Національний Технічний Університет України

Київський Політехнічний Інститут імені Ігоря Сікорського

Інститут Прикладного Системного Аналізу

Кафедра Системного Проектування

Паралельні обчислення

Лабораторна робота №4

Виконав:

Дєрюгін Є.О.

Група: ДА-81

Перевірив: Яременко В.С.

Київ – 2021

ЗМІСТ

[Мета роботи 3](#_Toc68637040)

[Завдання 3](#_Toc68637041)

[Лістинг програми мовою Kotlin 4](#_Toc68637042)

[Результати роботи програми 6](#_Toc68637043)

[Опис роботи програми 7](#_Toc68637044)

[Висновки 8](#_Toc68637045)

Мета роботи

Розробка і реалізація паралельного алгоритму з використанням механізмів асинхронних обчислень.

Завдання

Розробити програму, яка за допомогою Java класу CompletableFuture та відповідних методів асинхронних розрахунків (supplyAsync(), thenApplyAsync() та ін.) виконує завдання згідно варіанту. Можливе виконання мовою програмування за власним вибором з використанням відповідних конструкцій асинхронних алгоритмів.

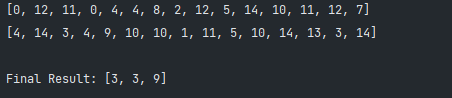
**Варіант 7**

Створити 2 масиви (або колекції) з випадковими числами. У першому масиві – залишити елементи, які більші 0.7 максимального значення масиву, в другому – залишити елементи кратні 3. Відсортувати масиви і злити в один відсортований масив тільки ті елементи, які входять в другий масив і не входять в перший.

Лістинг програми мовою Kotlin

import java.util.Collections.sort  
import java.util.concurrent.CompletableFuture  
import java.util.concurrent.ExecutionException  
import kotlin.collections.ArrayList  
import kotlin.random.Random  
  
  
const val *SIZE*: Int = 15  
  
fun main(args: Array<String>) {  
 // Создаю и заполняю 2 массива случаными числами  
 val a1 = IntArray(*SIZE*) **{** Random.nextInt(*SIZE*)**}**.*toList*()  
 val a2 = IntArray(*SIZE*) **{** Random.nextInt(*SIZE*)**}**.*toList*()  
  
 // Вывод массивов  
 *println*(a1)  
 *println*(a2)  
  
 // Первый фьючерс  
 val firstFuture: CompletableFuture<*List*<Int>?> = CompletableFuture  
 .supplyAsync **{** a1 **}** .thenApplyAsync **{** first: *List*<Int>? **->** // Создаю промежуточную переменную, присваиваю ей  
 // массив содержащий элементы, больше 0,7 максимального значения массива  
 val a: *List*<Int>? = first  
 ?.*filter* **{ it** > 0.7 \* first.*maxOrNull*()!! **}** // Сортируем  
 sort(a)  
 a  
 **}** // Создание второго фьючерса  
  
 val secondFuture: CompletableFuture<*List*<Int>?> = CompletableFuture  
 .supplyAsync **{** a2 **}** .thenApplyAsync **{** second: *List*<Int>? **->** // Создаю промежуточную переменную, присваиваю ей  
 // массив содержащий элементы, кратные 3-м  
 val a: *List*<Int>? = second?.*filter* **{ it** % 3 == 0 **}** // Сортируем  
 sort(a)  
 a  
 **}**// println()  
// println("Result: " + firstFuture.get())  
// println("Result: " + secondFuture.get())  
  
  
 // Финальный фьючерс, в котором получаем финальний массив  
 val finalResultFuture: CompletableFuture<*List*<Int>> = firstFuture.thenCombine(  
 secondFuture  
 ) **{** first: *List*<Int>?, second: *List*<Int>? **->** // Промежуточная переменная равная значению первого массива  
 val a: *MutableList*<Int> = ArrayList(second!!)  
 // Удаляю элементы со второго массива, которые содержатся в первом  
 first.*let* **{** if (**it** != null) {  
 a.removeAll(it)  
 }  
 **}** a  
 **}** // Вывод значений с последнего фьючерса  
 try {  
 *println*("\nFinal Result: " + finalResultFuture.get())  
 } catch (e: InterruptedException) {  
 e.printStackTrace()  
 } catch (e: ExecutionException) {  
 e.printStackTrace()  
 }  
}

Результати роботи програми



Опис роботи програми

Першочергово, підключив необхідні бібліотеки та задав розмір масивів константою SIZE.

Всередині функції **main()** я створив та заповнив 2 масиви випадковими числами та вивів масиви, аби переглянути їх.

Створюю перший *CompletableFuture* - ф’ючерс, який можна завершити явно, для створення якого використовую методи:

* .*supplyAsync* - аби повернути результат роботи ф’ючерса
* .*thenApplyAsync* – аби задача у ф’ючерсі була виконана асинхронно

У тілі ф’ючерса створив лямбда-функцію, в тілі якої я створив проміжну змінну, в яку записав значення масиву **a1**, в якому через .filter залишив елементи, які більші 0,7 максимального значення масиву. Далі відсортував масив у проміжній змінній та задав її результатом ф’ючерса.

Таким самим способом створив другий ф’ючерс, в якому провів операції над масивом **a2**, та залишив тільки ті елементи, що кратні трьом.

Далі я створив фінальний ф’ючерс для об’єднання результатів роботи перших двох ф’ючерсів. Я використав .*thenCombine* для того, щоб дочекатися завершення виконання перших двох ф’ючерсів, для початку виконання фінального ф’ючерса. Також я використав функцію *.removeAll*, яка видаляє з масиву другого ф’ючерса елементи, що входять в масив першого ф’ючерса.

Методом *.get()*, що чекає моменту, коли фінальний ф’ючерс поверне результат, вивожу результат роботи фінального ф’ючерса.

Висновки

У результаті виконання лабораторної роботи було створено програму, яка реалізує завдання згідно варіанту, за допомогою Java класу CompletableFuture та відповідних методів асинхронних розрахунків: .supplyAsync(), .thenApplyAsync(), .thenCombine.

Ознайомився з механізмами створення та оперування ф’ючерсами. Я не стикнувся з проблемами, під час виконання лабораторної роботи.