Advanced and Robot Programming Assignment

Roberto Albanese 4234506
Università degli Studi di Genova
Robotics Enginireeng

***Abstract*—POSIX is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems. The initial specification of this project aimed to simulate a network of multi-process systems, all identical, each one running on a machine in the same LAN, connected through sockets, exchanging a data token at a specified and measured speed. However, due to the Covid 19 pandemic, the processes initially used to connect several machines are used on a single computer in a closed loop.**

I. INTRODUCTION

Unix is a powerful family of operating system and it can be used for a large variety of topics. One of its main feature is the capability of perfectly managing communication between independent processes by using components like sockets, interrupts and pipes. This project focuses on these very aspects in order to simulate a network of multi-process systems. To do so, it is required a good knowledge of the main syscall involved. For a better comprehension, it follows a brief overview of the syscall used in this project:

- **fork()**: creates a son process ”identical” to the father. It returns ”0” to the son, ”PID of the son” to the father, ”-1” in case of error;
- **exec()**: substitutes the caller’s image with the executable file *pathname*, and executes it transferring *arguments*;
- **select()**: returns which file descriptor has data to read (hence preventing blocking when a process receives data on multiple pipes);
- **read()/write()**: used to read/write data from/to a pipe;
- **socket()**: establish an IP connection with machines in internet;
- **signal()**: defines a software equivalent of hardware interrupts.

II. PROBLEM DEFINITION

The initial requirement was to establish a networking communication between multiple machine and elaborate data within each machine through inter-processing communication. Due to Covid 19 it was not possible to achieve the initial requirement, so it was necessary to slightly modify it in order to obtain a closed loop communication within the same machine. In Fig. 1 it is shown the software architecture involved in the project: black boxes represent machines, circles represent processes and rectangles represent pipes.

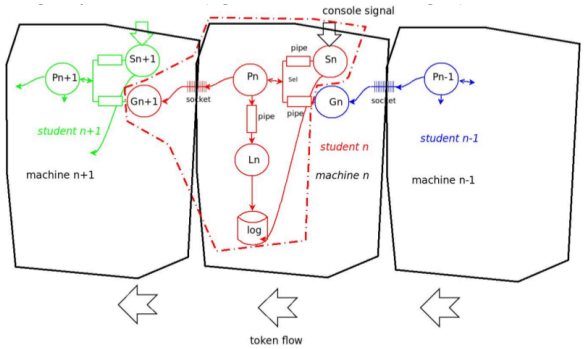

Fig. 1: Software Architecture

Process P receives tokens through a socket connection with G, computes and sends an updated token after a given delay. P is connected to Processes S and G through non-deterministic pipes (Posix *select*). P does the following computation:

$$new\ tok = rec\ token + DT * (1. - (rec\ token)^2/2) * 2pi * RF \quad (1)$$

Since (1) generates an exponential increment to $-inf$, the formula was modified as follows:

$$new\ tok = rec\ tok * \cos(2 * pi * RD * DT) \pm \sqrt{(1 - (rec\ token)^2/2) * \sin(2 * pi * RF * DT)} \quad (2)$$

where:

- *new tok* is the new computed token;
- *rec tok* is teh received token from G process;
- *RF* is reference frequencyof the generated token wave;
- *DT* is the delay time due to inter-process communication;

Token is a floating point number between -1. and 1, plus a time stamp with the absolute system time (by the operating system) of the instant P writes data on socket. *Process S* receives console messages as Posix signals. S can elaborate only three custom signals:

- **SIGCONT**: outputs the contents of the log file
- **SIGUSR1**: to stop processes P, G and L;
- **SIGUSR2**: to resume processes P, G and L.

Process L logs received data from P (timestamp and current token value) end events (received signal from S) in a log file. Data are logged in the following format:

- ”timestamp” ”from G — from S ” ”value”
- ”timestamp” ”sent value” (a sample of the wave)

Process G receives tokens and dispatches them to P.

III. IMPLEMENTATION

P, G, S, L start using the parameters stored in the configuration file. A **configuration file** in the machine contains in text format the necessary parameters, and is edited by the user before running the multi-process. The config file contains the following data:

- IP address of the machine and port numbers of processes (where relevant)
- waiting time (in microseconds) applied by process P before sending the updated token (initially: 1,000)
- reference frequency (RF) of the generated token wave
- other relevant data to be read before starting.

In order to compile, download the repository, open a terminal and move in the workspace where the source code is located. Then launch the following commands:

```
g++ -o <nameofExecMain> Final.c
g++ -o <nameofExecG> G.c
```

Inside *ConfigurationFile.txt* it is possible to edit the starting parameters. To execute the code type:

```
./<nameofExecMain>
```

Now it should be visible in the terminal a bunch of information, as shown in Fig. 2, including the configuration parameters, the successful creation of pipes between the processes, all the successful generation of all the processes with the respective PIDs, and the current token value received from G. In order to send custom signals to S it is necessary to open a new terminal and to launch the command in the following format:

```
>kill -<SignalName> <S PID>
```

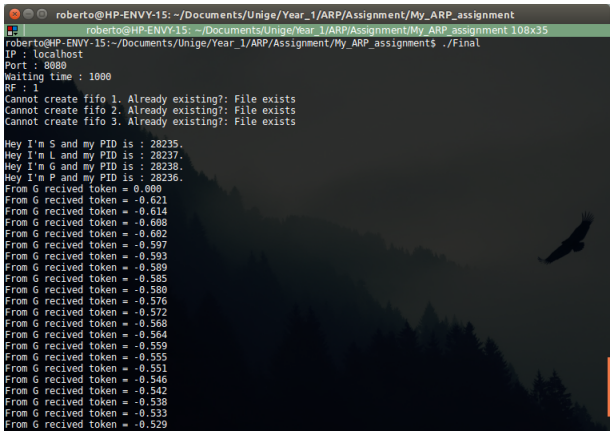

Fig. 2: Output from main execution

Whenever a custom signal is sent, in the main terminal it will be shown the successful reception of it as show in Fig. 3.

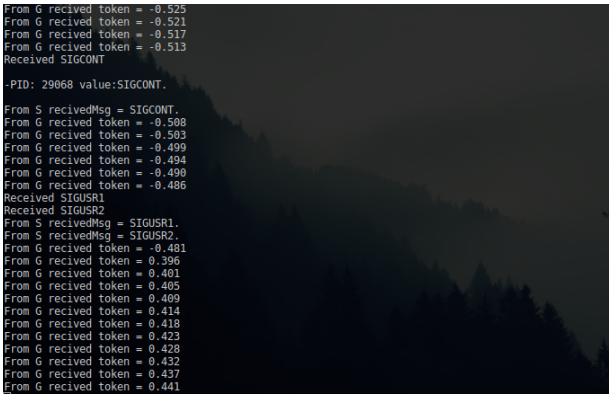

Fig. 3: Signal received from S

Finally, in the log file, as show in Fig. 4 it is possible to see all the logged data during the simulation.

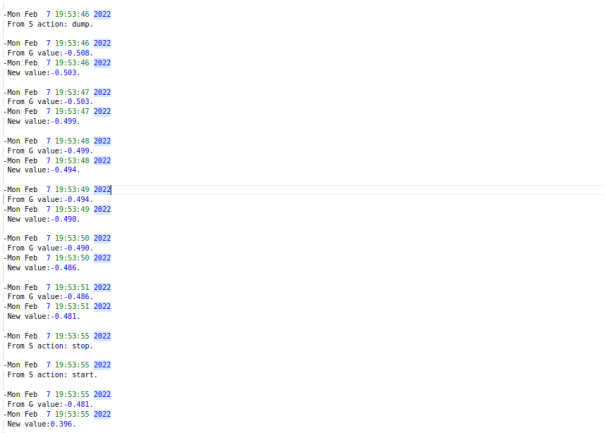

Fig. 4: Log file

IV. RESULTS

In this paper it is analyzed the concurrency of multiprocess communication within a Unix based machine using Posix standard. With the given code it is possible to test the architecture in order to transfer and receive data between process and alter the state of any process by stopping and resuming it during the execution. As an additional proof of work, the plot of the curve generated by (2) is shown with different RF values:

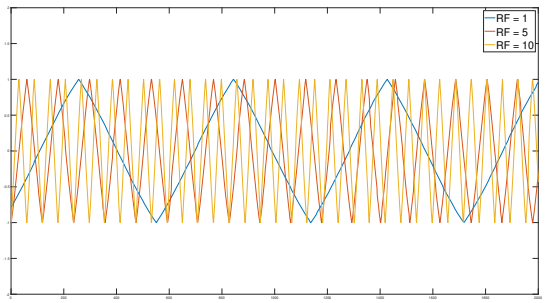

Fig. 5: Token Generated Curve

As expected, from Fig. 5, the token curve is successfully generated and by increasing *RF* value the curve period decreases.