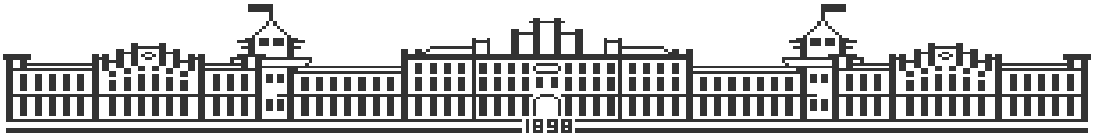


МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ. І.СІКОРСЬКОГО»
КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №4

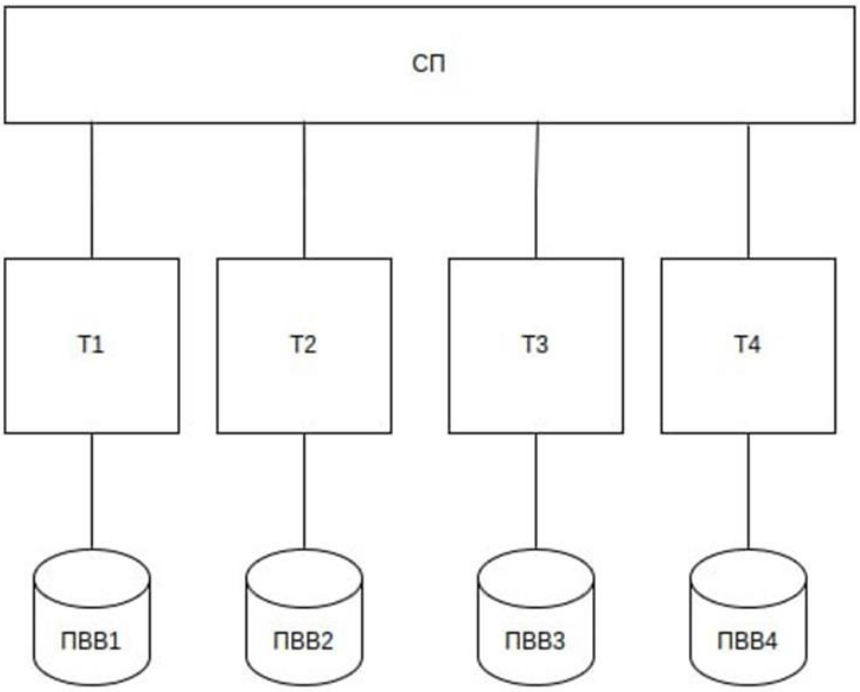
з дисципліни «Паралельне програмування»
на тему: «Мова Java. Монітори»

Виконав:
студент 3-го курсу
факультету ІОТ
групи ІО-14
Лупащенко А. А.

Перевірив:
доц.
Корочкін О. В.

Київ 2023

Варіант 27
MO = MB*(MC*MM)*d + min(Z)*MC



MB, MC MO - Z, d, MM

Етап 1. Паралельний алгоритм:

1. a_i = min(Zн) i = 1…4
2. a = mix(a,a_i) CP: a
3. MO_H = MB*(MC*MMн)*d + a*MCн CP: a, d

CP які змінюються: a
CP копіюємо: a, d

Етап 2. Опис алгоритму потоків та точки синхронізації:
T1:

1. Введення MB, MC
2. Сигнал Потокам T2,T3,T4 про введення MB, MC - - S(234,1)
3. Чекати на введення даних в потоках T2,T3,T4 - - W(234,1)
4. Копіювання d, d1 = d - - Критична ділянка
5. Обчислення a1 = min(Zн) - - Критична ділянка
6. Обчислення a = min(a, a1) - - Критична ділянка
7. Надсилаємо сигнал про завершення обчислень a – S(234,2)
8. Чекаємо на завершення обчислень a у інших потоках - W(234,2)
9. Копіювання a, a1 = a - - Критична ділянка
10. Обчислення MO_H = MB*(MC*MMн)*d + a*MCн
11. Сигнал T2 про завершення обчислень - - S(2,3)

T2:

1. Чекати на введення даних в потоках T1,T3,T4 - - W(134,1)
2. Копіювання d, d2 = d - - Критична ділянка
3. Обчислення a2 = min(Zн) - - Критична ділянка

4. Обчислення a = min(a, a2) - - Критична ділянка
5. Надсилаємо сигнал про завершення обчислень a – S(134,1)
6. Чекаємо на завершення обчислень a у інших потоках - W(134,2)
7. Копіювання a, a2 = a - - Критична ділянка
8. Обчислення MO_H = MB*(MC*MMн)*d + a*MCн
9. Чекаємо на завершення обчислень в T1,T3,T4 - -W(134,3)
10. Виведення результату MO

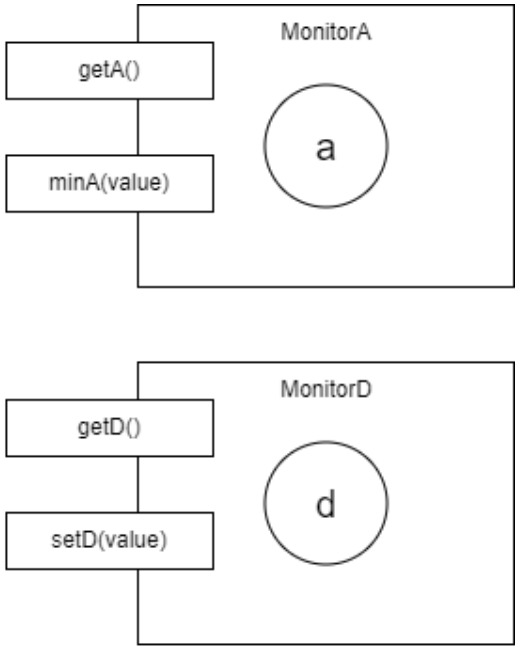
T3:

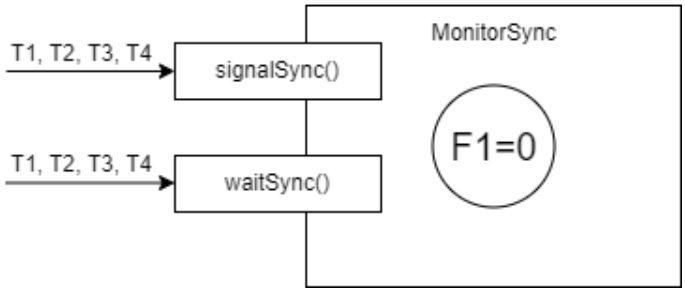
1. Чекати на введення даних в потоках T1,T2,T4 - - W(124,1)
2. Копіювання d, d3 = d - - Критична ділянка
3. Обчислення a3 = min(Zн) - - Критична ділянка
4. Обчислення a = min(a, a3) - - Критична ділянка
5. Надсилаємо сигнал про завершення обчислень a – S(124,1)
6. Чекаємо на завершення обчислень a у інших потоках - W(124,2)
7. Копіювання a, a3 = a - - Критична ділянка
8. Обчислення MO_H = MB*(MC*MMн)*d + a*MCн
9. Сигнал T2 про завершення обчислень - - S(2,2)

T4:

1. Введення Z, d, MM
2. Сигнал Потокам T1,T2,T3 про введення Z, d, MM - - S(123,1)
3. Чекати на введення даних в потоках T1,T2,T3 - - W(123,1)
4. Копіювання d, d4 = d - - Критична ділянка
5. Обчислення a4 = min(Zн) - - Критична ділянка
6. Обчислення a = min(a, a4) - - Критична ділянка
7. Надсилаємо сигнал про завершення обчислень a – S(123,2)
8. Чекаємо на завершення обчислень a у інших потоках - W(123,2)
9. Копіювання a, a4 = a - - Критична ділянка
10. Обчислення MO_H = MB*(MC*MMн)*d + a*MCн
11. Сигнал T2 про завершення обчислень - - S(2,3)

Етап 3. Схема взаємодії потоків:





Етап 4. Код програми:

```
import java.util.Arrays;

class Data {
        public static final int n = 4;
        public static final int P = 4;
        public static final int H = n / P;

        // Ініціалізація даних (MD, MB, MA, C, E, d)
        public static long[][] MB = new long[n][n];
        public static long[][] MO = new long[n][n];
        public static long[][] MC = new long[n][n];
        public static long[][] MM = new long[n][n];
        public static long[] Z = new long[n];

        // Метод для генерації матриці з одиницями по головній діагоналі
        public static long[][] generateMatrix(int size, int values) {
                long[][] matrix = new long[size][size];

                for (int i = 0; i < size; i++) {
                        for (int j = 0; j < size; j++) {
                                if (i==j) {
                                        matrix[i][j] = 1;
                                        continue;
                                }
                                matrix[i][j] = values;
                        }
                }

                return matrix;
        }

        // Метод для генерації вектора з послідовними значеннями
        public static long[] generateVector(int from, int count) {
                int size = count;
                long[] vector = new long[size];

                for (int i = 0; i < size; i++) {
                        vector[i] = i + from;
                }

                return vector;
        }

        // Метод для встановлення значень матриці з іншої матриці
        public static void setMatrixValues(long[][] targetMatrix, long[][] sourceMatrix, int
from, int to) {
                int rows = targetMatrix.length;

                for (int i = 0; i < rows; i++) {
                        for (int j = 0; j < to-from; j++) {
                                targetMatrix[i][j+from] = sourceMatrix[i][j];
                        }
                }
        }
```

```
        // Метод для множення матриці на скаляр
        public static long[][] multiplyMatrixByScalar(long[][] matrix, long scalar, int
from, int to) {
                int rows = matrix.length;
                int cols = matrix[0].length;

                if (to == 0) {
                        to = cols;
                }

                long[][] result = new long[rows][to-from];

                for (int i = 0; i < rows; i++) {
                        for (int j = 0; j < to-from; j++) {
                                result[i][j] = matrix[i][j+from] * scalar;
                        }
                }

                return result;
        }

        // Метод для множення матриць
        public static long[][] matrixMultiplication(long[][] matrix1, long[][] matrix2, int
from, int to) {
                int rows1 = matrix1.length;
                int cols1 = matrix1[0].length;
                int rows2 = matrix2.length;
                int cols2 = matrix2[0].length;

                if (cols1 != rows2) {
                        throw new IllegalArgumentException("Несумісні розміри матриці");
                }

                if (to == 0) {
                        to = cols2;
                }

                long[][] result = new long[rows1][to-from];

                for (int i = 0; i < rows1; i++) {
                        for (int j = 0; j < to-from; j++) {
                                for (int k = 0; k < cols1; k++) {
                                        result[i][j] += matrix1[i][k] * matrix2[k][j+from];
                                }
                        }
                }

                return result;
        }

        // Метод для додавання матриць
        public static long[][] matrixAddition(long[][] matrixA, long[][] matrixB) {
                int rows = matrixA.length;
                int cols = matrixA[0].length;

                long[][] result = new long[rows][cols];

                for (int i = 0; i < rows; i++) {
                        for (int j = 0; j < cols; j++) {
                                result[i][j] = matrixA[i][j] + matrixB[i][j];
                        }
                }
```

```
                return result;
        }

        // Метод для пошуку максимального значення у векторі
        public static long findMin(long[] vector, int from, int to) {
                long min = vector[from];
                for (int i = 1; i < to-from; i++) {
                        if (vector[i+from] < min) {
                                min = vector[i+from];
                        }
                }
                return min;
        }

        // Метод обчислення Wh
        public static long[][] calcMOh(int p, int H, long a, int d) {
                //Обчислення MC*MM
                long[][] result1 = matrixMultiplication(MC, MM, H*(p-1), H*p);
                //Обчислення MB*(MC*MM)
                long[][] result2 = matrixMultiplication(MB, result1, 0, 0);
                //Обчислення MB*(MC*MM)*d
                long[][] result3 = multiplyMatrixByScalar(result2, d, 0, 0);
                //Обчислення a*MC
                long[][] result4 = multiplyMatrixByScalar(MC, a, H*(p-1), H*p);
                //Обчислення MB*(MC*MM)*d + a*MC
                return matrixAddition(result3, result4);
        }
}

class MonitorA {
        private long a;

        // Конструктор монітора
        public MonitorA(long initialValue) {
                this.a = initialValue;
        }

        // Метод для отримання значення a
        public synchronized long getA() {
                System.out.println("Ресурс a використовує " +
Thread.currentThread().getName());
                return this.a;
        }

        // Метод для зміни значення a та знаходження мінімуму з аргументом
        public synchronized void minA(long value) {
                System.out.println("Мінімальне значення a обислює " +
Thread.currentThread().getName());
                this.a = Math.min(a, value);
                System.err.println("Ресурс a " + a);
        }
}

class MonitorD {
        private int d;

        // Метод для отримання значення d
        public synchronized int getD() {
                System.out.println("Ресурс d використовує " +
Thread.currentThread().getName());
                return this.d;
        }

        // Метод для задання значення d
```

```
        public synchronized void setD(int value) {
                this.d = value;
        }
}

class MonitorSync {
        private int F1;
        private int P;

        // Конструктор монітора
        public MonitorSync() {
                this.F1 = 0;
                this.P = Data.P;
        }

        // Метод для збільшення лічильника сигналів
        public synchronized void signalSync() {
                F1++; // Збільшення лічильника
                // Перевірити чи дорівнює кількість сигналів потокам
                if (F1 == P) {
                        notifyAll(); // Викликати notifyAll(), щоб розбудити всі потоки
                        F1 = 0; // Скидання лічильника сигналів
                }
        }

        // Метод для очікування інших потоків
        public synchronized void waitSync() {
                // Перевірити чи є сигнали
                if (F1 != 0) {
                        try {
                                wait(); // Чекати, коли буде досягнуто значення числа потоків
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }
}

class T1 implements Runnable {
        private int p = 1;
        private MonitorA monA;
        private MonitorD monD;
        private MonitorSync monSync;

        public T1(MonitorA monA, MonitorD monD, MonitorSync monSync) {
    this.monA = monA;
                this.monD = monD;
                this.monSync = monSync;
  }

        @Override
        public void run() {
                System.out.println("T1: Початок роботи");
                // Введення MB, MC
                Data.MB = Data.generateMatrix(Data.n, 3);
                Data.MC = Data.generateMatrix(Data.n, 2);
                // Очікування введення у T2, T3, T4
                monSync.signalSync();
                monSync.waitSync();

                // Критичний розділ - Копіювання та обчислення
                int d1 = monD.getD();
                long a1 = Data.findMin(Data.Z, Data.H * (p - 1), Data.H * p);
                monA.minA(a1);
```

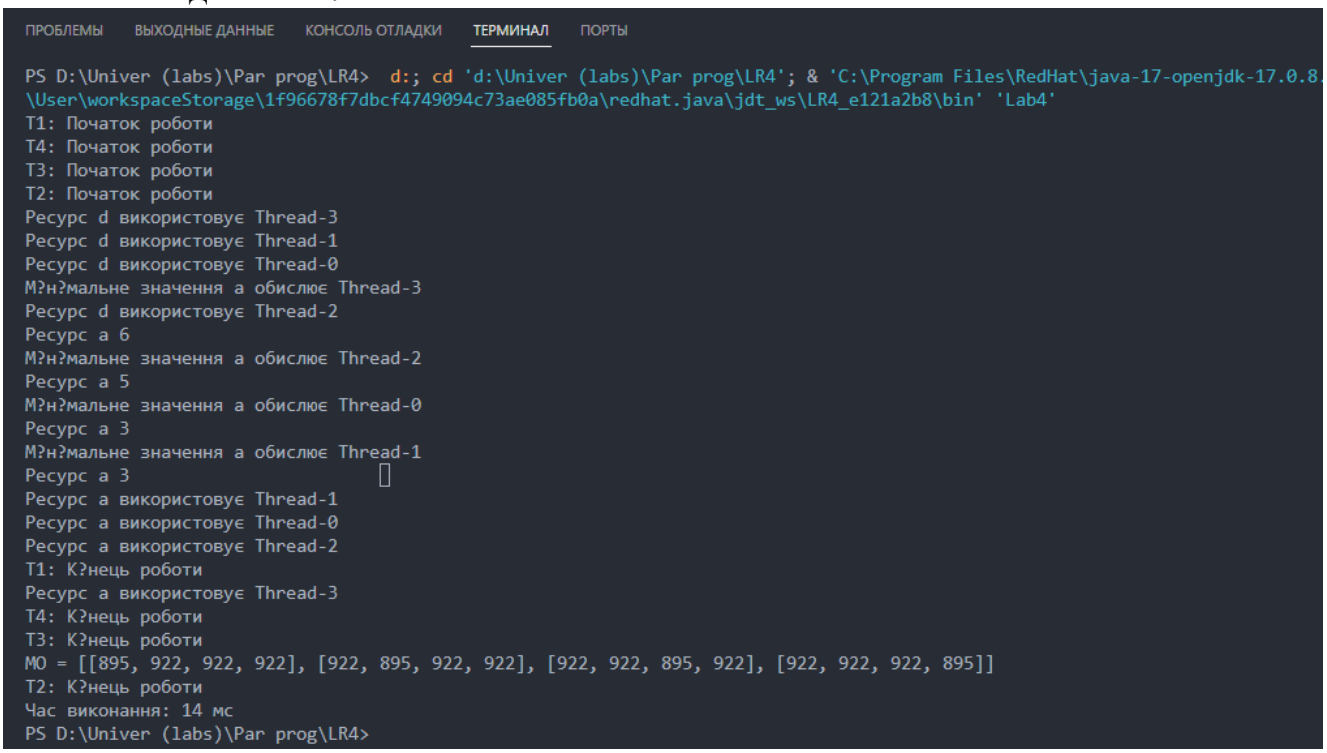
```
                // Сигнал про завершення обчислень іншим потокам
                monSync.signalSync();
                monSync.waitSync();

                // Критичний розділ - Копіювання
                a1 = monA.getA();

                // Обчислення MOh та запис до MO
                Data.setMatrixValues(Data.MO, Data.calcMOh(p, Data.H, a1, d1), Data.H * (p -
1), Data.H * p);
                // Сигнал про завершення T2
                monSync.signalSync();

                System.out.println("T1: Кінець роботи");
        }
}

class T2 implements Runnable {
        private int p = 2;
        private MonitorA monA;
        private MonitorD monD;
        private MonitorSync monSync;

        public T2(MonitorA monA, MonitorD monD, MonitorSync monSync) {
    this.monA = monA;
                this.monD = monD;
                this.monSync = monSync;
  }

        @Override
        public void run() {
                System.out.println("T2: Початок роботи");
                // Очікування введення у T1, T3, T4
                monSync.signalSync();
                monSync.waitSync();

                // Критичний розділ - Копіювання та обчислення
                int d2 = monD.getD();
                long a2 = Data.findMin(Data.Z, Data.H * (p - 1), Data.H * p);
                monA.minA(a2);

                // Сигнал про завершення обчислень іншим потокам
                monSync.signalSync();
                monSync.waitSync();

                // Критичний розділ - Копіювання
                a2 = monA.getA();

                // Обчислення MOh та запис до MO
                Data.setMatrixValues(Data.MO, Data.calcMOh(p, Data.H, a2, d2), Data.H * (p -
1), Data.H * p);
                // Сигнал про завершення та очікування інших потоків
                monSync.signalSync();
                monSync.waitSync();

                System.out.println("MO = " + Arrays.deepToString(Data.MO));

                System.out.println("T2: Кінець роботи");
        }
}

class T3 implements Runnable {
        private int p = 3;
```

```
        private MonitorA monA;
        private MonitorD monD;
        private MonitorSync monSync;

        public T3(MonitorA monA, MonitorD monD, MonitorSync monSync) {
    this.monA = monA;
                this.monD = monD;
                this.monSync = monSync;
  }

        @Override
        public void run() {
                System.out.println("T3: Початок роботи");
                // Очікування введення у T1, T2, T4
                monSync.signalSync();
                monSync.waitSync();

                // Критичний розділ - Копіювання та обчислення
                int d3 = monD.getD();
                long a3 = Data.findMin(Data.Z, Data.H * (p - 1), Data.H * p);
                monA.minA(a3);

                // Сигнал про завершення обчислень іншим потокам
                monSync.signalSync();
                monSync.waitSync();

                // Критичний розділ - Копіювання
                a3 = monA.getA();

                // Обчислення MOh та запис до MO
                Data.setMatrixValues(Data.MO, Data.calcMOh(p, Data.H, a3, d3), Data.H * (p -
1), Data.H * p);
                // Сигнал про завершення T2
                monSync.signalSync();

                System.out.println("T3: Кінець роботи");
        }
}

class T4 implements Runnable {
        private int p = 4;
        private MonitorA monA;
        private MonitorD monD;
        private MonitorSync monSync;

        public T4(MonitorA monA, MonitorD monD, MonitorSync monSync) {
    this.monA = monA;
                this.monD = monD;
                this.monSync = monSync;
  }

        @Override
        public void run() {
                System.out.println("T4: Початок роботи");
                // Введення Z, d, MM
                Data.Z = Data.generateVector(3, Data.n);
                monD.setD(4);
                Data.MM = Data.generateMatrix(Data.n, 4);
                // Очікування введення у T1, T2, T3
                monSync.signalSync();
                monSync.waitSync();

                // Критичний розділ - Копіювання та обчислення
                int d4 = monD.getD();
```

```
                long a4 = Data.findMin(Data.Z, Data.H * (p - 1), Data.H * p);
                monA.minA(a4);

                // Сигнал про завершення обчислень іншим потокам
                monSync.signalSync();
                monSync.waitSync();

                // Критичний розділ - Копіювання
                a4 = monA.getA();

                // Обчислення MOh та запис до MO
                Data.setMatrixValues(Data.MO, Data.calcMOh(p, Data.H, a4, d4), Data.H * (p -
1), Data.H * p);
                // Сигнал про завершення T2
                monSync.signalSync();

                System.out.println("T4: Кінець роботи");
        }
}

public class Lab4 {
        public static void main(String[] args) {
                MonitorA monitorA = new MonitorA(Long.MAX_VALUE);
                MonitorD monitorD = new MonitorD();
                MonitorSync monitorSync = new MonitorSync();
                // Створення потоків
                Thread t1 = new Thread(new T1(monitorA, monitorD, monitorSync));
                Thread t2 = new Thread(new T2(monitorA, monitorD, monitorSync));
                Thread t3 = new Thread(new T3(monitorA, monitorD, monitorSync));
                Thread t4 = new Thread(new T4(monitorA, monitorD, monitorSync));

                // Запуск потоків
                long startTime = System.currentTimeMillis();
                t1.start();
                t2.start();
                t3.start();
                t4.start();

                try {
                        // Очікування завершення роботи потоків
                        t1.join();
                        t2.join();
                        t3.join();
                        t4.join();
                } catch (InterruptedException e) {
                        // Обробка випадку переривання: вивід стеку виклику в консоль
                        e.printStackTrace();
                } finally {
                        long endTime = System.currentTimeMillis();
                        long executionTime = endTime - startTime;
                        System.out.println("Час виконання: " + executionTime + " мс");
                }
        }
}
```

Результати:

Виконання для n = 4:



Перевірка значень на вірність:

$$MC*MM = \begin{pmatrix} 1 & 2 & 2 & 2 \\ 2 & 1 & 2 & 2 \\ 2 & 2 & 1 & 2 \\ 2 & 2 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 4 & 4 & 4 \\ 4 & 1 & 4 & 4 \\ 4 & 4 & 1 & 4 \\ 4 & 4 & 4 & 1 \end{pmatrix} = \begin{pmatrix} 25 & 22 & 22 & 22 \\ 22 & 25 & 22 & 22 \\ 22 & 22 & 25 & 22 \\ 22 & 22 & 22 & 25 \end{pmatrix}$$

$$MB*(MC*MM) = \begin{pmatrix} 1 & 3 & 3 & 3 \\ 3 & 1 & 3 & 3 \\ 3 & 3 & 1 & 3 \\ 3 & 3 & 3 & 1 \end{pmatrix} \cdot \begin{pmatrix} 25 & 22 & 22 & 22 \\ 22 & 25 & 22 & 22 \\ 22 & 22 & 25 & 22 \\ 22 & 22 & 22 & 25 \end{pmatrix} = \begin{pmatrix} 223 & 229 & 229 & 229 \\ 229 & 223 & 229 & 229 \\ 229 & 229 & 223 & 229 \\ 229 & 229 & 229 & 223 \end{pmatrix}$$

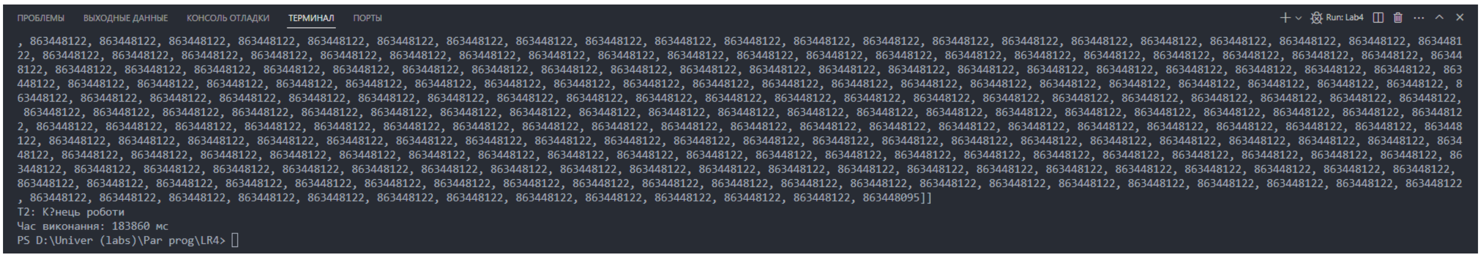
$$MB*(MC*MM)*d = 4 \cdot \begin{pmatrix} 223 & 229 & 229 & 229 \\ 229 & 223 & 229 & 229 \\ 229 & 229 & 223 & 229 \\ 229 & 229 & 229 & 223 \end{pmatrix} = \begin{pmatrix} 4 \cdot 223 & 4 \cdot 229 & 4 \cdot 229 & 4 \cdot 229 \\ 4 \cdot 229 & 4 \cdot 223 & 4 \cdot 229 & 4 \cdot 229 \\ 4 \cdot 229 & 4 \cdot 229 & 4 \cdot 223 & 4 \cdot 229 \\ 4 \cdot 229 & 4 \cdot 229 & 4 \cdot 229 & 4 \cdot 223 \end{pmatrix} = \begin{pmatrix} 892 & 916 & 916 & 916 \\ 916 & 892 & 916 & 916 \\ 916 & 916 & 892 & 916 \\ 916 & 916 & 916 & 892 \end{pmatrix}$$

$\min(Z) = \min(3\ 4\ 5\ 6) = 3$

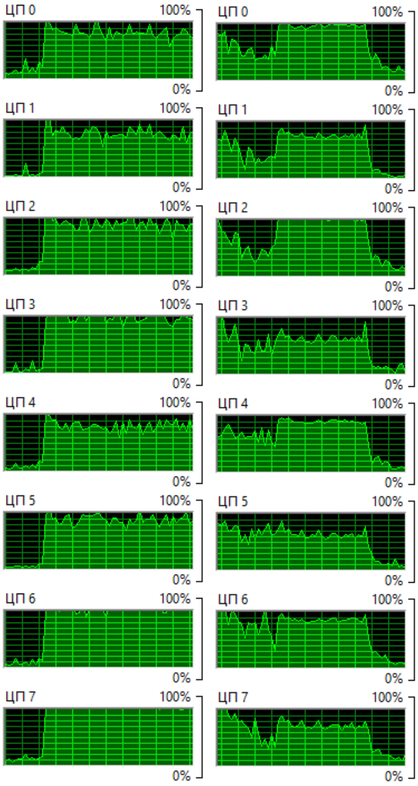
$$\min(Z)*MC = 3 \cdot \begin{pmatrix} 1 & 2 & 2 & 2 \\ 2 & 1 & 2 & 2 \\ 2 & 2 & 1 & 2 \\ 2 & 2 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 3 \cdot 1 & 3 \cdot 2 & 3 \cdot 2 & 3 \cdot 2 \\ 3 \cdot 2 & 3 \cdot 1 & 3 \cdot 2 & 3 \cdot 2 \\ 3 \cdot 2 & 3 \cdot 2 & 3 \cdot 1 & 3 \cdot 2 \\ 3 \cdot 2 & 3 \cdot 2 & 3 \cdot 2 & 3 \cdot 1 \end{pmatrix} = \begin{pmatrix} 3 & 6 & 6 & 6 \\ 6 & 3 & 6 & 6 \\ 6 & 6 & 3 & 6 \\ 6 & 6 & 6 & 3 \end{pmatrix}$$

$$MO = \begin{pmatrix} 892 & 916 & 916 & 916 \\ 916 & 892 & 916 & 916 \\ 916 & 916 & 892 & 916 \\ 916 & 916 & 916 & 892 \end{pmatrix} + \begin{pmatrix} 3 & 6 & 6 & 6 \\ 6 & 3 & 6 & 6 \\ 6 & 6 & 3 & 6 \\ 6 & 6 & 6 & 3 \end{pmatrix} = \begin{pmatrix} 892+3 & 916+6 & 916+6 & 916+6 \\ 916+6 & 892+3 & 916+6 & 916+6 \\ 916+6 & 916+6 & 892+3 & 916+6 \\ 916+6 & 916+6 & 916+6 & 892+3 \end{pmatrix} = \begin{pmatrix} 895 & 922 & 922 & 922 \\ 922 & 895 & 922 & 922 \\ 922 & 922 & 895 & 922 \\ 922 & 922 & 922 & 895 \end{pmatrix}$$

Виконання для n = 3000:



Навантаження ядер при n = 3000 становила майже 100%, просадка навантаження відбулась через закінчення обчислень та подальше навантаження через вивід результату:
Дані наведені з використанням Resource Monitor ОС Windows



Висновок:

1) Розроблено програму, що використовує паралельні потоки для вирішення задачі з обчисленням. Java була обрана як мова програмування через наявність необхідних засобів для вирішення заданої роботи.
2) Для забезпечення взаємного виключення та синхронізації використані монітори ресурсів та монітори синхронізації відповідно. Ці засоби дозволяють ефективно керувати доступом до спільних ресурсів та уникати конфліктів між паралельними обчисленнями, що є ключовим для успішної реалізації паралельного алгоритму.
3) Розроблені алгоритми для кожного з чотирьох потоків, а також створена схема взаємодії потоків, де використані монітори з відповідними методами. Ця схема не лише забезпечує відсутність дідлоків, але й гарантує правильну синхронізацію всіх потоків у програмі.
4) Під час тестування проблем не виявлено й перевірка показала ті самі значення, що підтверджує точність розрахунків та успішне паралельне виконання завдань.