存储器管理

- 存储器的层次结构
- 程序的装入和链接
- 连续分配方式
- 基本分页存储管理方式
- 基本分段存储管理方式

4.4 基本分页存储管理方式

引入分页式存储管理

- 分区存储管理的问题
  - 碎片：即使所有碎片容量的总和大于一个进程要求内存的容量，因其不连续而无法分配，浪费了内存资源。
  - 进程在分区中连续存放，故进程大小受分区大小和内存可用空间限制。
  - 紧凑开销大
  - 不利于程序段和数据段的共享。
- 原因
  - 一个作业的逻辑地址空间和物理地址空间要求占用连续区域。
- 解决思路
  - 如能避开这种连续性要求，即一个作业可以存放在不连续的存储空间，这样可以在不需要移动内存中原有的程序和数据就可以可以解决碎片问题。

4.4.1 页面与页表

- 1. 页面
  - 1) 页面和物理块
    - 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，并为各页加以编号，从0开始，如第0页、第1页等。
    - 把内存空间分成与页面相同大小的若干个存储块，称为(物理)块或页框(frame)， 也同样为它们加以编号，如0＃块、1＃块等等。
    - 在为进程分配内存时，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”。

4.4.1 页面与页表

- 2) 页面大小
  - 页面小：内存碎片减小， 有利于提高内存利用率，但使每个进程占用较多的页面，从而导致进程的页表过长，占用大量内存； 此外，还会降低页面换进换出的效率。
  - 页面大：减少页表，提高页面换进换出的速度；使页内碎片增大
  - 页长的划分与内存大小和内外存之间的传输速度有关；且页面大小应是2的幂，通常为512 B~8 KB

4.4.1 页面与页表

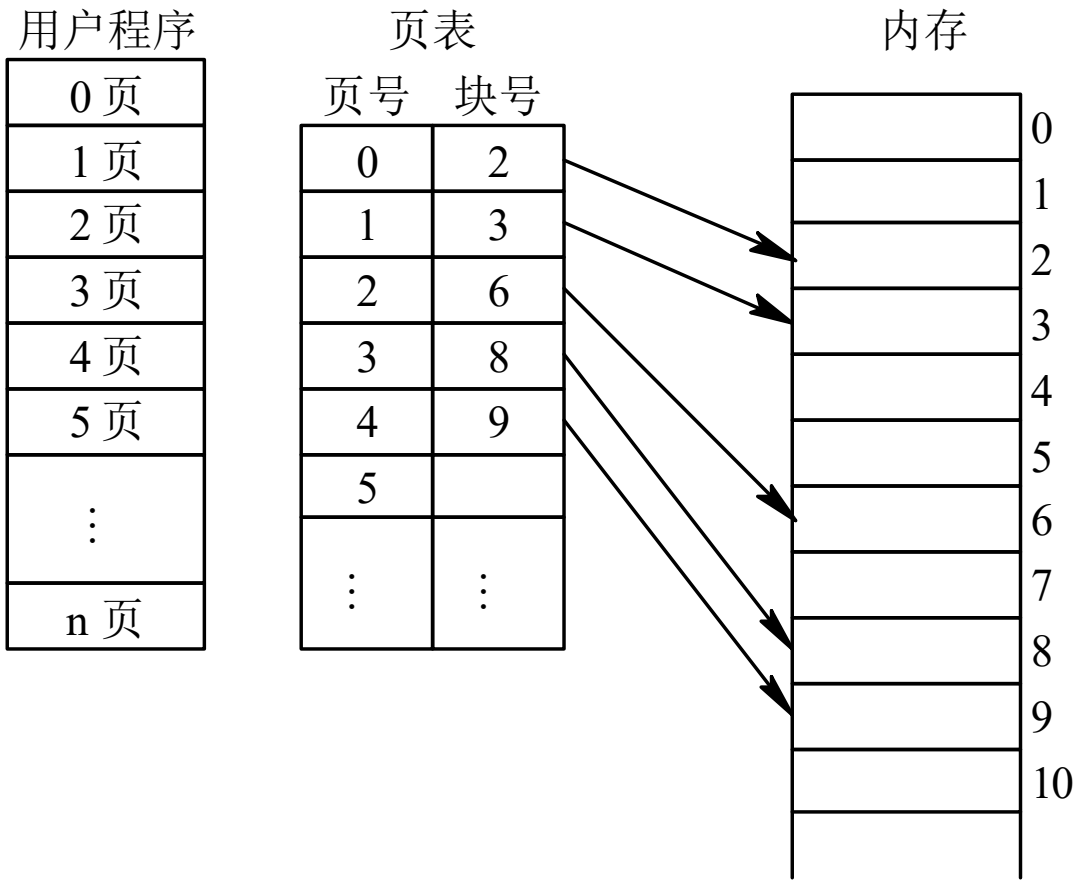
- 2. 地址结构
  - 页号、页内地址
  - 对某特定机器，其地址结构是一定的。若给定一个逻辑地址空间中的地址为A，页面的大小为L，则页号P和页内地址d可按下式求得：
  - 例：页面大小1kB，如A = 2170B；则，p=2、d=122

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

$$d = [A]MODL$$

4.4.1 页面与页表

- 3. 页表
  - 页面映象表：
    - 页——物理块号

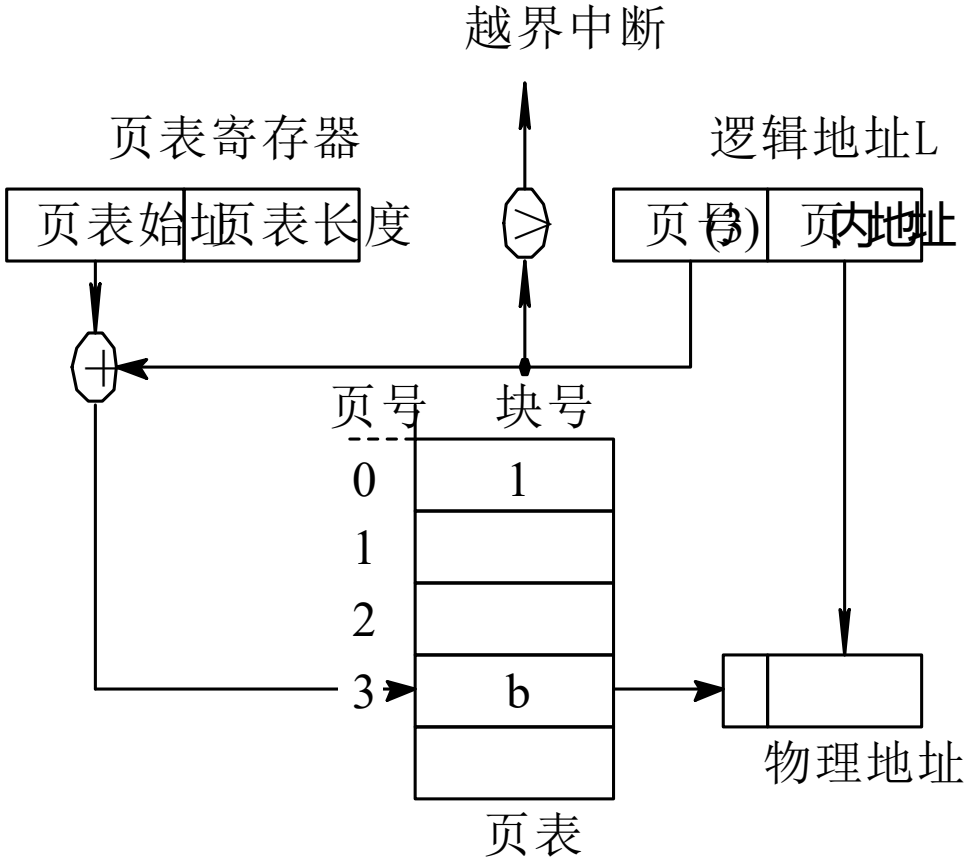


4.4.2 地址变换机构

- 逻辑地址到物理地址的转换
  - 页号——物理块号
- 1 基本地址变换机构，
  - 页表
    - 专门的寄存器
    - 内存中
  - 页表寄存器
    - 页表在内存中的开始地址和长度

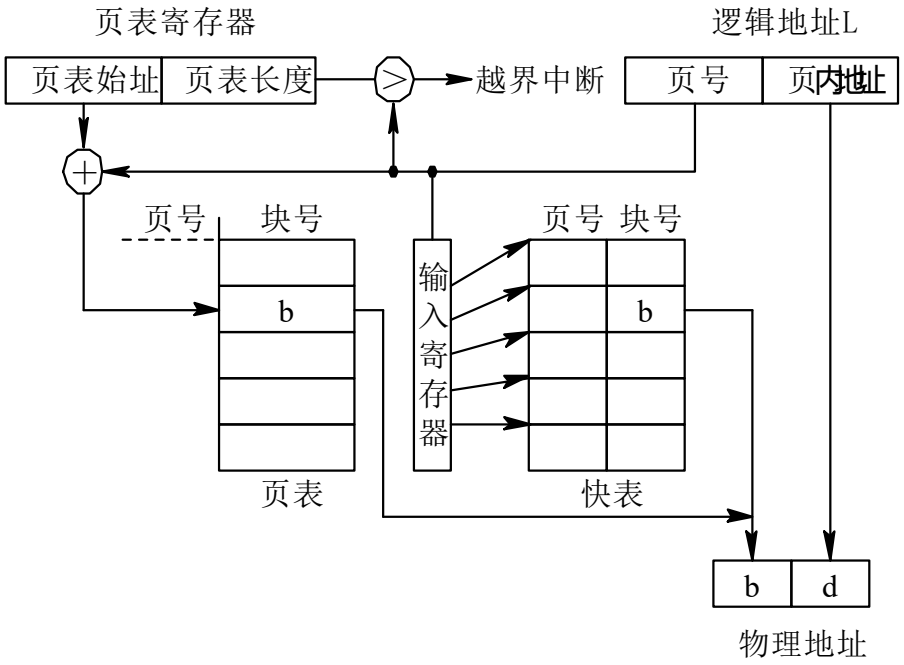
4.4.2 地址变换机构

- 1 基本地址变换机构
  - 变换过程
    - 有效地址（相对地址）→ 页号 + 页内地址
    - 检查页号是否越界
    - 通过控制寄存器得到页表位置；
    - 查页表，页号→物理块号
    - 物理块号×页面大小 + 块（页）内地址 = 物理地址



4.4.2 地址变换机构

- 2. 具有快表的地址变换机构
  - 提高速度
    - 取指（数据）需两次访存
    - 第一次访问页表计算出物理地址，第二次访问物理地址
- 快表
  - 高速联想存储器
  - 具有并行查找能力
  - 存放当前访问的页表项
  - 首先访问快表

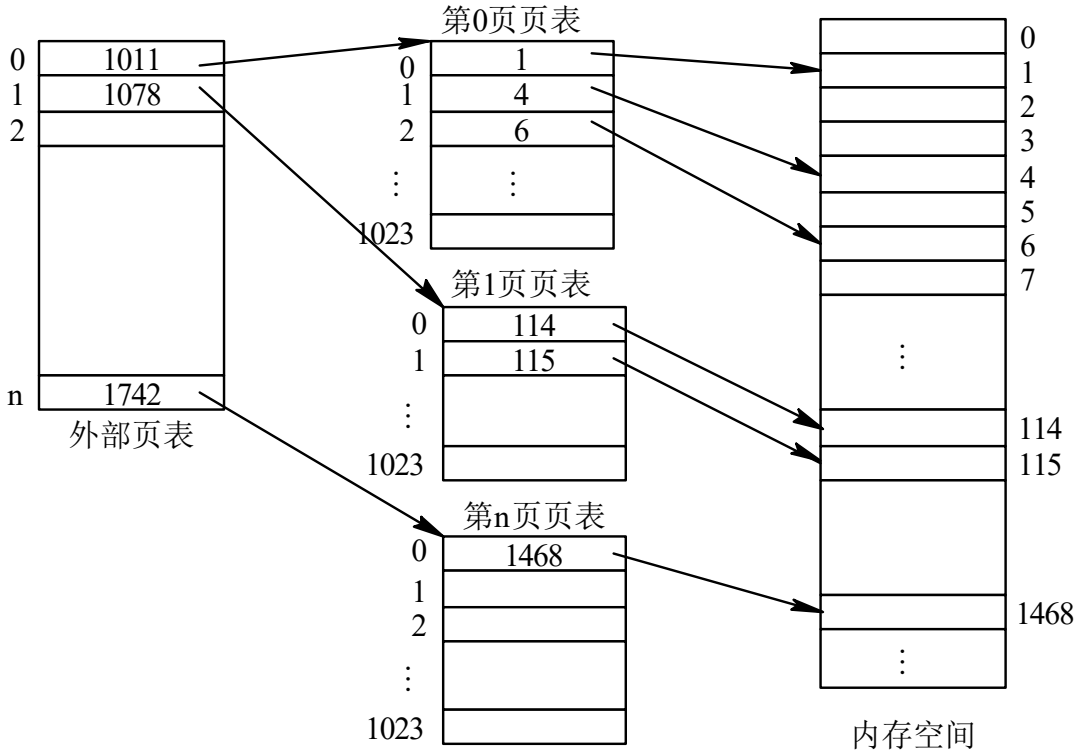


4.4.3 两级和多级页表

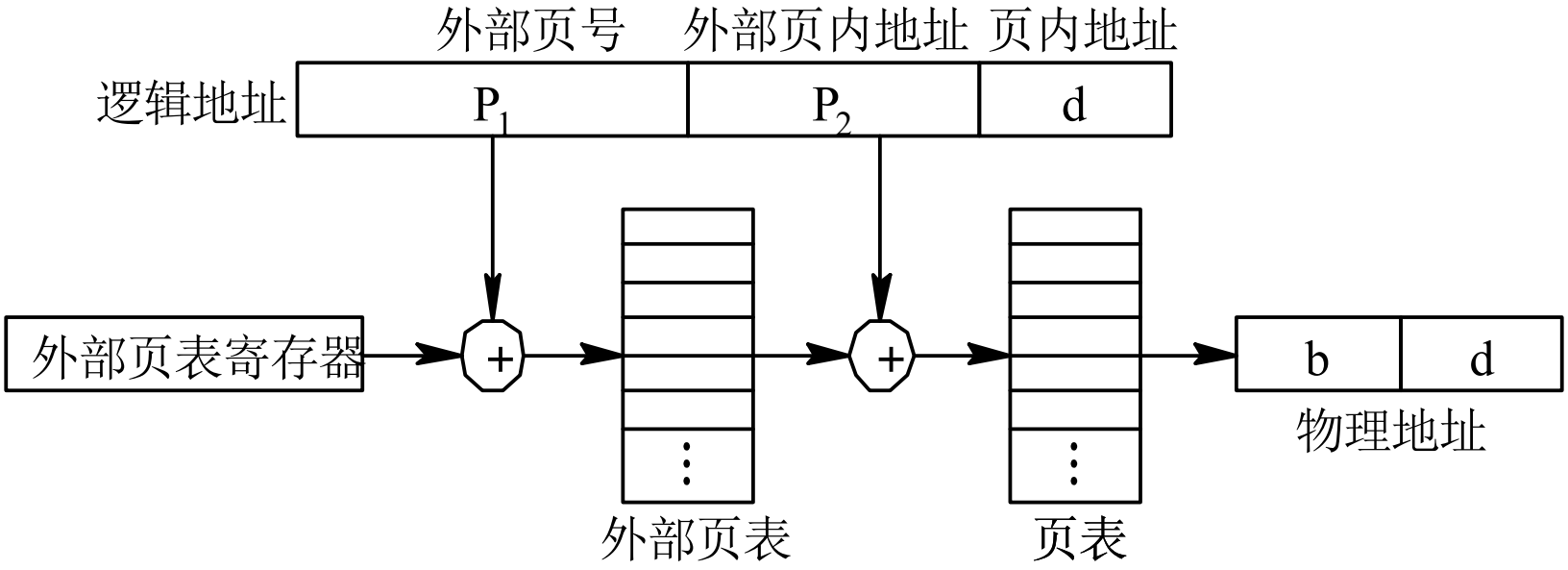
- 引入
  - 逻辑地址空间大导致页表非常大，要占用相当大的内存空间，并且，要求是连续的
  - 解决办法
    - ① 采用离散分配方式来解决难以找到一块连续的大内存空间的问题
    - ② 只将当前需要的部分页表项调入内存， 其余的页表项仍驻留在磁盘上，需要时再调入。

4.4.3 两级和多级页表

- 两级页表
  - 页表再分页，并将各页面放在不同的物理块
  - 为离散分配的页表建立页表，称为外层页表



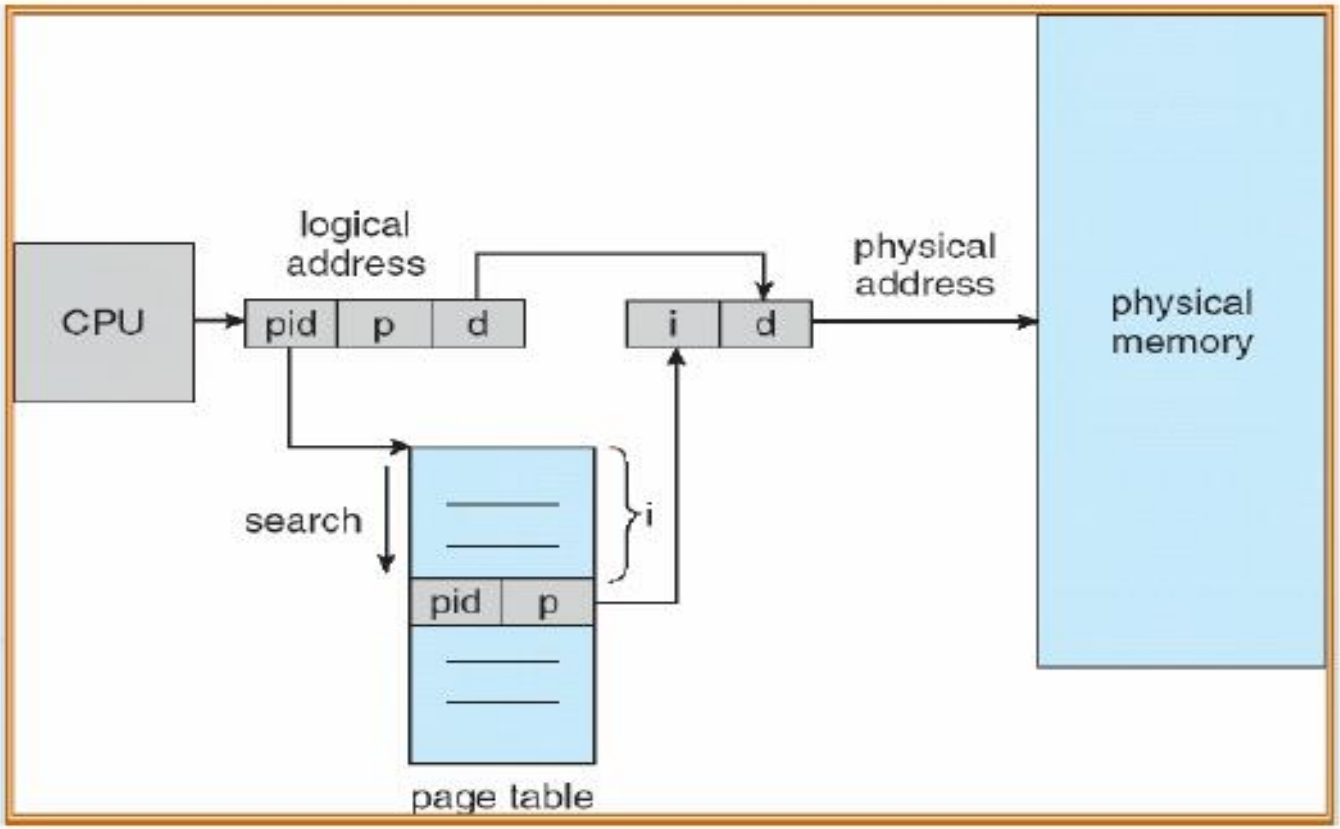
4.4.3 两级和多级页表



- 两级页表
  - 为减少内存空间
    - 外部页表放在内存
    - 部分页表在内存，部分页表在外存
- 多级页表

反置页表

Inverted Page Table Architecture



例1：有一系统采用页式存储管理，有一作业大小是8KB，页大小为2KB，依次装入内存的第7、9、A、5块，试将虚地址0AFEH，1ADDH转换成内存地址。

虚地址0AFEH

0000 1010 1111 1110

P＝1 W＝010 1111 1110

MR＝0100 1010 1111 1110

＝4AFEH

页号	块号
0	7
1	9
2	A
3	5

4.5 基本分段存储管理方式

分页：提高内存利用率

分段：方便用户（程序员）

4.5.1 分段存储管理方式的引入

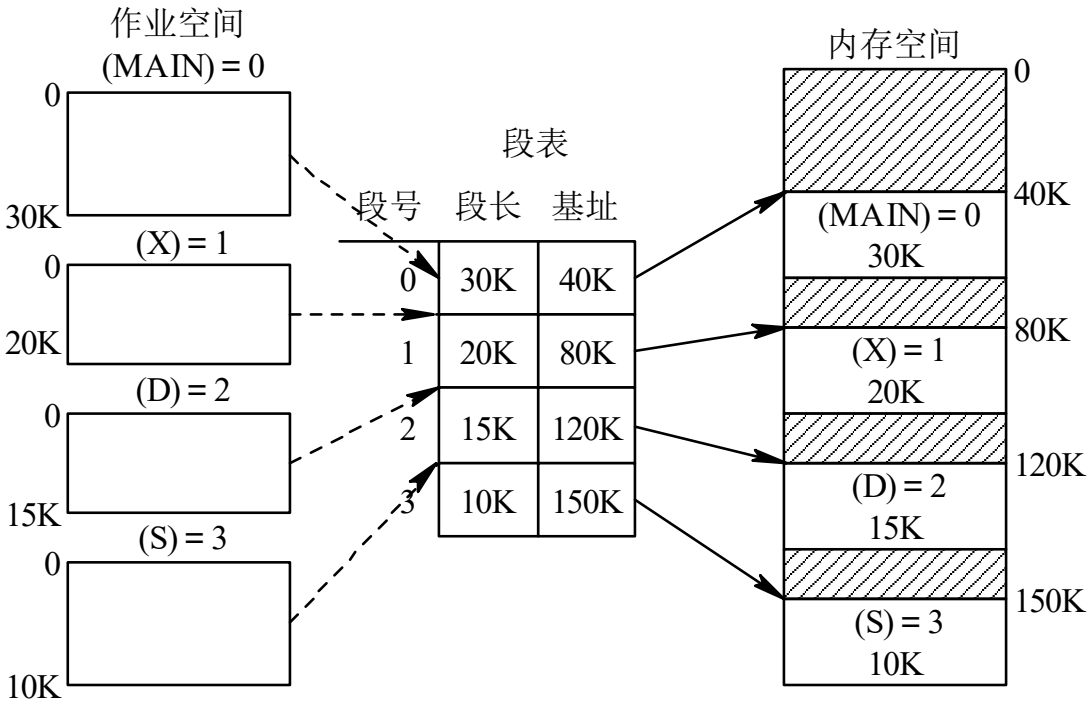
- 满足用户和程序员的下述一系列需要：
  - 1) 方便编程
    - 作业按照逻辑分段
  - 2) 信息共享
    - 页是按物理大小划分，无完整的信息
    - 段按逻辑划分，具有独立意义
    - 段适合共享
  - 3) 信息保护
  - 4) 动态增长
    - 事先不知道所用空间大小
  - 5) 动态链接
    - 运行时链接模块

4.5.2 分段系统的基本原理

- 1　分段
  - 把程序按内容或过程分成段，长度不固定
  - 每个段有自己的名字或编号；每段从0编址，占用连续的地址空间
  - 地址空间变为二维线性虚拟存储器。以段为单位分配内存。
    - 段号　＋　段内地址

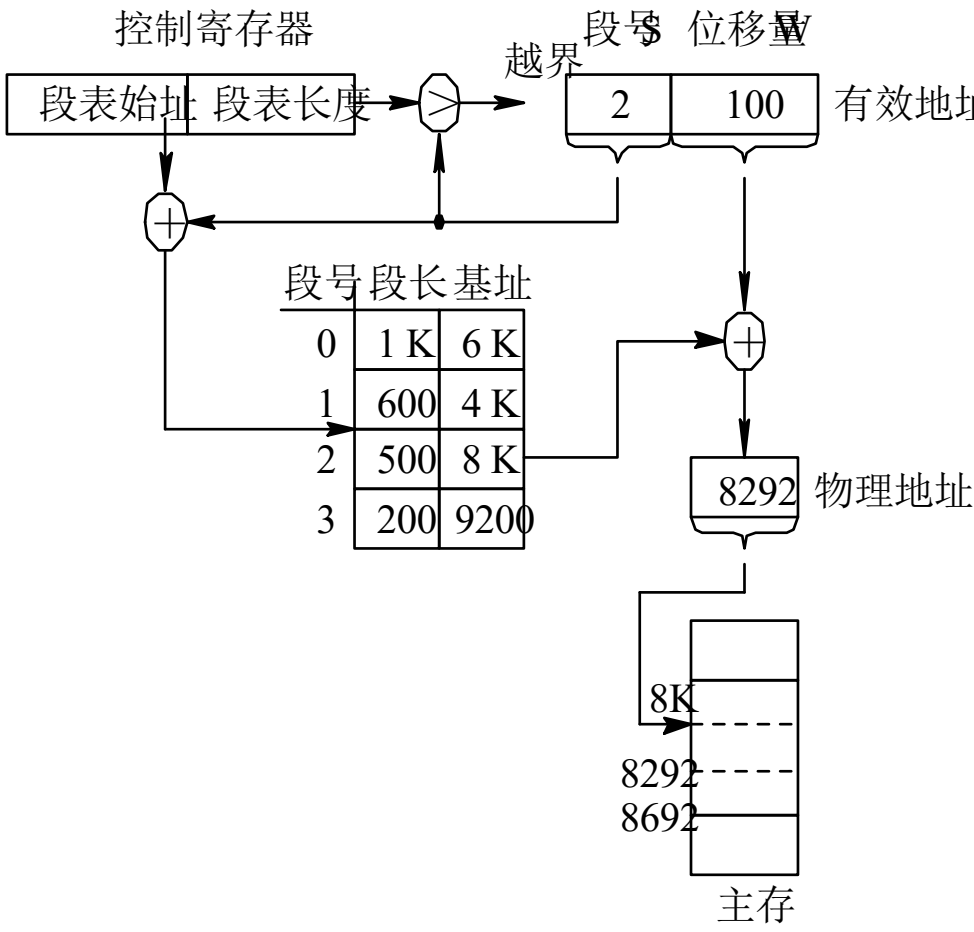
4.5.2 分段系统的基本原理

- 2 段表
  - 段映射表
    - 每个段占用一个表项，记录该段在内存中的起始地址和长度



4.5.2 分段系统的基本原理

- 3. 地址变换机构
  - 段表寄存器
    - 内容为段表始址和长度
  - 变换过程
    - 段表始址、段号 => 该段在段表中的表项位置 => 该段在内存中的起始地址
    - 起始地址 ＋ 段内地址 ＝ 物理地址
  - 快表
    - 最近常用的段表项



4.5.2 分段系统的基本原理

- 4. 分页和分段的主要区别
  - 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，它含有一组其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。
  - 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。
  - 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。

4.5.3 信息共享

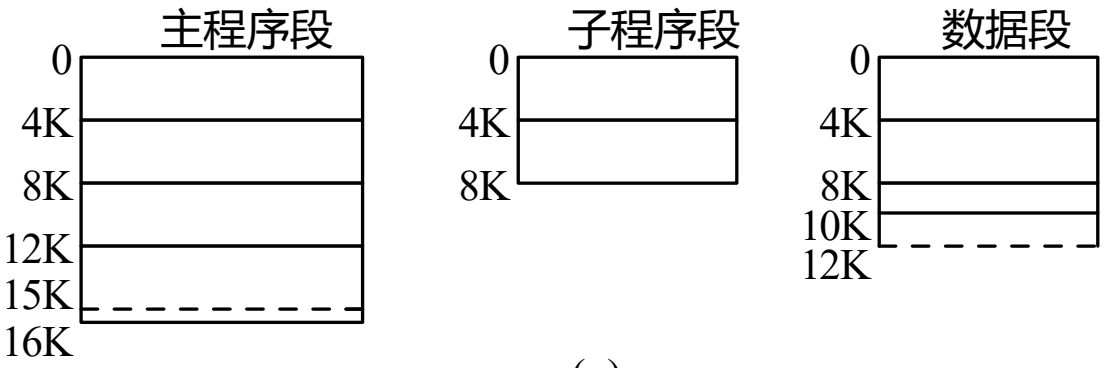
- 共享时的保护容易实现
- 段表的表项登记共享段信息
  - 可重入代码
    - 纯代码：允许多个进程同时访问的代码
    - 不允许进程对他修改

4.5.4 段页式存储管理方式

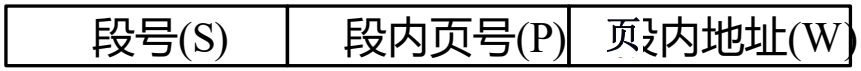
- 段式：反映了程序的逻辑结构。
- 页式：内存利用率高。
- 段式管理弊病的主要原因在于要求一个段在内存中连续存储。
- 如果以分段技术管理用户的逻辑地址空间，而以分页技术管理实际使用的主存空间，则兼并分段分页的优点。

4.5.4 段页式存储管理方式

- 1 基本原理
  - 用户程序分成若干段，每段分成若干页
  - 虚地址构成
    - 段号、段内页号、页内地址



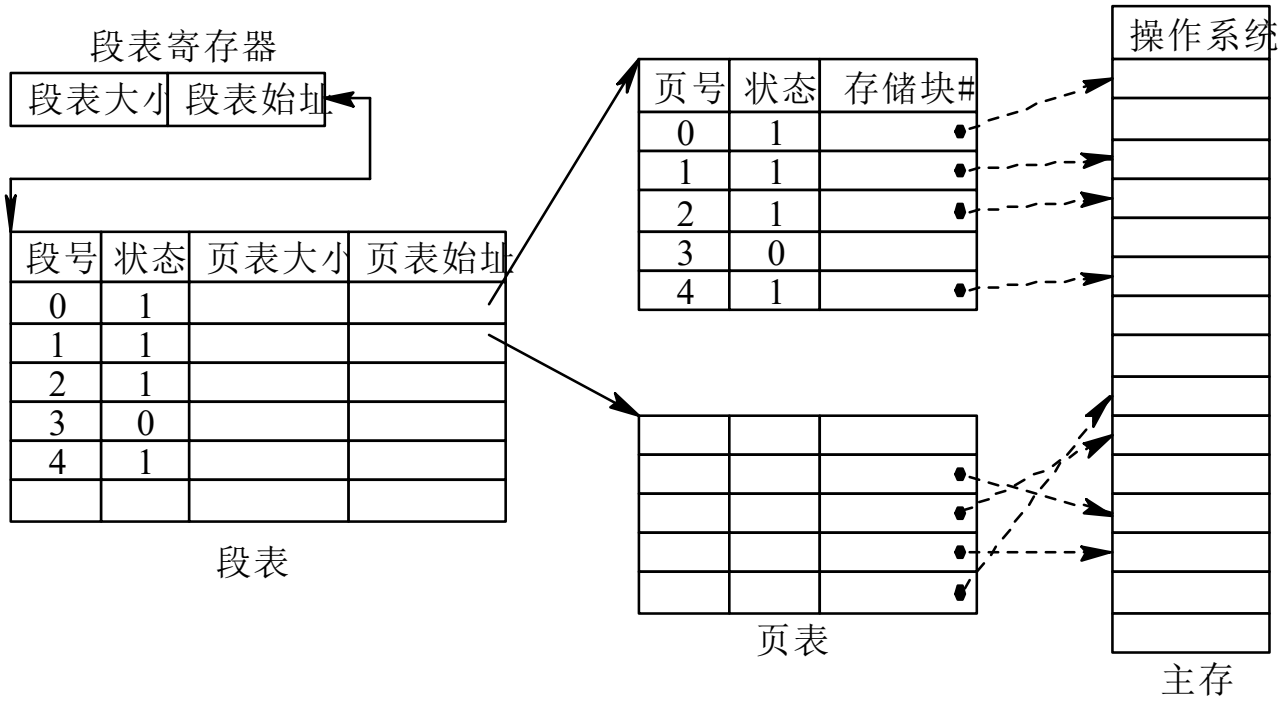
(*a*)



(*b*)

4.5.4 段页式存储管理方式

- 1 基本原理
  - 段表和页表
    - 每个作业或进程一张段表、每个段有一张页表
    - 段表寄存器：段表始址
    - 段表：页表始址　　；　　页表：块号



4.5.4 段页式存储管理方式

- 2 地址变换过程
  - 段表始址，段号 => 该段的页表始址
  - 页表始址 、 段内页号 => 物理块号
  - 物理块号 、 页内地址 => 物理地址
  - 加速
    - 快速联想寄存器