基于存储管理抖动问题的研究

陈 岚
(广东肇庆学院计算机科学系 肇庆526061)

On the Thrashing of Storage Management

CHEN Lan
(Department of Computer Science & Technology, GuangDong ZhaoQing University, Zhaoqing 526061)

Abstract The thrashing problem of virtual storage management is a result of complex factors of the system. It consumes system resources, and undermines the efficiency of the system. The author of this article analyzes the relationship between thrashing problem and working set and replacement algorithm and the behavior properties of programs, and raises the possible solutions to remove the thrashing problem.

Keywords Virtual storage, Thrashing, Working set, Replacement algorithm

1 抖动问题的形成

虚拟存储技术是实现主存扩充的有效方法,已被广泛使用在现代的各种计算机系统中。多用户系统的虚拟存储器,利用大容量的外存空间来逻辑扩充内存,提供了一种将内存和外存统一管理的方法,内存中只存放那些经常的、反复被调用和访问的程序段和数据,而进程或作业的其它部分则存放于外存中,待需要时再调入内存。虚拟存储器可理解成一个由主存储器和辅助存储器构成的地址空间,最大虚拟地址空间往往取决于指令中的地址长度。其实现方法实质上是系统自动进行内存和外存间的数据交换。正如美国著名IT咨询公司RFG在对虚拟存储的定义中提到,所谓虚拟存储是指那些架构和产品被仿真设计成一个物理设备,如磁带机,其特性被镜像到另一个物理设备上,通常是一个磁盘或磁盘子系统。逻辑设备和虚拟设备的特性可以完全不同,应用系统操作的是虚拟设备,而不必关心真正的物理设备是什么。对于这个定义可从两个方面来理解:从专业角度看,虚拟存储实际上是逻辑存储,是一种灵活、有效地管理存储数据的方式,虚拟存储克服了物理存储的局限。从用户角度看,虚拟存储将使用户使用存储空间而不是使用物理存储硬件,用户有认为计算机主存"无限大"的虚假感觉,其实现技术对用户来说是透明的。而管理上是管理存储空间而不是管理物理存储硬件。

实现虚拟存储,系统需在内外存之间不断交换信息,因此,就要不断地启动外部设备以及处理缺页中断、淘汰页面、页面的调入调出等相应的过程。被选中淘汰的页面,最好是那些今后不会再被访问或最长时间里不会被访问的页,但操作系统是难以预料程序执行的轨迹的,很可能产生这样的现象:对于刚淘汰出去的页,进程可能马上又要访问它,故又需将它调入,因无空闲内存块又要淘汰另一页,而后者很可能是即将被访问的页。于是造成了系统需花费大量的时间忙于进行这种频繁的页面交换,使得机器的主要开销大多用在反复调入调出数据和程序段上,致使系统的实际效率很低,严重时将导致系统的瘫痪。这种现象称之为"抖动"[1,2]。

抖动现象与内存的多道程序度、系统分配给每个进程的物理实页数、系统采用的置换策略(淘汰算法)以及程序的行为特性有关。

2 抖动与工作集

2.1 多道程序度及物理实页数对抖动的影响

考察单CPU的多道程序系统,CPU的利用率如图1所示。从图中的曲线可知,CPU利用率Up是多道程序度N的函数。

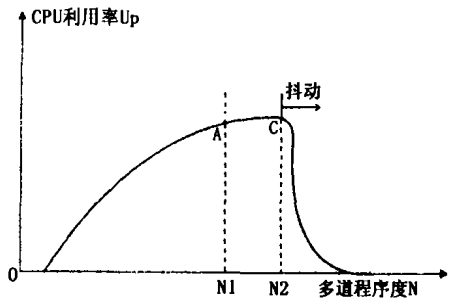

图1 CPU利用率曲线

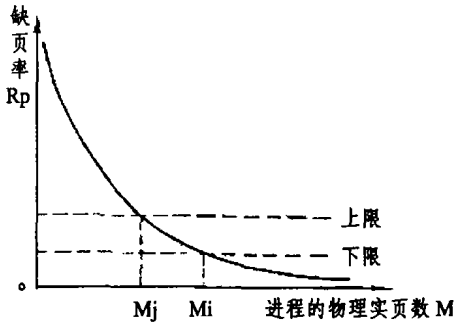

图2 缺页率与进程物理实页数的关系

当多道程序的道数N1≤N≤N2时,Up将达到峰值域A~C,如N继续增值,将引起系统抖动而使Up急剧下降,出现Up<<C的情况。从函数曲线可知,最理想的情况是使A≤Up(N)≤C时N的取值。

N的取值与系统给进程分配的内存空间的大小即物理实页数有关。在存储空间已定的情况下,N与进程可分配的物理实页数M以及M与进程的缺页率Rp之间均成反比关系,为进程分配的物理实页数越少,进程的缺页中断次数就越多,可能导致系统产生抖动现象。图2显示了进程的缺页率与进程占

用的物理实页数的关系。若每个进程的物理实页数M都能满足Mj≤M≤Mi，则内存中的多道程序度N就是合理的。此时该进程获得了有效运行所需的足够的内存空间，缺页率将保持在合理的范围内。

将系统中的多道程序度以及为各进程分配的物理实页数控制在一定的范围内，使得进程运行时所要调用的页面绝大部分都在内存，从而令CPU有效利用率接近最佳值，是防止系统发生抖动现象的有效方法，对这个问题的研究导致了工作集理论的提出。

所谓工作集[3]，就是进程在某段时间里实际上要访问的页的集合，程序要有效运行，其工作集必须在主存中。如果用S(t,Δt)表示从(t−Δt)开始到t之间所访问的页的集合，那么S就是作业在时间t上的工作集。工作集是对程序局部的一个近似模拟，如果我们能找出一个作业的各个工作集，并求出其页面数的最大者，就可确定该作业所需内存量，并由此确定系统内多道程序的最大数。在实践中可通过模拟程序执行的方法，每经过10ms或10000次内存访问输出一个工作集，以此找到所有工作集并求出其所需页面数的最大者，然后作为内存分配和防止抖动的依据。根据所有用户进程的工作集大小之和ΣS_i，系统可动态调整N和M的值。设B是内存用户区总物理实页数，MS_i是进程PR_i的初始工作集的大小，为保证各进程能够在工作集上有效运行，系统应维持$\Sigma MS_i<B$。为防止抖动的发生，系统可从以下几方面根据工作集的变化作相应的调整。

(1)若$\Sigma S_i>B$，则挂起某个进程。

(2)若$\Sigma S_i\leqslant B$，但$\Sigma MS_i>B$，则挂起某个进程。

(3)若$\Sigma S_i<\Sigma MS_i$且$MS_i<(B-\Sigma MS_i)$，则可激活进程PR_j(设PR_j为某个挂起的就绪进程)。

(4)若$\Sigma S_i\geqslant\Sigma MS_i$且$MS_i<(B-\Sigma S_i)$，则可激活进程$PR_j$。

2.2 对抖动的定量分析

利用统计模型进一步分析工作集与抖动之间的关系。设d为不发生缺页时满足一个内存请求所需的时间，g为发生缺页时系统从外存中读出一页数据所需的平均时间，P(s)为缺页率(s是当前进程在内存中的工作集)，则在虚存情况下平均的有效访问时间为(1−P(s))×d+P(s)×g。假定系统只运行一个进程且页交换时CPU空闲，若d为120ns，g为5ms，为了使有效访问时间达到1μs，则缺页率不能超过(1μs−120ns)/(5ms−120ns)=19.998%。

通过程序模拟可知，$P(s)=ae^{-bs}$(其中，0<a<1<b，$ae^{-bs}<<d$，a、b为与淘汰算法有关的待定常数)[4]。

以各并发进程具有相同的统计特性的假设为前提进行讨论，并且对于一个并发进程来说，只有发生缺页中断时才变成等待状态，那么在处理机访问一个内存单元的时间内，平均每秒引起的内外存之间的页传送率为P(s)/d，也就是每d/P(s)秒需要从外存向内存送一页，从而对一个在虚存范围内执行的进程，它可处于三种可能的状态之中，即：

(1)g<d/P(s)

(2)g>d/P(s)

(3)g=d/P(s)

第一种情况，缺页中断的次数较少，内外存之间的页面传送处于供过于求(等待页传送)的状态。第二种情况，缺页中断的次数较多，页面传送供不应求，系统已处于抖动状态。第三种情况是较理想的情况，即进程在执行过程中所需的页数正好等于从外存可以调入的页数。此时该进程在内存中占有最佳工作集。

将进程占有最佳工作集的条件g=d/P(s)与程序模拟结果$P(s)=ae^{-bs}$相结合，可得：

$ae^{-bs}=d/g$

$s=(1/b)\ln(ag/d)$

由式中可知，进程从外存中读取一页数据的时间越长，所需工作集越大。

抖动只有在g>d/P(s)时才会发生。而$P(s)=ae^{-bs}$是一个与工作集s、参数a和b都有关的概率值。P(s)是可以改变的，对于给定的系统来说，g和d已定，则解决抖动问题的策略是将P(s)减少到使g=d/P(s)。扩大工作集，即增加s，或选择不同的置换策略，即改变参数a和b，都可以满足要求。

3 抖动与置换策略

采用虚存技术能较好地解决空间不足的矛盾并提高系统的效率。但采用虚存技术后，如系统不能有效地将缺页率控制在一定范围内，则反而会使系统陷于缺页中断的频繁处理中，大部分的机器时间花费在来回进行页面置换上，只有一小部分时间用于程序的实际运行，从而直接影响整个系统的效率。从上面的讨论可知，改变参数a和b，亦即选择不同的置换策略，使进程的缺页率保持在合理的范围内，可防止抖动现象的发生[2,5]。置换策略的好坏直接关系到系统的性能，是虚拟存储技术的关键问题。

3.1 固定工作集的置换策略

若设m为工作集大小，s(t)为t时刻的工作集，r(t)为t时刻访问的页号(t是以访问序列的每一项为单位的时间。如访问序列为h1,h2,h3，即1时刻访问第h1页，2时刻访问第h2页，3时刻访问第h3页。记为：r(1)=h1，r(2)=h2，r(3)=h3)。则固定工作集的置换策略有如下控制过程：

$s(0)=\phi$

$s(t+1)=$

$$\begin{cases} s(t) & r(t+1)\in s(t) \\ s(t)+\{r(t+1)\} & r(t+1)\notin s(t),|s(t)|<m \\ s(t)+\{r(t+1)\}-\{y\} & r(t+1)\notin s(t),|s(t)|=m,y\in s(t) \end{cases}$$

式中y为被淘汰页。根据不同的置换策略可有不同的y的选择方法。

采用固定工作集的置换策略有很多，如FIFO(先进先出置换算法)、OPT(最佳置换算法)、LRU(最近最少使用算法)等。其中最具代表性的是LRU算法。

3.1.1 LRU(Least Recently Used)算法 LRU最近最少使用置换算法，其基本原理是选择最近一段时间内最久没有访问过的页淘汰。故若S(p,m,t)表示页面走向为p、主存容量为m、在时刻t的页面集合，对于LRU算法，存在如下关系，即$S(p,m,t)\subseteq S(p,m+1,t)$成立，因此工作集大小为m时出现的缺页中断一定会在工作集大小为m+1时出现，即对于任何时刻t(t=1,2,3…)，S(p,m,t)所选中的页号必定包含在S(p,m+1,t)之中。按LRU算法的思想，每访问一个页面时，都要留下一个“邮戳”，淘汰时选择一个最后一次访问离现在最久的页。由于要对各页访问的历史情况加以记录和更新，故描述LRU算法的数据结构中需引入记录时间的给定集，以标识页面的使用情况。图3显示了LRU算法的执行过程，假定系统分给进程的物理实页数为3，页面访问序列为4,3,2,1,2,5,1,4,5,1,3,5。在LRU控制下共出现7次缺页中

断。

访问序列	4	3	2	1	2	5	1	4	5	1	3	5
工作集	4	3	2	1	2	5	1	4	5	1	3	5
		4	3	2	1	2	5	1	4	5	5	3
			4	3	3	1	2	5	1	4	4	4
缺页中断	×	×	×	×	√	×	√	×	√	√	×	√

图3 LRU 策略控制下工作集的变化

在各种置换策略中,LRU 算法是一种比较理想的策略,但该策略实现过程需要实际硬件的支持,造成系统的"非生产性"开销太大,增加了硬件成本。目前很多操作系统采用 LRU 的近似方法。

3.1.2 LRU(Least Recently Used)算法的改进 LRU 算法的时间主要耗在大量的比较上,为减少算法时间复杂性,可对 LRU 方法加以改进,如采用矩阵表示法。其做法是对 n 个实页设置一个 n×n 的矩阵,矩阵初始时全为0。当 k 页被访问时,硬件置 k 行的所有位为1,再置 k 列的所有位为0。任何时刻二进制值最小的一行所对应的页是最近最少使用的页。图4显示了访问序列为4,3,2,1,2,5,1,4,5,1,3,5,物理实页数为3的情况下,淘汰的页面及工作集的变化。

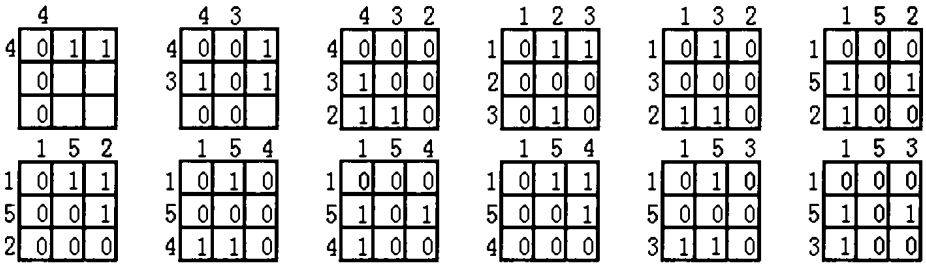

图4 LRU 改进策略控制下工作集的变化

3.2 可变工作集的置换策略

工作集是时间 t 的函数,随着时间的变化,工作集以及工作集的大小也发生变化,亦即局部集有大有小,时大时小,因此让工作集大小恒定就不能合理使用空间。更合理的策略应该是随着局部集的变化来动态调整工作集的大小。

3.2.1 WS(Working Set)策略 这种策略是基于工作集中只放置当前活跃的局部集中的页。若某页在工作集中有 Δ 个时间单位未被引用,则将其淘汰。图5显示了 Δ=5,访问序列为4,3,2,1,2,5,1,4,5,1,3,5,物理实页数为3的情况下,淘汰页面及工作集的变化过程。共产生8次缺页中断。工作集的平均大小为2.7个物理实页。

访问序列	4	3	2	1	2	5	1	4	5	1	3	5
工作集	4	4	4	1	1	1	1	1	5	5	5	5
		3	3	3	3	5	5	5	4	4	3	3
			2	2	2	2	2	4		1	1	1
缺页中断	×	×	×	×	√	×	√	×	√	×	×	√
工作集大小	1	2	3	3	3	3	3	3	2	3	3	3

图5 WS 策略控制下工作集的变化

对 WS 策略的一种改进是 SWS(Sampled Working Set)策略。它的基本做法是:每过 τ 个时间单位(如取 τ=100000)检查一次计数器,将计数器值大小等于 Δ 的页淘汰出去。这可用硬件时钟来计数,每访问一页,同时将当前时钟值记录在页表项中,经 τ 时间单位后便检查工作集,将当前时钟值与记录在页表项中的值相减,淘汰差值大小等于 Δ 的页。

访问序列	4	3	2	1	2	5	1	4	5	1	3	5
工作集	4	3	2	2	2	5	5	5	5	5	5	
				1	1	1	1	1	1	1	3	
								4				
缺页中断	×	×	×	×	√	×	√	×	√	√	×	√
工作集大小	1	1	1	2	2	2	2	3	2	2	2	1

图6 VMIN 策略控制下工作集的变化

3.2.2 VMIN(Variable MINimal replacement)策略 VMIN 策略需要一个控制参数 Δ,若某页下次访问的距离大于 Δ 则将其淘汰。图6显示了 Δ=5,访问序列为4,3,2,1,2,5,1,4,5,1,3,5,物理实页数为3的情况下,工作集变化过程。共产生7次缺页中断,工作集的平均大小为1.8个物理实页。

3.2.3 二种方法的比较 上述二种方法的有关参数如图7所示,从表中的参数可知,VMIN 策略控制下的工作集长度的均值小于在 WS 策略控制下的均值。

置换策略	时间控制参数	缺页中断次数	工作集长度
WS	Δ=5	8	2.7
VMIN	Δ=5	7	1.8

图7 WS 策略与 VMIN 策略的比较

对 VMIN 策略进行讨论。设处理一次缺页中断的时间为常数 P,淘汰一页的时间为常数 R,$C_k(i,i+1)$为第 k 页第 i 次被访问到第 i+1次被访问的开销。对第 k 页,构成 $C_k(i,i+1)$ 开销的情况有以下三种:

(1)第 K 页在第 i 次访问后即被淘汰,直到第 i+1次访问时,系统产生缺页中断再将其调入工作集,此时系统的开销为淘汰一页的时间与处理一次缺页中断的时间之和 R+P。

(2)第 K 页在第 i 次访问后仍然驻留在工作集中 X 个时间单位后才被淘汰,直到第 i+1次访问时产生缺页中断再调入工作集,则其开销为 R+P+X。

(3)第 k 页在第 i 次访问后没有被淘汰,一直驻留在工作集中,直到第 i+1次被访问。设这段时间距离为 L(i,i+1),则其开销为 L(i,i+1)。

综上所述,可有:

$$C_k(i,i+1)=\begin{cases}R+P\\R+P+X\\L(i,i+1)\end{cases}$$

$$\mathrm{Min}[C_k(i,i+1)]=\mathrm{Min}[R+P,R+P+X,L(i,i+1)]=\mathrm{Min}[R+P,L(i,i+1)]=\begin{cases}R+P\\L(i,i+1)\end{cases}$$

由于 VMIN 的控制参数为 Δ,若某页下次访问的距离大于 Δ,则此页在下次被调用前已被系统淘汰。所以

$$\mathrm{Min}[C_k(i,i+1)]=\begin{cases}R+P & L(i,i+1)>\Delta\\ L(i,i+1) & L(i,i+1)\leqslant\Delta\end{cases}$$

则当 $\Delta=R+P$ 时，VMIN 策略的效果可达最优。

VMIN 算法的前提是需预先知道进程的整个访问序列，这在实际操作中很难实现，此方法可以作为评价其他置换策略的标准。在置换策略的实施中，除用控制参数 Δ 的方法外，系统还可以通过缺页频率来调整工作集，缺页率高时则扩大工作集；缺页率低时则缩小工作集。

4 抖动与程序的行为特性

很多的存储管理方法考虑的出发点是基于程序的特性——局部性概念。程序局部化程度越高，程序执行时可经常集中在几个页面上访问，可减少缺页中断的次数，从而防止抖动现象的发生。程序的局部性包括时间局部性和空间局部性。

(1)时间局部性：是指一旦某个位置(数据或指令)被访问了，它常常很快又要被再次访问。这种现象体现在程序的循环结构以及变量和子程序等程序结构中。

(2)空间局部性：是指一旦某个位置被访问了，那么它附近的位置很快也要被访问到。这种现象体现在顺序的指令序列、线性的数据结构等程序结构中。

程序的行为特性与系统的缺页率，以至整个系统的效率有一定的关系。我们在进行程序设计的时候，要力求减少程序访问的离散性，提高程序访问的局部性。下面的例子很好地说明了程序的行为特性对缺页率的影响。

假定在页面大小为512字的分页系统中对矩阵 $A_{512\times512}$ 赋值，最基本的编码方式有以下两种：

程序编制方法1：

```
var A array [1..512] of array [1..512] of integer;
  For j:=1 to 512
    For i:=1 to 512
      A[i,j]:=0;
```

程序编制方法2：

```
var A array [1..512] of array [1..512] of integer;
  For i:=1 to 512
    For j:=1 to 512
      A[i,j]:=0;
```

方法1对矩阵赋值的执行顺序为：A[1,1]，A[2,1]，A[3,1]……A[512,1]，A[1,2]，A[2,2]，A[3,2]，……。方法2对矩阵赋值的执行顺序为：A[1,1]，A[1,2]，A[1,3]……A[1,512]，A[2,1]，A[2,2]，A[2,3]，……。矩阵在内存中是按行存放的，对512字的页面大小来说，每行刚好占一页。若系统分给这个进程只有二个物理实页，对方法1，因其使用矩阵的顺序是按列进行的，将引起512×(512/2)=131072次缺页中断。对方法2，因其使用矩阵的顺序是按行进行的，引起的缺页中断次数为512/2=256次。

结束语 抖动现象存在于虚拟存储管理的系统中，减少或消除抖动现象，对于充分发挥系统的性能，提高系统的效率具有非常重要的意义。引入工作集，采用适当的页面置换策略等方法，可防止系统抖动现象的发生，在满足各进程工作集的前提下，使内存的多道程序度接近最佳值。

参 考 文 献

1 冯耀霖，杜舜国. 操作系统. 西安电子科技大学出版社，2000. 92～100

2 Madnick S E, Donovan J J. OPERATING SYSTEM. Praha: Nakl. techn. Lit. ,1983. 564～575

3 汤子瀛，哲凤屏，汤小丹. 计算机操作系统. 西安电子科技大学出版社，2001. 165～186

4 张尧学，史美林. 计算机操作系统教程. 清华大学出版社，1993

5 邹鹏，王广芳. 操作系统原理. 国防科技大学出版社，1996. 110～120

(上接第156页)

类型声明描述中的继承关系。假定类型 E 是由类型 B 采用 public 方式派生的类型。如果程序运行中抛出了一个 E 类型的异常 e，这个异常可能传到一个在 E 类型定义的作用域之外的函数里，如果那里确实定义了要求捕捉 B 类型异常的处理器，那么这一处理器能否处理异常 e?这一问题无论怎样解释都有不合理的地方。按照实际的类型继承关系，e 应该在这里被捕捉和处理。但是这一实际类型关系在处理器的位置上又是不可见的。

此外，如果异常对象被传到其类型定义的作用域之外，还可能导致在需要复制或销毁该对象时，无法执行有关复制构造函数或析构函数。一种典型情况是异常在其类定义的作用域之外被通用处理器捕获，而这个类又定义了特殊析构函数，这时程序将无法调用该函数。注意，由于异常传播完全是动态确定的过程，编译处理时不可能对这种问题进行完全的检查。

这些情况告诉我们，表示异常的基类和派生类最好一起定义。这些类不宜具有复杂语义，例如定义具有特殊语义的复制语义和特定析构函数等等。

2.7 异常处理的实现和代价

可以将异常看成一种非局部控制转移机制。然而，一般来说，异常处理的代价远高于函数调用和退出。目前异常处理有两种基本实现方式：动态链方式和静态表方式。它们各有长短。动态链方式在出现异常时的效率略高，且能很好支持动态连接等。但采用这种方式，程序运行中即使没有抛出异常，也要付出额外开销。静态表方式采用静态的监视范围表，如果程序未实际抛出异常，就无需为异常处理机制的存在付出额外代价。但是，发生异常时就需要多次查询这个表，因此效率较低。这种方式的另一缺点是难以支持动态连接。C++异常机制的设计使之可以支持静态表方式，主要是为了它“不用就无需付出代价”的设计原则。

由于 C++异常的高代价特征，因此，人们建议只将异常处理机制用于真正需要处理异常的情况，而不要将它用作一种非局部转移的方便手段。当然，对于那些性能要求并不高，而可靠性要求很高的程序或者程序部分，这一问题的重要性就大大降低了。

总结 本文中对 C++语言的异常处理机制进行了深入细致的分析，包括其中的一些静态语法特征、动态性质、实现中的问题以及它们所可能造成的影响。通过这些分析，我们可以看到在程序语言，特别是面向对象语言中一般的异常处理机制所牵涉到的许多问题。这一分析也能帮助实际程序语言的使用者理解应该如何使用这方面的语言机制。

参 考 文 献

1 Koenig A, Stroustrup B. Exception Handling in C++. Jounal of Object Oriented Programming, 1990, 3(2): 16～33

2 Stroustroup B. The Design and Evolution of C++, Addison-Wesley, 1994

3 Buhr P A, Mok W Y R. Advanced exception handling mechanisms. IEEE Trans, on SE, 2000, 26(9): 820～836

4 Knudsen J L. Fault tolerance and exception handling in BETA, LNCS 2022, 1～17, Springer-Verlag, 2002

5 Dony C. A full object-orinted exception handling system: rationale and Smalltalk Implementation. Springer-Verlag, 2002. 18～38