The case of the orphaned LpdrLoaderLock critical section

devblogs.microsoft.com/oldnewthing/20140808-00

August 8, 2014

Raymond Chen

A customer found that under heavy load, their application would occasionally hang. Debugging the hang shows that everybody was waiting for the `LpdrLoaderLock` critical section. The catch was that the critical section was reported as locked, but the owner thread did not exist.

```
0:000> !critsec ntdll!LdrpLoaderLock
CritSec ntdll!LdrpLoaderLock+0 at 7758c0a0
WaiterWoken        No
LockCount          4
RecursionCount     2
OwningThread       150c
EntryCount         0
ContentionCount    4
*** Locked
0:000> ~~[150c]k
              ^ Illegal thread error in '~~[150c]k'
```

How can a critical section be owned by thread that no longer exists?

There are two ways this can happen. One is that there is a bug in the code that manages the critical section such that there is some code path that takes the critical section but forgets to release it. This is unlikely to be the case for the loader lock (since a lot of really smart people are in charge of the loader lock), but it's a theoretical possibility. We'll keep that one in our pocket.

Another possibility is that the code to exit the critical section never got a chance to run. For example, somebody may have thrown an exception across the stack frames which manage the critical section, or somebody may have tried to exit the thread from inside the critical section, or (gasp) somebody may have called `TerminateThread` to nuke the thread from orbit.

I suggested that the `TerminateThread` theory was a good one to start with, because even if it wasn't the case, the investigation should go quickly because the light is better. You're not so much interested in finding it as you are in ruling it out quickly.[1]

The customer replied, “We had one call to `TerminateThread` in our application, and we removed it, but the problem still occurs. Is it worth also checking the source code of the DLLs our application links to?”

Okay, the fact that they *found one at all* means that there’s a good chance others are lurking.

Before we could say, “Yes, please continue your search,” the customer followed up. “We found a call to `TerminateThread` in a component provided by another company. The code created a worker thread, and decided to clean up the worker thread by terminating it. We commented out the call just as a test, and it seems to fix the problem. So at least now we know where the problem is and we can try to fix it properly.”

[1] In the medical profession, there’s the term ROMI, which stands for *rule out myocardial infarction*. It says that if a patient comes to you with anything that could possibly remotely be the symptom of a heart attack, like numbness in the arm or chest pain, you should perform a test to make sure. Even though the test is probably going to come back negative, you have to check just to be safe. Here, we’re ruling out `TerminateThread`, which I guess could go by the dorky acronym ROTT.

Raymond Chen

Follow