Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Основи паралельного програмування

Лабораторна робота №2

**«C#. Семафори, мютекси, події, критичні секції»**

Виконала:

студентка групи ІВ-71

Молчанова В.С.

Перевірив:

Корочкін О.В.

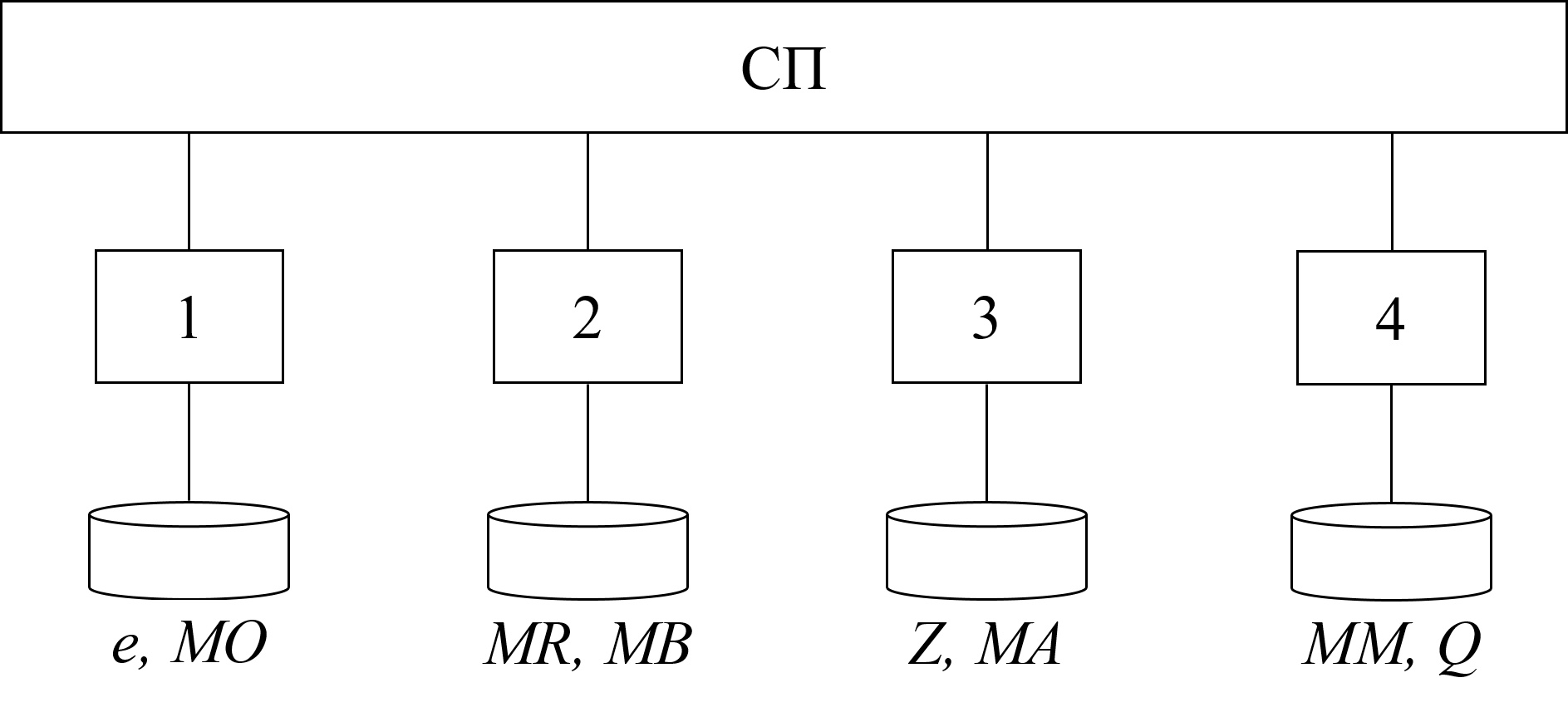
Київ

2020 р.

**Мета роботи**: розробка програми для ПКС зі СП.

**Завдання**:

* Р=4
* МA=e\*min(Z)\*(MO\*MB) + (Z\*Q)\*(MR\*MM)



* Мова програмування: С#
* Засоби: Семафори, мютекси, події та критичні секції.

## Виконання роботи

### Етап 1. Розробка паралельного алгоритму

Алгоритм обчислення виразу:

1. 
2. 
3. 
4. 
5. 

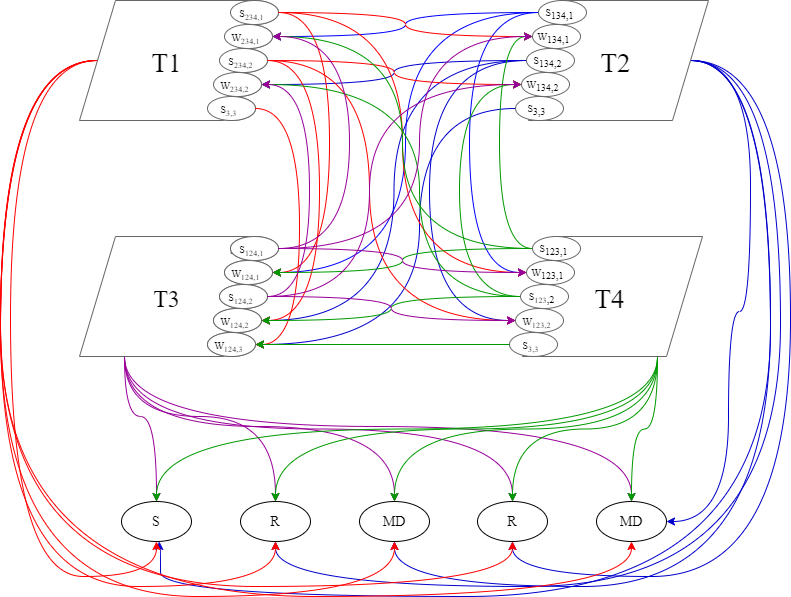
Спільні ресурси: .

### Етап 2. Розробка алгоритмів потоків

|  |  |  |
| --- | --- | --- |
|  | **Т3** | ТС КД |
| 1. | Ввести дані: Z |  |
| 2. | Сигнал решті потоків про введення даних |  |
| 3. | Очікувати введення даних в решті потоків |  |
| 4. |  |  |
| 5. |  |  |
| 6. |  |  |
| 7. | Сигнал решті потоків про завершення обчислення |  |
| 8. | Очікувати завершення обчислення в решті потоків |  |
| 9. | Копіювати | КД |
| 10. | Обчислити |  |
| 11. | Очікувати завершення обчислення в решті потоків |  |
| 12. | Вивести *МА* |  |

|  |  |  |
| --- | --- | --- |
|  | **Тi, де** | ТС КД |
| 1. | Ввести дані:   * Якщо i=1: *e, MO* * Якщо i=2: *MR, MB* * Якщо i=4: *MM, Q* |  |
| 2. | Сигнал решті потоків про введення даних |  |
| 3. | Очікувати введення даних в решті потоків |  |
| 4. |  |  |
| 5. |  |  |
| 6. |  |  |
| 7. | Сигнал решті потоків про завершення обчислення |  |
| 8. | Очікувати завершення обчислення в решті потоків |  |
| 9. | Копіювати | КД |
| 10. | Обчислити |  |
| 11. | Сигнал Т3 про завершення обчислення |  |

### Структурна схема взаємодії потоків



### Лістинг програми

#### Program.cs

using System;

using System.Collections.Generic;

using System.Threading;

namespace OPP\_Lab2

{

internal class Program

{

public static int N = 8;

public static int P = 4;

public static int H = N / P;

public static Matrix MO, MR, MB, MM;

public static Matrix MA;

public static Vector Z, Q;

public static int e, s;

public static int m;

private static List<Semaphore> \_inputSemaphores;

private static object lck = new object();

private static List<Semaphore> \_minAndSemaphores;

private static Mutex \_copyMutex;

private static List<EventWaitHandle> \_calculateEvents;

private static List<Thread> \_threads;

public static void Main(string[] args)

{

Console.ReadKey();

var startTime = DateTime.Now.Ticks;

\_inputSemaphores = new List<Semaphore>();

\_minAndSemaphores = new List<Semaphore>();

\_calculateEvents = new List<EventWaitHandle>();

\_threads = new List<Thread>();

MA = new Matrix(N);

m = int.MaxValue;

for (var i = 0; i < P; i++)

{

\_inputSemaphores.Add(new Semaphore(0, P-1));

\_minAndSemaphores.Add(new Semaphore(0, P - 1));

\_copyMutex = new Mutex();

\_calculateEvents.Add(new AutoResetEvent(false));

\_threads.Add(new Thread(MyTask));

}

for (var i = 0; i < P; i++)

{

\_threads[i].Start(i+1);

}

for (var i = 0; i < P; i++)

{

\_threads[i].Join();

}

var endTime = DateTime.Now.Ticks;

var totalTicks = endTime - startTime;

Console.WriteLine($"N = {N}, time = {totalTicks / 10000} ms");

Console.ReadKey();

}

private static void Input(int taskNumber)

{

switch (taskNumber)

{

case 1:

{

MO = new Matrix(N);

e = 1;

MO.InputMatrix();

break;

}

case 2:

{

MR = new Matrix(N);

MB = new Matrix(N);

MR.InputMatrix();

MB.InputMatrix();

break;

}

case 3:

{

Z = new Vector(N);

Z.InputVector();

break;

}

case 4:

{

MM = new Matrix(N);

MM.InputMatrix();

Q = new Vector(N);

Q.InputVector();

break;

}

}

}

private static void MinAndProduct(int taskNumber)

{

var Zi = Z.GetSubVector((taskNumber - 1) \* H, H);

var Qi = Q.GetSubVector((taskNumber - 1) \* H, H);

var mi = Zi.Min();

lock (lck)

{

m = m > mi ? mi : m;

}

var si = 0;

for (var i = 0; i < H; i++)

{

si += Zi[i] \* Qi[i];

}

lock (lck)

{

s += si;

}

}

private static void Calculate(int taskNumber, int ei, int mi, Matrix MOi, int si, Matrix MRi)

{

// MAH = ei \* mi \* (MOi \* MBH ) + si \* (MRi \* MDH)

for (var i = 0; i < N; i++)

{

for (var j = (taskNumber - 1) \* H; j < taskNumber \* H; j++)

{

var sum1 = 0;

var sum2 = 0;

for (var k = 0; k < N; k++)

{

sum1 += MOi[i][k] \* MB[k][j];

sum2 += MRi[i][k] \* MM[k][j];

}

MA[i][j] = ei \* mi \* sum1 + si \* sum2;

}

}

}

private static void MyTask(object num)

{

var taskNumber = (int)num;

Console.WriteLine($"T{taskNumber} start");

var otherTasksNumbers = new List<int> { 1, 2, 3, 4 };

otherTasksNumbers.Remove(taskNumber);

Input(taskNumber);

\_inputSemaphores[taskNumber - 1].Release(P - 1);

foreach (var otherTasksNumber in otherTasksNumbers)

{

\_inputSemaphores[otherTasksNumber - 1].WaitOne();

}

// Calculate m = min(Z) та d = Z \* Q

MinAndProduct(taskNumber);

\_minAndSemaphores[taskNumber - 1].Release(P - 1);

foreach (var otherTasksNumber in otherTasksNumbers)

{

\_minAndSemaphores[otherTasksNumber - 1].WaitOne();

}

int ei, mi, si;

Matrix MOi, MRi;

// Copy common data

\_copyMutex.WaitOne();

{

ei = e;

mi = m;

MOi = new Matrix(MO);

si = s;

MRi = new Matrix(MR);

}

\_copyMutex.ReleaseMutex();

// Calculate MAH = ei \* mi \* (MOi \* MBH ) + si \* (MRi \* MDH)

Calculate(taskNumber, ei, mi, MOi, si, MRi);

if (taskNumber == 3)

{

foreach (var otherTasksNumber in otherTasksNumbers)

{

\_calculateEvents[otherTasksNumber - 1].WaitOne();

}

if (N <= 8)

Console.WriteLine(MA);

}

else

{

\_calculateEvents[taskNumber - 1].Set();

}

Console.WriteLine($"T{taskNumber} end");

}

}

}

#### Vector.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace OPP\_Lab2

{

public class Vector

{

private readonly List<int> \_vector;

public int Length => \_vector.Count;

public Vector(int length)

{

\_vector = new List<int>(length);

for (var i = 0; i < length; i++)

{

\_vector.Add(0);

}

}

public Vector(Vector other)

{

\_vector = new List<int>(other.\_vector);

}

public Vector(IEnumerable<int> vector)

{

\_vector = new List<int>(vector);

}

public Vector GetSubVector(int index, int length)

{

var result = new Vector(new List<int>(\_vector.GetRange(index, length)));

return result;

}

public int Min() => \_vector.Min();

public void InputVector()

{

for (var i = 0; i < Length; i++)

{

\_vector[i] = 1;

}

}

public int this[int i]

{

get => \_vector[i];

set => \_vector[i] = value;

}

public override string ToString()

{

var s = "";

for (var i = 0; i < Length; i++)

{

s += $"{\_vector[i]} ";

}

return s;

}

}

}

#### Matrix.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace OPP\_Lab2

{

public class Matrix

{

private readonly List<Vector> \_matrix;

public int Length => \_matrix.Count;

public Matrix(int len)

{

\_matrix = new List<Vector>(len);

for (var i = 0; i < len; i++)

{

\_matrix.Add(new Vector(len));

}

}

public Matrix(List<Vector> matrix)

{

\_matrix = new List<Vector>(matrix);

}

public Matrix(Matrix mat)

{

\_matrix = new List<Vector>(mat.\_matrix);

}

public void InputMatrix()

{

for (var i = 0; i < Length; i++)

{

\_matrix[i] = new Vector(\_matrix.Count);

\_matrix[i].InputVector();

}

}

public int this[int i, int j]

{

get => \_matrix[i][j];

set => \_matrix[i][j] = value;

}

public Vector this[int i]

{

get => \_matrix[i];

set => \_matrix[i]= value;

}

public override string ToString()

{

var s = "";

foreach (var vector in \_matrix)

{

s += $"{vector}\n";

}

return s;

}

}

}