# Sistemas de Operação

**(Ano letivo de 2025-2026)**

## Guiões das aulas práticas

script #03 IPC — File redirection, pipes and fifos


### Summary

Understanding and dealing with redirecting the standard output of a process.

Using pipes and named pipes.


---

### Previous note

In the code provided, some system calls are not used directly. Instead, equivalent functions provided by the `process.{h,cpp}` library are used. The functions in this library deal internally with error situations, either aborting execution or throwing exceptions, thus releasing the programmer of doing so. This library will be available during the practical exams.

---

### Exercises

**Exercise 1** Redirecting the standard output (stdout) of a process.

(a) ***The `dup2` system call.***

- See the manual for a description of the `dup2` system call (`man dup2`).
- Read the `redirect.cpp` source code, create the `redirect` executable (`make redirect`), and run it.
- **Questions:** What happened to the program's second `printf` statement? Was it executed or not? If it was, where did the printed message go?
- Edit the `my.file` file that was created in the meantime in the working directory. Change its content by inserting your own text of at least 40 characters. Run the `redirect` program again.
- **Question:** Explain the change that occurred in the file after the previous execution. Was it what you were expecting?

(b) ***Training exercise***

An exercise in the previous lesson involves programs `fork4.cpp` and `child.cpp`. Both of them send messaged to the standard output. Without changing the `printf` statements of any of them, change program (`fork4.cpp`), and only it, such that the `printf`s of program `child.cpp` go to a file whose name is given to `fork4.cpp` as a command line argument.

---

**Exercise 2** Creating and using a pipe as a communication channel between two processes.

A pipe is a unidirectional data transfer channel that can be used to facilitate communication between directly related processes (parent-child or child-child, for example). A pipe is created calling system call `pipe` (see `man 2 pipe`).

(a) ***Creating a pipe.***

- Read the `pipe1.cpp` source code, create the `pipe1` executable (`make redirect`), and run it.
- **Question:** What led to the modification of the contents of the `buf` array?
- Change line 32 of the `pipe1.cpp` file, eliminating the `+1` in the expression assigned to the `sz` variable. Rebuild the executable file and run it in the new version. Try to interpret the changes.

(b) ***Communication between parent and child processes using a pipe.***

- See the manual for a description of the `sort` command (`man sort`).
- Read the `pipe2.cpp` source code, create the `pipe2` executable (`make redirect`), and run it.
- Execute the following sequence of commands:

```
ls -l > /tmp/zzz
sort -n -k 5 /tmp/zzz
```

- Execute the following command

```
ls -l | sort -n -k 5
```

- **Question:** Are there differences between the outputs of the 3 previous cases? Why or why not?

(c) ***Training exercise***

Change program `pipe2.cpp` such that:

- The second `exec` (the one related to the `sort`) also happens in a child process.
- The parent process waits for the termination of both children before quit.

---

**Exercise 3** Communication between two processes using a named pipe (FIFO)

A FIFO, or named pipe, is a particular type of pipe that is accessible through the file system (in fact, it constitutes a special type of file). This means that, unlike a conventional pipe, its existence is independent of the processes that use it, enabling communication between processes that are not directly related. See the manual for a description of the FIFO special file (`man fifo`). A FIFO can be created using command `mkfifo` (see `man mkfifo`).

(a) ***Creating a FIFO.***

- Create a FIFO in your working directory called `my.fifo`

  ```
  mkfifo my.fifo
  ```

  List the directory

  ```
  ls -l
  ```

  and see what happens.

(b) ***Communicating using a FIFO.***

- Analyze the code of the programs described by the files `sender.cpp` and `receiver.cpp` that implement a mechanism for transferring messages from one terminal to another.
- Create the sender (`make server`) and receiver (`make receiver`) executables and run them in separate terminals. Launch receiver first, then sender.
- **Question:** Why does the "Received through fifo" message only appear on the receiver terminal after launching the sender?
- **Question:** Try to explain how the message transfer mechanism works between the sender process and the receiver process.
- **Question:** Press the CTRL-D key combination in the terminal where you launched the sender process. Not only did the sender process terminate, but the receiver process as well. Why?

(c) ***Training exercise***

Modify the `sender.cpp` and `receiver.cpp` files to allow bidirectional communication. Suggestion: use two FIFOs.

---