Troubleshooting Pool Problems with Poolmon

Account Brian Desmond's Blog1
Category Blog

Having discussed some of the symptoms of resources shortages associated with paged and nonpaged pool as well as a bit about what these special types of memory are in this post, we can now talk a bit about troubleshooting these shortages with a tool called Poolmon. Poolmon will show us the top consumers of nonpaged as well as paged pool and sort it however you want. Exactly how you get Poolmon as a layperson is a bit of an interesting situation.

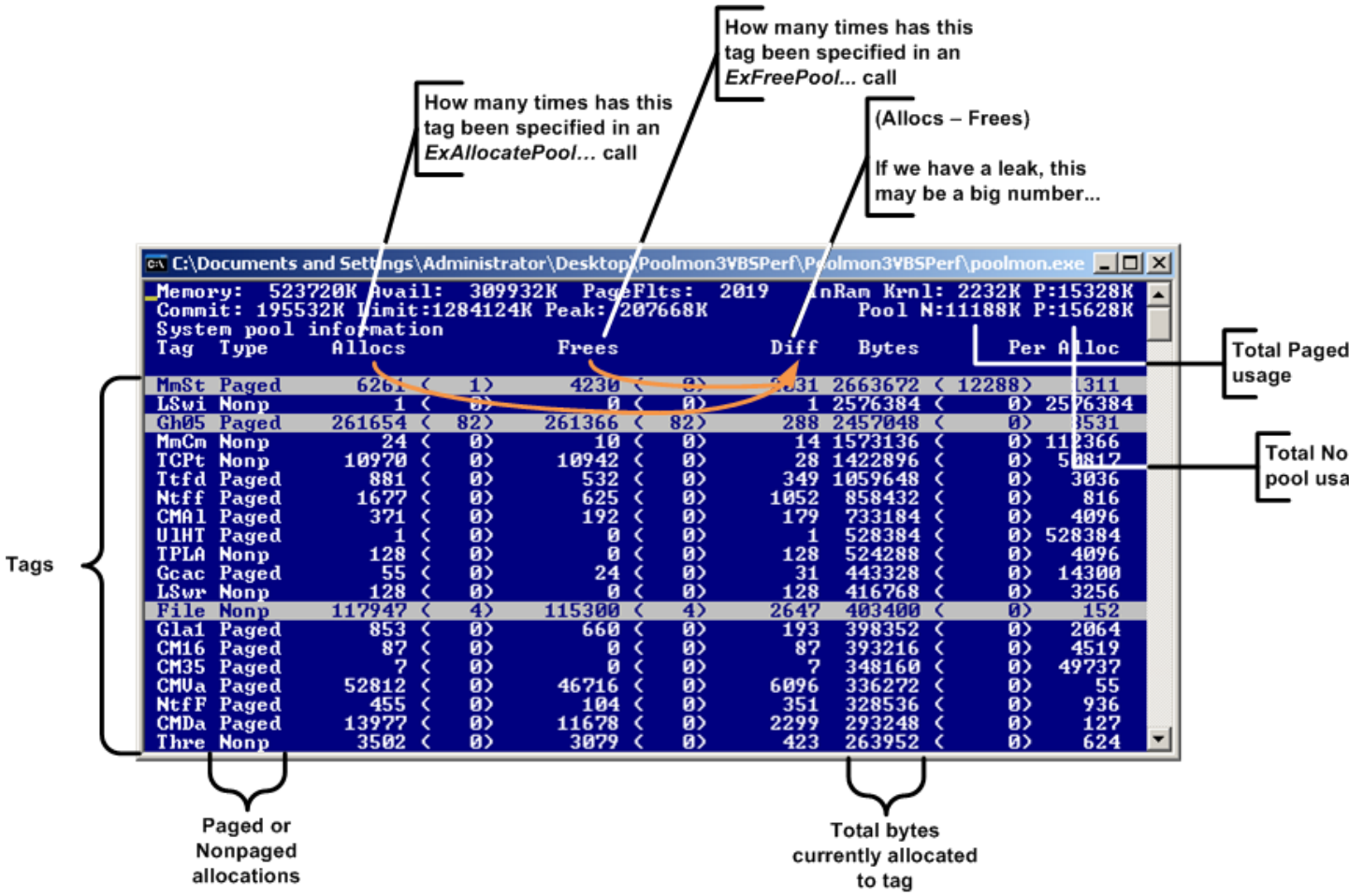
The latest Poolmon.exe binary is available from the Windows Driver Kit (WDK aka DDK). You can download the WDK free from either MSDN if you have a subscription, or from Microsoft Connect. The binaries are located in Tools\Other\i386 or Tools\Other\amd64 in the install directory. There's also an older version in some versions of the Windows Support Tools download. Older versions are typically fine, so if you're in a hurry, the Support Tools download will be the quickest way to get the tool.

Note: The current version of the WDK (version 6001.18002) ships with the wrong version of a DLL that Poolmon needs for certain operations. The DLL it depends on is msdis160.dll however the WDK ships with msdis150.dll. I was able to locate a copy of msdis160.dll in the Visual Studio 2008 install. I'm also told it's available in the Windows 7 beta WDK.

A more productive way to work with Poolmon when troubleshooting pool leaks is to use a package which is available from PSS. The archive is called Poolmon3VBSPerf.zip and presumably you should be able to get it from your TAM if you have one, or else next time you talk to an engineer they'll have it. I wasn't able to find the package on the Premier Support portal or elsewhere. The zip includes a number of scripts which wrap Poolmon for a bunch of useful logging and installation as a service. I keep this archive out on a share at customers I support so it's readily available.

Windows XP and earlier (e.g. Windows 2000) requires a special configuration flag be set in the registry in order to enable pool tagging. Without setting this option, Poolmon is useless. Pool tags are an essential part of the troubleshooting process, and this discussion assumes that drivers are using them. In a nutshell, what happens is a driver calls the ExAllocatePoolWithTag API and specifies a tag that's usually three or four characters in length.

When you launch Poolmon, you'll see output something like this, annotations mine:



There are a couple ways to enable pool tagging on a machine that makes it optional. The gflags utility which ships with the Debugging Tools for Windows is the best approach, however you can also set a registry setting directly. To enable pool tagging with gflags, run `gflags -r +ptg`. Alternatively, you can set `HKLM\System\CurrentControlSet\Control\SessionManager\GlobalFlag` to `0x00000400`. Either way, you'll need to reboot after you set this flag in order for the change to take effect.

Note: If you have the Poolmon package discussed earlier available from PSS, there is an EnablePoolTag batch script in the archive which you can use to make this change. You'll still need to reboot afterwards.

Now that we have Poolmon up and running, we can go to work. If you've got a SRV 2019 or SRV 2020 event being logged as discussed previously, you're in a situation where you'll want to use Poolmon to collect some data. To setup the screen for easy reading, after you launch Poolmon press B to sort by Bytes and then press P to cycle through showing only Paged, Nonpaged, or Both. If you're getting 2019 events, Nonpaged is what you want and likewise, 2020 events indicate you want to see Paged pool allocations. Take note of the data shown – you'll need the top tags in particular to search for a culprit.

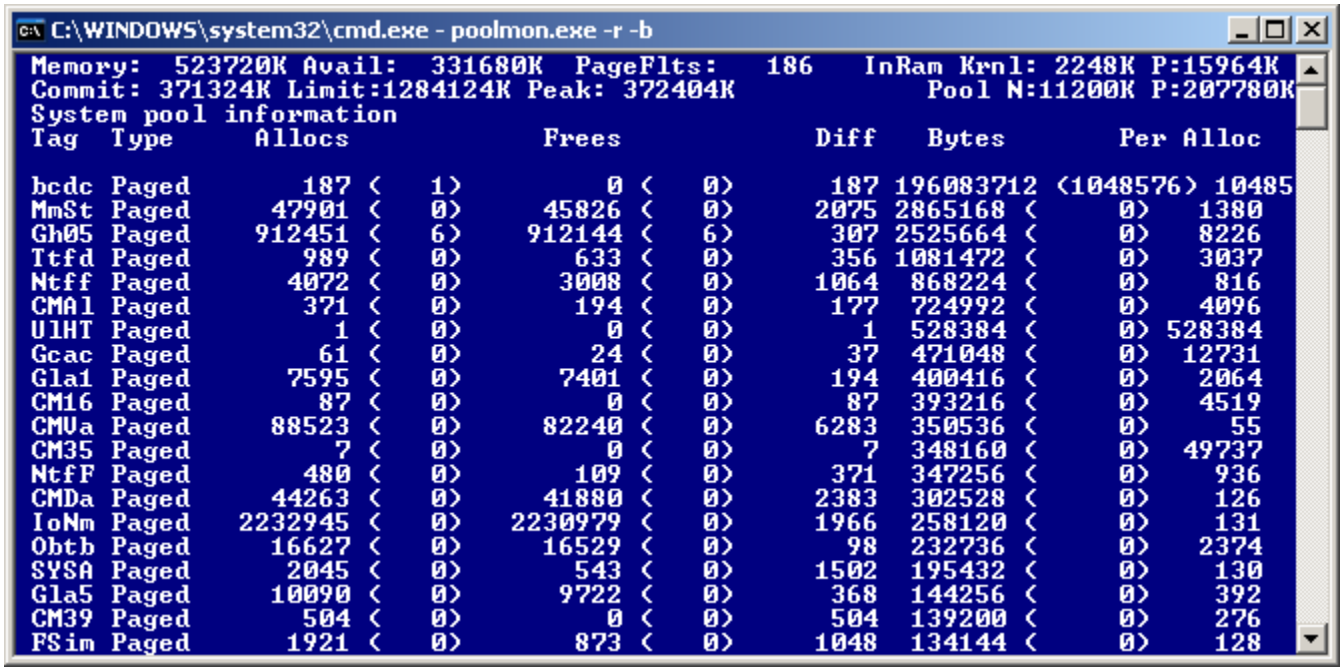
The difficulty of doing this by hand is that you need to be present when the problem is occurring. These types of resource shortages can take a substantial amount of time to develop sometimes. Collecting the data after the fact or right after a reboot is unlikely to yield any interesting results. You may however be

able to find the culprit during the time leading up to the actual problem by taking periodic snapshots of the pool data and comparing them.

The Poolmon package available from PSS makes this entire situation quite easy. There is a script in the archive called _LogPool-as-a-service.cmd. It will install a Windows service which captures a snapshot to a text file every 60 seconds as well as quite a few other data points. If you don't want to install the tool as a service, you can just launch LogPool.cmd and the logging will be generated as long as your session is logged in.

Note: A workaround you could potentially employ is to schedule the Poolmon utility to run every 60 seconds and log to a text file. The command you'll want is `Poolmon.exe -b -n C:\PoolMon\PoolLog.txt` (or similar).

With the data gathering step behind us, let's take the scenario where Poolmon looks something like this:



The top tag, bcdc is accounting for about 196MB of paged pool utilization. This isn't normal by any means and is definitely something to look into. So, we have the tag, *bcdc*, but, how do we correlate that to a device driver that's causing the issue? The first place to check is the pooltag.txt file which Microsoft provides. This text file has a listing of Windows tags and their purpose. You can find this file in the triage subfolder of the Debugging Tools for Windows installation, and under tools\other\i386 in the WDK.

If you search pooltag.txt for bcdc, you'll find that it's not there. This typically means the pool tag is owned by a third party (aka not Microsoft) driver. In order to find this driver we need to search through the strings of all the drivers loaded on the system. The tool of choice to do this is findstr.exe which is included with Windows. To search for the string do this:

1. Open a command prompt and switch to `%WinDir%\System32\Drivers`.
2. Execute `findstr /m /l bcdc *.sys`
3. If you don't get any results, expand your search to include the entire Windows folder, Program Files, and other extensions (such as *.dll and *.drv).

In this case, findstr returned one result, a driver called bcdcrash.sys. This happens to be a driver I wrote specifically to leak various resources for demonstration.

So, what if we can't find a suspect driver after doing all this, or a tag like MmSt or Thre is consuming an excessive amount of pool? I chunked these scenarios in to a few separate posts:

- MmSt Post Link
- Thre Post Link
- Ddk Post Link
- Generic handle leak link