The thread that gets the DLL_PROCESS_DETACH notification is not necessarily the one that got the DLL_PROCESS_ATTACH notification

devblogs.microsoft.com/oldnewthing/20090626-00

June 26, 2009

Raymond Chen

The thread that gets the `DLL_PROCESS_DETACH` notification is not necessarily the one that got the `DLL_PROCESS_ATTACH` notification. This is obvious if you think about it, because the thread that got the `DLL_PROCESS_ATTACH` notification might not even exist any longer when the DLL is unloaded. How can something that doesn't exist send a notification?

Even so, many people fail to realize this. You can't do anything with thread affinity in your `DLL_PROCESS_ATTACH` or `DLL_PROCESS_DETACH` handler since you have no guarantee about which thread will be called upon to handle these process notifications. Of course, you're not supposed to be doing anything particularly interesting in your `DLL_PROCESS_ATTACH` handler anyway, but things with thread affinity are doubly bad.

The classic example of this, which I'm told the Developer Support team run into with alarming frequency, is a DLL that creates a window in its `DLL_PROCESS_ATTACH` handler and destroys it in its `DLL_PROCESS_DETACH` handler. Now, creating a window in `DllMain` is already a horrifically bad idea since arbitrary code can run during the creation of a window (for example, there may be a global hook), but the lack of a thread guarantee makes it downright insane. The DLL calls `DestroyWindow` in its `DLL_PROCESS_DETACH` handler, but since that notification comes in on a thread different from the one that received the `DLL_PROCESS_ATTACH` notification, the attempt to destroy the window fails since you must call `DestroyWindow` from the same thread that created it.

Result: The DLL's attempt to destroy its window fails, a message comes in, and the process crashes since the window procedure no longer exists.

Raymond Chen

Follow