**14 «СОБЫТИЯ».**

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. Можно воспользоваться инструментами синхронизации потоков lock, AutoResetEvent и т.д.

Листинг программы:

using System;

using System.Threading;

class Program

{

static void Main(string[] args)

{

Thread thread1 = new Thread(CountNumbers);

Thread thread2 = new Thread(CountNumbers);

Thread thread3 = new Thread(CountNumbers);

thread1.Name = "Thread 1";

thread2.Name = "Thread 2";

thread3.Name = "Thread 3";

thread1.Start(0);

thread2.Start(10);

thread3.Start(20);

}

static void CountNumbers(object start)

{

int num = (int)start;

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

{

lock (typeof(Program))

{

Console.WriteLine(Thread.CurrentThread.Name + ": " + num);

num++;

}

Thread.Sleep(100);

}

}

}

Таблица 1.1 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
|  | Thread 1: 0  Thread 2: 10  Thread 3: 20  Thread 3: 21  Thread 1: 1  Thread 2: 11  Thread 3: 22  Thread 1: 2  Thread 2: 12  Thread 1: 3  Thread 2: 13  Thread 3: 23  Thread 2: 14  Thread 3: 24  Thread 1: 4  Thread 1: 5  Thread 3: 25  Thread 2: 15  Thread 2: 16  Thread 3: 26  Thread 1: 6  Thread 3: 27  Thread 2: 17  Thread 1: 7  Thread 3: 28  Thread 2: 18  Thread 1: 8  Thread 2: 19  Thread 1: 9  Thread 3: 29 |

Анализ результатов:

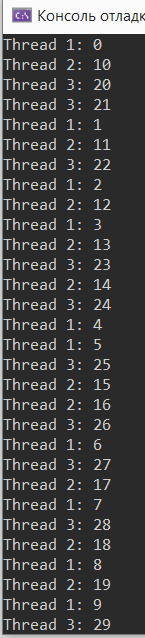


Рисунок 1.1 – Результат работы программы

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:

using System;

using System.Threading;

class Program

{

static void Main(string[] args)

{

Thread thread1 = new Thread(CalculateSum);

Thread thread2 = new Thread(CalculateSum);

DateTime startTime = DateTime.Now;

thread1.Start();

thread2.Start();

thread1.Join();

thread2.Join();

DateTime endTime = DateTime.Now;

TimeSpan duration = endTime - startTime;

Console.WriteLine($"Время выполнения первого потока: {duration.TotalMilliseconds} мс");

Console.ReadLine();

}

static void CalculateSum()

{

int sum = 0;

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

{

sum += i;

Thread.Sleep(100);

}

Console.WriteLine($"Сумма чисел от 1 до 10: {sum}");

}

}

Таблица 1.3 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
|  | Сумма чисел от 1 до 10: 55  Сумма чисел от 1 до 10: 55  Время выполнения первого потока: 1150,4937 мс |

Анализ результатов:

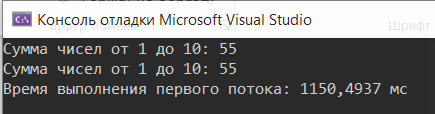


Рисунок 1.2 – Результат работы программы

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+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:

using System;

using System.Threading;

class Program

{

static void Main(string[] args)

{

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

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

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

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

Thread thread1 = new Thread(() => SumMethod(A, N));

Thread thread2 = new Thread(() => MultiplyMethod(A, N));

thread1.Start();

thread2.Start();

thread1.Join();

thread2.Join();

Console.ReadLine();

}

static void SumMethod(int A, int N)

{

long result = 0;

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

{

result += A \* i;

Console.WriteLine($"Поток 1: Промежуточный результат на итерации {i}: {result}");

Thread.Sleep(100);

}

Console.WriteLine($"Поток 1: Итоговый результат: {result}");

}

static void MultiplyMethod(int A, int N)

{

long result = 1;

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

{

result \*= A \* i;

Console.WriteLine($"Поток 2: Промежуточный результат на итерации {i}: {result}");

Thread.Sleep(100);

}

Console.WriteLine($"Поток 2: Итоговый результат: {result}");

}

}

Таблица 1.3 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| 4, 5 | Введите число A:  4  Введите число N:  5  Поток 2: Промежуточный результат на итерации 1: 4  Поток 1: Промежуточный результат на итерации 1: 4  Поток 1: Промежуточный результат на итерации 2: 12  Поток 2: Промежуточный результат на итерации 2: 32  Поток 2: Промежуточный результат на итерации 3: 384  Поток 1: Промежуточный результат на итерации 3: 24  Поток 1: Промежуточный результат на итерации 4: 40  Поток 2: Промежуточный результат на итерации 4: 6144  Поток 2: Промежуточный результат на итерации 5: 122880  Поток 1: Промежуточный результат на итерации 5: 60  Поток 1: Итоговый результат: 60  Поток 2: Итоговый результат: 122880 |

Анализ результатов:

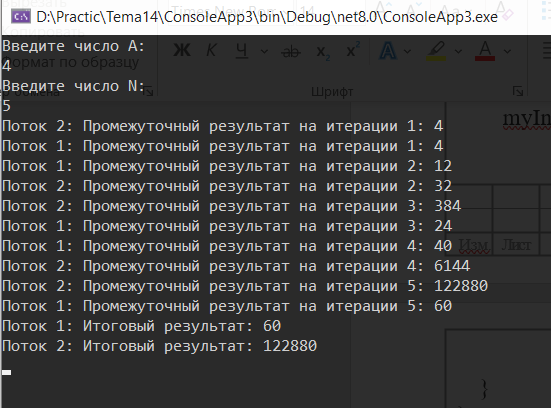


Рисунок 1.3 – Результат работы программы

Задание 4. Дана последовательность натуральных чисел {a 0 …a n–1 }. Создать многопоточное приложение для поиска суммы ∑a i , где a i – 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:

using System;

using System.Threading;

class Program

{

static int[] array = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

static void Main(string[] args)

{

int numThreads = 4;

Thread[] threads = new Thread[numThreads];

SumThreadResult[] threadResults = new SumThreadResult[numThreads];

int chunkSize = array.Length / numThreads;

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

{

int start = i \* chunkSize;

int end = (i == numThreads - 1) ? array.Length : (i + 1) \* chunkSize;

SumThreadResult threadResult = new SumThreadResult();

threadResults[i] = threadResult;

threads[i] = new Thread(() =>

{

threadResult.Sum = PartialSum(start, end);

});

threads[i].Start();

}

foreach (Thread thread in threads)

{

thread.Join();

}

int totalSum = 0;

foreach (var result in threadResults)

{

totalSum += result.Sum;

}

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

Console.ReadLine();

}

static int PartialSum(int start, int end)

{

int sum = 0;

for (int i = start; i < end; i++)

{

if (array[i] % 2 == 0)

{

sum += array[i];

}

}

return sum;

}

}

class SumThreadResult

{

public int Sum { get; set; }

}

Таблица 1.4 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
|  | Сумма четных чисел: 30 |

Анализ результатов:

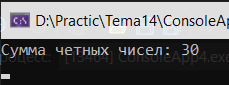


Рисунок 1.4 – Результат работы программы