Debugging a hang: Chasing the wait chain inside a process

devblogs.microsoft.com/oldnewthing/20141024-00

October 24, 2014

Raymond Chen

Today we’re going to debug a hang. Here are some of the (redacted) stacks of the process. I left some red herrings and other frustrations.

```
0: kd> !process ffffe000045ef940 7
PROCESS ffffe000045ef940
    SessionId: 1  Cid: 0a50    Peb: 7ff6b661f000  ParentCid: 0a0c
    DirBase: 12e5c6000  ObjectTable: ffffc0000288ae80  HandleCount: 1742.
    Image: contoso.exe
        THREAD ffffe000018d68c0  Cid 0a50.0a54  Teb: 00007ff6b661d000 Win32Thread:
fffff90143635a90 WAIT: (WrUserRequest) UserMode Non-Alertable
            ffffe000046192c0  SynchronizationEvent
        nt!KiSwapContext+0x76
        nt!KiSwapThread+0x14c
        nt!KiCommitThreadWait+0x126
        nt!KeWaitForSingleObject+0x1cc
        nt!KeWaitForMultipleObjects+0x44e
        0xfffff960`0038bed0
        0x1
        0xffffd000`24257b80
        0xfffff901`43635a90
        0xd
        0xffffe000`00000001
        0xfffff803`ffffff00
        THREAD ffffe000045f88c0  Cid 0a50.0a8c  Teb: 00007ff6b64ea000 Win32Thread:
0000000000000000 WAIT: (UserRequest) UserMode Non-Alertable
            ffffe000041c1830  SynchronizationEvent
        nt!KiSwapContext+0x76
        nt!KiSwapThread+0x14c
        nt!KiCommitThreadWait+0x126
        nt!KeWaitForSingleObject+0x1cc
        nt!NtWaitForSingleObject+0xb1
        nt!KiSystemServiceCopyEnd+0x13 (TrapFrame @ ffffd000`248ebc40)
        ntdll!ZwWaitForSingleObject+0xa
        ntdll!RtlpWaitOnCriticalSection+0xe1
        ntdll!RtlEnterCriticalSection+0x94
        ntdll!LdrpAcquireLoaderLock+0x2c
        ntdll!LdrShutdownThread+0x64
        ntdll!RtlExitUserThread+0x3e
        KERNELBASE!FreeLibraryAndExitThread+0x4c
        combase!CRpcThreadCache::RpcWorkerThreadEntry+0x62
        KERNEL32!BaseThreadInitThunk+0x30
        ntdll!RtlUserThreadStart+0x42
        THREAD ffffe00003c46080  Cid 0a50.0a9c  Teb: 00007ff6b64e6000 Win32Thread:
fffff90143713a90 WAIT: (UserRequest) UserMode Non-Alertable
            ffffe000041c1830  SynchronizationEvent
        nt!KiSwapContext+0x76
        nt!KiSwapThread+0x14c
        nt!KiCommitThreadWait+0x126
        nt!KeWaitForSingleObject+0x1cc
        nt!NtWaitForSingleObject+0xb1
        nt!KiSystemServiceCopyEnd+0x13 (TrapFrame @ ffffd000`367ece40)
        ntdll!ZwWaitForSingleObject+0xa
        ntdll!RtlpWaitOnCriticalSection+0xe1
        ntdll!RtlEnterCriticalSection+0x94
        ntdll!LdrpAcquireLoaderLock+0x4c
        ntdll!LdrpFindOrMapDll+0x75d
        ntdll!LdrpLoadDll+0x394
        ntdll!LdrLoadDll+0xc6
        kernelbase!LoadLibraryExW+0x142
        kernelbase!LoadLibraryExA+0x26
        contoso!__delayLoadHelper2+0x2b
        contoso!_tailMerge_Winmm_dll+0x3f
        contoso!PolarityReverser::OnCompleted+0x28
        contoso!PolarityReverser::Reverse+0xf4
        contoso!ListItem::ReversePolarity+0x7e
        contoso!View::OnContextMenu+0x8
        contoso!View::WndProc+0x25e
        user32!UserCallWinProcCheckWow+0x13a
        user32!DispatchClientMessage+0xf8
        user32!__fnEMPTY+0x2d
        ntdll!KiUserCallbackDispatcherContinue
        user32!ZwUserMessageCall+0xa
        user32!RealDefWindowProcWorker+0x1e2
        user32!RealDefWindowProcW+0x52
        uxtheme!_ThemeDefWindowProc+0x33e
        uxtheme!ThemeDefWindowProcW+0x11
        user32!DefWindowProcW+0x1b6
        comctl32!CListView::WndProc+0x25e
        comctl32!CListView::s_WndProc+0x52
        user32!UserCallWinProcCheckWow+0x13a
        user32!SendMessageWorker+0xa72
        user32!SendMessageW+0x10a
        comctl32!CLVMouseManager::HandleMouse+0xd10
        comctl32!CLVMouseManager::OnButtonDown+0x27
        comctl32!CListView::WndProc+0x1a4186
        comctl32!CListView::s_WndProc+0x52
        user32!UserCallWinProcCheckWow+0x13a
        user32!DispatchMessageWorker+0x1a7
        THREAD ffffe0000462b8c0  Cid 0a50.0ac0  Teb: 00007ff6b64dc000 Win32Thread:
0000000000000000 WAIT: (UserRequest) UserMode Non-Alertable
            ffffe0000462c980  NotificationEvent
        nt!KiSwapContext+0x76
        nt!KiSwapThread+0x14c
        nt!KiCommitThreadWait+0x126
        nt!KeWaitForSingleObject+0x1cc
        nt!NtWaitForSingleObject+0xb1
        nt!KiSystemServiceCopyEnd+0x13 (TrapFrame @ ffffd000`201e9c40)
        ntdll!ZwWaitForSingleObject+0xa
        KERNELBASE!WaitForSingleObjectEx+0xa5
        contoso!CNetworkManager::ThreadProc+0x94
        KERNEL32!BaseThreadInitThunk+0x30
        ntdll!RtlUserThreadStart+0x42
        THREAD ffffe000046ad340  Cid 0a50.0b38  Teb: 00007ff6b64b6000 Win32Thread:
0000000000000000 WAIT: (UserRequest) UserMode Non-Alertable
            ffffe000049108c0  Thread
        nt!KiSwapContext+0x76
        nt!KiSwapThread+0x14c
        nt!KiCommitThreadWait+0x126
        nt!KeWaitForSingleObject+0x1cc
        nt!NtWaitForSingleObject+0xb1
        nt!KiSystemServiceCopyEnd+0x13 (TrapFrame @ ffffd000`2563bc40)
        ntdll!ZwWaitForSingleObject+0xa
        KERNELBASE!WaitForSingleObjectEx+0xa5
        litware!CDiscovery::Uninitialize+0x8c
        litware!CApiInstance::~CApiInstance+0x48
        litware!CApiInstance::`scalar deleting destructor'+0x14
        litware!std::tr1::_Ref_count_obj<CApiInstance>::_Destroy+0x31
        litware!std::tr1::_Ref_count_base::_Decref+0x1b
        litware!std::tr1::_Ptr_base<CApiInstance>::_Decref+0x20
        litware!std::tr1::shared_ptr<CApiInstance>::{dtor}+0x20
        litware!std::tr1::shared_ptr<CApiInstance>::reset+0x3c
        litware!CSingleton<CApiInstance>::ReleaseRef+0x97
        litware!LitWareUninitialize+0xed
        fabrikam!CDoodadHelper::~CDoodadHelper+0x67
        fabrikam!_CRT_INIT+0xda
        fabrikam!__DllMainCRTStartup+0x1e5
        ntdll!LdrpCallInitRoutine+0x57
        ntdll!LdrpProcessDetachNode+0xfe
        ntdll!LdrpUnloadNode+0x77
        ntdll!LdrpDecrementNodeLoadCount+0xd0
        ntdll!LdrUnloadDll+0x34
        KERNELBASE!FreeLibrary+0x22
        combase!CClassCache::CDllPathEntry::CFinishObject::Finish+0x28
        combase!CClassCache::CFinishComposite::Finish+0x80
        combase!CClassCache::FreeUnused+0xda
        combase!CoFreeUnusedLibrariesEx+0x2c
        combase!CDllHost::MTAWorkerLoop+0x7d
        combase!CDllHost::WorkerThread+0x122
        combase!CRpcThread::WorkerLoop+0x4e
        combase!CRpcThreadCache::RpcWorkerThreadEntry+0x46
        KERNEL32!BaseThreadInitThunk+0x30
        ntdll!RtlUserThreadStart+0x42
        THREAD ffffe000046db8c0  Cid 0a50.0b50  Teb: 00007ff6b64aa000 Win32Thread:
fffff9014370da90 WAIT: (UserRequest) UserMode Non-Alertable
            ffffe000046dcae0  NotificationEvent
            ffffe000046dd3c0  SynchronizationEvent
        nt!KiSwapContext+0x76
        nt!KiSwapThread+0x14c
        nt!KiCommitThreadWait+0x126
        nt!KeWaitForMultipleObjects+0x22b
        nt!ObWaitForMultipleObjects+0x1f8
        nt!NtWaitForMultipleObjects+0xde
        nt!KiSystemServiceCopyEnd+0x13 (TrapFrame @ ffffd000`21801c40)
        ntdll!ZwWaitForMultipleObjects+0xa
        KERNELBASE!WaitForMultipleObjectsEx+0xe1
        USER32!MsgWaitForMultipleObjectsEx+0x14e
        contoso!EventManagerImpl::MessageLoop+0x32
        contoso!EventManagerImpl::BackgroundProcessing+0x134
        ntdll!TppWorkpExecuteCallback+0x2eb
        ntdll!TppWorkerThread+0xa12
        KERNEL32!BaseThreadInitThunk+0x30
        ntdll!RtlUserThreadStart+0x42
        THREAD ffffe000049108c0  Cid 0a50.06cc  Teb: 00007ff6b6470000 Win32Thread:
0000000000000000 WAIT: (UserRequest) UserMode Non-Alertable
            ffffe000041c1830  SynchronizationEvent
        nt!KiSwapContext+0x76
        nt!KiSwapThread+0x14c
        nt!KiCommitThreadWait+0x126
        nt!KeWaitForSingleObject+0x1cc
        nt!NtWaitForSingleObject+0xb1
        nt!KiSystemServiceCopyEnd+0x13
        ntdll!ZwWaitForSingleObject+0xa
        ntdll!RtlpWaitOnCriticalSection+0xe1
        ntdll!RtlEnterCriticalSectionContended+0x94
        ntdll!LdrpAcquireLoaderLock+0x2c
        ntdll!LdrShutdownThread+0x64
        ntdll!RtlExitUserThread+0x3e
        KERNEL32!BaseThreadInitThunk+0x38
        ntdll!RtlUserThreadStart+0x42
```

Since debugging is an exercise in optimism, let’s ignore the stacks that didn’t come out properly. If we can’t make any headway, we can try to fix them, but let’s be hopeful that the stacks that are good will provide enough information.

Generally speaking, the deeper the stack, the more interesting it is, because uninteresting threads tend to be hanging out in their message loop or event loop, whereas interesting threads are busy doing something and have a complex stack trace to show for it.

Indeed, one of the deep stacks belongs to thread `0a9c`, and it also has a very telling section:

```
        ntdll!RtlpWaitOnCriticalSection+0xe1
        ntdll!RtlEnterCriticalSection+0x94
        ntdll!LdrpAcquireLoaderLock+0x4c
        ntdll!LdrpFindOrMapDll+0x75d
        ntdll!LdrpLoadDll+0x394
        ntdll!LdrLoadDll+0xc6
        kernelbase!LoadLibraryExW+0x142
        kernelbase!LoadLibraryExA+0x26
        contoso!__delayLoadHelper2+0x2b
        contoso!_tailMerge_Winmm_dll+0x3f
```

The polarity reverser’s completion handler is trying to load `winmm` via delay-load. That load request is waiting on a critical section, and it should be clear both from the scenario and the function names that the critical section it is trying to claim is the loader lock. In real life, I just proceeded with that conclusion, but but just for demonstration purposes, here’s how we can double-check:

```
0: kd> .thread ffffe00003c46080
0: kd> kn
  *** Stack trace for last set context - .thread/.cxr resets it
 # Call Site
00 nt!KiSwapContext+0x76
01 nt!KiSwapThread+0x14c
02 nt!KiCommitThreadWait+0x126
03 nt!KeWaitForSingleObject+0x1cc
04 nt!NtWaitForSingleObject+0xb1
05 nt!KiSystemServiceCopyEnd+0x13 (TrapFrame @ ffffd000`367ece40)
06 ntdll!ZwWaitForSingleObject+0xa
07 ntdll!RtlpWaitOnCriticalSection+0xe1
08 ntdll!RtlEnterCriticalSection+0x94
09 ntdll!LdrpAcquireLoaderLock+0x4c
0a ntdll!LdrpFindOrMapDll+0x75d
0b ntdll!LdrpLoadDll+0x394
0c ntdll!LdrLoadDll+0xc6
0d kernelbase!LoadLibraryExW+0x142
0e kernelbase!LoadLibraryExA+0x26
0f contoso!__delayLoadHelper2+0x2b
10 contoso!_tailMerge_Winmm_dll+0x3f
```

We need to grab the critical section passed to `RtlEnterCriticalSection`, but since this is an x64 machine, the parameter was passed in registers, not on the stack, so we need to figure out where the `rcx` register got stashed.

I'm going to assume that the same critical section is the first (only?) parameter to `RtlpWaitOnCriticalSection`. I don't know this for a fact, but it seems like a reasonable guess. The guess might be wrong; we'll see.

We disassemble the function look to see where it stashes `rcx`.

```
0: kd> u ntdll!RtlpWaitOnCriticalSection
    mov     qword ptr [rsp+18h],rbx
    push    rbp
    push    rsi
    push    rdi
    push    r12
    push    r13
    push    r14
    push    r15
    mov     rax,qword ptr [ntdll!__security_cookie (000007ff`3099d020)]
    xor     rax,rsp
    mov     qword ptr [rsp+80h],rax
    mov   r14,qword ptr gs:[30h]
    xor     r12d,r12d
    lea     rax,[ntdll!LdrpLoaderLock (00007fff`d4f51cb8)]
    mov     r15d,r12d
    cmp     rcx,rax
    mov     ebp,edx
    sete    r15b
    mov     rbx,rcx // ⇐ Bingo
```

Awesome, we can suck `rbx` out of the trap frame.

```
0: kd> .trap ffffd000`367ece40
rax=0000000000000000 rbx=00007fffd4f51cb8 rcx=000007f8136f2c2a
rdx=0000000000000000 rsi=00000000000001e8 rdi=0000000000000000
rip=000007f8136f2c2a rsp=000000000cf7f798 rbp=0000000000000000
 r8=000000000cf7f798  r9=0000000000000000 r10=0000000000000000
r11=0000000000000344 r12=0000000000000000 r13=0000000000000000
r14=000007f696870000 r15=000000007ffe0382
iopl=0         nv up ei pl zr na po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
ntdll!ZwWaitForSingleObject+0xa:
000007f8`136f2c2a c3              ret
```

Okay, let’s see if that value in `rbx` pans out.

```
0: kd> !cs 0x00007fff`d4f51cb8
-----------------------------------------
Critical section   = 0x00007fffd4f51cb8 (ntdll!LdrpLoaderLock+0x0)
DebugInfo          = 0x00007fffd4f55228
LOCKED
LockCount          = 0x8
WaiterWoken        = No
OwningThread       = 0x0000000000000b38
RecursionCount     = 0x1
LockSemaphore      = 0x1A8
SpinCount          = 0x0000000004000000
```

Hooray, we confirmed that this is indeed the loader lock. I would have been surprised if it had been anything else! (If you had been paying attention, you would have noticed the `lea rax,[ntdll!LdrpLoaderLock (00007fff`d4f51cb8)]` in the disassembly which already confirms the value.)

We also see that the owning thread is 0xb38. Here's its stack again:

```
nt!KiSwapContext+0x76
nt!KiSwapThread+0x14c
nt!KiCommitThreadWait+0x126
nt!KeWaitForSingleObject+0x1cc
nt!NtWaitForSingleObject+0xb1
nt!KiSystemServiceCopyEnd+0x13 (TrapFrame @ ffffd000`2563bc40)
ntdll!ZwWaitForSingleObject+0xa
KERNELBASE!WaitForSingleObjectEx+0xa5
litware!CDiscovery::Uninitialize+0x8c
litware!CApiInstance::~CApiInstance+0x48
litware!CApiInstance::`scalar deleting destructor'+0x14
litware!std::tr1::_Ref_count_obj<CApiInstance>::_Destroy+0x31
litware!std::tr1::_Ref_count_base::_Decref+0x1b
litware!std::tr1::_Ptr_base<CApiInstance>::_Decref+0x20
litware!std::tr1::shared_ptr<CApiInstance>::{dtor}+0x20
litware!std::tr1::shared_ptr<CApiInstance>::reset+0x3c
litware!CSingleton<CApiInstance>::ReleaseRef+0x97
litware!LitWareUninitialize+0xed
fabrikam!CDoodadHelper::~CDoodadHelper+0x67
fabrikam!_CRT_INIT+0xda
fabrikam!__DllMainCRTStartup+0x1e5
ntdll!LdrpCallInitRoutine+0x57
ntdll!LdrpProcessDetachNode+0xfe
ntdll!LdrpUnloadNode+0x77
ntdll!LdrpDecrementNodeLoadCount+0xd0
ntdll!LdrUnloadDll+0x34
KERNELBASE!FreeLibrary+0x22
combase!CClassCache::CDllPathEntry::CFinishObject::Finish+0x28
combase!CClassCache::CFinishComposite::Finish+0x80
combase!CClassCache::FreeUnused+0xda
combase!CoFreeUnusedLibrariesEx+0x2c
combase!CDllHost::MTAWorkerLoop+0x7d
combase!CDllHost::WorkerThread+0x122
combase!CRpcThread::WorkerLoop+0x4e
combase!CRpcThreadCache::RpcWorkerThreadEntry+0x46
KERNEL32!BaseThreadInitThunk+0x30
ntdll!RtlUserThreadStart+0x42
```

Reading from the bottom up, we see that this thread is doing some work on behalf of COM; specifically, it is freeing unused libraries. The `fabrikam` library presumably responded `S_OK` to `DllCanUnloadNow`, so COM says, "Okay, then out you go."

As part of DLL_PROCESS_DETACH processing, the C++ runtime library runs global destructors. The `CDoodadHelper` destructor calls into the `LitWareUninitialize` function in `litware.dll`. That function decrements a reference count, and it appears that the reference count went to zero because it's destructing the `CApiInstance` object. The destructor for that function calls `CDiscovery::Uninitialize`, and that function waits on a kernel object.

The debugger was kind enough to tell us what the object is:

```
        THREAD ffffe000046ad340  Cid 0a50.0b38  Teb: 00007ff6b64b6000 Win32Thread: 0000000000000000 WAIT: (UserRequest) UserMode Non-Alertable
            ffffe000049108c0  Thread
```

It's a thread.

<!–

But in case we didn't know that, here's how we could have figured it out anyway: Look at the handle that was passed to `WaitForSingleObject`:

```
0: kd> kn
  *** Stack trace for last set context - .thread/.cxr resets it
 # Call Site
00 nt!KiSwapContext+0x76
01 nt!KiSwapThread+0x14c
02 nt!KiCommitThreadWait+0x126
03 nt!KeWaitForSingleObject+0x1cc
04 nt!NtWaitForSingleObject+0xb1
05 nt!KiSystemServiceCopyEnd+0x13 (TrapFrame @ ffffd000`2563bc40)
06 ntdll!ZwWaitForSingleObject+0xa
07 KERNELBASE!WaitForSingleObjectEx+0xa5
08 litware!CDiscovery::Uninitialize+0x8c
0: kd> .frame 7
07 KERNELBASE!WaitForSingleObjectEx+0xa5
0: kd> dv
        hHandle = 0x00000000`000016e8
 dwMilliseconds = 0xffffffff
     bAlertable = 0n0
...
```

The code is waiting infinitely (`0xffffffff`) for handle `0x16e8`. Let's see what that handle refers to.

```
0: kd> !handle 16e8 f
PROCESS ffffe000045ef940
    SessionId: 1  Cid: 0a50    Peb: 7ff6b661f000  ParentCid: 0a0c
    DirBase: 12e5c6000  ObjectTable: ffffc0000288ae80  HandleCount: 1742.
    Image: contoso.exe
Handle table at ffffc0000288ae80 with 1742 entries in use
16e8: Object: ffffe000049108c0  GrantedAccess: 001fffff (Protected) Entry:
ffffc00002b8aba0
Object: ffffe000049108c0  Type: (ffffe000001fff20) Thread
    ObjectHeader: ffffe00004910890 (new version)
        HandleCount: 2  PointerCount: 32770
```

That value `ffffe000049108c0` matches the value reported by the thread dump at the start.

–>

Going back to the thread dump at the start, we also can see what thread `ffffe000049108c0` is doing. Here it is again:

```
        nt!KiSwapContext+0x76
        nt!KiSwapThread+0x14c
        nt!KiCommitThreadWait+0x126
        nt!KeWaitForSingleObject+0x1cc
        nt!NtWaitForSingleObject+0xb1
        nt!KiSystemServiceCopyEnd+0x13
        ntdll!ZwWaitForSingleObject+0xa
        ntdll!RtlpWaitOnCriticalSection+0xe1
        ntdll!RtlEnterCriticalSectionContended+0x94
        ntdll!LdrpAcquireLoaderLock+0x2c
        ntdll!LdrShutdownThread+0x64
        ntdll!RtlExitUserThread+0x3e
        KERNEL32!BaseThreadInitThunk+0x38
        ntdll!RtlUserThreadStart+0x42
```

That thread is trying to acquire the loader lock so it can send `DLL_THREAD_DETACH` notifications. But the loader lock is held by the `FreeLibrary`. Result: Deadlock, as the two threads are waiting for each other. (You can also see that thread `0xa8c` is stuck in the same place because it too is trying to exit.)

The underlying problem is that the Fabrikam DLL is waiting on a thread (indirectly via LitWare) while inside its own `DllMain`.

The Fabrikam code could avoid this problem by calling `LitWareUninitialize` when its last object is destroyed rather than when the DLL is unloaded. (Of course, it also has to remember to call `LitWareInitialize` when its first object is created.)

Raymond Chen

Follow