Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Spring Term 2015

Operating Systems and Networks Assignment 1

Assigned on: **19th February 2015**
Due by: **27th February 2015**

In the exercise seesions of the Operating Systems and Networks course we will assume you have access to a Linux machine. Furthermore you will need to be able to program in C on that machine, so you will need a text editor and a C compiler as well. We also frequently refer to the *man(ual) pages*, which document almost all of the commands in the userland of a linux distribution, as well as the C API of the kernel and the C standard library, libc.

If you are not familiar with C or programming/using linux, please use online ressources to get up to speed. Not beeing able to try and play with the examples given, i.e., in this exercise sheet, will make it very hard for you to follow along later!

If you do not use linux on your personal computer we recommend using a virtual machine, such as Virtualbox. If you do not want to install linux at all we recommend a linux live cd, such as Knoppix (use the DVD version to make sure you have all required utilities). Alternatively you can use the public computer rooms at ETH.

1 General Operating Systems Questions

a) What is the purpose of having a kernel?

(a) What is the least functionality a kernel has to provide usually (Hint: Usually a minimal kernel provides three properties)?
(b) What does this functionality provide to the rest of the system?
(c) Where does the rest of the system reside?
(d) How does the rest of the system interact with the kernel?
(e) Why does it need to interact with the kernel?

b) The kernel can do and access everything. It exports functionality to user applications by syscalls. Does that mean that every user application can execute code in the kernel by doing a syscall?

c) Can you compile libc without the kernel sources?

2 fork()

Creating new processes in the Unix/Linux world is done using fork(). fork() clones an existing process and adds it to the runqueue, rather then really creating a new one. Since fork clones a process, they both execute the line after the fork() call. Now they need to distinguish whether they are parent or child process. This can be done by checking the return value of fork(): to the parent process, fork() returns the PID of the child process, to the child process, fork() returns 0

2.1 Playing with fork()

2.1.1 Calling fork() once

Create a program which forks itself once. The parent process should output "Im the parent and my child's PID is <pid>". The child should output "I'm the child and my PID is <pid>". Do the PIDs match?

2.1.2 fork() multiple times

What do you expect to happen here? Please explain what you think will happen.

```
int main(int argc, char **argv) {
    while(1) { fork(); }
}
```