МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего профессионального образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Систем и Технологий

Кафедра «Информационные системы»

Дисциплина «Распределенные вычисления и приложения»

**ОТЧЕТ**

**по лаборатороной работе № 1**

Тема Разработка многопоточного приложения с использованием Java Concurrency согласно варианту задания.

Выполнил студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_Кузнецов В. И.\_\_\_\_\_\_\_/

подпись инициалы, фамилия

Курс\_\_\_\_\_\_\_\_\_\_4\_\_\_\_\_\_\_\_\_\_\_\_\_ Группа\_\_\_\_\_\_\_ПИбд-41\_\_\_\_\_\_\_\_\_\_

Направление/специальность\_\_\_09.03.04. Программная инженерия\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель Андреев И.А.

должность, ученая степень, ученое звание

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

фамилия, имя, отчество

Дата сдачи:

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_201\_\_г.

Дата защиты:

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_201\_\_г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ульяновск

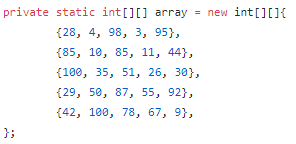
2020 г.

Задание:

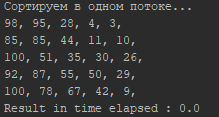
Разработка многопоточного приложения с использованием Java Concurrency согласноварианту задания.

Вариант: отсортировать строки матрицы по убыванию наибольших элементов.

*Исходная матрица*



*Сортировка в одном потоке*

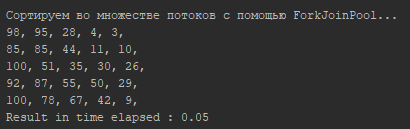


Сортировка в одном потоке.

Для сортировки используется класс реализующий сортировку Слиянием. Циклом проходим по каждой из строк матрицы и сортируем.

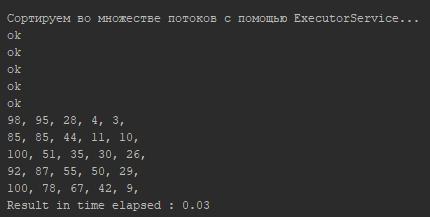
*Сортировка с помощью ForkJoinPool*

Реализация RecursiveTask

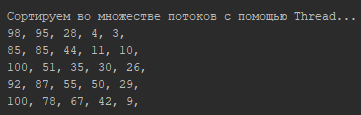


*Сортировка с помощью ExecutorService*

Реализация Callable и использование массива объектов Future.



*Сортировка с помощью нескольких Thread*



*Сортировка с помощью Java Concurrent Collections*

Используется Collections.synchronizedList() и реализация Runnable



Листинг:

import concurrency.RunnableSorting;

import java.util.Arrays;

import java.util.Collections;

import java.util.List;

public class main {

private static int[][] array = new int[][]{

{28, 4, 98, 3, 95},

{85, 10, 85, 11, 44},

{100, 35, 51, 26, 30},

{29, 50, 87, 55, 92},

{42, 100, 78, 67, 9},

};

private static String toStringArray(int[][] array) {

StringBuilder stringBuilder = new StringBuilder();

for (int[] row : array) {

for (int element : row) {

stringBuilder.append(element);

stringBuilder.append(", ");

}

stringBuilder.append("\n");

}

return stringBuilder.toString();

}

private static int[][] getCopy() {

final int[][] copiedArray = new int[array.length][];

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

final int[] row = array[i];

copiedArray[i] = new int[row.length];

System.arraycopy(row, 0, copiedArray[i], 0, row.length);

}

return copiedArray;

}

public static void main(String args[]) {

//

System.out.println("Сортируем в одном потоке...");

int[][] copy = getCopy();

singlethread.MergeSort singlethreadSorted = new singlethread.MergeSort();

String result = singlethreadSorted.SimpleSort(copy);

System.out.println(result + "\n");

//

//

//

concurrency.MergeSort concurrency = new concurrency.MergeSort();

//

//

//

System.out.println("Сортируем во множестве потоков с помощью ExecutorService...");

copy = getCopy();

result = concurrency.SortedWithExecutorService(copy);

System.out.println(result + "\n");

//

//

//

System.out.println("Сортируем во множестве потоков с помощью ForkJoinPool...");

copy = getCopy();

result = concurrency.SortedForkJoinPool(copy);

System.out.println(result + "\n");

//

//

//

System.out.println("Сортируем во множестве потоков с помощью Thread...");

copy = getCopy();

concurrency.SortedWithThreadAndRunnable(copy);

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

System.out.println("Thread has been interrupted");

}

System.out.println(toStringArray(copy));

//

//

//

System.out.println("Сортируем во множестве потоков с помощью Java Concurrent Collections....");

ExampleForSyncCollections();

}

private static void ExampleForSyncCollections() {

List<Integer> list = Collections.synchronizedList(Arrays.asList(28, 4, 98, 3, 95));

for (int step = 0; step < 10; step++) {

Runnable bubbleSorting = () -> {

synchronized (list) {

for (int x = 0; x < list.size(); x++) {

for (int i = 0; i < list.size() - x - 1; i++) {

if (list.get(i).compareTo(list.get(i + 1)) > 0) {

Integer temp = list.get(i);

list.set(i, list.get(i + 1));

list.set(i + 1, temp);

}

}

}

}

};

Thread thread = new Thread(bubbleSorting);

thread.setName("Thread " + step);

thread.start();

}

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

System.out.println("Thread has been interrupted");

}

System.out.println(toStringArray(new int[][]{

list.stream().mapToInt(i -> i).toArray()

}));

}

}

package singlethread;

public class MergeSort {

private int[][] array;

public String SimpleSort(int[][] matrix) {

this.array = matrix;

//

long start = System.currentTimeMillis();

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

array[index] = mergeSort(array[index]);

}

long finish = System.currentTimeMillis();

float timeElapsed = (finish - start) / 100F;

StringBuilder builder = new StringBuilder();

builder.append(toString());

builder.append("Result in time elapsed : ");

builder.append(timeElapsed);

return builder.toString();

}

private int[] mergeSort(int[] input) {

if (input.length <= 1) {

return input;

}

//

int middle = input.length / 2;

int[] left = new int[middle];

int[] right = new int[input.length - middle];

//

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

left[i] = input[i];

}

for (int i = middle; i < input.length; i++) {

right[i - middle] = input[i];

}

//

return merge(mergeSort(right), mergeSort(left));

}

private int[] merge(int[] right, int[] left) {

//

int size = right.length + left.length;

int i = right.length - 1;

int j = left.length - 1;

int[] sorted = new int[size];

//

for (int k = size - 1; k >= 0; k--) {

if (i == -1) {

sorted[k] = left[j];

j--;

continue;

}

if (j == -1) {

sorted[k] = right[i];

i--;

continue;

}

if (right[i] > left[j]) {

sorted[k] = left[j];

j--;

continue;

}

if (right[i] < left[j]) {

sorted[k] = right[i];

i--;

continue;

}

if (right[i] == left[j]) {

sorted[k] = left[j];

j--;

}

}

return sorted;

}

@Override

public String toString() {

StringBuilder stringBuilder = new StringBuilder();

for (int[] row : this.array) {

for (int element : row) {

stringBuilder.append(element);

stringBuilder.append(", ");

}

stringBuilder.append("\n");

}

return stringBuilder.toString();

}

}

package concurrency;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.concurrent.locks.ReentrantLock;

public class MergeSort {

public String SortedWithExecutorService(int[][] matrix) {

//

// Создаём разделяемый ресурс.

//

CommonResource commonResource = new CommonResource(matrix);

int length = matrix.length;

//

// Создаём лист объектов Future от которых будем получать

// данные из запущенных потоков.

//

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

//

// Создаём объект ExecutorService, которые будет управлять

// пуллом запущенных потоков.

//

ExecutorService executor = Executors.newFixedThreadPool(length);

ReentrantLock locker = new ReentrantLock();

//

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

//

// Интерфейс Callable в отличие от интрефейса Runnable умеет возвращать

// значение из потока.

//

Callable<String> callable = new CallableSorting(commonResource, locker, i);

//

// Передадим методу submit интерфейс Callable,

// после итерация продолжится.

//

Future<String> future = executor.submit(callable);

futures.add(future);

}

long start = System.currentTimeMillis();

for (Future<String> future : futures) {

try {

//

// Метод get блокирует текущий поток

// до тех пор пока не получит результат

// от выбранного потока.

//

String result = future.get();

System.out.println(result);

} catch (InterruptedException | ExecutionException e) {

System.err.println("task interrupted");

}

}

long finish = System.currentTimeMillis();

float timeElapsed = (finish - start) / 100F;

//

// Метод shutdown() завершит все потоки.

//

executor.shutdown();

//

//

//

StringBuilder builder = new StringBuilder();

builder.append(commonResource);

builder.append("Result in time elapsed : ");

builder.append(timeElapsed);

return builder.toString();

}

public String SortedForkJoinPool(int[][] matrix) {

//

// Создаём разделяемый ресурс.

//

CommonResource commonResource = new CommonResource(matrix);

int length = matrix.length;

StringBuilder builder = new StringBuilder();

ReentrantLock locker = new ReentrantLock();

var pool = new ForkJoinPool();

//

//

//

long start = System.currentTimeMillis();

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

var sorting = new RecursiveTaskSorting(commonResource, locker, i);

pool.invoke(sorting);

}

long finish = System.currentTimeMillis();

float timeElapsed = (finish - start) / 100F;

//

//

//

builder.append(commonResource);

builder.append("Result in time elapsed : ");

builder.append(timeElapsed);

return builder.toString();

}

public void SortedWithThreadAndRunnable(int[][] matrix) {

//

// Создаём разделяемый ресурс.

//

CommonResource commonResource = new CommonResource(matrix);

ReentrantLock locker = new ReentrantLock();

int length = matrix.length;

//

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

//

// Создаём новый поток Thread, которому передаём интрефейс

// Runnable содержащий разделяемый ресурс.

// Потоки не синхронизованы, поэтому выполняться они будут

// не по порядку как с помощью ExecutorService, а до тех пор

// пока они запущены и не будут остановлены планировщиком задач.

//

Thread thread = new Thread(new RunnableSorting(commonResource, locker, i));

thread.setName("Thread " + i);

thread.start();

}

}

}

package concurrency;

import java.util.concurrent.RecursiveAction;

import java.util.concurrent.locks.ReentrantLock;

public class RecursiveTaskSorting extends RecursiveAction {

private CommonResource resource;

private ReentrantLock locker;

private int indexOfRow;

public RecursiveTaskSorting(CommonResource resource, ReentrantLock locker, int indexOfRow) {

this.resource = resource;

this.locker = locker;

this.indexOfRow = indexOfRow;

}

private int[] mergeSort(int[] input) {

if (input.length <= 1) {

return input;

}

//

int middle = input.length / 2;

int[] left = new int[middle];

int[] right = new int[input.length - middle];

//

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

left[i] = input[i];

}

for (int i = middle; i < input.length; i++) {

right[i - middle] = input[i];

}

//

return merge(mergeSort(right), mergeSort(left));

}

private int[] merge(int[] right, int[] left) {

//

int size = right.length + left.length;

int i = right.length - 1;

int j = left.length - 1;

int[] sorted = new int[size];

//

for (int k = size - 1; k >= 0; k--) {

if (i == -1) {

sorted[k] = left[j];

j--;

continue;

}

if (j == -1) {

sorted[k] = right[i];

i--;

continue;

}

if (right[i] > left[j]) {

sorted[k] = left[j];

j--;

continue;

}

if (right[i] < left[j]) {

sorted[k] = right[i];

i--;

continue;

}

if (right[i] == left[j]) {

sorted[k] = left[j];

j--;

}

}

return sorted;

}

@Override

protected void compute() {

locker.lock();

try {

//

// Читаем из разделяемого ресурса.

// Строка матрицы.

//

int[] matrixRowToSort = resource.matrix[indexOfRow];

int[] sorted = mergeSort(matrixRowToSort);

//

// Изменяем разделяемый ресурс.

//

resource.matrix[indexOfRow] = sorted;

} catch (Exception e) {

System.out.println(e.getMessage());

} finally {

locker.unlock();

}

}

}

package concurrency;

import java.util.concurrent.locks.ReentrantLock;

public class RunnableSorting implements Runnable {

private CommonResource resource;

private ReentrantLock locker;

private int indexOfRow;

public RunnableSorting(CommonResource resource, ReentrantLock locker, int indexOfRow) {

this.resource = resource;

this.locker = locker;

this.indexOfRow = indexOfRow;

}

private int[] mergeSort(int[] input) {

if (input.length <= 1) {

return input;

}

//

int middle = input.length / 2;

int[] left = new int[middle];

int[] right = new int[input.length - middle];

//

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

left[i] = input[i];

}

for (int i = middle; i < input.length; i++) {

right[i - middle] = input[i];

}

//

return merge(mergeSort(right), mergeSort(left));

}

private int[] merge(int[] right, int[] left) {

//

int size = right.length + left.length;

int i = right.length - 1;

int j = left.length - 1;

int[] sorted = new int[size];

//

for (int k = size - 1; k >= 0; k--) {

if (i == -1) {

sorted[k] = left[j];

j--;

continue;

}

if (j == -1) {

sorted[k] = right[i];

i--;

continue;

}

if (right[i] > left[j]) {

sorted[k] = left[j];

j--;

continue;

}

if (right[i] < left[j]) {

sorted[k] = right[i];

i--;

continue;

}

if (right[i] == left[j]) {

sorted[k] = left[j];

j--;

}

}

return sorted;

}

@Override

public void run() {

locker.lock();

try {

//

// Читаем из разделяемого ресурса.

// Строка матрицы.

//

int[] matrixRowToSort = resource.matrix[indexOfRow];

int[] sorted = mergeSort(matrixRowToSort);

//

// Изменяем разделяемый ресурс.

//

resource.matrix[indexOfRow] = sorted;

} catch (Exception e) {

System.out.println(e.getMessage());

} finally {

locker.unlock();

}

}

}

package concurrency;

public class CommonResource {

public CommonResource(int[][] matrix) {

this.matrix = matrix;

}

public int[][] matrix;

@Override

public String toString() {

StringBuilder stringBuilder = new StringBuilder();

for (int[] row : this.matrix) {

for (int element : row) {

stringBuilder.append(element);

stringBuilder.append(", ");

}

stringBuilder.append("\n");

}

return stringBuilder.toString();

}

public String rowToString(int indexOfRow) {

StringBuilder stringBuilder = new StringBuilder();

for (int element : this.matrix[indexOfRow]) {

stringBuilder.append(element);

stringBuilder.append(", ");

}

return stringBuilder.toString();

}

}

package concurrency;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.concurrent.Callable;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\* Пример использования пакета syn

\*/

public class CallableSorting implements Callable<String> {

private CommonResource resource;

private ReentrantLock locker;

private int indexOfRow;

public CallableSorting(CommonResource resource, ReentrantLock locker, int indexOfRow) {

this.resource = resource;

this.locker = locker;

this.indexOfRow = indexOfRow;

}

private int[] mergeSort(int[] input) {

if (input.length <= 1) {

return input;

}

//

int middle = input.length / 2;

int[] left = new int[middle];

int[] right = new int[input.length - middle];

//

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

left[i] = input[i];

}

for (int i = middle; i < input.length; i++) {

right[i - middle] = input[i];

}

//

return merge(mergeSort(right), mergeSort(left));

}

private int[] merge(int[] right, int[] left) {

//

int size = right.length + left.length;

int i = right.length - 1;

int j = left.length - 1;

int[] sorted = new int[size];

//

for (int k = size - 1; k >= 0; k--) {

if (i == -1) {

sorted[k] = left[j];

j--;

continue;

}

if (j == -1) {

sorted[k] = right[i];

i--;

continue;

}

if (right[i] > left[j]) {

sorted[k] = left[j];

j--;

continue;

}

if (right[i] < left[j]) {

sorted[k] = right[i];

i--;

continue;

}

if (right[i] == left[j]) {

sorted[k] = left[j];

j--;

}

}

return sorted;

}

@Override

public String call() throws Exception {

locker.lock();

try {

//

// Читаем из разделяемого ресурса.

// Строка матрицы.

//

int[] matrixRowToSort = resource.matrix[indexOfRow];

int[] sorted = mergeSort(matrixRowToSort);

//

// Изменяем разделяемый ресурс.

//

resource.matrix[indexOfRow] = sorted;

} catch (Exception e) {

System.out.println(e.getMessage());

} finally {

locker.unlock();

}

return "ok";

}

}