**14 МНОГОПОТОЧНЫЕ ПРИЛОЖЕНИЯ**

Задание №1. Создать консольное приложение, в котором реализовано 3 потока. Первый выводит числа от 0 до 9, второй – от 10 до 19, третий – от 20 до 29. Вывод значений на экран происходит следующим образом: сначала своё число выводит первый поток, а затем второй и третий. Организовать возможность использования методов Start и Sleep, а также изменение свойства Priority, позволяющее изменить приоритет потока.

Листинг программы:

class Pogram

{

static void mythread1()

{

for (int i = 0; i < 10; i++)

{

Console.WriteLine("Поток 1 выводит " + i);

}

}

static void mythread2()

{

for (int i = 10; i < 20; i++)

{

Console.WriteLine("Поток 2 выводит " + i);

}

}

static void mythread3()

{

for (int i = 20; i < 30; i++)

{

Console.WriteLine("Поток 3 выводит " + i);

}

}

static void Main(string[] args)

{

Thread thread1 = new Thread(mythread1);

Thread thread2 = new Thread(mythread2);

Thread thread3 = new Thread(mythread3);

thread1.Priority = ThreadPriority.Highest;

thread2.Priority = ThreadPriority.AboveNormal;

thread3.Priority = ThreadPriority.Lowest;

thread1.Start();

thread2.Start();

Thread.Sleep(1000);

thread3.Start();

Console.ReadLine();

}

}

Таблица 14.1 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
|  | 0 – 9, 10 – 19, 20 – 29 |

Источник: собственная разработка

Анализ результатов представлен на рисунке 14.1.

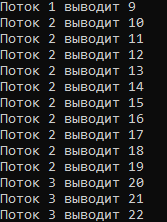


Рисунок 14.1 – Результат работы программы

Источник: собственная разработка

Задание №2. Создать консольное приложение, в котором несколько потоков будут выполнять один и тот же метод. (Количество потоков: 2, Метод: Сумма чисел от 1 до 10). Произвести расчет затраченного времени в миллисекундах на выполнение потока и вывести его на экран.

Листинг программы:

using System.Threading.Tasks;

using System.Diagnostics;

static void method()

{

    int sum = 0;

    for (int i = 1; i <= 10; i++)

    {

        sum += i;

        Console.WriteLine($"Добавление к переменной sum значения '{i}'");

    }

    Console.WriteLine($"Результат: {sum}");

}

Stopwatch sw = new Stopwatch();

Stopwatch sw2 = new Stopwatch();

Task task1 = new Task(method);

Task task2 = new Task(method);

sw.Start();

task1.Start();

task1.Wait();

sw.Stop();

sw2.Start();

task2.Start();

task2.Wait();

sw2.Stop();

TimeSpan ts = sw.Elapsed;

TimeSpan ts2 = sw2.Elapsed;

string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}", ts.Hours, ts.Minutes, ts.Seconds, ts.Milliseconds / 10);

Console.WriteLine("1. Программа завершилась за " + elapsedTime);

string elapsedTime2 = String.Format("{0:00}:{1:00}:{2:00}.{3:00}", ts2.Hours, ts2.Minutes, ts2.Seconds, ts2.Milliseconds / 10);

Console.WriteLine("2. Программа завершилась за " + elapsedTime);

Console.ReadLine();

Таблица 14.2 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
|  | 1 – 10, 55, 00:00:00.15 |

Источник: собственная разработка

Анализ результатов представлен на рисунке 14.2.

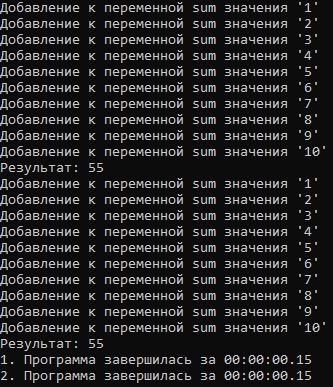


Рисунок 14.2 – Результат работы программы

Источник: собственная разработка

Задание №3. Создать консольное приложение, в котором будут два метода, один метод будет выполняться двумя потоками одновременно, а другой метод в каждый момент времени будет выполняться одним потоком. (Количество потоков: 2, Метод: 1. A+A 1 +A 2 +A 3 +...+A N, А и N вводятся с клавиатуры; 2. A\*A 1 \*A 2 \*A 3 \*…\*A N, А и N вводятся с клавиатуры).

Листинг программы:

try

{

Console.Write("Введите A: ");

int a = Convert.ToInt32(Console.ReadLine());

Console.Write("Введите N: ");

int n = Convert.ToInt32(Console.ReadLine());

var task1 = Task.Run(() => FirstMethod(a, n));

var task2 = Task.Run(() =>

{

FirstMethod(a, n);

SecondMethod(a, n);

});

task1.Wait();

task2.Wait();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

void FirstMethod(int a, int n)

{

int res = a;

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

{

res += (int)Math.Pow(a, i);

}

Console.WriteLine(res);

}

void SecondMethod(int a, int n)

{

int res = a;

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

{

res \*= (int)Math.Pow(a, i);

}

Console.WriteLine(res);

}

Таблица 14.3 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| 19, 23 | 943531298, 943531298, 0 |

Источник: собственная разработка

Анализ результатов представлен на рисунке 14.3.

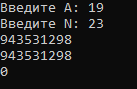


Рисунок 14.3 – Результат работы программы

Источник: собственная разработка

Задание №4. Дана последовательность натуральных чисел . Создать многопоточное приложение для поиска суммы , где – четные числа.

Примерный алгоритм реализации:

В качестве примера рассмотрим случай, когда в массиве хранятся номера элементов. Основной поток создает нужное число потоков, число которых задается параметром, каждому из которых передается параметр – номер процесса (или номер части массива, обрабатываемой потоком). Функция потока возвращает частичную сумму. Основной поток ожидает завершения всех потоков и находит общую сумму.

Листинг программы:

class Program

{

public static int b = 10, sum = 0, CurrentThread = 0;

public static int[] c;

static void Main(string[] args)

{

int n = 1000;

int threads = 10;

c = new int[n];

Random r = new Random();

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

{

c[i] = r.Next(20);

}

Thread[] threadsArray = new Thread[threads];

for (int i = 0; i < threads - 1; i++)

{

threadsArray[i] = new Thread(new ThreadStart(delegate () {

for (int j = n / threads \* i; j < n / threads \* (i + 1); j++)

{

if (c[j] % 2 == 0)

{

sum += c[j];

}

}

}));

CurrentThread++;

threadsArray[i].Start();

threadsArray[i].Join(); }

Console.WriteLine("count:{0}", sum);

Console.ReadKey();

}

}

Таблица 14.4 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| 10, 0, 0, 10 | 3746 |

Источник: собственная разработка

Анализ результатов представлен на рисунке 14.4.



Рисунок 14.4 – Результат работы программы

Источник: собственная разработка