I Assignment – Report

Authors: Ilenia D’Angelo, Giovanni Di Marco

Code

We have implemented two versions of the same program: one using three different unnamed pipes and the other using one named pipe and an unnamed pipe.

- **Unnamed Pipes version:** After the definition of pipes and file descriptors, we have generated the child processes (G1, G2, R, M) using the system call *fork ()*. Then we opted to struct our code in several modules for clarity and an efficient debugging, using several functions implemented by us. We have implemented, in execution order: *writeG, readR, writeR, readM*.
  Processes:
  1. **G1 and G2:** These two processes are the senders, each one of 10^6 messages. They call the function *writeG*. It creates messages in form of structs and write them, using the suitable system call, one at a time over a pipe, one for each process.
  2. **R:** is the receiver of those messages. To achieve this aim it calls the function *readR*, which reads, using the system call, in a cycle performed $2 * 10^6$ times. Since R receives messages from both G1 and G2, we have inserted a *select ()* function. If both pipes have data available, the choice is made randomly. Then it does some calculations with the received data and sends them, using *writeR*, to process M. In this function R writes, over a new pipe, another struct, which contains all the data useful for the aim of its receiver.
  3. **M:** This process uses our third pipe, obviously different from the first two, to read what R has sent. With this propose it calls the *ReadM* function, which reads data and prints out a string on the shell, where we can see results of the performance of our processor in sending and reading data.
- **Named pipe version:** it differs from the first one only for the part of the communication of G1 and G2 with R: the (unilateral) communication uses a single named pipe to take place, in this case it is not necessary to have a select function ( ). Messages are read in the order they are written using the default FIFO policy.

Some issues

We have encountered an unexpected behaviour in our program. We have been prevented from closing no more used file descriptors. Unfortunately, we have not been able to fix the error: “Bad file descriptor”, which occurs when some unused file descriptors are closed, so we have decided to let them close at the end of the program.

Results

We have done some experiments (contained in the attached file) to test our CPU’s performance. How we can see in both versions, the highest bandwidth value and the lowest latency value are reached by setting the offset value equal to zero. The trend is monotonic, so we don’t have a pitch, probably because of the new processor of last generation. Furthermore, we have implemented the code in a way that is impossible to do not receive some data, which have been sent. Because the process R acts as it has to perform an infinite cycle, when it waits for a message, so it terminates only when it has read every datum form G1 and G2.
Based on the data collected, the latency in the named pipes version is much lower than unnamed pipes’ one. This confirms the expected results: the named pipes are more efficient than others.