[秒杀多线程第五篇 经典线程同步 关键段CS](http://blog.csdn.net/morewindows/article/details/7442639)

分类： [Windows多线程](http://blog.csdn.net/MoreWindows/article/category/1115271) [Windows编程](http://blog.csdn.net/MoreWindows/article/category/862060)2012-04-11 09:06 49444人阅读 [评论](http://blog.csdn.net/morewindows/article/details/7442639#comments)(124) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/morewindows/article/details/7442639#report)

[多线程](http://www.csdn.net/tag/%e5%a4%9a%e7%ba%bf%e7%a8%8b)[thread](http://www.csdn.net/tag/thread)[microsoft](http://www.csdn.net/tag/microsoft)[fun](http://www.csdn.net/tag/fun)[null](http://www.csdn.net/tag/null)[struct](http://www.csdn.net/tag/struct)

上一篇《[秒杀多线程第四篇 一个经典的多线程同步问题](http://blog.csdn.net/morewindows/article/details/7442333)》提出了一个经典的多线程同步互斥问题，本篇将用关键段CRITICAL\_SECTION（临界区）来尝试解决这个问题。

本文首先介绍下如何使用关键段，然后再深层次的分析下关键段的实现机制与原理。

关键段CRITICAL\_SECTION一共就四个函数，使用很是方便。下面是这四个函数的原型和使用说明。

函数功能：初始化

函数原型：

void InitializeCriticalSection(LPCRITICAL\_SECTION lpCriticalSection);

函数说明：定义关键段变量后必须初始化。

函数功能：销毁

函数原型：

void DeleteCriticalSection(LPCRITICAL\_SECTION lpCriticalSection);

函数说明：用完之后记得销毁。

函数功能：进入关键区域

函数原型：

void EnterCriticalSection(LPCRITICAL\_SECTION lpCriticalSection);

函数说明：系统保证各线程互斥的进入关键区域。

函数功能：离开关关键区域

函数原型：

void LeaveCriticalSection(LPCRITICAL\_SECTION lpCriticalSection);

然后在经典多线程问题中设置二个关键区域。一个是主线程在递增子线程序号时，另一个是各子线程互斥的访问输出全局资源时。详见代码：

**[cpp]** [view plaincopy](http://blog.csdn.net/morewindows/article/details/7442639)

1. #include <stdio.h>
2. #include <process.h>
3. #include <windows.h>
4. **long** g\_nNum;
5. unsigned **int** \_\_stdcall Fun(**void** \*pPM);
6. **const** **int** THREAD\_NUM = 10;
7. //关键段变量声明
8. CRITICAL\_SECTION  g\_csThreadParameter, g\_csThreadCode;
9. **int** main()
10. {
11. printf("     经典线程同步 关键段\n");
12. printf(" -- by MoreWindows( http://blog.csdn.net/MoreWindows ) --\n\n");
14. //关键段初始化
15. InitializeCriticalSection(&g\_csThreadParameter);
16. InitializeCriticalSection(&g\_csThreadCode);
18. **HANDLE**  handle[THREAD\_NUM];
19. g\_nNum = 0;
20. **int** i = 0;
21. **while** (i < THREAD\_NUM)
22. {
23. EnterCriticalSection(&g\_csThreadParameter);//进入子线程序号关键域
24. handle[i] = (**HANDLE**)\_beginthreadex(NULL, 0, Fun, &i, 0, NULL);
25. ++i;
26. }
27. WaitForMultipleObjects(THREAD\_NUM, handle, TRUE, INFINITE);
29. DeleteCriticalSection(&g\_csThreadCode);
30. DeleteCriticalSection(&g\_csThreadParameter);
31. **return** 0;
32. }
33. unsigned **int** \_\_stdcall Fun(**void** \*pPM)
34. {
35. **int** nThreadNum = \*(**int** \*)pPM;

！！！！如果试图离开一个并不是有当前线程拥有的临界区，那么将是一个非常严重的错误。

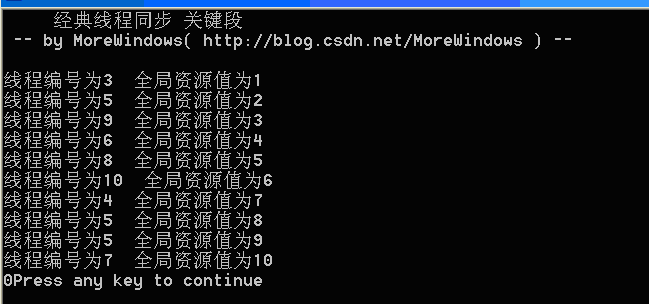
像下面这样是错误的，主线程中进入的临界区在在子线程中Leave。

1. LeaveCriticalSection(&g\_csThreadParameter);//离开子线程序号关键区域
3. Sleep(50);//some work should to do
5. EnterCriticalSection(&g\_csThreadCode);//进入各子线程互斥区域
6. g\_nNum++;
7. Sleep(0);//some work should to do
8. printf("线程编号为%d  全局资源值为%d\n", nThreadNum, g\_nNum);
9. LeaveCriticalSection(&g\_csThreadCode);//离开各子线程互斥区域
10. **return** 0;
11. }

注意：！！！！如果一个线程调用LeaveCriticalSection对不属于自己的临界区对象操作时会发生错误，将导致原本拥有这个对象的线程调用EnterCriticalSection时无限期的等待。

这个例子其实是为了证明关键段不能这样使用。上面的阴影部分的两句是不能这么使用的。

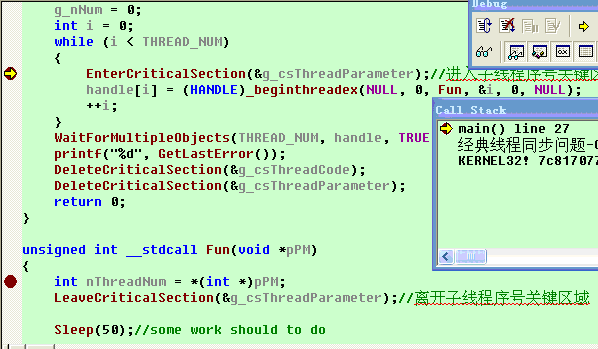
运行结果如下图：



可以看出来，各子线程已经可以互斥的访问与输出全局资源了，但主线程与子线程之间的同步还是有点问题。

       这是为什么了？

要解开这个迷，最直接的方法就是先在程序中加上断点来查看程序的运行流程。断点处置示意如下：



然后按F5进行调试，正常来说这两个断点应该是依次轮流执行，但实际调试时却发现不是如此，主线程可以多次通过第一个断点即

       EnterCriticalSection(&g\_csThreadParameter);//进入子线程序号关键区域

这一语句。这说明主线程能多次进入这个关键区域！找到主线程和子线程没能同步的原因后，下面就来分析下原因的原因吧^\_^

先找到关键段CRITICAL\_SECTION的定义吧，它在WinBase.h中被定义成RTL\_CRITICAL\_SECTION。而RTL\_CRITICAL\_SECTION在WinNT.h中声明，它其实是个结构体：

typedef struct \_RTL\_CRITICAL\_SECTION {

    PRTL\_CRITICAL\_SECTION\_DEBUG DebugInfo; //调试信息

    LONG LockCount; //

    LONG RecursionCount; //

    HANDLE OwningThread; // from the thread's ClientId->UniqueThread

    HANDLE LockSemaphore; //

    DWORD SpinCount; //

} RTL\_CRITICAL\_SECTION, \*PRTL\_CRITICAL\_SECTION;

各个参数的解释如下：

第一个参数：PRTL\_CRITICAL\_SECTION\_DEBUG DebugInfo;

调试用的。

第二个参数：LONG LockCount;

初始化为-1，n表示有n个线程在等待。

第三个参数：LONG RecursionCount;

表示该关键段的拥有线程对此资源获得关键段次数，初为0。

第四个参数：HANDLE OwningThread;

即拥有该关键段的线程句柄，微软对其注释为——from the thread's ClientId->UniqueThread

第五个参数：HANDLE LockSemaphore;

实际上是一个自复位事件。

第六个参数：DWORD SpinCount;

旋转锁的设置，单CPU下忽略

由这个结构可以知道关键段会记录拥有该关键段的线程句柄即**关键段是有“线程所有权”概念的**。事实上它会用第四个参数OwningThread来记录获准进入关键区域的线程句柄，如果这个线程再次进入，EnterCriticalSection()会更新第三个参数RecursionCount以记录该线程进入的次数并立即返回让该线程进入。其它线程调用EnterCriticalSection()则会被切换到等待状态，一旦拥有线程所有权的线程调用LeaveCriticalSection()使其进入的次数为0时，系统会自动更新关键段并将等待中的线程换回可调度状态。

**关键段可以用于线程间的互斥，但不可以用于同步。**

 但是有一点很容易照成误解应该强调一下，以下  
摘自《windows并发编程指南》第六章，P160：  
离开一个未拥有的临界区  
如果试图离开一个并不是有当前线程拥有的临界区，那么将是一个非常严重的错误。这意味着  
  
在程序存在一个错误，并且在发生这种情况时不会立即出现表示错误的信息。既不会返回错误  
  
码也不会抛出异常。外表上看来，所有的事情都是进展顺利，但却留下了一个定时炸弹。  
如果在LeaveCriticalSection时没有拥有临界区，那么接下来所有对EnterCriticalSection的  
  
调用都将被阻塞。这实际上使得所有在随后试图使用这个临界区的线程都将形成死锁(这里是  
  
说，调用Leave的那个线程如果接下来再调用Enter的话这个线程就会永远阻塞，而且在它调用了Enter之后，所有使用临界区的线程都将阻塞包括之前拥有这个临界区的线程。但是，在调用了Leave之后和调用Enter之前，拥有临界区的那个线程还是可以Enter进去的)。如果未拥有临界区的线程在尝试执行离开临界区操作时，这个临界区正被另一个线程所拥有，那么当前所有者仍然可以递归地获取和释放这个临界区。但当所有者退出这个临界区时，它将被永久性地破坏：随后将发生于前面相同的行为，即系统中的任何线程在随后的调用Enter时都将被永远地阻塞。

另外，由于将线程切换到等待状态的开销较大，因此为了提高关键段的性能，Microsoft将旋转锁合并到关键段中，这样EnterCriticalSection()会先用一个旋转锁不断循环，尝试一段时间才会将线程切换到等待状态。下面是配合了旋转锁的关键段初始化函数

函数功能：初始化关键段并设置旋转次数

函数原型：

BOOL InitializeCriticalSectionAndSpinCount(

  LPCRITICAL\_SECTION lpCriticalSection,

  DWORD dwSpinCount);

函数说明：旋转次数一般设置为4000。

函数功能：修改关键段的旋转次数

函数原型：

DWORD SetCriticalSectionSpinCount(

  LPCRITICAL\_SECTION lpCriticalSection,

  DWORD dwSpinCount);

《Windows核心编程》第五版的第八章推荐在使用关键段的时候同时使用旋转锁，这样有助于提高性能。值得注意的是如果主机只有一个处理器，那么设置旋转锁是无效的。无法进入关键区域的线程总会被系统将其切换到等待状态。

最后总结下关键段：

**1．关键段共初始化化、销毁、进入和离开关键区域四个函数。**

**2．关键段可以解决线程的互斥问题，但因为具有“线程所有权”，所以无法解决同步问题。**

**3．推荐关键段与旋转锁配合使用。**

下一篇《[秒杀多线程第六篇 经典线程同步 事件Event](http://blog.csdn.net/morewindows/article/details/7445233)》将介绍使用事件Event来解决这个经典线程同步问题。

[](http://blog.csdn.net/luochengor)

楼主，你好，在关键段结构体中第三个参数：LONGRecursionCount; 表示该关键段的拥有线程对此资源获得关键段次数，初为0。我想问一下在什么情况下会让这个参数的计数增加呢？也就是在什么场景下会在本线程调用LeaveCriticalSection之前连续调用EnterCriticalSection两次以上呢？本线程在调用LeaveCriticalSection之前已经获取到访问权了，还需要在调用EnterCriticalSection吗？麻烦楼主给指点一下。

[](http://blog.csdn.net/ponder2011)

为什么把线程数改为100，他什么都不输出？

回复ponder2011：哈哈 去看 WaitForMultipleObjects的第一参数 个数有个最大值

[](http://blog.csdn.net/ohmygodohyeah)

引用“MoreWindows”的评论：回复w2wfyh：这个例子其实是为了证明关键段不能这样使用，子线程中调用的...  
ohmygod！ 就是这个问题，我已经纠结很久了，始终没想明白为什么主线程中的Level会在子线程中使用。同时也在想若真的这样使用了，那么程序又是怎么执行的！一直 没想明白，LZ能点一下么，谢谢咯！

38楼 [DarkHorse](http://blog.csdn.net/lovewubo) 2013-06-27 17:45发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7442639#reply)

[http://avatar.csdn.net/9/2/6/3_lovewubo.jpg](http://blog.csdn.net/lovewubo)

MSDN:  
If a thread calls LeaveCriticalSection when it does not have ownership of the specified critical section object, an error occurs that may cause another thread using EnterCriticalSection to wait indefinitely.   
  
如果一个线程调用LeaveCriticalSection对不属于自己的临界区对象操作时会发生错误，将导致原本拥有这个对象的线程调用EnterCriticalSection时无限期的等待

33楼 [亚细亚](http://blog.csdn.net/yaxiya) 2013-05-23 15:56发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7442639#reply)

[](http://blog.csdn.net/yaxiya)

LZ对临界区的讲解和理解都很深刻，膜拜了。  
但是有一点很容易照成误解应该强调一下，以下  
摘自《windows并发编程指南》第六章，P160：  
离开一个未拥有的临界区  
如果试图离开一个并不是有当前线程拥有的临界区，那么将是一个非常严重的错误。这意味着  
  
在程序存在一个错误，并且在发生这种情况时不会立即出现表示错误的信息。既不会返回错误  
  
码也不会抛出异常。外表上看来，所有的事情都是进展顺利，但却留下了一个定时炸弹。  
如果在LeaveCriticalSection时没有拥有临界区，那么接下来所有对EnterCriticalSection的  
  
调用都将被阻塞。这实际上使得所有在随后试图使用这个临界区的线程都将形成死锁(这里是  
  
说，调用Leave的那个线程如果接下来再调用Enter的话这个线程就会永远阻塞，而且在它调用了Enter之后，所有使用临界区的线程都将阻塞包括之前拥有这个临界区的线程。但是，在调用了Leave之后和调用Enter之前，拥有临界区的那个线程还是可以Enter进去的)。如果未拥有临界区的线程在尝试执行离开临界区操作时，这个临界区正被另一个线程所拥有，那么当前所有者仍然可以递归地获取和释放这个临界区。但当所有者退出这个临界区时，它将被永久性地破坏：随后将发生于前面相同的行为，即系统中的任何线程在随后的调用Enter时都将被永远地阻塞。

回复lindeshi：LZ讲得很详细，通俗易懂，初学也能看得很明白。但层主这点补充的很有用啊，LZ的例子针对这点并不是很典型，差点漏过去了。。

[http://avatar.csdn.net/1/4/0/3_linrulei11.jpg](http://blog.csdn.net/linrulei11)

想请问一下，为什么输出的线程ID会有重复的？谢谢

回复linrulei11：应该是一只输出9对吧？  
因为他传递的是地址，传递的是&i，当到最后的时候i已经是9了，并且该地址传递进去了。  
所以即使之前i=1，但是由于最后传递的时候是9，并且传递的是地址，并非值，所以一直是9.

回复linrulei11：这是因为关键段不能用于线程同步。

回复MoreWindows：标号不一样，就代表关键代码段不能同步？  
  
标号不一样是因为你使用了地址传递！既然不能同步，为什么全局的值还是递增的呢？  
  
另外，子线程和主线程同步不同步有什么意义呢？线程本来就是不同步的，所谓线程同步是针对修改变量的顺序要做到同步。只要被修改的变量被同步了，那么线程就是同步的。

回复MoreWindows：不是很明白。。。即使不能同步，但是怎么能输出循环过的值呢？比如i=9,后又怎么会再输出6，8呢？是临界区有记录吗？就比如房卡会记住门牌号，但怎么会记往是谁在使用呢？

回复passball：“线程所有权”只要记录下线程的ID号就可以了，这个ID号是整个系统唯一的。

回复MoreWindows：线程ID应该是唯一的吧？也就是说虽然是同一个线程函数，但每次循环进入一次函数就会分配唯一的ID，对吗？  
那么应该是每一个i都要对应一个线程ID啊，即使有先后，所有的i也应该都输出啊，为什么有的i没有了呢？  
另外，同一个线程可以重复进入临界区，以结构体中的计数为准，但是多个enter（）可以用一个leave（）就结束了吧？

回复passball：你要看下《操作系统》(汤子瀛版)，不然很多知识点会搞混淆的。  
对于“同一个线程可以重复进入临界区，以结构体中的计数为准，但是多个enter（）可以用一个leave（）就结束了吧？”  
这个不能，因此关键段不但会记录线程ID号，还会记录线程占用关键段次数，只有占用次数为0后，该关键段才能被其它线程占用。

引用“MoreWindows”的评论：回复passball：你要看下《操作系统》(汤子瀛版)，不然很多知识点会搞...  
“同一个线程可以重复进入临界区，以结构体中的计数为准，但是多个enter（）可以用一个leave（）就结束了吧？”  
这句话如果出一个判断题的话,估计大部分人都会认为同一个线程不能重复进入临界区品;测试了你的代码,受益匪浅啊;THANKS!

回复MoreWindows：谢谢回复，我已经懂了！收获很多啊！

[http://avatar.csdn.net/2/9/2/3_jiancaesar.jpg](http://blog.csdn.net/JianCaesar)

对于旋转锁的概念不是很清楚，为什么单核，线程就会直接进入等待状态？和多核有什么区别吗？

回复JianCaesar：呵呵，细说的话，可以单独开一篇文章了。因此在这简单的回答下。假设一个单核CPU在执行线程A，而这个线程A是要等待线程B离开关键区域才能继续执行，因此线程A不应该再占用CPU了，因为它占着CPU，线程B是无法执行的。但在多核情况下，很可能CPU的另一个核正在执行线程B，所以线程A可以试试“手气”——先执行下旋转锁，尝试一段时间看看线程B是否能离开关键区域。如果是，线程A就不用切换到等待状态了。

回复MoreWindows：按照你的解释，旋转锁相对于对关键区域的一个轮询了？那对多核来说，线程A调用旋转锁也是要耗费CPU时间的不如直接进入等待状态。

回复JianCaesar：CPU在切换线程是比较费资源的，所以先尝试调用旋转锁，不行再让线程进入等待状态。这就好比CPU访问内存时先去访问cache，如未命中再去访问内存。

[http://avatar.csdn.net/9/9/C/3_galaxy_li.jpg](http://blog.csdn.net/Galaxy_Li)

博主使用了Sleep(0)，这个有什么特殊含义吗？

回复Galaxy\_Li：MSDN上面的解释是“If you specify 0 milliseconds, the thread will relinquish the remainder of its time slice but remain ready. ” 意思应该是说，该线程放弃当前分得时间片的剩余执行时间，从“进行态”直接进入“就绪态”，一旦再次获得时间片，可立即执行。

回复Galaxy\_Li：等待随机的一段时间

[http://avatar.csdn.net/3/A/8/3_eagleatustb.jpg](http://blog.csdn.net/eagleatustb)

不错，从API和代码角度解析了此同步方法，我觉得CRITICAL\_SECTION不能跨进程使用以及原因，还有多次进入关键段的情况也应该说明一下。

回复eagleatustb：CRITICAL\_SECTION不能跨进程使用，我觉得这个因为二点吧：  
1。没有这方面的接口，如事件的OpenEvent()。  
2。本身不是内核对象。  
还望高手指点。

回复eagleatustb：多次进入关键段会导致RecursionCount的增加。文章中提到了“第三个参数：LONG RecursionCount;   
表示该关键段的拥有线程对此资源获得关键段次数，初为0。”

[](http://blog.csdn.net/lixiaojun9688)

赞一个，很不错的讲解，我对这个为啥有“线程所有权”而就不能用于线程同步还是有点不太理解，希望楼主能给点详细的解释

回复lixiaojun9688：我个人理解是：同步是指线程执行的次序问题，如生产者和消费者（缓冲区个数为1），生产者要等消费者取走产品即缓冲区为空，消费者要等生产者投放产品即缓冲区非空。这对二个线程都是一种制约。而关键段的“线程拥有权”特性决定了拥有关键段的线程可以不受限制的运行，从而违反了二个线程都要受制约的要求。

回复MoreWindows：我觉得，有些同步问题用关键段来解决是有问题的，而有些同步问题是可以用关键段来解决的。

关键段不可以用来解决生产者消费者这种同步问题吗？回复MoreWindows：

回复Galaxy\_Li：关键段不能跨进程来使用，所以不能用于生产者消费者问题。

回复MoreWindows：那关键段能用于生产者线程的互斥吗？或者是消费者的互斥？

[](http://blog.csdn.net/fpgzs2)

CRITICAL\_SECTION通常说法是“临界区”吧，“关键段”这翻译感觉有点拐扭！

回复fpgzs2：俺也是觉得，“临界区”比“关键段”用得更普遍……

回复fpgzs2：《Windows核心编程》就译成关键段的。

[](http://blog.csdn.net/lanzhengpeng2)

关键段是有“线程所有权”概念的  
这个很重要.

回复lanzhengpeng2：正是因为这个“线程所有权”，所以关键段不能用于线程同步。

回复MoreWindows："拥有线程所有权"是关键段可以用于线程互斥的前提条件，但这并不能说明其不能用于线程同步。虽然只能在“拥有者”线程中Enter，却可以在任何线程中对该关键段进行Leave操作，这也就为线程同步提供了条件。  
  
我们只需要在main方法中的语句  
EnterCriticalSection(&g\_csThreadParameter);  
后面添加：  
while (g\_csThreadParameter.RecursionCount>1)  
{  
Sleep(0);  
}  
通过限制对g\_csThreadParameter的拥有次数来实现对主线程的约束，即可实现同步。  
  
另外，楼主的“房卡”比喻很形象，我觉得可以说的更完善一些，拥有房卡的人（即主人）可以多次进出房间，主人也可以带客人进房间，房门可以由主人或客人来关闭。

回复lichaoyin：确实是