方亮的专栏

[原]DllMain中不当操作导致死锁问题的分析--导致DllMain中死锁的关键隐藏因子2

2012-11-5 阅读1694 评论7

本文介绍使用Windbg去验证《DllMain中不当操作导致死锁问题的分析--导致DllMain中死锁的关键隐藏因子》中的结论，调试对象是文中刚开始那个例子。（转载请指明出于breaksoftware的csdn博客）

1 g 让程序运行起来

2 ctrl+break 中断程序

3 ~ 查看线程数

```
0:001> ~
   0  Id: 1c9c.afc Suspend: 1 Teb: 7ffdf000 Unfrozen
.  1  Id: 1c9c.18c4 Suspend: 1 Teb: 7ffde000 Unfrozen
```

其实该程序自己运行起来的线程只有ID为0、TID为afc的线程。18c4线程是我们在windbg中输入ctrl+break，导致windbg在我们调试的进程中插入的一个中断线程。以后我们看到是这个线程的操作，就可以忽略。

4 dd fs:[0] 寻找主线程TEB起始地址(7ffde000)

```
0:001> dd fs:[0]
0038:00000000  003cffe4 003d0000 003cf000 00000000
0038:00000010  00001e00 00000000 7ffde000 00000000
```

5 dt _TEB 7ffde000 查看主线程中PEB结构指针(0x7ffdc000)

```
0:001> dt _TEB 7ffde000
ntdll!_TEB
   +0x000 NtTib            : _NT_TIB
   +0x01c EnvironmentPointer : (null)
   +0x020 ClientId         : _CLIENT_ID
   +0x028 ActiveRpcHandle  : (null)
   +0x02c ThreadLocalStoragePointer : (null)
   +0x030 ProcessEnvironmentBlock : 0x7ffdc000 _PEB
   +0x034 LastErrorValue   : 0
```

6 dt _PEB 0x7ffdc000 寻找LoaderLock的指针(0x7c99e0174)

```
+0x0a0 LoaderLock       : 0x7c99e174
```

7 dt_RTL_CRITICAL_SECTION 0x7c99e174 查看临界区状态，我们看到看到LockCount值为-1，那么我们通过给它设置“写”断点，从而在每次“关键”时刻予以监控。

```
0:001> dt _RTL_CRITICAL_SECTION 0x7c99e174
ntdll!_RTL_CRITICAL_SECTION
   +0x000 DebugInfo        : 0x7c99e1a0 _RTL_CRITICAL_SECTION_DEBUG
   +0x004 LockCount        : -1
   +0x008 RecursionCount   : 0
   +0x00c OwningThread     : (null)
   +0x010 LockSemaphore    : (null)
   +0x014 SpinCount        : 0
```

8 baw2 0x7c99e178 对LockCount设置写断点

9 g

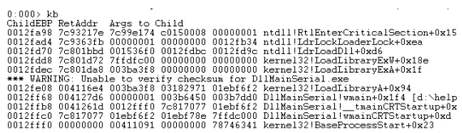
10kb 我们看到线程号是1，即Windbg插入的线程导致的断点，我们忽略之（我们看到关闭线程时也会进入临界区）

11 g

12 kb 同上，忽略之

13 g

14 kb 这次是主线程（0）触发了断点，断点原因是LdrLoadDll中要加锁。



我们使用IDA反编译LdrLoadDll，可以看到调用的位置

```
v4 = RtlDosApplyFileIsolationRedirection_Ustr(1, a3, &unk_7C99E214, &v11, &v14, &v17, 0, 0, 0);
 v5 = v4;
 if ( v4 >= 0 )
 {
   v9 = 1;
 }
 else
 {
   if ( v4 != -1072365560 )
     goto LABEL_6;
 }
 LdrLockLoaderLock(1, 0, &v10);
 ms_exc.disabled = 0;
```

15 g

16 kb 还是主线程（0）触发了断点，原因是LdrLoadDll中调用了LdrpLoadDll，该函数中需要进入临界区，这是第二次进临界区了。在《Best Practices for Creating DLLs》中有对这种现象允许的说明

```
The loader lock is recursive, which means that it can be acquired again by the same thread.
```

在LdrLoadDll中我们看到

```
LdrLockLoaderLock(1, 0, &v10);
 ms_exc.disabled = 0;
```

```
if ( LdrpTopLevelDllBeingLoaded )
{
  if ( ShowSnaps || LdrpShowRecursiveDllLoads || LdrpBreakOnRecursiveDllLoads )
  {
    DbgPrint("[%lx,%lx] LDR: Recursive DLL load\n");
    DbgPrint("[%lx,%lx]   Previous DLL being loaded: \"%wZ\"\n");
    DbgPrint("[%lx,%lx]   DLL being requested: \"%wZ\"\n");
    if ( LdrpCurrentDllInitializer )
      DbgPrint("[%lx,%lx]   DLL whose initializer was currently running: \"%wZ\"\n");
    else
      DbgPrint("[%lx,%lx]   No DLL initializer was running\n");
  }
}
LdrpTopLevelDllBeingLoaded = v17;
v6 = LdrpLoadDll(v9, a1, a2, v17, a4, 1);
```

在LdrpLoadDll中我们看到

```
loc_7C93656E:                           ; CODE XREF: LdrpLoadDll(x,x,x,x,x,x)+69Tj
            push    offset _LdrpLoaderLock
            call    _RtlEnterCriticalSection@4 ; RtlEnterCriticalSection(x)
            jmp     short loc_7C9364FC
```

```
0:000> kb
ChildEBP RetAddr  Args to Child
0012f8lc 7c936578 7c99e174 c0150008 00000000 ntdll!RtlEnterCriticalSection+0x15
0012fac8 7c93643d 00000000 001536f0 0012fdbc ntdll!LdrpLoadDll+0x75
0012fd70 7c801bbd 001536f0 0012fdbc 0012fd9c ntdll!LdrLoadDll+0x230
0012fdd8 7c801d72 7ffdfc00 00000000 00000000 kernel32!LoadLibraryExW+0x18e
0012fdec 7c801da8 003ba3f8 00000000 00000000 kernel32!LoadLibraryExA+0x1f
0012fe08 004116e4 003ba3f8 03102971 01ebf6f2 kernel32!LoadLibraryA+0x94
0012ff68 004127d6 00000001 003b6450 003b7dd0 DllMainSerial!wmain+0x1f4 [d:\help
0012ffb8 0041261d 0012fff0 7c817077 01ebf6f2 DllMainSerial!__tmainCRTStartup+0xJ
0012ffc0 7c817077 01ebf6f2 01ebf79e 7ffdc000 DllMainSerial!wmainCRTStartup+0xd [
0012fff0 00000000 00411091 00000000 78746341 kernel32!BaseProcessStart+0x23
```

17 g

18 kb 第三次进入临界区

```
0:000> kb
ChildEBP RetAddr  Args to Child
0012f5d4 7c937c90 7c99e174 00006d19 0012f6c8 ntdll!RtlEnterCriticalSection+0x15
0012f684 7c937e8f 7c920000 0012f6c8 00006d19 ntdll!LdrpGetProcedureAddress+0x117
0012f6d4 7c93c0bc 7c800000 7c920000 000180cc ntdll!LdrpSnapThunk+0x317
0012f758 7c93c3fb 00252010 002524a0 10018000 ntdll!LdrpSnapIAT+0x20e
0012f784 7c93c374 7ffdc000 001536f0 00000000 ntdll!LdrpHandleOneOldFormatImportDescriptor+0xcc
0012f79c 7c93b2c6 7ffdc000 001536f0 002524a0 ntdll!LdrpHandleOldFormatImportDescriptors+0x1f
0012f818 7c936227 001536f0 002524a0 c0150008 ntdll!LdrpWalkImportDescriptor+0x19e
0012fac8 7c93643d 00000000 001536f0 0012fdbc ntdll!LdrpLoadDll+0x24e
0012fd70 7c801bbd 001536f0 0012fdbc 0012fd9c ntdll!LdrLoadDll+0x230
0012fdd8 7c801d72 7ffdfc00 00000000 00000000 kernel32!LoadLibraryExW+0x18e
0012fdec 7c801da8 003ba3f8 00000000 00000000 kernel32!LoadLibraryExA+0x1f
0012fe08 004116e4 003ba3f8 03102971 01ebf6f2 kernel32!LoadLibraryA+0x94
0012ff68 004127d6 00000001 003b6450 003b7dd0 DllMainSerial!wmain+0x1f4 [d:\help others\dllmains
0012ffb8 0041261d 0012fff0 7c817077 01ebf6f2 DllMainSerial!__tmainCRTStartup+0x1a6 [f:\dd\vctoc
0012ffc0 7c817077 01ebf6f2 01ebf79e 7ffdc000 DllMainSerial!wmainCRTStartup+0xd [f:\dd\vctools\c
0012fff0 00000000 00411091 00000000 78746341 kernel32!BaseProcessStart+0x23
```

19 g 主线程第一次退出临界区

```
0:000> g
Breakpoint 0 hit
eax=00000000 ebx=0012f628 ecx=7c9300c4 edx=7c99e174 esi=100181b4 edi=00000000
eip=7c921114 esp=0012f5d4 ebp=0012f684 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
ntdll!RtlLeaveCriticalSection+0x34:
7c921114 c20400          ret     4
```

20 kb 主线程第四次进入临界区

```
ChildEBP RetAddr  Args to Child
0012f5d4 7c937c90 7c99e174 00006d19 0012f6c8 ntdll!RtlEnterCriticalSection+0x15
0012f684 7c937e8f 7c920000 0012f6c8 00006d19 ntdll!LdrpGetProcedureAddress+0x117
0012f6d4 7c93c0bc 7c800000 7c920000 00018070 ntdll!LdrpSnapThunk+0x317
0012f758 7c93c3fb 00252010 002524a0 10018000 ntdll!LdrpSnapIAT+0x20e
0012f784 7c93c374 7ffdc000 001536f0 00000000 ntdll!LdrpHandleOneOldFormatImportDescriptor+0xcc
0012f79c 7c93b2c6 7ffdc000 001536f0 002524a0 ntdll!LdrpHandleOldFormatImportDescriptors+0x1f
0012f818 7c936227 001536f0 002524a0 c0150008 ntdll!LdrpWalkImportDescriptor+0x19e
0012fac8 7c93643d 00000000 001536f0 0012fdbc ntdll!LdrpLoadDll+0x24e
0012fd70 7c801bbd 001536f0 0012fdbc 0012fd9c ntdll!LdrLoadDll+0x230
0012fdd8 7c801d72 7ffdfc00 00000000 00000000 kernel32!LoadLibraryExW+0x18e
0012fdec 7c801da8 003ba3f8 00000000 00000000 kernel32!LoadLibraryExA+0x1f
0012fe08 004116e4 003ba3f8 03102971 01ebf6f2 kernel32!LoadLibraryA+0x94
0012ff68 004127d6 00000001 003b6450 003b7dd0 DllMainSerial!wmain+0x1f4 [d:\help others\dllmains
0012ffb8 0041261d 0012fff0 7c817077 01ebf6f2 DllMainSerial!__tmainCRTStartup+0x1a6 [f:\dd\vctoc
0012ffc0 7c817077 01ebf6f2 01ebf79e 7ffdc000 DllMainSerial!wmainCRTStartup+0xd [f:\dd\vctools\c
0012fff0 00000000 00411091 00000000 78746341 kernel32!BaseProcessStart+0x23
```

21 g 主线程第二次退出临界区

```
0:000> g
Breakpoint 0 hit
eax=00000000 ebx=0012f628 ecx=7c92ff2d edx=7c99e174 esi=100181b8 edi=00000000
eip=7c921114 esp=0012f5d4 ebp=0012f684 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
ntdll!RtlLeaveCriticalSection+0x34:
7c921114 c20400          ret     4
```

22 g 有线程要进入临界区

```
0:000> g
Breakpoint 0 hit
eax=00000001 ebx=00000000 ecx=7c99e174 edx=00000000 esi=7ffdc000 edi=7ffde000
eip=7c92112b esp=0051fca8 ebp=0051fd1c iopl=0         nv up ei ng nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000282
ntdll!RtlTryEnterCriticalSection+0x13:
7c92112b 7518            jne     ntdll!RtlTryEnterCriticalSection+0x2d (7c921145) [br=1]
```

23 kb 这次是我们在代码中启动的工作线程（1）要尝试进入临界区

```
0:001> kb
ChildEBP RetAddr  Args to Child
0051fca4 7c9398b3 7c99e174 0051fd30 00000047 ntdll!RtlTryEnterCriticalSection+0x13
0051fd1c 7c92e457 0051fd30 7c920000 00000000 ntdll!_LdrpInitialize+0x8c
00000000 00000000 00000000 00000000 00000000 ntdll!KiUserApcDispatcher+0x7
```

24 ~ 查看线程 确定有两个线程了

```
0:001> ~
   0  Id: 1c9c.afc Suspend: 1 Teb: 7ffdf000 Unfrozen
.  1  Id: 1c9c.1e28 Suspend: 1 Teb: 7ffde000 Unfrozen
```

25 g

```
0:001> g
Breakpoint 0 hit
eax=00000000 ebx=00000000 ecx=7ffde000 edx=7c99e174 esi=7ffdc000 edi=7ffde000
eip=7c921015 esp=0051fca8 ebp=0051fd1c iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
ntdll!RtlEnterCriticalSection+0x15:
7c921015 7519            jne     ntdll!RtlEnterCriticalSection+0x30 (7c921030) [br=1]
```

26 kb 工作线程（1）要进入临界区，可是它不会进去的，因为它会被挂起

```
0:001> kb
ChildEBP RetAddr  Args to Child
0051fca4 7c944d2d 7c99e174 0051fd30 00000047 ntdll!RtlEnterCriticalSection+0x15
0051fd1c 7c92e457 0051fd30 7c920000 00000000 ntdll!_LdrpInitialize+0xf0
00000000 00000000 00000000 00000000 00000000 ntdll!KiUserApcDispatcher+0x7
```

27 g 死锁了

```
*BUSY* Debuggee is running..
```

28 control+break windbg要启动一个中断线程，中断线程触发了断点

```
Breakpoint 0 hit
eax=00000002 ebx=00000000 ecx=7c99e174 edx=00000000 esi=7ffdc000 edi=7ffdd000
eip=7c92112b esp=003dfca4 ebp=003dfd18 iopl=0         nv up ei ng nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000282
ntdll!RtlTryEnterCriticalSection+0x13:
7c92112b 7518            jne     ntdll!RtlTryEnterCriticalSection+0x2d (7c921145) [br=1]
```

29 ~ 查看线程，ID为2的就是windbg插入的线程

```
0:002> ~
   0  Id: 1c9c.afc Suspend: 1 Teb: 7ffdf000 Unfrozen
   1  Id: 1c9c.1e28 Suspend: 1 Teb: 7ffde000 Unfrozen
.  2  Id: 1c9c.1580 Suspend: 1 Teb: 7ffdd000 Unfrozen
```

30 ~0s 切换到主线程（0），发现主线程在内核态中出不来了

```
0:002> ~0s                                                          .    .
eax=00000000 ebx=100110dc ecx=00153e98 edx=7ffde1b4 esi=000007f4 edi=00000000
eip=7c92e514 esp=0012f52c ebp=0012f590 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
ntdll!KiFastSystemCallRet:
7c92e514 c3              ret
```

31 kb 查看主线程调用堆栈，确实是在等工作线程结束

```
0:000> kb
ChildEBP RetAddr  Args to Child
0012f528 7c92df5a 7c8025db 000007f4 00000000 ntdll!KiFastSystemCallRet
0012f52c 7c8025db 000007f4 00000000 00000000 ntdll!NtWaitForSingleObject+0xc
0012f590 7c802542 000007f4 ffffffff 00000000 kernel32!WaitForSingleObjectEx+
*** WARNING: Unable to verify checksum for D:\help others\DllMainSerial\Debu
0012f5a4 100113de 000007f4 ffffffff 00000001 kernel32!WaitForSingleObject+0x
0012f69c 10011b9b 10000000 00000001 00000000 DllWithoutDisableThreadLibraryC
0012f6e0 10011abf 10000000 00000001 00000000 DllWithoutDisableThreadLibraryC
0012f6f4 7c92118a 10000000 00000001 00000000 DllWithoutDisableThreadLibraryC
0012f714 7c93b5d2 100110dc 10000000 00000001 ntdll!LdrpCallInitRoutine+0x14
0012f81c 7c9362db 00000000 c0150008 00000000 ntdll!LdrpRunInitializeRoutines
0012fac8 7c93643d 00000000 001536f0 0012fdbc ntdll!LdrpLoadDll+0x3e5
0012fd70 7c801bbd 001536f0 0012fdbc 0012fd9c ntdll!LdrLoadDll+0x230
0012fdd8 7c801d72 7ffdfc00 00000000 00000000 kernel32!LoadLibraryExW+0x18e
0012fdec 7c801da8 003ba3f8 00000000 00000000 kernel32!LoadLibraryExA+0x1f
0012fe08 004116e4 003ba3f8 03182971 01ebf6f2 kernel32!LoadLibraryA+0x94
0012ff68 004127d6 00000001 003b6450 003b7dd0 DllMainSerial!wmain+0x1f4 [d:\h
0012ffb8 00412e1d 0012fff0 7c817077 01ebf6f2 DllMainSerial!__tmainCRTStartup
0012ffc0 7c817077 01ebf6f2 01ebf78e 7ffdc000 DllMainSerial!wmainCRTStartup+0
0012fff0 00000000 00411091 00000000 78746341 kernel32!BaseProcessStart+0x23
```

32 ~1s 切换到工作线程，发现它也在内核态中出不来了

```
0:000> ~1s
eax=00000000 ebx=00000000 ecx=7ffde000 edx=7c99e174 esi=7c99e174 edi=0000000(
eip=7c92e514 esp=0051fc14 ebp=0051fc9c iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=0000024(
ntdll!KiFastSystemCallRet:
7c92e514 c3              ret
```

33 kb 查看工作线程调用堆栈

```
0:001> kb
ChildEBP RetAddr  Args to Child
0051fc10 7c92df5a 7c939b23 000007d8 00000000 ntdll!KiFastSystemCallRet
0051fc14 7c939b23 00000748 00000000 00000000 ntdll!NtWaitForSingleObject+0xc
0051fc9c 7c921046 0199e174 7c944d2d 7c99e174 ntdll!RtlpWaitForCriticalSection+0x132
0051fca4 7c944d2d 7c99e174 0051fd30 00000047 ntdll!RtlEnterCriticalSection+0x46
0051fd1c 7c92e457 0051fd30 7c920000 00000000 ntdll!_LdrpInitialize+0xf0
00000000 00000000 00000000 00000000 00000000 ntdll!KiUserApcDispatcher+0x7
```

34 dt _RTL_CRITICAL_SECTION 0x7c99e174 查看临界区所有权，从线程TID中我们可以看到，临界区的确是被主线程占着。

```
0:001> dt _RTL_CRITICAL_SECTION 0x7c99e174
Dll!ithoutDisableThreadLibraryCalls_A!_RTL_CRITICAL_SECTION
   +0x000 DebugInfo        : 0x7c99e1a0 _RTL_CRITICAL_SECTION_DEBUG
   +0x004 LockCount        : 3
   +0x008 RecursionCount   : 2
   +0x00c OwningThread     : 0x00000afc
   +0x010 LockSemaphore    : 0x000007d8
   +0x014 SpinCount        : 0
```

上一篇 下一篇

发表评论

提 交

查看评论

7楼 Breaksoftware 2014-06-04 13:32

[reply]qweqweqwexdd3[/reply] 过奖，相互交流学习。

6楼 qweqweqwexdd3 2014-06-02 14:00

lz的调试技术很高超，学习！

5楼 mishio 2012-11-30 11:27

主要是被2010以前的manifest给整坏了习惯，发布出去的要么自己安装VS的redist，要不就是需要手工配置…… 其实后来发现VC2010和VC2012的MD编译出来的都只需要配置上一个DLL就可以了，但是之前的就没有再调整^^

4楼 Breaksoftware 2012-11-30 10:25

[reply]mishio[/reply] 是有可能的，一般比较大型的工程，需要公用模块时就需要使用DLL，这样方便重用。动dllmain的情况一般是为了管理资源，一般情况下是很少用到。

3楼 mishio 2012-11-29 23:14

[reply]Breaksoftware[/reply] =v= 其实自己一般都不写DLLMAIN，好像，也很少用DLL，大部分用静态库的

更多评论（7）