The case of the hung Explorer window

devblogs.microsoft.com/oldnewthing/20160805-00

August 5, 2016

Raymond Chen

A Windows Insider reported that Explorer stopped responding whenever they opened their Downloads folder.

We were able to obtain a memory dump during the hang, and observed that most threads were waiting for the loader lock. The loader lock was being held by this thread:

```
ntdll!RtlpWaitOnCriticalSection
ntdll!RtlpEnterCriticalSectionContended
GdiPlus!GdiplusStartupCriticalSection::{ctor}
GdiPlus!GdiplusStartup
ShellExtension+...
ShellExtension+...
ShellExtension+...
ntdll!LdrpCallInitRoutine
ntdll!LdrpInitializeNode
ntdll!LdrpInitializeGraphRecurse
ntdll!LdrpInitializeGraph
ntdll!LdrpPrepareModuleForExecution
ntdll!LdrpLoadDllInternal
ntdll!LdrpLoadDll
ntdll!LdrLoadDll
KERNELBASE!LoadLibraryExW
[...]
combase!CoCreateInstanceEx
combase!CoCreateInstance
windows_storage!_SHCoCreateInstance
windows_storage!CRegFolder::_CreateCachedRegFolder
windows_storage!CRegFolder::_CreateCachedRegFolder
windows_storage!CRegFolder::_BindToItem
windows_storage!CRegFolder::BindToObject
windows_storage!CShellItem::_BindToHandlerLegacy
windows_storage!CShellItem::BindToHandler
[...]
explorerframe!CNscEnumTask::InternalResumeRT
explorerframe!CRunnableTask::Run
```

This thread was waiting on a GDI+ critical section, which was being held here:

```
KERNELBASE!WaitForSingleObjectEx
GdiPlus!BackgroundThreadShutdown
GdiPlus!InternalGdiplusShutdown
GdiPlus!GdiplusShutdown
shell32!CGraphicsInit::~CGraphicsInit
shell32!CImageFactory::{dtor}
shell32!CImageFactory::`scalar deleting destructor'
shell32!CImageFactory::Release
shell32!IsImageSizeSufficientForRequestedSize
shell32!_ExtactIconFromImage
shell32!_ExtractIconsFromImage
shell32!ExtractIconsUsingResourceManager
shell32!_ExtractIcons
shell32!SHDefExtractIconW
[...]
windows_storage!CLoadSystemIconTask::InternalResumeRT
windows_storage!CRunnableTask::Run
windows_storage!CShellTask::TT_Run
windows_storage!CShellTaskThread::ThreadProc
windows_storage!CShellTaskThread::s_ThreadProc
```

It should now be clear what the problem is.

On the second thread, GDI+ is shutting down because its last client decided to uninitialize it. (In this case, the last client was the system image list, which extracting the icon for a Store app, and Store app icons are PNG files, which is why GDI+ entered the picture.)

GDI+ is waiting for its worker thread to exit so it can finish cleaning up.

Just at this moment, the folder tree was populating itself on the first thread, and it found a third party shell extension. It dutifully loaded the third party shell extension (because that's what shell extensions are for), and that shell extension, as part of its `DLL_PROCESS_ATTACH` tried to initialize GDI+.

Here comes the deadlock.

GDI+ was prepared for this possibility that somebody would try to initialize GDI+ while GDI+ was already in the process of shutting itself down. It solves this problem by making the shutdown run to completion (seeing as it already started), and then starting a new initialization pass.

That shutdown is waiting for a worker thread to finish up and exit. But the thread cannot exit until it sends out its `DLL_THREAD_DETACH` notifications. And since DLL notifications are serialized, the `DLL_THREAD_DETACH` cannot be sent until the `DLL_PROCESS_ATTACH` completes. But the `DLL_PROCESS_ATTACH` for the third party shell extension is waiting for GDI+. There's our deadlock.

The root cause for this is that the third party shell extension is initializing GDI+ inside its `DLL_PROCESS_ATTACH`. This is already highly suspect even without any special insight into GDI+, and the suspicious are confirmed in the documentation for `GdiplusStartup`:

> Do not call **GdiplusStartup** or **GdiplusShutdown** in `DllMain` or in any function that is called by DllMain.

My guess is that the vendor who wrote this shell extension thinks that the rule doesn't apply to them because they passed `SuppressBackgroundThread = true`, thinking that by removing the background thread, they successfully avoided any deadlocks with another thread. It didn't occur to them that the other thread might not be the GDI+ background thread.

It also didn't occur to them that GDI+ might *already be initialized* with a background thread. Furthermore, suppose the component that initialized GDI+ first (with a background thread) uninitialized GDI+ first. That call to `GdiplusShutdown` will not shut down GDI+ because there is still an outstanding client. And then when their DLL unloads, they call `GdiplusShutdown`, and that will cause a true shutdown of GDI+, which includes shutting down that background thread that they thought they had suppressed.[1]

So basically it was a bad idea all around.

I transferred this issue to the application compatibility team for outreach to the vendor, who happens to be a major corporation, so hopefully they can spare some developers to fix the deadlock.

Bonus chatter: Identifying the vendor was a bit tricky because of the extremely vague DLL name.

Bonus chatter: When I originally composed the email with my analysis of the bug, I wrote *application compatibility outrage* instead of *application compatibility outreach*. Unfortunately, I caught the mistake before hitting Send.

[1]Closer investigation shows that my guess was incorrect. The code that calls `GdiplusStartup` leaves the background thread enabled, so I have no idea how this ever worked in isolation. It "works" only because the calls to `GdiplusStartup` and `GdiplusShutdown` are no-op because somebody else initialized GDI+ first, and is still using GDI+ at the time they unload.

Raymond Chen

Follow