**Exercise 1: Hello World**

**Part 1: Thinking about elevators**

Not for handing in, just for thinking about. Talk to other groups, assistants, or even people who have taken the course in previous years.

Brainstorm some techniques you could use to prevent a user from being hopelessly stranded, waiting for an elevator that will never arrive. Think about the worst-case behaviour of the system.

* **What if the software controlling one of the elevators suddenly crashes?** The embedded system should have an independent SW code applied in a particular area of the Flash memory (Bootloader), which detects if the main SW code crashes, acting in consequence.
* **What if it doesn't crash, but hangs?** As the system should be programmed in a real time approach, some emergency functions might be activated if the processes do not meet the predefined deadline.
* **What if a message between machines is lost?** One system should be aware of expecting incoming messages from another one, asking for them again and/or executing another function in consequence.
* **What if the network cable is suddenly disconnected? Then re-connected?** There should be an incoming “life signal” value (a 4-bytes counter, for example) in order to know when the cable is disconnected and re-connected.
* **What if a user of the system is being a troll?** The algorithm should be smart enough to be aware of what an external element/user can do into the system, and limit its possibilities.
* **What if the elevator car never arrives at its destination?** A group of sensors (and an efficient algorithm) should be used by the system to know if the destination if never arrived because the elevator is physically not been moved or the condition to confirm the right destination (limit switch, mathematical calculation…) is never high.

**Part 2: Set up some control and build tools**

To understand how the Version Control System (VCS) Git works.

**Part 3: Reasons for concurrency and parallelism**

To complete this exercise you will have to use git. Create one or several commits that adds answers to the following questions and push it to your groups repository to complete the task.

When answering the questions, remember to use all the resources at your disposal. Asking the internet isn't a form of "cheating", it's a way of learning.

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

**Concurrency:** The capability of executing two tasks virtually at same time. They take advantage of CPU time-slicing feature of OS where each task runs part of it and then go to waiting state. When first task is in waiting state, the second task continues its part of task.

**Parallelism:** The capability of executing multiple tasks (or sub-tasks) at the same time using multi-core infrastructure of CPU. Parallelism is a subset of concurrency.

The difference between them is that concurrency does not run various tasks simultaneously, but they are distributed along the time, giving the impression of parallelism to the end user. A single-core architecture is valid enough.

Parallelism really execute different tasks at the same time, so it needs a multi-core infrastructure, assigning one core to each task or sub-task.

* **Why have machines become increasingly multicore in the past decade?**

Since the upper threshold of clock speeds has leveled out during recent years, multi-core processors have become a common means to improve computing performance. While adding more cores does not increase the overall computing performance by a proportional amount (two cores do not equal twice the speed), multi-core processors do provide a substantial performance boost over a single-core CPUs. Additionally, a multi-core processor can run more efficiently than a single processor, since not all cores need to be active unless needed.

* **What kinds of problems motivates the need for concurrent execution? (Or phrased differently: What problems do concurrency help in solving?)**

When tasks need to run independently of each other at the same time. For instance, a web server. The web server needs to be able to address different users at the same time, while not waiting for another users query to complete.

* **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)**

Concurrent programs speed up the system, but makes programming more complicated, as concurrency is non-deterministic and requires extra components (semaphores, mutex) to avoid the systems blocks.

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

1. **Process:** OS-managed truly concurrent (with the suitable hardware support). Each process exists within its own address space.
2. **Thread:** OS-managed, within the same address space as the parent and all its other threads. Truly concurrent, and multi-tasking is pre-emptive.
3. **Green Thread:** These are user-space projections of the same concept as threads, but are not OS-managed. Not truly concurrent, but interleaved or multiplexed instead.
4. **Coroutine:** Not OS-managed. Exactly as threads, except for co-operatively multitasking, and hence not truly concurrent.

Threads are located inside processes.

* **Which one of these do pthread\_create() (C/POSIX), threading.Thread() (Python), go (Go) create?**

pthread\_create() and threading.Thread() create a green thread (non-OS managed, but managed by the user). Go creates a goroutine which is like a green thread managed by the go runtime.

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

GIL is a mutual exclusion (mutex) module that avoids threads to access the same function (or better said Python object) at a time, making one of these two to wait until the first one finishes. This lock is necessary mainly because CPython's memory management is not thread-safe.

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

By using the multiprocessing package one can side-step the GIL by using subprocesses instead of threads.

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

Just for programming language Go, the function GOMAXPROCS sets the maximum number of CPUs that can be executing simultaneously and returns the previous setting. If n<1, it does not change the current setting.

## Part 4: Finally some code

Implement this in C, Python and Go:

global shared int i = 0

main:

spawn thread\_1

spawn thread\_2

join all threads (wait for them to finish)

print i

thread\_1:

do 1\_000\_000 times:

i++

thread\_2:

do 1\_000\_000 times:

i--

### **Delivery**

In this exercise you should take advantage of the starter code found in the subdirectories corresponding to the three different programming languages. Fill out the missing code and make sure your program is working before committing and pushing the changes to your repository.

Finally, create a new file called RESULT.md inside this directory explaining what happens and why. Remember to add, commit and push the new file.