方亮的专栏

[原]DllMain中不当操作导致死锁问题的分析--死锁介绍

2012-11-2 阅读1419 评论0

最近在网上看到一些关于在DllMain中不当操作导致死锁的问题，也没找到比较确切的解答，这极大吸引了我研究这个问题的兴趣。我花了一点时间研究了下，正好也趁机研究了下进程对DllMain的调用规律。因为整个研究篇幅比较长，我觉得还是分开写比较能突出重点。本文先说说死锁。（转载请指明出于breaksoftware的csdn博客）

介绍死锁之前，我说一个我小时候听过的一个故事：

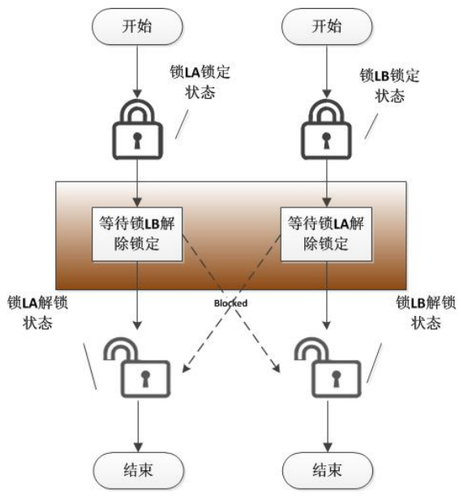
某国际实验机构将在全球各著名小学做个团队合作的实验。他们在中国选定了一个“被安排的”学校，然后“随机”选出一些学生，让这些学生作为实验的样本参与中国区的实验。实验是这样的：他们在N根细绳一头捆着一支短粉笔，将这些粉笔放到一个细口瓶中。该瓶口只能容一支粉笔自由出入。然后绳的另一头放在学生的手中，告知他们要迅速将自己手中绳子捆住的粉笔从瓶子中拽出来。中国学生经过讨论后，决定出他们的方案。于是123之后，“聪明谦让”的中国学生“一个个”并迅速的将各自的粉笔拽了出来。而同样的实验，在“苦大仇深”的外国学生中结果却不理想。因为他们同时一起往外拽绳子，导致所有的粉笔都卡在瓶口……

这个故事影响了我很久，我一直在思考：外国人这么笨么？但是现在我回忆这个故事，却想到了这个实验中发生的一些现象和我们在编程中遇到的一些问题是如此的类似。想想，“中国学生”的思路就是“序列化执行”，而外国学生的现象就是因为“竞争”而导致了“死锁”。

回到正题，我想熟悉计算机的同学应该对“死锁”这个概念并不陌生。我们看一下wiki对Deadlock这个词的解释：

```
A deadlock is a situation in which two or more competing actionsare each waiting for the other to finish, and thus neither ever does。
```

也就是说：多个操作相互等待其他结束从而导致它们都无法结束的一种场景。为简单描述，我以两个相互影响因素来描述死锁。



上图中红色的部分就是故事中“所有粉笔卡在瓶口”那个纠结的时期。于是左右两个例程都纠结于此，不再往下执行。

以下我列出比较典型的死锁案例

```
// A线程中 hEventA未激活
WaitforSingleObject(hEventA, INFINITE);
SetEvent(hEventB);
```

```
// B线程中 hEventB未激活
WaitforSingleObject(hEventB, INFINITE);
SetEvent(hEventA);
```

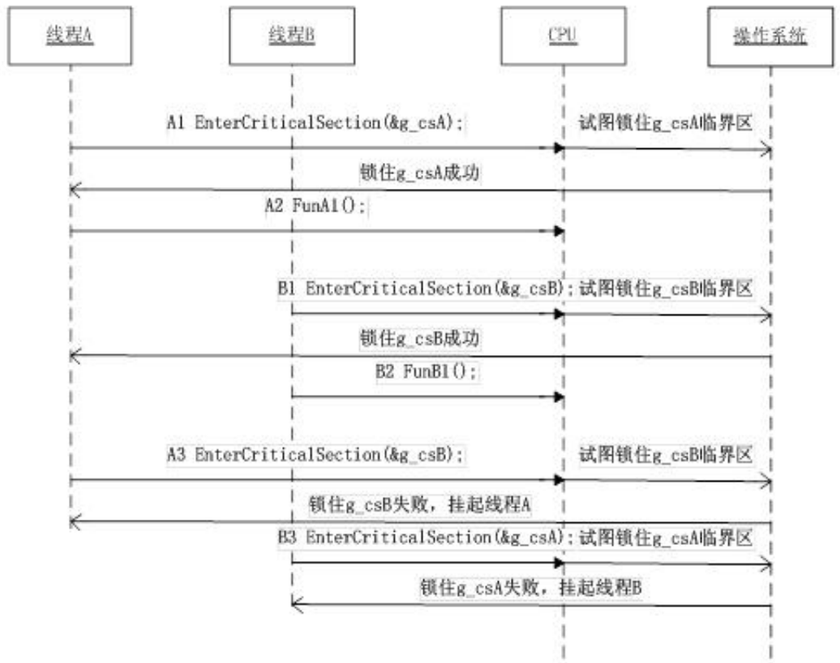
线程A、B都在等待对方释放一个事件再释放对方要获得的事件，否则它们都一直等下去。系统看不得它们不工作而傻等，于是就把它们都挂起了。它们就死锁了。想想这和古惑仔片中两个黑帮（线程A、B）相互扣押了对方的卧底（线程A扣押了hEventB，线程B扣押了hEventA），但是两方的老大都要求对方先放人(A线程WaitforSingleObject(hEventA, INFINITE(无线等待)); B线程WaitforSingleObject(hEventB, INFINITE(无线等待));)，于是他们就这样僵持下去了(死锁)。如果某方老大富有智慧，会先要求对方一段时间先放人以不失身份，但是等一会儿后(A线程WAIT_TIMEOUT == WaitforSingleObject(hEventA, 1000(小等一下));)，自己先把对方的人先放了(SetEvent (hEventB);)。然后对方老大释怀了(线程BWAIT_OBJECT_0==WaitforSingleObject(hEventB, INFINITE);)，再释放潜入的卧底(SetEvent (hEventA);)，这样的问题就解决了。可是编程中，出于种种原因，我们很难在一开始就发现是哪儿我们没转过弯。就像我题目中描述的问题，很多人无法理解为什么就在DllMain中加了点代码就死锁了，甚至代码中不包括一点”等“性质的函数（其实是有，只是很隐蔽）。

我们再看一个教科书式的死锁案例

```
// A线程
EnterCriticalSection(&g_csA);   //要进入临界区g_csA
FunA1();                        //该函数不影响死锁这个必然的结果，只是如果这个函数执行的消耗的时间很完美，将导致死锁出现的概率大增
EnterCriticalSection(&g_csB);   //要进入临界区g_csB
FunA2();
LeaveCriticalSection(&g_csB);   //要退出临界区g_csB
LeaveCriticalSection(&g_csA);   //要退出临界区g_csA
```

```
// B线程
EnterCriticalSection(&g_csB);   //要进入临界区g_csB
FunB1();                        //该函数不影响死锁这个必然的结果，只是如果这个函数执行的消耗的时间很完美，将导致死锁出现的概率大增
EnterCriticalSection(&g_csA);   //要进入临界区g_csA
FunB2();
LeaveCriticalSection(&g_csA);   //要退出临界区g_csA
LeaveCriticalSection(&g_csB);   //要退出临界区g_csB
```

如果A线程进入g_csA临界区并运行到FunA1()时，B线程进入g_csB临界区，并运行了FunB1()。这个时候，当A线程从FunA1()中退出后，试图进入临界区g_csB时是进入不了的，因为此时B线程还在运行FunB1()，B还在g_csB临界区中，于是A线程等待B退出临界区g_csB。而B线程运行完FunB1()时，将试图进入临界区g_csA，它也进入不了，因为线程A的操作在这个临界区中。于是B就等待A线程退出g_csA。它们相互等待对方退出，而自己却不去主动退出，这样就是挤破了头也没法一起进行下去。最后说一下，此处的FunA1()，FunB1()并不影响死锁产生的结果，但是会影响死锁产生的概率。



请大家记住这两个例子，我们会在之后分析的DllMain中不当操作导致死锁的案例中再次看到它们的身影。

上一篇 下一篇