# What does an invalid handle exception in LeaveCriticalSection mean?

devblogs.microsoft.com/oldnewthing/20061211-00

December 11, 2006

Raymond Chen

Internally, a critical section is a bunch of counters and flags, and possibly an event. (Note that the internal structure of a critical section is subject to change at any time—in fact, it changed between Windows XP and Windows 2003. The information provided here is therefore intended for troubleshooting and debugging purposes and not for production use.) As long as there is no contention, the counters and flags are sufficient because nobody has had to wait for the critical section (and therefore nobody had to be woken up when the critical section became available).

If a thread needs to be blocked because the critical section it wants is already owned by another thread, the kernel creates an event for the critical section (if there isn't one already) and waits on it. When the owner of the critical section finally releases it, the event is signaled, thereby alerting all the waiters that the critical section is now available and they should try to enter it again. (If there is more than one waiter, then only one will actually enter the critical section and the others will return to the wait loop.)

If you get an invalid handle exception in `LeaveCriticalSection`, it means that the critical section code thought that there were other threads waiting for the critical section to become available, so it tried to signal the event, but the event handle was no good.

Now you get to use your brain to come up with reasons why this might be.

One possibility is that the critical section has been corrupted, and the memory that normally holds the event handle has been overwritten with some other value that happens not to be a valid handle.

Another possibility is that some other piece of code passed an uninitialized variable to the `CloseHandle` function and ended up closing the critical section's handle by mistake. This can also happen if some other piece of code has a double-close bug, and the handle (now closed) just happened to be reused as the critical section's event handle. When the buggy code closes the handle the second time by mistake, it ends up closing the critical section's handle instead.

Of course, the problem might be that the critical section is not valid because it was never initialized in the first place. The values in the fields are just uninitialized garbage, and when you try to leave this uninitialized critical section, that garbage gets used as an event handle, raising the invalid handle exception.

Then again, the problem might be that the critical section is not valid because it has already been destroyed. For example, one thread might have code that goes like this:

```
EnterCriticalSection(&cs);
... do stuff...
LeaveCriticalSection(&cs);
```

While that thread is busy doing stuff, another thread calls `DeleteCriticalSection(&cs)`. This destroys the critical section while another thread was still using it. Eventually that thread finishes doing its stuff and calls `LeaveCriticalSection`, which raises the invalid handle exception because the `DeleteCriticalSection` already closed the handle.

All of these are possible reasons for an invalid handle exception in `LeaveCriticalSection`. To determine which one you're running into will require more debugging, but at least now you know what to be looking for.

Postscript: One of my colleagues from the kernel team points out that the Locks and Handles checks in Application Verifier are great for debugging issues like this.

Raymond Chen

**Follow**