Part 20: Introduction to Concurrency

27 JUNE 2017

Welcome to tutorial no. 20 in Golang tutorial series.

Go is a concurrent language and not a parallel one. Before discussing how concurrency is taken care in Go, we must first understand what is concurrency and how it is different from parallelism.

What is concurrency?

Concurrency is the capability to deal with lots of things at once. It's best explained with an example.

Let's consider a person jogging. During his morning jog, lets say his shoe laces become untied. Now the person stops running, ties his shoe laces and then starts running again. This is a classic example of concurrency. The person is capable of handling both running and tying shoe laces, that is the person is able to deal with lots of things at once :)

What is parallelism and how is it different from concurrency?

Parallelism is doing lots of things at the same time. It might sound similar to concurrency but its actually different.

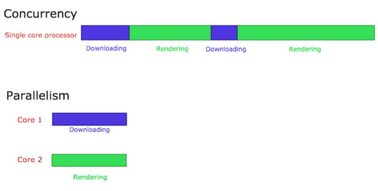
Lets understand it better with the same jogging example. In this case lets assume that the person is jogging and also listening to music in his iPod. In this case the person is jogging and listening to music at the same time, that is he is doing lots of things at the same time. This is called parallelism.

Concurrency and Parallelism - A technical point of view

We understood what is concurrency and how it is different from parallelism using real world examples. Now lets look at them from a more technical point of view as we are geeks :).

Lets say we are programming a web browser. The web browser has various components. Two of them are the web page rendering area and the downloader for downloading files from the internet. Lets assume that we have structured our browser's code in such a way that each of these components can be executed independently (This is done using threads in languages such as Java and in Go we can achieve this using Goroutines, more on this later). When this browser is run in a single core processor, the processor will context switch between the two components of the browser. It might be downloading a file for some time and then it might switch to render the html of a user requested web page. This is know as concurrency. Concurrent processes start at different points of time and their execution cycles overlap. In this case the downloading and the rendering start at different points in time and their executions overlap.

Lets say the same browser is running on a multi core processor. In this case the file downloading component and the HTML rendering component might run simultaneously in different cores. This is known as parallelism.



Parallelism will not always result in faster execution times. This is because the components running in parallel have might have to communicate with each other. For example, in the case of our browser, when the file downloading is complete, this should be communicated to the user, say using a popup. This communication happens between the component responsible for downloading and the component responsible for rendering the user interface. This communication overhead is low in concurrent systems. In the case when components run in parallel in multiple cores, this communication overhead is high. Hence parallel programs do not always result in faster execution times!

Support for concurrency in Go

Concurrency is an inherent part of the Go programming language. Concurrency is handled in Go using Goroutines and channels. We will discuss about them in detail in the upcoming tutorials.

That's it for introduction to concurrency. Please leave your feedback and comments. Have a good day.

Next tutorial - Goroutines

Naveen Ramanathan

iOS developer at dietco.de and Golang enthusiast.

Share this post