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

Листинг программы:

Thread t1 = new Thread(() => WriteNumFromTo("Первый поток", 1, 9));

Thread t2 = new Thread(() => WriteNumFromTo("Второй поток", 10, 19));

Thread t3 = new Thread(() => WriteNumFromTo("Третий поток", 20, 29));

t1.Priority = ThreadPriority.Highest; t2.Priority = ThreadPriority.BelowNormal;

t3.Priority = ThreadPriority.Lowest;

t1.Start(); Thread.Sleep(5000); t2.Start(); Thread.Sleep(5000); t3.Start();

static void WriteNumFromTo(string mes, int from, int to)

{ for (int i = from; i <= to; i++){Console.WriteLine($"{mes}: {i}");}}

Таблица 14.1 – Выходные и входные данные

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

Источник: собственная разработка

Анализ результатов:

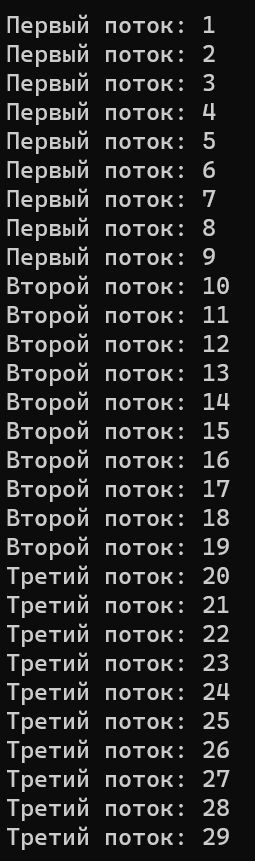


Рисунок 14.1 – Результат работы программы

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:

static void Main(string[] args)

{Thread t1 = new Thread(SumNumbers);Thread t2 = new Thread(SumNumbers);

Stopwatch stopwatch = new Stopwatch(); Stopwatch.Start();

t1.Start(); t2.Start(); t1.Join(); t2.Join();} static void SumNumbers()

{ int sum = 0; for (int i = 1; i <= 10; i++) {sum += i; Thread.Sleep(100);}

Console.WriteLine($"Поток {Thread.CurrentThread.ManagedThreadId}: " +

$"сумма чисел от 1 до 10 равна {sum}. " +

$"Время выполнения потока: {Stopwatch.ElapsedMilliseconds} мс."); }

static readonly Stopwatch Stopwatch = new Stopwatch();}

Таблица 14.2 – Выходные и входные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
|  | Поток 10: сумма чисел от 1 до 10 равна 55. Время выполнения потока: 1093 мс.  Поток 9: сумма чисел от 1 до 10 равна 55. Время выполнения потока: 1093 мс. |

Источник: собственная разработка

Анализ результатов:

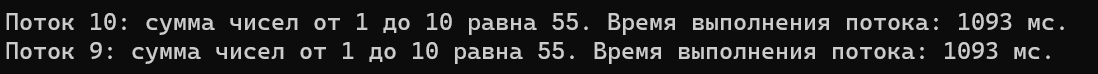


Рисунок 14.2 – Результат работы программы

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+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 Sum(int A, int N){int sum = 0;for (int i = 0; i <= N; i++)

{sum += (int)Math.Pow(A, i); Console.WriteLine($"Сумма: {sum}");

Thread.Sleep(500); }}

static void Product(int A, int N){int product = 1; for (int i = 1; i <= N; i++){

product \*= (int)Math.Pow(A, i); Console.WriteLine($"Произведение: {product}"); Thread.Sleep(500); }}

static void Main(string[] args){ Console.Write("Введите A:");

int A = int.Parse(Console.ReadLine()); Console.Write("Введите N:");

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

Thread t1 = new Thread(() => Sum(A, N));

Thread t2 = new Thread(() => Product(A, N));

t1.Start(); t2.Start(); t1.Join(); t2.Join();

Console.WriteLine("Работа потоков завершена.");}

Таблица 14.3 – Выходные и входные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| 2; 8 | Сумма: 1  Произведение: 2  Сумма: 3  Произведение: 8  Произведение: 64  Сумма: 7  Сумма: 15  Произведение: 1024  Сумма: 31  Произведение: 32768  Произведение: 2097152  Сумма: 63  Произведение: 268435456  Сумма: 127  Произведение: 0  Сумма: 255  Сумма: 511  Работа потоков завершена. |

Источник: собственная разработка

Анализ результатов:

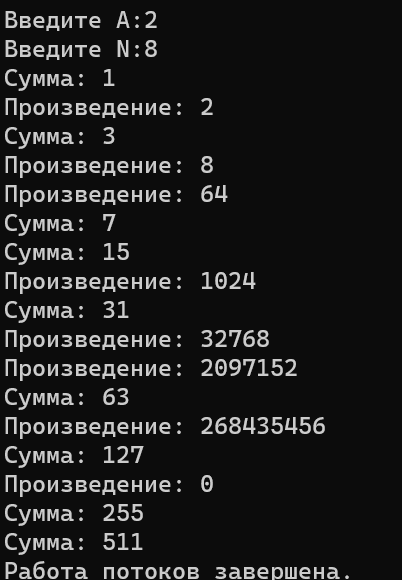


Рисунок 14.3 – Результат работы программы

Источник: собственная разработка

Задание 4. Дана последовательность натуральных чисел {a₀ …an-₁}. Создать многопоточное приложение для поиска суммы ∑ai, где ai – 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:

Console.Write("Введите последнее число последовательности: ");

int num = Convert.ToInt32(Console.ReadLine()); int[] arr = new int[num];

for (int i = 0; i < arr.Length; i++){ arr[i] = i + 1;}

int countThread = GetCountThread(num);

Thread[] threads = new Thread[countThread];

int[] arrResultThreads = new int[countThread];

int startWith = 0; int sizePart = 5;

for (int i = 0; i < countThread; i++){if (sizePart + startWith > arr.Length){

sizePart = arr.Length - startWith;}

int[] partArr = new int[sizePart];

Array.Copy(arr, startWith, partArr, 0, partArr.Length);

startWith += sizePart; int numThread = i;

threads[i] = new Thread(() =>{arrResultThreads[numThread] = SumEvenElemInArr(partArr);});

threads[i].Start();}

foreach (Thread thread in threads){thread.Join();}

Console.WriteLine("Сумма чётных элементов: "+ arrResultThreads.Sum());

int SumEvenElemInArr(int[] arr){ int sum = 0;

for (int i = 0; i < arr.Length; i++){if (arr[i] % 2 == 0){sum += arr[i];}}return sum;}

int GetCountThread(int num) => num % 5 != 0 ? Convert.ToInt32(num / 5) +1:num / 5;

Таблица 14.4 – Выходные и входные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| 10 | Сумма чётных элементов: 30 |

Источник: собственная разработка

Анализ результатов:

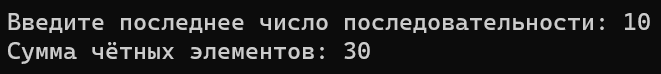


Рисунок 14.4 – Результат работы программы

Источник: собственная разработка