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 №6

Выполнил: Студент группы

БВТ2203

Дворянчиков Даниил

Москва

2023

Задание 1. Реализация многопоточной программы для вычисления суммы элементов массива. Вариант 1. Создать два потока, которые будут вычислять сумму элементов массива по половинкам, после чего результаты будут складываться в главном потоке.

Задание 2. Реализация многопоточной программы для поиска наибольшего элемента в матрице. Вариант 1. Создать несколько потоков, каждый из которых будет обрабатывать свою строку матрицы. 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 CyclicBarrier: Используйте барьеры для синхронизации грузчиков перед отправлением на другой склад.

Выполнение:

1. Создаём два потока и контейнер для хранения результатов. Передаём потокам части массива затем суммируем их.

public class MultithreadedSum {  
  
 public static void main(String[] args)  
 {  
 int[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
 int length = array.length;  
  
 // Разделение массива на две части  
 int mid = length / 2;  
  
 // Создание объекта для хранения результатов каждого потока  
 ResultContainer resultContainer = new ResultContainer();  
  
 // Создание и запуск первого потока  
 Thread thread1 = new Thread(new SumThread(array, 0, mid, resultContainer));  
 thread1.start();  
  
 // Создание и запуск второго потока  
 Thread thread2 = new Thread(new SumThread(array, mid, length, resultContainer));  
 thread2.start();  
  
 try  
 {  
 // Ожидание завершения обоих потоков  
 thread1.join();  
 thread2.join();  
  
 // Получение результатов и вывод суммы  
 int totalSum = resultContainer.getResult1() + resultContainer.getResult2();  
 System.*out*.println("Total Sum: " + totalSum);  
 }  
 catch (InterruptedException e)  
 {  
 e.printStackTrace();  
 }  
 }  
}  
  
// Класс для хранения результатов  
class ResultContainer  
{  
 private int result1;  
 private int result2;  
  
 public synchronized void setResult1(int result1)  
 {  
 this.result1 = result1;  
 }  
  
 public synchronized void setResult2(int result2)  
 {  
 this.result2 = result2;  
 }  
  
 public synchronized int getResult1()  
 {  
 return result1;  
 }  
  
 public synchronized int getResult2()  
 {  
 return result2;  
 }  
}  
  
// Класс, представляющий поток вычисления суммы  
class SumThread implements Runnable  
{  
 private int[] array;  
 private int start;  
 private int end;  
 private ResultContainer resultContainer;  
  
 public SumThread(int[] array, int start, int end, ResultContainer resultContainer)  
 {  
 this.array = array;  
 this.start = start;  
 this.end = end;  
 this.resultContainer = resultContainer;  
 }  
  
 @Override  
 public void run()  
 {  
 int sum = 0;  
 for (int i = start; i < end; i++) sum += array[i];  
  
 // Установка результата в объект ResultContainer  
 if (start == 0) resultContainer.setResult1(sum);  
 else resultContainer.setResult2(sum);  
 }  
}

Протестируем:



1. Создаём массив потоков и передаём каждому свою строку матрицы.

public class MaxElementFinder {  
 static class MaxElementRunnable implements Runnable {  
 private int[] row;  
 private int result;  
  
 public MaxElementRunnable(int[] row)  
 {  
 this.row = row;  
 this.result = Integer.*MIN\_VALUE*;  
 }  
  
 public int getResult()  
 {  
 return result;  
 }  
  
 @Override  
 public void run()  
 {  
 for (int element : row)  
 {  
 if (element > result) result = element;  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 // Пример матрицы  
 int[][] matrix = {  
 {1, 2, 3, 4},  
 {5, 6, 7, 8},  
 {9, 10, 11, 12},  
 {13, 14, 15, 16}  
 };  
  
 // Создаем массив потоков  
 Thread[] threads = new Thread[matrix.length];  
  
 // Создаем массив для хранения результатов каждого потока  
 int[] results = new int[matrix.length];  
  
 // Создаем и запускаем потоки для обработки строк матрицы  
 for (int i = 0; i < matrix.length; i++)  
 {  
 MaxElementRunnable maxElementRunnable = new MaxElementRunnable(matrix[i]);  
 threads[i] = new Thread(maxElementRunnable);  
 threads[i].start();  
 try  
 {  
 threads[i].join(); // Ждем завершения потока  
 }  
 catch (InterruptedException e)  
 {  
 e.printStackTrace();  
 }  
 results[i] = maxElementRunnable.getResult();  
 }  
  
 // Находим максимальный элемент из результатов  
 int maxElement = Integer.*MIN\_VALUE*;  
 for (int result : results)  
 {  
 if (result > maxElement) maxElement = result;  
 }  
  
 System.*out*.println("Наибольший элемент в матрице: " + maxElement);  
 }  
}

Протестируем:



class Warehouse {  
 private static final int *MAX\_WEIGHT\_PER\_LOAD* = 150;  
 private static final int *NUM\_LOADERS* = 3;  
 private static final int *NUM\_PRODUCTS* = 30;  
  
 private static Stack<Integer> *productWeights*;  
 private final CyclicBarrier cyclicBarrier;  
  
 public Warehouse() {  
 *productWeights* = new Stack<>();  
 cyclicBarrier = new CyclicBarrier(*NUM\_LOADERS*, this::unloadTruck);  
  
 // Генерируем случайные веса для товаров  
 Random random = new Random();  
 for (int i = 0; i < *NUM\_PRODUCTS*; i++)  
 {  
 int weight = random.nextInt(50) + 1; // Вес от 1 до 50 кг  
 *productWeights*.push(weight);  
 }  
 }  
  
 public void startLoading() {  
 for (int i = 0; i < *NUM\_LOADERS*; i++) {  
 new Thread(new Loader(i)).start();  
 }  
 }  
  
 private void unloadTruck()  
 {  
 System.*out*.println("Грузчики отправляются на другой склад...");  
 }  
  
 private class Loader implements Runnable {  
 private final int loaderId;  
  
 private static int *totalWeight* = 0;  
  
 private static int *productIndex* = 0;  
  
 public Loader(int loaderId) {  
 this.loaderId = loaderId;  
 }  
  
 @Override  
 public synchronized void run() {  
 while (!*productWeights*.empty())  
 {  
 int productWeight = *productWeights*.peek();  
  
 if (*totalWeight* + productWeight <= *MAX\_WEIGHT\_PER\_LOAD*) {  
 *productWeights*.pop();  
 *totalWeight* += productWeight;  
 *productIndex*++;  
 try  
 {  
 Thread.*sleep*(2000);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 System.*out*.println("Грузчик " + loaderId + " загружает товар с весом " + productWeight + " кг.");  
 }  
 else {  
 // Если суммарный вес товаров превысил лимит, грузчики отправляются на другой склад  
 try  
 {  
 cyclicBarrier.await();  
 }  
 catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 catch (BrokenBarrierException e)  
 {  
 throw new RuntimeException(e);  
 }  
 *totalWeight* = 0;  
 }  
 }  
 }  
 }  
}

Протестируем:

