What can go wrong if you release an SRWLock from a thread different from the one that acquired it?

devblogs.microsoft.com/oldnewthing/20230623-00

June 23, 2023

Raymond Chen

A customer found that if they ran their program under Application Verifier, the tool identified that their program was acquiring an SRWLock in exclusive mode on one thread, but was releasing it on another thread. They found a remark in the documentation:

> The SRW lock must be released by the same thread that acquired it.

They asked what goes wrong if you break this rule, because the documentation doesn't say.

The documentation doesn't say what goes wrong because the operation is not supported, so the behavior is undefined. You don't have to define undefined behavior.

A member of the kernel team nevertheless explained what goes wrong.

The system keeps track of which thread have acquired an SRWLock so that it can deal with priority inversion scenarios: If a low-priority thread acquires an SRWLock, and a higher-priority thread blocks waiting to acquire the SRWLock, then the priority of the waiting thread is transferred to the owner thread.[1]

This information to assist in avoiding priority inversion is kept in preallocated per-thread data.

If you release an SRWLock from a thread different from the one that acquired it, then the internal bookkeeping gets all messed up. The most likely result is that the priority transfer to the acquiring thread is never removed (since it was cleaned up by the wrong thread). From an outsider's point of view, it looks like the acquiring thread's priority got permanently boosted for no reason.

The kernel folks didn't say, but I suspect that another consequence is that any priority transfer to the releasing thread is removed prematurely, so it looks like the releasing thread's priority got demoted for no reason.

All of this is implementation detail and not contractual. The contractual requirement is that you release the SRWLock from the same thread that acquired it. If you fail to follow this rule, then the behavior is undefined, and your program will behave erratically in unspecified ways.

So don't do that. Follow the rules and nobody gets hurt.

[1] Priority inversion was famously the source of hangs on the Mars Pathfinder mission in 1997.