

Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

**Лабораторна робота №6**

“Java. Монітори”

з дисципліни “Програмування для Паралельних Компьютерних Систем”

Виконав:

студент 3 курсу групи ІО-52

Бояршин Ігор

Номер заліковки: 5207

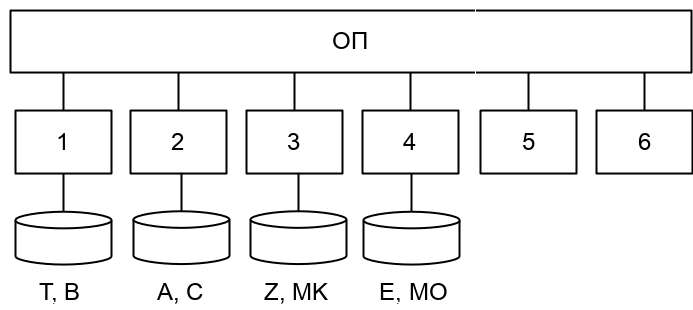
Перевірив:

Корочкін О.В.

Київ 2018 р.

**Техническое задание**

Структура паралельной компъютерной системы с общей памятью:



Выражение для подсчёта:

A = (B\*C)\*Z + min(E) \* T \* (MO\*MK)

Язык программирования:

Java

Средства взаимодействия задач:

Мониторы

**Выполнение работы**

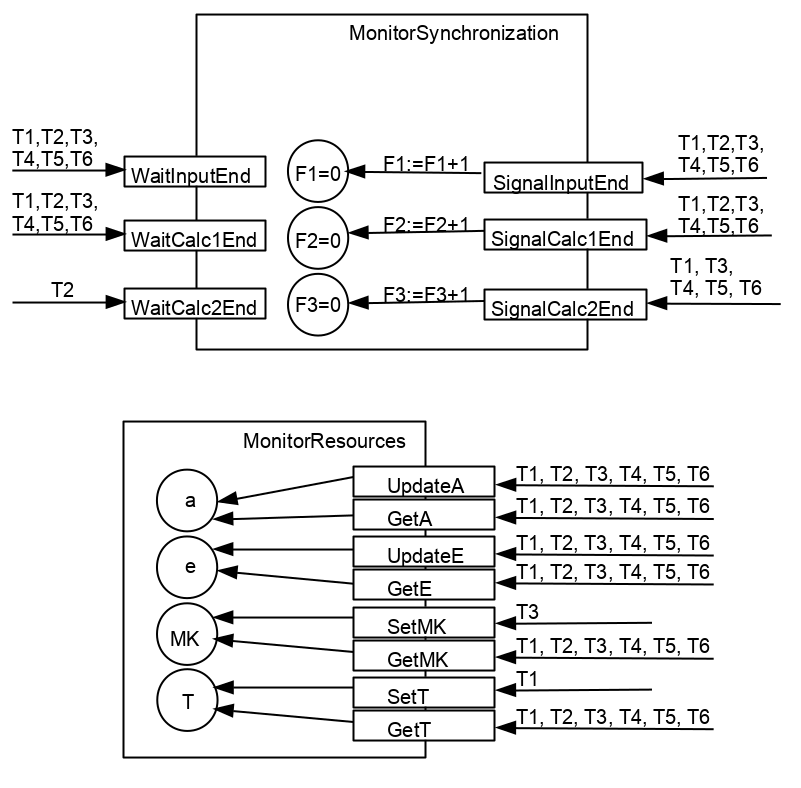
**Этап 1:** Разработка параллельного математического алгоритма

1. ai = BH \* CH , i = (1..P), P – количество потоков, P = 6
2. a = a + ai , i = (1..P)
   * ОР: а
3. ei = min(EH) , i = (1..P)
4. e = min(e; ei) , i = (1..P)
   * ОР: е
5. AH = b \* ZH + e \* T \* (MOH \* MK)
   * ОР: T, MK, b, e

**Этап 2:** Разработка алгоритмов потоков

|  |  |  |
| --- | --- | --- |
| # | Ti (1 <= I <= 6) | ТС | КУ |
| 1 | **Если** i=1 **то**:   * Ввод T, B   **Если** i=2 **то**:   * Ввод C   **Если** i=3 **то**:   * Ввод Z, MK   **Если** i=4 **то**:   * Ввод E, MO |  |
| 2 | **Сигнал** Tj (j=1..6, j != i) о завершении ввода | S1..6-1 |
| 3 | **Ждать** сигнала о завершения ввода в Tj (j=1..6, j != i) | W1..6-1 |
| 4 | ai = BH \* CH |  |
| 5 | a = a + ai | КУ |
| 6 | ei = min(EH) |  |
| 7 | e = min(e, ei) | КУ |
| 8 | **Сигнал** Tj (j=1..6, j != i) о завершении счёта1 | S1..6-2 |
| 9 | **Ждать** сигнала о завершения счёта1 в Tj (j=1..6, j != i) | W1..6-2 |
| 10 | Копирование MKi := MK, Ti := T, ai := a, ei := e | КУ |
| 11 | Счёт AH = ai \* ZH + ei \* Ti \* (MOH \* MKi) |  |
| 12 | **Если** i != 2 **то**:   * **Сигнал** T2 о завершении счёта2   **Иначе** **Если** i = 2 **то**:   * **Ждать** завершения счёта в Tj (1 <= j <= 6, j != 2) | S2-3  W1,3..6-3 |
| 13 | **Если** i=2 **то**:   * Вывод A |  |

**Этап 3:** Разработка схемы взаимодействия потоков



**Этап 4:** Разработка программы

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

// Lab6: Java. Monitors

// Task: A = (B\*C) \* Z + min(E) \* T \* (MO\*MK)

// Author: Igor Boyarshin

// Date: 24.04.2018

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

import java.util.concurrent.RecursiveAction;

import java.util.concurrent.ForkJoinPool;

import java.util.concurrent.ForkJoinTask;

import java.util.List;

import java.util.ArrayList;

public class Main {

private static final int N = 12;

private static final int P = 6;

private static final int H = N / P;

private static final int OUTPUT\_THRESHOLD = 8;

private static ForkJoinPool thePool = new ForkJoinPool(P);

private MonitorResources monitorResources = new MonitorResources();

private MonitorSynchronization monitorSynchronization = new MonitorSynchronization();

private Vector A = new Vector(N, 0);

private Vector B;

private Vector C;

private Vector Z;

private Vector E;

private Matrix MO;

public class Task extends RecursiveAction {

private static final long serialVersionUID = 42L;

private final int tid;

private final int start;

private final int finish;

public Task(int tid, int start, int finish) {

this.tid = tid;

this.start = start;

this.finish = finish;

}

@Override

protected void compute() {

if (finish - start == H) {

process();

} else {

List<Task> subtasks = new ArrayList<>();

subtasks.add(new Task(tid, start, start + H));

subtasks.add(new Task(tid + 1, start + H, finish));

ForkJoinTask.invokeAll(subtasks);

}

}

private void process() {

System.out.println(":> Starting Thread " + tid);

// Input

switch (tid) {

case 1:

final Vector T = fillVectorOnes();

monitorResources.setT(T);

B = fillVectorOnes();

break;

case 2:

C = fillVectorOnes();

break;

case 3:

Z = fillVectorOnes();

final Matrix MK = fillMatrixOnes();

monitorResources.setMK(MK);

break;

case 4:

E = fillVectorOnes();

MO = fillMatrixOnes();

break;

default:

break;

}

// Sync on input

monitorSynchronization.signalInputEnd();

monitorSynchronization.waitInputEnd();

// Calc1 a

int ai = 0;

for (int i = start; i < finish; i++) {

ai += B.get(i) \* C.get(i);

}

monitorResources.updateA(ai);

// Calc1 e

int ei = E.get(start);

for (int i = start; i < finish; i++) {

if (E.get(i) < ei) {

ei = E.get(i);

}

}

monitorResources.updateE(ei);

// Sync on calc1

monitorSynchronization.signalCalc1End();

monitorSynchronization.waitCalc1End();

// Copies

ai = monitorResources.getA();

ei = monitorResources.getE();

final Matrix MKi = monitorResources.getMK();

final Vector Ti = monitorResources.getT();

// Calc2

for (int h = start; h < finish; h++) {

int temp = 0;

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

int prod = 0;

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

prod += MO.get(h, j) \* MKi.get(j, i);

}

temp += Ti.get(i) \* prod;

}

A.set(h, ai \* Z.get(h) + ei \* temp);

}

// Sync on calc2

switch (tid) {

case 2:

monitorSynchronization.waitCalc2End();

break;

default:

monitorSynchronization.signalCalc2End();

break;

}

// Output

if (tid == 2) {

if (N <= OUTPUT\_THRESHOLD) {

outputVector(A);

}

}

System.out.println(":> Finished Thread " + tid);

}

}

public Main() {

thePool.invoke(new Task(1, 0, N));

}

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

public static void main(String[] args) {

System.out.println(":> Main program started!");

new Main();

}

public void outputVector(Vector vector) {

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

System.out.print(vector.get(i) + " ");

}

System.out.println();

}

public void outputMatrix(Matrix matrix) {

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

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

System.out.print(matrix.get(row, column) + " ");

}

System.out.println();

}

System.out.println();

}

public Vector fillVectorOnes() {

return new Vector(N, 1);

}

public Matrix fillMatrixOnes() {

return new Matrix(N, 1);

}

}

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

// Monitor Resources

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

public class MonitorResources {

private int a = 0;

private int e = Integer.MAX\_VALUE;

private Matrix MK;

private Vector T;

public synchronized void updateA(int a) {

this.a += a;

}

public synchronized int getA() {

return a;

}

public synchronized void updateE(int e) {

if (e < this.e) {

this.e = e;

}

}

public synchronized int getE() {

return e;

}

public synchronized void setMK(Matrix MK) {

this.MK = MK;

}

public synchronized Matrix getMK() {

return MK;

}

public synchronized void setT(Vector T) {

this.T = T;

}

public synchronized Vector getT() {

return T;

}

}

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

// Monitor Synchronization

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

public class MonitorSynchronization {

private int F1 = 0;

private int F2 = 0;

private int F3 = 0;

public synchronized void signalInputEnd() {

F1++;

if (F1 == 6) {

notifyAll();

}

}

public synchronized void signalCalc1End() {

F2++;

if (F2 == 6) {

notifyAll();

}

}

public synchronized void signalCalc2End() {

F3++;

if (F3 == 5) {

notifyAll();

}

}

public synchronized void waitInputEnd() {

while (F1 < 6) {

try {

wait();

} catch (InterruptedException e) {}

}

}

public synchronized void waitCalc1End() {

while (F2 < 6) {

try {

wait();

} catch (InterruptedException e) {}

}

}

public synchronized void waitCalc2End() {

while (F3 < 5) {

try {

wait();

} catch (InterruptedException e) {}

}

}

}

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

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

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

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.regex.Pattern;

public class Vector {

private final int SIZE;

private int[] elements;

public Vector(int size) {

this.SIZE = size;

this.elements = new int[SIZE];

}

public Vector(int size, int value) {

this.SIZE = size;

this.elements = new int[SIZE];

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

this.elements[i] = value;

}

}

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

this.SIZE = size;

this.elements = elements;

}

public int get(int index) {

return this.elements[index];

}

public void set(int index, int value) {

this.elements[index] = value;

}

}

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

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

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

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.regex.Pattern;

public class Matrix {

private final int SIZE;

private int[][] elements;

public Matrix(int size) {

this.SIZE = size;

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

}

public Matrix(int size, int value) {

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] = value;

}

}

}

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

this.SIZE = size;

this.elements = elements;

}

public int get(int row, int column) {

return this.elements[row][column];

}

public void set(int row, int column, int value) {

this.elements[row][column] = value;

}

}