

# 颠簸现象与工作集

团队成员：杨雨婷 王梓柔 刘韦晶 谭楷睿 王越洋

时间：2024.5.13

汇报人：王越洋



# 目录

CONTENTS

01 颠簸现象的概念

02 颠簸的原因

03 工作集的概念

04 工作集如何解决颠簸问题

在计算机系统中，颠簸/抖动现象特指请求分页式虚拟存储器系统中的一种关键问题，即由于频繁地发生缺页中断，导致系统性能急剧下降的现象。

## 1. 多道程序度过高

当同时运行的进程数过多，且这些进程频繁访问的页面数高于可用的物理内存块数时，就可能导致进程运行时频繁缺页。系统需要不断地从外存中调入页面到内存中，以满足进程的执行需求，这就导致了页面在内存与外存之间的频繁调度。

## 2. 缺页率过高

缺页率与系统为进程分配的物理内存块的多少（即驻留集的大小）密切相关。当物理内存块数小于某个数值时，减少一块都会对缺页率有较大影响。随着缺页率的增加，系统需要花费更多的时间来调度页面，从而导致颠簸现象的发生。



## 3. CPU利用率低

当CPU利用率较低时，调度程序可能会增加多道程序度，试图通过引入新进程来提高CPU的利用率。然而，如果新引入的进程同样需要频繁地访问外存中的页面，那么反而会进一步加剧页面的频繁调度，导致颠簸现象的发生。

为了降低缺页率、防止颠簸现象的发生，操作系统需要采用一些有效的内存管理方法，如工作集策略。工作集是指进程在一段时间内实际访问的页面集合。操作系统可以根据进程的工作集大小来动态地调整分配给进程的物理内存块数，从而确保进程在运行时能够有足够的内存空间来容纳其工作集，减少缺页和页面调度的次数。

工作集（或驻留集）是指在某段时间间隔内，进程要访问的页面集合。经常被使用的页面需要在工作集中，而长期不被使用的页面要从工作集中被丢弃。

工作集的大小对于系统的性能和效率有着重要的影响。通过预取、置换和清理等方式，操作系统可以对工作集进行管理，以提高系统的响应速度和吞吐量。

工作集的概念

工作集的作用

## 工作集模型的原理：

让操作系统跟踪每个进程的工作集，并为进程分配大于其工作集的物理块。如果还有空闲物理块，则可以再调一个进程到内存以增加多道程序数。如果所有工作集之和增加以至于超过了可用物理块的总数，那么操作系统会暂停一个进程，将其页面调出并且将其物理块分配给其他进程，防止出现抖动现象。

**常驻集：**常驻集是指在当前时刻，进程实际驻留在内存当中的页面集合。

- 常驻集与缺页率之间的关系：
  - 当常驻集包含了工作集，也就是工作集都在内存中，缺页较少；
  - 工作集发生剧烈变动时，缺页较多；
  - 当常驻集大小达到某个数目后，再分配物理页帧 也不会有明显下降的缺页率

















































## 工作集的页置换算法:

工作集的页置换算法并不是只有发生缺页中断时才把页换出到外存。随着程序在执行，工作集的窗口会跟着挪动，如果页不在工作集窗口中就会把页换出

追踪之前  $\tau$  个的引用

➤ 在之前  $\tau$  个内存访问的页引用是工作集， $\tau$  被称为窗口大小

Example:  $\tau = 4$  references:

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			<i>c</i>	<i>c</i>	<i>d</i>	<i>b</i>	<i>c</i>	<i>e</i>	<i>c</i>	<i>e</i>	<i>a</i>	<i>d</i>
Pages in Memory	Page <i>a</i>											
	Page <i>b</i>											
	Page <i>c</i>											
	Page <i>d</i>											
	Page <i>e</i>											
Faults												

[https://blog.csdn.net/superSmart\\_Dong](https://blog.csdn.net/superSmart_Dong)

## 常驻集大小与置换范围的配合的三种策略

- 固定分配+局部置换
- 可变分配+全局置换
- 可变分配+局部置换

## 固定分配+局部置换

主要问题是进程开始前要依据进程的类型决定分配多少页面。多了会影响并发水平，少了会使缺页率过高。

## 可变分配+全局置换

这时操作系统会一直维持一定数目的空闲页面，以进行快速置换。这时的主要问题是置换策略的选择，即如何决定哪个进程的页面将被调出。较好的选择是页面缓冲算法

## 页面缓冲算法：

设立空闲页面链表和已修改页面链表。采用可变分配和基于先进先出的局部置换策略，并规定被淘汰页先不做物理移动，而是依据是否修改，分别挂到空闲页面链表或已修改页面链表的末尾；空闲页面链表同时用于物理块分配。当已修改页面链表达达到一定长度如Z个页面时，一起将所有已修改页面写回磁盘，故可显著减少磁盘I/O操作次数

在PBA算法下，内存中的内存块应该分为三类。一类是已经正在使用的内存块，一个是空闲链表上挂载的内存块，另一类是修改页链表。

书上说，如果内存块要被淘汰了，不会被直接换出内存，而是会挂载到 空闲页链表或者修改页链表的尾部，重点来了，这种方法会选择空闲页链表的第一个内存块进行读入，也就是说不管第一个内存块有没有东西，都会直接覆盖，空闲页链表里的页面都是空闲空间，如果你需要，随时都可以用新的页面替换掉里面原有的内容。

不知道题主是否做过一些栈相关的题目，我们在出栈的时候，只是会移动栈顶指针，而不会真的把顶部的数据删除，当我们向下移动栈顶指针的时候，原来的数据就已经算脏数据了，可以随意覆盖。

所以在PBA算法下，只是给了先前被“淘汰”的页面一些缓冲的机会，如果在它被后续的页面替换之前再次被使用，那么就可以直接恢复到进程的驻留集中。

## · 可变分配+局部置换

这时的操作系统也要维持一定数目的空间页面，但是算法的选择却比第二种策略简单，因此它是比较好的策略。

对于页面置换算法来说，“容易”和“简单”两个词有一些不同的含义：

### 局部置换的容易

“容易”指的是对于性能分析而言更容易。因为局部置换只涉及当前进程的页面置换，所以性能分析相对简单，可以更准确地衡量当前进程的性能。通过分析当前进程的页面置换情况，可以更好地了解该进程的内存访问模式、页面置换频率等性能指标，从而更好地优化页面置换算法。

### 全局置换的简单

“简单”指的是实现上更为简单。因为全局置换不需要考虑当前进程的页面集合，只需根据全局的页面置换算法进行置换即可。全局置换不需要维护每个进程的页面置换信息，因此实现更加简单，运行开销也 smaller。

因此，“容易”主要是指性能分析上的方便，而“简单”则是指实现上的简便。



```
While(I got it!){  
    printf ( "Thanks" )  
}
```

