Why does my attempt to acquire an SRW lock block even though !locks report no locks held?

devblogs.microsoft.com/oldnewthing/20170301-00

March 1, 2017

Raymond Chen

A customer asked for some help debugging a problem with their application:

> We have an application that synchronizes access to resources by using SRW locks rather than critical sections. We have traced execution in the debugger up to the point at which we're about to call `AcquireSRWLockShared`. Trying to step over the call in WinDbg sends the program into Running mode. Breaking back into the debugger and using the `!locks` command reports "no locks", confirming that that there are no deadlocks on the lock. But why is `AcquireSRWLockShared` not returning?

The debugger's `!locks` command reports on critical section objects. Says so right on the tin:

> The **!locks** extension in Ntsdexts.dll displays a list of critical sections associated with the current process.

It does not have any insight into SRW locks or any other synchronization objects.

The program is entering Running mode when you try to step over the call to `AcquireSRWLockShared` because the call is blocking, presumably because another thread has acquired the SRW lock in exclusive mode and has yet to release that lock. (Other possibilities are that the acquirer of the exclusive SRW lock released it incorrectly, or that the SRW lock is corrupted.)

There are no built-in diagnostics for SRW locks. These locks retain no diagnostic information; that's what makes them slim. If you need to debug your program's use of SRW locks, you can wrap the SRW lock functions inside your own helper functions that record additional diagnostic information. Or you can use Application Verifier, which is basically the same thing, just that the diagnostic information is recorded by Application Verifier. Or you can add additional logging to your program to try to reconstruct the history of the SRW lock to find out who acquired it exclusively and failed to release it properly.

Raymond Chen

Follow