What is this inconsistent heap state that MSDN warns me about during DLL_PROCESS_DETACH?

devblogs.microsoft.com/oldnewthing/20150213-00

February 13, 2015

Raymond Chen

In the MSDN documentation for the `DllMain` entry point, MSDN notes:

> When handling **DLL_PROCESS_DETACH**, a DLL should free resources such as heap memory only if the DLL is being unloaded dynamically (the *lpReserved*[1] parameter is **NULL**). If the process is terminating (the *lpvReserved* parameter is non-**NULL**), all threads in the process except the current thread either have exited already or have been explicitly terminated by a call to the **ExitProcess** function, which might leave some process resources such as heaps in an inconsistent state. In this case, it is not safe for the DLL to clean up the resources. Instead, the DLL should allow the operating system to reclaim the memory.

A customer wanted to know, "What is this inconsistent heap state that MSDN is talking about here?"

The information is actually right there in the sentence. "... explicitly terminated by a call to the **ExitProcess** function, which might leave some process resources such as heaps in an inconsistent state."

When you see text that says "X might lead to Y," then when you ask "What could lead to Y?" you might consider that it is X.

> **Background reading**: Quick overview of how processes exit on Windows XP, plus bonus electrification of critical sections and slim reader/writer locks.

Okay, I'll give the customer the benefit of the doubt and assume that the question was more along the lines of "Why would termination by a call to the **ExitProcess** function lead to an inconsistent state?"

Remember why `TerminateThread` is a horrible idea: It terminates the thread in the middle of whatever it is doing, not giving it a chance to restore consistency or otherwise provide for an orderly end to operations. The thread may have been in the middle of updating a data

structure, which usually involves perturbing an invariant, then re-establishing it. If it was terminated before it could re-establish the invariant, you now have an inconsistent data structure.

That's the sort of inconsistency the paragraph is talking about. If one of the threads terminated by `ExitProcess` was executing heap code at the moment it was terminated, then the heap may be inconsistent because the thread never got a chance to re-establish consistency. (The heap tries to detect that this has happened but all that does is transform one failure mode into another. The failure is still there.)

Of course, the heap isn't the only resource that suffers from this problem. Any resource that is available to more than one thread is susceptible to this. It's just that the heap is a very popular shared resource, so it gets an extra mention in the documentation.

[1] Typo. Should be *lpvReserved*.

Raymond Chen

Follow