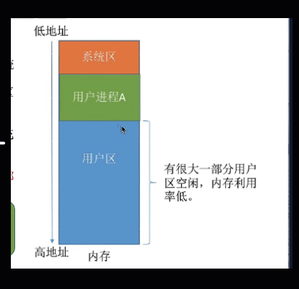
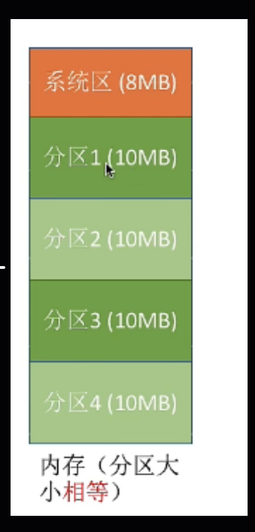
连续分配管理(分配与回收)

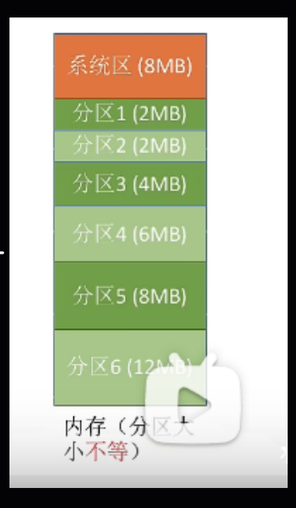
- 非连续分配
  - 基本分页存储管理
  - 基本分段存储管理
  - 段页式存储管理
- 连续分配
(用户进程分配的必须是一个连续的内存空间)
  - 单一连续分配
    - 框架



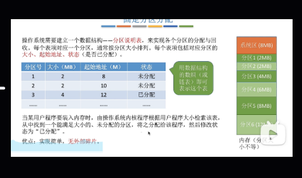
    - 分区
      - 系统区 —— 位于内存的低地址位置，存放操作系统的相关数据
      - 用户区 —— 存放用户进程的相关数据，内存中只能有一道用户程序，独占整个用户区
    - 优点
      - 实现简单，无外部碎片
      - 可以采用覆盖技术进行扩充
      - 不一定要采取内存保护 —— 早起的PC MS-DOS
    - 缺点
      - 只使用与单用户，单任务的操作系统
      - 有内部碎片，存储器利用率极低 —— 内部碎片 —— 分配给进程的内存区域中，有些部分没有用上，即为内部碎片
  - 固定分区分配
    - 思想 —— 用户空间划分成若干个固定大小的分区，每个分区中只装入一道作业
    - 分区分配
      - 分区大小相等 —— 缺乏灵活性，适用于一台计算机控制多个相同对象的场所



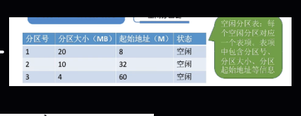
      - 分区大小不等 —— 灵活性强，满足了不同大小进程的需求
常根据运行中作业大小情况进行划分



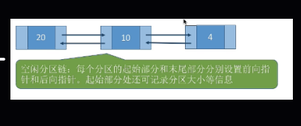
    - 记录分区
      - 数据结构 分区说明表 —— 每个表项对应一个分区，记录分区的大小，起始地址，状态等
      - 装入内存中，操作系统根据用户程序大小检索该表，找到满足条件的分区，然后修改分区说明表状态
    - 优点
      - 实现简单
      - 无外部碎片
    - 缺点
      - 用户程序太大的话，分区不满足要求，必须采用覆盖技术解决，降低性能
      - 产生内部碎片，利用率低
    - 框架



  - 动态分区分配
(可变分区分配)
    - 思想 —— 不会预先分配内存分区，进程装入内存时，根据内存大小动态分配分区，使分区刚好符合进程需要。
    - 问题解决
      - 数据结构记录使用情况
        - 空闲分区表 —— 每个表项对应一个空分区



        - 空闲分区链 —— 链表链接



      - 多个空闲分区都能满足，如何分配 —— 动态分区分配算法
        - 从空闲分区表选出一个分区分配给该作业
        - 四种分配算法
      - 如何分配与回收
        - 分配
          - 情形一 —— 修改起始地址

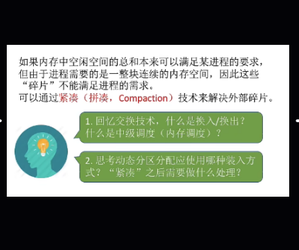
          - 情形二 —— 刚好满足，分区数目减一

        - 回收
          - 情形一、二 —— 回收的前后面有相邻空闲分区 —— 修改分区大小，起始地址

          - 情形三 —— 前后都有相邻 —— 三者合三为一

          - 情形四 —— 前后都没有相邻 —— 添加表项

        - 分配或者回收
          - 相邻 合并
          - 不相邻 添加或删除
      - 碎片
        - 定义
          - 内部碎片 —— 分配给进程的内存区域中，有部分没有用上
          - 外部碎片 —— 内存中的某些空闲分区由于太小而难以利用
        - 动态分配
          - 没有内部碎片
          - 但是有外部碎片 —— 可以使用紧凑和拼凑技术 解决外部碎片 —— 动态重定位装入



  - 框架