Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні систем та технологій

ЛАБОРАТОРНА РОБОТА №7

з дисципліни «Основи програмування - 2»

Тема: «Синхронізація потоків виконання»

Виконали:
студенти групи ІА-31
Клим’юк В.Л, Самелюк А.С,
Дук М.Д, Сакун Д.С

Перевірив:
асистент кафедри ІСТ
Степанов А. С.

Київ-2024

Тема: Синхронізація потоків виконання

Мета: Дослідження принципів синхронізації потоків виконання в мові програмування Java для забезпечення безпечного доступу до спільних ресурсів у мультипотокових програмах.

Хід роботи

1. Пригадати API для здійснення операцій вводу-виводу. Особливу увагу звернути на такі класи та інтерфейси:

- InputStream
    - FileInputStream
- Reader
    - FileReader
- AutoCloseable
    - Closable
- IOException
    - FileNotFoundException

2. Пригадати Collection Framework. Особливу увагу звернути на такі класи та інтерфейси:

- Map
    - HashMap
    - TreeMap
- List
    - ArrayList

3. Пригадати засоби синхронізації потоків виконання

4. Виконати завдання свого варіанту з л/р №5 другого семестру таким чином, щоб програма збирала єдину статистику використання слів одночасно з декількох файлів

в паралельних потоках (необхідно передбачити синхронізацію при доступі до об’єктів збереження статистики). Крім того:

4.1. Перевірити яким чином кількість та/або розмір файлів, а також кількість потоків впливає на швидкодію. Проаналізувати результати, виявити «вузьке місце» та надати рекомендації щодо збільшення загальної продуктивності.

4.2. Перевірити, що для одних і тих самих файлів результати паралельної обробки не відрізняються від послідовної обробки усіх файлів в одному потоці виконання.

```
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

public class WordFrequency {

    public static void main(String[] args) {
        String[] filenames = new String[]{
                "text1.txt",
                "text2.txt",
                "text3.txt",
                "text4.txt"
        };
        int threads = 4;

        //Multiple threads
        try {
            ArrayList<String> rarestWords = rarestWords(filenames, threads);
            System.out.println("Rarest words in multiple threads:");
            for (String word : rarestWords) {
                System.out.println(word);
            }
        } catch (IllegalArgumentException | NullPointerException | IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }

        //Single thread
        try {
            ArrayList<String> rarestWords = rarestWordsSingleThread(filenames);
            System.out.println("Rarest words in one thread:");
            for (String word : rarestWords) {
                System.out.println(word);
            }
        } catch (IllegalArgumentException | NullPointerException | IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }

    public static ArrayList<String> rarestWords(String[] filenames, int threadsCount)
throws IOException {
        Map<String, Integer> wordFrequency = new HashMap<>();
        Thread[] threads = new Thread[threadsCount];
```

```
        for(int i = 0; i < filenames.length; ){
            for(int j = 0; j < threads.length; j++){
                if (i >= filenames.length)
                    break;

                if(threads[j] == null || !threads[j].isAlive()){
                    final String filename = filenames[i];
                    System.out.println("Thread created" );
                    threads[j] = new Thread(new Runnable() {
                        @Override
                        public void run() {
                            try (BufferedReader reader = new BufferedReader(new
FileReader(filename))) {
                                String line;
                                while ((line = reader.readLine()) != null) {
                                    String[] words = line.trim().split("(\\s*[,\\-
+%#№=/.!?*:$^`;\"&\\[\\]\\(\\)\\{\\}]\\s*)|(\\s+)");

                                    for (String word : words) {
                                        synchronized (wordFrequency){
                                            wordFrequency.put(word,
wordFrequency.getOrDefault(word, 0) + 1);
                                        }
                                    }
                                }
                            catch (Exception e){
                                System.out.println("Error happened");
                            }

                            synchronized (System.out){
                                System.out.println("Thread end");
                            }
                        }
                    });
                    threads[j].start();
                    i++;
                }

            }
        }

        for(var t : threads){
            try{
                t.join();
            }catch (InterruptedException e){
                System.out.println("Thread interrupted");
            }
        }

        int minFrequency = Integer.MAX_VALUE;
        for (int frequency : wordFrequency.values()) {
            if (frequency < minFrequency) {
                minFrequency = frequency;
            }
        }

        ArrayList<String> rarestWords = new ArrayList<>();

        for (Entry<String, Integer> entry : wordFrequency.entrySet()) {
            if (entry.getValue() == minFrequency) {
                rarestWords.add(entry.getKey());
            }
```

```
        }

        return rarestWords;
    }

    public static ArrayList<String> rarestWordsSingleThread(String[] filenames)
throws IOException {
        Map<String, Integer> wordFrequency = new HashMap<>();

        for(var filename : filenames){
            try (BufferedReader reader = new BufferedReader(new
FileReader(filename))) {
                String line;
                while ((line = reader.readLine()) != null) {
                    String[] words = line.trim().split("(\\s*[,\\-
+%#№=/.!?*:$^`;\"&\\[\\]\\(\\)\\{\\}]\\s*)|(\\s+)");

                    for (String word : words) {
                        wordFrequency.put(word, wordFrequency.getOrDefault(word, 0) +
1);
                    }
                }
            } catch (FileNotFoundException e) {
                throw new FileNotFoundException("File not found: " + filename);
            } catch (IOException e) {
                throw new IOException("Error reading file: " + e.getMessage());
            }
        }

        int minFrequency = Integer.MAX_VALUE;
        for (int frequency : wordFrequency.values()) {
            if (frequency < minFrequency) {
                minFrequency = frequency;
            }
        }

        ArrayList<String> rarestWords = new ArrayList<>();

        for (Entry<String, Integer> entry : wordFrequency.entrySet()) {
            if (entry.getValue() == minFrequency) {
                rarestWords.add(entry.getKey());
            }
        }

        return rarestWords;
    }
}
```

Код 1.1

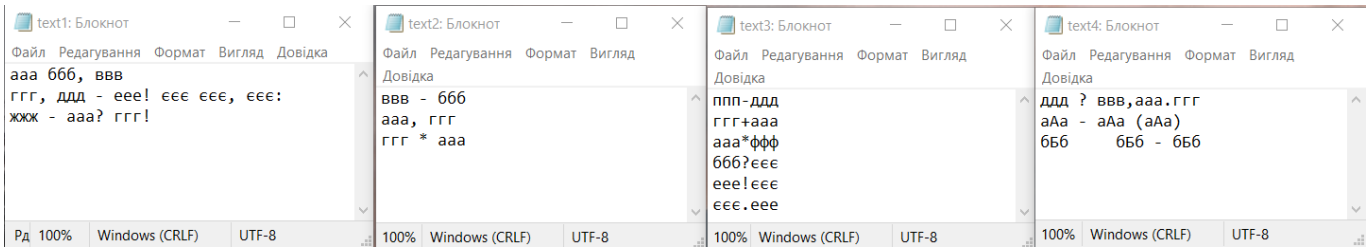

Рисунок 1.1 – Приклади тестових файлів

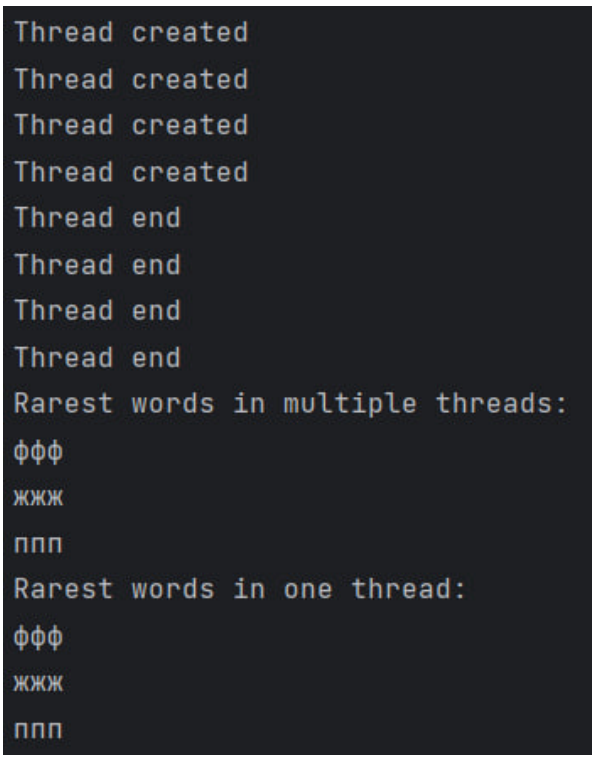

Рисунок 1.2 – Результат роботи коду

Висновки: У результаті виконання лабораторної роботи ми отримали розуміння того, як застосовувати механізми синхронізації в Java для координації виконання потоків та уникнення гонок за ресурсами. Використання ключових конструкцій, таких як synchronized, wait та notify, дозволяє ефективно керувати взаємодією потоків та забезпечити консистентність даних у багатопотокових програмах.