Operating Systems
Lab-3: Processes

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. To the child process, fork() returns 0. There are some sample programs available on the course webpage. You are advised to run all those programs before beginning with the following questions.

1. Use the *ps*, *ps lx*, *ps tree* and *ps -aux* command to display the process attributes.

2. Learn the top command to display the resource utilization statistics of processes:

   - Open a terminal and type the top command
   - Start a browser and see the effect on the top display
   - Compile a C program and observe the same effect (Use a long loop - say while(1) to observe the effect)
   - From the top display, answer the following:
     - How much memory is free in the system?
     - Which process is taking more CPU?
     - Which process has got maximum memory share?
   - Write a CPU bound C program and a I/O bound C program (e.g. using more printf statements within while(1) loop), compile and execute both of them.

3. Write a program in C that creates a child process, waits for the termination of the child and lists its PID, together with the state in which the process was terminated (in decimal and hexadecimal)

4. Test the codes for creation of orphan process and zombie process given in the reading resource section of Assignment 3 in the course website.

5. Write a C program such that it forks a new process. Then the parent process and the child process should create one more process such that the program in all has four running processes. Each process should print its process ID and its parent process ID. Draw process hierarchy starting from parent process.

6. In a C program, print the address of the variable and enter into a long loop (say using while(1)).

- Start three to four processes of the same program and observe the printed address values.
- Show how two processes which are members of the relationship parent-child are concurrent from execution point of view, initially the child is copy of the parent, but every process has its own data.

7. Test the source code below:

```
for(i = 1; i  10; i + +){
   fork();
   printf(The process with the PID=%d,getpid());
}
```

   In the next phase, modify the code, such as after all created processes have finished execution, in a file process management.txt the total number of created processes should be stored.

8. Write two programs: one called client.c, the other called server.c. The client program lists a prompter and reads from the keyboard two integers and one of the characters $+$ or $-$. The read information is transmitted with the help of the system call excel to a child process, which executes the server code. After the child (server) process finishes the operation, it transmits the result to parent process (client) with the help of the system call exit. The client process prints the result on the screen and also reprints the prompter, ready for a new reading.

9. Write a C program that takes a file name as a command line parameter and sorts a set of integers stored in the file (use any sorting method). You can assume that the file will always be there in the current directory and that it will always contain a set of integers (maximum no. of integers is 1000). The sorted output is written to the display and the input file is left unchanged. Compile the C file into an executable named "sort1". Name the C file sort1.c. Now write a C program (xsort.c) that implements a command called "xsort" that you will invoke from the shell prompt. The syntax of the command is "xsort <filename>". When you type the command, the command opens a new xterm window (terminal), and then sorts the integers stored in the file <filename> using the program "sort1". Look up the man pages for xterm, fork and the different variations of exec* system calls (such as execv, execve, execlp etc.) to do this assignment. Submit the C files sort1.c and xsort.c.