The dangers of mixing synchronous and asynchronous state

devblogs.microsoft.com/oldnewthing/20090618-00

June 18, 2009

Raymond Chen

The window manager distinguishes between synchronous state (the state of the world based on what messages your program has received) and asynchronous state (the actual state of the world this very instant). We saw this earlier when discussing the difference between `GetKeyState` and `GetAsyncKeyState`. Here are some other functions and their relationship to the queue state:

Use synchronous state	Use asynchronous state
`GetActiveWindow`	`GetForegroundWindow`
`GetMessagePos`	`GetCursorPos`
`GetMessageTime`	`GetTickCount`

If you query the asynchronous state while processing a message, you can find yourself caught in a race condition, because the synchronous state of the system when the message was generated may not match the asynchronous state of the system when you receive it. For example, if the users presses a key, and then moves the mouse, calling `GetCursorPos` from your keypress handler will tell you were the cursor is right now, which is not the same as where the cursor was when the key was pressed.

Generally speaking, you should use the synchronous state during message handling so that you react to the state of the system at the time the input event took place. Reacting to the asynchronous state of the system introduces race conditions if there is a change to the system state between the time the message was generated and the time the message is processed.

Of the above functions, `GetTickCount` is the only one I can think of that has a legitimate usage pattern in common use, namely, when creating timing loops. But if you want to know what time it was when a key was pressed, then `GetMessageTime` is the function to use.

This is all a rather length lead-in for my remarks regarding a comment claiming that there is no practical reason why you can't use `GetForegroundWindow` to determine which window was the one that had focus when a keyboard message was generated. Well, actually, there is, and it's precisely the race condition I've spent most of this article describing. Suppose the user presses a key and then switches to another program. Now your program gets around to processing the keyboard input, and you call `GetForegroundWindow`, and instead of getting a window from your application, you get some other window from another program. You then pass that window handle to `TranslateAccelerator`, the keyboard event matches an entry in the accelerator, and boom, you just sent a random `WM_COMMAND` message to a program that will interpret it to mean something completely different.

Remember, just because your program has the line

```
#define IDC_REFRESH    814
```

doesn't mean that another program can't have the line

```
#define IDC_DELETEALL  814
```

Now the user presses `F5` and switches from your program to that other program. Your program processes the message, queries the *asynchronous* foreground state with `GetForegroundWindow`, and gets that other program's window back. You then translate the accelerator, and `TranslateAccelerator` posts the `WM_COMMAND(814)` message to that other program, which interprets it as "delete all".

The great thing about this is that the users will probably blame the other program. "Sometimes, when I use this program, it spontaneously deletes all my items. Stupid program. It's so buggy."

Commenter poenits correctly points out that I failed to take into account the case where the message is posted directly to the dialog. (The dialog manager tries not to put keyboard focus on the dialog itself, but if you play weird games, you can find yourself backed into that situation, such as if you delete all the controls on a dialog!) The fix, however, is not to translate the message directly to the window with keyboard focus, because the window with keyboard focus might belong to a *third* dialog that you don't want to translate accelerators for. (That other window might have used the other header file which defines message 814 to be `IDC_DELETEALL`.) Just check for your specific window directly:

```
if (hwnd1== msg.hwnd || IsChild(hwnd1, msg.hwnd))
    TranslateAccelerator(hwnd1, hAccel, &msg);
else if (hwnd2 == msg.hwnd || IsChild(hwnd2, msg.hwnd))
    TranslateAccelerator(hwnd2, hAccel, &msg);
```

Think of `TranslateAccelerator` as `MaybePostWM_COMMAND`. The first parameter to `TranslateAccelerator` must be a window you are certain knows how to interpret the `WM_COMMAND` message that you might end up posting. You know which windows understand your custom `WM_COMMAND` messages. Pass one of those known windows, not some random unknown window that you calculated from unknown sources.

Passing an unknown window as the first parameter to `TranslateAccelerator` is like falling for one of those phishing scams. If you get a random piece of email telling you "Hey, call this number and give me your personal information," you're not going to do it. If you really want to contact your bank, you ignore the phone number in the email and just call the number you know and trust to be your bank's service desk. Similarly, you shouldn't be posting your personal messages to some random window you receive. Post it to the known trusted window. Otherwise you're just sending your money to some unknown recipient in Nigeria.

Raymond Chen

Follow