Posted messages are processed ahead of input messages, even if they were posted later

devblogs.microsoft.com/oldnewthing/20130531-00

May 31, 2013

Raymond Chen

Regardless of which interpretation you use, it remains the case that posted messages are processed ahead of input messages. Under the MSDN interpretation, posted messages and input messages all go into the *message queue*, but posted messages are pulled from the queue before input messages. Under the Raymond interpretation, posted messages and input messages are kept in separate queues, and the message retrieval functions will look first in the posted message queue before looking in the input queue.

Let's run an experiment to see posted messages get processed ahead of input messages. Start with the new scratch program and make these changes:

```
#include <strsafe.h>


class RootWindow : public Window
{
public:
 virtual LPCTSTR ClassName() { return TEXT(“Scratch”); }
 static RootWindow *Create();


 void AppendText(LPCTSTR psz)
 {
    ListBox_SetCurSel(m_hwndChild,
                      ListBox_AddString(m_hwndChild, psz));
 }


 void AppendFormat(LPCTSTR pszFormat, …)
 {
  va_list ap;
  va_start(ap, pszFormat);
  TCHAR szMsg[256];
  StringCchVPrintf(szMsg, ARRAYSIZE(szMsg), pszFormat, ap);
  AppendText(szMsg);
  va_end(ap);
 }


 void LogMessage(const MSG *pmsg)
 {
   AppendFormat(TEXT(“%d\t%04x\t%p\t%p”),
                pmsg->time,
                pmsg->message,
                pmsg->wParam,
                pmsg->lParam);
 }


 …
};


LRESULT RootWindow::OnCreate()
{
 m_hwndChild = CreateWindow(
      TEXT(“listbox”), NULL,
      LBS_HASSTRINGS | LBS_USETABSTOPS |
      WS_CHILD | WS_VISIBLE | WS_TABSTOP | WS_VSCROLL,
      0, 0, 0,0, GetHWND(), (HMENU)1, g_hinst, 0);
 return 0;
}
```

```
int WINAPI WinMain(HINSTANCE hinst, HINSTANCE hinstPrev,
                   LPSTR lpCmdLine, int nShowCmd)
{
   …
   while (GetMessage(&msg, NULL, 0, 0)) {


    switch (msg.message) {
    case WM_KEYDOWN:
     prw->AppendText(TEXT("Sleeping"));
     UpdateWindow(prw->GetHWND());
     Sleep(1000);
     prw->AppendText(TEXT("Posting"));
     PostMessage(prw->GetHWND(), WM_USER, 0, 0);
     break;
    case WM_KEYUP:
    case WM_USER:
     prw->LogMessage(&msg);
     break;
    }


    TranslateMessage(&msg);
    DispatchMessage(&msg);
   …
}
```

This program creates a list box so we can display some output. In the message loop, it sniffs at all the queued messages and does the following:

- If the message is `WM_KEYUP` or `WM_USER`, then it logs the message timestamp and some parameters.
- If the message is `WM_KEYDOWN`, then it sleeps without processing messages for one second, and then posts a `WM_USER` message to the main window (which ignores it).

Run this program, and then tap the shift key.

The window gets a `WM_KEYDOWN` for the shift key. It sleeps for one second (plenty of time for you to release the shift key), and then posts a `WM_USER` message.

The `WM_USER` and `WM_KEYUP` messages arrive, and observe via the log window that they arrive *out of order*. `WM_USER` message arrived first!

That's because of the rule that says that posted messages are processed ahead of input messages. (Depending on how you want to look at it, you might say that posted messages are "called out for preferential treatment" in the queue, or you might say that posted messages are placed in a different queue from input messages, and the posted message queue has higher priority.)

Observe also that the timestamp on the `WM_USER` message is *greater than* the timestamp on the `WM_KEYUP` message, because the key went up before the `WM_USER` message was posted. *Time has gone backward.*

Make the following change to our program: Change the message we post from `WM_USER` to `WM_KEYUP`:

```
        PostMessage(hwnd, WM_KEYUP, 0, 0);
```

Run the program again, and again tap the shift key. Observe that the posted `WM_KEYUP` message is processed ahead of the `WM_KEYUP` input message. (You can see the difference because we posted the `WM_KEYUP` message with `wParam` and `lParam` both zero, whereas the `WM_KEYUP` input message has information in those parameters.)

This little demonstration also reinforces some other things we already knew. For example, it once again shows that the input manager does not wiretap your posted messages. If you post a `WM_KEYUP` message, it is treated like a posted message not an input message. We saw earlier that posting a keyboard message does not update internal input states. The keyboard shift states are not updated to match your prank call message. If somebody calls `GetQueue-Status`, they will not be told that there is input waiting. It will not wake a `MsgWaitFor-MultipleObjects` function that is waiting for `QS_INPUT`. And as we saw here today, the message gets processed out of order.

Raymond Chen

Follow