МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

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

Кафедра «Математическая Кибернетика и Информационные технологии»

Лабораторная работа №7

Выполнил: Студент группы

БВТ2203

Георгян Сергей

Москва

2023

Задание 1: 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: cоздать 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.

Ход работы:

Задание 1:

import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.concurrent.TimeUnit;  
  
public class ArraySum {  
  
 public static void main(String[] args) {  
 int[] array = {1, 2, 3, 5, 12, 1, 1, 3, 7, 8, 23};  
 int numThreads = 10;  
  
 ExecutorService executorService = Executors.*newFixedThreadPool*(numThreads);  
  
 int chunkSize = array.length / numThreads;  
 for (int i = 0; i < numThreads; i++) {  
 int startIndex = i \* chunkSize;  
 int endIndex = (i == numThreads - 1) ? array.length : (i + 1) \* chunkSize;  
  
 int[] subArray = new int[endIndex - startIndex];  
 System.*arraycopy*(array, startIndex, subArray, 0, subArray.length);  
  
 executorService.execute(new ArrayProcessorTask(subArray));  
 }  
  
 executorService.shutdown();  
 try {  
 executorService.awaitTermination(1, TimeUnit.*HOURS*);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 int totalSum = ArrayProcessorTask.*getTotalSum*();  
 System.*out*.println("Total Sum: " + totalSum);  
 }  
}  
  
class ArrayProcessorTask implements Runnable {  
  
 private int[] array;  
 private static int *totalSum* = 0;  
 private static final Object *lock* = new Object();  
  
 public ArrayProcessorTask(int[] array) {  
 this.array = array;  
 }  
  
 @Override  
 public void run() {  
 int sum = 0;  
 for (int num : array) {  
 sum += num;  
 System.*out*.println(num);  
 }  
 synchronized (*lock*) {  
 *totalSum* += sum;  
 }  
 }  
  
 public static int getTotalSum() {  
 synchronized (*lock*) {  
 return *totalSum*;  
 }  
 }  
}

Файл.1 – программа для подсчета суммы массива

Задание 2:

import java.util.ArrayList;  
import java.util.List;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.concurrent.TimeUnit;  
  
public class MatrixMaxFinder {  
  
 public static void main(String args[]) {  
 int[][] matrix = {  
 {2, 34, 1},  
 {543, 58, 2},  
 {43, 765, 20},  
 };  
  
 List<Integer> oneDMatrix = new ArrayList<>();  
 for (int[] row : matrix) {  
 for (int j : row) {  
 oneDMatrix.add(j);  
 }  
 }  
  
 int numThreads = 3;  
 ExecutorService executorService = Executors.*newFixedThreadPool*(numThreads);  
  
 int chunkSize = oneDMatrix.size() / numThreads;  
 for (int i = 0; i < numThreads; i++) {  
 int startIndex = i \* chunkSize;  
 int endIndex = (i == numThreads - 1) ? oneDMatrix.size() : (i + 1) \* chunkSize;  
  
 List<Integer> subList = oneDMatrix.subList(startIndex, endIndex);  
  
 executorService.execute(new MatrixHandlerTask(subList));  
 }  
  
 executorService.shutdown();  
  
 try {  
 executorService.awaitTermination(1, TimeUnit.*HOURS*);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 int maxElement = Integer.*MIN\_VALUE*;  
 for (MatrixHandlerTask task : MatrixHandlerTask.*getTasks*()){  
 int taskMaxElement = task.getMaxi();  
 if (taskMaxElement > maxElement) {  
 maxElement = taskMaxElement;  
 }  
 }  
  
 System.*out*.println("Max element: " + maxElement);  
 }  
}  
  
class MatrixHandlerTask implements Runnable {  
 private static List<MatrixHandlerTask> *tasks* = new ArrayList<>();  
  
 private List<Integer> array;  
 private int maxi;  
  
 public MatrixHandlerTask(List<Integer> array) {  
 this.array = array;  
 *addTask*(this);  
 }  
  
  
 public void run() {  
 maxi = array.get(0);  
 for (int i = 1; i < array.size(); i++) {  
 int current = array.get(i);  
 if (current > maxi) {  
 maxi = current;  
 }  
 }  
 }  
  
 public int getMaxi() {  
 return maxi;  
 }  
  
 private static synchronized void addTask(MatrixHandlerTask task) {  
 tasks.add(task);  
 }  
  
 public static synchronized List<MatrixHandlerTask> getTasks() {  
 return tasks;  
 }  
}

Файл.2 – программа для нахождения максимального элемента матрицы

Задание 3:

import java.util.ArrayList;  
import java.util.List;  
import java.util.concurrent.CountDownLatch;  
  
public class Transfer {  
 public static void main(String[] args) {  
 List<Integer> weights = new ArrayList<>(List.*of*(20, 30, 50, 70, 20, 40, 80, 110));  
  
 int numWorkers = 3;  
 int maxWeightPerTrip = 150;  
  
 CountDownLatch startSignal = new CountDownLatch(1);  
 CountDownLatch doneSignal = new CountDownLatch(numWorkers);  
  
 List<Loader> loaders = new ArrayList<>();  
 for (int i = 0; i < numWorkers; i++) {  
 loaders.add(new Loader(startSignal, doneSignal, weights, maxWeightPerTrip, i + 1));  
 }  
  
 loaders.forEach(Loader::start);  
 startSignal.countDown();  
  
 try {  
 doneSignal.await();  
 System.*out*.println("Работа выполнена.");  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}  
  
class Loader extends Thread {  
 private CountDownLatch startSignal;  
 private CountDownLatch doneSignal;  
 private List<Integer> weights;  
 private int maxWeightPerTrip;  
 private int workerNum;  
  
 public Loader(CountDownLatch startSignal, CountDownLatch doneSignal, List<Integer> weights, int maxWeightPerTrip, int workerNum) {  
 this.startSignal = startSignal;  
 this.doneSignal = doneSignal;  
 this.weights = weights;  
 this.maxWeightPerTrip = maxWeightPerTrip;  
 this.workerNum = workerNum;  
 }  
  
 @Override  
 public void run() {  
 try {  
 startSignal.await();  
  
 int totalWeight = 0;  
 while (!weights.isEmpty()) {  
 int nextWeight = weights.get(0);  
  
 if (totalWeight + nextWeight <= maxWeightPerTrip) {  
 totalWeight += nextWeight;  
 weights.remove(0);  
  
 System.*out*.println("Рейс " + workerNum + ": товар весом " + nextWeight + " кг. Всего: " + totalWeight + " кг.");  
 } else {  
 break;  
 }  
 }  
  
 System.*out*.println("Рейс " + workerNum + ": успешно донесен товар.");  
 doneSignal.countDown();  
  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}

Файл 3 – программа отвечающая за работу 3 грузчиков

Вывод: Я научился работать с потоками