МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Звіт

Лабораторна робота №6

З дисципліни:

Об’єктно орієнтоване програмування

Виконав

студент групи КН-111

Бойко Андрій

Викладач:

Грабовська Н. Р.

Тема роботи

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

1. Вимоги

1.1 Розробник

Бойко Андрій Віталійович

КН-111

3 варіант

1.2 Загальне завдання

Ознайомлення з моделлю потоків Java

. ● Організація паралельного виконання декількох частин програми. ● Вимірювання часу паралельних та послідовних обчислень. ● Демонстрація ефективності паралельної обробки.

1.3 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.

1.Опис програми

Main

```
package lab_6;

import java.util.ArrayList;
import java.util.Scanner;

public class Main {
	static MultiThread multi = new MultiThread();
	static Linear linear = new Linear();

	public static void main(String[] args) {
		Scanner scn = new Scanner(System.in);
		System.out.print("Enter count of rounds: ");
		int n = scn.nextInt();	scn.close();

		ArrayList<Integer> mList;
		ArrayList<Integer> lList;

		long mStart = System.nanoTime();
		mList = multi.begin(n);
		long mFInish = System.nanoTime();

		long lStart = System.nanoTime();
		lList = linear.begin(n);
		long lFInish = System.nanoTime();

		double m = (double)Math.round((mFInish-mStart)/10000000)/100;
		double l = (double)Math.round((lFInish-lStart)/10000000)/100;

		System.out.println("MultiThread time is: "+ m+" mid time is: "+m/n);
		System.out.println("Linear run time is: " + l+" mid time is: "+l/n);

		if(args.length > 0)
			if(args[0].compareTo("-d") == 0) {
				System.out.println();
				addInfo(mList , lList);
			}
	}

	private static void addInfo(ArrayList<Integer> m, ArrayList<Integer> l) {
		for(int i : m)
			System.out.print((double)i/100 + "sec ");

		System.out.println();

		for(int i : l)
			System.out.print((double)i/100 + "sec ");
	}
}
```

Linear

```
package lab_6;

import java.util.ArrayList;

public class Linear {
	public ArrayList<Integer> begin(int n) {
		ArrayList<Integer> l = new ArrayList<Integer>();
```

```
		for(int i = 0 ; i < n ; i++) {
			long s = System.nanoTime();

			runFunc();

			l.add( (int) Math.round((System.nanoTime() - s)/10000000) );
		}
		return l;
	}

	private void runFunc() {    //To short the main
		for(int i = 1 ; i <= 100 ; i++) {
			System.out.println("m  "+i);
			try { Thread.sleep(10);
			} catch (InterruptedException e) { }
		}

		for(int i = 1 ; i <= 100 ; i++) {
			System.out.println("t1 " + i);
			try { Thread.sleep(10);
			} catch (InterruptedException e) { }
		}
		for(int i = 1 ; i <= 100 ; i++) {
			System.out.println("l1 " + i);
			try { Thread.sleep(10);
			} catch (InterruptedException e) { }
		}



	}


}
```

MultiThread

```
package lab_6;

import java.util.ArrayList;

public class MultiThread {
	static Thread1 t1;
	static Thread2 t2;

	public ArrayList<Integer> begin(int n) {
		ArrayList<Integer> l = new ArrayList<Integer>();

		for(int i = 0 ; i < n ; i++) {  //Run n times
			long s = System.nanoTime();

			runFunc();
			check();

			l.add( (int) Math.round((System.nanoTime() - s)/10000000) );
		}
		return l;
	}

	private void runFunc() {    //To short the main
		t1 = new Thread1();
		t1.start();

		t2 = new Thread2();
		t2.start();
```

```
			for(int i = 1 ; i <= 100 ; i++) {
				try { Thread.sleep(10);
				} catch (InterruptedException e) { }
				System.out.println("m  "+i);
			}
		}

		private void check() {    //If alive - continue
			if(t1.isAlive()) {
				try {
					t1.join();
				} catch (InterruptedException e) { }
			}
			if(t2.isAlive()) {
				try {
					t2.join();
				} catch (InterruptedException e) { }
			}

		}
}
```

Thread1

```
package lab_6;

public class Thread1 extends Thread{
	@Override
	public void run() {
		for(int i = 1 ; i <= 100 ; i++) {
			System.out.println("t1 " + i);
			try {
				sleep(10);
			} catch (InterruptedException e) { e.printStackTrace(); }
		}
	}
}
```

Thread2

```
package lab_6;

public class Thread2 extends Thread{
	@Override
	public void run() {
		for(int i = 1 ; i <= 100 ; i++) {
			System.out.println("t1 " + i);
			try {
				sleep(10);
			} catch (InterruptedException e) { e.printStackTrace(); }
		}
	}
}
```

Висновки

На цій лабораторної роботи, я ознайомився з моделлю потоків java.