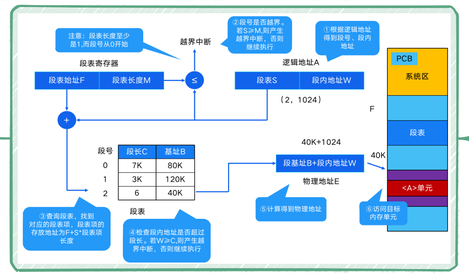
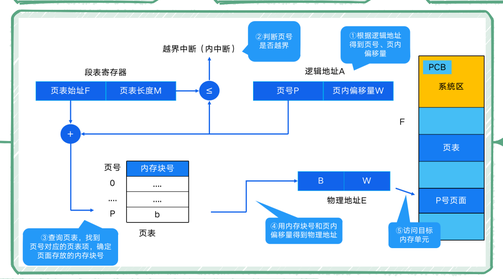
第三章 内存管理

分页与分段存储管理方式

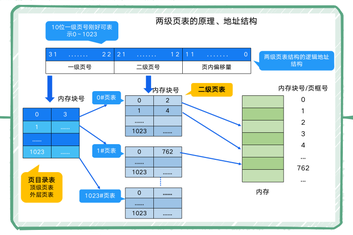
- 概念串联（哈工大）
  - 操作系统为了让用户方便使用，我们需要分段，仅仅分段会导致内存利用率低（外部碎片）
    - 引出分页，然后分段+分页，两个结合的核心是虚拟内存，如果要有虚拟内存，就必须要有换入与换出机制
- 分段
  - 程序员眼中程序由若干段组成，每个段有各自特点：代码段只读、代码/数据段不会动态增长（段内连续，段间不要求连续）
  - 有段表程序进入内存的故事
    - 程序分成多个段，每个段在内存中找到空闲的地方
      - 每个空闲内存的基址放在LDT表（局部描述符表，段表）中，最后将该LDT表赋给PCB
        - 每次执行指令时都查LDT表，将逻辑地址转变为物理地址
  - 地址变换机构



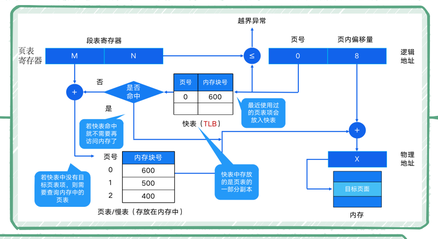
    - 地址结构：段号S+段内偏移量W
    - 段表
      - 每个进程都有一张映射逻辑地址空间和物理地址空间的段表，每个段表项对应进程的一段
      - 段号（隐含）+段长+本段在主存中的始址
  - 段的共享与保护
    - 共享：两个作业的段表中相应表项指向被共享段的同一物理副本实现
      - 保护：存取控制保护+地址越界保护
- 内存分区（连续）
  - 单一连续分配（内存仅有一道程序）
    - 系统分为系统区和用户区，系统区仅供操作系统使用，通常在低地址部分，用户区为用户提供，内存中只存在一道程序
      - 只适用于单用户单任务操作系统，存在内部碎片
  - 固定分区分配
    - 将用户空间划分为若干固定大小的区域，每个分区仅装入一道程序（适合多道程序存储，无外部碎片）
      - 分区大小相等
      - 分区大小不等
        - 通常将分区按大小排队，建立分区说明表（分区始址、大小、状态）
  - 动态分区分配
    - 进程装入内存时，根据进程大小动态建立分区
      - 存在外部碎片
        - 内存紧缩：空闲分区合并（需要移动）
    - 实现
      - 空闲分区表/空闲分区链（双向链表）
    - 首次适应算法
      - 空闲分区按照地址递增的次序进行查找，第一个满足条件的进行分配
        - 性能最好，开销小，回收分区后一般无需重新排序
    - 最佳适应算法
      - 按容量递增的顺序进行查询分区，将第一个满足条件的进行分配
        - 尽可能留下大片空闲区，外部碎片最多，回收分区后可能需要重新排序
    - 最坏适应算法（最大适应算法）
      - 空闲分区按照容量递减的次序进行查找，第一个满足条件的进行分配
        - 无可用大内存块，性能差，算法开销大
    - 邻近适应算法（循环首次适应算法）
      - 分配内存从上次查找结束的位置开始继续查找
        - 常在末尾分配空间，导致尾部空间分裂成小碎块，每次分配无需排序链表
- 分页
  - 引出分页：从连续到离散
  - 概念
    - 将主存空间划分为大小相等且固定的块作为主存的基本单位，进程以块为单位进行空间申请
    - 页：逻辑地址空间中的块
      - 页框：物理地址空间中的块
        - 页面太小：进程页面过多，增加内存占用，降低硬件地址转换效率，缺页频繁（若请求分页）
          - 页面太大：内部碎片过多，内存利用率低
            - 页面大小由操作系统及CPU架构决定
  - 基本地址变换机构



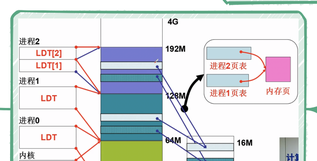
    - 地址结构：页号+页内偏移
    - 页表
      - 逻辑地址到物理地址的映射
        - 由页表项组成，页表项由页号（隐含）和对应物理块号组成
    - 页表寄存器（唯一，每个进程页表始址存放在PCB中，进程调度后放入页表寄存器）
      - 存放页表在内存中的起始地址和页表长度
      - 每个进程拥有一张页表，且进程的页表驻留在内存中
    - 计算方式
      - 页号P = A / L，页内偏移量W = A % L
        - 比较页号P和页表长度M，若P >= M产生越界中断
      - 页表中页号P对应的页表项地址 = 页表始址F + 页号P × 页表项长度取出该页表项内物理块号b
        - 计算物理地址E = b × L + W
- 多级页表及快表
  - 为了提高内存空间利用率，页应该小，页小了则页表大引出多级页表
  - 第一种尝试：只存放用到的页（用到的逻辑页才有页表项）会导致页表中页号不连续，不能使用偏移找到页表项，用顺序查找或折半查找会导致多次访存，效率极低
  - 第二种尝试：多级页表（页目录表+页表）
    - 好比看一本书单级页表只有节目录，每次要顺序查节目录，目录占用连续内存过大，多级目录就像增加了章目录，可以直接找到需要的章，在章下找小节，不用的章或节可以调出内存



    - 逻辑地址空间格式 = 一级页号 + 二级页号 + 页内偏移
    - 内存利用率提高、访存次数提高
  - 快表
    - 多级页表提高了空间效率，但时间上增加了访存次数，引入TLB（相联快速存储，寄存器）
      - 保存页表副本，用于地址变换
    - 地址变换机构中增加一个具有并行查找能力的高速缓冲存储器（快表），又称相联存储器（TLB）
      - 相联存储器既可以按照地址查找，也可以按照内容查找
    - 具有快表的地址变换机构



      - 快表命中仅访存一次，快表未命中，需要访存两次（单级页表）
      - TLB有效访问时间 = HitR × (TLB + MA) + (1-HitR) × (TLB+2MA)
        - 命中率：HitR、内存访问时间：MA、TLB时间：TLB，设为单级页表：仅2MA，e.g. 有效访问时间=98%×(20ns+100ns)+2%×(20ns+200ns)=120ns
- 段页式的故事
  - 概念：作业的地址空间首先被分为若干逻辑段，每段有自己的段号，每个段分为若干大小固定的页
    - 程序员希望用段，物理内存希望用页（段面向用户，页面向硬件）
  - 虚拟内存（实现分段与分页的关键所在）
    - 逻辑地址->虚拟地址（段表实现）
      虚拟地址->物理地址（用户透明，页表实现）
      - 此处有些疑惑，王道及查阅资料都认为逻辑地址和虚拟地址是一个东西
        但此处讲解时，逻辑地址可以认为是程序员编写代码时用的，虚拟地址是段表映射到虚拟地址空间的地址



    - 首先在虚拟内存上分配区域给用户的数据段，建立映射（建立段表）
      - 将用户数据段分割为多页，每页与物理内存的页框关联在一起
    - 利用段表记录虚拟内存是如何对应的，用页表记录虚拟存储的区域是咋么放到物理内存的
    - 父子进程用相同的物理页，子进程在自己的虚拟内存空间中建立新的顶级页表，并分配内存建立新的二级页表，拷贝父进程的二级页表

若父进程创建子进程

父子进程共用只读的物理页，当子进程都需要写入时，需要重新分配一块物理页，修改页表映射

父子进程通过段表和页表进行内存分离

地址转变机构

一个进程对应一个段表多个页表

地址结构：段号S+页号P+页内偏移量W

Presented with xmind