How To Deadlock Yourself (Don't Do This)

ntdebug 19 Jul 2012 10:52 AM **1**

Some APIs should come with a warning in big red letters saying "**DANGER!**", or perhaps more subtly "**PROCEED WITH CAUTION**". One such API is **ExSetResourceOwnerPointer**. Although the documentation contains an explanation of what limited activity you can do with the resource after making this call, its warning is not very strongly worded.

You may see evidence of a call to ExSetResourceOwnerPointer in a debug. A lock in !locks will have an unusual owner field, such as the one shown below:

```
2: kd> !locks
**** DUMP OF ALL RESOURCE OBJECTS ****
KD: Scanning for held locks...
Resource @ 0xfffffa8011efede8 Exclusively owned
Contention Count = 20
NumberOfSharedWaiters = 16
Threads: fffff88007fab7f3-02<*> *** Unknown owner, possibly FileSystem
fffffa80169538a0-01 fffffa801ea69b60-01 fffffa8017dfd430-01 fffffa800cd76b60-01
fffffa801512a410-01 fffffa801279b340-01 fffffa8016d079a0-01 fffffa8015452aa0-01
fffffa801607bb60-01 fffffa8012f79b60-01 fffffa8013b4e040-01 fffffa801b03e300-01
fffffa800cd77040-01 fffffa8013a8e040-01 fffffa800cd76040-01 fffffa80172d7490-01
```

The error "*** Unknown owner, possibly FileSystem" is an indicator that the owner field of this eresource has likely been modified by ExSetResourceOwnerPointer. Fortunately for us debuggers, programmers often point the owner field to a location on the thread stack. You can pass an address to the !thread command and it will interpret the address as a stack value.

```
2: kd> !thread fffff88007fab7f3 e
fffff88007fab7f3 is not a thread object, interpreting as stack value...
THREAD fffffa80169538a0 Cid 0004.0638 Teb: 0000000000000000 Win32Thread: 0000000000000000 WAIT: (WrResource) KernelMode Non-Alertable
fffffa80139ea3e0 Semaphore Limit 0x7fffffff
IRP List:
fffffa8016fd1010: (0006,0310) Flags: 00000884 Mdl: 00000000
Not impersonating
DeviceMap fffff8a000008aa0
Owning Process fffffa800cd6a5f0 Image: System
Attached Process N/A Image: N/A
Wait Start TickCount 27606952 Ticks: 141 (0:00:00:02.199)
Context Switch Count 90787
UserTime 00:00:00.000
KernelTime 00:00:02.496
Win32 Start Address nt!ExpWorkerThread (0xfffff80002293a50)
Stack Init fffff88007fabdb0 Current fffff88007fab3a0
Base fffff88007fac000 Limit fffff88007fa6000 Call 0
Priority 14 BasePriority 13 UnusualBoost 1 ForegroundBoost 0 IoPriority 2 PagePriority 5
Child-SP RetAddr Call Site
fffff880`07fab3e0 fffff800`0228da52 nt!KiSwapContext+0x7a
fffff880`07fab520 fffff800`0228fbaf nt!KiCommitThreadWait+0x1d2
fffff880`07fab5b0 fffff800`0224ec9e nt!KeWaitForSingleObject+0x19f
fffff880`07fab650 fffff800`022ad98c nt!ExpWaitForResource+0xae
fffff880`07fab6c0 fffff880`0140fc10 nt!ExAcquireSharedStarveExclusive+0x1bc
fffff880`07fab720 fffff880`0140f8e2 sis!SipDereferenceCSFile+0x40
fffff880`07fab750 fffff880`0140f608 sis!SipDereferencePerLink+0x62
fffff880`07fab780 fffff880`014102e7 sis!SipDereferenceScb+0x184
fffff880`07fab7c0 fffff800`025796e6 sis!SiFilterContextFreedCallback+0xaf
fffff880`07fab7f0 fffff880`016b9bcc nt!FsRtlTeardownPerStreamContexts+0xe2
fffff880`07fab840 fffff880`016b98d5 Ntfs!NtfsDeleteScb+0x108
fffff880`07fab880 fffff880`0162ccb4 Ntfs!NtfsRemoveScb+0x61
fffff880`07fab8c0 fffff880`016b72dc Ntfs!NtfsPrepareFcbForRemoval+0x50
fffff880`07fab8f0 fffff880`01635882 Ntfs!NtfsTeardownStructures+0xdc
fffff880`07fab970 fffff880`016ce813 Ntfs!NtfsDecrementCloseCounts+0xa2
fffff880`07fab9b0 fffff880`016a838f Ntfs!NtfsCommonClose+0x353
fffff880`07faba80 fffff880`016cd7ef Ntfs!NtfsFspClose+0x15f
fffff880`07fabb50 fffff880`01635c0d Ntfs!NtfsCommonCreate+0x193f
fffff880`07fabd30 fffff800`0227e787 Ntfs!NtfsCommonCreateCallout+0x1d
fffff880`07fabd60 fffff800`0227e741 nt!KySwitchKernelStackCallout+0x27 (TrapFrame @ fffff880`07fabc20)
```

```
fffff880`085fffe0 fffff800`0229620a nt!KiSwitchKernelStackContinue
fffff880`08600000 fffff880`01635b2f nt!KeExpandKernelStackAndCalloutEx+0x29a
fffff880`086000e0 fffff880`016d29c0 Ntfs!NtfsCommonCreateOnNewStack+0x4f
fffff880`08600140 fffff880`013330b6 Ntfs!NtfsFsdCreate+0x1b0
fffff880`086002f0 fffff800`0258d717 fltmgr!FltpCreate+0xa6
fffff880`086003a0 fffff800`0258379f nt!IopParseDevice+0x5a7
fffff880`08600530 fffff800`02588b16 nt!ObpLookupObjectName+0x32f
fffff880`08600630 fffff800`0258f827 nt!ObOpenObjectByName+0x306
fffff880`08600700 fffff800`02599438 nt!IopCreateFile+0x2b7
fffff880`086007a0 fffff880`01405bcf nt!NtCreateFile+0x78
fffff880`08600830 fffff880`01405fbf sis!SipOpenBackpointerStream+0x10b
fffff880`086008f0 fffff880`0140657d sis!SipOpenCSFileWork+0x3bf
fffff880`08600c70 fffff800`02293b61 sis!SipOpenCSFile+0x21
fffff880`08600cb0 fffff800`0252ea26 nt!ExpWorkerThread+0x111
fffff880`08600d40 fffff800`02264866 nt!PspSystemThreadStartup+0x5a
fffff880`08600d80 00000000`00000000 nt!KxStartSystemThread+0x16
```

Looking at the call stack for the above thread we can see that sis.sys is trying to acquire the eresource shared. Ordinarily, if a thread already owns an eresource exclusive, it can obtain it shared without first releasing the exclusive ownership. In this scenario the kernel will compare the eresource's owner field to the current thread and if they match the thread will be allowed to take shared ownership of the eresource. This is where the danger of ExSetResourceOwnerPointer comes into play. If you change the owner field with ExSetResourceOwnerPointer then this check fails because the owner field doesn't match the current thread.

The result of this scenario is that the thread waits for the exclusive owner to release the lock so this thread can get shared access. Unfortunately this thread is the exclusive owner, and it is the shared waiter. The thread has deadlocked on itself.

Even if you are careful in your handling of the resource after calling ExSetResourceOwnerPointer, there is often a risk that your driver may be re-entered in the same thread and you may end up in a scenario you didn't initially anticipate. This is why using this API is dangerous, and should be avoided when not absolutely necessary.

This issue demonstrated in this article was addressed in **KB2608658** (issue 3), which is available for download from the **Microsoft Download Center**.

Comments

Janine Patterson
30 Jul 2012 1:59 AM

Great Sometimes i see very nice and easy created blogs but in the most ways they are very usefull like your blog