Signals

Orlando Sousa, António Barros, Luis Lino Ferreira, André Andrade, Carlos Gonçalves, Jorge Pinto Leite, Nuno Morgado, David Freitas, Luis Miguel Pinho

February 2023

1. Open a terminal and type the command `kill -l` to list all the signals the host operating system supports.

   Type `man kill` to read the manual page and learn about the options and functionalities of this command.

2. The following program prints the sentence *"I love SCOMP"* on the screen every second, indefinitely.

```c
#include <stdio.h>
#include <unistd.h>

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

```

   Write this program in a source code file, and proceed as follows.

   (a) On one terminal, compile and run the program.

   (b) Open another terminal, and type the command `ps` to discover the PID of the process in which the program displaying the "I love SCOMP!" message is executing.

   (c) Type the command `kill -s STOP target_PID` to send the STOP signal and suspend that process.
   Note: `target_PID` is the PID of the process that will receive the signal.

   (d) Type the command `ps -l` to check that the process became "stopped" (status should be "T").

   (e) Send the signal `CONT` such the process resumes its normal execution. Check that the process left the "stopped" state.

3. Write a program that sends a specific signal to a process. The program asks the user for the target PID and the number of the signal to send.

   Use the previous question approach to test this program.

   Suggestion: Type the command `man 2 kill` to consult the manual page of the `kill()` system call.

4. Check the documentation to determine the differences between the `signal()` and `sigaction()` functions.

   (a) Which of the two is more suitable for handling signals? Justify your choice.

   (b) Document with detail all the parameters of `sigaction()`, as well as its possible values.

5. Typically, the following actions can be assigned for handling a signal:

   - terminate a process,
   - ignore the signal,

- stop a process, (The process becomes marked with the "T" state on the process table, which can be visualized with `ps` command. Check with `man ps` the section about "PROCESS STATE CODES".)
- resume a process (previously "stopped"),
- end a process and generate a *core dump* by the reception of a `SIGQUIT` signal.

Check and document the default behaviours for some signals of your POSIX 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 pressing CTRL-C.
...		

6. Recall the meaning of *global* variables and *static* variables. Is it safe to use global or static variables inside a signal handler? Justify your answer.

7. *Reentrant* and *non-interruptable by signal* functions can be called safely inside a signal handler because they will not present unexpected behaviour.

   A reentrant function does not use global or static data and does not call non-reentrant functions. As such, its behaviour is predictable, even if interrupted by a new signal.

   A non-interruptable by signal function executes in their entirety, even if a new signal is received during its execution. As such, it is not affected by assynchronous signals.

   (a) Check which functions are safe to use inside a signal handler. On Linux, consult the signal safety manual page by typing the command `man signal-safety`; on other Unix operating systems, consult the `sigaction` manual page.

   (b) Why you should not use the `printf()` function inside a signal handler and, on the other hand, it is safe use the `write()` function?

8. Write a program that when the `SIGUSR1` signal is received, the message *"I captured a SIGUSR1 sent by the process with PID XX"*, where *XX* is the PID of the process which sent the `SIGUSR1` signal.

   Suggestion: You can use `sprintf()` to easily write a text into an array of chars, and then use the `write()` system call to output that text, as the example in the following listing.

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>

void foo() {
    char msg[80];

    sprintf(msg, "Here is a number: %d\n", 13);
    write(STDOUT_FILENO, msg, strlen(msg));
}

```

9. Write the following program and compile it.

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    for( ; ; ) {
        printf("I like signals\n");
        sleep(1);
    }
}

```

(a) The usual terminal line discipline assigns control characters to the SIGINT SIGKILL and SIGTSTP signals:

- CTRL-C sends SIGINT;
- CTRL-\ sends SIGQUIT;
- CTRL-Z sends SIGTSTP.

Confirm these associations typing the command `stty -a`.

(b) Run the program. Once the program is executing, press CTRL-C.

Explain what happened, considering the reception of the signal and the action performed.

(c) Modify the program to print the "I won't let the process end with CTRL-C!" message each time a `SIGINT` signal is received. Recompile and run the program, then test it by pressing CTRL-C several times.

(d) 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.

(e) 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-\.

(f) Open another terminal and send a SIGINT signal to the target process using the kill command. What happened?

(g) This time, send a SIGQUIT signal to the target process, instead. What happened?

(h) Start the program again and perform as follows.

i. Press CTRL-Z to suspend the process.

ii. Access the process table with command `ps` to check the state of the process as well as its PID.

iii. Execute the `jobs` command and check the job number (between square brackets) of the process.

iv. Type the `kill %1` command to terminate the suspended process.
What is the meaning of `%1` in this last command?

10. Write a program that executes an infinite loop. For each iteration of the loop, the program displays the message "I'm working!" and calls the `sleep()` function to pause for one second.

(a) Add to your program the `handle_USR1()` function to handle the `SIGUSR1` signal. This function increments the global `USR1_counter` global variable and displays the message "SIGUSR1 signal captured: USR1_counter = XX".

Note: The variable should be declared as `volatile sig_atomic_t USR1_counter`.

- *The* `volatile` *keyword indicates the compiler that the variable may be changed asynchronously, so any code optimisation on this variable must be avoided.*
- *The* `sig_atomic_t` *type is an integer that is loaded/stored from/to memory in a single instruction.*

(b) Set your program to block all signals whenever the `SIGUSR1` is handled, i.e. no other signal (not even the `SIGUSR1` signal) can interrupt `handle_USR1()` during its execution.

Compile and test your program, sending signals from another terminal.

(c) Modify your program to create a child process before entering the infinite loop. The child sends to his parent a burst of 12 SIGUSR1s with a 10 ms interval between consecutive signals; finally, the child sends a SIGINT.

Test this version of your program.

(Suggestion: Check the `nanosleep()` *function manual page.)*

(d) Change the `SIGUSR1` handler to take more than 1 second to be executed.

Recompile, 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 with the previous experiments.
Suggestion: check for the behaviour of the `SA_NODEFER` flag.

11. 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 by mentioning those which are not eventually blocked.
Suggestion: use `NULL` as the value for the second parameter of the `sigprocmask` function.

12. Write a program that meets the following requirements.
    - The program starts by generating five children processes.
      Each child $i \in \{0, 1, 2, 3, 4\}$ prints on the screen the sequence of all integer values in the interval $[i \times 200, (i + 5) \times 200[$.
      Once the job is complete, the child notifies the parent with a SIGUSR1 before exiting.
    - The parent process enters a loop executing the `pause()` function.
      The parent updates a variable – of type `volatile sig_atomic_t` – that counts how many children are still executing.
      The parent exits the loop when no more children are executing. Only then, the parent calls the `wait()`/`waitpid()` function five times.

    Suggestion: The `SA_NOCLDWAIT` and `SA_NOCLDSTOP` flags will be useful for solving this problem.

13. Write a program that creates a child process.
    - **Workflow of the parent process.**
      The parent executes task A for 3 seconds.
      Once task A finishes, the parent displays on the screen the message "Task A: done!" and notifies the child sending it a `SIGUSR1` signal.
      Finally, the parent waits for the child to complete, to display the message "Job is complete!".
    - **Workflow of the child process.**
      The child executes task B for a random amount of time, between 1 to 5 seconds.
      Once task B completes, the child displays on the screen the message "Task B: done!".
      Once tasks A and B are complete, the child will execute task C for 1 second.
      When task C is complete, the child displays the message "Task C: done!" and exits.

14. Write a program that (1) reads a string from the keyboard and (2) displays the size of the string on the screen.
    However, the user has 10 seconds to input the string. After this time limit, the program displays the message "You were too slow and this program will end!" and exits.

    (a) Implement a solution that solves the problem with a single process.
    Suggestion: Use the `alarm()` function.

    (b) Implement a solution that uses two processes to manage the time limit.

15. Consider an array of commands in which each position has the command name and the time limit (in seconds) to complete. This is expressed in the defined type `command_t`:

```c
typedef struct {
    char cmd[32];  // Command name.
    int time_cap;  // Time limit to complete (in seconds).
} command_t;

```

A command is terminated if it exceeds its time limit; additionally, the message "The command *XX* did not complete in its allowed time!" (where *XX* is the command name) is displayed.

Implement a program that sequentially executes all the commands in the array, enforcing the time limits.

Suggestion: In order to test the program, you can develop another program that (1) display its name on the screen, (2) sleeps for 10 seconds and finally (3) present the message "Execution ended!".

16. A software house wants to evaluate the potential of a new algorithm to process data in parallel to extract valuable information in a faster way than the solution in production. For this purpose, this algorithm will be tested in a simulated application that uses parallel/concurrent programming.

    You should develop a program where the master process (i.e. the parent) creates 50 worker processes (i.e. children).

    - **Workflow of the worker processes.**

      Each worker process starts executing of the `simulate1()` function. This function returns 1 (one) if it finds relevant data or 0 (zero) if not.

      When this function completes, the worker process sends the parent process a `SIGUSR1` signal in the case of sucess or a `SIGUSR2` signal otherwise.

      After that, the process will wait for a signal from the master process before starting to execute `simulate2()` function.

    - **Workflow of the master process.**

      The master process waits for the notifications sent by the workers. After 25 processes have finished, the master process will take one of two possible actions:

      (a) If none of the reported searches are successful, the parent process will display the message "Inefficient algorithm!" and terminate all children.

      (b) Otherwise, the master process will send the SIGUSR1 signal to all the worker processes. Every worker process will immediately start the execution of the `simulate2()` function, even if it means interrupting the `simulate1()` function.

    Use randomly generated values to simulate the execution time and the result of both `simulate1()` and `simulate2()` functions.

    Set a very low probability of success for the `simulate1()` function to ensure both scenarios will occur.