Debug session: Why is an LPC server not responding?

devblogs.microsoft.com/oldnewthing/20130215-00

February 15, 2013

Raymond Chen

A particular scenario was hanging, and the team responsible for the scenario debugged it to the point where they saw that their X component was waiting for their Y component, which was waiting for Explorer, so they asked for help chasing the hang into Explorer.

The team was kind enough to have shared what they've learned so far:

```
kd> !alpc /m 9c14d020


Message 9c14d020
  MessageID             : 0x0274 (628)
  CallbackID            : 0xCCA5 (52389)
  SequenceNumber        : 0x00000016 (22)
  Type                  : LPC_REQUEST
  DataLength            : 0x0094 (148)
  TotalLength           : 0x00AC (172)
  Canceled              : No
  Release               : No
  ReplyWaitReply        : No
  Continuation          : Yes
  OwnerPort             : 82bb9db8 [ALPC_CLIENT_COMMUNICATION_PORT]
  WaitingThread         : 834553c0
  QueueType             : ALPC_MSGQUEUE_MAIN
  QueuePort             : 84646730 [ALPC_CONNECTION_PORT]
  QueuePortOwnerProcess : 846209c0 (explorer.exe)
  ServerThread          : 00000000 <————————
  QuotaCharged          : No
  CancelQueuePort       : 00000000
  CancelSequencePort    : 00000000
  CancelSequenceNumber  : 0x00000000 (0)
  ClientContext         : 02a56b80
  ServerContext         : 00000000
  PortContext           : 0701ea20
  CancelPortContext     : 00000000
  SecurityData          : 962f89b8
  View                  : 00000000
kd> !process 846209c0 0
PROCESS 846209c0  SessionId: 1  Cid: 0804    Peb: 7fbac000  ParentCid: 0724
    DirBase: 3e546380  ObjectTable: 97195300  HandleCount: 1041.
    Image: explorer.exe
```

Yikes, there is no thread signed up to service the request.

I don't know much about ALPC, but I can fumble around. Fortunately, this is debugging and not rocket surgery, so you still get full points if you stumble across the answer by guessing.

I decided to start guessing by looking at what the `!alpc` command can tell me.

```
kd> !alpc -?


  !alpc /m MessageAddress
    Dumps the message at the specified address.


  !alpc /p PortAddress
    Dumps the port at the specified address.
```

Well, I already saw what the result was for dumping the message, so I may as well dump the port.

```
kd> !alpc /p 84646730


…
  8 thread(s) are registered with port IO completion object:
    THREAD 84658d40  Cid 0804.0888  Teb: 7fa7e000 Win32Thread: 8214a748 WAIT
    THREAD 8466a040  Cid 0804.08c4  Teb: 7fa74000 Win32Thread: 8214c800 WAIT
    THREAD 84659a00  Cid 0804.08ec  Teb: 7fa72000 Win32Thread: 82158d08 WAIT
    THREAD 8466c8c0  Cid 0804.08f0  Teb: 7fa6e000 Win32Thread: 82160420 WAIT
    THREAD 84671040  Cid 0804.0910  Teb: 7fa68000 Win32Thread: 8217c4e8 WAIT
    THREAD 8460d180  Cid 0804.099c  Teb: 7fa5e000 Win32Thread: 820bad08 WAIT
    THREAD 834278c0  Cid 0804.0c80  Teb: 7fa6b000 Win32Thread: 820b9620 WAIT
    THREAD 8345ad40  Cid 0804.0da0  Teb: 7fba9000 Win32Thread: 821c6d08 WAIT
…
```

So it looks like there are eight threads signed up to process events on this port. (Is that what this means? I don't know, but I'm going to assume that it does, because this is debugging. Debugging is an exercise in optimism.) Let's see what they're doing.

```
kd> .thread 84658d40;k
Implicit thread is now 84658d40
  *** Stack trace for last set context – .thread/.cxr resets it
ChildEBP RetAddr
940ef394 80f1505f nt!KiSwapContext+0x19
940ef3d0 80f184e0 nt!KiSwapThread+0x34b
940ef3fc 80f163fc nt!KiCommitThreadWait+0x26f
940ef46c 80f4d2df nt!KeWaitForSingleObject+0x459
940ef4b4 80e20838 nt!KiSchedulerApc+0x298
940ef4c8 00000000 hal!KfLowerIrql+0x2c


[the others look the same]
```

Well, I don't know what they're doing, but it looks like they're waiting for something. But one of the threads looks different:

```
kd> .thread 84671040;k
Implicit thread is now 84671040
  *** Stack trace for last set context – .thread/.cxr resets it
ChildEBP RetAddr
9415f864 80f1505f nt!KiSwapContext+0x19
9415f8a0 80f184e0 nt!KiSwapThread+0x34b
9415f8cc 80eb3d6e nt!KiCommitThreadWait+0x26f
9415f934 810c0527 nt!KeWaitForMultipleObjects+0x4e3
9415fbe4 810c0703 nt!ObWaitForMultipleObjects+0x2fd
9415fd38 80ef113c nt!NtWaitForMultipleObjects+0xca
9415fd38 77945e04 nt!KiFastCallEntry+0x12c
07e2f1c4 779437f6 ntdll!KiFastSystemCallRet
07e2f1c8 7515c136 ntdll!NtWaitForMultipleObjects+0xa
07e2f34c 77752658 KERNELBASE!WaitForMultipleObjectsEx+0xee
07e2f368 777fbe60 KERNEL32!WaitForMultipleObjects+0x19
07e2f3d4 777fc5de KERNEL32!WerpReportFaultInternal+0x1a3
07e2f3e8 777df654 KERNEL32!WerpReportFault+0x6d
07e2f3f4 751e517c KERNEL32!BasepReportFault+0x19
07e2f490 77a0f95a KERNELBASE!UnhandledExceptionFilter+0x1e0
07e2f4a0 77a0fd4d ntdll!TppExceptionFilter+0x1b
07e2f4b4 77a1c66b ntdll!TppWorkerpInnerExceptionFilter+0x13
07e2fb34 77753278 ntdll!TppWorkerThread+0xa6092
07e2fb40 779761a6 KERNEL32!BaseThreadInitThunk+0xe
07e2fb80 77976152 ntdll!__RtlUserThreadStart+0x4a
07e2fb90 00000000 ntdll!_RtlUserThreadStart+0x1c
```

Ah, well that explains why Explorer isn't responding: It crashed on an unhandled exception! Windows Error Reporting is busy trying to generate a report.

Now to see what the crash was. I don't know for sure, but I'm pretty confident that one of the parameters to `BasepReportFault` is an `EXCEPTION_POINTERS`. Why am I confident of that? Because it would be hard to report the fault without it!

```
kd> dd 07e2f3f4 l4
07e2f3f4  07e2f490 77a0f95a 07e2f4e8 00000001
          ChildEBP RetAddr  Param1


kd> dd 07e2f4e8 l2
07e2f4e8  07e2f620 07e2f63c
                ^ ^
   ExceptionRecord ContextRecord
kd> .cxr 0x07e2f63c
eax=00000000 ebx=0451e2f8 ecx=e2af034f edx=77945e00 esi=00000000 edi=0451e2e0
eip=1df7fc6a esp=07e2f920 ebp=07e2f938 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
contoso!ContosoPower::Disconnect+0xdd:
001b:1df7fc6a 8b08  mov ecx,dword ptr [eax] ds:0023:00000000=????????
```

Aha, Explorer crashed due to a null pointer crash in the `ContosoPower::Disconnect` function.

Passing the buck onward to Contoso, the report back was that this was a known issue, and a hotfix was available.

Raymond Chen

Follow