Why does my window get a WM_ACTIVATE message when it isn't active?

devblogs.microsoft.com/oldnewthing/20131016-00

October 16, 2013

Raymond Chen

Say you launch a program, and for whatever reason the program takes a long time to start up, so you start doing something else, say launching Calculator and balancing your checkbook. Eventually, the program you launched a while back gets itself off the ground and creates its main window. And the window sits in the background (since the window manager won't let it steal foreground activation), but the caret is blinking in the edit control, and the program seems to think it's the active window. If you write a test program to do this, say by sticking a `Sleep(10000)` at the start of your `WinMain`, you'll see that your window gets a `WM_ACTIVATE` message that says "Yup, you're active." But you're not. Or are you? What's going on here? Let's rewind to 16-bit Windows. The active window was the top-level window that receives input. You could make a window active by clicking on it or by selecting it via `Alt`+`Tab` or any number of other operations. A program could change the active window by calling the `SetActiveWindow` function explicitly, or by performing a number of other operations which imply changing the active window. (For example, depending on the parameters you pass, `ShowWindow` and `SetWindowPos` may make the window active as well as showing/repositioning it.) There was only one active window at a time; if a program set itself as the active window, then the previous active window lost activation. Okay, now let's move forward to Win32. Recall that in Win32, most of the state that used to be global became thread-local.[1] This was done to permit the asynchronous input model, where each thread gets its own input queue. This means each thread gets its own mouse cursor show state, each thread gets its own caret, each thread gets its own focus window, and so on. And one of the and-so-on bits is that each thread gets its own active window. What you're seeing is a thread which has an active window which is not the foreground window. The thread also has a focus window, and when an edit control gets the focus, it draws a blinky caret. Mind you, the window is not the *foreground* window, so your input doesn't actually go to it, but the window doesn't know that. It's sitting around as an active window, wondering why nobody is typing anything. So now you know what's going on. Mind you, there's nothing actually wrong with this situation. In fact, it's a sign that the virtualization is doing what it's supposed to do: The thread is living in its own world, a world designed to be compatible with the 16-bit world where there was only one active window. **Footnote**

[1] Or, more nitpickily, local to the input thread group. Since most input thread groups consist of a single thread, I'll just write thread and leave you to insert "or input thread group" as necessary. But you knew that, because it's one of the five things every Win32 programmer should know.

Raymond Chen

Follow