Universitat Pompeu Fabra Barcelona

Lab 02: Delivery Exercise

Understanding Binary files for Storing Data

Objective: The objective of this project is to familiarize yourself on how to create other processes in linux in order to efficiently perform a complex task using concurrency.

Create a program which will open a directory, and list all its content. When it finds a file ending in *.dat* which it finds, it will create a son, which will open that file, read all its content as integers, write the total sum of these integers in a pipe to the father, and finish. The father will continue the exploration of the directory without waiting for its childs. Then, when it has visited all the content in the directory, it will read all the content written in the pipe, and print it in the standard output. It will then finish when all of its childs have also finished.

- **Hints:**
  - Beware of the path, when the folder is not in the current directory. To open a file, you will have to provide its full path, combining
  - Notice that several processes can write simultaneously in the same pipe.
  - Beware when you read from pipes, that you must close its reading endpoint before writing.

Delivery Instructions:

- Submit the following files using the same specified naming conventions below:
  - directoryExplorer.c
  - integrants.txt include all participants of the group, separated by lines, in the format *NIA Name Lastname1 LastName2*
- 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.
- 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. You are not allowed to use **any other functions** that directly or indirectly use a system call (i.e. fwrite).