# [C++中Event使用](http://blog.csdn.net/xiaobai1593/article/details/6990275)

之前线程同步用互斥锁Mutex或用户模式的CriticalSection等来进行同步，而且使用效果一直很好，直到最近遇到新问题，朋友推荐用事件去处理。但是对事件了解比较少，所以摘了篇文章看，另外自己做了下试验（所以代码不是转的哦）。

**概念：**

事件是用来同步地位不相等的线程的，事件可以用来使一个线程完成一件事情，然后另外的线程完成剩下的事情。

事件的使用很灵活，自动事件的激发态是由人工来控制的（超级重要，尤其是在需要精确掌控是否处于有信号状态的时候），而Mutex在释放（releaseMetux）后就一直处于激发态，直到线程WaitForSingleObject。事件可以用来控制经典的读写模型和生产者和消费者模型。相应的方式为，生成者等待消费者的消费，再消费者消费完后通知生产者进行生产。

Mutex是排他的占有资源，一般用于地位相等的线程进行同步。每个线程都可以排他的访问一个资源或代码段，不存在哪个线程对资源访问存在优先次序。一个线程只能在Mutex处于激发态的时候访问被保护的资源或代码段，线程可以通过WaitForSingelObject来等待Mutex，在访问资源完成之后，ReleaseMutex释放Mutex，此时Mutex处于激发态。

**注意：对句柄使用WaitForSIngleObject()之后的副作用（句柄状态的自动变化）**

Mutex具有成功等待的副作用，在用WaitForSingleObject()等待到Mutex后，Mutex自动变为无信号状态，直到调用ReleaseMutex()使Mutex变为有信号状态为止；

自动事件也具有成功等待的副作用；

手动事件没有，必须调用ResetEvent()使手动事件变为未激发态；

进程和线程也没有成功等待的副作用。当线程或者进程函数返回时，线程内核对象变为激发态，但WaitForSingleObject并没有使线程或者进程的内核对象变为未激发态。

总之，事件一般用于控制线程的先后顺序，而Mutex一般用于排他的访问资源。

**Event的使用：**

首先创建事件:

HANDLE CreateEvent(

　　LPSECURITY\_ATTRIBUTES lpEventAttributes, // 安全属性

　　BOOL bManualReset, // 复位方式，1为手动复位，0为自动复位

　　BOOL bInitialState, // 初始状态，0为不可用状态即无信号，1为可用状态即有信号

　　LPCTSTR lpName // 对象名称

　　);

SetEvent()设置事件为有信号；ResetEvent()设置事件为无信号。

WaitForSingleObject()来等待Event变为有信号：

DWORD WaitForSignalObject(HANDLE hObject, DWORD dwMilliseconds);

返回值有以下几种：

WAIT\_OBJECT\_0 ：表示等待对象已经变为有信号状态，如果设置为自动复位，还会把此信号再次变为无信号状态。

WAIT\_TIMEOUT   ：表示等待超时

WAIT\_FAILED      ：表示等待对象句柄是一个无效句柄。

代码示例：

功能：用主线程来等待子线程的执行结束

**[cpp]** [view plain](http://blog.csdn.net/xiaobai1593/article/details/6990275) [copy](http://blog.csdn.net/xiaobai1593/article/details/6990275)

[print?](http://blog.csdn.net/xiaobai1593/article/details/6990275)

1. **HANDLE** hEvent;
2. **int** num=1;
4. **DWORD** WINAPI \_threadProc(**LPVOID** lpParam)
5. {
6. for(**int** i=0; i<10000; i++)
7. {
8. for(**int** j=0; j<10000; j++)
9. ;
10. }
12. num=10;
13. SetEvent(hEvent);
14. return 0;
15. }
17. void main()
18. {
19. //手动复位，初始时无信号
20. hEvent=CreateEvent(NULL, TRUE, FALSE, NULL);
21. if(hEvent==NULL)
22. {
23. printf("hEvent is null\n");
24. return;
25. }
27. **HANDLE** hThread=CreateThread(NULL, 0, \_threadProc, NULL, 0, NULL);
29. **DWORD** dwRet=WaitForSingleObject(hEvent, INFINITE);
30. if(dwRet==WAIT\_ABANDONED)
31. printf("WAIT\_ABANDONED\n");
32. else if(dwRet==WAIT\_TIMEOUT)
33. printf("WAIT\_TIMEOUT\n");
34. else if(dwRet==WAIT\_OBJECT\_0)
35. printf("有信号状态\n");
36. else if(dwRet==WAIT\_FAILED)
37. printf("WAIT\_FAILED\n");
39. printf("sub thread run out, num is %d\n", num);
40. }

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

HANDLE hEvent;

int num=1;

DWORD WINAPI \_threadProc(LPVOID lpParam)

{

for(int i=0; i<10000; i++)

{

for(int j=0; j<10000; j++)

;

}

num=10;

SetEvent(hEvent);

return 0;

}

void main()

{

//手动复位，初始时无信号

hEvent=CreateEvent(NULL, TRUE, FALSE, NULL);

if(hEvent==NULL)

{

printf("hEvent is null\n");

return;

}

HANDLE hThread=CreateThread(NULL, 0, \_threadProc, NULL, 0, NULL);

DWORD dwRet=WaitForSingleObject(hEvent, INFINITE);

if(dwRet==WAIT\_ABANDONED)

printf("WAIT\_ABANDONED\n");

else if(dwRet==WAIT\_TIMEOUT)

printf("WAIT\_TIMEOUT\n");

else if(dwRet==WAIT\_OBJECT\_0)

printf("有信号状态\n");

else if(dwRet==WAIT\_FAILED)

printf("WAIT\_FAILED\n");

printf("sub thread run out, num is %d\n", num);

}

# CEvent类详解

CEvent类祥解

CEvent 类提供了对事件的支持。事件是一个允许一个线程在某种情况发生时，唤醒另外一个线程的同步对象。事件告诉线程何时去执行某一给定的任务，从而使多个线程流平滑。例如在某些网络应用程序中，一个线程（记为A）负责监听通信端口，另一个线程（记为B）负责更新用户数据。通过使用CEvent类，线程A可以通知线程B何时更新用户数据，这样线程B可以尽快地更新用户数据。每一个CEvent对象可以有两种状态：有信号状态（signaled）和无信号状态（nonsignaled）。线程监视位于其中的CEvent类对象的状态，并在相应的时候采取相应的操作。

     在MFC中，CEvent类对象有两种类型，分别是所谓的人工事件和自动事件。对于自动事件，当其获得信号后，就会释放下一个可用的线程。一个自动 CEvent对象在被至少一个线程释放后会自动返回到无信号状态；而人工事件对象获得信号后，释放所有可利用线程，直到调用成员函数ReSetEvent ()将其设置为无信号状态时为止。注意，在创建CEvent类的对象时，默认创建的是自动事件。

CEvent的各成员函数的原型与参数说明如下。

CEvent(

BOOL bInitiallyOwn = FALSE,

BOOL bManualReset = FALSE,

LPCTSTR lpszName = NULL,

LPSECURITY\_ATTRIBUTES lpsaAttribute = NULL );

bInitiallyOwn：若bInitiallyOwn为TRUE，则使CMultilock类对象和CSingleLock类对象的线程可用；否则，要访问资源的线程必须等待。该参数的默认值为FALSE。

bManualReset：指定要创建的CEvent对象是属于手工事件还是自动事件。为TRUE，则为手工事件，否则为自动事件。该参数默认值为FALSE。

lpszName：指定要创建的事件对象的名，如果该事件对象将跨进程使用，则此参数不能为NULL。如果该参数和一个已经存在的CEvent对象相同，则该构造函数返回一个对这个已存在对象的引用；如果参数和一个已存在的非CEvent类的同步对象（如CMutex）相同，则对象创建失败；

lpsaAttribute：指向SECURITY\_ATTRIBUTES结构的指针，该参数决定要创建的事件对象的安全属性，一般置为NULL。

在事件对象建成后，可以调用其成员函数来改变其状态。

BOOL CEvent::SetEvent ();

将CEvent类对象的状态设置为有信号状态，并且释放所有等待的线程；如果该事件是人工事件，则CEvent类对象保持为有信号状态，直到调用成员函数ResetEvent()将其重新设为无信号状态时为止，这样该事件就可以释放多个线程；如果CEvent类对象为自动事件，则在SetEvent()将事件设置为有信号状态后，CEvent类对象由系统自动重置为无信号状态，除非一个线程被释放。

如果该函数执行成功，则返回非零值，否则返回零。

BOOL CEvent::ResetEvent();

该函数将事件的状态设置为无信号状态，并保持该状态直至SetEvent()被调用时为止。由于自动事件是由系统自动重置，故自动事件不需要调用该函数。

如果该函数执行成功，返回非零值，否则返回非零。

BOOL CEvent::PulseEvent()

发送一个事件脉冲，该函数完成一系列操作后才返回。对于自动事件，PulseEvent()将事件设置为有信号状态，等待一个线程被释放，将事件重置为无信号状态，然后PulseEvent()返回；对于人工事件，则将等待该事件的所有线程被释放，事件被自动重置为无信号状态，然后PulseEvent()返回。

一个CEvent对象在线程中被创建后，自动处于无信号状态，但在另一个线程中可以调用Win32 API WaitForSingleObject()函数来监视其状态。

该函数的原型及参数说明如下：

DWORD WaitForSingleObject(HANDLE hHandle，DWORD dwMilliseconds);

其中hHandle为指向要监视的同步对象的句柄，dwMilliseconds为监视hHandle所指向的对象所设置的超时值，单位为毫秒。当在线程的执行函数中调用该函数时，线程暂时挂起，系统监视hHandle所指向的对象的状态。

如果经过dwMilliseconds毫秒后，hHandle指向的对象变为有信号状态，则WaitForSingleObject()返回，线程被释放，且返回值为WAIT\_TIMEOUT；

如果在挂起的dwMilliseconds毫秒内，线程所等待的对象在某一时刻变为有信号，则该函数立即返回，返回值为WAIT\_OBJECT\_0。

参数dwMilliseconds有两个具有特殊意义的值：0和INFINITE。若为0，则该函数立即返回；若为INFINITE，则线程一直被挂起，直到hHandle所指向的对象变为有信号状态时为止。如果CEvent对象为自动事件，则当WaitForSingleObject(hHandle,INFINITE)返回时，自动把CEvent对象重置为无信号状态。CEvent::SetEvent()把对象设置为有信号状态，释放等待的线程。CEvent::ResetEvent()把对象设置为无信号状态，程序在WaitForSingleObject(hHandle,INFINITE)处等待。

例程9 MultiThread9

建立一个基于对话框的工程MultiThread9，在对话框IDD\_MULTITHREAD9\_DIALOG中加入一个按钮和两个编辑框控件，按钮的ID为IDC\_WRITEW，标题为“写'W’”；两个编辑框的ID分别为IDC\_W和IDC\_D，属性都选中Read-only;

在MultiThread9Dlg.h文件中声明两个线程函数：

UINT WriteW(LPVOID pParam);

UINT WriteD(LPVOID pParam);

使用ClassWizard分别给IDC\_W和IDC\_D添加CEdit类变量m\_ctrlW和m\_ctrlD；

在MultiThread9Dlg.cpp文件中添加如下内容：

为了文件中能够正确使用同步类，在文件开头添加

#include "afxmt.h"

定义事件对象和一个字符数组，为了能够在不同线程间使用，定义为全局变量。

CEvent eventWriteD;

char g\_Array[10];

添加线程函数：

UINT WriteW(LPVOID pParam)

{

 CEdit \*pEdit=(CEdit\*)pParam;

 pEdit->SetWindowText("");

 for(int i=0;i<10;i++)

 {

  g\_Array[i]=''W'';

      pEdit->SetWindowText(g\_Array);

  Sleep(1000);

 }

 eventWriteD.SetEvent();

 return 0;

}

UINT WriteD(LPVOID pParam)

{

 CEdit \*pEdit=(CEdit\*)pParam;

 pEdit->SetWindowText("");

 WaitForSingleObject(eventWriteD.m\_hObject,INFINITE);

 for(int i=0;i<10;i++)

 {

  g\_Array[i]=''D'';

      pEdit->SetWindowText(g\_Array);

  Sleep(1000);

 }

 return 0;

}

　　仔细分析这两个线程函数, 您就会正确理解CEvent 类。线程WriteD执行到 WaitForSingleObject(eventWriteD.m\_hObject,INFINITE);处等待，直到事件eventWriteD为有信号该线程才往下执行，因为eventWriteD对象是自动事件，则当WaitForSingleObject()返回时，系统自动把eventWriteD对象重置为无信号状态。

双击按钮IDC\_WRITEW，添加其响应函数：

void CMultiThread9Dlg::OnWritew()

{

 CWinThread \*pWriteW=AfxBeginThread(WriteW,

  &m\_ctrlW,

  THREAD\_PRIORITY\_NORMAL,

  0,

  CREATE\_SUSPENDED);

 pWriteW->ResumeThread();

 CWinThread \*pWriteD=AfxBeginThread(WriteD,

  &m\_ctrlD,

  THREAD\_PRIORITY\_NORMAL,

  0,

  CREATE\_SUSPENDED);

 pWriteD->ResumeThread();

}编译并运行程序，单击“写'W’”按钮，体会事件对象的作用。

一个Event被创建以后,可以用OpenEvent()API来获得它的Handle,用CloseHandle()

　　来关闭它,用SetEvent()或PulseEvent()来设置它使其有信号,用ResetEvent()

　　来使其无信号,用WaitForSingleObject()或WaitForMultipleObjects()来等待

　　其变为有信号.