

Re: Memory fragmentation issue in kernel mode

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.development.device.drivers/2007-02/msg00727.html>

- *From:* "Kevin" <kevin11@xxxxxxxxxxxx>
 - *Date:* Tue, 27 Feb 2007 16:23:35 -0800
-

Use Task Manager and look at the Handles column for your test process. If what Anton describes is happening, it will be plainly obvious. And, you may as well display USER and GDI objects, Threads, etc., looking for numbers that grow without limit.

Also, on the Performance tab, watch the Kernel Memory section. This should be growing without limit, according to the symptoms you are seeing. If this is happening, but the Handles, USER or GDI numbers do not, then a driver is not releasing memory it allocated.

"Anton Bassov" <AntonBassov@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in message <news:A80F332A-923D-40BA-B4EB-B8303DB4B871@xxxxxxxxxxxxxxxxxxxx>

3. Is it application's or system's responsibility to prevent memory fragmentation? i.e. Is it an application bug or OS problem?

Well, a user level application can do very little with this kind of fragmentation, which is actually fragmentation of the system virtual address space. A user level application doesn't allocate and even access the system virtual address space.

I am afraid you grossly underestimate app's "potential" when it comes to causing non-paged memory leaks – it can do quite a lot, despite being unable to allocate anything directly.

For example, quite often you can see the code like the following:

do things.....

CreateThread(...)

Re: Memory fragmentation issue in kernel mode

do things.....

I hope that, as a kernel-level programmer, you immediately see the problem with the above code – it ignores the fact that CreateThread() returns a handle to the newly created thread, and, hence, never calls CloseHandle() on it. As a result, when newly created thread exits, it will enter the terminated state, but once a handle to it is still open, i.e. reference count on ETHREAD that describes it is non-zero, its ETHREAD (which is allocated from non-paged pool and happens to be not that small), instead of getting returned to non-paged pool, will remain on the thread list until the app exits. You may not notice it if the program does not create that many threads and/or exits shortly, but if this buggy app is intended to run for quite a while and creates many threads, the effect that it has on non-paged pool will become noticeable....

Actually, I don't exclude the possibility that the OP's app is written or more or less similar fashion....

Anton Bassov

"Gianluca Varenni" wrote:

"killme" <killme@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in message news:553CE535-CD8D-4D53-8F5B-84BF514BF713@xxxxxxxxxxxxxxxxxxxx

Hi All,

I am having one peculiar problem in Windows XP SP2 (with all patches applied). I have my driver that uses direct IO to perform data transfer (the data transfer will be in 256 KB buffers). I have one third party test application which will be accessing my device to test the driver. When I run this application for around 1 million passes, then MmGetSystemAddressForMdlSafe() fails in my driver.

Re: Memory fragmentation issue in kernel mode

When I ran the ioctl sample provided in DDK with different buffer sizes, it succeeded for 512 bytes and 4KB but failed for 64KB and afterwards indicating some memory fragmentation. One more thing is that I never allocate any memory in the driver during the process other than using MmGetSystemAddressxxx().

I have following questions:

1. In such scenario, what should a driver writer do? He can simply fail the request and continue or try to access the MDL in a scatter/gather manner and continue based upon the importance of the device?

Well, if your driver can use this workaround, I would try that.

2. Will the system ever recover after such memory fragmentation?
3. Is it application's or system's responsibility to prevent memory fragmentation? i.e. Is it an application bug or OS problem?

Well, a user level application can do very little with this kind of fragmentation, which is actually fragmentation of the system virtual address space. A user level application doesn't allocate and even access the system virtual address space. A driver does (by allocating memory or mapping memory from the process address space to system address space with MmGetSystemAddress). But even in a driver, this is one the usual way to dispatch IRPs. So strictly speaking, i think it's more of an OS problem.

...having said that, it seems quite weird that your machine's memory gets so fragmented that the OS cannot map 64k of memory into system address space. This operation is done literally thousands of times by the OS for every communication between kernel mode and user mode. Are your direct io requests synchronous or asynchronous? If asynch, how many concurrent IO requests

Re: Memory fragmentation issue in kernel mode

(and related mappings of MDLs to system address space) do you think you might have?

Have a nice day
GV