When DLL_PROCESS_DETACH tells you that the process is exiting, your best bet is just to return without doing anything

devblogs.microsoft.com/oldnewthing/20120105-00

January 5, 2012

Raymond Chen

When the `DllMain` function receives a reason code of `DLL_PROCESS_DETACH`, the increasingly-inaccurately-named `lpReserved` parameter to is used to indicate whether the process is exiting. And if the process is exiting, then you should just return without doing anything. No, really. Don't worry about freeing memory; it will all go away when the process address space is destroyed. Don't worry about closing handles; handles are closed automatically when the process handle table is destroyed. Don't try to call into other DLLs, because those other DLLs may already have received their `DLL_PROCESS_DETACH` notifications, in which case they may behave erratically in the same way that a Delphi object behaves erratically if you try to use it after its destructor has run. The building is being demolished. Don't bother sweeping the floor and emptying the trash cans and erasing the whiteboards. And don't line up at the exit to the building so everybody can move their in/out magnet to *out*. All you're doing is making the demolition team wait for you to finish these pointless housecleaning tasks. Okay, if you have internal file buffers, you can write them out to the file handle. That's like remembering to take the last pieces of mail from the mailroom out to the mailbox. But don't bother closing the handle or freeing the buffer, in the same way you shouldn't bother updating the "mail last picked up on" sign or resetting the flags on all the mailboxes. And ideally, you would have flushed those buffers as part of your normal wind-down before calling `ExitProcess`, in the same way mailing those last few letters should have been taken care of before you called in the demolition team. I regularly use a program that doesn't follow this rule. The program allocates a lot of memory during the course of its life, and when I exit the program, it just sits there for several minutes, sometimes spinning at 100% CPU, sometimes churning the hard drive (sometimes both). When I break in with the debugger to see what's going on, I discover that the program isn't doing anything productive. It's just methodically freeing every last byte of memory it had allocated during its lifetime. If my computer wasn't under a lot of memory pressure, then most of the memory the program had allocated during its lifetime hasn't yet been paged out, so freeing every last drop of memory is a CPU-bound operation. On the other hand, if I had kicked off a build or done something else memory-intensive, then most of the memory the

program had allocated during its lifetime has been paged out, which means that the program pages all that memory back in from the hard drive, just so it could call `free` on it. Sounds kind of spiteful, actually. “Come here so I can tell you to go away.”

All this anal-rententive memory management is pointless. The process is exiting. All that memory will be freed when the address space is destroyed. *Stop wasting time and just exit already*.

Raymond Chen

Follow