第5章 存储管理

在计算机系统中，内存是关键资源，对内存如何处理在很大程度上将影响整个系统的性能。

存储管理目前仍是人们研究操作系统的中心问题之一，甚至操作系统的命名也往往取决于存储管理的策略。

主要内容

5.1 存储管理概述

5.2 程序的装入与链接

5.3 连续存储管理

5.4 页式存储管理

5.5 段式存储管理

5.6 段页式存储管理

5.7 虚拟存储管理方式

5.8 Linux存储管理

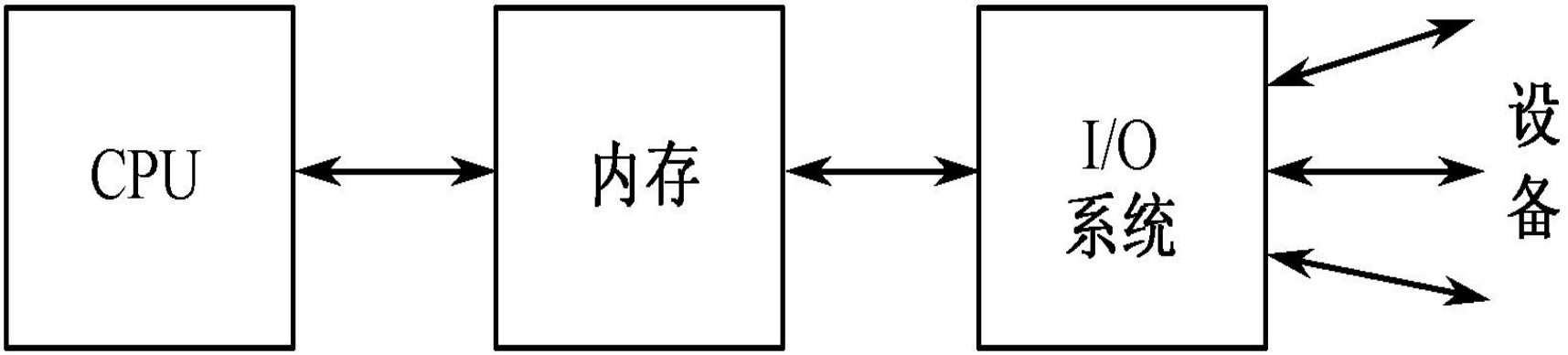
5.9 本章小结

本章要点

- 了解存储管理的主要功能；
- 理解程序的装入与连接方式；
- 掌握连续存储管理方式和非连续分配管理方式（页式存储管理；段式存储管理；段页式存储管理）；掌握虚拟存储器的基本概念、请求分页存储管理以及常用的**页面置换算法**。

5.1 存储管理概述

- 主存储器简称主存或内存（Main Memory或Primary Memory或Real Memory），是指CPU能直接存取指令和数据的存储器。



内存在计算机系统中的地位

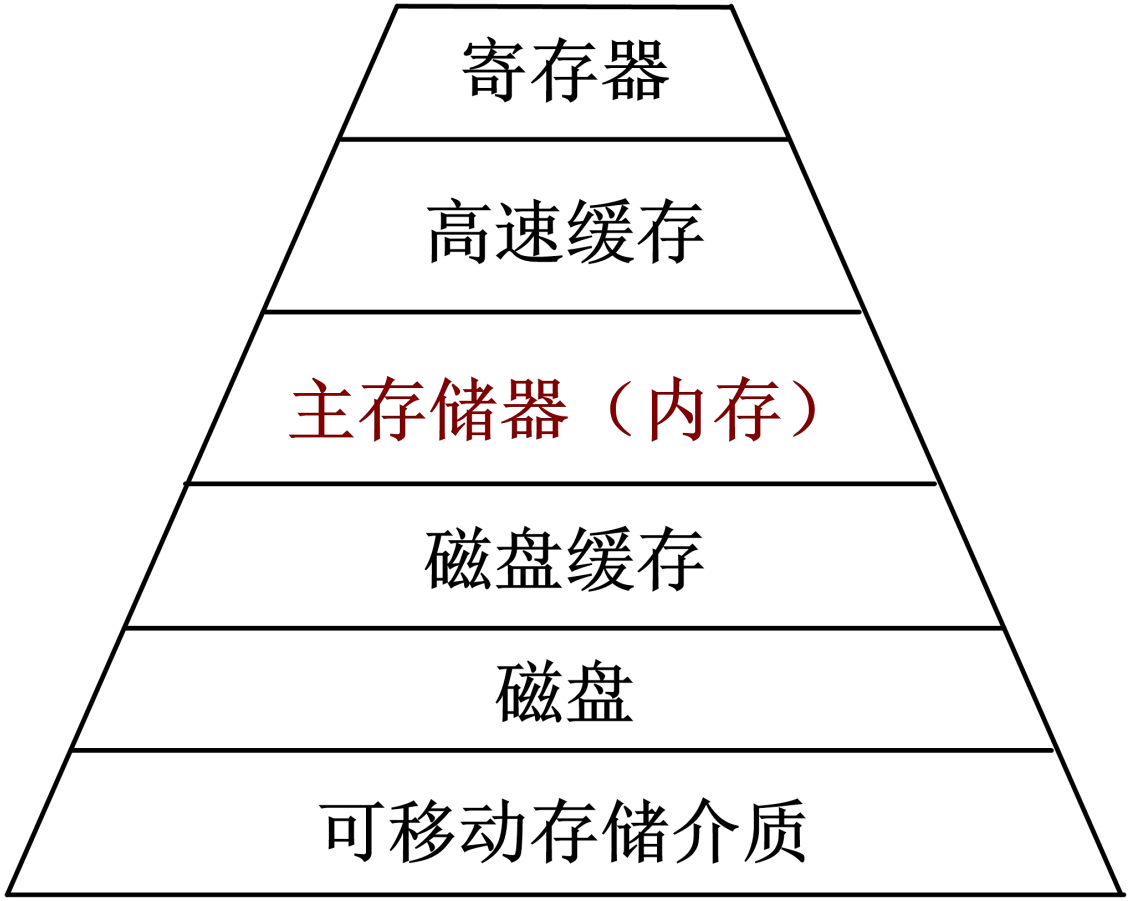
5.1 存储管理概述

- 主存可分为系统区和用户区。
  - 系统区位于主存的底端，存放操作系统内核的代码和静态数据结构，可能被其他程序或数据覆盖。
  - 用户区存放用户程序和数据，用户程序运行时进行分配，执行结束时释放。

- 存储管理是对主存中的用户区进行管理，其目的是尽可能地方便用户和提高主存空间的利用率，使主存在成本、速度和规模之间获得较好的平衡。

5.1.1 存储器的存储结构

- 存储器的层次



5.1.2 存储管理的功能

- **主存空间的分配与去配**
  - 分配或回收进程主存空间
- **实现地址转换**
  - 逻辑地址到物理地址的映射，重定位
- **主存空间的共享和保护**
  - 共享主存资源、共享主存的某些区域
  - 读写控制
- **主存空间的扩充**
  - 虚拟存储器

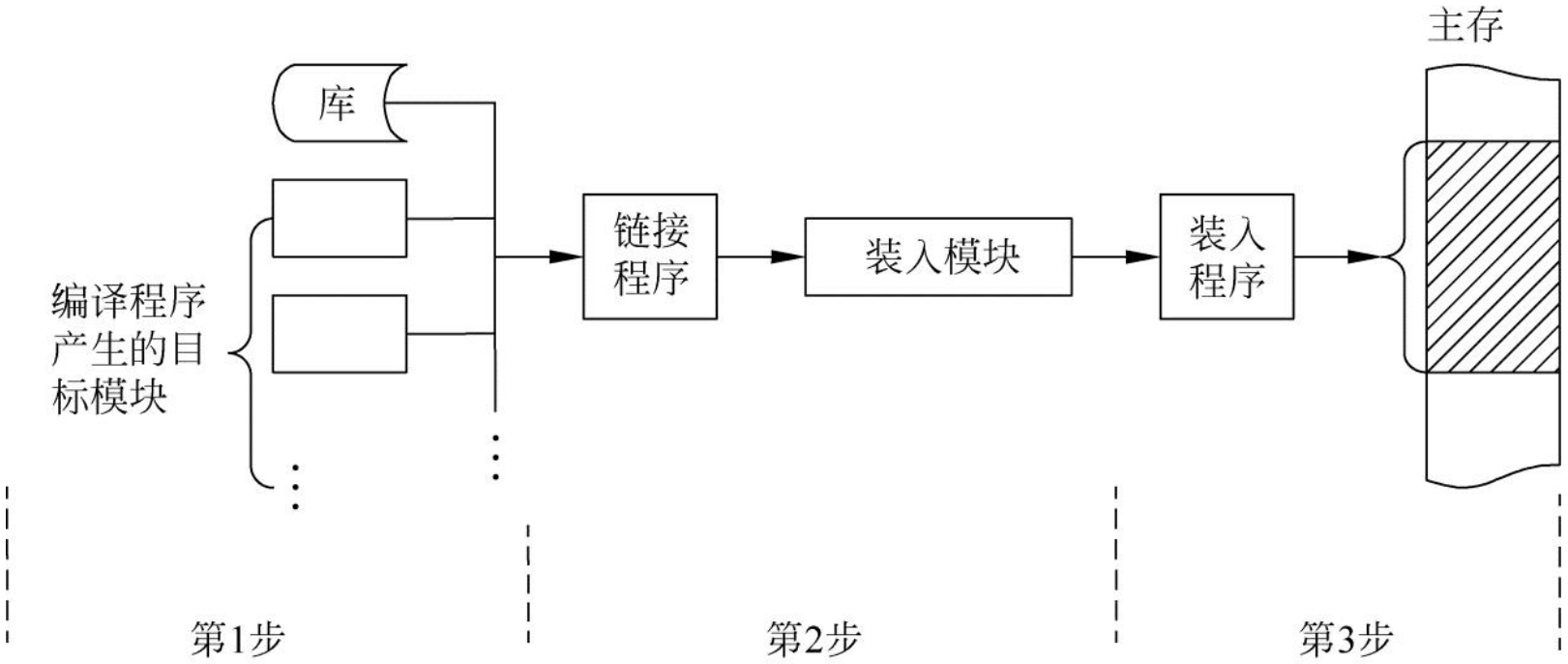
5.2.1 物理地址与逻辑地址

- **物理地址空间**：物理地址是内存中存储单元的真实地址，所以又称为绝对地址或实地址。由物理地址所对应的主存空间称为物理地址空间。

- **逻辑地址空间**：逻辑地址也称为相对地址或虚地址，它不是内存的物理地址，不能作为程序运行时的实际地址进行读写访问。由逻辑地址对应的存储空间称为逻辑地址空间。

5.2.2 程序的装入

- 程序主要处理阶段
  - 编辑、编译、链接、装入、运行



5.2.2 程序的装入

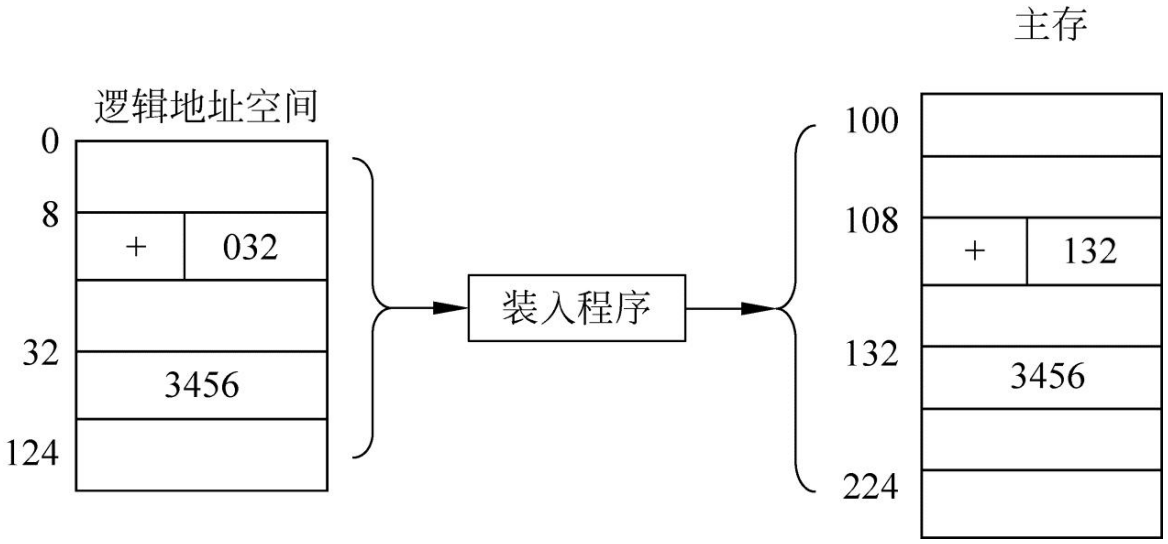
- 装入方式
  1. 绝对装入方式：编译程序产生绝对地址目标代码
  2. 可重定位装入方式：目标模块起始地址为0，其余为相对地址
  3. 动态运行时装入方式：进程的内存映像在不同时候可以处于不同的位置

5.2.2 程序的装入

- 绝对装入方式
  - 如果在编译时知道程序驻留在主存的具体位置，则编译程序将产生物理地址的目标代码。
  - 绝对装入程序按照装入模块中的地址，将程序和数据装入主存。
  - 模块装入后，由于程序中的逻辑地址与实际主存的地址完全相同，所以不需要对程序和数据的地址进行修改。
  - 绝对装入方式只能将目标模块装入到主存储器事先指定的固定位置，它只适用于单道程序环境。

5.2.2 程序的装入

- 静态重定位装入方式
  - 在装入作业时，把该作业中的指令地址和数据地址一次性全部转换成物理地址，这样在作业执行过程中无需再进行地址转换工作，这种地址转换方式称“静态重定位”。而这种作业装入的方式称为“静态重定位装入方式”。



5.2.2 程序的装入

- 动态重定位装入方式
  - 在装入作业时，装入程序直接把作业装入到所分配的主存区域中。在作业执行过程中，随着每条指令或数据的访问由硬件地址转换机制自动地将指令中的逻辑地址转换成对应的物理地址。这种地址转换的方式是在作业执行过程中动态完成的，故称为“动态重定位”。而这种作业装入的方式称为“动态重定位装入方式”。

5.2.2 程序的装入

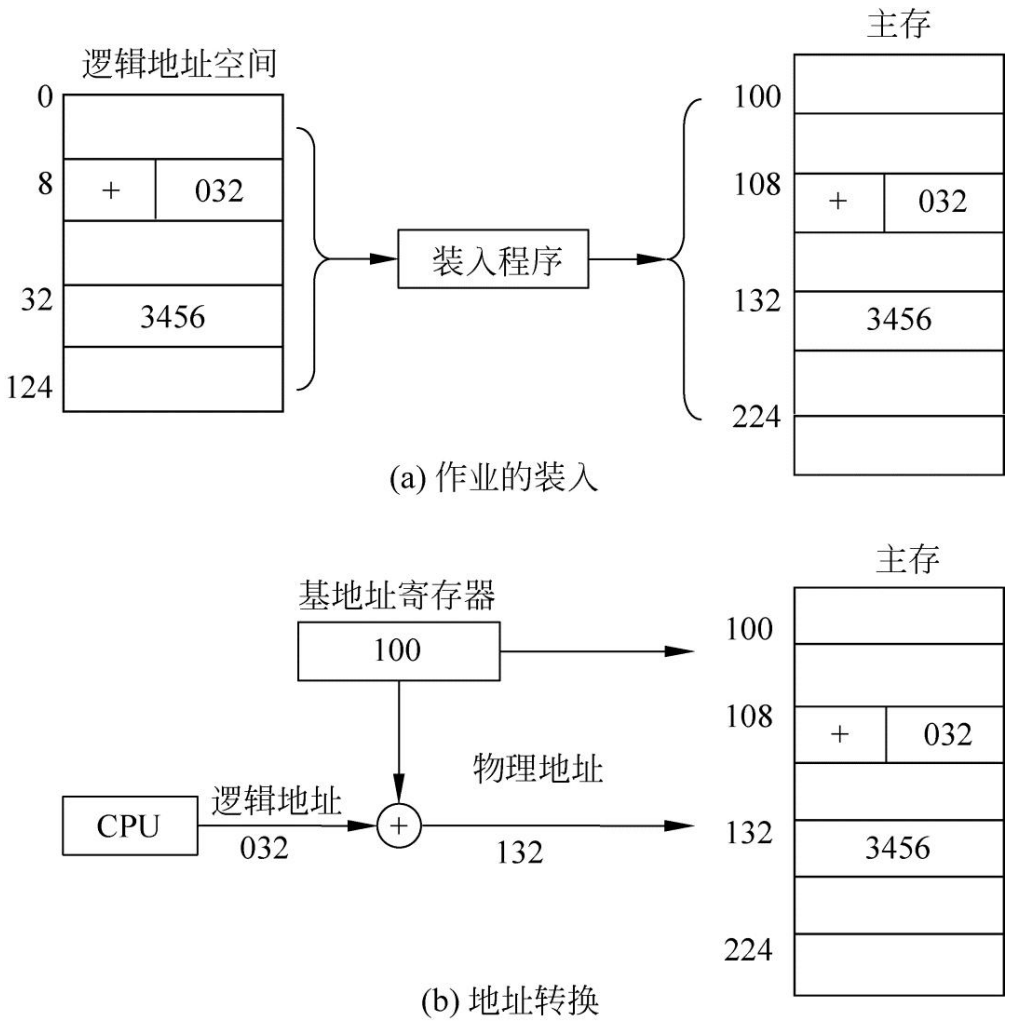


图5-4 动态重定位

5.2 程序的装入与链接

- 采用动态重定位时，由于装入主存的作业仍保持逻辑地址，所以必要时可改变作业在主存储器中的存放区域。作业在主存中被移动位置后，只需要把新区域的起始地址代替原来基址寄存器中的地址，这样，作业执行时，硬件地址转换机制将按新区域的起始地址与逻辑地址相加，转换成新的物理地址，使作业仍可正确执行。
- 若作业执行时被改变了存储区仍能正确运行，则称程序是可浮动的。
- 采用动态重定位的系统支持“程序浮动”。
- 采用静态重定位的系统不支持“程序浮动”。

5.2.3 程序的链接

- 源程序经过编译后，得到一组目标模块，再通过链接程序将这组目标模块链接，形成一个完整的装入模块。

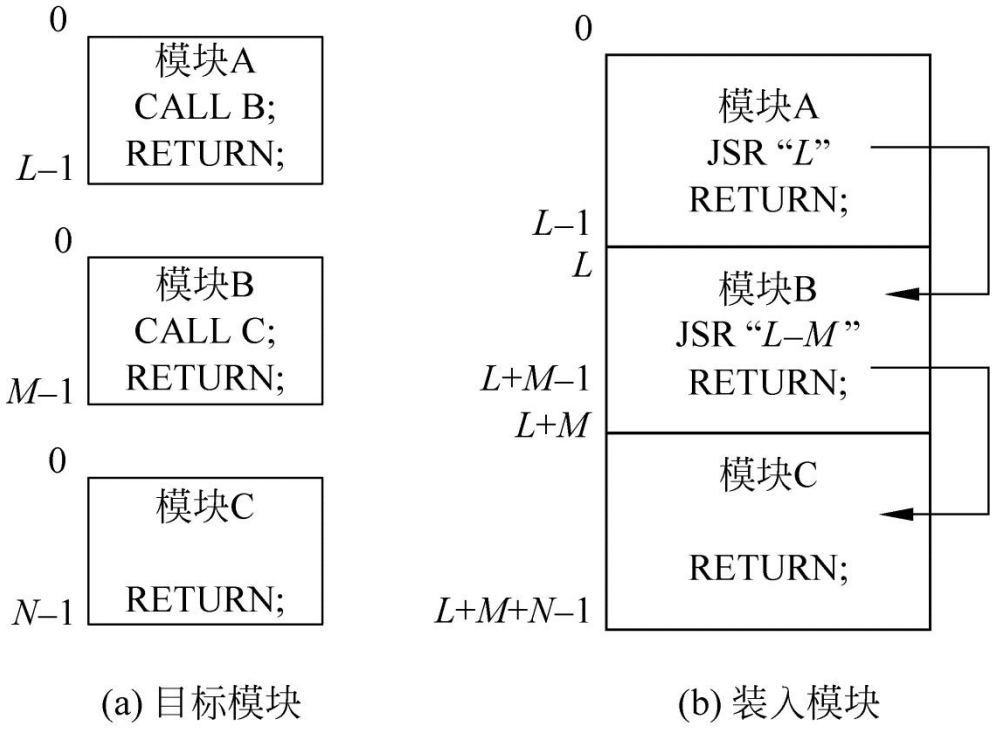


图5-5 程序链接示意图

5.2.3 程序的链接

- **静态链接**
  - 在程序运行之前，首先将各个目标模块以及所需要的库函数，链接成一个完整的装入模块，又称为可执行文件。
  - 经过编译后得到的目标模块，每个模块的起始地址都为“0”，模块中的地址都是相对于起始地址计算的，在链接成一个装入模块后，程序中被调用模块的起始地址不再是“0”，此时须修改被调用模块的逻辑地址，同时每个模块中使用的外部调用符号也相应转变为逻辑地址。如图5-5（b）中，B和C模块的起始地址分别变更为L和（L+M），而原B模块中的所有逻辑地址都要加上L，原C模块中的所有逻辑地址都要加上（L+M）。

5.2.3 程序的链接

- **装入时动态链接**
  - 用户源程序经编译后得到的一组目标模块，在装入主存时，采用边装入边链接的方式，即在装入一个目标模块时，若需要调用一个模块，则找出该模块，并将它装入主存，并修改目标模块中的逻辑地址。
  - 采用动态链接方式，可以容易地实现对目标模块的修改和更新，同时便于实现对目标模块的共享。
- **运行时动态链接**
  - 是指在程序执行过程中当需要该目标模块时，才把该模块装入主存，并进行链接。
  - 运行时动态链接不仅可以加快程序装入的速度，而且可以节省大量的主存空间。

5.3 连续存储管理

- 连续存储空间管理，是指把主存储器中的用户区作为一个连续区域或者分成若干个连续区域进行管理。
- 连续存储管理方式可分为单一连续分配、固定分区分配以及可变分区分配等方式。

5.3.1 单一连续存储管理

- 单一连续存储管理是一种最简单的存储管理方式。在单一连续存储管理方式下，操作系统占用一部分主存空间，其余的主存空间作为一个连续分区全部分配给一个作业使用，即在任何时刻主存储器中最多只存有一个作业。

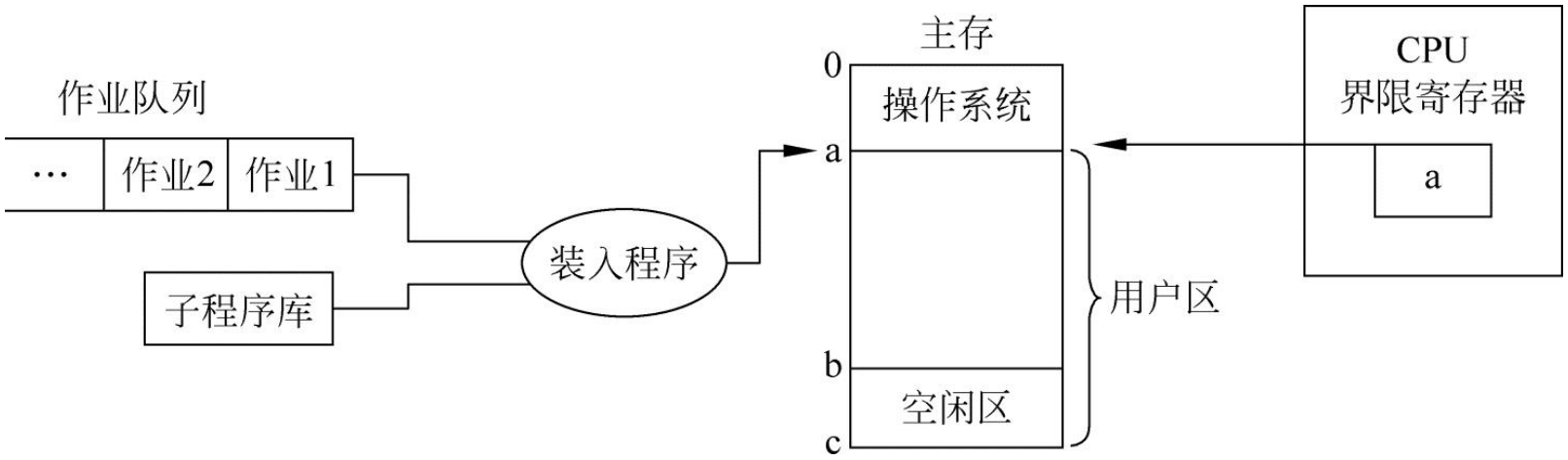


图5-6 单一连续存储管理的示意图

5.3.1 单一连续存储管理

- 单一连续存储管理每次只允许一个作业装入主存储器，因此可以不必考虑作业在主存中的移动问题，所以当作业被装入主存时，系统一般采用静态重定位方式进行地址转换，当然也可以采用动态重定位方式。
- 单一连续存储管理方式下的存储保护比较简单，即判断物理地址≥界限地址，并且物理地址≤主存最大地址，若条件成立，则可执行，否则有地址错误，形成“地址越界”程序性中断事件。当采用静态重定位装入方式时，由装入程序检查其物理地址是否超过界限地址，若是，则可以装入；否则，将产生地址错误，程序不能装入。这样，一个被装入的作业，总能保证在用户区中执行，避免破坏系统区中的信息，达到“存储保护”的目的。

5.3.1 单一连续存储管理

- 单一连续存储管理方式适用于单道程序系统，适合于单用户、单任务的操作系统，在个人计算机和专用计算机系统中可采用。

- 采用这种管理方式存在几个主要缺点：
  - CPU利用率比较低。当正在执行的作业出现某个等待事件时，CPU便处于空闲状态。
  - 存储器得不到充分利用。不管用户作业的程序和数据量的多少，都是一个作业独占主存的用户区。
  - 计算机的外围设备利用率不高。

5.3.2 固定分区存储管理

- 预先把主存储器中的用户区分割成若干个连续区域，每一个连续区域称为一个分区，每个分区的大小可以相同，也可以不同。但是一旦分割完成后，主存储器中分区的个数固定不变，每个分区的大小固定不变。每个分区可以装入一个作业，不允许一个作业跨分区存储，也不允许多个作业同时存放在同一个分区中。
- 固定分区存储管理适合多道程序设计系统。

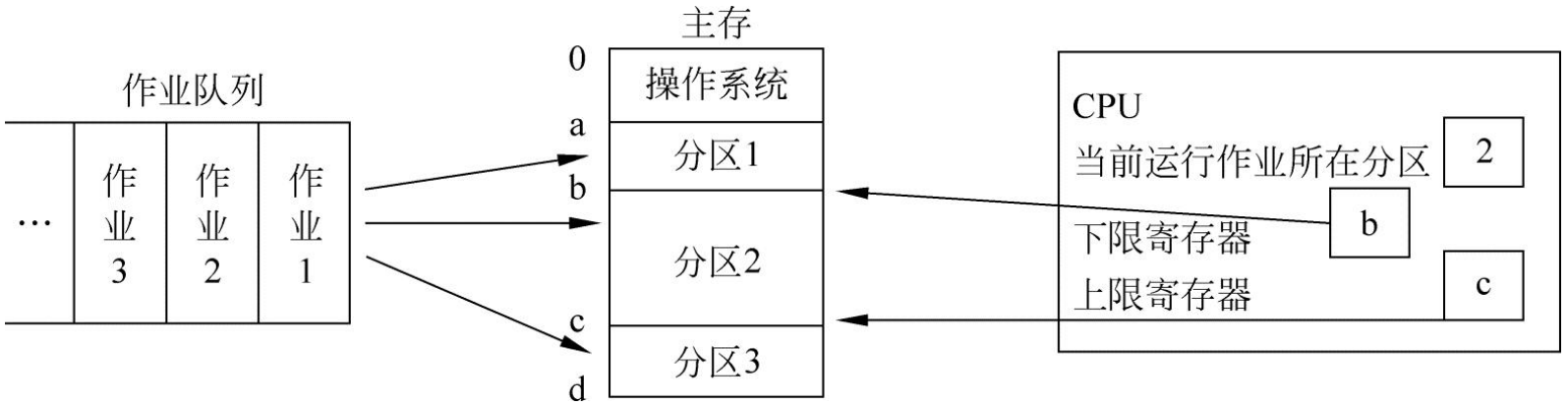


图5-7 固定分区存储管理方式的示意图

5.3.2 固定分区存储管理

- **空间的分配与去配**
  - 为了管理各分区的分配和使用情况，系统需要设置一张“主存分配表”，以说明各分区的分配情况。一个系统中分区分配表的长度固定，由主存储器中分区的个数所决定。

分区号	大小/KB	起址/KB	状态
1	12	20	JOB A
2	32	32	0
3	64	64	JOB B
4	128	128	JOB C

(a) 主存分配表

(b) 存储空间分配情况

图 5-8 四个分区的分区分配表

5.3.2 固定分区存储管理

- 当作业队列中有作业要求装入主存时，存储管理可采用“顺序分配算法”进行主存空间的分配。分配时顺序查找主存分配表，选择标志位为“0”的分区，将作业的地址空间大小与该分区长度进行比较，如果能容纳该作业，则将此分区分配给该作业，且在此分区的占用标志栏中填入该作业名。如果作业的地址空间大于此空闲分区的长度，则重复上述过程继续查找，查找空闲区长度大于等于该作业的地址空间大小且标志位为“0”的空闲分区，若有，则分配，否则该作业暂时不能装入主存储器。
- 装入后的作业在执行结束后必须归还所占用的分区。存储管理根据作业名查找主存分配表，找到该作业所占用的分区，将该分区的状态标志位重新设置为“0”，表示该分区现在是空闲区，可以装入新的作业。

5.3.2 固定分区存储管理

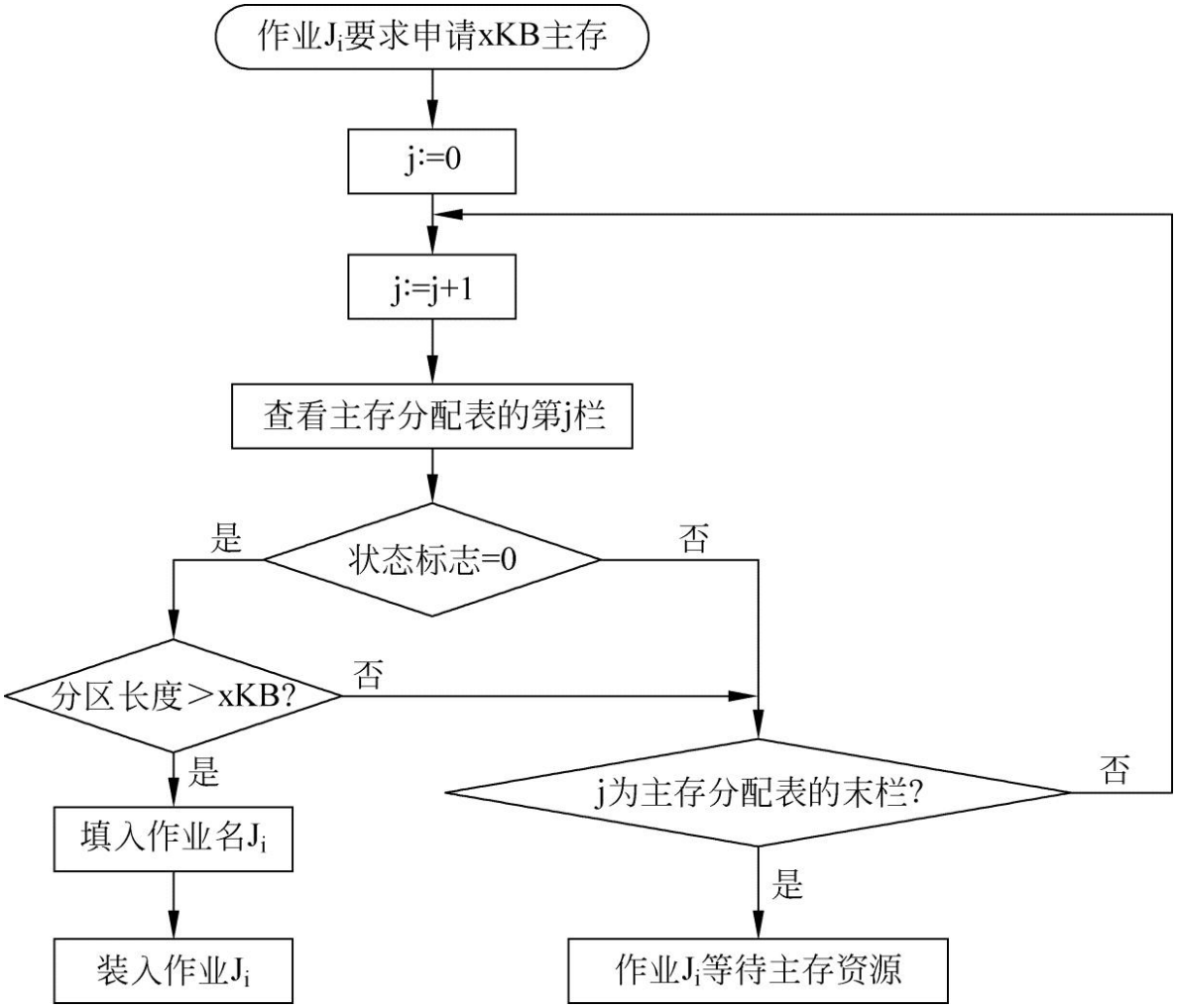
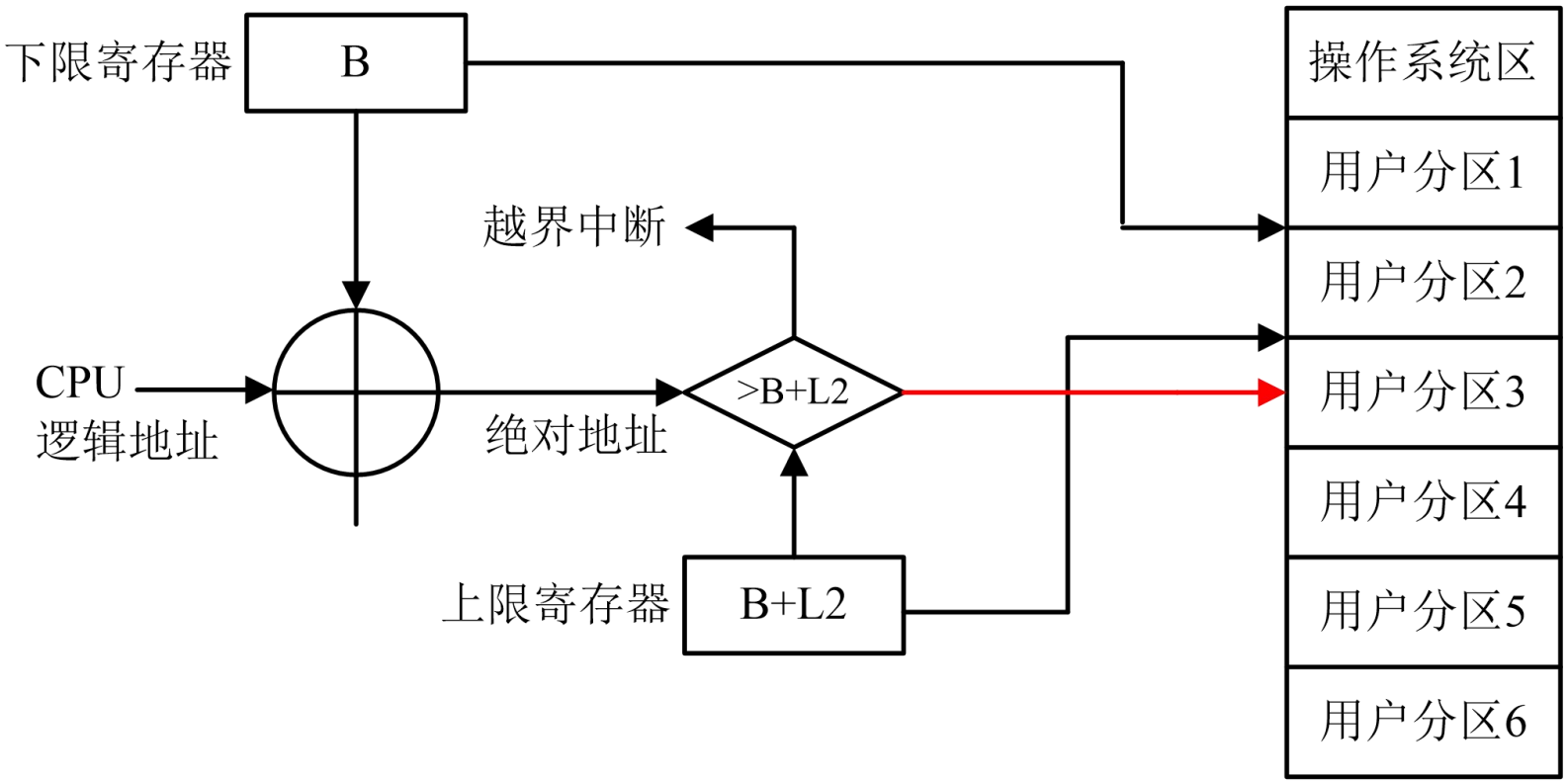


图 5-9 固定分区顺序分配算法流程

5.3.2 固定分区存储管理

- **地址转换和存储保护**
  - 固定分区存储管理，采用静态重定位装入方式装入作业。
  - 由装入程序把作业中的逻辑地址与分区的下限地址相加，得到对应的物理地址。当一个已经被装入主存的作业占有处理器运行时，进程调度程序将该作业所在分区的下限地址和上限地址分别存储在处理器的“下限寄存器”和“上限寄存器”中。处理器执行该作业指令时必须判断：下限地址≤物理地址<上限地址。
  - 如果物理地址在上、下限地址范围内，则按物理地址访问主存储器；如果条件不成立，则产生“地址越界”的中断事件，达到存储保护的目的。
  - 作业运行结束时，调度程序选择另一个可运行的作业，同时修改下限寄存器和上限寄存器的内容，以保证处理器能控制该作业的正确执行。

固定分区管理的地址映射和存储保护



5.3.2 固定分区存储管理

- **主存空间的利用率**
  - 固定分区存储管理方式总是为作业分配一个不小于作业地址空间的分区，因此在分区中产生一部分空闲区域，影响了主存空间的利用率。为了提高主存空间的利用率采用如下几种方法：

① 根据经常出现的作业的大小和频率划分分区。
② 划分分区时按分区大小顺序排列。
③ 按作业对主存空间的需求量排成多个作业队列，规定每个作业队列中的各作业只能依次装入对应的分区中；不同作业队列中的作业分别依次装入不同的分区中，不同的分区中可同时装入作业；某作业队列为“空”时，该作业队列对应的分区也不能用来装其他作业队列中的作业。

5.3.2 固定分区存储管理

- 采用多个作业队列的固定分区法可以有效地防止小作业占用大分区，从而减少了闲置的主存空间量。但是如果分区划分不合适，则会造成某个作业队列经常为空队列，使对应分区经常无作业被装入，反而使分区的利用率不高。所以采用多个作业队列的固定分区法时，可结合作业的大小和出现的频率划分分区，以达到更好的空间利用率。

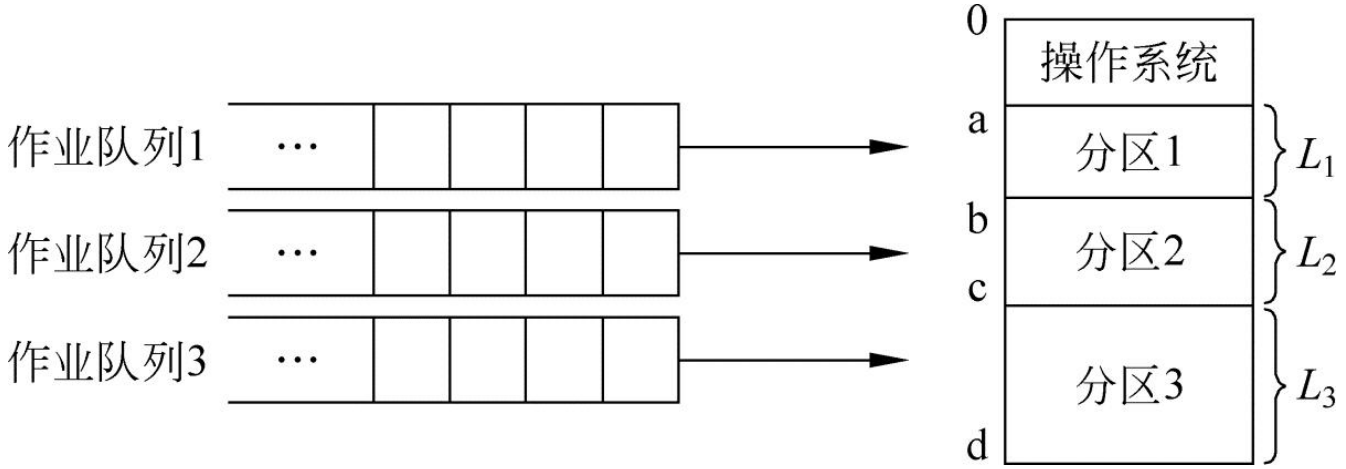


图5-10 多个作业队列的固定分区示意图

5.3.3 可变分区存储管理

- 可变分区存储管理是在作业要求装入主存时，根据作业需要的地址空间的大小和当时主存空间的实际使用情况决定是否为该作业分配一个分区。如果有足够的连续空间，则按需要分割一部分空间分区给该作业；否则令其等待主存空间。

- 在可变分区存储管理中
  - 主存储器中分区的大小是可变的；
  - 主存中分区的个数是可变的；
  - 主存中的空闲分区个数也随着作业的装入与撤离而发生变化。

5.3.3 可变分区存储管理

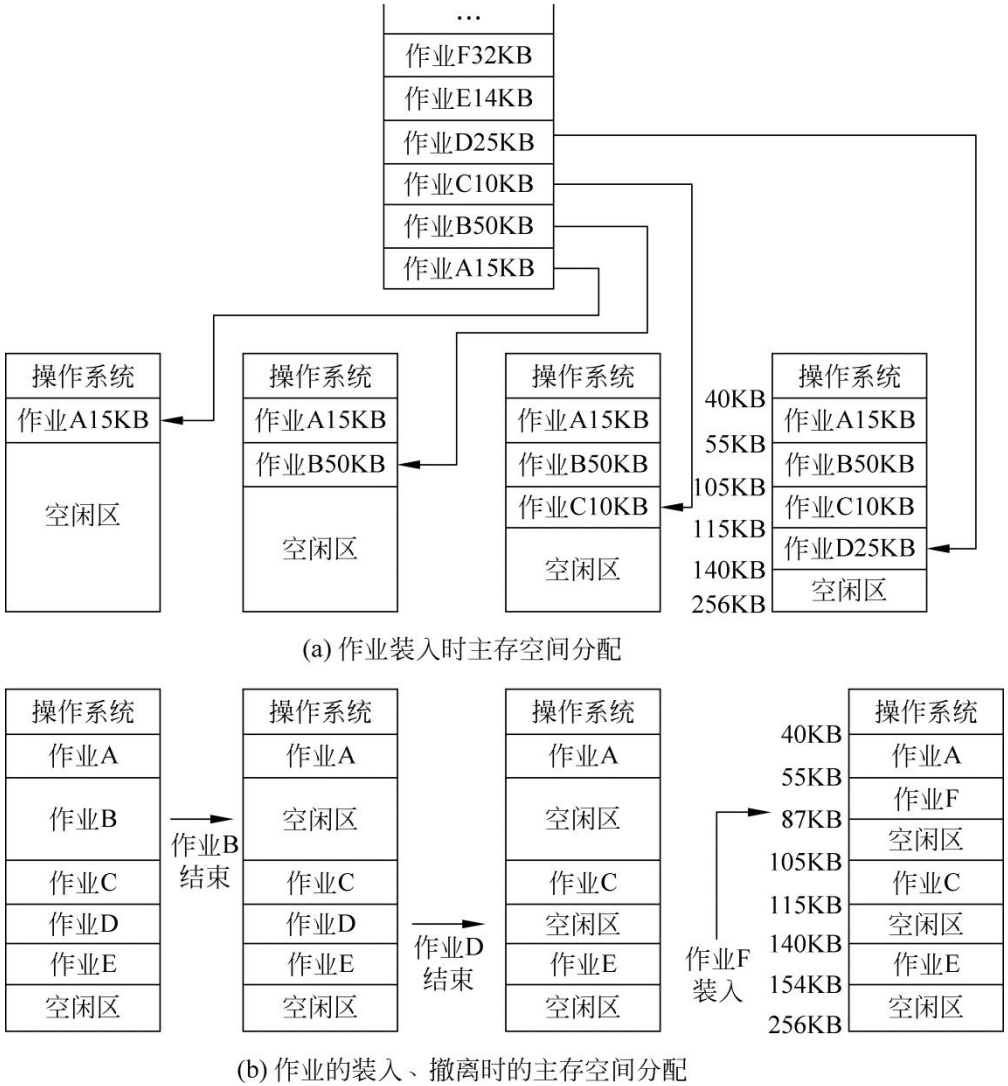


图 5-11 可变分区存储分配与回收示意图

5.3.3 可变分区存储管理

- **1.空间的分配和去配**
- 系统初始时，把主存中整个用户区看作一个大的空闲分区。当作业要求装入主存时，系统根据作业对主存空间的实际需求量进行分配。图5-11（a）给出了作业被装入时主存空间分配的情况。
- 装入主存储器中的作业执行结束后，它所占用的分区被收回，成为一个新的空闲区，可以用来装入新的作业。随着作业不断的装入和作业执行完后的撤离，主存储区被分成若干分区，其中有的分区被作业占用，有的分区空闲。图5-11（b）给出了作业被装入、执行结束后撤离的主存空间分配情况。

5.3.3 可变分区存储管理

- 可变分区数据结构
- 采用可变分区存储管理方式管理诸存储空间时，主存中已占用分区和空闲区的数目和大小都是可变的。

- 常用的数据结构有以下两种形式。
- 分区表
  - 系统设置“空闲分区表”和“已分配分区表”，用来描述空闲分区和已分配分区的情况，为系统空间分配提供依据。已分配分区表记录已装入的作业在主存空间中占用分区的始址和长度，用标志位指出占用该分区的作业名。空闲分区表记录主存中可供分配的空闲分区的始址和长度，用标志位指出该分区是未分配的空闲分区。

5.3.3 可变分区存储管理

- 系统按一定的规则组织主存分配表，当作业要求装入时，从空闲分区表查找一个长度大于作业要求的空闲分区。

始址	长度	标志
40	15K	作业A
55	32K	作业F
105K	10K	作业C
140K	14K	作业E
		空
	…	

(a) 已分配分区表

始址	长度	标志
87	18K	未分配
115	25K	未分配
154K	102K	未分配
		空
	…	

(b) 空闲分区表

图5-12 可变分区存储管理方式的主存分配表

5.3.3 可变分区存储管理

- **分区链**
  - 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，设置一些用于控制分区分配的信息，以及用于链接前一个分区的前向指针；在分区尾部则设置一后向指针，通过前、后向链接指针，可以将所有的空闲分区联结成一个双向链。为了检索的方便，在分区尾部重复设置状态位和分区大小表目。

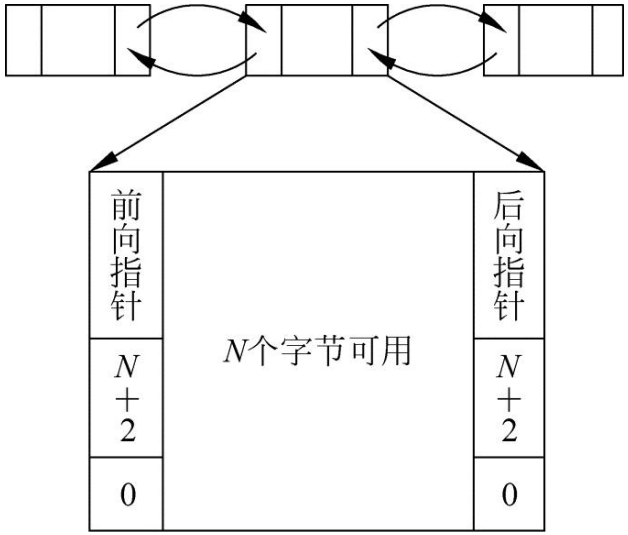


图5-13 空闲链结构示意图

5.3.3 可变分区存储管理

- **可变分区分配算法**
- 将一个作业装入主存，须按照一定的分配算法，从空闲分区表或空闲分区链中选出一个分区分配给该作业。

- 可变分区存储管理常用的空间分配算法有：
  - “最先适应”分配算法
  - “最优适应”分配算法
  - “最坏适应”分配算法

5.3.3 可变分区存储管理

- **最先适应分配算法**
- 最先适应分配算法在主存空间分配时，总是**顺序查找空闲分区表，选择第一个满足作业地址空间要求的空闲分区**进行分割，一部分分配给作业，而剩余部分仍为空闲分区。
- 最先适应分配算法实现简单，但是经过若干次作业的装入与撤离后，有可能把较大的主存空间分割成若干个小的、不连续的新空闲分区，这些空闲分区的长度可能比较小，不能满足主存再次分配的需要，从而使主存空间的利用率大大降低，这些空闲分区称为“**碎片**”。

5.3.3 可变分区存储管理

- **最优适应分配算法**
- 最优适应分配算法总是选择一个满足作业地址空间要求的最小空闲分区进行分配，这样每次分配后总能保留下较大的分区，使装入大作业时比较容易获得满足。
- 在实现过程中，空闲分区按其长度以递增顺序登记在空闲分区表中。这样，系统分配时顺序查找空闲分区表，找到的第一个满足作业空间要求的空闲分区一定是能够满足该作业要求的所有分区中的最小一个分区。
- 采用最优适应分配算法，每次分配后分割的剩余空间总是最小的，这样形成的“碎片”非常“零散”，往往难以再次分配使用，从而影响了主存空间的利用率。

5.3.3 可变分区存储管理

- **最坏适应分配算法**
- 最坏适应分配算法与最优适应分配算法相反，总是选择一个满足作业地址空间要求的最大空闲分区进行分割，按作业需要的空间大小进行分配给作业使用后，剩余部分的空间不致于太小，仍然可以供系统再次分配使用。这种分配算法对中小型作业是有利的。
- 实现最坏适应分配算法时，空闲分区按其长度以递减顺序登记在空闲分区表中。系统分配时顺序查找空闲分区表，表中的第一个登记项即对应着当前主存的最大空闲分区。同样，当作业归还主存空间时，则需要按分区的大小按递减顺序登记到空闲分区表的适当位置。

5.3.3 可变分区存储管理

图 5-14 可变分区分配算法示意图

5.3.3 可变分区存储管理

- 装入的作业执行结束后，它所占据的分区将被回收，回收后的空闲分区登记在空闲分区表中，用来装入新的作业。回收空间时应检查是否存在与回收区相邻的空闲分区，如果有，则将其合并成为一个新的空闲分区进行登记管理。
- 一个回收区可能存在上邻空闲分区，也可能存在下邻空闲分区，或既存在上邻又存在下邻空闲分区，或既无上邻又无下邻空闲分区。假定作业归还的分区始址为S，长度为L 。

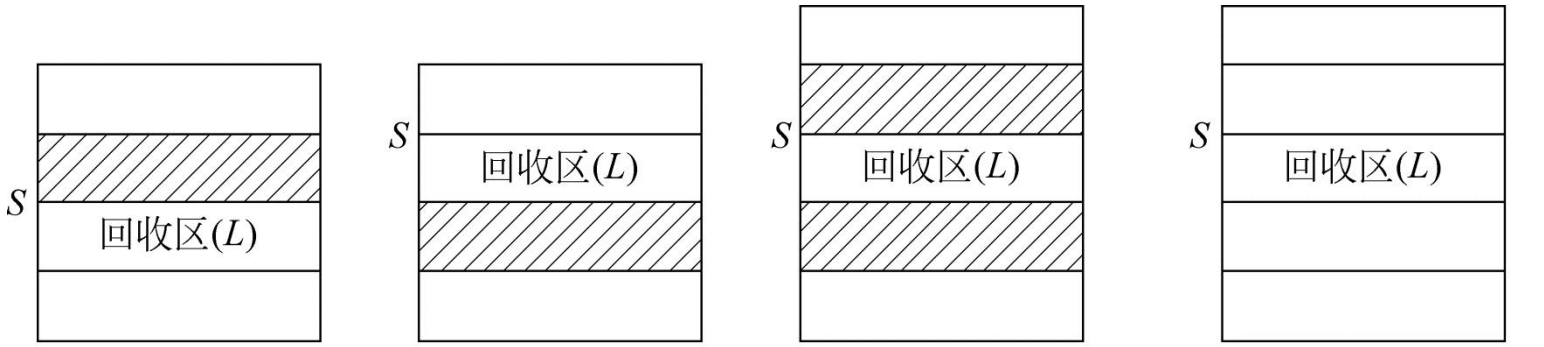


图 5-15 主存回收示意图

5.3.3 可变分区存储管理

① 回收区有下邻空闲区

如果S+L正好等于空闲区表中第j个登记栏目所示分区的始址，则表明回收区有一个下邻空闲区。此时应将回收区与下邻空闲区合并，只须修改第j栏登记项的内容：始址修改为回收区的始址，长度为二者之和。

② 回收区有上邻空闲区

如果空闲区中表第j个登记栏中的“始址+长度”正好等于S，则表明回收区有一个上邻空闲区。此时应将回收区与上邻空闲分区合并，只须修改第j栏登记项的内容：始址不变，长度为上邻空闲区长度加上回收区长度L。

5.3.3 可变分区存储管理

③ 回收区既有上邻空闲区又有下邻空闲区

如果S正好等于第j个登记栏中的“始址+长度”，并且S+L正好等于空闲区表中某个登记栏目（设为第k栏），则表明回收区既有上邻空闲区又有下邻空闲区，此时应将三个分区合并为一个新的空闲分区，只须修改第j栏登记项的内容：始址不变，长度为上邻空闲区的长度+回收区长度L+下邻空闲区长度，第k栏的标志应修改成“空”状态（空表目，既不是分配分区也不是空闲分区）。

④ 回收区既无上邻空闲区又无下邻空闲区

此时应将回收区单独建立一个新表项，查找一个标志为“空”的登记栏，把回收区的始址和长度登记入表，且把该栏目中的标志位修改成“未分配”，表示该栏指示了一个空闲分区。

5.3.3 可变分区存储管理

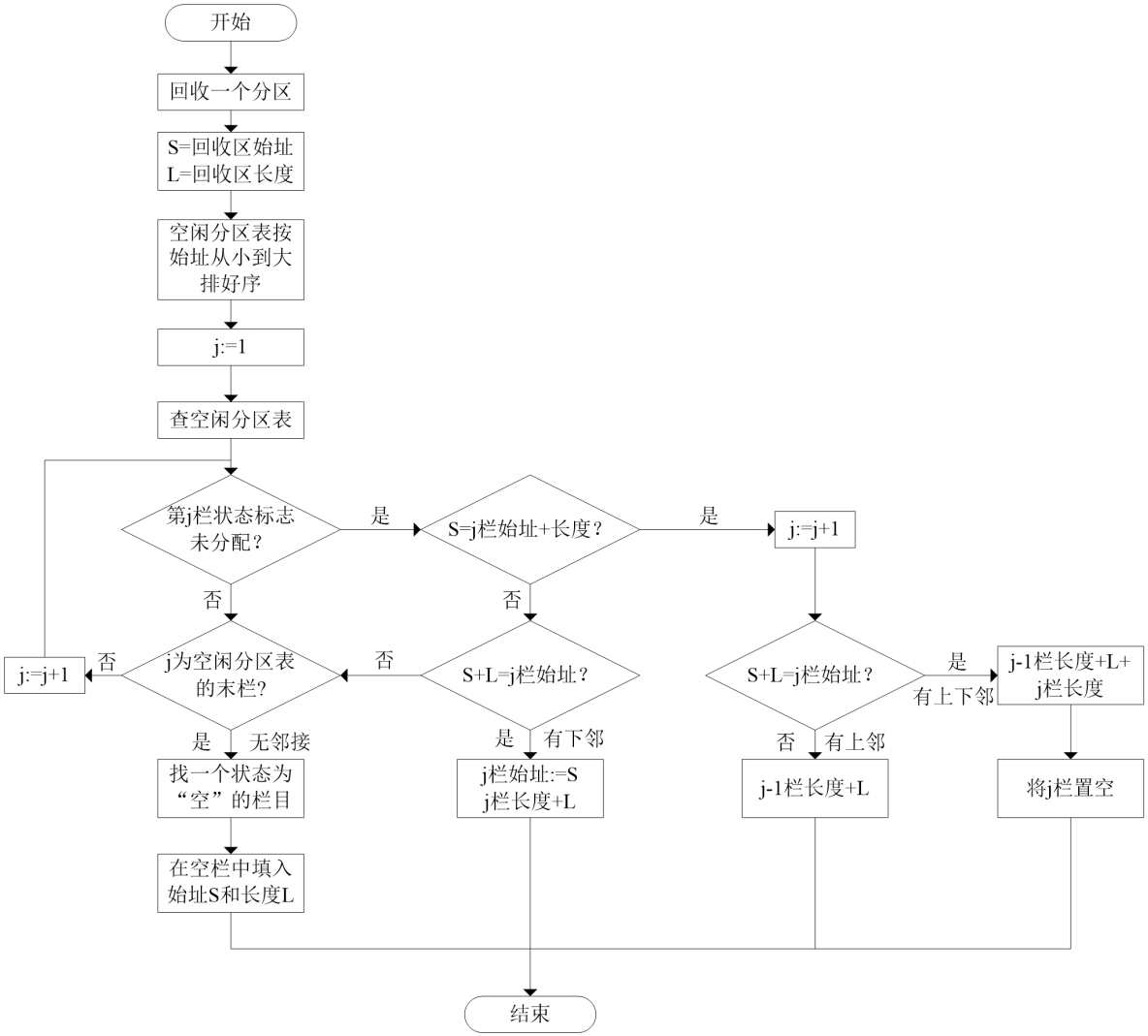


图5-16 合并下邻/上邻空闲分区的回收算法流程

5.3.3 可变分区存储管理

- **2.地址转换和存储保护**
- 采用可变分区存储管理方式，一般均采用动态重定位方式装入作业。为使地址的转换不影响到指令的执行速度，必须有硬件地址转换机构的支持。硬件地址转换机构包括两个专用控制寄存器：基址寄存器和限长寄存器，以及一些加法、比较线路等。基址寄存器用来存放作业所占分区的起始地址，限长寄存器用来存放作业所占分区长度。

5.3.3 可变分区存储管理

- 正在运行的作业所占分区的起始地址和长度被送入基址寄存器和限长寄存器中。执行过程中，处理器每执行一条指令都要将该指令的逻辑地址与限长寄存器中的值进行比较，当逻辑地址小于限长值时，则把逻辑地址与基址寄存器的值相加就可得到对应的物理地址。当逻辑地址大于限长寄存器中的限长值时，表示欲访问的地址超出了所分配的分区范围，这时形成一个“地址越界”的程序性中断事件，达到存储保护的目的。

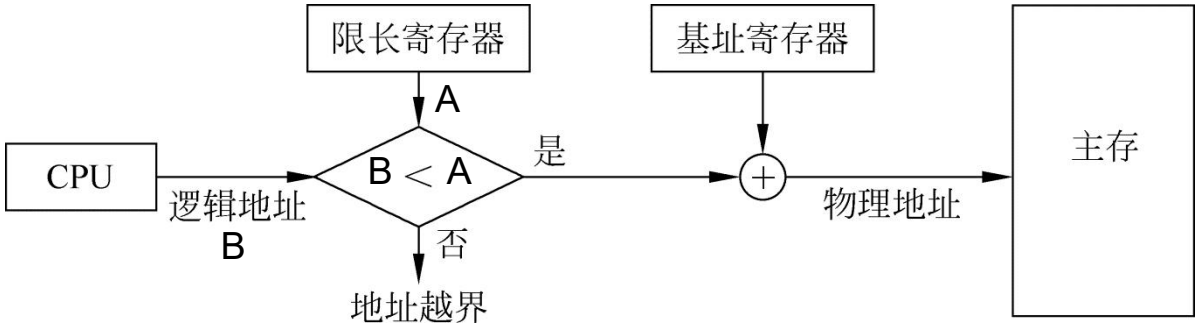


图5-17 可变分区管理地址转换示意图

5.3.3 可变分区存储管理

- **3.移动技术**
- **采用可变分区存储管理主存时，可采用移动技术使分散的空闲分区集中起来以容纳新的作业，如图5-18所示。**

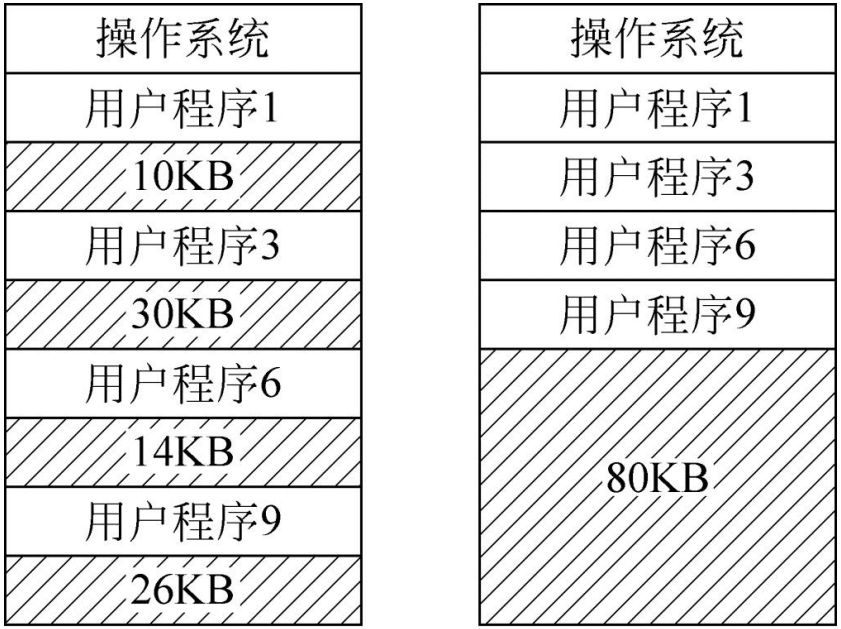


图5-18 移动技术示意图

5.3.3 可变分区存储管理

- 移动技术为作业执行过程中扩充主存空间提供方便，一道作业在执行中要求增加主存量时，只要适当移动邻近的作业就可增加它所占的分区长度。
- 移动可以集中分散的空闲分区，提高主存空间的利用率，移动也为作业动态扩充主存空间提供了方便。

- 采用移动技术时必须注意：
  - 移动会增加系统开销。
  - 移动是有条件的。

5.3.3 可变分区存储管理

- 采用移动技术时应该尽量减少移动的作业数和信息量，以降低系统的开销、提高系统的效率。图5-19给出了两种作业装入主存的方式。

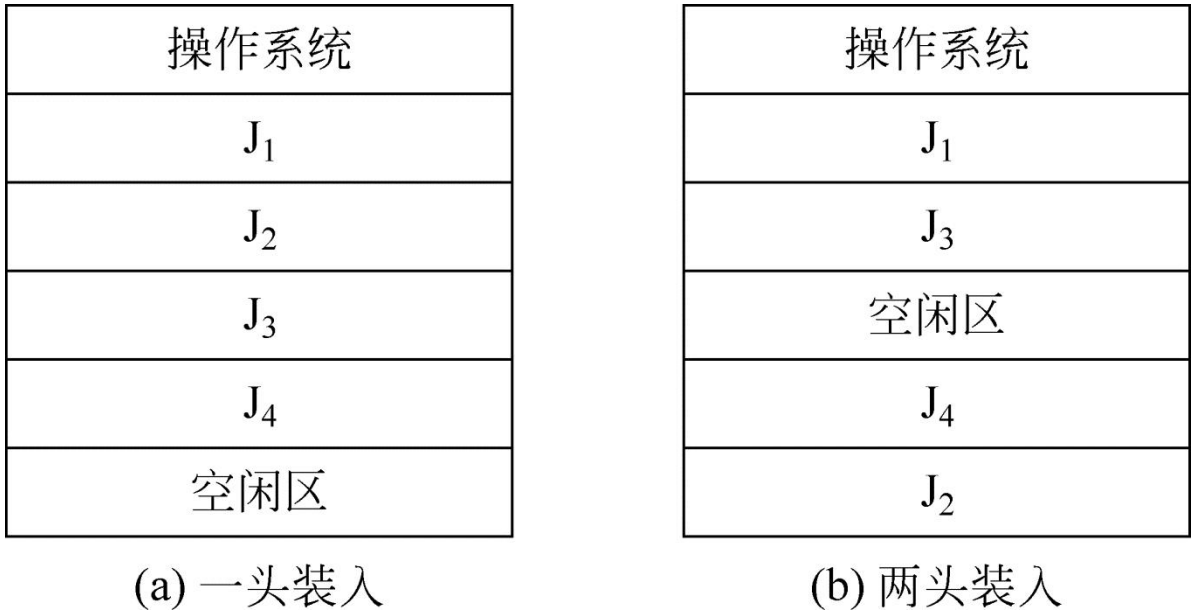


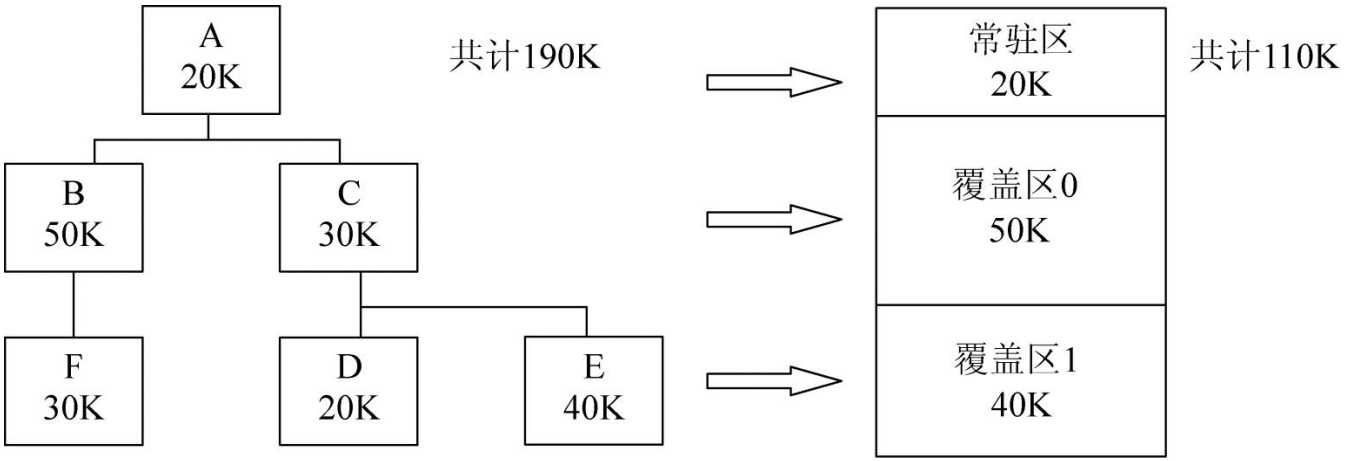
图5-19 作业装入主存的方式

5.3.4 覆盖与交换技术

- 覆盖技术和交换技术是在多道环境下用来扩充主存的两种方法。覆盖技术主要用于早期的操作系统中，而交换技术在现代操作系统中仍有较强的生命力。
- 单一连续存储管理和分区存储管理对作业的大小都有严格的限制，当作业要求运行时，系统将作业的全部信息一次装入主存，并一直驻留在主存直至运行结束。当作业的大小大于主存可用空间时，该作业就无法运行，这就限制了计算机系统上开发较大程序的可能。
- 覆盖和交换技术是解决大作业与小主存矛盾的两种存储管理技术，他们实质上是对主存进行了逻辑扩充。

5.3.4 覆盖与交换技术

- **覆盖技术的基本思想是一个程序不需要把所有的指令和数据都装入主存**。可以把程序划为若干功能上相对独立的程序段，让那些不会同时执行的程序段共享一块主存。
- 将程序的必要部分（常用功能）的代码和数据常驻主存；可选部分（不常用功能）在其他程序模块中实现，平时存放在外存中（覆盖文件），在需要用到时才装入到主存；不存在调用关系的模块不必同时装入到主存，从而可以相互覆盖(即不同时用的模块可共用一个分区)。把可以相互覆盖的程序段叫做覆盖，可共享的主存区叫做覆盖区。

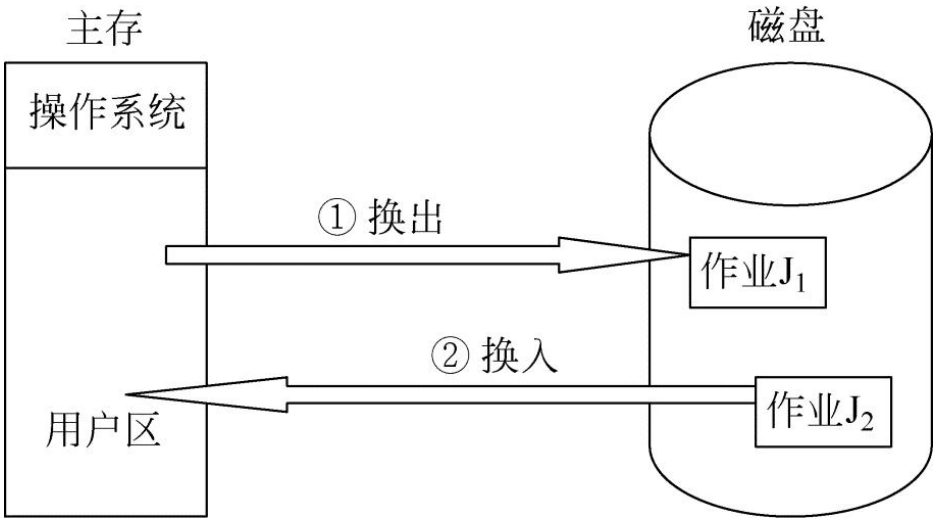


5.3.4 覆盖与交换技术

- 为了实现覆盖管理，系统必须提供相应的覆盖管理控制程序。当作业装入运行时，由系统根据用户提供的覆盖结构进行覆盖处理。当程序中引用当前尚未装入覆盖区的覆盖中的例程时，则调用覆盖管理控制程序，请求将所需的覆盖装入覆盖区中，系统响应请求，并自动将所需覆盖装入主存覆盖区中。
- 覆盖技术打破了必须将一个作业的全部信息装入主存后才能运行的限制。在一定程度上解决了小主存运行大作业的矛盾。
- 但是，采用覆盖技术编程时必须划分程序模块和确定程序模块之间的覆盖关系，增加了编程复杂度；从外存装入覆盖文件，是以时间延长来换取空间节省。

5.3.4 覆盖与交换技术

- **交换**的基本思想是把主存中暂时不能运行的进程或暂时不使用的程序和数据换出到外存，以腾出足够的主存空间，把已具备运行条件的进程或进程所需要的程序和数据换入主存。
- 交换技术并不要求程序员做特殊的工作。交换完全由操作系统进行，整个过程对进程是透明的。交换的对象可以是整个进程，此时成为“整体交换”或“进程交换”。



5.3.4 覆盖与交换技术

- 实现交换技术需要后援存储器，通常是硬盘，它必须具备两个显著的特征：
  - 数据传输快
  - 容量足够大

- 交换技术主要是在进程或作业之间进行，而覆盖则主要是在同一个作业或进程内进行。

5.4 页式存储管理

- 连续分配方式要求作业一次性、连续装入在主存空间，对空间的要求较高，而且可能形成很多“碎片”，虽然可以通过移动技术将零散的空闲分区汇集成可用的大块空间，但需要增加系统的额外开销。如果采用不连续存储的方式，把逻辑地址连续的作业分散存放到几个不连续的主存区域中，并能保证作业的正确执行。这样既可充分利用主存空间，减少主存的碎片，又可避免移动所带来的额外开销。

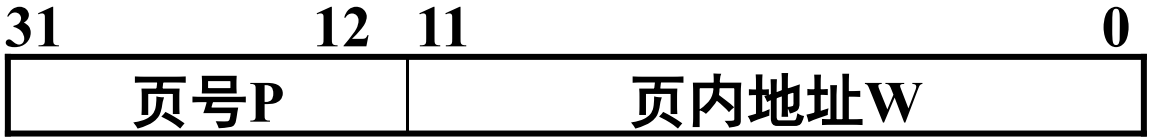
5.4 页式存储管理

- 页式存储管理允许程序的存储空间不一定连续，这样可以把一个进程分散地放在各个空间的内存块中，它既不需要移动内存中的原有的信息，也解决了外部碎片的问题，从而提高了内存的利用率。

5.4.1 基本原理

- 采用页式存储管理，系统需要将进程的逻辑地址空间划分成若干大小相等的页(page)，并对各页加以编号。相应地，也要把内存的存储空间划分为若干个与页相同大小的块(block)，同样也对各块加以编号。分页后的地址结构将由两部分构成，即页号P和页内偏移地址W。

5.4.1 基本原理



页式存储管理地址结构示意图

- 给定的逻辑地址是*A*，页面的大小为*L*，则页号*P*和页内地址*W*可按下式求得：

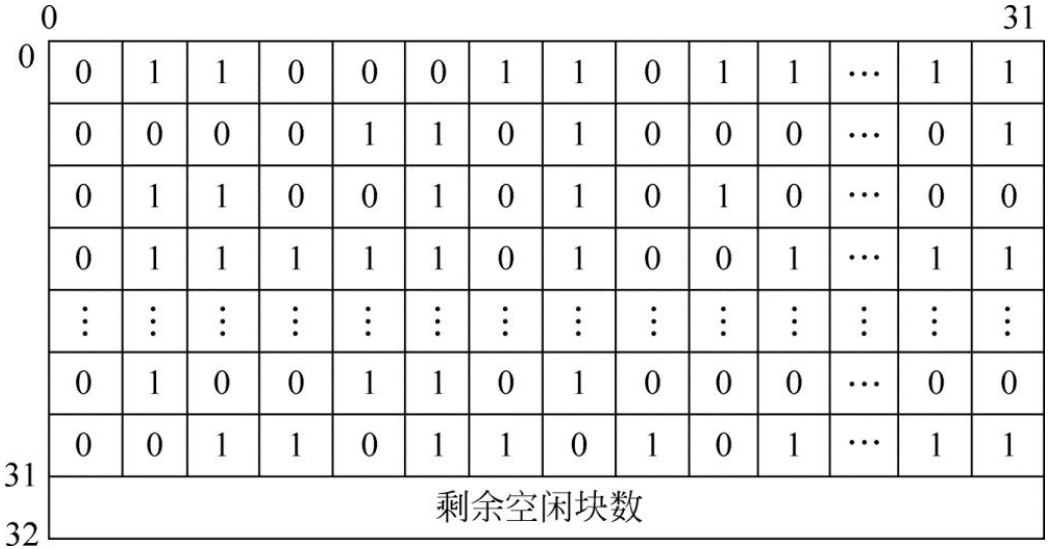
$$P = \mathrm{INT}(A/L)\ ,\ W = A\ \mathrm{MOD}\ L$$

式中，INT是向下整除的函数，MOD是取余函数。

5.4.2 存储空间的分配与去配

- 页式存储管理把主存空间划分成若干块，以块为单位进行主存空间的分配。由于块的大小是固定的，系统可以采用一张主存分配表来记录已分配的块、尚未分配的块以及当前剩余的空闲块总数。最简单的办法可用一张“位示图”来记录主存的分配情况。
- 例如一个划分成1024块的主存，则可用字长为32位的32个字的位示图来构成一张主存分配表。

5.4.2 存储空间的分配与去配



- 进行主存分配时，首先查看空闲块总数是否能够满足作业要求，若不能满足，则不进行分配；若能满足，则从位示图中找出为“0”的位，并且将其占用标志置为“1”，并从空闲块总数中减去本次占用的块数，按找到的位计算出对应的块号，建立该作业的页表，并把作业装入对应的物理块中。

5.4.2 存储空间的分配与去配

- 由于每一块的大小相等，在位示图中查找到一个为“0”的位后，根据它所在的字号、位号，按如下公式可计算出对应的块号：
  - 块号＝字号×字长＋位号
- 当一个作业执行结束时，则应该收回作业所占的主存块。根据归还的块号计算出该块在位示图中对应的位置，将占用标志修改为“0”，同时把归还块数加入到空闲块总数中。假定归还块的块号为i，则在位示图中对应的位置为：
  - 字号＝[i / 字长]，　位号＝i mod 字长
  - 其中[ ]表示对i除以字长后取其整数，而mod表示对i除以字长后取其余数部分。

5.4.3 页表与地址转换

- 在页式存储管理系统中，允许将作业的每一页离散地存储在主存的物理块中，为此，系统为每个作业建立了一张**页面映像表**，简称页表。
- 页表实现了从页号到主存块号的地址映像。作业中的所有页（0~n）依次地在页表中记录了相应页在主存中对应的物理块号。页表的长度由进程或作业拥有的页面数决定。

5.4.3 页表与地址转换

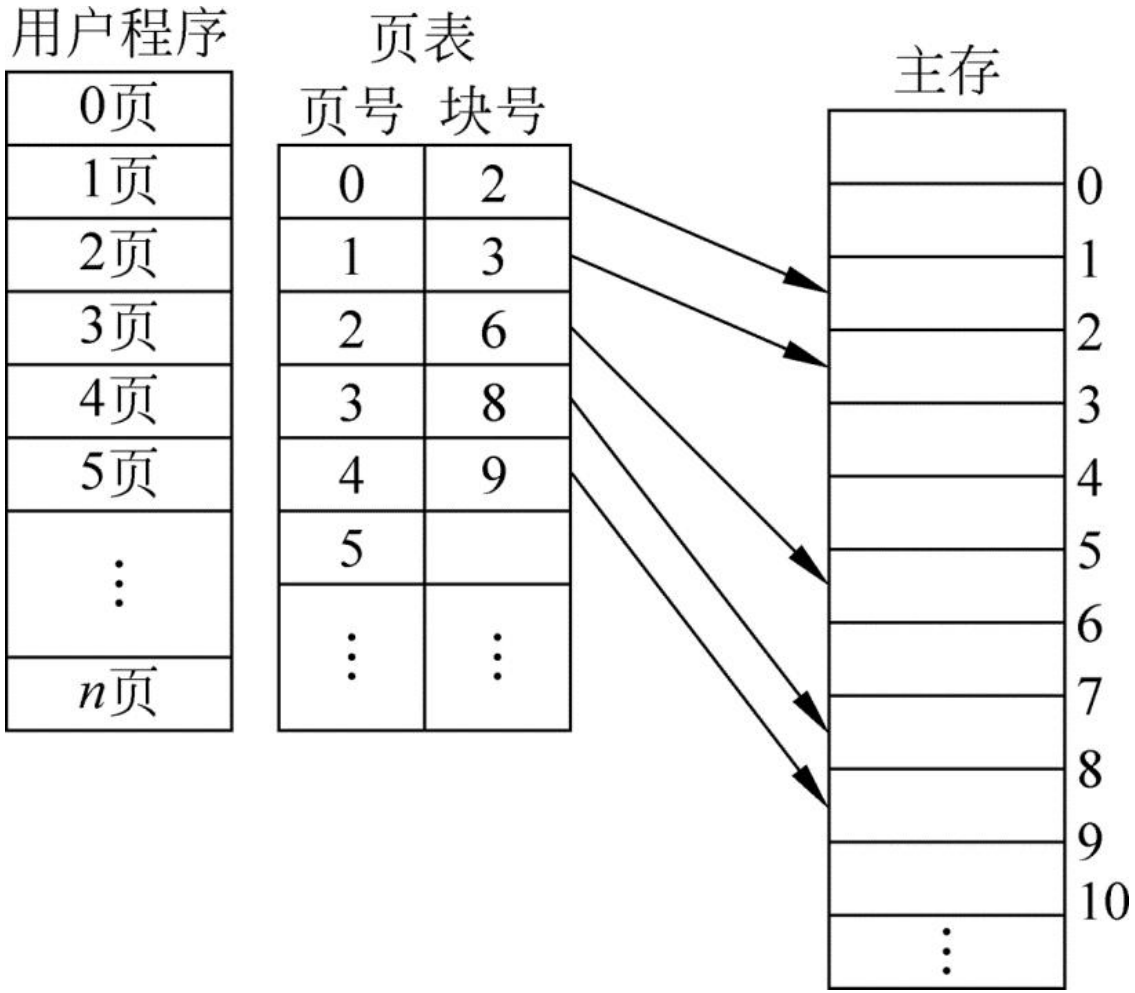


图5-23 页表示意图

5.4.3 页表与地址转换

- 页式存储管理采用动态重定位方式装入作业，作业执行时通过硬件的地址转换机构实现从用户空间中的逻辑地址到主存空间中物理地址的转换工作。由于页内地址和物理块内的地址是一一对应的。因此，地址变换机构的任务，实际上是将逻辑地址中的页号，转换成为主存中的物理块号。
- 页表是硬件进行地址转换的依据。

5.4.3 页表与地址转换

- 调度程序在选择作业后，将选中作业的页表始址送入硬件设置的页表控制寄存器中。地址转换时，只要从页表寄存器中就可找到相应的页表。当作业执行时，分页地址变换机构会自动将逻辑地址分为页号和页内地址两部分，以页号位索引检索页表，如果页表中无此页号，则产生一个“地址错”的程序性中断事件；如果页表中有此页号，则可得到对应的主存块号，再按逻辑地址中的页内地址计算出欲访问的主存单元的物理地址。因为块的大小相等，所以
  - 物理地址=块号×块长＋页内地址

5.4.3 页表与地址转换

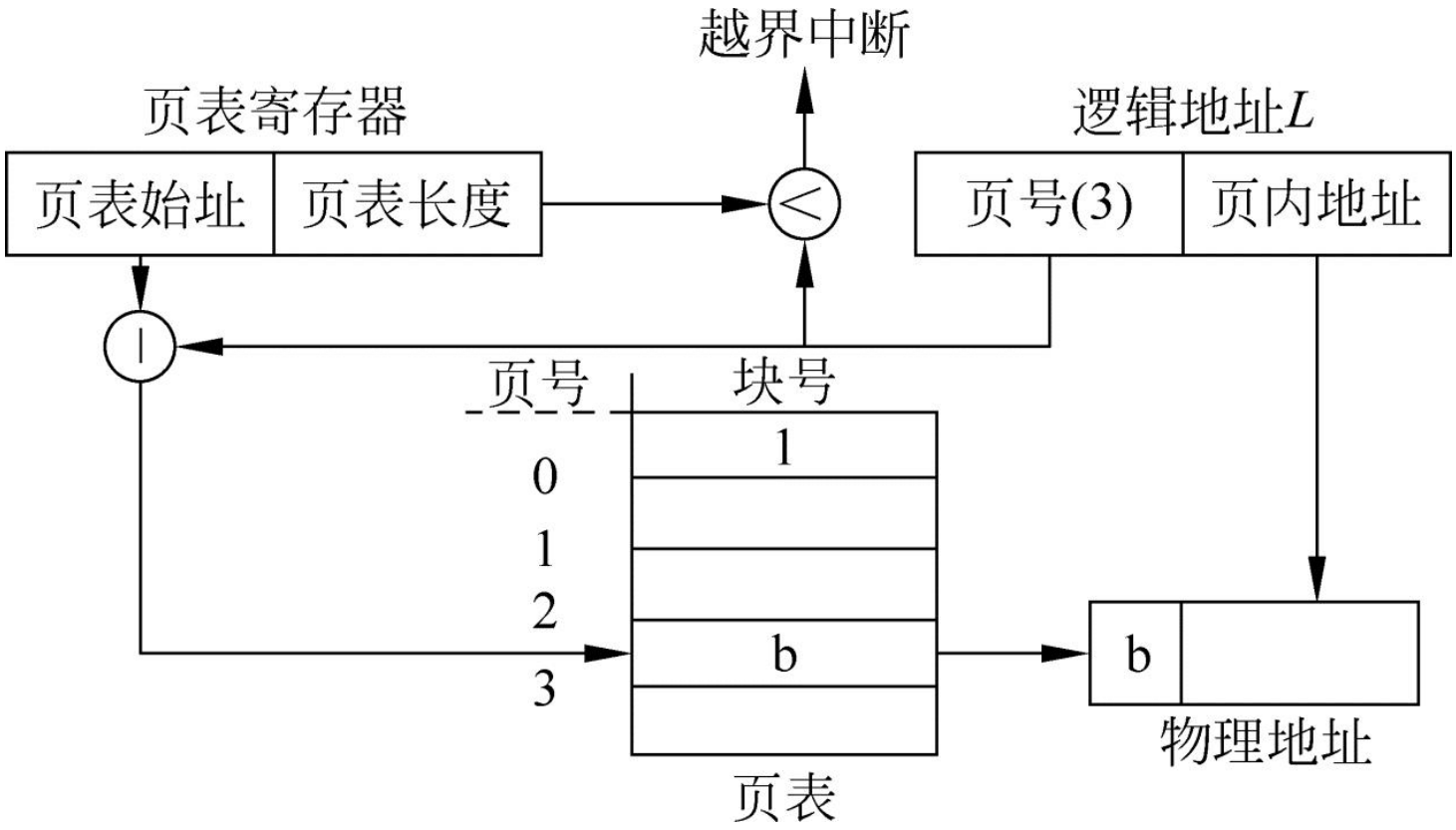


图5-24　分页系统的地址变换机构示意图

5.4.4 快表

- 由于页表是存放在主存储器中的，取一个数据或指令至少需要访问主存两次以上。
- 第一次是访问主存中的页表，查找到指定页面所对应的物理块号，将块号与页内地址拼接，计算出数据或指令的物理地址。
- 第二次根据第一次得到的物理地址进行数据的存取操作。
- 为了提高存取速度，通常在地址变换机构中增设一个具有并行查找能力的小容量高速缓冲寄存器，又称“相联寄存器”。利用高速缓冲寄存器存放页表的一部分，把存放在高速缓冲寄存器中的这部分页表称“快表”。快表中登记了当前执行程序中最常用的页号与主存中块号的对应关系，图5-25给出具有快表的地址变换机构示例。

5.4.3 页表与地址转换

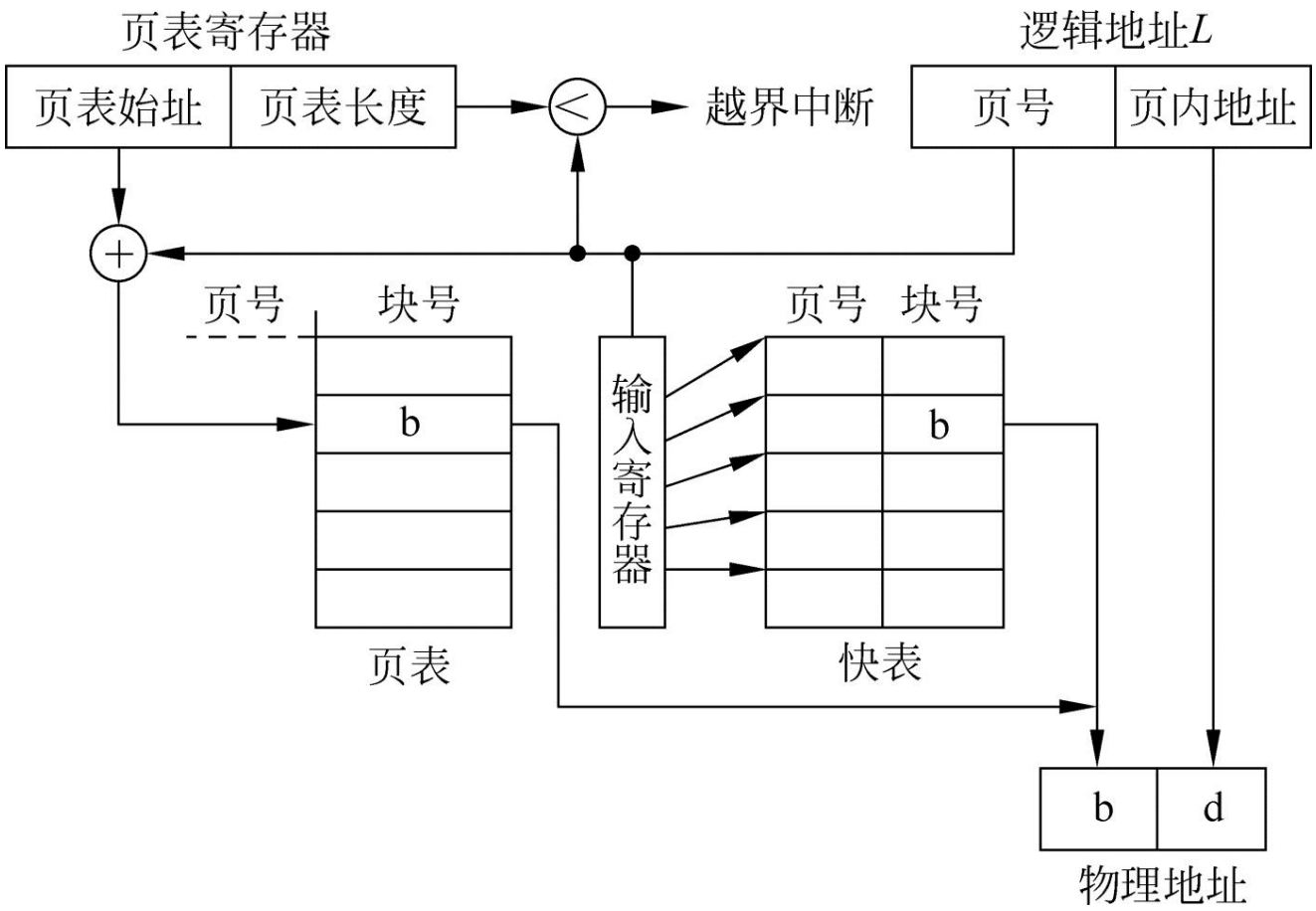


图5-25　具有快表的地址变换机构示意图

5.4.5 页的共享与保护

- 采用页式存储管理能方便地实现程序和数据的共享。
- 在多道程序系统中，编译程序、编辑程序、解释程序、公共子程序、公共数据等都是可共享的，这些共享的信息在主存储器中只需要保留一个副本，大大提高了主存空间的利用率。
- 在实现共享时，必须区分**数据的共享**和**程序的共享**。
- 实现数据共享时，可允许不同的作业对共享的数据页采用不同页号，只需要将各自的有关表目指向共享的数据信息块即可。
- 实现程序共享时，由于页式存储结构要求逻辑地址空间是连续的，所以在程序运行前它们的页号是确定的。

5.4.5 页的共享与保护

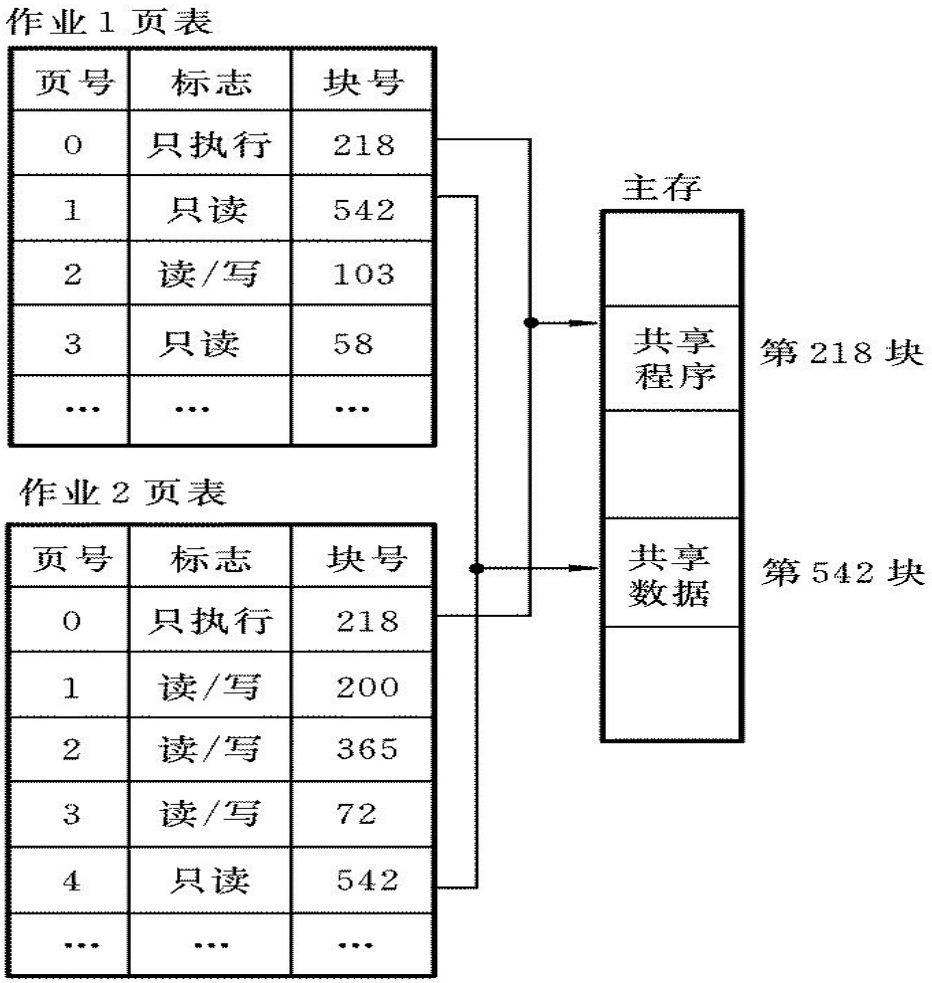


图5-26　页的共享示意图

5.5 段式存储管理

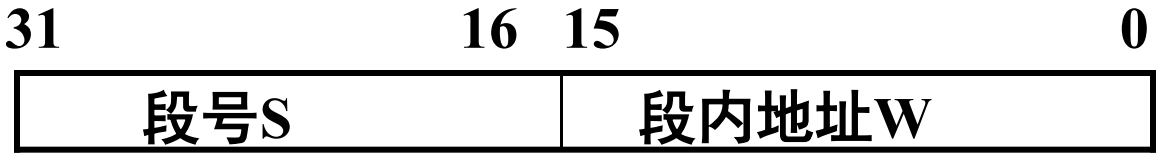
- 用户编制的程序是由若干段组成的：一个程序可以由一个主程序、若干子程序、符号表、栈以及数据等若干段组成。每一段都有独立、完整的逻辑意义，每一段程序都可独立编制，且每一段的长度可以不同。
- 段式存储管理支持用户的分段观点，具有逻辑上的清晰和完整性，它以段为单位进行存储空间的管理。

5.5 段式存储管理

- 将程序的地址空间按照程序自身的逻辑关系划分为若干段，如每个函数一个段，各段占用一片内存空间，这样程序在内存的存放情况就与程序的逻辑结构对应起来。
- 把程序和数据分隔为逻辑上独立的地址空间，有助于存储共享和保护。
- 为了满足用户在编程和使用等方面的需求，引入了段式存储管理技术。

5.5.1 基本原理

- 每个作业由若干个相对独立的段组成，每个段都有一个段名，为了实现简单，通常可用段号代替段名，段号从“0”开始，每一段的逻辑地址都从“0”开始编址，段内地址是连续的，而段与段之间的地址是不连续的。
- 其逻辑地址由段号和段内地址两部分所组成：



5.5.2 空间的分配与去配

- 段式存储管理是在可变分区存储管理方式的基础上发展而来的。
- 为了能从主存中正确找出每个段所在的分区位置，系统为每个进程建立一张段映射表，简称“段表”。每个段在表中占有一个表项，记录该段在主存储器中的起始地址和长度，段表实现了从逻辑段到主存空间之间的映射。

5.5.2 空间的分配与去配

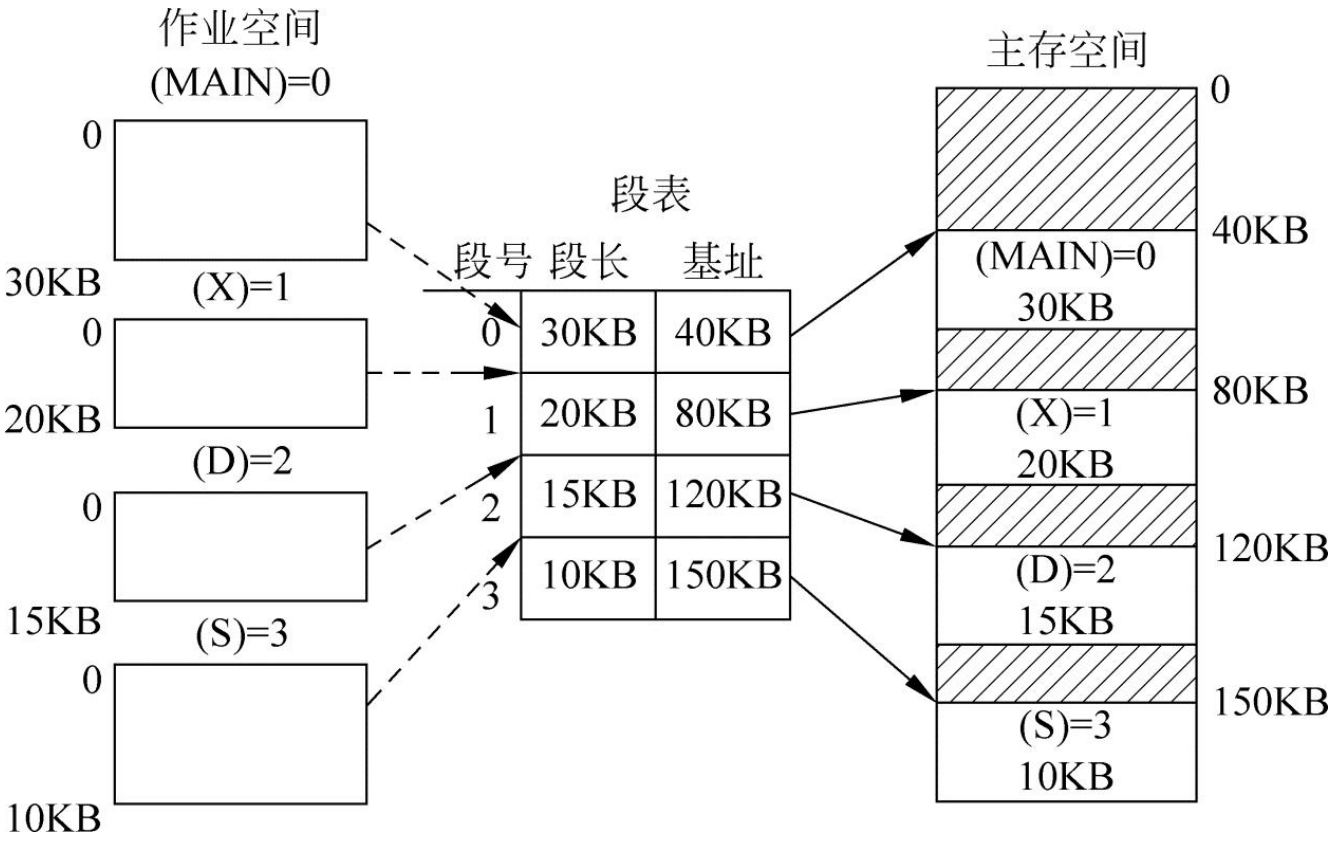


图5-27　段的装入示意图

5.5.2 空间的分配与去配

- 如果在装入某段信息时找不到满足该段地址空间大小的空闲区，则可采用移动技术合并分散的空闲区，以利于大作业的装入。
- 当采用分段式存储管理的作业执行结束后，它所占据的主存空间将被回收，回收后的主存空间登记在空闲分区表中，可以用来装入新的作业。系统在回收空间时同样需要检查是否存在与回收区相邻的空闲分区，如果有，则将其合并成为一个新的空闲分区进行登记管理。
- 段表存放在主存储器中，在访问一个数据或指令时至少需要访问主存两次以上。为了提高对段表的存取速度，通常增设一个相联寄存器，利用高速缓冲寄存器保存最近常用的段表项。

5.5.3 地址转换与存储保护

- 段式存储管理采用**动态重定位**方式装入作业，作业执行时通过硬件的地址转换机构实现从逻辑地址到物理地址的转换工作，段表的表目起到了基址寄存器和限长寄存器的作用，是硬件进行地址转换的依据。

5.5.3 地址转换与存储保护

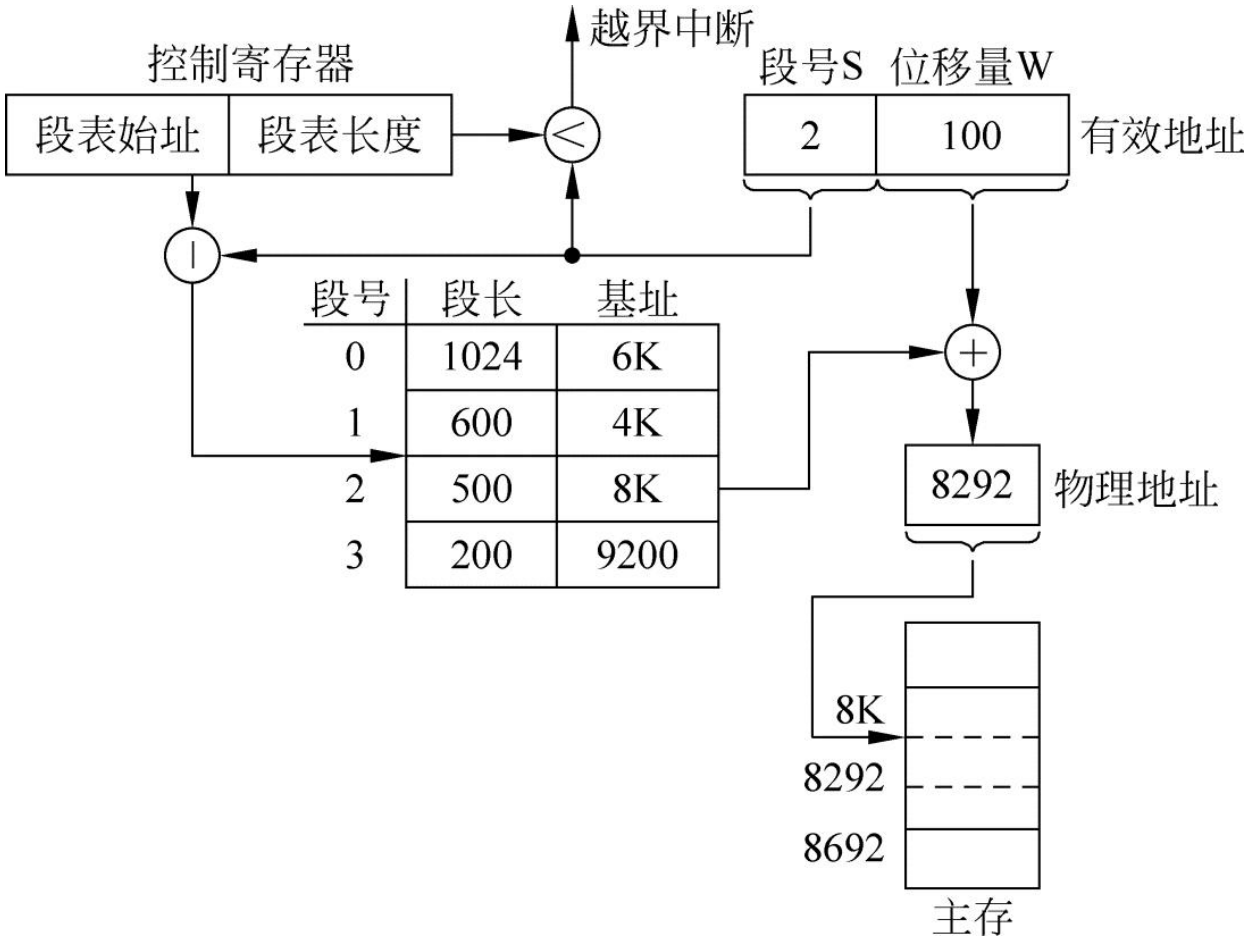


图5-28 分段式存储管理地址变换示意图

5.5.4 段的共享

- 段按逻辑意义来划分的，可以按段名进行访问，因此分段式存储管理系统的一个突出优点，是可以方便地实现段的共享，即允许若干个进程共享一个或多个段。
- 为实现某段代码的共享，各个进程对共享段使用相同的段名，在各自的段表中填入共享段的起始地址，并置以适当的读写控制权，即可做到共享一个逻辑上完整的主存段信息。

5.5.4 段的共享

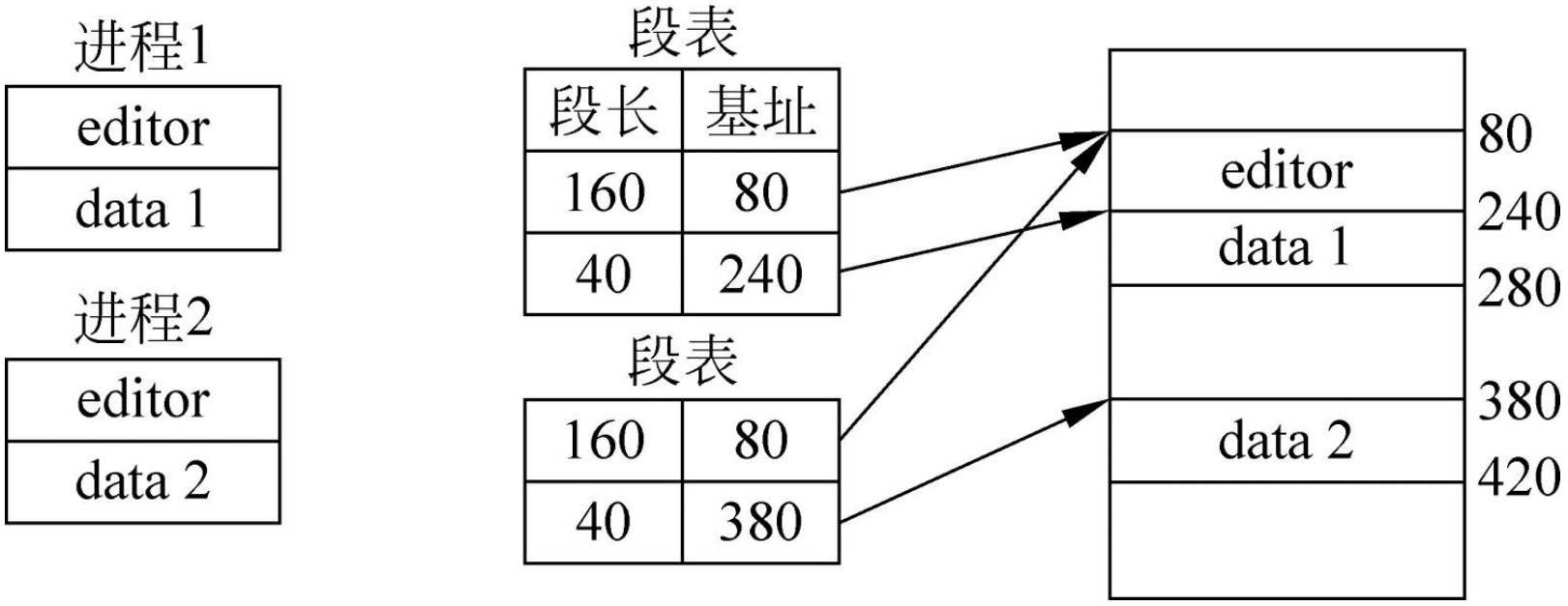


图5-29 段的共享示意图

5.5.4 段的共享

- 例如一个多用户系统，可同时接纳40个用户，他们都需要执行一个文本编辑程序(Text Editor)。如果文本编辑程序有160KB的代码和40KB的数据区，则总共需要8MB的主存空间来支持40个用户的访问。如果160KB的代码是可重入的，则该代码只需要在主存中保留一份文本编辑程序的副本，此时所需要的主存空间仅为1760KB(40×40+160），只需要在每个进程的段表中为文本编辑程序设置一个段表项。

5.5.4 段的共享

- 在多道程序设计系统中，由于进程的并发执行，一个程序段为多个进程共享时，有可能出现多次同时重复执行该段程序的情况，而该共享程序段的指令和数据在执行过程中是不能被修改的。此外，共享段也有可能被置换出主存，显然，一个正在被某个进程使用或即将被某个进程使用的共享段是不应该置换出主存的，因此可以在段表中设置共享位来判别该段是否正在被某个进程调用。

5.5.5 分页和分段存储管理的主要区别

- 分页和分段系统都采用离散分配主存方式，都需要通过地址映射机构来实现地址变换，有许多相似之处。但两者又是完全不同的。具体表现如下。

① 页是信息的物理单位，是系统管理的需要而不是用户的需要；而段则是信息的逻辑单位，它含有一组意义相对完整的信息，分段是为了更好地满足用户的需要。

② 页的大小固定且由系统决定，因而一个系统只能有一种大小的页面；而段的长度却不固定，由用户所编写的程序决定，通常由编译程序对源程序进行编译时根据信息的性质来划分。

③ 分页式作业的地址空间是一维的，页间的逻辑地址是连续的；而分段式作业的地址空间则是二维的，段间的逻辑地址是不连续的。

5.6 段页式存储管理

- 段式存储管理支持了用户的观点，但每段必须占据主存储器的连续区域，有可能需要采用移动技术汇集主存空间，为此，兼用分段和分页的方法，构成可分页的段式存储管理，通常被称为是“段页式存储管理”。段页式存储管理兼顾了段式在逻辑上的清晰和页式在管理上方便的优点。
- 用户对作业采用分段组织，每段独立编程，在主存空间分配时，再把每段分成若干个页面，这样每段不必占据连续的主存空间，可把它按页存放在不连续的主存块中。
- 段页式存储管理的逻辑地址格式如下：

段号（S）	页号（P）	页内地址（W）

5.6 段页式存储管理

- 段页式存储管理为每一个装入主存的作业建立一张段表，且对每一段建立一张页表。段表的长度由作业分段的个数决定，段表中的每一个表目指出本段页表的始址和长度。页表的长度则由对应段所划分的页面数所决定，页表中的每一个表目指出本段的逻辑页号与主存物理块号之间的对应关系。

5.6 段页式存储管理

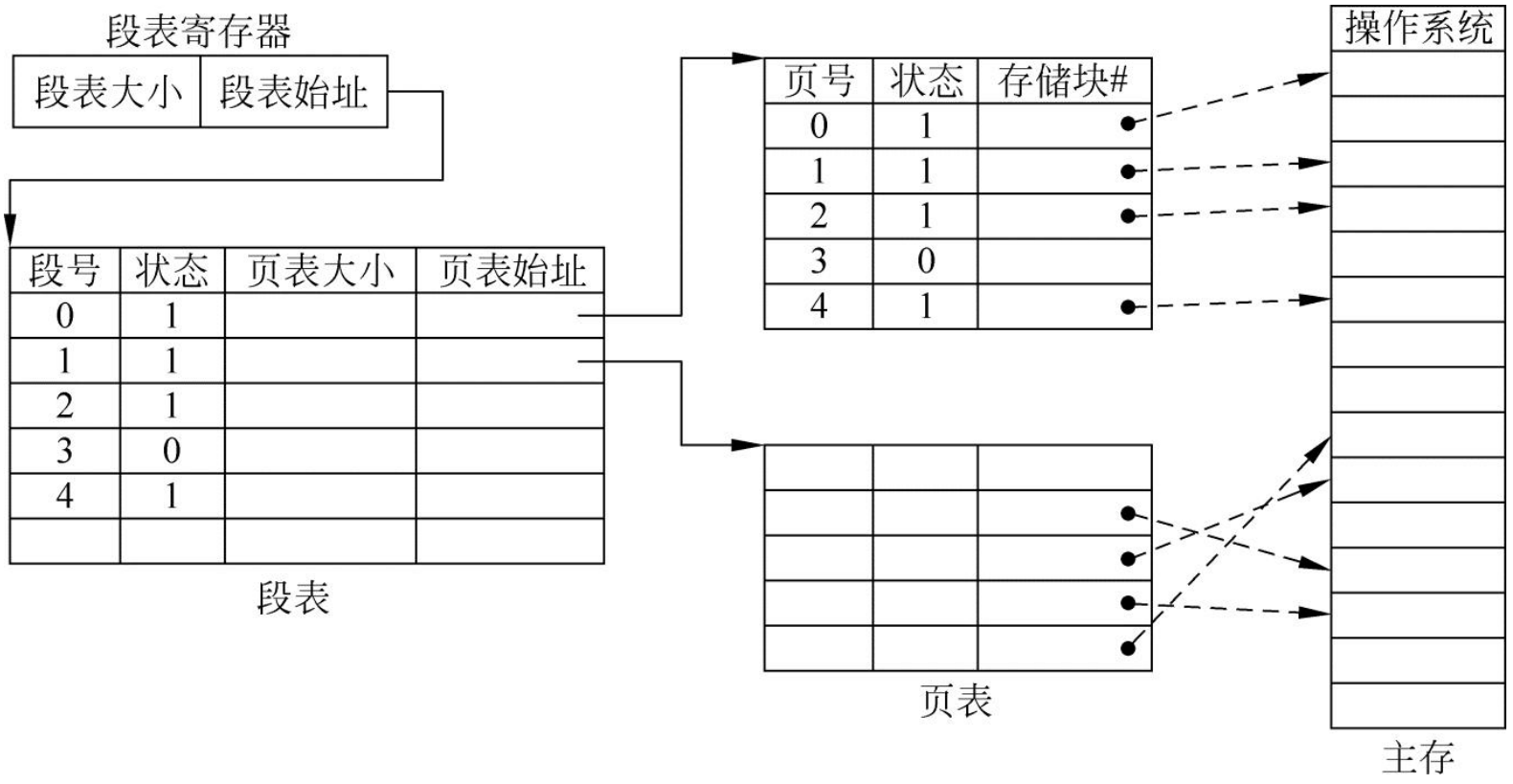


图5-30 段页式存储管理中段表、页表之间的关系

5.6 段页式存储管理

- 执行指令时，地址机构根据逻辑地址中的段号查找段表，得到该段的页表始址，然后根据逻辑地址中的页号查找该页表，得到对应的主存块号，由主存块号与逻辑地址中的页内地址形成可访问的物理地址。如果逻辑地址中的段号超出了段表中的最大段号或者页号超出了该段页表中的最大页号，都将形成“地址越界”的程序性中断事件。

5.6 段页式存储管理

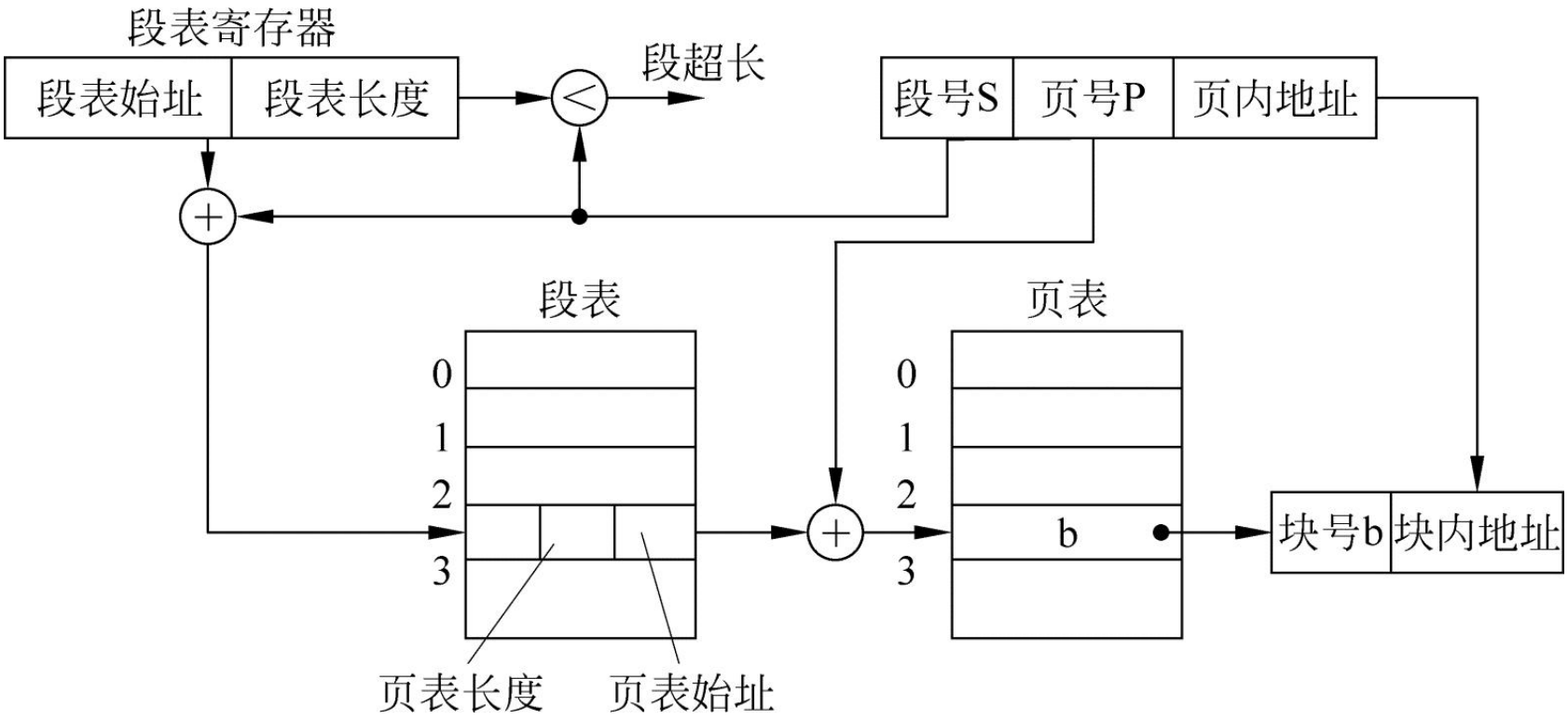


图 5-31 段页式存储管理的地址转换机构示意图

5.6 段页式存储管理

- 由逻辑地址到物理地址的变换过程中，需要三次访问主存：

① 第一次是访问主存中的段表，获得该段对应页表的始址；

② 第二次是访问页表，获得指令或数据的物理地址；

③ 最后再按物理地址存取信息。

5.7 虚拟存储管理方式

- 前面所介绍的各种存储管理方式具有一个共同的特点，即作业必须一次性全部装入主存空间后才能运行，直至作业运行结束后才释放所占有的全部主存资源。这样就会出现以下情况：

① 当主存空间不能满足作业地址空间要求时，作业就不能装入主存，无法运行。

② 当有大量作业要求运行时，由于主存容量有限，只能将少数作业装入主存运行，而其他作业留在辅存上等待。

5.7 虚拟存储管理方式

- 早在1968年，彼得·詹姆斯·丹宁（Peter James Denning）就曾指出：程序执行时呈现出局部性特征，即在一较短的时间内，程序的执行仅局限于某个部分；而它所访问的存储空间也局限在某个区域中。
- 局限性表现在时间局限性和空间局限性两方面。
  - **时间局限性**：如果程序中的某条指令一旦执行，则不久以后该指令可能再次执行；如果某数据被访问，则不久以后该数据可能再次被访问。例如程序中存在着的大量的迭代循环、临时变量和子程序调用等。
  - **空间局限性**：一旦程序访问了某个存储单元，在不久以后，其附近的存储单元也将被访问，即程序在一段时间内所访问的地址，可能集中在一定的范围之内。例如对数组、表或数据堆栈进行操作。

5.7.1 虚拟存储器

- 基于局部性原理，把作业信息保存在磁盘上，当作业请求装入时，只需将当前运行所需要的一部分信息先装入主存，作业执行过程时，如果所要访问的信息已调入主存，则可继续执行；如果信息尚未装入主存，再将这些信息调入主存，使程序继续执行；如果此时主存已满，无法再装入新的信息，则系统将主存中暂时不用的信息置换到磁盘上，腾出主存空间后，再将所需要的信息调入主存，使程序继续执行。

- 这不仅使主存空间能充分地被利用，而且用户编制程序时可以不必考虑主存储器的实际容量，允许用户的逻辑地址空间大于主存的实际容量。从用户的角度来看，好像计算机系统提供了一个容量很大的主存储器，我们称它为“虚拟存储器”。

5.7.1 虚拟存储器

- 虚拟存储器指一种实际上并不存在的虚假存储器，它是系统为了满足应用对存储器容量的巨大需求而构造的一个非常大的地址空间。它使用户在编程时无需担心存储器之不足，好像有一个无限大的存储器供其使用一样。

- 虚拟存储器是建立在离散分配的存储管理方式的基础上，它允许将一个作业分多次调入主存。虚拟存储器实际上是为了扩大主存容量而采用的一种设计技巧，虚拟存储器的容量由计算机的地址结构和辅助存储器的容量所决定，与实际主存储器的容量无关，其逻辑容量由主存和辅存容量之和决定，其运行速度接近于主存储器的速度，而每位的成本却又接近辅助存储器。

5.7.1 虚拟存储器

- 虚拟存储器具有离散性、多次性、对换性和虚拟性四大主要特征。
  - **离散性**：打破连续存放的限制，离散化放置。
  - **多次性**：部分装载，多次调入。
  - **对换性**：运行过程中在主存和外存间对换。
  - **虚拟性**：逻辑上扩充主存容量。

5.7.2 请求分页式存储管理

- 请求分页式存储管理是在页式存储管理的基础上，增加请求分页功能和页面置换功能实现的虚拟存储系统。请求分页式存储管理允许作业只装入部分页面，就启动运行，在执行过程中，如果所要访问的页已调入主存，则进行地址转换，得到欲访问的主存物理地址；如果所要访问的页面不在主存中，则产生一个“缺页中断”，如果此时主存能容纳新页，则启动磁盘I/O将其调入主存；如果主存已满，则通过页面置换功能将当前所需的页面调入。

5.7.2 请求分页式存储管理

- 为了实现请求分页和页面置换功能，系统必须提供必要的硬件支持和相应的软件支持。一般需要以下支持：
  - 请求分页的页表机制
  - 缺页中断机构
  - 地址变换机构
  - 页面置换算法

5.7.2 请求分页式存储管理

- **页表机制**
  - 请求分页式存储管理的主要依据就是页表。由于只是将作业的部分页面调入主存，其余部分仍存放在辅助存储器上，因此必须指出哪些页面已在主存，哪些页面还没有装入。为此需要将页表增加若干项，修改后的页表格式如下所示：

页号	物理块号	状态位	访问字段	修改位	辅存地址

5.7.2 请求分页式存储管理

- **缺页中断机构**
  - 当访问的页面不在主存时，则由硬件发出一个缺页中断，操作系统必须处理这个中断，将所需要的页面调入主存。

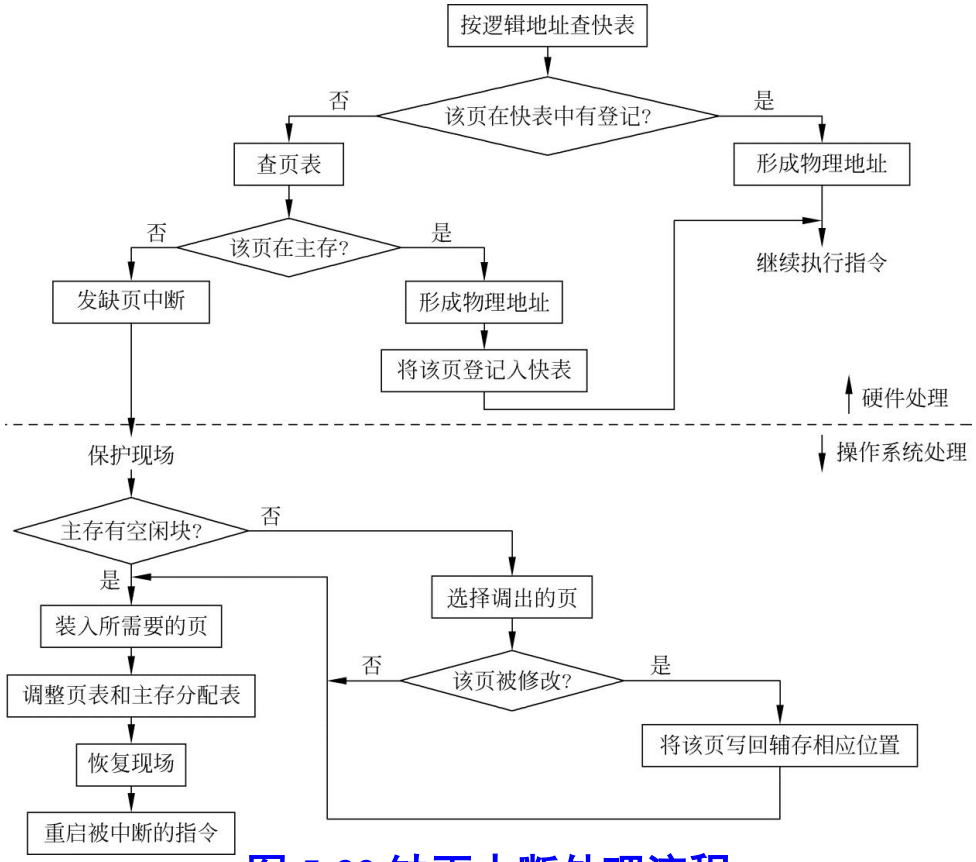


图 5-32 缺页中断处理流程

5.7.2 请求分页式存储管理

- 在处理缺页中断的过程中，同样需要保护现场、分析中断源、转入中断处理程序进行处理、恢复现场等步骤，但缺页中断又与一般的中断有着明显的区别，主要表现在：

  ① 在指令执行期间产生和处理中断信号。通常，CPU在一条指令结束后接收中断请求并响应。而缺页中断则是在**指令执行期间**，所要访问的指令或数据不在主存时所产生和处理的。

  ② 一条指令在执行期间可能产生**多次**缺页中断。

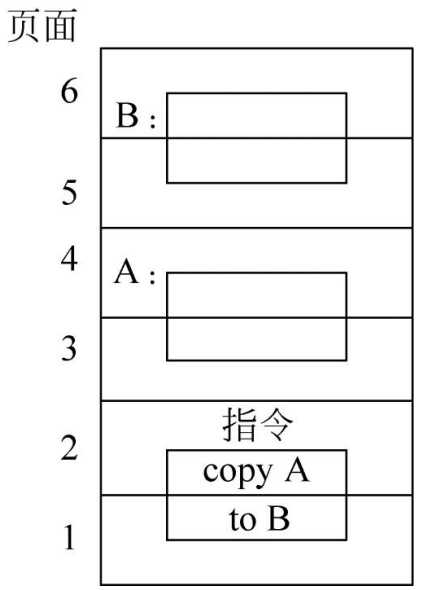


图 5-33 涉及6次缺页中断的指令

5.7.2 请求分页式存储管理

- **地址变换机构**
  - 在请求分页式存储管理中，当作业访问某页时，硬件的地址转换机构首先查找快表，若找到，并且其状态位为“1”，则按指定的物理块号进行地址转换，得到其对应的物理地址；若该页的状态位为“0”，则由硬件发出一个“缺页中断”，按照页表中指出的辅存地址，由操作系统将其调入主存，并在页表中填上其分配的物理块号，修改状态位、访问位，对于写指令，置修改位为“1”，然后按页表中的物理块号和页内地址形成物理地址..
  - 如果快表中未找到该页的页表项时，则到主存中查找页表，若该页尚未调入主存，系统产生缺页中断，请求操作系统将该页面调入，同时将此页表项写入快表。

5.7.2 请求分页式存储管理

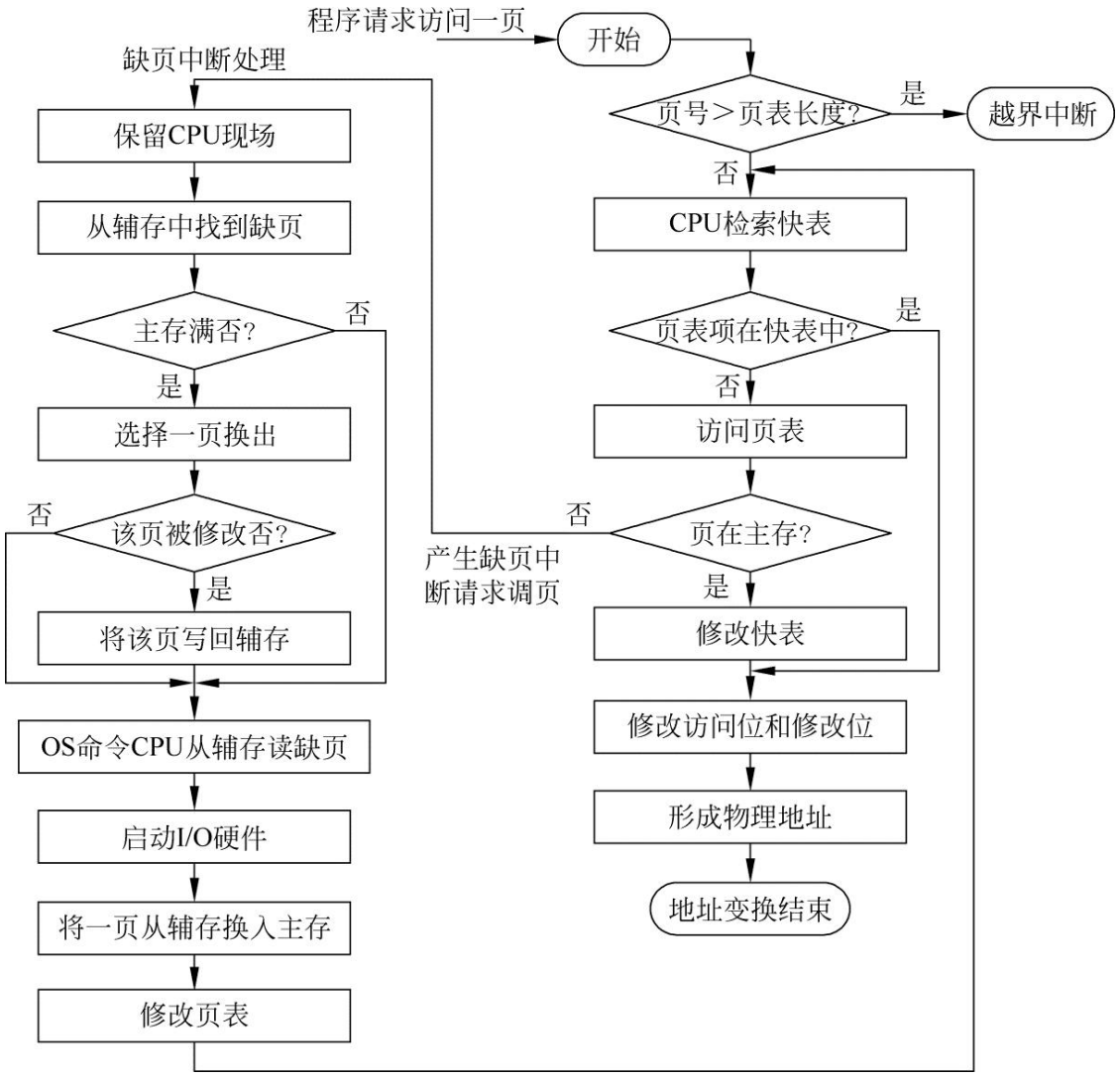


图 5-34 请求式分页存储管理地址变换过程

5.7.2 请求分页式存储管理

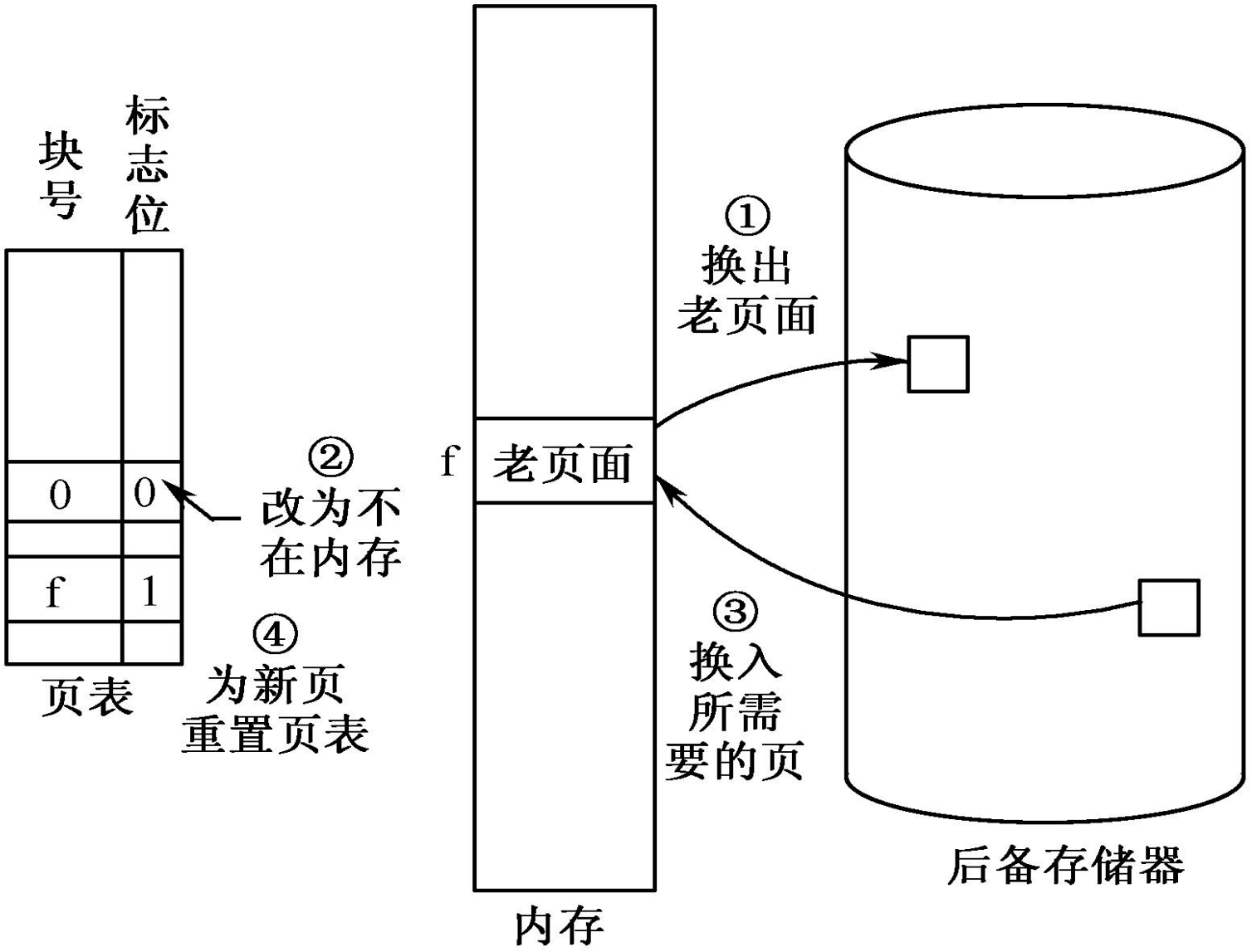
- **页面置换策略**
  - 在请求分页式存储管理中，可采用两种主存分配策略：**固定分配**和**可变分配**。
  - 在进行页面转换时，也可采用两种策略，**全局置换**和**局部置换**。
  - 可组合成三种适用的策略

① **固定分配局部置换策略**：进程主存物理块数目运行期间不变，页面置换时只能从该进程在主存的页面进行选择。

② **可变分区局部置换策略**：

③ **可变分区全局置换策略**：

页面置换流程示意


块号
标志位
0
0
f
1
页表
②
改为不
在内存
④
为新页
重置页表
f
老页面
内存
①
换出
老页面
③
换入
所需
要的页
后备存储器

页面置换算法

- 在作业运行过程中，如果所要访问的页面不在主存中，需要把他们调入主存，但主存中已没有空闲空间时，为了保证作业的运行，**系统必须按一定的算法选择一个已在主存中的页面，将它暂时调出主存，让出主存空间，用来存放所需调入的页面，这个工作称为“页面置换”**。选择换出页面的算法称为“页面置换算法”。
- 刚被调出的页面又立即要用，因而又要把它调入，而调入不久又被选中调出，调出不久又被调入，如此反复，使调度非常频繁，以至于大部分时间都花费在来回调度上。这种现象称为“**抖动**”或称“**颠簸**”。
- 一个好的置换算法应该尽可能地减少和避免抖动现象的发生。

页面走向

- **页面走向：存储的访问序列**
- **对于给定的页面大小，仅考虑其页号，不关心完整的地址。**
- **如果当前对页面p进行了访问，那么，马上又对该页访问就不会缺页。这样连续出现的同一页面就简化为一个页号。**

页面走向

- 如有以下地址序列（用十进制数表示）：
  - 0100，0432，0101，0612，0102，0103，0104，0101，0611，0102，0103，0104，0101，0610，0102，0103，0104，0101，0609，0102，0105
- 若每页100个字节，则页面走向简化为：
  - 1，4，1，6，1，6，1，6，1，6，1

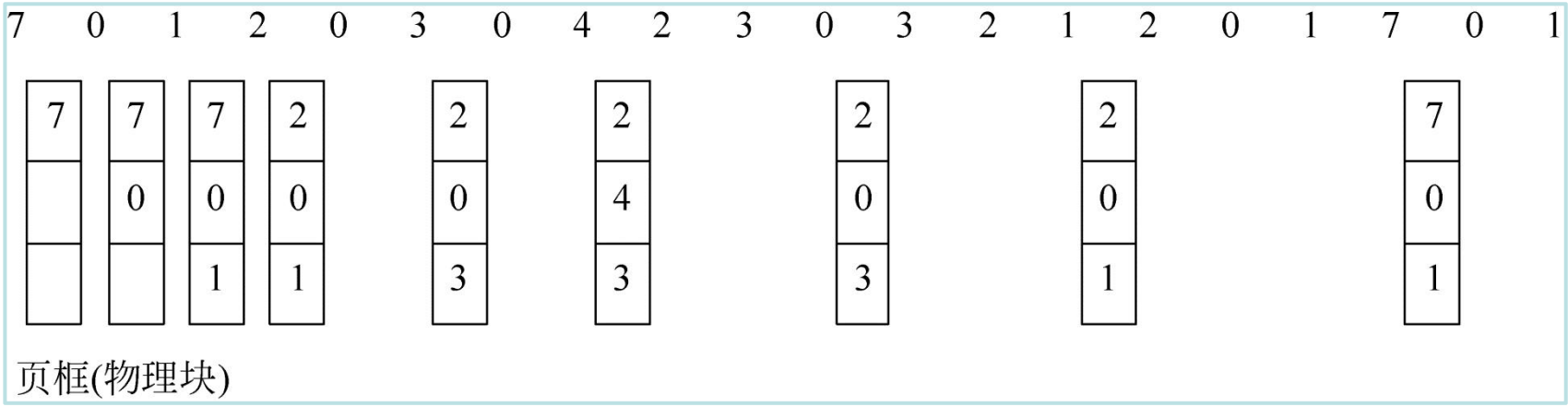
（1）最佳置换算法

- 最佳置换算法（Optimal Replacement, OPT）的实质是：为调入新页面而必须预先淘汰某个老页面时，所选择的老页面应在将来不被使用，或者是在将来最长时间内不再被访问。

- 采用最佳置换算法通常可获得最低的缺页中断率，但这是一种理想化的算法，无法实现。但是这个算法可以作为衡量其它算法的标准。

（1）最佳置换算法

- 假定某进程共有8页，且系统为之分配了3个物理块，并有以下页面调度序列：7，0，1，2，0，3，0，4，2，3，0，3，2，1，2，0，1，7，0，1。



采用最佳置换算法，只发生了9次页面置换，缺页中断率为45%。

（2）先进先出置换算法

- 先进先出页面置换算法认为刚被调入的页面在最近的将来被访问的可能很大，而在主存中驻留时间最长的页面在最近的将来被访问的可能性最小。因此，FIFO算法总是淘汰最先进入主存的页面，即淘汰在主存中驻留时间最长的页面。

- FIFO算法只需要把装入主存的页面按调入的先后次序链接成一个队列，并设置一个替换指针，指针始终指向最先装入主存的页面，每次页面置换时，总是选择替换指针所指示的页面调出。

（2）先进先出置换算法

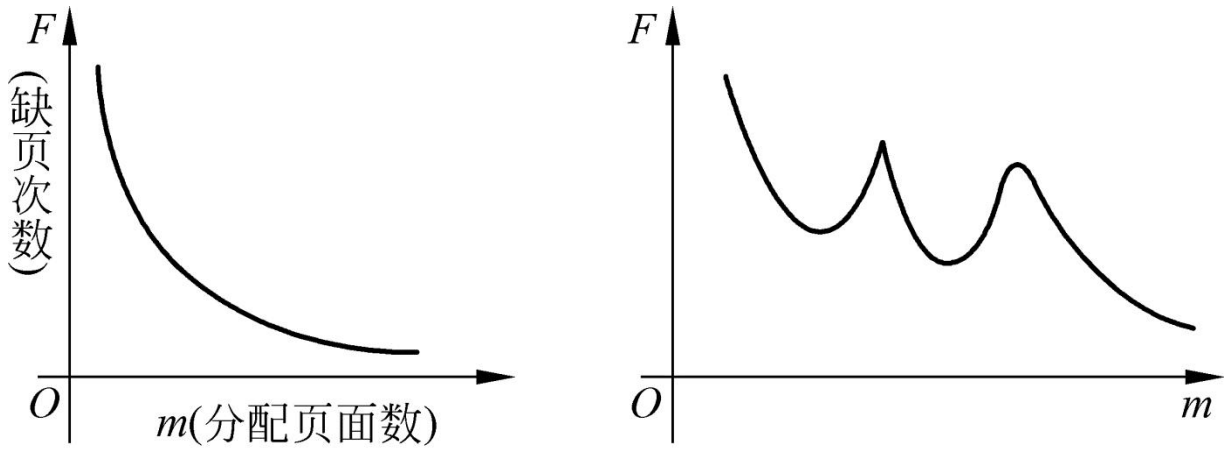
	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
最后进入主存的页	7	0	1	2	2	3	0	4	2	3	0	0	0	1	2	2	2	7	0	1
		7	0	1	1	2	3	0	4	2	3	3	3	0	1	1	1	2	7	0
最先进入主存的页			7	0	0	1	2	3	0	4	2	2	2	3	0	0	0	1	2	7
	+	+	+	+		+	+	+	+	+	+			+	+			+	+	+

(注：+表示产生一次缺页中断)

- 采用FIFO置换算法一共发生了15次页面置换，缺页中断率为75%，页面淘汰的顺序为7，0，1，2，3，0，4，2，3，0，1，2。
- FIFO算法简单，易实现，但效率不高。

（2）先进先出置换算法

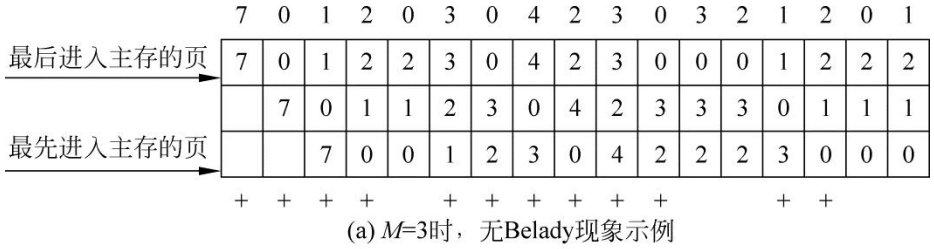
- 采用FIFO算法时，在未给作业分配满足它所要求的页面时，有时会出现这样的奇怪现象：分配的物理块数增多，而缺页中断次数反而增加。这种现象称之为Belady现象。*原因：没有考虑程序执行的动态特征。*



(a) 正常情况　　(b) Belady现象

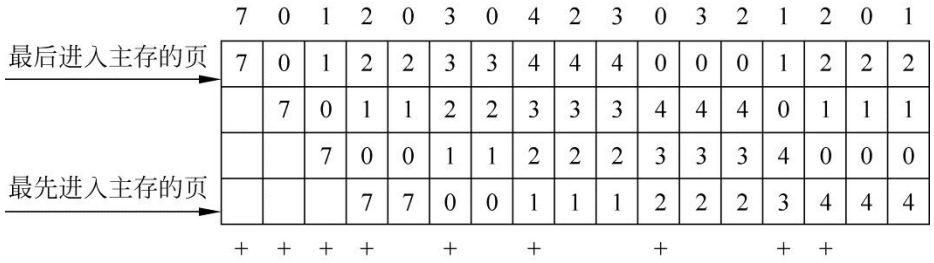
图5-37 FIFO置换算法的Belady现象

（2）先进先出置换算法



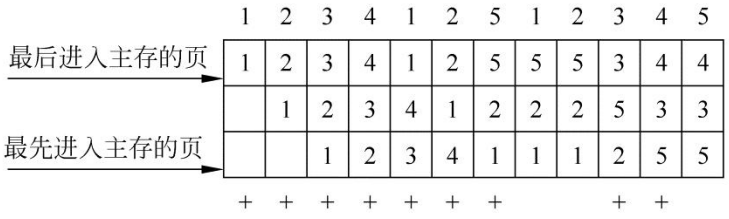
(a) M=3时，无Belady现象示例

缺页率为12/17=70.5%



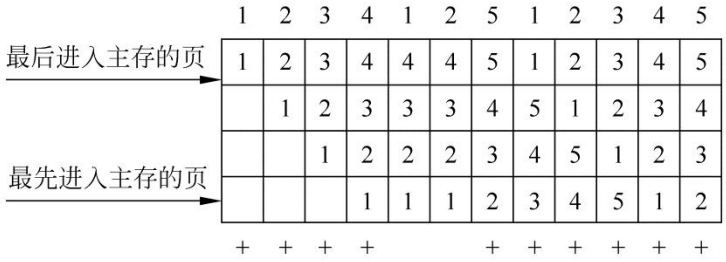
(b) M=4时，无Belady现象示例

缺页率为9/17=52.9%



(c) M=3时，Belady现象示例

缺页率为9/12=75%



(d) M=4时，Belady现象示例

(注：+表示产生一次缺页中断)

缺页率为10/12=83.3%

（3）最近最少使用置换算法

- 最近最少使用（Least Recently Used，LRU）置换算法总是选择**最近一段时间内最长时间没有被访问过**的页面调出。

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
最近被访问的页	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
最长时间未被访问的页		7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0
			7	0	1	2	2	3	0	4	2	2	0	3	3	1	2	0	1	7
	+	+	+	+		+		+	+	+	+			+		+		+		

（注：+ 表示产生一次缺页中断）

采用LRU置换算法一共产生12次缺页中断，
缺页中断率为60%，
页面淘汰的顺序为7，1，2，3，0，4，0，3，2。

（3）最近最少使用置换算法

- LRU为了记录页面自上次被访问以来所经过的时间，需要在页表中增加一个“引用位”标志，在每次被访问后将引用位置零，重新计时，这样，在发生缺页中断需要调入新的页面时，通过检查页表中各页的引用位，选择计时最长时间没有被访问过的页面淘汰，并且把主存中所有页面的引用位全部清零，重新计时。

（3）最近最少使用置换算法

- LRU近似算法（时钟置换算法Clock）是在页表中为每一页增加一个引用位信息，当该页被访问时，由硬件将它的引用位信息置为1，操作系统选择一个时间周期T，每隔一个周期T，将页表中所有页面的引用位信息置 0，这样，在时间周期T内，被访问过的页面的引用位为 1，而没有被访问过的页面的引用位仍为0。当产生缺页中断时，可以从引用位为0的页面中选择一页调出，同时将所有页面的引用位信息全部重置0。
- 这种近似算法的实现比较简单，但关键在于时间周期 T 大小的确定。T 太大，可能所有的引用位都变成1，找不出最近最少使用的页面淘汰；T太小，引用位为0的页面可能很多，而无法保证所选择的页面是最近最少使用的。

（3）最近最少使用置换算法

- 如果把页表中的“引用位”和“修改位”结合起来使用可以改进时钟页面替换算法，它们一共组合成四种情况：
  ① 最近没有被引用，没有被修改（r=0，m=0）
  ② 最近被引用，没有被修改（r=1，m=0）
  ③ 最近没有被引用，但被修改（r=0，m=1）
  ④ 最近被引用过，也被修改过（r=1，m=1）

- 采用了这种策略，其主要优点是没有被修改过的页面会被优先选出来，淘汰这种页面时不必写回磁盘，从而节省时间，但查找一个淘汰页面可能会经过多轮扫描，算法的实现开销较大。

（4）最近最不常用置换算法

- 最近最不常用（Least Frequently Used, LFU）置换算法总是选择被访问次数最少的页面调出，即认为在过去的一段时间里被访问次数多的页面可能经常需要访问。
- 一种简单的实现方法是为每一页设置一个计数器，页面每次被访问后其对应的计数器加1，每隔一定的时间周期T，将所有计数器全部清零。这样，在发生缺页中断时，选择计数器值最小的对应页面淘汰，显然它是最近最不常用的页面，同时把所有计数器清零。这种算法实现比较简单，但代价很高，同时有一个关键问题是如何选择一个合适的时间周期T。

LRU vs. LFU

内存页面	时间T内 是否访问	时间T内 访问次数	LRU 淘汰页面	LFU 淘汰页面
2	1	1	7	2
4	1	3		
7	0	6		

5.4.4 时钟页面置换算法

- 建立一个类似钟面的环形链表以保存进入内存的页面，一个指针指向最老的页。
- 当发生缺页中断时，首先检查指针指向的页：
  - 如果R为0就淘汰该页，并将新页插入该位置，然后指针前移一个位置；
  - 如果R为1清除R位并把指针前移一个位置，重复这个过程直到找到一个R位为0的页为止。

缺页中断率分析

- 虚拟存储系统解决了有限主存的容量限制问题，能使更多的作业同时多道运行，从而提高系统的效率，但缺页中断处理需要系统的额外开销，影响系统效率，因此应尽可能减少缺页中断的次数，降低缺页中断率。

- 假定一个作业共有n页，系统分配给它的主存块是m块（m、n均为正整数，且1≤m≤n）。因此，该作业最多有m页可同时被装入主存。如果作业执行中**访问页面的总次数为A**，其中有F次访问的页面尚未装入主存，故产生了**F次缺页中断**。现定义：**f＝F/A**，则f称为“**缺页中断率**”。

缺页中断率分析

- 缺页中断率与缺页中断的次数有关。

- 影响缺页中断率的因素有：
  ① 分配给作业的主存块数
  ② 页面的大小
  ③ 程序编制方法
  ④ 页面调度算法

缺页中断率分析

- （1）分配给作业的主存块数
  - 系统分配给作业的主存块数多，则同时装入主存的页面数就多，那么缺页中断次数就少，缺页中断率就低，反之，缺页中断率就高。

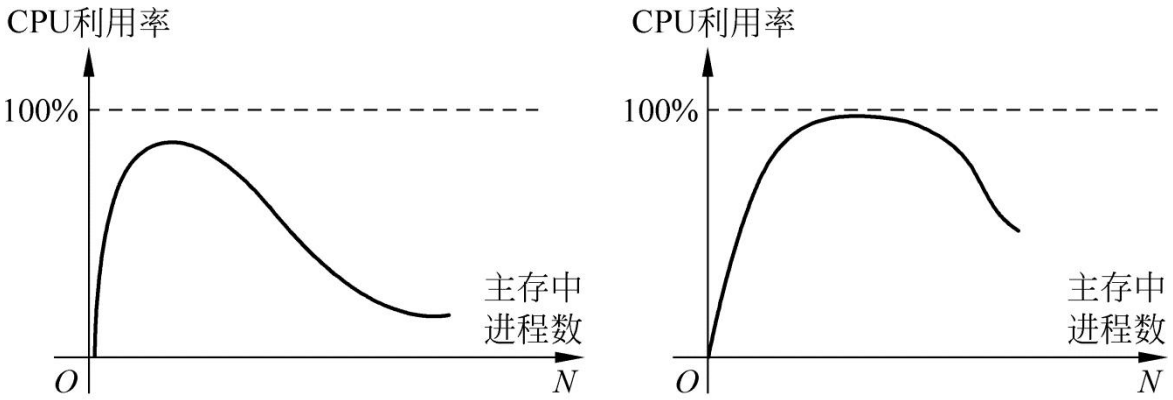


图5-40 处理器使用率与进程数的关系

缺页中断率分析

- 主存物理块数增加到临界值以后，即使再增加较多的物理块，进程的缺页中断次数也不再明显减少。大多数程序都有这样一个特定点，在这个特定点以后再增加主存容量，缺页中断次数的减少并不明显。

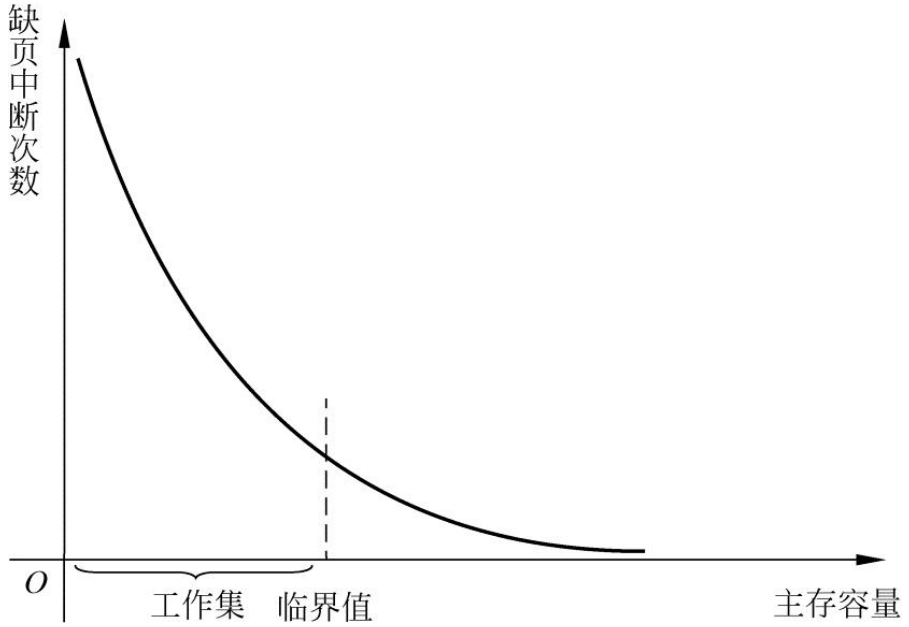


图5-41 主存容量与缺页中断的关系

工作集

- Denning认为：程序在运行时对页面的访问是不均匀的，即往往在某段时间内访问仅局限于较少的若干页面，如果能够预知程序在某段时间间隔内要访问哪些页面，并能将它们提前调入主存，将会大大地降低缺页率，从而减少置换工作，提高CPU的利用率。

- 对于给定的访问序列选取定长的时间间隔(⊿)，称为工作集窗口，落在工作集窗口中的页面集合称为工作集。即工作集是指在某段时间间隔里，进程实际要访问的页面的集合。把某进程在时间t的工作集记为w(t,⊿),把变量⊿称为工作集“窗口尺寸”。

工作集

假如有以下页面访问序列，窗口尺寸⊿=9：

2615777516234123444343441327

|----------| |----------|

⊿ t1 ⊿ t2

t1 时刻的工作集

w(t1)={1,2,5,6,7}

t2 时刻的工作集

w(t2)={3,4}

工作集

页面访问序列1	窗口尺寸 2	3	4	5
24	24	24	24	24
15	15 24	15 24	15 24	15 24
18	18 15	18 15 24	18 15 24	18 15 24
23	23 18	23 18 15	23 18 15 24	23 18 15 24
24	24 23	24 23 18	*	*
17	17 24	17 24 23	17 24 23 18	17 24 23 18 15
18	18 17	18 17 24	*	*
24	24 18	*	*	*
18	18 24	*	*	*
17	17 18	*	*	*
17	17	*	*	*
15	15 17	15 17 18	15 17 18 24	*
24	24 15	24 15 17	*	*
17	17 24	*	*	*
24	24 17	*	*	*
18	18 24	18 24 17	*	*

工作集

- 在有些操作系统中，如Windows，虚拟存储管理程序为每一个进程分配固定数量的物理块，并且这个数目可以进行动态的调整。这个数目就是由每个进程的工作集来确定，并且根据主存的负荷和进程的缺页情况动态地调整其工作集。

- 当每个工作集都已达到最小值时，虚存管理程序跟踪进程的缺页数量，根据主存中自由页面数量可以适当增加其工作集的大小。

页面大小

- 页面的大小影响页表的长度、页表的检索时间、页面置换时间、可能的页内零头大小等，对缺页中断的次数也有一定的影响。
- 如果划分的页面大，一个作业的页面数就少，页表所占用的主存空间少且查表速度快，在系统分配相同主存块的情况下，发生缺页中断的次数减少，降低了缺页中断率，但是一次换页需要的时间延长，可能产生的页内零头所带来的空间浪费较大。
- 反之，一次换页需要的时间就短，可能产生的页内零头少，空间利用率提高，但是一个作业的页面数多，页表所占用的主存空间长且查表速度慢，在系统分配相同主存块的情况下，发生缺页中断的次数增多，增加了缺页中断率。

程序编制方法

- 程序编制的方法不同，对缺页中断的次数也有很大影响。缺页中断率与程序的局部化程度密切相关。一般，希望编制的程序能经常集中在几个页面上进行访问，以减少缺页中断次数，降低缺页中断率。

- 例如，一个程序要将128×128数组置初值“0”。现假设分配给该程序的主存块数只有一块，页面的大小为每页128个字，数组中每一行元素存放在一页中。开始第一页已经调入主存。

程序编制方法

- 程序编制的方法不同，对缺页中断的次数也有很大影响。缺页中断率与程序的局部化程度密切相关。一般，希望编制的程序能经常集中在几个页面上进行访问，以减少缺页中断次数，降低缺页中断率。

- 例如，一个程序要将128×128数组置初值“0”。现假设分配给该程序的主存块数只有一块，页面的大小为每页128个字，数组中每一行元素存放在一页中（按行存储）。开始第一页已经调入主存。

程序编制方法

```
int A［128］［128］;
for (j=0;j<128;j++ )
   for (i=0;i<128;i++)
      A［i］［j］=0;
```

按列处理，与数据存储顺序不一致，

将产生（128*128-1）次缺页中断。

程序编制方法

```
int A［128］［128］;
for (i=0;i<128;i++ )
   for (j=0;j<128;j++)
      A［i］［j］=0;
```

按行处理，与数据存储顺序一致，

只产生（128-1）次缺页中断。

5.7.3 请求分段式存储管理

- 请求分段式存储管理以段式存储管理为基础，为用户提供比主存实际容量更大的虚拟空间。请求分段式存储管理把作业的各个分段信息保留在磁盘上，当作业被调度进入主存时，首先把当前需要的一段或几段装入主存，便可启动执行，当所要访问的段已在主存，则将逻辑地址转换成绝对地址；如果所要访问的段尚未调入主存，则产生一个“缺段中断”，请求操作系统将所要访问的段调入。为实现请求分段式存储管理，需要有一定的硬件支持和相应的软件。
- 请求分段式存储管理所需要的硬件支持有段表机制、缺段中断机构，以及地址变换机构等。

5.7.3 请求分段式存储管理

- **段表机制**
- 在请求分段式存储管理中，段表是进行段调度的主要依据。在段表中需要增设段是否在主存，各段在磁盘上的存储位置，已在主存的段需要指出该段在主存中的起始地址和占用主存区的长度，还可设置该段是否被修改，是否可扩充等标志信息。请求分段式存储管理的段表结构如下：

段名	段长	段基址	存取方式	访问字段A	修改位M	状态位P	扩充位	辅存始址

5.7.3 请求分段式存储管理

- **缺段中断机构**
- 在请求分段式存储管理中，当所要访问的段尚未调入主存时，则由缺段中断机构产生一个“缺段中断”信号，请求操作系统将所要访问的段调入主存。操作系统处理缺段中断的步骤如下：
  ① 空间分配；
  ② 修改段表；
  ③ 新的段被装入后应让作业重新执行被中断的指令，这时就能在主存中找到所要访问的段，可以继续执行下去。

5.7.3 请求分段式存储管理

- **地址变换机构**
- 请求分段式存储管理方式中的地址变换在分段式存储管理系统的地址变换机构的基础上增加缺段中段请求及处理等形成。
- 请求分段式存储管理还是以段为单位进行主存空间的分配，整段信息的装入、调出，有时需要主存空间的移动，这些都增加了系统的开销。

5.7.3 请求分段式存储管理

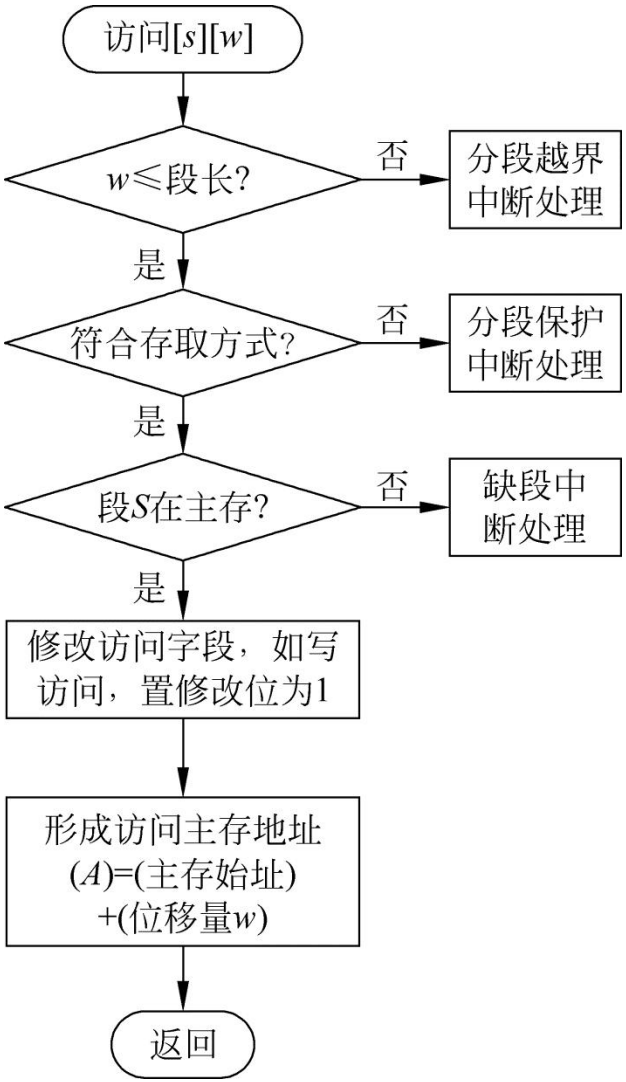


图5-43 请求分段式存储管理系统地址变换过程

5.7.4 请求段页式存储管理

- 如果按段页式存储管理方式，每个作业仍然**按逻辑分段，把每一段再分成若干页面**，这样，在请求式存储管理中，每一段不必占用连续的存储空间，可按页存放在不连续的主存块中，甚至当主存块不足时，可以将一段中的部分页面装入主存。这种管理方式称为“**请求段页式存储管理**”。

5.7.4 请求段页式存储管理

- 采用请求段页式存储管理，需要对每一个装入主存的作业建立一张段表，对每一段建立一张页表，段表中指出该段对应页表所存放的起始地址及其长度，页表中应指出该段的每一页在磁盘上的位置以及该页是否在主存，若在主存，则填上占用的主存块号。作业执行时按段号查找段表，找到相应的页表再根据页号查找页表，由状态位判定该页是否已在主存，若在，则进行地址转换，否则进行页面调度。
- 请求段页式存储管理结合了请求分段式虚拟管理和请求分页式虚拟管理的优点，但增加了设置表格（段表、页表）和查表等开销。

5.9 本章小结

- 存储管理对主存中的用户区进行管理，其目的是尽可能地**提高主存空间的利用率**，使主存在成本、速度和规模之间获得较好的权衡。存储管理的基本功能有主存空间的分配与去配、地址转换、主存空间的共享与保护、主存空间的扩充。
- 在现代计算机系统中，存储部件采用层次结构来组织，具体可分为寄存器、高速缓存、主存储器、磁盘缓存、磁盘、可移动存储介质等组成，这样在成本、速度和规模等诸因素中能获得较好的性能价格比。
- 多道程序设计系统中，为了方便程序编制，用户程序中使用逻辑地址，而CPU则是按物理地址访问主存，读取指令和数据，为了保证程序的正确执行，需要进行**地址转换**。地址转换又称为重定位，有**静态重定位**和**动态重定位**。采用动态重定位的系统支持程序的浮动。

5.9 本章小结

- 早期单用户、单任务操作系统中主存管理采用单用户连续存储管理方式。现代操作系统支持多道程序设计，满足多道程序设计最简单的存储管理技术是分区管理，有固定分区管理和可变分区管理。固定分区管理采用顺序分配算法进行主存空间的分配，采用静态重定位方式将作业装入主存，通过上、下限寄存器实现存储保护。可变分区管理的空间分配算法包括：最先适应、最优适应和最坏适应等算法。回收空间时检查是否存在与回收区相邻的空闲分区，如果有，则将其合并成为一个新的空闲分区进行登记管理。可变分区管理一般采用动态重定位方式将作业装入主存，通过基址寄存器和限长寄存器等硬件实现存储保护。

5.9 本章小结

- 可变分区管理可以有条件地采用移动技术使分散的空闲区汇集起来容纳新的作业。分区管理中，主存空间不足时，**交换技术**和**覆盖技术**可以达到扩充主存的目的。分区管理方式要求作业信息一次性连续装入主存，对空间的要求较高，为了解决这个矛盾，操作系统引入离散存储管理方式。离散存储管理方式主要有页式存储管理、段式存储管理和段页式存储管理。离散存储管理允许将一个作业信息存储在不相邻的主存空间中，通过操作系统建立页表（或段表）数据结构实现逻辑地址空间与物理地址空间的映射，实现地址空间的保护。

5.9 本章小结

- 虚拟存储器的实现借助于大容量的辅助存储器存放作业信息，操作系统利用作业的局部性特点把当前作业信息装入主存，并且利用页表、段表等数据结构构造一个用户的虚拟空间。操作系统根据中断（缺页中断、缺段中断）进行处理，选择一种合适的调度算法对主存和辅存中的信息进行调入和调出，尽可能地避免“抖动”现象的发生。虚拟存储管理有请求式分页存储、请求式分段存储和请求式段页存储。请求式分页存储管理的实现需要必要的硬件支持和相应的软件支持，涉及到请求分页的页表机制；缺页中断机构；地址变换机构；页面置换算法等。其中页面置换算法主要有：最佳置换算法、先进先出置换算法、最近最少用置换算法和最近最不常用置换算法等。虚拟存储器的性能与缺页中断率密切相关，系统分配给作业的主存物理块数、页面的大小以及程序的编制方法等对缺页中断率都有影响。