**ПРАКТИЧЕСКОЕ ЗАДАНИЕ 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

static void Main(string[] args)

{

Thread Thread1 = new Thread(First\_Thread);

Thread Thread2 = new Thread(Second\_Thread);

Thread Thread3 = new Thread(Third\_Thread);

Thread1.Priority = ThreadPriority.Highest;

Thread2.Priority = ThreadPriority.Normal;

Thread3.Priority = ThreadPriority.Lowest;

Thread1.Start();

Thread1.Join();

Thread2.Start();

Thread2.Join();

Thread3.Start();

}

static void First\_Thread()

{

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

{

Thread.Sleep(20);

Console.WriteLine("Вывод первого потока: " +i);

}

}

static void Second\_Thread()

{

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

{

Thread.Sleep(100);

Console.WriteLine("Вывод второго потока: "+ i);

}

}

static void Third\_Thread()

{

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

{

Thread.Sleep(90);

Console.WriteLine("Вывод третьего потока: " +i);

}

}

}

Таблица 1.1 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| Числа от 1 до 30 | Вывод чисел от 1 до 30 с помощью потоков |

Анализ результатов:

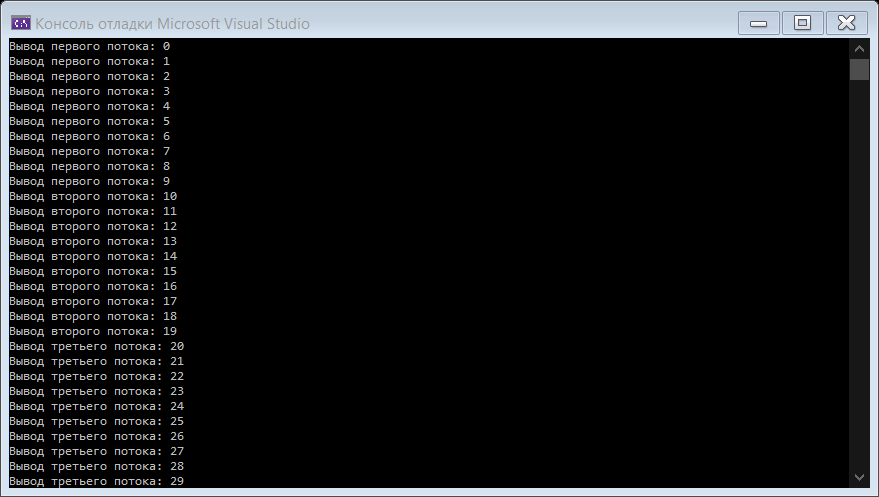


Рисунок 1.1 – Результат работы программы

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:

static void Sum()

{

int sum = 0;

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

{ sum += i; }

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

}

static void Main(string[] args)

{

System.Diagnostics.Stopwatch swatch = new System.Diagnostics.Stopwatch();

swatch.Start();

ThreadStart thread1 = new ThreadStart(Sum);

Thread thread2 = new Thread(thread1);

thread2.Start();

swatch.Stop();

Console.ForegroundColor = ConsoleColor.Blue;

Console.ForegroundColor = ConsoleColor.White;

Console.WriteLine($"Время, потраченное на выполнение задачи: {0}", swatch.Elapsed);

Console.ForegroundColor = ConsoleColor.Green;

}

}

Таблица 1.2 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
|  | Время, потраченное на выполнение задачи: 00:00:00.017153  Сумма чисел:55 |

Анализ результатов:



Рисунок 1.2 – Результат работы программы

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+A1+A2+A3+..+AN , А и N вводятся с клавиатуры; 2. A\*A1\*A2\*A3\*…\*AN , А и N вводятся с клавиатуры ).

Листинг программы:

static void First(int A)

{

double sum = 0;

for (int i = 1; i < 10; i++)

{

sum = sum + Math.Pow(A, i);

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

}

}

static void Second(double A)

{

double umn = 1;

for (int j = 1; j < 10; j++)

{

umn = umn \* Math.Pow(A, j);

Console.WriteLine($" {umn} ");

}

}

static void Main(string[] args)

{

int A;

Console.WriteLine("Input A: ");

A = Convert.ToInt32(Console.ReadLine());

int N;

Console.WriteLine("Input N: ");

N = Convert.ToInt32(Console.ReadLine());

Thread thread1 = new Thread(() => First(A));

Thread thread2 = new Thread(() => Second(N));

Stopwatch time = new Stopwatch();

thread1.Start();

thread2.Start();

Console.ReadLine();

}

}

Таблица 1.3 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| A =1  N = 2 | 1  2  3  8  64  1024  4  5  6  7  8  32768  2097152  9 |

Анализ результатов:

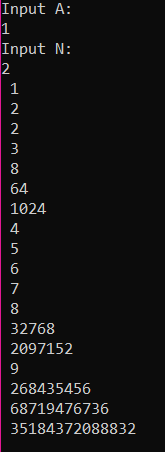


Рисунок 1.3 – Результат работы программы

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 | 4354 |

Источник: собственная разработка

Анализ результатов представлен на рисунке 1.4.



Рисунок 1.4 – Результат работы программы

Источник: собственная разработка