第3章 存储管理

多道程序系统要求主存储器中存放多路进程，以提高系统的利用率。主存储器是仅次于CPU的宝贵资源。众多进程共用一个存储器，必然涉及存储器的分配、安全、利用率、共享以及扩展等诸多问题。在分配存储器空间时，应解决以下问题：采用何种区域划分形式才能使存储器达到最有效的利用？采用何种安全保障手段才能使不同的进程既能够相互配合又不至于相互破坏？如何实现进程对存储空间的共享？在存储空间不足的情况下如何借用辅助存储器空间来给用户提供更大的使用空间？

更具体地说，存储管理需要做的事情是：将用户程序所用的地址空间转换为主存储器中的实际地址空间，将用户程序的操作地址变换为存储器上的具体位置，为存储空间提供安全和共享的手段，为用户程序实现虚拟存储空间等。

3.1 实用系统中的存储管理方法

3.1.1 DOS分区及分段

DOS对主存储器的大小有限制，它只能使用1MB的内存空间。该空间被分为两部分：处于低端的640KB的基本内存和处于高端的扩展内存。系统启动后将操作系统调入基本内存的低端位置，大概占几十KB的空间，基本内存的剩余部分便是用来存放用户程序的用户区。

在DOS发展的后期，操作系统除了驻留在低端内存以外，还可以利用扩展内存来存放系统的数据结构、驱动程序以及某些库文件等内容，但用户不能对扩展内存中的内容进行修改。图3.1展示了DOS对存储器的划分情况。

用户区是用户程序所使用的区域，该区中可以存放用户程序和数据，可以作为用户的工作区来对数据进行处理，但用户不能突破基本内存的限制，因此用户程序的大小必须低于640KB。用户区内只能存放一个用户程序，因此，DOS只支持单道程序。

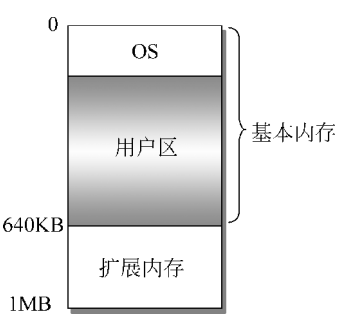


图3.1 DOS的内存划分

3.1.2 Windows XP的存储器

Windows XP对存储器的大小限制最低为64MB，原因是操作系统本身需要较大的空间。内存被划分为大小为4KB的块，又称为页面。用户在编制程序时，其大小最高可达4GB，但在程序运行时，并不是全部程序都装入内存，而是只装入程序的部分页面来运

行。内存中可以存放多个用户任务的页面,因此,Windows 支持多任务同时运行。

当需要装入新的程序页面而内存中又没有足够的空闲区域时,操作系统将内存中长期未使用的页面换出到辅助存储器上早已安排的页面(paging file)文件中,腾出空间后再将需要换进的页面调入。这样无论用户任务需要多少内存空间,操作系统都能实现对它们的运行和控制,这便是 Windows XP 中的虚拟存储器。

通过 Windows XP 的系统监视器,我们可以查看内存使用变化情况(双击"控制面板"→"管理工具"→"性能",然后分别添加计数器:Memory//Page Faults/sec、Memory//Pages Input/sec、Memory//Pages Output/sec、Memory//Available MBytes(见图 3.2 和图 3.3)。

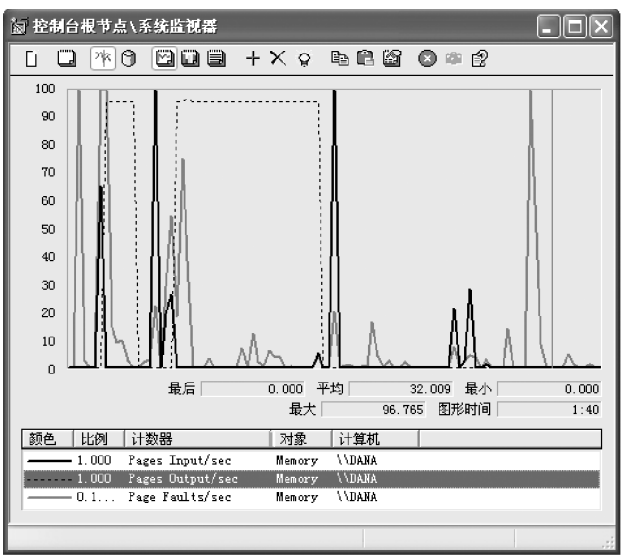


图 3.2　页面在内存中换出换进

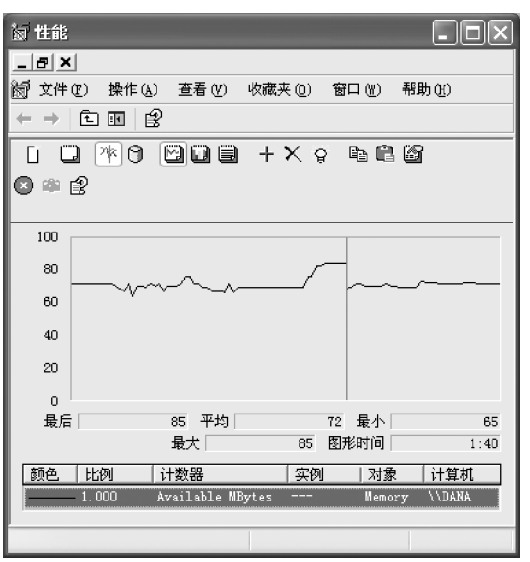


图 3.3　可用物理内存

图 3.2 表示页面在内存中换出换进情况。Page Faults/sec 是每秒发生页面缺失的平均数量。页面缺失将直接导致页面换进。Pages Input/sec 是从磁盘换进页面的速度。当一个进程引用一个虚拟内存的页面，而此页面不存在于内存，就会发生页面缺失。Pages Output/sec 是指为了释放物理内存空间而将页面写入磁盘的速度。当物理内存不足时，Windows 会将页面写回到磁盘以便释放空间。我们可以看到，随着时间的变化，各个参数都在发生变化，其中出页的峰值往往与进页峰值接近，这说明出页多是因为有进页需求，即只有当内存中没有可分配空间，同时又必须调入内存新的页面时，才需要换出页面。只要内存中还存在未使用的物理内存，即使有页面换进要求，也不需要将存在于内存的页面换出。

图 3.3 反映了可用物理内存与虚拟内存变化情况。Available MBytes 是计算机上运行进程的可用物理内存大小，它是将零的、空闲的和备用内存列表的空间添加在一起来计算的。

3.1.3 Linux 存储管理

Linux 也是将存储器空间划分成页面，根据进程运行时的需要来对页面进行换进、换出的。同样在磁盘上也安排了交换区来与内存协调工作，以达到扩大内存的目的。但是 Linux 的交换区多采用在硬盘上划分出一个指定区域来作为交换区，因此交换区的大小不可变化。

3.2 存储管理功能

要了解存储管理的功能，有必要先了解以下概念，然后再进行功能说明。

3.2.1 用户实体与存储空间

1. 用户实体与存储器的关系

从用户的角度来看，由于操作系统要处理各种不同表现形态的实体，而这些实体的存在就意味着对存储空间的占有，又由于这些实体都有其不同的存在状态，因此其占有存储空间的种类、位置、大小都各不相同。一般情况下，作业存放在辅助存储器上，其内容包含作业控制块、作业步序列、作业所对应的程序和数据。当作业的状态变为执行状态时，其内容进入内存并以进程的形式存在。进程的内容包含进程控制块、进程所对应的程序和数据。任务在被激活之前存放在辅助存储器上，包含任务描述信息、子任务链接数据、任务所对应的程序及要处理的数据。当任务被激活时，它成为进程进入主存储器。进程的描述部分及主程序部分始终存放于主存储器，其他程序和数据部分视需要由操作系统在内存与外存之间交换。程序一般被存放于辅助存储器上，它是否进入主存储器完全依赖于进程的运行。如果程序是被进程处理的数据，则一般存放于主存储器，其他数据则以文件的形式存放于辅助存储器。

对于存储管理程序来说，它主要考虑用户和存储器两个部分的衔接。当用户向计算机提交自己的任务时，存储管理是以一种逻辑形式来进行描述，而当操作系统处理用户的

任务时,是对具体的存储器地址进行操作,这两者是不相同的,存储管理的工作就是圆满地处理发生在衔接逻辑和物理存储时所产生的各种问题。

2. 存储空间与存储地址

下面介绍逻辑和物理存储空间与地址的概念。

(1) 逻辑地址

用户在编制程序时无法预知程序将在内存中所占的位置,因此他无法直接使用内存地址,于是用户以地址0为起点来安排程序指令和数据。每一条程序指令要访问的数据都有一个对应的地址,这个地址被称为逻辑地址。由于它是相对于0的地址,因此又被称为相对地址。当用户程序被编译为目标代码时也使用的是相对地址。原则上讲,相对地址的最大值没有限制,因此用户可以无限制地加长自己的程序。但在具体应用中相对地址的大小受相对地址寄存器位数的限制,如在Windows XP中相对地址寄存器为32位,表示相对地址最大可达4GB。

(2) 逻辑地址空间

一个完整的用户作业,一定存在着一系列的逻辑地址。这些逻辑地址形成一个范围,用户程序、数据、工作区域都包含在该范围之内,这就是逻辑地址空间。逻辑地址空间虽然没有具体的存在形式,但不同的操作系统赋予它不同的表现形式(我们将在后面几节中针对具体的存储管理进行介绍),它的大小也是可以确定的。用户可以直接对逻辑地址和逻辑地址空间进行访问和操作。逻辑地址空间可以定义为:实体(用户、作业、任务、进程或程序)所用的所有逻辑地址的集合。逻辑地址空间又称为相对地址空间,有时候也被简称为用户空间或者作业空间。逻辑地址空间的大小被限制在0到相对地址最大值之间。

(3) 物理地址

内存中的实际地址被称为物理地址。由于它并不和任何相对地址相关,因此,物理地址又称为绝对地址。物理地址的最小值为0,最大值取决于内存的大小和内存地址寄存器的所能表现的最大值,两者中较小的那一个值为物理地址的最大值。

(4) 物理地址空间

当作业进入主存储器时,其逻辑地址空间就会被操作系统安排到具体物理位置上,作业因此而占有的内存空间就是物理地址空间。物理地址空间可以定义为:当逻辑地址空间被映射到内存时所对应的物理地址的集合。物理地址空间又称为绝对地址空间。物理地址空间并不是指物理内存,只有当逻辑地址空间存在时,才会有物理地址空间,物理地址空间受存储器大小的限制,也就是说物理地址空间最大只能达到内存的大小。物理空间所使用的地址是物理地址,但并不是说只有物理空间内的地址才是物理地址,内存中所有的地址都是物理地址。

3.2.2 存储分配、释放及分配原则

1. 存储分配

当作业向系统提出存储空间的要求时,存储分配程序在内存中选择一个大小相当的区域分配给作业。存储分配实际上是将作业的逻辑地址空间映射成为内存中的物理地址

空间。内存中有许多尚未使用的区域即自由区都可以被分配，但到底选择哪一个自由区必须依据分配算法来确定。

2. 存储释放

当作业运行完毕不再需要所分配的内存时，它便会释放内存区域。存储释放实际上是解除逻辑地址空间与物理地址空间的联系，并释放物理空间。存储释放程序将回收的内存区域重新设定为自由区，并将其安排进入自由区队列。进入自由区队列的具体位置也必须依据分配算法。

3. 分配原则

在设计分配程序时需要考虑如下诸多的因素。

(1) 内存空间的划分

内存空间的划分有利于作业的存放及存储空间的管理。好的空间划分既可以存放大小不同的作业，也可以使存储空间得到充分的利用，还可以使管理简单易行。通常的划分方法有分区、分块等。

(2) 数据结构的确定

数据结构用来对存储空间进行描述。所有存储空间统一描述或者将已分配区域与未分配区域分开描述，可使用存储分区表以及存储区队列等方法。

(3) 作业空间的划分

作业空间的划分决定了用户在编制程序时必须遵守的空间规则。这种划分有利于在存储分配时合理安排对应的内存区域。通常采用作业空间连续、分页或者分段的形式。

(4) 淘汰算法

当需要分配内存空间而又没有足够的内存可分配时，就需要考虑将已存在内存的某些分区淘汰，以腾出空间来进行分配。选择哪些分区和页面来进行淘汰将直接影响到内存与外存之间的交换频率，因此，淘汰算法将影响系统的工作效率。

(5) 分配算法

当作业申请内存空间而内存中又有若干可以分配的自由区域时，选择哪一个自由区域依据分配算法，可以按先后顺序、大小顺序等来进行选择。

3.2.3 地址映射

用户在逻辑地址空间中安排自己的作业，作业中的程序、数据等各部分的地址取决于它们之间的逻辑关系，而作业的运行依赖于操作系统为其安排的物理地址空间，因此用户作业必须经过装入和地址映射以后才能够运行。

1. 装入

装入是指将逻辑地址空间安排到内存中具体的物理位置上。装入针对的是整个逻辑地址空间，对应的物理地址空间可以是连续存放的，也可以是分开存放的。装入后的作业并不能立即运行，因为作业中每一个指令要访问的地址依然是相对地址，相对地址是逻辑地址空间中的地址，并不是内存中的绝对地址，因此不能够访问。

2. 地址映射

对于指令要访问的地址进行相对地址到绝对地址的变换，就是地址映射。地址映射

就是将逻辑地址空间中的地址映射到物理地址空间中去。采用的办法为重定位。

3. 重定位

在装入过程完成后,根据装入的起始位置来修改程序中指令要访问的地址,将相对地址改为绝对地址就是重定位。方法如下:

绝对地址=(BR)+相对地址

其中,BR是基地址寄存器,用来存放内存中段的起始地址;(BR)表示该寄存器中的内容,为绝对地址空间的起始地址。

上述地址的修改可以在不同的时候进行。根据不同的地址修改时间可将重定位划分为静态重定位和动态重定位。

1) 静态重定位

静态重定位是在装入过程完成后在程序运行前,一次将所有的指令要访问的地址全部修改为绝对地址,在程序运行过程中不再修改(见图3.4)。静态重定位要求程序一旦装入,其绝对地址空间就不能发生变化了。

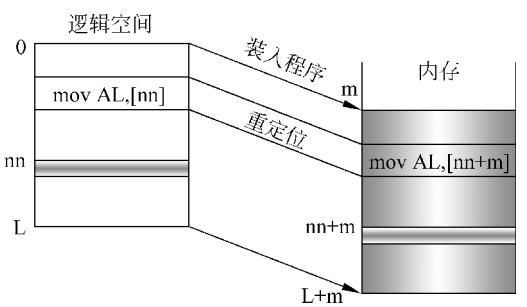


图3.4 静态重定位

采用静态重定位方式的主要优点如下:

(1) 不需要增加任何硬件设备,可以在一般机器上全部用软件实现。

(2) 静态重定位装入程序可以对由多个程序段组成的程序进行静态连接,而且实现起来比较简单。

静态重定位方式的主要缺点如下:

(1) 因为程序是在执行之前一次装入到主存储器中的,在执行期间程序不能在主存储器中移动,所以对提高主存储器的利用率不利。

(2) 若程序所需要的存储容量超过了分配给它的主存物理空间,则程序员必须在程序设计时自行采取某种手段来解决存储空间不足的问题,如采用覆盖结构。

(3) 多个用户不能共享已经存放在主存储器中的同一个程序,如果几个用户要使用同一个程序,则每个用户必须在各自的主存空间中存放一个程序副本。

2) 动态重定位

动态重定位是在程序的运行过程中,当指令需要执行时对将要访问的地址进行修改,修改过程如图3.5所示。动态重定位允许在程序运行过程中,其绝对地址空间发生变化或被分割为不同的区域,变化后只需要将基地址寄存器中的内容进行对应修改。

采用动态重定位方式的主要优点如下:

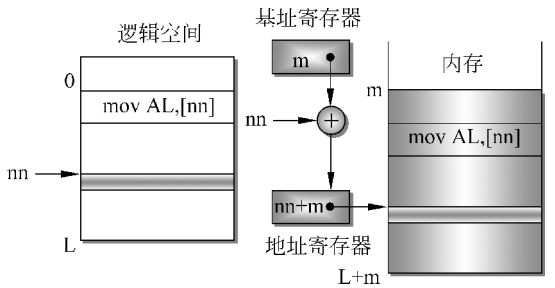


图 3.5　动态重定位

(1) 由于程序是在实际执行时,由硬件形成主存物理地址的,因此在程序开始执行之前,不一定要把整个程序都调入到主存中。而且一个程序可以被分配在多个不连续的主存物理空间内,从而可以使用较小的存储分配单位,以提高主存储器的利用率。

(2) 几个程序可以共享存放在主存储器中的同一个程序段,而不必在主存储器中存放多个副本,这是静态重定位方式无法实现的。

(3) 支持虚拟存储器。它可以为用户提供一个比实际主存储器的物理空间大得多的逻辑地址空间。对于大程序,不需要采用覆盖结构,程序的调度完全由系统管理程序来实现。

动态重定位方式的主要缺点如下:

(1) 需要有硬件支持。

(2) 实现存储管理的软件算法比较复杂。

3.2.4　虚拟存储器

虚拟存储器也是一种虚拟资源,它是将内存进行虚拟,使用户能使用比实际内存大得多的虚拟空间。要实现虚拟内存必须具备如下条件:

(1) 实际内存空间。由于用户程序需要在主存储器中运行,内存空间是构成虚拟空间的基础,因而内存空间越大,所构成的虚拟空间的运行速度越快,一次能够并行的用户数目也越多,虚拟空间的性能也越好。

(2) 辅助存储器上的内存交换区。为了将主存储器空间扩大,可借用辅助存储器上的一部分区域来进行,该区域被称为内存交换区。交换区的大小可以设定,一般最小不得低于主存储器的大小,最大可达整个辅助存储器的大小。当然,它必须受虚拟地址的限制。

(3) 虚拟地址。针对虚拟存储器的使用,用户在编制程序时应使用逻辑地址。因此,逻辑地址也称为虚拟地址,逻辑地址空间也称为虚拟地址空间。对虚拟地址的重定位就是将程序指令中的虚拟地址修改为物理地址。虚拟地址的大小可以超过主存储器的大小,它只是受地址寄存器的位数的限制,如一个 32 位的地址寄存器其虚拟地址最大可达 4GB(即 2^{32})。

(4) 换进、换出机制。它表现为中断请求机构和淘汰算法。中断请求机构用于对外存作业空间的换进,淘汰算法用于对内存页面的换出。

具备了以上条件,存储管理就可以通过程序来实现虚拟存储器了。因为用户所使用的虚拟空间远大于主存储器的大小,操作系统将用户程序部分装入内存而其余部分装入辅助存储器的交换区中,在程序的运行过程中按需要在主存储器和交换区之间不断地进行换进、换出,使用户程序最终得以全部运行。

目前所使用的操作系统几乎全部具备虚拟存储器功能,虽然不同的系统其实现虚拟存储器的基本条件都相似,但在数据的换进、换出策略上是可以不同的。

3.2.5 存储保护与共享

由于存储器中存放的进程不止一个,并且若干进程又是并行的,这就很难保证进程所在主存区域的数据不被其他进程在非授权情况下访问,甚至破坏。存储保护就是要保护进程的数据不被非法访问者破坏。存储保护的手段有以下几种。

1. 界地址寄存器保护法

每一个进程的绝对地址空间都存在上界和下界,界地址寄存器保护法要求进程在运行时不得超过上、下界地址寄存器所指定的地址范围。采用基地址寄存器和长度寄存器来描述绝对地址空间的起始位置和长度,进程每运行一条指令都要将其所访问的地址与基地址寄存器和长度寄存器的值进行比较,如在范围之内可以正常运行,如超过范围则视为非法。从图 3.6 可见界地址寄存器保护法的具体运作。

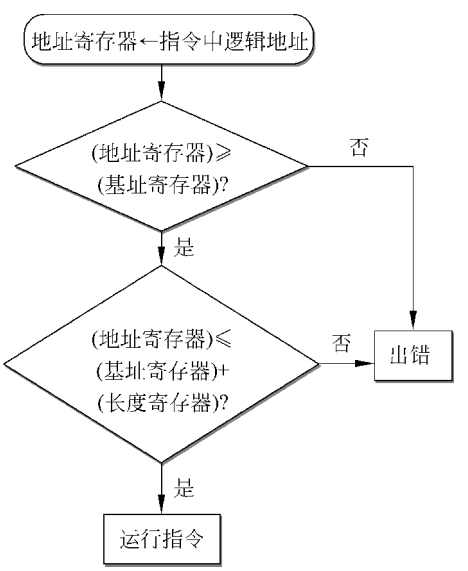


图 3.6 界地址寄存器保护法

纯粹采用界地址寄存器保护法也存在问题,它使进程之间合法的访问也受到限制,特别是当进程之间需要共享某些数据时,使用界地址寄存器就表现得无能为力。

2. 访问授权保护

系统为每一个存储区域都给定一个访问权限值,同时也为每一个进程赋予一个访问权限值。当进程访问某个区域时,若进程的访问权限大于等于被访问区域的权限值,访问可以进行,否则视为非法。由于访问权限值可以在一个范围内变化,因此进程访问权利的灵活性得以体现,一个进程可以对不同存储区域有不同的访问权限,一个存储区域也可以被多个具有不同访问权限的进程按权限级别进行访问。访问授权保护还有一个好处是它允许存储区域的共享。具体运作如图 3.7 所示。

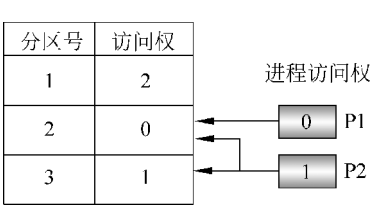


图 3.7 访问授权保护

图 3.7 表明,进程 P1 对 2 号区域具有访问权,P2 对 2 号和 3 号区域具有访问权,但由于 1 号区域的访问权为 2,大于 P1 和 P2 所持有的访问权,因此无法被 P1 和 P2 所访问。

3.2.6 存储区整理

当系统运行一段时间后，内存空间经过不断分配与释放，可能已经变得不再整洁，可能出现如下问题：产生许多被称为碎片的不能够使用的小的自由区；当新的进程进入内存时被过分分散存储；存储管理在虚拟存储器的内存和外存之间换进、换出的次数过多，导致系统运行缓慢；系统不断地告诉用户内存空间不够等，都表明存储区需要整理。存储区的整理就像一个家庭主妇对自家住房中的家具重新挪动，使其看起来空间更大，使用起来更方便。

存储器的整理方法可以是定期将内存中的碎片合并，或者将某些进程的分散存储区域移动到一起。经过整理后，系统中将有更大的自由分区，进程存储的分割也更为合理，存储管理的效率就必然会提高。存储器整理的副作用是在整理时所有进程都不能执行，并且需要消耗较多的 CPU 时间。

由于不同的系统都有不同的主存储器管理方法，在其功能的实现上采用的方法也各不相同，下面针对不同的存储区域的划分方法来进行具体的分析。

3.3 分区管理

分区管理是存储管理方法中一种最简单的形式，存储管理的发展也都是以它为基础的。从静态单一连续分区、固定分区到动态的多重分区，我们将分别从实现原理、分配与释放、地址映射、存储分配、存储保护等几个方面来进行讨论。

3.3.1 单一连续分区

1. 实现原理

除了操作系统所占有的内存空间以外，存储管理提供给用户程序的物理空间是从某一低地址开始到某一高地址结束的单一的连续的区域。用户逻辑地址空间也是一个连续的整体，经过装入程序直接装入分区的低地址部分(见图 3.8)。用户区经过分配，一部分区域被作业占有，剩下的那一部分无法再利用。

2. 分配与释放

由于内存中只有一个可用区域，分配和释放都是针对这一个区域。分配流程如图 3.9所示。当一个作业申请内存空间时，分配程序将作业的大小与内存中用户可用区域比较，如果作业申请的区域小于或等于用户可用区域，则进行正常分配，否则告诉用户内存不够无法分配。释放程序更加简单，只需将该区域标志为未分配。

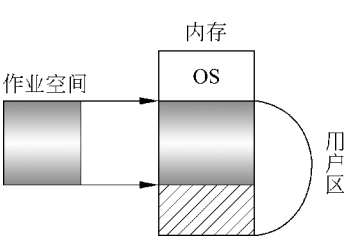


图 3.8 单一分区

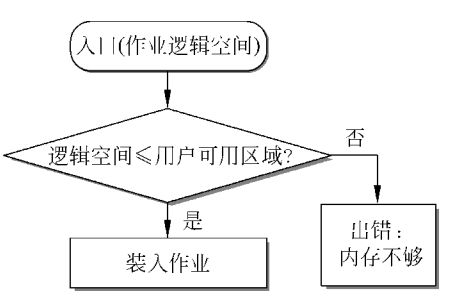


图 3.9 单一分区分配流程

3. 地址映射

采用静态重定位的方式在作业装入时一次性对所有指令将要访问的地址进行修改。由于作业的物理地址空间不会发生变化,因此单一连续分区不适合使用动态重定位。

4. 存储保护

使用界地址寄存器保护法。其中,基址寄存器的内容是操作系统常驻内存部分以后的首地址,长度寄存器的内容便是用户可用区域的长度。由于操作系统不会发生变化,甚至可以不使用界地址寄存器,而将基址和长度用两个常量来代替。

5. 单一连续分区的优缺点

单一连续分区的优点是:

(1) 管理简单。除了分配程序以外,几乎不再需要任何程序来参与管理,因此,系统开销极小。

(2) 使用安全。由于系统中只有一个用户程序,因此,系统遭到破坏的可能性随之降低,加上用户程序和操作系统之间采用固定分界,很容易在操作系统的程序中加入分界地址以防止用户程序的非法访问。

(3) 不需要任何附加的硬件设备。

单一连续分区的缺点是:

(1) 作业的大小受用户可用区域大小的限制。由于作业必须一次性连续存放于内存区域中,比用户可用区域大的作业就无法存放,因此无法运行。

(2) 不支持多用户。

(3) 容易造成系统资源的浪费。系统中一次只能运行一个作业,CPU 的利用率必然受到影响。又由于用户区中只能存放一个作业,即使内存中有较大的剩余空间也不可能得到利用。

单一连续分区适用于单用户系统,DOS 是一个典型的例子。

3.3.2 多重固定分区

1. 实现原理

将内存空间由小到大划分为若干个位置固定大小不等的区域,每个区域可以存放一个作业,存放于不同区域的作业可以并行。用户逻辑地址空间依然是一个连续的整体,在作业申请进入内存时一次性装入。

2. 数据结构

采用内存分区表来描述内存中每一个区域的情况,内容包括区域的起始位置、区域的大小、区域的使用状态。采用作业表来描述存放于区域中的作业,内容包括作业号、作业占用的区域、区域的大小。图 3.10 表现了内存分区表、作业表和内存之间的关系。

3. 分配与释放

由于内存中存在着不止一个自由区,分配算法在作业申请内存空间时需要进行选择,

内存分区表

区号	大小	起始地址	状态
0	20KB	40KB	未分配
1	40KB	60KB	已分配
2	80KB	100KB	已分配
3	160KB	180KB	未分配
4	320KB	340KB	已分配
⋮	⋮	⋮	⋮

作业表

作业号	大小	区号
0	55KB	2
1	10KB	0
2	150KB	4
⋮	⋮	⋮

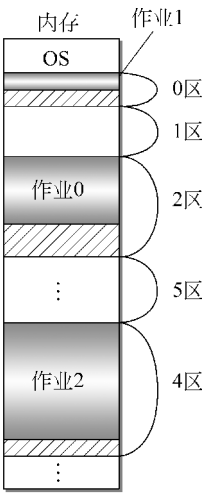


图 3.10　多重固定分区

可以采用最先适应算法、最佳适应算法和最坏适应算法等方法。算法的具体内容将在动态分区部分讨论。

4. 地址映射

由于作业被分配进入内存后位置不再发生变化，因此地址映射可以采用静态重定位方法。不过我们要注意到每一个作业的物理地址空间的起始位置是不相同的，因此对每一个作业进行重定位时要修正基址寄存器的值。

5. 存储保护

存储保护可以采取界地址寄存器的方法和访问授权保护，由于作业在内存中的位置保持不变，可以用两个常量替代界地址寄存。

6. 多重固定分区的优缺点

(1) 提高了 CPU 的利用率。多个作业的并存保证了 CPU 不因为等待某一个作业而停止运行。

(2) 作业大小受到最大分区大小的限制。作业仍然需要一次性连续装入，内存中自由分区的总量即使大于作业的大小也可能无法分配。

(3) 空间浪费。如果一个较小的作业占有一个较大的区域，该区域中剩余的空间就被浪费(见图 3.10 中的阴影区域)。

(4) 碎片问题。每一个分区都存在一部分不能再利用的空间，这就是碎片。碎片的存在必然使存储器的利用率下降。

3.3.3　多重动态分区

为了更多地利用内存空间，可以采用动态分区的方法。

1. 实现原理

多重动态分区是一种灵活的分区方式，它根据作业对内存空间的申请来划分主存区域，区域的大小可变、位置可变、数量也可变(见图 3.11)。

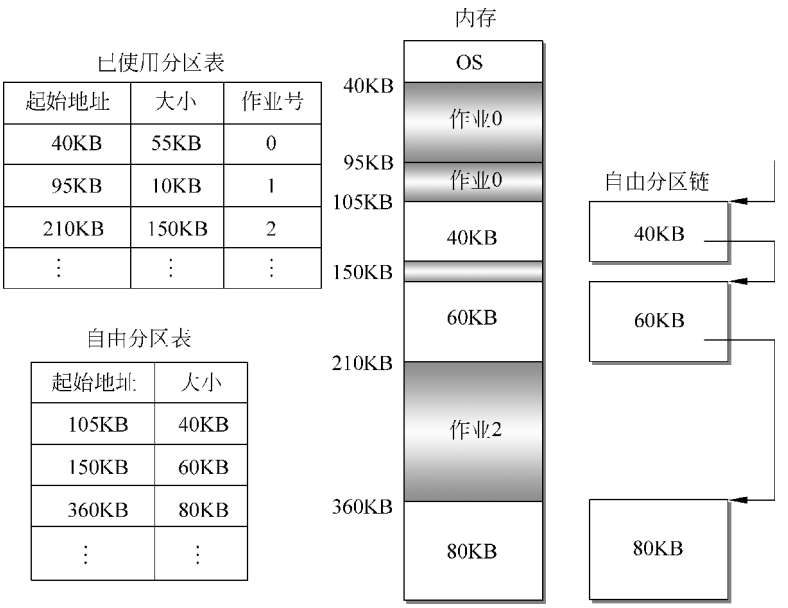


图 3.11　多重动态分区

2. 数据结构

(1) 已使用分区表

描述已被分配的区域,内容包括起始位置、区域大小、对应的作业号。

(2) 自由分区表

描述内存中的自由区域,内容包括起始位置及区域大小。

(3) 自由分区链

为每一个自由分区设置一个链接指针来指向下一个自由分区,使所有的自由分区形成一个链表,内容包括链接指针和分区大小。

3. 分配与释放

当作业申请内存空间,分配哪一个区域需进行选择,选择方式称为分配算法。分配算法有如下几种。

(1) 最先适应算法

该算法可将作业分配到内存中第一个碰到的大于或等于作业申请空间的未分配区(见图 3.12(a))。

(2) 最佳适应算法

该算法可将作业申请大小与内存中所有未分配区的大小进行比较,直到找到最小的大于或等于作业空间的区分配给作业(见图 3.12(b))。

(3) 最坏适应算法

该算法可将作业申请大小与内存中所有未分配区的大小进行比较,直到找到最大的大于或等于作业空间的区分配给作业(见图 3.12(c))。

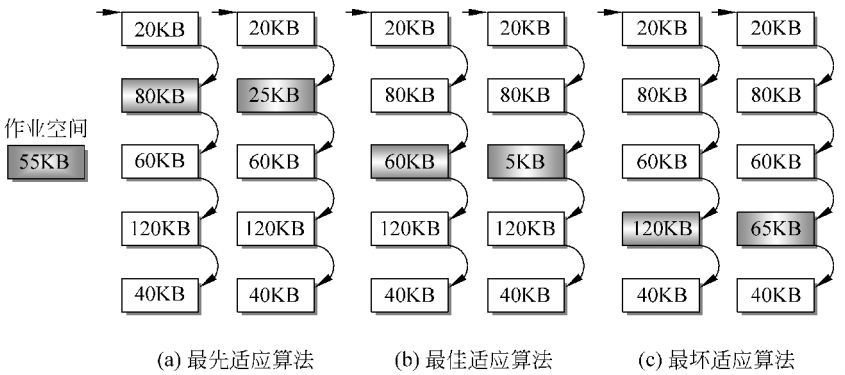


图 3.12　不同分配算法对空间的处理

以上三种算法各有其优缺点，最先适应算法简单但分配比较盲目，可能造成较小的作业分割了较大的空间，使大作业无法被分配。最佳适应算法优先使用小的自由空间，但每次分配以后的剩余空间可能变得过小而成为碎片。最坏适应算法优先使用大的自由空间，在进行分割后剩余空间还可以被使用，但也使大的自由空间无法保留给需要大空间的作业。如果将自由分区表或自由分区块链进行组织，使其按自由区的大小排序，就可以使最佳适应算法也是最先适应算法，或使最坏适应算法也是最先适应算法，算法由此变得更简单。

在进行作业分区的释放时，需要完成自由分区表的插入、修改、合并等。

4. 地址映射

动态分区采用动态重定位方式来实现地址映射，这样作业的基地址发生变化也不会影响执行。当作业被选择运行时，其物理空间起始地址被装入基地址寄存器中，CPU 每执行一条指令之前重定位硬件对指令要访问的地址进行修改。

5. 存储保护

存储保护可以采用界地址寄存器的方法和访问授权保护，不过由于作业被分配于内存一个连续的区域中，访问授权保护的作用似乎并不大，因为作业并没有对其他作业空间的访问权力。

6. 存储区整理

经过不断地分配和释放后，内存中自由分区会变得越来越多和越来越小，这就使很多小自由分区成为碎片。这时，可以用紧缩的方法来解决碎片。紧缩是将内存中已使用区域经过移动沉淀到低地址部分，从而使碎片浮动到内存的高地址部分合并成较大的可使用空间。用紧缩方法来消除碎片需要占用大量的 CPU 时间，并且在移动过程中稍有不慎就有可能破坏全部数据。在 Windows 中对磁盘空间的整理就是采用的紧缩方法（见图 3.13），它与操作系统对内存的紧缩大同小异。

先将低地址部分的已占有区内容转移到高地址部分，清理低地址空间后再将转移出去的内容写入，然后地址后移再进行同样的整理，直到从低地址开始的所有空间都已被占

图 3.13 Windows XP 的磁盘碎片整理

有,全部碎片合并到高地址部分。经过整理,访问的速度明显增快,出差错的次数明显降低。

7. 多重动态分区的优缺点

(1) 多道程序得以提高。

(2) 提高了内存的利用率。由于采用了紧缩碎片的方法,内存中不再存在不能使用的区域。

(3) 作业大小依然受内存容量的限制。

(4) 对碎片问题的解决需要以增加系统开销为代价。

(5) 不便共享。由于分区管理中作业需要连续存放,共享问题始终无法解决。

3.4 分 页 管 理

分区管理中要求一次性将作业空间分配到内存的某一个区域中,即使内存中空闲区域总和超过作业要求的空间,如果没有合适的区域,分配将无法完成,其原因是作业空间始终是一个整体。能否将作业空间分割成若干部分呢? 答案是肯定的,分页管理就是其中一种。分页管理分静态分页管理和动态分页管理两种,下面分别介绍。

3.4.1 静态分页管理

1. 原理

作业地址空间划分成连续的大小相同的页面,内存划分成连续的大小相等的块(也称为页框),页面的大小与内存块的大小完全相同。作业进入内存时其不同的页面对应于内存中不同的块,连续页面可以对应不连续的块。原理如图 3.14 所示。

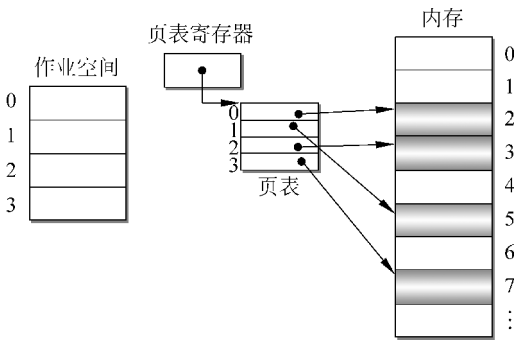
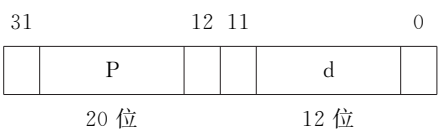


图 3.14　静态分页管理原理图

2. 逻辑地址

逻辑地址被分为两部分：页面号和页内位移。页内位移范围与内存块的大小有关，页号的范围还取决于逻辑地址的位数。下面举例说明。

逻辑地址寄存器如下：



假定逻辑地址为 32 位，内存块的大小为 4KB。4KB 的页内位移变化范围为 0～4095 字节，这需要 12 位来进行描述（$2^{12}=4\times1024$），剩下的 20 位就是页面号的范围 0～1 048 575（$2^{20}-1$）。由此可见内存块的划分有一定的规则，它只能是 2^N。

分页管理的逻辑地址实际上是线性变化的。用户编程时不需要确定逻辑地址处于哪一个页面号和页内位移，只有逻辑地址空间进入物理地址空间后，页面号和页内位移才通过计算电路来划分。对于用户而言，逻辑地址就是一个受地址寄存器位数限制的连续变化的地址范围，可以根据需要来确定页面大小和页面号的范围，因此我们称分页管理的逻辑地址空间为线性空间。

3. 数据结构

数据结构包括以下 3 部分。

（1）页表：页表用来说明作业页面号与内存块号的对应关系，内容包括页面号、内存块号。

（2）页表寄存器：页表寄存器用来存放作业所对应的页表的起始地址。

（3）内存分块表和作业表：内存分块表内容为块号、块使用状态（已使用或未使用），作业表内容为作业号、页表起始地址。

4. 地址映射

地址转换是通过页表寄存器所指定的页表来实现的。假定逻辑地址为页号 P 和页内位移 d，转换方法如下：

页表起始地址＝（页表寄存器）

页表中页号为P的表目地址=(页表寄存器)+表目长度×P,由此获得对应的内存块号P′。

$$绝对地址=P'\times 页框长度+d$$

以上转换是靠硬件来执行的,可参见图3.15。

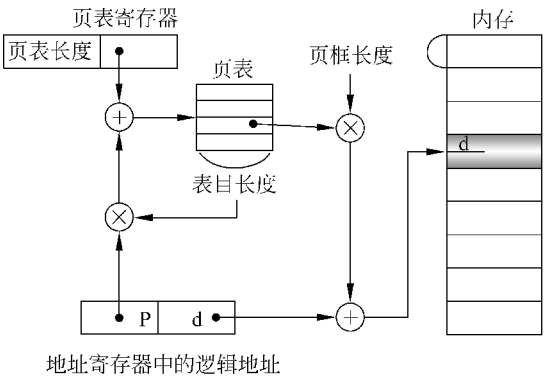


图3.15 静态分页管理地址映射

5. 分配与释放

分配与释放都比较简单,当作业需要内存空间时,可根据作业页面数来分配对应大小的页表,然后根据存储分块表来查找未使用块分配给作业,并在页表上做相应的记录,直到所有页面分配完毕。释放时只需将存储分块表上对应块的状态改为未使用,再释放作业对应的页表及作业表中的对应项。分页管理分配流程图如图3.16所示。

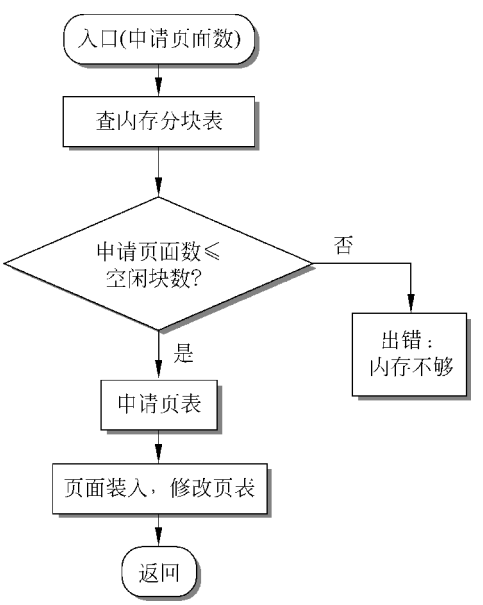


图3.16 静态分页管理分配流程

6. 存储保护与共享

用页表寄存器来记录页表的起始地址和页表长度，它相当于界地址寄存器的功能。

采用分页管理可以实现共享（见图 3.17），不同作业的不同的页面如果同时对应于同一个内存块号，该内存块就被共享了。图 3.16 中显示了作业 A、作业 B、作业 C 共享一个内存块的情况，三个作业的对应页面都通过页表指向同一个内存块。

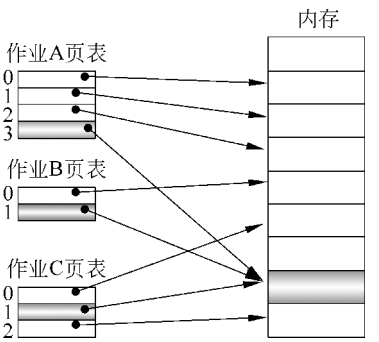


图 3.17　静态分页管理的共享

但页面管理的共享实现起来非常困难，由于页面的划分并没有考虑作业地址空间的逻辑意义，系统无法根据内存块中的数据区分哪些可以共享，哪些不能共享，如果使用页表来硬性共享内存中某一块，则可能导致对该块数据的破坏。

7. 存储区整理

内存被分为大小相等的块，每块对应于作业中的某个页面，表面看来不存在内存空间的浪费。但作业的大小并不一定是内存块大小的整数倍，作业的最后一个页面必然不能完全占据一个内存块，因此每个作业平均有半块内存块的浪费，这种无法再用的内存空间称为页内碎片。内存分块越大，页内碎片也越大，因此可以通过适当减小内存块大小来减少内存的浪费。不过内存块也不能分得过小，过小将使分页管理失去它管理简单的优势。在实际应用中内存块的大小从 512B 到 4KB 不等。

8. 静态分页管理的优缺点

（1）管理简单。由于内存最小管理单位是被划分成大小相等的块，因此分页管理从针对所有内存物理地址，变为只针对内存中的块，操作相对简单。

（2）每访问一次内存数据需要经过二次寻址，即对页表地址的访问和对内存块内地址的访问。

（3）解决了碎片问题。虽然页内碎片依然存在，但内存空间不再存在越来越多的浮动碎片，无须内存碎片整理。

（4）无法实现共享。由于作业地址空间是线性空间，页面的划分并未依据作业的逻辑意义，因此无法实现真正意义上的共享。

（5）作业大小受内存可用页面数的限制。

3.4.2　动态分页管理

静态分页管理是指一次将作业的所有页面都分配到内存，因此作业的大小依然受内存空间的限制。如果想在内存中运行较大的作业，则必须把眼光放到内存以外的存储空间上，动态分页管理开始了对辅助存储器的利用。

1. 原理

内存分块和作业页面的划分与静态分页管理相同，但不是所有的作业页面都一次性

分配进内存,根据作业的使用情况将需要运行的页面存放于内存,暂时不需要运行的页面存放于辅助存储器上,当需要运行存放于辅助存储器上的页面时,再将对应的页面调入内存(见图3.18)。

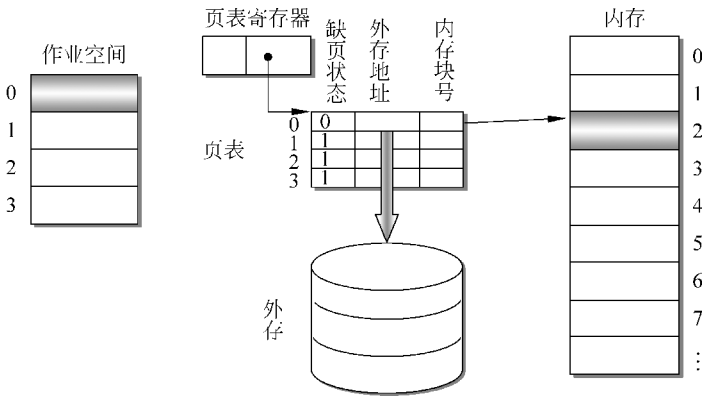


图3.18　动态分页管理原理图

2. 数据结构

动态分页管理的页表内容:页面号、缺页状态、页面外存地址、页面在内存的块号。其中,缺页状态指该页面是否在内存,系统安排该页运行时首先检查缺页状态,如发现该页不在内存,则启动缺页中断将该页面调入内存运行。每个页面都有其对应的外存地址,但每个页面是否有内存块号,则取决于缺页状态。

3. 地址映射

在进行动态分页管理的地址映射时,首先检测该页面的缺页状态,如果该页面在内存,则映射方法与静态分页管理相同;如果该页面不在内存,则先调用缺页中断,然后重新开始本过程。

4. 分配与淘汰算法

(1) 分配

分配是动态进行的,在为作业分配空间时往往调入最先使用的页面,其余页面都置成缺页状态,在需要时再调入。但是,当页面需要调入时可能内存中没有可使用的空闲块,这时就需要将某些已分配的内存块内容调至外存,选定哪一个页面调至外存是淘汰算法要做的事。

图3.19中显示的是执行一条指令的全过程。首先通过页表来查询需要访问的页面是否在内存,如果缺页状态为0,则表示该页面已在内存,接下来通过地址映射找到要访问的内存物理地址实现对指令的运行。如果缺页状态为1,表示该页面不在内存,由此激活缺页中断机构,由缺页中断机构调入需要运行的页面。在调入页面时还需要考虑是否有空闲的内存块,如果没有,则需要运行淘汰算法,将内存中的某一个页面换出后,才能换入新的页面。

(2) 淘汰算法

当内存中没有空闲块可供装入新的页面时,就需要换出已经存在于内存的页面。选

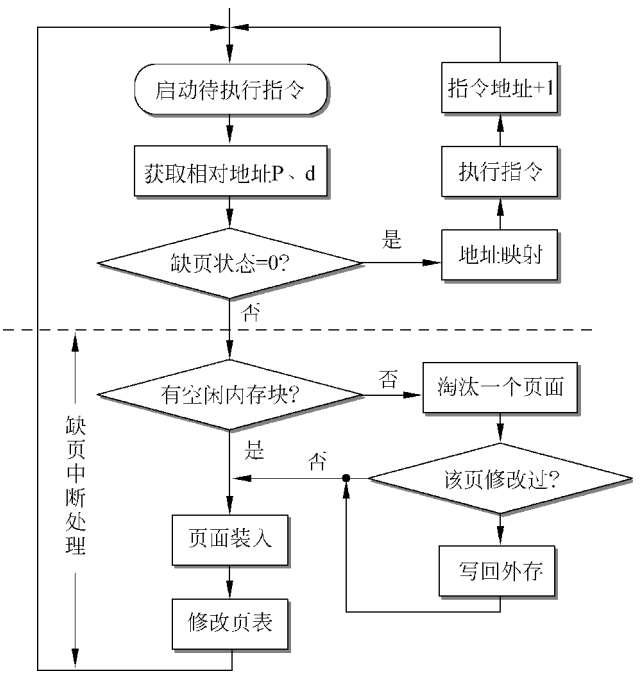


图 3.19　动态分页的动态分配

择哪一个页面淘汰至外存，需要考虑的因素有管理是否复杂、系统是否有稳定性、公平性等。

衡量淘汰算法可以依据缺页率和淘汰率：

$$\text{缺页率}=\frac{\text{缺页次数}}{\text{所有页面访问次数}}$$

$$\text{淘汰率}=\frac{\text{淘汰页面数}}{\text{所有页面访问次数}}$$

好的淘汰算法应该有较低的缺页率和淘汰率。

具体的淘汰算法有以下几个。

① 最佳淘汰算法(OPT)。该算法选择在最远的将来才被访问的页面淘汰(见图 3.20)。假定内存中有 3 个空闲块可以使用，当需要调入页面 3 时，已没有新的空闲块可被使用，需要选择一个页面淘汰。相对于页面 0 和页面 2，页面 4 是最远的将来才会使用的页面，因此它被选择淘汰。

该算法照顾了系统的稳定性，尽量减少了淘汰次数。但由于存在于内存中的页面不经过运行很难判定谁是最远的、将来才被访问的，所以这种算法实际上是不可实现的。

② 先进先出算法(FIFO)。该算法选择最早进入内存的页面淘汰(见图 3.20)。在需要调入页面 3 时，由于页面 0 最早进入，因此被选择淘汰。

这种方法包含一个假定：最早进入内存的页面就是目前最不会被使用的页面。这种假定是否成立依赖于用户编制程序时的考虑，如果是顺序程序，则假定成立，如果是循环程序，则假定不成立。当假定不成立时可能碰到这样的问题，最先进入内存的页面可能是

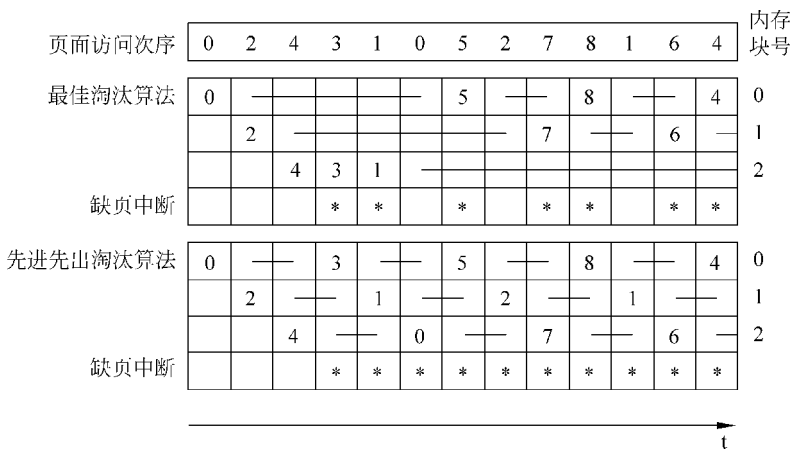


图 3.20　最佳淘汰算法与缺页中断次数

经常使用的页面,在被淘汰出局后可能马上就需要调入内存,这将导致频繁的调出、调进发生,引起系统不稳定,这种现象被称为抖动。系统抖动是我们不愿看到的现象。

③ 最近最少使用算法(LRU)。该算法选择最近一段时间内最长时间未被使用的页面淘汰(见图 3.21)。该算法的假定是:长时间未使用的页面不会马上被使用。这正好符合内存局部性原理(内存中某个位置现在被访问,很快将再次被访问;某个位置现在被访问,其邻近位置也将被访问。在此不作具体的解释,有兴趣者可参见其他书籍),因此从理论上讲,它是一个较好的算法。

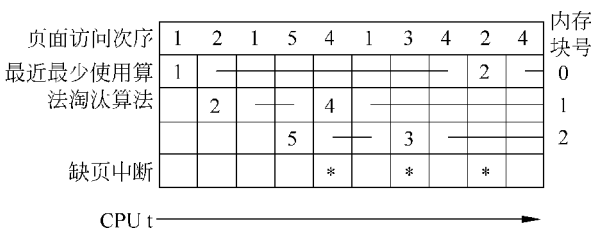


图 3.21　最近最少使用算法与缺页中断次数

最近最少使用算法的问题是,需要确定一个比较时间段来反映哪一个页面长期未被使用,时间段过长时该算法将变为先进先出算法,时间段过短又会使系统频繁地记录访问次数并进行比较,从而增加系统开销。由于该算法难以实现,因而需要寻找比它更为简洁又与之相似的算法。

④ 最近未使用算法(NRU)。该算法选择页面选择指针遇到的最近未被访问的页面淘汰(见图 3.22)。更为简化的方法是:页面选择指针下移,只要遇到刚才未使用的页面就可以淘汰。需要为每个页面安排一个使用位来记录该页面是否被使用过,使用为 1,未使用为 0,在选择时如果发现该页面的使用位为 1,则将其置为 0;如果发现该页面的使用位为 0,则准备淘汰该页,在淘汰之前还要判断其修改位,修改位为 1,表明该页修改过,需将该页面复制到外存储器后再淘汰,修改位为 0,表明未修改过,可直接淘汰。

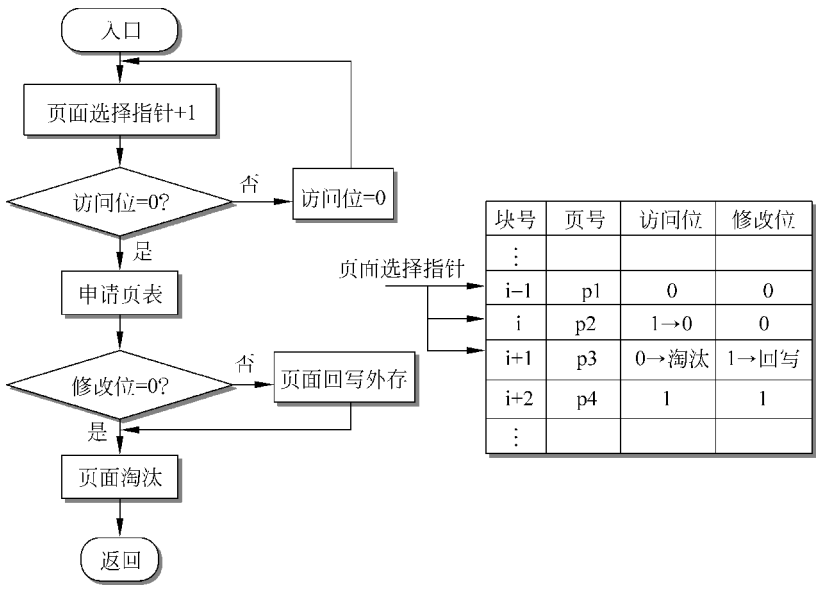


图 3.22　最近未使用算法

5. 虚拟存储器

动态分页技术实现了虚拟存储器。它是通过缺页机构和淘汰算法将作业页面在内存与外存之间换出、换进的，使大的作业能够在较小的内存空间中运行。因此用户可以使用比内存空间大得多的作业逻辑地址空间。

6. 加速寻址

从地址映射我们可以看到，每一次页面的使用都需经过寻找页表表目和寻找内存地址的二次寻址过程，这将使系统整体运行速度降低 1 半。改进方法是采用比内存速度更快的高速存储器来存放常用页面的快速页表(见图 3.23)，该快速页表被称为快表。从图中我们可以看到寻址是经过两条路进行的，一条是原来的二次寻址路径，另一条是通过快表。如果页面是常用页面，就会经过快表被快速定位；如果页面是不常用页面，则通过正常的寻址路径来定位。由于本方法加快了常用页面的寻址过程，系统整体运行速度得以提高。

7. 动态分页管理的优缺点

(1) 由于分页管理以内存中的块为单位，比对单个地址逐一管理要简单得多，因此分页存储管理最大的优点是管理简单。

(2) 采用了动态分页技术以后，增加缺页中断机制和淘汰程序，使分页存储管理能够提供虚拟存储器。

(3) 分页存储管理最大问题是无法实现共享，主要原因是页面的划分未考虑作业空间的逻辑意义。

为了克服分页管理无法实现共享的问题，分段与段页式管理在作业空间的划分上进行了考虑。

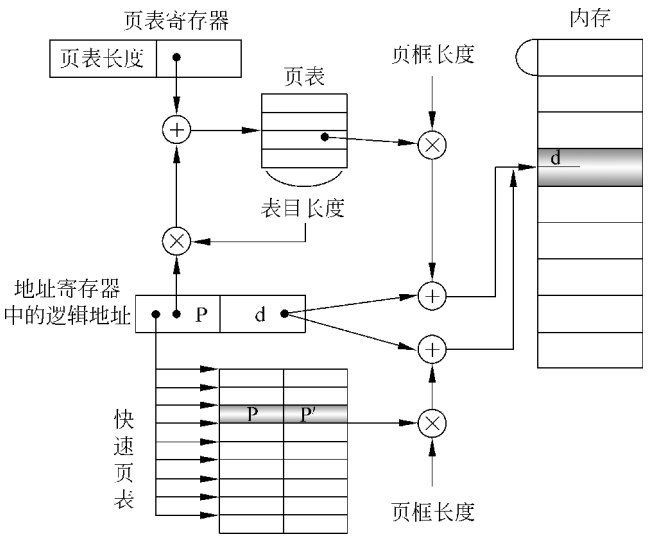


图 3.23　使用快表后的寻址路径

3.5　分段与段页式管理

3.5.1　分段管理

1. 原理

分段管理将作业地址空间按逻辑意义划分成段,每段都有其对应的段号和段长,对分段数量和分段的长度没有限制。内存空间采用多重动态分区的形式,作业中的段对应于内存中的分区,分区的长度和位置没有限制(见图 3.24)。

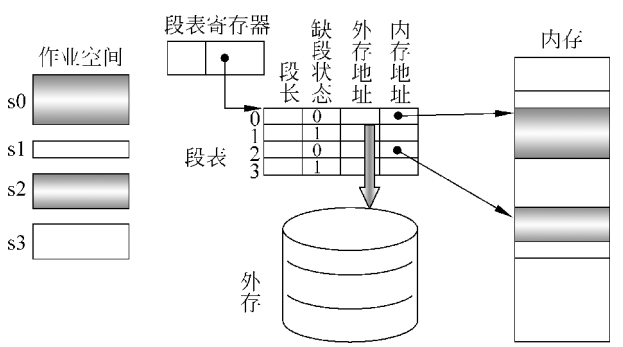


图 3.24　分段管理原理图

2. 逻辑地址

逻辑地址被分为两部分:段号 S 和段内位移 d。

S	d

由于每个段号可对应不同的段长，因此，每个段的段内位移的范围是不同的。在分页管理中，逻辑地址页号 P 和页内位移 d 实际上是一个线性地址，页号和页内位移只是为了对应内存中的块号和块内位移而进行了人为的划分。但是，段号 S 和段内位移 d 不能形成一个线性地址，因为它实际上是代表着段长和段内位移两个变量。由于这两个变量没有特定的限制范围而无法用一个变量来替代，因此分段管理的逻辑地址是二维地址，分段管理的逻辑地址空间是二维空间。

3. 数据结构

分段管理的数据结构有：

(1) 段表。最直接反映逻辑地址与物理地址对应关系的是段表，内容包括段号、段长、缺段状态、段在外存的地址、段在内存的地址。

(2) 段表寄存器。段表寄存器指定作业的段表在内存中的起始位置。

(3) 内存分块表。它记录内存中各分区的情况，内容包括起始地址、分区长度、分配状态、对应作业名，也可以将内存分块表分为两部分：已使用分区表和自由分区表或自由分区块链。

(4) 作业表。它记录作业的存储情况，内容有作业名、作业分段数、段表在内存中的起始地址。

4. 地址映射

地址转换是通过段表寄存器所指定的段表来实现的。假定逻辑地址的段号为 S，段内位移为 d，转换方法如下：

段表起始地址＝(段表寄存器)

段表中段号为 S 的表目地址＝(段表寄存器)＋表目长度×S，由此获得对应的内存分区地址 S′。

绝对地址＝S′＋d

与分页管理一样，分段管理的地址转换机构自动执行以上的转换(见图 3.25)。

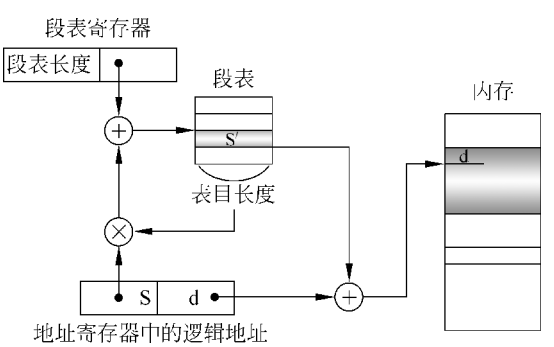


图 3.25　分段管理地址映射

5. 分配与释放

当作业申请内存空间时，分段管理首先为作业申请段表空间，再将代表作业主程序的段按一定的分配算法(内存空间采用动态分区形式，因此分配算法可采用最先适应算法、最佳适应算法或者最坏适应算法)调入内存，填入对应段表表目中该段的内存地址，其他暂时不调入内存的段，缺段状态置为 1，在需要运行时再调入。

由于内存空间有限,可能在某段需要调入内存时无法找到所需的自由分区,这也需要使用淘汰算法。

分段管理的释放与动态多重分区相同。

6. 连接

当作业空间逻辑分段后,用户是靠调用来实现段与段之间的关系,如图3.26(a)中的Call Sub;而存储管理将其实现称为连接(见图3.26(b))。

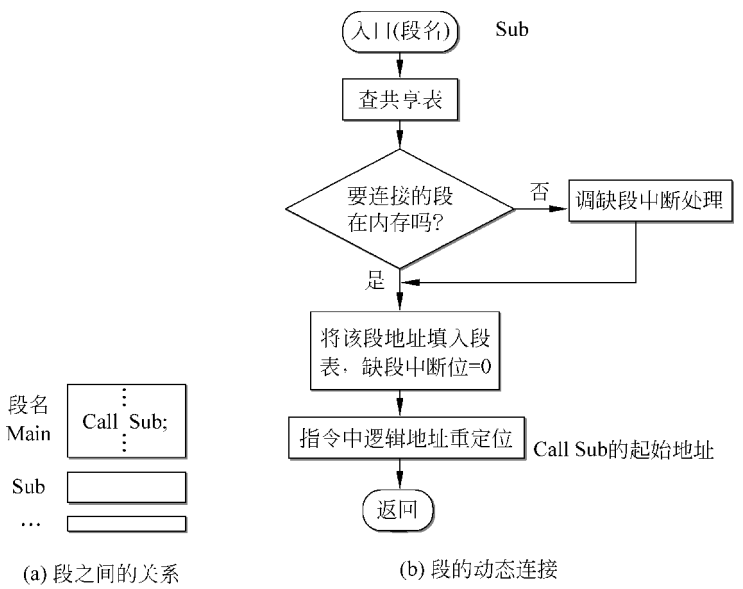


图3.26 分段管理中的连接

一般是通过查看系统共享表来判断要连接的段是否在内存,系统共享表记录了所有可以共享的程序模块,这些程序模块通常来自于子程序库或者公用函数库。如果该段在内存中,就需要修改该段所对应的段表表目,将该段的内存地址填入到段表中。然后对导致连接的指令进行重定位,用该段在内存中的起始地址替换指令中的段的逻辑名称。如果从共享表中获知该段不在内存,则需要调用缺段中断处理将存在于外存的段调入,然后修改段表并进行重定位。由此看来,连接最直接的表现就是将要连接的逻辑模块名修改为该模块在内存中的物理地址,从而实现调用者与被调用者之间的关系。

对Sub段的连接,首先看Sub是否已经在内存,如果在内存,就将其对应的内存地址填入到段表中的内存地址部分;如果不在内存,就将其调入内存,再将地址填入到段表中。然后对Call指令的逻辑地址Sub进行重定位,从而连接完成。

连接也可以分为静态连接与动态连接。

静态连接发生在作业刚装入内存还没有被执行前,因此连接以后各作业段在内存的位置不能发生变化,于是要求所有的作业段必须一次调入内存,这样就限制了作业的大小,对内存的容量要求也高。

动态连接发生在执行过程中,每当作业段与段之间产生调用要求时才进行连接。因此作业中的各段没有必要一次全部调入,这极大地提高了内存的利用率。分段管理采用的是动态连接方式。

7. 共享

从连接我们看到，Sub 段有可能已经存在于内存，但在这之前本作业并没有要求过对 Sub 段的使用，是谁将该段调入内存的呢？一定是别的作业，这说明有另外的作业也在使用 Sub 段，这就是共享（见图 3.27）。由于分段管理允许共享，每一次进行连接时就应该先查看该段是否在内存。从图中我们可以看出作业 A 的 3 号段、作业 B 的 1 号段、作业 C 的 1 号段通过段表指向内存中同一个区，从而实现了共享。

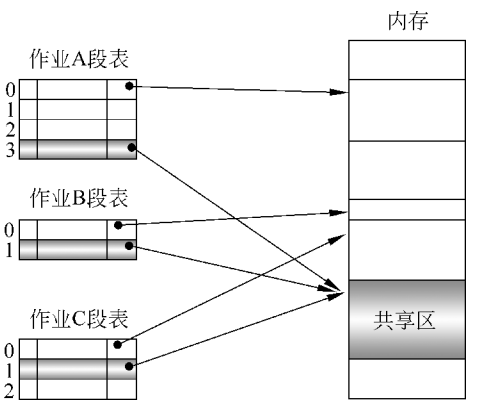


图 3.27　分段管理的共享

8. 存储保护

在段表中既有段的起始位置也有段的长度，它们其实就是基地址寄存器和长度寄存器的内容，分段管理很容易实现作业区域的界限管理。另外，可以在段表中增加一个访问权限项，给段赋予不同的访问权值来实现访问授权控制。分段管理的存储保护显然优于分页存储管理。

9. 虚拟存储器

由于分段管理使用了缺段中断机制，使作业的段可以在内存与外存之间换进、换出，从而实现较小的内存空间运行较大的作业，因此它提供了虚拟存储器。

10. 存储区整理

分段存储管理对内存采用的是动态分区形式，在进行多次运行后系统中也会存在许多小得不能再用的内存碎片，可以使用紧缩的方法来进行存储区的整理。

11. 分段管理的优缺点

（1）易于实现共享。由于将作业地址空间按逻辑意义划分成段，因此，每段很容易被赋予指定的访问权限，又由于段表的地址映射作用，不同的作业可以通过段表指向同一个内存区。

（2）实现了动态连接。通过段的动态连接可以提高内存的利用率，同时动态连接也是实现共享的基础。

（3）实现了虚拟存储器。实现虚拟存储器的关键机构是缺段中断。

（4）段长受内存容量的限制。一个作业段必须存放于内存中的连续区域，因此，该作业段的长度不能超过内存中最大的可用区域，否则就不能够运行。

（5）内存管理复杂。由于内存划分采用的是多重动态分区的形式，内存中的区域位置和大小都在不断地发生变化，造成管理程序复杂并增加了系统开销，因此需要管理程序

经常对内存空间进行整理。

分页管理和分段管理各自存在不可克服的缺点,同时又有着不可替代的优点。显然有一种要求就是利用分页管理和分段管理的优点,克服它们各自的弱点来形成一种新的管理方法,这就是段页式管理。

3.5.2 段页式管理

1. 原理

内存空间划分成大小相等位置固定的块,作业地址空间按逻辑意义划分成大小不等的段,每段再按内存块的大小划分成大小相等的页面,段内的页面对应于内存中的块。图3.28是段页式管理的原理图。

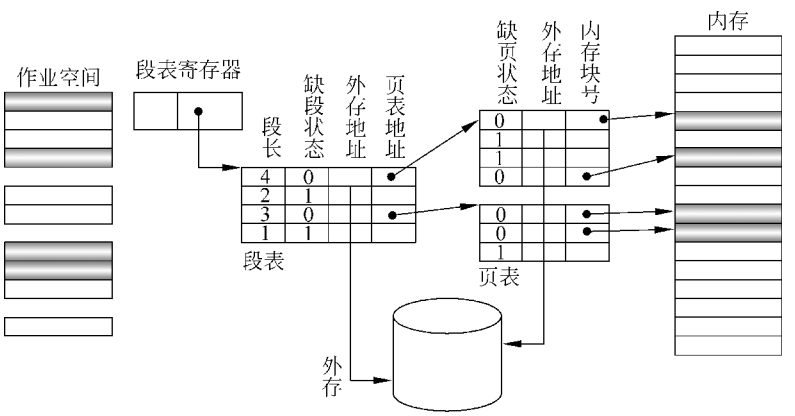


图3.28 段页式管理原理图

2. 逻辑地址

逻辑地址使用段号S、页号P、页内位移d来表示:

S	P	d

由于页号和页内位移共同构成一个线性地址,加上段号所代表的段长变量,段页式管理的逻辑地址空间也是二维空间。

3. 数据结构

段页式管理的数据结构有:

(1) 段表。每一个作业对应一个段表,由它来形成作业段与该段页表的对应关系,内容包括段号、段长、缺段状态、访问控制、内存地址等。其中,缺段状态表示该段页表是否在内存。如果状态为1,表明该段还未进入内存;如果状态为0,则表示该段已在内存。访问控制表示系统赋予该段的访问权限,内存地址是该段页表在内存的地址。

(2) 页表。每一个段对应一个页表,用它来表示段中页面与内存块的对应关系,内容有页面号、缺页状态、外存地址、内存块号。其中,缺页状态表示该页面是否在内存,外存地址是该页面在外存的地址,该页面如果在内存,则有对应的内存块号。

(3) 段表寄存器。它可用来记录作业所对应的段表在内存中的起始位置。

(4) 存储分块表。它记录内存中每一块的使用状态和对应的作业。

(5) 作业表。它记录作业的存储情况,内容包括作业名、作业分段数、段表的起始地址等。

4. 地址映射

地址映射通过段表寄存器、段表和页表来实现地址转换。假定逻辑地址为段号 S、页号 P 和页内位移 d,转换方式如下:

段表起始地址=(段表寄存器)

段表中段号为 S 的表目地址=(段表寄存器)+段表表目长度×S,由此获得对应的页表地址 S′。

页表中页号为 P 的表目地址=S′+页表表目长度×P,由此获得所对应的内存块号 P′。

绝对地址=P′×页框长度+d

段页式管理的地址转换机构自动执行以上的转换(见图 3.29)。

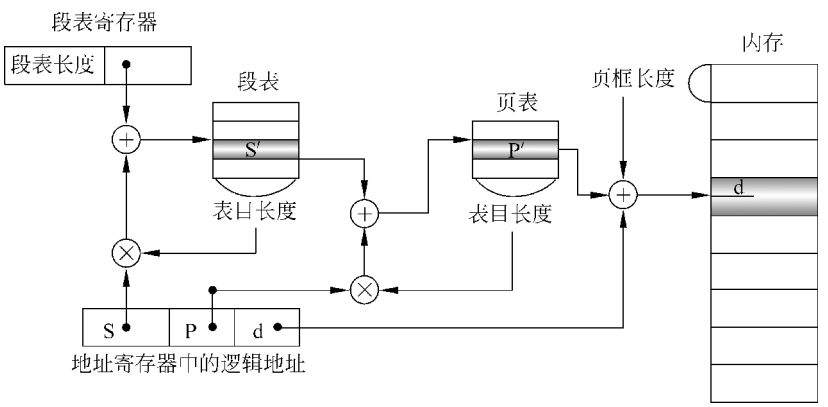


图 3.29 段页式管理地址映射

5. 分配与释放

段页式管理的分配与释放涉及段表分配、页表分配、内存块分配和淘汰算法。

当一个作业申请内存空间时,首先根据作业表中的分段数来申请段表,段表的大小由分段数来确定;再为该作业中的主程序段申请页表,并将页表的起始地址存放于段表对应的内存地址,缺段状态置为 0。其他段暂时不分配,对应的缺段状态置为 1;再来处理主程序段的页表,为最先执行的页面申请内存块,如果获得,则将该页面的内存块号置为刚分配的块号,否则运行淘汰算法来获取一个空的内存块进行分配,将已分配页面的缺页状态置为 0,将未分配页面的缺页状态置为 1。其他许多未分配的段和页面将在程序的运行过程中根据缺页状态来进行动态分配。

6. 连接与共享

(1) 连接

当用户需要调用新的段时,就必须进行连接,首先看需要连接的段是否已经在内存,如果在内存,则表明可以找到其对应的页表,将页表地址填入到段表中的内存地址部分;如果不在内存,则需为该段申请建立页表。其次修改段表的对应部分。最后对用户指令

的逻辑地址进行重定位,从而连接完成。在这里要注意的是,段的连接只是对该段对应的页表的连接,因此,连接以后并不意味着该段的所有内容都已存在于内存,只有当某一个页面需要被访问时,才会通过缺页中断机构将其调入内存。

(2) 共享

通过不同的作业段表中的不同段指向相同的页表地址来实现段的共享,它和分段管理的共享是一样的。

7. 存储保护

与分段管理的存储保护一样,既具有界地址寄存器保护的功能,也使用了存储访问控制。

8. 虚拟存储器

段页式管理的虚拟存储器功能十分强大,它既有缺段中断,也有缺页中断,只要一个作业中的某一段的某一个页面存放于内存,该作业就可以被运行。作业地址空间基本上不受内存空间大小的限制,只要在外存上开辟出足够大的交换区,几乎任意大小的作业都可以在虚拟存储器上运行。

9. 存储区整理

由于内存空间被划分成块,因此只存在页内碎片,页内碎片无法通过存储区的整理来取消。因此段页式管理不需要对存储区进行整理。

10. 快速寻址

快速寻址同样可以使用高速存储器来存放常用页面的页表,针对段页式管理其页表内容为段号、页号和块号。

我们已经介绍了分区管理、分页管理、分段管理、段页式管理等,下面通过表3.1来对各种管理进行比较。

表3.1 各种存储管理方案比较

管理方法 / 比较项	分区管理			分页管理	分段管理	段页式管理
	单一分区	固定多区	动态多区			
内存划分	单区	多区	变化	固定块	变化	固定块
作业划分	不分	不分	不分	页面	段	段、页面
重定位	静态	静态	动态	动态	动态	动态
分配	静态	静态	静态	静态、动态	动态	动态
存储保护	界地址	界地址	界地址	页表界地址	界地址、访问控制	界地址、访问控制
连接	静态	静态	静态	静态	动态	动态
共享				较难	能	能
硬件支持			地址变换	地址变换、缺页机构	地址变换、缺段机构、连接机构	地址变换、缺页缺段机构、连接机构
软件支持		分配算法	紧缩	淘汰算法	紧缩、淘汰算法	淘汰算法
虚拟存储器				支持	支持	支持

通过比较我们发现，不能一概而论说哪一种管理方法更为合理，每一种方法都有它的侧重面以及它的不足，在具体的操作系统中以上各种方法都有应用。

3.6 常用系统的存储管理方案

3.6.1 DOS 的存储管理

DOS 使用的是单一连续分区方式，该区域紧接着操作系统在内存的驻留部分，最大可达 640KB。当需要运行用户程序时，操作系统将用户程序一次全部调入内存。用户地址空间可以分为 4 段，它们是用来存放程序的代码段、用来存放数据的数据段、用来实现数据操作的堆栈段、用来处理数组的地址段。这些段不需要连续存放，用户甚至可以指定段的相对位置。段的起始地址由对应的段寄存器来指定，绝对地址由段地址加上段内位移来确定。

DOS 的分段只是为了方便用户程序的编写，经过分段的程序条理更清晰。由于内存的用户区最大容量限制为 640KB，用户程序也不可以突破这个界限。如果用户程序必须突破 640KB 的限制，可以采用称为覆盖的技术来进行编程。覆盖技术将程序划分为不同层次的模块，上层模块可以调用下层模块，同层模块之间由于没有相互调用关系可以在运行时间上串行，并且使用同一个覆盖区域。因此覆盖技术实际上是同层模块对同一个内存区域的覆盖使用，以此来节约内存空间。如果用户程序采用了覆盖技术，只要不同层次的最大模块的总和不超过 640KB，该程序就可以运行。不过，覆盖技术对于普通用户来说要求太高，并不受欢迎，人们也就渐渐放弃了对该技术的使用。

DOS 只能识别 1MB 以内的内存空间，如果内存容量大于 1MB，多出来的内存就会被浪费，这极大地限制了各种应用程序的开发。在 DOS 的基础上，演变开发出来的有些新的操作系统都力图冲破 640KB 的限制，但由于观念没有彻底改变，各种 DOS 最终未能走出困境。

另外，作为一个单用户系统，DOS 越来越不能适应当今多用户多任务的发展趋势，这也是它走向衰落的另一个原因。

3.6.2 Windows XP 的存储管理

Windows XP 要求的内存最小容量为 64MB，使用的是 486 以上 I386 系列 CPU 提供的段页式管理。CPU 提供了一个 16 位的段选择器和 32 位的偏移地址寄存器，因此允许用户空间分段，每段最大可达 4GB。内存空间分为大小为 4KB 的块。

Windows XP 为每个进程提供 4GB 的虚拟存储器，它由物理内存和用户驱动器根目录上的页面文件(paging file)构成，虚拟空间的实现是由在内存与页面文件之间的交换来完成的，交换的基本单位为 4KB 的页面。虚拟地址有前、后两个 64KB 的保护区，该保护区的应用是为了防止编程错误而设定的。用户程序的虚拟地址起始位实际为 0X00400000。

4GB的虚拟地址又被分割成两个2GB空间,低端的2GB提供给用户进程使用,内装用户进程的程序代码、数据等,高端的2GB为系统进程使用。

用户进程的虚拟地址空间被分为自由区、确认区和保留区三个部分:

(1) 虚拟内存的确认区:具有备用的物理内存,根据该区域设定的访问权限,用户可以进行写、读或在其中执行程序等操作。

(2) 虚拟内存的保留区:没有备用的物理内存,但具有一定的访问权限。

(3) 虚拟内存的自由区:不限定其用途,其相应的权限为不可访问。

与虚拟内存相关的访问权限告诉系统进程可在内存中进行何种类型的操作,如只读权限(可以在该区域上读,但不能写或者执行)、执行权限(不能在该区域上读或写,但能执行)、非访问权限(不允许进程对其地址进行任何操作)。

Windows XP还提供了在小块内存中动态分配和释放内存的处理程序——应用程序堆,应用程序堆对这种内存管理来说,无论是资源上还是运行上都是很高效的。另外,Windows XP还提供了其他几个内存扩展工具,如内存块清空、内存块覆盖、内存块复制等。

图3.30显示了缺页次数与时间的关系(双击"我的电脑"→"控制面板"→"管理工具"→"性能",然后添加计数器:memory→page faults/sec)。

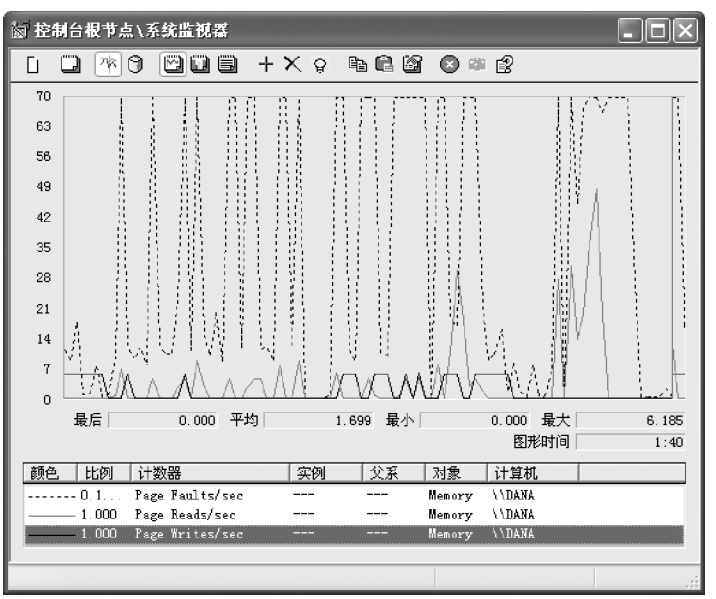


图3.30 Windows XP缺页次数变化

Page Reads/sec是系统在缺页中断后单位时间内读取磁盘页面的数量。当一个进程引用一个虚拟内存的页面,并且此页面必须从磁盘检索时,就会发生缺页中断。Page Writes/sec是指为了淘汰内存中页面而将页面回写到磁盘的速度。只有在内存中修改过的页面才会写入磁盘。

图 3.31 是 Windows XP 中虚拟内存情况。Windows XP 中的内存分配情况可通过“控制面板”→“管理工具”→“计算机管理”→“计算机管理(本地)”→“属性”→“高级”→“虚拟内存”命令查看。

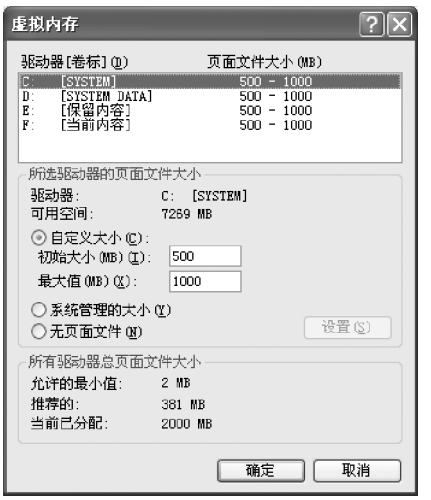


图 3.31 Windows XP 中的虚拟内存

3.6.3 Linux 的存储管理

由于 Linux 也使用的是 486 以上的微型机，因此，它也利用了分页技术来实现虚拟存储器。在系统安装时，从硬盘上划出一块区域作为交换区。系统运行时，通过页面的换进、换出来扩大存储空间。

整个虚拟空间可以划分为 16KB 个段，每个段的大小可变，最大能够达到 4GB，每个段可以提供独立的段内保护。每个虚拟地址空间(16KB 个段)可以分为相等的两个部分，一半称为全局虚拟地址空间，由全局段描述符表(Global Descriptor Tabel，GDT)映射；另一半称为局部虚拟地址空间，由局部段描述符表(Local Descriptor Tabel，LDT)映射。

支持二级分页机制，每个页面 4KB，提供段页式存储管理的硬件支持。地址转换要经过相对独立的两级地址变换。第一级使用分段机制，把包含段地址和段内偏移地址的二维虚拟地址空间转换为一个线性地址空间(也是虚拟地址空间)；第二级使用分页机制，把线性地址空间转换为物理地址空间(见图 3.32)。

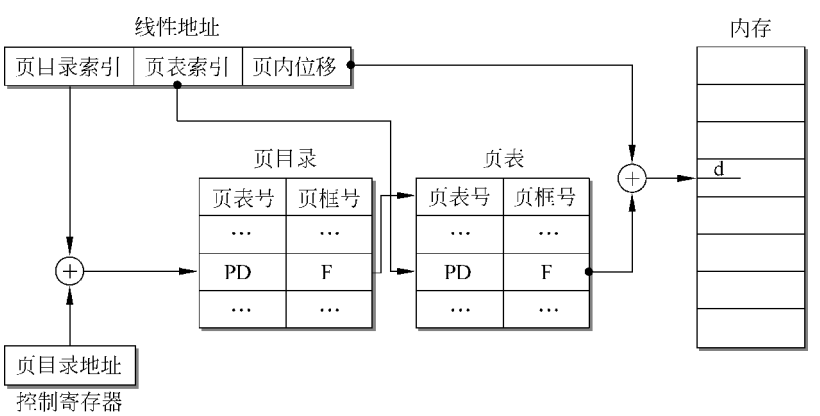


图 3.32 Linux 地址映射

线性地址由三个部分组成，分别为页目录索引、页表索引和页内位移。第一部分页目录索引描述要访问地址在页目录表中的位置，通过这个索引，就可以得到一个记录页表的内存单元；第二部分页表索引描述待访问地址在页表中的位置，通过查找页表，就得到待访问地址所在的页面；第三个部分页内位移描述待访问地址相对于页面基地址的偏移量。

在同一个任务内部,还提供4种(0～3)保护特权级,某一级特权i只可以访问所有其他大于等于这一特权级(>i)的程序段。进程的虚拟空间包括了系统(内核)空间和用户空间两个部分,它们分别处于不同的特权级。内核空间特权级为0,进程执行这个空间的指令,称为处于内核态(或者系统态);用户空间的特权级为3,进程执行这个空间的指令,称为处于用户态。用户态和核心态是同一进程的两种不同运行模式,进程在用户态和核心态执行时分别访问位于用户空间和核心空间的堆栈和数据结构。处于内核态的进程可以访问同一进程的用户空间,反之则不可以访问。

Linux采用记龄(aging)淘汰算法。Linux根据访问次数来决定是否适合换出,优先换出那些很长时间没有访问的页面。

3.7 本章小结

存储管理涉及对内存的划分、分配与释放、地址映射、存储保护、存储共享、虚拟存储器等诸多方面。分页管理简单易行,分段管理利于共享,段页式管理集中了所有存储管理方式的优点。存储管理采用动态分配和淘汰算法进行内存和外存页面的换进、换出来实现虚拟存储器。也采用动态连接技术和用户空间的合理划分来实现对存储区的共享。DOS采用了单一连续分区的方案,作业大小受内存容量的限制。Windows XP采用了动态分页管理,因此,可以实现最大为4GB的虚拟空间。Linux也是采用的动态分页管理,在内存与外存之间进行大小为4KB的页面交换。

习　题

3.1　什么是逻辑地址和逻辑地址空间?举例说明。

3.2　什么是物理地址和物理地址空间?它们和逻辑地址空间有什么关系?

3.3　为什么要进行存储分配?有哪些影响存储分配的因素?

3.4　已知主存有512KB容量,其中操作系统占用顶端40KB,有如下的一个作业序列:

作业1　要求160KB;
作业2　要求32KB;
作业3　要求280KB;
作业1　完成;
作业3　完成;
作业4　要求160KB;
作业5　要求240KB;

请用首次适应算法和最佳适应算法来处理上述的作业序列,并完成以下步骤(1)～(4),要求从空白区下端分割一块作为已分配区。

(1) 画出作业1、2、3进入主存后,主存的分配情况。

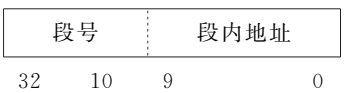
(2) 作业1、3完成后,画出主存分配情况。

(3) 画出两种算法下空白区的链接情况。

(4) 哪种算法对该作业序列是适合的?

3.5　什么是重定位？解释静态重定位和动态重定位的异同点。

3.6　构造虚拟存储器必须具备哪些条件？有一计算机系统，内存容量为 512KB，辅存容量为 2GB，逻辑地址形式如下：



求其虚拟存储器的实际容量。

3.7　有哪些存储保护的方法？如果你设计操作系统，你将采用何种方法？为什么？

3.8　以 DOS 为例说明单一分区管理。

3.9　从各个方面比较多重固定分区和多重动态分区。

3.10　有哪些分配算法？比较它们的优缺点。

3.11　何谓存储碎片，如何解决这个问题？

3.12　为什么要引入分页管理？其最大优点是什么？

3.13　比较静态分页和动态分页的差别，对于单用户机，采用哪一种方案更好？为什么？

3.14　为什么说分页管理的逻辑地址为一维地址，而分段管理的逻辑地址为二维地址？

3.15　若在一分页存储管理系统中，某作业的页表如下所示：

页　号	页框号	页　号	页框号
0	2	2	1
1	3	3	6

已知页面大小为 1024 字节，试将逻辑地址 1000、2000、3000、5012 转化为相应的物理地址。

3.16　动态分页技术支持虚拟存储器，说明对应的软硬件机构。

3.17　何谓“抖动”？它是由什么引起的？如何消除这种现象？

3.18　简述最近最少使用算法（LRU）和最近未使用算法（NRU）两种页面置换算法的思想。

3.19　若某进程分得 4 个内存块，其页面访问顺序为 1、3、4、5、2、3、4、8、6、7、5、6、5、4、2，分别求采用 OPT、FIFO、LRU 算法下的缺页次数和缺页率。

3.20　为什么引入分段管理？其最大特点是什么？需要哪些软硬件支持？

3.21　简述分段管理的动态连接及其意义。

3.22　某分段存储管理系统中，有一作业的段表如下：

段　　号	段长（容量）	主存起始地址	缺 段 位
0	200	600	1
1	50	850	1
2	100	1000	1
3	150	—	0

逻辑地址[0,100]、[1,60]、[2,85]、[3,100]对应的内容是否在主存？为什么？

3.23　为什么说相对于分页管理，分段管理更易于实现信息共享和保护？如何实现？

3.24　为什么说段页式管理集中了分页与分段管理的优点？解释之。

3.25　段页式存储系统中，为了获得一条指令或数据，需几次访问内存？

3.26　说明在分页、分段和段页式虚拟存储技术中的存储保护是如何实现的？有无碎片问题？

3.27　Windows采用怎样的存储管理方法？你认为它最出色的地方在哪里？