The dangers of buffering up posted messages and then reposting them later

devblogs.microsoft.com/oldnewthing/20140328-00

March 28, 2014

Raymond Chen

A customer wanted to check that their idea for solving a re-entrancy problem doesn't have any hidden gotchas.

> We have a window which processes incoming work. One of our work items enters a modal loop, and if new work gets posted to the window while the modal loop is running, our work manager gets re-entered, and Bad Things happen.
>
> Our proposed solution is to alter the modal loop so that it buffers up all messages destined for the worker window. (Messages for any other window are dispatched normally.) When the modal loop completes, we re-post all the messages from the buffer, thereby allowing the worker window to resume processing.

The danger here is that reposting messages can result in messages being processed out of order. Depending on how your worker window is designed, this might or might not be a problem. For example, suppose that during the modal operation, somebody posts the `WWM_FOOSTARTED` message to the worker window. You buffer it up. When your modal operation is complete, you are about to post the message back into the queue, but another thread races against you and posts the `WWM_FOOCOMPLETED` message before you can post your buffered messages back into the queue. Result: The worker window receives the `WWM_FOOCOMPLETED` message before it receives the `WWM_FOOSTARTED` message. This will probably lead to confusion.

The place to solve this problem is in the window itself. That gets rid of the race condition.

```
LRESULT CALLBACK WorkerWindow::WndProc(
    HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
 if (uMsg is a work message) {
  if (m_cBusy) {
   // Now is a bad time to process the work message.
   // Queue it up for later.
   m_queue.Append(uMsg, wParam, lParam);
  } else {
   m_cBusy++; // prevent re-entrancy
   do {
    ProcessWorkMessage(uMsg, wParam, lParam);
   } while (m_queue.RemoveFirst(&uMsg, &wParam, &lParam));
   m_cBusy--; // re-entrancy no longer a problem
  }
  return 0;
 }
 ...  // handle the other messages
}
```

By queueing up the work inside the window itself, you ensure that the messages are processed in the same order they were received.

This technique can be extended to, say, have the worker window do some degree of work throttling. For example, you might keep track of how long you've been processing work, and if it's been a long time, then stop to pump messages for a while in case any system messages came in, and somebody is waiting for your answer.

Raymond Chen

Follow