# 第六章 基本存储管理

存储器是价格昂贵，数量不足的资源。存储管理的效率直接影响到系统的性能，也最能够反映一个操作系统的特色。因此，存储管理的问题是操作系统的核心问题。

这里只讨论内存管理，外存管理在文件管理中讨论。

## 存储管理的目的

1. 为用户使用存储器提供方便：
   ① 在逻辑空间编程。
   ② 提供足够大的存储空间。
2. 充分发挥内存的利用率。

# 本章主要内容

6.1 存储管理的基本功能

6.2 分区存储管理

6.3 内存扩充技术

6.4 分页存储管理

6.5 分段存储管理

6.6 段页式存储管理

# 6.1 存储管理的基本功能

## 6.1.1 存储系统的基本概念

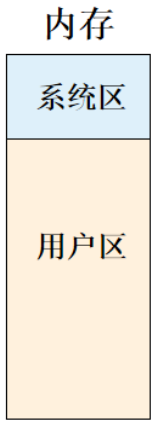
在计算机系统中，存储是有层次的。

表 6-1　计算机的存储层次及特点

| 存 储 层 次 | 特　　点 |
| --- | --- |
| 寄存器组 | 在 CPU 内部，访问速度最快；但数量有限，只用于暂存正在执行的指令的中间数据结果 |
| 高速缓存 | 在 CPU 内部，访问速度介于寄存器和内存之间；主要用于备份内存中常用的数据，减少 CPU 对内存的访问次数 |
| 内存 | CPU 可以直接访问内存的数据，访问速度快；但价格高，且具有易失性不能提供永久存储，主要用于保存当前正在使用的程序和数据 |
| 外存（硬盘、光盘、U 盘、软盘等） | 外存的数据需调入内存后才能被 CPU 访问，访问速度慢；但价格成本低，可长期存储，用于长期存储程序和数据 |

# 6.1.1 存储系统的基本概念

- 通常，内存被划分为系统区和用户区两部分。
  - 系统区： 用于加载操作系统常驻内存部分（内核）；
  - 用户区：除系统区以外的全部内存空间，供当前正在执行的用户程序使用。
- 在多道程序环境中，操作系统的存储管理模块需要对内存的用户区进行细分，以加载多个用户程序。




# 1.物理地址和逻辑地址

(1)物理地址和物理地址空间

- 内存单元的地址称为物理地址，也称为绝对地址。物理地址反映了数据在内存的实际存放位置。
- 物理地址的集合称为物理地址空间，也称为绝对地址空间。
- 在单道程序环境中，一个程序总是被装入内存用户区的起始空间去执行。因此，程序员在编写程序时，可以使用物理地址指明要访问的数据或执行的指令在内存的位置。
- 在多道程序环境中，程序员事先并不知道程序会被装入内存的哪个区域执行，因此编程时就无法使用物理地址了。

# 1.物理地址和逻辑地址

## (2)逻辑地址和逻辑地址空间

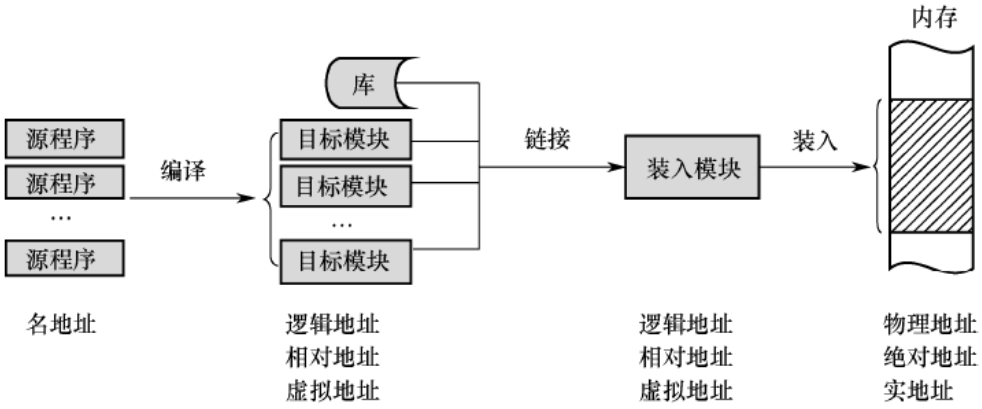



图 6-1　对用户程序中地址的处理过程

在多道程序环境中，装入模块被加载到内存的位置是不确定的，因此编译链接环节无法将名地址转换成内存物理地址，便对目标模块从0开始为其编址。

- 这种与物理地址无关的访问地址称为逻辑地址/相对地址。
- 逻辑地址的集合称为逻辑地址空间/相对地址空间。

# 2.地址重定位

- 地址重定位负责把用户程序中的逻辑地址转换为内存中的物理地址。地址重定位又称为地址变换、地址映射。
- 根据地址重定位的时机不同，地址重定位又分为：
  - 静态地址重定位
  - 动态地址重定位

# (1)静态地址重定位

- 静态地址重定位发生在程序被装入内存的过程中，在程序运行之前就完成了地址重定位。

  物理地址 = 装入内存的起始地址 + 逻辑地址

- 优点：
  - 静态地址重定位不需要硬件支持。
- 缺点：
  - 用户程序只能装入内存的一个连续存储区域上，不能装入多个离散区域。
  - 用户程序地址重定位后不允许在内存中移动。

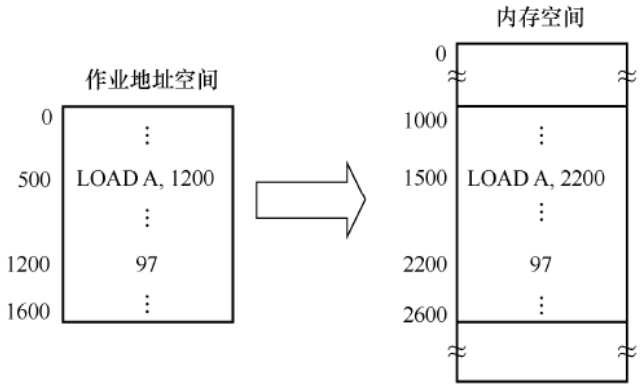



图 6-2　静态地址重定位示意图

# (2)动态地址重定位

- 动态地址重定位发生在程序运行过程中，即当执行到某条指令且该指令需要进行内存访问时，再将逻辑地址转换为相应的物理地址。
- 使用重定位寄存器存放正在执行的用户进程在内存的起始地址。

物理地址 = 重定位寄存器的值 + 逻辑地址

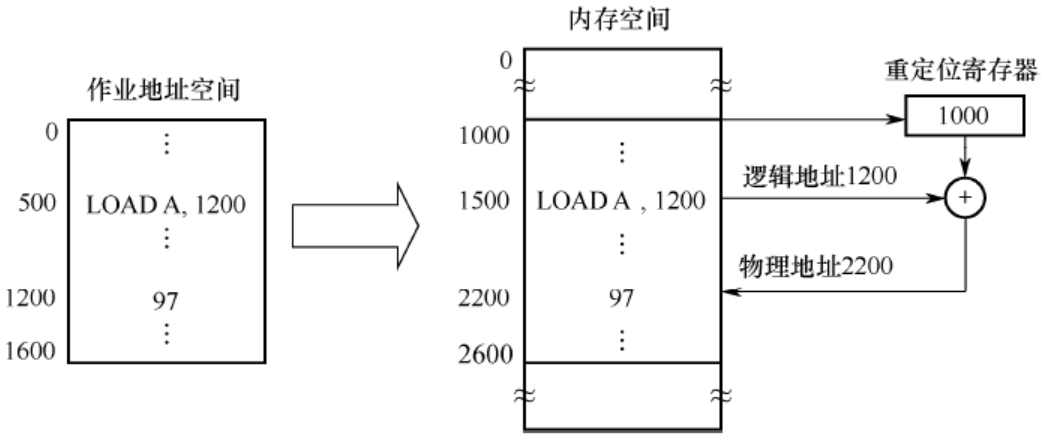
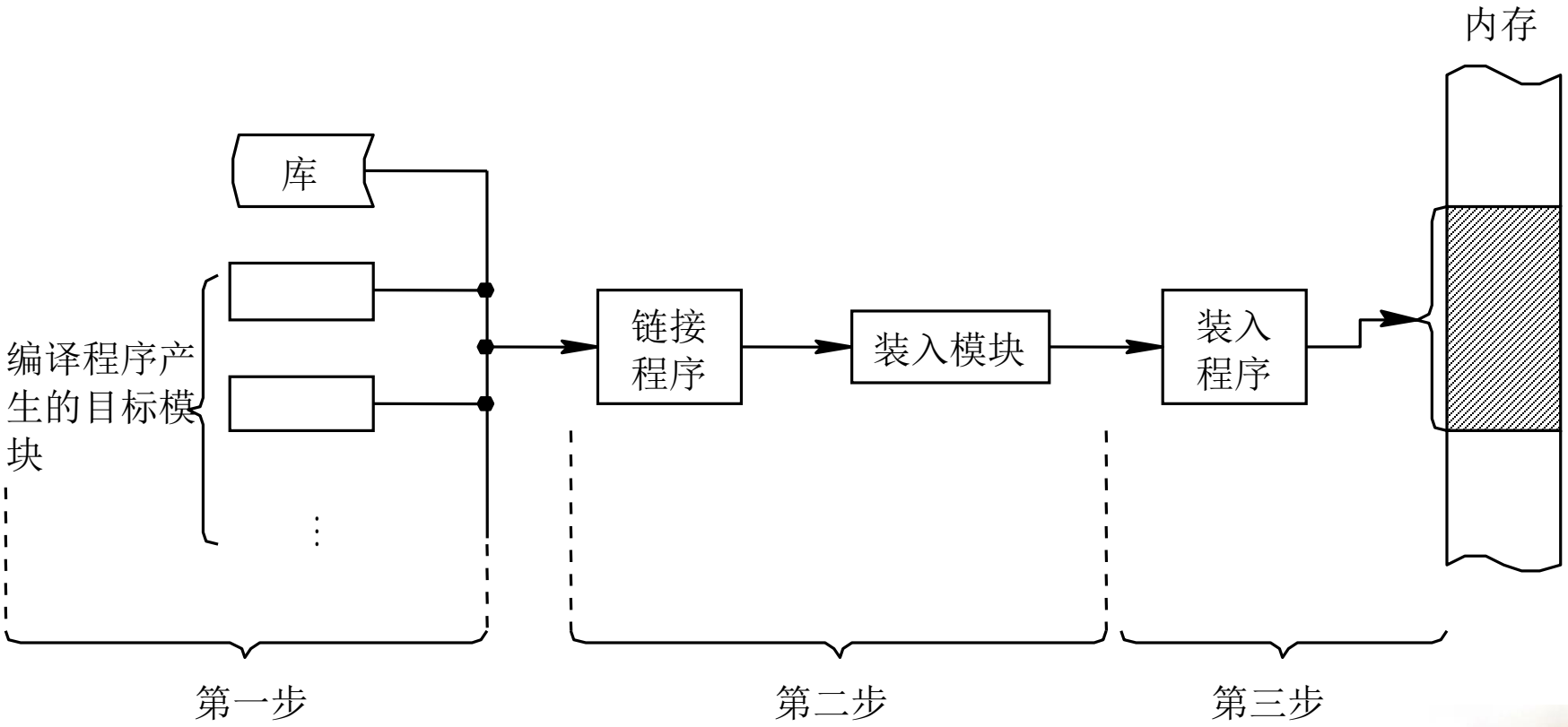



图 6-3　动态地址重定位示意图

# (2)动态地址重定位

- 优点：
  - 支持程序执行过程在内存中移动。当程序在内存发生移动时，只需修改重定位寄存器为新的内存起始地址，就可以实现正确的地址重定位。
  - 支持程序在内存中离散存储。
- 缺点：
  - 与静态地址重定位相比，动态地址重定位需要附加硬件支持（重定位寄存器），增加了机器成本。

# 对用户程序的处理步骤



## 3.程序的装入方式

只能用于单道程序环境

### （1）绝对装入方式(Absolute Loading Mode)

装入模块中是绝对地址，按照装入模块中的绝对地址将程序和数据装入内存。既可在编译或汇编时给出，也可由程序员直接赋予。通常在程序中采用符号地址，在编译或汇编时，再将这些符号地址转换为绝对地址。

可用于多道程序环境

## （2）可重定位装入方式(Relocation Loading Mo

装入模块为相对地址（逻辑地址），装入程序按照当前内存使用情况，将装入模块装入内存的某个物理地址。但是装入后不允许移动。— 静态重定位

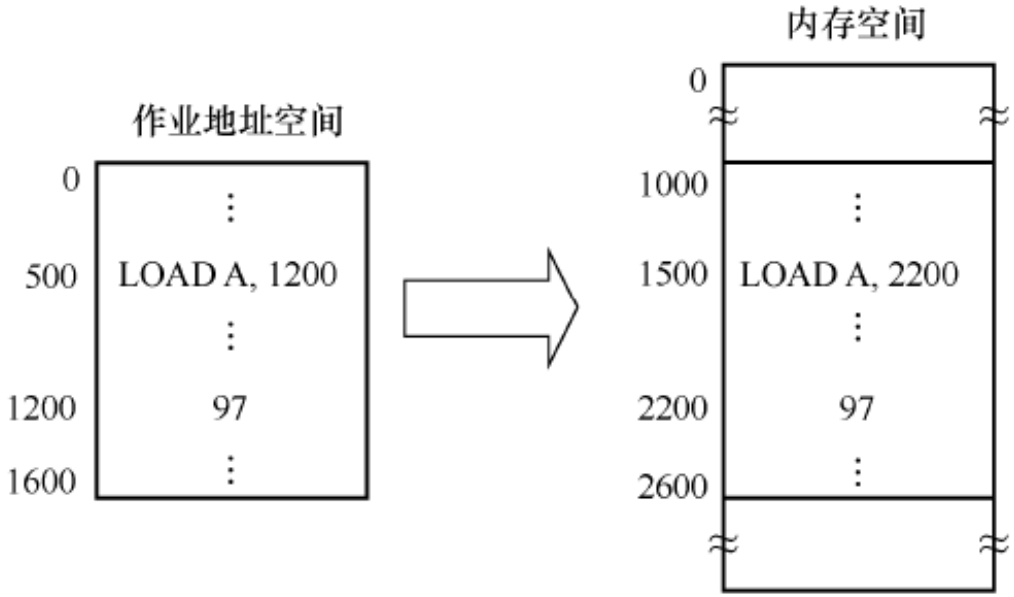


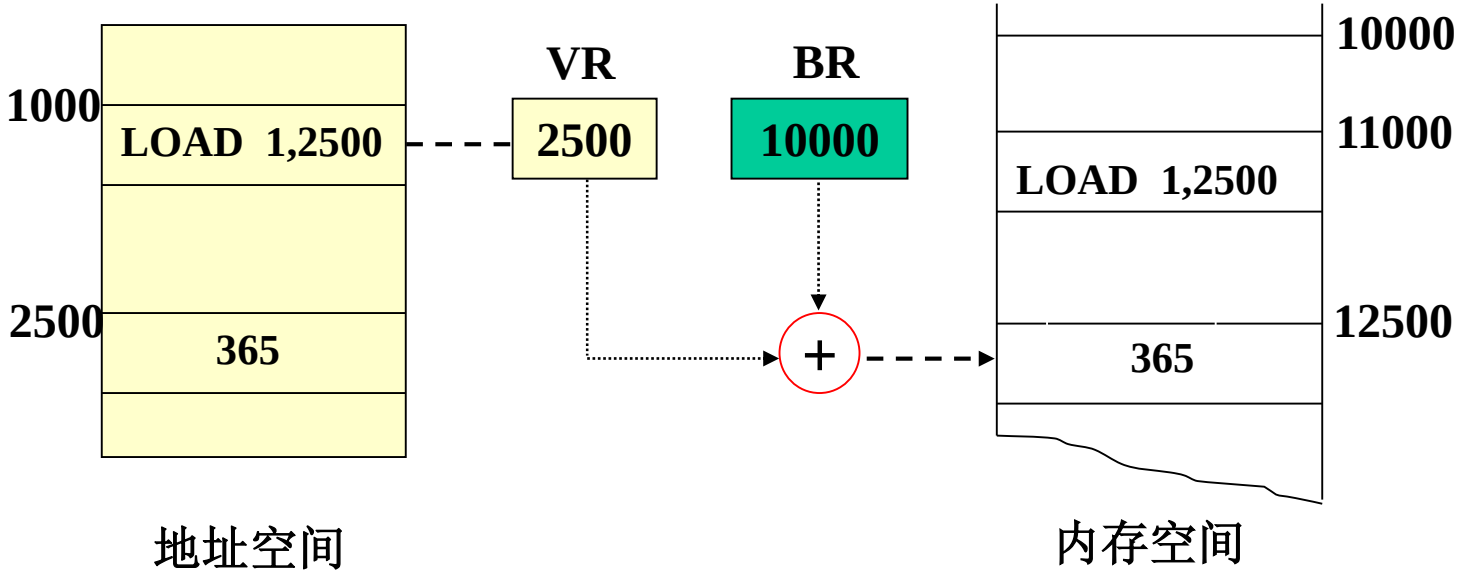

图 6-2　静态地址重定位示意图

可用于多道程序环境

## （3）动态运行时装入方式(Denamle Run-time Loading)

把装入模块装入内存后，并不立即把装入模块中的相对地址转换为绝对地址，而是把这种地址转换推迟到程序真正要执行时才进行。装入内存后的所有地址都仍是相对地址。此方式需要特殊硬件的支持。

——动态重定位


1000
LOAD 1,2500
2500
365
VR
2500
BR
10000
+
10000
11000
LOAD 1,2500
12500
365
地址空间
内存空间

# 6.1.2 存储管理的基本功能

- 存储管理的目标是合理有效地组织存储空间管理，进行内存的分配和回收，以支持多个进程并发执行，同时提高内存空间的利用率并方便用户使用。因此，存储管理应具有以下基本功能：

- 内存分配和回收
- 内存共享和保护
- 地址映射
- 内存扩充

# 6.2　分区存储管理

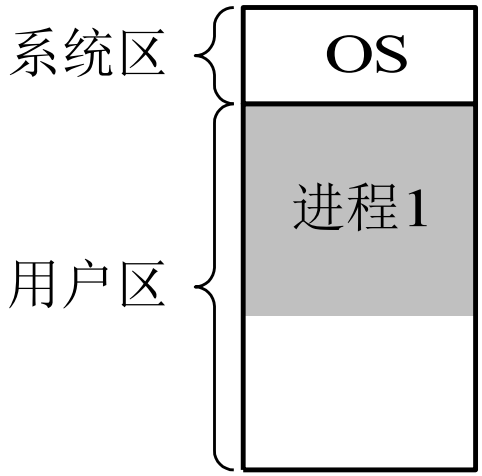
- 6.2.1 固定分区存储管理
- 6.2.2 动态分区存储管理
- 6.2.3 动态重定位分区存储管理

# 连续分配存储管理方式

## 1. 单一连续分配

只能用于单用户、单任务操作系统

采用这种存储管理方式时，可把内存分为系统区和用户区两部分，系统区仅提供给OS使用，通常是放在内存的低址部分；用户区是指除系统区以外的全部内存空间，提供给用户使用。



(1) 内存分配：一道用户程序独占用户区。

(2) 地址映射：物理地址=用户区基地址+逻辑地址。

(3) 内存保护：通过基址寄存器保证用户程序不会从系统区开始；另外需要一个界限寄存器，存储程序的逻辑地址范围，若进行映射的逻辑地址超过了界限寄存器中的值，则产生一个越界中断信号送CPU。

优点：易于管理。

缺点：对要求内存空间少的程序，造成内存浪费；程序全部装入，很少使用的程序部分也占用内存。

# 6.2.1 固定分区存储管理

固定式分区分配的“固定”主要体现在系统的分区数目固定和分区大小固定两个方面。

1. 划分分区的方法

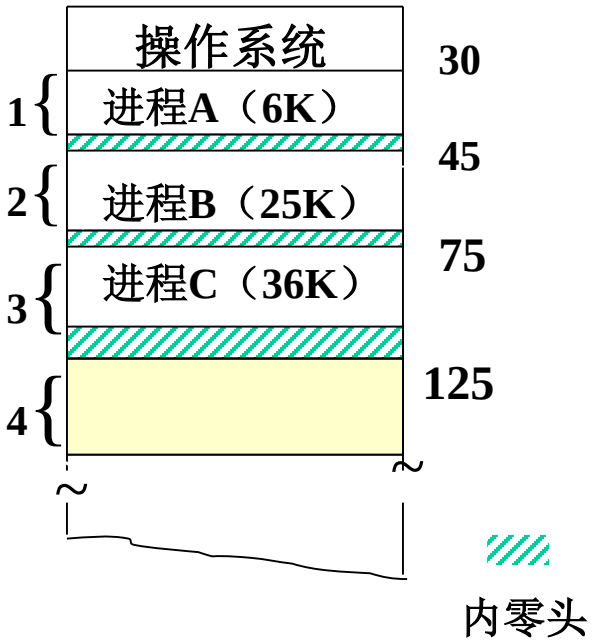
a. 分区大小相等， 使所有的内存分区大小相等。分区太大，内存空间浪费。太小，不足以装入程序，无法运行。例：一台计算机控制多台相同的冶炼炉。

b. 分区大小不等。在内存中划分出多个较小的分区、适量的中等分区及少量大分区。

# 2. 内存分配

| 分区号 | 大小（K） | 始址（K） | 状态 |
|---|---|---|---|
| 1 | 15 | 30 | 已分配 |
| 2 | 30 | 45 | 已分配 |
| 3 | 50 | 75 | 已分配 |
| 4 | 100 | 125 | 未分配 |

图　固定分区使用表




简单、可靠，但产生分区“内零头”。内存利用率低。

# 6.2.2　动态分区存储管理

- 分区不是固定划分好的，而是根据进程的实际需要动态地划分。动态分区存储管理也称为可变分区存储管理。
- 系统初始化后，除操作系统常驻内存部分之外，内存的用户区只有一个空闲分区。随后，分配程序根据进程的实际需求依次划分出分区，供进程使用。

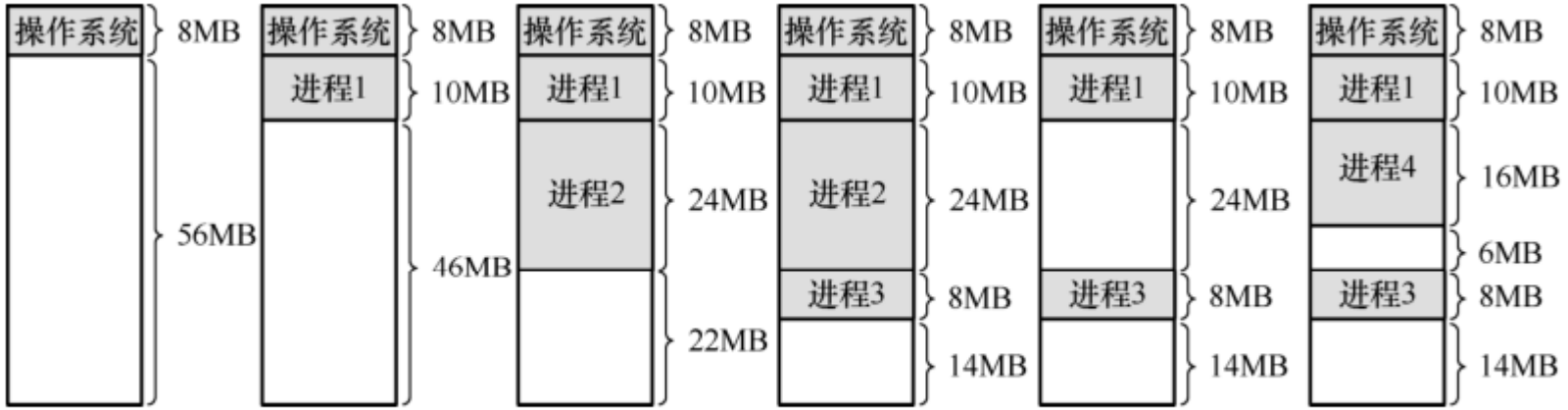



图 6-6　动态分区存储管理示意图

# 6.2.2　动态分区存储管理

**1.分区分配中的数据结构**

**a.空闲分区表。**

**b.空闲分区链。**

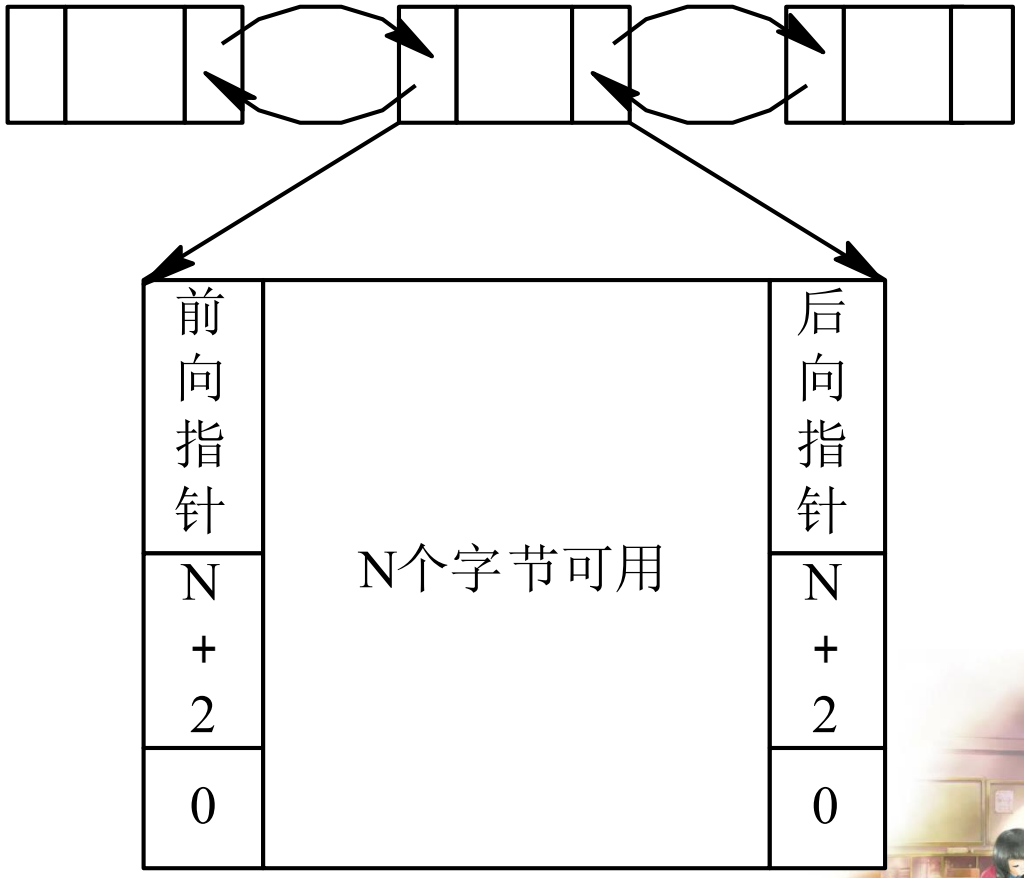



图　空闲链结构

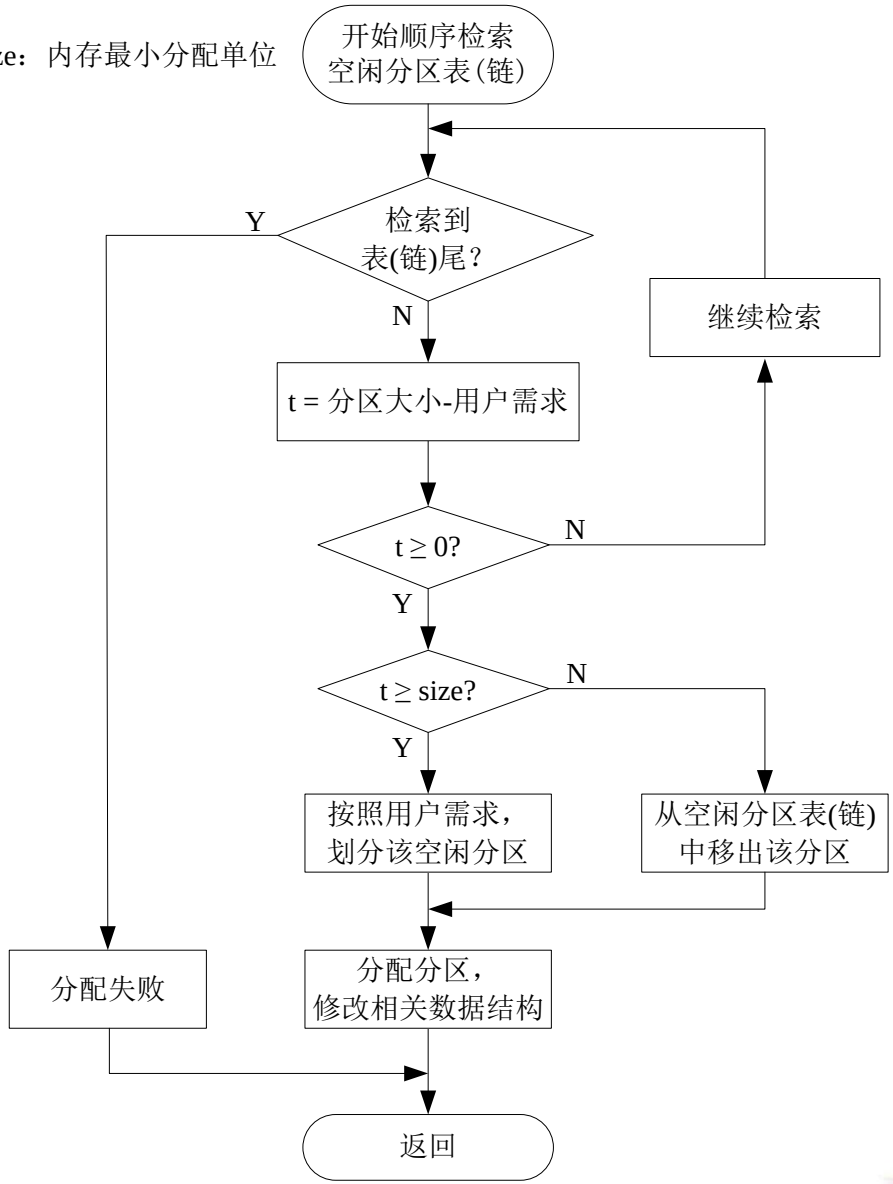
**2.分区分配算法**

## 3.分区分配操作

size：内存最小分配单位

### 1) 分配内存

图　内存分配流程



## 2) 回收内存

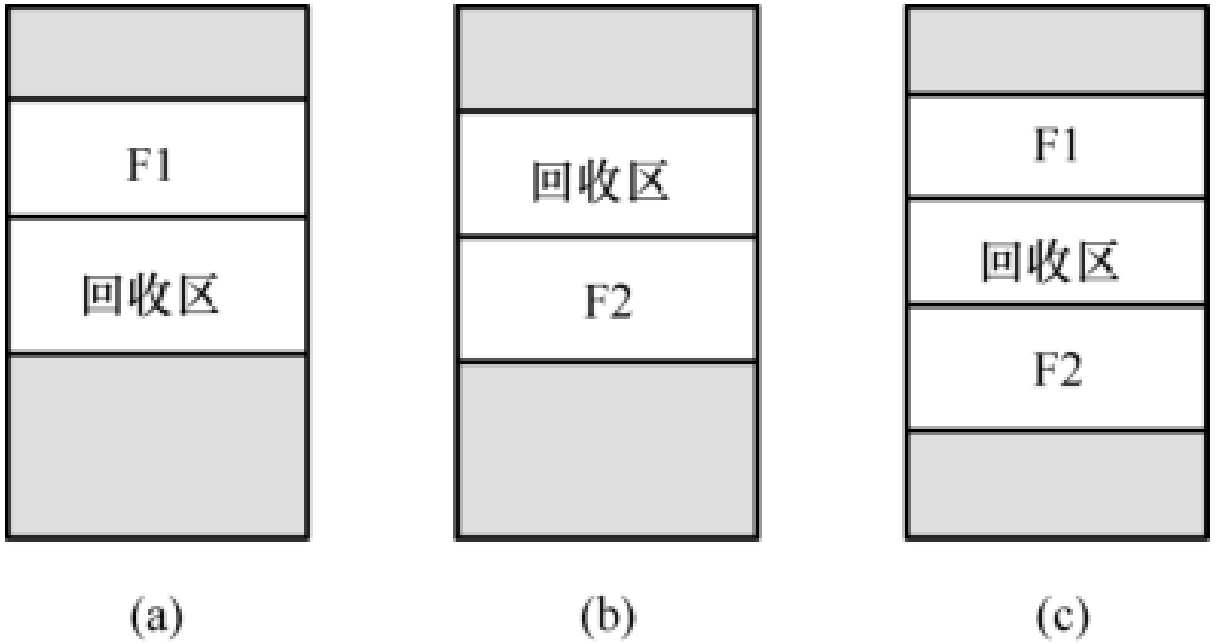



图 6-8　内存回收时的情况

# 3．动态分区分配算法

- 顺序搜索分配算法
  - 空闲分区表的表头或链首开始依次搜索，直到找到第一个满足要求的分区或直到最后也未找到。
  - 分为首次适应算法、循环首次适应算法、最佳适应算法和最坏适应算法，区别主要在于各个空闲分区在空闲分区表或空闲分区链中排列的先后顺序不同。
  - 分区数目很多时，顺序搜索分配算法的效率会降低。
- 索引搜索分配算法
  - 在大、中型操作系统中通常会采用索引搜索分配算法，如伙伴系统。

# 基于顺序搜索的动态分区分配算法

## （1）首次适应算法FF

要求空闲分区链以地址递增的次序链接。在进行内存分配时，从链首开始顺序查找，直至找到一个能满足其大小要求的空闲分区，从中划出一块内存空间分配给请求者，余下的留在空闲链中。

优点：优先利用内存中低址部分的空闲分区，保留了高址部分的大空闲区。

缺点：低址部分不断划分，留下难以利用的小空闲分区；每次查找从低址部分开始，增加查找开销。

**（2）循环首次适应算法，该算法是由首次适应算法演变而成的。**

为进程分配内存时，不再每次从链首开始查找，而是从上次找到的空闲分区的下一个空闲分区（查询指针）开始查找，直至找到第一个满足要求的空闲分区，从中划出一块分配给作业。

优点：使内存中的空闲分区分布得更均匀，减少查找开销。

缺点：缺乏大的空闲分区。

## （3）最佳适应算法。

“最佳”：每次为作业分配内存时，总是把既能满足要求、又是最小的空闲分区分配给作业，避免“大材小用”。

所有空闲分区按其大小以递增的顺序形成一空闲区链，这样第一次找到的满足要求的空闲区，必然是最优的。

优点：较大的空闲分区可以被保留。

缺点：留下很多难以利用的小空闲区（外碎片）。

## （4）最坏适应算法WF（Worst Fit）

未分配分区按照大小从大到小排列。分配时顺序选择当前最大区。

优点：基本上不留下小的空闲分区。

缺点：较大的空闲分区不被保留。

- 若一个8MB的进程提出了分配请求，下图给出了最佳适应算法、首次适应算法、循环首次适应算法和最坏适应算法的分配结果。

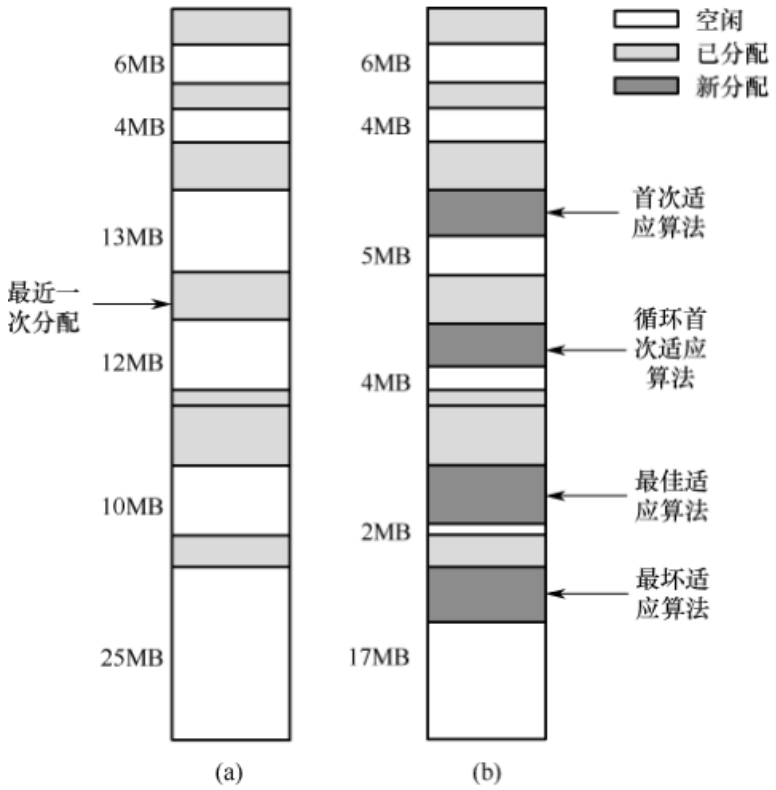



图 6-9　顺序搜索分配算法分区示意图

# 基于索引搜索的动态分区分配算法

## 1. 伙伴系统

- 固定分区方式限制了活动进程的数目，当进程大小与空闲分区大小不匹配时，内存空间利用率很低。
- 动态分区方式算法复杂，回收空闲分区时需要进行分区合并等，系统开销较大。
- 伙伴系统方式是对以上两种方式的一种折衷方案。
- 伙伴系统规定，无论已分配分区或空闲分区，其大小均为2的$k$次幂，$k$为整数，$1 \leqslant k \leqslant m$，其中：$2^1$表示分配的最小分区大小，$2^m$表示分配的最大分区大小，通常$2^m$是整个可分配内存的大小。

假设系统的可用空间容量为$2^m$个字，则系统开始时，整个内存区是一个大小为$2^m$的空闲分区。在系统运行过程中，由于不断划分，可能会形成若干个不连续的空闲分区，将这些空闲分区根据分区的大小进行分类，对于每一类具有相同大小的所有空闲分区，单独设立一个空闲分区双向链表。这样，不同大小的空闲分区形成了$k(0 \leqslant k \leqslant m)$个空闲分区链表。

当需要为进程分配一个长度为$n$的存储空间时，首先计算一个$i$值，使$2^{i-1}<n\leqslant 2^{i}$，然后在空闲分区大小为$2^{i}$的空闲分区链表中查找。若找到，即把该空闲分区分配给进程。否则，表明长度为$2^{i}$的空闲分区已经耗尽，则在分区大小为$2^{i+1}$的空闲分区链表中寻找。若存在$2^{i+1}$的一个空闲分区，则把该空闲分区分为相等的两个分区，这两个分区称为一对伙伴，其中的一个分区用于分配，而把另一个加入分区大小为$2^{i}$的空闲分区链表中。若大小为$2^{i+1}$的空闲分区也不存在，则需要查找大小为$2^{i+2}$的空闲分区，若找到则对其进行两次分割：第一次，将其分割为大小为$2^{i+1}$的两个分区，一个用于分配，一个加入到大小为$2^{i+1}$的空闲分区链表中；第二次，将第一次用于分配的空闲区分割为$2^{i}$的两个分区，一个用于分配，一个加入到大小为$2^{i}$的空闲分区链表中。若仍然找不到，则继续查找大小为$2^{i+3}$的空闲分区，以此类推。由此可见，在最坏的情况下，可能需要对$2^{k}$的空闲分区进行$k$次分割才能得到所需分区。

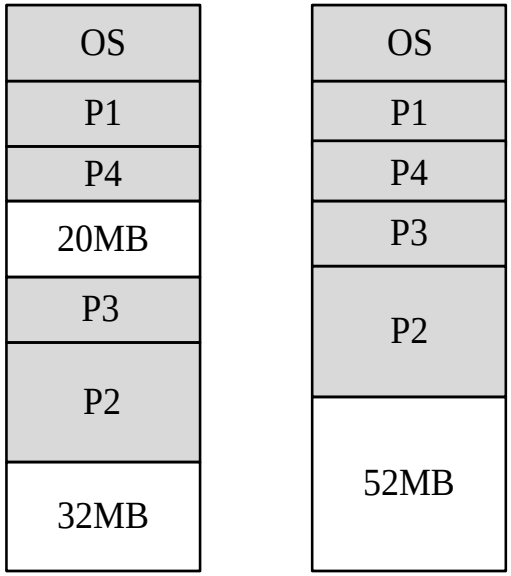
一次回收也可能要进行多次合并，如回收大小为$2^i$的空闲分区时，若事先已存在$2^i$的空闲分区时，则应将其与伙伴分区合并为大小为$2^{i+1}$的空闲分区，若事先已存在$2^{i+1}$的空闲分区时，又应继续与其伙伴分区合并为大小为$2^{i+2}$的空闲分区，依此类推。

在伙伴系统中，其分配和回收的时间性能取决于查找空闲分区的位置和分割、合并空闲分区所花费的时间。该算法在回收空闲分区时，需要对空闲分区进行合并，所以其时间性能比分类搜索算法差，但比顺序搜索算法好，而其空间性能则远优于分类搜索法，比顺序搜索法略差。

# 6.2.3 动态重定位分区存储管理

## 1. 动态重定位的引入

- 如图，如果现在有一个大小为**48MB**的进程申请装入内存，由于无法为该进程分配一个连续的空闲分区，此种情况下，动态分区存储管理分配失败，该进程无法装入。




(1) 紧凑前　　(2) 紧凑后

紧凑是指移动内存中原来的进程，将分散的多个空闲分区拼接成一个大的空闲分区的技术。
也称为“拼接”。

## 2. 动态重定位的实现

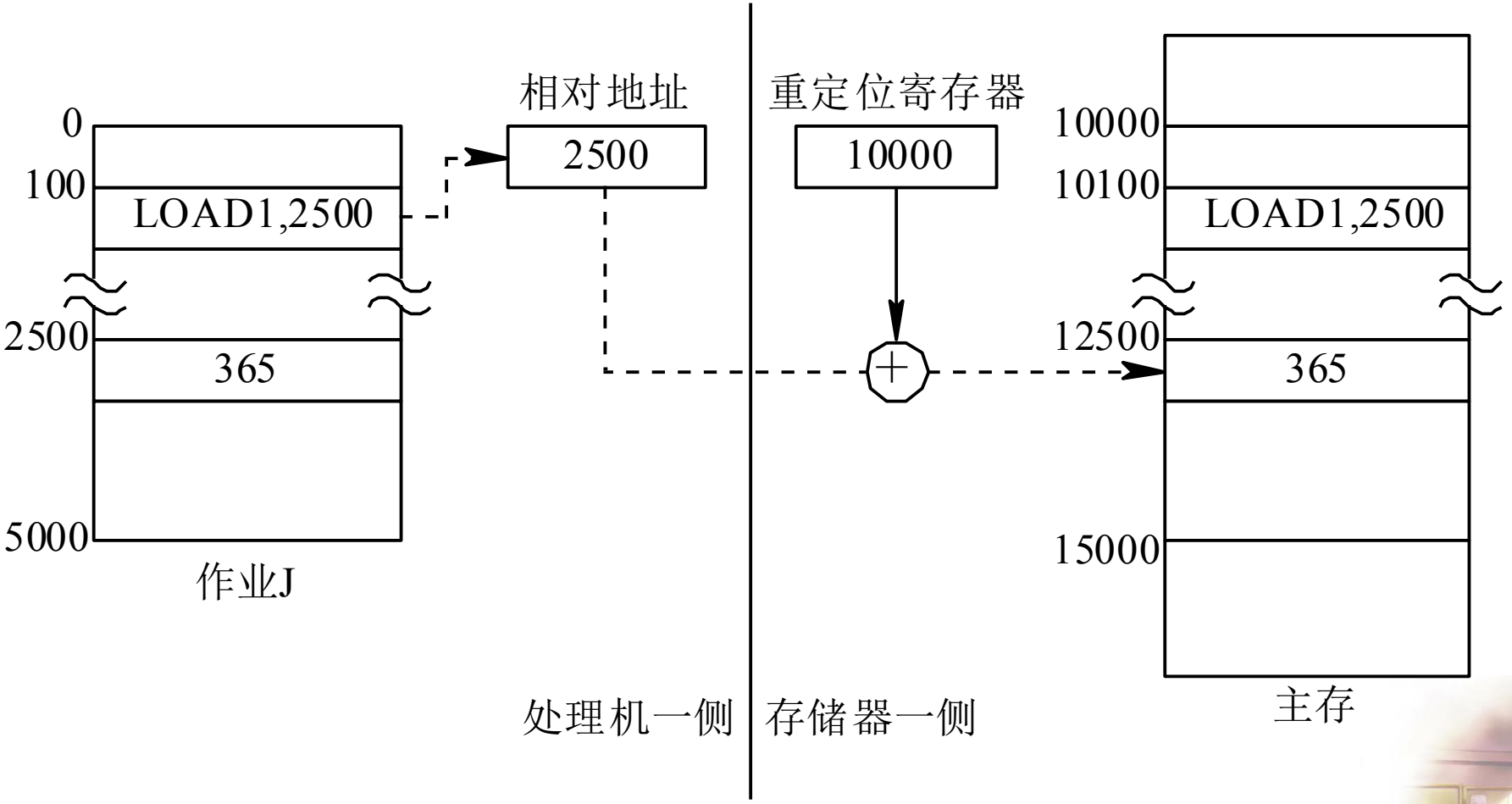
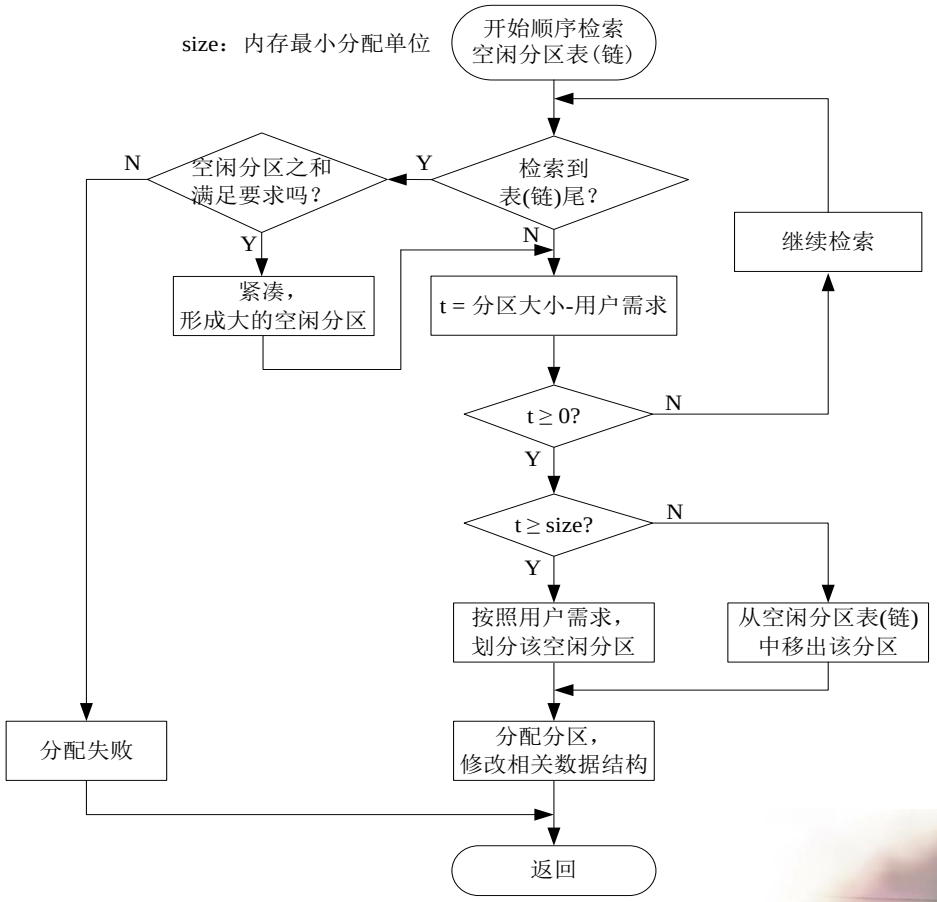



图　动态重定位示意图

## 3. 动态重定位分区分配算法

- 动态重定位分区与动态分区存储管理的分配算法基本上相同，差别仅在于：
  - 前者增加了紧凑功能，即在找不到足够大的空闲分区来满足用户需求时，实施紧凑技术。



# 6.3 内存扩充技术

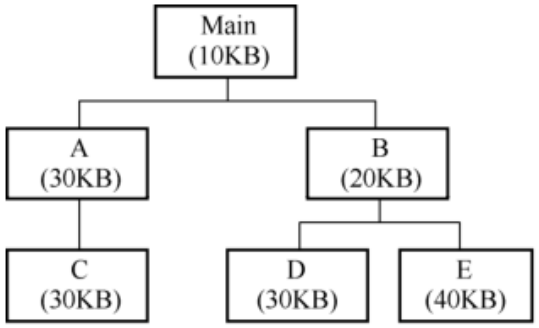
在基本存储管理系统中，当并发运行的多个进程的长度之和大于内存可用空间时，多道程序设计的实现就会遇到很大的困难。内存扩充技术就是借助大容量的辅存在逻辑上实现内存的扩充，以解决内存容量不足的问题。

介绍两种虚拟内存出现之前的内存扩充技术。

- 6.3.1 覆盖技术
- 6.3.2 交换技术

# 6.3.1 覆盖技术

- 用于早期的单用户系统。
- 覆盖技术：
  - 将一个程序按照逻辑结构划分成若干程序段。
  - 将不同时执行的程序段组成一个覆盖段小组。
  - 一个覆盖段小组的多个程序段可装入同一块内存区域，这个内存区域称为覆盖区。




(a) 一个程序的调用结构

| 覆盖 | 空间分配 |
|---|---|
| Main | 10KB |
| A和B | 30KB |
| C、D和E | 40KB |

(b) 覆盖区分配情况

图 6-12　覆盖技术示例

# 6.3.1 覆盖技术

- 覆盖对用户是不透明的。为了提高覆盖的效果，用户在编写程序时就要精心安排好程序的覆盖结构，并用覆盖描述语言描述覆盖段小组。
- 覆盖描述语言可以写入独立的覆盖描述文件中，并和目标程序一起提交给系统，也可附加在源文件中一起编译。

# 6.3.2　交换技术

## 1. 交换技术的概念

- 交换技术是指把内存中暂时不能运行的进程或者暂时不用的代码和数据调到外存上，腾出足够的内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从外存调入内存。也称“对换”。
- 交换技术与覆盖技术比较：
  - 交换技术不要求用户给出程序段之间的覆盖结构。交换技术主要在程序或进程之间进行。
  - 覆盖技术主要发生在同一个程序或进程内部。而且覆盖技术只能覆盖那些与覆盖程序段无关的程序段。
- 交换技术最早用于单一连续分配系统解决多道程序运行的问题，后来加以发展用于分区存储管理。

# 6.3.2 交换技术


图 6-13　某系统实施交换技术的过程

借助交换技术，有限的内存空间装入运行了更多的作业。因此，交换技术提高了内存的利用率。

# 6.3.2 交换技术

2. 交换技术的类型

- 整体交换。以整个进程为单位在内存和外存之间进行交换，其目的是减轻内存负荷，广泛用于多道程序系统。处理机中级调度的核心就是交换技术。
- 部分交换。以进程的一部分为单位，如一个页或一个段，在内存和外存之间进行交换。部分交换是第7 章将要介绍的请求分页存储管理和请求分段存储管理的基础，其目的是支持虚拟存储器。

### 3. 对换空间的管理

连续分配方式

外存分为文件区（存放文件）和对换区（存放从内存换出的进程）。

离散分配方式

为了能对对换区中的空闲盘块进行管理，在系统中应配置相应的数据结构，以记录外存的使用情况。其形式与内存在动态分区中所用数据结构相似，可以用空闲分区表或空闲分区链。在空闲分区表中的每个表目中应包含两项，即对换区的首址及其大小，单位是盘块号和盘块数。

# 6.4　分页存储管理方式

内存连续分配的问题：

- 产生内零头或者外零头。
- 采用紧凑技术增加额外开销。

离散式内存分配 — 允许一个进程分配在不相连接的内存区域中，以利于提高存储效率。

## 一. 分页存储管理的基本思想

将用户程序的地址空间划分为大小相等的页面（页），将内存空间也划分为大小相等的物理块（页框），页和物理块的大小相等，页面的典型大小为**1KB**。一个作业的所有页面一次装入，但可不连续存放。

## 1、 页面和物理块

分页存储管理，是将一个进程的逻辑地址空间分成若干个大小相等的片，称为页面或页。相应地，把内存空间分成与页面大小相同的若干个存储块，称为(物理)块或页框(frame)。为进程分配内存时，以块为单位将进程中的若干个页分别装入到多个可以不相邻接的物理块中。

由于进程的最后一页经常装不满一块而形成了不可利用的碎片，称之为“页内碎片”。

## 2. 页表

通常放在内存中

- 系统为每个进程创建一个数据结构，用于记录页与分配的物理块的对应关系，这个数据结构称为页表。
- 页表的作用是实现从页号到物理块号的地址映射。

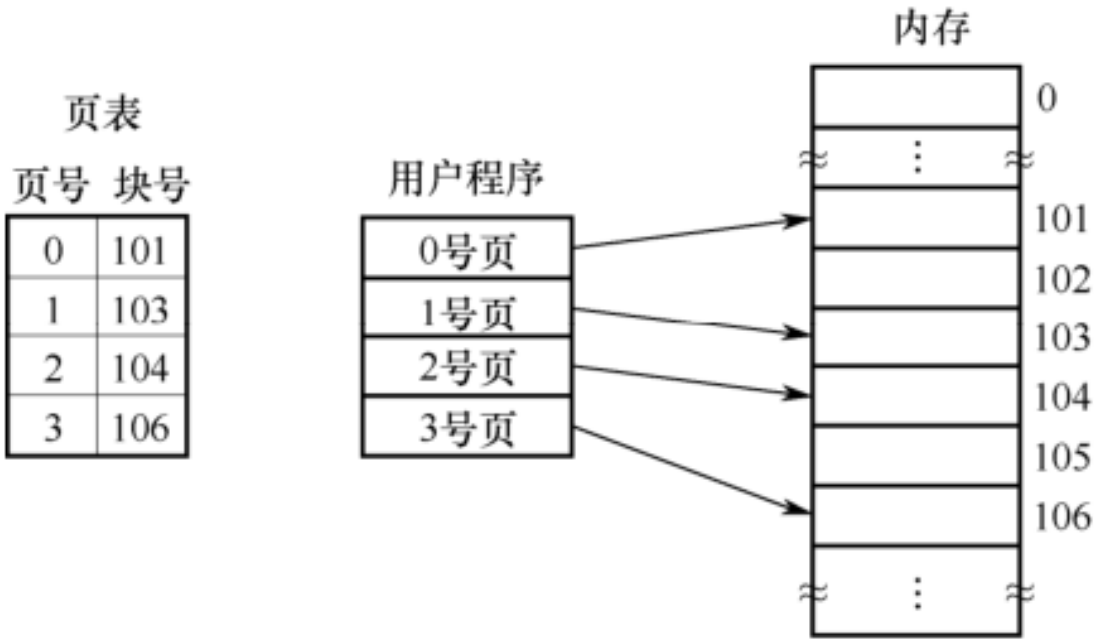



图 6-14　页表的作用

3. 页面大小

由硬件决定

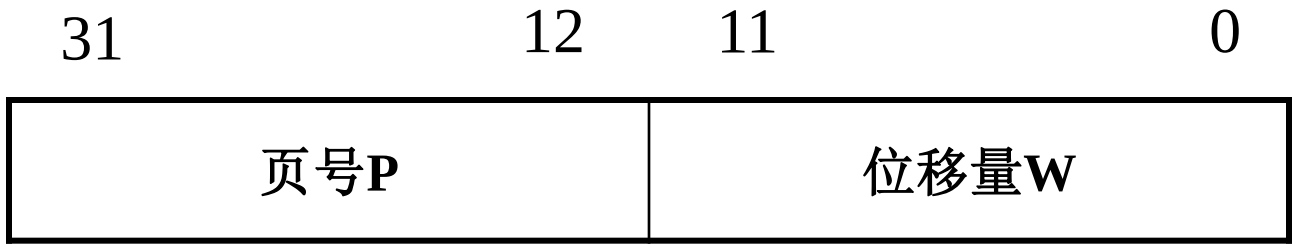
应适中

页面若太小，一方面可使内存碎片减小，有利于提高内存利用率；但另一方面也会使每个进程占用较多的页面，从而导致页表过长，占用大量内存；此外，还会降低页面换进换出的效率。

如果页面较大，虽然可减少页表的长度，提高页面换进换出的速度，但却又会使页内碎片增大。

## 4. 地址结构

分页地址中的地址结构如下：




对某特定机器，其地址结构是一定的。若给定一个逻辑地址空间中的地址为A，页面的大小为L，则页号P和页内地址d可按下式求得：

$$P = INT\left[\frac{A}{L}\right]$$

$$d = [A]MODL$$

## 二.地址变换机构

### 1.基本的地址变换机构

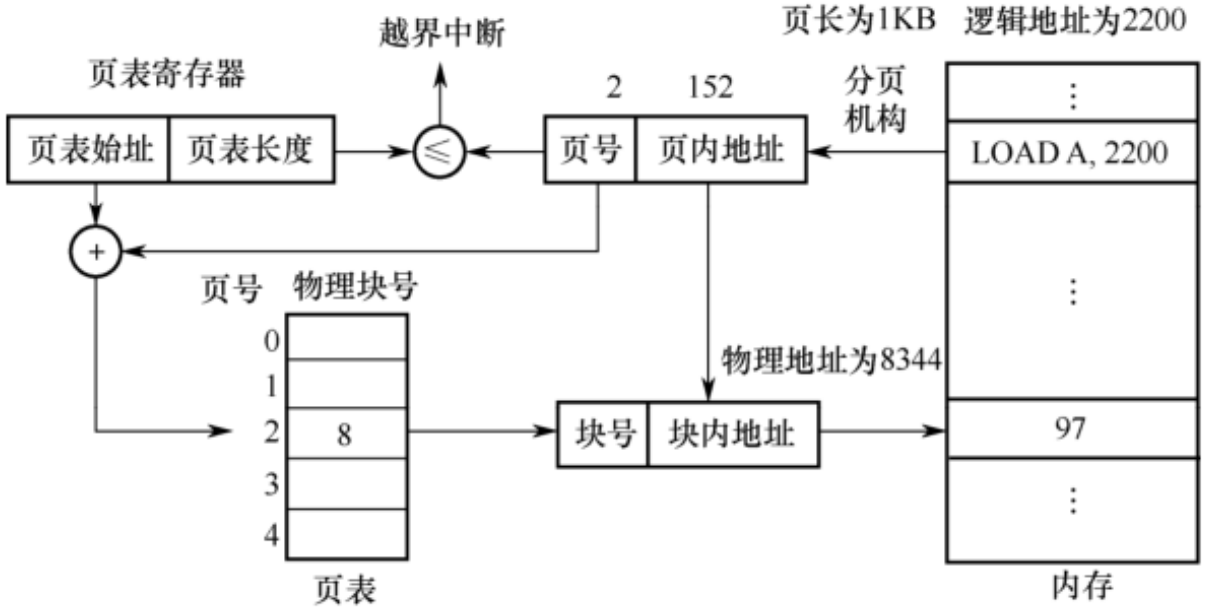



图 6-17　基本地址变换机构

- 整个地址变换过程由硬件自动完成。
- 可以看出，页表实际上是动态重定位技术的一种延伸。

## 2. 具有快表的地址变换机构

### (1)为什么要设置快表?

由于页表存放在内存，每存取一个数据，CPU 要访问两次内存，这导致读写内存的指令处理速度降低了近一半。分页存储管理采用离散分配方式提高内存空间利用率是以降低内存访问指令执行效率为代价的，这个代价是高昂的。

为了加快地址变换的速度，在地址变换机构中建立一个高速缓冲存储器，也称为联想存储器(Associative Memory)或快表。

### (2)具有快表的地址变换机构

在Cache中，存放现行进程的页表副本，即记录现行进程中最常用的页表项。

页表表目是在访问内存的过程中按需要动态装入快表的，当进程切换时，快表被清零。

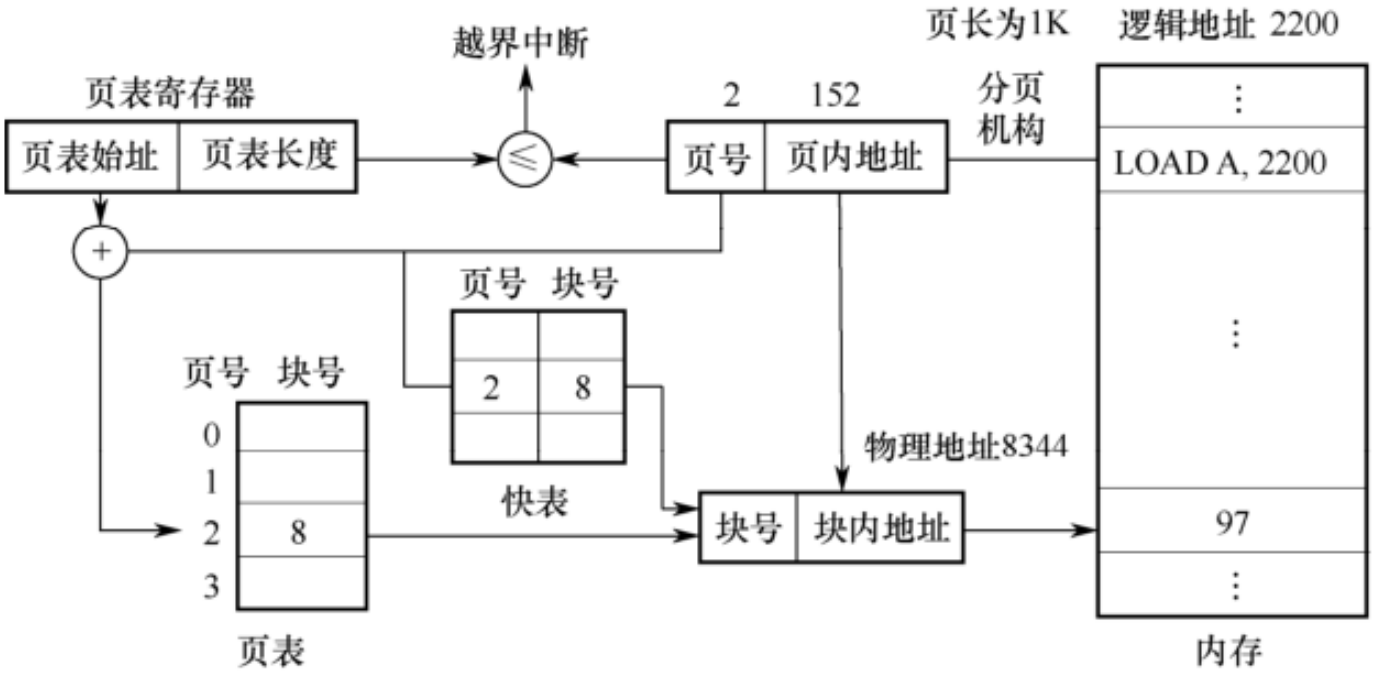



图 6-18　具有快表的地址变换机构

- 有统计数据表明，从快表中找到所需页表项的概率可达90%以上。这样，因增加地址变换机构而造成的速度损失可减少到10%以下。

## 三. 两级页表、多级页表和反置页表

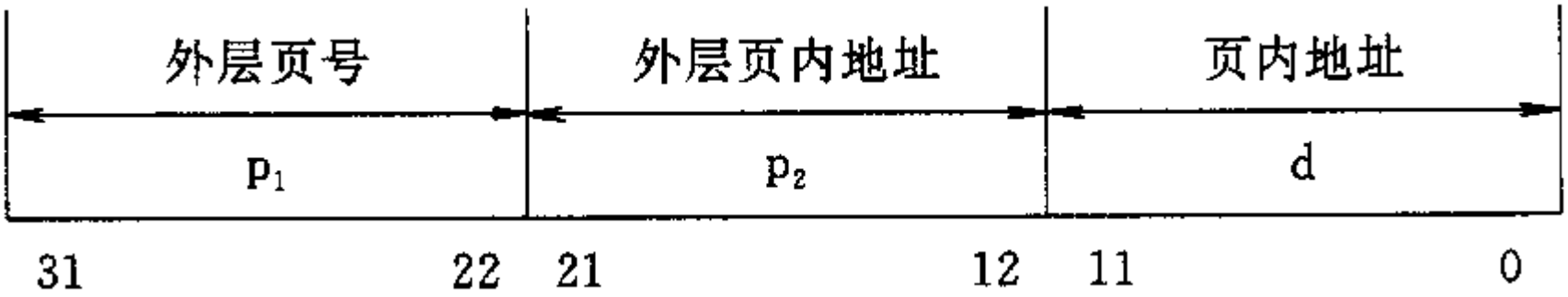
- 问题：页表很大，有可能一个物理块已经装不下页表。
- 解决方法：
  - 采用离散方式存储页表，即对页表进行分页，加载到多个物理块来存储，这种方法就是两级页表或多级页表。
  - 只将当前需要的部分页表项调入内存，其余的页表项放在外存，当有需要时再调入内存，这种方法需要借助第7 章的虚拟存储技术来实现。

设置多级页表是为了解决逻辑空间过大，所造成的页表过大，占有太大的连续内存空间的问题。

## 1. 两级页表(Two-Level Page Table)

页表内再分页，分为外层页表和内层页表。

逻辑地址结构可描述如下：



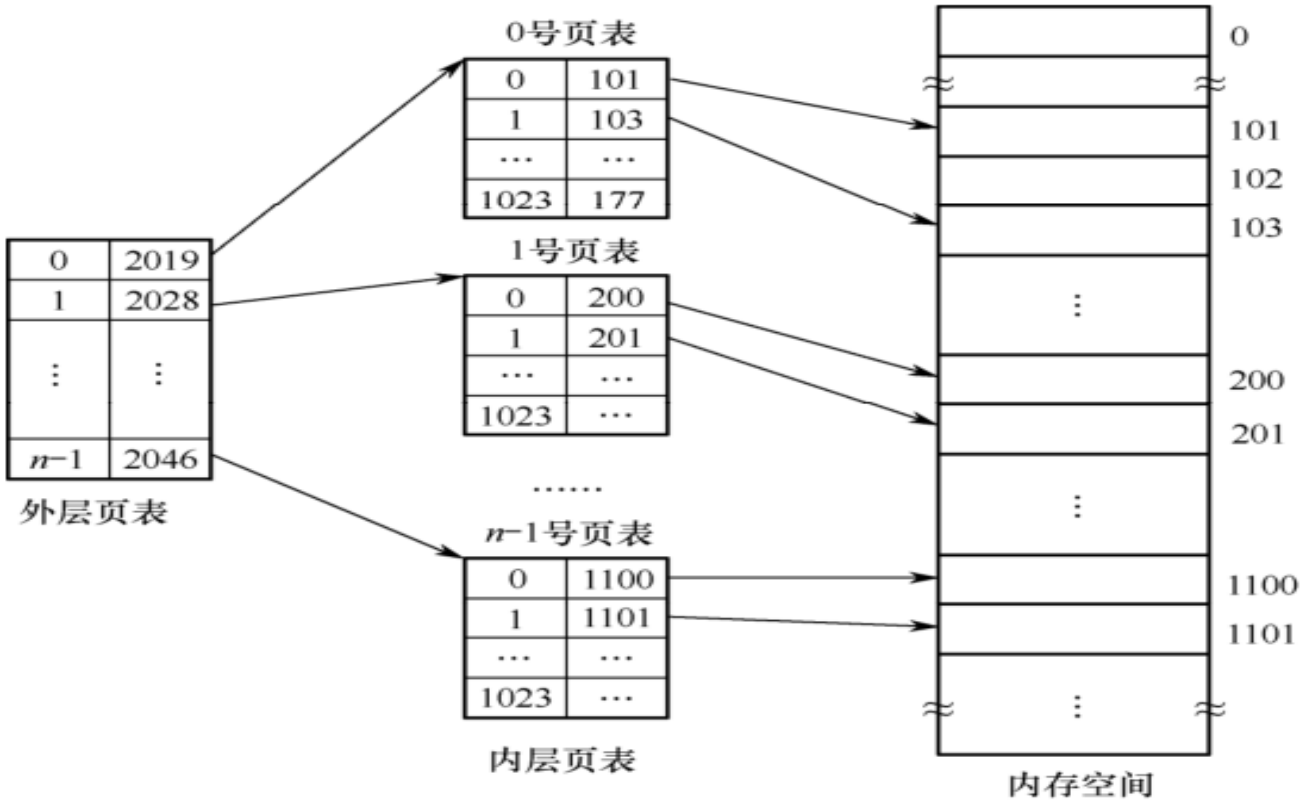



图 6-19　两级页表结构

- 外层页表：记录页表的每个分页在内存的存放情况，实现原始页表的离散存放。
- 内层页表：原始页表的各个分页。

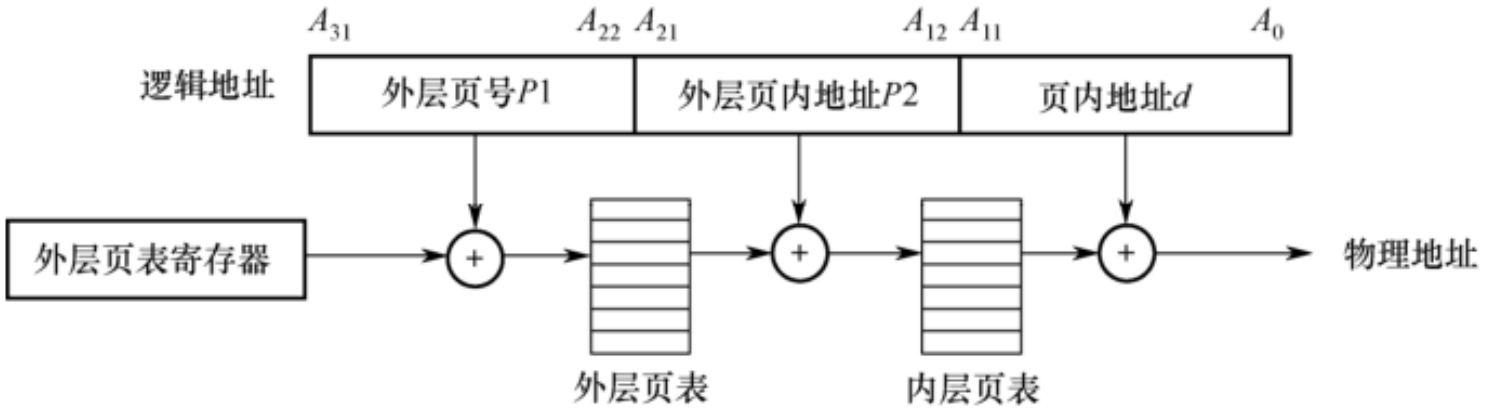



图 6-20　两级页表的逻辑地址结构及地址变换过程

- 实现上增设一个外层页表寄存器，用于存放外层页表的始址。由地址变换结构将逻辑地址分页，即逻辑地址变为（**P1, P2, d**）的形式。
- 显然，采用两级页表后，访问内存的一条指令在执行时需要访问内存三次，更有必要采用具有快表的地址变换机构。

## 2. 多级页表

对于32位的机器，采用两级页表结构是合适的；但对于64位的机器，如果页面大小仍采用4 KB即$2^{12}$B，那么还剩下52位，假定仍按物理块的大小($2^{12}$位)来划分页表，则将余下的40位用于外层页号。此时在外层页表中可能有4096G个页表项， 要占用16384 GB的连续内存空间。因此，必须采用多级页表。

对于64位的计算机，如果要求它能支持$2^{64}$(=1844744 TB)规模的物理存储空间，则即使是采用三级页表结构也是难以适应的；在当前的实际应用中也无此必要。

解决的方法有下面两种：

- 近两年推出的64 位操作系统把可直接寻址的存储器空间减少为45位，可利用三级页表结构实现分页存储管理。
- 采用反置页表。

## 3.反置页表

- 常规页表：按页号进行排序，页表项中的内容是物理块号。
- 反置页表：按物理块号排序，页表项中的内容是该物理块装入的页号P及所属进程的标识符pid。
  - 整个系统只有一个页表，每个物理块在表中有唯一对应的表项。
  - 反置页表为每一个物理块设置一个页表项，并将它们按物理块的编号排序，其中的内容则是页号和其所隶属进程的标识符。在进行地址变换时，根据进程标识符和页号，去检索反置页表。如果检索到与之匹配的页表项，则该页表项的序号i便是该页所在的物理块号。

## 四. 分页存储管理的特点

分页存储管理便于多道程序设计，实现了离散存储，提高了内存利用率，不必像动态分区分配那样执行紧凑操作。但分页存储管理仍然存在如下严重缺点：

(1) 采用动态地址映射会增加计算机成本和降低处理机的速度。

(2) 各种表格要占用一定容量的内存空间，而且还要花费一部分处理机时间来建立和管理这些表格。

**(3) 虽然消除了大量碎片，但每个作业的最后一页一般都有不能充分利用的空白区（页内碎片）。**

例如，假定页面大小4 KB，如果某作业需要9 KB内存，则该为它分配三个物理块，最后一块的3 KB空间被浪费了。如果减少页面大小，使页面大小为2 KB，则应分配5个存储块，其中有1 KB的内存被浪费了。减少页面大小，可以减少内存的浪费，但页表的长度又增加了，这也是一个矛盾。

**(4) 不支持动态增长。**

**(5) 不方便进行信息共享。**

**(6) 存储扩充问题仍未得到解决。**

# 6.5　基本分段存储管理

## 一.分段存储管理方式的引入

分页存储管理无法满足用户在编程和使用上的需要，主要表现在：

1) 不方便编程

2) 不便于信息共享

3) 不便于信息保护

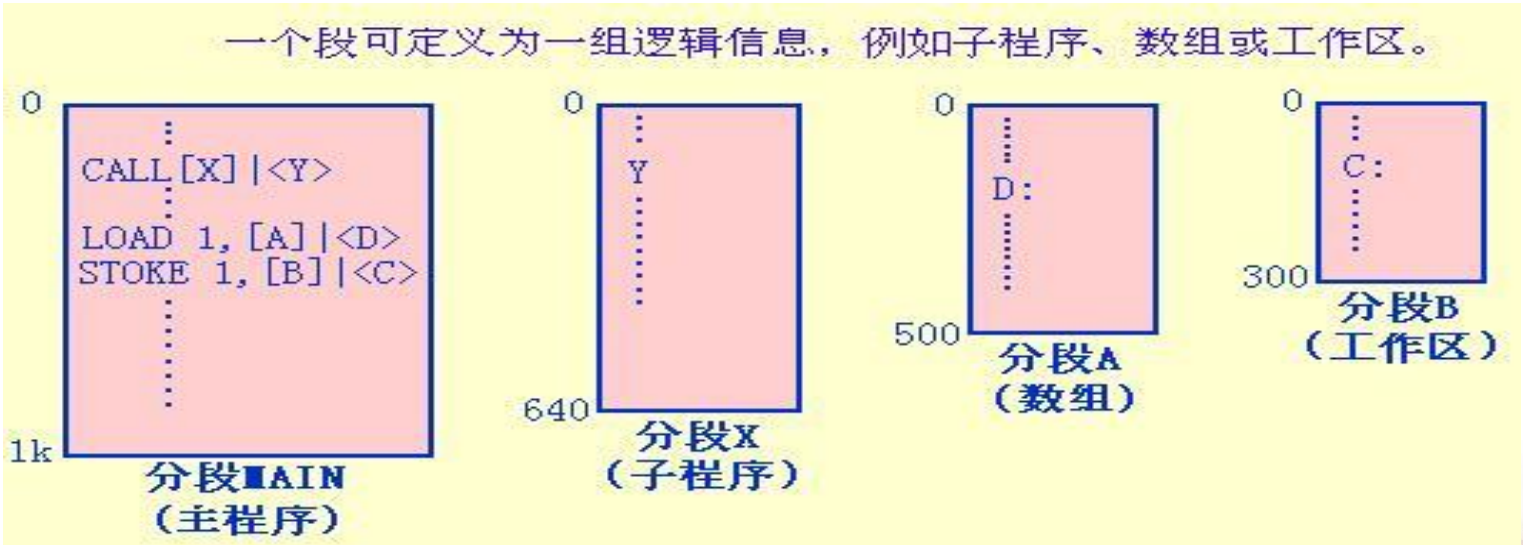
4) 不允许动态增长

5) 不便于动态链接

# 二. 分段系统的基本原理

## 1. 分段

页是信息的物理单位，段（segments）是信息的逻辑单位，在逻辑上是一组整体的信息，每个段都是从0开始相对编址的。



- 程序分段后，逻辑地址由两部分组成：
  - 段号s（或段名）和段内地址d。
  - 如指令“LOAD A, [1] | <160>;”的含义是将1 段中160 单元的值读出，给寄存器A 赋值。
- 在分段存储管理中，作业的逻辑地址空间是二维的。

## 2. 段表

- 系统为每个进程建立一个段表，用来记录每个逻辑段在内存的存放始址和相关信息。每个段对应一个段表项，包含段号、段长、基址等信息。
- 段表实现了从逻辑段到物理内存的映射。

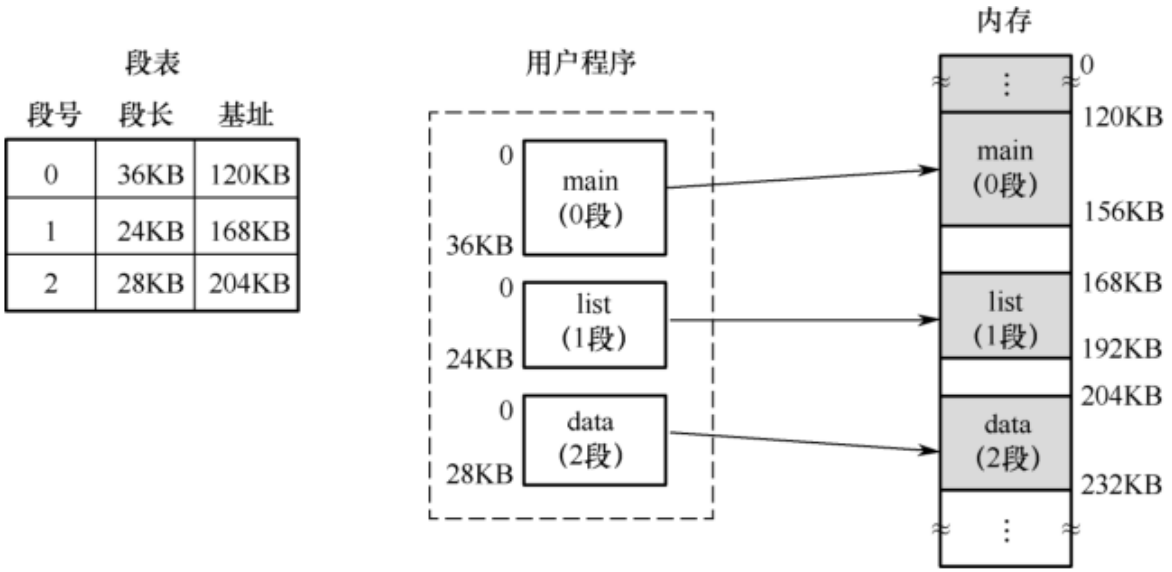



图 6-21　利用段表实现地址映射

# 3.分段地址变换过程

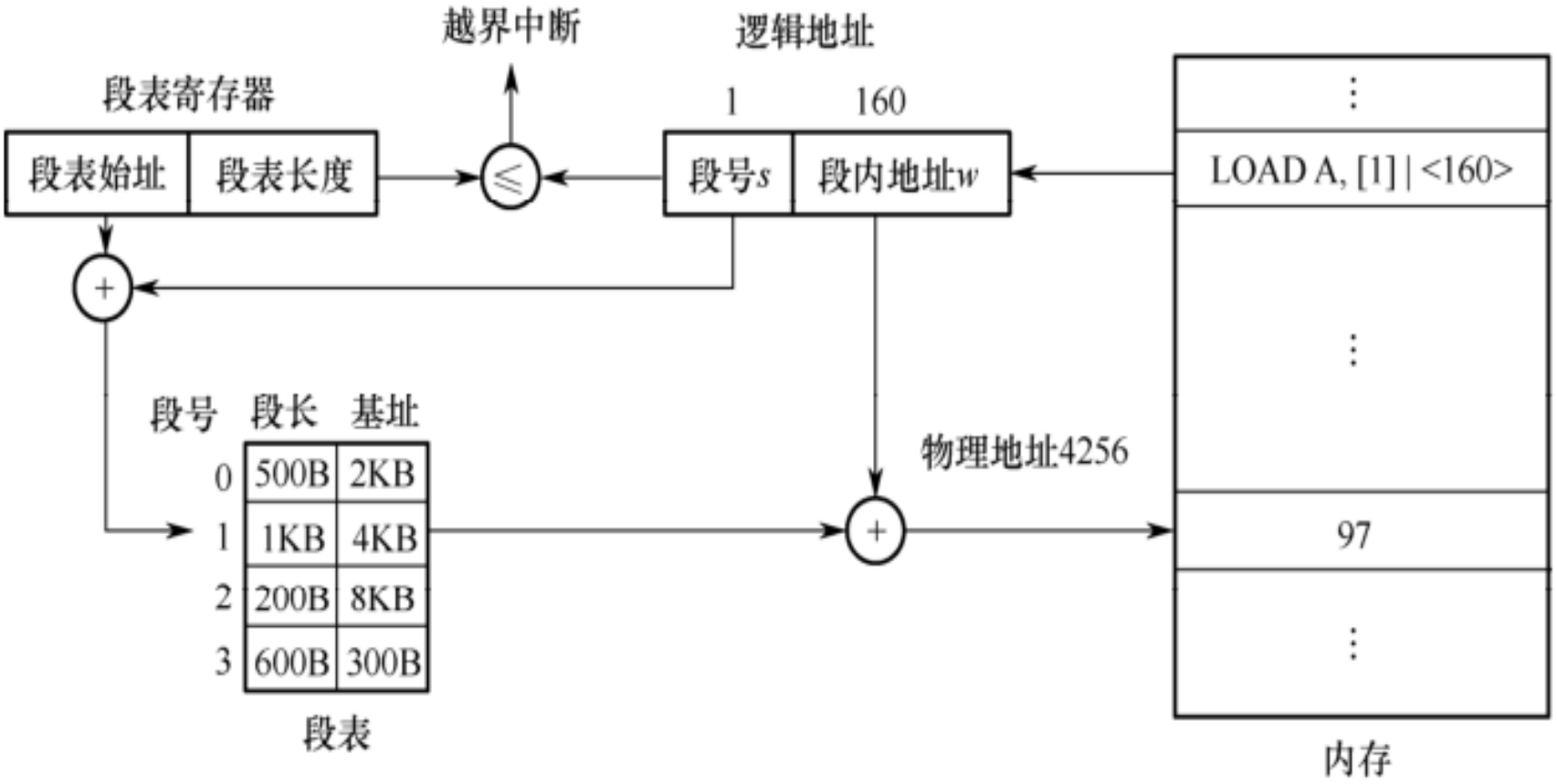



图 6-22　分段存储管理的地址变换过程

**4. 分页和分段的主要区别**

**(1)** 页是信息的物理单位，分页是为实现离散分配方式，以消减内存的外零头，提高内存的利用率。或者说，分页仅仅是由于系统管理的需要而不是用户的需要。段则是信息的逻辑单位，它含有一组意义相对完整的信息。分段的目的是为了能更好地满足用户的需要。

(2) 页的大小固定且由系统决定，由系统把逻辑地址划分为页号和页内地址两部分，是由机器硬件实现的，因而在系统中只能有一种大小的页面；而段的长度却不固定，决定于用户所编写的程序，通常由编译程序在对源程序进行编译时，根据信息的性质来划分。

(3) 分页的作业地址空间是一维的，即单一的线性地址空间，程序员只需利用一个记忆符，即可表示一个地址；而分段的作业地址空间则是二维的，程序员在标识一个地址时，既需给出段名， 又需给出段内地址。

# 5.信息共享

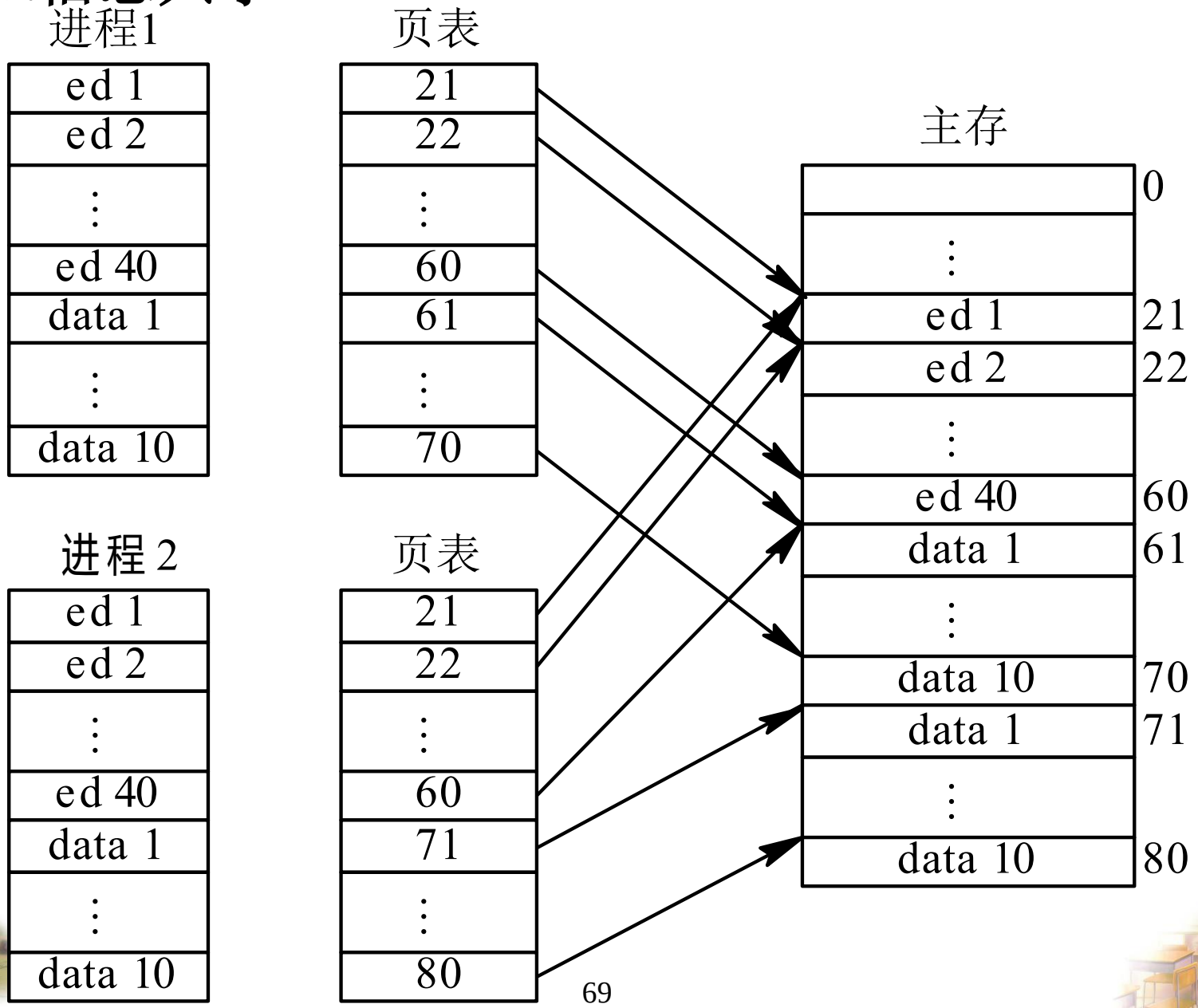



图　分页系统中共享editor的示意图

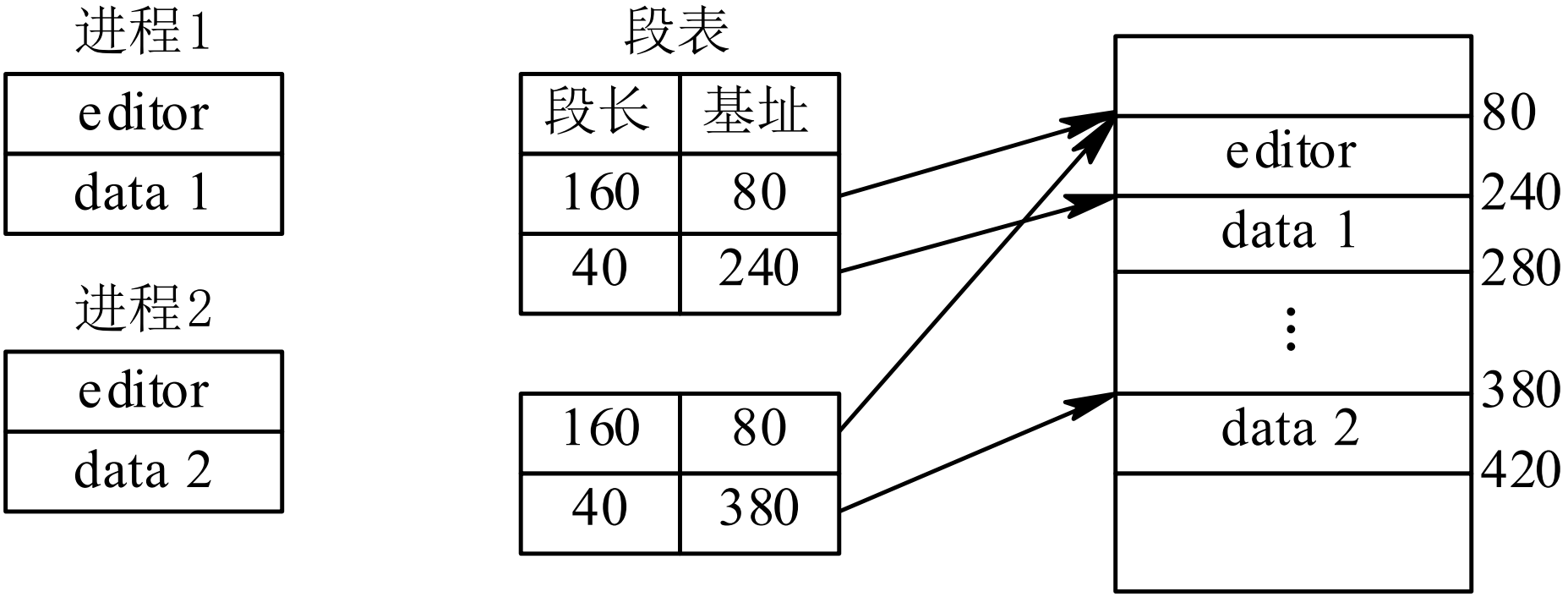



**图　分段系统中共享editor的示意图**

# 6.6 段页式存储管理方式

- 分段存储管理，虽然段与段之间可以离散存储，但同一个段必须连续存储，即要为每个段找一个连续的存储区，若找不到连续的存储区，则分配失败。因此，
- 问题：一个段是否可以离散存储？
  - 将分页存储管理和分段存储管理结合起来——段页式存储管理。

# 6.6 段页式存储管理方式

## 1. 基本原理

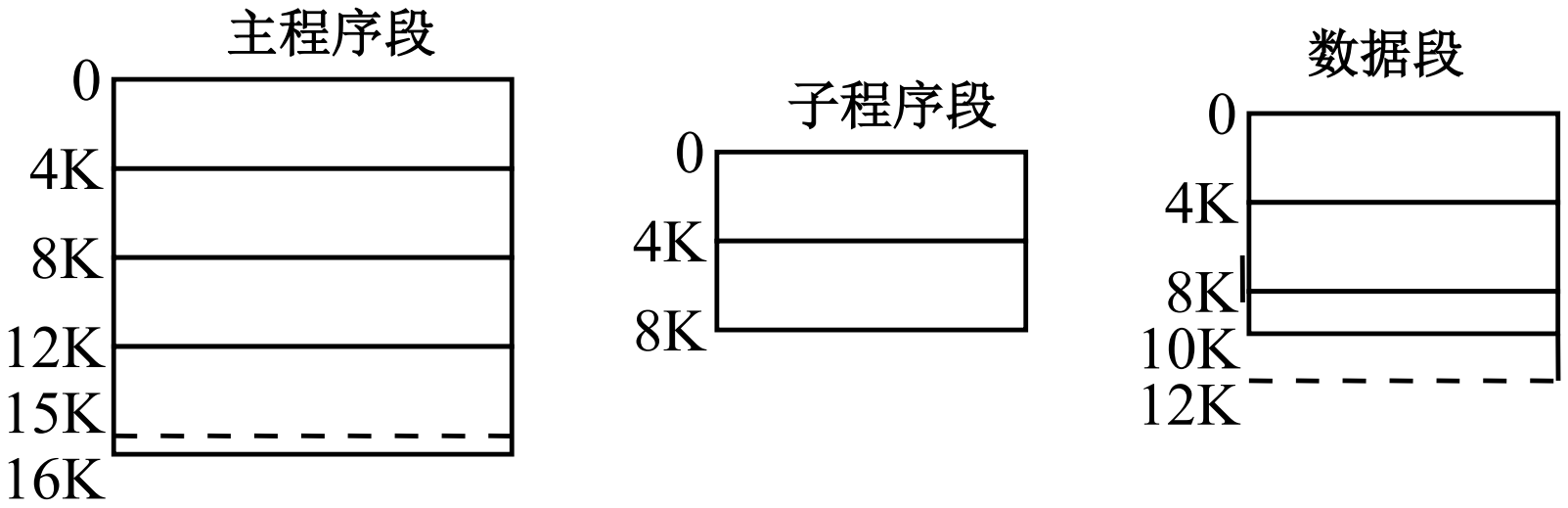
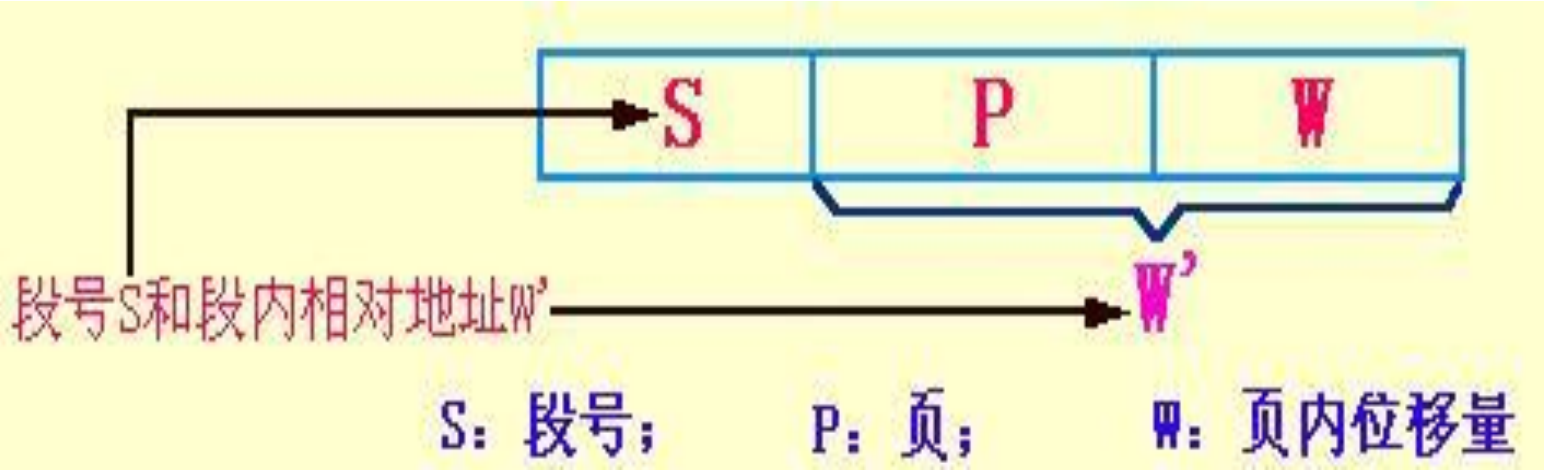



图　作业地址空间

- 段页式地址结构

实际上地址由（S，W’）⟶（S，P，W）




通常，分段由程序员完成，分页由系统自动完成。

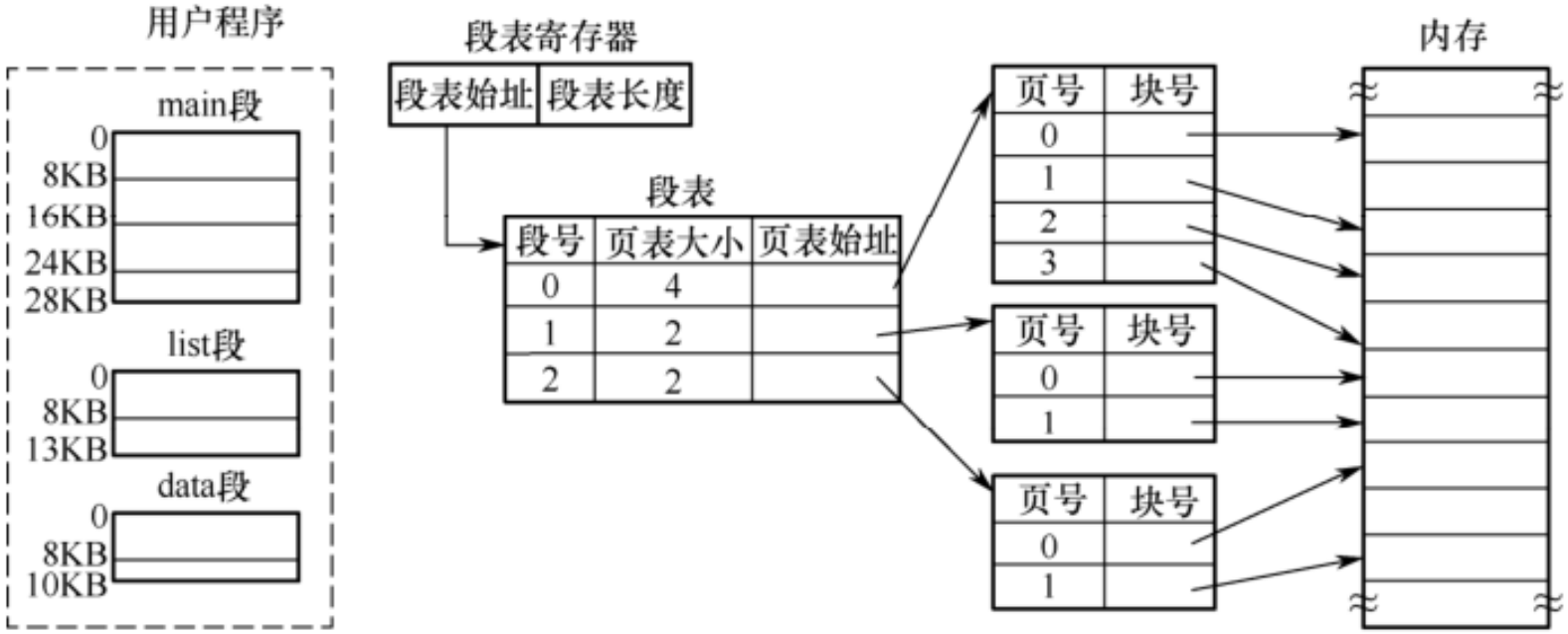



图 6-23　段页式存储管理下段表和页表的映射关系图

## 2. 地址变换过程

例如给定某个逻辑地址中，段号为2，段内地址为6015，若系统规定块大小为1 KB，则采用段页式管理，该逻辑地址表示：段号为2，段内页号为5，页内地址为895。其地址变换过程如图所示。

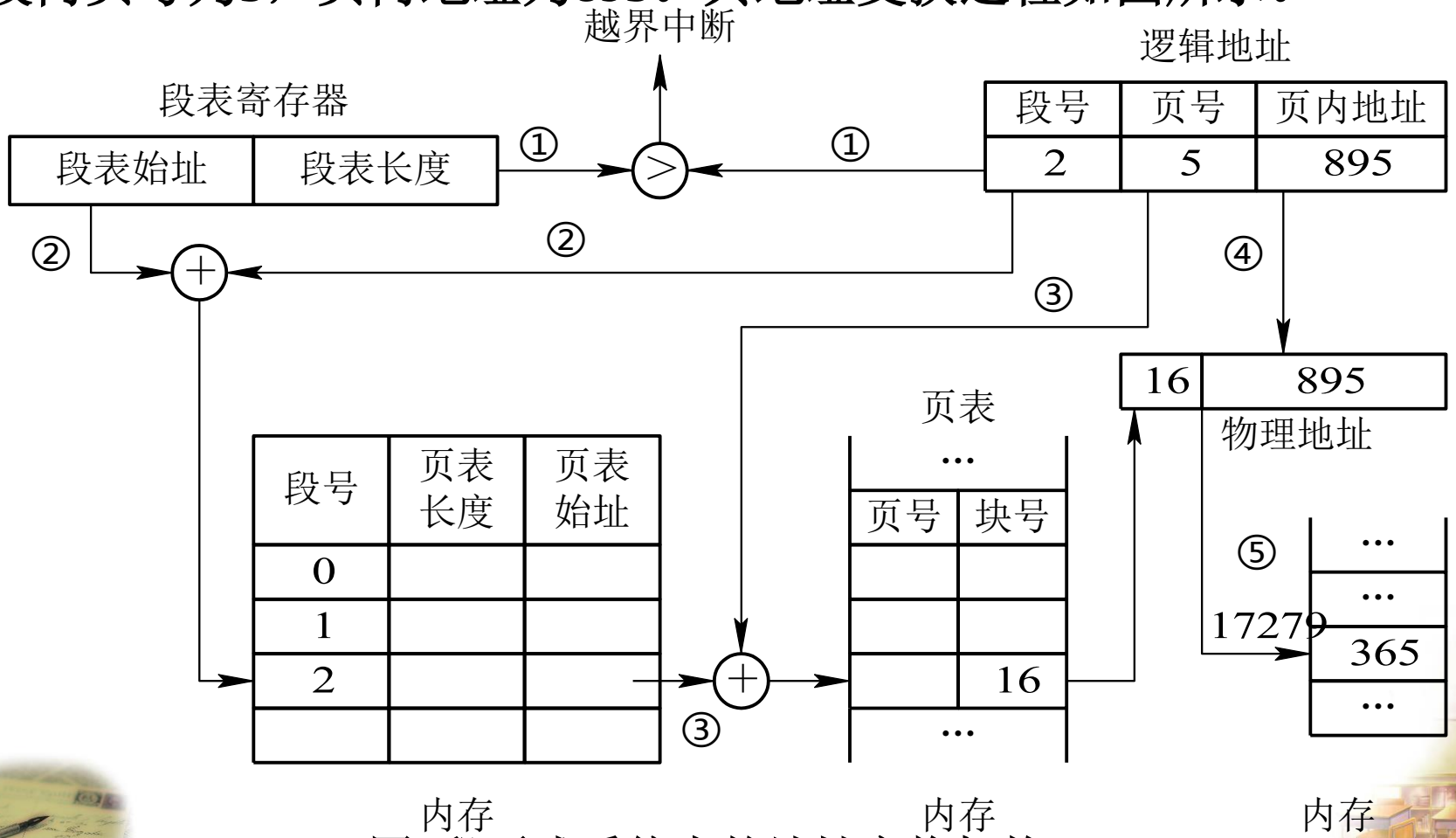



图 段页式系统中的地址变换机构

### 3. 段页式存储管理的评价

段页式存储管理方案保留了分段存储管理和分页存储管理的全部优点，满足了用户和系统两方面的需求。这种性能提升是有代价的，增加了硬件成本、系统的复杂性和管理上的开销，程序碎片在每个段都存在，段表、段内页表等表格占用相对较大的内存空间。

段页式存储管理技术对当前的大、中型计算机来说，算是最通用、最灵活的一种方法。

# 小　结

程序的几种装入、链接方式；

内存连续分配与回收算法；

交换概念；覆盖技术；

分页与分段机制、页表与段表、地址变换（应理解并熟悉）；

理解段页式管理。

# 比较几种实存管理方案的异同，并说明其特点。

| | 内存分配方法描述 | 优点 | 缺点 |
|---|---|---|---|
| 固定分区 | 在系统生成时，内存分为若干个分区。等于或小于分区大小的进程可以调入内存。 | 易于实现，开销小。 | 存在内部碎片。 |
| 动态分区 | 动态建立分区，所以分区大小与进程大小相同。 | 无内部碎片，内存利用率较高。 | 存在外部碎片，紧凑耗时。 |
| 分页存储 | 内存和进程都分成同样大小的若干页面和页，通过将进程的所有页调入内存的空闲块来将进程调入内存，这些内存中的页可以不连续。 | 无外部碎片。 | 少量内部碎片。 |
| 分段存储 | 每个进程按逻辑分为若干个段，通过将进程所有的段调入内存，这些内存中的段可不连续。 | 无内部碎片。 | 硬件支持，每个段的大小长度受内存可用区大小的限制。 |
| 段页存储 | 先将进程按逻辑分成若干个段，然后将内存和每个段分成相同大小的若干页面和页，然后将进程调入内存，这些内存中的段、页可不连续。 | 段可动态增长，便于共享，动态链接和控制访问。 | 很少的内部碎片，需要硬件支持，软件复杂。 |