Non-psychic debugging: Looking for leaked objects by their vtable

devblogs.microsoft.com/oldnewthing/20140124-00

January 24, 2014

Raymond Chen

A programmer on the GHI team reported that they were hitting an assertion failure using an internal library and asked for help debugging it.

```
ABC!CFactoryBase::AssertZeroRef+0x2e
ABC!CFactoryBase::Unregister+0x1c
ABC!CWidgetFactory::Unregister+0x2d
DEF!CWidgetManager::Uninitialize+0x4a
DEF!CWidget::`scalar deleting destructor'+0xd
ABC!operator delete()+0x6
ABC!CFrame::Destroy+0xb
ABC!CFrame::WndProc+0x362
USER32!InternalCallWinProc+0x23
USER32!UserCallWinProcCheckWow+0x121
USER32!DispatchClientMessage+0x12d
USER32!__fnEMPTY+0x24
ntdll!KiUserCallbackDispatcher+0x3d
USER32!NtUserDestroyWindow+0xa
ABC!CFrame::WndProc+0x34
USER32!InternalCallWinProc+0x23
USER32!UserCallWinProcCheckWow+0x121
USER32!DispatchMessageWorker+0x411
USER32!DispatchMessageW+0x10
GHI!wWinMain+0xa0
GHI!__wmainCRTStartup+0x153
KERNEL32!BaseThreadInitThunk+0xe
ntdll!__RtlUserThreadStart+0x23
ntdll!_RtlUserThreadStart+0x1b
```

I didn't work on this internal library, but on the other hand I'm also not afraid to look inside and around.

The assertion failure said, "Assertion failed: All widgets from a factory must be destroyed before you can unregister the factory."

The factory does not keep a list of all the widgets it created. It merely keeps a count and asserts that the count is zero when the factory is unregistered."

A good start would be to find the widgets that are still outstanding, so we can try to figure out why they weren't destroyed.

```
0:000> u ABC!CWidget::CWidget
...
1071158b mov     dword ptr [esi],offset ABC!CWidget::`vftable' (106da08c)
```

This gives us the widget vtable, so a memory scan should find all the outstanding widgets.

```
0:000> !heap -search 106da08c
    _HEAP @ 950000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        01eb12d8 000e 0000  [00]   01eb12e0    00064 - (busy)
          ABC!CWidget::`vftable'
```

Okay, so a search of the heap shows that there is only one widget, and it is at `0x01eb12e0`. Let's see what that widget can tell us about who it is.

```
0:000> dt ABC!CWidget 01eb12e0
   +0x000 __VFN_table : 0x106da08c
   +0x004 m_uBucketId      : 2
   +0x008 m_rgClassData    :
   +0x050 m_rgSharedData   :
   +0x05c m_fLocked        : 1
   +0x060 m_pszName        : 0x01eba4c0  "GHI_widget"
```

Hey, how about that. The widget conveniently has the name `GHI_widget`, which seems like a pretty good sign that the GHI component leaked a widget.

Notice that I didn't use any special knowledge of Widgets, Widget Factories, the ABC component, or the GHI component. All I did was take the error message that said, "You leaked a widget" and said, "Maybe I should go look for that widget. That may tell us something." I disassembled the widget constructor to look for a unique tag common to all widgets, and then scanned memory looking for that vtable. From the found object, I dumped its member variables looking for some sort of clue as to its identity, and by an amazing stroke of luck, the widget had a name.

Back in my trainee days in tech support, if a customer asked a question that we couldn't answer, we escalated the problem to the next higher level and were encouraged to tag along and learn from the subject matter expert. That way, when the problem came up again, we could solve it ourselves.

In other words, we were encouraged not to run away from information, but to run *toward* it.

(It helped that we weren't graded on "number of cases closed per second.")

One of the most important skills in a programmer is the willingness to look at code that you didn’t write. When I joined Microsoft, this instinct to run toward information led me to watch as somebody else debugged a problem and learn from them. I would then go back and read the code that they debugged to see how much of it I could understand. And if I ran into a problem of my own, I dove in and read the source code to the component that was giving me trouble, even if it was not a component I remotely had any responsibility for. Maybe I could figure out what it was doing, maybe I couldn’t, but at least I gave it a try. And when I went to another developer with my theory, I was told either that my understanding was correct, or that I had gotten it wrong and was told the correct answer. Either way, I learned a little bit more that day.

Exercise: If the widget had not had a name, what would be a reasonable next step in the investigation?

Raymond Chen

Follow