Матюков Николай Юрьевич ПС-31

Лабораторная работа №3

2 вариант

**Постановка задачи.**

Знакомство с возможностями стандартной библиотеки языка c# для работы с потоками. Изучение классов Mutex, Interlocked, AutoResetEvent. Реализация критической секции с использованием данных классов.

Задание: Написать класс (на языке c#), реализующий следующий интерфейс:

public interface ICriticalSection

{

//Вход в критическую секцию

void Enter();

//Вход в критическую секцию (timeout – таймаут в мс). Если произошел вход в критическую секцию, то возвращается - true. Если произошел выход по таймауту, то false

bool TryEnter(int timeout);

//Устанавливается количество попыток входа в критическую секцию в методе Enter перед блокированием потока

void SetSpinCount(int count);

//Выход из критической секции

void Leave();

}

Для демонстрации работы необходимо написать программу вычисления числа π методом численного интегрирования как описано в: <https://habr.com/ru/post/147796/>.

**Описание Алгоритма.**

Считаем число ПИ путем численного интегрирования. Указываем потоковой функции в каком диапазоне считать наш интервал. Результат вычисления из каждого потока будем прибавлять к переменной пи. Чтобы память не испортилась в случае если два потока будут одновременно изменять одну ячейко нам надо как-то гарантировать что в один момент времени только один поток получает доступ к переменной pi, т.е. создать так называемую «критическую секцию».

**Таблица.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Общее количество итераций | Вычисленное значение числа π | Коли-чество spin’ов | Длительность timeout, мс | | | | | |
| 10 | | 20 | | 30 | |
| Enter | TryEnter | Enter | TryEnter | Enter | TryEnter |
| 10000000 | 3.14159265358956 | 50 | 594 | 1285 | 647 | 1387 | 555 | 1331 |
| 250 | 1307 | 1714 | 903 | 1897 | 1287 | 1770 |
| 500 | 1969 | 2133 | 1790 | 2035 | 2077 | 2230 |
| 50000000 | 3.14159265359045 | 50 | 2685 | 6948 | 2724 | 6771 | 3186 | 6647 |
| 250 | 6107 | 8776 | 6211 | 8892 | 4828 | 8891 |
| 500 | 10788 | 11132 | 10638 | 10943 | 10551 | 10595 |
| 7000000000 | 3.14159265358949 | 50 | 3743 | 9420 | 4076 | 9046 | 3801 | 9358 |
| 250 | 8809 | 11842 | 8665 | 12190 | 10947 | 13203 |
| 500 | 13950 | 14677 | 13977 | 15232 | 14793 | 14862 |

**Выводы.**

При увеличении количества спинов уменьшаяется производительность. Длительность таймаута практически не влияет на производительность. Но tryEnter занимает больше времени чем Enter скорее всего и-за того, что в цикле мы каждый раз проверяем время.

**Листинг программы.**

using System;  
using System.Diagnostics;  
  
namespace lw03  
{  
 class Program  
 {  
 private const int **NUMBER\_OF\_THREADS** = 4;   
 static void Main(string[] args)  
 {  
 if (args.Length > 3)  
 {  
 Console.WriteLine("Incorrect number of arguments.");  
 return;  
 }  
   
 int numberOfSteps= Int32.Parse(args[0]);  
 int timeout = Int32.Parse(args[1]);  
 int numberOfSpins = Int32.Parse(args[2]);  
   
 Stopwatch time = new Stopwatch();  
   
 Console.WriteLine("Enter: ");  
 time.Start();  
 double **pi** = PiCalculator.GetPi(**NUMBER\_OF\_THREADS**, numberOfSteps, numberOfSpins, timeout, true);  
 time.Stop();  
 Console.WriteLine(" Time: " + time.ElapsedMilliseconds);  
 Console.WriteLine(" Pi = " + **pi**);  
   
 time.Reset();  
 Console.WriteLine("TryEnter: ");  
 time.Start();  
 **pi** = PiCalculator.GetPi(**NUMBER\_OF\_THREADS**, numberOfSteps, numberOfSpins, timeout, false);  
 time.Stop();  
 Console.WriteLine(" Time: " + time.ElapsedMilliseconds);  
 Console.WriteLine(" Pi = " + **pi**);  
 }  
 }  
}

namespace lw03  
{  
 public interface ICriticalSection  
 {  
 //Вход в критическую секцию  
 void Enter();  
  
 //Вход в критическую секцию (timeout – таймаут в мс). Если произошел вход в критическую секцию, то возвращается - true. Если произошел выход по таймауту, то false  
 bool TryEnter(int timeout);  
  
 //Устанавливается количество попыток входа в критическую секцию в методе Enter перед блокированием потока  
 void SetSpinCount(int count);  
  
 //Выход из критической секции  
 void Leave();  
 }  
}

using System;  
using System.Diagnostics;  
using System.Threading;  
  
namespace lw03  
{  
 public class CriticalSectionImpl : ICriticalSection  
 {  
 private int value;  
 private int spinCount;  
   
 public CriticalSectionImpl(int count)  
 {  
 this.value = 1;  
 this.spinCount = count;  
 }  
  
 public void Enter()  
 {  
 while(true)  
 {  
 for(int **i** = 0; **i** < spinCount; ++**i**)  
 {  
 if(Interlocked.CompareExchange(ref value, 2, 1) == 1)  
 {  
 return;  
 }  
 }  
 Thread.Sleep(10);  
 }  
 }  
  
 public void Leave()  
 {  
 if(Interlocked.CompareExchange(ref value, 1, 2) != 2)  
 {  
 throw new Exception("Cant leave from section.");  
 }  
 }  
  
 public void SetSpinCount(int count)  
 {  
 spinCount = count;  
 }  
  
 public bool TryEnter(int timeout)  
 {  
 var time = new Stopwatch();  
 time.Start();  
 while(time.ElapsedMilliseconds < timeout)  
 {  
 for(int **i** = 0; **i** < spinCount; ++**i**)  
 {  
 if(Interlocked.CompareExchange(ref value, 2, 1) == 1)  
 {  
 return true;  
 }  
 }  
 Thread.Sleep(10);  
 }  
   
 return false;  
 }  
   
 }  
}

namespace lw03  
{  
 public class ArgsThread  
 {  
 public long left;  
  
 public long right;  
  
 public double step;  
  
 public ArgsThread(long left, long right, double step)  
 {  
 this.left = left;  
 this.right = right;  
 this.step = step;  
 }  
 }  
}

using System;  
using System.Threading;  
  
namespace lw03  
{  
 public class PiCalculator  
 {  
 private static ICriticalSection criticalSection;  
  
 private static double pi = 0.0;  
  
 private static int timeout;  
  
 public static double GetPi(int numberOfThreads, int numberOfSteps, int numberOfSpins, int time, bool isEnter)  
 {  
 Thread[] threads = new Thread[numberOfThreads];  
 int steps = numberOfSteps;  
 timeout = time;  
 criticalSection = new CriticalSectionImpl(numberOfSpins);  
 int iterations = numberOfSteps / numberOfThreads;  
 pi = 0;  
   
 double step = 1.0 / steps;  
 for (int **i** = 0; **i** < numberOfThreads; ++**i**)  
 {  
 Thread thread = isEnter ? new Thread(CalculateWithEnter) : new Thread(CalculateWithTryEnter);  
 threads[**i**] = thread;  
   
 ArgsThread argsThread = new ArgsThread(**i** \* iterations, (**i** + 1) \* iterations, step);  
 thread.Start(argsThread);  
 }  
   
 for(int **i** = 0; **i** < numberOfThreads; ++**i**)  
 {  
 threads[**i**].Join();  
 }  
  
 return pi;  
  
 }  
  
 private static void CalculateWithEnter(object arg)  
 {  
 if (arg.GetType() != typeof(ArgsThread))  
 {  
 throw new Exception("Incorrect type");  
 }  
 ArgsThread argsThread = (ArgsThread)arg;  
 for(long **i** = argsThread.left; **i** < argsThread.right; ++**i**)  
 {  
 double x = (**i** + 0.5) \* argsThread.step;  
 double value = 4.0 / (1 + x \* x) \* argsThread.step;  
   
 criticalSection.Enter();  
 pi += value;  
 criticalSection.Leave();  
 }  
 }  
  
 private static void CalculateWithTryEnter(object arg)  
 {  
 if (arg.GetType() != typeof(ArgsThread))  
 {  
 throw new Exception("Incorrect type");  
 }  
 ArgsThread argsThread = (ArgsThread)arg;  
 for(long **i** = argsThread.left; **i** < argsThread.right;)  
 {  
 double x = (**i** + 0.5) \* argsThread.step;  
 double value = 4.0 / (1 + x \* x) \* argsThread.step;  
   
 if(criticalSection.TryEnter(timeout))  
 {  
 pi += value;  
 criticalSection.Leave();  
 ++**i**;  
 }  
 }  
 }  
 }  
}