Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №2**

З дисципліни «Паралельне програмування»

**Потоки в мові C#**

ВИКОНАВ:

Студент ІІІ курсу ФІОТ

Групи ІО-91

Діденко В.В.

ПЕРЕВІРИВ:

Корочкін О.В.

Київ 2021 р.

**Мета роботи:** вивчення засобів мови C# для роботи з потоками.

**Виконання роботи:** Розробити програму, яка містить паралельні потоки, що реалізують відповідну функцію F1, F2, F3 з Додатку Б згідно отриманому варіанту. Вимоги що до створення потоків і завдання дослідження особливості виконання паралельної програми визначені в лабораторної роботі 1.

**Необхідні теоретичні відомості:** мова C# забезпечує можливість програмування паралельних процесів за допомогою потоків класу Thread з використанням концепції потокових функцій.

**Варіант завдання:**

F1: A = B + C + D\*(MD\*ME)

F2: MG = SORT(MF - MH \* MK)

F3: S = SORT(O\*MO)\*(MS \*MT)

**Виконання роботи**

**Лістинг програми:**

**Lab2.cs**

**using System;**

**using System.Threading;**

**/\***

**\* Parallel programming.**

**\* Labwork 2. C#.**

**\***

**\* F1 1.12 A = B + C + D \* (MD \* ME)**

**\* F2 2.24 MG = sort(MF - MH \* MK)**

**\* F3 3.21 S = sort(O \* MO) \* (MS \* MT)**

**\***

**\* Didenko Vladyslav**

**\* IO-91**

**\* 16.09.2021**

**\*/**

**namespace Lab2 {**

**class Lab2 {**

**public const int N = 2;**

**public static void Main(string[] args) {**

**Console.WriteLine("Lab 3 start\n");**

**// var T1 = new Thread(new Function1(Function.FillType.Random).Run);**

**// var T2 = new Thread(new Function2(Function.FillType.Random).Run);**

**// var T3 = new Thread(new Function3(Function.FillType.Random).Run);**

**var T1 = new Thread(new Function1(1).Run);**

**var T2 = new Thread(new Function2(1).Run);**

**var T3 = new Thread(new Function3(1).Run);**

**T1.Priority = ThreadPriority.Lowest;**

**T2.Priority = ThreadPriority.Highest;**

**T3.Priority = ThreadPriority.Normal;**

**T1.Start();**

**//T1.Join();**

**T2.Start();**

**//T2.Join();**

**T3.Start();**

**//T3.Join();**

**// Thread.Sleep(100);**

**Console.WriteLine("Lab 3 finished \n");**

**}**

**}**

**public static class DefaultOperations {**

**public static int Sum(int a, int b) => a + b;**

**public static int Difference(int a, int b) => a - b;**

**public static int Multiply(int a, int b) => a \* b;**

**}**

**public class WrongInputException : Exception {**

**public override string ToString() {**

**return $"{base.ToString()}\nInput must be int";**

**}**

**}**

**}**

**F1.cs**

**using System;**

**namespace Lab2 {**

**public class Function1 : Function {**

**Vector b = new Vector(Lab2.N);**

**Vector c = new Vector(Lab2.N);**

**Vector d = new Vector(Lab2.N);**

**Matrix md = new Matrix(Lab2.N, Lab2.N);**

**Matrix me = new Matrix(Lab2.N, Lab2.N);**

**public Function1(FillType type) : base(type) { }**

**public Function1(int num) : base(num) { }**

**public override void FillWithRandom() {**

**b.FillWithRandom();**

**c.FillWithRandom();**

**d.FillWithRandom();**

**md.FillWithRandom();**

**me.FillWithRandom();**

**}**

**public override void FillWithNumber(int num = 1) {**

**b.FillWithNumber(num);**

**c.FillWithNumber(num);**

**d.FillWithNumber(num);**

**md.FillWithNumber(num);**

**me.FillWithNumber(num);**

**}**

**public override void FillWithInput() {**

**Console.WriteLine("Enter B:");**

**b.FillWithInput();**

**Console.WriteLine("Enter C:");**

**c.FillWithInput();**

**Console.WriteLine("Enter D:");**

**d.FillWithInput();**

**Console.WriteLine("Enter MD:");**

**md.FillWithInput();**

**Console.WriteLine("Enter ME:");**

**me.FillWithInput();**

**}**

**public override void Run() {**

**// A = B + C + D \* (MD \* ME)**

**// lock (Lab2.Locker) {**

**// Thread.Sleep(1000);**

**Console.WriteLine("Function 1 started");**

**var result = b + c + d \* (md \* me);**

**Console.WriteLine("Function 1 result:\n" + result);**

**Console.WriteLine("Function 1 finished");**

**// }**

**}**

**}**

**}**

**F2.cs**

**using System;**

**namespace Lab2 {**

**public class Function2 : Function {**

**Matrix mf = new Matrix(Lab2.N, Lab2.N);**

**Matrix mh = new Matrix(Lab2.N, Lab2.N);**

**Matrix mk = new Matrix(Lab2.N, Lab2.N);**

**public Function2(FillType type) : base(type) { }**

**public Function2(int num) : base(num) { }**

**public override void FillWithRandom() {**

**mf.FillWithRandom();**

**mh.FillWithRandom();**

**mk.FillWithRandom();**

**}**

**public override void FillWithNumber(int num = 2) {**

**mf.FillWithNumber(num);**

**mh.FillWithNumber(num);**

**mk.FillWithNumber(num);**

**}**

**public override void FillWithInput() {**

**Console.WriteLine("Enter MF:");**

**mf.FillWithInput();**

**Console.WriteLine("Enter MH:");**

**mh.FillWithInput();**

**Console.WriteLine("Enter MK:");**

**mk.FillWithInput();**

**}**

**public override void Run() {**

**// MG = Sort(MF - MH \* MK)**

**// lock (Lab2.Locker) {**

**// Thread.Sleep(1000);**

**Console.WriteLine("Function 2 start");**

**var result = (mf - mh \* mk).Sort();**

**Console.WriteLine("Function 2 result:\n" + result);**

**Console.WriteLine("Function 2 finished\n");**

**// }**

**}**

**}**

**}**

**F3.cs**

**using System;**

**namespace Lab2 {**

**public class Function3 : Function {**

**Vector o = new Vector(Lab2.N);**

**Matrix mo = new Matrix(Lab2.N, Lab2.N);**

**Matrix ms = new Matrix(Lab2.N, Lab2.N);**

**Matrix mt = new Matrix(Lab2.N, Lab2.N);**

**public Function3(FillType type) : base(type) { }**

**public Function3(int num) : base(num) { }**

**public override void FillWithRandom() {**

**o.FillWithRandom();**

**mo.FillWithRandom();**

**ms.FillWithRandom();**

**mt.FillWithRandom();**

**}**

**public override void FillWithNumber(int num = 0) {**

**o.FillWithNumber(num);**

**mo.FillWithNumber(num);**

**ms.FillWithNumber(num);**

**mt.FillWithNumber(num);**

**}**

**public override void FillWithInput() {**

**Console.WriteLine("Enter O:");**

**o.FillWithInput();**

**Console.WriteLine("Enter MO:");**

**mo.FillWithInput();**

**Console.WriteLine("Enter MS:");**

**ms.FillWithInput();**

**Console.WriteLine("Enter MT:");**

**mt.FillWithInput();**

**}**

**public override void Run() {**

**// S = Sort(O \* MO) \* (MS \* MT)**

**// lock (Lab2.Locker) {**

**// Thread.Sleep(1000);**

**Console.WriteLine("Function 3 start");**

**var result = (o \* mo).Sort() \* (ms \* mt);**

**Console.WriteLine("Function 3 result:\n" + result);**

**Console.WriteLine("Function 3 finished\n");**

**// }**

**}**

**}**

**}**

**F.cs**

**using System;**

**namespace Lab2 {**

**public abstract class Function {**

**public enum FillType {**

**Random,**

**Number,**

**Input**

**}**

**public Function(FillType type) {**

**switch (type) {**

**case FillType.Random:**

**FillWithRandom();**

**break;**

**case FillType.Number:**

**FillWithNumber();**

**break;**

**case FillType.Input:**

**FillWithInput();**

**break;**

**default:**

**throw new ArgumentOutOfRangeException(nameof(type), type, null);**

**}**

**}**

**public Function(int num) {**

**FillWithNumber(num);**

**}**

**public abstract void FillWithRandom();**

**public abstract void FillWithNumber(int num = 0);**

**public abstract void FillWithInput();**

**public abstract void Run();**

**}**

**}**

**Matrix.cs**

**using System;**

**namespace Lab2 {**

**public class Matrix {**

**private int[,] \_matr;**

**public Matrix(int n, int m) {**

**\_matr = new int[n, m];**

**FillWithNumber(1);**

**}**

**public Matrix(int[,] arr) {**

**\_matr = (int[,]) arr.Clone();**

**}**

**public static Matrix operator \*(Matrix ma, Matrix mb) {**

**if (ma.\_matr.GetLength(1) != mb.\_matr.GetLength(0)) {**

**throw new MatrixLengthException();**

**}**

**var n = ma.\_matr.GetLength(0);**

**var m = mb.\_matr.GetLength(1);**

**var matr = new int[n, m];**

**for (var i = 0; i < n; ++i) {**

**for (var j = 0; j < m; ++j) {**

**matr[i, j] = 0;**

**for (var k = 0; k < n; ++k) {**

**matr[i, j] += ma.\_matr[i, k] \* mb.\_matr[k, j];**

**}**

**}**

**}**

**return new Matrix(matr);**

**}**

**public static Matrix operator \*(Matrix ma, Vector a) {**

**var n = ma.\_matr.GetLength(0);**

**var m = a.GetLength();**

**if (ma.\_matr.GetLength(1) != 1) {**

**throw new MatrixLengthException();**

**}**

**var matr = new int[n, m];**

**for (var i = 0; i < n; i++) {**

**for (var j = 0; j < m; j++) {**

**matr[i, j] += ma.\_matr[i, 0] \* a.GetElem(j);**

**}**

**}**

**return new Matrix(matr);**

**}**

**public static Matrix operator \*(Matrix ma, int a) {**

**var n = ma.\_matr.GetLength(0);**

**var m = ma.\_matr.GetLength(1);**

**var matr = new int[n, m];**

**for (var i = 0; i < n; i++) {**

**for (var j = 0; j < m; j++) {**

**matr[i, j] = a \* ma.\_matr[i, j];**

**}**

**}**

**return new Matrix(matr);**

**}**

**public static Matrix operator +(Matrix ma, Matrix mb) {**

**return new Matrix(MatrixOperation(ma, mb, DefaultOperations.Sum));**

**}**

**public static Matrix operator -(Matrix ma, Matrix mb) {**

**return new Matrix(MatrixOperation(ma, mb, DefaultOperations.Difference));**

**}**

**private static int[,] MatrixOperation(Matrix ma, Matrix mb, Func<int, int, int> operation) {**

**var n = ma.\_matr.GetLength(0);**

**var m = ma.\_matr.GetLength(1);**

**if (n != mb.\_matr.GetLength(0) || m != mb.\_matr.GetLength(1)) {**

**throw new MatrixLengthException();**

**}**

**var matr = new int[n, m];**

**for (var i = 0; i < n; i++) {**

**for (var j = 0; j < m; j++) {**

**matr[i, j] = operation(ma.\_matr[i, j], mb.\_matr[i, j]);**

**}**

**}**

**return matr;**

**}**

**public int GetLenFirstDimension() => \_matr.GetLength(0);**

**public int GetLenSecondDimension() => \_matr.GetLength(1);**

**public void FillWithNumber(int number) {**

**var n = Lab2.N;**

**for (var i = 0; i < n; i++) {**

**for (var j = 0; j < n; j++) {**

**\_matr[i, j] = number;**

**}**

**}**

**}**

**public void FillWithRandom() {**

**var n = \_matr.GetLength(0);**

**var m = \_matr.GetLength(1);**

**var r = new Random();**

**for (var i = 0; i < n; i++) {**

**for (var j = 0; j < m; j++) {**

**\_matr[i, j] = r.Next(20);**

**}**

**}**

**}**

**public void FillWithInput() {**

**var n = \_matr.GetLength(0);**

**var m = \_matr.GetLength(1);**

**for (var i = 0; i < n; i++) {**

**for (var j = 0; j < m; j++) {**

**if (!int.TryParse(Console.ReadLine(), out \_matr[i, j])) {**

**throw new WrongInputException();**

**}**

**}**

**}**

**}**

**public int GetElem(int i, int j) {**

**return \_matr[i, j];**

**}**

**public int GetSize() {**

**return \_matr.GetLength(0);**

**}**

**public int Min() {**

**var res = \_matr[0, 0];**

**var n = \_matr.GetLength(0);**

**var m = \_matr.GetLength(1);**

**for (var i = 0; i < n; i++) {**

**for (var j = 0; j < m; j++) {**

**if (res < \_matr[i, j]) {**

**res = \_matr[i, j];**

**}**

**}**

**}**

**return res;**

**}**

**public int Max() {**

**var res = \_matr[0, 0];**

**var n = \_matr.GetLength(0);**

**var m = \_matr.GetLength(1);**

**for (var i = 0; i < n; i++) {**

**for (var j = 0; j < m; j++) {**

**if (res > \_matr[i, j]) {**

**res = \_matr[i, j];**

**}**

**}**

**}**

**return res;**

**}**

**public Vector GetVectorByRow(int indexOfRow) {**

**var length = \_matr.GetLength(1);**

**var vect = new int[length];**

**for (var j = 0; j < length; j++) {**

**vect[j] = \_matr[indexOfRow, j];**

**}**

**return new Vector(vect);**

**}**

**public Matrix Sort() {**

**var n = \_matr.GetLength(0);**

**var m = \_matr.GetLength(1);**

**var matr = new Matrix(n, m);**

**for (var i = 0; i < n; i++) {**

**var sortedVect = GetVectorByRow(i).Sort();**

**for (var j = 0; j < m; j++) {**

**matr.\_matr[i, j] = sortedVect.GetElem(j);**

**}**

**}**

**return matr;**

**}**

**public override string ToString() {**

**var res = "";**

**var n = GetSize();**

**for (var i = 0; i < n; ++i) {**

**for (var j = 0; j < n; ++j) {**

**res += \_matr[i, j] + "\t";**

**}**

**res += "\n";**

**}**

**return res;**

**}**

**}**

**public class MatrixLengthException : Exception {**

**public override string ToString() {**

**return $"{base.ToString()}\nMatrices must have same length";**

**}**

**}**

**}**

**using System;**

**using System.Linq;**

**namespace Lab2 {**

**public class Vector {**

**private int[] \_vect;**

**public Vector(int[] arr) => \_vect = arr;**

**public Vector(int n) {**

**\_vect = new int[n];**

**FillWithNumber(1);**

**}**

**public static Vector operator \*(Vector a, Matrix ma) {**

**var n = ma.GetLenFirstDimension();**

**var m = ma.GetLenSecondDimension();**

**if (a.GetLength() != n) {**

**throw new MatrixLengthException();**

**}**

**var vect = new int[m];**

**for (var i = 0; i < m; i++) {**

**vect[i] = 0;**

**for (var j = 0; j < m; j++) {**

**vect[i] += ma.GetElem(i, j) \* a.GetElem(j);**

**}**

**}**

**return new Vector(vect);**

**}**

**public static Vector operator \*(Vector a, Vector b) {**

**return new Vector(VectorsOperation(a, b, DefaultOperations.Multiply));**

**}**

**public static Vector operator \*(Vector a, int b) {**

**var n = a.\_vect.Length;**

**var vect = new int[n];**

**for (var i = 0; i < n; i++) {**

**vect[i] = a.\_vect[i] \* b;**

**}**

**return new Vector(vect);**

**}**

**public static Vector operator +(Vector a, Vector b) {**

**return new Vector(VectorsOperation(a, b, DefaultOperations.Sum));**

**}**

**public static Vector operator -(Vector a, Vector b) {**

**return new Vector(VectorsOperation(a, b, DefaultOperations.Difference));**

**}**

**private static int[] VectorsOperation(Vector a, Vector b, Func<int, int, int> operation) {**

**var n = a.\_vect.Length;**

**if (n != b.\_vect.Length) {**

**throw new VectorLengthException();**

**}**

**var vect = new int[n];**

**for (var i = 0; i < n; i++) {**

**vect[i] = operation(a.\_vect[i], b.\_vect[i]);**

**}**

**return vect;**

**}**

**public int GetLength() => \_vect.Length;**

**public void FillWithNumber(int number) {**

**var n = Lab2.N;**

**for (var i = 0; i < n; i++) {**

**\_vect[i] = number;**

**}**

**}**

**public void FillWithRandom() {**

**var r = new Random();**

**var n = \_vect.Length;**

**for (var i = 0; i < n; i++)**

**\_vect[i] = r.Next(20);**

**}**

**public void FillWithInput() {**

**var n = \_vect.Length;**

**for (var i = 0; i < n; i++) {**

**if (!int.TryParse(Console.ReadLine(), out \_vect[i])) {**

**throw new WrongInputException();**

**}**

**}**

**}**

**public int GetElem(int index) => \_vect[index];**

**public Vector Sort() {**

**var buf = new Vector((int[])\_vect.Clone());**

**Array.Sort(buf.\_vect);**

**return buf;**

**}**

**public int Max() => \_vect.Max();**

**public int Min() => \_vect.Min();**

**public override string ToString() {**

**var res = "";**

**var n = \_vect.Length;**

**for (var i = 0; i < n; ++i)**

**res += \_vect[i] + "\t";**

**return res;**

**}**

**}**

**public class VectorLengthException : Exception {**

**public override string ToString() {**

**return $"{base.ToString()}\nVectors must have same length";**

**}**

**}**

**}**

**Результат виконання програми:**

