第5章

存储管理

存储管理主要是研究进程(或作业)如何占用主存资源。当前计算机都是基于冯·诺依曼存储程序式的计算机,程序和数据在运行和使用时都需要存放在主存中。设计操作系统的重要目标之一是提高计算机资源的利用率,其根本途径是采用多道程序设计技术。因此,必须合理地管理主存储空间,使尽量多的进程(或作业)能够同时存放于主存以竞争处理器,保证处理器和I/O设备能够并行运行。

存储管理所研究的内容包括三个方面:取(Fetch)、放(Placement)、替换(Replacement)。

"取"是研究该将哪个进程(或进程的某些部分)从辅存调入主存。调入进程占用主存或有资格占用主存是中级调度的工作。在主存资源有限的情况下,也可以调入进程的某些部分占用主存,它一般有请调(Demand Fetch)和预调(Anticipatory Fetch)之分。前者按照进程运行需要来确定调入进程的某个部分;后者是采用某种策略,预测出即将使用的进程的某部分并调入主存。

"放"则是研究将"取"来的某个进程(或进程的某部分)按何种方式放在主存的什么地方。

"替换"是研究将哪个进程(或进程的某部分)暂时从主存移入辅存,以腾出主存空间供其他进程(或进程的某部分)占用。

在这三个方面中,"放"是存储管理的基础。目前,"放"的技术可归结成两类:一类是连续的,即运行的程序和数据必须放在主存的一片连续空间中(本章介绍的单道连续分配、多道连续固定分区和多道连续可变分区方法均属此类);另一类是不连续的,即运行的程序和数据可以放在主存的多个不相邻的块中(本章介绍的页式管理、段式管理和段页式管理即属此类)。

除介绍上述各种存储管理方法外,本章还将介绍虚拟存储管理技术,在虚拟存储管理技术中涉及替换问题。

本章内容以将进程或作业放在主存中的方式为线索,按照存储管理的发展历程详细描述连续存储分配、不连续存储分配、虚存管理策略。并且介绍在各种策略下的存储空间保护和地址变换问题。

5.1 连续空间分配

在早期的操作系统设计中，都采用连续空间分配策略。那时还没有引入进程的概念，主存分配还是以作业为单位。本节介绍将作业分配到一段连续存储空间的方法，虽然是针对将主存分配给作业，但这些方法对任何形式的空间分配都具有参考意义。连续空间分配具有易理解、访问效率高、空间利用率低等特点。

5.1.1 单道连续分配、覆盖与交换技术

操作系统中只有单道用户程序，单道用户程序连续存放于主存中。

1. 空间划分与空间保护方法

在没有操作系统的时期，整个存储空间由单个用户使用。随着系统“监督程序”的出现，存储管理也随之出现了。当时的存储管理十分简单，仅将主存空间分成两部分，如图 5.1 所示。操作系统在低地址部分(0～a 单元)。

有了操作系统后，用户不再独占主存资源。为了避免用户程序执行时随意访问操作系统占用的主存空间，应将用户程序的执行严格控制在用户区域中。这种存储保护的控制措施主要是通过硬件提供的界地址寄存器和越界检查机制来实现的。将操作系统所在空间的下界 a 存放在界地址寄存器中，用户程序执行时(即处理器在用户态下运行时)，每次访问主存时，越界检查机制都会将访问主存的地址和界地址寄存器中的值进行比较，若越界，则终止程序的执行，如图 5.2 所示。

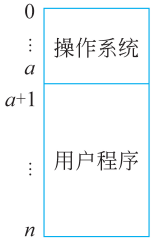

图 5.1 单道连续分配法的空间安排

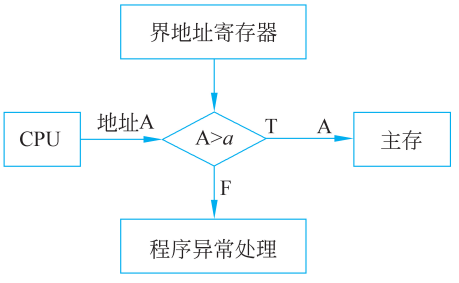

图 5.2 地址越界检查机制

2. 覆盖(Overlay)技术

早期，主存十分昂贵，因此主存的容量较小。虽然主存中仅存放一道用户程序，但是存储空间容不下用户程序的现象也常会发生，这一矛盾可用覆盖方法来解决。覆盖的基本思想是，由于程序运行时并非任何时候都要访问程序及数据的各个部分(尤其是大程序)，因此可以把用户空间分成一个固定区和一个或多个覆盖区。将经常活跃的部分放在固定区，其余部分按调用关系分段。首先将那些即将要用的段放在覆盖区，其他段放入辅存。在需要调用前，用户安排“调入覆盖系统调用”将其调入覆盖区，替换覆盖区中原有

的段。

例如，某作业各过程间有如图 5.3 所示的关系。过程 A 调用过程 B 和 C，过程 B 调用过程 F，过程 C 调用 D 和 E。由于 B 不会调用 C，C 也不会调用 B，所以过程 B，C 不必同时存入主存，同样的关系也发生在过程 D，E 之间，以及过程 D，E 和过程 F 之间。因此，用户可以建立如下的覆盖结构：将主存分成一个容量为 4KB 的固定区（由过程 A 占用）和两个覆盖区，容量分别为 6KB 和 10KB，如图 5.4 所示。将作业分成两个覆盖段：覆盖段 0 由过程 B，C 组成，覆盖段 1 由过程 F，D，E 组成。组成覆盖段的那些过程称为覆盖。

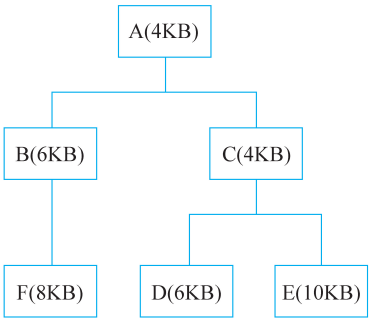

图 5.3　过程间的调用关系

图 5.4　主存区域的划分

在覆盖结构中，每个覆盖用(i,j)来表征，i 指覆盖所在的覆盖段号，j 指覆盖段中的覆盖号。本例的覆盖结构如图 5.5 所示。当作业运行时，过程段 A 占用固定区，覆盖区 0 由覆盖段 0 中的覆盖根据需要占用，覆盖区 1 由覆盖段 1 中的覆盖根据需要占用。例如，覆盖区 0 由覆盖(0,1)占用，覆盖区 1 由覆盖(1,1)占用。这时，过程 C 要调用过程 E，于是将覆盖(1,1)移入辅存，将覆盖(1,2)从辅存调入主存，占用覆盖区 1。这一变化过程如图 5.6 所示。

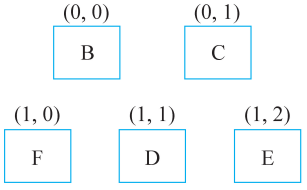

图 5.5　覆盖结构

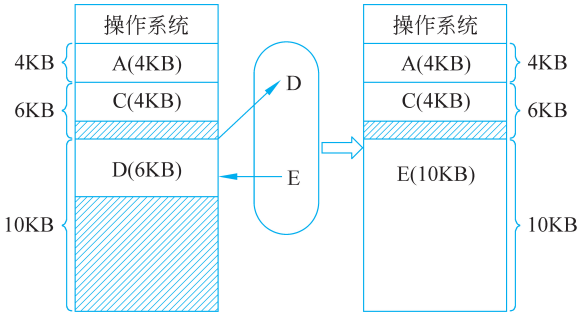

图 5.6　E 覆盖 D 的过程

采用覆盖技术是把解决空间不足的问题交给了用户。操作系统提供帮助用户将覆盖段调入主存的系统调用，但用户自己必须安排程序调入覆盖段，由此可见，覆盖技术用户参与过多，会给用户带来麻烦。

后面的虚存请调技术思想来自覆盖，但是无须额外用户编程了，访存指令能够自动判断所访问的数据不在主存而通知操作系统把数据从辅存调入主存。

3. 交换(Swapping)技术

引入交换技术的目的是想让那些在等 I/O 完成的作业先把内存腾出来给别的作业来占用处理器运行,从而让处理器与 I/O 能够并行。

交换的基本思想是,把处于等待状态的作业从主存移入辅存,这一过程称为换出;把准备好竞争处理器运行的作业从辅存移入主存,这一过程称为换入。

应特别注意,当作业处于等待状态而准备换出时,作业正在进行 I/O 操作,按理正在 I/O 操作的作业要与外设交换数据,如果是输出则要从用户空间取数据输出,如果是输入则要把数据放到用户空间,怎么能把作业滚出去呢?为了解决这个问题,在系统空间中必须开辟 I/O 缓冲区,用户作业输出时把数据先放入系统空间缓冲区或输入时从系统空间缓冲区取数据。将数据输出到外部设备中或将数据从外部设备输入的 I/O 操作必须在系统缓冲区中进行,这样,在系统缓冲区与外部设备进行 I/O 操作时,作业交换不受限制。

后面的虚存请调技术实现时,要调入新的数据或程序,但是没有空闲空间怎么办?交换技术可以作为通用技术,与后面所述的各种存储方法相结合,借助辅存的空间在逻辑上实现主存空间的扩展。

5.1.2 多道固定分区、链接与重定位技术

随着多道程序设计技术的出现及主存空间的增长,要求主存中可以存放多道作业(或进程,这时已出现进程概念),若仍利用简单的用户与系统界地址的存储保护方法,操作系统虽得以保护,但如果在某道作业运行时可能误访问其他作业空间,使其他作业的空间得不到应有的保护。因此,在多道程序设计中,为了保护用户访存不越界,硬件和软件应提供更多的支持。

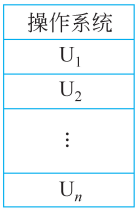

图 5.7 多道连续固定分区法的空间安排

将用户空间分成如图 5.7 所示的大小固定的几块,各块大小的选取很重要。系统初启时,可根据系统中常运行的作业(或进程)的大小来划分各块。以后在系统运转过程中不断收集统计信息,再重新修订各块的大小。

1. 链接

用户编程时一般是用高级语言编写程序源代码,会调用已经编译好的库函数,在经过编译后高级语言编写的源代码会变成二进制的目标码,这时库函数调用指令地址未确定,然后要与库函数链接,用户程序目标码和库函数的目标码被并到一起从 0 依次编址,函数的地址这时被确定,即按照库函数入口所在地址修改函数调用指令的地址域,形成最后的用户程序目标码文件,这一过程叫作**静态链接**。

后来为了解决静态链接技术出现的多个不同进程调用相同库函数引起的库函数目标码多份副本的问题,引入了动态链接技术。**动态链接**是在主存保存一份库函数代码,在编译链接时不能把库目标码合并进用户程序目标码了,只能在将进程的用户程序代码加载

到主存时或用户程序代码执行到调用函数时再确定调用函数地址。我们可以把要调用的库函数地址放到指针变量中，链接时根据库函数实际地址改变这个指针变量的值即可，如C语言的函数指针变量，调用函数指针变量代表的函数是很容易的。读者可以到网上搜索有关资料。其实**系统调用也是一种特殊的动态链接**，无须在用户程序编译链接时确定操作系统内核的系统调用处理程序地址，其特殊性在于根本就没有用函数地址来调用，只是用系统调用号查系统调用函数表，再转调函数表中的函数。

2. 地址重定位与空间保护方法

地址重定位是指将用户程序目标码中相对于 0 地址开始的所有指令、数据逻辑地址变换成指令、数据所在的主存物理地址。主要的重定位方法有两种：静态重定位，即在操作系统将目标代码加载到主存时，将目标代码所有地址域改为“原地址＋目标代码所在主存起始地址”；动态重定位，由硬件地址转换机制实现，在执行访存指令时将“原地址＋目标代码所在主存起始地址”后进行访问。

注意，链接主要是强调将原来不在一起的模块中的函数地址在统一的逻辑地址空间中确定好，而重定位强调把逻辑地址定位到物理地址。

多道连续固定分区法所依赖的**保护机制**有两种：一种是上、下界寄存器和地址检查机制；另一种是基址寄存器、长度寄存器和动态地址转换机制。前者要求用户代码是静态重定位的(用户代码中使用相对于 0 的地址，加载程序在确定其主存存放位置并加载到主存后将其修改成绝对地址)，后者要求用户的代码是动态重定位的(用户代码中的相对地址在指令执行时才被动态地转换成绝对地址)。

上、下界寄存器是硬件提供的一对寄存器，分别存放正在运行程序的上、下界。当处理器资源分给某作业时，即将该作业的上、下界地址分别装入上、下界寄存器。

地址检查机制是指，当用户程序被执行时，每次访问主存时，该机制都会将指令的访存地址与上、下界寄存器的值进行比较。若其值介于上、下界之间，则可用该地址访问存储器，否则，终止程序的运行，如图 5.8 所示。

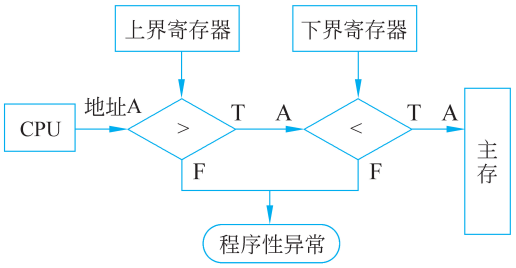

图 5.8　地址检查机制

基地址寄存器、长度寄存器分别存放运行程序在主存的起始地址及其总长度。当处理器资源分给某作业时，即将其主存的起始地址和长度分别装入基地址寄存器和长度寄存器。

动态地址转换机制是指，当用户程序运行时，每次访问主存时，该机制都会将指令的访存地址（相对地址）与长度寄存器中的值进行比较。若越界，则终止该程序；否则，与基地址寄存器中的值相加成为访问主存的绝对物理地址，如图 5.9 所示。

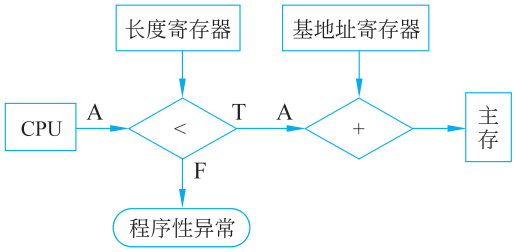

图 5.9　动态地址转换机制

如果某多道连续固定分区存储管理系统使用基地址寄存器和长度寄存器进行地址变换与存储保护，如第 3 章所述，进程控制块（PCB）包含进程映像位置信息。PCB 中的位置信息应该包括进程的基地址与程序长度。当进程调度程序选中某个进程后，应将其 PCB 中的这些位置信息内容装入基地址寄存器和长度寄存器。

3. 存储碎片

多道连续固定分区法与单道连续分配法相比，虽然提高了空间利用率，但对空间的利用仍不充分。由于进入各存储块的作业长度往往短于该块的长度，因而存在一些未加利用的存储空间。另外，若大作业较多，则小存储块常处于空闲状态，从而造成浪费。这些未得到利用的空间称为存储碎片（Memory Fragmentation）。

存储碎片分为内部碎片和外部碎片。若存储块的长度为 n，其存储的作业长度为 m，则剩下的（长度为 $n-m$）空间称为该块的内部碎片；若存储块的长度为 n，如果一直没有适合该块的作业，长时间得不到使用，则称该块为外部碎片。在多道连续固定分区法中，这两种碎片都存在。因此人们提出了要多少空间给多少空间的可变分区法。

5.1.3　多道连续可变分区法

多道连续固定分区法存在碎片问题，故人们又引入了多道连续可变分区法。这种方法对用户存储区域实施动态分割，申请者要多大空间给多大空间，从而改善了空间的利用效果。这种方法虽然是作业空间分配被引入，现在也被广泛用于各种存储空间分配中。

1. 管理方法

系统设置一张表，登记主存空间用户区域中未占用的块（空闲块）。当作业被中级调度选中后，即可在空闲块中分配空间。例如，主存的总存储量若为 257KB，操作系统占用 40KB。假设在任何一段时间里，驻留在主存中的每道作业都获得相等的处理器时间。作业队列见表 5.1（假设作业运行时间是指处理器时间，不含 I/O），则存储区域的变化如图 5.10 所示。

表 5.1　作业队列

作业队列次序	所需存储量/KB	运行时间/s
1	060	10
2	100	05
3	030	20
4	070	08
5	050	15

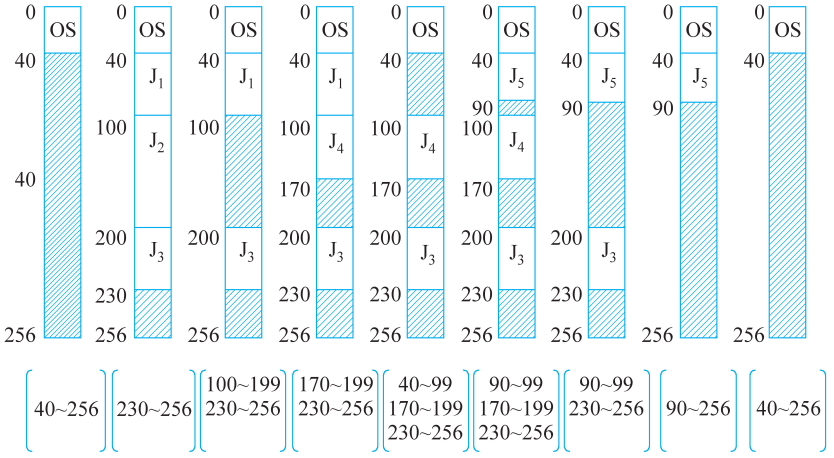

图 5.10　存储区域的变化

多道连续可变分区法的存储管理较复杂，需要用到作业分配和回收存储空间的算法。

(1) 分配存储空间。

中级调度程序为选中的作业分配存储空间，则在可用块集合中按某种策略选择一个大小满足该用户作业要求的可用块分配给用户。若选中的块比该用户作业需要的量大，则应将剩余部分回收到可用块集合中。假设 F 为可用块集合，size(k)为块 k 的大小，size(v)为用户所需的存储量。算法可表示如下。

① 如果对所有的 $k \in F$，均有 size(k)＜size(v)，则分配失败。

② 否则，按下述一种“分配策略”选出 $k \in F$，使得 size(k)≥size(v)。

③ $F = F - \{k\}$。

④ 如果 size(k)－size(v)＜ε(ε 为基本存储分配单位的大小)，则将块 k 分给该用户。

⑤ 否则分割 k 为 k' 和 k''，其中，size(k')＝size(v)。将 k' 分给用户同时回收 k''，$F = F \cup \{k''\}$。

分配策略：满足作业要求的可用块可能有很多块，那么应该选哪块分给该作业呢？有下述三种选择方法。

① 首次满足法(First Fit)。搜索 F 时，选择所碰到的第一个满足作业要求的存储块

分配给用户。

② 最佳满足法(Best Fit)。在 F 中选出所有满足作业要求的存储块中最小的一块分给用户。

③ 最大满足法(Largest Fit)。在 F 中选出所有满足作业要求的存储块中最大的一块分给用户。

基本存储分配单位:ε 为系统规定的基本存储分配单位,若分配后的剩余量比 ε 还小,则把该块全部分给用户。

对于上述三种分配策略,若采用后两种方法,则需将可用块排序。采用最大满足法仅需搜索 F 中的第一个元素(F 中元素按从大到小的次序排列),而最佳满足法搜索的平均次数为 $n/2(n=\sharp(F))$。首次满足法则无须对可用块排序。Knuth 和 Shore 分别就这三种方法对存储空间的利用情况做了模拟实验。结果表明,首次满足法可能比最佳满足法好,而首次满足法和最佳满足法一定比最大满足法好。

(2) 回收空间。

当作业撤出时,需要回收作业所占空间。收回的空间加到可用块集合 F 中。若收回的块与 F 中某些块相邻,则应合并这些块。例如,图 5.10 中,在作业 J_4 撤离时,所释放的块(100～169)与块(90～99)和块(170～199)合并。

2. 可用空间的管理

就存储保护而言,多道连续可变分区法所需的硬件支持与多道连续固定分区法一样。多道连续可变分区法一般用数组或链表管理可用空间。

(1) 数组管理。用一个数组登记可用空间的分配情况。数组的最大项数为用户空间的总存储量/基本存储分配单位。当数组项为 0 时,表示该项对应的存储分配单位为空闲;为 1 时,表示占用。为节省空间,数组项可以用 1 位表示,这就用 bitmap 表来表示基本存储单位的空闲情况。

(2) 链表管理。在每个可用块的低地址部分设两个域,分别是指针域和表示块长的长度域。将所有可用块使用指针串起来,系统掌握表头。

采用可变分区,没有内部碎片(一般不把小于基本存储分配单位的未利用的空间看成碎片),但是有外部碎片,并且外部碎片现象经常很严重。这可以用**紧致(Compact)空间**的方法予以消除。紧致空间的基本思想是,通过移动主存中的作业位置,使可用空间连成一片。要实现紧致空间,必须要求作业代码可动态重定位。可以设计一个系统进程负责紧致空间的工作,该进程平时处于睡眠状态,当外部碎片较多时将其唤醒,或者系统定期唤醒它。紧致空间需要花费很多时间,并且在紧致空间时不允许被移动的作业运行,否则难以保证正确性。

本节介绍的对主存管理的三种管理方法有一个共同特点,即用户作业在主存中是连续存放的。表 5.2 对这三种方法进行了比较和总结。这类方法的优点是硬件支持简单,但无论何种方法都有大量的存储碎片。多道连续可变分区法虽可利用紧致空间的方法消除外部碎片,但时间和空间耗费都太大。

表 5.2　连续空间分配小结

项目 / 方法	作业个数	内部碎片	外部碎片	硬件支持	可用空间管理	解决碎片方法	解决空间不足	提高作业个数
单道连续分配法	1	有	无	界地址寄存器越界检查机制	—	—	覆盖	交换
多道连续固定分区法	≤ N（用户空间划成 N 块）	有	有	1. 上、下界寄存器，越界检查机制 2. 基地址寄存器、长度寄存器、动态地址转换机制	—	—		
多道连续可变分区法	不定	无	有		1. 数组 2. 链表	紧致		

5.2　不连续空间分配

连续存储分配容易出现大段的连续空间因不能容纳作业或进程而不可用。因此，为了充分利用存储空间资源而引入了不连续空间分配策略。

5.2.1　页式管理

连续分配存储空间存在的许多存储碎片和空间管理较复杂（指多道连续可变分区法）的问题，其原因在于，连续分配要求把作业（进程）放在主存的一片连续区域中。页式管理避开了这种连续性要求。例如，有一个长度为 3 的进程，而主存中当前仅有两个长度为 2 和一个长度为 1（都小于该进程长度）的可用块，但总长度又可以满足。连续分配法对此的唯一解决办法就是紧致空间，然而紧致空间的操作开销很大。在页式系统中，将进程和主存都分成较小的块，可将进程的各块非连续地分配到可用块中。这样做既不用移动进程又解决了碎片问题。

1. 逻辑空间与物理空间

在连续存储分配中，用户作业的地址与主存地址有简单的对应关系。在页式系统中，因为连续性被破坏，所以用户程序目标代码所用的地址与程序和数据在主存中所对应的地址已失去这种简单对应关系。为此需要对这两种空间加以区分。用户程序目标代码所设想的空间和所用地址称为逻辑空间和逻辑地址；其所占主存空间称为物理空间，对应的地址称为物理地址。在页式系统中，逻辑空间、物理空间均以相同长度为单位进行等分。逻辑空间所划分出的每个区域称为页（Page）或页面；物理空间所划分出的每个区域称为页帧（Page Frame）。

系统在最初启动时，把所有页帧作为可用页帧放在一个队列中。当用户作业申请空间时，便从可用队列中按申请的量分配页帧。整个逻辑空间中的页集合可以离散地以页为单位存储在主存中。当某用户释放空间时，系统将释放的页帧回收到可用队列中。

2. 动态地址转换机制

页式方法中逻辑地址与物理地址之间失去了自然联系。必须在程序运行时，使用由

硬件提供的动态地址转换机制把逻辑地址映射成对应的物理地址，程序才能正确运行。

1）页表

由于逻辑地址和物理地址不一致，因此必须把每页第一个单元逻辑地址所对应的物理地址登记在一张称为页表的表中。逻辑空间若有 n 页，页表就应该有 n 项。为了节省空间，页表中登记的物理地址可以由页帧号替代。

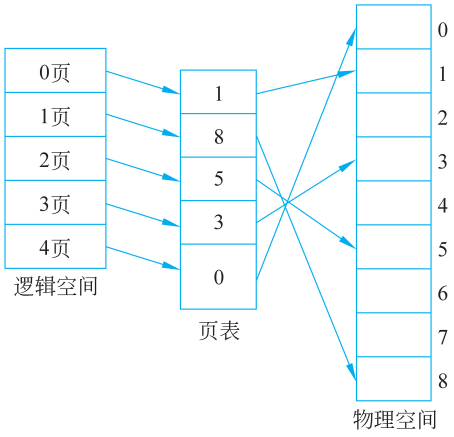

图 5.11　页表项的内容

页表的第 i 项描述第 i 页。例如，用户作业由 5 页组成，分别放在第 1，8，5，3，0 号页帧中。页表项的内容如图 5.11 所示。在页式系统中，系统空间设置一片区域作为页表区，系统为每道作业（进程）提供一个页表。如果是进程，则页表的起始地址存放在进程 PCB 表的页表始地址信息栏目中。

2）地址结构

在页式系统中，为了通过页表把逻辑地址（LA）转换成物理地址（PA），必须把线性的逻辑地址分解成页号、页内位移，分别记为 P，d。利用页号通过查页表得到页帧号，再由页帧号和页内位移可以计算出线性的物理地址。页帧号、页帧内位移记为 f，d（因为页与页帧大小相同，故页内位移与页帧内位移等值。例如，页大小为 512B，地址 539 属于第 1 页，位移为 27）。在求解逻辑地址对应的物理地址时，首先应分解出逻辑地址的页号和页内位移，然后按页号查找对应的页表项得到 f。按空间的划分规则可知：

$$P=\mathrm{LA}/\text{页大小},\quad d=\mathrm{LA}-P\times\text{页大小}$$

地址转换过程为：通过逻辑地址分别求出 P，d；将页表始地址加上 P 得到页表项地址；从页表项中获得该页所驻留的页帧号 f；再将 f 乘以页大小加上 d 就得到所要的物理地址，如图 5.12 所示。即：$\mathrm{PA}=f\times\text{页大小}+d$。

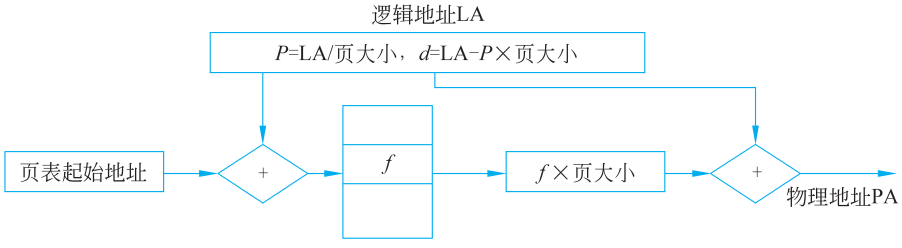

图 5.12　地址转换原理

3）把页大小设为 2 的幂可以节省地址转换开销

上述过程要做加、减、乘、除运算，耗时太多。因为计算机采用二进制编码，所以如果取页大小为 2 的正整数次幂，乘、除运算就变成了位移运算。例如，取页大小为 2^9（512）B，则逻辑地址的低 9 位为页内位移，高位为页号。这样，进行地址转换时可以不做乘、除运算。