# Operating Systems
# Program Execution

Group of Distributed Systems
University of Minho

## 1 Objectives

Become familiar with and use system calls related to program execution.

## 2 Chamadas ao sistema

```
#include <unistd.h>     /* system calls: essential defs and decls */

int execl(const char *path, const char *arg0, ..., NULL);
int execlp(const char *file, const char *arg0, ..., NULL);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
```

## 3 Exercises

1. Implement a program that runs the command `ls -l`. Note that if the execution is successful, no further instructions are executed from the original program.
2. Implement a program similar to the one above that executes the same command but now in the context of a child process.
3. Implement a simplified version of the `system()` function. Unlike the original function, do not try to support any kind of redirection, or composition/chaining of executable programs. The only argument should be a *string* that specifies an executable program and a possible list of arguments. Make sure that the behavior and return value of your function are compatible with the original one.
4. Implement a `controlador` program that concurrently executes a set of programs specified as arguments to your command line. The `controlador` should re-execute each program as long as it does not terminates with null exit code. At the end of its execution, the `controlador` should print out the number of times each program was executed. Consider that the programs are specified without any argument.

   ```
   $ ./controlador ./a.out ./b.out ./c.out
   a.out 2
   b.out 4
   c.out 4
   ```

Tip: You can reuse the `mysystem()` function implemented in the previous exercise. To test the `controlador`, use the auxiliary program *auxilar.c* provided along with this guide. The program executes and returns a random value between 0 and 3.

## 4 Additional Exercises

1. Implement a very simple command interpreter inspired by by the `bash`. The interpreter should execute commands specified in a line of text entered by the user. The commands are composed of the name of the program to be executed and any list of arguments. The commands can also be executed in the foreground, or in the background if the user ends the line with `&`. The interpreter should end its execution when the user invokes the internal command `exit` or when you mark the end of the file (`Control-D` at the beginning of a line on Unix-based systems).