Darea de seamă la Java: Lucrare de laborator nr. 5.2

Tema: Programare paralela in java

Varianta: Nr. 16

Efectuat: Țurcanu Cristian, studentul grupei IA1901

Verificat: Epifanova Irina, lector universitar, magistru în informatică

Formularea problemei:

De scris un program în Java care, pentru o matrice bidimensională, în locul elementelor scrie suma vecinilor acestora, numărul de iterații și matricea se introduce de la tastatură. În program de folosit cîte un fir de execuție, care va face calcule, pentu fiecare linie a matricei. De afișat punctul de inceput și sfirșit pentru fiecare fir, precum și momentul de rescriere a matricei. Clasa firelor de execuție trebuie să fie moștenită de la Thread.

Codul sursă

MatrixThread.java

```
public class MatrixThread extends Thread{
    static int Matrix[][];
    static int rows, cols, finishedThreadCount;
    int threadIdRow;
    int generations;

    private static final Object threadCounterLock = new Object();
    private static final Object waitNotifyLock = new Object();

    static {
        rows = Helper.randomizer.nextInt(6) + 5;
        cols = Helper.randomizer.nextInt(6) + 5;
        Matrix = new int[rows][cols];

        Matrix[rows/2][cols/2] = 1;
    }

    static void PrintMatrix() {
        for (int i = 0; i < rows; i++)  {
            for (int j = 0; j < cols; j++) {
                Helper.print(Matrix[i][j] + " ");
            }

            Helper.println("");
        }
    }

    public MatrixThread(int id, int generations) {
        this.threadIdRow = id;
```

```
            this.generations = generations;
        }

        public void run() {
            while (generations != 0) {
                Helper.println("Started reading");

                int localResult[] = new int[cols];

                for (int j = 0; j < cols; j++) {
                    int cellResult = Matrix[threadIdRow][j];

                    if (threadIdRow + 1 < rows) {
                        if (j - 1 >= 0)
                            cellResult += Matrix[threadIdRow + 1][j - 1];

                        if (j + 1 < cols)
                            cellResult += Matrix[threadIdRow + 1][j + 1];

                        cellResult += Matrix[threadIdRow + 1][j];
                    }

                    if (threadIdRow - 1 >= 0) {
                        if (j - 1 >= 0)
                            cellResult += Matrix[threadIdRow - 1][j - 1];

                        if (j + 1 < cols)
                            cellResult += Matrix[threadIdRow - 1][j + 1];

                        cellResult += Matrix[threadIdRow - 1][j];
                    }

                    if (j - 1 >= 0)
                        cellResult += Matrix[threadIdRow][j - 1];

                    if (j + 1 < cols)
                        cellResult += Matrix[threadIdRow][j + 1];

                    localResult[j] = cellResult;
                }

                Helper.println("Finished reading");

                IncrementFinishedThreadCount();

                if (GetFinishedThreadCount() != rows) {
                    synchronized (waitNotifyLock) {
                        try {
                            waitNotifyLock.wait();
                        } catch (InterruptedException e) {}
                    }
                } else {
                    synchronized (waitNotifyLock) {
                        waitNotifyLock.notifyAll();
                    }
                }

                Helper.println("Started writing");

                for (int j = 0; j < cols; j++) {
                    Matrix[threadIdRow][j] = localResult[j];
                }

                Helper.println("Finished writing");

                DecrementFinishedThreadCount();

                if (GetFinishedThreadCount() != 0) {
                    synchronized (waitNotifyLock) {
                        try {
                            waitNotifyLock.wait();
                        } catch (InterruptedException e) {}
                    }
                } else {
                    synchronized (waitNotifyLock) {
                        PrintMatrix();

                        waitNotifyLock.notifyAll();
                    }
                }



                generations--;
            }

            synchronized (Main.mainLock) {
                Main.mainLock.notifyAll();
            }
        }

        public static void IncrementFinishedThreadCount() {
            synchronized (threadCounterLock) {
                finishedThreadCount++;
            }
        }

        public static void DecrementFinishedThreadCount() {
            synchronized (threadCounterLock) {
                finishedThreadCount--;
            }
        }

        public static int GetFinishedThreadCount() {
            synchronized (threadCounterLock) {
                return finishedThreadCount;
            }
        }
    }
```

Main.java

```
public class Main {
    public static final Object mainLock = new Object();


    public static void main(String[] args) {
        MatrixThread.PrintMatrix();

        for (int i = 0; i < MatrixThread.rows; i++)
            (new MatrixThread(i, 3)).start();

        synchronized (mainLock) {
            try {
                mainLock.wait();
            } catch (InterruptedException e) {}
        }
    }
}
```

Helper.java

```
import java.util.Random;
import java.io.*;

public class Helper {

    public static Random randomizer = new Random();

    public static String InputString() {
        BufferedReader box = new BufferedReader(new InputStreamReader(System.in));

        String str = "";
        try {
            str = box.readLine();
        } catch (Exception e) {
            System.out.println("Failed row reading at Helper.InputString");
            System.out.println(e);
        }

        return str;
    }

    public static int InputInt() {
        boolean success = false;
        int result = 0;

        do {
            try {
                result = (Integer.valueOf(InputString())).intValue();
                success = true;
            } catch (Exception e) {
                System.out.println(e);
                System.out.print("Dati valoarea inca o data: ");
                success = false;
            }
        } while (!success);

        return result;
    }

    public static float InputFloat() {
        boolean success = false;
        float result = 0;

        do {
            try {
                result = (Float.valueOf(InputString())).floatValue();
                success = true;
            } catch (Exception e) {
                System.out.println(e);
                System.out.print("Dati valoarea inca o data: ");
                success = false;
            }
        } while (!success);

        return result;
    }

    public static int InputIntLimit(int min, int max) {
        int result = min - 1;

        do {
            result = InputInt();
            if (result < min || result > max) {
                println("Outside of limits [" + min + ", " + max + "], try
again");
            }
        } while (result < min || result > max);

        return result;
    }

    public static int InputIntLimit(int min) {
        return InputIntLimit(min, Integer.MAX_VALUE);
    }

    public static float InputFloatLimit(float min, float max) {
        float result = min - 1f;
        do {
            result = InputFloat();
            if (result < min || result > max) {
                println("Outside of limits [" + min + ", " + max + "], try
again");
            }
        } while (result < min || result > max);

        return result;
    }

    public static boolean InputBoolean() {
        boolean success = false;
        boolean result = false;
        do {
            try {
                result = Boolean.parseBoolean(InputString());
                success = true;
            } catch (Exception e) {
                println("Shit's fucked man");
                println(e.toString());
                println("Try again");
                success = false;
            }
        } while (!success);

        return result;
    }

    public static float InputFloatLimit(float min) {
        return InputFloatLimit(min, Float.MAX_VALUE);
    }

    public static void println(String text) {
        System.out.println(text);
    }

    public static void print(String text) {
        System.out.print(text);
    }

    public static String FileRead(String path) {
        String result = "";
        try {
            BufferedReader box = new BufferedReader(new FileReader(path));

            String line;
            while ((line = box.readLine()) != null) {
                result += line;
            }
            box.close();
        } catch (Exception e) {
            println("Failed file reading at Helper.FileRead");
            println(e.toString());
        }

        return result;
    }

    public static void FileWrite(String path, String text) {
        try {
            File file = new File(path);
            FileOutputStream fileOutputStream = new FileOutputStream(file);

            if (!file.exists()) {
                file.createNewFile();
            }

            byte b[] = text.getBytes();

            fileOutputStream.write(b);
            fileOutputStream.flush();
            fileOutputStream.close();
        } catch (Exception e) {
            println("Failed file writing at Helper.FileWrite");
            println(e.toString());
        }
    }
}
```

Rezultatele rulării programului

```
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
Started reading
Finished reading
Started reading
Started reading
Started reading
Finished reading
Finished reading
Started reading
Finished reading
Finished reading
Started reading
Finished reading
Started writing
Started writing
Finished writing
Started writing
```

```
Finished writing
Started writing
Finished writing
Started writing
Finished writing
Finished writing
Started writing
Finished writing
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 1 1 1 0 0
0 0 1 1 1 0 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
Started reading
Started reading
Finished reading
Started reading
Started reading
Started reading
Started reading
Finished reading
Finished reading
Finished reading
Finished reading
Finished reading
Started writing
Started writing
Finished writing
Started writing
Finished writing
Finished writing
Started writing
Finished writing
Started writing
Finished writing
Started writing
Finished writing
0 0 0 0 0 0 0
0 1 2 3 2 1 0
0 2 4 6 4 2 0
0 3 6 9 6 3 0
0 2 4 6 4 2 0
0 1 2 3 2 1 0
Started reading
Started reading
Started reading
Started reading
Finished reading
Started reading
Started reading
Finished reading
Finished reading
Finished reading
Finished reading
Finished reading
Started writing
Started writing
Finished writing
Started writing
Finished writing
Started writing
Finished writing
Started writing
Finished writing
Started writing
Finished writing
Finished writing
1 3 6 7 6 3 1
3 9 18 21 18 9 3
6 18 36 42 36 18 6
7 21 42 49 42 21 7
6 18 36 42 36 18 6
3 9 18 21 18 9 3
```