Part 3

What is concurrency? What is parallelism? What's the difference?

Concurrency happens when two tasks can start, run and complete in overlapping time periods.

Parallelism is the technique that makes performing several computations at the same time possible, for example using multiple CPUs in parallel, hence making programs run faster.

The main difference is that while concurrence is about dealing with multiple computations at once, parallelism is about performing a lot of computations at once. Concurrency also deals with independently processes, while parallelism simultaneously executes possibly related computations.

Why have machines become increasingly multicore in the past decade?

During the past decade each single core has been developed to be smaller in size and more energy efficient. This makes it possible to replace one old core with multiple new cores, in the same chip, due to the smaller sizes and less energy consumption.

What kinds of problems motivates the need for concurrent execution?

(Or phrased differently: What problems do concurrency help in solving?)

One of the problems that motivates the need for concurrency is that it makes complex algorithms simpler by splitting tasks into separate concurrent executions.

A concurrent approach also helps to utilize the hardware to its maximum capacity.

Does creating concurrent programs make the programmer's life easier? Harder? Maybe both?

(Come back to this after you have worked on part 4 of this exercise)

It is both harder and easier. Harder because we still have to consider the problem with parallelism, like when executing two tasks working on the same global variable needs

some kind of synchronization. However it still makes it possible to split complex algorithms into separated tasks, thereby making the life of the programmer easier.

What are the differences between processes, threads, green threads, and coroutines?

Processes: A series of steps taken in order to achieve a particular end or goal. It can also be described as a collection of threads.

Threads: A way for a program to split itself in multiple simultaneously running tasks.

Green threads: Threads that are scheduled by a runtime library.

Coroutines: A computer program that generalizes subroutines for non-preemptive multitasking, by allowing execution to be suspended and resumed.

Which one of these do `pthread_create()` (C/POSIX), `threading.Thread()` (Python), `go` (Go) create?

pthread_create() creates a new thread.

threading.Thread() also creates a new thread.

go creates coroutines.

How does pythons Global Interpreter Lock (GIL) influence the way a python Thread behaves?

GIL is a mutex, or a lock, that allows only one thread to hold the control of the python interpreter, so only one thread can be in a state of execution at any point in time. This means that the GIL is very unsuitable for cases where a developer wishes to use shared memory threading to exploit multiple cores on a single machine or write their entire application in Python including CPU bound elements.

With this in mind: What is the workaround for the GIL (Hint: it's another module)?

One can use Python multiprocessing module, which can efficiently side-step the GIL using subprocesses instead of threads. The module then allows a programmer to fully leverage multiple processors on a given machine.

What does `func GOMAXPROCS(n int) int` change?

This function sets the maximum number of CPUs that can be executing simultaneously and returns the previous setting. Setting n < 1 does not change the current setting.

Part 4