**Міністерство освіти і науки України  
Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки  
Кафедра обчислювальної техніки**

**Лабораторна робота №2**

«Програмування для комп’ютерних систем зі спільною пам’яттю. Семафори, критичні секції, атомік-змінні, бар’єри»

з дисципліни  
«Програмне забезпечення високопродуктивних комп’ютерних систем»

Виконала:

студентка групи ІМ-11   
Бащак Ярина Володимирівна

Перевірив:

доц. Корочкін О. В.

Київ 2024

**Завдання.**

Побудувати паралельний алгоритм для обчислення формули. Розробити паралельну програму за допомогою мови Java, яка б виконувала це паралельне обчислення. Формула, схема введення та виведення даних потоками обирається згідно варіанту.

Програма має містити класи Lab2, Data та потоки T1, T2, T3, T4.

**Варіант 10:**

Завдання: MR = MB\*(MC\*MM)\*d + min(Z)\*MC

|  |  |  |  |
| --- | --- | --- | --- |
| Введення – виведення даних | | | |
| 1 | 2 | 3 | 4 |
| MB | MR | MC | Z, d, MM |

**Виконання роботи.**

**Етап 1. Побудова паралельного математичного алгоритму.**

Розроблено наступний алгоритм:

1. zi = min(Zн), де Zн - це H елементів вектора Z;
2. z = min(z, zi); СР: z;
3. MRн = MB \* (MC \* MMн) \* d + z \* MCн; СР: MB, MC, d, z; де MRн, ММн, МСн - це Н стовпців відповідних матриць.

Всі вказані спільні ресурси, окрім змінної z, яка повинна бути перезаписана, використовуються лише для читання, тому не потребують окремого захисту.

**Етап 2. Розробка алгоритмів потоків.**

Далі наведено покровий алгоритм для кожної з чотирьох задач.

Задача T1 Точки синхронізації

1. Вивід повідомлення про старт потоку
2. Ведення MB
3. Сигнал задачам T2, T3, T4 про введення MB -- S2-1, S3-1, S4-1
4. Чекати на введення даних в задачах T3, T4 -- W3-1, W4-1
5. Обчислення 1: zi = min(Zн)
6. Обчислення 2: z = min(z, zi) -- КД1
7. Сигнал задачам T2, T3, T4 про завершення обчислення 2 -- S2-2, S3-2, S4-2
8. Чекати на завершення обчислення 2 в задачах T2, T3, T4 -- W2-1, W3-2, W4-2
9. Копіювання zi = z -- КД2
10. Обчислення 3: MRн = MB \* (MC \* MMн) \* d + zi \* MCн
11. Сигнал задачі T2 про завершення обчислення 3 -- S2-3
12. Вивід повідомлення про завершення потоку

Задача T2 Точки синхронізації

1. Вивід повідомлення про старт потоку
2. Чекати на введення даних в задачах T1, T3, T4 -- W1-1, W3-1, W4-1
3. Обчислення 1: zi = min(Zн)
4. Обчислення 2: z = min(z, zi) -- КД1
5. Сигнал задачам T1, T3, T4 про завершення обчислення 2 -- S1-1, S3-1, S4-1
6. Чекати на завершення обчислення 2 в задачах T1, T3, T4 -- W1-2, W3-2, W4-2
7. Копіювання zi = z -- КД2
8. Обчислення 3: MRн = MB \* (MC \* MMн) \* d + zi \* MCн
9. Чекати на завершення обчислення 3 в задачах T1, T3, T4 -- W1-3, W3-3, W4-3
10. Виведення результату MR
11. Вивід повідомлення про завершення потоку

Задача T3 Точки синхронізації

1. Вивід повідомлення про старт потоку
2. Ведення MC
3. Сигнал задачам T1, T2, T4 про введення MC -- S1-1, S2-1, S4-1
4. Чекати на введення даних в задачах T1, T4 -- W1-1, W4-1
5. Обчислення 1: zi = min(Zн)
6. Обчислення 2: z = min(z, zi) -- КД1
7. Сигнал задачам T1, T2, T4 про завершення обчислення 2 -- S1-2, S2-2, S4-2
8. Чекати на завершення обчислення 2 в задачах T1, T2, T4 -- W1-2, W2-1, W4-2
9. Копіювання zi = z -- КД2
10. Обчислення 3: MRн = MB \* (MC \* MMн) \* d + zi \* MCн
11. Сигнал задачі T2 про завершення обчислення 3 -- S2-3
12. Вивід повідомлення про завершення потоку

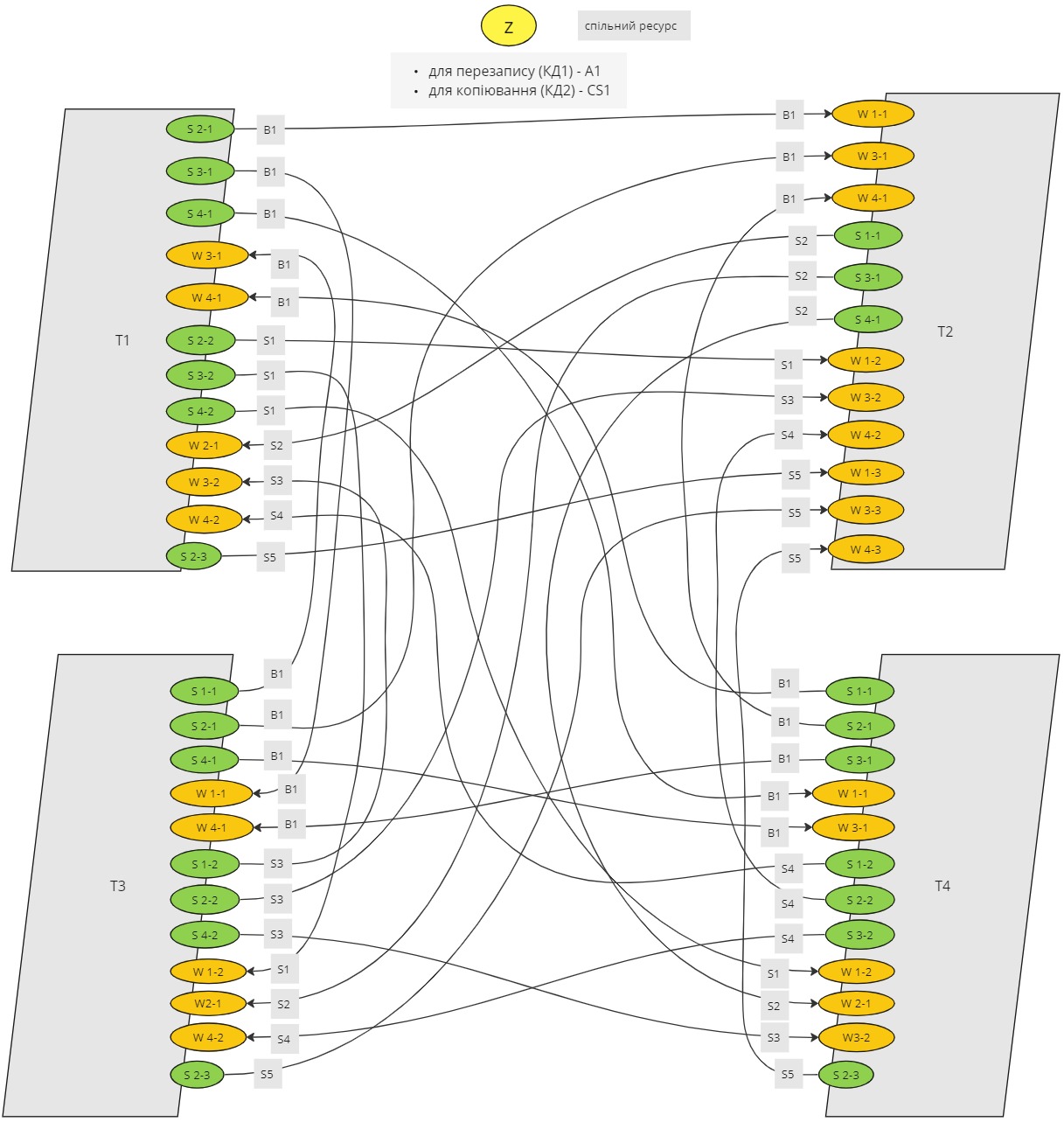
Задача T4 Точки синхронізації

1. Вивід повідомлення про старт потоку
2. Ведення Z, d, MM
3. Сигнал задачам T2, T3, T1 про введення Z, d, MM -- S2-1, S3-1, S1-1
4. Чекати на введення даних в задачах T3, T1 -- W3-1, W1-1
5. Обчислення 1: zi = min(Zн)
6. Обчислення 2: z = min(z, zi) -- КД1
7. Сигнал задачам T2, T3, T1 про завершення обчислення 2 -- S2-2, S3-2, S1-2
8. Чекати на завершення обчислення 2 в задачах T2, T3, T4 -- W2-1, W3-2, W1-2
9. Копіювання zi = z -- КД2
10. Обчислення 3: MRн = MB \* (MC \* MMн) \* d + zi \* MCн
11. Сигнал задачі T2 про завершення обчислення 3 -- S2-3
12. Вивід повідомлення про завершення потоку

**Етап 3. Розробка схеми взаємодії задач**

При розробці структурної схеми взаємодії задач, що наведена на мал. 1, було враховано які засоби організації синхронізації реалізовані в мові Java, а саме семафори, критичні секції, атомік-змінні і бар’єри. Далі наведено опис використання кожного з них:

* бар’єр В1 для синхронізації введення даних з потоків Т1, Т3, Т4;
* атомік-змінна z для реалізації КД1 (перезапис спільного ресурсу);
* семафори S1, S2, S3, S4 для сигналів про завершення обчислення 2;
* критична секція CS1 для реалізації КД2 (копіювання спільного ресурсу);
* семафор S5 для сигналів до Т2 про завершення задачами обчислення 3.



Мал. 1 Схема взаємодії задач

**Етап 4. Розроблення програми.**

На основі розроблених алгоритмів на етапі 2 і структурної схеми взаємодії задач на етапі 3 було реалізовано програмі на мові Java.

Програма складається з основного класу Lab2, класу Data, що зберігає всі спільні змінні та містить статичні методи для роботи над векторами і матрицями, а також класи T1, T2, T3, T4 для відповідних потоків. Лістинг програми представлено далі.

// Програмне забезпечення високопродуктивних комп'ютерних систем

// Лабораторна робота №2: програмування для КС із СП. Семафори, критичні секції, атомік-змінні, бар’єри

// варіант 10

// MR = MB\*(MC\*MM)\*d + min(Z)\*MC

// Бащак Ярина Володимирівна

// група ІМ-11

// 20.03.2024

public class Lab2 {

    public static void main(String[] args) {

        Data D = new Data();

        Thread1 T1 = new Thread1(1, D);

        Thread2 T2 = new Thread2(2, D);

        Thread3 T3 = new Thread3(3, D);

        Thread4 T4 = new Thread4(4, D);

        T1.start();

        T2.start();

        T3.start();

        T4.start();

    }

}

public class Data {

    public int N = 20;

    public int P = 4;

    public int H = N / P;

    public int[][] MB = new int[N][N];

    public int[][] MC = new int[N][N];

    public int[] Z = new int[N];

    public AtomicInteger z = new AtomicInteger(Integer.MAX\_VALUE);

    public int d;

    public int[][] MM = new int[N][N];

    public int[][] MR = new int[N][N];

    Semaphore S1 = new Semaphore(0);

    Semaphore S2 = new Semaphore(0);

    Semaphore S3 = new Semaphore(0);

    Semaphore S4 = new Semaphore(0);

    Semaphore S5 = new Semaphore(0);

    CyclicBarrier B1 = new CyclicBarrier(4);

    public synchronized int CS1() {

        return z.intValue();

    }

    public static int findMin(int[] vector) {

        int min = vector[0];

        for (int i = 0; i < vector.length; i++) {

            if (vector[i] < min) {

                min = vector[i];

            }

        }

        return min;

    }

    public static int[][] multiplyMatrices(int[][] matrix1, int[][] matrix2) {

        int m = matrix1.length;

        int n = matrix2.length;

        int p = matrix2[0].length;

        int[][] result = new int[m][p];

        for (int i = 0; i < m; i++) {

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

                result[i][j] = 0;

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

                    result[i][j] += matrix1[i][k] \* matrix2[k][j];

                }

            }

        }

        return result;

    }

    public static int[][] multiplyMatrixByNumber(int[][] matrix, int number) {

        int rows = matrix.length;

        int columns = matrix[0].length;

        int[][] result = new int[rows][columns];

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

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

                result[i][j] = matrix[i][j] \* number;

            }

        }

        return result;

    }

    public static int[][] addMatrices(int[][] matrix1, int[][] matrix2) {

        int rows = matrix1.length;

        int columns = matrix1[0].length;

        int[][] result = new int[rows][columns];

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

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

                result[i][j] = matrix1[i][j] + matrix2[i][j];

            }

        }

        return result;

    }

    public static int[][] getSubmatrixFromColumns(int[][] matrix, int start, int end) {

        int rows = matrix.length;

        int columns = end - start;

        int[][] submatrix = new int[rows][columns];

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

            for (int j = start; j < end; j++) {

                submatrix[i][j - start] = matrix[i][j];

            }

        }

        return submatrix;

    }

    public static void insertSubmatrixIntoMatrix(int[][] targetMatrix, int[][] submatrix, int start) {

        int rows = targetMatrix.length;

        int submatrixColumns = submatrix[0].length;

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

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

                targetMatrix[i][start + j] = submatrix[i][j];

            }

        }

    }

    public void calculateStep3(int start, int end, int zi) {

        int[][] MM\_H = Data.getSubmatrixFromColumns(MM, start, end);

        int[][] MC\_H = Data.getSubmatrixFromColumns(MC, start, end);

        int[][] MC\_MMh = Data.multiplyMatrices(MC, MM\_H);

        int[][] MB\_MC\_MMh = Data.multiplyMatrices(MB, MC\_MMh);

        int[][] MB\_MC\_MMh\_d = Data.multiplyMatrixByNumber(MB\_MC\_MMh, d);

        int[][] z\_MCh = Data.multiplyMatrixByNumber(MC\_H, zi);

        int[][] MR\_H = Data.addMatrices(MB\_MC\_MMh\_d, z\_MCh);

        Data.insertSubmatrixIntoMatrix(MR, MR\_H, start);

    }

    public static void printMatrix(int[][] matrix) {

        for (int i = 0; i < matrix.length; i++) {

            for (int j = 0; j < matrix[i].length; j++) {

                System.out.print(matrix[i][j] + " ");

            }

            System.out.println();

        }

    }

}

public class Thread1 extends Thread {

    private Data data;

    private int threadId;

    private int start;

    private int end;

    private int zi;

    public Thread1(int id, Data D) {

        data = D;

        threadId = id;

        start = (threadId - 1) \* data.H;

        end = threadId \* data.H;

    }

    private void fillData() {

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

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

                data.MB[i][j] = 1;

            }

        }

    }

    @Override

    public void run() {

        System.out.println("T" + threadId + " start");

        try {

            fillData();

            // сигнал про введення даних і чекати, щоб інші потоки ввели дані - бар'єр B1

            data.B1.await();

            // обчислення 1

            int[] Z\_H = Arrays.copyOfRange(data.Z, start, end);

            zi = Data.findMin(Z\_H);

            // доступ до спільного ресурсу - КД1 - атомік-змінна z

            data.z.updateAndGet(current -> Math.min(current, zi));

            // сигнал про завершення обчислення 2 - семафор S1

            data.S1.release(3);

            // чекати, щоб всі потоки виконали обчислення 2 - семафори S2, S3, S4

            data.S2.acquire();

            data.S3.acquire();

            data.S4.acquire();

            // копіювання zi = z - КД2 - критична секція CS1

            zi = data.CS1();

            // обчислення 3 і запис результату в MR

            data.calculateStep3(start, end, zi);

            // сигнал про завершення обчислення 3 - семафор S5

            data.S5.release();

        } catch (Exception e) {

            e.printStackTrace();

        } finally {

            System.out.println("T" + threadId + " finish");

        }

    }

}

public class Thread2 extends Thread {

    private Data data;

    private int threadId;

    private int start;

    private int end;

    private int zi;

    public Thread2(int id, Data D) {

        data = D;

        threadId = id;

        start = (threadId - 1) \* data.H;

        end = threadId \* data.H;

    }

    @Override

    public void run() {

        System.out.println("T" + threadId + " start");

        try {

            // чекати, щоб інші потоки ввели дані - бар'єр B1

            data.B1.await();

            // обчислення 1

            int[] Z\_H = Arrays.copyOfRange(data.Z, start, end);

            zi = Data.findMin(Z\_H);

            // доступ до спільного ресурсу - КД1 - атомік-змінна z

            data.z.updateAndGet(current -> Math.min(current, zi));

            // сигнал про завершення обчислення 2 - семафор S2

            data.S2.release(3);

            // чекати, щоб всі потоки виконали обчислення 2 - семафори S1, S3, S4

            data.S1.acquire();

            data.S3.acquire();

            data.S4.acquire();

            // копіювання zi = z - КД2 - критична секція CS1

            zi = data.CS1();

            // обчислення 3 і запис результату в MR

            data.calculateStep3(start, end, zi);

            // чекати, щоб інші потоки завершили обчислення 3 - семафор S5

            data.S5.acquire(3);

            // вивід MR як результату

            Data.printMatrix(data.MR);

        } catch (Exception e) {

            e.printStackTrace();

        } finally {

            System.out.println("T" + threadId + " finish");

        }

    }

}

public class Thread3 extends Thread {

    private Data data;

    private int threadId;

    private int start;

    private int end;

    private int zi;

    public Thread3(int id, Data D) {

        data = D;

        threadId = id;

        start = (threadId - 1) \* data.H;

        end = threadId \* data.H;

    }

    private void fillData() {

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

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

                data.MC[i][j] = 1;

            }

        }

    }

    @Override

    public void run() {

        System.out.println("T" + threadId + " start");

        try {

            fillData();

            // сигнал про введення даних і чекати, щоб інші потоки ввели дані - бар'єр B1

            data.B1.await();

            // обчислення 1

            int[] Z\_H = Arrays.copyOfRange(data.Z, start, end);

            zi = Data.findMin(Z\_H);

            // доступ до спільного ресурсу - КД1 - атомік-змінна z

            data.z.updateAndGet(current -> Math.min(current, zi));

            // сигнал про завершення обчислення 2 - семафор S3

            data.S3.release(3);

            // чекати, щоб всі потоки виконали обчислення 2 - семафори S1, S2, S4

            data.S1.acquire();

            data.S2.acquire();

            data.S4.acquire();

            // копіювання zi = z - КД2 - критична секція CS1

            zi = data.CS1();

            // обчислення 3 і запис результату в MR

            data.calculateStep3(start, end, zi);

            // сигнал про завершення обчислення 3 - семафор S5

            data.S5.release();

        } catch (Exception e) {

            e.printStackTrace();

        } finally {

            System.out.println("T" + threadId + " finish");

        }

    }

}

public class Thread4 extends Thread {

    private Data data;

    private int threadId;

    private int start;

    private int end;

    private int zi;

    public Thread4(int id, Data D) {

        data = D;

        threadId = id;

        start = (threadId - 1) \* data.H;

        end = threadId \* data.H;

    }

    private void fillData() {

        data.d = 1;

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

            data.Z[i] = 1;

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

                data.MM[i][j] = 1;

            }

        }

    }

    @Override

    public void run() {

        System.out.println("T" + threadId + " start");

        try {

            fillData();

            // сигнал про введення даних і чекати, щоб інші потоки ввели дані - бар'єр B1

            data.B1.await();

            // обчислення 1

            int[] Z\_H = Arrays.copyOfRange(data.Z, start, end);

            zi = Data.findMin(Z\_H);

            // доступ до спільного ресурсу - КД1 - атомік-змінна z

            data.z.updateAndGet(current -> Math.min(current, zi));

            // сигнал про завершення обчислення 2 - семафор S4

            data.S4.release(3);

            // чекати, щоб всі потоки виконали обчислення 2 - семафори S1, S2, S3

            data.S1.acquire();

            data.S2.acquire();

            data.S3.acquire();

            // копіювання zi = z - КД2 - критична секція CS1

            zi = data.CS1();

            // обчислення 3 і запис результату в MR

            data.calculateStep3(start, end, zi);

            // сигнал про завершення обчислення 3 - семафор S5

            data.S5.release();

        } catch (Exception e) {

            e.printStackTrace();

        } finally {

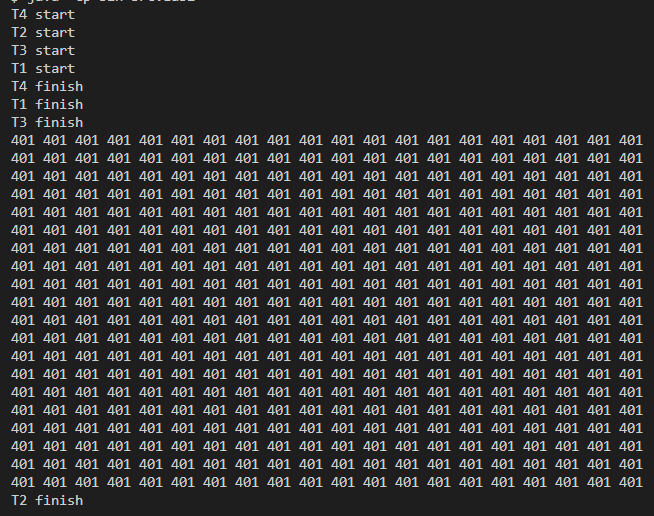
            System.out.println("T" + threadId + " finish");

        }

    }

}

На мал. 2 надано скриншот результату роботи програми для N = 20.



Мал. 2 Вивід програми

**Висновки:**

1. для реалізації багатопоточної програми використано клас Thread, який дозволяє створювати та керувати потоками в мові Java;
2. розроблено паралельний математичний алгоритм, алгоритм кожної з задач та побудовано структурну схему їх взаємодії;
3. в програмі представлено роботу всіх вказаних засобів організації синхронізації, що реалізовані в мові Java, а саме семафорів, критичних секцій, атомік-змінних, бар’єрів.