Lab N°3.1 : Unix / Linux Processes

Exercise IV.1: (Signal Handling)

Q1) Create a program "**Prog.c**" that loops to empty infinity

Q2) Compile and run the "**Prog.c**" program.

Q3) Stop this process by two methods?

- Using the keyboard.
- Using commands in another **Shell**

Exercise IV.2: (Signal Handling)

Write a **C** program that creates a child process.

- The **child process** must send a personal signal to the **parent** process.
- The **parent process** should display "**Signal received**" when it receives the signal.

Exercise IV.3: (Signal Management Using the Infinite Loop)

Write a **C** program that creates a process.

- The process must execute **an infinite loop** in which it waits for a signal ("**SIGINT**").
- When it receives the signal, it should display "**Interrupt signal received**" and terminate.

Exercise IV.4: (Signals & Shell)

Q1) Write a program that displays "**Good morning**" when it receives the **USR1** signal, "**Good evening**" when it receives the **USR2** signal and exits after displaying "**End of program**" when **Ctrl-C** is pressed on the keyboard.

Q2) Test the program by sending the following signals from the shell

Exercise IV.5: (Signals & Zombie Processes)

Write a program that creates a child. The parent diverts the **SIGCHLD** signal to delete the zombie child, then it forks. The child displays its **pid**. The program will not terminate; it must be **killed** from the command line.

Exercise IV.6: (Synchronization)

Write a program in which a process creates a child and initializes a handler (display **HELLO**) on **SIGUSR1**. The child displays information on the screen and then sends the **SIGUSR1** signal to its parent.

- Note that the child program must terminate before the parent process.
- The output (messages on the screen) of the program should look like this:

```
Parent: I am process 27406
Child : I am process 27407
Parent: I am waiting for a HELLO signal
Child : End of process
Parent: End of process
```

Exercise IV.7: (Pipe & One Child Process)

Write a C program that creates a pipe, then creates a child process. The parent process sends the message "**Hello!**" to the child process via the pipe, and the child process reads the message and displays it on the screen.

Exercise IV.8: (Pipe & Two Child Processes)

Write a **C** program that creates a pipe, then creates two child processes, **P1** and **P2**. Process **P1** sends even numbers from **1** to **10** to process **P2** via the pipe, and process **P2** reads the numbers and displays them on the screen.

Exercise IV.9: (Two Pipes)

Write a **C** program that creates two pipes, then creates three child processes (**P1**, **P2** and **P3**).

- Process **P1** sends numbers from **1** to **10** to process **P2** via the first pipe.
- Process **P2** multiplies these numbers by **2** and sends them to process **P3** via the second pipe.
- Process **P3** displays the final numbers.

Exercise IV.10: (Inheritance of an Ordinary Pipe)

Write a program that creates an ordinary pipe and then a child.

Successively, the parent and child exchange messages ("**Hi Dad**" and "**Hi Child**").

Exercise IV.11: (Pipe Without Writers)

Q1) Write a program that creates an ordinary pipe **p** and reads from **p[0]**.

Q2) What happens if the program does not close **p[1]** before reading from **p[0]**?

Q3) Same question if the program closes **p[1]** before reading from **p[0]**?

Exercise IV.12: (Pipe Without Readers)

Q1) Write a program that creates an ordinary pipe **p** and writes to **p[1]**.

Q2) What happens if the program does not close **p[0]** before writing to **p[1]**?

Q3) Same question if the program closes **p[0]** before writing to **p[1]**?