Why did HeapFree fail with ERROR_POSSIBLE_DEADLOCK?

devblogs.microsoft.com/oldnewthing/20120106-00

January 6, 2012

Raymond Chen

A customer reported that they were receiving some assertion failures because the `HeapFree` function was failing with what they believed to be a valid heap block, and the `GetLast-Error` function reported that the reason for failure was `ERROR_POSSIBLE_DEADLOCK`. What's going on? One of my colleagues asked the psychic question, "Is the process exiting?" "Why yes, in fact it is. How did you know?" Recall how processes exit. One of the first things that happens is that all the other threads in the process are forcible terminated, which has as a consequence that any synchronization resources owned by those threads are now orphaned. And in this case, the synchronization resource in question was the heap. When the function calls `HeapFree`, the heap code tries to take the heap lock but finds that it can't because the heap lock was owned by another thread. And that other thread no longer exists. (Perhaps it was terminated while it was in the middle of its own `HeapFree` operation.) The heap code detects this and instead of deadlocking on its own custom synchronization object, it fails with the error `ERROR_POSSIBLE_DEADLOCK`.

By the same logic, you can demonstrate that you cannot reliably allocate memory at process shutdown either. So now you can't allocate memory; you can't free memory. As we saw last time, when you are told that the process is exiting, you should not do any cleanup at all. The memory will get freed when the process address space is torn down. No need to free it manually; that's just a waste of time.

Raymond Chen

Follow