Министерство цифрового развития, связи и массовых коммуникаций

Российской Федерации

Ордена Трудового Красного Знамени

Федерально государственное бюджетное образовательное учреждение

высшего образования

МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ

Кафедра «Математической кибернетики и информационных технологий»

Лабораторная работа №6­­7

По дисциплине

«Информационные технологии и программирование»

Выполнил:

Студент группы БВТ2203

Бородин К.Н.

Москва 2023

ЗАДАНИЯ

1. Реализация многопоточной программы для вычисления суммы элементов массива. 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.

2. Реализация многопоточной программы для поиска наибольшего элемента в матрице. 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.

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. Используйте семафоры для ограничения доступа к складу и контроля над весом товаров.

ХОД РАБОТЫ.

ЗАДАНИЕ 1.

Создаём массив и заполняем его элементами. Затем задаём количество потоков в пуле и создаём сервис для операции с этими потоками. Определяем размер кусков массива, на которые он будет разбиваться, создаём массив для хранения результата каждого потока. Далее запускам потоки: определяем начало и конеч массива, создаём задачу и помещаем её в пул потоков, а также сохраняем результат. После выполнения завершаем работу пула потоков. Полученные рузультаты суммируем и сохраняем.

Класс выполняемой задачи будет переопределять метод call() так, чтобы вычислялась сумма подмассива.

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;

public class Task1 {

    /\*

    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.

     \*/

    public static void main(String[] args){

        int [] array = new int[10000];

        for (int i=0; i<10000; i++){

            array[i] = i;

        }

        //количество потоков в пуле

        int numThreads = 3;

        // Сам пул потоков

        //executorService описывает сервис для запуска runnable и callable задач

        //в качестве возвращаемого значения Future

        //callable возвращает значение.

        ExecutorService executorService = Executors.newFixedThreadPool(numThreads);

        // мы будем разбивать массив на части. Это часть массива

        int chunkSize = array.length / numThreads;

        //Массив для хранения результатов выполнения каждого потока

        Future<Integer>[] results = new Future[numThreads];

        //запуск потоков для обработки частей массива

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

            int start = i\*chunkSize;

            int end = (i == numThreads-1) ? array.length : (i+1)\*chunkSize;

            // Создаём задачу для каждого потока и передаём ей часть массива

            ArraySumTask task = new ArraySumTask(array, start, end);

            // помещаем задачу в пул потоков и сохраняем результат выопленения

            results[i] = executorService.submit(task);

        }

        // Завершение работы пула потоков

        executorService.shutdown();

        // Суммируем результаты выполнения потоков

        int totalSum = 0;

        try {

            for (Future<Integer> result : results){

                totalSum += result.get();

            }

        }

        catch (Exception e){

            e.printStackTrace();

        }

        System.out.println(totalSum);

    }

}

class ArraySumTask implements java.util.concurrent.Callable<Integer>{

    private int[] array;

    private int startIndex;

    private int endIndex;

    public ArraySumTask(int[] array, int start, int end){

        this.array = array;

        this.startIndex = start;

        this.endIndex = end;

    }

    @Override

    public Integer call(){

        int sum = 0;

        for (int i=startIndex; i<endIndex; i++){

            sum+=array[i];

        }

        return sum;

    }

}

Вывод программы:



ЗАДАНИЕ 2.

Решение задачи аналогично предыдущей за исключением того, что мы не суммируем элементы, а ищем максимальный.

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;

public class Task2 {

    /\*

    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.

     \*/

    public static void main(String[] args){

        int [] array = new int[10000];

        for (int i=0; i<10000; i++){

            array[i] = i;

        }

        //количество потоков в пуле

        int numThreads = 3;

        // Сам пул потоков

        //executorService описывает сервис для запуска runnable и callable задач

        //в качестве возвращаемого значения Future

        //callable возвращает значение.

        ExecutorService executorService = Executors.newFixedThreadPool(numThreads);

        // мы будем разбивать массив на части. Это часть массива

        int chunkSize = array.length / numThreads;

        //Массив для хранения результатов выполнения каждого потока

        Future<Integer>[] results = new Future[numThreads];

        //запуск потоков для обработки частей массива

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

            int start = i\*chunkSize;

            int end = (i == numThreads-1) ? array.length : (i+1)\*chunkSize;

            // Создаём задачу для каждого потока и передаём ей часть массива

            ArrayMaxTask task = new ArrayMaxTask(array, start, end);

            // помещаем задачу в пул потоков и сохраняем результат выопленения

            results[i] = executorService.submit(task);

        }

        // Завершение работы пула потоков

        executorService.shutdown();

        // Суммируем результаты выполнения потоков

        int totalMax = 0;

        try {

            for (Future<Integer> result : results){

                totalMax = Math.max(totalMax, result.get());

            }

        }

        catch (Exception e){

            e.printStackTrace();

        }

        System.out.println(totalMax);

    }

}

class ArrayMaxTask implements java.util.concurrent.Callable<Integer>{

    private int[] array;

    private int startIndex;

    private int endIndex;

    public ArrayMaxTask(int[] array, int start, int end){

        this.array = array;

        this.startIndex = start;

        this.endIndex = end;

    }

    @Override

    public Integer call(){

        int max = 0;

        for (int i=startIndex; i<endIndex; i++){

            max= Math.max(max, array[i]);

        }

        return max;

    }

}

Вывод программы:



ЗАДАНИЕ 3.

Создаю массив, в котором буду храниться веса товаров. Также задаю количество потоков. Создаю переменную для подсчёта суммы. Класс будет описан далее. Создаю семафор для контроля доступа к этой переменной – за раз только один грузчик. Создаю семафор для доступа к товарам. К каждому товару может обратиться только один грузчик. Далее я запускаю поток.

Класс Worker реализовывает интерфейс Runnable, он принимает на вход айди рабочего, массив, с которым будут производиться манипуляции, общуу сумму, семафор для неё и семафоры для каждого товара. Далее переопределяю метод run(). Я циклически буду проходиться по элементам массива бесконечно много раз. Я буду пробовать получать доступ к товарам. Если у меня получиться, то я проверяю могу ли добавить этот товар к общей куче. Если могу, то я отгружаю товары. После единоразового прохождения по массиву я проверю, заблокированы ли все семафоры. Если да, то цикл прерывается.

import java.util.concurrent.Semaphore;

public class WarehouseTransfer {

    public static void main(String[] args) {

        int[] array = {30,30,30,30,30,30, 50, 123, 100, 30};

        int numThreads = 3;

        // Общая переменная суммы

        SumHolder sumHolder = new SumHolder();

        // Создаем семафор для общей переменной суммы

        Semaphore sumSemaphore = new Semaphore(1);

        // Создаем семафоры для каждого элемента массива

        Semaphore[] elementSemaphores = new Semaphore[array.length];

        for (int i = 0; i < array.length; i++) {

            elementSemaphores[i] = new Semaphore(1);

        }

        // Создаем и запускаем потоки

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

            new Thread(new Worker(i, array, sumHolder, sumSemaphore, elementSemaphores)).start();

        }

    }

}

class Worker implements Runnable {

    private final int id; //айди рабочего

    private final int[] array; //Массив, из которого мы будем брать значения

    private final SumHolder sumHolder; //общая сумма

    private final Semaphore sumSemaphore; //семафор для работы с общей суммой

    private final Semaphore[] elementSemaphores; //семафор для доступа к элементама массива

    public Worker(int id, int[] array, SumHolder sumHolder, Semaphore sumSemaphore, Semaphore[] elementSemaphores) {

        this.id = id;

        this.array = array;

        this.sumHolder = sumHolder;

        this.sumSemaphore = sumSemaphore;

        this.elementSemaphores = elementSemaphores;

    }

    @Override

    public void run() {

        while (true) { //идти будет всё по кругу, чтобы гарантированно забрать все грузы со склада

            for (int i = 0; i < array.length; i++) {

                try {

                    int flag=0; //флаг того, что можно грузить товар

                    // Пытаемся захватить семафор для элемента массива

                    if (elementSemaphores[i].tryAcquire()) {//Если получилось, то мы элемент захватили

                        int element = array[i]; //Получаем этот элемента

                        // Захватываем семафор для общей переменной суммы

                        sumSemaphore.acquire(); //Для того, чтобы один грузчик работал с суммой, а не все сразу, мы даём ему доступ

                        if (sumHolder.getSum()+element <=150){//Можем ли мы добавить

                            sumHolder.add(element);

                            System.out.println("Worker " + id + ": Added " + element + ", Current weight: " + sumHolder.getSum());

                        }

                        else{flag=1;elementSemaphores[i].release();}//Если не может, то мы разблокируем этот семафор для других грузчиков

                        sumSemaphore.release(); //Разрешаем другим грузчикам работать с переменной суммы

                        // Если сумма превышает 150, обнуляем сумму

                        if (flag==1) { //отгрузка товара на другой склад

                            sumSemaphore.acquire();

                            sumHolder.reset();

                            System.out.println("Worker " + id + ": can't bring weight. They're/he's going to another warehouse. Current weight = 0");

                            sumSemaphore.release();

                            flag=0;

                        }

                        Thread.sleep(200);//Имитация загрузки, чтобы демонстрировать, что рабочие работают одновременное

                    }

                } catch (InterruptedException e) {

                    e.printStackTrace();

                }

            }

            // Если все семафоры элементов заблокированы, завершаем поток

            boolean allSemaphoresBlocked = true;

            for (Semaphore semaphore : elementSemaphores) {

                if (semaphore.availablePermits() > 0) {

                    allSemaphoresBlocked = false;

                    break;

                }

            }

            if (allSemaphoresBlocked) {

                System.out.println("Worker " + id + " finished its work");

                break;

            }

        }

    }

}

// Несколько потоков будут работать с одной и той же переменной, поэтому необходимо синхронизировать доступ к этому ресурсу

class SumHolder {

    private int sum = 0;

    public synchronized void add(int value) {

        sum += value;

    }

    public synchronized void reset() {

        sum = 0;

    }

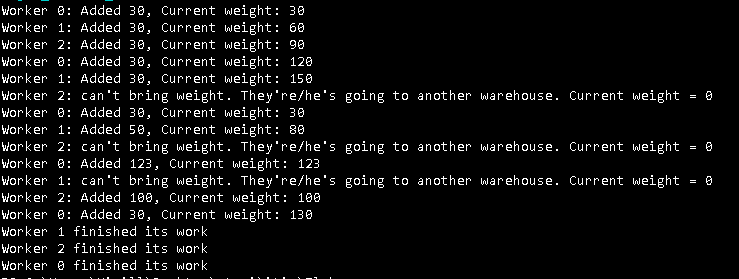
    public synchronized int getSum() {

        return sum;

    }

}

Вывод программы:



Ура ура, уйди