CoUninitalize will ask a DLL if it is okay to unload now, but the answer is a foregone conclusion

devblogs.microsoft.com/oldnewthing/20131106-00

November 6, 2013

Raymond Chen

The `DllCanUnloadNow` entry point is exported by COM in-proc servers. COM host applications call `CoFreeUnusedLibraries` periodically to ask COM to do DLL housecleaning, and in response, COM asks each DLL if it is okay to be unloaded. If so, then COM unloads the DLL.

What is not well-known is that COM also does DLL housecleaning when you shut down the last apartment by calling `CoUninitialize`. When that happens, COM will still go around asking each DLL whether it's okay to be unloaded, but the question is merely a formality, because *regardless of your answer, COM will unload you anyway*.

The story here is that COM is being shut down for the process, so COM knows that when the last `CoUninitialize` is finished, all COM objects will be destroyed. After all, if you don't have COM, then you can't have any COM objects.

As a courtesy, COM will ask you, "Is it okay to unload you?" in case you want to do some early cleanup. But it will ignore your answer.

This means that you need to exercise caution if you call `CoUninitialize` or `CoFreeUnusedLibraries` from your COM in-proc server, because the call may end up freeing your code out from under you.

For example, one third-party crash I investigated boiled down to a COM object whose destructor went like this:

```
MyComObject::~MyComObject()
{
 .. blah blah blah ..
 // Let DllCanUnloadNow know that we have one
 // fewer active COM object
 _Module.Unlock();
 CoFreeUnusedLibraries();
}
```

It so happened that this was the last COM object created by the DLL, so the `_Module.Unlock()` call dropped the DLL object count to zero. The COM server then inexplicably called `CoFreeUnusedLibraries` (something that is supposed to be called by the host, not a plug-in), and `CoFreeUnusedLibraries` did what it was told and asked each DLL, "Hey, do you mind if I unload you now?" The DLL's `DllCanUnloadNow` function saw that the active COM object count was zero, so it said, "Sure, go ahead."

I hope you see where this is going.

COM unloads your DLL because you said you were okay with it. The call to `CoFreeUnused-Libraries` eventually returns, but its return address is inside the `MyComObject` destructor, which was unloaded because *you said it was okay to unload*.

The fix here is to remove the call to `CoFreeUnusedLibraries`. It shouldn't have been there in the first place.

A more common error is creating a background thread without bumping the DLL reference count. When the last COM apartment shuts down, COM will free your DLL, thereby stranding your worker thread. You need to use the `FreeLibraryAndExitThread` trick to keep your DLL loaded until the background thread finishes.

Raymond Chen

Follow