Some reasons not to do anything scary in your DllMain

devblogs.microsoft.com/oldnewthing/20040127-00

January 27, 2004

Raymond Chen

As everybody knows by now, you're not supposed to do anything even remotely interesting in your DllMain function. Oleg Lvovitch has written two very good articles about this, one about how things work, and one about what goes wrong when they don't work.

Here's another reason not to do anything remotely interesting in your DllMain: It's common to load a library without actual intent to invoke its full functionality. For example, somebody might load your library like this:

```
// error checking deleted for expository purposes
hinst = LoadLibrary(you);
hicon = LoadIcon(you, MAKEINTRESOURCE(5));
FreeLibrary(hinst);
```

This code just wants your icon. It would be very surprised (and perhaps even upset) if your DLL did something heavy like starting up a timer or a thread.

(Yes, this could be avoided by using LoadLibraryEx and LOAD_LIBRARY_AS_DATAFILE, but that's not my point.)

Another case where your library gets loaded even though no code is going to be run is when it gets tugged along as a dependency for some other DLL. Suppose "middle" is the name of some intermediate DLL that is linked to your DLL.

```
hinst = LoadLibrary(middle);
pfn = GetProcAddress(hinst, "SomeFunction");
pfn(...);
FreeLibrary(hinst);
```

When "middle" is loaded, your DLL will get loaded and initialized, too. So your initialization runs even if "SomeFunction" doesn't use your DLL.

This "intermediate DLL loaded for a brief time" scenario is actually quite common. For example, if somebody does "Regsvr32 middle.dll", that will load the middle DLL to call its DllRegisterServer function, which typically doesn't do much other than install some registry keys. It almost certainly doesn't call into your helper DLL.

Another example is the opening of the Control Panel folder. The Control Panel folder loads every *.cpl file so it can call its CplApplet function to determine what icon to display. Again, this typically will not call into your helper DLL.

And under no circumstances should you create any objects with thread affinity in your DLL_PROCESS_ATTACH handler. You have no control over which thread will send the DLL_PROCESS_ATTACH message, nor which thread will send the DLL_PROCESS_DETACH message. The thread that sends the DLL_PROCESS_ATTACH message might terminate immediately after it loads your DLL. Any object with thread-affinity will then stop working since its owner thread is gone.

And even if that thread survives, there is no guarantee that the thread that calls FreeLibrary is the same one that called LoadLibrary. So you can't clean up those objects with thread affinity in DLL_PROCESS_DETACH since you're on the wrong thread.

And absolutely under no circumstances should you be doing anything as crazy as creating a window inside your DLL_PROCESS_ATTACH. In addition to the thread affinity issues, there's the problem of global hooks. Hooks running inside the loader lock are a recipe for disaster. Don't be surprised if your machine deadlocks.

Even more examples to come tomorrow.

Raymond Chen

Follow