How bad is it to delay closing a thread handle for a long time after the thread has exited?

devblogs.microsoft.com/oldnewthing/20161215-00

December 15, 2016

Raymond Chen

A customer has a component that creates a thread with `CreateThread` in order to do *something*, and eventually that thread exits normally. The code hangs onto the thread handle for the lifetime of the component, because the component wants to wait until the worker thread has fully exited before it will shut down. The component eventually closes the thread handle, but it may take a very long time before the handle gets closed.

The customer's question was basically, "How bad is it to delay closing a thread handle for a long time after the thread has exited?" They were concerned that failing to close the handle would have a noticeable impact upon the host process, like leaving a megabyte of memory reserved for the thread's stack.

On the other hand, if the impact is *de minimis*, then the customer would rather not add complexity and tinker with code that has been working just fine so far.

Fortunately, the answer is, "It's not that bad." When the thread exits, nearly all of its resources are released. There may be some straggling resources like a (now-empty) data structure to keep track of the outstanding I/O for the thread, and data members to record the thread's exit code, thread times, security descriptor, processor affinity, and other miscellaneous information.

But it's not a significant amount of extra data in the grand scheme of things if you're going to have only one of these "long-lived thread handles" per process. Just don't make it a habit. (Now, if you're going to have thousands of them, we may need to talk.)

Raymond Chen

Follow