

Operating System - P53

P59 - ~~sendotol~~

页面置换算法
特征：不可实现，但可以用作基准
在当前页面中置换最后要访问的页面
但没有办法判断哪个页面是最后访问的

NRU(最近未使用)算法	LRU的很粗糙的近似 从编号最小的类中随机选择置换一个页面	根据R位和M位将页面4类 易于实现但性能不好
FIFO(先进先出)算法	可能会抛弃重要页面 淘汰最老但可能仍在使用的页面	维护链表记录页面装入内存顺序
第二次机会算法	相比FIFO算法有较大改善 如果正在使用则保留页面	找出页面前先检查页面是否正在使用 相比FIFO大大提高了性能
时钟算法	现实的算法 用环形链表代替普通链表	第二次机会算法的另一种实现 性能特征相同，执行时间更少
LRU(最近最少使用)算法	很优秀，但很难实现 通常需要特定的硬件实现 如果缺少硬件则无法使用	性能不是非常好
LFU(最不经常使用)算法	LRU的相对粗糙的近似 非常类似于LRU的有效算法	更有效地实现 实际应用中最重要的算法之一
老化算法	实现起来开销较大	使用了工作集 具有良好的性能
工作集算法	好的有效算法	工作集算法的一种变体 有良好性能且可高效地实现
工作集时钟算法	在可运行的进程间动态地分配页框，因此分配给各进程的页框数随时间变化	实际应用中最重要的算法之一

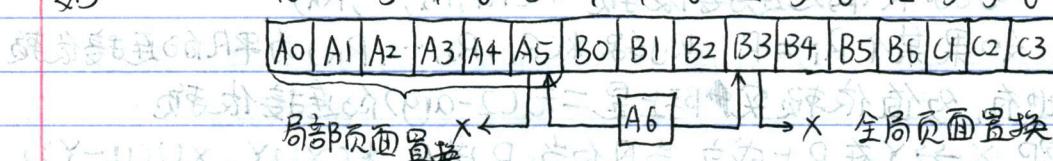
分配策略：指如何在相互竞争的可运行进程之间分配内存

局部(local)页面置换算法，只考虑发生缺页中断进程的其他页面

可以有效地为每个进程分配固定的内存片段

全局(global)页面置换算法，考虑所有当前存在于内存中的页面

如：10 7 5 4 6 3 9 4 6 2 5 6 12 3 5 6



Operating

System - P54

unit 10: 9

229 - step 3

比较全局算法与局部页面置换算法。全局算法的优点：简单，易于实现。

在通常情况下全局算法工作得更好，尤其是在工作集大小固定时。

特别是当工作集的大小随进程运行时间发生变化时。

如果使用局部算法，即使存在大量空闲页框，工作集增长依旧会导致真页缺

（即从内存中调入新页框以满足工作集增长）

使用全局算法时，需要系统不断确定应该给每个进程分配自己的页框数量。

方法一：监测工作集的大小，工作集大小由“老化”位指出。

但是注意这个方法无法防止颠簸。

由于工作集的大小可能几ms内发生变化。

而对应的“老化”位可能要经历一定的时钟滴答数才会更新。

从而使得“老化”位无法真实地反映工作集大小。

方法二是使用一个为进程分配页框的算法，并有不同的算法选择。

算法一是定期确定进程运行的数目。

为所有运行进程分配数量相等的页框份额。

剩余部分放入一个公共池中，以应对可能发生的缺页中断。

问题是为大小相差很大的不同进程分配相同的份额是不合理的。

算法二是按照进程大小的比例来分配相应比例的页框数目。

但是问题是，进程大小并不必然反映其所需的工作集大小。

如某些机器上仅使用一两个操作数的指令可能需要使用多个页框。

而如果分配的页框数过少，则可能导致指令无法执行。

一个改进方法是对每个进程规定一个最小的页框数。

则保证了过小的进程也可以顺利运行。

但产生了一定程度的空闲页框浪费。

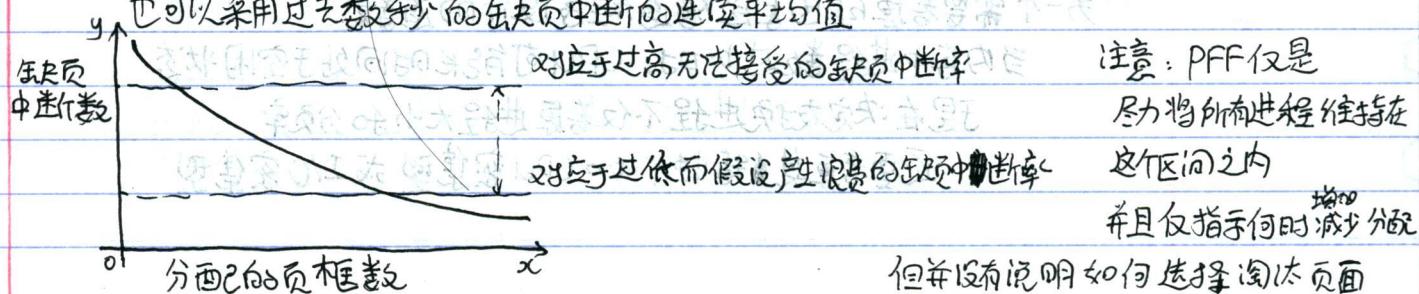
缺页中断率 (page fault frequency, PFF)，用于全局页面置换算法下管理内存动态分配的方法。

基于假设，对于一类页面置换算法（如 LRU），

进程的缺页中断率应随着分配页框数目增加而降低。

对于测量缺页中断率，可以直接采用最近一秒内进程的缺页中断数。

也可以采用过去数秒的缺页中断的连续平均值。



但并没有说明如何选择淘汰页面。

Operating System - P55

对于页面置换算法，有些既适用于局部页面置换算法，也适用于全局页面置换算法

如 先进先出页面置换算法 (FIFO)

在全局算法中置换所有内存中最老的页面

在局部算法中置换当前进程中最老的页面

相似地有最近最少使用页面置换算法 (LRU)

相对的，有些页面置换算法只在局部策略有意义

如 工作集和 WSClock 算法，针对某些特定进程

于是必须应用于这些进程的上下文 (context) 中

实际上也不存在针对整个计算机的工作集

当使用所有工作集的并集作为机器工作集时，会损失局部特性，从而降低性能

负载控制

只要所有进程的组合工作集超过内存容量，就会产生颠簸

即使使用最优页面置换算法并假设进程采用理想的全局页框分配

如 缺页中断率算法 (PFF) 指出，

一些进程需要更多内存，但没有进程需要更少的内存

唯一解决方案是从内存中暂时地去掉一些进程

将一部分进程交换到磁盘，以减少竞争内存的进程数

被交换到磁盘的页框可以由处于 ~~真实~~ 空闲状态的进程分享

如果颠簸停止，系统可以如此运行直到再次出现颠簸

如果颠簸持续，则继续换出进程，直到颠簸停止

注意即使使用了分页，也需要交换

区别在于此时交换用于减少对内存的潜在需求，而非收回页面

将进程交换到磁盘借用了两级调度思想

当一些进程交换到磁盘，则用一个短周期调度程序调度剩余进程

通过将恰好好的进程交换出去可以获得可接受的缺页中断率

此时一些进程周期性地从磁盘调入

另一些进程周期性地换出到磁盘

另一个需要考虑的因素是多道程序设计的道数

当内存中进程数过多时，CPU 可能长时间处于空闲状态

于是在决定交换进程不仅考虑进程大小和频率

还要考虑其特性，如 CPU 密集型或 I/O 密集型

Operating System - P56

页面大小 在 Operating system 中，页面大小是可以自定义的参数
即使硬件设计固定了页面大小，操作系统可以通过分配已连接页面来作为更大的页面
从结果上看，页面大小不存在全局最优，即没有最佳的页面大小

对于选择相对较小的页面，内部碎片 (internal fragmentation)，指对于一个随机的正文段、数据段或堆栈段很可能不会恰好装满整个页面，即在最后一个页面有多余的空间

平均而言，最后一个页面可能有一半的空间为空

这种被浪费的空间称为内部碎片

当内存中有 n 个段，页面大小为 p 时，平均而言有 $\frac{n}{2}p$ 字节以内部碎片的形式浪费

另外考虑一个程序，分成 8 个阶段顺序地执行 每个阶段需要 4KB 内存
则当页面大小为 32KB 时，则始终分配 32KB

但如果页面大小为 4KB，则只需分配 4KB
可以保证程序顺序地执行每个阶段

但是更小的页面意味着程序需要更多的页面，进而需要一个更大的页表
由于内存与磁盘间的传输是以页面为单位的
而传输的大部分时间会花在寻道和旋转延迟
所以传输大页面和小页面的时间大致相同

更小的页面更容易消耗 TLB 空间
由于 TLB 表项相对稀缺，且对于性能更重要，因此条件允许时大页面是值得的

在部分机器上，CPU 每次切换进程时都必须把新进程的页表装入硬件寄存器
更小的页面意味着更长的装入页面寄存器时间，页表占用的空间也更大
有时操作系统会为系统的不同部分使用不同的页面大小
如对内核使用大页面，而对用户进程使用小页面

假设进程平均大小 s 字节，页面大小 p 字节，每个页表项需要 e 字节
则每个进程需要的页面数大约为 s/p ，而最后一个页面浪费为 $p/2$
于是有总开销 $se/p + p/2 = F(p)$
对 p 求导有 $F'(p) = -se/p^2 + \frac{1}{2} = 0$ ，则有 $p = \sqrt{2se}$
得到仅考虑页表所需内存和碎片浪费的最优页面大小