YSC4231: Parallel, Concurrent and Distributed Programming

Programming Assignment 5

This assignment is dedicated to Scala's `Futures` and `Promise` and consists of three problems that should be solved by means of utilising these mechanisms. The template repository is accessible via the GitHub Classroom link posted on canvas.

While you can use synchronisation mechanisms such as Java's `synchronised` and `AtomicInteger`, you implementation shouldn't contain explicit starting/joining of threads—use `Futures` and `Promises` instead. As usual, portion of this assignment's total score is reserved for additional testing (templates for tests are provided). Some "default" test scenarios are provided, but feel free to add more of your own.

Problem 1. The template for this problem is given in the object `LinkExtractor`. Using Scala's `Futures`, implement its three methods:

- `askForUrl(): Future[String]` asks the user for a URL;
- `fetchUrl(url: String): Future[String]` reads the web page at that URL;
- `getLinks(doc: String): Future[List[String]]` returns all the hyperlinks in the text `doc`.

Each function should use a separate `Future` for each of these three steps. Check your implementation via the `main` method. Some hints:

- Study carefully the tests for programs that rely on Futures, which are available in the code for the lecture: tests that use plain `assert` (not "wrapped" into the future) may behave non-deterministically due to timing issues.
- Make sure to handle possible failures via `recover` method of `Futures`.
- In case of getting `java.nio.charset.MalformedInputException`, look for a solution on the Internet.[1]
- You can detect links via scala regular expressions:

```
val pattern: Regex = """href="(http[^"]+)"""".r // Defines the pattern
```

Next, find out (e.g., via Google) how to find sub-strings matching regex patterns in Scala.

Problem 2. In the object `ServerObserver`, implement the method

```
getLinkHttpServerCounts(url: String): Future[Map[String, Int]]
```

which, given a URL, (a) reads the web page at that URL, (b) finds all the hyperlinks, (c) visits each of them concurrently, and (d) locates the Server HTTP header for each of them. It then collects a map of which servers were found how often. Use Futures for visit each page and return its Server header. Hints:

- Consider reusing some of the functionality from **Problem 1**.
- Feel free to use the provided utility functions `fetchServerName` and `toUrl`.
- Use the `Future.sequence` utility method to combine a *sequence of Futures* into a *future that returns a sequence of values*.
- Avoid using `Await.result` anywhere except for the `main` method to synchronise the futures, as this kills parallelism.

Problem 3. The template is in the file `PalindromeSearch.scala`. Implement the `Future`-returning function `firstPalindromePrime`, whose result finds the *first* prime number in a given range `[from ... upTo]` that is a palindrome, or returns `-1`, if no such number exists. To do so, split the range into sub-ranges that will be searched concurrently by tasks enclosed into Futures. To implement cancellation, use a `Promise`: all tasks should check from time to time if the promise is completed, in which case they should stop their attempts. Some hints:

- The parameter `workers` determines how many Scala `Futures` running concurrently should be allocated.
- Use `tryComplete` and `tryFailure` of the `Promise` for CAS-like installing of a result into the promise.
- A performant concurrent hash map is provided by Scala's `TrieMap`.

[1] https://stackoverflow.com/questions/29987146/using-result-from-scalas-fromurl-throws-exception

- Think if the palindrome-seeking futures need to synchronise with each other or exchange any information.
- Use `BigInt(x).isProbablePrime(10)` to check a number of primality of `x` with high probability.
- Do not use explicit `await`s or `Thread.sleep` except for testing the futures, as it would defeat the point of asynchronous computations.
- That said, a bit of spinning in individual futures (on some variable to be updated) is okay.