# [秒杀多线程第七篇 经典线程同步 互斥量Mutex](http://blog.csdn.net/morewindows/article/details/7470936)

阅读本篇之前推荐阅读以下姊妹篇：

《[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)》

《[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)》

《[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)》

前面介绍了[关键段CS](http://blog.csdn.net/morewindows/article/details/7442639)、[事件Event](http://blog.csdn.net/morewindows/article/details/7445233)在[经典线程同步问题](http://blog.csdn.net/morewindows/article/details/7442333)中的使用。本篇介绍用互斥量Mutex来解决这个问题。

互斥量也是一个内核对象，它用来确保一个线程独占一个资源的访问。互斥量与关键段的行为非常相似，并且互斥量可以用于不同进程中的线程互斥访问资源。使用互斥量Mutex主要将用到四个函数。下面是这些函数的原型和使用说明。

第一个 CreateMutex

函数功能：创建互斥量（注意与事件Event的创建函数对比）

函数原型：

HANDLECreateMutex(

  LPSECURITY\_ATTRIBUTESlpMutexAttributes,

  BOOLbInitialOwner,

  LPCTSTRlpName

);

函数说明：

第一个参数表示安全控制，一般直接传入NULL。

第二个参数用来确定互斥量的初始拥有者。如果传入TRUE表示互斥量对象内部会记录创建它的线程的线程ID号并将递归计数设置为1，由于该线程ID非零，所以互斥量处于未触发状态。如果传入FALSE，那么互斥量对象内部的线程ID号将设置为NULL，递归计数设置为0，这意味互斥量不为任何线程占用，处于触发状态。

第三个参数用来设置互斥量的名称，在多个进程中的线程就是通过名称来确保它们访问的是同一个互斥量。

函数访问值：

成功返回一个表示互斥量的句柄，失败返回NULL。

第二个打开互斥量

函数原型：

HANDLEOpenMutex(

 DWORDdwDesiredAccess,

 BOOLbInheritHandle,

 LPCTSTRlpName     //名称

);

函数说明：

第一个参数表示访问权限，对互斥量一般传入MUTEX\_ALL\_ACCESS。详细解释可以查看MSDN文档。

第二个参数表示互斥量句柄继承性，一般传入TRUE即可。

第三个参数表示名称。某一个进程中的线程创建互斥量后，其它进程中的线程就可以通过这个函数来找到这个互斥量。

函数访问值：

成功返回一个表示互斥量的句柄，失败返回NULL。

第三个触发互斥量

函数原型：

BOOLReleaseMutex (HANDLEhMutex)

函数说明：

访问互斥资源前应该要调用等待函数，结束访问时就要调用ReleaseMutex()来表示自己已经结束访问，其它线程可以开始访问了。

最后一个清理互斥量

由于互斥量是内核对象，因此使用CloseHandle()就可以（这一点所有内核对象都一样）。

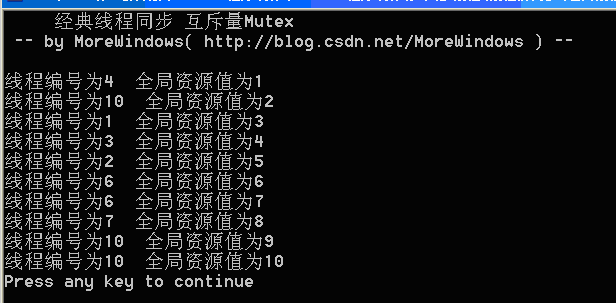
接下来我们就在经典多线程问题用互斥量来保证主线程与子线程之间的同步，由于互斥量的使用函数类似于事件Event，所以可以仿照上一篇的实现来写出代码：

**[cpp]** [view plaincopy](http://blog.csdn.net/morewindows/article/details/7470936)

1. //经典线程同步问题 互斥量Mutex
2. #include <stdio.h>
3. #include <process.h>
4. #include <windows.h>
6. **long** g\_nNum;
7. unsigned **int** \_\_stdcall Fun(**void** \*pPM);
8. **const** **int** THREAD\_NUM = 10;
9. //互斥量与关键段
10. **HANDLE**  g\_hThreadParameter;
11. CRITICAL\_SECTION g\_csThreadCode;
13. **int** main()
14. {
15. printf("     经典线程同步 互斥量Mutex\n");
16. printf(" -- by MoreWindows( http://blog.csdn.net/MoreWindows ) --\n\n");
18. //初始化互斥量与关键段 第二个参数为TRUE表示互斥量为创建线程所有
19. g\_hThreadParameter = CreateMutex(NULL, FALSE, NULL);
20. InitializeCriticalSection(&g\_csThreadCode);
22. **HANDLE**  handle[THREAD\_NUM];
23. g\_nNum = 0;
24. **int** i = 0;
25. **while** (i < THREAD\_NUM)
26. {
27. handle[i] = (**HANDLE**)\_beginthreadex(NULL, 0, Fun, &i, 0, NULL);
28. WaitForSingleObject(g\_hThreadParameter, INFINITE); //等待互斥量被触发
29. i++;
30. }
31. WaitForMultipleObjects(THREAD\_NUM, handle, TRUE, INFINITE);
33. //销毁互斥量和关键段
34. CloseHandle(g\_hThreadParameter);
35. DeleteCriticalSection(&g\_csThreadCode);
36. **for** (i = 0; i < THREAD\_NUM; i++)
37. CloseHandle(handle[i]);
38. **return** 0;
39. }
40. unsigned **int** \_\_stdcall Fun(**void** \*pPM)
41. {
42. **int** nThreadNum = \*(**int** \*)pPM;
43. ReleaseMutex(g\_hThreadParameter);//触发互斥量
45. Sleep(50);//some work should to do
47. EnterCriticalSection(&g\_csThreadCode);
48. g\_nNum++;
49. Sleep(0);//some work should to do
50. printf("线程编号为%d  全局资源值为%d\n", nThreadNum, g\_nNum);
51. LeaveCriticalSection(&g\_csThreadCode);
52. **return** 0;
53. }

说明：访问互斥资源前应该要调用等待函数，结束访问时要调用ReleaseMutex()来表示自己已经结束访问，其它线程可以开始访问了。（这个例子是个反例，用于反证不在没占有互打斥资源的线程中使用ReleaseMutex），本例中在拿到互斥量的线程是主线程，做ReleaseMutex的却是子线程，这么做是错误的，返回的将是FALSE。

运行结果如下图：



可以看出，与关键段类似，互斥量也是不能解决线程间的同步问题。

       联想到关键段会记录线程ID即有“线程拥有权”的，而互斥量也记录线程ID，莫非它也有“线程拥有权”这一说法。

       答案确实如此，互斥量也是有“线程拥有权”概念的。“线程拥有权”在[关键段](http://blog.csdn.net/morewindows/article/details/7442639)中有详细的说明，这里就不再赘述了。另外由于互斥量常用于多进程之间的线程互斥，所以它比关键段还多一个很有用的特性——“**遗弃**”情况的处理。比如有一个占用互斥量的线程在调用ReleaseMutex()触发互斥量前就意外终止了（相当于该互斥量被“遗弃”了），那么所有等待这个互斥量的线程是否会由于该互斥量无法被触发而陷入一个无穷的等待过程中了？这显然不合理。因为占用某个互斥量的线程既然终止了那足以证明它不再使用被该互斥量保护的资源，所以这些资源完全并且应当被其它线程来使用。因此在这种“**遗弃**”情况下，系统自动把该互斥量内部的线程ID设置为0，并将它的递归计数器复置为0，表示这个互斥量被触发了。然后系统将“公平地”选定一个等待线程来完成调度（被选中的线程的WaitForSingleObject()会返回WAIT\_ABANDONED\_0）。

下面写二个程序来验证下：

第一个程序创建互斥量并等待用户输入后就触发互斥量。第二个程序先打开互斥量，成功后就等待并根据等待结果作相应的输出。详见代码：

第一个程序：

**[cpp]** [view plaincopy](http://blog.csdn.net/morewindows/article/details/7470936)

1. #include <stdio.h>
2. #include <conio.h>
3. #include <windows.h>
4. **const** **char** MUTEX\_NAME[] = "Mutex\_MoreWindows";
5. **int** main()
6. {
7. **HANDLE** hMutex = CreateMutex(NULL, TRUE, MUTEX\_NAME); //创建互斥量
8. printf("互斥量已经创建，现在按任意键触发互斥量\n");
9. getch();
10. //exit(0);
11. ReleaseMutex(hMutex);
12. printf("互斥量已经触发\n");
13. CloseHandle(hMutex);
14. **return** 0;
15. }

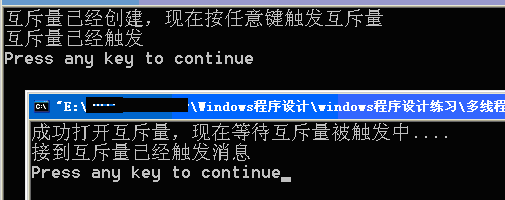
第二个程序：

**[cpp]** [view plaincopy](http://blog.csdn.net/morewindows/article/details/7470936)

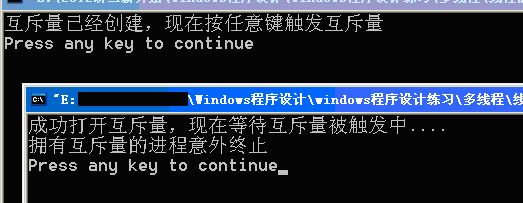
1. #include <stdio.h>
2. #include <windows.h>
3. **const** **char** MUTEX\_NAME[] = "Mutex\_MoreWindows";
4. **int** main()
5. {
6. **HANDLE** hMutex = OpenMutex(MUTEX\_ALL\_ACCESS, TRUE, MUTEX\_NAME); //打开互斥量
7. **if** (hMutex == NULL)
8. {
9. printf("打开互斥量失败\n");
10. **return** 0;
11. }
12. printf("等待中....\n");
13. **DWORD** dwResult = WaitForSingleObject(hMutex, 20 \* 1000); //等待互斥量被触发
14. **switch** (dwResult)
15. {
16. **case** WAIT\_ABANDONED:
17. printf("拥有互斥量的进程意外终止\n");
18. **break**;
20. **case** WAIT\_OBJECT\_0:
21. printf("已经收到信号\n");
22. **break**;
24. **case** WAIT\_TIMEOUT:
25. printf("信号未在规定的时间内送到\n");
26. **break**;
27. }
28. CloseHandle(hMutex);
29. **return** 0;
30. }

运用这二个程序时要先启动程序一再启动程序二。下面展示部分输出结果：

结果一．二个进程顺利执行完毕：



结果二．将程序一中//exit(0);前面的注释符号去掉，这样程序一在触发互斥量之前就会因为执行exit(0);语句而且退出，程序二会收到WAIT\_ABANDONED消息并输出“拥有互斥量的进程意外终止”：



有这个对“**遗弃**”问题的处理，在多进程中的线程同步也可以放心的使用互斥量。

最后总结下互斥量Mutex：

**1．互斥量是内核对象，它与关键段都有“线程所有权”所以不能用于线程的同步。**

**2．互斥量能够用于多个进程之间线程互斥问题，并且能完美的解决某进程意外终止所造成的“遗弃”问题。**

只用一个mutex是无法实现同步的吧，博主看下我的实现，是不是这个意思？

**[cpp]** [view plaincopy](http://blog.csdn.net/morewindows/article/details/7470936)

1. #include <stdio.h>
2. #include <process.h>
3. #include <windows.h>
4. #include <iostream>
5. #include <sstream>
6. #include <string>
8. **using** **namespace** std;
10. **long** g\_nNum;
11. unsigned **int** \_\_stdcall Fun(**void** \*pPM);
12. **const** **int** THREAD\_NUM = 10;
13. **HANDLE** g\_hThdMutex;
14. CRITICAL\_SECTION g\_csThreadCode;
16. **const** string szcMainMutexName("MainMutex");
17. **const** string szcThreadMutexName("ThreadMutex");
19. **int** main()
20. {
21. **HANDLE** hMainMutex = CreateMutex(NULL, TRUE, szcMainMutexName.c\_str());
22. InitializeCriticalSection(&g\_csThreadCode);
24. **HANDLE**  handle[THREAD\_NUM];
25. g\_nNum = 0;
26. **int** i = 0;
28. **while** (i < THREAD\_NUM)
29. {
30. handle[i] = (**HANDLE**)\_beginthreadex(NULL, 0, Fun, &i, 0, NULL);
32. **int** count = 0;
33. stringstream ss;
34. ss << i;
35. **while** ( NULL == (g\_hThdMutex = OpenMutex(MUTEX\_ALL\_ACCESS, TRUE, (szcThreadMutexName + ss.str()).c\_str())))
36. {
37. count++;
38. **if** (count > 1000)
39. {
40. printf("打开互斥量失败\n");
41. **return** 0;
42. }
43. Sleep(50);
44. }
46. //续...
    1. //接
47. ReleaseMutex(hMainMutex);
49. WaitForSingleObject(g\_hThdMutex, INFINITE);
50. CloseHandle(g\_hThdMutex);
51. i++;
52. }
53. WaitForMultipleObjects(THREAD\_NUM, handle, TRUE, INFINITE);
55. CloseHandle(hMainMutex);
56. DeleteCriticalSection(&g\_csThreadCode);
57. **for** (i = 0; i < THREAD\_NUM; i++)
58. CloseHandle(handle[i]);
59. **return** 0;
60. }
61. unsigned **int** \_\_stdcall Fun(**void** \*pPM)
62. {
63. **int** nThreadNum = \*(**int** \*)pPM;
64. stringstream ss;
65. ss << nThreadNum;
66. g\_hThdMutex = CreateMutex(NULL, TRUE, (szcThreadMutexName + ss.str()).c\_str());
68. **HANDLE** hMainMutex = OpenMutex(MUTEX\_ALL\_ACCESS, TRUE, szcMainMutexName.c\_str());
69. **if** (hMainMutex == NULL)
70. {
71. printf("打开互斥量失败\n");
72. **return** 0;
73. }

76. WaitForSingleObject(hMainMutex, INFINITE);
78. //Sleep(50);//some work should to do
80. EnterCriticalSection(&g\_csThreadCode);

83. g\_nNum++;
84. Sleep(0);//some work should to do
85. printf("线程编号为%d  全局资源值为%d\n", nThreadNum, g\_nNum);

88. LeaveCriticalSection(&g\_csThreadCode);
90. ReleaseMutex(g\_hThdMutex);
91. **return** 0;
92. }

47楼 [ljj13223452342342](http://blog.csdn.net/ll0815053) 2014-05-15 16:34发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7470936#reply)

[](http://blog.csdn.net/ll0815053)

博主你的意思是mutex拥有线程所有权因此必须由得到Mutex所有权的主线程调用ReleaseMutex才行，而Event没有所谓线程所有权因此只需在线程函数里SetEvent（），所以Mutex不能实现同步，是这样吧！ 感觉受益匪浅啊！！thx

46楼 [zhouda829829](http://blog.csdn.net/zhouda829829) 2014-04-29 14:55发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7470936#reply)

[](http://blog.csdn.net/zhouda829829)

**[cpp]** [view plaincopy](http://blog.csdn.net/morewindows/article/details/7470936)

1. #include <pthread.h>
2. #include <stdio.h>
3. #include <windows.h>
5. **int** g\_Flag = 0;
6. **void** \* thread1(**void** \* pthreadid);
7. **static** pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;
9. **void** main(**void**)
10. {
11. **int** a,i;
12. pthread\_t tid1;
13. **for**(i=0;i<10;i++)
14. {
15. a = pthread\_create(&tid1,NULL,thread1,(**void**\*)i);
16. }
18. getchar();
19. }
21. **void** \* thread1(**void** \* pthreadid)
22. {
23. **int** nThreadNum = (**int**) pthreadid;
24. Sleep(50);//some work should to do
25. pthread\_mutex\_lock(&mutex);
26. g\_Flag++;  //处理全局资源
28. Sleep(0);//some work should to do
29. printf("线程编号为%d  全局资源值为%d\n", nThreadNum, g\_Flag);
30. pthread\_mutex\_unlock(&mutex);
31. **return** 0;
32. }

我就用了互斥实现了你说的要求，是不是我的理解有什么偏差啊，求指正。

回复zhouda829829：我也有同样疑问呀博主!!  
zh兄的代码确实实现了同步和互斥呀!!  
可以麻烦博主说明一下吗??

43楼 [第XXX个小号](http://blog.csdn.net/ccy0815ccy) 2014-03-07 18:10发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7470936#reply)

[](http://blog.csdn.net/ccy0815ccy)

已经反复看了楼主文章好几遍，每次看完文章后，最大的感悟 是，是时候认真学习英文了

40楼 [yy610879290](http://blog.csdn.net/yy610879290) 2013-10-26 11:37发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7470936#reply)

[](http://blog.csdn.net/yy610879290)

楼主互斥量的例子有问题：  
1、首先，拿到互斥量的线程是主线程，在子线程里面去ReleaseMutex肯定失败，返回的是FALSE。  
2、这里创建的互斥量没有任何作用，while循环里面的WaitForSingleObject发现访问互斥量的线程是同一个线程，所以会把当前线程置为可调度的。

回复yy610879290：对的，文章就是为了证明互斥量不能这样用。

30楼 [亚细亚](http://blog.csdn.net/yaxiya) 2013-06-13 10:40发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7470936#reply)

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

**[cpp]** [view plaincopy](http://blog.csdn.net/morewindows/article/details/7470936)

1. **HANDLE** WINAPI CreateMutex(
2. \_In\_opt\_  LPSECURITY\_ATTRIBUTES lpMutexAttributes,
3. \_In\_      **BOOL** bInitialOwner,
4. \_In\_opt\_  **LPCTSTR** lpName
5. );
6. lpMutexAttributes：互斥量安全访问属性，一般置为NULL；
8. bInitialOwner：控制互斥量的初始状态，一般设置为**false**，是互斥量的线程ID和递归计数都设为0，表示互斥量不被任何进程占用，处于触发状态，其他进程可以进行调用等待函数，获得该互斥量对象，获得对同步资源的占用。如果初始值设为**true**，互斥量的线程ID设置为当前线程，递归计数设为1，表示当前线程占用互斥量对象，拥有对同步资源的独占，互斥量处于未触发状态。
10. lpName：用于创建有名的内核对象，即用来创建跨进程边界的内核对象。
12. CreateMutex用于创建名为lpName的互斥量内核对象，并返回指向该内核对象的句柄。如果该内核对象已经存在，那么CreateMutex会返回该内核对象的句柄，并通过系统返回错误ERROR\_ALREADY\_EXISTS，通过GetLastError（）获得。

回复yaxiya：刚开始对触发和未触发理解比较模糊,现在明白了^\_^

回复yaxiya：未触发其实就是已经有线程独占了,所以其它线程就只能等待;当该线程释放互斥量后,也就是互斥量中的线程ID为0,计数器为0,这时,互斥量处于触发状态,其它线程可以进入,从而拥有该互斥量对象了;

28楼 [jianxianbai](http://blog.csdn.net/jianxianbai) 2013-04-17 15:40发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7470936#reply)

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

我觉得这样理解比较好，mutex和关键段就像一张房卡（或者通行证），每扇门只能有一张房卡（每个关卡只有一张通行证）。每个线程就像一个人，只有拿到房卡（通行证）的人可以进入门中（通过关卡），wait。。。。（ENTER...）函数就相当于申请房卡(通行证)，ReleaseMutex （leave。。。）就像归还房卡（通行证），但是没有申请到房卡的人是无法归还房卡的，申请到房卡的人就一直拥有了这张房卡了，直到归还为止。  
  
而他们的区别就在于，如果把某个用户比作一个国家。mutex更像是某个关卡的通行证，是属于整个国家的，在整个国家都需要这张通行证才能过这个关卡。而如果把每个进程比作一个国家里的一座座大楼，那关键段就像是真正的房卡，房卡在不同的大楼里面可以重名。A楼的卡也不会影响B楼的卡

23楼 [小九](http://blog.csdn.net/windyitian) 2012-12-19 20:04发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7470936#reply)

[http://avatar.csdn.net/A/C/A/3_windyitian.jpg](http://blog.csdn.net/windyitian) ------ 我也想问这个问题。。！！！！

兄弟这个系列写的不错。比我强多了。正巧我最近在看windows核心编程。有点问题说一下。  
正如之前回复中有人提到的，例子一确实有问题，它无法作为mutex不能进行同步的例证，因为你那么写代码跟没用Mutex一模一样。  
1、你主线程创建的互斥量本身就是已触发状态。因此主线程中的第一次Wait时直接将mutex归属于主线程，然后主线程继续向下执行，下次在Wait时也无需等待，因为此mutex归属于主线程，直接计数加一后向下执行。而新生线程中的release无作用，它本身就没占有过互斥量，怎么能release呢，所以返回fasle向下执行。因此可能出现创建了一个子线程后，子线程尚未运行任何代码时，你主线程又创建了多个子线程。当这些子线程运行代码时，取了重复的i值。  
2、信号量的时候这么做就可以了。因为信号量不归属于任何线程，所以你那么编码保证了主线程创建一个子线程->主线程等待->子线程取i值->主线程创建另一个子线程->主线程等待->子线程取的新i值的有序循环。  
3、当然，为了实现你这种有序的循环，靠mutex确实做不到。就算正确的在同一个线程中对mutex进行wait和release，依旧只能保证一次只有一个线程处理临界资源i而不能保证次序。  
你最后总结的时候我觉得是对的。mutex的线程相关性导致无法进行线程同步。其实名字翻译过来本身就是互斥量。不能同步线程的次序也是题中应有之意。

回复windyitian：楼主的意思就是要说明，这个mutex是属于某个线程的这个结论。明白吗?.......

回复jianxianbai：我有同样的疑问  
g\_hThreadParameter = CreateMutex(NULL, FALSE, NULL); 楼主此处初始化有问题吧，第二个参数为FALSE表示互斥量对象的线程ID和递归计数都将被设为0，这意味着互斥量不为任何线程占用，因此处于触发状态，既然是触发状态那接下来在线程中调用的ReleaseMutex释放触发状态不就是多余了吗，因此第二个参数应该是TRUE，处于未触发状态才对吧

[](http://blog.csdn.net/lfh05)

《windows 核心编程》的代码中文版？

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

Mutex的进程间同步，仅限于同一个windows登录用户启动的进程，如果是不同用户启动的（比如控制台用户和远程桌面用户），进程间用Mutex是无效的

回复SpbDev：MSDN上有解释，如果要创建全局的，名字前面加globe。所有用户都有效。 The name can have a "Global\" or "Local\" prefix to explicitly create the object in the global or session name space.

回复chmlqw：正解。

[http://avatar.csdn.net/D/1/D/3_qinfengxiaoyue.jpg](http://blog.csdn.net/qinfengxiaoyue)

回复MoreWindows：例子一对mutex的使用欠妥当。同意5楼同学的说法。以下是MSDN对mutex使用的一些说明。  
The state of a mutex object is signaled when it is not owned by any thread. The creating thread can use the bInitialOwner flag to request immediate ownership of the mutex. Otherwise, a thread must use one of the wait functions to request ownership. When the mutex's state is signaled, one waiting thread is granted ownership, the mutex's state changes to nonsignaled, and the wait function returns. Only one thread can own a mutex at any given time. The owning thread uses the ReleaseMutex function to release its ownership.

回复qinfengxiaoyue：文章前面其实也说明了：  
第三个触发互斥量  
函数原型：  
BOOL ReleaseMutex (HANDLEhMutex)  
函数说明：  
访问互斥资源前应该要调用等待函数，结束访问时要调用ReleaseMutex()来表示自己已经结束访问，其它线程可以开始访问了。  
例一本来也就是为了直观形象的证明其不能用于同步，所以就违反这一规定了来试了一下。

5楼 [xianyunhe1234](http://blog.csdn.net/xianyunhe1234) 2012-04-19 09:55发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7470936#reply)

[](http://blog.csdn.net/xianyunhe1234)!!!!!!!不理解-#############################

楼主的例子1中并没有正确的使用Mutex，所有的子线程并没有使用 WaitForSingleObject来获得对Mutex的拥有权。所有的ReleaseMutex的也就完全没有效果，ReleaseMutex返回的全是false。

引用“xianyunhe1234”的评论：楼主的例子1中并没有正确的使用Mutex，所有的子线程并没有使用 WaitForSingleObje...  
我同意，在MSDN中的xxxxMutex相关的说明中很清楚了，你这个线程都没有wait到它，所有权都不是该线程，Release是没用的。  
代码拷贝了下来，略作修改，可以使用的。另外那个cs可以去掉了。

回复xianyunhe1234：那例子一中应该如何来使用Mutex了？

回复MoreWindows：每一个线程要想通过mutex进入互斥区，必须使用WaitForSingleObject来加锁，然后 使用ReleaseMutex来解锁，一般情况下必须成对出现，否则互斥就无效了。例子1中的WaitForSingleObject只对主线程有效，对子线程无效。

回复xianyunhe1234："例子1中的WaitForSingleObject只对主线程有效，对子线程无效。"那是因为互斥量的线程所有权的原因。不是WaitForSingleObject要与WaitForSingleObject必须成对的原因。

**[cpp]** [view plaincopy](http://blog.csdn.net/morewindows/article/details/7470936)

1. **int** main(**void**)
2. {
3. //前面的省略
4. **while** (i < THREAD\_NUM)
5. {
6. handle[i] = (**HANDLE**)\_beginthreadex(NULL, 0, Fun, &i, 0, NULL);
7. WaitForSingleObject(g\_hThreadParameter, INFINITE); //等待互斥量被触发
8. ReleaseMutex(g\_hThreadParameter);
9. i++;
10. }
11. // 后面也省略
12. }
13. unsigned **int** \_\_stdcall Fun(**void** \*pPM)
14. {
15. **int** nThreadNum = \*(**int** \*)pPM;
17. WaitForSingleObject(g\_hThreadParameter, INFINITE);
18. ReleaseMutex(g\_hThreadParameter);//触发互斥量
20. // 后面省略
21. **return** 0;
22. }

他说的应该是这个意思，但是这样的话，主线程和子线程获取mutex的机会是一样的，所以有可能主线程先获取了mutex,那么就无法达到预期的要求了。所以mutex不方便用于同步。不像EVENT，可以确保子线程先得到信号。

回复chmlqw：嗯，是这个意思。

回复xianyunhe1234：17. WaitForSingleObject(g\_hThreadParameter, INFINITE);   
18. ReleaseMutex(g\_hThreadParameter);//触发互斥量   
先加锁然后什么也不做又马上解锁？

回复MoreWindows：WaitForSingleObject的作用是想等待子线程获取了变量值，然后主线程在继续下一个线程的创建。所以得到以后，就可以马上释放了，只有这里释放了，子线程才有机会获取。但是我前面也讲了，因为Mutex的特殊性（就是主线程和子线程wait之后，获取信号的机会是一样的），所以无法达到同步的目的，只能用来互斥。

回复MoreWindows：仅使用mutex无法实现同步，我是同意的。