What does it mean when I get an access violation at a very low address when entering a critical section?

devblogs.microsoft.com/oldnewthing/20170922-00

September 22, 2017

Raymond Chen

Warning: This article talks about implementation details which can change at any time. The information provided is for debugging and diagnostic purposes only.

A customer found that their server program occasionally crashes in the internal function `RtlpWaitOnCriticalSection` trying to dereference the address `0x00000014`.

```
7789dde3 ff4014          inc     dword ptr [eax+14h]
```

The dereference was due to a null pointer in the `EAX` register. This was particularly difficult to debug because the problem usually didn't surface until the program had been running continuously for a week or more.

The customer chased the null pointer backwards and found that it came from the `DebugInfo` field of the `RTL_CRITICAL_SECTION` structure.

```
typedef struct _RTL_CRITICAL_SECTION
{
                                             // value in memory:
     PRTL_CRITICAL_SECTION_DEBUG DebugInfo;  // 0x00000000
     LONG LockCount;                         // 0xFFFFFFFC
     LONG RecursionCount;                    // 0x00000000
     PVOID OwningThread;                     // 0x00000000
     PVOID LockSemaphore;                    // 0x00005CDC
     ULONG SpinCount;                        // 0x00000000
} RTL_CRITICAL_SECTION, *PRTL_CRITICAL_SECTION;
```

The customer confirmed that, yes, the `DebugInfo` of the critical section they were trying to enter was indeed null.

Although the customer didn't do it in their application (at least not knowingly), they did try a test application which passed the `CRITICAL_SECTION_NO_DEBUG_INFO` flag to the `InitializeCriticalSectionEx` function, in the hopes of inducing a null pointer for the `DebugInfo`, but it didn't work. When initialized in that way, the `DebugInfo` was set to `0xFFFFFFFF`.

Is it possible that this is a critical section that was initialized with the traditional `InitializeCriticalSection` function, but the attempt to allocate the debug info failed, so the kernel left it null?

No, that's not why the the `DebugInfo` is null. If a critical section has no debug info (either explicitly requested as such with the `CRITICAL_SECTION_NO_DEBUG_INFO` flag, or because the system couldn't allocate any debug info), then the `DebugInfo` is set to the special value `0xFFFFFFFF`. The `DebugInfo` for a valid initialized critical section is never null.

So what does it mean when the `DebugInfo` is null? The most likely reason is that you are using an uninitialized critical section. Either you never initialized it, or you deleted an initialized critical section (which resets it back to the uninitialized state).

Other evidence that you have an uninitialized critical section is that the critical section is locked, yet has no owner. Furthermore, the spin count is zero, which occurs only on uniprocessor systems. I suspect the server they are running the program on has more than one core. (Heck, my *phone* has more than one core.)

Bonus reading: Displaying a critical section in the debugger.

Related: I hope you werent using those undocumented critical section fields.

Raymond Chen

Follow