

CSS SQL Server Engineers

This is the official team Web Log for Microsoft Customer Service and Support (CSS) SQL Support. Posts are provided by the CSS SQL Escalation Services team.

Fun with Locked Pages, AWE, Task Manager, and the Working Set...

I realize that the topic of "locked pages" and AWE can be confusing. I don't blame anyone for being confused on this. I also realize we have blogged and talked about this topic many times perhaps "beating it to death". And I certainly know this is not really fun to anyone (but it made for a catchy title). But I still get questions both from customers and internally within Microsoft about these topics for both 32bit and 64bit SQL Server systems. So I thought a blog post that summarizes and clarifies a few points might be valuable. Maybe this will be the one resource that "turns on the light" for you. Many people respond well to a "FAQ" so I'll create this topic in that format. I recommend you read through each FAQ one at a time in order, because some of the answers need to be read first to understand the ones below it:

1. What is the difference between AWE on 32bit systems and "Locked Pages" on 64bit systems?

I think it is all about using the right terms. SQL Server introduced a "feature" called AWE before we even shipped a 64bit version of SQL Server. The concept was to extend the ability of SQL Server to address more memory than the limits of the virtual address space (VAS) on 32bit systems (which is 2Gb-3Gb). But the SQL team couldn't do this alone. So the Windows team built an API to support this capability called [Address Windowing Extensions](#) (AWE). If you want to go a step further and find out what is the difference between Physical Address Extensions (PAE) and AWE, read this [resource](#).

One of the interesting effects for any application that uses the AWE API is that any memory allocated with these APIs is not part of the process working set. Therefore, Windows considered this memory as "locked" (i.e. cannot be paged to disk). Therefore, the user who launches an app that uses the AWE API must have the "Locked Pages in Memory" privilege (for example the service account for SQL Server) set.

So now the SQL team introduces a x64 edition of SQL Server with SQL Server 2005. Because a process running on x64 doesn't have the same VAS limits as 32bit (the numbers are crazy here. Theoretically, a 64bit process has a 16 Exabyte address limit which Windows doesn't even support yet. In fact, Windows limits VAS to 8TB but physical memory is limited to 2TB....today), there is no need to use any special APIs to address memory bigger than the VAS. In other words, SQL Server should be free to go back and just use plain ol' VirtualAlloc() to allocate memory. Well...the developers of the product discovered that if they still use the AWE APIs to allocate memory even though it is not really needed, two things would happen:

- A small performance gain occurs within the kernel. For more details, read this blog post from Slava Oks: <http://blogs.msdn.com/slavao/archive/2005/04/29/413425.aspx>
- Just as with 32bit systems, any memory allocated using the AWE API is not part of the working set and therefore cannot be paged to disk. Therefore it is considered "locked".

Thus was born the concept most refer to as "locked pages" for 64bit SQL Server editions.

Now remember the requirement to use the AWE APIs for any Windows application? The user account running the process must have the "Locked Pages in Memory" privilege set. So in the 64bit SQL Server engine, if this privilege is set (and a SKU check. See a different FAQ below for more discussion on this), we internally use the AWE APIs to allocate memory. By using the AWE APIs, we have in effect enabled a "locked pages" feature for the SQL Server engine.

So in summary the AWE APIs for 32bit and 64bit SQL Server systems are used for different purposes. In 32bit it is really to extend memory access beyond 4Gb or to enable the AWE feature. For 64bit systems, it is to possibly gain performance and to "lock pages" for the buffer pool.

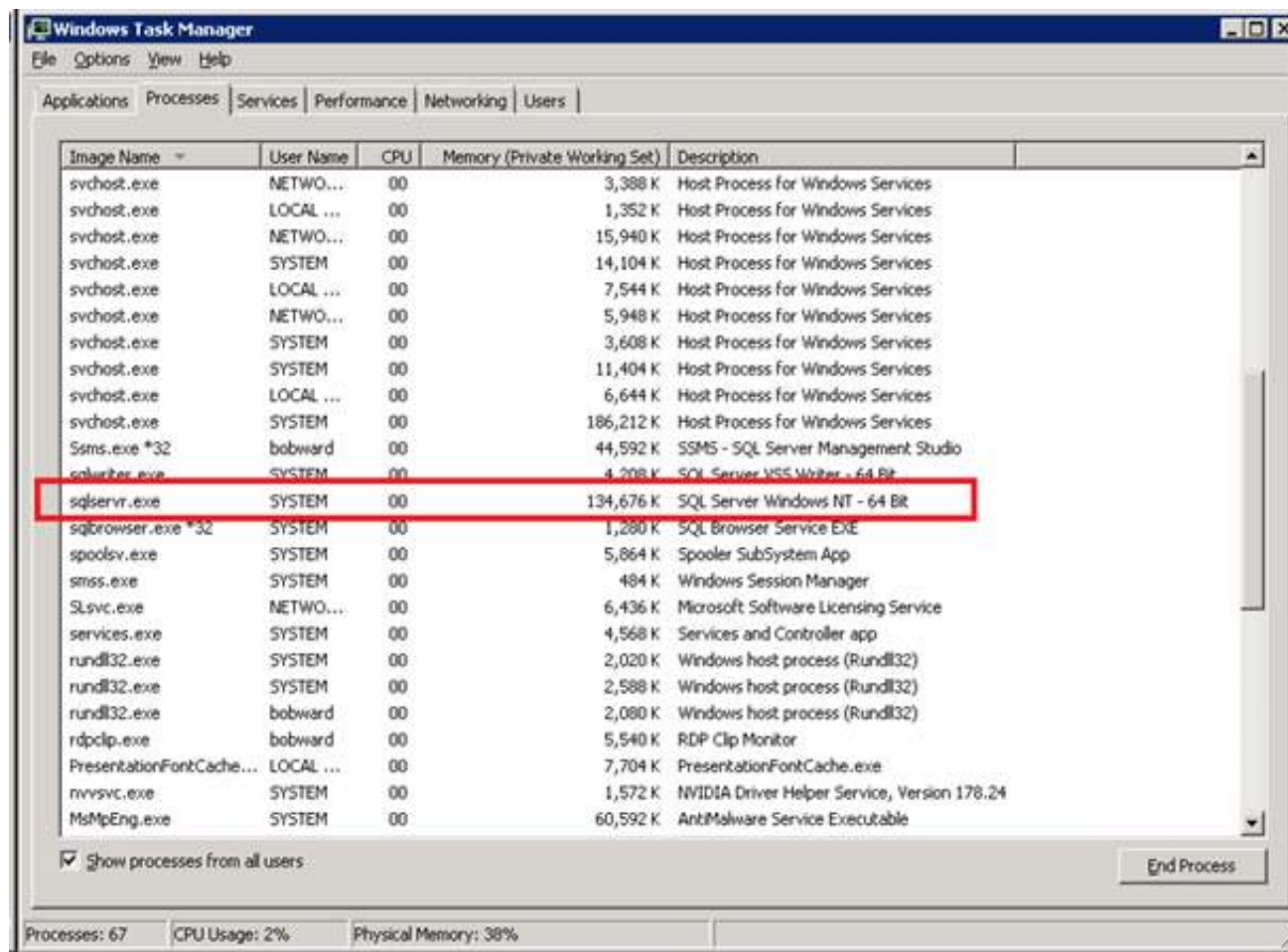
2. Do I need to use the "awe enabled" sp_configure option on 64bit systems for SQL Server to "lock pages"?

No. In fact, the code for SQL Server for 64bit systems ignores this sp_configure option. It is a "no-op" for 64bit SQL Server systems. You may ask why is this the case if I just told you that AWE APIs are used in 64bit SQL Server systems to "lock pages"?

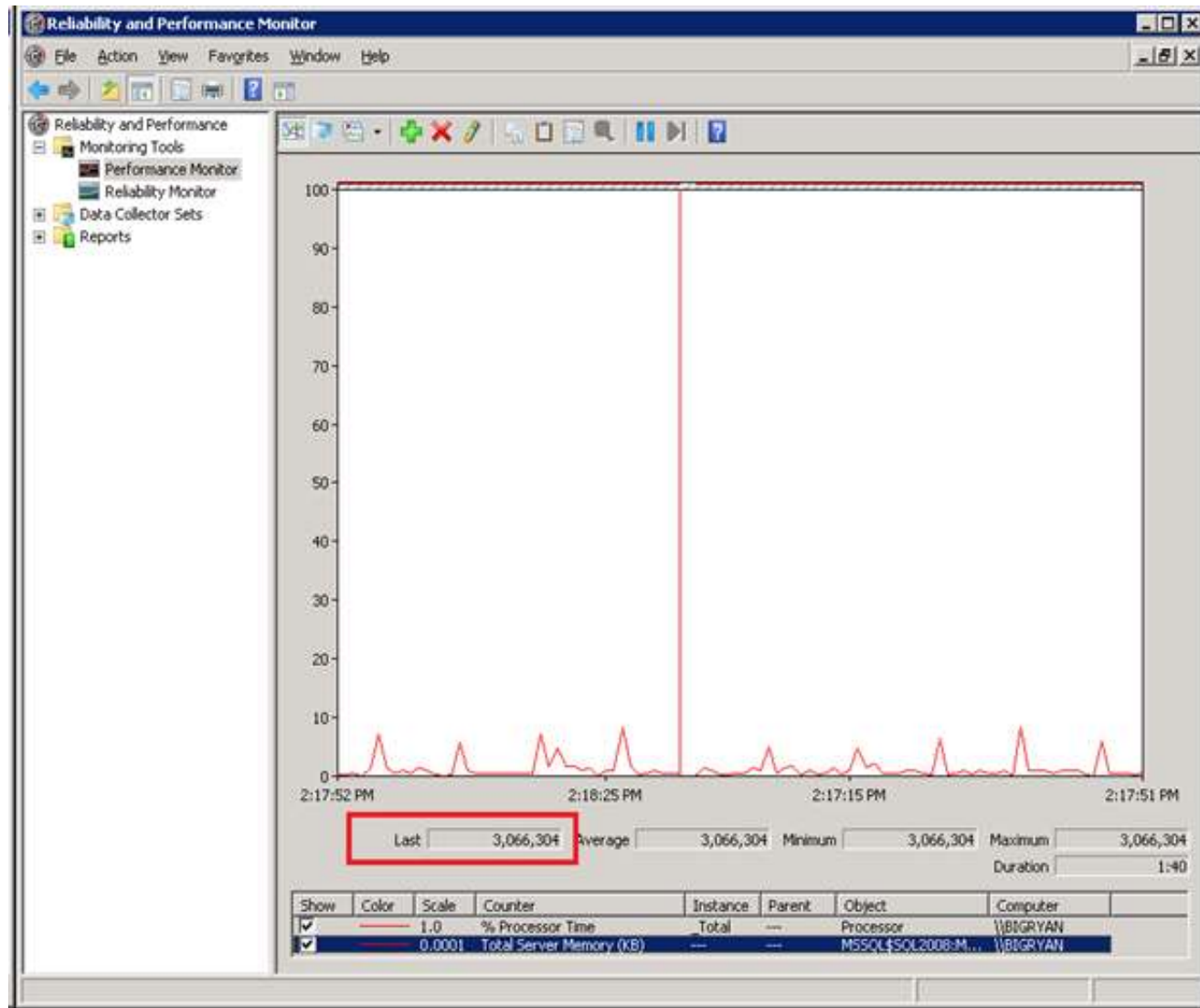
The answer is based on the purpose for that sp_configure option. The purpose of this sp_configure option on 32bit systems is for the user to "enable" the "AWE" feature, which is I explained above is to extend the ability to reference memory > 4Gb. Now as I mentioned already in order to use the AWE APIs you must have the "Locked Pages in Memory" Privilege. So, when you try to use sp_configure to set 'awe enabled' on a 32bit we actually will fail this command if "Locked Pages in Memory" is not set.

3. Why is Task Manager not showing all of the memory allocated for SQL Server?

Imagine you walk up to a 64bit SQL Server installation and the customer tells you the computer has 8Gb of physical memory and SQL Server perfmon counters such as "Total Server Memory" show SQL Server is using around 3Gb of that. But you open up Task Manager on this Windows Server 2008 machine and look at the columns for SQLSERVER.EXE in Task Manager and see something like this:

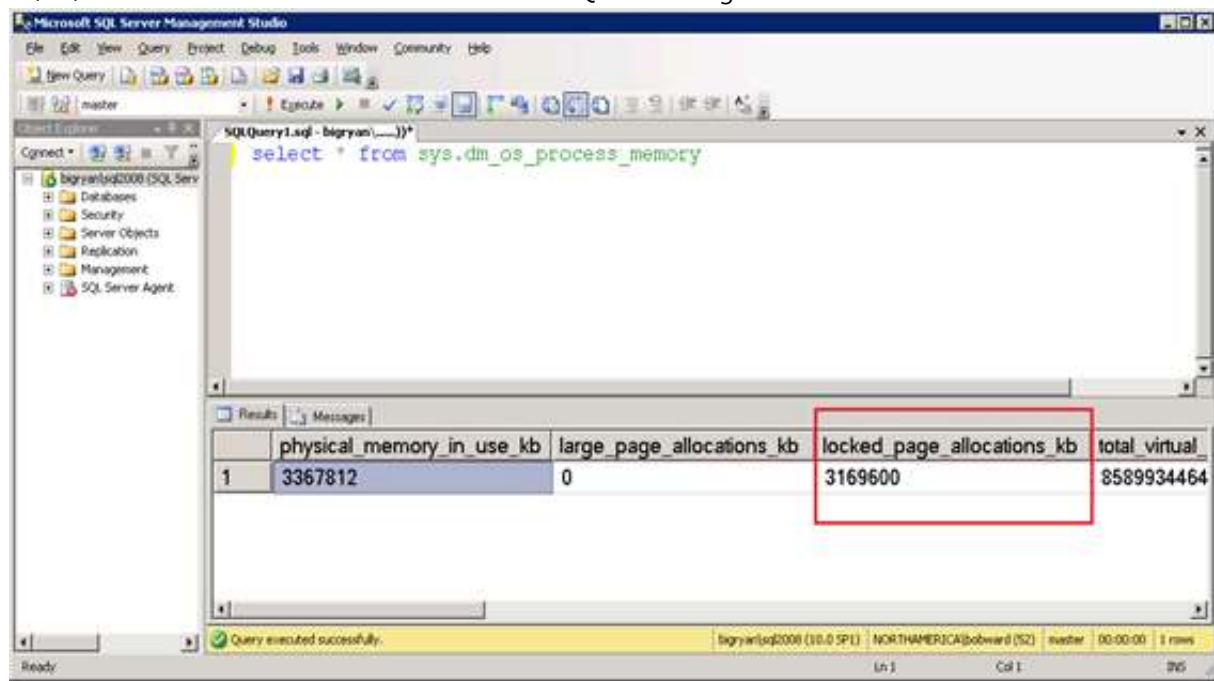


You can see in this example, that the column for "Memory" for Task Manager shows only ~135Mb. But on this computer, if I look at Perfmon and show Total Server Memory, I see this:



Why the difference? Well, the most likely reason is that this SQL Server 64bit instance is using "locked pages" as I've described in an earlier question. Notice the name on the Task Manager column is called **Memory (Private Working Set)**. Remember we also said that if SQL Server 64bit instances use "locked pages" this memory would not be part of the working set (because remember AWE APIs are used on 64bit to "lock" pages and that memory is not part of the working set). So since the locked pages are not part of the working set, they won't appear in this column in Task Manager. On the Task Manager for Windows Server 2003, this column is called "Mem Usage" but it also reflects the working set of the process.

How can we prove this "difference" in memory is due to locked pages? Well, there are several ways to do this, but one way to easily see this in SQL Server 2008 is to query the sys.dm_os_process_memory DMV. On my computer where I saw this behavior I queried this DMV and saw these results:



	physical_memory_in_use_kb	large_page_allocations_kb	locked_page_allocations_kb	total_virtual_memory_kb
1	3367812	0	3169600	8589934464

You can see from this DMV that the column `locked_page_allocations_kb` is close to the Total Server Memory perfmon counter and shows that this memory is actually "locked".

4. Now I know that SQL Server on x64 can use "Locked Pages", what exactly is locked?

The simple, direct answer to this question is any memory allocated through the Buffer Pool Manager for SQL Server. But what is the "Buffer Pool Manager"? Does this mean only "database" pages are locked? Starting in SQL Server 2005, all memory allocations (in other words all access to the Windows API for memory) go through the SQLOS component of the engine. Any code in the SQL engine that need to allocate memory $\leq 8\text{Kb}$ use something called the "Single Page Allocator" (SPA) in SQLOS. It just so happens that SQLOS redirects any SPA requests to the Buffer Pool Manager. This is the same buffer pool code that has been used since SQL 7.0 to allocate memory. So any memory needed in the engine that wants "single pages" ultimately goes through the Buffer Pool. And... the Buffer Pool is the code that has the logic to lock pages via the AWE APIs.

So...any code using the Single Page Allocator (SPA) which uses the Buffer Pool manager, will have its memory "locked" if the Buffer Pool Manager is using locked pages. What code uses the Single Page Allocator? Aside from the what you may already guess which are database pages, you can find out what "type" of memory uses the SPA by querying the `sys.dm_os_memory_clerks` DMV. Look for any row where the `single_pages_kb` column is > 0 . If you run this query on your server you are likely to see clerk types of `CACHESTORE_OBJCP` and `CACHESTORE_SQLCP`. This procedure cache memory. I won't list out all of them here but you can see several "types" of memory use SPA which means they use Buffer Pool which means this memory can be locked.

5. Does the Standard Edition of SQL Server 32bit support AWE? What about "Locked Pages" for 64bit?

I've seen some people ask me and others at Microsoft whether the Standard Edition of SQL Server for 32bit supports the "AWE" feature. I think the confusion is related to the fact that Standard Edition for SQL Server for 64bit until recently did not support locked pages.

So let's dispel this myth here:

- The Standard Edition of SQL Server for 2005 and 2008 32bit DOES support the AWE feature. One source to confirm this is at <http://msdn.microsoft.com/en-us/library/cc645993.aspx>. This lists "Dynamic AWE" as a feature support for both Standard

and Enterprise Editions. As I wrote this blog I wanted to confirm this by going to the source..literally. I checked our code and our logic to "enable AWE" is based on Enterprise OR Standard Editions.

- Up until recently the Standard Edition of SQL Server 2005 and 2008 64bit did NOT support the "locked pages" feature as I've described it earlier in this blog post. Based on customer feedback we changed this in recent cumulative updates for both 2005 and 2008. Enabling this for Standard requires a trace flag and you can read more about this at <http://blogs.msdn.com/psssql/archive/2009/05/19/an-update-for-standard-sku-support-for-locked-pages.aspx>

6. How do I know if AWE is enabled on a 32bit SQL Server? How do I know if "Locked Pages" is working on a 64bit SQL Server?

Read this blog post first which gives details on the algorithm in the engine for enabling AWE or for enabling "Locked Pages" <http://blogs.msdn.com/psssql/archive/2007/10/18/do-i-have-to-assign-the-lock-privilege-for-local-system.aspx>.

For 32bit systems, one tip. If you think you set "awe enabled" to 1 and are not sure why you are not seeing **Address Windowing Extensions is enabled** in the ERRORLOG, it is likely you either didn't run RECONFIGURE after changing "awe enabled" to 1 OR the "Lock Pages in Memory" privilege is not set. If you try to reconfigure after changing "awe enabled" to 1 and the "Lock Pages in Memory" privilege is not set, the RECONFIGURE command will fail with:

Msg 5845, Level 16, State 1, Line 1

Address Windowing Extensions (AWE) requires the 'lock pages in memory' privilege which is not currently present in the access token of the process.

I hope this information has helped you understand the concepts of AWE and "Locked Pages" with respect to SQL Server. I know this can be a confusing topic and I don't blame anyone for asking question to understand this. If I see comments to this post or run into any common questions that make sense I'll add this to the FAQ on this blog.

Bob Ward
Microsoft

Published Friday, September 11, 2009 8:40 PM by [psssql](#)

Filed under: [Memory](#)

Comments

No Comments

Anonymous comments are disabled
