Some reasons not to do anything scary in your DllMain, part 3

devblogs.microsoft.com/oldnewthing/20140821-00

August 21, 2014

Raymond Chen

In the same week, the shell team was asked to investigate two failures. The first one was a deadlock in Explorer. The participating threads look like this:

- Thread 1 called `FreeLibrary` on a shell extension as part of normal `CoFreeUnused-Libraries` processing. That DLL called `OleUninitialize` from its `DllMain` function. This thread blocked because the COM lock was held by thread 2.
- Thread 2 called `CoCreateInstance`, and COM tried to load the DLL which handles the object, but the thread blocked because the loader lock was held by thread 1.

The shell extension caused this problem because it ignored the rule against calling shell and COM functions from the `DllMain` entry point, as specifically called out in the `DllMain` documentation as examples of functions that should not be called. The authors of this shell extension may never have caught this problem in their internal testing (or if they did they didn't understand what it meant) because hitting this deadlock requires that a race window be hit: The shell extension DLL needs to be unloaded on one thread at the exact same moment another thread is inside the COM global lock trying to load another DLL. Meanwhile, another failure was traced back to a DLL calling `CoInitialize` from their `DllMain`. This extra COM initialization count means that when the thread called `Co-Uninitialize` thinking that it was uninitializing COM, it actually merely decremented the count to 1. The code then proceeded to do things that are not allowed in a single-threaded apartment, believing that it had already torn down the apartment. But the secret `Co-Initialize` performed by the shell extension violated that assumption. Result: A thread that stopped responding to messages. The authors of both of these shell extensions seemed be calling `CoUninitialize` / `OleUninitialize` in order to cancel out a `Co-Initialize` / `OleInitialize` which they performed in their `DLL_PROCESS_ATTACH`. This is fundamentally unsound not only because of the general rule of not calling COM functions inside `DllMain` but also because OLE initialization is a per-thread state, whereas the thread that gets the `DLL_PROCESS_DETACH` notification is not necessarily the one that receives the `DLL_PROCESS_ATTACH` notification. It so happens that in the second case, the DLL in question was a shell copy hook, and the hang was occuring not in Explorer but in an application which was using `SHFileOperation` to delete some files. We could at least advise the application authors to pass the `FOFX_NOCOPYHOOKS` flag to `IFileOperation::SetOperationFlags` to prevent copy hooks from being loaded.

Previous articles in this series: Part 1, Part 2.

Raymond Chen

Follow