**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 FirstThgread()

{

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

{

Console.WriteLine($"First thread display: {i}");

}

}

static void SecondThread()

{

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

{

Console.WriteLine($"Second thread display: {i}");

}

}

static void ThirdThread()

{

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

{

Console.WriteLine($"Third thread display: {i}");

}

}

static void Main(string[] args)

{

Thread fThread = new Thread(FirstThgread);

Thread sThread = new Thread(SecondThread);

Thread tThread = new Thread(ThirdThread);

fThread.Priority = ThreadPriority.Highest;

sThread.Priority = ThreadPriority.AboveNormal;

tThread.Priority = ThreadPriority.Lowest;

fThread.Start();

Thread.Sleep(500);

sThread.Start();

Thread.Sleep(1000);

tThread.Start();

}

}

Таблица 14.1 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
|  | 1 – 9, 10 – 19, 20 – 29 |

Источник: собственная разработка

Анализ результатов представлен на рисунке 14.1.

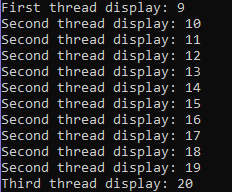


Рисунок 14.1 – Результат работы программы

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:

static void Adding()

{

int sum = 0;

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

{

sum += i;

Console.WriteLine($"Adding {i} to sum");

}

Console.WriteLine($"Result of adding = {sum}");

}

Stopwatch stop1 = new Stopwatch();

Stopwatch stop2 = new Stopwatch();

Task fTask = new Task(Adding);

Task sTask = new Task(Adding);

stop1.Start();

fTask.Start();

fTask.Wait();

stop1.Stop();

stop2.Start();

sTask.Start();

sTask.Wait();

stop2.Stop();

TimeSpan timespanF = stop1.Elapsed;

TimeSpan timespanS = stop2.Elapsed;

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

Console.WriteLine("1) Elapsed time: " + elapsedTime);

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

Console.WriteLine("2) Elapsed time: " + elapsedTime);

Console.ReadLine();

Таблица 14.2 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
|  | 1 – 10, 55, 00:00:00.21 |

Источник: собственная разработка

Анализ результатов представлен на рисунке 14.2.

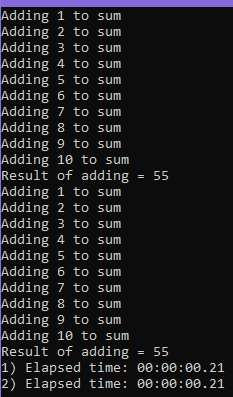


Рисунок 14.2 – Результат работы программы

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("Enter A value: ");

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

Console.Write("Enter N value: ");

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

var fTask = Task.Run(() => Method1(a, n));

var sTask = Task.Run(() =>

{

Method1(a, n);

Method2(a, n);

});

fTask.Wait();

sTask.Wait();

}

catch (Exception e)

{

Console.WriteLine($"An error occured! {e.Message}");

}

void Method1(int a, int n)

{

int sum = a;

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

{

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

}

Console.WriteLine(sum);

}

void Method2(int a, int n)

{

int sum = a;

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

{

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

}

Console.WriteLine(sum);

}

Таблица 14.3 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| 3, 3 | 42, 42, 2187 |

Источник: собственная разработка

Анализ результатов представлен на рисунке 14.3.

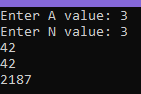


Рисунок 14.3 – Результат работы программы

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 E = 0, ThdreadInUse = 0;

public static int[] c;

static void Main(string[] args)

{

int n = 1000;

int threads = 10;

c = new int[n];

Random rand = new Random();

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

{

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

}

Thread[] ArrayOfThdreads = new Thread[threads];

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

{

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

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

{

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

{

E += c[j];

}

}

}));

ThdreadInUse++;

ArrayOfThdreads[i].Start();

ArrayOfThdreads[i].Join();

}

Console.WriteLine($"Sum - {E}");

Console.ReadKey();

}

}

Таблица 14.4 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| 0, 0, 10 | 3990 |

Источник: собственная разработка

Анализ результатов представлен на рисунке 14.4.



Рисунок 14.4 – Результат работы программы

Источник: собственная разработка