Why is the debugger telling me I crashed because my DLL was unloaded, when I see it loaded right here happily executing code?

devblogs.microsoft.com/oldnewthing/20140523-00

May 23, 2014

Raymond Chen

A customer was puzzled by what appeared to be contradictory information coming from the debugger.

> We have Windows Error Reporting failures that tell us that we are executing code in our DLL which has been unloaded. Here's a sample stack:
>
> ```
> Child-SP          RetAddr           Call Site
> 00000037`7995e8b0 00007ffb`fe64b08e ntdll!RtlDispatchException+0x197
> 00000037`7995ef80 000007f6`e5d5390c ntdll!KiUserExceptionDispatch+0x2e
> 00000037`7995f5b8 00007ffb`fc977640 <Unloaded_contoso.dll>+0x3390c
> 00000037`7995f5c0 00007ffb`fc978296 RPCRT4!NDRSRundownContextHandle+0x18
> 00000037`7995f610 00007ffb`fc9780ed RPCRT4!DestroyContextHandlesForGuard+0xea
> 00000037`7995f650 00007ffb`fc9b5ff4
> RPCRT4!ASSOCIATION_HANDLE::~ASSOCIATION_HANDLE+0x39
> 00000037`7995f680 00007ffb`fc9b5f7c RPCRT4!LRPC_SASSOCIATION::`scalar deleting
> destructor'+0x14
> 00000037`7995f6b0 00007ffb`fc978b25
> RPCRT4!LRPC_SCALL_BROKEN_FLOW::FreeObject+0x14
> 00000037`7995f6e0 00007ffb`fc982e44
> RPCRT4!LRPC_SASSOCIATION::MessageReceivedWithClosePending+0x6d
> 00000037`7995f730 00007ffb`fc9825be RPCRT4!LRPC_ADDRESS::ProcessIO+0x794
> 00000037`7995f870 00007ffb`fe5ead64 RPCRT4!LrpcIoComplete+0xae
> 00000037`7995f910 00007ffb`fe5e928a ntdll!TppAlpcpExecuteCallback+0x204
> 00000037`7995f980 00007ffb`fc350ce5 ntdll!TppWorkerThread+0x70a
> 00000037`7995fd00 00007ffb`fe60f009 KERNEL32!BaseThreadInitThunk+0xd
> 00000037`7995fd30 00000000`00000000 ntdll!RtlUserThreadStart+0x1d
> ```
>
> But if we ask the debugger what modules are loaded, our DLL is right there, loaded as happy as can be:
>
> ```
> 0:000> lm
> start             end                 module name
> ...
> 000007f6`e6000000 000007f6`e6050000   contoso    (deferred)
> ...
> ```
>
> In fact, we can view other threads in the process, and they are happily running code in our DLL. What's going on here?

All the information you need to solve this problem is given right there in the problem report. You just have to put the pieces together.

Let's take a closer look at that `<Unloaded_contoso.dll>+0x3390c` entry. The address that the symbol refers to is the return address from the previous frame: `000007f6`e5d5390c`. Subtract `0x3390c` from that, and you get `000007f6`e5d20000`, which is the base address of the unloaded module.

On the other hand, the `lm` command says that the currently-loaded copy of `contoso.dll` is loaded at `000007f6`e6000000`. This is a *different address*.

What happened here is that `contoso.dll` was loaded into memory at `000007f6`e5d20000`, and then it ran for a while. The DLL was then unloaded from memory, and later loaded back into memory. When it returned, it was loaded at a different address `000007f6`e6000000`. For some reason (improper cleanup when unloading the first copy, most likely), there was still a function pointer pointing into the old unloaded copy, and when `NDRSRundownContextHandle` tries to call into that function pointer, it calls into an unloaded DLL, and you crash.

When faced with something that seems impossible, you need to look more closely for clues that suggest how your implicit assumptions may be incorrect. In this case, the assumption was that there was only one copy of `contoso.dll`.

Raymond Chen

Follow