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

Иляков С.А.

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.

Вариант 2: Использование Runnable: Создайте интерфейс Грузчик и реализуйте его в классе LoaderRealization. Используйте ExecutorService для управления потоками.

Ход работы:

Задание 1:

import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.concurrent.TimeUnit;  
  
public class ArrayProcessor {  
  
 public static void main(String[] args) {  
 int[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
 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(1000, TimeUnit.*SECONDS*);  
 } 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*;  
 }  
 }  
}

Рисунок 1 – Задание 1

Задание 2:

import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.concurrent.TimeUnit;  
  
public class ArrayProcessor {  
  
 public static void main(String[] args) {  
 int[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
 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(1000, TimeUnit.*SECONDS*);  
 } 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*;  
 }  
 }  
}

Рисунок 2 – Задание 2

Задание 3:

public interface Loader extends Runnable{  
 int take();  
 void put(int cargo);  
}

Рисунок 3 – Интерфейс Load

import java.util.Stack;  
  
public class LoaderRealization implements Loader {  
 static int *created* = 1;  
 static int *sum* = 0;  
 static final Object *monitor* = new Object();  
 private static boolean *isFull* = false;  
 private static boolean *isStockEmpty* = false;  
 private static final boolean[] *isCargoEmpty* = new boolean[]{true, true, true};  
  
 Stack<Integer> cargos;  
 Stack<Integer> firstStock;  
 Stack<Integer> secondStock;  
 int id;  
  
 public LoaderRealization(Stack<Integer> firstStock, Stack<Integer> secondStock) {  
 this.firstStock = firstStock;  
 this.secondStock = secondStock;  
 id = *created*;  
 *created*++;  
 cargos = new Stack<>();  
 }  
  
 @Override  
 public synchronized int take() {  
 if (firstStock.isEmpty()) {  
 *isStockEmpty* = true;  
 return -1;  
 }  
 int cargo = firstStock.peek();  
 if (*sum* + cargo > 150) {  
 *isFull* = true;  
 return -1;  
 }  
 cargo = firstStock.pop();  
 *sum* += cargo;  
 System.*out*.println("Loader " + id + " took " + cargo + ". Sum is " + *sum* + " now");  
 *isCargoEmpty*[id-1] = false;  
 return cargo;  
 }  
  
 @Override  
 public synchronized void put(int cargo) {  
 secondStock.push(cargo);  
 *sum* -= cargo;  
 System.*out*.println("Loader " + id + " put " + cargo + ". Sum is " + *sum* + " now");  
 }  
  
 private boolean areCargosEmpty() {  
 return *isCargoEmpty*[0] && *isCargoEmpty*[1] && *isCargoEmpty*[2];  
 }  
  
 @Override  
 public void run() {  
 while (!*isStockEmpty*) {  
 while (!*isStockEmpty* && !*isFull*) {  
 int cargo = take();  
 if (cargo != -1) {  
 cargos.push(cargo);  
 }  
 try {  
 Thread.*sleep*(id \* 500L);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 while (!cargos.isEmpty()) {  
 put(cargos.pop());  
 try {  
 Thread.*sleep*(id \* 500L);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 *isCargoEmpty*[id-1] = true;  
 synchronized (*monitor*) {  
 if (areCargosEmpty()) {  
 System.*out*.println(id + " notified all");  
 *monitor*.notifyAll();  
 } else {  
 System.*out*.println(id + " is waiting");  
 try {  
 while (!areCargosEmpty()) {  
 *monitor*.wait();  
 }  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 }  
 *isFull* = false;  
 }  
  
 }  
}

Рисунок 4 – LoaderRealization

import java.util.\*;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.concurrent.TimeUnit;  
  
public class Task3 {  
 public static void main(String[] args) throws InterruptedException {  
 int[] cargos = new int[]{8, 7, 9, 16, 19, 9, 3, 15, 9, 1, 20, 9, 12, 16, 11, 1, 1, 19, 17, 11, 13, 10, 2, 15, 20, 12, 1, 4, 4, 2, 4, 14, 19, 10, 2, 20, 10, 1, 20, 3, 2, 8, 17, 7, 7, 18, 8, 9, 17, 5, 12};  
 Stack<Integer> firstStock = new Stack<>();  
 for (int cargo :  
 cargos) {  
 firstStock.push(cargo);  
 }  
 Stack<Integer> secondStock = new Stack<>();  
 ExecutorService service = Executors.*newFixedThreadPool*(3);  
 for (int i = 0; i < 3; i++) {  
 service.submit(new LoaderRealization(firstStock, secondStock));  
 }  
 service.shutdown();  
 if (service.awaitTermination(1000, TimeUnit.*SECONDS*)) {  
 System.*out*.println("Program has been completed");  
 System.*out*.println("Second stock: " + secondStock);  
 Arrays.*sort*(cargos);  
 System.*out*.println("Sorted first stock: " + Arrays.*toString*(cargos));  
 System.*out*.println("Sorted second stock: " + secondStock.stream().sorted().toList());  
 }else {  
 System.*out*.println("Program hasn't been completed");  
 }  
 }  
}

Рисунок 5 – Task 3

**Вывод:** Я научился работать с потоками

Ссылка на Git - <https://github.com/Wex187/-.git>