# PIPE & FIFO

Samuele Germiniani
samuele.germiniani@univr.it


UNIVERSITÀ
di VERONA
Department
of ENGINEERING FOR INNOVATION
MEDICINE

# Exercise templates

- Download the templates of the exercises

```
git clone https://github.com/SamueleGerminiani/ex_pipe_fifo_templates
```

# PIPE - Exercise 1

Write a program that implements the Producer-Consumer paradigm based on PIPEs.

- The Consumer process receives the pathname of a text file from the command line.
- Subsequently, the Consumer process creates a PIPE and a Producer process.
- The Producer process reads the content of the file in chunks of up to 100 characters.
- Each chunk read is sent to the Consumer process through the PIPE.
- The Consumer process prints the content of the received file through the PIPE.

# PIPE - Exercise 2

Extend Exercise 1 so that the Consumer process receives from the command line a list of text file pathnames.

- For each text file, the Consumer process creates a Producer process, which sends the content of the text file to the Consumer through a single PIPE.
- The PIPE is a byte stream channel. If two or more Producers write to the same PIPE

**How can we distinguish the data written by one Producer from the data written by another Producer?**

**Possible Solution**

The Producer prefixes the number of bytes it intends to write before the actual data.

---

PIPE read end ← | 2 | byte-1 byte-2| 4| byte-1 byte-2 byte-3 byte-4| ----

---

The content of the PIPE above shows two data chunks. The first chunk has a size of 2 bytes, while the second one has a size of 4 bytes

Use the following struct:

```
struct Item {
   ssize_t size;
   char value[MSG_BYTE];
};
```

The size field contains the number of bytes that the Producer process intends to write to the PIPE. The value field contains the actual data sent via the PIPE.

```
struct Item item;
// ... the Producer initializes item as described above
// Chunk size and data are atomically copied to the PIPE using write()
write(PIPE, &item, item.size + sizeof(item.size));

// The Consumer process reads from the PIPE first the size field and then the data.
ssize_t size;
read(PIPE, &size, sizeof(size)); // Read chunk size
read(PIPE, &buffer, size);       // Read chunk
```

# FIFO - Exercise 3

Write a Client-Server application based on FIFO.

**Server**

- The server creates a FIFO and waits for a message; the message is a vector of 2 integers [a, b].
- If a < b, the server prints: "a is less than b"; if a >= b, the server prints: "a is greater than or equal to b".
- After printing the string on the screen, the server removes the FIFO, and finally terminates.

**Client**

- The client asks the user for two integers, sends the two numbers to the server via the FIFO, and finally terminates.

## FIFO - Exercise 4

- Extend Exercise 3 so that the server continues to read messages from the created FIFO.
- The server removes the FIFO and finally terminates only when the two received numbers are equal or more than 30 seconds have passed since the last received message.

# FIFO - Exercise 5

Create a Client-Server application based on FIFO.
The messages sent from a Client to the Server are instances of the Request structure.
The messages sent from the Server to a Client are instances of the Response structure.
The structs are defined in **request_response.h**

**Server**
The Server process performs the following operations:
1.Creates a FIFO named "fifo_server" in the /tmp directory.
2.Reads a message req of type Request from fifo_server.
3.Creates an instance resp of the Response structure by initializing the value of resp.result equal to the square of req.code.
4.Opens the FIFO /tmp/fifo_client.cPid, where cPid is the value of the req.cPid variable (if req.cPid is 123, the Server process opens the FIFO /tmp/fifo_client.123).
5.Writes resp to /tmp/fifo_client.cPid.
6.Repeats from step 2

**Client**
The Client process performs the following operations:
1.Creates a FIFO named "fifo_client.pid" in /tmp, where pid is the process's PID (if the Client has PID 123, the process creates the FIFO /tmp/fifo_client.123).
2.Creates an instance req of the Request structure (see above), where cPid contains the process's PID, and code is a randomly generated number (rand()).
3.Writes req to /tmp/fifo_client.cPid.
4.Reads a message resp of type Response (see above) from /tmp/fifo_client.cPid.
5.Prints the resp.result field to the screen.
6.Deletes the FIFO /tmp/fifo_client.cPid, and finally terminates.
If no message is received within 30 seconds or the SIGINT signal is sent to the Server process, which deletes fifo_server, it finally terminates.