**新建项目**

**计算机学院（软件学院）实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **学生姓名** | 徐可可 | **学号** | 171530425 | **实验成绩** |  |
| **专业** | 软件工程 | **班级** | 软工四班 | **实验日期** | 19年11月13日 |
| **课程名称** | C#高级程序设计 | | | **任课教师** | 彭伟国 |
| **实验名称** | 线程的使用 | | | **实验序号** | 5 |
| **实验地点** | S409 | **实验台号** | 25 | **指导教师** | 彭伟国 |
| 1. **实验目的及要求**   1．掌握线程的创建与控制，包括创建线程、暂停和恢复线程、终止线程、设置线程执行的优先级。 2．理解实现线程同步方法。 | | | | | |
| 1. **实验内容（或实验原理、实验拓扑）**   1. 尝试开发一个程序，要求通过使用线程休眠控制程序的执行效果。  2. 写两个线程，交替打印1-100，看谁先执行完；写两个线程，分别打印1-100，看谁先执行完。  3. 写两个线程，运用互斥锁（在一个时刻内只允许一个线程进入执行，而其他线程必须等待）原理解决取钱问题。 | | | | | |
| 1. **实验设备与环境**   1. 高配笔记本：Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz  、16G内存、48TB分布式硬盘、Windows 10 1909政府版、Visual Studio 2019 企业版、.NET Framework 4.7.2  2. 实验室电脑：英特尔 酷睿2 双核 T7700 @ 2.40GHz、3G内存、128 G虚拟硬盘、Windows 7 旗舰版 32位 SP1、Microsoft Visual Studio Ultimate 2012、.NET Framework 4.5 | | | | | |
| 1. **实验设计方案（包括实验步骤、设计思想、算法描述或开发流程等）** | | | | | |

|  |
| --- |
| 1. **实验结果（包括设计效果、测试数据、运行结果等）** 2. 尝试开发一个程序，要求通过使用线程休眠控制程序的执行效果。 在这里插入图片描述 3. 写两个线程，交替打印1-100，看谁先执行完；写两个线程，分别打印1-100，看谁先执行完。 在这里插入图片描述 4. 写两个线程，运用互斥锁（在一个时刻内只允许一个线程进入执行，而其他线程必须等待）原理解决取钱问题。 在这里插入图片描述 |
| 1. **实验小结（包括收获、心得体会、注意事项、存在问题及解决办法、建议等）**   进程和线程、协程   1. 进程 进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位。每个进程都有自己的独立内存空间，不同进程通过进程间通信来通信。由于进程比较重量，占据独立的内存，所以上下文进程间的切换开销（栈、寄存器、虚拟内存、文件句柄等）比较大，但相对比较稳定安全。 2. 线程 线程是进程的一个实体,是CPU调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源。线程间通信主要通过共享内存，上下文切换很快，资源开销较少，但相比进程不够稳定容易丢失数据。 3. 协程 协程是一种用户态的轻量级线程，协程的调度完全由用户控制。协程拥有自己的寄存器上下文和栈。协程调度切换时，将寄存器上下文和栈保存到其他地方，在切回来的时候，恢复先前保存的寄存器上下文和栈，直接操作栈则基本没有内核切换的开销，可以不加锁的访问全局变量，所以上下文的切换非常快。 |

|  |
| --- |
| 1. **附录（包括作品、流程图、源程序及命令清单等）** 2. 尝试开发一个程序，要求通过使用线程休眠控制程序的执行效果。 **Program.cs**   using System;  using System.Threading;  namespace Test\_5  {  class Program  {  //1.尝试开发一个程序，要求通过使用线程休眠控制程序的执行效果。  static void Main(string[] args)  {  Program obj1 = new Program();  Thread thread1 = new Thread(new ThreadStart(obj1.Count));  thread1.Name = "线程1";  Program obj2 = new Program();  Thread thread2 = new Thread(new ThreadStart(obj2.Count));  thread2.Name = "线程2";  Program obj3 = new Program();  Thread thread3 = new Thread(new ThreadStart(obj3.Count));  thread3.Name = "线程3";  thread1.Start();  thread2.Start();  thread3.Start();  Console.ReadKey();  }  private int cnt = 0;  private void Count()  {  while (cnt < 10)  {  cnt++;  Console.WriteLine(Thread.CurrentThread.Name + "数到" + cnt);  Thread.Sleep(100);  }  }  }  }   1. 写两个线程，交替打印1-100，看谁先执行完；写两个线程，分别打印1-100，看谁先执行完。 **Program.cs**   using System;  using System.Threading;  namespace \_5\_2  {  class Program  {  static void Main(string[] args)  {  //2.写两个线程，交替打印1-100，看谁先执行完；写两个线程，分别打印1-100，看谁先执行完。  //3.写两个线程，运用互斥锁（在一个时刻内只允许一个线程进入执行，而其他线程必须等待）原理解决取钱问题。    Program obj1 = new Program();  Thread thread1 = new Thread(new ThreadStart(obj1.Count1));  thread1.Name = "线程1";  Program obj2 = new Program();  Thread thread2 = new Thread(new ThreadStart(obj2.Count1));  thread2.Name = "线程2";  thread1.Start();  thread2.Start();  Console.ReadKey();  }  private static int cnt1 = 0;  private int cnt2 = 1;  private void Count1()  {  while (cnt1 < 100)  {  cnt1++;  Console.WriteLine(Thread.CurrentThread.Name + "数到" + cnt1);  Thread.Sleep(100);  }  }  private void Count2()  {  while (cnt2 < 100)  {  cnt2++;  Console.WriteLine(Thread.CurrentThread.Name + "数到" + cnt2);  Thread.Sleep(100);  }  }  }  }   1. 写两个线程，运用互斥锁（在一个时刻内只允许一个线程进入执行，而其他线程必须等待）原理解决取钱问题。 **Program.cs**   using System;  using System.Threading;  namespace \_5\_3  {  class Program  {  public static int balance = 3000;  public static object o = new object();  static void Main(string[] args)  {  Program p = new Program();  Thread t1 = new Thread(p.GetMoney);  t1.Name = "t1";  Thread t2 = new Thread(p.GetMoney);  t2.Name = "t2";  t1.Start(1000);  t2.Start(500);  //Console.WriteLine("\nPress any key to quit.");  Console.ReadKey();  }  public void GetMoney(object num)  {  //互斥段在一个时刻内只允许一个线程进入执行，而其他线程必须等待。  //这是通过在代码块运行期间为给定对象获取互斥锁来实现的。  lock (o)//互斥锁  {  int a = (int)num;  balance -= a;  Thread.Sleep(100);  Console.WriteLine(Thread.CurrentThread.Name + ":" + balance);  }  }  }  } |