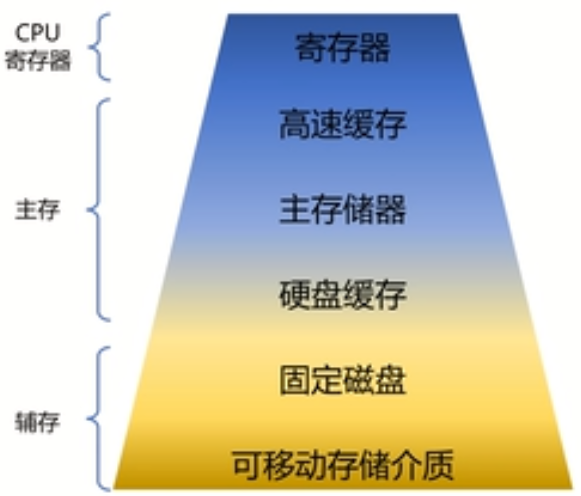
# 内存管理概念

## 准备工作

### 存储器结构寄存器

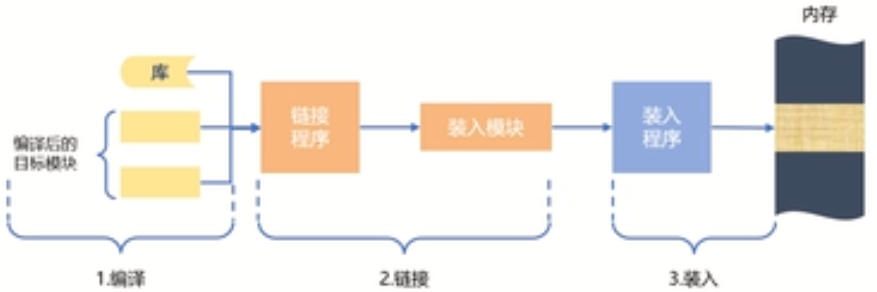
* 高速缓存（独立硬件）
* 主存储器
* 硬盘缓存（属于硬盘，进程挂起放在这）
* 固定磁盘
* 可移动存储介质



### 进程运行原理

#### 用户程序 —> 进程

* 编译
* 链接
  + 静态链接
  + 装入时动态链接
  + 运行时动态链接
* 装入
  + 绝对装入（绝对地址）（全部装入）
  + 可重定位装入（全部装入）
  + 动态运行时装入



#### 两个装入细节

* 逻辑地址与物理地址
* 内存保护（记录边界值）

#### 内存扩充的两种方式

* 覆盖
* 交换

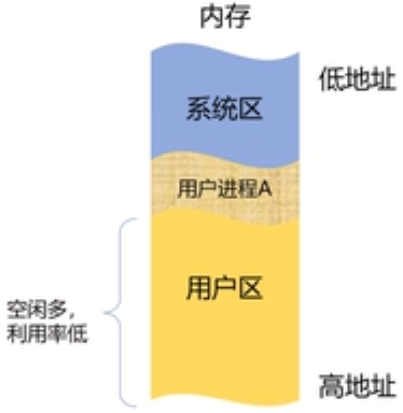
## 内存管理方式

### 连续分配管理方式

* 单一连续分配
* 固定分区分配
* 动态分区分配

#### 单一连续分配

* 优点：
  + 实现简单
  + 无外部碎片
  + 不一定需要内存保护
* 缺点：
  + 只能用于单用户、单任务OS
  + 有内部碎片
  + 存储器利用率低

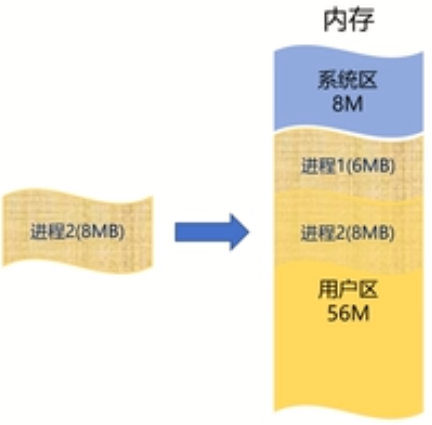


#### 固定分区分配

* 优点：
  + 实现简单
  + 无外部碎片
* 缺点：
  + 较大用户程序时，需要采用覆盖技术，降低了性能
  + 会产生内部碎片，利用率低



#### 动态分区分配——怎么记录内存的使用情况？



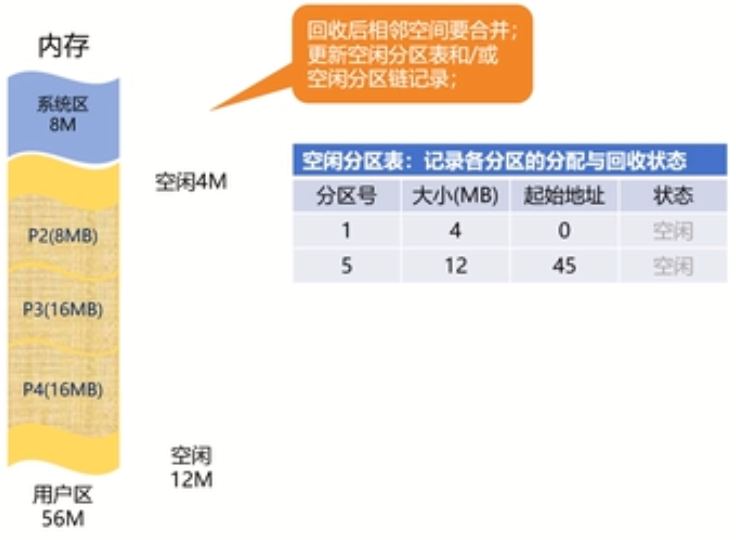
#### 动态分区分配——选择哪个分区给新进程？

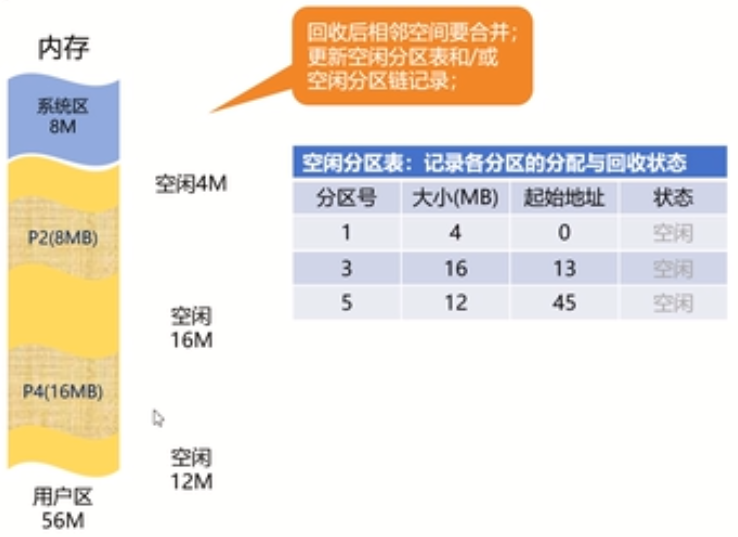
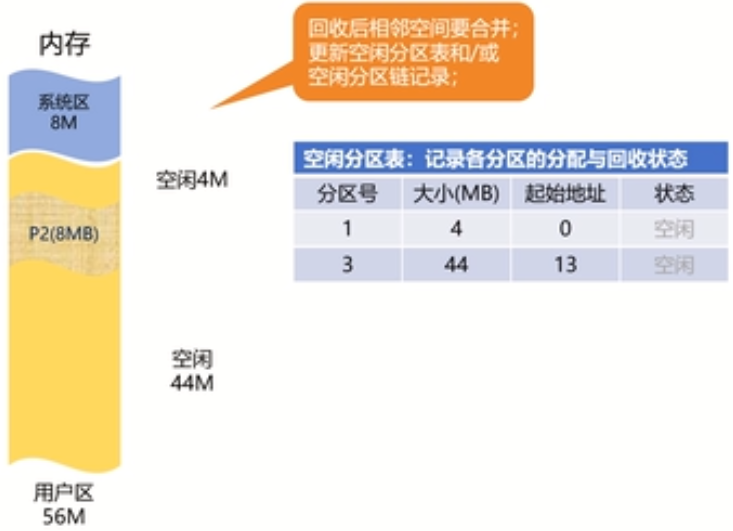
* 首次适应算法：从低地址查找合适空间
* 最佳适应算法：优先使用最小空闲空间
* 最坏适应算法：优先使用最大连续空间
* 临近适应算法：从上次查找处向后查找

|  |  |  |  |
| --- | --- | --- | --- |
| 算法 | 算法思想 | 分区排序 | 优缺点 |
| 首次适应 | 从低地址查找合适空间 | 地址递增排序 | 综合性能最好，开销小  不需要（对空闲分区）重排序 |
| 最佳适应 | 优先使用最小空闲空间 | 容量递增排序 | 更容易调足大进程常求  小碎片多，开销大，需要重排序 |
| 最坏适应 | 优先使用最大连续空间 | 容量递减排列 | 小碎片少  不利于大进程，开销大 |
| 临近适应 | 从上次查找处向后查找 | 地址递增排列（循环链表） | 不用每次从链表头查找，开销小  会使高地址大分区被用完 |

#### 动态分区分配——已使用的分区怎么回收？

* 回收后相邻空间要合并
* 更新空闲分区表和/或空闲分区链记录



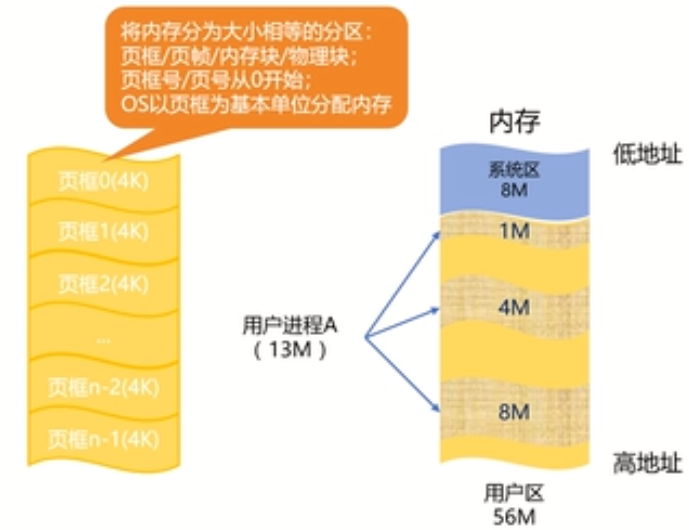
 

### 非连续分配管理方式

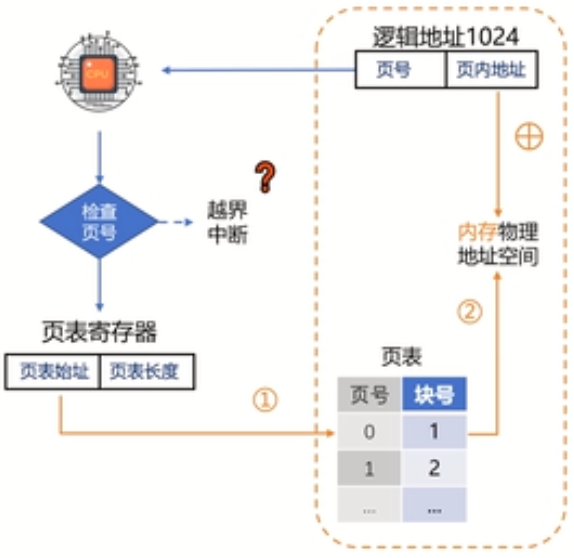
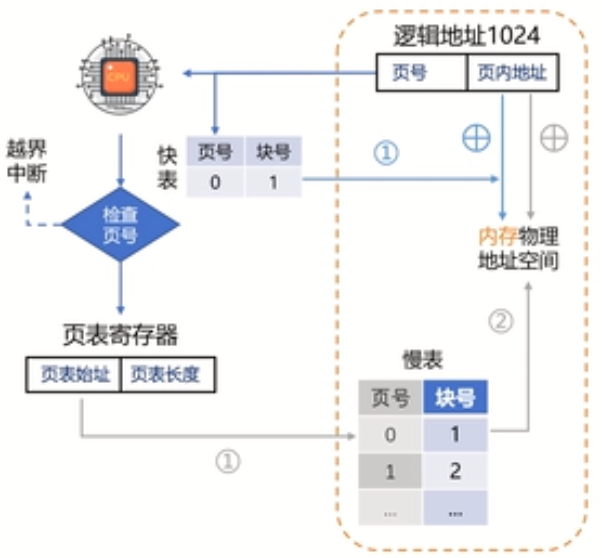
* 基本分页存储管理方式
* 基本分段存储管理方式
* 段页式管理方式

#### 基本分页存储管理方式（本质是固定分区）

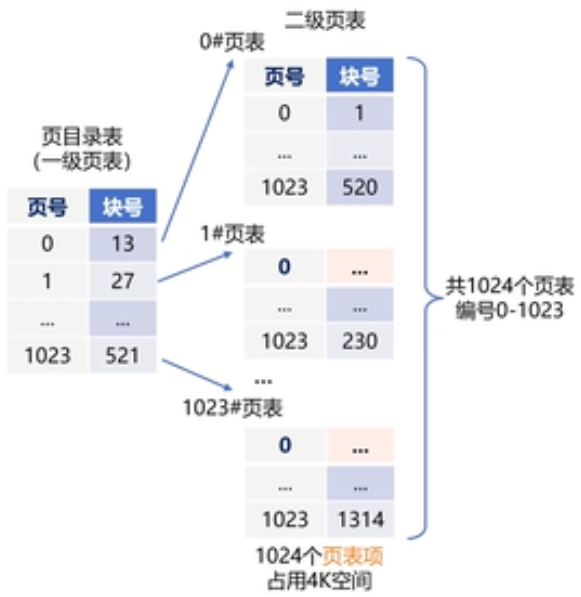
* 页/页面、页框、块
* 页表



* 基本地址变换机构
  + 物理地址 = (页号->块号) + 偏移量
  + 页号P = 逻辑地址A/页面长度（大小）L
  + 偏移量W = 逻辑地址A % 页面长度L
  + P = A >> 12; W = A & 4095
* 基本地址变换机构特点
  + 页式管理中地址空间是一维的
  + 每次访存都需要地址转换，必须足够快
  + 页表不能太大，会降低内存利用率（页表在内存中）

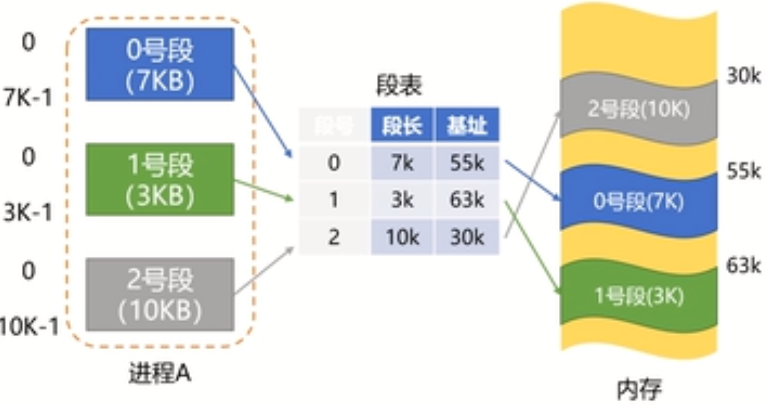
* 具有快表的地址变换机构（快表在高速缓存中）
  + 直接将页号与快表页号比较
  + 匹配成功，取块号+偏移量形成地址
  + 匹配失败，访问主存页表，并同步到快表(局部性原理)
* 两级页表
  + 页表连续存放，占用大量连续空间
  + 一段时间内只需要访问部分特定页面
  + 页表项分组/分页离散存储
  + 建页目录表管理离散页表
* 两级页表创建
  + 将逻辑地址拆分成三部分
  + 从PCB中读取页目录表始址
  + 根据一级页号查出二级页表位置
  + 根据二级页号查内存块号，加偏移量计算物理地址



#### 基本分段存储管理方式

* 分段
* 段表
* 地址变换机构
* 段的共享与保护



#### 分页与分段方式对比

* 页->物理单位
* 段->逻辑单位
* 分页->一维地址空间
* 分段->二维地址空间
* 分段更容易信息共享和保护

|  |  |  |
| --- | --- | --- |
| 管理方式 | 优点 | 缺点 |
| 分页管理 | 内存利用率高，不会产失外部碎片，少量内部碎片 | 不好按照逻辑模块实现信息共享和保护 |
| 分段管理 | 容易按逻辑模块实现信息共享和保护 | 段长较大时，不便分配空间，会产生外部碎片 |

#### 段页式管理方式

* 先分段，再分页
* 1个进程->1个段表
* 1个段表项>1个页表
* 1个页表->多个物理块



# 虚拟内存管理