CT213
Computing Systems & Organisation

University of Galway
2022/2023

Lab

Inter Process Communitation (IPC) 1: Named Pipes

Semester 1

This practical will introduce named pipes as a way for processes to communicate. As an introduction you can probably have a look at the following (old) articles (from the Linux Journal):

- `http://www.linuxjournal.com/content/using-named-pipes-fifos-bash`
- `http://www.linuxjournal.com/article/2156`

You'll obviously find plenty of other resources online...

1 A Ring of Communicating Processes

In this exercise we want to create several (similar) processes and make them communicate (communication between them is one-way in this exercise). The general architecture of our system can be seen in Figure 1 with 6 processes and 6 named pipes between them. Note the order (names) of the pipes and the processes: `pipe1` is between processes `p1` and `p2` etc. Actually what we want is for messages from `p1` to `p2` (and only those ones) to use `pipe1` – there will not be any symmetric message between `p2` and `p1` in our exercise. This means that if `p1` can send messages to `p2`, `p2` cannot send anything to `p1` and vice-versa, and so on for every couple of processes.

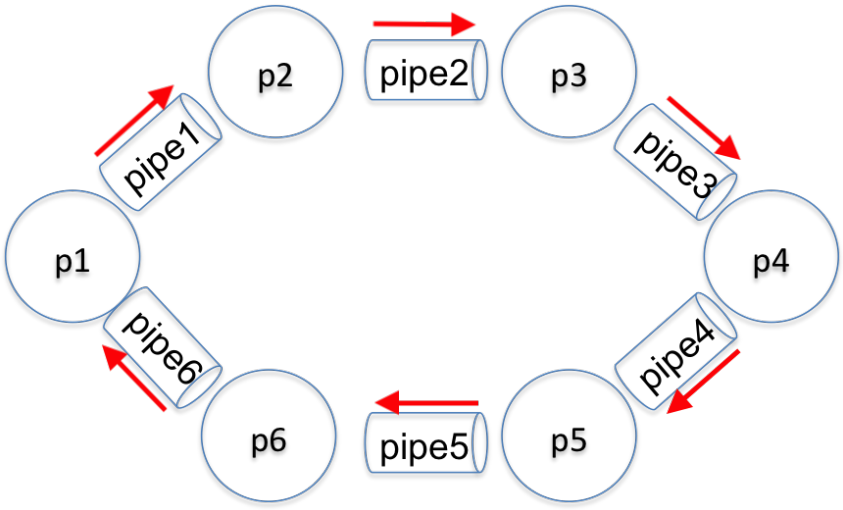

Figure 1: Architecture of our System

1.1 Simple Test

Create a named pipe called `test_pipe` (using the command `mkfifo test_pipe`). As a first example we write/read one message to/from the named pipe.

The commands `echo` and `read` should be used to write and read to/from a named pipe. Use **two** terminals to test your scripts.

- Terminal 1, write using the command: `echo "something" > test_pipe`
- Terminal 2, read using the command: `read input < test_pipe; echo $input`

1.2 Test with Loops

Now create two scripts `write_sh` and `read_sh` so that:

- `write_sh`: the writer continuously writes in the pipe something that is given by the user on Terminal 1
- `read_sh`: the reader continuously reads from the pipe and displays it on Terminal 2.

The loop for the writer should be:

```
#!/bin/bash

while true; do
        read input
        echo $input > test_pipe
done
```

Listing 1: write.sh

The loop for the reader should be:

```
#!/bin/bash

while true; do
        read input < test_pipe
        echo received from the pipe: $input
done
```

Listing 2: read.sh

Again, use two terminals to test your scripts. You can stop both scripts with `control+c`.

1.3 In and Out

Now write one additional script (name it `inout.sh`) that does one thing: read from one named pipe and write in another named pipe. this script should have two parameters (the two named pipes). Use an endless loop for this and reuse the two scripts `write_test.sh` and `read_test.sh` that you have implemented before (remember: read THEN write).

Note that you have **2 DIFFERENT** pipes here, one for the input and one for the output (in the scripts: `write_test.sh` and `read_test.sh` the pipe was the same).

Create two named pipes `pipe_test1` and `pipe_test2`, and test your script: write in the "in" named pipe and check what is in the "out" pipe (from another terminal using `read input < pipe; echo $input}`.

The following is a STEP in the right direction (things are missing):

```
#!/bin/bash

in_pipe=$1
out_pipe=$2

while true; do
        read input < XXX
        echo I found this in the pipe: $input and I am going to send it on my out pipe
        echo $input > YYY
done
```

Listing 3: inout.sh

1.4 First Attempt

Now create 6 named pipes and start 6 of your `inout.sh` processes with the correct parameters (previous and next pipes)–each process in its own terminal. Then, open another terminal and start the writing process: write something in one pipe and observe the processes sending the input to each other. Stop the processes with control+c. You probably realise there is something wrong here.

1.5 Stop!

Now we want to stop the input to be sent again and again and again. When a script reads the same word twice in a row, it does not forward it on to its neighbour. This would stop the problem you've seen before. Test that the processes are not in an endless loop anymore, sending the same message for ever.

The following is a STEP in the right direction (things are missing):

```
#!/bin/bash

in_pipe=$1
out_pipe=$2

prev_input="init"

while true; do
        read input < XXX
        echo I found this in the pipe: $input and I am going to send it on my out pipe
        if [ YYY ]; then
                echo $input > ZZZ
                prev_input=$input
        fi
done
```

Listing 4: inout.sh

1.5.1 A Better Solution *(Optional)*

What if we don't want to initialise `prev_input`? What other condition/if/test do you need to add?

1.6 A Nice Solution *(Optional)*

Obviously the problem at the moment is that you need to know the name of the two named pipes (in and out) for each process. Ideally, the scripts should create a named pipe at the start of the script (say the in named pipe), then ask for the other one (say, the out one). To do so, they should communicate with another script, let's call it a bootstrap process, which knows who's who and can tell every process which pipe they should write to. The idea is simple: at the start of every "normal" process, the process creates a named pipe and sends a message to the bootstrap process which stores the name of the named pipe and writes in it the other named pipe.

```
#!/bin/bash

read prev_pipe < pipeb
first_pipe=$prev_pipe

for var in 1 2 3 4 5; do
        read next_pipe < pipeb
        echo in pipe: $next_pipe > $prev_pipe
        echo sent $next_pipe in $prev_pipe
        prev_pipe=$next_pipe
done
echo in pipe: $next_pipe > $first_pipe
echo finally send $next_pipe in $first_pipe
```

Listing 5: bootstrap.sh

Modify your previous "repeater" script to (i) create a named pipe (`mkfifo`) and then send the name of the named pipe on the boostrap process' named pipe (ii) read on the pipe the name of the out pipe. Launch the bootstrap process. Launch the repeater processes and check they get the correct named pipes.

Test that the ring is connected.

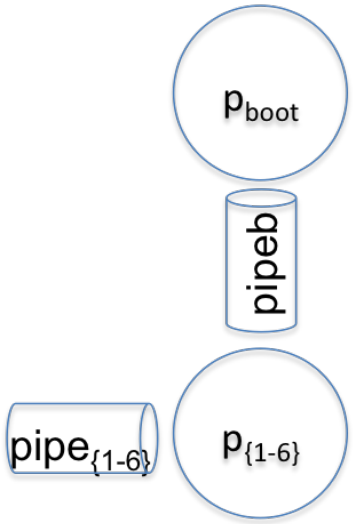

Figure 2: Architecture of the Bootstrap Mechanism