CLOUD COMPUTING

The Memory-Optimization Hoax

RAM optimizers make false promises

Mark Russinovich | Dec 29, 2003

As you've surfed the Web, you've probably seen browser pop-ups such as "Defragment your memory and improve performance" and "Minimize application and system failures and free unused memory." The links lead you to utilities that promise to do all that and more for a mere $9.95, $14.95, or $29.95. Sound too good to be true? It is. These utilities appear to do useful work, but at best, RAM optimizers have no effect, and at worst, they seriously degrade performance.

Literally dozens of so-called "memory optimizers" are available—some are commercial products and others are freeware. You might even be running such a product on your system. What do these products really do, and how do they try and fool you into thinking that they live up to their claims? Let's take a look inside memory optimizers to see exactly how they manipulate visible memory counters in Windows.

The Memory Optimizer UI

also display the processes running on the system.

When a scheduled optimization job runs, the utility's available-memory counter often goes up, sometimes dramatically, which appears to imply that the tool is actually freeing up memory for your applications to use. To understand how these utilities cause the available-memory line to rise, you first need to understand how Windows manages physical memory.

Windows Memory Management

Like most modern OSs, Windows implements a demand-paged virtual-memory system. An OS uses virtual memory to give applications the illusion that a computer has more physical memory than it actually does.

On 32-bit Windows systems, processes have a virtual-memory address space of 4GB that the OS typically divides equally between the process and the system. Thus, a process can allocate as much as 2GB of virtual memory, depending on the amount available. The total amount of virtual memory allocated to all processes can't exceed the sum of the system's paging files and most of its physical memory (the OS reserves a small portion of physical memory).

Given that processes can, with a large enough paging file, allocate virtual memory that exceeds the computer's physical memory capacity, the Windows Memory Manager subsystem must share physical memory among processes and the Cache Manager's cached file data. As Figure 1 illustrates, the Memory Manager assigns each process (e.g., Microsoft Word, Notepad, Windows Explorer) a part of physical memory, which is known as the process's working set. The portions of the kernel and drivers that are pageable, in addition to pageable kernel memory buffers

The computer's memory-management hardware requires that Windows manage working sets and virtual memory in page-size blocks. (On 32-bit x86 processors, pages are typically 4096 bytes in size. However, the OS and memory-intensive applications also use large pages of 4MB as an optimization, when possible.)

When a process accesses a page of its virtual memory that isn't present in its working set, the process incurs a *page fault* hardware exception. When that happens, the Memory Manager assigns a page of available physical memory to hold the newly accessed data. Additionally, the Memory Manager might decide to expand the process's working set by adding the page to the working set. However, if the Memory Manager deems the process's working set to be large enough, it will exchange a page already in the working set with the new page, choosing for replacement the page that the process accessed least recently, under the assumption that the process is least likely to access that page in the near future.

When the Memory Manager removes a page from a process working set, it must decide what to do with that page. If the page has been modified, the Memory Manager first puts it on the *modified page list,* a list of pages that eventually will be written to the paging file or to the memory-mapped files to which the pages correspond. From the modified page list, the Memory Manager moves pages to a pool called the *standby list*. Unmodified pages go directly to the standby list. Thus, you can view the standby list as a cache of file data.

Available Memory

I stated earlier that the Memory Manager gives a page of available physical memory to a process that experiences a page fault, but I haven't told you what defines available memory. The standby list is part of physical memory that the

Figure 2 shows the transitions that occur between working sets and the page lists. Once per second, a system thread wakes up and calls the Memory Manager's working set manager to examine the System and processes' working sets. If available memory is low, the working set manager removes pages from the processes that haven't incurred many page faults in the past second. The removed pages go to the modified or standby list as appropriate and contribute to available memory. An important side effect of this tuning mechanism is that if the system needs memory for other processes, the Memory Manager takes pages from idle processes' working sets. Thus, those working sets eventually disappear, meaning that processes that remain idle for a sufficient length of time eventually consume no physical memory.

When a process needs a new page of physical memory, the Memory Manager first determines whether the page the process is accessing is in the standby or modified page list. The page will be in one of these lists if the page was removed from the process working set and not reused for another purpose. Placing that page back into the process working set is called a *soft page fault* because, unlike hard page faults, it doesn't involve a read from the paging file or other file on disk.

If the page isn't in the standby list or the modified page list, the Memory Manager takes a page from a list that has a page on it, checking first the free list, then the zeroed page list, and finally the standby list. If no memory is available, the Memory Manager triggers the Balance Set Manager to trim the process working sets and populate one of the three lists that make up available memory. If the Memory Manager has to remove a page from the zeroed page, free, or standby list for reuse, it determines how to access the targeted code or data, which can include reading

attention to the workings of RAM optimizers. The available-memory value that RAM optimizers display is the same value that the Task Manager shows as Available in the Physical Memory section on the Performance tab, which Figure 3 shows. That value is the sum of the sizes of the standby, zeroed page, and free lists. System Cache is the sum of the sizes of the standby list and the System working set. (In Windows NT 4.0 and earlier, File Cache reflects the size of only the System working set.)

RAM optimizers take advantage of the Memory Manager's behavior by allocating, then freeing, large amounts of virtual memory. Figure 4, page 21, shows the effect a RAM optimizer has on a system. The first bar depicts the working sets and available memory before optimization. The second bar shows that the RAM optimizer creates a high memory demand, which it does by incurring many page faults in a short time. In response, the Memory Manager increases the optimizer's working set. This working-set expansion occurs at the expense of available memory and—when available memory becomes low—at the expense of other processes' working sets. The third bar illustrates how, after the RAM optimizer frees its memory, the Memory Manager moves all the pages that were assigned to the RAM optimizer to the free list, thus contributing to the available-memory value. Most optimizers hide the rapid decline in available memory that occurs during the first step, but if you run Task Manager during an optimization, you can often see the decline as it takes place.

Although gaining more available memory might seem beneficial, it isn't. As RAM optimizers force the available-memory counter up, they force other processes' data and code out of memory. Say that you're running Word, for example. As the optimizer forces the available-memory counter up, the text of open documents and the program code that was part of Word's working set before the optimization (and

Other RAM Optimizer Claims

Some vendors make additional claims for their RAM-optimizer products. One claim you might see is that a product frees memory that's needlessly consumed by unused processes, such as those that run in the taskbar tray. All such claims are untrue because Windows automatically trims idle processes' working sets. The Memory Manager handles all necessary memory optimization.

Developers of RAM optimizers also claim that their products defragment memory. The act of allocating, then freeing a large amount of virtual memory might, as a conceivable side effect, lead to blocks of contiguous available memory. However, because virtual memory masks the layout of physical memory from processes, processes can't directly benefit from having virtual memory backed by contiguous physical memory. As processes execute and undergo working-set trimming and growth, their virtual-memory–to–physical-memory mappings will become fragmented despite the availability of contiguous memory.

Having contiguous available memory can improve performance in one case: when the Memory Manager, to maximize the behavior of the CPU memory caches, uses a mechanism called *page coloring* to decide which page from the free or zeroed page list to assign to a process. However, any minor benefit that might result from making available physical memory contiguous is heavily outweighed by the negative impact of discarding valuable code and data from memory.

Finally, vendors often claim that RAM optimizers regain memory lost to leaks. This claim is perhaps the most patently false assertion of all.

The Memory Manager knows at all times what physical and virtual memory belongs to a process. However, if a process allocates memory, then doesn't free it

working-set trimming eventually will steal from the process's working set any physical pages that are assigned to leaked virtual memory. That process sends the leaked pages to the paging file and lets the system use physical memory for other purposes. Thus, a memory leak has only a limited impact on available physical memory. The real impact is on virtual-memory consumption (which Task Manager calls both PF Usage and Commit Charge). No utility can do anything about virtual-memory consumption other than kill processes that are consuming memory.

Fraudware

I have yet to see a RAM optimizer that lives up to any of its claims. If you look closely, you'll often see that vendors have buried long-winded disclaimers on their Web sites that state what I've explained—that the product might not have any impact on a system's performance and might actually degrade it. Even without knowing how these products take advantage of the Memory Manager to inflate a highly visible and provocatively named memory metric, common sense suggests that if RAM optimization were possible (and could be implemented by so many small-time upstarts), Microsoft developers would have long since integrated the technology into the kernel.

2175 COMMENTS

RECOMMENDED READING

Cloud-Based Virtual Desktops Can Replace Physical Devices
DEC 07, 2020

Deutsche Bank to Move 'Heart' of IT Systems Into Google's Cloud
DEC 04, 2020

What Must Enterprises Learn to Increase ROI from the Cloud?

CIOs to Hit the Gas on Digital Business in 2021