Signals **SCOMP-PL1b**

1. On a terminal of your Linux machine obtain the names and integer values of the signals.
   Sugestion: use the **"kill -l"** to list all the signal. Check with **"man kill"** the options and functionalities of this command.

2. Consider the following code:

```
...
int main() {
    for(;;){
        printf("I Love SCOMP!\n");
        sleep(1);
    }
}
...
```

   Identify all the steps necessary to:

   a. On a terminal, compile and run the program.

   b. Open another terminal and through the **kill** command send a signal that suspends the process that is displaying the "I Love SCOMP!" message. Check that the process became "stopped". Which signal made this possible?

   c. Now, send another signal that will allow the previous process to resume its normal execution. Check that the process left the stopped state. Which signal is used?

   d. By using signal, end this process.

   e. Write a program that by using the signals primitives is able to send a specific signal to a process. Both values should be read before sending.

   f. Apply the previously implemented program in e. to replace the **kill** command and repeat the item on a. and items b. through d.

3. Check the differences between the **signal** and **sigaction** functions.
   a. Which is most suitable to handle signals? Justify your choice.
   b. Document with detail all the parameters of **sigaction** as well as its possible values.

c. Write a program that when the SIGUSR1 signal is received the message "SIGUSR1 has been captured and was sent by the process with PID XX" where XX is the process PID which sent the SIGUSR1 signal.
d. On the function code which handles the signal is it possible to use global or static variables safely? Justify your answer.
e. Which functions can be called in an adequate manner on a signal handler? (Sugestion: "man signal-safety").

4. Generally, one of the following behaviors can be associated to a signal: ignore the signal, resume a process (previously "stopped"), end a process, stop a process (marked with "T" state on the process table – can be visualized with **ps** command, check with **"man ps"** the section "PROCESS STATE CODES"), end a process and generate a "core dump" by the reception of a SIGQUIT signal.

   Check and document the default behaviors for some signals of your Linux machine, e.g., SIGHUP, SIGINT, SIGQUIT, SIGILL, SIGABRT, SIGTRAP, SIGFPE, SIGKILL, SIGSEGV, SIGPIPE, SIGALRM, SIGTERM, SIGUSR1, SIGUSR2, SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU.

   Do it by creating a table such as:

Signal	Number	Action
SIGINT	**2**	End the process; Interruption generated when doing CTRL-C on the keyboard.

5. Consider the following code:

```
...
int main() {
    for(;;){
        printf("I Like Signal\n");
        sleep(1);
    }
}
...
```

a. On a terminal compile and run the program. On the same terminal press CTRL-C. What happened? For the explanation bear in mind the reception of a signal and action performed.

b. Change the program so that each time a SIGINT signal is received the "I won't let the process end with CTRL-C!" message is displayed. Recompile and run the program and test by pressing several times CTRL-C.
c. On the terminal where the process was launched in b. press the CTRL-\ keys. What happened? For the explanation bear in mind the reception of a signal and action performed.
d. Change the program so that each time a SIGQUIT signal is received the "I won't let the process end by pressing CTRL-\!" message is displayed. Recompile and run the program and test by pressing several times CTRL-\.
e. On another terminal send, by using the **kill** command a SIGINT and a SIGQUIT signal to the process created in d. What happened?
f. Press CTRL-Z to suspend the process. Access the process table to check the state of the process as well as its PID. Use the **kill** command to kill the suspended process.

   Sugestion: the **"kill %1"** can be used. Execute the **"jobs"** command. What is the meaning of **"kill %1"**?

g. On the signal handlers instead of **printf** the **write** function should be used. Why is that?

6. Write a program that:
   a. Assigns the **handle_signal()** function to SIGUSR1 signal. The function should increase the USR1_COUNTER and display the "SIGUSR1 signal captured, USR1_COUNTER = XX".
   b. Block all signals whenever SIGUSR1 is received by the process. While the process is executing the instructions for the SIGUSR1 handler, no other signal (and even the SIGUSR1 signal) may interrupt the code associated with the handler. The program will have an infinite loop and each iteration should display the "I'm working!" message and call the **sleep(1)** function. To test the code, send signal from another terminal.
   c. Change the program from a. so that a child process is created, and within that process, 12 signals are sent to the parent process (e.g., SIGUSR1, SIGINT, *etc.*) with a 10ms interval between them.

      (Sugestion: **"man nanosleep"**)

d. Change the SIGUSR1 handler to take more than 1 second to be executed. Recompile and run the program and check for differences.
e. Change the program so that no signal is blocked, recompile and run it again. Compare the results of this change facing the results from item d.

Note: To make sure that all works as intended, a *volatile sig_atomic_t* variable type should be used.

Suggestion: check for the behavior of the SA_NODEFER flag.

7. Using **sigprocmask**, **sigismember** and **sigfillset** functions implement a function that lists all signals blocked when a process receives the SIGUSR1 signal. Are all signals blocked? Justify your answer mentioning those who aren't eventually blocked.

   Suggestion: use NULL as value for the second parameter of the **sigprocmask** function.

8. Write a program that:
   - Creates 5 children processes. Each process I presents on the screen the numbers of the interval [i*200, (i+5)*200].
   - The parent process is on a loop executing the **pause()** function. Use signals from the children process to the parent to state that a children has ended, update a variable that is still counting how many children are still executing. The loop of the parent process should end when no more children are executing and only after this happens the **wait()/waitpid()** function can be called five times.

     Suggestion: the SA_NOCLDWAIT and SA_NOCLDSTOP flags will be useful with the resolution. To guarantee that all works the best way, a *volatile sig_atomic_t* variable type should be used.

9. Write a program that creates a process and:
   - The parent process executes task A for 3 seconds, presenting the seconds passed, and afterward sends a SIGUSR1 signal to its child.

- The child process executes task B for 1 to 5 seconds, and the duration is randomly generated.
- The child process will then execute task C. This task can only be executed once there's a notification that task A has ended.

  Suggestion: To guarantee that all works the best way, a global *volatile sig_atomic_t* variable type should be used to verify task A execution.

10. Write a program that:
    - Reads a string from the keyboard-
    - After being read, the program should count the number of characters and display the number on the screen. After that the **sleep(**20**)** function should be executed to simulate the execution of other operations and in the end display the "Successful execution!" message.
    - The user has 10 seconds to input the string. If that time passes and no string is inputted the "To slow, that is why the program will end!" message is displayed and execution ends.

    a. Implement a solution without creating processes.
       Suggestion: you can use the **alarm()** function.
    b. Implement a solution that uses processes to manage the time limit.

11. Consider an array of commands in which each position has the command name and the maximum execution time. If a particular command exceeds its time limit, it should be ended and a message "The command XX didn't end in its allowed time!" where XX is the command name.

    Implement a program that sequentially executes all the array's commands according to the described work mode. Use the processes primitives, signals, and **exec** functions. The array should be of the following structure type:

```
typedef struct {
    char cmd[32];
    int tempo;
} comando;
```

    Note: In order to test the program, other programs can be implemented that performs as follows. They display its name on the screen, sleep for N second and present the message "Execution ended!".

12. A software house wants to evaluate the application's feasibility for a new algorithm to process data in order to extract some kind of information. For that, it is intended to use this new algorithm in a simulation that uses parallel/concurrent programming. The program to be implemented should create 50 processes and test the usefulness of the new algorithm according to the following procedure:
    - Each process starts by executing simulate1() function. This function returns true if data is found or false if data wasn't found. When its execution ends, each child process sends a SIGUSR1 signal if the function succeed, or a SIGUSR2 signal if the function failed, to the process that's managing the simulation. After that, the process should wait for a signal from the managing process to start executing simulate2() function.
    - The parent process will manage the simulation by running the following procedure after 25 processes have finished:
        i. Suppose at that moment none of the searches have been done successfully. In that case, the parent process will display an "Inefficient algorithm!" message and terminate all remaining children processes that are still running.
        ii. But, if at that moment at least one child process has done a successful search, the parent process should send a signal to all the remaining running children processes to start the execution of the **simula2()** function even if it means interrupting the **simula1()** function.

    Note: use randomly generated values to implement both **simula1()**, and **simula2()** functions. To test all the procedure described, the simula1() function can be changed to guarantee that both scenarios will occur.