虚拟内存

前引

- 为什么需要虚拟内存
  - 程序大小受到实际内存大小的限制
  - 内存容量限制并发的进程数量
- 之前介绍过解决上述问题的方法
  - 覆盖技术：需要用户程序支持
  - 交换技术：进程的挂起

虚拟内存

- **虚拟内存的思想**
  - 基于分页或分段以及局部性原理
  - 进程运行的过程中，主存中只存放几个块/帧/段，而进程的全部信息存放在辅存中
- **虚拟内存的流程**
  - 进程的常驻集存放主存中
  - 当访问不在主存的块/帧/段时，产生**缺块中断**
    1. OS发出I/O请求，进程阻塞，进行进程切换
    2. I/O请求结束，产生中断，阻塞的进程转为就绪态
- **虚拟内存的优点**
  - 可以有更多的进程驻存在主存中
  - 进程可以大于整个主存空间
- **虚拟内存的缺点**
  - 系统抖动（块的调入调出），处理器需要处理I/O操作
- 虚拟内存需要的支撑
  - 是否采用虚拟存储技术
  - 是否使用分页、分段或段页
  - 存储管理所使用的算法/内存管理策略

虚拟内存需要的硬件支撑

- 分页
  - 每一个进程同样需要维护一个页表
    - ```
      # 页表
      页号：|主/辐标志 P|修改位 M|other control bits|Frame Number|
      P:表示该页是否在主存中有存储
      M:若标记修改，则在修改辅存中的该块对应的信息时，需要先从主存中将该块移到辅存中，
      若未标记修改，则不需要进行该I/O操作
      ```
    - 通过virtual address查找相应的块的流程：
      ```

1. 通过virtual address的页号在页表中找到对应的项
2. 如果P标志主，则直接根据Frame Number找到在主存中存放的帧，之后根据offset+帧的起始位置获得数据
3. 如果P标志辅，那么发出I/O中断，将辅存中该页移入主存中，I/O中断完成后，再次根据Frame Number找到在主存中存放的帧，之后根据offset+帧的起始位置获得数据

- 页表往往会很大，如何解决
  - 多级页表
  - 反向页表
- **转移后备存储器（TLB）**
  - 为什么需要
    - 虚拟内存访问需要两次访问内存：访问页表，访问数据
  - 是什么
    - 将页表移入高速缓存，减少访问内存的次数
    - 移入最近使用过的页表
- 页越小
  - 页内碎片总量越小
  - 每个进程的页表越大
  - I/O效率低（一小块一小块地搬和一大块一大块地搬相比）

- 分段
- 段页

内存管理策略

读取策略：何时将页读入主存

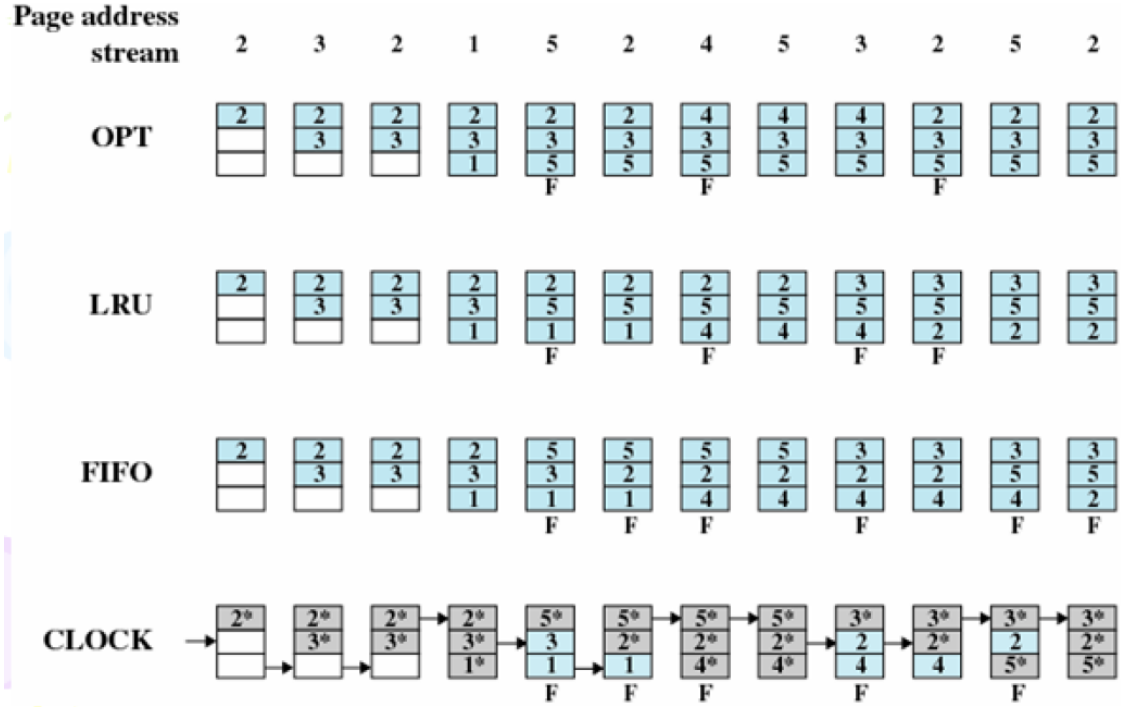
- 请求分页：当发生缺页中断时，need and fetch a page
- 预约分页：预测将会用到哪些页，提前读入more pages
- 二者结合：当进程刚开始运行时，通过预约分页读入较多页，之后运行过程中，利用请求分页

放置策略：确定读入内存的块的放置位置

- 分页和段页 无需考虑存放的位置
- 分段需要考虑采用最佳匹配、首次匹配或者邻近匹配

替换策略：解决读入的新页替换主存中的哪一页

- **最佳算法**：替换页时选择**未来最晚被访问的页**进行替换，不可实现
- **最近最少使用算法（LRU）**：选择**过去最早被使用的页**进行替换
- **先进先出(FIFO)算法**：抽象出一个循环队列，指针指向最先进来的页，最先进来的页先被替换
- **时钟策略**
  - 设置使用标志位U，当页被使用时标记为1
  - 当有缺页中断时，从指针指向的当前位置起，查找第一个使用标志位为0的页面作为替换页
    - 在查找过程中，将遇到的使用标志位为1 的页标志位修改为0
  - 当没有缺页中断时，指针不移动

- **时钟策略的改进**
  - 为什么需要改进
    - 当替换页时，需要先将主存中的页存入辅存中，再将辅存中的页存入主存中
    - 如果主存中的页没有被修改，那么可以省略将主存中给的页存入辅存中的步骤
    - 所以在进行替换页时，优先替换主存中没有修改的页
  - 增加一个修改标志位M，如果主存中的页被修改过标记为1
  - 步骤
    1. 在循环队列中查找U=0,M=0的块/页，如果找到，则将该页作为替换页。
    2. 若没有找到U=0,M=0的页，则再查找U=0,M=1的页，如果找到，则将该页作为替换页
    3. 若没有找到U=0,M=1的页，则表明当前循环队列中的页都被使用，则将循环队列中所有的U标记由1改为0。
    4. 指针回到初始位置，重复步骤1~3
- 

驻留集管理：管理分配给一个进程的主存容量大小（页的多少）

- **固定分配**
- **可变分配**
  - **工作集策略**：使当前时刻驻留集的大小与工作集的大小尽可能相等
    - 工作集$w(t,\Delta)$：一个进程当前正在使用的页面的集合
      - ```
        eg:
        页面使用顺序：
        | t |   0   1   2   3   4   5   6   7   8   9
        |页 | 24| 15|18 | 23| 25| 17| 17| 19| 21| 18|
                        ↑                       ↑
        w=(24,15,18) Δ=3       w=(21,19,17,25) Δ=5
        ```
    - 思想
      - 监视每个进程的工作集
      - 周期性的从驻留集中移去不在工作集中的页
      - 驻留集包含工作集才能执行该进程
        ```

- **缺页中断频率算法**
  - 缺页中断频率高，驻留集增大
  - 缺页中断频率低，驻留集减小

清除策略：确定何时将一个被修改的页写回辅存

- **请求式清除**：**当修改页被替换时，执行写回操作**
- **预约式清除**：预先（修改页被替换前）执行写回操作
- **结合页缓存清除**
  - 在替换页时，先将页存放于缓存中
  - 缓存中有两个列表
    - 修改页
    - 未修改页
  - 操作系统周期性地将修改页写入辅存，并将修改页移入未修改页列表中，同时清除未修改页列表中的页

加载控制：负责进程的装入，管理主存中进程的数量

- 驻留进程数量太少，处理器效率低
- 驻留进程数量太多，进程的驻留集小，频繁发生缺页中断
- 进程挂起策略：
  - 最低优先级进程
  - 缺页进程（阻塞状态）
  - 最后一个被唤醒的进程（该进程可能还没有移入主存中）
  - 驻留集最小的进程
  - 最大空间的进程
  - 最大剩余时间的进程