What happens to a sent message when SendMessageTimeout reaches its timeout?

devblogs.microsoft.com/oldnewthing/20110915-00

September 15, 2011

Raymond Chen

The `SendMessageTimeout` function tries to send a message, but gives up if the timeout elapses. What exactly happens when the timeout elapses? It depends. The first case is if the receiving thread never received the message at all. (I.e., if during the period the sender is waiting, the receiving thread never called `GetMessage`, `PeekMessage`, or a similar message-retrieval function which dispatches inbound sent messages.) In that case, if the timeout is reached, then the entire operation is canceled; the window manager cleans up everything and makes it look as if the call to `SendMessageTimeout` never took place. The message is removed from the list of the thread's non-queued messages, and when it finally gets around to calling `GetMessage` (or whatever), the message will not be delivered. The second case is if the receiving thread received the message, and the message was delivered to the destination window procedure, but the receiving thread is just slow to process the message and either return from its window procedure or call `ReplyMessage`. In that case, if the timeout is reached, then the sender is released from waiting, but the message is allowed to proceed to completion. Since people seem to like tables, here's a timeline showing the two cases.

Sending thread	Case 1	Case 2
`SendMessageTimeout(WM_X)` called	… not responding …	… not responding …
	… not responding …	… not responding …
	… not responding …	`GetMessage()` called
	… not responding …	`WndProc(WM_X)` called
	… not responding …	`WndProc(WM_X)` still executing
timeout elapses	… not responding …	`WndProc(WM_X)` still executing

`SendMessageTimeout(WM_X)` returns	… not responding …	`WndProc(WM_X)` still executing
	… not responding …	`WndProc(WM_X)` returns
	`GetMessage()` called	
	(message `WM_X` *not* received)	

Notice that in case 2, the window manager has little choice but to let the window procedure continue with the message. After all, time travel has yet to be perfected, so the window manager can't go back in time and tell the younger version of itself, (Possibly with a slow-motion "Noooooooooooo" for dramatic effect.) "No, don't give him the message; he won't finish processing it in time!" If you are in case 2 and the message `WM_X` is a system-defined message that is subject to marshaling, then the data is not unmarshaled until the window procedure returns. It would be bad to free the memory out from under a window procedure. On the other hand, if the message is a custom message, then you are still on the hook for keeping the values valid until the window procedure is done.

But wait, how do I know when the window procedure is done? The `SendMessageTimeout` function doesn't tell me! Yup, that's right. If you need to do cleanup after message processing is complete, you should use the `SendMessageCallback` function, which calls you back when the receiving thread completes message processing. When the callback fires, that's when you do your cleanup.

Raymond Chen

Follow