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

Звіт до лабораторної роботи №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-13
Мартинюк Марія Павлівна
номер у списку групи 13

Перевірив:
Корочкін О. В.

Київ 2024

Мета роботи: розробка програми для ПКС зі СП
Мова програмування: Java
Засоби організації взаємодії процесів: семафори, критичні секції, атомік змінні(типи), бар'єри

Вхідні дані:

- комп'ютерна система зі спільною пам'яттю включає чотири процесори і три пристрої введення-виведення (рис. 1.1);
- математичне завдання згідно **варіанту №16**:
  A = p * (sort(d * B + Z * MM) * (MX * MT)) + (B * Z) * Z,
  де A, B, Z - вектори розміру N; MM, MX, MT - матриці розміру N; p, d - скаляри;
- введення значень для MM, B, MX виконується у процесорі 1;
  введення значення скаляру p виконується у процесорі 3;
  введення значень для Z, MT, d виконується у процесорі 4;
  виведення результату A виконується у процесорі 4.

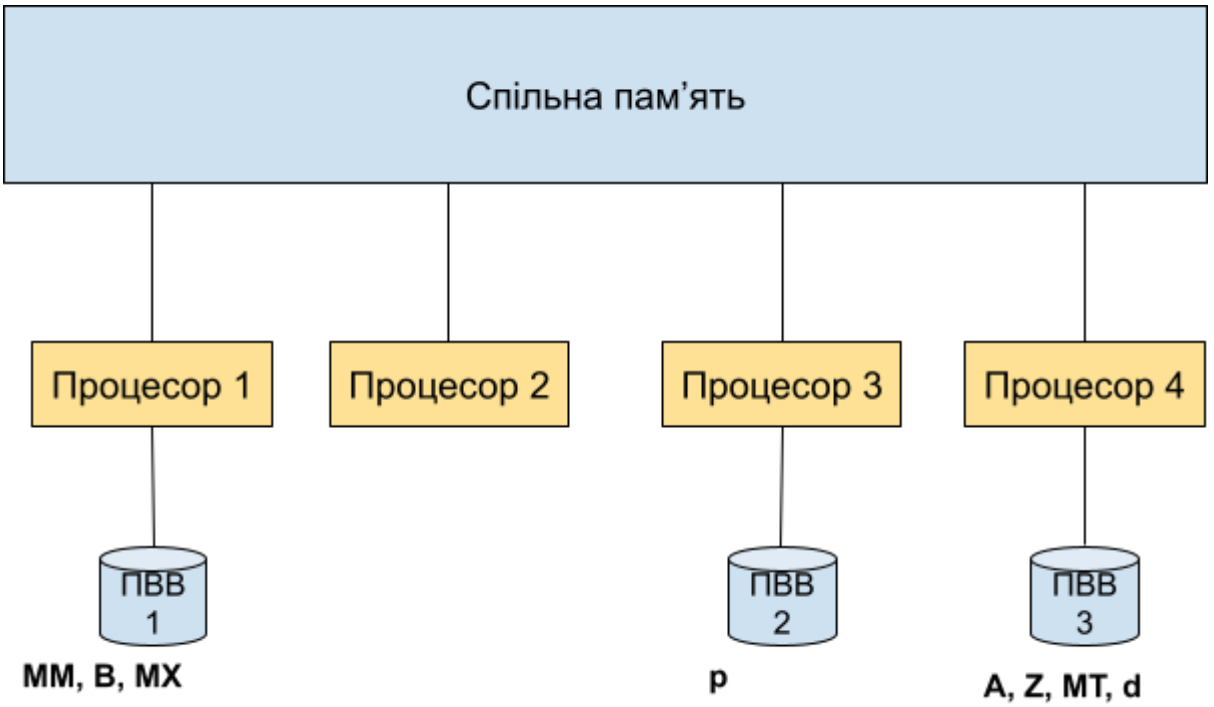

рис. 1.1 Структурна схема 4 процесорної системи

ВИКОНАННЯ РОБОТИ

Етап 1. Побудова паралельного алгоритму

A = p * (sort(d * B + Z * MM) * (MX * MT)) + (B * Z) * Z,

1. Xн = sort((d * Bн + Z * MMн)	Потоки T1-T4	CP: d, копія
2. X2н = sortzl(Xн, Xн)	Потоки T1 і T3	
3. X = sortzl(X2н, X2н)	Потік T1	
4. MAн = MX * MTн		
5. Dн = p * (X * MAн)		CP: p, копія
6. ai = (Bн * Zн), i = 1...P		
7. a = a + ai		CP: a
8. Aн = Dн + a * Zн		CP: a, копія

,де н = N/P частина вектору або матриці, N - розмір вектору/матриці, P = 4 - кількість процесорів (потоків).

Етап 2. Розробка алгоритмів потоків

Задача T1	**Точки синхронізації, КД**
1. Введення MM, B, MX	
2. Сигнал задачі T2, T3, T4 про введення MM, B, MX	S2-1 S3-1 S4-1
3. Чекати на введення p у задачі T3; Z, MT, d у задачі T4	W3-1 W4-1
4. Копія d1 = d	КД1
5. Обчислення1 Xн = sort((d1 * Bн + Z * MMн))	
6. Чекати на завершення обчислень Xн в T2	W2-2
7. Обчислення2 X2н = sortzl(Xн, Xн)	

8. Чекати на завершення обчислень X2н в T3	W3-3
9. Обчислення3 X = sortzl(X2н, X2н)	
10. Сигнал T2, T3, T4 про завершення обчислень X	S2-2 S3-2 S4-2
11. Обчислення4 MAн = MX * MTн	
12. Копія p1 = p	КД2
13. Обчислення5 Dн = p1 * (X * MAн)	
14. Обчислення6 a1 = (Bн * Zн)	
15. Обчислення7 a = a + a1	КД3
16. Сигнал T2, T3, T4 про завершення обчислень a	S2-3 S3-3 S4-3
17. Чекати на завершення обчислень a в T2, T3, T4	W2-4 W3-4 W4-4
18. Копія a1 = a	КД4
19. Обчислення8 Aн = Dн + a1 * Zн	
20. Сигнал T4 про завершення обчислень Aн	S4-4

Задача T2	**Точки синхронізації**
1. Чекати на введення MM, B, MX у задачі T1; p у задачі T3; Z, MT, d у задачі T4	W1-1 W3-1 W4-1
2. Копія d2 = d	КД1
3. Обчислення1 Xн = sort((d2 * Bн + Z * MMн)	

4. Сигнал T1 про завершення обчислень Xн	S1-1
5. Чекати на завершення обчислень X в T1	W1-2
6. Обчислення2 MAн = MX * MTн	
7. Копія p2 = p	КД2
8. Обчислення3 Dн = p2 * (X * MAн)	
9. Обчислення4 a2 = (Bн * Zн)	
10. Обчислення5 a = a + a2	КД3
11. Сигнал T1, T3, T4 про завершення обчислень a	S1-2 S3-2 S4-2
12. Чекати на завершення обчислень a в T1, T3, T4	W1-3 W3-3 W4-3
13. Копія a2 = a	КД4
14. Обчислення6 Aн = Dн + a2 * Zн	
15. Сигнал T4 про завершення обчислень Aн	S4-3

Задача T3	**Точки синхронізації**
1. Введення p	
2. Сигнал задачі T1, T2, T4 про введення p	S1-1 S2-1 S4-1
3. Чекати на введення MM, B, MX у задачі T1; Z, MT, d у задачі T4	W1-1 W4-1
4. Копія d3 = d	КД1
5. Обчислення1 Xн = sort((d3 * Bн + Z * MMн)	

6. Чекати на завершення обчислень Xн в T4	W4-2
7. Обчислення2 X2н = sortzl(Xн, Xн)	
8. Сигнал T1 про завершення обчислень X2н	S1-2
9. Чекати на завершення обчислень X в T1	W1-3
10. Обчислення3 MAн = MX * MTн	
11. Копія p3 = p	
12. Обчислення4 Dн = p3 * (X * MAн)	КД2
13. Обчислення5 a3 = (Bн * Zн)	
14. Обчислення6 a = a + a3	КД3
15. Сигнал T1, T2, T4 про завершення обчислень a	S1-3 S2-3 S4-3
16. Чекати на завершення обчислень a в T1, T2, T4	W1-4 W2-4 W4-4
17. Копія a3 = a	КД4
18. Обчислення7 Aн = Dн + a3 * Zн	
19. Сигнал T4 про завершення обчислень Aн	S4-4

Задача T4	**Точки синхронізації**
1. Введення A, Z, MT, d	
2. Сигнал задачі T1, T2, T3 про введення A, Z, MT, d	S1-1 S2-1 S3-1

3. Чекати на введення MM, B, MX у задачі T1; p у задачі T3	W1-1 W3-1
4. Копія d4 = d	КД1
5. Обчислення1 Xн = sort((d4 * Bн + Z * MMн)	
6. Сигнал T3 про завершення обчислень Xн	S3-2
7. Чекати на завершення обчислень X в T1	W1-2
8. Обчислення2 MAн = MX * MTн	
11. Копія p4 = p	КД2
12. Обчислення3 Dн = p4 * (X * MAн)	
13. Обчислення4 a4 = (Bн * Zн)	
14. Обчислення5 a = a + a4	КД3
15. Сигнал T1, T2, T3 про завершення обчислень a	S1-3 S2-3 S3-3
16. Чекати на завершення обчислень a в T1, T2, T3	W1-3 W2-3 W3-3
17. Копія a4 = a	КД4
18. Обчислення6 Aн = Dн + a4 * Zн	
19. Чекати на завершення обчислень Aн в T1, T2, T3	W1-4 W2-4 W3-4
20. Вивід Aн	

Етап 3. Розробка схеми взаємодії задач

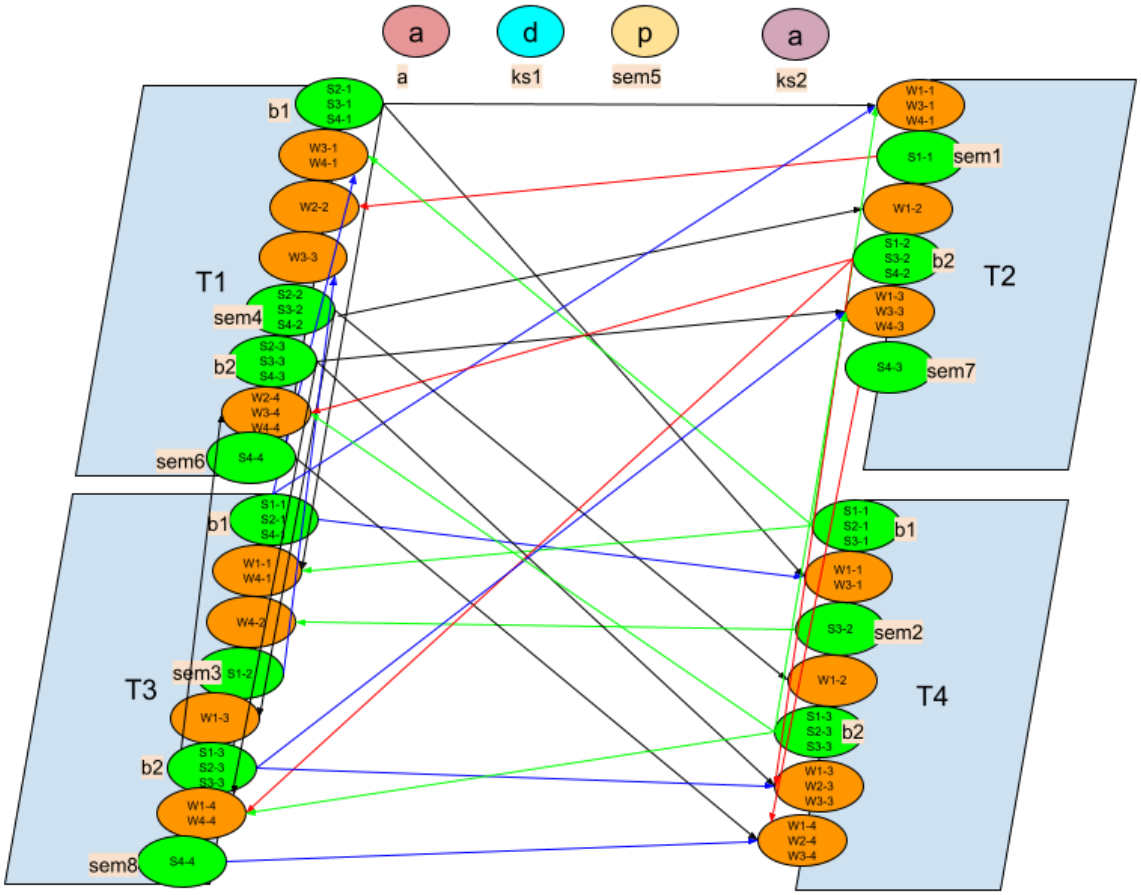

Рис.1.2 Структурна схема взаємодії задач

На структурній схемі взаємодії задач (рис.1.2) наведено такі засоби синхронізації потоків:

- Семафори:
  - **sem1, sem2 -** для синхронізації завершення обчислень Xн в потоках T2, T4 відповідно;
  - **sem3 -** для синхронізації завершення обчислень X2н в потоці T3
  - **sem4 -** для синхронізації завершення обчислень X в потоці T1;
  - **sem5 -** для захисту КД2 при копіюванні змінної p в потоках T1, T2, T3, T4 відповідно;
  - **sem6, sem7, sem8 -** для синхронізації завершення обчислень Ан в потоках T1, T2, T3 відповідно і подальшого виведення результату в потоці T4;

- Atomic змінні (типи): **a** - для захисту КД3 при обрахунку значення a в потоках T1, T2, T3, T4 відповідно;

- Критичні секції (lock, synchronized):
  - **ks1** - для захисту КД1 при копіюванні змінної d в потоках T1, T2, T3, T4 відповідно;
  - **ks2** - для захисту КД4 при копіюванні змінної a в потоках T1, T2, T3, T4 відповідно;

- Бар'єри:
  - **b1** - для синхронізації завершення введення даних в потоках T1, T3, T4 відповідно
  - **b2** - для синхронізації завершення обчислень a в потоках T1, T2, T3, T4 відповідно

Структурна схема взаємодії базується на розробленому в етапі 2 алгоритмі взаємодії потоків та дозволяє наочно проконтролювати зв'язок у багатопотоковій програмі, використовуючи при цьому вище описані засоби синхронізації.

Етап 4. Розробка програми

Програма складається з основного класу Lab2, допоміжного класу з методами для обчислення математичних виразів Data та 4 класів для кожного з 4 потоків: T1, T2, T3, T4.

Lab2.java

```
//Дисципліна "Програмне забезпечення високопродуктивних комп'ютерних систем"
//Лабораторна робота ЛР2 Варіант 16
//A = p * (sort(d * B + Z * MM) * (MX * MT)) + (B * Z) * Z
//Мартинюк Марія Павлівна
//Дата 24.03.2024

import java.util.concurrent.Semaphore;

public class Lab2 {
   static int N = 8;
   static int P = 4;
```

```
   static int H = N/P;

   public static void main(String[] args) throws
InterruptedException {
        double start, end, totalTime;
        start = System.currentTimeMillis();

        Semaphore sem1 = new Semaphore(0);
        Semaphore sem2 = new Semaphore(0);
        Semaphore sem3 = new Semaphore(0);
        Semaphore sem4 = new Semaphore(0);
        Semaphore sem6 = new Semaphore(0);
        Semaphore sem7 = new Semaphore(0);
        Semaphore sem8 = new Semaphore(0);

        T1 T1 = new T1(N, H, sem1, sem3, sem4, sem6);
        T2 T2 = new T2(H, sem1, sem4, sem7);
        T3 T3 = new T3(H, sem2, sem3, sem4, sem8);
        T4 T4 = new T4(N, H, sem2, sem4, sem6, sem7, sem8);

        //threads start
        T1.start();
        T2.start();
        T3.start();
        T4.start();

        try {
            T1.join();
            T2.join();
            T3.join();
            T4.join();
        } catch (InterruptedException e) {
            System.out.println(e.getMessage());
        }

        end = System.currentTimeMillis();;
        totalTime = end - start;

        System.out.println("TIME " + totalTime + "ms");
   }
}
```

Data.java

```
import java.util.Arrays;
import java.util.concurrent.CyclicBarrier;
import java.util.concurrent.Semaphore;
import java.util.concurrent.atomic.AtomicInteger;
```

```
import java.util.concurrent.locks.ReentrantLock;

public class Data {
    static Semaphore sem5 = new Semaphore(1);
    static CyclicBarrier b1 = new CyclicBarrier(4);
    static CyclicBarrier b2 = new CyclicBarrier(4);
    static final Object ks2 = new Object();
    static final ReentrantLock ks1 = new ReentrantLock();
    static AtomicInteger a = new AtomicInteger(0);

    //допоміжний метод для заповнення вектору одиницями
    public static int[] vectorInput(int N) {
        int[] vector = new int[N];
        Arrays.fill(vector, 1);
        return vector;
    }

    //допоміжний метод для заповнення матриці одиницями
    public static int[][] matrixInput(int N) {
        int[][] matrix = new int[N][N];
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                matrix[i][j] = 1;
            }
        }
        return matrix;
    }

    //допоміжний метод для розрахунку Xн = sort((d * Bн + Z * MMн)
    public static int[] XнCalculate(int[] Bн, int[] Z, int[][] MMн,
int d, int startRow, int endRow) {
        int[][] MMpart = new int[endRow - startRow][MMн[0].length];
        for (int i = startRow; i < endRow; i++) {
            MMpart[i - startRow] = Arrays.copyOf(MMн[i],
MMн[i].length);
        }

        int[] result = new int[Bн.length];

        for (int i = 0; i < Bн.length; i++) {
            result[i] = d * Bн[i];
        }

        for (int i = 0; i < MMpart.length; i++) {
            int rowSum = 0;
            for (int j = 0; j < Z.length; j++) {
                rowSum += Z[j] * MMpart[i][j];
            }
```

```
            result[i] += rowSum;
        }

        Arrays.sort(result);
        return result;
    }

    //допоміжний метод для розрахунку МАн = МХ * МТн
    public static int[][] multiplyMatricesPartialResult(int[][] MX,
int[][] МТн, int startRow, int endRow) {
        int[][] partialResult = new int[endRow -
startRow][МТн[0].length];
        for (int i = startRow; i < endRow; i++) {
            for (int j = 0; j < МТн[0].length; j++) {
                for (int k = 0; k < МТн.length; k++) {
                    partialResult[i - startRow][j] += MX[i][k] *
МТн[k][j];
                }
            }
        }
        return partialResult;
    }

    //допоміжний метод для розрахунку  Дн = р * (Х * МАн)
    public static int[] calculateДн(int[] X, int[][] МАн, int p) {
        int[] Дн = new int[МАн.length];

        for (int i = 0; i < X.length; i++) {
            X[i] *= p;
        }
        for (int i = 0; i < МАн.length; i++) {
            for (int j = 0; j < X.length; j++) {
                Дн[i] += X[j] * МАн[i][j];
            }
        }
        return Дн;
    }

    // допоміжний метод для розрахунку аі = Вн * Zн
    public static int vectorMultiply(int[] Вн, int[] Zн) {
        int result = 0;

        for (int i = 0; i < Вн.length; i++) {
            result += Вн[i] * Zн[i];
        }

        return result;
    }
```

```
    // допоміжний метод для розрахунку Ан = Дн + (а * Z)н
    public static int[] calculateАн(int[] Дн, int a, int[] Zн) {
        int[] Ан = new int[Дн.length];

        for (int i = 0; i < Zн.length; i++) {
            Zн[i] *= a;
        }

        for (int i = 0; i < Дн.length; i++) {
            Ан[i] = Дн[i] + Zн[i];
        }
        return Ан;
    }
}
```

T1.java

```
import java.util.Arrays;
import java.util.concurrent.Semaphore;
import java.util.concurrent.BrokenBarrierException;

public class T1 extends Thread {
    int N, H;
    static int [] B;
    static int [][] MM;
    static int [][] MX;
    Semaphore sem1, sem3, sem4;
    Semaphore sem6;
    static int[] X, Ан;
    int threadIndex = 1;
    private int a1;

    public T1(int N, int H, Semaphore sem1, Semaphore sem3,
Semaphore sem4, Semaphore sem6) {
        this.N = N;
        this.H = H;
        this.sem1 = sem1;
        this.sem3 = sem3;
        this.sem4 = sem4;
        this.sem6 = sem6;
    }

    public void inputAndCalculate() {
        try {
            // Введення MM, B, MX
            B = Data.vectorInput(N);
            MM = Data.matrixInput(N);
```

```
/*                MM = new int[N][N];
                for (int i = 0; i < MM.length; i++) {
                    for (int j = 0; j < MM[i].length; j++) {
                        if (i == 1) {
                            MM[i][j] = 2; // Set the second column to
2
                        } else {
                            MM[i][j] = 1; // Set other elements to 1
                        }
                    }
                }*/
            MX = Data.matrixInput(N);

            // Чекати на введення p у задачі T3; Z, MT, d у задачі T4
            try {
                Data.b1.await();
            } catch (InterruptedException | BrokenBarrierException
e) {
                e.printStackTrace();
            }

            // ks1
            int d1;
            Data.ks1.lock();
                d1 = T4.d;// Копія d1 = d  КД1
            Data.ks1.unlock();

            // Обчислення1 Xн = sort((d1 * Bн + Z * MMн))
            int [] Bн = new int[H];
            int startIndex = (threadIndex * H) - H;
            System.arraycopy(T1.B, startIndex, Bн, 0, H);

            int [] Xн = Data.XнCalculate(Bн, T4.Z, MM, d1,
startIndex, startIndex + H);
            //System.out.println("X1 calcaulated Xn: " +
Arrays.toString(Xн));

            sem1.acquire(); // Чекати на завершення обчислень Xn у
задачі T2

            // Обчислення2 X2н = sortz1(Xн, Xн)
            int[] X2н = new int[Xн.length + T2.Xн.length];
            System.arraycopy(Xн, 0, X2н, 0, Xн.length);
            System.arraycopy(T2.Xн, 0, X2н, Xн.length,
T2.Xн.length);
            Arrays.sort(X2н);
```

```
            sem3.acquire(); // Чекати на завершення обчислень X2n у
задачі T3

            // Обчислення3 X = sortzl(X2н, Xн)
            X = new int[X2н.length + T3.X2н.length];
            System.arraycopy(X2н, 0, X, 0, X2н.length);
            System.arraycopy(T3.X2н, 0, X, X2н.length,
T3.X2н.length);
            Arrays.sort(X);

            sem4.release(3); // Сигнал T2, T3, T4 про завершення
обчислень X

            // Обчислення4 MAн = MX * MTн
            int [][] MAн = Data.multiplyMatricesPartialResult(MX,
T4.MT,startIndex, startIndex + H);

            Data.sem5.acquire();
               int p1 = T3.p; // Копія p1 = p  КД2
            Data.sem5.release();

            // Обчислення5 Dн = p1 * (X * MAн)
            int [] Dн = Data.calculateDн(X, MAн, p1);

            int [] Zн;

            // Обчислення6 a1 = ( Bн * Zн)
            Zн = new int[H];
            System.arraycopy(T4.Z, startIndex, Zн, 0, H);
            a1 = Data.vectorMultiply(Bн, Zн);

            // Обчислення7 a = a + a1
            Data.a.updateAndGet(a -> a + a1); // КД3

            try {
                Data.b2.await();
            } catch (InterruptedException | BrokenBarrierException
e) {
                e.printStackTrace();
            }

            // ks2
            synchronized (Data.ks2) {
                a1 = Data.a.intValue(); // Копія a1 = a  КД4
            }

            // Обчислення8 Aн = Dн + a1 * Zн
            Aн = Data.calculateAн(Dн, a1, Zн);
```

```
            sem6.release(); // Сигнал T4 про завершення обчислень Ан

        } catch (InterruptedException e) {
            System.out.println("ERROR: " + e.getMessage());
        }
    }


    @Override
    public void run(){
        System.out.println("T1 is started");
        inputAndCalculate();
        System.out.println("T1 is finished");
    }
}
```

T2.java

```
import java.util.Arrays;
import java.util.concurrent.BrokenBarrierException;
import java.util.concurrent.Semaphore;

public class T2 extends Thread {
    int H;
    static int [] Хн;
    static int [][] МАн;
    Semaphore sem1, sem4;
    Semaphore sem7;
    static int [] Ан;
    int threadIndex = 2;
    private int a2;

    public T2(int H, Semaphore sem1, Semaphore sem4, Semaphore
sem7) {
        this.H = H;
        this.sem1 = sem1;
        this.sem4 = sem4;
        this.sem7 = sem7;
    }

    public void calculate() {
        try {
            // Чекати на введення MM, B, MX у задачі T1; p у задачі
T3; Z, MT, d у задачі T4
            try {
                Data.b1.await();
            } catch (InterruptedException | BrokenBarrierException
e) {
```

```
                e.printStackTrace();
            }

            // ks1
            int d2;
            Data.ks1.lock();
                d2 = T4.d; // Копія d2 = d  КД1
            Data.ks1.unlock();

            // Обчислення1 Хн = sort((d1 * Вн + Z * ММн))
            int [] Вн = new int[H];
            int startIndex = (threadIndex * H) - H;
            System.arraycopy(T1.B, startIndex, Вн, 0, H);

            Хн = Data.ХнCalculate(Вн, T4.Z, T1.MM, d2, startIndex,
startIndex + H);
            //System.out.println("X2 calcaulated Xn: " +
Arrays.toString(Хн));

            sem1.release(); // Сигнал T1 завершення обчислень Xn

            sem4.acquire(); // Чекати на завершення обчислень X у
задачі T1

            // Обчислення2 X2н = sortzl(Хн, Хн)
            МАн = Data.multiplyMatricesPartialResult(T1.MX,
T4.MT,startIndex, startIndex + H);

            Data.sem5.acquire();
            int p2 = T3.p; // Копія p2 = p  КД2
            Data.sem5.release();

            // Обчислення3 Dн = p2 * (X * МАн)
            int [] Dн = Data.calculateDH(T1.X, МАн, p2);

            // Обчислення4 a2 = ( Вн * Zн)
            int[] Zн = new int[H];
            System.arraycopy(T4.Z, startIndex, Zн, 0, H);
            a2 = Data.vectorMultiply(Вн, Zн);

            // Обчислення5 a = a + a2
            Data.a.updateAndGet(a -> a + a2); // КД3

            try {
                Data.b2.await();
            } catch (InterruptedException | BrokenBarrierException
e) {
                e.printStackTrace();
```

```
            }

            // ks2
            synchronized (Data.ks2) {
                a2 = Data.a.intValue(); // Копія a2 = a  КД4
            }

            // Обчислення6 Ан = Dн + a2 * Zн
            Ан = Data.calculateАн(Dн, a2, Zн);

            sem7.release(); // Сигнал Т4 про завершення обчислень Ан

        } catch (InterruptedException e) {
            System.out.println("ERROR: " + e.getMessage());
        }
    }

    @Override
    public void run(){
        System.out.println("T2 is started");
        calculate();
        System.out.println("T2 is finished");
    }
}
```

T3.java

```
import java.util.Arrays;
import java.util.concurrent.BrokenBarrierException;
import java.util.concurrent.Semaphore;

public class T3 extends Thread {
    int H;
    static int p;
    static int [] X2н;
    static int [][] МАн;
    Semaphore sem2, sem3, sem4;
    Semaphore sem8;
    static int [] Ан;
    int threadIndex = 3;
    private int a3;

    public T3(int H, Semaphore sem2, Semaphore sem3, Semaphore
sem4, Semaphore sem8) {
        this.H = H;
        this.sem2 = sem2;
        this.sem3 = sem3;
        this.sem4 = sem4;
```

```
        this.sem8 = sem8;
    }

    public void inputAndCalculate() {
        try {
            // Введення p
            p = 1;

            try {
                Data.b1.await();
            } catch (InterruptedException | BrokenBarrierException
e) {
                e.printStackTrace();
            }

            // ks1
            int d3;
            Data.ks1.lock();
                d3 = T4.d; // Копія d3 = d  КД1
            Data.ks1.unlock();

            // Обчислення1 Хн = sort((d3 * Вн + Z * ММн)
            int [] Вн = new int[H];
            int startIndex = (threadIndex * H) - H;
            System.arraycopy(T1.B, startIndex, Вн, 0, H);

            int [] Хн = Data.ХнCalculate(Вн, T4.Z, T1.MM, d3,
startIndex, startIndex + H);
            //System.out.println("X3 calcaulated Xn: " +
Arrays.toString(Хн));

            sem2.acquire(); // Чекати на завершення обчислень Xn у
задачі T4

            // Обчислення2 X2н = sortzl(Хн, Хн)
            X2н = new int[Хн.length + T4.Хн.length];
            System.arraycopy(Хн, 0, X2н, 0, Хн.length);
            System.arraycopy(T4.Хн, 0, X2н, Хн.length,
T4.Хн.length);
            Arrays.sort(X2н);

            sem3.release(); // Сигнал T1 про завершення обчислень
X2n

            sem4.acquire(); // Чекати на завершення обчислень X у
задачі T1

            // Обчислення3 МАн = МХ * МТн
```

```
            MAн = Data.multiplyMatricesPartialResult(T1.MX,
T4.MT,startIndex, startIndex + H);

            Data.sem5.acquire();
            int p3 = T3.p; // Копія p3 = p  КД2
            Data.sem5.release();

            // Обчислення4 Dн = p3 * (X * MAн)
            int[] Dн = Data.calculateDн(T1.X, MAн, p3);

            // Обчислення5 a3 = ( Bн * Zн)
            int[] Zн;
            Zн = new int[H];
            System.arraycopy(T4.Z, startIndex, Zн, 0, H);
            a3 = Data.vectorMultiply(Bн, Zн);

            // Обчислення6 a = a + a3
            Data.a.updateAndGet(a -> a + a3); // КД3

            try {
                Data.b2.await();
            } catch (InterruptedException | BrokenBarrierException
e) {
                e.printStackTrace();
            }

            // ks2
            synchronized (Data.ks2) {
                a3 = Data.a.intValue(); // Копія a3 = a  КД4
            }

            // Обчислення7 Aн = Dн + a3 * Zн
            Aн = Data.calculateAн(Dн, a3, Zн);

            sem8.release(); // Сигнал T4 про завершення обчислень Aн

        } catch (InterruptedException e) {
            System.out.println("ERROR: " + e.getMessage());
        }
    }

    @Override
    public void run() {
        System.out.println("T3 is started");
        inputAndCalculate();
        System.out.println("T3 is finished");
    }
}
```

T4.java

```
import java.util.Arrays;
import java.util.concurrent.BrokenBarrierException;
import java.util.concurrent.Semaphore;

public class T4 extends Thread {
   public static int [] Xн;
   int N, H;
   static int[] Z;
   static int[][] MT;
   static int d;
   Semaphore sem2, sem4;
   Semaphore sem6, sem7, sem8;
   static int[][] MAн;
   int threadIndex = 4;
   private int a4;

   public T4(int N, int H, Semaphore sem2, Semaphore sem4,
Semaphore sem6, Semaphore sem7, Semaphore sem8) {
       this.N = N;
       this.H = H;
       this.sem2 = sem2;
       this.sem4 = sem4;
       this.sem6 = sem6;
       this.sem7 = sem7;
       this.sem8 = sem8;
   }

   public void inputAndCalculate() {
       try {
           // Введення A, Z, MT, d
           d = 1;
           Z = Data.vectorInput(N);
           MT = Data.matrixInput(N);

           // Чекати на введення MM, B, MX у задачі T1; p у задачі
T3
           try {
               Data.b1.await();
           } catch (InterruptedException | BrokenBarrierException
e) {
               e.printStackTrace();
           }

           // ks1
           int d4;
           Data.ks1.lock();
```

```
                d4 = d; // Копія d4 = d  КД1
            Data.ks1.unlock();

            // Обчислення Xн = sort((d1 * Bн + Z * MMн))
            int [] Bн = new int[H];
            int startIndex = (threadIndex * H) - H;
            System.arraycopy(T1.B, startIndex, Bн, 0, H);

            Xн = Data.XнCalculate(Bн, T4.Z, T1.MM, d4, startIndex,
startIndex + H);
            //System.out.println("X4 calcaulated Xn: " +
Arrays.toString(Xн));

            sem2.release(); // Сигнал T3 завершення обчислень Xn

            sem4.acquire(); // Чекати на завершення обчислень X у
задачі T1

            // Обчислення2 MAн = MX * MTн
            MAн = Data.multiplyMatricesPartialResult(T1.MX,
T4.MT,startIndex, startIndex + H);

            Data.sem5.acquire();
            int p4 = T3.p; // Копія p4 = p  КД2
            Data.sem5.release();

            // Обчислення3 Dн = p4 * (X * MAн)
            int [] Dн = Data.calculateDн(T1.X, MAн, p4);

            // Обчислення4 a4 = ( Bн * Zн)
            int[] Zн = new int[H];
            System.arraycopy(T4.Z, startIndex, Zн, 0, H);
            a4 = Data.vectorMultiply(Bн, Zн);

            // Обчислення5 a = a + a4
            Data.a.updateAndGet(a -> a + a4); // КД3

            try {
                Data.b2.await();
            } catch (InterruptedException | BrokenBarrierException
e) {
                e.printStackTrace();
            }

            // ks2
            synchronized (Data.ks2) {
                a4 = Data.a.intValue(); // Копія a4 = a  КД4
            }
```

```
            // Обчислення6 Ан = Dн + a_4 * Zн
            int [] Ан = Data.calculateAн(Dн, a4, Zн);

            sem6.acquire(); // Чекати на завершення обчислень Ан у
задачі T1
            sem7.acquire(); // Чекати на завершення обчислень Ан у
задачі T2
            sem8.acquire(); // Чекати на завершення обчислень Ан у
задачі T3

            // Вивід Ан
            int totalLength = T1.Ан.length + T2.Ан.length +
T3.Ан.length + Ан.length;
            int[] A = new int[totalLength];
            System.arraycopy(T1.Ан, 0, A, 0, T1.Ан.length);
            System.arraycopy(T2.Ан, 0, A, T1.Ан.length,
T2.Ан.length);
            System.arraycopy(T3.Ан, 0, A, T1.Ан.length +
T2.Ан.length, T3.Ан.length);
            System.arraycopy(Ан, 0, A, T1.Ан.length + T2.Ан.length +
T3.Ан.length, Ан.length);

            System.out.println("A = " + Arrays.toString(A));

        } catch (InterruptedException e) {
            System.out.println("ERROR: " + e.getMessage());
        }
    }

    @Override
    public void run() {
        System.out.println("T4 is started");
        inputAndCalculate();
        System.out.println("T4 is finished");
    }
}
```

Результат виконання:

На рис.1.3 наведений результат виконання багатопотокової програми при N = 8, P = 4. Значення усіх елементів використовуваних матриць та векторів рівні 1.

```
T1 is started
T2 is started
T3 is started
T4 is started
T2 is finished
T1 is finished
T3 is finished
A = [584, 584, 584, 584, 584, 584, 584, 584]
T4 is finished
TIME 11.0ms
```

рис.1.3 Результат виконання розробленої програми №1

Задля впевненості у правильності розрахунків через використання у математичній формулі операції сортування, було проведено додаткове тестування програми. При тих же значеннях N = 8, P = 4 елементи 2 рядка матриці MM мали значення 2. Усі інші елементи, а також елементи інших використовуваних векторів та матриць мали значення 1. Перевірка коректності розрахунків відбувалась через виведення проміжних результатів обчислення сортування у консоль. На рис.1.4 відображено результат тестування, що дозволяє переконатись у правильності виконаних обчислень.

```
T1 is started
T2 is started
T3 is started
T4 is started
X1 calcaulated Xn: [9, 17]
X3 calcaulated Xn: [9, 9]
X2 calcaulated Xn: [9, 9]
X4 calcaulated Xn: [9, 9]
T1 is finished
T3 is finished
T2 is finished
A = [648, 648, 648, 648, 648, 648, 648, 648]
T4 is finished
TIME 13.0ms
```

рис.1.4 Результат виконання розробленої програми №2

Дослідження завантаженості ядер процесору:

Варто зазначити, що конкретно мій комп’ютер містить 8 логічних ядер та 4 фізичних. Нижче наведений рис.1.4 демонструє, що при використанні 4 ядер процесору розподілення обчислювальних можливостей відбувається рівномірно між кожним з них. Поєднуючи цю інформацію із раніше отриманими остаточними значеннями обрахунків, можна зробити висновок, що чотирьох потокова програма написана коректно.

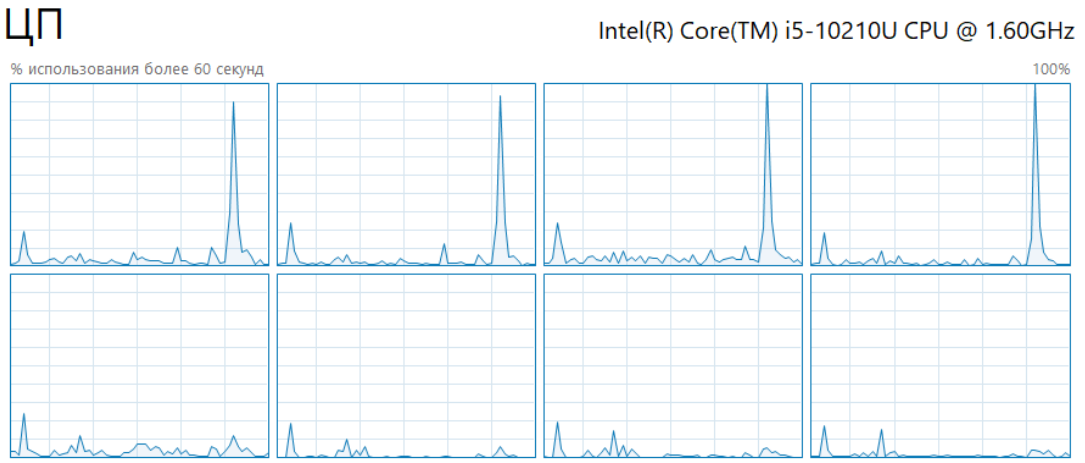

рис. 1.4 Графіки завантаженості ядер процесору

Додатково було обраховано коефіцієнт прискорення при запуску програми на 4 ядрах та на 1 ядрі при N = 8: Кп = 27 / 7 = 3,86. Тобто при використанні 4 ядер замість 1, програма завершується у 3,86 рази швидше.

Висновок:

Виконавши лабораторну роботу було побудовано та розроблено паралельний алгоритм для обчислення математичної задачі згідно варіанту 16 Додатка В для комп’ютерної системи зі спільною пам’яттю, що включає 4 процесори і 3 пристрої введення-виведення. За отриманим алгоритмом стало зрозуміло, що критичні ділянки з використання спільних ресурсів a, d, p, a потребують захистів, механізм яких був визначений після створення схеми взаємодії.

Задля реалізації завдання було обрано мову програмування Java та такі механізми синхронізації, як семафори, atomic змінні (типи), критичні секції (lock, synchronized) та бар’єри.

Завдяки додатковому тестуванню програми, можна зробити висновок, що розрахунок остаточного значення відбувається коректно. Аналіз завантаженості ядер процесору при запуску програми підтвердив рівномірне розподілення навантаження між задіяними ядрами, що свідчить про успішну оптимізацію паралельних обчислень. Кп = 3,86, що також сповіщає про пришвидшення виконання у разі використання 4 ядер процесора замість 1.