# 3_内存管理

## 3.1_1 内存的基础知识

- 内存的定义|作用
  - 程序执行前需要先放到内存中才能被CPU处理
  - 存储单元：每个地址对应一个存储单元
  - 补充：
    - 1K=$2^{10}$
    - 1M=$2^{20}$
    - 1G=$2^{30}$
- 逻辑地址|物理地址
  - 逻辑地址就是相对地址
- 从写程序到程序运行
  - 编辑
    - 编写程序
  - 编译
    - 将高级语言转化成机器语言，由源代码文件生成目标模块
  - 链接
    - 由目标模块生成装入模块，链接后生成完整的逻辑地址
    - 链接的三种方式
      - 静态链接
        - 在程序运行前，先将各目标模块及它们所需的库函数连接成一个完整的可执行文件
      - 装入时动态链接
        - 将各目标模块装入内存时，边装入边链接的链接方式
      - 运行时动态链接

        - 在程序执行中需要该模块时，才对它进行链接，其优点时便于修改和更新
    - 装入
        - 将装入模块装入内存，装入后形成物理地址
        - 装入的三种方式
            - 绝对装入
                - 在编译的时候就知道程序放在内存的哪个位置
                - 只适用于单道程序环境
            - 可重定位装入|静态重定位
                - 装入时将逻辑地址转表为物理地址
                - 一个作业装入内存的时候，必须分配其要求的全部内存空间，如果没有足够的内存，就不能装入该作业
                - 再程序运行期间就不能再移动
            - 动态运行时装入|动态重定位
                - 把地址转化推迟到程序真正要执行时才进行
                - 需要重定位寄存器
                - 允许程序在内存中发生移动

## 3.1_2 内存管理的概念

- 内存管理要实现的功能
    - 内存空间的分配与回收
    - 需要提供某种技术从逻辑上对内存空间进行扩充（实现虚拟性）
    - 逻辑地址到物理地址的转化（三种装入方式）
    - 提供内存保护功能，保证各进程在各自的存储空间内运行，互不干扰
        - 设置上下限寄存器
        - 采用重定位寄存器（基址寄存器）和界地址寄存器（限长寄存器）
            - 重定位寄存器记录起始物理地址

- 界地址寄存器存放最大逻辑地址

## 3.1_3 进程的内存映像

- 存储区域分类


## 3.1_4 覆盖与交换

- 内存空间的扩充
  - 覆盖技术
    - 用于解决“程序大小超过物理内存总和”的问题
    - 将程序分为多个段，内存分为一个”固定区“和若干个”覆盖区“，需要常驻的放在”固定区“，调入后就不再调出，不常用的段放在”覆盖区“，需要用到时调入内存，用不到时掉出内存
    - 必须由程序员声明覆盖结构
    - 缺点
      - 对用户不透明
  - 交换技术
    - 内存空间紧张时，系统将内存中某些进程暂时换出外存，把外存中某些已具备运行条件的进程换入内存（PCB会常驻内存，不会被换出）（进程在内存与磁盘之间动态调度）

        - 磁盘分为文件区和对换区，换出的进程放在对换区
    - 覆盖与交换的区别
        - 覆盖是在同一个程序或进程中的
        - 交换是在不同进程（或作业）之间的

## 3.1_5 连续分配管理方式

- 连续分配方式
    - 指为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
    - 内存被分配为系统区和用户区，系统区通常在低地址，用于存放操作系统相关数据，用户区用于存放用户进程相关数据
    - 优点
        - 实现简单
        - 无外部碎片
        - 可以采用覆盖技术扩充内存
        - 不一定需要采取内存保护‘
    - 缺点
        - 只用于单用户、单任务的操作系统
        - 有内部碎片
        - 存储器利用率极低
- 固定分区分配
    - 将用户区分割为若干固定分区给各道程序，分割策略有分区大小相等和分区大小不相等，可以建议一个分区说明表来管理各个分区
        - 分区大小说明表
            - 包括对应分区的大小、起始位置、状态（是否已分配）
    - 优点
        - 实现简单
        - 无外部碎片
    - 缺点

        - 如果用户程序太大，无分区满足，则需要覆盖技术，这会降低性能
        - 会产生内部碎片，内存利用率低
- 动态分区分配
    - 可变分区分配，不会预先划分内存分区，而是在进程装入内存时，根据进程的大小动态地建立分区，并使分区的大小正好适合进程的需要。
    - 两种常用的数据结构
        - 空闲分区表
            - 包括对应分区的大小、起始位置、状态（是否已分配）
        - 空闲分区链
            - 设置前向指针和后向指针
    - 优点
        - 没有内部碎片
    - 缺点
        - 有外部碎片
- 内部碎片：分配给某进程的内存区域中，如果有些部分没有用上
- 外部碎片：是指内存中的某些空闲分区由于太小而难以利用（如果有外部碎片，可以采用紧凑技术）

## 3.1_6 动态分区分配算法

- 首次适应算法（First Fit）
    - 算法思想：
        - 每次从低地址开始查找，找到第一个能满足大小的空闲分区
- 最佳适应算法（Best Fit）
    - 算法思想：
        - 为了保证“大进程”到来时能有连续的大片区域，可以尽可能留下大片的空闲区，优先使用更小的空闲区。

        - 空闲分区按容量递增次序链接，分配内存时顺序查找空闲分区链
    - 缺点：
        - 会留下小碎片
- 最坏适应算法（Worst Fit）
    - 算法思想：
        - 最佳适应算法相反，按容量递减次序排列，每次尽可能用大的分区
        - 空闲分区按容量递减次序链接，分配内存时顺序查找空闲分区链
    - 缺点：
        - 后续大空间容易没有位置存放
- 领进适应算法（Next Fit）
  - 算法思想：
  - 每次从上次查找结束的位置开始检索
  - 缺点：
  - 大空间容易被用完

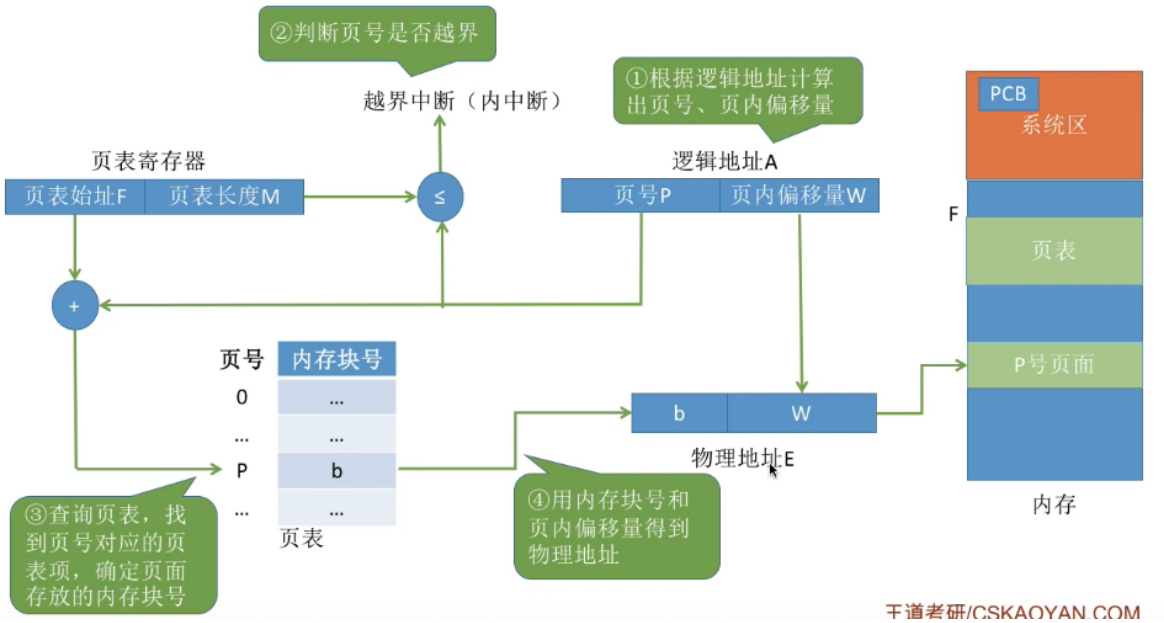
| 算法 | 算法思想 | 分区排列顺序 | 优点 | 缺点 |
| --- | --- | --- | --- | --- |
| 首次适应 | 从头到尾找适合的分区 | 空闲分区以地址递增次序排列 | 综合看性能最好。***算法开销小***，回收分区后一般不需要对空闲分区队列重新排序 | |
| 最佳适应 | 优先使用更小的分区，以保留更多大分区 | 空闲分区以容量递增次序排列 | 会有更多的大分区被保留下来，更能满足大进程需求 | 会产生很多太小的、难以利用的碎片；***算法开销大***，回收分区后可能需要对空闲分区队列重新排序 |
| 最坏适应 | 优先使用更大的分区，以防止产生太小的不可用的碎片 | 空闲分区以容量递减次序排列 | 可以减少难以利用的小碎片 | 大分区容易被用完，不利于大进程；***算法开销大***（原因同上） |
| 邻近适应 | 由首次适应演变而来，每次从上次查找结束位置开始查找 | 空闲分区以地址递增次序排列（可排列成循环链表） | 不用每次都从低地址的小分区开始检索。***算法开销小***（原因同首次适应算法） | 会使高地址的大分区也被用完 |

## 3.1_7 基本分页存储管理的基本概念

- 连续分配：为用户进程分配连续的内存空间
- 非连续分配：为用户进程分配分散的内存空间
- 分页存储
  - 将内存分为大小相等的小分区“页框”，每个页框有一个编号，即“页框号”，页框号从0开始。
    - 页框=页帧=内存块=物理块=物理页面
    - 页框号=页帧号=内存块号=物理块号=物理页号
  - 将逻辑地址空间也分为与页框大小相等的一个个部分，每一个部分称为一个“页”或“页面”
  - 进程的页面与内存的页框有一一对应的关系
- 页表
  - 页表项由“页号”和“块号”组成
  - 页表记录进程页面和实际存放的内存块之间的映射关系
  - 页表记录的只是内存块号，而不是内存块的起始地址
    - J号内存块的起始地址=J * 内存块大小
- 地址的转换
  - 分页后，各页面是离散存放的，但页面内部是连续存放的
  - 逻辑地址A对应的物理地址=P号页面在内存中的起始位置+页内偏移量W
  - 页号=逻辑地址/页面长度（去除法的整数部分）
  - 页内偏移量=逻辑地址%页面长度（取除法的余数部分）
  - 如果每个页面大小为$2^k$B，用二进制数表示逻辑地址，则末尾的K位即为页内偏移量，其余部分就是页号
- 逻辑地址结构
  - 如果有K位表示“页内偏移量”，则说明该系统中一个页面的大小是$2^k$个内存单元
  - 如果有M位表示“页号”，则说明在该系统中，一个进程最多允许$2^M$个页面

## 3.1_8 基本地址变换机构

- 页表寄存器（PTR）
  - 存放页表在内存中的起始地址F和页表长度M，进程未执行时，页表的起始地址和页表的长度放在进程控制块（PCB）中，当进程被调度时，操作系统内核会把它们放在页表寄存器中。
- 计算步骤




  - 若页号P>=页表长度M时会越界
  - 其中：页号P对应的==页表项地址=页表起始地址F+页号P * 页表项长度
  - 计算E=b * L+W

## 3.1_9 具有快表的地址变换机构

- 快表
  - 又称联想寄存器（TLB），是一种访问速度比内存快很多的高速缓冲存储器（TLB不是内存），用来存放最近访问的若干页表项的副本，以加速地址变换的过程。与此对应，内存中的页表常称为慢表。
  - 如果快表查询命中，则访问某个逻辑地址仅需一次访存
  - 如果快表查询没有命中，则需要两次访存（第一次是访问查询内存中存放的页表，第二次是访问最终需要访问的内存单元），同时将查询到的内容从慢表复制到快表。
- 局部性原理

- 时间局部性
  - 访问某个变量后，在不久的将来还会被访问
  - 执行某个指令后，在不久后这条指令可能会被再次执行
- 空间局部性
  - 程序访问了某个存储单元，不久之后，其附近的存储单元也很有可能被访问

- 总结

| | 地址变换过程 | 访问一个逻辑地址的访存次数 |
|---|---|---|
| 基本地址变换机构 | ①算页号、页内偏移量<br>②检查页号合法性<br>③查页表，找到页面存放的内存块号<br>④根据内存块号与页内偏移量得到物理地址<br>⑤访问目标内存单元 | 两次访存 |
| 具有快表的地址变换机构 | ①算页号、页内偏移量<br>②检查页号合法性<br>③查快表。若命中，即可知道页面存放的内存块号，可直接进行⑤；若未命中则进行④<br>④查页表，找到页面存放的内存块号，并且将页表项复制到快表中<br>⑤根据内存块号与页内偏移量得到物理地址<br>⑥访问目标内存单元 | 快表命中，只需一次访存<br><br>快表未命中，需要两次访存 |

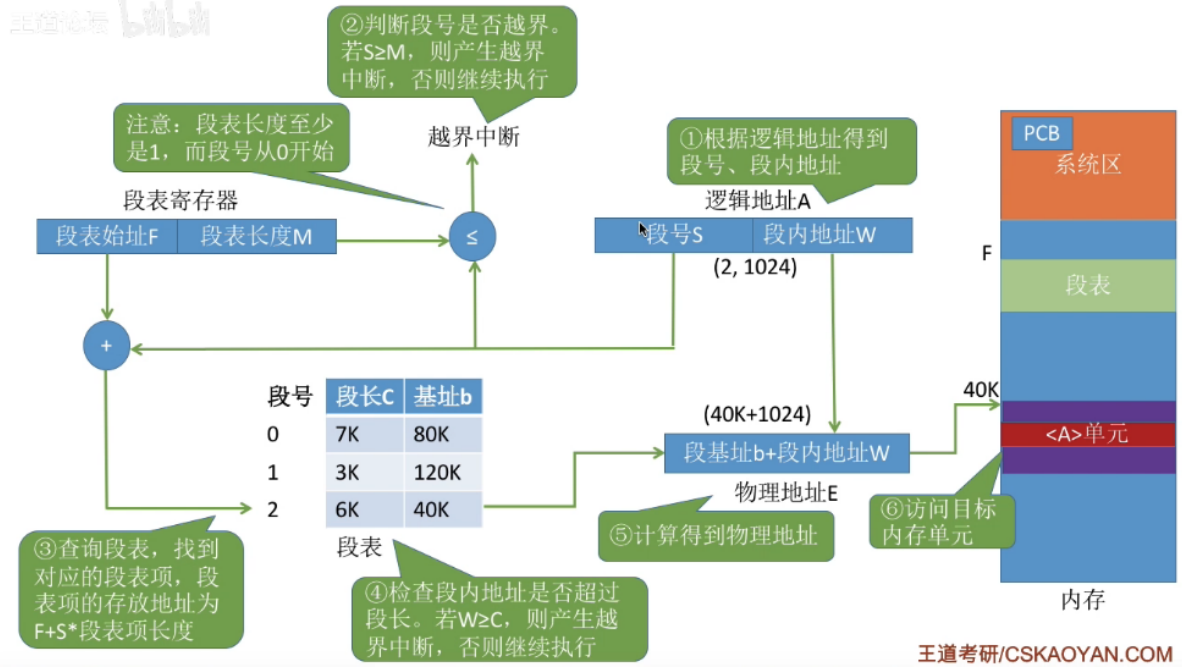
## 3.1_10 两级页表

- 单极页表存在的问题
  - 所有页表项必须连续存放，页表过大时需要很大的连续空间
  - 在一段时间内并非所有页面都用得到，因此没必要让整个页表常驻内存
- 两级页表的原理、逻辑地址结构
  - 逻辑地址结构：
    - 页目录表、外层页表、顶级页表
    - 一级页号、二级页号、页内偏移量
- 实现地址变换
  - 按照地址结构将逻辑地址拆分成三部分
  - 从PCB中读出页目录表始址，根据一级页号查页目录表，找到下一级页表在内存中的存放位置

    - 根据二级页号查表，找到最终想访问的内存块号
    - 结合页内偏移量得到物理地址
- 细节
    - 多级页表中，各级页表的大小不能超过一个页面。若两级页表不够，可以分更多级
    - 多级页表的访问次数（假设没有快表结构）——N级页表访问一个逻辑地址需要N+1次访存

## 3.1_11 基本分段存储管理

- 分段
    - 进程的地址空间
        - 按照程序自身的逻辑关系划分为若干个段，每段有段名，每段从0开始编址
    - 内存分配规则
        - 以段为单位进行分配，每个段在内存中占据连续空间，但各段之间可以不相邻
    - 逻辑地址由段号（段名）+段内地址（段内偏移量）组成
    - 段号的位数决定了每个进程最多可以分几个段
    - 段内地址位数决定了每个段的最大长度是多少
- 段表
    - 段号，段长，基址
    - 每个程序被分段后，用段表记录该程序在内存中存放的位置
- 地址变换
    - 段表寄存器
        - 段表初始地址F+段表长度M



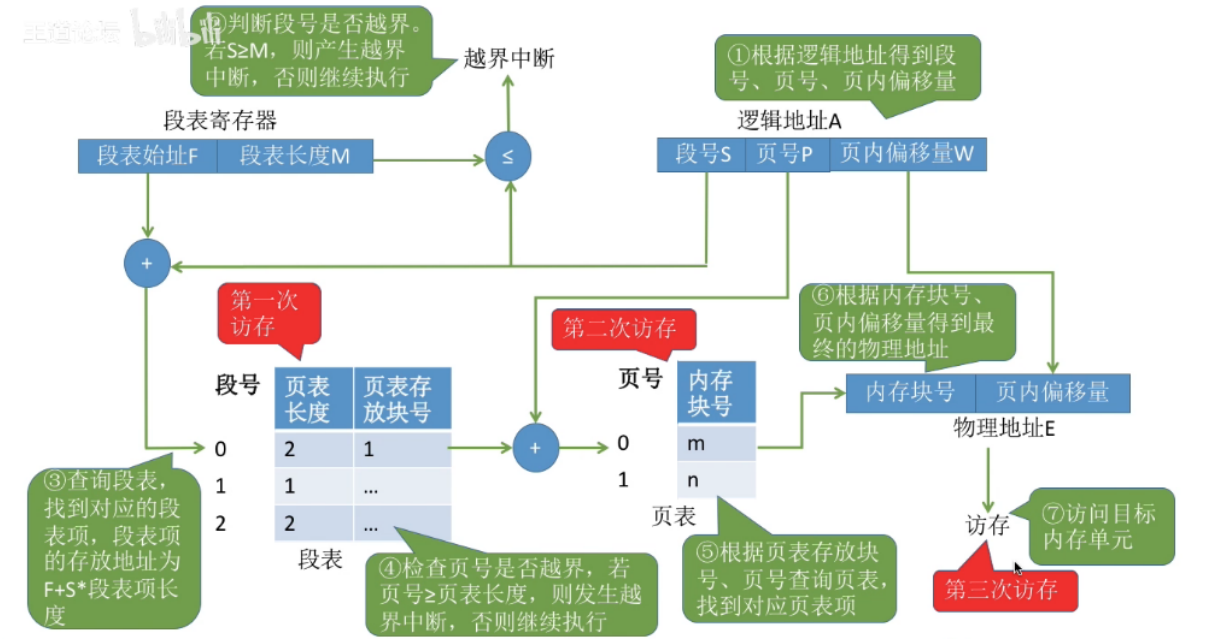

- 分段、分页管理的对比
  - 页：信息的物理单位，实现离散分配，提高内存利用率，对用户是不可见的，地址是一维的
  - 段：信息的逻辑单位，对用户可见，地址是二维的
  - 分段比分页更容易实现信息的共享和保护（纯代码/可重入代码可以共享）
  - 分页（单级页表）：两次访问
    - 第一次：查内存中的页表
    - 第二次：访问目标内存单元
  - 分段：两次访问
    - 第一次：查内存中的段表
    - 第二次：访问目标内存单元
  - 两者都可以采用快表机构，减少内存的访问次数

## 3.1_12 段页式管理方式

- 分页、分段管理方式最大的优缺点
  - 分页：
    - 内存空间利用率高，不会产生外部碎片，少量内部碎片碎、
    - 不方便进行信息共享和保护

- 分段：
  - 方便信息共享和保护，
  - 如果段长过大，容易产生外部碎片
- 段页式管理方式——分段+分页的结合
  - 先分段再分页
  - 逻辑地址：段号，页号，页内偏移量
  - 注意：
    - 段号的位数觉得每个进程最多有几个段
    - 页号位数决定了每个段最多有多少页
    - 页内偏移量决定了页面大小，内存块大小是多少
  - 地址结构是二维的
- 段表、页表
  - 段表项=段号+页表长度+页表存放块号（段表项长度固定，段号隐含）
  - 页表=页号+内存块号
  - 一个进程只会对应一个段表，但是有可能对应多个页表
- 实现地址变换

---




  - 第一次：查段表
  - 第二次：查页表

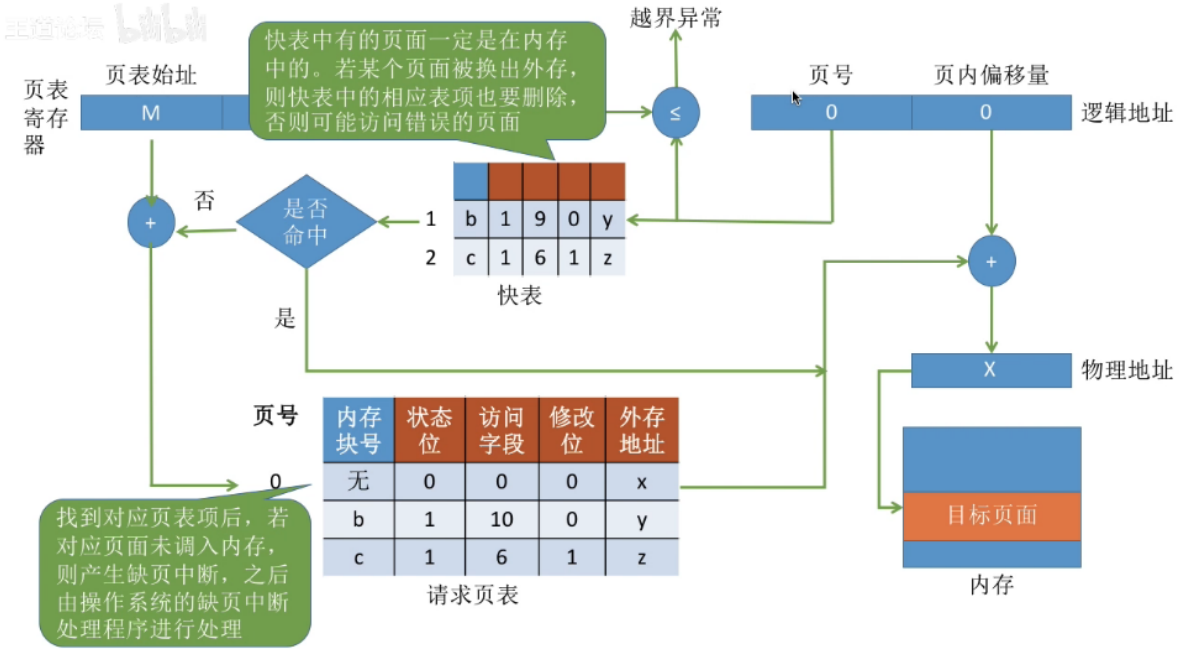
        - 第三次：访问目标单元

## 3.2_1 虚拟内存基本概念

- 传统存储管理方式的特征、缺点
    - 一次性：
        - 作业必须全部装入内存后才能开始运行
        - 大作业无运行
        - 并发性下降
    - 驻留性：
        - 一旦作业被装入内存，就会一直驻留在内存
- 局部性原理
    - 时间局部性
    - 空间局部性
    - 高速缓存技术
- 虚拟内存的定义和特征
    - 虚拟内存
        - 在程序装入时，可以将程序中很快会用到的部分装入内存，暂时用不上的部分留在外存
        - 运行时，访问信息不在内存时，由操作系统负责将所需信息从外存调入内存
        - 当内存空间不够时，将暂时用不上的信息换出到外存
    - 主要特征
        - 多次性：无需一次装入内存
        - 对换性：无需一直常驻
        - 虚拟性：逻辑上扩充内存
    - 虚拟内存实际容量
        - 虚拟内存最大容量是计算机地址结构确定的
        - 虚拟内存的实际容量=min(内存和外存容量之和，CPU寻址范围)

## 3.2_2 请求分页管理方式

- 页表机制
  - 请求分页存储的页表：
    - 内存块号 状态位 访问字段 修改位 外存地址
- 缺页中断机构
  - 当访问页面不在内存时，产生一个缺页中断（内中断----故障）
- 地址变换机构




## 3.2_3 页面置换算法

- 最佳置换算法（OPT）
  - 每次选择淘汰的页面是以后永不使用或者在最长时间内不再被访问的页面，这样可以保证最低的缺页率
  - 缺页时未必发生页面置换
  - 缺页率=缺页中断次数/总访问次数
  - 实际上不知道后面的序列，最佳置换算法实际上是无法实现的
- 先进先出置换算法（FIFO）
  - 每次选择淘汰的页面是最早进入内存的页面
  - Belady异常：当分配的内存块增大时，缺页次数反而增加

- 最近最久未使用置换算法（LRU）
  - 每次淘汰最近最久未使用的页面
  - 在逆向扫描过程中最后一个出现的页号就是淘汰的页面
  - 优点：
    - 性能好，最接近OPT算法的性能
  - 缺点：
    - 实现困难，开销大
- 时钟置换算法（最近未用算法，CLOCK）
  - 最多经历两轮扫描，初始为1，扫一下为0，再扫一下被踢
- 改进型的时钟置换算法
  - 最多会进行四轮扫描，优先淘汰没有被修改过的，因为没有修改过的不用进行IO操作00->01（改）->00->01

    算法规则：将所有可能被置换的页面排成一个循环队列
    第一轮：从当前位置开始扫描到第一个（0, 0）的帧用于替换。本轮扫描不修改任何标志位
    第二轮：若第一轮扫描失败，则重新扫描，查找第一个（0, 1）的帧用于替换。本轮将所有扫描过的帧访问位设为0
    第三轮：若第二轮扫描失败，则重新扫描，查找第一个（0, 0）的帧用于替换。本轮扫描不修改任何标志位
    第四轮：若第三轮扫描失败，则重新扫描，查找第一个（0, 1）的帧用于替换。

| | 算法规则 | 优缺点 |
|---|---|---|
| OPT | 优先淘汰最长时间内不会被访问的页面 | 缺页率最小，性能最好；但无法实现 |
| FIFO | 优先淘汰最先进入内存的页面 | 实现简单；但性能很差，可能出现Belady异常 |
| LRU | 优先淘汰最近最久没访问的页面 | 性能很好；但需要硬件支持，算法开销大 |
| CLOCK（NRU） | 循环扫描各页面<br>第一轮淘汰访问位=0的，并将扫描过的页面访问位改为1。若第一轮没选中，则进行第二轮扫描。 | 实现简单，算法开销小；但未考虑页面是否被修改过。 |
| 改进型CLOCK（改进型NRU） | 若用（访问位, 修改位）的形式表述，则<br>第一轮：淘汰（0, 0）<br>第二轮：淘汰（0, 1），并将扫描过的页面访问位都置为0<br>第三轮：淘汰（0, 0）<br>第四轮：淘汰（0, 1） | 算法开销较小，性能也不错 |

# 3.2_4 页面分配策略

- 驻留集
  - 指请求分页存储管理中给进程分配的物理块的集合
- 固定分配
  - 每个进程的物理块数目一定，驻留集大小不可变
- 可变分配
  - 先分配一定数目的物理块，后根据运行情况动态调整
- 局部置换
  - 发生缺页使，只能选择进程自己的物理块进行置换
- 全局置换
  - 可将操作系统保留的空闲物理块分配给缺页进程
- 页面分配、置换策略
  - 固定分配局部替换：驻留集大小不可改变
  - 可变分配全局替换：可以将操作系统保留的空闲物理块分配给缺页进程
  - 可变分配局部替换：只能选进程自己的物理块置换
    - 可变分配全局替换----只要缺页就分配新得物理块
    - 可变分配局部替换----要根据发生缺页的频率来动态增加或者减少进程的物理块
- 调入页面的时机
  - 预调页策略：一次调用若干个相邻页面，运行前调入，主要用于进程的首次调入
  - 请求调页策略：运行时缺页再调入内存
- 从何处调页
  - 对换区：快，采用==连续分配方式
  - 文件区：慢，采用==离散分配方式
- 抖动（颠簸）现象
  - 刚刚换出的又要换入，刚刚换入的又要换出
  - 主要原因：分配给进程的物理块不够

- 工作集
  - 指在某段时间间隔里，进程实际访问页面的集合
  - 工作集大小可能小于窗口尺寸
  - 驻留集的大小不能小于工作集的大小，否则进程运行过程中将频繁的抖动缺页