Лабораторна робота №6.

Паралельне виконання.

Багатопоточність. Ефективність використання.

Мета:

- **Ознайомлення з моделлю потоків Java.**
- **Організація паралельного виконання декількох частин програми.**
- **Вимірювання часу паралельних та послідовних обчислень.**
- **Демонстрація ефективності паралельної обробки.**

1.1 Розробник

Гірник Юрій, КН-108, номер варіанту індивідуального завдання – 6.

1.2 Загальна завдання

1. Використовуючи програми рішень попередніх задач, продемонструвати можливість паралельної обробки елементів контейнера: створити не менше трьох додаткових потоків, на яких викликати відповідні методи обробки контейнера.

2. Забезпечити можливість встановлення користувачем максимального часу виконання(таймаута) при закінченні якого обробка повинна припинятися незалежно від того знайдений кінцевий результат чи ні.

3. Для паралельної обробки використовувати алгоритми, що не змінюють початкову колекцію.

4. Кількість елементів контейнера повинна бути досить велика, складність алгоритмів обробки колекції повинна бути зіставна, а час виконання приблизно однаковий, наприклад:
   - ○ пошук мінімуму або максимуму;
   - ○ обчислення середнього значення або суми;
   - ○ підрахунок елементів, що задовольняють деякій умові;
   - ○ відбір за заданим критерієм;
   - ○ власний варіант, що відповідає обраній прикладної області.

5. Забезпечити вимірювання часу паралельної обробки елементів контейнера за допомогою розроблених раніше методів.

6. Додати до алгоритмів штучну затримку виконання для кожної ітерації циклів поелементної обробки контейнерів, щоб загальний час обробки був декілька секунд.

7. Реалізувати послідовну обробку контейнера за допомогою методів, що використовувались для паралельної обробки та забезпечити вимірювання часу їх роботи.

8. Порівняти час паралельної і послідовної обробки та зробити висновки про ефективність розпаралелювання:

○ результати вимірювання часу звести в таблицю;

○ обчислити та продемонструвати у скільки разів паралельне виконання швидше послідовного.

2 Опис програми

Дана програма, на прикладі пошуку найбільшого/найменшого/суми числа, наочно показує переваги паралельно виконання над послідовним під час будь-яких розрахунків при програмування.

2.1 Засоби ООП

Використано інкапсуляцію даних, створено 4 пов’язаних класи.

2.2 Ієрархія та структура класів

А) Клас Main з консольною обробкою даних.

Б) Клас ThreadOne з методом run для пошуку мінімального елементу.

В) Клас ThreadTwo з методом run для пошуку максимального елементу.

Г) Клас ThreadThree з методом run для пошуку суми всіх елементів.

Д) Допоміжиний клас TimeLimit для зупинки потоків.

2.3 Важливі фрагменти програми.

```
public class ThreadOne extends Thread {
    private long array[];
    private int lower;
    private long sleep;
    private double  timeout;
    private TimeLimit time = new TimeLimit();

    ThreadOne(long array[], long sleep, double timeout){
        this.array = array;
        this.sleep = sleep;
        this.timeout = timeout * 1_000_000_000;
    }

    @Override
    public void run(){
        boolean isTime = false;
        double en, st = System.nanoTime();
        System.out.println("Початок роботи потоку 1");
        lower = (int) array[0];
        for(int i = 1; i < array.length ;i++ ){
        if (lower > array[i])  lower = (int) array[i];
        try {
                Thread.sleep(sleep);
            } catch (InterruptedException ex) {
                Logger.getLogger(ThreadThree.class.getName()).log(Level.SEVERE, null, ex);
            }
        en = System.nanoTime();
       if(en - st > timeout) {
            isTime = true;
            break;
        }
        }
        if (isTime)
            System.out.println("\nПотік 1 - таймаут!\nЗУПИНКА!");
        else {
            en  = (double) (System.nanoTime() - st) / 1_000_000_000;
         System.out.println("Час виконання потоку 1 - " + en  + " сек"
                 + "\nМінімальний елемент: " + lower);
        }
    }
```

```
case "1":
    first.start();
    second.start();
    third.start();
    first.join();
    second.join();
    third.join();
    break;
case "2":
    st = System.nanoTime();
    timelimit = first.run(timelimit.rest);
    if (timelimit.timeout) {
        timelimit.timeout = false;
        break;
    }
    timelimit = second.run(timelimit.rest);
    if (timelimit.timeout){
        timelimit.timeout = false;
        break;
    }
    timelimit = third.run(timelimit.rest);
    if (timelimit.timeout){
        timelimit.timeout = false;
        break;
    }
    en = System.nanoTime();
    elapsedTimeInSecond = (double) (en - st) / 1_000_000_000;
    System.out.println("Час виконання послідовно: " + elapsedTimeInSecond + " сек");
    break;
```

3. Варіанти використання

Дана програма може використовуватись для наочного показу різниці в часу між паралельним та послідовним виконанням.

ВИСНОВКИ

Дана лабораторна робота ознайомила з моделлю потокі в Java. Я навчився розділяти виконання програми частинно, парарельними потоками. Було реалізовано таймаут, обчислення часу виконання потоками певних обрахунків.