МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ   
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчёт по лабораторной работе №4  
по дисциплине «Параллельные вычисления»

**Синхронизация в C#**

Выполнил: студент группы ФИб-4301-51-00     / К.О. Дёмин /

Проверил: ст.преподаватель каф. ПМИ     / А.В. Торбеева /

Киров 2019

Оглавление

[Цель работы 3](#_Toc25497353)

[Задание 3](#_Toc25497354)

[Листинг 4](#_Toc25497355)

[Полученные результаты 10](#_Toc25497356)

[Выводы по лабораторной работе 12](#_Toc25497357)

# Цель работы

Рассмотреть различные способы синхронизации и исследовать их применение.

# Задание

1. Два потока работают с целочисленным массивом. Первый поток произвольным образом изменяет элементы массива, второй поток выводит случайный диапазон длины 5 элементов данного массива (например, с 3 по 7). Организовать правильную работу имеющихся потоков.
2. Имеется хранилище данных, с которым работают одновременно несколько потоков. Первые *N* потоков случайным образом изменяют данные в хранилище (при этом изменять данные в некоторый момент времени может ровно один поток), остальные *M* потоков периодически читают данные из хранилища. Несколько читателей могут находиться в хранилище одновременно, но читатель и писатель не могут одновременно использовать хранилище. Эмулировать работу хранилища.
3. Разработать параллельный алгоритм вычисления определенного интеграла методом прямоугольников с точностью ε (задаётся пользователем).



Реализовать полученный алгоритм с использованием потоков.

Замерить среднее время выполнения программы для малых значений ε (по 2-3 значения, на каждое – не менее 10 запусков) на 1, 2, 4 и 8 потоках. Вычислить среднее ускорение для 2, 4 и 8 потоков. Рассчитать эффективность параллельного алгоритма. Построить диаграммы для ускорения и эффективности.

# Листинг

Задание 1

using System;

using System.Threading.Tasks;

namespace task\_1

{

    class Program

    {

        static void Main(string[] args)

        {

            Console.WriteLine("Введите размер массива:");

            int arraySize = int.Parse(Console.ReadLine());  //  читаем размер массива

            ChangeArray changeArray = new ChangeArray(arraySize);  //  создаём класс

            changeArray.Run();  //  запускаем изменение массива

            Console.ReadKey();

        }

    }

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

namespace task\_1

{

    class ChangeArray

    {

        int[] array;

        int n;

        Thread changeThread;

        Thread printThread;

        Random random = new Random();

        Semaphore[] semaphores;

        public ChangeArray(int n)

        {

            this.n = n;

            array = Enumerable.Range(0, n).ToArray();

            semaphores = new Semaphore[n].Select(i => new Semaphore(1, 1)).ToArray();

        }

        public void Run()

        {

            changeThread = new Thread(() => Change());  //  создаём поток изменения

            printThread = new Thread(() => Print());  //  создаём поток вывода

            changeThread.Start();

            printThread.Start();

        }

        private void Change()

        {

            while(true)

            {

                int number = random.Next(n);  //  генерируем индекс

                semaphores[number].WaitOne();

                Console.WriteLine($"INIT CHANGE Index: {number}");

                array[number] = random.Next(n); //  изменяем значение

                Console.WriteLine($"array[{number}] = {array[number]}");

                semaphores[number].Release();

            }

        }

        private void Print()

        {

            while(true)

            {

                int a = random.Next(n - 4);  //  генерируем начало

                Console.WriteLine($"INIT PRINT from {a} to {a + 4}");

                //  выводим интервал

                List<int> data = new List<int>();

                for (int i = a; i < a + 5; ++i)

                {

                    semaphores[i].WaitOne();

                    data.Add(array[i]);

                    semaphores[i].Release();

                }

                Console.WriteLine($"Print From {a} to {a + 4}: " + string.Join(',', data));

            }

        }

    }

}

Задание 2

using System;

using System.Collections.Generic;

using System.Threading;

namespace StorageEmulator

{

    class Program

    {

        static void Main(string[] args)

        {

            Console.WriteLine("Введите число читателей:");

            int readers = int.Parse(Console.ReadLine());

            Console.WriteLine("Введите число писателей:");

            int writers = int.Parse(Console.ReadLine());

            Storage storage = new Storage();  //  создаём хранилище

            List<Thread> threads = new List<Thread>();  //  список потоков

            bool online = true;  //  статус программы

            //  создаём читателей

            for (int r = 0; r < readers; ++r)

            {

                int id = r + 1;

                Thread thread = new Thread(i => Read(id, storage, ref online));

                threads.Add(thread);

            }

            //  создаём писателей

            for (int w = 0; w < writers; ++w)

            {

                int id = w + 1;

                Thread thread = new Thread(i => Write(id, storage, ref online));

                threads.Add(thread);

            }

            //  запускаем

            threads.ForEach(i => i.Start());

            //  любая кнопка завершает программу

            Console.ReadKey();

            online = false;

            //  ожидаем завершение

            foreach (var t in threads)

                t.Join();

            storage.Dispose();

            Console.WriteLine("Работа завершена");

        }

        //  Поток читает

        static void Read(int id, Storage storage, ref bool online)

        {

            Random random = new Random();  //  рандом

            while (online)  //  пока онлайн

            {

                int index = random.Next() & 8;  //  генерируем индекс

                string data = storage.Read(index);  //  читаем дату и выводим

                Console.WriteLine($"Читатель №{id} прочитал №{index}: {data}");

            }

        }

        //  Поток записывает

        static void Write(int id, Storage storage, ref bool online)

        {

            Random random = new Random();

            while (online)  //  пока онлайн

            {

                int index = random.Next() & 8;  //  генерируем индекс

                string data = Guid.NewGuid().ToString();  //  генерируем текст

                storage.Write(index, data);  //  записываем текст

                Console.WriteLine($"Писатель №{id} записал в №{index} значение - {data}");

            }

        }

    }

}

using System;

using System.Collections.Generic;

using System.Text;

using System.Threading;

namespace StorageEmulator

{

    class Storage : IDisposable

    {

        Dictionary<int, string> data = new Dictionary<int, string>();  //  хранилище

        ReaderWriterLockSlim readerWriterLockSlim;  //  ридер райтер лок, 1 запись, множественное чтение

        public Storage()

        {

            readerWriterLockSlim = new ReaderWriterLockSlim();

        }

        public void Dispose()

        {

            readerWriterLockSlim.Dispose();  //  очищаем объект

        }

        //  чтение из хранилища

        public string Read(int id)

        {

            readerWriterLockSlim.EnterReadLock();  //  входим на чтение

            string temp = null;

            if (!data.TryGetValue(id, out temp))  //  пытаемся считать

                temp = null;

            readerWriterLockSlim.ExitReadLock();  //  разблокируем

            return temp;

        }

        //  запись в хранилище

        public void Write(int id, string value)

        {

            readerWriterLockSlim.EnterWriteLock();  //  блокируем чтение и запись

            if (!data.TryGetValue(id, out string temp))  //  пытаемся найти id в хранилище

                data.Add(id, value);  //  добавляем

            else data[id] = value;  //  изменяем значение

            readerWriterLockSlim.ExitWriteLock();  //  разблокируем

        }

    }

}

Задание 3

using System;

using System.Diagnostics;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

namespace task\_3

{

    class Program

    {

        const int Iterations = 10;  //  число итераций

        static double A = 0.0001d;  //  А

        static double B = 1d;  //  B

        static int P = 10000;

        static Semaphore semaphore;

        static CountdownEvent countdownEvent;

        static bool next = false;

        static void Main(string[] args)

        {

            Console.WriteLine("Введите кол-во потоков:");

            int threadCount = int.Parse(Console.ReadLine());

            Console.WriteLine("Введите степень");

            int numbers = int.Parse(Console.ReadLine());

            double e = Math.Pow(10, -numbers);

            Stopwatch stopwatch = new Stopwatch();  //  объект для замерения времени

            TimeSpan timeSpan = new TimeSpan();

            countdownEvent = new CountdownEvent(threadCount);

            semaphore = new Semaphore(0, threadCount);

            //  вычисляем итерациями

            for (int iter = 0; iter < Iterations; ++iter)

            {

                countdownEvent.Reset();

                double[] s0 = new double[P];

                int[] ns = new int[P];

                int step = -1;

                double result = 0;

                double oldRes = 0;

                double eps = e;

                stopwatch.Restart();  //  перезапускаем замер времени

                                      //  создаём потоки в списке

                var threads = Enumerable.Range(0, threadCount).Select(i => new Thread(j => Calc(ref step, ref eps, s0, ns)))

                    .ToList();

                //  запускаем потоки

                threads.ForEach(i => i.Start());

                do

                {

                    oldRes = result;

                    countdownEvent.Wait();  //  ждём все потоки

                    result = s0.Sum();  //  считаем результат

                    if (Math.Abs(result - oldRes) > e)

                    {

                        eps \*= 0.5d;

                        countdownEvent.Reset();

                        next = true;

                        step = -1;

                    } else next = false;

                    semaphore.Release(threadCount);

                }

                while (next);

                stopwatch.Stop();  // завершаем замер времени

                timeSpan += stopwatch.Elapsed;  //  добавляем время

                Console.WriteLine("Ответ: {0}", result.ToString($"f{numbers}"));

            }

            Console.WriteLine($"Время: {timeSpan.TotalSeconds / Iterations:f3} сек.");  //  выводим информацию о времени

            Console.ReadKey();

        }

        /// <summary>

        /// функция интеграла

        /// </summary>

        /// <param name="t"></param>

        /// <returns></returns>

        static double Func(double t)

        {

            return Math.Sin(t) / t;

        }

        /// <summary>

        /// Метод вычисления на поток

        /// </summary>

        /// <param name="step"></param>

        /// <param name="e"></param>

        /// <param name="results"></param>

        /// <param name="ns"></param>

        static void Calc(ref int step, ref double e, double[] results, int[] ns)

        {

            double dx = (B - A) / P;  //  шаг

            int num;  //  номер интервала для вычисления

            while (true)

            {

                //  получаем номер интервала для вычисления и вычисляем интервалы до тех пор, пока номер меньше количества интервалов

                while ((num = Interlocked.Increment(ref step)) < P)

                {

                    double start = A + dx \* num;

                    double finish = start + dx;

                    DefiniteIntegral(start, finish, e, ref results[num], ref ns[num]);

                }

                //  говорим, что поток завершил работу

                countdownEvent.Signal();

                //  ожидаем семафор

                semaphore.WaitOne();

                if (!next) break;

                //  если конец, то выходим

            }

        }

        /// <summary>

        /// Функция вычисления интеграла на промежутке с заданной точностью с возвращаемым значением и степенью n

        /// </summary>

        /// <param name="a"></param>

        /// <param name="b"></param>

        /// <param name="eps"></param>

        /// <param name="S0"></param>

        /// <param name="n"></param>

        static void DefiniteIntegral(double a, double b, double eps, ref double S0, ref int n)

        {

            double dx = (b - a) / n;

            double S = S0;

            double x = 0;

            if (n == 0)

            {

                dx = (b - a) \* 0.5d;

                n = 2;

                S0 = 0; S = 0;

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

                {

                    x = a + i \* dx;

                    S0 += Func(x);

                }

                S = S0 \*= dx;

            }

            do

            {

                S0 = S;

                n = 2 \* n;

                dx = dx \* 0.5d;

                S = 0;

                for (int i = 1; i < n; i += 2)

                {

                    x = a + i \* dx;

                    S += Func(x);

                }

                S = S \* dx + S0 \* 0.5d;

            }

            while (Math.Abs(S - S0) > eps);

            S0 = S;

        }

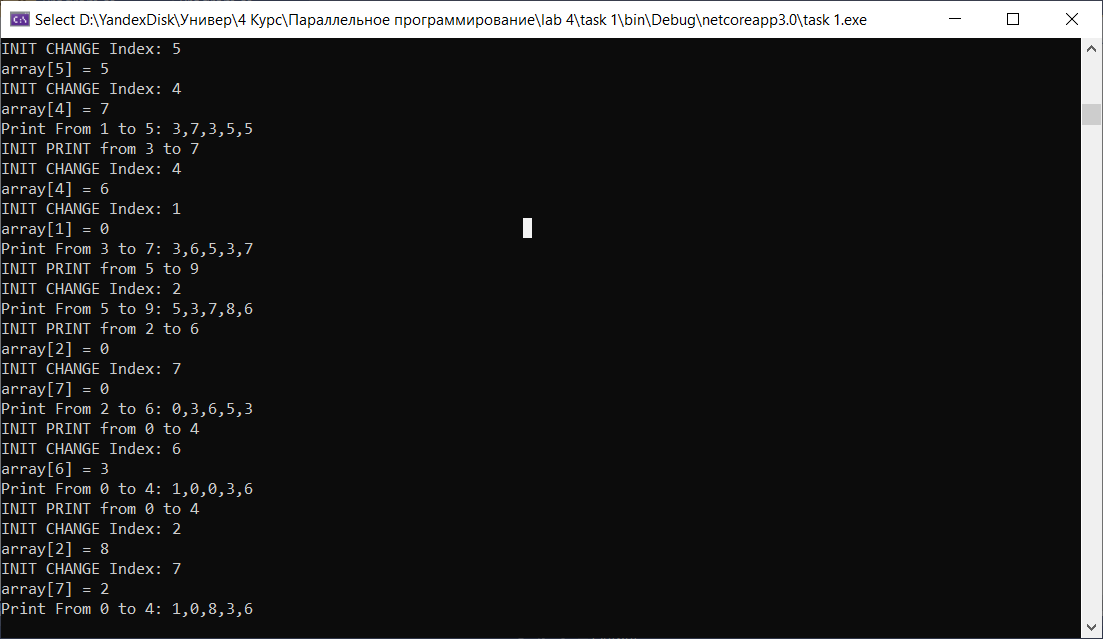
    }

}

# Полученные результаты

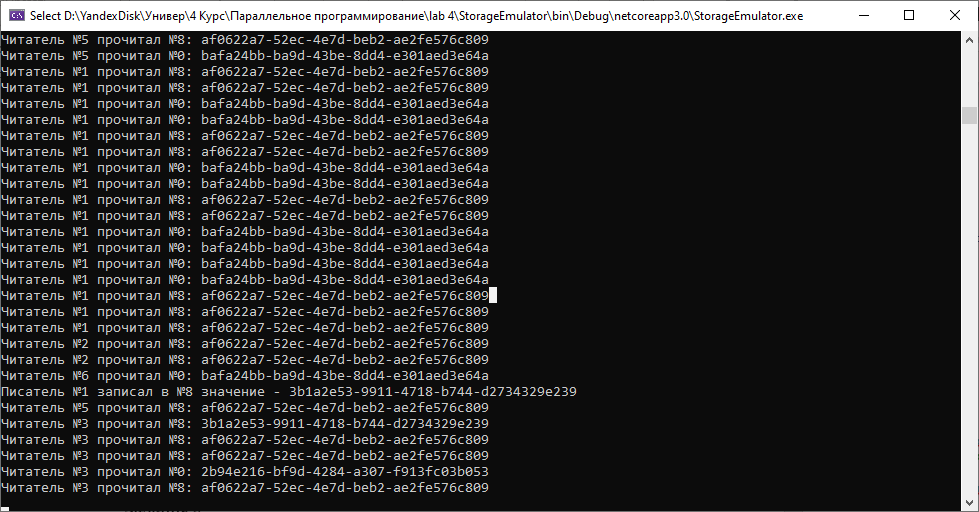
Тестирование проводилось на четырёхъядерном и восьми поточном процессоре Intel Core i7 4770 с частотой 3.7 ГГц, 16 гб ОЗУ в двухканальном режиме с частотой 1600 МГц.

Задание 1



Если индекс не попадает в диапазон, то выводится диапазон и изменяется значение диапазона. Если индекс попадает в диапазон, то сначала изменяется значение, а затем выводится диапазон.

Задание 2



Задание 3

Результаты тестирования алгоритма приведены в таблице 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Точность (ε)** | **Число потоков** | | | |
| **1** | **2** | **4** | **8** |
| **10–7** | 0,011 | 0,008 | 0,006 | 0,005 |
| **10–8** | 0,063 | 0,036 | 0,020 | 0,018 |
| **10–9** | 0,471 | 0,248 | 0,132 | 0,119 |
| Таблица 1. Время работы алгоритма. | | | | |

Ускорение приведено в таблице 2.

Ниже представлены графики ускорения и эффективности:

Как видно из таблиц и графиков, на 8 потоках для ускорение получилось почти в 4 раза. При этом эффективность составляет примерно 0,5. Наибольшая эффективность наблюдается на двух потоках для .

# Выводы по лабораторной работе

Для синхронизации в c# могут использоваться различные способы, методы и классы: mutex, lock, semaphore, CountdownEvent, ReaderWriterLockSlim и т.д. Правильная синхронизация позволяет сократить время простоя и ожидания, а также максимально эффективно использовать вычислительные ресурсы системы.