MarkRussinovich Microsoft

06-26-2019 11

Pushing the Limits of Windows: Physical Memory

First published on TechNet on Jul 21, 2008

This is the first blog post in a series I'll write over the coming months called Pushing the Limits of Windows that describes how Windows and applications use a particular resource, the licensing and implementation-derived limits of the resource, how to measure the resource's usage, and how to diagnose leaks. To be able to manage your Windows systems effectively you need to understand how Windows manages physical resources, such as CPUs and memory, as well as logical resources, such as virtual memory, handles, and window manager objects. Knowing the limits of those resources and how to track their usage enables you to attribute resource usage to the applications that consume them, effectively size a system for a particular workload, and identify applications that leak resources.

Here's the index of the entire Pushing the Limits series. While they can stand on their own, they assume that you read them in order.

Physical Memory

One of the most fundamental resources on a computer is physical memory. Windows' memory manager is responsible with populating memory with the code and data of active processes, device drivers, and the operating system itself. Because most systems access more code and data than can fit in physical memory as they run, physical memory is in essence a window into the code and data used over time. The amount of memory can therefore affect performance, because when data or code a process or the operating system needs is not present, the memory manager must bring it in from disk.

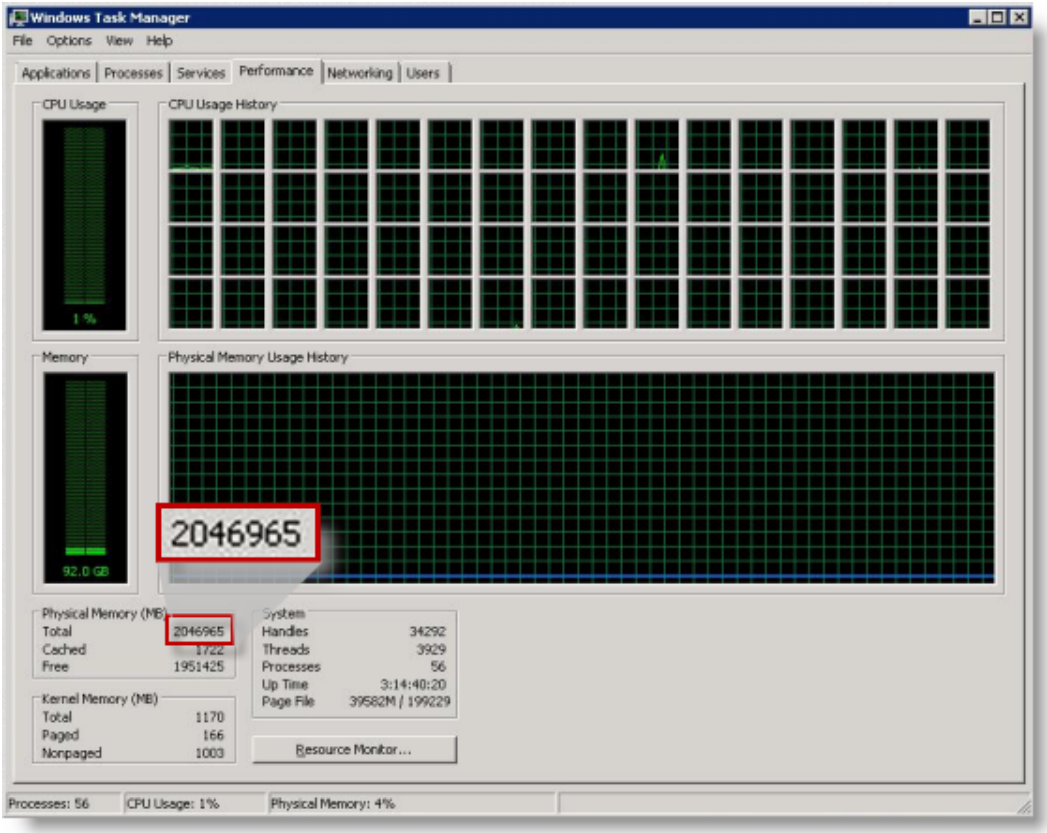
Besides affecting performance, the amount of physical memory impacts other resource limits. For example, the amount of non-paged pool, operating system buffers backed by physical memory, is obviously constrained by physical memory. Physical memory also contributes to the system virtual

memory limit, which is the sum of roughly the size of physical memory plus the maximum configured size of any paging files. Physical memory also can indirectly limit the maximum number of processes, which I'll talk about in a future post on process and thread limits.

Windows Server Memory Limits

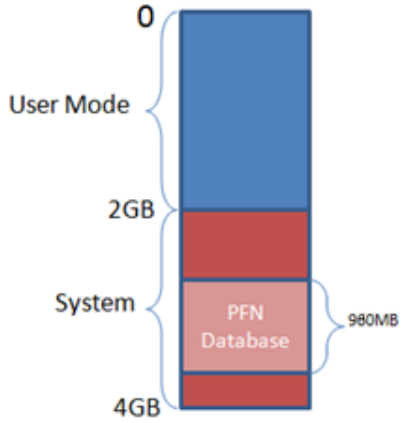
Windows support for physical memory is dictated by hardware limitations, licensing, operating system data structures, and driver compatibility. The Memory Limits for Windows Releases page in MSDN documents the limits for different Windows versions, and within a version, by SKU.

You can see physical memory support licensing differentiation across the server SKUs for all versions of Windows. For example, the 32-bit version of Windows Server 2008 Standard supports only 4GB, while the 32-bit Windows Server 2008 Datacenter supports 64GB. Likewise, the 64-bit Windows Server 2008 Standard supports 32GB and the 64-bit Windows Server 2008 Datacenter can handle a whopping 2TB. There aren't many 2TB systems out there, but the Windows Server Performance Team knows of a couple, including one they had in their lab at one point. Here's a screenshot of Task Manager running on that system:



The maximum 32-bit limit of 128GB, supported by Windows Server 2003 Datacenter Edition, comes from the fact that structures the Memory Manager uses to track physical memory would consume too much of the system's virtual address space on larger systems. The Memory Manager keeps track of each page of memory in an array called the PFN database and, for performance, it maps the entire PFN database into virtual memory. Because it represents each page of memory with a 28-byte data structure, the PFN database on a 128GB system requires about 980MB. 32-bit Windows has a 4GB virtual address space defined by hardware that it splits by default between the currently executing user-mode process (e.g.

Notepad) and the system. 980MB therefore consumes almost half the available 2GB of system virtual address space, leaving only 1GB for mapping the kernel, device drivers, system cache and other system data structures, making that a reasonable cut off:



That's also why the memory limits table lists lower limits for the same SKU's when booted with 4GB tuning (called 4GT and enabled with the Boot.ini's /3GB or /USERVA, and Bcdedit's /Set IncreaseUserVa boot options), because 4GT moves the split to give 3GB to user mode and leave only 1GB for the system. For improved performance, Windows Server 2008 reserves more for system address space by lowering its maximum 32-bit physical memory support to 64GB.

The Memory Manager could accommodate more memory by mapping pieces of the PFN database into the system address as needed, but that would add complexity and possibly reduce performance with the added overhead of map and unmap operations. It's only recently that systems have become large enough for that to be considered, but because the system address space is not a constraint for mapping the entire PFN database on 64-bit Windows, support for more memory is left to 64-bit Windows.

The maximum 2TB limit of 64-bit Windows Server 2008 Datacenter doesn't come from any implementation or hardware limitation, but Microsoft will only support configurations they can test. As of the release of Windows Server 2008, the largest system available anywhere was 2TB and so Windows caps its use of physical memory there.

Windows Client Memory Limits

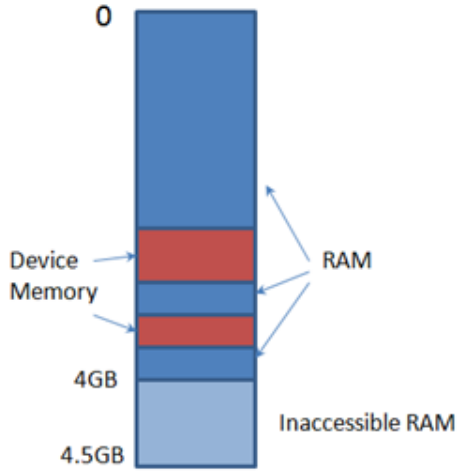
64-bit Windows client SKUs support different amounts of memory as a SKU-differentiating feature, with the low end being 512MB for Windows XP Starter to 128GB for Vista Ultimate and 192GB for Windows 7 Ultimate. All 32-bit Windows client SKUs, however, including Windows Vista, Windows XP and Windows 2000 Professional, support a maximum of 4GB of physical memory. 4GB is the highest physical address accessible with the standard x86 memory management mode. Originally, there was no need to even consider support for more than 4GB on clients because that amount of memory was rare, even on servers.

However, by the time Windows XP SP2 was under development, client systems with more than 4GB were foreseeable, so the Windows team started broadly testing Windows XP on systems with more than 4GB of memory. Windows XP SP2 also enabled Physical Address Extensions (PAE) support by default on hardware that implements no-execute memory because its required for Data Execution Prevention (DEP), but that also enables support for more than 4GB of memory.

What they found was that many of the systems would crash, hang, or become unbootable because some device drivers, commonly those for video and audio devices that are found typically on clients but not servers, were not programmed to expect physical addresses larger than 4GB. As a result, the drivers truncated such addresses, resulting in memory corruptions and corruption side effects. Server systems commonly have more generic devices and with simpler and more stable drivers, and therefore hadn't generally surfaced these problems. The problematic client driver ecosystem led to the decision for client SKUs to ignore physical memory that resides above 4GB, even though they can theoretically address it.

32-bit Client Effective Memory Limits

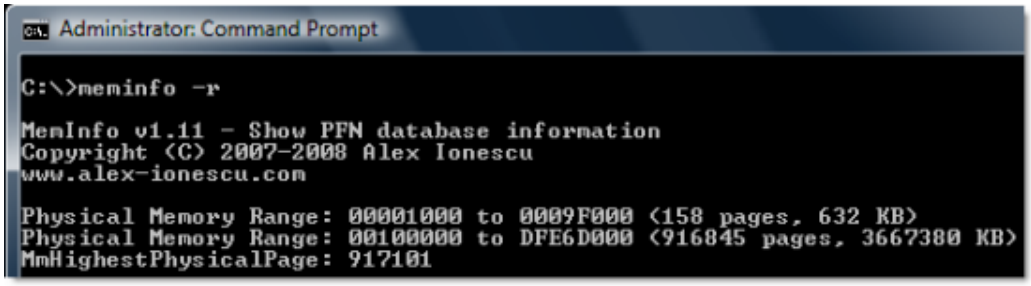
While 4GB is the licensed limit for 32-bit client SKUs, the effective limit is actually lower and dependent on the system's chipset and connected devices. The reason is that the physical address map includes not only RAM, but device memory as well, and x86 and x64 systems map all device memory below the 4GB address boundary to remain compatible with 32-bit operating systems that don't know how to handle addresses larger than 4GB. If a system has 4GB RAM and devices, like video, audio and network adapters, that implement windows into their device memory that sum to 500MB, 500MB of the 4GB of RAM will reside above the 4GB address boundary, as seen below:



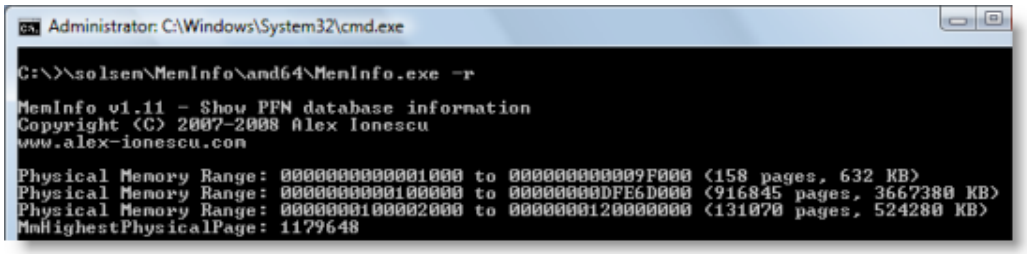
The result is that, if you have a system with 3GB or more of memory and you are running a 32-bit Windows client, you may not be getting the benefit of all of the RAM. On Windows 2000, Windows XP and Windows Vista RTM, you can see how much RAM Windows has accessible to it in the System Properties dialog, Task Manager's Performance page, and, on Windows XP and Windows Vista (including SP1), in the Msinfo32 and Winver utilities. On Window Vista SP1, some of these locations changed to show installed RAM, rather than available RAM, as documented in this Knowledge Base article .
On my 4GB laptop, when booted with 32-bit Vista, the amount of physical memory available is 3.5GB, as seen in the Msinfo32 utility:

Installed Physical Memory (RAM)	4.00 GB
Total Physical Memory	3.50 GB

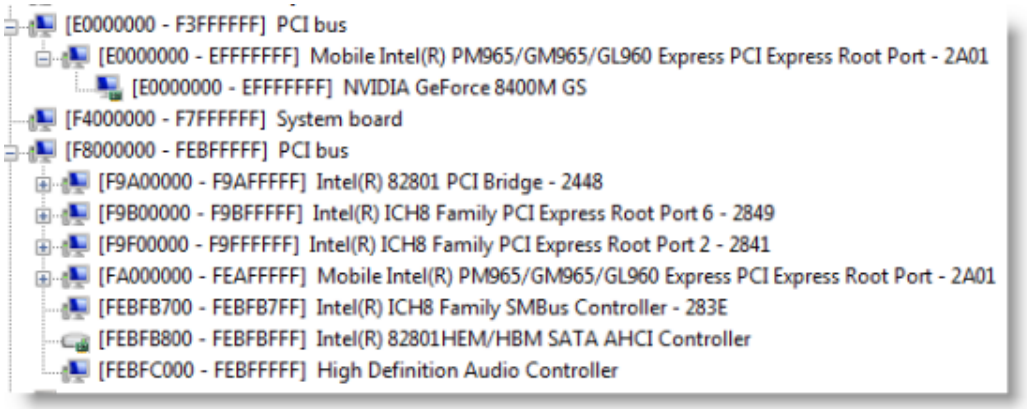
You can see physical memory layout with the Meminfo tool by Alex Ionescu (who's contributing to the 5th Edition of the Windows Internals that I'm coauthoring with David Solomon). Here's the output of Meminfo when I run it on that system with the -r switch to dump physical memory ranges:



Note the gap in the memory address range from page 9F0000 to page 100000, and another gap from DFE6D000 to FFFFFFFF (4GB). However, when I boot that system with 64-bit Vista, all 4GB show up as available and you can see how Windows uses the remaining 500MB of RAM that are above the 4GB boundary:

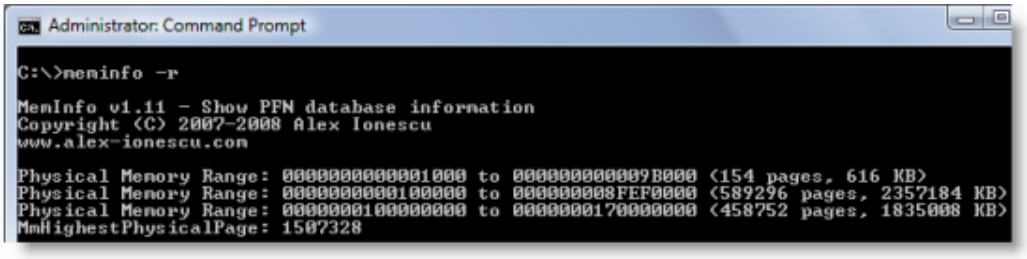


What's occupying the holes below 4GB? The Device Manager can answer that question. To check, launch "devmgmt.msc", select Resources by Connection in the View Menu, and expand the Memory node. On my laptop, the primary consumer of mapped device memory is, unsurprisingly, the video card, which consumes 256MB in the range E0000000-EFFFFFFF:

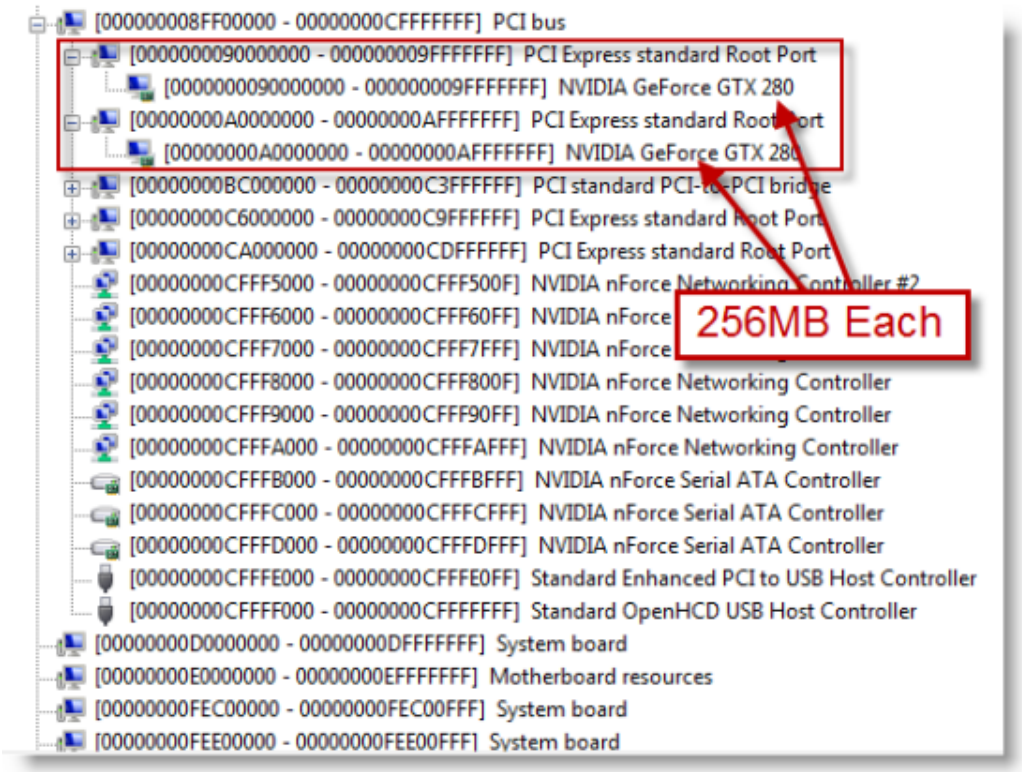


Other miscellaneous devices account for most of the rest, and the PCI bus reserves additional ranges for devices as part of the conservative estimation the firmware uses during boot.

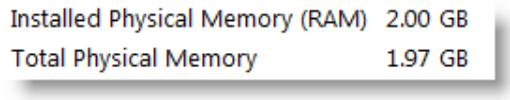
The consumption of memory addresses below 4GB can be drastic on high-end gaming systems with large video cards. For example, I purchased one from a boutique gaming rig company that came with 4GB of RAM and two 1GB video cards. I hadn't specified the OS version and assumed that they'd put 64-bit Vista on it, but it came with the 32-bit version and as a result only 2.2GB of the memory was accessible by Windows. You can see a giant memory hole from 8FEF0000 to FFFFFFFF in this Meminfo output from the system after I installed 64-bit Windows:



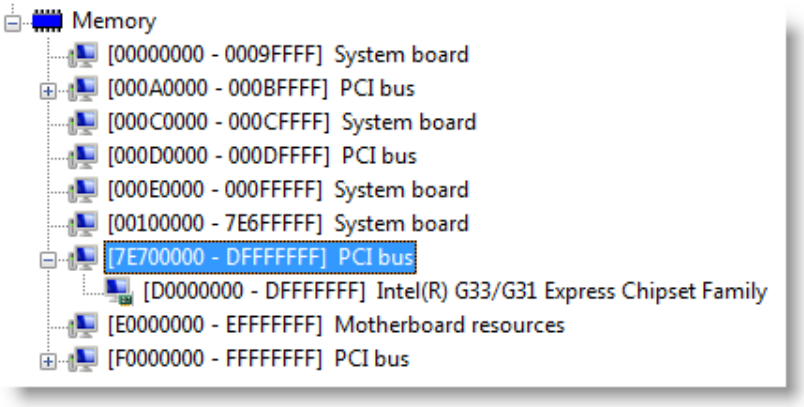
Device Manager reveals that 512MB of the over 2GB hole is for the video cards (256MB each), and it looks like the firmware has reserved more for either dynamic mappings or because it was conservative in its estimate:



Even systems with as little as 2GB can be prevented from having all their memory usable under 32-bit Windows because of chipsets that aggressively reserve memory regions for devices. Our shared family computer, which we purchased only a few months ago from a major OEM, reports that only 1.97GB of the 2GB installed is available:



The physical address range from 7E700000 to FFFFFFFF is reserved by the PCI bus and devices, which leaves a theoretical maximum of 7E700000 bytes (1.976GB) of physical address space, but even some of that is reserved for device memory, which explains why Windows reports 1.97GB.



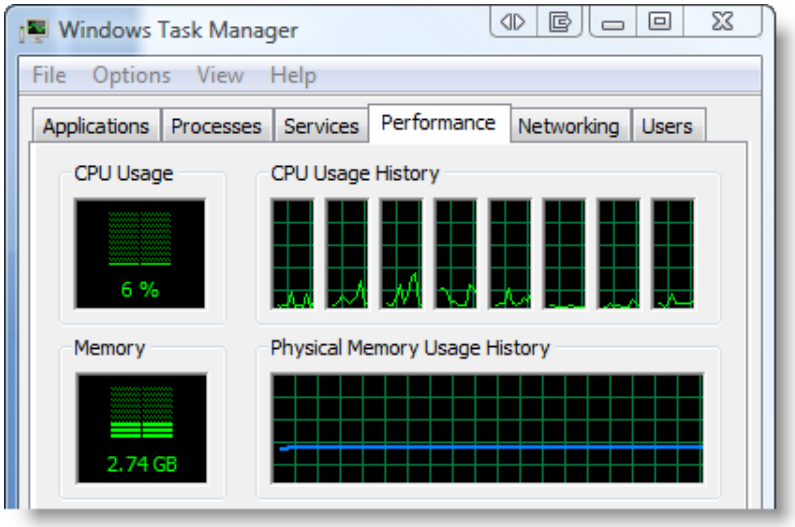
Because device vendors now have to submit both 32-bit and 64-bit drivers to Microsoft's Windows Hardware Quality Laboratories (WHQL) to obtain a driver signing certificate, the majority of device drivers today can probably handle physical addresses above the 4GB line. However, 32-bit Windows will continue to ignore memory above it because there is still some difficult to measure risk, and OEMs are (or at least should be) moving to 64-bit Windows where it's not an issue.

The bottom line is that you can fully utilize your system's memory (up the SKU's limit) with 64-bit Windows, regardless of the amount, and if you are purchasing a high end gaming system you should definitely ask the OEM to put 64-bit Windows on it at the factory.

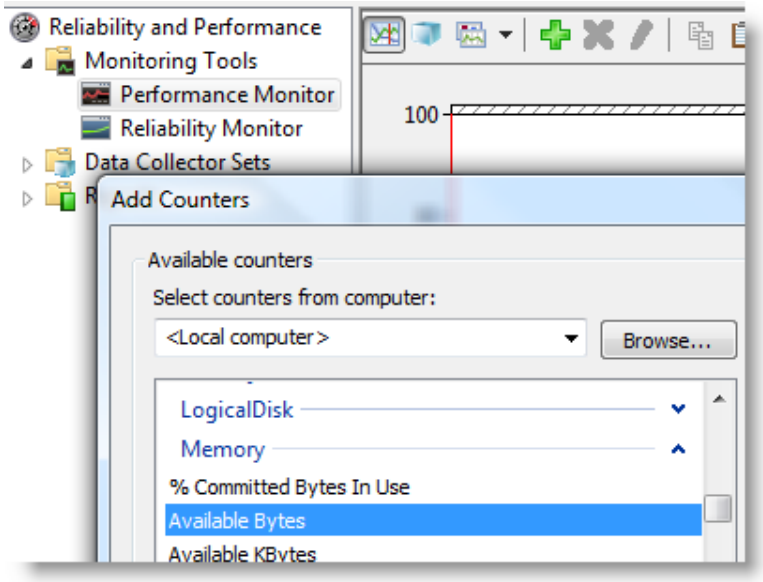
Do You Have Enough Memory?

Regardless of how much memory your system has, the question is, is it enough? Unfortunately, there's no hard and fast rule that allows you to know with certainty. There is a general guideline you can use that's based on monitoring the system's "available" memory over time, especially when you're running memory-intensive workloads. Windows defines available memory as physical memory that's not assigned to a process, the kernel, or device drivers. As its name implies, available memory is available for assignment to a process or the system if required. The Memory Manager of course tries to make the most of that memory by using it as a file cache (the standby list), as well as for zeroed memory (the zero page list), and Vista's Superfetch feature prefetches data and code into the standby list and prioritizes it to favor data and code likely to be used in the near future.

If available memory becomes scarce, that means that processes or the system are actively using physical memory, and if it remains close to zero over extended periods of time, you can probably benefit by adding more memory. There are a number of ways to track available memory. On Windows Vista, you can indirectly track available memory by watching the Physical Memory Usage History in Task Manager, looking for it to remain close to 100% over time. Here's a screenshot of Task Manager on my 8GB desktop system (hmm, I think I might have too much memory!):



On all versions of Windows you can graph available memory using the Performance Monitor by adding the Available Bytes counter in the Memory performance counter group:



You can see the instantaneous value in Process Explorer's System Information dialog, or, on versions of Windows prior to Vista, on Task Manager's Performance page.

Pushing the Limits of Windows

Out of CPU, memory and disk, memory is typically the most important for overall system performance. The more, the better. 64-bit Windows is the way to go to be sure that you're taking advantage of all of it, and 64-bit Windows can have other performance benefits that I'll talk about in a future Pushing the Limits blog post when I talk about virtual memory limits.