OPERATING SYSTEMS

Virtual Memory Policies

Operating Systems

郑贵锋 博士
中山大学计算机学院
zhenggf@mail.sysu.edu.cn
https://gitee.com/code_sysu

中山大學 SUN YAT-SEN UNIVERSITY

- 目录
  - 虚拟内存策略需要决定
    - 读取策略
    - 放置策略
    - 置换策略
    - 常驻集管理
    - 加载控制

Virtual Memory Policies

- **内存管理软件**
  - 内存管理软件取决于硬件是否支持分页或分段，或者两者都支持。
  - 纯粹的分段系统很少见。段通常是分页的——内存管理问题就是分页的问题。
  - 因此，我们将集中讨论与分页相关的问题。
  - 为了获得良好的性能，我们需要低缺页中断率。

Virtual Memory Policies

- **读取策略**
  - 确定页面何时应读入主内存。两种常用政策：
    - 当引用页面上的某个位置时，请求调页才将页面读入主存（仅按需分页）。
      - 进程第一次启动时出现许多缺页中断，但随着更多页面被读入，缺页中断应会减少。
    - 预先调页读入了预期使用的页面：
      - 引用局部性表明，一次性读入磁盘上连续的页面更有效。
      - 效率不确定：读入的额外页面"通常"未被引用。
      - 进程首次启动时（需程序员指定需要的页）采用；发生缺页中断时（对程序员透明）采用，更可取。

Virtual Memory Policies

- **放置策略**
  - 确定进程块在实际内存中的驻留位置。
  - 对于分页（和段页式）：
    - 硬件决定页面的放置位置：选择的帧位置无关紧要，因为所有内存帧都是等效的（一样大小，不是问题）。
  - 对于纯分段系统：
    - 第一次匹配、下一次匹配……都是可能的选择（真正的问题）。

Virtual Memory Policies

- **置换策略**
  - 当新页面被调入时，处理在主内存中选择一个页面以被置换。
  - 只要主内存已满（没有可用的空闲帧），就会发生这种情况。
  - 经常发生，因为操作系统试图将尽可能多的进程（页面）调入主内存以提高多道程序的程度。
  - 并非主内存中的所有页面都可以被选择置换。
  - 某些帧已锁定（无法调出）：
    - 内核的大部分、以及关键控制结构、I/O缓冲区都保存在锁定的物理帧上。
  - 操作系统需要作出决策，计划可被置换的页面集应为：
    - 仅限于出现缺页中断的进程的页面集。
    - 或者，未锁定帧中所有页面的集合（包含所有页均在内存的进程）。

Virtual Memory Policies

- **置换策略**
  - 计划可被置换的页面集的决定，与常驻页面集管理策略有关：
    - 要为每个进程分配多少页帧？
  - 无论计划可被置换的页面集是什么，置换策略都执行算法，在该集中选择被置换的页面。

Virtual Memory Policies

- **帧的分配**
  - 常驻集管理
    - 操作系统必须决定分配给进程的页面帧数：
      - 如果分配的帧太少，则缺页中断率高。
      - 如果分配的帧太多，则多道程序级别较低。
  - 常驻集大小
    - 固定分配政策：
      - 分配随时间保持不变的固定帧数：
        - 该数量在加载时确定，并取决于应用程序的类型。
    - 可变分配政策：
      - 分配给进程的帧数可能随时间而变化：
        - 如果缺页中断率较高，则可能会增加。
        - 如果缺页中断率非常低，则可能会降低。
      - 需要更多的操作系统开销来评估活动进程的行为。

Virtual Memory Policies

- **帧的分配**
  - 固定分配
    - 平均分配
      - 例如，如果有100帧和5个进程，则给每个进程20帧。
      - 保留一些作为空闲帧缓冲池
    - 比例分配
      - 根据进程的大小进行分配。
      - 随着多道程序程度的提高，进程大小也会发生变化
        - 设 s_i 为进程 P_i 的大小，m为总帧数
        - 那么所有进程大小 $S = \Sigma s_i$
        - 给进程 P_i 的分配 $a_i = (s_i / S) \times m$
      - 实例

$s_1 = 10$, $s_2 = 127$, $m = 64$
$S = \Sigma s_i = 10 + 127 = 137$
$a_1 = (10 / 137) \times 64 \approx 5$
$a_2 = (127 / 137) \times 64 \approx 59$

Virtual Memory Policies

- **帧的分配**
  - 固定分配
    - 优先级分配
      - 使用优先级而不是大小的比例来分配。
      - 如果进程P_i产生缺页中断：
        - 从自己的帧中选择一个帧进行置换。
        - 从优先级较低的进程中选择帧进行置换。

Virtual Memory Policies

- **帧的分配**
  - 置换范围
    - 置换范围是发生缺页中断时要考虑置换的帧集。
    - 全局置换 – 进程从所有帧集中选择置换帧；一个进程可以从另一个进程获取帧
      - 但是，进程的执行时间可能会有很大的差异
      - 因吞吐量更大，更常用
    - 局部置换 – 每个进程仅从分配给自己的帧集中选择置换
      - 每个进程的性能更加一致
      - 但可能有未使用的已分配内存
    - 我们将考虑置换范围和驻留集大小策略的可能组合。

Virtual Memory Policies

- **帧的分配**
  - 固定分配+局部范围
    - 每个进程分配固定数量的页面：
      - 在加载时确定；取决于应用程序类型。
    - 发生缺页中断时，考虑置换的页面帧是缺页中断进程的局部帧：
      - 因此，分配的帧数是恒定的。
      - 可以使用以前的置换算法。
    - 问题：难以提前确定分配帧的正确数目：
      - 如果太低：缺页中断率将很高。
      - 如果太大：多道程序级别将太低。
  - 固定分配+全局范围
    - 不可能实现：
      - 如果所有未锁定的帧都是置换的候选帧，则分配给进程的帧数将随着时间的推移（进程的变化）而变化。

Virtual Memory Policies

- **帧的分配**
  - 可变分配+全局范围
    - 易于实现——被许多操作系统（如Unix SVR4）采用。
    - 将维护一个空闲帧列表：
      - 当进程发出缺页中断时，将为其分配一个此列表的空闲帧
      - 因此，分配给缺页中断进程的帧数增加。
      - 对于将失去帧的进程的选择是任意的；这远非最优做法。
    - 页面缓冲可以缓解这个问题，因为如果页面很快再次被引用，它可被重新使用。

Virtual Memory Policies

- **帧的分配**
  - 可变分配+局部范围
    - 可能是最佳组合（由Windows NT使用）。
    - 根据应用程序类型，在加载时为新进程分配一定数量的帧：
      - 使用预先调页或请求调页来填充分配。
    - 发生缺页中断时，从发生缺页中断的进程的常驻集中选择要置换的页面。
    - 定期重新评估提供的分配，并增加或减少分配，以提高整体性能。

Virtual Memory Policies

- **帧的分配**
  - 工作集策略
    - 工作集策略是一种基于引用局部性假设的局部范围可变分配方法。
    - 在时间 t, 进程的工作集 W(D, t) 是在最后 D 个虚拟时间单位中引用的页面集：
      - 虚拟时间 = 进程执行时经历的时间
      - D 是一个时间窗口。
      - W(D, t) 是程序局部的近似。
    - 进程的工作集在开始执行时首先增长。
    - 然后因局部性原则而稳定。
    - 当进程向新的局部区域过渡时，它再次增长：
      - 直到工作集包含来自两个局部的页面为止。
    - 然后在新的局部呆了足够长的时间后下降。

Virtual Memory Policies

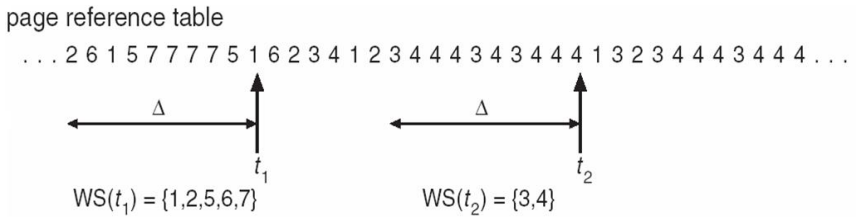
- **帧的分配**
  - 工作集策略
    - 工作集概念建议采用以下策略来确定常驻集的大小：
      - 监视每个进程的工作集。
      - 定期从进程的常驻集中删除那些不在工作集中的页面。
      - 当进程的常驻集小于其工作集时，为其分配更多帧：
        - 如果没有足够的可用帧，则暂停进程（直到有更多的帧可用），即，只有当进程的工作集在主内存中时，才可以执行进程。

Virtual Memory Policies

- **帧的分配**
  - 工作集策略
    - 工作集模型
      - $\Delta \equiv$ 工作集窗口 $\equiv$ 固定次数的页面引用（虚拟时间单位）
        - 例如：10,000条指令
      - 进程 P_i 的工作集大小 $WSS_i \equiv$ 最近 Δ 中引用的总页数
        - 如果 Δ 太小，它不能覆盖整个局部。
        - 如果 Δ 太大了，它将包括几个局部。
        - 如果 $\Delta = \infty \Rightarrow$ 将包含整个程序。
      - $D = \Sigma\ WSS_i \equiv$ 总需求帧
      - 如果 $D > m$, 其中m为总可用帧数 $\Rightarrow$ 抖动
        - 防止抖动策略：挂起其中一个进程

Virtual Memory Policies

- **帧的分配**
  - 工作集策略
    - 工作集模型



Virtual Memory Policies

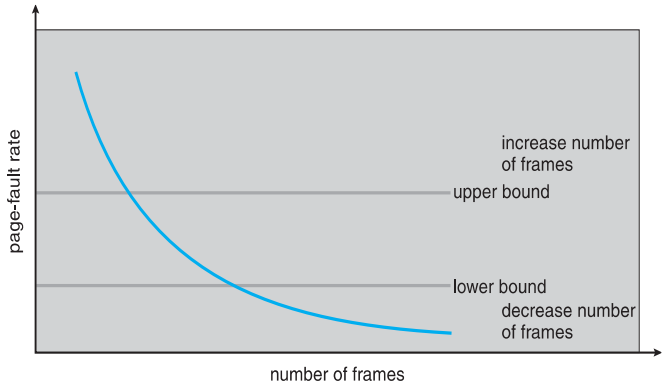
- **帧的分配**
  - 工作集策略
    - 跟踪工作集
      - 使用间隔计时器+引用位进行近似。
      - 例子： Δ = 10,000次引用
        - 计时器每5000次引用中断一次。
        - 在内存中为每页保留2位引用历史。
        - 当计时器中断时，将所有页的引用位复制（到历史位）并清除
        - 如果引用历史中的一位为1 $\Rightarrow$ 工作集中的页面
          - 同时检查引用位，可确定最近10000~15000次引用中是否引用过该页
      - 为什么这不完全准确？
        - 未知5000次引用的中间是否被引用
      - 改进：历史位增加到10位，每1000次引用中断一次。

Virtual Memory Policies

- **帧的分配**
  - 工作集策略
    - 此工作集策略的实现问题：
      - 对每个进程的工作集进行测量是不切实际的：
        - 需要在每次内存引用时对引用的页加上时间戳。
        - 需要为每个进程维护一个按时间顺序排列的引用页面队列。
      - 总需求帧数 D 的最优值未知且随时间变化。
    - 解决方案：与其监视工作集，不如监视缺页中断率！

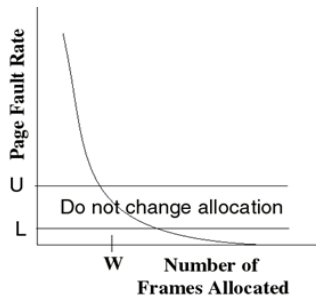
Virtual Memory Policies

- **帧的分配**
  - 缺页中断频率(PFF)方案
    - 确定"可接受"缺页中断率：
      - 若实际比率过低，进程将减少帧数。
      - 若实际比率过高，进程将增加帧数。



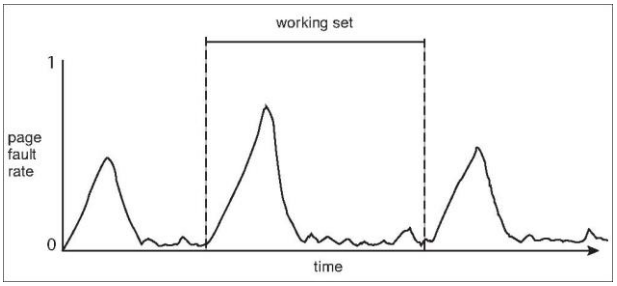
Virtual Memory Policies

- **帧的分配**
  - 缺页中断频率 (PFF) 策略
    - 定义缺页中断率的上限 U 和下限 L
    - 如果频率高于 U, 则为进程分配更多帧。
    - 如果频率小于 L, 则分配较少的帧。
    - 常驻集大小应接近工作集大小 W.
    - 如果 PFF > U 并且没有更多可用帧，将挂起该进程。



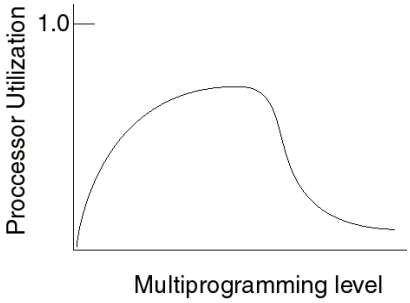
Virtual Memory Policies

- **帧的分配**
  - 工作集和缺页中断率
    - 进程的工作集与其缺页中断率之间的直接关系
    - 工作集随时间而变化
    - 缺页中断率随时间推移起伏：新局部的高峰，之后的低谷



Virtual Memory Policies

- **加载控制**
  - 确定将驻留在主存中的进程数（即多道程序级别）：
    - 进程太少：所有进程都被阻塞的情况经常发生，此时处理器将处于空闲状态
    - 太多进程：每个进程的驻留大小将太小，缺页中断将导致抖动



Virtual Memory Policies

- **加载控制**
  - 工作集或缺页中断频率算法隐式包含加载控制：
    - 只有驻留集足够大的进程才允许执行。
  - 另一种方法是明确调整多道程序级别，使缺页中断之间的平均时间等于处理缺页中断的时间：
    - 性能研究表明，这是处理器使用率最高的点。

Virtual Memory Policies

- **加载控制**
  - 进程挂起
    - 显式加载控制有时要求挂起（换出）进程。
    - 可能的挂起选择标准：
      - 缺页中断进程
        - 可能此进程的工作集不在主内存中，因此将被阻塞
      - 最后一个激活的进程
        - 此进程的工作集最有可能未驻留内存。
      - 具有最小驻留集的进程
        - 此进程需要最少的重新加载代价，不利于局部性较小的进程。
      - 最大空间进程
        - 将产生最多的空闲帧。