The case of the recursively hung WM_DRAWCLIPBOARD message

devblogs.microsoft.com/oldnewthing/20221223-00

December 23, 2022

Raymond Chen

An application hang report showed that the application was stuck in this stack:

```
win32u!ZwUserMessageCall+0x14
user32!SendMessageWorker+0x823
user32!SendMessageW+0xda
contoso!CContosoWindow::WndProc+0xa5d
user32!UserCallWinProcCheckWow+0x2f8
user32!DispatchClientMessage+0x9c
user32!__fnDWORD+0x33
ntdll!KiUserCallbackDispatcherContinue
win32u!ZwUserMessageCall+0x14
user32!SendMessageWorker+0x823
user32!SendMessageW+0xda
contoso!CContosoWindow::WndProc+0xa5d
user32!UserCallWinProcCheckWow+0x2f8
user32!DispatchClientMessage+0x9c
user32!__fnDWORD+0x33
ntdll!KiUserCallbackDispatcherContinue
win32u!ZwUserMessageCall+0x14
user32!SendMessageWorker+0x823
user32!SendMessageW+0xda
contoso!CContosoWindow::WndProc+0xa5d
user32!UserCallWinProcCheckWow+0x2f8
user32!DispatchClientMessage+0x9c
user32!__fnDWORD+0x33
ntdll!KiUserCallbackDispatcherContinue
win32u!ZwUserMessageCall+0x14
user32!SendMessageWorker+0x823
user32!SendMessageW+0xda
contoso!CContosoWindow::WndProc+0xa5d
user32!UserCallWinProcCheckWow+0x2f8
user32!DispatchClientMessage+0x9c
user32!__fnDWORD+0x33
ntdll!KiUserCallbackDispatcherContinue
win32u!ZwUserMessageCall+0x14
user32!SendMessageWorker+0x823
user32!SendMessageW+0xda
contoso!CContosoWindow::WndProc+0xa5d
user32!UserCallWinProcCheckWow+0x2f8
user32!DispatchClientMessage+0x9c
user32!__fnDWORD+0x33
ntdll!KiUserCallbackDispatcherContinue
win32u!ZwUserMessageCall+0x14
user32!SendMessageWorker+0x823
user32!SendMessageW+0xda
contoso!CContosoWindow::WndProc+0xa5d
user32!UserCallWinProcCheckWow+0x2f8
user32!DispatchClientMessage+0x9c
user32!__fnDWORD+0x33
ntdll!KiUserCallbackDispatcherContinue
win32u!ZwUserMessageCall+0x14
user32!SendMessageWorker+0x823
user32!SendMessageW+0xda
contoso!CContosoWindow::WndProc+0xa5d
```

```
user32!UserCallWinProcCheckWow+0x2f8
user32!DispatchClientMessage+0x9c
user32!__fnDWORD+0x33
ntdll!KiUserCallbackDispatcherContinue
win32u!ZwUserMessageCall+0x14
user32!SendMessageWorker+0x823
user32!SendMessageW+0xda
contoso!CContosoWindow::WndProc+0xa5d
user32!UserCallWinProcCheckWow+0x2f8
user32!DispatchClientMessage+0x9c
user32!__fnDWORD+0x33
ntdll!KiUserCallbackDispatcherContinue
win32u!ZwUserMessageCall+0x14
user32!SendMessageWorker+0x823
user32!SendMessageW+0xda
contoso!CContosoWindow::WndProc+0xa5d
user32!UserCallWinProcCheckWow+0x2f8
user32!DispatchClientMessage+0x9c
user32!__fnDWORD+0x33
ntdll!KiUserCallbackDispatcherContinue
win32u!ZwUserMessageCall+0x14
user32!SendMessageWorker+0x823
user32!SendMessageW+0xda
contoso!CContosoWindow::WndProc+0xa5d
user32!UserCallWinProcCheckWow+0x2f8
user32!DispatchClientMessage+0x9c
user32!__fnDWORD+0x33
ntdll!KiUserCallbackDispatcherContinue
user32!_InternalCallWinProc+0x2a
user32!InternalCallWinProc+0x1b
user32!DispatchClientMessage+0xea
user32!__fnDWORD+0x3f
ntdll!KiUserCallbackDispatcher+0x4c
win32u!NtUserGetMessage+0xc
user32!GetMessageW+0x30
contoso!WindowThreadProc+0x9b
kernel32!BaseThreadInitThunk+0x14
ntdll!RtlUserThreadStart+0x21
```

Inspecting the local variables at each recursive call shows that the message is always `WM_DRAWCLIPBOARD`. The Contoso window receives the `WM_DRAWCLIPBOARD` message, does its work, and then forwards the message to the next clipboard viewer window, just like the book says. While waiting for that window to respond, another `WM_DRAWCLIPBOARD` message arrives, and the cycle repeats.

The clipboard viewer chain is a linked list of windows that have all subscribed to clipboard notifications. This linked list is managed cooperatively: When you add yourself to the chain, you are given the handle of the previous head of the chain. And when you finish dealing with a clipboard notification, you forward the notification to the next window in the chain. That way, all the windows in the chain eventually learn about the clipboard.

The clipboard viewer chain was developed back in the days of 16-bit Windows, when all programs were cooperatively multi-tasked and generally were trusted to behave properly. The clipboard viewer chain used the same trick that window hooks used to save space: It externalized the cost.

Here's a sketch of how the clipboard viewer chain worked in 16-bit Windows:

```
HWND hwndClipboardViewer;

HWND SetClipboardViewer(HWND hwndNewViewer)
{
  HWND hwndOldViewer = hwndClipboardViewer;
  hwndClipboardViewer = hwndNewViewer;
  return hwndOldViewer;
}

HWND GetClipboardViewer()
{
  return hwndClipboardViewer;
}

HWND ChangeClipboardChain(HWND hwndRemove, HWND hwndNewNext)
{
  if (hwndClipboardViewer == hwndRemove) {
    hwndClipboardViewer = hwndNewNext;
  } else {
    SendMessage(hwndClipboardViewer, WM_CHANGECBCHAIN,
        (WPARAM)hwndRemove, (LPARAM)hwndNewNext);
  }
}

void NotifyClipboardViewers()
{
  if (hwndClipboardViewer) {
    SendMessage(hwndClipboardViewer, WM_DRAWCLIPBOARD, 0, 0);
  }
}
```

And that's it! The entire clipboard viewer feature in 30 lines of code.

Okay, so back to our customer's problem.

The window registered itself as a clipboard viewer, and the clipboard contents changed, causing it to receive a `WM_DRAWCLIPBOARD` message. The window dealt with the clipboard change, and then dutifully called `SendMessage` to forward the `WM_DRAWCLIPBOARD` message down the chain. Every window in the chain deals with the message, and then calls `SendMessage`.

What happened here is that some window in the chain is hung, and that causes all the other windows in the chain to hang, since they are all blocked on each other via `SendMessage`:

Window 1	
↓	SendMessage
Window 2	
↓	SendMessage
Window 3	
↓	SendMessage
Window 4	hung

In order for Window 1's `SendMessage` to complete, Window 2 needs to return. But Window 2 is stuck in a `SendMessage` to Window 3, which is in turn stock in a `SendMessage` to Window 4, which is hung. That one hung window has caused a chain of windows to stop responding.

The Contoso window got caught in the chain of windows that are all waiting for that other hung window to process the `WM_DRAWCLIPBOARD` message.

So what can Contoso do about this?

The best solution is to leave the game. Instead of using the old and busted clipboard viewer chain, use the new hotness `AddClipboardFormatListener` function to register to be notified when the clipboard contents change and escape the clipboard viewer chain.

Fortunately, converting from a clipboard viewer to a clipboard format listener is fairly simple and even involves deleting some code, so that's a nice bonus.

- Change `SetClipboardViewer` to `AddClipboardFormatListener`.
- Delete the variable that held the previous clipboard viewer.
- Delete the code that handled the `WM_CHANGECBCHAIN` message.
- Change `case WM_DRAWCLIPBOARD` to `case WM_CLIPBOARDUPDATE`.
- Delete the `SendMessage(hwndNextViewer, WM_DRAWCLIPBOARD, wParam, lParam)`.
- Change `ChangeClipboardChain` to `RemoveClipboardFormatListener`.

If for some reason you really want to be a clipboard viewer, you can at least switch to using `SendNotifyMessage` to forward the `WM_DRAWCLIPBOARD` message to the next window in the chain. The `SendNotifyMessage` function is like `SendMessage` except that it doesn't

want for the recipient to return. It's a fire-and-forget `SendMessage`.

Raymond Chen

Follow