Отчет по лабораторной работе № 2

«Синхронизация потоков при помощи семафоров и критических секций»

Выполнил: студент группы P3317

Плюхин Д.А.

Преподаватель: Лаздин Артур Вячеславович

# **Задание**

Написать программу для консольного процесса, который состоит из трёх потоков:

Поток **main** должен выполнить следующие действия:

создать массив, размерность и элементы которого вводятся пользователем с консоли; вывести размерность и элементы исходного массива на консоль;

ввести число k;

запустить поток **work**;

запустить поток **SumElement**;

освобождение выходной поток stdout после вывода на консоль каждого нового элемента массива.

выводить на экран поэлементно элементы массива (итогового) параллельно с работой потока **work**;

известить поток **SumElement** о начале суммирования (момент запуска произойдёт после того,

будут готовы к печати k - элементов массива).

Поток **work** должен выполнить следующие действия *(Для синхронизации с потоком* ***main*** *–* *использовать*

***семафор. Проверить работу программы используя критическую секцию для*** *синхронизации с потоком* ***main, объяснить отличия, если есть!****)*:

запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;

Поиск в массиве простых чисел (разместить их в массиве слева, остальные элементы массива - справа).

Элементы - целые числа без знака.

извещать поток main о новом элементе;

после каждого готового элемента отдыхать в течение заданного интервала времени;

Поток **SumElement** должен выполнить следующие действия (*Для синхронизации с потоком* *main****,*** *использовать* *бинарный семафор!*):

ждёт от потока **main** сигнал о начале суммирования;

выполнить суммирование элементов итогового массива до заданной позиции k; вывести итоговую сумму.

# **Исходный код**

using System;

using System.Threading;

namespace FirstLab

{

class Threads

{

static void Main()

{

Thread mainThread = new Thread(main);

mainThread.Start();

}

static void main()

{

Semaphore main\_work\_semaphore;

Semaphore main\_sum\_element\_semaphore;

main\_work\_semaphore = new Semaphore(0, 1);

main\_sum\_element\_semaphore = new Semaphore(0, 1);

int arraySize = getIntFromUser("Please, type size of array:");

uint[] array = new uint[arraySize];

for (int i = 0; i < arraySize; i++){

array[i] = getUintFromUser("Please, type "+(i+1)+" element of array:");

}

showArray(array);

int k = getIntFromUser("Please, type k:");

//Create threads

Thread workThread = new Thread(() => work(ref array, k, ref main\_work\_semaphore));

Thread sumElementThread = new Thread(() => SumElement(ref array, k, ref main\_sum\_element\_semaphore));

//Run threads

workThread.Start();

sumElementThread.Start();

int current\_index = 0;

while (current\_index < k){

main\_work\_semaphore.WaitOne();

Console.WriteLine("main: "+(current\_index+1)+" element is known: "+array[current\_index]);

current\_index++;

}

Console.WriteLine("main: "+k+" elements are known, so signaling to the SumElement...");

main\_sum\_element\_semaphore.Release();

}

static void work(ref uint[] array, int k, ref Semaphore main\_work\_semaphore)

{

int rest\_interval = getIntFromUser("Please, type rest interval (milliseconds):");

for (int i = 0; i < array.Length; i++){

int selected\_element = i;

for (int j = i; j < array.Length; j++){

if (isSimple(array[j])){

selected\_element = j;

break;

}

}

uint tmp = array[i];

array[i] = array[selected\_element];

array[selected\_element] = tmp;

Console.WriteLine("work: "+(i+1)+" element is ready!");

if (i < k) main\_work\_semaphore.Release();

Thread.Sleep(rest\_interval);

}

}

static void SumElement(ref uint[] array, int k, ref Semaphore main\_sum\_element\_semaphore)

{

main\_sum\_element\_semaphore.WaitOne();

Console.WriteLine("SumElement: Sum of first "+k+" elements of array is "+countSum(array,k));

}

static void showArray(uint[] array){

Console.Write("Given array with size "+array.Length+" and elements ");

for (int i = 0; i < array.Length; i++){

if (i != 0) Console.Write(", ");

Console.Write(array[i]);

}

Console.WriteLine();

}

static uint countSum(uint[] array, int k){

uint sum = 0;

for (int i = 0; i < k; i++){

sum += array[i];

}

return sum;

}

static bool isSimple(uint number){

if (number < 2) return true;

for (uint i = 2; i < (uint)(Math.Floor(Math.Sqrt(number)) + 1); i++){

if (0 == number % i) return false;

}

return true;

}

static uint getUintFromUser(string msg){

Console.WriteLine(msg);

return Convert.ToUInt32(Console.ReadLine());

}

static int getIntFromUser(string msg){

Console.WriteLine(msg);

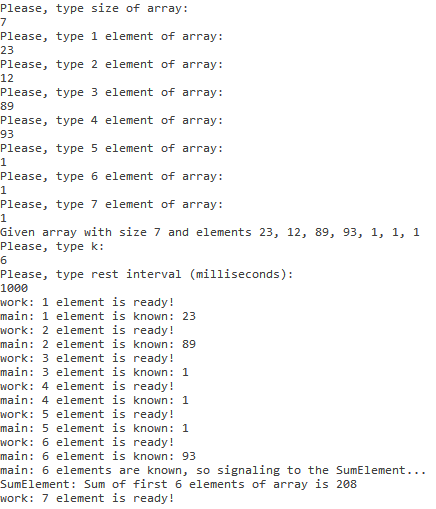
return Convert.ToInt32(Console.ReadLine());

}

}

}

# **Результат**



# **Вывод**

Таким образом, взаимодействие потоков достаточно просто реализуется при помощи таких объектов синхронизации, как семафоры, и, в частности, бинарные семафоры, которые, по сути, ничем не отличаются от критических секций, и именно по этой причине в данном случае реализация взаимодействия потоков work и main не принципиальна и может быть выполнена равноценно как с помощью семафора, так и с помощью критической секции, поскольку оба варианта синхронизации предусматривают использование двоичной блокирующей переменной, операции изменения которой атомарны.