**МАССОВЫХ КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра «Информационные технологии»

Лабораторная работа по ИТП №7

Выполнил: Студент группы

БВТ2204

Сарыков Эрбол

Москва

2023

Задание 1.

Реализация многопоточной программы для вычисления суммы

элементов массива.

Вариант 2. Создать пул потоков с помощью класса ExecutorService и

разделить массив на равные части, каждую из которых будет

обрабатывать отдельный поток. После завершения работы всех потоков

результаты будут складываться в главном потоке.

Задание 2.

Реализация многопоточной программы для поиска наибольшего

элемента в матрице.

Вариант 2. Создать пул потоков с помощью класса ExecutorService и

разделить матрицу на равные части, каждую из которых будет

обрабатывать отдельный поток. После завершения работы всех потоков

результаты будут сравниваться в главном потоке для нахождения

наибольшего элемента.

Задание 3:

У вас есть склад с товарами, которые нужно перенести на другой склад. У

каждого товара есть свой вес. На складе работают 3 грузчика. Грузчики

могут переносить товары одновременно, но суммарный вес товаров,

которые они переносят, не может превышать 150 кг. Как только грузчики

соберут 150 кг товаров, они отправятся на другой склад и начнут

разгружать товары.

Напишите программу на Java, используя многопоточность, которая

реализует данную ситуацию.

Использование Semaphore: Используйте семафоры для

ограничения доступа к складу и контроля над весом товаров.

Задание 1

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.\*;

public class SumCalculator {

    public static void main(String[] args) throws InterruptedException, ExecutionException {

        int[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

        int numOfThreads = 2; // количество потоков

        ExecutorService executorService = Executors.newFixedThreadPool(numOfThreads);

        List<Future<Integer>> futures = new ArrayList<>();

        int chunkSize = array.length / numOfThreads;

        for (int i = 0; i < numOfThreads; i++) {

            final int start = i \* chunkSize;

            final int end = (i + 1) \* chunkSize;

            Future<Integer> future = executorService.submit(() -> calculateSum(array, start, end));

            futures.add(future);

        }

        int totalSum = 0;

        for (Future<Integer> future : futures) {

            totalSum += future.get();

        }

        System.out.println("Сумма: " + totalSum);

        executorService.shutdown();

    }

    private static int calculateSum(int[] array, int start, int end) {

        int sum = 0;

        for (int i = start; i < end; i++) {

            sum += array[i];

        }

        return sum;

    }

}



1. Метод main

Метод main представляет собой точку входа в программу.

В этом методе создается массив array, который содержит набор чисел

Задается количество потоков numOfThreads, которые будут использоваться для распараллеливания вычислений.

Создается пул исполнителей executorService с фиксированным числом потоков, равным numOfThreads.

Создается список futures, в который будут добавляться результаты выполнения задач в виде объектов Future.

Вычисляется размер "куска" массива, который будет обрабатываться каждым потоком.

Запускается цикл, который разбивает массив на части и отправляет задачи на выполнение в потоки пула. Результаты задач сохраняются в списке futures.

Затем вычисляется общая сумма, суммируя результаты из futures.

Выводится общая сумма.

После завершения вычислений пул исполнителей закрывается с помощью метода shutdown.

2. Метод calculateSum

Этот метод вычисляет сумму элементов массива в заданном диапазоне.

Принимает на вход массив array, начальный индекс start и конечный индекс end

Вычисляет сумму элементов массива от start до end.

Задание 2

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.\*;

public class Matrix {

    public static void main(String[] args) throws ExecutionException, InterruptedException {

        int[][] matrix = {

            {1, 2, 3},

            {4, 5, 6},

            {7, 8, 9}

        };

        int numOfThreads = 3; // количество потоков

        ExecutorService executorService = Executors.newFixedThreadPool(numOfThreads);

        List<Future<Integer>> futures = new ArrayList<>();

        int rowsPerThread = matrix.length / numOfThreads;

        for (int i = 0; i < numOfThreads; i++) {

            final int startRow = i \* rowsPerThread;

            final int endRow = (i + 1) \* rowsPerThread;

            Future<Integer> future = executorService.submit(() -> findMax(matrix, startRow, endRow));

            futures.add(future);

        }

        int maxElement = Integer.MIN\_VALUE;

        for (Future<Integer> future : futures) {

            maxElement = Math.max(maxElement, future.get());

        }

        System.out.println("MAX: " + maxElement);

        executorService.shutdown();

    }

    private static int findMax(int[][] matrix, int startRow, int endRow) {

        int max = Integer.MIN\_VALUE;

        for (int i = startRow; i < endRow; i++) {

            for (int j = 0; j < matrix[i].length; j++) {

                max = Math.max(max, matrix[i][j]);

            }

        }

        return max;

    }

}



1. Метод main

Метод main представляет собой точку входа в программу.

В этом методе создается двумерный массив matrix, представляющий собой матрицу с числами.

Задается количество потоков numOfThreads, которые будут использоваться для распараллеливания вычислений.

Создается пул исполнителей executorService с фиксированным числом потоков, равным numOfThreads.

Создается список futures, в который будут добавляться результаты выполнения задач в виде объектов Future.

Вычисляется количество строк "на порцию", которое будет обрабатываться каждым потоком.

Запускается цикл, который разбивает матрицу на части и отправляет задачи на выполнение в потоки пула. Результаты задач сохраняются в списке futures.

Затем вычисляется общий максимальный элемент, сравнивая результаты из futures.

Выводится общий максимальный элемент.

После завершения вычислений пул исполнителей закрывается с помощью метода shutdown.

2. Метод findMax

Этот метод осуществляет поиск максимального элемента в заданной части матрицы.

Принимает на вход матрицу matrix, начальный индекс строки startRow и конечный индекс строки endRow.

Внутри циклов перебираются элементы матрицы в заданной части, и максимальное значение обновляется, если найден элемент больше текущего максимума.

Задание 3

import java.util.concurrent.Semaphore;

class Warehouse {

    private static final int MAX\_WEIGHT = 150;

    private int currentWeight = 0;

    private Semaphore semaphore = new Semaphore(3); // 3 грузчика

    public void addWeight(int weight) throws InterruptedException {

        semaphore.acquire();

        currentWeight += weight;

        System.out.println(Thread.currentThread().getName() + " добавил вес: " + weight + ". Текущий вес: " + currentWeight);

        if (currentWeight >= MAX\_WEIGHT) {

            System.out.println("Достигнут максимальный вес. Начинаем перенос на другой склад.");

            unloadGoods();

        }

        semaphore.release();

    }

    private void unloadGoods() {

        currentWeight = 0;

        System.out.println("Товары разгружены на другом складе.");

    }

}

class WarehouseWorker implements Runnable {

    private Warehouse warehouse;

    private int weight;

    public WarehouseWorker(Warehouse warehouse, int weight) {

        this.warehouse = warehouse;

        this.weight = weight;

    }

    @Override

    public void run() {

        try {

            while (true) {

                warehouse.addWeight(weight);

                Thread.sleep(2000);

            }

        } catch (InterruptedException e) {

            Thread.currentThread().interrupt();

        }

    }

}

public class Sklad {

    public static void main(String[] args) {

        Warehouse warehouse = new Warehouse();

        Thread worker1 = new Thread(new WarehouseWorker(warehouse, 50), "Грузчик 1");

        Thread worker2 = new Thread(new WarehouseWorker(warehouse, 60), "Грузчик 2");

        Thread worker3 = new Thread(new WarehouseWorker(warehouse, 40), "Грузчик 3");

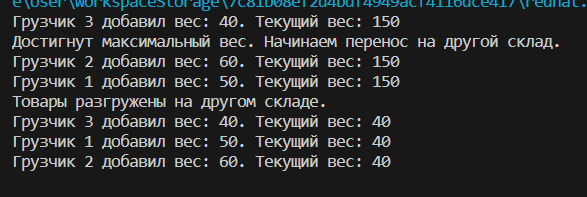
        worker1.start();

        worker2.start();

        worker3.start();

    }

}



1. Класс Warehouse

Класс Warehouse представляет склад и содержит следующие поля:

MAX\_WEIGHT: константа, определяющая максимальный вес груза на складе (в данном случае 150).

currentWeight: текущий вес груза на складе.

semaphore: семафор с ограничением на количество грузчиков (3 грузчика).

Метод addWeight(int weight) throws InterruptedException: этот метод выполняет добавление веса груза на склад. Грузчики используют семафор для контроля доступа к складу. Если текущий вес достигает максимального значения (MAX\_WEIGHT), вызывается метод unloadGoods() для разгрузки на другой склад.

Приватный метод unloadGoods(): этот метод сбрасывает текущий вес груза на складе и выводит сообщение о разгрузке.

2. Класс WarehouseWorker

Класс WarehouseWorker представляет грузчика и реализует интерфейс Runnable.

Конструктор WarehouseWorker(Warehouse warehouse, int weight): принимает ссылку на склад и вес груза, который будет переносить грузчик.

Метод run(): переопределяет метод run() из интерфейса Runnable. Грузчик постоянно добавляет вес груза на склад и засыпает на 2 секунды, имитируя работу.

3. Класс Sklad (главный класс)

Главный класс Sklad создает объект класса Warehouse и несколько потоков грузчиков.

Создаются три грузчика (worker1, worker2, worker3), которые работают с одним и тем же складом.

Грузчики запускаются в отдельных потоках методом start().