System Programming
3rd Laboratory (15 and 17 March 2017)

During this laboratory students should read and understand the man pages of several commands and functions. These functions are listed in the end of each exercise

I

Implement a program that creates 10 children processes. Each of these children should sleep for a random number of seconds (between 0 and 10 seconds).
At the end of the execution, each child should print its PID and the number of seconds it was asleep.
Use the following functions:

- **`fork`**
- **`random`**
- **`sleep`**

Process organizations

II

Change the previous program so that the parent prints in the end the total sleep time for each child.
Each child should return the number of seconds it was asleep.
The parent should wait for the death of each of its child and retrieve the returned value.
The parent should print the information of the dead children (PID and sleep time) before printing the total sleep time.
Use the following functions:

- **`exit`**
- **`wait`**

III

Observe the **lab-3-mult-debug.c** file.
What does it doe?
Compile the file: **`gcc lab3-mult-debug.c -g -o mult-debug`**
Run the program using the command **`mult-debug &`**
Run the command **`ps f`**
What is the state of the **mult-debug** processes?
Send the signal SIGCONT (kill -SIGCONT xxxxx) to the various processes. After each kill observe the result of **`ps f`**
Send the other signals to the various processes. After each kill observe the result of **`ps f`**
Look at the following man pages:

- **`kill`**
- **`ps`**

IV

Since each of the processes are stopped, it is now possible to run the debugger, attach it to any of such processes and resume computation following the following steps:

- start the debugger running **`ddd mult-debug`**
- start the processes issuing the command **`run`**

- take note of the PID of the process you want to debug
- execute **ps f** in the terminal
- detach from the current process issuing the command **detach**
- attach to one of the processes command **attach** or:
  - **`menu File -> menu attach -> select process`**
- press **`Step`** (3 times) until the program restarts executing step by step just like in a regular debug session

V

Implement a program that continuously prints increasing numbers and at random intervals (from 0 to 10 seconds) resets the counter.
Use the following functions:

- **`alarm`**
- **`sigaction`**

VI

Implement a simple batch system that executes commands defined in a script file and accounts for total execution time. This system will perform the following operations:

- sequentially read a script file containing the commands to be executed
- execute all of the commands, commands should be executed by the order they appear on the script file.
- the commands execution should not overlap: one command should only be started after the conclusion of the previous
- the system should measure the total execution **time** for each command and print the overall time after the last command

Use the following functions:

- you may want to look at **`man system`**
  - on opensuse it presents the following line
    - **`execl("/bin/sh", "sh", "-c", command, (char *) 0);`**
- **`fork`**
- **`execve`**
- **`sigaction`**
- **`clock_gettime`**

Why does the **`clock`** function does not work?