A process shutdown puzzle, Episode 2

devblogs.microsoft.com/oldnewthing/20120427-00

April 27, 2012

Raymond Chen

A customer reported that their program would very sporadically crash in the function `Close-ThreadpoolCleanupGroupMembers`. The customer was kind enough to provide a stack trace at the point of the crash:

```
ntdll!RtlUnhandledExceptionFilter2+0x31e
KERNELBASE!UnhandledExceptionFilter+0x175
ntdll!RtlUserThreadStart$filt$0+0x3f
ntdll!__C_specific_handler+0x8f
ntdll!RtlpExecuteHandlerForException+0xd
ntdll!RtlDispatchException+0x3a6
ntdll!RtlRaiseException+0x223
ntdll!TppRaiseInvalidParameter+0x48
ntdll!TpReleaseCleanupGroupMembers+0x246
litware!CThreadPool::UnInitialize+0x22
litware!_CRT_INIT+0xbf
litware!__DllMainCRTStartup+0x18b
ntdll!LdrpCallInitRoutine+0x3f
ntdll!LdrShutdownProcess+0x205
ntdll!RtlExitUserProcess+0x90
kernel32!ExitProcessImplementation+0xa
contoso!wmain+0x193
contoso!__wmainCRTStartup+0x13d
kernel32!BaseThreadInitThunk+0xd
ntdll!RtlUserThreadStart+0x1d
```

The customer wondered, “Could the problem be that my cleanup group does not have a callback? MSDN seems to suggest that this is okay.”

The exception being thrown is `STATUS_INVALID_PARAMETER`, but that doesn’t really say much.

But that’s okay, because the smoking gun isn’t the exception being raised. It’s in the stack.

Do you see it?

The code is calling `CloseThreadpoolCleanupGroupMembers` from inside `DllMain` while handling the `DLL_PROCESS_DETACH` notification. Looking further up the stack, you can see this was triggered by a call to `ExitProcess`, and now all the stuff you know about how processes exit kicks in.

For example, that the first thing that happens is that all threads are forcibly terminated.

That's your next clue.

Observe that the customer's DLL is trying to communicate with the thread pool during process termination. But wait, all the threads have already been terminated. It's trying to communicate with a nonexistent thread pool.

The thread pool realizes, "Hey, like I've already been destroyed. I can't do what you ask because there is no thread pool any more. You want me to block until all currently executing callback functions finish, but those callback functions will never finish (if they even exist at all) *because the threads hosting their thread pool got destroyed*. Not that I can tell whether they are executing or not, because I am already destroyed. The only options are to hang or crash. I think I'll crash."

The customer needs to restructure the program so that it either cleans up its thread pool work before the `ExitProcess`, or it can simply skip all thread pool operations when the reason for the `DLL_PROCESS_DETACH` is process termination.

Raymond Chen

Follow