Modality, part 8: A timed MessageBox, the better version

devblogs.microsoft.com/oldnewthing/20050304-00

March 4, 2005

Raymond Chen

A few days ago, we saw a simple version of a timed message box which had a limitation that it could be used from only one thread at a time. Today we'll work to remove that limitation.

As you may recall, the reason why it could be used from only one thread at a time was that we kept the "Did the message box time out?" flag in a global. To fix it, we will move the flag to a per-instance location, namely a helper window.

Start with the scratch program, add the code for the scratch window class, *change the name of the scratch window class* so it doesn't conflict with the class name of the scratch program (thanks to reader Adrian for pointing this out), then add the following:

```
#define IDT_TOOLATE     1
typedef struct TOOLATEINFO {
 BOOL fTimedOut;
 HWND hwndReenable;
} TOOLATEINFO;
void CALLBACK
MsgBoxTooLateProc(HWND hwnd, UINT uiMsg, UINT_PTR idEvent, DWORD dwTime)
{
  TOOLATEINFO *ptli = reinterpret_cast<TOOLATEINFO*>(
    GetWindowLongPtr(hwnd, GWLP_USERDATA));
  if (ptli) {
    ptli->fTimedOut = TRUE;
    if (ptli->hwndReenable) {
       EnableWindow(ptli->hwndReenable, TRUE);
    }
    PostQuitMessage(42);
  }
}
int TimedMessageBox(HWND hwndOwner, LPCTSTR ptszText,
    LPCTSTR ptszCaption, UINT uType, DWORD dwTimeout)
{
  TOOLATEINFO tli;
  tli.fTimedOut = FALSE;
  BOOL fWasEnabled = hwndOwner && IsWindowEnabled(hwndOwner);
  tli.hwndReenable = fWasEnabled ? hwndOwner : NULL;
  HWND hwndScratch = CreateScratchWindow(hwndOwner, DefWindowProc);
  if (hwndScratch) {
      SetWindowLongPtr(hwndScratch, GWLP_USERDATA,
                       reinterpret_cast<LPARAM>(&tli));
      SetTimer(hwndScratch, IDT_TOOLATE, dwTimeout, MsgBoxTooLateProc);
  }
  int iResult = MessageBox(hwndOwner, ptszText, ptszCaption, uType);
  if (hwndScratch) {
    KillTimer(hwndScratch, IDT_TOOLATE);
    if (tli.fTimedOut) { // We timed out
      MSG msg;
      // Eat the fake WM_QUIT message we generated
      PeekMessage(&msg, NULL, WM_QUIT, WM_QUIT, PM_REMOVE);
      iResult = -1;
    }
    DestroyWindow(hwndScratch);
  }
  return iResult;
}
void OnChar(HWND hwnd, TCHAR ch, int cRepeat)
{
  switch (ch) {
  case ' ':
    TimedMessageBox(hwnd, TEXT("text"), TEXT("caption"),
                    MB_OK, 2000);
    break;
  }
```

```
}
// add to WndProc
    HANDLE_MSG(hwnd, WM_CHAR, OnChar);
// add to InitApp
    RegisterScratchWindowClass();
```

This is basically the same as the previous cheap version, just with slightly different bookkeeping.

The state of the timed message box is kept in the structure `TOOLATEINFO`. But how to pass this state to the timer callback? You can't pass any parameters to timer callbacks.

Aha, but timer callbacks do get a window handle. But as we discovered a few days ago, we can't just hang the callback off the `hwndOwner` window because we don't know how to pick a timer ID that doesn't conflict with an existing one.

The solution: Hang it on a window of our own window creation. That way, we get a whole new space of timer IDs to play in, separate from the timer IDs that belong to `hwndOwner`. The scratch window is a convenient window to use. We don't pass an interesting window procedure to `CreateScratchWindow` because there is no need; all we wanted was a window to own our timer.

Raymond Chen

Follow