Laboratory #9 – Semaphore – May 19th 2021

Exercise 1– Create a multi-thread program composed of 2 threads. Each thread must simply print on the screen a message reporting its thread id. The threads must be synchronized in such a way that messages are alternated on the screen, i.e., message from thread 1 always followed by message from thread 2.

Exercise 2– Create a multi-thread program that writes information in a shared file. The main program creates an empty file whose name is provided by the user. Then, it creates N threads (with N still specified by the user).
Each thread performs 100 times the following operations:

1. Open the file.
2. Read the last line of the file (each line contains a number).
3. Add 3 to the number.
4. Append the new number at the end of the file.
5. Close the file.

Make sure that steps from 1 to 5 are executed in an atomic way by the different processes.
Hint: *fopen*, *fprintf* and *fscanf* are the functions you are required to know in order to properly manage the file access. Also remember that files accept three basic way of opening: reading, writing and append. Choose accordingly.

Exercise 3– A program is composed of several processes that share an integer variable (SV) saved in shared memory. SV must be protected using a semaphore so that only one process at a time can access it. The parent process (referred to as P) initializes SV to -1 and then creates a child process (referred to as p1).
After creating this process, it enters into an infinite loop in which it performs the following operations:

1. Generate an integer random number R between 1 and 10
2. Sleep for R seconds
3. Wait until it can access SV
4. Write R in the shared variable
5. Release the shared variable
6. Go back to 1

P1 performs an infinite loop creating processes as follows:

1. Generate an integer random number R between 1 and 3
2. Sleep for R seconds
3. Generate a child process
4. Go back to 1

Every child process generated by P1 performs the following operations:

1. Generate an integer random number R between 1 and 5
2. Sleep for R seconds
3. Wait until it can access SV
4. Check if SV == -1
5. If yes: release SV and go back to 1
6. If No: print SV, set SV to -1 and exit

Note: to share a semaphore between processes, the semaphore variable must be saved in a shared memory region shared among all processes.