

Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

**Лабораторна робота №3**

“C#. Семафоры, События, Мютексы, Локи, Мониторы”

з дисципліни “Програмування для Паралельних Компьютерних Систем”

Виконав:

студент 3 курсу групи ІО-52

Бояршин Ігор

Номер заліковки: 5207

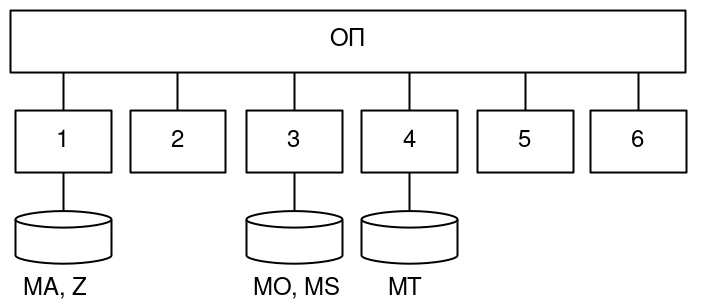
Перевірив:

Корочкін О.В.

Київ 2018 р.

**Техническое задание**

Структура паралельной компъютерной системы с общей памятью:



Выражение для подсчёта:

MA = max(Z) \* (MO\*MT) + min(Z) \* MS

Язык программирования:

C#

Средства взаимодействия задач:

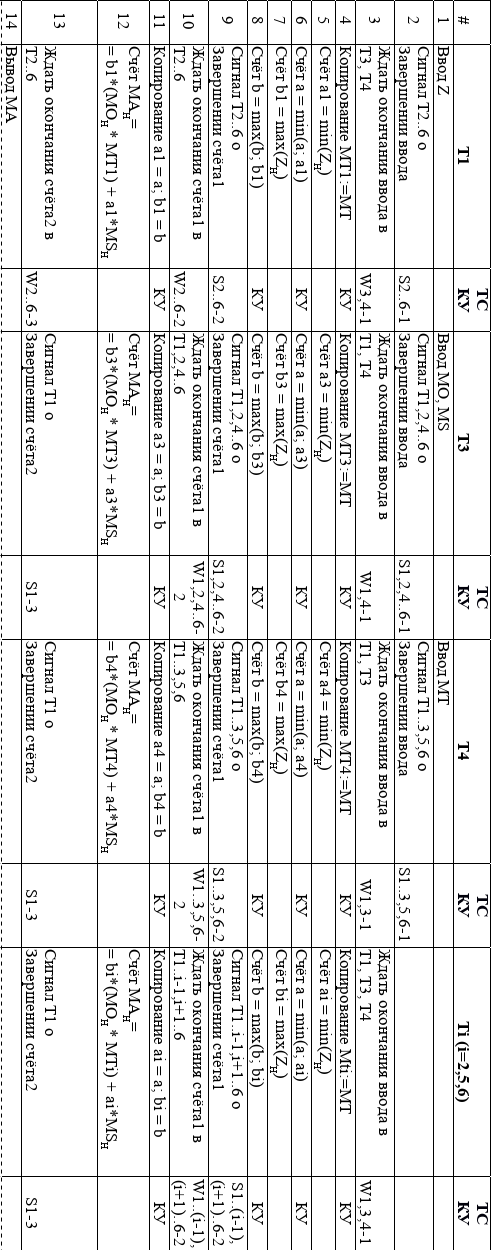
Семафоры, События, Мютексы, Локи, Мониторы, Атомарные переменные

**Выполнение работы**

**Этап 1:** Разработка параллельного математического алгоритма

1. ai = min(ZH)
2. a = min(a; ai)
   * ОР: a
3. bi = max(ZH)
4. b = max(b; bi)
   * ОР: b
5. MAH = b\* (MOH \* MT) + a\* MSH
   * ОР: MT, a, b

**Этап 2:** Разработка алгоритмов потоков



**Этап 3:** Разработка схемы взаимодействия потоков

**Этап 4:** Разработка программы

//-----------------------------------------------------------------------------

// Lab3: C#. Semaphores, Events, Mutexes, Locks, Monitors

// Task: MA = max(Z) \* (MO \* MT) + min(Z) \* MS

// Author: Igor Boyarshin

// Date: 18.03.2018

//-----------------------------------------------------------------------------

using System;

using System.Threading;

public class Lab3 {

// Constants

private static readonly int N = 12;

private static readonly int P = 6;

private static readonly int H = N / P;

private static readonly int outputThreshold = 12;

private static readonly bool doOutput = true;

// Data

private int a;

private volatile int b;

private Vector Z;

private Matrix MO, MT, MS, MA;

// Semaphores

private static Semaphore[] Semaphore\_Calculation1EndIn;

// Mutexes

private static Mutex Mutex\_SetA;

// private static Mutex Mutex\_SetB;

// Locks

private static readonly object Lock\_CopyMt = new object();

// Events

private static EventWaitHandle Event\_InputFinishIn1;

private static EventWaitHandle Event\_InputFinishIn3;

private static EventWaitHandle Event\_InputFinishIn4;

private static EventWaitHandle[] Event\_Calculation2EndIn;

// Monitors

private static object Monitor\_SetAb = new object();

//-----------------------------------------------------------------------------

public Lab3(int N) {

// Preparations

MA = new Matrix(N, 0);

b = 0;

a = 400000000;

// Semaphores

Semaphore\_Calculation1EndIn = new Semaphore[P];

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

Semaphore\_Calculation1EndIn[i] = new Semaphore(0, P);

}

// Mutexes

Mutex\_SetA = new Mutex(false);

// Events

// manual

Event\_InputFinishIn1 = new EventWaitHandle(false, EventResetMode.ManualReset);

Event\_InputFinishIn3 = new EventWaitHandle(false, EventResetMode.ManualReset);

Event\_InputFinishIn4 = new EventWaitHandle(false, EventResetMode.ManualReset);

// automatic

Event\_Calculation2EndIn = new EventWaitHandle[P];

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

Event\_Calculation2EndIn[i] =

new EventWaitHandle(false, EventResetMode.AutoReset);

}

// Threads

Thread[] threads = new Thread[P];

//-----------------------------------------------------------------------------

for (int thread = 0; thread < P; thread++) {

int index = thread;

threads[thread] = new Thread(() => {

int tid = index + 1;

int low = (tid - 1) \* H;

int high = tid \* H;

if (doOutput) {

Console.WriteLine(":> Thread " + tid + " started!");

}

// 1. Input

switch (tid) {

case 1:

Z = FillVectorOnes();

break;

case 3:

MO = FillMatrixOnes();

MS = FillMatrixOnes();

break;

case 4:

MT = FillMatrixOnes();

break;

default:

break;

}

// 2. Signal T1..6 about input finish

switch (tid) {

case 1:

Event\_InputFinishIn1.Set();

break;

case 3:

Event\_InputFinishIn3.Set();

break;

case 4:

Event\_InputFinishIn4.Set();

break;

default:

break;

}

// 3. Wait for input finish in T1, T3, T4

Event\_InputFinishIn1.WaitOne();

Event\_InputFinishIn3.WaitOne();

Event\_InputFinishIn4.WaitOne();

// 4. Copy MTi = MT

Matrix MTi;

lock (Lock\_CopyMt) {

MTi = MT.copy();

}

// 5.

int ai = FindMin(ref Z, low, high);

// 6.

Mutex\_SetA.WaitOne();

a = Min(a, ai);

Mutex\_SetA.ReleaseMutex();

// 7.

int bi = FindMax(ref Z, low, high);

// 8.

b = Max(b, bi);

// 9. Signal T1..6 about Calculation1 finish

Semaphore\_Calculation1EndIn[tid - 1].Release(P);

// 10. Wait for Calculation1 finish in T1..6

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

Semaphore\_Calculation1EndIn[i].WaitOne();

}

// 11. Copy a, b

Monitor.Enter(Monitor\_SetAb);

try {

ai = a;

bi = b;

} finally {

Monitor.Exit(Monitor\_SetAb);

}

// 12. Calculation2

for (int i = low; i < high; i++) {

for (int j = 0; j < N; j++) {

int product = 0;

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

product += MO[i, k] \* MTi[k, j];

}

MA[i, j] = bi \* product + ai \* MS[i, j];

}

}

// 13. Wait for Calculation2 finish in T1..6

Event\_Calculation2EndIn[tid - 1].Set();

switch (tid) {

case 1:

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

Event\_Calculation2EndIn[i].WaitOne();

}

break;

default:

break;

}

// 14. Output

switch (tid) {

case 1:

if (N <= outputThreshold) {

OutputMatrix(ref MA);

}

break;

default:

break;

}

if (doOutput) {

Console.WriteLine(":> Finished Thread " + tid);

}

});

}

//-----------------------------------------------------------------------------

for (int i = 0; i < 6; i++) {

threads[i].Start();

}

for (int i = 0; i < 6; i++) {

threads[i].Join();

}

}

//-----------------------------------------------------------------------------

public void OutputMatrix(ref Matrix matrix) {

for (int row = 0; row < N; row++) {

for (int column = 0; column < N; column++) {

Console.Write(matrix[row, column] + " ");

}

Console.WriteLine();

}

}

public Vector FillVectorOnes() {

return new Vector(N, 1);

}

public Matrix FillMatrixOnes() {

return new Matrix(N, 1);

}

private int Min(int a, int b) {

return a < b ? a : b;

}

private int Max(int a, int b) {

return a > b ? a : b;

}

private int FindMin(ref Vector v, int low, int high) {

return FindMinOrMax(ref v, low, high, true);

}

private int FindMax(ref Vector v, int low, int high) {

return FindMinOrMax(ref v, low, high, false);

}

private int FindMinOrMax(ref Vector v, int low, int high, bool findMinElement) {

int current = v[low];

for (int i = low; i < high; i++) {

int element = v[i];

if (findMinElement ?

(element < current) : (element > current)) {

current = element;

}

}

return current;

}

//-----------------------------------------------------------------------------

public static void Main(String[] args) {

// Console.WriteLine(":> Main program started!");

new Lab3(N);

}

}

// ------------------------------------------------------------------------ //

// -------------------------------Vector----------------------------------- //

// ------------------------------------------------------------------------ //

public class Vector {

private readonly int SIZE;

private int[] elements;

public Vector(int size) {

this.SIZE = size;

this.elements = new int[SIZE];

}

public Vector(int size, int v) {

this.SIZE = size;

this.elements = new int[SIZE];

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

this.elements[i] = v;

}

}

public Vector(int size, int[] elements) {

this.SIZE = size;

this.elements = elements;

}

public int this[int index] {

get {

return elements[index];

}

set {

elements[index] = value;

}

}

}

// ------------------------------------------------------------------------ //

// -------------------------------Matrix----------------------------------- //

// ------------------------------------------------------------------------ //

public class Matrix {

private readonly int SIZE;

private int[,] elements;

public Matrix(int size) {

this.SIZE = size;

this.elements = new int[SIZE, SIZE];

}

public Matrix(int size, int v) {

this.SIZE = size;

this.elements = new int[SIZE, SIZE];

for (int row = 0; row < SIZE; row++) {

for (int column = 0; column < SIZE; column++) {

this.elements[row, column] = v;

}

}

}

public Matrix(int size, int[,] elements) {

this.SIZE = size;

this.elements = elements;

}

public int this[int x, int y] {

get {

return elements[x, y];

}

set {

elements[x, y] = value;

}

}

public Matrix copy() {

int[,] elems = new int[SIZE, SIZE];

for (int row = 0; row < SIZE; row++) {

for (int column = 0; column < SIZE; column++) {

elems[row, column] = elements[row, column];

}

}

return new Matrix(SIZE, elems);

}

}