Coroutine against Goroutine: comparison between *Kotlin* and *GoLang* concurrency

Activity Project in *Operating Systems M*

Luca Marchegiani

August 14, 2022

Contents

Abstract

In this paper we are going to make a comparison between `Kotlin` and `Go` concurrency that is the main focus of the activity project in *Operating Systems M* course of the master degree in *Computer Science Engineering* at the *Alma Mater Studiorum* University of Bologna.

After a description of the concurrency management in these two languages, we will try to go into an experimental comparison using a previously made projects of the author in the courses of *Software System Engineering* and *Mobile Systems M*.

1 Introduction

`Kotlin` is a modern multiplatform programming language developed by Jetbrains that works on JVM such as `Scala`. `Kotlin` is completely interoperable with `Java`[1] and it is *Object-oriented* with strong elements of functional programming that make it more powerful than his father `Java`. As specified in the main page of the official website, `Kotlin` has also the advantages to be *concise*, *safe in nullability*, *expressive interoperable* and *multiplatform*.

In addition to this, `Kotlin` is now the official `Android` language, and now supports also multiplatform allowing the developer to write `Kotlin` code that can be compiled for native platform (including `Android` and `iOS`), `JVM` and `JavaScript`.

`Go` is an open source programming language developed and supported by `Google`. It's an *imperative* and *object-oriented* language strongly designed for concurrency thanks to its very easy way to launch process. The idea of this language is to maintain the run-time

[1] All classes written in `Kotlin` are callable from `Java` code and vice-versa.

efficiency of `C` but with more readability and usability. Differently from `C`, `Go` has *memory safety*, *garbage collection* and *structural typing* as said by Wikipedia.

Both of this language supports coroutines as concurrent units of execution. *Coroutines* are lightweight processes that can run over multiple OS threads allowing to save on thread management costs.

Coroutines and threads are very similar but the main difference is that the firsts are *non-preemptive* (or *cooperatives*) differently from the seconds that are typically *preemptive* and scheduled by the OS. Indeed, the execution of a coroutine can be suspended and resumed by the developer, calling some operations, and not by the OS.