Sharing an input queue takes what used to be asynchronous and makes it synchronous, like focus changes

devblogs.microsoft.com/oldnewthing/20130607-00

June 7, 2013

Raymond Chen

As I noted earlier in the series, attaching input queues puts you back into the world of coöperative multitasking, where the two attached threads need to work together to get anything done.

Back in the old 16-bit days, when input was synchronous, there was only one active window, only one focus window, only one window with capture, only one caret, only one cursor show count, only one keyboard state, only one input state. Furthermore, if you called `SetFocus`, you had to wait until the previous focus window responded to the `WM_KILLFOCUS` message before your `SetFocus` call returned.

With asynchronous input, these sorts of operations are now local to your input queue. If you call `SetFocus`, then that steals focus only from other windows which belong to your input queue. Windows which belong to other input queues are unaffected. (Conversely, you can set focus only to windows which belong to your input queue, since those are the only windows your input queue has access to.)

This is probably not very exciting, until you look at the one thing that can reach across input queues: The foreground window.

The concept of the foreground window was introduced when input was desynchronized in order to express the "really global active window", as opposed to `SetActiveWindow`, which continued to refer to the local active window. It's something that was originally intended to be used only in emergencies since it violates the isolation of input queues, but as we learned before, eventually nothing is special any more, and what used to be the special function for stepping outside the box has become the function you use every day for getting things done.

What most people don't realize is that `SetForegroundWindow` is still subject to the rules on synchronous input. If you call `SetForegroundWindow`, and the previous foreground window also belongs to your input queue, then your call to `SetForegroundWindow` will wait until the previous foreground window processes its `WM_ACTIVATE(WA_INACTIVE)` message.

A lot of people use `AttachThreadInput` thinking that it’s a Get Out of Jail Free card, letting them manipulate windows of other programs and bypass the normal rules for focus and activation. But in fact it’s a *Get Into the Same Jail* card, because you tied your thread’s fate to that other thread. If that thread has stopped responding to messages, then your thread will also stop responding to messages, since you are sharing the same input queue and operations within an input queue are synchronous.

Bonus reminder: If two windows are related by a parent/child relationship or owner/owned relationship, then their input queues are automatically attached. For example, if you do a `SetParent` where the parent and child are in different threads, you have just synchronized the two threads. This sort of cross-thread relationship is technically legal, but it is very difficult to manage correctly, so it should be avoided unless you really know what you’re doing. And if you are doing cross-thread or cross-process between windows that were not designed to participate in cross-thread or cross-process parenting, you are almost certainly doomed.

Raymond Chen

Follow