

Mark's Blog

Mark Russinovich's technical blog covering topics such as Windows troubleshooting, technologies and security.

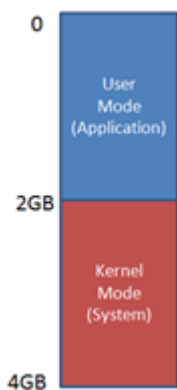
Pushing the Limits of Windows: Virtual Memory

In my [first Pushing the Limits of Windows post](#), I discussed physical memory limits, including the limits imposed by licensing, implementation, and driver compatibility. This time I'm turning my attention to another fundamental resource, virtual memory. Virtual memory separates a program's view of memory from the system's physical memory, so an operating system decides when and if to store the program's code and data in physical memory and when to store it in a file. The major advantage of virtual memory is that it allows more processes to execute concurrently than might otherwise fit in physical memory.

While virtual memory has limits that are related to physical memory limits, virtual memory has limits that derive from different sources and that are different depending on the consumer. For example, there are virtual memory limits that apply to individual processes that run applications, the operating system, and for the system as a whole. It's important to remember as you read this that virtual memory, as the name implies, has no direct connection with physical memory. Windows assigning the file cache a certain amount of virtual memory does not dictate how much file data it actually caches in physical memory; it can be any amount from none to more than the amount that's addressable via virtual memory.

Process Address Spaces

Each process has its own virtual memory, called an address space, into which it maps the code that it executes and the data that the code references and manipulates. A 32-bit process uses 32-bit virtual memory address pointers, which creates an absolute upper limit of 4GB (2^{32}) for the amount of virtual memory that a 32-bit process can address. However, so that the operating system can reference its own code and data and the code and data of the currently-executing process without changing address spaces, the operating system makes its virtual memory visible in the address space of every process. By default, 32-bit versions of Windows split the process address space evenly between the system and the active process, creating a limit of 2GB for each:



Applications might use Heap APIs, the .NET garbage collector, or the C runtime malloc library to allocate virtual memory, but under the hood all of these rely on the [VirtualAlloc](#) API. When an application runs out of address space then VirtualAlloc, and therefore the memory managers layered on top of it, return errors (represented by a NULL address). The Testlimit utility, which I wrote for the [4th Edition of Windows Internals](#) to demonstrate various Windows limits, calls VirtualAlloc repeatedly


U
T

http://blogs.technet.com/markrussinovich/archive/2008/11/17/3155406.aspx Go NOV DEC JAN
135 captures
8 Dec 2008 - 6 Nov 2020 2007 2008 2010 About this capture

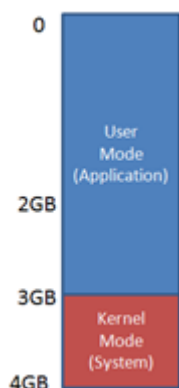
```
Testlimit v5.0 - tests windows limits
By Mark Russinovich

Reserving private bytes (MB)...
Reserved 2010 MB of private memory. Lasterror: 8
Not enough storage is available to process this command.
```

2010 MB isn't quite 2GB, but Testlimit's other code and data, including its executable and system DLLs, account for the difference. You can see the total amount of address space it's consumed by looking at its Virtual Size in [Process Explorer](#):

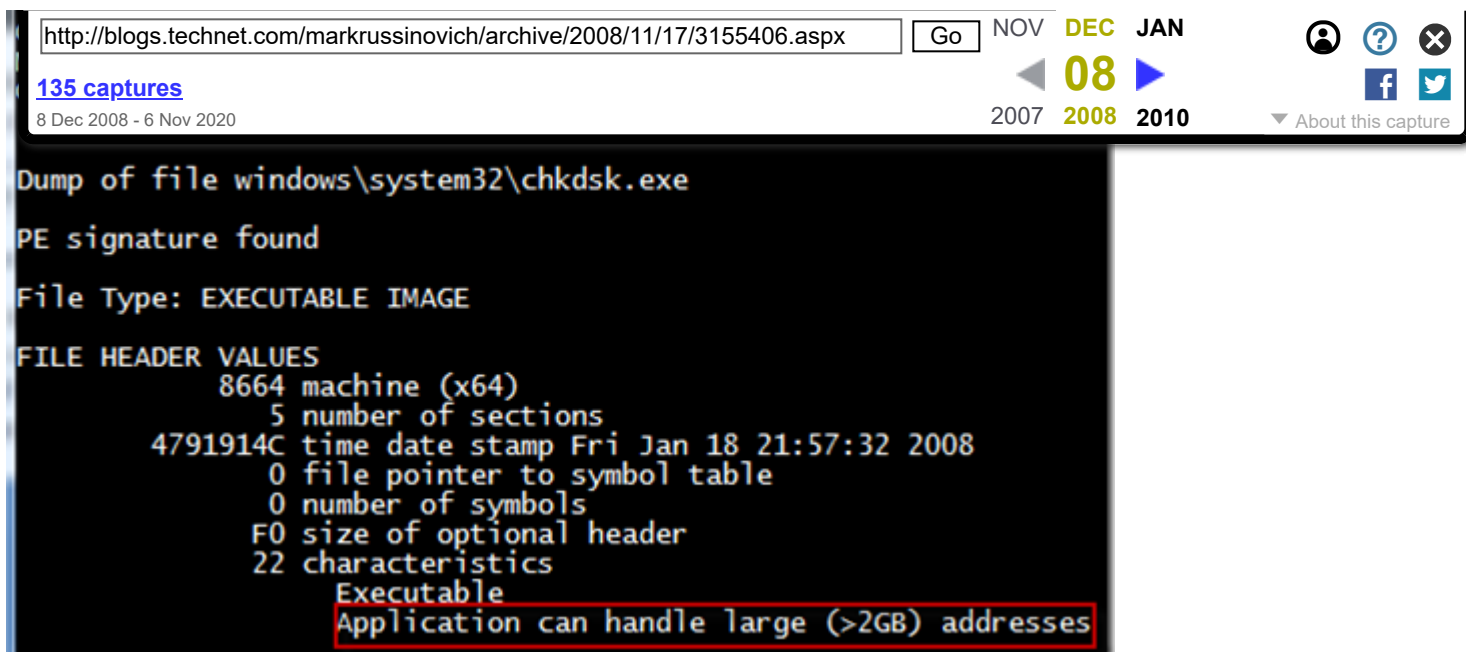
Process	Virtual Size
 Testlimit.exe	2,089,792 K

Some applications, like SQL Server and Active Directory, manage large data structures and perform better the more that they can load into their address space at the same time. Windows NT 4 SP3 therefore introduced a boot option, [/3GB](#), that gives a process 3GB of its 4GB address space by reducing the size of the system address space to 1GB, and Windows XP and Windows Server 2003 introduced the [/userva](#) option that moves the split anywhere between 2GB and 3GB:



To take advantage of the address space above the 2GB line, however, a process must have the 'large address space aware' flag set in its executable image. Access to the additional virtual memory is opt-in because some applications have assumed that they'd be given at most 2GB of the address space. Since the high bit of a pointer referencing an address below 2GB is always zero, they would use the high bit in their pointers as a flag for their own data, clearing it of course before referencing the data. If they ran with a 3GB address space they would inadvertently truncate pointers that have values greater than 2GB, causing program errors including possible data corruption.

All Microsoft server products and data intensive executables in Windows are marked with the large address space awareness flag, including Chkdsk.exe, Lsass.exe (which hosts Active Directory services on a domain controller), Smss.exe (the session manager), and Esentutl.exe (the Active Directory Jet database repair tool). You can see whether an image has the flag with the Dumpbin utility, which comes with Visual Studio:



Testlimit is also marked large-address aware, so if you run it with the `-r` switch when booted with the 3GB of user address space, you'll see something like this:

```
C:\>testlimit -r

Testlimit v5.0 - tests windows limits
By Mark Russinovich

Reserving private bytes (MB)...
Reserved 2962 MB of private memory. Lasterror: 8
Not enough storage is available to process this command.
```

Because the address space on 64-bit Windows is much larger than 4GB, something I'll describe shortly, Windows can give 32-bit processes the maximum 4GB that they can address and use the rest for the operating system's virtual memory. If you run Testlimit on 64-bit Windows, you'll see it consume the entire 32-bit addressable address space:

```
C:\>testlimit -r

Testlimit v5.0 - tests windows limits
By Mark Russinovich

Reserving private bytes (MB)...
Reserved 4026 MB of private memory. Lasterror: 8
Not enough storage is available to process this command.
```

64-bit processes use 64-bit pointers, so their theoretical maximum address space is 16 exabytes (2^{64}). However, Windows doesn't divide the address space evenly between the active process and the system, but instead defines a region in the address space for the process and others for various system memory resources, like system page table entries (PTEs), the file cache, and paged and non-paged pools.

The size of the process address space is different on IA64 and x64 versions of Windows where the sizes were chosen by balancing what applications need against the memory costs of the overhead (page table pages and translation lookaside buffer - TLB - entries) needed to support the address space. On x64, that's 8192GB (8TB) and on IA64 it's 7168GB (7TB - the 1TB difference from x64 comes from the fact that the top level page directory on IA64 reserves slots for Wow64 mappings). On both IA64 and x64 versions of Windows, the size of the various resource address space regions is 128GB (e.g. non-paged pool is assigned 128GB of the address space), with the exception of the file cache, which is assigned 1TB. The address space of a 64-bit process therefore looks something like this:

http://blogs.technet.com/markrussinovich/archive/2008/11/17/3155406.aspx Go NOV DEC JAN

135 captures 8 Dec 2008 - 6 Nov 2020 2007 2008 2010 About this capture

2⁶⁴

File Cache
Non-paged Pool
Paged Pool
System PTEs

The figure isn't drawn to scale, because even 8TB, much less 128GB, would be a small sliver. Suffice it to say that like our universe, there's a lot of emptiness in the address space of a 64-bit process.

When you run the 64-bit version of Testlimit (Testlimit64) on 64-bit Windows with the `-r` switch, you'll see it consume 8TB, which is the size of the part of the address space it can manage:

```
C:\>testlimit64 -r

Testlimit v5.0 - tests Windows limits
By Mark Russinovich

Reserving private bytes (MB)...
Reserved 8388544 MB of private memory. Lasterror: 8
Not enough storage is available to process this command.
```

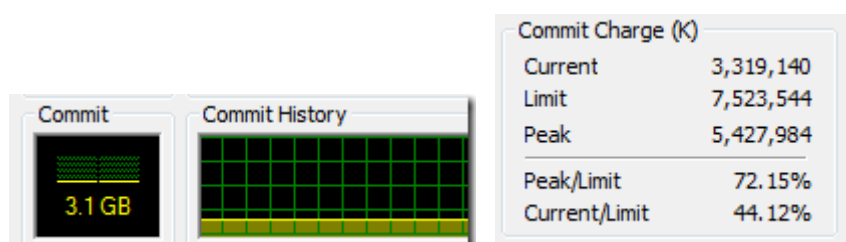
Process	Virtual Size
testlimit64.exe	8,589,913,132 K

Committed Memory

Testlimit's `-r` switch has it reserve virtual memory, but not actually *commit* it. Reserved virtual memory can't actually store data or code, but applications sometimes use a reservation to create a large block of virtual memory and then commit it as needed to ensure that the committed memory is contiguous in the address space. When a process commits a region of virtual memory, the operating system guarantees that it can maintain all the data the process stores in the memory either in physical memory or on disk. That means that a process can run up against another limit: the *commit limit*.

As you'd expect from the description of the commit guarantee, the commit limit is the sum of physical memory and the sizes of the paging files. In reality, not quite all of physical memory counts toward the commit limit since the operating system reserves part of physical memory for its own use. The amount of committed virtual memory for all the active processes, called the *current commit charge*, cannot exceed the system commit limit. When the commit limit is reached, virtual allocations that commit memory fail. That means that even a standard 32-bit process may get virtual memory allocation failures before it hits the 2GB address space limit.

The current commit charge and commit limit is tracked by Process Explorer in its System Information window in the Commit Charge section and in the Commit History bar chart and graph:



12/31/2020

Mark's Blog : Pushing the Limits of Windows: Virtual Memory

http://blogs.technet.com/markrussinovich/archive/2008/11/17/3155406.aspx Go NOV DEC JAN

135 captures

8 Dec 2008 - 6 Nov 2020

2007 2008 2010

About this capture

Totals		Physical Memory (K)	
Handles	5522	Total	523760
Threads	329	Available	389268
Processes	25	System Cache	210728

Commit Charge (K)		Kernel Memory (K)	
Total	115516	Total	25840
Limit	1278776	Paged	21336
Peak	242464	Nonpaged	4504

On Vista and Server 2008, Task Manager doesn't show the commit charge graph and labels the current commit charge and limit values with "Page File" (despite the fact that they will be non-zero values even if you have no paging file):

System	
Handles	31401
Threads	1107
Processes	79
Up Time	1:41:00
Page File	3153M / 16571M

You can stress the commit limit by running Testlimit with the -m switch, which directs it to allocate committed memory. The 32-bit version of Testlimit may or may not hit its address space limit before hitting the commit limit, depending on the size of physical memory, the size of the paging files and the current commit charge when you run it. If you're running 32-bit Windows and want to see how the system behaves when you hit the commit limit, simply run multiple instances of Testlimit until one hits the commit limit before exhausting its address space.

Note that, by default, the paging file is configured to grow, which means that the commit limit will grow when the commit charge nears it. And even when the paging file hits its maximum size, Windows is holding back some memory and its internal tuning, as well as that of applications that cache data, might free up more. Testlimit anticipates this and when it reaches the commit limit, it sleeps for a few seconds and then tries to allocate more memory, repeating this indefinitely until you terminate it.

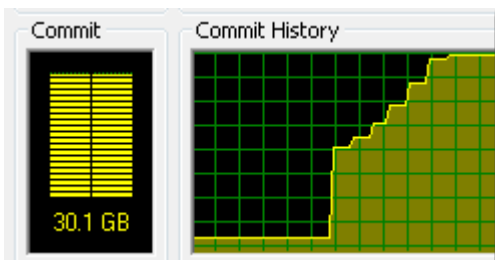
If you run the 64-bit version of Testlimit, it will almost certainly will hit the commit limit before exhausting its address space, unless physical memory and the paging files sum to more than 8TB, which as described previously is the size of the 64-bit application-accessible address space. Here's the partial output of the 64-bit Testlimit running on my 8GB system (I specified an allocation size of 100MB to make it leak more quickly):

http://blogs.technet.com/markrussinovich/archive/2008/11/17/3155406.aspx Go NOV DEC JAN 08 2007 2008 2010 About this capture

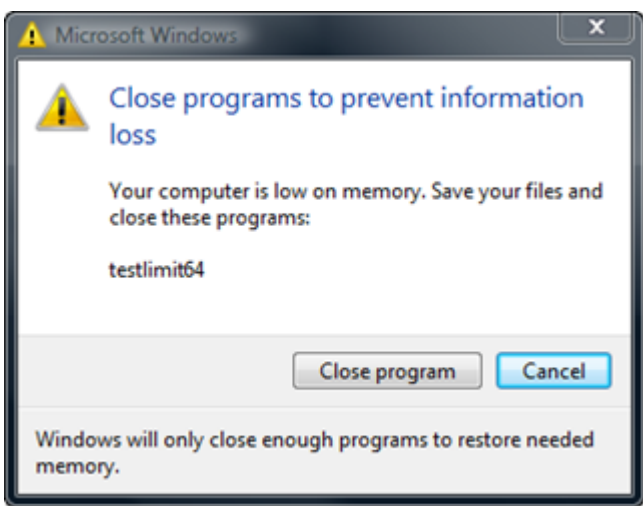
135 captures
8 Dec 2008 - 6 Nov 2020

```
Leaking private bytes 100 MB at a time...
Leaked 13900 MB of private memory (13900 MB total leaked). Lasterror: 1455
The paging file is too small for this operation to complete.
Leaked 2200 MB of private memory (16100 MB total leaked). Lasterror: 1455
The paging file is too small for this operation to complete.
Leaked 2600 MB of private memory (18700 MB total leaked). Lasterror: 1455
The paging file is too small for this operation to complete.
Leaked 2600 MB of private memory (21300 MB total leaked). Lasterror: 1455
The paging file is too small for this operation to complete.
Leaked 3400 MB of private memory (24700 MB total leaked). Lasterror: 1455
The paging file is too small for this operation to complete.
Leaked 3900 MB of private memory (28600 MB total leaked). Lasterror: 1455
The paging file is too small for this operation to complete.
Leaked 200 MB of private memory (28800 MB total leaked). Lasterror: 1455
The paging file is too small for this operation to complete.
Leaked 0 MB of private memory (28800 MB total leaked). Lasterror: 1455
The paging file is too small for this operation to complete.
Sleeping for 5 seconds to allow for paging file expansion...
```

And here's the commit history graph with steps when Testlimit paused to allow the paging file to grow:



When system virtual memory runs low, applications may fail and you might get strange error messages when attempting routine operations. In most cases, though, Windows will be able present you the low-memory resolution dialog, like it did for me when I ran this test:








After you exit Testlimit, the commit limit will likely drop again when the memory manager truncates the tail of the paging file that it created to accommodate Testlimit's extreme commit requests. Here, Process Explorer shows that the current limit is well below the peak that was achieved when Testlimit was running:

<http://blogs.technet.com/markrussinovich/archive/2008/11/17/3155406.aspx> NOV DEC JAN
◀ 08 ▶
2007 2008 2010

135 captures
8 Dec 2008 - 6 Nov 2020

Peak/Limit 197.01%
Current/Limit 18.95%

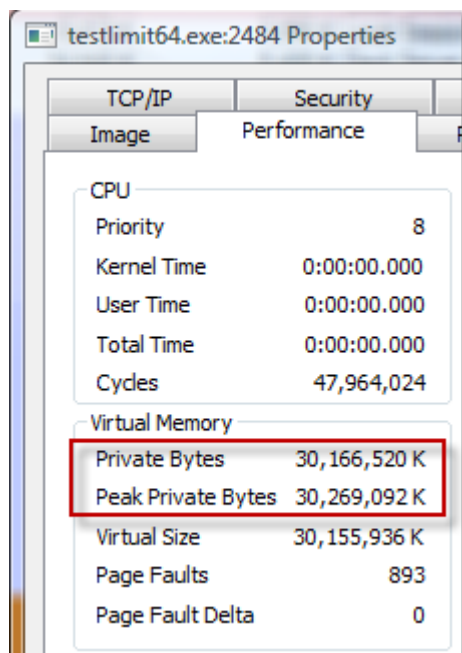
About this capture

Process Committed Memory

Because the commit limit is a global resource whose consumption can lead to poor performance, application failures and even system failure, a natural question is 'how much are processes contributing the commit charge'? To answer that question accurately, you need to understand the different types of virtual memory that an application can allocate.

Not all the virtual memory that a process allocates counts toward the commit limit. As you've seen, reserved virtual memory doesn't. Virtual memory that represents a file on disk, called a file mapping view, also doesn't count toward the limit unless the application asks for copy-on-write semantics, because Windows can discard any data associated with the view from physical memory and then retrieve it from the file. The virtual memory in Testlimit's address space where its executable and system DLL images are mapped therefore don't count toward the commit limit. There are two types of process virtual memory that do count toward the commit limit: private and pagefile-backed.

Private virtual memory is the kind that underlies the garbage collector heap, native heap and language allocators. It's called private because by definition it can't be shared between processes. For that reason, it's easy to attribute to a process and Windows tracks its usage with the Private Bytes performance counter. Process Explorer displays a process private bytes usage in the Private Bytes column, in the Virtual Memory section of the Performance page of the process properties dialog, and displays it in graphical form on the Performance Graph page of the process properties dialog. Here's what Testlimit64 looked like when it hit the commit limit:



http://blogs.technet.com/markrussinovich/archive/2008/11/17/3155406.aspx Go NOV DEC JAN

135 captures 8 Dec 2008 - 6 Nov 2020 2007 2008 2010 About this capture

28.8 GB

Pagefile-backed virtual memory is harder to attribute, because it can be shared between processes. In fact, there's no process-specific counter you can look at to see how much a process has allocated or is referencing. When you run Testlimit with the -s switch, it allocates pagefile-backed virtual memory until it hits the commit limit, but even after consuming over 29GB of commit, the virtual memory statistics for the process don't provide any indication that it's the one responsible:

Virtual Memory	
Private Bytes	2,024 K
Peak Private Bytes	2,024 K
Virtual Size	49,588 K
Page Faults	883
Page Fault Delta	0

For that reason, I added the -l switch to Handle a while ago. A process must open a pagefile-backed virtual memory object, called a section, for it to create a mapping of pagefile-backed virtual memory in its address space. While Windows preserves existing virtual memory even if an application closes the handle to the section that it was made from, most applications keep the handle open. The -l switch prints the size of the allocation for pagefile-backed sections that processes have open. Here's partial output for the handles open by Testlimit after it has run with the -s switch:

```
C:\Windows\system32>handle -a -l -p testlimit64

Handle v3.42
Copyright (C) 1997-2008 Mark Russinovich
Sysinternals - www.sysinternals.com

-----
testlimit64.exe pid: 2676 NTDEV\markruss
38: Section
   Pagefile      1048576 bytes
3C: Section
   Pagefile      1048576 bytes
40: Section
   Pagefile      1048576 bytes
44: Section
   Pagefile      1048576 bytes
```

You can see that Testlimit is allocating pagefile-backed memory in 1MB blocks and if you summed the size of all the sections it had opened, you'd see that it was at least one of the processes contributing large amounts to the commit charge.

How Big Should I Make the Paging File?

Perhaps one of the most commonly asked questions related to virtual memory is, how big should I make the paging file? There's no end of ridiculous advice out on the web and in the newsstand magazines that cover Windows, and even Microsoft has published misleading recommendations. Almost all the suggestions are based on multiplying RAM size by some factor, with common values being 1.2, 1.5 and 2. Now that you understand the role that the paging file plays in defining a system's commit limit and how processes contribute to the commit charge, you're well positioned to see how useless such formulas truly are.

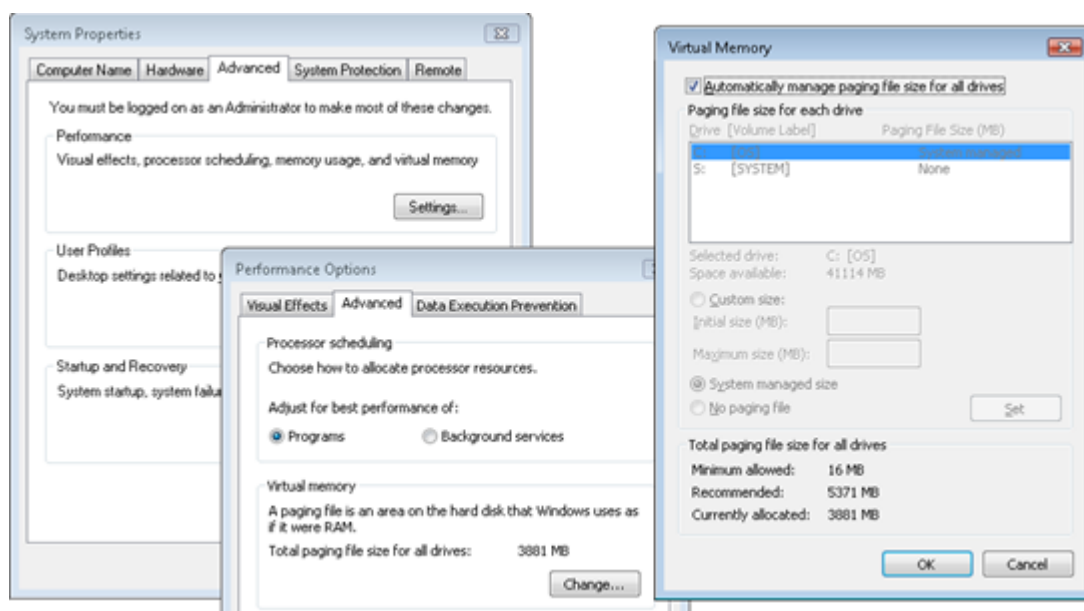
http://blogs.technet.com/markrussinovich/archive/2008/11/17/3155406.aspx Go NOV DEC JAN 08 2007 2008 2010 About this capture

allocate the virtual memory they want and will fail to run properly.

So how do you know how much commit charge your workloads require? You might have noticed in the screenshots that Windows tracks that number and Process Explorer shows it: Peak Commit Charge. To optimally size your paging file you should start all the applications you run at the same time, load typical data sets, and then note the commit charge peak (or look at this value after a period of time where you know maximum load was attained). Set the paging file minimum to be that value minus the amount of RAM in your system (if the value is negative, pick a minimum size to permit the kind of crash dump you are configured for). If you want to have some breathing room for potentially large commit demands, set the maximum to double that number.

Some feel having no paging file results in better performance, but in general, having a paging file means Windows can write pages on the modified list (which represent pages that aren't being accessed actively but have not been saved to disk) out to the paging file, thus making that memory available for more useful purposes (processes or file cache). So while there may be some workloads that perform better with no paging file, in general having one will mean more usable memory being available to the system (never mind that Windows won't be able to write kernel crash dumps without a paging file sized large enough to hold them).

Paging file configuration is in the System properties, which you can get to by typing "sysdm.cpl" into the Run dialog, clicking on the Advanced tab, clicking on the Performance Options button, clicking on the Advanced tab (this is *really* advanced), and then clicking on the Change button:



You'll notice that the default configuration is for Windows to automatically manage the page file size. When that option is set on Windows XP and Server 2003, Windows creates a single paging file that's minimum size is 1.5 times RAM if RAM is less than 1GB, and RAM if it's greater than 1GB, and that has a maximum size that's three times RAM. On Windows Vista and Server 2008, the minimum is intended to be large enough to hold a kernel-memory crash dump and is RAM plus 300MB or 1GB, whichever is larger. The maximum is either three times the size of RAM or 4GB, whichever is larger. That explains why the peak commit on my 8GB 64-bit system that's visible in one of the screenshots is 32GB. I guess whoever wrote that code got their guidance from one of those magazines I mentioned!

A couple of final limits related to virtual memory are the maximum size and number of paging files supported by Windows. 32-bit Windows has a maximum paging file size of 16TB (4GB if you for some reason run in non-PAE mode) and 64-bit Windows can having paging files that are up to 16TB in size

http://blogs.technet.com/markrussinovich/archive/2008/11/17/3155406.aspx

Go

NOV DEC JAN

08

2007 2008 2010

135 captures

8 Dec 2008 - 6 Nov 2020

About this capture

Comment Notification

If you would like to receive an email when updates are made to this post, please register [here](#)

Subscribe to this post's comments using [RSS](#)

Comments

re: Pushing the Limits of Windows: Virtual Memory

Brilliant post Mark. Thanks for writing this.

As an Active Directory guy I wanted to remind readers to beware of configuring DCs without any page file, even on DCs with lots of RAM and where the commit charge peak is low relative to the commit limit. The database engine depends on the page file per KB 889654.

Cheers

Tuesday, November 18, 2008 9:47 PM by Matthew Reynolds

re: Pushing the Limits of Windows: Virtual Memory

Damn

Love reading your posts, keep it up!

Tuesday, November 18, 2008 11:30 PM by [Michael Sainz](#)

re: Pushing the Limits of Windows: Virtual Memory

I was involved in choosing the default min/max sizes for system managed pagefiles in Vista, and I'm pretty sure those numbers were not just copied from some magazine :)

The 1 GB minimum was chosen based on the actual commit charge observed on small machines (512 MB of RAM). The 3*RAM maximum might seem excessive on machines with lots of RAM, but remember that pagefile will only grow this large if there is actual demand. Also, running out of commit (for example, because of a leak in some app) can bring the entire system to a halt, and a higher maximum size can make the difference between a system that does not respond and has to be rebooted and a system that can be recovered by restarting a process.

I will admit that scaling the maximum size linearly with the size of RAM is somewhat arbitrary. Perhaps it should have been a fixed constant instead.

Wednesday, November 19, 2008 2:30 AM by Pavel Lebedinsky

re: Pushing the Limits of Windows: Virtual Memory

Great post (as always)! I love the amount of detail in your posts.

Everyone looks at task manager and tries to draw some conclusions, but thanks to you I am starting to really understand what the numbers mean.

Hugo

Wednesday, November 19, 2008 2:40 AM by [Hugo Peeters](#)

re: Pushing the Limits of Windows: Virtual Memory

A few more points:

1. Reserved memory does contribute to commit charge, because the memory manager charges commit for pageable space necessary to map the entire reserved range. On 64 bit this can be a significant number (reserving 1 TB of memory will consume approximately 2 GB of commit).

http://blogs.technet.com/markrussinovich/archive/2008/11/17/3155406.aspx

Go

NOV DEC JAN

08

2007 2008 2010

About this capture

135 captures

8 Dec 2008 - 6 Nov 2020

Facebook Twitter

3. Configuring a system with lots of RAM to run without pagefile may have either negative or positive perf impact depending on what the system is doing. The general recommendation in this case is to create a reasonably sized pagefile (for example, 4 GB) and increase it if the Paging file\% Usage counter gets close to 100%.

Note that this counter is completely different from what task manager calls "pagefile usage" (which is actually the system commit charge). Paging file\% Usage of 100% would mean that some unused pagefile-backed pages are sitting on the modified page list, unnecessarily taking up RAM. If pagefile was larger, those pages could have been written to disk, resulting in more RAM available for other purposes.

Wednesday, November 19, 2008 3:16 AM by Pavel Lebedinsky

Excellent article!

I think I never read a more fundamental article about managing the Windows Virtual Memory. Thank you very much for this, I think many people will now be able to reasonably set the size of the Virtual Memory.

"I guess whoever wrote that code got their guidance from one of those magazines I mentioned!" made me giggle! :-)

Wednesday, November 19, 2008 3:47 AM by Osvaldo Tabasco

Multiple page files on one volume

It's worth noting that previous Windows versions (as late as Windows 2000) permitted the creation of multiple page files on a single volume with a registry edit.

This was primarily used when the formal recommendation from Microsoft was for 1.5x RAM (Exchange Server anyone?) and the system had more than 4GB of RAM. Using the "one page per volume" strategy required 2 or more drive letters.

Instead, each pagefile could be 4GB and placed in a separate directory.

I think it was "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\Pagingfiles" but my memory is not perfect.

Wednesday, November 19, 2008 6:46 AM by David Rawling

re: Pushing the Limits of Windows: Virtual Memory

nice one!

but what do i do with the pagefile on a 64 bit server 2003 system with 32gb ram, with a memory-intensive application? ... let's call the application ... exchange 2k7 :-)

Wednesday, November 19, 2008 7:57 AM by robad

re: Pushing the Limits of Windows: Virtual Memory

I just bought a Gigabyte i-RAM RAM DISK to put my "virtual memory" on a RAM DISK. I didn't realize accessing virtual memory is application specific, so this is great information! Thank you.

Wednesday, November 19, 2008 11:48 AM by TekGems

re: Pushing the Limits of Windows: Virtual Memory

"To take advantage of the address space above the 2GB line, however, a process must have the 'large address space aware' flag set in its executable image."

Hmmm, I looked around and the closest I could find in the PE executable file format was from this:

<http://msdn.microsoft.com/en-us/library/ms809762.aspx>

http://blogs.technet.com/markrussinovich/archive/2008/11/17/3155406.aspx

Go

NOV DEC JAN

08

2007 2008 2010

135 captures

8 Dec 2008 - 6 Nov 2020

About this capture

The amount of virtual memory to reserve for the initial process heap. This heap's handle can be obtained by calling `GetProcessHeap`. Not all of this memory is committed (see the next field)."

So windows uses something like the `SizeOfHeapReserve` field in the executable file to be able to tell whether the process should be allocated more than 2GB of virtual memory to the usermode process? Interesting post as usual Mark, I learned quite a bit.

Wednesday, November 19, 2008 11:56 AM by o.s.

re: Pushing the Limits of Windows: Virtual Memory

o.s.: no, its a bit flag in `DllCharacteristics IIRC`

Wednesday, November 19, 2008 12:35 PM by asf

re: Pushing the Limits of Windows: Virtual Memory

Another fantastic post!! The detailed level of your articles is completely awesome!

Wednesday, November 19, 2008 12:55 PM by Lester Eliasquevitch

re: Pushing the Limits of Windows: Virtual Memory

Large address awareness is specified with the `IMAGE_FILE_LARGE_ADDRESS_AWARE` flag in the `IMAGE_FILE_HEADER` structure of the PE header:

<http://msdn.microsoft.com/en-us/library/ms680313.aspx>

Wednesday, November 19, 2008 1:27 PM by [markrussinovich](#)

re: Pushing the Limits of Windows: Virtual Memory

Thanks Mark for the reply. I was so busy researching asf's comment that I hadn't checked back on the latest comments. Thanks everyone.

Wednesday, November 19, 2008 1:56 PM by o.s

re: Pushing the Limits of Windows: Virtual Memory

<quote>I was involved in choosing the default min/max sizes for system managed pagefiles in Vista, and I'm pretty sure those numbers were not just copied from some magazine :)</quote>

[snip]

Also, running out of commit (for example, because of a leak in some app) can bring the entire system to a halt, and a higher maximum size can make the difference between a system that does not respond and has to be rebooted and a system that can be recovered by restarting a process.</quote>

I think you got that backwards. Running out of physical RAM without running out of commit is what brings the system to a halt, because the system is paging in and out continually and thrashing the disk I/O subsystem, and may be unrecoverable without a reboot. OTOH running out of commit generally causes a fatal error to the application performing the more allocations, allowing other applications to continue normally and at full speed.

And it scares me that the person in charge of setting pagefile limits at Microsoft has such a fundamental misunderstanding of the problem.

Pagefiles are a hack to permit applications written to process data in batches but without regard to memory usage (i.e. data once processed should be freed immediately, but isn't until the task is completed) to run on larger datasets which don't fit fully in RAM but the working set does. If the working set doesn't fit in RAM, theoretically the pagefile allows processing to continue but practically the 1000000X slowdown from paging means this isn't really enabling anything. And an application which is designed to handle large problems can handle data sets bigger than memory as long as the working set fits without any pagefile whatsoever (allocate only the working set). The OS disk cache

is NOV DEC JAN
135 captures
8 Dec 2008 - 6 Nov 2020 2007 2008 2010 About this capture

or non-cached I/O accordingly. But that's something that only server-class software like SQL server would want to worry about.

Or at least that's my understanding.

Wednesday, November 19, 2008 2:30 PM by Ben Voigt [C++ MVP]

re: Pushing the Limits of Windows: Virtual Memory

Is this a fair summary of why the 3GB is an option that is off by default:

Either lots of 2GB processes at the same time with a 2GB system virtual memory to manage all those resources, or a limited number of 3GB processes with a smaller system virtual memory of 1GB to manage them?

And what is wrong with just letting Windows manage the page file?

Won't it choose a reasonable size and grow it if necessary? I always just assumed it would never shrink, but Mark seemed to suggest the page file will be truncated when the commit charge drops. Is setting a fixed size to avoid the stall as the swap file is grown or shrunk in size, or to avoid the on-screen message that, 'Windows will increase the size of the virtual memory file' - message that cannot be clicked if running on an unattended server?

On a workstation, is there really any point? Will people notice a difference?

Wednesday, November 19, 2008 2:50 PM by Joe Butler

re: Pushing the Limits of Windows: Virtual Memory

Great article, but I there's one important thing that should be mentioned: address space fragmentation. So I recommend reading also http://forall.ru-board.com/egor23/online/FAQ/Virtual_Memory/Limits_Virtual_Memory.html

Wednesday, November 19, 2008 4:08 PM by m^22

re: Pushing the Limits of Windows: Virtual Memory

three things

1. In the last paragraph, the maximum page file size on 32 Bit windows and 64 Bit windows seem to be the same, at 16 TB. Is this really right?
2. How is memory managed for DirectX (aka Games!). Obviously they still go through Kernel and such, but I know games often allocate memory to try to force only physical memory usage. Will having paging file help at all? How does OS behave?
3. On Vista 32-bit, is 3GB option enabled or disabled? And if disabled, how do I get it going on my 4gb PC?

Wednesday, November 19, 2008 7:40 PM by Adi R

re: Pushing the Limits of Windows: Virtual Memory

Hello and thank you for this excellent post.

I have a question and comment about virtualized server environments

To avoid swapping within a virtual W2K3 VM, I configure them with a chockfull (1,2,4GB) of RAM and let the hypervisor manage the pseudo physical allocation.

With your post, I am thinking I might still need to provide some paging within the VM.

Any comments about either statements?

Wednesday, November 19, 2008 9:41 PM by Louis

re: Pushing the Limits of Windows: Virtual Memory

http://blogs.technet.com/markrussinovich/archive/2008/11/17/3155406.aspx

Go

NOV DEC JAN

08

2007 2008 2010

About this capture

nonpaged pool leak (which I've traced to pool "TdxA", but I'm not sure where to go from there). My computer (Vista32) gets dangerously flaky once the non-paged pool gets to around 1GB. (It's also not happy when the total handle count gets above 45,000 or so, but that's less serious.)

Thursday, November 20, 2008 1:05 AM by [Miral](#)

re: Pushing the Limits of Windows: Virtual Memory

Great article! Maybe a stupid question but what is the disadvantage of setting a too big paging file?

Thursday, November 20, 2008 3:49 AM by PeterR

re: by Pavel Lebedinsky

Pavel, you said that:

3. Configuring a system with lots of RAM to run without pagefile may have either negative or positive perf impact depending on what the system is doing.

Could you please give me some examples of negative performance impact, if I have 4GB XP32 w/o pagefile?

And similar question: my colleagues was once set and experiment with 2GB pagefile on the 4GB RAMDrive on the 8GB XP64. I think that this doesn't make sense at all. What do you think?

Sorry for probably bad English =)

Thursday, November 20, 2008 12:19 PM by vadim

re: Pushing the Limits of Windows: Virtual Memory

I do think that setting the pagefile to a fixed size is better because of fragmentation (and therefor performance) reasons.

Thursday, November 20, 2008 12:32 PM by [Pieter](#)

re: Pushing the Limits of Windows: Virtual Memory

"I do think that setting the pagefile to a fixed size is better because of fragmentation (and therefor performance) reasons. "

Just defrag the pagefile and then you don't have to worry about it...

Thursday, November 20, 2008 12:42 PM by [Tim Bolton](#)

re: Pushing the Limits of Windows: Virtual Memory

Excellent post, can't wait for the next edition of Windows Internals to hit the shelves.

Thursday, November 20, 2008 4:41 PM by [Clay](#)

re: Pushing the Limits of Windows: Virtual Memory

Great article!

May I suggest a topic for a future article? I would love to see discussion about Windows memory management policies. E.g., when pages get swapped out and how that is decided; if and how those policies are customizable; etc.

I often see situations where recently-inactive pages are swapped out too soon, when there is plenty of available physical memory. This is especially problematic with Java (or any other garbage-collected environment for that matter) applications, where most of the pages have to be swapped in for each garbage collection.

Thursday, November 20, 2008 8:10 PM by Eran

re: Pushing the Limits of Windows: Virtual Memory

Thank you for the great article.

http://blogs.technet.com/markrussinovich/archive/2008/11/17/3155406.aspx

Go

NOV DEC JAN

08

2007 2008 2010

135 captures

8 Dec 2008 - 6 Nov 2020

▼ About this capture

Friday, November 21, 2008 2:23 AM by [Hofi](#)

re: Pushing the Limits of Windows: Virtual Memory

I am running Vista with 4GB of physical memory. I also have a USB drive with Readyboost enabled. Is Mark saying it is foolish to allow Windows to manage my virtual memory? Am I supposed to manually tweak my pagefile size for maximum performance?

Friday, November 21, 2008 2:33 AM by [mrogi](#)

re: Pushing the Limits of Windows: Virtual Memory

Is it [still] true that, when the low-memory resolution dialog appears, some random thread of some random process has already been commandeered for that purpose and therefore the system must axiomatically be regarded as unstable?

Friday, November 21, 2008 9:59 AM by [Frank Wilhoit](#)

re: Pushing the Limits of Windows: Virtual Memory

To Ben Voight:

- > Pagefiles are a hack to permit applications
- > written to process data in batches but
- > without regard to memory usage

I don't think of it that way. Demand-paged virtual memory OSs take advantage of the "80-20 rule" (or 90-10, or whatever): Most workloads spend most of their time accessing only a small portion of their address space. There's no particular need to keep things whose access is not in your performance path in RAM all the time.

Evidence for this can be found by increasing RAM while keeping the workload constant. You will reach a point of diminishing returns after which adding more RAM won't improve performance. And you'll find that this point occurs well before everything is in RAM.

(Of course, a lot of code from the exe's and dll's you're running will **never** be in RAM, unless you happen to execute **all** of it.)

This is also the argument against disabling the pagefile completely. Doing so will force the OS (any virtual memory OS) to keep all pagefile-backed v.m.... or rather, everything that's supposed to be pagefile-backed... in RAM all the time, no matter how long ago it was referenced. This of course cuts down on the RAM available for code, the file cache, etc. It's usually a net loss. Not absolutely always, but usually. (And then of course there are the apps that simply won't work at all without pagefile space.)

Saturday, November 22, 2008 4:32 PM by [Jamie Hanrahan](#)

re: Pushing the Limits of Windows: Virtual Memory

Brilliant read as always Mark. I adore the work you do, the blogs you write, the apps create and Events you speak at. I am waiting in anticipation for the 5th edition of windows internals to drop through my door. Sadly its 2 months away yet :'(

Cheers!

Monday, November 24, 2008 7:33 AM by [Wayne Robinson](#)

re: Pushing the Limits of Windows: Virtual Memory

<quote>(Of course, a lot of code from the exe's and dll's you're running will **never** be in RAM, unless you happen to execute **all** of it.)</quote>

http://blogs.technet.com/markrussinovich/archive/2008/11/17/3155406.aspx

Go

NOV DEC JAN

08

2007 2008 2010

About this capture

135 captures

8 Dec 2008 - 6 Nov 2020

is at best as good performance as reading memory-mapped or on an as-needed basis, and usually worse. Why? Because the file cache will satisfy your I/O from memory anyway for workloads that fit, and for workloads that don't fit the initial loading step reduces to a disk to disk (file to pagefile) copy of your data, after which the OS still has to read the data back in from the pagefile on demand.

Do you agree that the "bringing the system to a halt" is not due to running out of commit, but due to excessive paging in the process of allocating up to the commit limit after RAM is exhausted?

Monday, November 24, 2008 10:32 AM by Ben Voigt [C++ MVP]

re: Pushing the Limits of Windows: Virtual Memory

Even though my earlier comment never got posted, here's an update for whoever is reading/moderating...

I had mentioned that 32-bit testlimit on my 4GB XP64 system was reporting something just a bit shy of 2GB when it should've been reporting just shy of 4GB. I finally discovered that I was using an ancient version of testlimit that was not marked large-address aware.

I downloaded the latest version of testlimit (v5) and it is now reporting 4GB as expected.

Alan

Monday, November 24, 2008 3:13 PM by AlanH

re: Pushing the Limits of Windows: Virtual Memory

@Ben: I'm not in charge of setting pagefile limits, I was just one of the people who reviewed proposed changes for Vista.

I do however have some experience with debugging machines that can't make forward progress because of resource allocation failures, so I'll comment on this:

- > running out of commit generally causes a
- > fatal error to the application performing
- > the more allocations, allowing other
- > applications to continue normally and at full speed.

This might be true for a short period of time, but if the leaking application keeps allocating more memory (especially if it does this by committing one page at a time), you'll quickly get to a point where you can't start new processes anymore. So unless you happened to have task manager running, you won't even be able to kill the misbehaving app.

Cleanly exiting an existing process (for example, saving a modified document in Word) will also be impossible because doing this would require allocating more memory.

If you wait even longer, you'll see more problems, such as screen repaint issues, not being able to switch desktops, crashes or weird behavior from apps that do not handle errors correctly, etc. Code that doesn't allocate memory or call any external APIs might be able to continue running at full speed, but for all practical purposes the machine will be unusable.

Monday, November 24, 2008 8:28 PM by Pavel Lebedinsky

re: Pushing the Limits of Windows: Virtual Memory

By the way, there are actually 2 separate reasons why pagefiles are necessary.

The first reason is to allow dirty pages that are never (or very rarely) referenced to be moved to disk, freeing up more RAM for other purposes.

The other reason is to enable better use of *virtual* memory, given that physical memory is allocated on demand. Remember that when a process calls VirtualAlloc(MEM_COMMIT) there are no

http://blogs.technet.com/markrussinovich/archive/2008/11/17/3155406.aspx

Go

NOV DEC JAN

◀ 08 ▶

2007 2008 2010

135 captures

8 Dec 2008 - 6 Nov 2020

▼ About this capture

Now, even though committing memory does not allocate physical pages, it still guarantees to the application that reading from/writing to the committed pages will never fail (or deadlock). It might be slow if other physical pages have to be moved to disk in order to make room, but it will eventually succeed.

In order to make that guarantee the memory manager has to assume that every committed page in the system might eventually be written to. And that in turn means that there has to be enough space in the physical memory and all the pagefiles combined to hold all the resulting data. In other words, the total commit charge has to be less than the system commit limit. Once the limit is reached, the memory manager will refuse to commit any more memory, even if there is still plenty of unused (free+zeroed) physical pages, or plenty of unused space in the pagefile.

In a sense, pagefiles are like stormwater ponds. Most of the time they are (almost) empty, but they have to be large enough in case a big storm happens.

Monday, November 24, 2008 11:06 PM by Pavel Lebedinsky

re: Pushing the Limits of Windows: Virtual Memory

Good article - thanks.

And now a carping criticism for which you personally are not responsible: why is it that Task Manager seems to want to mislabel important system metrics?

Like, for example in XP, displaying the commit charge as 'PF Usage' (was 'Mem' in Windows 2000).

I suppose whoever's in charge of these things has decided that the average user can't understand 'commit charge', but is lying to them going to help? I've lost count of the number of times I've had to point out that just because Task Manager says you're using 1GB of pagefile, it doesn't mean you're using 1GB of pagefile (on your system that has no pagefiles configured, even).

dave

Tuesday, November 25, 2008 10:57 AM by dave

re: Pushing the Limits of Windows: Virtual Memory

Mark - could you talk about observing the size of applications using AWE?

Friday, November 28, 2008 12:25 PM by Greg

re: Pushing the Limits of Windows: Virtual Memory

In the last paragraph, the phrase "32-bit Windows has a maximum paging file size of 16TB" surely should have been 32GB, not 32TB, right?

Sunday, November 30, 2008 2:29 AM by Marc Brooks

re: Pushing the Limits of Windows: Virtual Memory

I concur with Mark regarding judging the size of the page file. Measure the maximum commit charge and set accordingly.

This would support the somewhat vague scientific maxim that judgement without observation or measurement is, in effect, meaningless (unless, of course, you're Einstein.)

Monday, December 01, 2008 5:17 AM by Rik Mayell

re: Pushing the Limits of Windows: Virtual Memory

Everyone downloaded the 5.0 Version of Testlimit?

It seems to have a bug....

I am running on a 32Bit WinXP SP3 Machine and when I use Testlimit with the -s or -m switch and so on it doesn't take the MB Parameter and allocates the maximum instantly....

http://blogs.technet.com/markrussinovich/archive/2008/11/17/3155406.aspx

Go

NOV DEC JAN

08

2007 2008 2010

135 captures

8 Dec 2008 - 6 Nov 2020

▼ About this capture

Regarding the comment/question above

32-bit Windows, running in PAE mode, does has a maximum paging file size of 16TB.

(or in practice the size of the underlying disk)

Most 32bit systems run in PAE mode by default these days.

Without PAE mode, the limit, for any single pagefile, is 4GB

Monday, December 01, 2008 12:39 PM by stephc_msft

re: Pushing the Limits of Windows: Virtual Memory

Pavel: "enable better use of *virtual* memory, given that physical memory is allocated on demand" and then "reading from/writing to the committed pages [...] might be slow if other physical pages have to be moved to disk in order to make room"... and/or if the page had been swapped out so it needs to be brought back from disk.

The crux of the problem with virtual memory is in there. What I see happen often is a system that is working fine... but where some components are chuggish to respond. I open Windows Explorer and it takes 15 seconds to show up. Or I click on the start button and it takes 5 seconds to respond. Or I interact with any application and it takes "too long" to do what I ask of it.

It's like part of the application's memory is swapped out, and now Windows has to bring in the needed pages on demand, as it sees them being accessed, and oftentimes this can take seconds to complete. Running without a page file or with a small one minimizes the occurrences of this sort of issue. At least, it certainly did on Windows 2000, which was the last time I tried it, I must confess. And this sort of issue is very common in my experience, even today on Windows Vista. I think I'm leaning towards Ben on this one, with the caveat that having Pavel's "stormwater ponds" sounds like a good thing. Then again, I'm on videogames, where responsiveness is paramount, so I might be a trifle biased. Paradoxically, most videogames tend to constantly exercise the same memory in a tight loop, so this sort of issue doesn't crop up so often, there.

In any case, I've always suspected that swapping out application memory to make space for a bigger file cache is not such a good idea in many scenarios, although it definitely improves performance for certain tasks (mostly background and batched tasks, IMHO). The question is: is the current strategy used by Windows the best strategy to provide the best user-interaction responsiveness on consumer machines? Somehow, I don't think so. The problem is certainly very complex, so maybe I'm just deluding myself in thinking that there must be a better way.

Thursday, December 04, 2008 3:20 PM by JCAB

Historical Reasons for 3x RAM for Paging

I seem to recall the reason for the 3x paging file recommendation was made by Intel long back in the days when the 80386 was first introduced.

Thursday, December 04, 2008 9:32 PM by Todd Bandrowsky

re: Pushing the Limits of Windows: Virtual Memory

Typo I believe:

The figure isn't drawn to scale, because even 8TB (should be 8GB instead of TB) , much less 128GB, would be a small sliver. Suffice it to say that like our universe, there's a lot of emptiness in the address space of a 64-bit process.

Great article!

Friday, December 05, 2008 5:09 AM by John Westher

re: Pushing the Limits of Windows: Virtual Memory

http://blogs.technet.com/markrussinovich/archive/2008/11/17/3155406.aspx

Go

NOV DEC JAN

08

2007 2008 2010

135 captures

8 Dec 2008 - 6 Nov 2020

About this capture

Esentutl.exe is a database repair tool that isn't specific to Active Directory. Its file description says "Server Database Storage Utilities".

These are fairly common things one should know about Windows, IMO. As a result of finding those inaccuracies I have to fully test everything written here before I can trust any of it. Your conclusions are probably right, but you can see how including incorrect information can tarnish an otherwise perfect article, right?

Friday, December 05, 2008 8:53 AM by Kerry C

re: Pushing the Limits of Windows: Virtual Memory

Intel's 32 bit processors since the 80386 all address 48 bits of logical memory (2^{48}) via a 16 bit segment register and 32bit offset pointer.

IMHO, the modern generation of operating systems does not exploit this because 32 bits of addressable memory is "more than anyone would ever need" and dealing with "NEAR" and "FAR" pointers was a major fiasco in the early days of Windows coding.

Thought I would point it out since you failed to mention this.

Friday, December 05, 2008 2:24 PM by [John Cairns](#)

© 2008 Microsoft Corporation. All rights reserved. [Terms of Use](#) | [Trademarks](#) | [Privacy Statement](#)