

# HW4 2150276 沈卓成

---

## 1 创建线程

---

Code:

```
// 创建线程
using System;
using System.Threading;
class Program
{
    static void Main()
    {
        Thread t = new Thread(PrintNumbers);
        t.Start();
        PrintNumbers();
    }
    static void PrintNumbers()
    {
        Console.WriteLine("Starting...");
        for (int i = 1; i <= 10; i++)
        {
            Console.WriteLine(i);
        }
    }
}
```

Ans:

```
(base) PS D:\dotnet\C-\HW4> dotnet run
Starting...
1
2
3
4
5
6
7
8
9
10
Starting...
1
2
3
4
5
6
7
8
9
10
(base) PS D:\dotnet\C-\HW4>
```

## 2. 暂停线程

Code:

```
// 暂停线程
using System;
using System.Threading;
class Program
{
    static void Main()
    {
        Thread t = new Thread(PrintNumberswithDelay);
        t.Start();
        PrintNumbers();
    }
    static void PrintNumbers()
    {
        Console.WriteLine("PrintNumbers Starting...");
        for (int i = 1; i <= 10; i++)
        {
            Console.WriteLine(i);
        }
    }
    static void PrintNumberswithDelay()
    {
        Console.WriteLine("PrintNumberswithDelay Starting...");
```

```

        for (int i = 1; i <= 10; i++)
        {
            Thread.Sleep(TimeSpan.FromSeconds(2));
            Console.WriteLine(i);
        }
    }
}

```

Ans:

```

(base) PS D:\dotnet\C-\HW4> dotnet run
PrintNumbers Starting...
PrintNumbersWithDelay Starting...
1
2
3
4
5
6
7
8
9
10
1
2
3
4
5
6
7
8
9
10
(base) PS D:\dotnet\C-\HW4>

```

### 3. 等待线程结束join

Code:

```

// 等待线程结束
using System;
using System.Threading;
class Program
{
    static void Main()
    {
        Thread t = new Thread(PrintNumbersWithDelay);
        t.Start();
        t.Join();
        Console.WriteLine("Thread t has ended!");
    }
    static void PrintNumbers()
    {
        Console.WriteLine("PrintNumbers Starting...");
    }
}

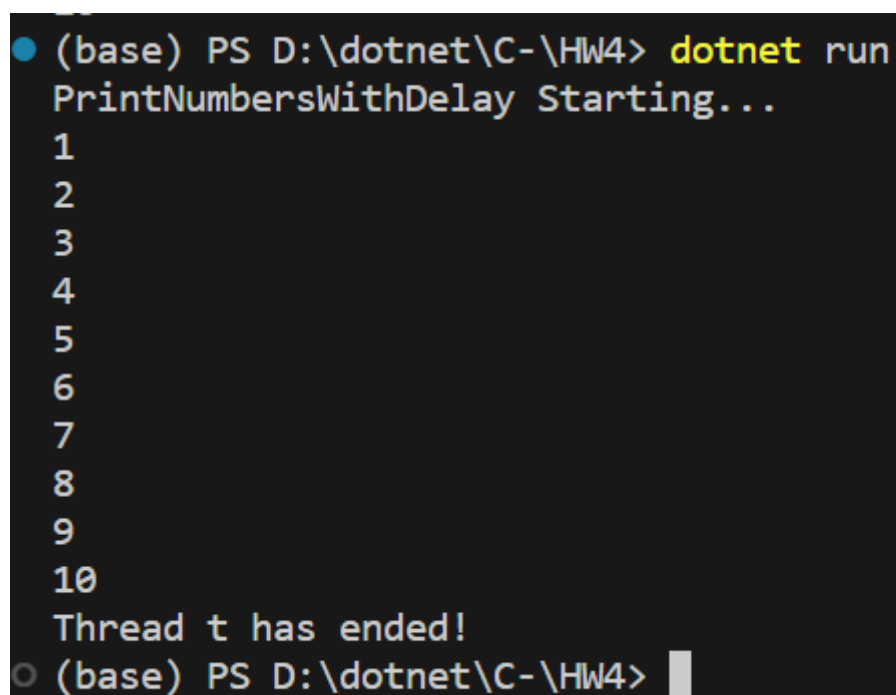
```

```

        for (int i = 1; i <= 10; i++)
        {
            Console.WriteLine(i);
        }
    }
    static void PrintNumbersWithDelay()
    {
        Console.WriteLine("PrintNumbersWithDelay Starting...");
        for (int i = 1; i <= 10; i++)
        {
            Thread.Sleep(TimeSpan.FromSeconds(2));
            Console.WriteLine(i);
        }
    }
}

```

Ans:



```

● (base) PS D:\dotnet\C-\HW4> dotnet run
PrintNumbersWithDelay Starting...
1
2
3
4
5
6
7
8
9
10
Thread t has ended!
○ (base) PS D:\dotnet\C-\HW4>

```

## 4. Abort

Code:

```

// 终止线程
using System;
using System.Threading;
class Program
{
    static void PrintNumberWithDelay()
    {
        Console.WriteLine("PrintNumberWithDelay starting...");
        try
        {

```

```

        for (int i = 0; i < 10; i++)
        {
            Thread.Sleep(TimeSpan.FromSeconds(2));
            Console.WriteLine(i);
        }
    }
    catch (ThreadAbortException)
    {
        Console.WriteLine("PrintNumberWithDelay is being aborted");
        Thread.ResetAbort(); // 重置线程的中止状态
    }
    finally
    {
        Console.WriteLine("PrintNumberWithDelay is done! Exit code is 1");
    }
    Console.WriteLine("PrintNumberWithDelay is done! Exit code is 2");
}
static void Main()
{
    Console.WriteLine("Main thread: starting a dedicated thread to do an asynchronous operation");
    Thread t = new Thread(PrintNumberWithDelay);
    t.Start();
    Thread.Sleep(TimeSpan.FromSeconds(6));
    t.Abort();
    t.Join();
    Console.WriteLine("Main thread is done!");
}
}

```

Ans:

```

(base) PS D:\dotnet\C-\HW4> dotnet run
D:\dotnet\C-\HW4\4.cs(20,13): warning SYSLIB0006: "Thread.ResetAbort()"已过时:"Thread.ResetAbort is not supported and throws PlatformNotSupportedException." (https://aka.ms/dotnet-warnings/SYSLIB0006) [D:\dotnet\C-\HW4\HW4.csproj]
D:\dotnet\C-\HW4\4.cs(34,9): warning SYSLIB0006: "Thread.Abort()"已过时:"Thread.Abort is not supported and throws PlatformNotSupportedException." (https://aka.ms/dotnet-warnings/SYSLIB0006) [D:\dotnet\C-\HW4\HW4.csproj]
Main thread: starting a dedicated thread to do an asynchronous operation
PrintNumberWithDelay starting...
0
1
Unhandled exception. 2
System.PlatformNotSupportedException: Thread abort is not supported on this platform.
   at System.Threading.Thread.Abort()
   at Program.Main() in D:\dotnet\C-\HW4\4.cs:line 34
3

```

在windows10平台上发生运行时错误: System.PlatformNotSupportedException: Thread abort is not supported on this platform.at System.Threading.Thread.Abort(): t.Abort()

## 5. 线程运行状态

Code:

```

// 线程运行状态
using System;
using System.Threading;
class Program
{
    static void DoNothing()
    {
    }
}

```

```

        Thread.Sleep(TimeSpan.FromSeconds(2));
    }
    static void PrintNumberWithStatus()
    {
        Console.WriteLine("PrintNumberWithStatus starting...");
        Console.WriteLine("Thread state: {0}",
Thread.CurrentThread.ThreadState); // 理解底层实现逻辑
        for (int i = 0; i < 10; i++)
        {
            Thread.Sleep(TimeSpan.FromSeconds(2));
            Console.WriteLine(i);
        }
    }
    static void Main()
    {
        Console.WriteLine("start program...");
        Thread t1 = new Thread(PrintNumberWithStatus);
        Thread t2 = new Thread(DoNothing);
        Console.WriteLine("t1 state: {0}", t1.ThreadState);
        t2.Start();
        t1.Start();
        for (int i = 0; i < 30; i++)
        {
            Console.WriteLine("t1 state: {0}, t2 state: {1}", t1.ThreadState,
t2.ThreadState);
        }
        Thread.Sleep(TimeSpan.FromSeconds(6));
        t1.Abort();
        Console.WriteLine("t1 thread has been aborted.");
        Console.WriteLine("t1 state: {0}, t2 state: {1}", t1.ThreadState,
t2.ThreadState);
    }
}

```

Ans:

```
Start program...  
t1 state: Unstarted  
t1 state: Running, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
t1 state: WaitSleepJoin, t2 state: WaitSleepJoin  
PrintNumberWithStatus starting...  
t1 state: Running, t2 state: WaitSleepJoin
```

可以发现：运行状态有：Unstarted, Running, WaitSleepJoin, AbortRequested, Stopped, Background, SuspendRequested和Suspended。

## 6. 线程优先级

Code:

```
// 线程优先级
using System;
using System.Threading;
using System.Diagnostics;
class Program
{
    class ThreadSample
    {
        private bool _isStopped = false;
        public void Stop()
        {
            _isStopped = true;
        }
    }
}
```

```

    }
    public void CountNumbers()
    {
        long counter = 0;
        while (!_isStopped)
        {
            counter++;
        }
        Console.WriteLine("{0} with {1,11} priority has a count = {2,13}",
Thread.CurrentThread.Name, Thread.CurrentThread.Priority.ToString(),
counter.ToString("N0"));
    }
}
static void RunThreads()
{
    var sample = new ThreadSample();
    var threadOne = new Thread(sample.CountNumbers);
    threadOne.Name = "ThreadOne";
    var threadTwo = new Thread(sample.CountNumbers);
    threadTwo.Name = "ThreadTwo";
    threadOne.Priority = ThreadPriority.Highest;
    threadTwo.Priority = ThreadPriority.Lowest;
    threadOne.Start();
    threadTwo.Start();
    Thread.Sleep(TimeSpan.FromSeconds(2));
    sample.Stop();
}
static void Main()
{
    Console.WriteLine("Current thread priority: {0}",
Thread.CurrentThread.Priority);
    Console.WriteLine("Runing on all cores available");
    RunThreads();
    Thread.Sleep(TimeSpan.FromSeconds(2));
    Console.WriteLine("Runing on a single core");
    Process.GetCurrentProcess().ProcessorAffinity = new IntPtr(1); // 这里试试
1, 2, 3, 5, 7
    RunThreads();
    Console.WriteLine("Main method complete.");
}
}

```

Ans:

1. 处理器个数为1

```

Current thread priority: Normal
Runing on all cores available
ThreadTwo with      Lowest priority has a count = 1,254,812,958
ThreadOne with      Highest priority has a count = 1,260,407,910
Runing on a single core
Main method complete.
ThreadOne with      Highest priority has a count = 13,317,079,673
ThreadTwo with      Lowest priority has a count = 62,788,035

```

2. 处理器个数为2



```

Current thread priority: Normal
Runing on all cores available
ThreadOne with Highest priority has a count = 1,086,753,647
ThreadTwo with Lowest priority has a count = 1,086,778,922
Runing on a single core
Main method complete.
ThreadOne with Highest priority has a count = 9,260,118,581
ThreadTwo with Lowest priority has a count = 35,439,147

```

### 3. 处理器个数为3

```

Current thread priority: Normal
Runing on all cores available
ThreadTwo with Lowest priority has a count = 757,520,691
ThreadOne with Highest priority has a count = 992,184,627
Runing on a single core
Main method complete.
ThreadOne with Highest priority has a count = 389,665,972
ThreadTwo with Lowest priority has a count = 378,429,817

```

### 4. 处理器个数为5

```

Current thread priority: Normal
Runing on all cores available
ThreadTwo with Lowest priority has a count = 757,520,691
ThreadOne with Highest priority has a count = 992,184,627
Runing on a single core
Main method complete.
ThreadOne with Highest priority has a count = 389,665,972
ThreadTwo with Lowest priority has a count = 378,429,817

```

### 5. 处理器个数为7

```

Current thread priority: Normal
Runing on all cores available
ThreadTwo with Lowest priority has a count = 1,340,168,108
ThreadOne with Highest priority has a count = 1,356,420,310
Runing on a single core
Main method complete.
ThreadOne with Highest priority has a count = 1,188,231,995
ThreadTwo with Lowest priority has a count = 1,151,109,080

```

## 7. 前后台线程

Code:

```

// 前后台线程
using System;
using System.Threading;
class Program
{
    static void Main()
    {
        var sampleForeground = new ThreadSample(10);
        var sampleBackground = new ThreadSample(20);
        var threadOne = new Thread(sampleForeground.CountNumbers);
        threadOne.Name = "ForegroundThread";
    }
}

```

```

var threadTwo = new Thread(sampleBackground.CountNumbers);
threadTwo.Name = "BackgroundThread";
threadTwo.IsBackground = true; // 当主线程结束时，后台线程会被终止。
// IsBackground属性用于控制线程的生命周期。通过将线程设置为后台线程，可以使应用程序
在主线程结束时更容易退出，而无需等待后台线程完成。
threadOne.Start();
threadTwo.Start();
// threadTwo.Join(); // 等待后台线程结束 这里是我自己加的
}
class ThreadSample
{
    private readonly int _iterations;
    public ThreadSample(int iterations)
    {
        _iterations = iterations;
    }
    public void CountNumbers()
    {
        for (int i = 0; i < _iterations; i++)
        {
            Thread.Sleep(TimeSpan.FromSeconds(0.5));
            Console.WriteLine("{0} prints {1}", Thread.CurrentThread.Name,
i);
        }
    }
}
}

```

Ans:

```
● (base) PS D:\dotnet\C-\HW4> dotnet run
ForegroundThread prints 0
BackgroundThread prints 0
BackgroundThread prints 1
ForegroundThread prints 1
BackgroundThread prints 2
ForegroundThread prints 2
ForegroundThread prints 3
BackgroundThread prints 3
ForegroundThread prints 4
BackgroundThread prints 4
BackgroundThread prints 5
ForegroundThread prints 5
BackgroundThread prints 6
ForegroundThread prints 6
BackgroundThread prints 7
ForegroundThread prints 7
ForegroundThread prints 8
BackgroundThread prints 8
ForegroundThread prints 9
BackgroundThread prints 9
○ (base) PS D:\dotnet\C-\HW4> █
```

## 8. 向线程传递参数

---

Code:

```
// 向线程传递参数
using System;
using System.Threading;
class Program
{
    class ThreadSample
    {
        private readonly int _iterations;
        public ThreadSample(int iterations)
        {
            _iterations = iterations;
        }
        public void CountNumbers()
        {
            for (int i = 1; i <= _iterations; i++)
            {
                Thread.Sleep(TimeSpan.FromSeconds(0.5));
                Console.WriteLine("{0} prints {1}", Thread.CurrentThread.Name,
i);
            }
        }
    }
}
```

```

    }
}
static void Count(object iterations)
{
    CountNumbers((int)iterations);
}
static void CountNumbers(int iterations)
{
    for (int i = 1; i <= iterations; i++)
    {
        Thread.Sleep(TimeSpan.FromSeconds(0.5));
        Console.WriteLine("{0} prints {1}", Thread.CurrentThread.Name, i);
    }
}
// 使用ref传递参数
static void PrintNumbers(int number)
{
    Console.WriteLine(number++);
}
static void Main()
{
    var sample = new ThreadSample(10);
    var threadOne = new Thread(sample.CountNumbers);
    threadOne.Name = "ThreadOne";
    threadOne.Start();
    threadOne.Join();

    Console.WriteLine("-----");

    var threadTwo = new Thread(Count);
    threadTwo.Name = "ThreadTwo";

    threadTwo.Start(8);
    threadTwo.Join();

    Console.WriteLine("-----");

    var threadThree = new Thread(() => PrintNumbers(12));
    threadThree.Name = "ThreadThree";
    threadThree.Start();
    threadThree.Join();

    Console.WriteLine("-----");

    int number = 10;
    var threadFour = new Thread(() => PrintNumbers(number));
    number = 20;
    var threadFive = new Thread(() => PrintNumbers(number));
    threadFour.Name = "ThreadFour";
    threadFive.Name = "ThreadFive";
    threadFour.Start();
    threadFive.Start();

    Console.WriteLine("-----");
}
// 使用out参数实验
static void TestOut(out int o)

```

```

{
    o = 10;
    // 使用未赋值的参数
    // 所有路径都需要设定
    // int n = o;
    // o=1;
}

// 参数数组
static void AnyNumberInts(params int[] intArray)
{
    Console.WriteLine("Number of ints: {0}", intArray.Length);
    foreach (var number in intArray)
    {
        Console.WriteLine(number);
    }
}

// 可选参数和命名参数
static void OptionalParameters(int first, int second = 10, int third = 20)
{
    Console.WriteLine("first = {0}, second = {1}, third = {2}", first,
second, third);
}
static void Test()
{
    OptionalParameters(1, third: 30);
    AnyNumberInts();
    AnyNumberInts(1, 2, 3, 4, 5);
}
}

```

Ans:

```
ThreadOne prints 1
ThreadOne prints 2
ThreadOne prints 3
ThreadOne prints 4
ThreadOne prints 5
ThreadOne prints 6
ThreadOne prints 7
ThreadOne prints 8
ThreadOne prints 9
ThreadOne prints 10
-----
ThreadTwo prints 1
ThreadTwo prints 2
ThreadTwo prints 3
ThreadTwo prints 4
ThreadTwo prints 5
ThreadTwo prints 6
ThreadTwo prints 7
ThreadTwo prints 8
-----
12
-----
20
-----
20
```

Note: 创建一个 lambda 匿名方法时，它可以捕获周围作用域中的变量。

## 9. 线程局部存储

Code:

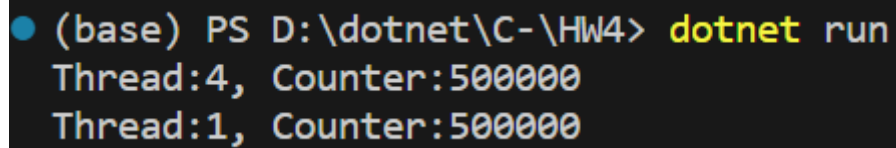
```
// 使用线程局部存储
using System;
using System.Threading;
class Program
{
    [ThreadStatic] // 要是没有这个，那么counterPerThread就会变成共享的，又因为没有加锁，
    所以会出现问题
    static int counterPerThread = 0;
    static void PrintNumbers()
    {
        for (int i = 0; i < 500000; i++)
        {
            counterPerThread++;
        }
        Console.WriteLine("Thread:{0}, Counter:{1}",
            Thread.CurrentThread.ManagedThreadId, counterPerThread);
    }
}
```

```

    }
    static void Main()
    {
        Thread t1 = new Thread(PrintNumbers);
        t1.Start();
        PrintNumbers();
    }
}

```

Ans:



```

(base) PS D:\dotnet\C-\HW4> dotnet run
Thread:4, Counter:500000
Thread:1, Counter:500000

```

Note: ThreadStatic作用很重要，如果没有，counterPerThread会变成共享的，且没有锁，所以会出现意想不到的问题。

## 10. 线程函数处理异常

Code:

```

// 线程函数要处理异常
using System;
using System.Threading;
class Program
{
    static void BadFaultyThread()
    {
        Console.WriteLine("Starting a faulty thread...");
        Thread.Sleep(TimeSpan.FromSeconds(2));
        throw new Exception("Boom!");
    }
    static void FaultyThread()
    {
        try
        {
            Console.WriteLine("Starting a faulty thread...");
            Thread.Sleep(TimeSpan.FromSeconds(1));
            throw new Exception("Boom!");
        }
        catch (Exception ex)
        {
            Console.WriteLine("Exception handled: {0}", ex.Message);
        }
    }
    static void Main()
    {
        var t = new Thread(FaultyThread);
        t.Start();
        t.Join();
        Console.WriteLine("-----");
    }
}

```

```

        // 无法捕获到新线程里面的异常
        try
        {
            var badThread = new Thread(BadFaultyThread);
            badThread.Start();
            badThread.Join();
        }
        catch (Exception ex)
        {
            Console.WriteLine("We won't get here!");
        }
        // 异常导致进程结束，这里无法执行
        Console.WriteLine("Main end.");
    }
}

```

Ans:

```

ⓧ (base) PS D:\dotnet\C-\HW4> dotnet run
D:\dotnet\C-\HW4\10.cs(38,26): warning CS0168: 声明了变量“ex”，但从未使用过 [D:\dotnet\C-\HW4\HW4.csproj]
Starting a faulty thread...
Exception handled: Boom!
-----
Starting a faulty thread...
Unhandled exception. System.Exception: Boom!
   at Program.BadFaultyThread() in D:\dotnet\C-\HW4\10.cs:line 10
   at System.Threading.Thread.StartCallback()

```

可以看到，try catch部分并没有捕获System.Exception异常。

## 11. Monitor lock

Code:

```

// 使用Monitor Lock（分析错误的原因）
using System;
using System.Threading;
class Program
{
    abstract class CountBase
    {
        public abstract void Increment();
        public abstract void Decrement();
    }
    class Count : CountBase
    {
        private int _count { get; set; }
        public override void Increment()
        {
            _count++;
        }
        public override void Decrement()
        {
            _count--;
        }
        public int GetCount()
        {
            return _count;
        }
    }
}

```



```

    }
}
class CounterWithMonitor : CountBase
{
    private readonly object _syncRoot = new object();
    public int _count { get; private set; }
    public override void Increment()
    {
        bool lockTaken = false;
        try
        {
            Monitor.Enter(_syncRoot, ref lockTaken);
            _count++;
        }
        finally
        {
            if (lockTaken)
            {
                Monitor.Exit(_syncRoot);
            }
        }
    }
    public override void Decrement()
    {
        bool lockTaken = false;
        try
        {
            Monitor.Enter(_syncRoot, ref lockTaken);
            _count--;
        }
        finally
        {
            if (lockTaken)
            {
                Monitor.Exit(_syncRoot);
            }
        }
    }
}
class CounterWithLock : CountBase
{
    private readonly object _syncRoot = new object();
    public int _count { get; private set; }
    public override void Increment()
    {
        lock (_syncRoot)
        {
            _count++;
        }
    }
    public override void Decrement()
    {
        lock (_syncRoot)
        {
            _count--;
        }
    }
}

```

```

static void TestCounter(CountBase counter)
{
    for (int i = 0; i < 100000; i++)
    {
        counter.Increment();
        counter.Decrement();
    }
}

static void Main()
{
    Console.WriteLine("Incorrect counter");
    var counter = new Count();
    var thread1 = new Thread(() => TestCounter(counter));
    var thread2 = new Thread(() => TestCounter(counter));
    var thread3 = new Thread(() => TestCounter(counter));
    thread1.Start();
    thread2.Start();
    thread3.Start();
    thread1.Join();
    thread2.Join();
    thread3.Join();
    Console.WriteLine("Count: {0}", counter.GetCount());

    Console.WriteLine("Counter with Monitor");
    var counterWithMonitor = new CounterWithMonitor();
    var thread4 = new Thread(() => TestCounter(counterWithMonitor));
    var thread5 = new Thread(() => TestCounter(counterWithMonitor));
    var thread6 = new Thread(() => TestCounter(counterWithMonitor));
    thread4.Start();
    thread5.Start();
    thread6.Start();
    thread4.Join();
    thread5.Join();
    thread6.Join();
    Console.WriteLine("Count: {0}", counterWithMonitor._count);

    Console.WriteLine("Counter with Lock");
    var counterWithLock = new CounterWithLock();
    var thread7 = new Thread(() => TestCounter(counterWithLock));
    var thread8 = new Thread(() => TestCounter(counterWithLock));
    var thread9 = new Thread(() => TestCounter(counterWithLock));
    thread7.Start();
    thread8.Start();
    thread9.Start();
    thread7.Join();
    thread8.Join();
    thread9.Join();
    Console.WriteLine("Count: {0}", counterWithLock._count);
}
}

```

Ans:

```
● (base) PS D:\dotnet\C-\HW4> dotnet run
Incorrect counter
Count: 201
Counter with Monitor
Count: 0
Counter with Lock
Count: 0
```

Note: 使用Monitor lock可以保证线程安全。

## 12. 死锁

Code:

```
// 死锁
using System;
using System.Threading;
class Program
{
    static void LockTooMuch(object lock1, object lock2)
    {
        lock (lock1)
        {
            Console.WriteLine("Got lock1");
            Thread.Sleep(1000);
            lock (lock2)
            {
                Console.WriteLine("Got lock2");
            }
        }
    }
    static void Main()
    {
        object lock1 = new object();
        object lock2 = new object();

        new Thread(() => LockTooMuch(lock1, lock2)).Start();
        lock (lock2)
        {
            Console.WriteLine("Got lock2");
            Thread.Sleep(1000);
            Console.WriteLine("Monitor.TryEnter returning false after a
specified timeout is elapsed");
            if (Monitor.TryEnter(lock1, TimeSpan.FromSeconds(5)))
            {
                Console.WriteLine("Got lock1,(Acquired a protected resource
successfully)");
            }
            else
            {
                Console.WriteLine("Failed to get lock1, timeout expired,(Failed
to acquire a protected resource within a specified time)");
            }
        }
    }
}
```

```

    }
}

Console.WriteLine("-----");
new Thread(() => LockTooMuch(lock1, lock2)).Start();
lock (lock2)
{
    Console.WriteLine("This will be deadlocked");
    Thread.Sleep(1000);
    lock (lock1)
    {
        Console.WriteLine("Acquired a protected resource successfully");
    }
}
}
}

```

Ans:

```

Got lock2
Got lock1
Monitor.TryEnter returning false after a specified timeout is elapsed
Failed to get lock1, timeout expired,(Failed to acquire a protected resource within a specified time)
-----
Got lock2
This will be deadlocked
Got lock1

```

使用lock会导致死锁，而使用TryEnter可以避免死锁时的无限等待。

## 13. 使用互锁函数

Code:

```

// 使用互锁函数
using System;
using System.Threading;
internal class Program
{
    abstract class CountBase
    {
        public abstract void Increment();
        public abstract void Decrement();
    }
    class Counter : CountBase
    {
        public int _count = 0;
        public override void Increment()
        {
            _count++;
        }
        public override void Decrement()
        {
            _count--;
        }
    }
    class CounterNoLock : CountBase
    {

```

```

        public int _count = 0;
        public override void Increment()
        {
            Interlocked.Increment(ref _count);
        }
        public override void Decrement()
        {
            Interlocked.Decrement(ref _count);
        }
    }
    static void TestCounter(CountBase c)
    {
        for (int i = 0; i < 1000000; i++)
        {
            c.Increment();
            c.Decrement();
        }
    }
    static void Main()
    {
        Console.WriteLine("Incorrect Counter");
        var counter = new Counter();
        var t1 = new Thread(() => TestCounter(counter));
        var t2 = new Thread(() => TestCounter(counter));
        var t3 = new Thread(() => TestCounter(counter));
        t1.Start();
        t2.Start();
        t3.Start();
        t1.Join();
        t2.Join();
        t3.Join();
        Console.WriteLine("Count: {0}", counter._count);

        Console.WriteLine("-----");

        Console.WriteLine("Counter with Interlocked");
        var counterNoLock = new CounterNoLock();
        var t4 = new Thread(() => TestCounter(counterNoLock));
        var t5 = new Thread(() => TestCounter(counterNoLock));
        var t6 = new Thread(() => TestCounter(counterNoLock));
        t4.Start();
        t5.Start();
        t6.Start();
        t4.Join();
        t5.Join();
        t6.Join();
        Console.WriteLine("Count: {0}", counterNoLock._count);
    }
}

```

Ans:

```
● (base) PS D:\dotnet\C-\HW4> dotnet run
Incorrect Counter
Count: 3970
-----
Counter with Interlocked
Count: 0
```

## 14. 使用生产者消费者队列

Code:

```
// 生产者消费者问题
using System;
using System.Threading;
using System.Collections.Concurrent;
using System.Threading.Tasks;
using System.Collections.Generic;
using System.Linq;
public class Program{
    public static void Main(){
        PCProblem pcProblem = new PCProblem();
        Thread producer1 = new Thread(new ThreadStart(pcProblem.producer));
        producer1.Name = "生产者1";
        Thread producer2 = new Thread(new ThreadStart(pcProblem.producer));
        producer2.Name = "生产者2";
        Thread producer3 = new Thread(new ThreadStart(pcProblem.producer));
        producer3.Name = "生产者3";
        Thread consumer1 = new Thread(new ThreadStart(pcProblem.consumer));
        consumer1.Name = "消费者1";
        Thread consumer2 = new Thread(new ThreadStart(pcProblem.consumer));
        consumer2.Name = "消费者2";
        producer1.Start();
        producer2.Start();
        producer3.Start();
        consumer1.Start();
        consumer2.Start();
    }
}
internal class PCProblem{
    int goodsNum = 0;
    Queue<int> goods = new Queue<int>();
    Semaphore buffer = new Semaphore(1, 1); // 缓冲区
    Semaphore empty = new Semaphore(5, 5); // 生产者能够生产的物品数量
    Semaphore full = new Semaphore(0, 5); // 消费者能够消费的物品数量

    public void producer(){
        while(true){
            empty.WaitOne(); // 等待生产者信号量
            buffer.WaitOne();
            goodsNum++;
            goods.Enqueue(goodsNum);
            Console.WriteLine(Thread.CurrentThread.Name + "生产者生产了物品" +
            goodsNum.ToString());
        }
    }

    public void consumer(){
        while(true){
            full.WaitOne();
            goods.Dequeue();
            goodsNum--;
            Console.WriteLine(Thread.CurrentThread.Name + "消费者消费了物品" +
            goodsNum.ToString());
        }
    }
}
```

```

        Thread.Sleep(1000);
        Console.WriteLine("当前有" + goods.Count + "个物品");
        Thread.Sleep(1000);
        full.Release(); // 释放一个消费者信号量
        buffer.Release();
    }
}

public void consumer(){
    while(true){
        full.WaitOne(); // 等待消费者信号量
        buffer.WaitOne();
        int goodNum = goods.Dequeue();
        Console.WriteLine(Thread.CurrentThread.Name + "消费者消费了物品" +
goodNum.ToString());
        Thread.Sleep(1000);
        Console.WriteLine("当前有" + goods.Count + "个物品");
        Thread.Sleep(1000);
        empty.Release(); // 释放一个生产者信号量
        buffer.Release();
    }
}
}

```

Ans:

```
(base) PS D:\dotnet\C-\HW4> dotnet run
```

```
生产者3生产者生产了物品1  
当前有1个物品  
生产者2生产者生产了物品2  
当前有2个物品  
生产者1生产者生产了物品3  
当前有3个物品  
生产者3生产者生产了物品4  
当前有4个物品  
消费者1消费者消费了物品1  
当前有3个物品  
消费者2消费者消费了物品2  
当前有2个物品  
生产者2生产者生产了物品5  
当前有3个物品  
消费者1消费者消费了物品3  
当前有2个物品  
生产者1生产者生产了物品6  
当前有3个物品  
消费者2消费者消费了物品4  
当前有2个物品  
生产者3生产者生产了物品7  
当前有3个物品  
消费者1消费者消费了物品5  
当前有2个物品  
生产者2生产者生产了物品8  
当前有3个物品  
生产者1生产者生产了物品9  
当前有4个物品  
消费者2消费者消费了物品6  
当前有3个物品  
消费者1消费者消费了物品7  
当前有2个物品  
生产者3生产者生产了物品10  
当前有3个物品  
生产者2生产者生产了物品11
```

生产者每次生产一个物品，消费者每次消费一个物品，生产者和消费者之间通过信号量进行同步，保证了生产者和消费者之间的同步。缓冲区size=1保证互斥性。