WinDbg 软件调试
经典 BUG 分析报告

THINCT

February 19, 2024

线程死锁

临界区互锁

线程死锁

临界区互锁

临界区互锁

```
4   CRITICAL_SECTION cs1, cs2;
5
6   void DeadlockThread1(int threadId)
7   {
8       EnterCriticalSection(&cs1);
9       std::cout << "Thread " << threadId << " entered cs1\n";
10      Sleep(100); // 让线程暂停一段时间，以便另一个线程有机会执行
11      EnterCriticalSection(&cs2);
12      std::cout << "Thread " << threadId << " entered cs2\n";
13      LeaveCriticalSection(&cs2);
14      LeaveCriticalSection(&cs1);
15  }
16
17  void DeadlockThread2(int threadId)
18  {
19      EnterCriticalSection(&cs2);
20      std::cout << "Thread " << threadId << " entered cs2\n";
21      Sleep(100); // 让线程暂停一段时间，以便另一个线程有机会执行
22      EnterCriticalSection(&cs1);
23      std::cout << "Thread " << threadId << " entered cs1\n";
24      LeaveCriticalSection(&cs1);
25      LeaveCriticalSection(&cs2);
26  }
```

Figure 1: **思考:** 为什么会产生死锁呢?

点击查看完整源代码

临界区互锁

```
0:006> !cs -l
-------------------------------------
DebugInfo = 0x77e64a40
Critical section = 0x00405058 (CriticalSectionDead!cs1+0x0)
LOCKED
LockCount = 0x1
WaiterWoken = No
OwningThread = 0x0000491c
RecursionCount = 0x1
LockSemaphore = 0xFFFFFFFF
SpinCount = 0x020007d0
-------------------------------------
DebugInfo = 0x77e64a60
Critical section = 0x00405070 (CriticalSectionDead!cs2+0x0)
LOCKED
LockCount = 0x1
WaiterWoken = No
OwningThread = 0x00004eac
RecursionCount = 0x1
LockSemaphore = 0xFFFFFFFF
SpinCount = 0x020007d0
```

!cs -l

是一个 *WinDbg* 的扩展命令，用于列出所有当前存在的临界区以及它们的状态。

Critical Section

这行显示了一个临界区的地址，以及它属于哪个模块和哪个临界区变量。在这里，临界区属于 *CriticalSectionDead* 模块，并且它的名字是 *cs1*。

LockCount

LockCount 表示这个临界区被锁定的次数。

SpinCount

SpinCount 表示在尝试获取临界区锁定时，线程会进行自旋（忙等待）的次数。如果自旋后仍然无法获取锁，线程就会进入睡眠状态。

OwningThread

OwningThread 显示当前拥有这个临界区的线程的线程 *ID*。

RecursionCount

RecursionCount 表示拥有这个临界区的线程已经重新进入这个临界区的次数。在某些情况下，一个线程可能会多次进入同一个临界区，这个计数就是用来跟踪这种情况的。

临界区互锁

DebugInfo

DebugInfo=0x77e64a40 是指向调试信息的指针。

LOCKED

这表示该临界区当前是锁定的状态，也就是说，有一个线程正在使用这个临界区，其他线程不能进入。

WaiterWoken

这表示目前没有线程在等待这个临界区被释放。如果有一个或多个线程在等待，这个值会是 *Yes*。

LockSemaphore

LockSemaphore 是一个信号量，用于实现临界区的锁定和解锁。*0xFFFFFFFF* 通常表示信号量当前不可用。

临界区互锁

```
0:006> !locks

CritSec CriticalSectionDead!cs1+0 at 00405058
WaiterWoken No
LockCount 1
RecursionCount 1
OwningThread 491c
EntryCount 0
ContentionCount 1
*** Locked

CritSec CriticalSectionDead!cs2+0 at 00405070
WaiterWoken No
LockCount 1
RecursionCount 1
OwningThread 4eac
EntryCount 0
ContentionCount 1
*** Locked

Scanned 5 critical sections
```

!locks

显示调试目标进程中所有被锁定的临界区（*Critical Sections*）的信息。

EntryCount

表示尝试进入临界区的次数。*0* 通常表示没有线程尝试进入这个临界区，但这可能是一个不准确的值，因为它取决于具体的调试器扩展实现。

ContentionCount

表示线程在尝试获取这个临界区时发生争用的次数。*1* 表示发生了一次争用。

临界区互锁

使用 ~*kvn 查看线程状态

```
   4  Id: 4078.491c Suspend: 1 Teb: 010e2000 Unfrozen
 # ChildEBP RetAddr      Args to Child
00 019ff8d8 77d9f999     00405074 00000000 00405070 ntdll!NtWaitForAlertByThreadId+0xc (FPO: [2,0,0])
01 019ff908 77d9f6ed     00000000 00000000 00405070 ntdll!RtlpWaitOnAddressWithTimeout+0x64 (FPO: [Non-Fpo])
02 019ff9a8 77d8011a     004013a0 00000001 00000001 ntdll!RtlpWaitOnCriticalSection+0x18d (FPO: [Non-Fpo])
03 019ff9e0 77d7ff69     00000000 019ff9fc 004013ef ntdll!RtlpEnterCriticalSectionContended+0x1aa (FPO: [Non-Fpo])
04 019ff9ec 004013ef     00405070 004013a0 019ffa0c ntdll!RtlEnterCriticalSection+0x49 (FPO: [1,0,0])
05 019ff9fc 7679fcc9     00000001 7679fcb0 019ffa68 CriticalSectionDead!DeadlockThread1+0x4f (FPO: [Non-Fpo]) (CONV: cdecl) [E:\debug-demo
06 019ffa0c 77da7c5e     00000001 4659192c 00000000 KERNEL32!BaseThreadInitThunk+0x19 (FPO: [Non-Fpo])
07 019ffa68 77da7c2e     ffffffff 77dc8c10 00000000 ntdll!__RtlUserThreadStart+0x2f (FPO: [SEH])
08 019ffa78 00000000     004013a0 00000001 00000000 ntdll!_RtlUserThreadStart+0x1b (FPO: [Non-Fpo])

   5  Id: 4078.4eac Suspend: 1 Teb: 010e5000 Unfrozen
 # ChildEBP RetAddr      Args to Child
00 01b3fad0 77d9f999     0040505c 00000000 00405058 ntdll!NtWaitForAlertByThreadId+0xc (FPO: [2,0,0])
01 01b3fb00 77d9f6ed     00000000 00000000 00405058 ntdll!RtlpWaitOnAddressWithTimeout+0x64 (FPO: [Non-Fpo])
02 01b3fba0 77d8011a     00401430 00000002 00000002 ntdll!RtlpWaitOnCriticalSection+0x18d (FPO: [Non-Fpo])
03 01b3fbd8 77d7ff69     00000000 01b3fbf4 0040147f ntdll!RtlpEnterCriticalSectionContended+0x1aa (FPO: [Non-Fpo])
04 01b3fbe4 0040147f     00405058 00401430 01b3fc04 ntdll!RtlEnterCriticalSection+0x49 (FPO: [1,0,0])
05 01b3fbf4 7679fcc9     00000002 7679fcb0 01b3fc60 CriticalSectionDead!DeadlockThread2+0x4f (FPO: [Non-Fpo]) (CONV: cdecl) [E:\debug-demo
06 01b3fc04 77da7c5e     00000002 46751f24 00000000 KERNEL32!BaseThreadInitThunk+0x19 (FPO: [Non-Fpo])
07 01b3fc60 77da7c2e     ffffffff 77dc8c10 00000000 ntdll!__RtlUserThreadStart+0x2f (FPO: [SEH])
08 01b3fc70 00000000     00401430 00000002 00000000 ntdll!_RtlUserThreadStart+0x1b (FPO: [Non-Fpo])
```

4 Id: 4078.491c Suspend: 1 Teb: 010e2000 Unfrozen
00405070 ntdll!RtlEnterCriticalSection+0x49

根据!cs -l 可知, 线程 491c 拥有临界区 5058, 结合上面线程的栈帧发现, 当前线程试图进入临界区 5070.

5 Id: 4078.4eac Suspend: 1 Teb: 010e5000 Unfrozen
00405058 ntdll!RtlEnterCriticalSection+0x49

根据!cs -l 可知, 线程 4eac 拥有临界区 5070, 结合上面线程的栈帧发现, 当前线程试图进入临界区 5080.

总结:

综上所述, 分别是 4 号和 5 号线程发生了临界区互锁造成了程序卡死.