Replaying input is not the same as reprocessing it

devblogs.microsoft.com/oldnewthing/20121206-00

December 6, 2012

Raymond Chen

Once upon a time, there was an application that received some input and said, "Okay, this input cancels my temporary state. I want to exit my temporary state, but I also want the input that took me out of the temporary state to go to whatever control would have received the input if I hadn't been in the temporary state in the first place." (For example, you might want the input that dismisses a pop-up window to be acted upon rather than eaten by the pop-up.) The application decided to solve this problem by regenerating the input message via `SendInput`, so that it goes back into the input queue. The theory, is that when the message pump pulls the regenerated input out of the queue, the temporary state will not be present, and the message will be routed to the correct window. I raised concerns that this technique would create problems with input reordering and multiple-processing, but the customer decided to stick with their original design. Time passed, and I had forgotten about this application. Some months later, another question came in: "We find that when the system is under load, we sometimes get into a state where dismissing our temporary state results in the mouse button getting 'stuck' down. *i.e.*, the user physically releases the mouse button, but we get spurious `WM_LBUTTONDOWN` with no matching `WM_LBUTTONUP`." The customer, it turns out, was the same one I had cautioned earlier about the dangers of replaying input. When you get input, *that is your chance to process the input*. If you decide you don't want to deal with the input right now and replay it via `SendInput`, you create a few new problems: First, you've caused everybody else who is looking at input states to see a second copy of your replayed events. If it were a keyboard event you replayed, a keyboard hook (or any code which subclassed your window) would see a key go down twice. If there were any mouse hooks, they would see the button go down twice. This is particularly confusing because the mouse button doesn't autorepeat. How can it go Down two times in a row without an intervening Up? Second, if there is other input in your queue, you just rearranged input events. For example, suppose the input queue consists of the following events:

`WM_LBUTTONDOWN`
`WM_LBUTTONUP`

You retrieve the first message (the button-down), resulting in the following input queue:

~~WM_LBUTTONDOWN~~
WM_LBUTTONUP

For illustrative purposes, I crossed out the message that is no longer in the queue, so you can see where it used to be. Now you decide to replay that message via `SendInput`. This appends the event to your queue, resulting in

~~WM_LBUTTONDOWN~~
WM_LBUTTONUP
WM_LBUTTONDOWN

Your message pump runs, it processes the button-up event ("Huh? How did I get an Up without a Down?"), and then it processes the button-down event. There are no further events, so the mouse button is down and gets stuck that way. You can imagine what other sorts of bad things can happen if an event in the queue is, say, a press or release of the shift key. Oops, the user clicked the Delete button and then hit the shift key afterwards to type a capital letter A, but due to your input reordering, your code saw it as a Shift+Click on the Delete button, and the item was deleted without confirmation.

When you get an input message, that is your chance to process it. If you decide that you want to hand the message off to somebody else, you have to do it during the processing of that message. If you try to process it at some other time, the input states may not be right.

Raymond Chen

Follow