Universitat Pompeu Fabra
Barcelona

Lab 03: Delivery Exercise

Understanding signal handling

Write a program where a father creates a child process (son) and plays a game of Nim against him (for reference: https://en.wikipedia.org/wiki/Nim). At the start of the game, a certain number of tokens will be provided as a command-line argument and written in the file name `"game.dat"`. The father goes first, and the players take turns removing 1, 2, or 3 tokens per move. The player who takes the last token wins, while the loser must print the following message to standard output:

> "My PID is X, I am the <Father/Son>, and I have lost."

The number of tokens each player removes is determined randomly using `rand()%3 + 1`.

Players communicate using a shared file named `"game.dat"`, where the remaining number of tokens is stored as a binary integer, at the beginning of the file (the size of the file will always be 4 bytes). To avoid data conflicts, you have to make sure that turns are operated sequentially, and there is no access to the shared file in between the turns. After making a move and updating the file, a player signals the opponent that their turn has ended by sending `SIGUSR1`. If a player takes the last token and wins the game, they instead send `SIGUSR2`.

Hint: To ensure proper signal handling, the receiving process must have its signal handler set before the first signal is sent. A simple way to achieve this is by introducing a **1-second sleep** before sending the first signal. While more complex synchronization strategies exist, this approach is sufficient for avoiding race conditions.

Delivery Instructions:

- Submit the following files using the same specified naming conventions below:
    - Nim.c
    - integrants.txt include all participants of the group, separated by lines, in the format *NIA Name Lastname1 LastName2 , and the practice group number.*
- Ensure each program compiles successfully and adheres to the CLI argument structure provided. Include the instruction you used for compiling.
- Take a screenshot showing your program successfully compiled without errors.
- Submit the programs as a compressed file named `practice1_<student1_student2>.zip`, replacing `<student1>` and `<student2>` with your full names (first and last names). Along with the compressed file, include

screenshots showing the program successfully compiled and implemented without errors.
- Only 1 submission per pair is required.
- Be prepared to explain the implementation and use of each program to your peers in the next class. Ensure that all programs are written in the C programming language and follow the provided naming conventions.
- Allowed function calls: open, close, read, write, dup2, fork, wait, waitpid, pipe, getpid, getppid, printf, scanf, malloc, free, signal, kill. You are not allowed to use **any other functions** that directly or indirectly use a system call (i.e. fwrite).