第 5 章 内存管理

冯·诺依曼的存储机制要求任何一个程序必须装入内存才能执行，现代计算机系统就是基于这种机制的。在计算机系统中，内存管理在很大程度上影响着这个系统的性能，这使得存储管理成为人们研究操作系统的中心问题之一。

现代计算机系统中的存储器通常由内存和外存组成。CPU 直接存取内存中的指令和数据，内存的访问速度快，但容量小、价格贵；外存不与 CPU 直接交互，用来存放暂不执行的程序和数据，但可以通过启动相应的 I/O 设备进行内存与外存信息的交换，外存的访问速度慢，但容量大大超过内存的容量，价格便宜。虽然随着硬件技术和生产水平的迅速发展，内存的成本急速下降，但内存容量仍是计算机资源中最关键且最紧张的资源。因此，对内存的有效管理仍是现代操作系统中十分重要的问题。

5.1 概　　述

5.1.1 基本概念

CPU 能直接存取指令和数据的存储器是内存，又称主存、实存，它的结构和实现方法将很大程度地决定整个计算机系统的性能，内存的大小由系统硬件决定，是实实在在的存储，它的存储容量受到实际存储单元的限制。内存是现代计算机系统操作的中心。如图 5.1 所示，CPU 和 I/O 系统都要和内存打交道。

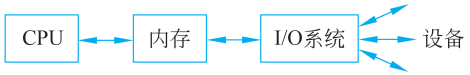


图 5.1　内存在计算机系统中的地位

内存用来存放内核、程序指令和数据，计算机当前要用到的每一项信息都存放在内存的特定单元中。内存是一个由字或字节构成的大型一维数组，每一个单元都有自己的地址，通过对指定地址单元进行读/写操作来实现对内存的访问。

1. 存储器的层次

尽管内存的访问速度大大高于外存，还是不能与高速的 CPU 相匹配，从而影响整个系统的处理速度。为此，往往把存储器分为 3 级，采用高速缓存器来存放 CPU 近期要用的程序和数据，如图 5.2 所示。高速缓存器由硬件寄存器构成，其存取速度比内存快，但成本远远高于内存，因此，在一个实际系统中的高速缓存器的容量不会很大。往往把某段程序或数

据预先从内存调入高速缓存，CPU 直接访问高速缓存，从而减少 CPU 访问内存的次数，提高系统处理速度。

在图 5.2 所示的 3 级存储器结构中，从高速缓存到外存，其容量越来越大，访问数据的速度则越来越慢，价格也越来越便宜。由于本章主要介绍内存管理，因此，不对缓存进行详细介绍。

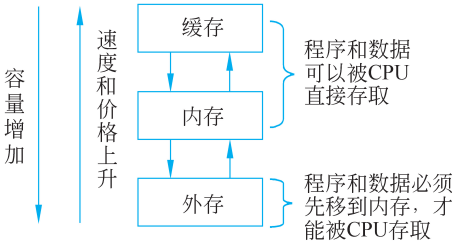


图 5.2　3 级存储器结构

2. 存储管理

系统中内存的使用一般分为两部分：一部分为系统空间，存放操作系统本身及相关的系统数据；另一部分为用户空间，存放用户的程序和数据。在单道程序系统中，内存一次只调入一个用户进程，并且该进程可以使用除操作系统占用外的所有内存空间，存储管理就是分配和回收内存区。

在多道程序系统中，多个作业可以同时装入内存，因而对存储管理提出了一系列要求：如何有效地将内存分配给多个作业，如何共享和保护内存，等等。存储管理既要提高存储资源的利用效率，同时又要方便用户使用，因此，要求存储管理具有内存空间管理、地址转换、内存扩充、内存共享和保护等功能。

1）内存空间管理

内存空间管理负责记录每个内存单元的使用状态，负责内存的分配与回收。内存分配有静态分配和动态分配两种方式。静态分配不允许作业在运行时再申请内存空间，在目标模块装入内存时就一次性地分配了作业所需的所有空间；而动态分配允许作业在运行时再请求分配附加空间，在目标模块装入内存时只分配了作业所需的基本内存空间。

采用动态分配方法的系统中，常使用合并自由区的方法，使一个空区尽可能地大。

2）地址转换

程序在装入内存执行时对应一个内存地址，而用户不必关心程序在内存中的实际位置。用户程序一旦编译之后每个目标模块都以 0 为基地址进行编址，这种地址称为相对地址或逻辑地址，而内存中各个物理存储单元的地址是从统一的基地址顺序编址，此地址也称为绝对地址或物理地址。当内存分配确定后，需要将逻辑地址转换为物理地址，这种转换过程称为地址转换，又称重定位。

3）内存扩充

内存的容量是受实际存储单元限制的，而运行的程序又不受内存大小的限制，这就需要用有效的存储管理技术来实现内存的逻辑扩充。这种扩充不是增加实际的存储单元，而是通过虚拟存储、覆盖、交换等技术来实现的。内存扩充后，就可执行比内存容量大得多的程序。存储扩充需要考虑放置策略、调入策略和淘汰策略。

4）内存的共享和保护

为了更有效地使用内存空间，需要共享内存。共享内存指共享在内存中的程序或数据。例如，当两个进程都要调用 C 编译程序时，操作系统只把一个 C 编译程序装入内存，让两个进程共享内存中的 C 编译程序，这样可以减少内存空间占有，提高内存的利用率。再者，内存共享可实现两个同步进程访问内存区。

由于多道程序共享内存，每个程序都应有它单独的内存区域，各自运行，互不干扰。当

多道程序共享内存空间时,就需要对内存信息进行保护,以保证每个程序在各自的内存空间正常运行;当信息共享时,也要对共享区进行保护,防止任何进程去破坏共享区中的信息。常用的内存保护方法有硬件方法、软件方法和软硬件结合方法。硬件方法包括界地址保护法和存储访问键方法。界地址保护法是为每个进程设置上下界地址。存储访问键法为每一个受保护的存储块分配一个保护键,不同的进程有不同的权限代码,仅当权限代码和存储保护键匹配时才允许访问。

5.1.2 虚拟存储器

在实存储器存储管理方式中,要求作业在运行前先全部装入内存,这是一种对内存空间的严重浪费,因为实际上有许多作业在每次运行时,并非用到其全部程序。再者,作业装入内存后,就一直驻留在内存中直到作业运行结束,其中有些程序运行一次后就不再需要运行了,却长期占据着内存资源,使得一些需要运行的作业无法装入内存运行,从而降低了内存的利用率,减少了系统的吞吐量,为此,引入了虚拟存储器。

虚拟存储器并不是以物理方式存在的存储器,而是具有请求调入和交换功能、能从逻辑上对内存容量进行扩充、给用户提供了一个比真实的内存空间大得多的地址空间,在作业运行前可以只将一部分装入内存便可运行以逻辑方式存在的存储器。虚拟存储器并不是实际的内存,这样的存储器实际上并不存在,只是由于系统提供了自动覆盖功能后,给用户造成的一种幻觉,仿佛有一个很大的内存供他使用一样。虚拟内存的容量比实际内存空间大得多,其逻辑容量由内存和外存容量之和所决定,其运行速度接近于实际内存速度,而每位的成本都又接近外存。虚拟存储技术将内存和外存有机地结合起来,把用户地址空间和实际的存储空间区分开,在程序运行时将逻辑地址转换为物理地址,以实现动态定位,所以实现虚拟存储技术的物质基础是:①有相当容量的辅助存储器以存放所有并发作业的地址空间;②有一定容量的内存来存放运行作业的部分程序;③有动态地址转换机构(DAT),实现逻辑地址的转换。虚拟存储器的核心,实质上是让程序的访问地址和内存的可用地址相脱离。

具体实现和管理虚拟存储器的技术主要有分页(paging)、分段(segmentation)和段页式(segmentation with paging)存储管理技术等。

虚拟存储器最显著的特点是虚拟性,在此基础上它还有离散性、多次性和交换性等基本特征。

1) 虚拟性

虚拟性是指在逻辑上扩大了内存容量,使得用户所看到的内存地址空间远大于实际内存的容量。例如,实际内存只有4MB,而用户程序和数据所用的空间都可以达到20MB或者更多。这是虚拟存储器所表现出来的最重要的特征,也是实现虚拟存储器的最重要目标。

2) 离散性

离散性是指内存在分配时采用的是离散分配方式。一个作业分多次装入内存,在装入时占用的内存也不是彼此连续的,而有可能是散布在内存的不同地方,从而避免内存空间的浪费。如果将作业装入一个连续的内存区域中,那么就会需要事先为它一次性地申请足够大的内存空间,这样使一部分内存空间暂时处于空闲状态。而采用离散分配方式,在需调入某些程序和数据时再申请内存空间,就避免了内存空间的浪费。

3）多次性

多次性是指一个作业不是全部一次性地装入内存，而是分成若干部分，当作业要装入时，只需将当前运行的那部分程序和数据装入内存，以后运行到其他部分时，再分别把它们从外存调入内存。这是任何其他存储管理方式都不具备的特征。因此，可以认为虚拟存储器是具有多次性特征的存储器系统。

4）交换性

交换性是指在一个进程运行期间，允许将那些暂不使用的程序和数据，从内存调至外存的交换区，以腾出尽量多的内存空间使其他运行进程调入内存使用，并且被调出的程序和数据在需要时再调入内存中。这种换出、换进技术，能够有效地提高内存利用率。

5.1.3 重定位

内存有物理内存和逻辑内存两个概念。物理内存由系统实际提供的存储单元所组成，相应的，由内存中的一系列存储单元所限定的地址范围称为内存空间，又称物理空间或绝对空间；而逻辑内存不考虑物理内存的大小和信息存放的实际地址，只考虑相互关联的信息之间的相对位置，其容量只受计算机地址位的限制。若处理器有 32 位地址，则它的虚拟地址空间为 2^{32}，约 4000MB，程序中的逻辑地址范围称为逻辑地址空间，又称地址空间。

在多数情况下，一个作业在装入时分配到的存储空间和它的地址空间是不一致的，因此，作业在 CPU 上运行时，其所要访问的指令和数据的实际地址与地址空间的地址不同，这种把地址空间中使用的逻辑地址转换为内存空间中的物理地址的地址转换称为重定位，又称地址映射或地址映像。

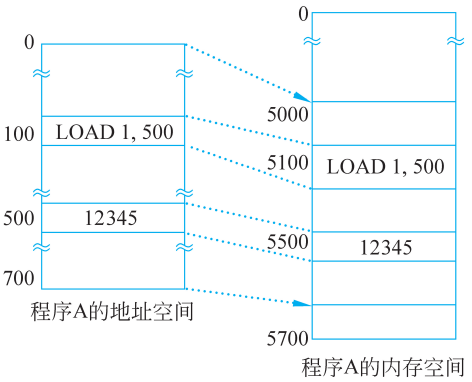


图 5.3 程序 A 装入内存时的情况

用户程序和数据装入内存时，需要进行重定位。图 5.3 描述了程序 A 装入内存前后的情况。在地址空间 100 号单元处有一条指令“LOAD 1，500”，它实现把 500 号单元中的数据 12345 装入寄存器 1。如果现在将程序 A 装入内存单元 5000～5700 的空间中，而不进行地址转换，则在执行内存中 5100 号单元中的“LOAD 1，500”指令时，系统仍然会从内存的 500 号单元中取出数据，送到寄存器 1 中，很显然，数据出了错。

由图 5.3 可以看出，程序 A 的起始地址 0 不是内存空间的物理地址 0，它与物理地址 5000 相对应。同样，程序 A 的 100 号单元中的指令放在内存 5100 号单元中，程序 A 的 500 号单元中的数据放在内存 5500 号单元中。因此，正确的方法是，CPU 执行程序 A 在内存 5100 号单元中的指令时，要从内存的 5500 号单元中取出数据（即 12345）送至寄存器 1 中，就是说，程序装入内存时要进行重定位。

根据地址转换的时间及采用技术手段的不同，把重定位分为静态重定位和动态重定位两种。静态重定位是在目标程序装入内存区时由装配程序来完成地址转换；而动态重定位是在目标程序执行过程中，在 CPU 访问内存之前，由硬件地址映射机构来完成将要访问的

指令或数据的逻辑地址向内存的物理地址的转换。

1. 静态重定位

静态重定位是由专门设计的重定位装配程序来完成的。对每个程序来说,这种地址转换只在装入时一次完成,在程序运行期间不再进行重定位。如果物理地址要发生改变,则需要进行重新装入。假设目标程序分配的内存区起始地址为FA,每条指令或数据的逻辑地址为LA,则该指令或数据映射的内存地址MA=FA+LA。例如,经过静态重定位,原100号单元中的指令放到内存5100号单元,该指令中的相对地址500相应变成5500,以后程序A执行时,CPU是从绝对地址5500号单元中取出数据12345,装入寄存器1中。如果程序A被装入8000~8700号内存单元中,那么,上述那条指令在内存中的形式将是"LOAD 1,8500"。以此类推,程序中所有与地址有关的量都要做相应变更。

这种静态重定位方式的优点是:无须增加地址转换机构,在早期的多道程序系统中大多采用此方案。它的主要缺点:一是程序的存储空间是连续的一片区域,程序在执行期间不能移动,因而就不能实现重新分配内存,这不利于内存的利用;二是用户必须事先确定所需的存储量,若所需的存储量超过可用存储空间时,则必须采用覆盖技术;三是每个用户进程很难共享内存的同一程序的副本,需各自使用一个独立的副本。

2. 动态重定位

动态重定位是靠硬件地址转换来完成的。通常由一个重定位寄存器(RR)和一个逻辑地址寄存器LR组成,其中,RR存放当前程序分配到存储空间的起始位置;LR中存放的是当前被映射的逻辑地址,映射得到的物理地址MA=LR+RR。动态映射过程如图5.4所示,只要改变LR的内容,就可以改变程序的内存空间,实现程序在内存中移动。由于这种地址转换是在作业执行期间随着每条指令的数据访问自动地、连续地进行的,所以称为动态重定位。

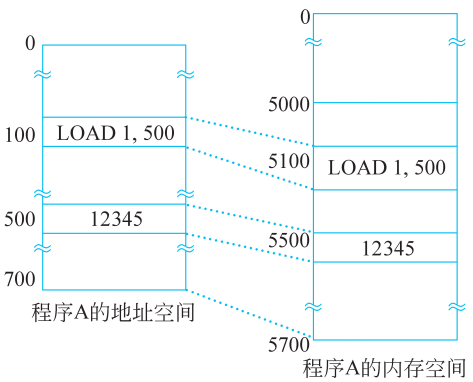


图5.4 动态映射过程示意图

动态重定位的主要优点是:内存的使用更加灵活有效,几个作业共享一程序段的单个副本比较容易,并且有可能向用户提供一个比内存空间大得多的地址空间,因而无须用户干预,而由系统负责全部的存储管理。它的主要缺点是:需附加硬件支持,且实现存储器管理的软件比较复杂。

5.2 存储管理的基本技术

最基本的4种存储管理技术是分区法、可重定位分区法、覆盖技术、交换技术,下面进行简要介绍。

5.2.1 分区法

分区管理是满足多道程序设计的一种最简单的存储管理方法。内存划分成若干大小不同的区域,除操作系统占用一个区域之外,其余由多道环境下的各并发进程共享。其基本原

理是给每一个内存中的进程划分一个适当大小的存储块，以连续存储各进程的程序和数据，使各进程能并发进行。按照分区的划分方式，又包括两种常见的分配方法：固定分区法和动态分区法。

1. 固定分区法

固定分区法就是把内存固定划分为若干不等的区域，划分的原则由系统决定。在整个执行过程中保持分区长度和分区个数不变。

图 5.5 为固定分区存储管理的示意图。为了便于内存分配，系统对内存的管理和控制采取分区说明表这样的数据结构，每一分区对应表中的一位，其中包括分区号、分区大小、起始地址和分区状态，并且内存的分配与释放、内存保护以及地址转换等都通过分区说明表来进行。

当某个用户程序装入内存时，就向系统提出要求分配内存的申请，以及需要多大内存空间，这时系统按照用户的申请去检索分区说明表，从中找出一个满足条件的空闲分区给该程序，并修改分区说明表中的状态栏，将状态置为“已分配”；如果找不到足够的分区，则拒绝为该用户程序分配内存。

当一个用户程序执行完后，就不再使用分给它的分区，于是释放相应的内存空间，系统根据分区的起始地址或分区号在分区说明表中找到相应的表项，并将其状态置为“空闲”。

固定分区法管理方式虽然简单，但内存利用率不高。如图 5.5(b)所示，若有作业 D 提出内存申请，需要 64KB 空间，系统将分区 4 分给它，这样分区 4 就有 64KB 的空间浪费了，因为作业 D 占用这个分区后，不管还有多大的剩余空间，都不能再分配给别的作业使用了。

分区号	大小	起址	状态
1	20KB	20KB	已分配
2	32KB	32KB	已分配
3	64KB	64KB	已分配
4	128KB	128KB	未分配

(a) 分区说明表

(b) 存储空间分配情况

图 5.5　固定分区的存储管理

2. 动态分区法

动态分区分配是根据进程的实际需要，动态地为它分配连续的内存空间。也就是说，各个分区是在相应作业装入内存时建立的，其大小恰好等于作业的大小。为了实现分区分配，系统中设置了相应的数据结构来记录内存的使用情况，常用的数据结构形式有空闲分区表和空闲分区链两种。

1) 空闲分区表

图 5.6 给出了空闲分区表的例子。内存中每一个未被使用的分区对应该表的一个表项，一个表项中包括分区序号、分区大小、分区的起始地址以及该分区的状态。当分配内存空间时，就从中进行查找，如找到合乎条件的空闲区就分配给作业。

2) 空闲分区链

空闲分区链是使用链指针把所有的空闲分区链接成一条链，为了实现对空闲分区的分配和链接，在每个分区的起始部分设置状态位、分区大小和链接各个分区的前向指针，由状态位指示该分区是否分配出去了；同时，在分区尾部还设置有一后向指针，用来链接后面的

序号	分区大小	分区始址	状态
1	64KB	44KB	空闲
2	24KB	156KB	空闲
3	48KB	200KB	空闲
4	30KB	300KB	空闲
5	40KB	480KB	空闲
⋮	⋮	⋮	⋮

图 5.6　空闲分区表示例

一个分区;分区的中间部分是用来存放作业的空闲内存空间,当该分区分配出去后,状态位就由“0”置为“1”,如图 5.7 所示。

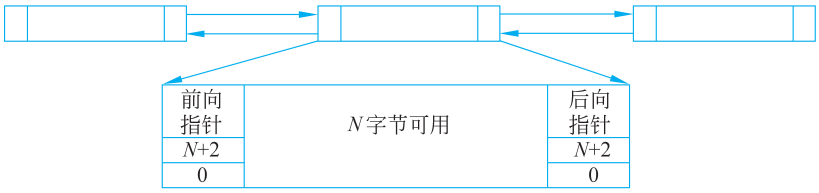


图 5.7　空闲分区链结构

采取动态分区法时,开始整个内存中只装有操作系统,当作业申请内存时,系统就查表找一个空闲区,该空闲区应足以放下这个作业,如果大小恰好一样,则把该分区分给这个作业使用,然后在空闲分区表或空闲链中再做相应的修改;如果这个空闲区比作业需要的空间大,则将该区分为两部分,一部分给作业使用,另一部分设置为较小的空闲区,当作业完成后就释放占有的分区,系统会设法将它和邻接的空闲区合并起来,使它们成为一个连续的、更大的空闲区。

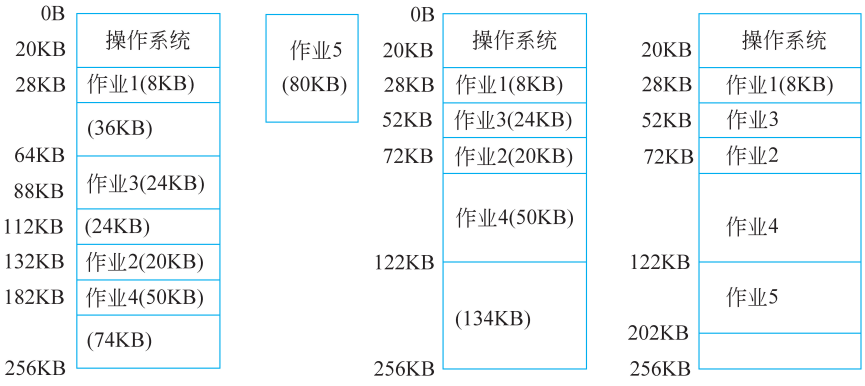
5.2.2　可重定位分区法

不管是使用固定分区法还是动态分区法,都必须把一个系统程序和用户程序装入一个连续的内存空间中。虽然动态分区法比固定分区法的内存利用率要高,但由于各作业申请和释放内存的结果,在内存中经常可能出现大量分散的小空闲区。内存中这种容量太小、无法被利用的小分区称为“碎片”或“零头”,大量碎片的出现减少了内存中作业的道数,还造成了内存空间的大量浪费。为了使分散、较小的空闲区得到合理的使用,可以定时或在分配内存时就把所有的碎片合并成为一个连续区。也就是说,移动某些已分配区的内容,使所有作业的分区紧凑地连在一起,而把空闲区留在另一端,这就是“紧凑”技术。

紧凑过程中作业在内存中要移动,因而所有关于地址的项均得做相应的修改,如基址寄存器、访问内存的指令、参数表和使用地址指针的数据结构等,采用动态重定位技术可以较好地解决这个问题。

动态重定位经常是用硬件来实现的。硬件支持包括一对寄存器,其中一个用于存放用户程序在内存中的起始地址,即基址寄存器;另一个用于表示用户程序的逻辑地址的最大范围,即限长寄存器。如图 5.8 所示,开始时作业 3 的起始地址是 64KB,其作业大小是 24KB,当执行作业 3 时,系统就把它的起始地址放入基址寄存器中,把其作业大小放入限长寄存器

中。由于系统对内存进行了紧凑，作业 3 移动到新位置，起始地址变为 28KB。再执行作业 3 时，就把 28KB 和 24KB 分别放入基址寄存器和限长寄存器中。



(a) 初始状态　　(b) 移动之后　　(c) 分配作业5之后

图 5.8　可重定位分区的紧缩

动态重定位的实现过程如图 5.9 所示。从图中可以看出，作业 3 装入内存后，其内存空间中的内容与地址空间中的内容是一样的。也就是说，将用户的程序和数据完全地装入内存中。当调度该作业在 CPU 上执行时，操作系统就自动将该作业在内存的起始地址(64KB)装入基址寄存器，将作业的大小(24KB)装入限长寄存器。当执行“LOAD 1，3000”指令时，操作对象的相对地址首先与限长寄存器的值进行比较，如果前者小于后者，则表示地址合法，在限定范围内；然后，将相对地址与基址寄存器中的地址相加，所得的结果就是真正访问的内存地址，如果该地址大于限长寄存器的内容，则表示地址越界，激发相应中断，并进行处理。

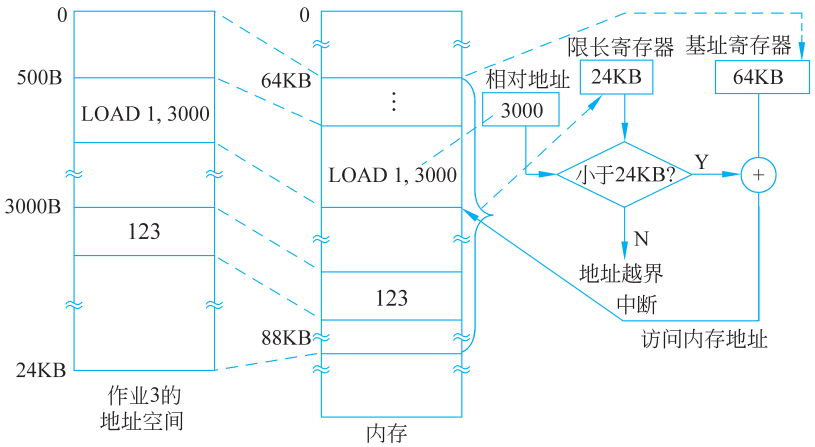


图 5.9　动态重定位的实现过程

动态重定位分区分配算法与动态分区分配算法基本上相同。二者差别仅在于：在这种分配算法中，增加了“紧凑”功能，通常是在找不到足够大的空闲分区来满足用户需求时，进行紧凑处理。

5.2.3 覆盖技术

覆盖技术主要用在早期的操作系统中,因为在早期的单用户系统中内存的容量一般少于64KB,可用的存储空间受到相当限制,某些大作业不能一次全部装入内存中,这就发生了大作业和小内存的矛盾。为此,引入“覆盖”管理技术,就是把大的程序划分为一系列的覆盖,每个覆盖是一个相对独立的程序单位,把程序执行时不需要同时装入内存的覆盖构成一组,称之为覆盖段。这个覆盖段中的覆盖都分配到同一个存储区域,这个存储分配区域称为覆盖区,它与覆盖段一一对应。并且,为了使覆盖区能被相应覆盖段中的每个覆盖在不同时刻共享,其大小应由其中最大的覆盖来确定。

覆盖技术要求程序员必须把一个程序划分成不同的程序段,并规定好它们的执行和覆盖顺序,操作系统根据程序员提供的覆盖结构来完成程序段之间的覆盖。因此,要求用户明确地描述作业中各个程序模块间的调用关系,这将加重用户负担。这种覆盖技术的例子如图5.10所示。从图中可以看出,程序段B不会调用C,程序段C也不会调用B,因此,程序段B和C不需要同时驻留在内存,它们可以共享同一内存区;同理,程序段D、E、F也可共享同一内存区。整个程序段被分为两部分:一部分是常驻内存部分,称为根程序,它与所有的被调用程序段有关,因而不能被覆盖,图中的程序A是根程序;另一部分是覆盖部分,被分为两个覆盖区,一个覆盖区由程序段B、C共享,其大小为B、C中所要求容量大者,另一个覆盖区为程序段D、E、F共享,两个覆盖区的大小分别为50KB与40KB。这样,虽然该进程正文段所要求的内存空间是190KB,但由于采用了覆盖技术,只需110KB的内存空间就可执行。

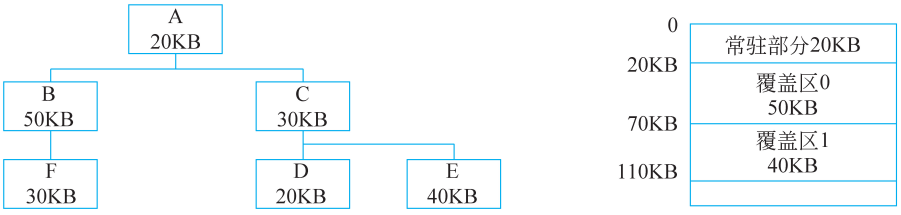


图5.10 覆盖示例

5.2.4 交换技术

交换技术就是把暂时不用的某个程序及数据的一部分或全部从内存移到外存中,以便腾出必要的内存空间,或者把指定的程序或数据从外存读到相应的内存中,并将控制权转给它,让其在系统上运行的一种内存扩充技术。实际上这是用外存作缓冲,让用户程序在较小的存储空间中,通过不断地换出/换进作业或进程来执行较大的程序。与覆盖技术相比,交换不要求程序员给出程序段之间的覆盖结构。交换主要是在进程或作业之间进行的,而覆盖则主要在同一个作业或进程中进行,而且其只能覆盖与覆盖程序段无关的程序段。

交换进程由换出和换进两个过程组成,换出是把内存中的数据和程序换到外存的交换区,而换进则是把外存交换区中的数据和程序换到内存的分区中。

交换技术是一种很有效的管理技术,通常可以解决如下问题:

(1) 作业要求增加存储空间,而分配请求受到阻塞。

(2) CPU 时间片用完。

(3) 作业等待某一 I/O 事件发生,如等待打印机资源。

(4) 紧凑存储空间,需把作业移动到存储空间的新位置上。

交换技术最早被用在 MIT 的兼容分时系统(CTSS)中,任何时刻在该系统的内存中只有一个完整的用户作业,当其运行一段时间后,或由于分配给它的时间片用完,或由于需要其他资源而等待,系统就把它交换到外存上,同时把另一个作业调入内存让其运行。这样,可以在存储容量不大的小型机上实现分时运行,早期的一些小型分时系统多数都是采用这种交换技术。

5.3 分页存储管理

无论是分区技术还是交换技术均要求作业存放在一片连续的内存区域中,这就造成了内存中的碎片问题。可以通过移动信息,利用紧缩法使空闲区变成连续的较大一块来解决问题。这种方法的实质是让存储器去适应程序对连续性的要求,以花费 CPU 时间为代价。另外,分页管理也是解决碎片问题的一种有效办法,它允许程序的存储空间是不连续的,用户程序的地址空间被划分为若干固定大小的区域,称为“页”。页面的典型大小为 1KB;相应的,也可将内存空间分成若干物理块,页和块的大小相等。这样,可将用户程序的任一页放在内存的任一块中,实现了离散分配。这时内存中的碎片大小显然不会超过一页。

5.3.1 基本概念

1. 页面和物理块

在分页存储管理中,将一个进程的逻辑地址空间划分成若干大小相等的部分,每部分称为页面或页,并且每页都有一个编号,即页号,页号从 0 开始依次编排,如 0,1,2,…。同样,将内存空间也划分成与页面大小相同的若干存储块,即为物理块或页框。相应地,它们也进行了编号,块号从 0 开始依次顺序排列为 0 号块、1 号块、2 号块……在为进程分配内存时,以块为单位将进程中的若干页分别装入多个可以不相邻的块中。由于进程的最后一页经常装不满一块,而形成不可利用的碎片,称为页内碎片。

页面和物理块的大小是由硬件即机器的地址结构所决定的。在确定地址结构时,若选择的页面较小,一方面可使内存碎片小,并减少了内存碎片的总空间,有利于提高内存的利用率;另一方面,也会使每个进程要求较多的页面,从而导致页表过长,占用大量内存;此外,还会降低页面换出换进的效率。若选择的页面较大,虽然可减少页表长度,提高换进换出的效率,但又会使页内碎片增大。因此,页面的大小应该选择适中,通常页面的大小是 2 的次幂,大约是 512B～4MB。

在分页存储管理中,表示地址的结构如图 5.11 所示。它由两部分组成:第一部分表示该地址所在页面的页号 P;第二部分表示页内位移 d,即位移量。地址长度为 32 位,其中 0 到 11 为位移量,即每页的大小为 4KB,12～31 位为页号,表示地址空间最多允许容纳 2^{20} 个页面。对于某一特定机器来说,它的地址结构是一定的,如果给定的逻辑地址空间中的地址是 A,页面大小为 L,则页号和位移量 d 可按下式求得

$$P = \mathrm{INT}\lfloor A/L \rfloor$$