Operating Systems

Bachelor's Degree on Computer Science & Engineering

Course 2020-21

Lab 2: Minishell

Date: **16/04/21**

GROUP: **89** TEAM: **09**

Members: **Ignacio Arnaiz Tierraseca & Luis Daniel Casais Mezquida & Juan Del Pino Vega**

100428997@alumnos.uc3m.es // 100429021@alumnos.uc3m.es // 100429063@alumnos.uc3m.es

INDEX

1 Execution of commands

As the parser gives us the commands in the argvv parameter, we just execute a fork and in the child process we execute the command in position 0 of the argvv parameter by execvp(argvv[0][0], argvv[0]).

For two commands, the child executes the first command and the parent executes the second one. And for three commands, two children are created, one for the first command and another for the second one, while the parent executes the last one.

Here we provide several tests for the simple command execution. We executed in the Minishell the commands: ls, ls -l, who and pwd.

2 Background

Commands to be executed in background are indicated with “&”, and do not show the result of their execution on the terminal, instead the shell keeps working on them while executing the rest of commands. This is done by not waiting for the children to end on the parent process.

3 Redirection

Redirection was implemented by setting the specified file as the standard I/O, by first closing the standard file number (0 in case of input, 1 in case of output, and 2 in case of error), and then opening the file, thus setting the file descriptor of the file as the same of the stdfileno, because when opening a file, the OS searches for the lowest available file descriptor. To check if there is redirection or not, we check filev (such that it's different from 0), a vector that contains the file names in case of redirection (for input, it's filev[0], for output it's filev[1], and for error redirection it's filev[2]). The error is checked before the fork(), because it's common to all commands, but for the other ones, on the first command the input is checked and on the last command the output is checked.

The following commands were tested:

- `ls > out.txt` (file existed)
- `ls > out.txt` (file did not exist)
- `in.txt > wc` (file existed)
- `in.txt > wc !> error.txt` (neither file existed)

4 Pipes

For three commands, two pipes are created, for 2 commands, one pipe, and for 1 commands no pipes are needed. As the execution of commands will change the image of the process, the standard I/O is replaced by the pipes for command intercommunication. For the first command, there is only need to change the standard output for the write next pipe, for the last command, the standard input was changed for the read of the previous pipe, and for the middle command, both were changed.

The following commands were tested:

- `ls | wc`
- `ls | wc | wc`
- `caca | wc` (command did not exist)

5 Execution of internal commands: Mycalc

5.1 Code description

Mycalc internal command has been developed in a separate function, in the child process of the fork this function is called when the command input is mycalc (argvv[0][0] is mycalc). Inside the mycalc function the structure of the command is checked; if the operand1 (argvv[0][1]), the operator (argvv[0][2]) or the operand2 (argvv[0][3]) is null then the error given in the statement is shown in the standard output (file descriptor 1). If everything is not null now we check that the operator is add or mod otherwise the error is shown in the standard output.

After checking the command we get the operands and if the add is the operator the output shows the addition and the accumulated sum preceded by the [OK] label in the standard error output (file descriptor 2). However, if the operator is mod the remainder and the quotient are calculated and shown in the specific way given in the statement by the standard error output (file descriptor 2).

5.2 Test cases

First we start we wrong inputs for mycalc:

- No operands and no operators.
- One operand.
- Two operands.
- One operand and an operator.
- The operator and an operand.
- One operator.
- One operand, an invalid operator and another operand.

For the valid test we just test one case of each operator, first for add and then for mod.

```
MSH>>mycalc
[ERROR] The structure of the command is <operand 1> <add/mod> <operand 2>
MSH>>mycalc 2
[ERROR] The structure of the command is <operand 1> <add/mod> <operand 2>
MSH>>mycalc 2 8
[ERROR] The structure of the command is <operand 1> <add/mod> <operand 2>
MSH>>mycalc 2 add
[ERROR] The structure of the command is <operand 1> <add/mod> <operand 2>
MSH>>mycalc add 2
[ERROR] The structure of the command is <operand 1> <add/mod> <operand 2>
MSH>>mycalc add
[ERROR] The structure of the command is <operand 1> <add/mod> <operand 2>
MSH>>mycalc 3 sub 8
[ERROR] The structure of the command is <operand 1> <add/mod> <operand 2>
MSH>>mycalc 3 add -8
[OK] 3 + -8 = -5; Acc -5
MSH>>mycalc 10 mod 7
[OK] 10 % 7 = 7 * 1 + 3
MSH>>
```

6 Execution of internal commands: Mycp

6.1 Code description

The mycp internal command is created as a separate function that is called by the child process of the fork when the command received is “mycp” and receives as arguments argvv

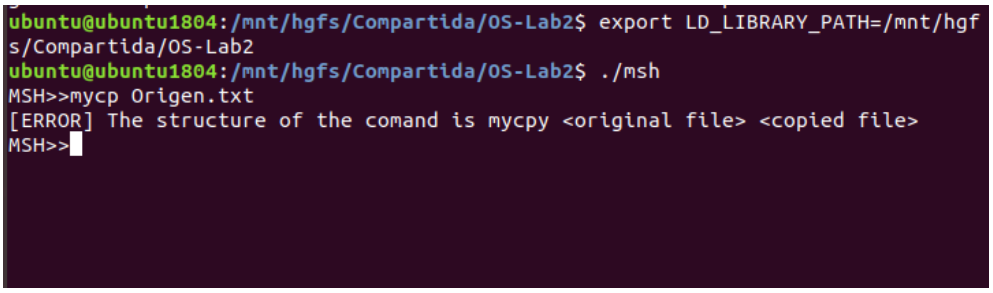
When called, the function stores the two strings following the command that is argvv[x][0], that are the original file, and the destination file. Then it checks if any of those parameters is null, in that case it will stop and show an error.

When it is checked that two parameters have been passed, the function checks if the origin file exists by using the struct stat for it, in the case this returns a -1 that means that the file has not been found, so it stops and shows an error message.

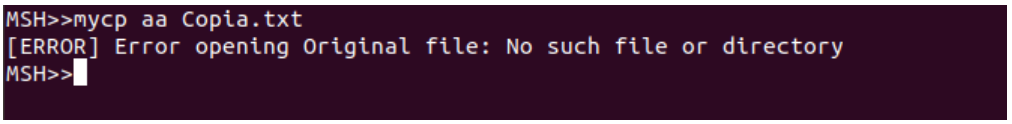
After checking that the parameters are correct, it calls “execvp” and pass to it as parameters “cp” and argv[x], so the unix command cp is call with our arguments

6.2 Test cases

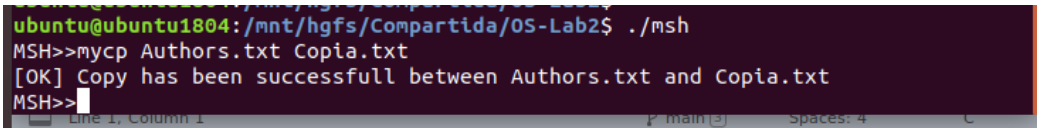
First test case, an incorrect number of parameters is passed to the command, so the shell shows an error indicating the correct format



Second test case, an inexistent file is passed to the command as parameter. The shell checks with the struct stat if that file exists, as it doesn't an error indicating that the original file doesn't exists



Third test case, two existent files are passed as parameters to the command, the program checks that both files are not null and that both files can be opened. After checking that both parameter are correct it copies the original file (Authors.txt) into the copy one (Copia.txt)



7 Conclusions

This lab has helped us to improve our skills in C programming with the use of fork, exec and pipes, as well as how to manage processes in order to execute different commands sequentially and manage redirections.

Although we have encountered some problems while developing the project. The first ones were mainly regarding closing child processes, as in some points child processes became zombies and prevented the shell to continue working. Then we had problems with pipes and doing n commands, so we decided to focus on only 1, 2 and 3 commands to better control the pipes. Finally, we weren't able to implement the accumulative function for mycalc, even using environment variables with setenv and getenv system calls.

In the end, we think this lab assignment has been teachingful and has reinforced our knowledge on how os systems work internally.