In pursuit of the message queue

devblogs.microsoft.com/oldnewthing/20060221-09

February 21, 2006

Raymond Chen

In 16-bit Windows, every thread (or "task" as it was called then) had a message queue, end of story. In the transition to 32-bit Windows, this model broke down because Win32 introduced the concepts of "worker threads" and "console applications", neither of which had much need for messaging. Creating a queue for every thread in the system would have been quite a waste, so the window manager deferred creating the input queue for a thread until that thread actually needed an input queue. That way, threads that didn't use the GUI didn't have to pay for something they weren't using. But once you send a message or peek for a message or create a window or do anything else that requires a message queue, ***poof*** a message queue would be created just in time to accomodate your request. As far as you knew, the queue was always there. The create-on-demand queue model worked out great: Queues were created only when they were needed, threads that didn't need message queues didn't get one, and nobody knew the difference. There was only one catch: `PostThreadMessage`. When I started writing this entry, I was going to write that the behavior of the `PostThreadMessage` function is a mistake. Instead of failing if the thread doesn't have a queue, it should have preserved the conceit that the queue was there all along by creating the queue on demand. But thinking about it more (as the writing process forces you to do), I've now convinced myself that the current design is correct, even though it violates the "as far as you can tell, the queue is always there" principle. The `PostThreadMessage` function is peculiar among all the other queue-related functions in that it operates on the queue of **another thread** that may not already have a queue. All the other queue functions operate on the queue of the thread making the call or operate on the queue of a thread that is known to have a queue (because it created a window, for example). As a result, a thread is in control of whether it gets a message queue or not. If `PostThreadMessage` created a queue on demand, this would allow one process to start creating queues in other processes without those other processes knowing about it. It could then start filling that message queue with thousands upon thousands of posted messages, and the victim thread would have no idea not only that it had a message queue, but also that the message queue that somebody else created was full of unprocessed messages! No thread would be able to defend itself from this sort of attack.

Making the `PostThreadMessage` an exception to the "as if there always were a thread queue" rule keeps a thread in control of its own queue destiny.

Raymond Chen

Follow