# 实验五 Windows多线程编程和线程同步

### 一、 实验目的

了解Windows中的进程和线程，了解多线程编程，理解基于互斥对象的线程同步的实现。

### 二、 实验要求

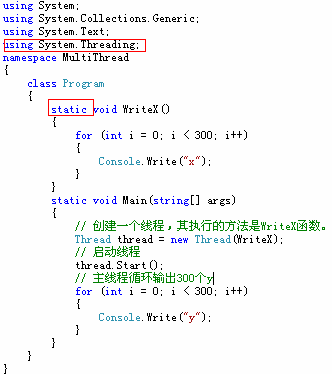
跟着实验指导书步骤边做边思考，同时完成实验报告。提交报告至学者网。

### 三、 实验内容

### （一）线程创建

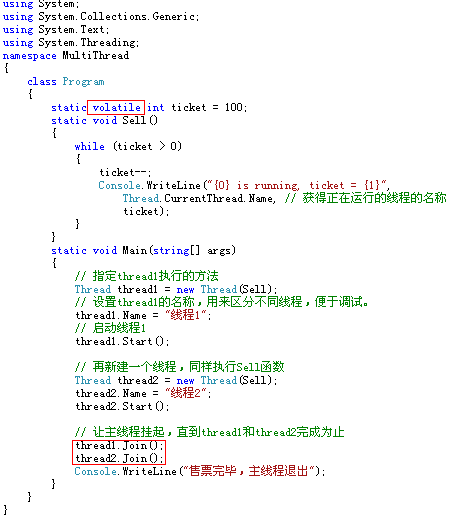
1. 新建一个新的空的工程，工程名为\*\*MultiThread，其中\*\*为你的姓名拼音首字母+学号末两位，如张三学号末两位为21， 则其工程名为ZS21MultiThread。

2. 在Program.cs中输入以下代码，边输入边理解：



回答：程序运行结果是什么？解释运行结果。

3. 输入以下代码，边输入边理解。



代码理解关键点：

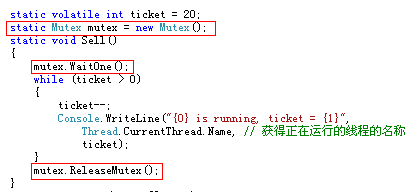
1. 代码创建了两个线程，两个线程均执行方法void Sell();
2. ticket是进程的全局变量，在进程执行过程中，thread1和thread2并发操作ticket。关键字volatile用于通知编译器，将有多个线程访问ticket数据成员，因此它不应当对此成员的状态做任何优化假设。通过将 volatile 与 ticket数据成员一起使用，可以从多个线程安全地访问此成员，而不需要使用正式的线程同步技术，但这仅仅是因为ticket 是 int类型。这意味着只需要执行单个原子操作就能修改ticket。但是，如果此数据成员是类、结构或数组，那么，从多个线程访问它可能会导致间歇的数据损坏。假设有一个更改数组中的值的线程。Windows 会定期中断线程，以便允许其他线程执行。因此，此线程会在分配某些数组元素之后和分配其他元素之前被停止。由于数组现在有了一个程序员从不想要的状态，因此，读取此数组的另一个线程可能会失败。
3. Thread类对象有一个Name属性，用于指定线程的名称，方便调试。可通过静态方法Thread.CurrentThread获得当前正在执行的线程。
4. thread1.join()方法让主线程阻塞，直到thread1终止为止，再唤醒主线程继续执行。thread2.join()功能同理。若注释掉thread1.Join()和thread2.Join()，信息“售票完毕，主线程退出”会在thread1和thread2终止后再显示吗？为什么？

### （二） 线程同步

1． 续上述程序，取消thread1.join()和thread2.join()的注释，运行程序。程序运行结果符合预期吗？为什么？运行几次程序。程序每次执行结果一样吗？为什么？

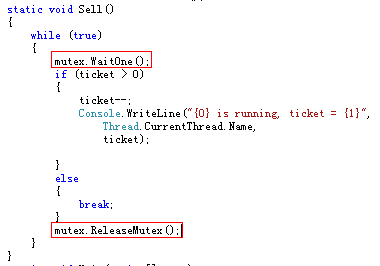
2. 在函数Fun()内，ticket—前增加一句“Thread.Sleep(500);”，其作用是让当前线程阻塞500毫秒，然后再唤醒线程继续工作。再次运行程序（或多运行几次程序），会不会出现ticket=-1的情况？为什么？提示：分析当ticket = 1时，两线程的可能调度情况。

3. 产生上述问题，是因为两线程共享全局变量ticket，而没有任何互斥措施。下面我们实现对tickets的互斥访问。在void Sell()函数中修改代码如下：



1. 采用Mutex类创建一个互斥对象mutex。WaitOne(); 相当于互斥信号量的wait操作，ReleaseMutex()相当于互斥信号量的signal操作。
2. 修改代码后，程序运行结果是什么？分析运行结果。

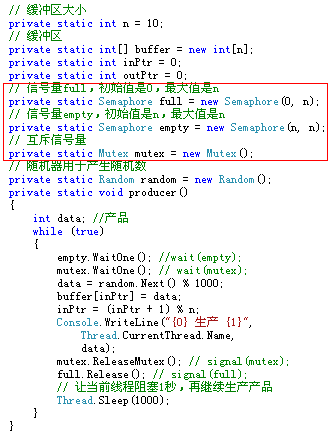
4. 更改上述代码如下：

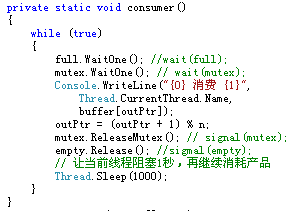


程序运行结果是什么？为什么会出错？该如何修改代码？

### （三） 用信号量实现生产者-消费者问题

输入以下代码，边输入边理解。





在main函数中输入以下代码：

Thread producerThread = new Thread(producer);

producerThread.Name = "producer";

producerThread.Start();

Thread consumerThread = new Thread(consumer);

consumerThread.Name = "consumer";

consumerThread.Start();

1. 学习使用Semaphore类和Mutex类。
2. 在main函数中创建三个生产者和三个消费者线程，查看程序运行结果，理解生产者线程与消费者线程之间如何同步。

Thread[] threads = new Thread[6];

for (int i = 0; i < 3; i++)

{

threads[i] = new Thread(producer);

threads[i].Name = string.Format("producer{0}", i);

}

for (int i = 3; i < 6; i++)

{

threads[i] = new Thread(consumer);

threads[i].Name = string.Format("consumer{0}", i - 3);

}

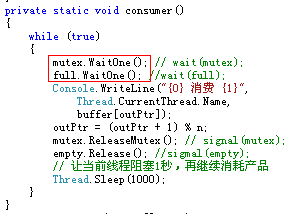
for (int i = 0; i < 6; i++)

{

threads[i].Start();

}

1. 将consumer函数中的full.WaitOne()和mutex.WaitOne()交换位置。



反复运行程序几次。会出现死锁情况吗？若有，是什么原因导致死锁？

1. （选做题）将consumer函数中的full.WaitOne()和mutex.WaitOne()位置交换回来。交换mutex.ReleaseMutex()和empty.Release(); 重新运行程序。运行结果正常吗？可以交换mutex和empty的signal操作的位置吗？若不可以，为什么？若可以，先signal(mutex)再signal(empty)，与先signal(empty)再signal(mutex)，哪个操作更好？

### （四） 用信号量实现读写者问题·

用信号量实现读写者问题，可作为课设。