

microsoft.public.vc.language: deadlock on SMP machines – perhaps caused by wldap32.dll?

deadlock on SMP machines – perhaps caused by wldap32.dll?

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2004-07/0452.html>

From: Marc Elliott (marce808_at_hotmail.com)

Date: 07/09/04

Date: 9 Jul 2004 10:19:33 -0700

Hello,

Apologies in advance, this is a very long post, but I'm trying to be as complete as possible. I'm also not sure if this is the correct group, but it seemed appropriate.

We have a process that runs as part of our product that has been working fine on Windows. However, we recently had an issue onsite where these processes (there are usually multiple instances) would hang and become completely unresponsive on SMP (Compaq 8-way) machines running Windows NT and 2000. I've managed to recreate the issue in our test lab on a dual Xeon (hyperthreaded) box running Windows 2003. There is definitely some sort of deadlock situation here, but I'm at a loss how to proceed after banging my head against this for a week now. There is some evidence that suggests the DllInit function of wldap32 is somehow a player in this – the details are further down.

First, let me provide a little context: This service (it's not really an application) is written in VC++ 6.0, and is spawned via a daemon that is also part of our application. In a typical deployment there is one of these service processes for each processor on the local machine. The service instantiates a Java Virtual Machine via JNI, and most of our core code executes within the VM.

One of the functions of this service is to spawn processes and feed them input and capture their output. This is achieved by the VM loading a VC++ 6.0 bridge library, which in turn loads another VC++ 6.0 library (we'll call this the "work library") which does the actual work. When all of the assigned work is through, the work library is unloaded until the next time. The work library is fairly simple, and the core behavior involves invoking CreateProcess (usually cmd.exe, which in my test setup just echoes some msg), and creating threads to write to the input handle, and read from the output and error handles. This has been working on all flavors of Windows (uniprocessor) for quite some time.

microsoft.public.vc.language: deadlock on SMP machines – perhaps caused by wldap32.dll?

Now, to the details. Here is the basic sequence leading up to the hang:

- 1) Process loads the work library
- 2) Process performs some number of brief "work units" (CreateProcess() + accompanying threads, and then cleanup) happily and healthily
- 3) Process CPU utilization drops to 0, and there is no more activity, ever. This lockup appears to happen faster when more instances are running.

Attaching cdb to these hung processes consistently reveals two scenarios:

Scenario A – the "Is wldap32.dll doing something bad?" scenario

The work library thread which called CreateProcess has successfully waited on the process to exit. It waits on an Event that will be set by the thread which reads the output from the process, but that Event is never set. Mainly because those helper threads are stuck. The debugger always has to forcibly break into the process b/c the loader lock is held.

!locks reveals two locks in contention:

```
CritSec ntdll!LdrpLoaderLock+0 at 77FC2340
LockCount 8
RecursionCount 1
OwningThread b80
EntryCount 2a
ContentionCount 2a
*** Locked
```

```
CritSec WLDAP32!PerThreadListLock+0 at 76F34010
LockCount 1
RecursionCount 1
OwningThread d64
EntryCount 1
ContentionCount 1
*** Locked
```

```
-----
Stack trace for thread b80:
0f5dfba8 77f43741 77f5d64e 000005a0 00000000
SharedUserData!SystemCallStub+0x4
0f5dfbac 77f5d64e 000005a0 00000000 00000000
ntdll!ZwWaitForSingleObject+0xc
0f5dfbe8 77f42044 000005a0 76f1129f 76f34010
ntdll!RtlpWaitForCriticalSection+0x126
0f5dfbf0 76f1129f 76f34010 00000000 0f5dfc1c
ntdll!RtlEnterCriticalSection+0x46
0f5dfc00 76f11257 00000b80 77f47dc9 76f10000
WLDAP32!AddPerThreadEntry+0x2f
0f5dfc08 77f47dc9 76f10000 00000002 00000000 WLDAP32!LdapDllInit+0x3e
```

deadlock on SMP machines – perhaps caused by wldap32.dll?

microsoft.public.vc.language: deadlock on SMP machines – perhaps caused by wldap32.dll?

```
0f5dfc28 77f5d067 76f11180 76f10000 00000002
ntdll!LdrpCallInitRoutine+0x14
0f5dfcbc 77f5cef6 0f5dfd30 0f5dfd30 00000000
ntdll!LdrpInitializeThread+0xd4
0f5dfd1c 77f45241 0f5dfd30 77f40000 00000000
ntdll!LdrpInitialize+0x171
00000000 00000000 00000000 00000000 00000000
ntdll!KiUserApcDispatcher+0x7
```

To me, this begs the question: why is this library referred to in the first place? We certainly don't use LDAP and we aren't using ActiveDirectory, either. 'lm' reveals that it is an unloaded module. My only guess is that the library init routine is called in order to see if LDAP is enabled in case it's needed to check process/thread creation security tokens.

The interesting thing about WLDAP32!PerThreadListLock+0 is that the owning thread (d64) is nowhere to be found! So, no stack trace. There are several other threads waiting on locks. None of them are in my input/output/error read/write user code, they are all in LdrpInitialize or kernel32!ExitThread. I suspect the exiting threads are mine.

5 threads have the same trace (with differing addresses, but waiting on the same critical section - 77fc2340):

```
0f6dfbec 77f43741 77f5d64e 00000330 00000000
SharedUserData!SystemCallStub+0x4
0f6dfbf0 77f5d64e 00000330 00000000 00000000
ntdll!ZwWaitForSingleObject+0xc
0f6dfc2c 77f42044 00000330 77f5cfcc 77fc2340
ntdll!RtlpWaitForCriticalSection+0x126
0f6dfc34 77f5cfcc 77fc2340 00000000 7ffdf000
ntdll!RtlEnterCriticalSection+0x46
0f6dfcbc 77f5cef6 0f6dfd30 0f6dfd30 00000000
ntdll!LdrpInitializeThread+0x2f
0f6dfd1c 77f45241 0f6dfd30 77f40000 00000000
ntdll!LdrpInitialize+0x171
00000000 00000000 00000000 00000000 00000000
ntdll!KiUserApcDispatcher+0x7
```

And here are the exiting threads:

c18:

```
0e0dfe70 77f43741 77f5d64e 00000330 00000000
SharedUserData!SystemCallStub+0x4
0e0dfe74 77f5d64e 00000330 00000000 00000000
ntdll!ZwWaitForSingleObject+0xc
0e0dfeb0 77f42044 00000330 77f7c658 77fc2340
ntdll!RtlpWaitForCriticalSection+0x126
0e0dfeb8 77f7c658 77fc2340 00000000 7ffa4000
ntdll!RtlEnterCriticalSection+0x46
0e0dff44 77e57eb3 00000000 0c0bd418 00a87b88
ntdll!LdrShutdownThread+0x30
0e0dff7c 77bc917d 00000000 77bc91f3 00000000 kernel32!ExitThread+0x41
0e0dff84 77bc91f3 00000000 00000000 00000000 msvcrt!_endthreadex+0x25
0e0dffb8 77e4a990 00a87b88 00000000 00000000 msvcrt!_endthreadex+0x9b
0e0dffec 00000000 77bc917e 00a87b88 00000000
kernel32!BaseThreadStart+0x34
```

9cc:

```
0e1dfe70 77f43741 77f5d64e 00000330 00000000
SharedUserData!SystemCallStub+0x4
0e1dfe74 77f5d64e 00000330 00000000 00000000
ntdll!ZwWaitForSingleObject+0xc
0e1dfeb0 77f42044 00000330 77f7c658 77fc2340
ntdll!RtlpWaitForCriticalSection+0x126
```

microsoft.public.vc.language: deadlock on SMP machines – perhaps caused by wldap32.dll?

```
0e1dfefb8 77f7c658 77fc2340 00000000 7ffa3000
ntdll!RtlEnterCriticalSection+0x46
0e1dff44 77e57eb3 00000000 0b2e6c48 0b38b590
ntdll!LdrShutdownThread+0x30
0e1dff7c 77bc917d 00000000 77bc91f3 00000000 kernel32!ExitThread+0x41
0e1dff84 77bc91f3 00000000 00000000 00000000 msvcrt!_endthreadex+0x25
0e1dffb8 77e4a990 0b38b590 00000000 00000000 msvcrt!_endthreadex+0x9b
0e1dffec 00000000 77bc917e 0b38b590 00000000
kernel32!BaseThreadStart+0x34
I can provide further information if it is needed.
```

Scenario B - the "what happened to the malloc lock?" scenario

As in scenario A, the work library thread which called CreateProcess has successfully waited on the process to exit. It waits on an Event that will be set by the thread which reads the output from the process, but that Event is never set. These helper threads are also stuck, but this time they are actually in user code. The debugger does not have to forcibly break into this process.

!locks only reveals one lock in contention:

```
CritSec MSVCRTD!__app_type+34 at 10264780
LockCount          1
RecursionCount     0
OwningThread       0
EntryCount         4
ContentionCount    4
*** Locked
```

The owning thread is 0. Uh oh. My two helper threads also happen to be waiting on this lock:

ea8:

```
0f2eed90 77f43741 77f5d64e 00000680 00000000
SharedUserData!SystemCallStub+0x4
0f2eed94 77f5d64e 00000680 00000000 00000000
ntdll!ZwWaitForSingleObject+0xc
0f2eedd0 77f42044 00000680 1020b5a3 10264780
ntdll!RtlpWaitForCriticalSection+0x126
0f2eedd8 1020b5a3 10264780 0f2eeec4 0f2eee18
ntdll!RtlEnterCriticalSection+0x46
0f2eede8 1021352a 00000009 0f2efed8 0f2eee74 MSVCRTD!_lock+0x93
0f2eee18 102134ce 0de09de0 00000001 0f2eee34 MSVCRTD!_free_dbg+0x2a
0f2eee28 104bd0ed 0de09de0 0f2eee44 104ad0f0 MSVCRTD!free+0xe
0f2eee34 104ad0f0 0de09de0 0f2eeec4 0f2eee5c MSVCP60D!operator
delete+0xd
0f2eee44 1048be9e 0de09de0 00000021 ffffffff
MSVCP60D!std::allocator<char>::deallocate+0x10
0f2eee5c 10489b71 00000001 0f2eeec4 0f2efee4
MSVCP60D!std::basic_string<char,std::char_traits<char>,std::allocator<char>
>::_Tidy+0x6e
0f2eee6c 0c0e3c99 0f2eff74 0f2efeec 0de09ce0
MSVCP60D!std::basic_string<char,std::char_traits<char>,std::allocator<char>
>::~~basic_string<char,std::char_traits<char>,std::allocator<char>
>+0x11
0f2efee4 0c0d9ddb 00000000 00000000 0de09ce0
worklib!NullCopyRunner::run+0x119
0f2eff84 1020bf53 0de09c98 00000000 00000000
worklib!JobThread::execute+0x4b
0f2effb8 77e4a990 0de09ce0 00000000 00000000
MSVCRTD!_beginthreadex+0x133
0f2effec 00000000 1020bee0 0de09ce0 00000000
kernel32!BaseThreadStart+0x34
7d0:
```

microsoft.public.vc.language: deadlock on SMP machines – perhaps caused by wldap32.dll?

```
-----  
0f1ee64c 77f43741 77f5d64e 00000680 00000000  
SharedUserData!SystemCallStub+0x4  
0f1ee650 77f5d64e 00000680 00000000 00000000  
ntdll!ZwWaitForSingleObject+0xc  
0f1ee68c 77f42044 00000680 1020b5a3 10264780  
ntdll!RtlpWaitForCriticalSection+0x126  
0f1ee694 1020b5a3 10264780 00000021 0f1ee6d8  
ntdll!RtlEnterCriticalSection+0x46  
0f1ee6a4 102129da 00000009 00000000 0f1ee7cc MSVCRTD!_lock+0x93  
0f1ee6d8 102129a6 00000021 00000001 00000001  
MSVCRTD!_nh_malloc_dbg+0x2a  
0f1ee6f4 1020e2cf 00000021 00000001 0f1eff44 MSVCRTD!_nh_malloc+0x16  
0f1ee708 104b7a69 00000021 0f1ee728 104ad0d2 MSVCRTD!operator new+0xf  
0f1ee714 104ad0d2 00000021 00000000 0f1eff44  
MSVCP60D!std::_Allocate+0x19  
0f1ee728 1048ba18 00000021 00000000 00000000  
MSVCP60D!std::allocator<char>::allocate+0x12  
0f1ee770 1048bccc 00000017 00000000 0f1eff44  
MSVCP60D!std::basic_string<char,std::char_traits<char>,std::allocator<char>  
>::_Copy+0x58  
0f1ee784 1048a005 00000017 00000001 0f1eff44  
MSVCP60D!std::basic_string<char,std::char_traits<char>,std::allocator<char>  
>::_Grow+0x10c  
0f1ee798 1048a060 0c136674 00000017 0f1eff44  
MSVCP60D!std::basic_string<char,std::char_traits<char>,std::allocator<char>  
>::assign+0x15  
0f1ee7ac 10489ad7 0c136674 0f1eff44 0f1eff84  
MSVCP60D!std::basic_string<char,std::char_traits<char>,std::allocator<char>  
>::assign+0x20  
0f1ee7bc 0c0d9e1f 0c136674 0f1eff40 1020d797  
MSVCP60D!std::basic_string<char,std::char_traits<char>,std::allocator<char>  
>::basic_string<char,std::char_traits<char>,std::allocator<char>  
>+0x27  
0f1eff84 1020bf53 0de09ba8 00000000 00000000  
worklib!JobThread::execute+0x8f  
0f1effb8 77e4a990 0de09bf0 00000000 00000000  
MSVCRTD!_beginthreadex+0x133  
0f1effec 00000000 1020bee0 0de09bf0 00000000  
kernel32!BaseThreadStart+0x34  
So it appears that the malloc/heap/whatever-you-call-it lock is  
somehow munged. None of the other threads reveal anything interesting.  
-----
```

Summary: I considered the possibility of heap contamination, but using gflags I enabled full page heap and every other form of heap checking, but there appear to be no problems there. I am suspicious that there is something else at work because these scenarios happen exactly the same every time.

As I mentioned before, this code has been working perfectly on Windows uniprocessor systems for more than a year now. The only variable is the multiprocessor environment. I am seriously thinking that this is system related.

I've noticed that there have been some deadlock related fixes for wldap32.dll, including a recent hotfix that postdates the release of Windows 2003. I'm wondering if this would fix our problem?

I eagerly await any new ideas on this issue! I am pretty much out of them at this point. If I can provide further information on this, please ask. Thanks for reading!

Regards,

Marc