Міністерство освіти, науки, молоді та спорту України
Національний університет “Львівська політехніка”

Кафедра СШІ

Лабораторна робота №13

Виконав:

ст. групи КН-107

Древницький Ю.А.

Прийняв :

Асистент

Кафедри СШІ

Швороб І.Б.

Обов’язковий розділ - Зауваження

Тема:

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.

Вимоги:

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.

Розробник:

Студент академічної групи КН-107

Древницький Юрій Анатолійович

Варіант 18

Розв’язок:

```
package lab_13;

import company.Listt;
```

```
import static lab_13.Timer.*;

public class Main {

    static Listt numbers = new Listt();

    // каунтер кількості двійок, трійок і сімок
    static int count2 = 0;
    static int count3 = 0;
    static int count7 = 0;

    // вміст контейнера
    public static final int N = 100;

    // час паралельного та послідовного виконання(сек)
    static double time1 = 0;
    static double time2 = 0;

    // ліміт часу виконання програми
    static int timeLimit = 100000;


    // для заповнення колекції числами
    static synchronized Listt numListt (Listt listt)
    {
        for (int i = 0; i < N; i++)
        {
            if (i<32)
            {
                listt.addFirst(2);
```

```
        }
        else if (i<67)
        {
            listt.addFirst(3);
        }
        else
            {
                listt.addFirst(7);
            }


    }

    return listt;
}


public static void main(String[] args) throws Exception {

    // початок відліку часу роботи програми
    setTimerStart();

    // заповнення колекції числами
    numbers = Main.numListt(numbers);

    /** провірка вмісту колекції */
    //int count = 0;
    //for ( Object o : numbers)
    //{
    //    System.out.println(o);
```

```
//    count++;
//}



Thread2 thread2 = new Thread2();
Thread3 thread3 = new Thread3();
Thread7 thread7 = new Thread7();

thread2.start();
thread3.start();
thread7.start();

thread2.join();
thread3.join();
thread7.join();

//кінець відліку часу роботи програми
setTimerFinish();

time1 = getResult()/Math.pow(10,9);

System.out.println("\nдвійок: "+count2+ "\nтрійок: "+count3 + "\nсімірок: "+ count7);
System.out.println("Час паралельного виконання: "+getResult()/Math.pow(10,9));
System.out.println("\n");



//послідовне виконання
count2 = 0;
count3 = 0;
count7 = 0;
```

```
        setTimerStart();
        method237();
        setTimerFinish();

        time2 = getResult()/Math.pow(10,9);

        System.out.println("\nдвійок: "+count2+ "\nтрійок: "+count3 + "\nсімірок: "+ count7);
        System.out.println("Час послідовного виконання: "+getResult()/Math.pow(10,9));
        System.out.println("\n");

        System.out.println("час послідовного виконання більший за паралельного у: "+ time2/time1 + " разів");
    }

    static class Thread2 extends Thread
    {
        @Override
        public void run() {

            method2();
        }
    }

    static class Thread3 extends Thread
    {
        @Override
        public void run() {

            method3();
```

```
    }
}

static class Thread7 extends Thread
{
    @Override
    public void run() {

        method7();
    }
}

static void method237()
{
    method2();
    method3();
    method7();
}

static void method2()
{
    //рахуємо кількість двійок у колекції
    for ( Object o : numbers)
    {
        // штучне сповільнення виконання для кожної ітерації циклів обробки контейнерів
        try {
            Thread.sleep(N/2);
        } catch (Exception e) {}

        if (o.equals(2)){
```

```
            count2++;
        }
        // перевірка часового ліміту
        setTimerFinish();
        limitChecker(timeLimit);

    }
}

static void method3()
{
    //рахуємо кількість трійок у колекції
    for ( Object o : numbers)
    {
        // штучне сповільнення виконання для кожної ітерації циклів обробки контейнерів
        try {
            Thread.sleep(N/2);
        } catch (Exception e) {}

        if (o.equals(3)){
            count3++;
        }

        // перевірка часового ліміту
        setTimerFinish();
        limitChecker(timeLimit);

    }
}
```

```
    static void method7()
    {
        //рахуємо кількість сімок у колекції
        for ( Object o : numbers)
        {
            // штучне сповільнення виконання для кожної ітерації циклів обробки контейнерів
            try {
                Thread.sleep(N/2);
            } catch (Exception e) {}

            if (o.equals(7)){
                count7++;
            }

            // перевірка часового ліміту
            setTimerFinish();
            limitChecker(timeLimit);
        }
    }
}

package lab_13;

public class Timer {

    private static long timerStart;
    private static long timerFinish;
    private static long timerWorkingLimit;
```

```
public static void setTimerStart() {
   Timer.timerStart = System.nanoTime();
}
public static void setTimerFinish() {
   Timer.timerFinish = System.nanoTime();
}
public static void setTimerWorkingLimit(long timerWorkingLimit) {
   Timer.timerWorkingLimit = timerWorkingLimit*(long)Math.pow(10,6);
}

public static long getTimerStart() {
   return timerStart;
}
public static long getTimerFinish() {
   return timerFinish;
}
public static long getTimerWorkingLimit() {
   return timerWorkingLimit;
}

public static long getResult()
{

   return timerFinish-timerStart;
}

public static void limitChecker(long timerWorkingLimit)
{
```

```
        if (getResult() > timerWorkingLimit*(long)Math.pow(10,6))
        {
           System.out.println("Program time Limit");
           System.exit(0);
        }
    }
}
```

Висновок:

Отже, я ознайомився з моделлю багато потоковості, навчився створювати багатопотокові програми, зрозумів переваги застосування паралельного виконання.