Even though mouse-move, paint, and timer messages are generated on demand, it’s still possible for one to end up in your queue

devblogs.microsoft.com/oldnewthing/20130523-00

May 23, 2013

Raymond Chen

We all know that the generated-on-demand messages like `WM_MOUSEMOVE`, `WM_PAINT`, and `WM_TIMER` messages are not posted into the queue when the corresponding event occurs, but rather are generated by `GetMessage` or `PeekMessage` when they detect that they are about to conclude that there is no message to return and the generated-on-demand message can be returned. When this happens, the window manager creates the message on the fly, posts it into the queue, and *hey, how about that*, the `GetMessage` or `PeekMessage` function now has a message to return!

Note that this auto-generate can happen even though the queue is not empty, because the message filters control what messages in the queue can be returned. For example, suppose the message queue contains the following messages:

- `{ hwnd1, WM_CLIPBOARDUPDATE }`
- `{ hwnd2, WM_LBUTTONDOWN }`

(Note that the above diagram is not strictly correct, because the `WM_LBUTTONDOWN` message goes into the input queue, not the message queue, but the distinction is not important here.)

Suppose you now call `GetMessage(&msg, hwnd1, WM_MOUSEFIRST, WM_MOUSELAST)`. None of the messages in the queue satisfy the message filter: The first message meets the window filter, but the message is not in range. The second message meets the message range filter, but does not meet the window filter. The `GetMessage` function is about to give up and say “I guess I need to wait for a message,” but before it finally concedes defeat, it says, “Hang on there. I see a note that tells me that I should auto-generate a `WM_MOUSEMOVE` message for window `hwnd1`. And that message satisfies the message filter. I’ll generate it now!”

The `GetMessage` function posts the `{ hwnd1, WM_MOUSEMOVE }` message into the queue (assigning it the current time as the timestamp), and then it says, “Hey, lookie here! A message that satisfies the filter!” It then removes the message from the queue and returns it.

(Note that this algorithm is conceptual. It doesn't actually work this way internally. In particular, the window manager does not literally talk to itself, at least not out loud.)

Okay, so in the `GetMessage` case, even if the message conceptually goes into the queue, it comes right back out immediately, so you never actually observe it there.

Now repeat the exercise with the `PeekMessage` function. As before, the `WM_MOUSEMOVE` message is posted into the queue with the current time as the timestamp. If the `PM_REMOVE` flag is passed, then the message is removed from the queue and returned, just like `Get-Message`. If the `PM_NOREMOVE` flag is passed, then things get interesting: The message is returned but not removed from the queue.

You now have a `WM_MOUSEMOVE` message *physically residing in the queue*!

This is the answer to the puzzle: If auto-generated messages are generated on demand, how is it possible for them to end up sitting in your message queue?

I recall a bug investigation from nearly two decades ago which basically boiled down to this issue: Somebody `PM_NOREMOVE`'d an auto-generated message and not only left it in the queue, but kept generating new ones without processing the old ones. Eventually, the message queue filled up.

(Note that this is also the answer to the puzzle: If `WM_MOUSEMOVE` is generated on demand, how can it be possible to retrieve a `WM_MOUSEMOVE` message with a timestamp different from the current time?)

Raymond Chen

Follow