中国科学院大学
University of Chinese Academy of Sciences

按需加载与页替换

中国科学院大学计算机学院

2025-11-26

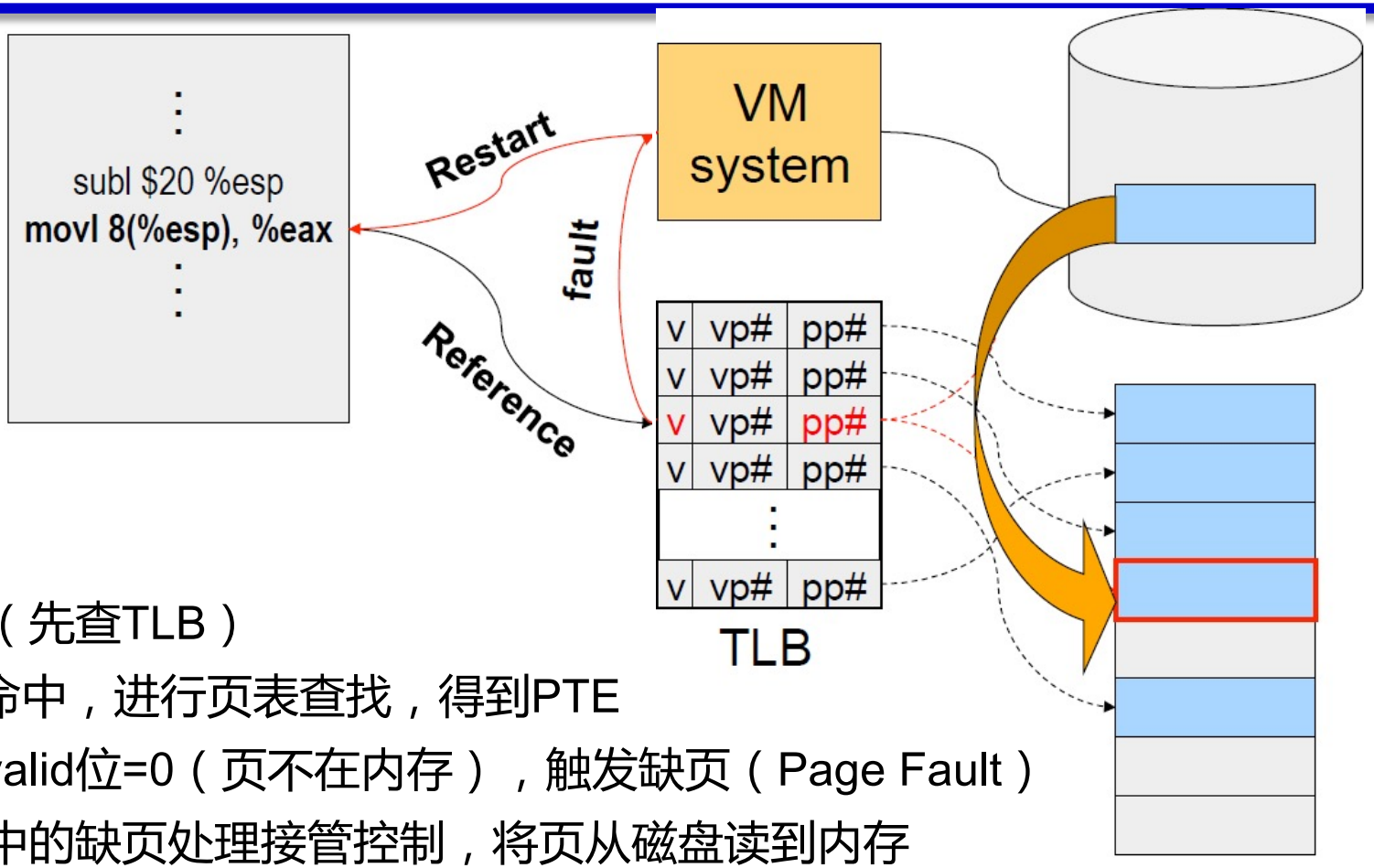
内容提要

- 缺页与换页
- 页替换算法

进程加载

- 简单办法
  - 将整个进程加载进内存 → 运行它 → 退出
- 问题
  - 慢（特别是对于大进程）
  - 浪费空间（一个进程并不是每时每刻都需要所有的内存）
- 解决办法
  - 按需加载页：只将实际使用的页加载进内存
  - 换页：内存空间有限，只放频繁使用的那些页
- 机制
  - 一部分虚存映射到内存，另一部分虚存映射到磁盘

缺页处理

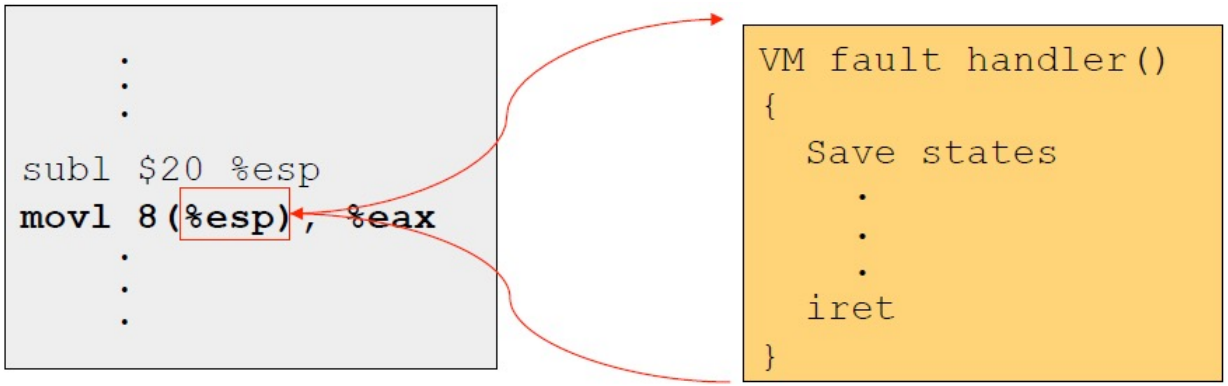


步骤：

- 内存访问（先查TLB）
- 若TLB不命中，进行页表查找，得到PTE
- 若PTE的valid位=0（页不在内存），触发缺页（Page Fault）
- 虚存管理中的缺页处理接管控制，将页从磁盘读到内存
- 更新PTE：填入pp#，将valid位置1
- 把PTE加载进TLB
- 重新执行该指令：重新进行内存访问

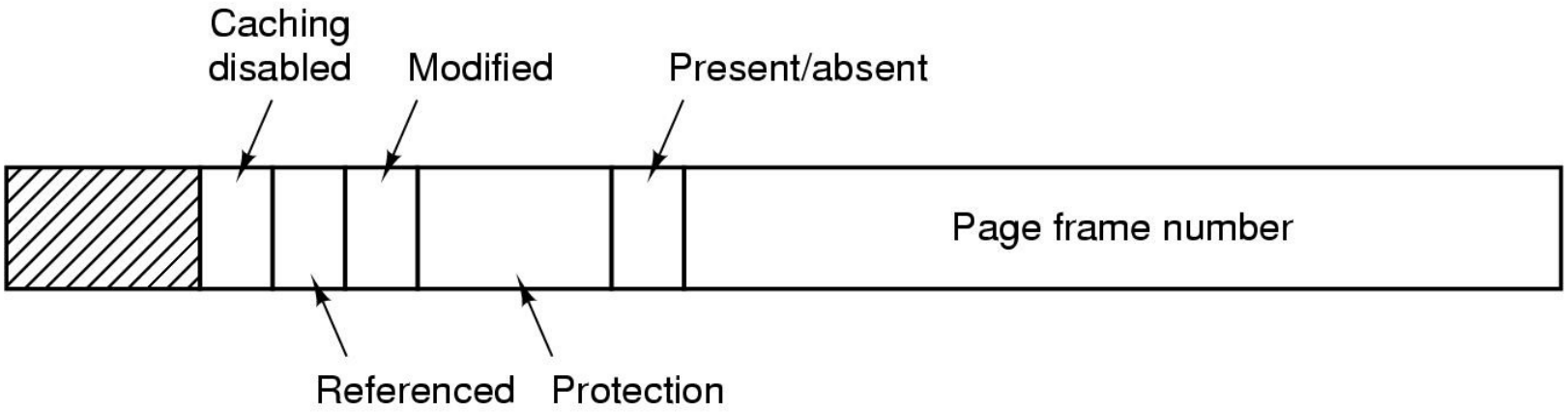
缺页处理

- 发生缺页后，如何切换到缺页处理？
  - 缺页可能发生在一条指令执行的中途
  - 应用程序透明：必须让用户程序不感知缺页
  - 需要保存状态并从断点处继续执行



- 页替换
  - 需要的页不在内存里 ➔ 需换入 ➔ 需为它分配一个页框
  - 可能此时没有空闲页框
  - VM需要进行页替换

回顾PTE



- 用于换页的位
  - V位：在不在内存（Valid），把页加载进内存时置位
  - M位：修改标志位（Modify），当对该页中的某个位置进行写时置位
  - R位：访问标志位（Reference），当访问该页中的某个位置时置位

缺页处理

- 进程A发生缺页，发生缺页的页记为VP
  - 陷入内核，保存进程A的当前状态：PC、寄存器
  - 调用OS的缺页处理模块
    - 检查地址和操作类型的合法性，不合法，则给进程A发signal或者kill
    - 为VP分配一个物理页框 ，记为PP
      - 如果有空闲页框PP1，则用它，PP=PP1
      - 如果没有空闲页框，选择一个状态为used页框 PP2
        - » 如果它是脏的（M位=1），则把它写回磁盘
        - » PTE表项valid位 置为0，flush TLB表项
        - » 写回完成后，PP=PP2

      页替换
    - 找到VP对应的磁盘页，把它读到这个页框（PP）中
    - 修改VP的PTE：填入PP#，将valid位置为1，并把该PTE加载进TLB
  - 恢复进程A的状态，重新执行发生缺页的指令
- 通用数据结构
  - 空闲页框链表
  - 一张映射表：页框→PID和虚页

内容提要

- 缺页与换页
- 页替换算法

换入谁

- 换入发生缺页的页
  - 简单，但是每次换页都有很大的开销
- 每次换入更多的页
  - 如果多换入的页以后会用，可减少缺页发生
  - 如果以后不会用它们，浪费空间和时间
  - 实际系统会进行预取一定量的页（例如，32、64页）
- 由应用控制换入谁
  - 一些系统支持用户控制预取
  - 但很多应用也不知道未来哪些页会用

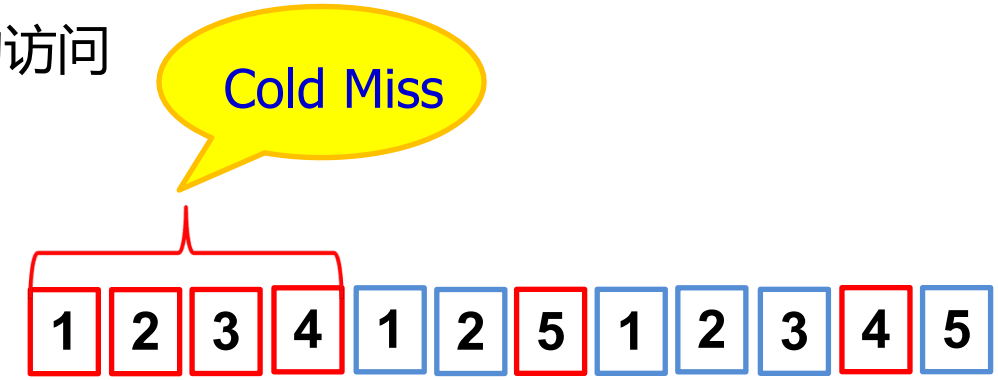
替换谁

页替换算法

- 随机选一页
- 最优算法（MIN）
- NRU (Not Recently Used)
- FIFO (First-In-First-Out)
- FIFO with second chance
- NFU (Not Frequently Used)
- LRU (Least Recently Used)
- Aging (approximate LRU)
- Clock
- Working Set
- WSClock

最优算法（MIN）

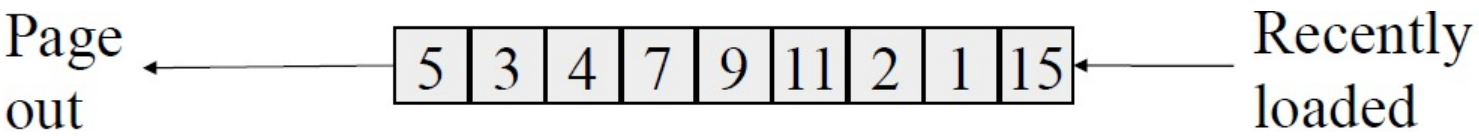
- 算法
  - 替换在未来最长一段时间里不用的页
  - 前提：知道未来所有的访问
- 例子
  - 内存大小为4个页
  - 访问序列：
  - 6次缺页



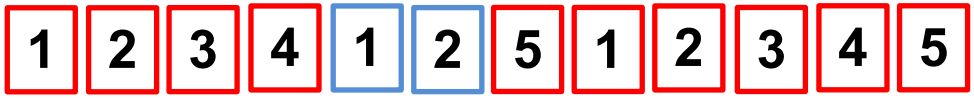
- 好处
  - 最优方案，可作为一种离线分析手段
- 坏处
  - 在线系统无法采用，因为不知道未来的访问顺序

FIFO

- 算法
  - 选择最老的页扔掉



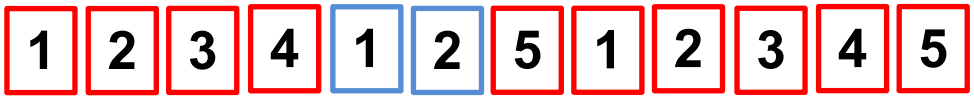
- 例子
  - 内存大小为4个页
  - 访问序列
  - 10个缺页



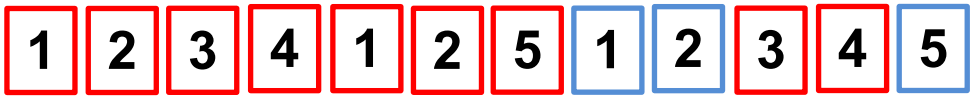
- 好处
  - 实现开销最小
- 坏处
  - 频繁使用的页被替换

增加页的数量 → 减少缺页？

- 假设内存大小为4个页
  - 算法：FIFO替换
  - 访问序列
  - 10个缺页



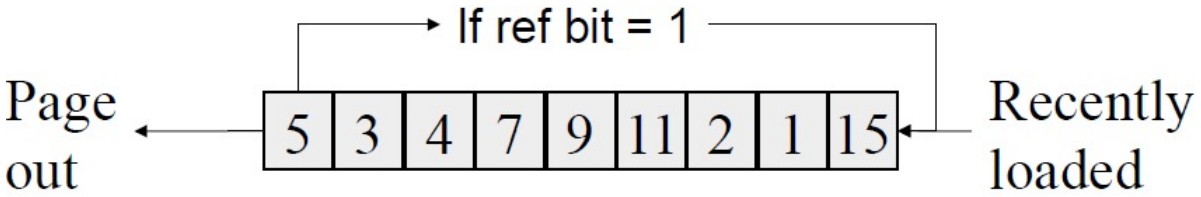
- 假设内存大小为3个页
  - 算法： FIFO替换
  - 同样的访问序列
  - 9个缺页



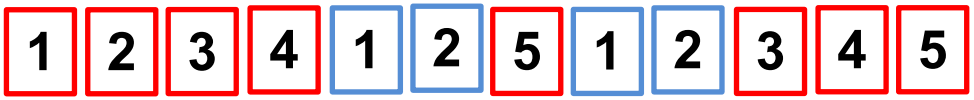
- “Belady’s anomaly”现象
  - Belady, Nelso, Shedler 1969
  - 采用FIFO算法时，有时会出现分配页数增多，缺页率反而升高的异常现象

Second Chance（有第二次机会的FIFO ）

- 核心思想
  - 尽量让频繁使用的页留在内存，不被替换
  - 替换时给访问过的页第二次机会，在内存呆更长时间
- 算法
  - 检查最老页的R位，如果为0，替换它
  - 如果为1，将它清0，并把它移动队尾，继续查找



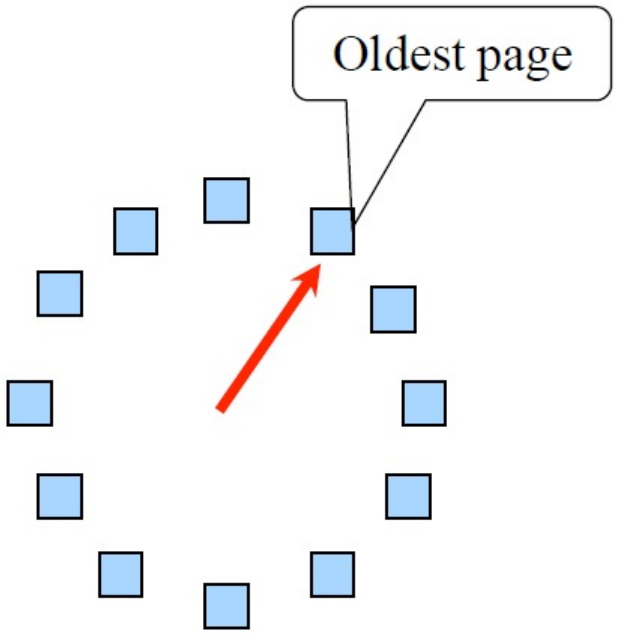
- 例子
  - 内存大小为4个页
  - 访问序列
  - 8个缺页



- 好处
  - 实现简单
- 坏处
  - 最坏情况时可能需要很长时间

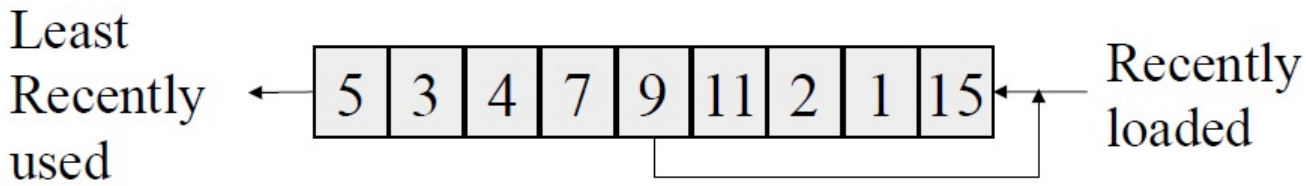
Clock

- 改进Second Chance的替换开销
  - 移动开销
- 算法：把所有页框组织成环形链表
  - 用一个表针指向最老的页
  - 发生缺页时，按表针走动方向来检查页
- 第二次机会
  - 如果其R位为1，将其置为0，且表针向前移一格
  - 如果其R位为0，替换它，且表针向前移一格
- 与Second Chance算法相比
  - 更加高效
- 如果内存很大
  - 轮转一遍需要很长时间

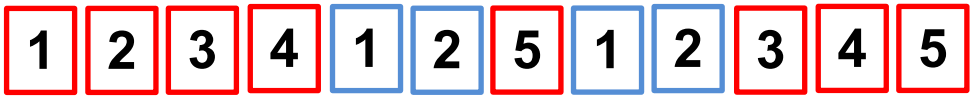


LRU

- 替换最长时间没有使用的页
  - 将所有页框组织成一个链表
  - 前端为最久未访问的页（LRU端）：替换的页
  - 后端为最近刚访问的页（MRU端）：新加载的页和命中的页
  - 每次命中将页重新插入MRU端



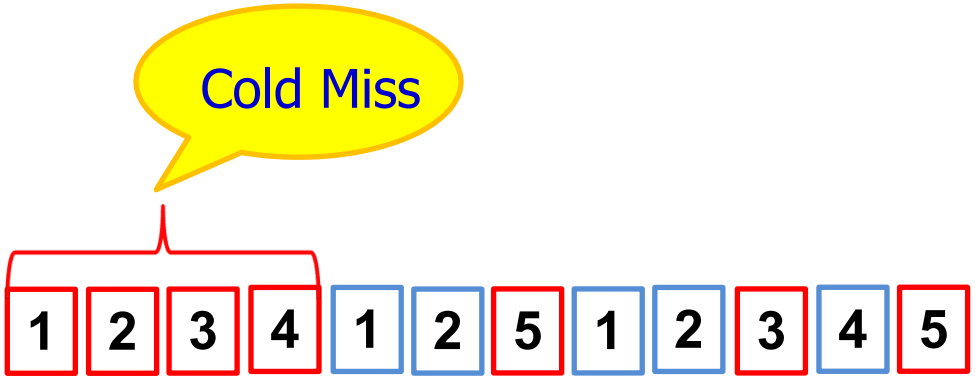
- 例子
  - 内存大小为4个页
  - 访问序列
  - 8个缺页



- 好处
  - 对MIN算法的很好近似
- 坏处
  - 实现困难

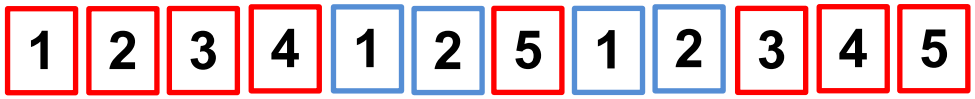
NRU（Not Recently Used)

- 算法
  - 按下面顺序，随机选择一个页
    - 未访问过 且 未修改过
    - 未访问过 且 修改过
    - 访问过 且 未修改过
    - 访问过 且 修改过
- 例子
  - 内存大小为4个页
  - 访问序列
  - 7个缺页
- 好处
  - 容易实现
- 坏处
  - 需要扫描内存中所有页的R位和M位



LFU(Least Frequent Used)

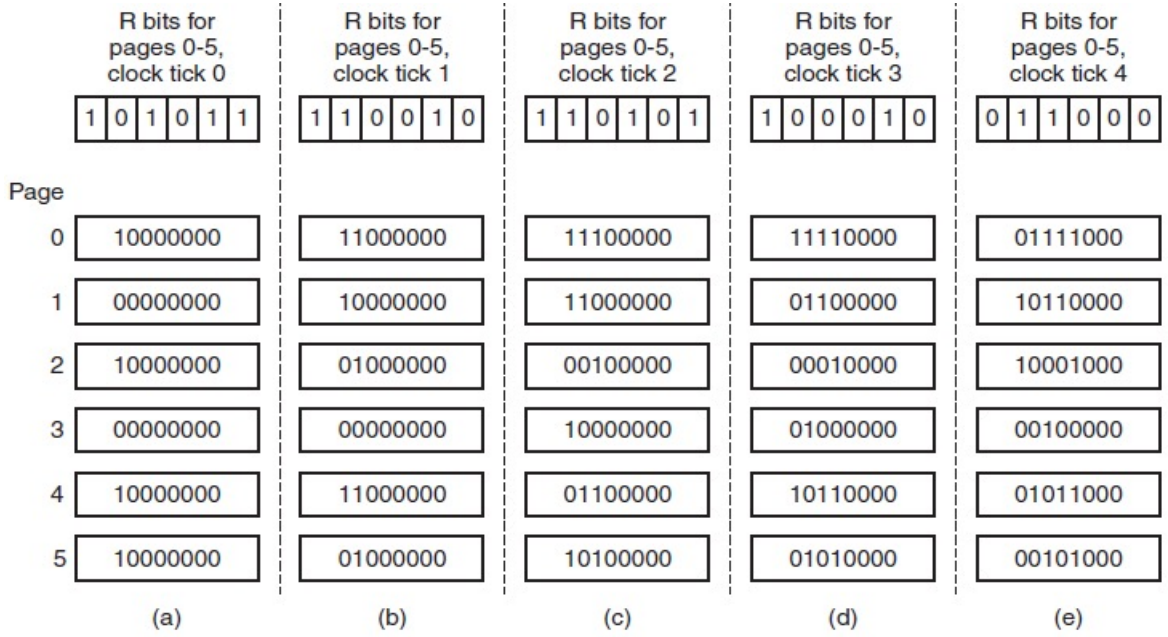
- LFU：记录每个页的访问次数，替换访问次数最少的页
  - 每页有一个访问计数器，用软件模拟
  - 每个时钟中断时，所有页的计数器 分别与它的R位值相加
- 例子
  - 内存大小为4个页
  - 访问序列
  - 8个缺页



- 坏处
  - never forget
  - 过去访问频繁、现在不访问的页，替换不出去

Aging

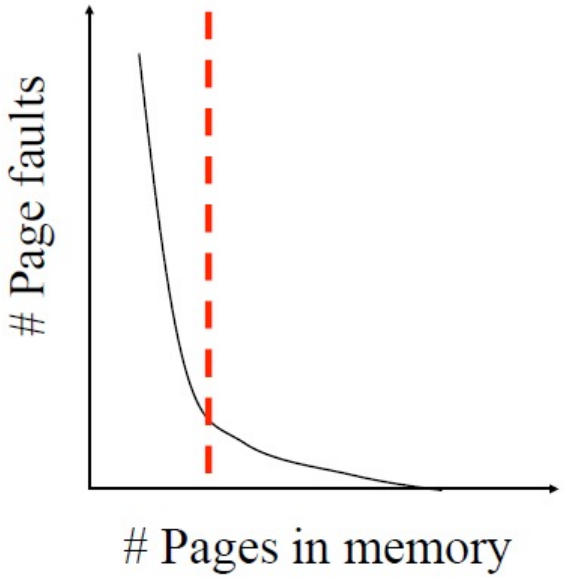
- 消除过去访问的影响
  - 每个时钟中断时，先将所有页计数器右移1位，再将每页计数器最高位与该页的R位相加
  - 替换时，选择计数器值最小的页



- Aging与LRU的主要差别
  - 记录下来的历史更短（计数器长度）
  - 无法区分访问的先后顺序（同一tick内）
- 多少位才够用？
  - 实际使用中，8位就工作得很好

程序的行为（Denning, 1968）

- 80/20原则
  - > 80%的访问只涉及20%的内存空间
  - > 80%的访问来自20%的代码
- 空间局部性
  - 相邻的页很可能会被访问
- 时间局部性
  - 被访问的页很可能在不远的将来再被访问



工作集（Working Set）

- 主要思想（Denning 1968, 1970）
  - 工作集被定义为在最近K次访问的那些页
  - 把工作集放进内存能大大地减少缺页
- 工作集的近似
  - 一个进程在过去T秒钟里使用的页
- 一个算法：记录页的“上次访问时间”
  - 在缺页处理时，扫描该进程所有的页
  - 如果R位为1，将该页的上次访问时间设置为当前时间
  - 如果R位为0，计算当前时间与上次访问时间之差Δ
    - 如果Δ>T，该页在过去T秒里没有访问过，则替换它
    - 否则，检查下一页
  - 将发生缺页的页加入工作集

WSClock

- 将页框组织成环形链表（类似Clock）
- 按表针走动的顺序来检查页
- 如果R位为1
  - 将R位置为0，该页的上次访问时间设置为当前时间
  - 检查下一页
- 如果R位为0
  - Δ = 当前时间 – 上次访问时间
  - 如果Δ<=T，该页在过去T秒里访问过，检查下一页
  - 如果Δ>T，该页在T秒里没有访问过，而且M位为1
    - 将该页写回加入写回链表（异步进行写回），并检查下一页
  - 如果Δ>T，该页在T秒里没有访问过， 且M位为0
    - 替换该页

算法对比

算法	特点
MIN	最优算法，但无法在实际系统中实现
FIFO	没有考虑重复访问的情况，仅根据第一次访问时的顺序进行替换
Second Chance	在FIFO基础上考虑了重复访问
Clock	基于Second Chance，可以更加高效实现
LRU	考虑了访问时效性，实现时有频繁链表操作，开销较大
NRU	粗粒度近似LRU，只区分有无访问，同时优先保留脏页；与LRU比，时效性考虑较少
LFU	考虑访问的次数，将访问次数多的数据留在内存中；容易受短期高频访问的影响
Aging	近似LFU，更容易实现；记录窗口短；相同访问次数时，无法区分时效性
Working Set	实现开销大
WSClock	基于WS优化实现时的数据结构（环形链表），实现开销小

算法对比

- LRU很好但实现困难，Aging是一个较好的近似LFU
- Clock被认为是一个很好的实际解决方案
- 所有替换算法都不优化由Cold miss带来的缺页