

Uncovering the Cause of a Server Hang

Nischay Anikar

My name is Nischay Anikar from the Escalation Engineer team in Global Escalation Services. In today's post I'll present a weird problem I worked through with a client. When we started to work on the problem, we found the following:

- Ping to the box worked.
- Keyboard was responding.
- Shares on the system were accessible remotely.
- Could not Remote Desktop into the box.
- Existing sessions were responsive to some extent – no new processes were getting created.
- No new processes/application could be launched. Sometimes application would get launched but after waiting for a long time.
- Remote computer management would work, but not all snap-ins would work. (Event logs showed up, but disk management did not respond).

At this time, perfmon was collected and nothing in it indicated any kind of resource contention. This was certainly not the case of any process/thread pegging the CPU. The above observations told us this was not a hard hang, as the system was responsive at DPC level. Rather system was even responding to certain extent at passive level – remember, shares were accessible. SMB requests are processed basically by the worker threads created by SRV.SYS.

Remote management snap-ins and remote registry responsiveness showed that RPC was working fine. However some snap-ins like disk management were behaving inconsistently.

This is the stage at which we had the Kernel dump of the system and we started our normal analysis trying to find the root of the problem.

From the dump there were no blocked threads on locks (ERESOURCE, !locks), no memory pressure (perfmon confirmed it too - !vm 1), no CPUs stuck in spinlocks, no DPCs pending(!dpcs), no ready threads pending to execute(!ready), and no alarming LPC wait chain among threads leading to system hang. These are some of the common causes that could lead to system hang. None of these were seen in the dump.

Then I said, enough of running behind the debugger commands to look for known problems, they didn't yield me anything useful up front. When the dump was given we were told that they attempted to launch notepad from explorer (Start->Run->Notepad) which never launched. If we start chasing from this point we are likely going to hit the root of the problem or at least get some leads. With this in mind, when we dumped out the explorer threads we saw one of the threads that was indeed trying to launch notepad (there was one more in the same state but trying to launch some other application).

```
THREAD 892ef4a0 Cid 0d2c.0ea8 Teb: 7ffd8000 Win32Thread: e108e6c0 WAIT: (Unknown)
KernelMode Non-Alertable
    f573bc2c NotificationEvent
    892ef518 NotificationTimer
Not impersonating
DeviceMap e12bf190
Owning Process 892027f0 Image: explorer.exe
Wait Start TickCount 40848 Ticks: 8313 (0:00:02:09.890)
Context Switch Count 284 LargeStack
```

Uncovering the Cause of a Server Hang

Nischay Anikar

```
UserTime          00:00:00.000
KernelTime       00:00:00.078
Win32 Start Address ntdll!RtlpWorkerThread (0x7c839f2b)
Start Address kernel32!BaseThreadStartThunk (0x77e617ec)
Stack Init f573c000 Current f573bb8c Base f573c000 Limit f5736000 Call 0
Priority 14 BasePriority 8 PriorityDecrement 0
ChildEBP RetAddr  Args to Child
f573bba4 8082ffd7 892ef4a0 892ef548 00000100 nt!KiSwapContext+0x25 (FPO: [Uses EBP]
[0,0,4])
f573bbbc 808287d4 895c6548 80a560c6 00000000 nt!KiSwapThread+0x83 (FPO: [Non-Fpo])
f573bc00 80810135 f573bc2c 00000000 00000000 nt!KeWaitForSingleObject+0x2e0 (FPO: [Non-
Fpo])
f573bc48 80842608 005c6548 00000000 00000000 nt!CcWaitForUninitializeCacheMap+0xa5 (FPO:
[Non-Fpo])
f573bcd0 8091f8e7 f573bd20 000f001f 00000000 nt!MmCreateSection+0x1fc (FPO: [Non-Fpo])
f573bd40 80883938 0190d51c 000f001f 00000000 nt!NtCreateSection+0x12f (FPO: [Non-Fpo])
f573bd40 7c82860c 0190d51c 000f001f 00000000 nt!KiFastCallEntry+0xf8 (FPO: [0,0] TrapFrame
@ f573bd64)
0190d174 7c826ed9 77e6cc9a 0190d51c 000f001f ntdll!KiFastSystemCallRet (FPO: [0,0,0])
0190d178 77e6cc9a 0190d51c 000f001f 00000000 ntdll!NtCreateSection+0xc (FPO: [7,0,0])
0190d994 77e424b0 00000000 001394f4 0013725c kernel32!CreateProcessInternalW+0x99c (FPO:
[Non-Fpo])
0190d9cc 7c916750 001394f4 0013725c 00000000 kernel32!CreateProcessW+0x2c (FPO: [Non-Fpo])
0190e450 7c916b45 00030064 00000000 00139904 SHELL32!_SHCreateProcess+0x387 (FPO: [Non-
Fpo])
0190e4a4 7c91617b 00136008 0190e4c4 7c915a76 SHELL32!CShellExecute::_DoExecCommand+0xb4
(FPO: [Non-Fpo])
0190e4b0 7c915a76 00000001 00000009 00136008
SHELL32!CShellExecute::_TryInvokeApplication+0x49 (FPO: [Non-Fpo])
0190e4c4 7c91599f 00000000 00000009 0190e500 SHELL32!CShellExecute::ExecuteNormal+0xb1
(FPO: [Non-Fpo])
0190e4d8 7c915933 0190e500 00000000 00000009 SHELL32!ShellExecuteNormal+0x30 (FPO: [Non-
Fpo])
0190e4f4 7c9a3416 0190e500 0000003c 04000b00 SHELL32!ShellExecuteExW+0x8d (FPO: [Non-Fpo])
0190e954 7c9e3f92 00030064 0190e988 0190f828 SHELL32!ShellExecCmdLine+0x143 (FPO: [Non-
Fpo])
0190ee20 7c9e4517 0190eea8 7c9e43f6 0190ee5c SHELL32!CRunDlg::OKPushed+0x179 (FPO: [Non-
Fpo])
0190ee30 7739b6e3 000400b4 00000111 00000001 SHELL32!RunDlgProc+0x121 (FPO: [Non-Fpo])
0190ee5c 77395f82 7c9e43f6 000400b4 00000111 USER32!InternalCallWinProc+0x28
0190eed8 77395e22 00095fb4 7c9e43f6 000400b4 USER32!UserCallDlgProcCheckWow+0x147 (FPO:
[Non-Fpo])
0190ef20 77395ffa 00000000 00000111 00000001 USER32!DefDlgProcWorker+0xa8 (FPO: [Non-Fpo])
0190ef3c 7739b6e3 000400b4 00000111 00000001 USER32!DefDlgProcW+0x22 (FPO: [Non-Fpo])
0190ef68 7739b874 77395fd8 000400b4 00000111 USER32!InternalCallWinProc+0x28
0190efe0 7739bfce 00095fb4 77395fd8 000400b4 USER32!UserCallWinProcCheckWow+0x151 (FPO:
[Non-Fpo])
0190f010 7739bf74 77395fd8 000400b4 00000111 USER32!CallWindowProcAorW+0x98 (FPO: [Non-
Fpo])
0190f030 77431848 77395fd8 000400b4 00000111 USER32!CallWindowProcW+0x1b (FPO: [Non-Fpo])
0190f04c 77431b9b 000400b4 00000111 00000001 comctl32!CallOriginalWndProc+0x1a (FPO: [Non-
Fpo])
0190f0a8 77431d5d 001060a8 000400b4 00000111 comctl32!CallNextSubclassProc+0x3c (FPO: [Non-
Fpo])
0190f0cc 75ed2f80 000400b4 00000111 00000001 comctl32!DefSubclassProc+0x46 (FPO: [Non-Fpo])
0190f0f0 77431b9b 000400b4 00000111 00000001 BROWSEUI!CAutoComplete::_s_ParentWndProc+0xec
(FPO: [Non-Fpo])
0190f14c 77431dc0 001060a8 000400b4 00000111 comctl32!CallNextSubclassProc+0x3c (FPO: [Non-
Fpo])
0190f1a0 7739b6e3 000400b4 00000111 00000001 comctl32!MasterSubclassProc+0x54 (FPO: [Non-
Fpo])
0190f1cc 7739b874 77431d6c 000400b4 00000111 USER32!InternalCallWinProc+0x28
0190f244 7739c2d3 00095fb4 77431d6c 000400b4 USER32!UserCallWinProcCheckWow+0x151 (FPO:
[Non-Fpo])
```

Uncovering the Cause of a Server Hang

Nischay Anikar

```
0190f280 7739c337 004f8a78 004f5df0 00000001 USER32!SendMessageWorker+0x4bd (FPO: [Non-Fpo])
0190f2a0 77386cea 000400b4 00000111 00000001 USER32!SendMessageW+0x7f (FPO: [Non-Fpo])
0190f2d0 77396199 000400b4 00503dc8 00030064 USER32!IsDialogMessageW+0x41c (FPO: [Non-Fpo])
0190f30c 7738965e 000400b4 00030064 00000001 USER32!DialogBox2+0x144 (FPO: [Non-Fpo])
```

This is the first parameter to CreateProcess, and this thread coincided with the notepad launch from explorer.

```
kd> du 001394f4
001394f4 "C:\WINDOWS\system32\notepad.exe"
```

This thread has been waiting longer than 2 minutes. Looking at what this thread is doing, we see that it's waiting for a Cache Map to be uninitialized (tear down of the existing references on this cache map) as part of creating the Image section during process creation.

Examining the state of threads in the whole box, you see there are a few more threads in different processes that are waiting on the CreateProcess while creating an Image section and waiting to uninitialized the cache map.

```
kd> !thread 891910a8
THREAD 891910a8 Cid 0180.0184 Teb: 7ffdf000 Win32Thread: e1442bb8 WAIT: (Unknown)
KernelMode Non-Alertable
    f6d44c2c NotificationEvent
    89191120 NotificationTimer
IRP List:
    894f0298: (0006,0094) Flags: 00000800 Mdl: 00000000
Impersonation token: e105d028 (Level Impersonation)
DeviceMap e12bf190
Owning Process 89138708 Image: winlogon.exe
Wait Start TickCount 48380 Ticks: 781 (0:00:00:12.203)
Context Switch Count 1617 LargeStack
UserTime 00:00:00.156
KernelTime 00:00:00.468
Start Address winlogon!__report_gsfailure (0x0103e1b0)
Stack Init f6d45000 Current f6d44b8c Base f6d45000 Limit f6d40000 Call 0
Priority 15 BasePriority 15 PriorityDecrement 0
ChildEBP RetAddr Args to Child
f6d44ba4 8082ffd7 891910a8 89191150 00000100 nt!KiSwapContext+0x25 (FPO: [Uses EBP]
[0,0,4])
f6d44bbc 808287d4 896b20e8 80a560c6 00000000 nt!KiSwapThread+0x83 (FPO: [Non-Fpo])
f6d44c00 80810135 f6d44c2c 00000000 00000000 nt!KeWaitForSingleObject+0x2e0 (FPO: [Non-Fpo])
f6d44c48 80842608 006b20e8 00000000 00000000 nt!CcWaitForUninitializeCacheMap+0xa5 (FPO:
[Non-Fpo])
f6d44cd0 8091f8e7 f6d44d20 000f001f 00000000 nt!MmCreateSection+0x1fc (FPO: [Non-Fpo])
f6d44d40 80883938 0006eedc 000f001f 00000000 nt!NtCreateSection+0x12f (FPO: [Non-Fpo])
f6d44d40 7c82860c 0006eedc 000f001f 00000000 nt!KiFastCallEntry+0xf8 (FPO: [0,0] TrapFrame
@ f6d44d64)
0006eb34 7c826ed9 77e6cc9a 0006eedc 000f001f ntdll!KiFastSystemCallRet (FPO: [0,0,0])
0006eb38 77e6cc9a 0006eedc 000f001f 00000000 ntdll!NtCreateSection+0xc (FPO: [7,0,0])
0006f354 7d1ec670 00000818 00000000 0006fadc kernel32!CreateProcessInternalW+0x99c (FPO:
[Non-Fpo])
0006f3a0 75842db7 00000818 00000000 0006fadc ADVAPI32!CreateProcessAsUserW+0x108 (FPO:
[Non-Fpo])
0006f424 75842f3a 0008c260 0006f8d4 0008c208 MSGINA!ExecApplication+0x8e (FPO: [Non-Fpo])
0006f884 0103be76 0008c208 0006f8d4 00710000 MSGINA!WlxStartApplication+0xbb (FPO: [Non-Fpo])
0006f8a8 01036d59 0007a868 0006f8d4 00008001 winlogon!StartApplication+0x40 (FPO: [Non-Fpo])
0006faf8 01036fa4 0007a868 00000001 0007a868 winlogon!HandleLoggedOn+0x203 (FPO: [Non-Fpo])
```

Uncovering the Cause of a Server Hang

Nischay Anikar

```
0006fb14 0103b24d 00050020 00000659 00000001 winlogon!LoggedonDlgProc+0x8b (FPO: [Non-Fpo])
0006fb38 7739b6e3 00050020 00000659 00000001 winlogon!RootDlgProc+0x6e (FPO: [Non-Fpo])
0006fb64 77395f82 0103b1df 00050020 00000659 USER32!InternalCallWinProc+0x28
0006fbe0 77395e22 0008fca4 0103b1df 00050020 USER32!UserCallDlgProcCheckWow+0x147 (FPO:
[Non-Fpo])
0006fc28 77395ffa 00000000 00000659 00000001 USER32!DefDlgProcWorker+0xa8 (FPO: [Non-Fpo])
0006fc44 7739b6e3 00050020 00000659 00000001 USER32!DefDlgProcW+0x22 (FPO: [Non-Fpo])
0006fc70 7739b874 77395fd8 00050020 00000659 USER32!InternalCallWinProc+0x28
0006fce8 7739ba92 0008fca4 77395fd8 00050020 USER32!UserCallWinProcCheckWow+0x151 (FPO:
[Non-Fpo])
0006fd50 7739bad0 0006fda0 00000000 0006fd84 USER32!DispatchMessageWorker+0x327 (FPO: [Non-
Fpo])
0006fd60 77395d78 0006fda0 00000000 004f2cd0 USER32!DispatchMessageW+0xf (FPO: [Non-Fpo])
0006fd84 77396199 00050020 004f2cd0 00000000 USER32!IsDialogMessageW+0x56b (FPO: [Non-Fpo])
0006fdc0 7738965e 00050020 00000000 00000010 USER32!DialogBox2+0x144 (FPO: [Non-Fpo])
0006fde8 773896a0 01000000 0107cbc8 00000000 USER32!InternalDialogBox+0xd0 (FPO: [Non-Fpo])
0006fe08 773896e8 01000000 0107cbc8 00000000 USER32!DialogBoxIndirectParamAorW+0x37 (FPO:
[Non-Fpo])
0006fe2c 0103de0a 01000000 00000578 00000000 USER32!DialogBoxParamW+0x3f (FPO: [Non-Fpo])
0006fe50 0102d838 01000000 00000578 00000000 winlogon!Fusion_DialogBoxParam+0x24 (FPO:
[Non-Fpo])
0006fe8c 0103b6e0 0007a868 01000000 00000578 winlogon!TimeoutDialogBoxParam+0x28 (FPO:
[Non-Fpo])
0006fec4 0103746e 0007a868 01000000 00000578 winlogon!WlxDialogBoxParam+0x80 (FPO: [Non-
Fpo])
0006fee4 01038042 0007a868 77e62f9d 77e42014 winlogon!BlockWaitForUserAction+0x3a (FPO:
[Non-Fpo])
0006ff08 01031b33 0007a868 ffffffff 00000004 winlogon!MainLoop+0x42d (FPO: [Non-Fpo])
0006fff0 0103e33b 0007a868 00000000 000724e4 winlogon!WUNotify+0x515 (FPO: [Non-Fpo])
0006fff4 00000000 7ffd7000 000000c8 000001c9 winlogon!__report_gsfailure+0x267 (FPO: [Non-
Fpo])
```

```
kd> !thread 88alc3a0
```

```
THREAD 88alc3a0 Cid 01b0.072c Teb: 7ff9d000 Win32Thread: 00000000 WAIT: (Unknown)
```

```
KernelMode Non-Alertable
```

```
    f5ea7c2c NotificationEvent
```

```
    88alc418 NotificationTimer
```

```
Not impersonating
```

```
DeviceMap                e1000128
```

```
Owning Process            8911fd88      Image:         services.exe
```

```
Wait Start TickCount      32679        Ticks: 16482 (0:00:04:17.531)
```

```
Context Switch Count      2043
```

```
UserTime                  00:00:00.015
```

```
KernelTime                00:00:00.140
```

```
Win32 Start Address 0x0000ald5
```

```
LPC Server thread working on message Id ald5
```

```
Start Address kernel32!BaseThreadStartThunk (0x77e617ec)
```

```
Stack Init f5ea8000 Current f5ea7b8c Base f5ea8000 Limit f5ea5000 Call 0
```

```
Priority 10 BasePriority 9 PriorityDecrement 0
```

```
ChildEBP RetAddr  Args to Child
```

```
f5ea7ba4 8082ffd7 88alc3a0 88alc448 00000100 nt!KiSwapContext+0x25 (FPO: [Uses EBP]
```

```
[0,0,4])
```

```
f5ea7bbc 808287d4 895c83f0 80a560c6 00000000 nt!KiSwapThread+0x83 (FPO: [Non-Fpo])
```

```
f5ea7c00 80810135 f5ea7c2c 00000000 00000000 nt!KeWaitForSingleObject+0x2e0 (FPO: [Non-
Fpo])
```

```
f5ea7c48 80842608 005c83f0 00000000 00000000 nt!CcWaitForUninitializeCacheMap+0xa5 (FPO:
[Non-Fpo])
```

```
f5ea7cd0 8091f8e7 f5ea7d20 000f001f 00000000 nt!MmCreateSection+0x1fc (FPO: [Non-Fpo])
```

```
f5ea7d40 80883938 0359f270 000f001f 00000000 nt!NtCreateSection+0x12f (FPO: [Non-Fpo])
```

```
f5ea7d40 7c82860c 0359f270 000f001f 00000000 nt!KiFastCallEntry+0xf8 (FPO: [0,0] TrapFrame
@ f5ea7d64)
```

```
0359eec8 7c826ed9 77e6cc9a 0359f270 000f001f ntdll!KiFastSystemCallRet (FPO: [0,0,0])
```

```
0359eccc 77e6cc9a 0359f270 000f001f 00000000 ntdll!NtCreateSection+0xc (FPO: [7,0,0])
```

Uncovering the Cause of a Server Hang

Nischay Anikar

```
0359f6e8 77e424b0 00000000 00000000 000b5100 kernel32!CreateProcessInternalW+0x99c (FPO:
[Non-Fpo])
0359f720 0100928b 00000000 000b5100 00000000 kernel32!CreateProcessW+0x2c (FPO: [Non-Fpo])
0359f80c 01008a4c 0064a8b0 000b5100 0359f844 services!ScLogonAndStartImage+0x28b (FPO:
[Non-Fpo])
0359f84c 010069b1 0064a8b0 00000000 00000000 services!ScStartService+0x1c6 (FPO: [Non-Fpo])
0359f87c 01005e57 0064a8b0 00000000 00000000 services!ScStartMarkedServices+0x9c (FPO:
[Non-Fpo])
0359f8b4 01005ee0 0064a8b0 00000000 00000000 services!ScStartServiceAndDependencies+0x1f1
(FPO: [Non-Fpo])
0359f8d8 77c80193 000a0180 00000000 00000000 services!RStartServiceW+0x8c (FPO: [Non-Fpo])
0359f8f8 77ce33e1 01005e78 0359fae0 00000003 RPCRT4!Invoke+0x30
0359fcf8 77ce35c4 00000000 00000000 000abe9c RPCRT4!NdrStubCall12+0x299 (FPO: [Non-Fpo])
0359fd14 77c7ff7a 000abe9c 000a06d0 000abe9c RPCRT4!NdrServerCall12+0x19 (FPO: [Non-Fpo])
0359fd48 77c8042d 010024ef 000abe9c 0359fdec RPCRT4!DispatchToStubInCNoAvrf+0x38 (FPO:
[Non-Fpo])
0359fd9c 77c80353 00000013 00000000 0101c148
RPCRT4!RPC_INTERFACE::DispatchToStubWorker+0x11f (FPO: [Non-Fpo])
0359fdc0 77c811dc 000abe9c 00000000 0101c148 RPCRT4!RPC_INTERFACE::DispatchToStub+0xa3
(FPO: [Non-Fpo])
0359fdfc 77c812f0 000abc30 0009ff08 000d5c58
RPCRT4!LRPC_SCALL::DealWithRequestMessage+0x42c (FPO: [Non-Fpo])
0359fe20 77c88678 0009ff40 0359fe38 000abc30 RPCRT4!LRPC_ADDRESS::DealWithLRPCRequest+0x127
(FPO: [Non-Fpo])
0359ff84 77c88792 0359ffac 77c8872d 0009ff08 RPCRT4!LRPC_ADDRESS::ReceiveLotsaCalls+0x430
(FPO: [Non-Fpo])
0359ff8c 77c8872d 0009ff08 00000000 00000000 RPCRT4!RecvLotsaCallsWrapper+0xd (FPO: [Non-
Fpo])
0359ffac 77c7b110 0009e2b8 0359ffec 77e6482f RPCRT4!BaseCachedThreadRoutine+0x9d (FPO:
[Non-Fpo])
0359ffb8 77e6482f 000d4d78 00000000 00000000 RPCRT4!ThreadStartRoutine+0x1b (FPO: [Non-
Fpo])
0359ffec 00000000 77c7b0f5 000d4d78 00000000 kernel32!BaseThreadStart+0x34 (FPO: [Non-Fpo])
```

These threads stuck in Cache Manager while attempting to launch a process, can potentially lead to the symptoms that were described to us. Let's try to prove it.

While we will not go into the details of Cache Manager mechanics (Refer to Cache Manager, Chapter 11 in Windows Internals), a quick note on how these threads will be unblocked is needed for the sake of this problem. When image sections are created if there is any existing shared cache map associated, we wait for any references on the shared cache map for this image section to drop to zero.

The thread waiting on the cache map to be un-initialized will get signaled when the reference drops to zero on the shared cache map. The code that signals the un-initialization executes in the context of Cache Manager Worker and is queued onto a System Worker thread. Looking at so many threads, all waiting for Cache Manager Worker thread to signal the cleanup of the section, it appears that either- The Cache Manager Worker kicked off but never reached a point to signal these blocked threads. Cache Manager Worker has not had a chance to run yet.

The Cache Manager globals below indicate the maximum number of CC worker that can be active or queued at any time, and current active count. The counts below indicate we are already at the peak. The "nt!CcNumberActiveWorkerThreads" counter indicates the number of threads that already have work to do, but not necessarily currently executing Cache manager worker.

```
kd> x nt!CcNumberActiveWorkerThreads
80896144 nt!CcNumberActiveWorkerThreads = <no type information>
kd> dd 80896144 11
80896144 00000008 <<This indicates the work items queued that will/or have
engaged worker
kd> x nt!CcNumberWorkerThreads
```

Uncovering the Cause of a Server Hang

Nischay Anikar

```
80896140 nt!CcNumberWorkerThreads = <no type information>
kd> dd 80896140 l1
80896140 00000008 <<This is the counter for Max Cc worker threads
kd> x nt!CcWorkerThread
8081211e nt!CcWorkerThread = <no type information>
```

So what are these work queue items that are being executed?

If the first condition is true then we should find these worker (nt!CcWorkerThread) executing on top of a system worker thread. Yes we did search the stacks of all the threads in the dump, but we weren't fortunate enough to find any System Worker Threads executing the Cache Manager Worker.

Only other possibility is these Cache Manager Worker threads never got a chance to run, likely system has no System Worker Threads idle enough to pick these Cache Manager work. So how do we prove/disprove this? (We could have started dumping out the System Worker Queues and its associated threads) We take a quicker approach - !exqueue. This command displays information and state of system worker queue and work items queued in each of its worker queue.

Let's dump out the state of the System Worker Queue/Threads.

```
kd> !exqueue
Dumping ExWorkerQueue: 808A76C0
```

```
**** Critical WorkQueue( current = 0 maximum = 1 )
THREAD 898f9b40 Cid 0004.0010 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 898f98d0 Cid 0004.0014 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 898f8020 Cid 0004.0018 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 898f8db0 Cid 0004.001c Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 898f8b40 Cid 0004.0020 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 898f88d0 Cid 0004.0024 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 898f8660 Cid 0004.0028 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 898f83f0 Cid 0004.002c Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 898f7020 Cid 0004.0030 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 898f7db0 Cid 0004.0034 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 89652868 Cid 0004.0ed0 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 895faa40 Cid 0004.0ed4 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 891fb9b8 Cid 0004.0ed8 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 89129db0 Cid 0004.0edc Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 892c4780 Cid 0004.0ee0 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 8961b6a0 Cid 0004.0ee4 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 8917a730 Cid 0004.0ee8 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 88a31b10 Cid 0004.0eec Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 895eacb0 Cid 0004.0ef0 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 891d7db0 Cid 0004.0ef8 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 89667b08 Cid 0004.0f14 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 8920a490 Cid 0004.0f48 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 892f3cb0 Cid 0004.0fa8 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 8962bdb0 Cid 0004.0fb0 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 89661350 Cid 0004.0fb8 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 8918adb0 Cid 0004.0fbc Teb: 00000000 Win32Thread: 00000000 WAIT
```

<Pending Work Items list for this queue>

```
PENDING: WorkerRoutine nt!CcWorkerThread (8081211e) Parameter 898f51e0
PENDING: WorkerRoutine nt!CcWorkerThread (8081211e) Parameter 898f9670
PENDING: WorkerRoutine nt!IopProcessWorkItem (808e419a) Parameter 891f8648
PENDING: WorkerRoutine nt!CcWorkerThread (8081211e) Parameter 898fald8
PENDING: WorkerRoutine Ntfs!NtfsCheckpointAllVolumes (f7135a57) Parameter 00000000
PENDING: WorkerRoutine srv!SrvResourceAllocThread (f5edfa28) Parameter 00000000
PENDING: WorkerRoutine nt!IopProcessWorkItem (808e419a) Parameter 89308f00
PENDING: WorkerRoutine nt!ObpProcessRemoveObjectQueue (8092b70e) Parameter 00000000
PENDING: WorkerRoutine srv!SrvResourceThread (f5ee026d) Parameter 00000000
PENDING: WorkerRoutine netbt!NTEExecuteWorker (f67cdcb2) Parameter f67eb6bc
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 89191008
```

Uncovering the Cause of a Server Hang

Nischay Anikar

```
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 8965d1e8
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 895edea0
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 892b8be8
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 895e11e8
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 89607210
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 896634a8
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 8915dce0
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 89221110
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 8922a968
PENDING: WorkerRoutine nt!CcWorkerThread (8081211e) Parameter 898f7278
PENDING: WorkerRoutine nt!CcWorkerThread (8081211e) Parameter 8998cd38
PENDING: WorkerRoutine nt!CcWorkerThread (8081211e) Parameter 898f9688
PENDING: WorkerRoutine nt!CcWorkerThread (8081211e) Parameter 898f8298
PENDING: WorkerRoutine nt!CcWorkerThread (8081211e) Parameter 8998c030
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 891fe578
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 891817c0
PENDING: WorkerRoutine Ntfs!NtfsCheckUsnTimeOut (f71489b8) Parameter 00000000
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 89648fd0
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 89207618
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 895fc7d0
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 89268950
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 8921e008
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 892acbe98
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 89685e98
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 8921ae60
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 896521a0
PENDING: WorkerRoutine termdd!_IcaDelayedWorker (f767d29a) Parameter 8920ab68
```

```
**** Delayed WorkQueue( current = 0 maximum = 1 )
THREAD 898f7b40 Cid 0004.0038 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 898f78d0 Cid 0004.003c Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 898f7660 Cid 0004.0040 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 898f73f0 Cid 0004.0044 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 898f6020 Cid 0004.0048 Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 898f6db0 Cid 0004.004c Teb: 00000000 Win32Thread: 00000000 WAIT
THREAD 898f6b40 Cid 0004.0050 Teb: 00000000 Win32Thread: 00000000 WAIT
```

```
**** HyperCritical WorkQueue( current = 0 maximum = 1 )
THREAD 898f68d0 Cid 0004.0054 Teb: 00000000 Win32Thread: 00000000 WAIT
```

This command examines the state of the System Work queue and associated System Worker threads. It's telling us that there are three set of queues prioritized as hyper-Critical, Critical and Delayed-Worker queues. While Delayed-worker and Hyper-Critical queues are empty, the Critical Worker queue has enough pending items to keep it busy. This is not good. On an ideal case we expect all the work to be processed immediately and almost no work pending in the queue.

Before we move on, let's take a step back and see why we came here. We were chasing down the cache manager workers and we came here to find if there is any Cache Manager work pending in the worker queue to be picked up. Indeed yes, we can see all of the 8 ("nt!CcNumberActiveWorkerThreads") still pending. This answers the puzzle as far as threads that were blocked at Cache Manager's shared cache map un-initialization. And "lexqueue" did come to our rescue here.

It's always like this! You get an answer to one question, but at the same time the next question is readied for you, i.e. why are these work items still pending and not being processed?

For this we need a little bit of background on how System Worker Threads work. Several system components and drivers may need to execute the code at PASSIVE LEVEL and in a thread context. For this they could always create new threads and use them to execute the code they want. Other option is to rely on the pre-created threads by the system called "System worker Threads" and get

Uncovering the Cause of a Server Hang

Nischay Anikar

relieved from the burden of thread management itself. Based on the priority of the work, work is queued to any of the three queues (Critical, Hyper-Critical, and Delayed-Worker). By default there will be certain number of worker threads (Refer to Chapter 3, System Mechanisms - System Worker Threads – in Windows Internals) created for each of these queues and they will wait on the respective queues for any new work to come in, pick the work and get back to wait on the queue after the completion of the work.

At a certain point it could so happen that all these pre-created threads would be executing some work, and may get blocked on another work item to complete. But as there are no idle worker threads to pick up this work, it would sit in the pending queue, resulting in blockage of all the work to be done by these set of system worker threads.

The Operating System tries to address this kind of a problem to some extent by running deadlock detection algorithm in a timely manner. When this code runs and the system sees that the pending work items are increasing (that is to say that no work items are being picked up, or work is coming in a higher rate than the existing number of threads could handle), it may decide to create additional worker threads to help with the pending work items load. These threads are special worker threads called “Dynamic Worker Threads”. These threads exist as long as there is enough work to be done. However they terminate on being idle for a certain amount of time, so the system doesn't tie up resources for unused worker threads. Even if these dynamic worker threads get blocked, the system cannot keep creating the additional dynamic worker threads forever, as this will lead to the system filling up with worker threads and all getting blocked.

The sole intention of Dynamic worker threads is to try to help any immediate additional load or help system recover from deadlock among existing worker threads. However a couple of dynamic threads should suffice this need if it's indeed a transient state. But if this is not a transient state and there is a real software problem then System should have to stop creating these dynamic threads at some point. This will eventually lead to hung Worker threads with work items getting just queued.

This dynamic thread count is limited to 16 for the Critical worker queue, and System will not create any more as soon we reach this limit. (Refer to Chapter 3, System Mechanisms - System Worker Threads – in Windows Internals and/or Documentation in DDK/WDK).

With this knowledge on System Worker threads, the next step ahead is to determine what the existing Worker threads in the Critical Worker queue are doing which is preventing them from picking up our work items. Below is one of those threads, waiting on a Notification event as part of processing the work from WorkerDrv.SYS. Checking what every single thread in the Critical Worker queue is doing, we see they all are waiting in WorkerDrv.SYS driver (All these threads may not be occupied by the same driver always, but could be a similar deadlock among different drivers).

```
kd> !thread 898f9b40
THREAD 898f9b40 Cid 0004.0010 Teb: 00000000 Win32Thread: 00000000 WAIT: (Unknown)
KernelMode Alertable
    f78aed5c NotificationEvent
Not impersonating
DeviceMap                e1000128
Owning Process            898fa648      Image:           System
Wait Start TickCount     28506          Ticks: 20655 (0:00:05:22.734)
Context Switch Count     2
UserTime                 00:00:00.000
KernelTime               00:00:00.000
Start Address nt!ExpWorkerThread (0x8087acfe)
Stack Init f78af000 Current f78aecc4 Base f78af000 Limit f78ac000 Call 0
Priority 13 BasePriority 13 PriorityDecrement 0
ChildEBP RetAddr  Args to Child
```

Uncovering the Cause of a Server Hang

Nischay Anikar

```
f78aecdc 8082ffd7 898f9b40 898f9be8 00000400 nt!KiSwapContext+0x25 (FPO: [Uses EBP]
[0,0,4])
f78aecf4 808287d4 891eac00 00000000 895b3268 nt!KiSwapThread+0x83 (FPO: [Non-Fpo])
f78aed38 f77b30fe f78aed5c 00000000 00000000 nt!KeWaitForSingleObject+0x2e0 (FPO: [Non-
Fpo])
WARNING: Stack unwind information not available. Following frames may be wrong.
f78aed6c 808e41ad 88a80c08 f78ced5c 808a76c0 WorkerDrv+0x40fe
f78aed80 8087ade9 895b3268 00000000 898f9b40 nt!IopProcessWorkItem+0x13 (FPO: [Non-Fpo])
f78aedac 809418f4 895b3268 00000000 00000000 nt!ExpWorkerThread+0xeb (FPO: [Non-Fpo])
f78aeddc 80887f7a 8087acfe 00000000 00000000 nt!PspSystemThreadStartup+0x2e (FPO: [Non-
Fpo])
00000000 00000000 00000000 00000000 00000000 nt!KiThreadStartup+0x16
```

And yes, we know who the culprit is. This Driver has utilized all of the default System Critical Worker threads and the additional Dynamic threads for this queue.

```
kd> x nt!ExWorkerQueue
808a76c0 nt!ExWorkerQueue = <no type information>

kd> dt nt!_EX_WORK_QUEUE 808a76c0 .
+0x000 WorkerQueue      :
+0x000 Header           : _DISPATCHER_HEADER
+0x010 EntryListHead    : _LIST_ENTRY [ 0x898f51e0 - 0x8920ab70 ]<---Pending
(QueueDepthLastPass)
+0x018 CurrentCount     : 0
+0x01c MaximumCount    : 1
+0x020 ThreadListHead  : _LIST_ENTRY [ 0x898f9c48 - 0x8918aeb8 ]<---Threads
attached to this queue
+0x028 DynamicThreadCount : 0x10 <-----Count of
additional threads created as per deadlock detection
+0x02c WorkItemsProcessed : 0x10f3
+0x030 WorkItemsProcessedLastPass : 0x10f3
+0x034 QueueDepthLastPass : 0x26 <<Count of pending work items
+0x038 Info             :
+0x000 QueueDisabled   : 0y0
+0x000 MakeThreadsAsNecessary : 0y1
+0x000 WaitMode        : 0y0
+0x000 WorkerCount     : 0y00000000000000000000000011010 (0x1a)
+0x000 QueueWorkerInfo : 210
```

Looking at the pending work items we know what kind of impact this deadlock could have on the system. Any operation that is dependent on this set of worker threads will surely be impacted, and over a period of time you expect the system to crawl and slowly could possibly reach a dead end with components having direct/indirect dependency on this component of the system. We see Termdd, NTFS, and Cache manager work items in the pending queue which explains RDP not working, new processes not getting launched and so on.

Closure

At the point when we know this driver has consumed all the Critical Worker threads, the quickest way to get the system up and running is to disable this driver. And I could work on fixing our WorkerDrv.SYS so that this driver understands the importance of System Worker threads and doesn't flood the worker queue with work items that will block for a long time or with work items that are dependent on other work items, leading to this situation.

To summarize, we started with a problem description of crawling/almost hung system (a few components were indeed responding). We found why application launch was being blocked, which lead us to cache manager threads. Chasing down cache manager threads, we ended up with System Worker Threads. Then to my driver WorkerDrv.SYS which never understood the importance of System Worker threads, and used them too freely.