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

ЛАБОРАТОРНА РОБОТА №6

з дисципліни «Основи програмування - 2»

Тема: «MultiThreading»

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: MultiThreading

Мета: Метою цієї лабораторної роботи є вивчення концепції багатопотоковості (MultiThreading) в програмуванні. Зокрема, мета полягає в ознайомленні з основними принципами роботи з потоками, їх створенням, управлінням та взаємодією між ними. Також в рамках цієї лабораторної роботи буде досліджено переваги та можливі проблеми, пов'язані з багатопотоковим програмуванням, а також засоби синхронізації доступу до ресурсів.

Хід роботи

1. Пригадати API для здійснення паралельних обчислень. Особливу увагу звернути на такі інтерфейси, класи та методи:

- Runnable;
- Thread;
- run();
- start();
- join().

2. Знайти суму арифметичної прогресії

$f(n, N) = n + 2n + 3n + 4n + 5n + ... + Nn,$

де n - номер варіанту (n=1),
N - 100_000_000,

наступними способами:

1) за допомогою формули розрахунку суми арифметичної прогресії;

2) «в лоб» за допомогою оператора циклу, що працює в одному треді;

3-7) «в лоб» за допомогою оператора циклу, що працює в декількох тредах (k=2, 4, 8, 16, 32).

Порівняти точність (має співпасти для усіх способів) та час t отримання результатів (для випадків 2 - 7). У звіті навести таблицю та побудувати графік t(k), пояснити отримані результати.

```
import java.util.concurrent.TimeUnit;

interface Calculator {
    long calculateSum(int n, long N);
}

class FormulaCalculator implements Calculator {
    public long calculateSum(int n, long N) {
        return (N * (N + 1) / 2) * n;
    }
}

class SingleThreadCalculator implements Calculator {
    public long calculateSum(int n, long N) {
        long sum = 0;
        for (int i = 1; i <= N; i++) {
            sum += n * i;
        }
        return sum;
    }
}

class ParallelCalculator implements Calculator {
    private int threads;

    public ParallelCalculator(int threads) {
        this.threads = threads;
    }

    public long calculateSum(int n, long N) {
        final long[] sum = {0};
        Thread[] threadPool = new Thread[threads];
        for (int k = 0; k < threads; k++) {
            final int start = k;
            threadPool[k] = new Thread(new Runnable() {
                @Override
                public void run() {
                    long localSum = 0;
                    for (long i = start + 1; i <= N; i += threads) {
                        localSum += n * i;
                    }
                    synchronized (sum) {
                        sum[0] += localSum;
                    }
                }
            });
            threadPool[k].start();
        }
        try {
            for (Thread thread : threadPool) {
                thread.join();
            }
        } catch (InterruptedException e) {
```

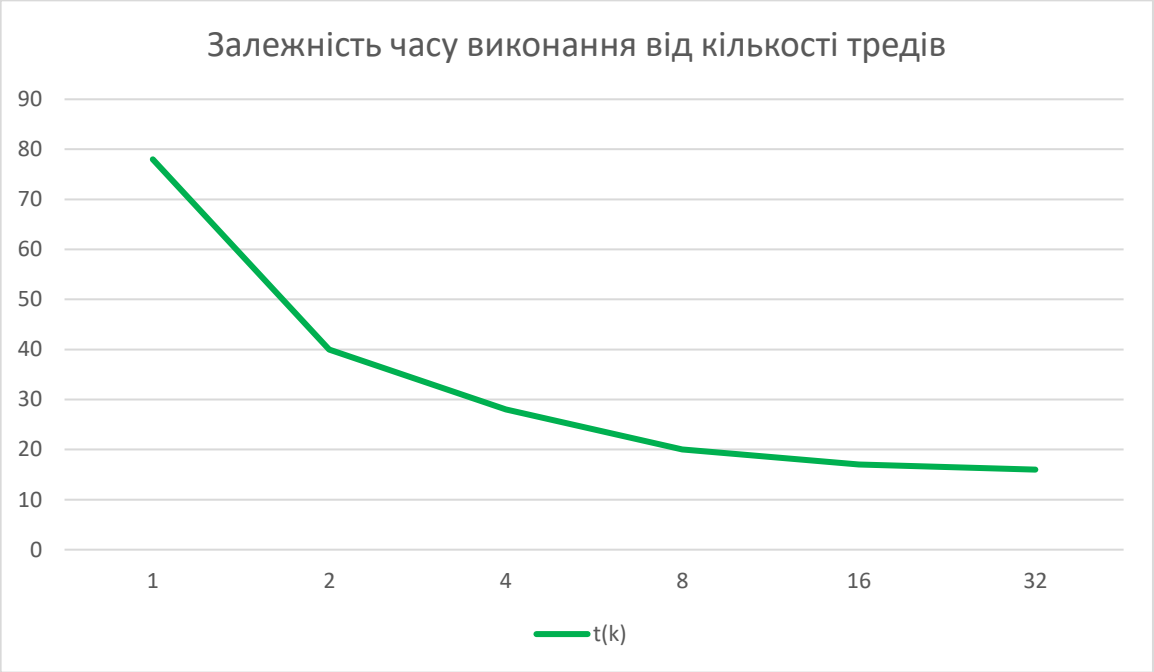
```
            e.printStackTrace();
        }
        return sum[0];
    }
}


public class Main {
    public static void main(String[] args) {
        int n = 1;
        long N = 100_000_000;

        Calculator formulaCalculator = new FormulaCalculator();
        calculateAndPrint("Formula", formulaCalculator, n, N);

        Calculator singleThreadCalculator = new SingleThreadCalculator();
        calculateAndPrint("Single thread", singleThreadCalculator, n, N);

        for (int threads : new int[]{2, 4, 8, 16, 32}) {
            Calculator parallelCalculator = new ParallelCalculator(threads);
            calculateAndPrint("Parallel with " + threads + " threads",
parallelCalculator, n, N);
        }
    }

    private static void calculateAndPrint(String method, Calculator calculator, int
n, long N) {
        long startTime = System.nanoTime();
        long result = calculator.calculateSum(n, N);
        long endTime = System.nanoTime();
        long time = TimeUnit.NANOSECONDS.toMillis(endTime - startTime);

        System.out.println(method + " result: " + result);
        System.out.println(method + " time: " + time + "ms");
    }
}
```

Код 1.1

```
Formula result: 5000000050000000
Formula time: 0ms
Single thread result: 5000000050000000
Single thread time: 78ms
Parallel with 2 threads result: 5000000050000000
Parallel with 2 threads time: 40ms
Parallel with 4 threads result: 5000000050000000
Parallel with 4 threads time: 28ms
Parallel with 8 threads result: 5000000050000000
Parallel with 8 threads time: 20ms
Parallel with 16 threads result: 5000000050000000
Parallel with 16 threads time: 17ms
Parallel with 32 threads result: 5000000050000000
Parallel with 32 threads time: 16ms
```

Рисунок 1.1 – Результат роботи коду

Кількість тредів (k)	Час виконання (t)
1	78
2	40
4	28
8	20
16	17
32	16



3. Відповісти на контрольні питання.

Висновки: Під час виконання лабораторної роботи ми отримали важливі знання щодо роботи з багатопотоковістю в програмуванні. Було розглянуто основні концепції створення, управління та взаємодії потоків. Ми виявили, що використання багатопотоковості може значно покращити ефективність програм, особливо в умовах багатозадачності та паралельності обчислень. Однак, варто бути обережними при роботі з потоками, оскільки неправильне їх використання може призвести до проблем з одночасним доступом до спільних ресурсів та виникнення гонок даних. Тому важливо використовувати засоби синхронізації для забезпечення правильної роботи програми в умовах багатопотоковості.