Laboratory Session 2

This session aims at revising multiple processes programming on Linux. The focus is on implementing programs to do some particular tasks using system calls.

Problem 2.1: *Process and process control library in Linux* (0 points)

Linux OS supports several system calls used to generate and control processes.

• Create a child process

```
#include <unistd.h>
pid_t fork(void);

Example:
int pid = fork()
if (pid < 0) {
    perror("fork: cannot create a new process");
    exit(0);
} else if (pid == 0) {
    /* The child process executes here */
} else {
    /* The parent process executes here */
    waitpid(pid, &status[i], 0);
}
```

• Wait for a process to change state

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t waitpid(pid_t pid, int *status, int options);
```

• Execute a file

```
#include <sys/types.h>
int execvp(const char *file, char *const argv[]);
```

On Linux's terminal, try command *man* [*function name*] to refer the manual page of the function.

Problem 2.2: *Multi-process Prime program* (10 points)

Write a C/C++ program that accepts N positive integral numbers from the command line and verifies whether those numbers are prime. The program contains multiple processes and each process verifies a number with the answer at the end. An execution of the program on the command line might look like this:

```
$ mpprime 12 3 19 4
12 is not a prime
3 is a prime
19 is a prime
4 is not a prime
```

The program must handle error situations (including wrong input) in a meaningful way. Make sure the program compiles cleanly with gcc -O2 -Wall -lm.

The solution (only one .c text file) is formatted in *name_id_l2.c, no space* and submitted to the Blackboard system by the end of the lab class. Note that students are responsible for missing/duplicated files due to wrong formats. Copying the whole source code from various sources such as the Internet is disallowed.