How likely is it that a window will receive a WM_NULL message out of the blue?

devblogs.microsoft.com/oldnewthing/20170602-00

June 2, 2017

Raymond Chen

A customer discovered a bug in their control that resulted in a crash:

```
LRESULT CALLBACK MyWindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
  switch (uMsg) {
  ...
  default:
    if (uMsg == g_customRegisteredMessage) {
      // For this message, the lParam is a pointer
      return HandleCustomMessage((SOMETHING*)lParam);
    }
    break;
  }
  return DefWindowProc(hwnd, uMsg, wParam, lParam);
}
```

The problem is that under certain conditions, the control doesn't initialize the `g_customRegisteredMessage` variable. If a `WM_NULL` message arrives, the test

```
    if (uMsg == g_customRegisteredMessage) {
```

is `true`, and we take the `lParam` of the `WM_NULL` message and treat it as a pointer. Since the `lParam` of the `WM_NULL` message is usually zero, this causes the program to crash with a null pointer.

The customer fully acknowledged the bug. But their question was one of risk management. How likely is a window going to receive the `WM_NULL` mesasge? Knowing the likelihood of the scenario would help them decide how critical the fix is. (And they weren't able to reproduce the problem in-house, so as far as they could determine, the likelihood was effectively zero. And yet it was happening.)

The `WM_NULL` message is not a common one, but it's not uncommon either. Posting a `WM_NULL` is usually done by a window to itself in order to wake up its message loop. This is typically done when the program has a custom message loop, and it needs some of the non-

message code to run. We saw an example of this some time ago where we posted a `WM_NULL` to let our message loop know that the pseudo-dialog has exited.

Posted `WM_NULL` messages are usually done from a program to itself, and they are usually posted as thread messages, not window messages, so they don't normally come through the window procedure.

Sending a `WM_NULL` is a different story, though. It is a relative common technique to send a `WM_NULL` message to a window for the purpose of checking whether the window is responding to messages. We used it to wait for a window to finish processing a foreground change. Some system monitoring tools will periodically call `SendMessageTimeout` to send a `WM_NULL` to all windows, just to see if they are responding. Windows UI Automation uses `WM_NULL` messages help determine the window interaction state.

The customer could try running system monitoring tools or accessibility tools to increase the likelihood of receiving a `WM_NULL` message under normal use. (I mean, sure, they could write a program that explicitly sends a `WM_NULL` message to their window, but that wouldn't be anything a normal end-user would have.)

I suspect the customer will bump up the priority of this issue due to the accessibility angle. People who use accessibility tools tend to really need them. It's not like you can tell a person with poor visual acuity, "Oh, just suck it up for a while."

Bonus chatter: The customer wrote back. After further investigation, they found that the problem was traced to a third party tool that their client was using, specifically this line of code that sends a `WM_NULL` message to the foreground window to determine whether it is responding.

Raymond Chen

Follow