[秒杀多线程第九篇 经典线程同步总结 关键段 事件 互斥量 信号量](http://blog.csdn.net/morewindows/article/details/7538247)

前面《[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，这个问题包括了主线程与子线程的同步，子线程间的互斥，是一道非常经典的多线程同步互斥问题范例，后面分别用了四篇

《[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)》

《[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)》

《[秒杀多线程第八篇经典线程同步信号量Semaphore](http://blog.csdn.net/morewindows/article/details/7481609)》

来详细介绍常用的线程同步互斥机制——关键段、事件、互斥量、信号量。下面对它们作个总结，帮助大家梳理各个知识点。

首先来看下关于线程同步互斥的概念性的知识，相信大家通过前面的文章，已经对线程同步互斥有一定的认识了，也能模糊的说出线程同步互斥的各种概念性知识，下面再列出从《计算机操作系统》一书中选取的一些关于线程同步互斥的描述。相信先有个初步而模糊的印象再看下权威的定义，应该会记忆的特别深刻。

1．线程（进程）同步的主要任务

答：在引入多线程后，由于线程执行的异步性，会给系统造成混乱，特别是在急用临界资源时，如多个线程急用同一台打印机，会使打印结果交织在一起，难于区分。当多个线程急用共享变量，表格，链表时，可能会导致数据处理出错，因此线程同步的主要任务是使并发执行的各线程之间能够有效的共享资源和相互合作，从而使程序的执行具有可再现性。

2．线程（进程）之间的制约关系？

当线程并发执行时，由于资源共享和线程协作，使用线程之间会存在以下两种制约关系。

（1）．间接相互制约。一个系统中的多个线程必然要共享某种系统资源，如共享CPU，共享I/O设备，所谓间接相互制约即源于这种资源共享，打印机就是最好的例子，线程A在使用打印机时，其它线程都要等待。

（2）．直接相互制约。这种制约主要是因为线程之间的合作，如有线程A将计算结果提供给线程B作进一步处理，那么线程B在线程A将数据送达之前都将处于阻塞状态。

间接相互制约可以称为**互斥**，直接相互制约可以称为**同步**，对于互斥可以这样理解，线程A和线程B互斥访问某个资源则它们之间就会产个顺序问题——要么线程A等待线程B操作完毕，要么线程B等待线程操作完毕，这其实就是线程的同步了。因此**同步包括互斥，互斥其实是一种特殊的同步**。

3．临界资源和临界区

在一段时间内只允许一个线程访问的资源就称为临界资源或独占资源，计算机中大多数物理设备，进程中的共享变量等待都是临界资源，它们要求被互斥的访问。每个进程中访问临界资源的代码称为临界区

看完概念性知识，下面用几个表格来帮助大家更好的记忆和运用多线程同步互斥的四个实现方法——关键段、事件、互斥量、信号量。

关键段CS与互斥量Mutex

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 创建或初始化 | 销毁 | 进入互斥区域 | 离开互斥区域 |
| 关键段CS | Initialize-  CriticalSection | Delete-  CriticalSection | Enter-  CriticalSection | Leave-  CriticalSection |
| 互斥量Mutex | CreateMutex | CloseHandle | 等待系列函数如WaitForSingleObject | ReleaseMutex |

关键段与互斥量都有“线程所有权”概念，可以将“线程所有权”理解成旅馆的房卡，在旅馆前台登记名字拥有房卡后是可以多次进出房间的，其它人则无法进入直到你交出房卡。每个线程必须先通过EnterCriticalSection或WaitForSingleObject来尝试获得“线程所有权”才能调用LeaveCriticalSection或ReleaseMutex。否则会调用失败，这就相当于伪造房卡去办理退房手续——由于登记本上没有你的名字所以会被拒绝。

互斥量能很好的处理“遗弃”情况，因此在多进程之间可以放心的使用。

对于关键段,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。

对于互斥量、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。

事件Event

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 创建 | 销毁 | 使事件触发 | 使事件未触发 |
| 事件Event | CreateEvent | CloseHandle | SetEvent | ResetEvent |

注意事件的手动置位和自动置位要分清楚，不要混淆了。

手动置位与自动置位在PulseEvent函数中的不同情况：

函数功能：在事件触发后立即将事件设置为未触发，相当于触发一个事件脉冲。

函数原型：BOOL PulseEvent(HANDLEhEvent);

函数说明：这是一个不常用的事件函数，此函数相当于SetEvent()后立即调用ResetEvent();此时情况可以分为两种：

1.对于手动置位事件，所有正处于等待状态下的线程都变成可调度状态。

2.对于自动置位事件，所有正处于等待状态下线程只有一个变成可调度状态。（因为自动置位中某一线程获得资源后会将事件置为未触发）

此后事件是末触发的。该函数不稳定，因为无法预知在调用PulseEvent ()时哪些线程正处于等待状态。

信号量Semaphore

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 创建 | 销毁 | 递减计数 | 递增计数 |
| 信号量  Semaphore | Create-  Semaphore | CloseHandle | 等待系列函数如WaitForSingleObject | Release-  Semaphore |

信号量在计数大于0时表示触发状态，调用WaitForSingleObject不会阻塞，等于0表示未触发状态，调用WaitForSingleObject会阻塞直到有其它线程递增了计数。

注意：互斥量，事件，信号量都是内核对象，可以跨进程使用（通过OpenMutex，OpenEvent，OpenSemaphore）。不过为什么只有互斥量能解决“遗弃”情况了，请看《[秒杀多线程第十五篇 关键段,事件,互斥量,信号量的“遗弃”问题](http://blog.csdn.net/morewindows/article/details/7823572)》。

呵呵^\_^，本系列一共使用了六篇文章来讲解了上面三个表格，如果读者能轻松写出这个表格并能解释下各函数的用法，那么对多线程的同步互斥问题也就有了良好的基础。

通过经典线程同步问题的学习，我们已经初步练好了解决多线程同步互斥的各种“招式”，下面再通过学习二个著名的实例《[秒杀多线程第十篇 生产者消费者问题](http://blog.csdn.net/morewindows/article/details/7577591)》和《[秒杀多线程第十一篇 读者写者问题](http://blog.csdn.net/morewindows/article/details/7596034)》来使我们在解决多线程同步时更加熟练。

9楼 [tdl1001](http://blog.csdn.net/tdl1001) 2012-06-13 15:41发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7538247#reply)

[](http://blog.csdn.net/tdl1001)

每个线程必须先通过EnterCriticalSection或WaitForSingleObject来尝试获得“线程所有权”才能调用LeaveCriticalSection或ReleaseMutex。  
  
好像楼主前面专门讲CS和Mutex的例子都不是这样啊

回复tdl1001：呵呵，那些例子就是为了证明要像你说的这样的使用，不然无法达到要求。（那些例子是反例）

7楼 [linteng2003](http://blog.csdn.net/goodhongwei) 2012-05-17 11:31发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7538247#reply)

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

“Mutex在没有被任何线程所拥有时是有信号的，这时任何线程可以使用Wait functions去获取该对象的所有权。当Mutex被某个线程拥有后就处于没信号状态，并且只有当该线程使用ReleaseMutex释放该互斥对象时，它才能被其它线程获得。在一个线程中试图释放另个线程所拥有的Mutex是不会成功的，Mutex只能被所有者线程所释放。”

回复linteng2003：这段话对理解Mutex很有帮助;