User interface code + multi-threaded apartment = death

devblogs.microsoft.com/oldnewthing/20080424-00

April 24, 2008

Raymond Chen

There are single-threaded apartments and multi-threaded apartments. Well, first there were only single-threaded apartments. No wait, let's try that again.

First, applications had only one thread. Remember, 16-bit Windows didn't have threads. Each process had one of what we today call a thread, end of story. Compatibility with this ancient model still exists today, thanks to the dreaded "main" threading model. The less said about that threading model the better.

OLE was developed back in the 16-bit days, so it used window messages to pass information between processes, there being no other inter-process communication mechanism available. When you initialized OLE, it created a secret `OleMainThreadWnd` window, and those secret windows were used to communicate between processes (and in Win32, threads). As we learned some time ago, window handles have thread affinity, which means that these communication windows have thread affinity, which means that OLE has thread affinity. When you made a call to an object that belonged to another apartment, OLE posted a message to the owner thread's secret `OleMainThreadWnd` window to tell it what needs to be done, and then it went into a private message loop waiting for the owner thread to do the work and post the results back.

Meanwhile, the OLE team realized that there were really two parts to what they were doing. There was the low-level object and interface management stuff (`IUnknown` , `CoMarshalInterThreadInterfaceInStream`) and the high-level "object linking and embedding" stuff (`IOleWindow` , `IOleDocument`) that was the impetus for the OLE effort in the first place. The low-level stuff got broken out into a functional layer known as COM; the high-level stuff kept the name OLE.

Breaking the low-level and high-level stuff apart allowed the low-level stuff to be used by non-GUI programs, which for quite some time were eyeing that object management functionality with some jealousy. As a result, COM grew two personalities, one focused on the GUI customers and another focused on the non-GUI customers. For the non-GUI customers, additional functionality such as multi-threaded apartments were added, and since the

customers didn't do GUI stuff, multi-threaded apartments weren't burdened by the GUI rules. They didn't post messages to communicate with each other; they used kernel objects and `WaitForSingleObject`. Everybody wins, right?

Well, yes, everybody wins, but you have to know what side your bread is buttered on. If you initialize a GUI thread as a multi-threaded apartment, you have violated the assumptions under which multi-threaded apartments were invented! Multi-threaded apartments assume that they are not running on GUI threads since they don't pump messages; they just use `WaitForSingleObject`. This not only clogs up broadcasts, but it can also deadlock your program. The thread that owns the object might try to send a message to your thread, but your thread can't receive the message since it isn't pumping messages.

That's why COM objects involved with user interface programming nearly always require a single-threaded apartment and why `OleInitialize` initializes a single-threaded apartment. Because multi-threaded apartments were designed on the assumption that there was no user interface. Once you're doing user interface work, you have to use a single-threaded apartment.

Raymond Chen

Follow