内容，则不必从磁盘上读入内存，重新将该页分配给它，并置为有效

续表

状态	说明
修改	类似于备用状态，只是该页中的数据被修改过，并且它的内容尚未写入磁盘中的对应虚页
修改不写入	类似于修改页面，内存管理器的修改页面写入程序不把它写入磁盘
空闲	页面处于空闲状态，但其中包含无用的数据，即页面还没被清零，还不能被用户进程使用
清零	页面处于空闲状态，且已经被零页面线程初始化为零页面
坏页	产生奇偶校验错误或其他硬件错误而不能使用的页面

2. 页块数据库

页块数据库由结构数组组成，该数组从 0 到系统中内存的物理页面的数量来编号。在系统的每个进程的页表都与页块数据库相联系，来得到每页的块号和状态。页块数据库和页表的关系如图 4-29 所示。在图中可用的页表项指向页块数据库中的项，并且页块数据库中的项指向正在使用它们的页表。

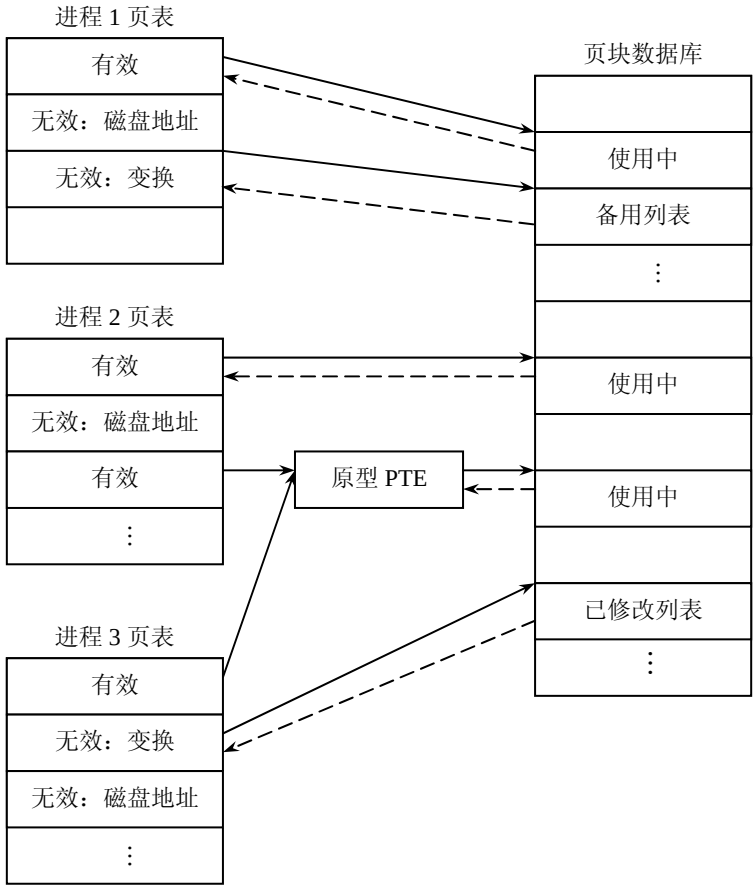

图 4-29　页块数据库和页表的关系

在页面的状态中，除了“有效”和“过渡”页面不在任何页面页表中，其他 6 种状态被

以链表的形式组织在一起构成页面列表，以便内存管理器能够迅速定位指定类型的页面。页块数据库的页面列表的连接情况如图 4-30 所示。

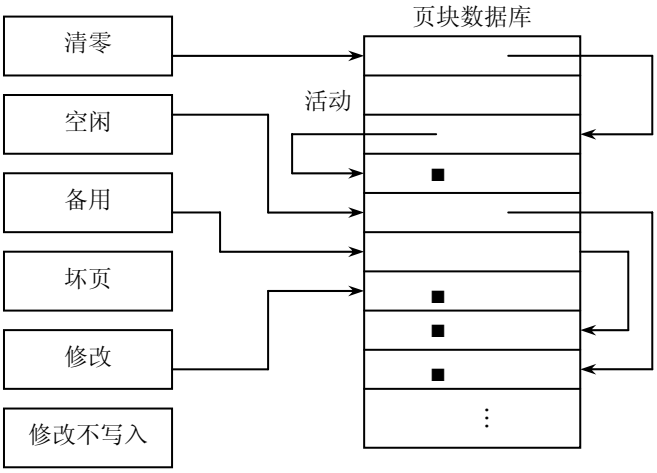

图 4-30　页块数据库中的页面列表

3. 页面状态的变换

在页面调度过程中，随着页面的调入和调出，页块状态也跟着的变化，而造成在页面列表中移动。它的变换情况如下所述。

当内存管理器需要一个零初始化的页面来为要求零页面错误服务时，它首先试图从零页面列表中得到一个页面，如果该表为空，它就从空闲页面列表中得到一个页面，并用零初始化该页面。如果空闲列表为空，它就会转到备用列表，并用零初始化页面。

需要把页面零初始化的原因是安全性的要求。用户模式进程必须得到初始化过的页块，以免读出先前进程在内存中的内容。因此，内存管理器给用户模式进程零初始化的页块，除非页面正在从映射文件读入。如果是这种情况，内存管理器就要使用非零初始化的页块，用磁盘中的数据初始化这些页块。

当内存管理器不需要零初始化页面时，它将首先转向空闲列表，如果空闲列表为空，它就转到备用列表。在内存管理器能够在备用或更改列表中使用页块之前，它必须首先从仍然指向它的无效 PTE 中返回并删除引用。因为在页块数据库中的项含有指向先前用户的页表的指针，内存管理器能够迅速地查找 PTE 并做出适当的更改。

进程引用新的页面并且它的工作集已满，或内存管理器调整了它的工作集，或进程退出了等，这时它不得不从它的工作集中放弃一页，如果页面没有被更改过，页面就进入备用列表，如果页面在驻留期间被更改了，它就进入更改列表。

当更改列表变得过大，或零列表和备用列表的大小低于最小值时，就要激活更改页面写入程序将页面写到磁盘，并将页面移入备用列表。

页块状态的变换如图 4-31 所示。

4. 页块项（PFN）的数据结构

虽然页块项有固定的长度，但是它们有几种不同的状态，这主要与页面的状态有关。主要有四种 PFN 结构。

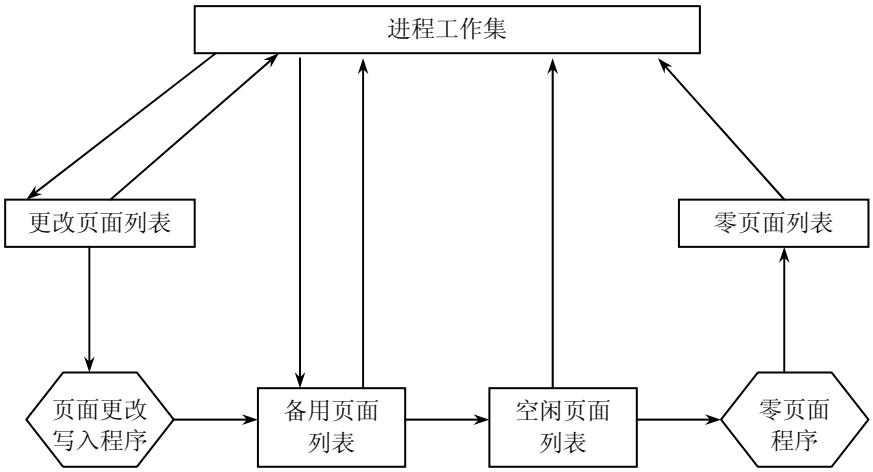

图 4-31　页块状态变换图

（1）工作集中的页面的 PFN 结构，如表 4-5 所示。

表 4-5　在工作集中的页面的 PFN 结构

工作集索引		
页表项（PTE）的地址		
共享计数		
标志	类型	引用计数
初始的页表项（PTE）的内容		
页表项（PTE）的页块地址		

工作集索引：这是一个进程工作集列表的索引，在工作集列表中驻留着映射这个物理页面的虚拟地址。如果该页面是专用页面，工作集索引就是直接在工作集列表中的项。如果是共享页面，工作集索引是一种提示，它只保证对使页面有效的第一个进程是正确的。

页表项（PTE）的地址：指向该页面的 PTE 的虚拟地址。

共享计数：表示指向页面的 PTE 的数量。

引用计数：该页面的引用数量。当一个页面被首次添加到工作集中或当用于 I/O 的页面被锁定在内存中时，引用计数就要增加。当共享计数为 0 或页面从内存中解锁时，引用计数将减少。当引用计数变为 0 时，工作集就不再拥有该页面，并且根据它的状态更新描述此页面的 PFN 项以把页面添加到空闲、备用或更改列表中。

类型：这个 PFN 代表的页面的状态类型，即有效、过渡、备用、修改、修改不写入、空闲、清零和坏页等。

标志：表示该页块的工作情况，是否被修改，奇偶校验是否错误、读取和写入是否正在进行，是否用于非页交换区和在页面操作时是否有错误等。

初始的页表项（PTE）的内容：所有的 PFN 项都包含该页面的 PTE 的原始内容，通过保存 PTE 的内容，允许当物理页面不再驻留时恢复它。

页表项（PTE）的页块地址：包含指向这个 PTE 的页表页的物理页面地址。

（2）在备用或更改列表中的页面的 PFN 结构，如表 4-6 所示。

表 4-6　在备用或更改列表中的页面的 PFN 结构

向前的链接		
页表项（PTE）的地址		
向后的链接		
标志	类型	引用计数
初始的页表项（PTE）的内容		
页表项（PTE）的页块地址		

这种结构用于备用或更改列表中的页面，在这种情况下，向前和向后的链接就把列表中的元素在列表内链接在一起。这种链接允许很容易地操作页面以满足页错误。当页面处于这些列表中的一个时，共享计数已经被向后链接替代，引用计数可能不为 0，因为用于这个页面的 I/O 可能正在进行，例如，页面正被写到磁盘。

（3）在置零或空闲列表中的页面的 PFN 结构，如表 4-7 所示。

表 4-7　在置零或空闲列表中的页面的 PFN 结构

向前的链接		
页表项（PTE）的地址		
颜色链 PFN 地址		
标志	类型	引用计数
初始的页表项（PTE）的内容		
页表项（PTE）的页块地址		

这种结构用于空闲或清零列表上的页面。除了在这两种列表内链接以外，使用颜色链 PFN 项还指向它们在处理机内存高速缓存种的位置即“颜色”。通过把虚拟的连续页面映射到在高速缓存中的物理上连续的页面，从而把 CPU 中内存高速缓存的不必要的变化减到最小。对于直接映射高速缓存的处理机，利用硬件的能力进行优化可以显著地提高系统的性能。

（4）正在进行 I/O 的页面的 PFN 结构，如表 4-8 所示。

表 4-8　正在进行 I/O 的页面的 PFN 结构

事件的地址		
页表项（PTE）的地址		
共享计数		
标志	类型	引用计数
初始的页表项（PTE）的内容		
页表项（PTE）的页块地址		

这种结构用于 I/O 正在进行中的页面。在进行 I/O 处理期间，“事件的地址”指向当 I/O 完成时被设置为有信号态的事件对象，如果出现页错误，该部分含有表示 I/O 错误的错误状态代码，这样可用于解决冲突页错误。

存储管理是操作系统的重要组成部分，目的是尽可能地方便用户和提高内存的效率。存储管理主要完成四个功能：内存的分配和管理、内存空间的共享、存储保护和存储扩充。

覆盖技术和交换技术都属于内存扩充技术，缓冲技术提高了整个系统的效率。

存储管理方案有分区管理、分页管理、分段管理和段页式管理四大类，分区管理主要使用在早期的计算机系统上。分页管理、分段管理和段页式管理中引入虚拟存储技术，使内存的利用率有了极大的提高，也便于实现内存的共享、保护。

Windows Server 2008 的存储管理方案属于请求页式管理方案，把地址空间划分成两部分：进程地址空间和系统地址空间，采用二级页表地址变换方法，用工作集表示进程所获得的内存，以页面错误实现页面的调度，在区域对象中提供视口可以让多个进程共享内存，使用页块数据库表示内存中页块的状态和变化，来管理物理内存。

一、选择题

1．把作业地址空间中使用的逻辑地址变成内存中物理地址称为（　　）。

A．加载　　B．重定位　　C．物理化　　D．编译

2．在存储管理中，采用覆盖与交换技术的目的是（　　）。

A．减少程序占用的主存空间　　B．物理上扩充主存容量

C．提高 CPU 效率　　D．代码在主存中共享

3．通常所说的“存储保护”的基本含义是（　　）。

A．防止存储器硬件受损　　B．防止程序在内存丢失

C．防止程序间相互越界访问　　D．防止程序被人偷看

4．在下列有关请求分页管理的叙述中，正确的是（　　）。

A．程序和数据是在开始执行前一次性装入的

B．产生缺页中断一定要淘汰一个页面

C．一个被淘汰的页面一定要写回外存

D．在页表中要有“访问位”和“改变位”等信息

5．在下面关于虚拟存储器的叙述中，正确的是（　　）。

A．要求程序运行前必须全部装入内存且在运行过程中一直驻留在内存

B．要求程序运行前不必全部装入内存且在运行过程中不必一直驻留在内存

C．要求程序运行前不必全部装入内存但是在运行过程中必须一直驻留在内存

D．要求程序运行前必须全部装入内存但在运行过程中不必一直驻留在内存

6．在请求分页系统中，页表中的访问位是供（　　）参考的。

A．页面置换　　B．内存分配

C．页面换出　　D．页面调入

7．选择在最近的过去最久未访问的页面予以淘汰的算法称为（　　）。

A．Opt　　B．LRU　　C．MFU　　D．LFU

8．虚存最基本的特征是（　　）。

A．一次性　　B．多次性　　C．交换性　　D．离散性

9．在虚拟存储系统中，若进程在内存中占 3 块（开始时为空），采用先进先出页面淘汰算法，当执行访问页号序列为 1，2，3，4，1，2，5，1，2，3，4，5，6 时，将产生（　　）次缺页中断。

A．7　　B．8　　C．9　　D．10

10．采用段式存储管理的系统中，若地址用 24 位表示，其中 8 位表示段号，则允许每段的最大长度是（　　）。

A．2^{24}　　B．2^{16}　　C．2^{8}　　D．2^{32}

11．在一页式存储管理系统中，页表内容如下所示。若页的大小为 4K，则地址转换机构将逻辑地址 0 转换成物理地址为（　　）。

A．8192　　B．4096　　C．2048　　D．1024

页号	块号
0	2
1	4
2	3
3	8
4	7

12．下面哪种内存管理方法不能实现虚存？（　　）

A．动态页式管理　　B．静态页式管理

C．分段存储管理　　D．段页式存储管理

13．解决碎片问题，以及使程序可浮动的最好的办法是采用（　　）技术。

A．静态重定位　　B．动态重定位

C．内存静态分配　　D．内存动态分配

14．设主存容量为 1MB，辅存容量为 400MB，计算机系统的地址寄存器有 24 位，那么虚存的最大容量是（　　）。

A．1MB　　B．401MB

C．1MB+2^{24}B　　D．2^{24}B

15．在硬件条件和环境条件相同的条件下，一个作业在（　　）存储管理系统中执行的时间最多。

A．分区　　B．分页

C．分段　　D．段页

二、填空题

1．使每道程序能在内存中“各得其所”是通过存储管理的_____功能实现的，保证每道程序在不受干扰的环境下运行，是通过存储管理的_____功能实现的，为缓和内存紧张的情况而将内存中暂时不能运行的进程调至外存，这是通过存储管理的_____功能实现的，能让较大的用户程序在较小的内存空间中运行，是通过存储管理的_____功能实现的。

2．在内存分配技术中，_____是一种允许作业在运行中、在内存中进行移动的技术。

3．分段保护中的越界检查是通过_____中存放的_____和段表中的_____实现。

4．在分页系统中若页面较小，虽有利于_____，但会引起_____；而页面较大，虽有利于_____，但会引起_____。

5．在分页系统中的地址结构可分为_____和_____两部分；在分段系统中的地址结构可分为_____和_____两部分。

6．引入分段系统，主要是为了满足用户的一系列要求，主要包括_____、_____、_____和_____几个方面。

7．页是信息的_____单位，进行分页是出于_____的需要；段是信息的_____单位，进行分段是出于_____的需要。

8．非虚拟存储管理方式最基本的特征是_____。在动态分区存储管理方式中的另一个重要特征是_____。在分段存储管理方式中的另一个特征是_____。

9．在段页式系统中（无快表），为获得一条指令或数据，都需三次访问内存。第一次从内存中取得_____，第二次从内存中取得_____，第三次从内存中取得_____。

10．虚拟存储器的基本特征是_____和_____，因而决定了实现虚拟存储器的关键功能是_____和_____功能。

11．为实现请求分页管理，应在页表中增加_____、_____、_____、_____等几项。

12．为实现段的共享，系统中应设置一张共享段表，其中包含_____、_____和_____等数据项。

三、简答题

1．操作系统的存储管理目标是什么？

2．为什么把进程的地址空间分为逻辑地址空间和物理地址空间？

3．什么是重定位？如何区分静态重定位和动态重定位？

4．覆盖技术与虚拟存储技术有何不同？

5．交换技术和虚拟存储技术有什么异同？

6．描述缓冲存储器的工作原理。

7．什么是虚拟存储器？其容量通常由什么因素决定？

8．试画出分页情况下相关映像的地址变换过程。

9．说明分页、分段和段页式虚拟存储技术中的存储保护是如何实现的？

10．说明请求页式管理中的取页策略、置页策略和淘汰策略，并指出 Windows Server 2008 系统中是如何实现的？

11．什么是动态连接？描述动态连接过程。

12．说明分区、分页、分段和段页式存储管理的优缺点。

13．Windows Server 2008 系统为什么采用二级页表结构？

14．说明 Windows Server 2008 系统中的地址变换过程。

15．在 Windows Server 2008 系统中是如何实现访问内存数据的速度与处理机的处理速度相匹配的？

16．在 Windows Server 2008 系统中共享内存技术的特点是什么？

17．在 Windows Server 2008 系统中工作集是如何进行调整的？

18．说明在 Windows Server 2008 系统中页块状态的变换过程。