Why is there a special PostQuitMessage function?

devblogs.microsoft.com/oldnewthing/20051104-33

November 4, 2005

Raymond Chen

Why is there a special PostQuitMessage function? Because it's not really a posted message.

Commenter A. Skrobov asked, "What's the difference between `PostQuitMessage` and `PostThreadMessage(GetCurrentThreadId, WM_QUIT)`?"

They are not equivalent, though they may look that way at first glance. The differences are subtle but significant.

Like the `WM_PAINT`, `WM_MOUSEMOVE`, and `WM_TIMER` messages, the `WM_QUIT` message is not a "real" posted message. Rather, it is one of those messages that the system generates **as if** it were posted, even though it wasn't. And like the other messages, the `WM_QUIT` message is a "low priority" message, generated only when the message queue is otherwise empty.

When a thread calls `PostQuitMessage`, a flag in the queue state is set that says, "If somebody asks for a message and there are no posted messages, then manufacture a `WM_QUIT` message." This is just like the other "virtually posted" messages. `WM_PAINT` messages are generated on demand if there are any invalid regions, `WM_MOUSEMOVE` messages are generated on demand if the mouse has moved since the last time you checked, and `WM_TIMER` messages are generated on demand if there are any due timers. And since the message is "virtually posted", multiple calls coalesce, in the same way that multiple paint messages, multiple mouse motions, and multiple timer messages also coalesce.

Why is `WM_QUIT` handled like a low-priority message?

Because the system tries not to inject a `WM_QUIT` message at a "bad time"; instead it waits for things to "settle down" before generating the `WM_QUIT` message, thereby reducing the chances that the program might be in the middle of a multi-step procedure triggered by a sequence of posted messages.

If you `PeekMessage(..., PM_NOREMOVE)` a `WM_QUIT` message, this returns a `WM_QUIT` message but does not clear the flag. The `WM_QUIT` message virtually "stays in the queue".

As another special behavior, the generated `WM_QUIT` message bypasses the message filters passed to the `GetMessage` and `PeekMessage` functions. If the internal "quit message pending" flag is set, then you will get a `WM_QUIT` message once the queue goes quiet, regardless of what filter you pass.

By comparison, `PostThreadMessage` just places the message in the thread queue (for real, not virtually), and therefore it does not get any of the special treatment that a real `PostQuitMessage` triggers.

Raymond Chen

Follow