Windows中的进程的Working Set，Private Bytes和Virtual Bytes

转载 fw0124 于 2011-04-27 17:01:00 发布 阅读量3.1w 收藏 12 点赞数 6
分类专栏： Windows 文章标签： windows dll application manager system thread

总结：

1）Working Set看成一个进程可以用到(但不一定会使用)的物理内存。即不引起page fault异常就能够访问的内存。

Working Set包含了可能被其他程序共享的内存, 例如DLL就是一个典型的可能被其他程序共享的资源。

所以所有进程的Working Set加起来有可能大于实际的物理内存。

2） Private Bytes是只被本进程用占用的虚拟地址空间，不包括其他进程共享的内存。

Private Bytes既包括不引起page fault异常就能够访问的内存也包括引起page fault异常才能够访问的内存。

所以一般Private Bytes大于Working Set。但是如果一个进程和其他进程共享较多内存，也可能造成Working Set大于Private Bytes。

3） Virtual Byte是整个进程占用的全部虚拟地址空间。32位Windows 用户模式下，进程最大可以使用2GB，可以通过修改Boot.ini文件扩展为最大可以使用到3GB。

4）Windows Task Manager中看到内存使用量是Working Set。

==========

原文：

http://my.opera.com/Returner/blog/show.dml/573233

http://www.cnblogs.com/awpatp/archive/2010/01/26/1656651.html

*The working set of a process is the set of memory pages currently visible to the process in physical RAM memory. These pages are resident and available for an application to use without triggering a page fault.
*The working set of a process consists of the set of resident physical pages visible to the process. When a thread accesses a page that is not in the working set of its process, a page fault occurs. Before the thread can continue, the virtual memory manager must add the page to the working set of the process. A larger working set increases the probability that a page will be resident in memory, and decreases the rate of page faults.
可以把Working Set看成一个进程可以用到(但不一定会使用)的物理内存（物理内存页的集合）。内存管理单元在进行虚拟内存地址到物理内存地址转换时，如果虚拟地址不在物理内

存中，会引起page fault异常。足够的WorkingSet就可以保证常用的虚拟地址都位于物理内存中，减少这种异常，避免了异常处理（例如访问swap文件，将页面读入物理内存）带来的性能损耗。因此，对于时间比较敏感的程序，应该分配足够的WorkingSet以保证程序性能。

*When you increase the working set size of an application, you are taking away physical memory from the rest of the system.
*Suppose you have a 16-megabyte system and you set your minimum to four megabytes. In effect, this takes away four megabytes from the system. Other applications may be unable to get their minimum working set.
可见，Working Set是被真实地从物理内存中划分出来的。一个程序占用了多少Working Set，物理内存中就有多少空间不能被其它程序使用。

*Reducing memory consumption is always a beneficial goal. If you call SetProcessWorkingSetSize(0xffffffff, 0xffffffff), this tells the system that your working set can be released. This does not change the current sizing of the working set, it just allows the memory to be used by other applications. It is a good idea to do this when your application goes into a wait state.
*Windows NT 3.5 allows processes to increase their working set size by using SetProcessWorkingSetSize(). This API is also useful to trim your minimum working set size if you want to run many processes at once, because each process has the default minimum working set size reserved, no matter how small the process actually is.
Windows会为每个进程保留一个默认数值的working set，然而这个保留的Working set可能会比该process实际需要的大。可以用SetProcessWorkingSetSize()来对进程的Working set进行裁剪，只保留进程目前已经占用的页面，空闲的就释放掉给其它应用使用。

一个有趣的问题是, working set指目前程序所消耗的物理内存, private bytes指的是commit的内存, 那么为什么有些进程的working set比private bytes还大? 要回答这个问题, 需要仔细看看两者的定义:

- "Working Set refers to the numbers of pages of virtual memory committed to a given process, both shared and private."
- "Private Bytes is the current size, in bytes, of memory that this process has allocated that cannot be shared with other processes."

所以, Working Set包含了可能被其他程序共享的内存, 而Private Bytes只包括被当前进程使用的内存.

DLL是一个典型的可能被其他程序共享的资源. DLL的加载使用文件映像, 因此包含DLL的物理内存可以被同时映像到多个进程上. 所以在进程中加载DLL的内存只能算到working set上, 而不能被算到private bytes上.

在解决内存问题的时候Shared的部分一般可以不用考虑.

一个进程使用内存的时候, 它占用的内存会被分为两部分, 一部分是working set, 另一部分是private byte减去working set. 其中working set是贮存在物理内存中的, 而剩下的另一部分是paging file, 存在磁盘上.

一般来说把所有进程的working set加起来会比你机器上所拥有的物理内存要大, 这是因为有Shared的资源(比如DLL)的缘故.

Virtual Bytes.

The current size, in bytes, of the virtual address space for this process. The virtual address space limit of a user mode process is 2 GB, unless 3 GB address space is enabled by using the /3GB switch in boot.ini.

原文:

Virtual Bytes are the total **virtual address space** occupied by the entire process, including memory-mapped files such as shared DLLs. This is like the working set except it includes data that has already been paged out and is sitting in a pagefile somewhere. The total virtual bytes used by every process on a system under heavy load will add up to significantly more memory than the machine actually has.

翻译:

Virtual Byte是整个进程占用的全部虚拟地址空间, 包括诸如共享dll在内的内存映射文件. 它跟working set很像, 不一样的是, 它还得算上已经被page out的存放在别的什么地方的pagefile中的数据. 一个高负荷系统中所有进程的全部的virtual bytes加起来, 会比机器实际拥有的内存多很多.

个人理解

==========

假设我有一台机器, 机器有物理内存1024M. 机器上面仅运行着三个进程A, B, C.

下面的情况是可能的:

A和B仅共用一个叫做common.dll的库文件. common.dll为A分配了10M的内存, 为B分配了20M内存.

common.dll本身占30M内存.

A的可执行程序以及其数据本身占用了40M, B的使用了50M, C的使用了60兆.

A的可执行程序本身使用的内存中有5兆不在内存中, 在pagefile中.

A没有引用其他的dll文件.

那么A的private bytes是40-5+10 = 45M

A的working set是40-5+30 = 65M

A的virtual bytes是40+30+10=80M

工作集

项目 • 2023/06/13

进程的工作集是进程的虚拟地址空间中当前驻留在物理内存中的一组页面。 工作集仅包含可分页内存分配;工作集中不包括不可分页的内存分配，例如 地址窗口扩展 (AWE) 或 大型页面分配 。

当进程引用当前不在工作集中的可分页内存时，会发生 *页面错误* 。 系统页错误处理程序尝试解决页面错误，如果成功，该页将添加到工作集。 (访问 AWE 或大型页面分配永远不会导致页面错误，因为这些分配不可分页。)

必须通过从页面*的后备存储*（系统分页文件或进程创建的内存映射文件）读取页面内容来解决*硬页面错误*。 无需访问后备存储即可解决*软页面错误* 。 在出现软页面错误时：：

- 该页位于其他某个进程的工作集中，因此它已驻留在内存中。
- 该页处于转换状态，因为它已从正在使用该页的所有进程的工作集中删除并且尚未重新调整用途，或者由于内存管理器预提取操作而已驻留。
- 进程首次引用分配的虚拟页面， (有时称为 *"零需求"故障*) 。

由于执行以下操作，可以从进程工作集中删除页面：

- 该进程通过调用 SetProcessWorkingSetSize、SetProcessWorkingSetSizeEx 或 EmptyWorkingSet 函数减少或清空工作集。
- 进程在未锁定的内存范围上调用 VirtualUnlock 函数。
- 进程使用 UnmapViewOfFile 函数取消映射文件的映射视图。
- 内存管理器从工作集中剪裁页，以创建更多可用内存。
- 例如，内存管理器必须从工作集中删除页，以便为新页 (腾出空间，因为工作集处于其最大大小) 。

如果多个进程共享一个页面，则从一个进程的工作集中删除该页不会影响其他进程。 从使用该页的所有进程的工作集中删除页面后，该页将成为 *过渡页*。 转换页将一直缓存在 RAM 中，直到某个进程再次引用该页或重新调整 (用途，例如，用零填充并提供给另一个进程) 。 如果转换页自上次写入磁盘 (即，如果页面是"脏") ，则必须将页面写入其后备存储，然后才能重新调整用途。 一旦这些页面可用，系统可能会开始将脏转换页面写入其后备存储。

每个进程都有影响进程的虚拟内存分页行为的最小和最大工作集大小。 若要获取指定进程的工作集的当前大小，请使用 GetProcessMemoryInfo 函数。 若要获取或更改最小和最大工作集大小，请使用 GetProcessWorkingSetSizeEx 和 SetProcessWorkingSetSizeEx 函数。

PSAPI) (进程状态应用程序编程接口提供了许多函数，这些函数返回有关进程的工作集的详细信息。 有关详细信息，请参阅 工作集信息。

反馈

此页面是否有帮助? 是 否

提供产品反馈 | 在 Microsoft Q&A 获取帮助

Use Performance Monitor to find a user-mode memory leak

Article • 08/27/2024

If you suspect there's a user-mode memory leak but aren't sure which process causes it, use Performance Monitor to measure the memory usage of individual processes.

Run Performance Monitor as Administrator. Right click on the *Performance Monitor* under *Monitoring Tools* and select **Properties** to add the following counters:

- **Process** > **Private Bytes** (for each process you want to examine)
- **Process** > **Virtual Bytes** (for each process you wish to examine)

Set the *Duration* to capture enough activity. For example, change the update time to 600 seconds to capture a graph of the leak over time. You might also want to log the data to a file for later examination.

The **Private Bytes** counter indicates the total amount of memory that a process has allocated, not including memory shared with other processes.

The **Virtual Bytes** counter indicates the current size of the virtual address space that the process uses.

Some memory leaks appear in the data file in the form of an increase in private bytes allocated. Other memory leaks show up in the form of an increase in the virtual address space.

After you've determined which process is leaking memory, use the UMDH tool to determine the specific routine that's at fault. For details, see Using UMDH to find user-mode memory leaks.