Lab Operating Systems
Session 1 version 2: User level programming

Rules

You may perform the labs individually or in pairs (the latter is preferred). 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 sessions in total. Your final lab grade is computed as a weighted average of your three lab grades using the following weights: 30% for Session 1 and 35% for the sessions 2 and 3.
Each session consists of a number of programming exercises. In the sessions 2 and 3, a (short and compact) report is required (including codes). For the first sessions, it suffices to digitally submit your codes. These will be compiled and tested by the teaching assistant on Linux. 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 grade of the first sessions will be based on:

- 60% correctness: determined by performing tests
- 40% code quality: coding style, sufficient documentation, etc.

Please submit your files in a `tar`-file, including a proper `Makefile`. The email address for submitting is `operatingsystems17@gmail.com`.

Exercise 1: Parsing of command line arguments

Make a command `execute` that mimics a very minimal shell. The command must accept on its command line a string (between quotes) and 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 ans start executables. It is explicitly not allowed to use the standard library function `system()`.

Here is the log of an example session:

```
$ execute "/bin/ls -l"
-rwxr-xr-x 1 ronlam ronlam 13339 2010-04-23 17:05 a.out
-rw-r--r-- 1 ronlam ronlam   178 2010-04-23 15:52 test.c
$ execute "ls -l"
-rwxr-xr-x 1 ronlam ronlam 13339 2010-04-23 17:05 a.out
-rw-r--r-- 1 ronlam ronlam   178 2010-04-23 15:52 test.c
$ execute "lss -l"
execute: command lss not found
```

Exercise 2: Inter Process Communication (IPC)

Make a program `ring` that simulates a ring of communicating processes. The command accepts on its command line an integer, which denotes the number of processes in the ring (at least 2 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 process-id and the number on the screen, increases the number by one 1, 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.

Here is the log of an example session:

```
$ ring 5
pid=20235: 0
pid=20236: 1
pid=20237: 2
pid=20238: 3
pid=20239: 4
pid=20235: 5
.......
pid=20235: 50
```

Exercise 3: File system traversal

Make a program `duplicates` that traverses the file system sub tree of which the root is the directory in which it was started. The program should compare the files it finds for equality, and print all unique pairs of files that have the same content.

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

Here is the log of an example session:

```
$ duplicates
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
```