# Operating Systems Lab 1: User Level Programming

## Rules

Please perform the lab in pairs, with the same pairs you enrolled in on Nestor. In the case of a pair, both partners receive the same mark for the labs.

This is the first lab session of the course *Operating Systems*. There are three labs in total. Your final lab grade is computed as a weighted average of your three lab grades using the following weights: 30% for lab 1 and 35% for labs 2 and 3.

Each lab consists of a number of programming exercises. In labs 2 and 3, a (short and compact) report is required (including codes). For the first lab, it suffices to digitally submit your codes. So, there is no need for a report in this session but we do require you to hand in code that is provided with sufficient comment lines that show that you understand the related system calls.

The exercises will be submitted through Themis, available on `https://themis.housing.rug.nl`. Before submitting your code, **make sure to enroll in the correct group first!** Your code will automatically be tested for functionality.

The grade of the first lab will be based on:

- 60% correctness: whether or not you passed all tests in Themis. If you did not pass all test cases, you might still get partial points.
- 40% code quality: coding style, sufficient documentation, etc.

## Exercise 1: Parsing of command line arguments

Make a program that mimics a very minimal shell. The program must accept on its `stdin` a string (optionally between quotes) and it should execute the corresponding command. The program must search in the user's standard search path (you can use the standard library function `getenv()` and the environment variable `PATH`).

**Note** that you may only use the system calls `fork()` and (any variation of) `exec()` to make processes and start executables. It is explicitly not allowed to use the standard library function `system()`.

| **Example 1** | **Example 2** |
|---|---|
| **Input** | **Input** |
| `cat /etc/hostname` | `nonExisting -someParam` |
| **Output** | **Output** |
| `computer-name` | `Command nonExisting not found!` |

## Exercise 2: Inter Process Communication (IPC)

Make a program that simulates a ring of communicating processes. The command accepts on its `stdin` an integer, which denotes the number of processes in the ring (at least 1 and at most 16). Each process communicates uni-directional (so, in one direction: read from the left neighbour, and write to the right neighbour). The communication is started by the initial (oldest) process.

Each process receives from its left neighbour an integer, prints its **relative** process-id and the number on the screen, increases the number by one, and sends the obtained number to its right neighbour. When the value 50 has been reached, all processes should terminate. Make sure, that no zombies processes remain. The relative process-id is defined as the relative position of the process in the ring. The parent is number 0, the right neighbour is 1, and so on.

Some important notes:

- On Themis, `stdout` is redirected to a file, which means that due to buffering the output from the different processes will be ordered by process id. Make sure to disable buffering of `stdout` using `setbuf(stdout, NULL);`.
- Make sure to check that you don't exit with an error, even though the program output seems to be correct. Check the exit code using `echo $?`, this should be 0.

**Example:**
**Input**
```
5
```
**Output**
```
relative pid=0: 0
relative pid=1: 1
relative pid=2: 2
relative pid=3: 3
relative pid=4: 4
relative pid=0: 5
relative pid=1: 6
...
relative pid=0: 50
```

## Exercise 3: File system traversal

Make a program that traverses the file system tree starting from a folder that is given on `stdin` (which can be relative to the folder the program is started in). The program should compare the files it finds for equality, and print all unique pairs of files that have the same content. It should print the paths of these files relative to the folder given as input. Do not worry about the order in which you print these pairs, Themis will automatically sort the output before checking the result.

**Hint:** it does not make sense to compare two files byte-by-byte, if their file size differs.

**Example:**
**Input**
```
./tempFolder
```
**Output**
```
"./hello.c" and "./test/hello2.c" are the same file
"./a.out" and "./test/a.out" are the same file
"./a.out" and "./bin/a.out" are the same file
"./test/a.out" and "./bin/a.out" are the same file
```