P2: Implementing a shell

Sergio López and Xavier Marquès

February 2021

3 Fork and Exec example

1. The program creates a new process with fork and in the child one we do the execvp to change the process and execute a command we pass in the shell.
   (a) runs the program so creates 2 process
   (b) runs the program and list all archive so creates 2 process
   (c) runs the program and list all archive with their permission so creates 2 process
   (d) runs the program and shows all the process . Creates 2 process
   (e) runs the program and shows all process from all users without any restriction. Creates 2 process
   (f) runs the program and list the cmds.txt . Creates 2 process
   (g) In this case, the program finds this command invalid and crashes and only creates 1 process
   (h) executes the program in the background. 2 process
   (i) runs the program with success, so 2 process are created
   (j) runs the program twice so 2 forks are done and 3 process are created
   (k) runs the program twice and list the archives so 3 process
   (l) runs the program twice and list all archive with their permission so creates 3 process
2. The execution of execvp may failed because the command was not found or there were an error in the execution of that. So that, the child needs to inform to the father that he has exit with an error code, and we can control them after that.
3. Always, the child it is needed to exit in some case in order that there was not and orphaned child. So in case that there is no command to execute, we have to finish it and inform the father of the situation.

4 Programming the shell

4.1 Declaring variables and defining the command struct

The first step is to define and declare all variables. For the commands, we defined an struct in which will have an array of chars for the command , a group of arrays of chars for the arguments and two integers: one for the number of arguments and the other a flag to determine if the command is executed concurrently.
Then, for reading the file and storing all the commands, we will need several variables: a command counter, a parameter counter, a flag to see if the string we're reading is a command, a buffer and a char for the split function, a integer to open the file and check if it exist and an integer that we will use later on to store an input from the terminal. Finally, we'll need an array of commands.

4.2 Reading a file and storing the commands

The next step is to read and store all commands from a text file. The idea is to use the read_split function provided and read string per string until we reach and EOF, where the function will return 0 since 0 bytes are read. Then, we need to check the length of the buffer because maybe we're not reading a word but a white space and if we have a word, we have to see the command flag. Since this is defined as a true at the beginning, then the first word of each line will be a command so we store it in the command variable and set the flag to 0 because the next words in the line, if there are, will be arguments.
If the buffer is a white space, we can have '&' and that means we're executing this command concurrently, so we set the flag is_concurrent as a true and store it as a parameter; or we can have a new line and, therefore, we have to reset all variables and set the flag command as true.

4.3 Starting the shell

Once we have all commands stored, we can start the shell prompt with an infinite while and we can have 3 cases:

1. If we type "list", then the program has to print all commands. So we just do a double iteration, one for each command and the other for the arguments of all commands, if they have. Since we predefined 10 number of commands, to not iterate always to 10, we added a condition that iterates until one of the pointers points to an empty space.

2. If we type "exit", we just break the while.

3. If we type a number, the program must execute the command in this line. To store this input, we have to convert it with sscanf to an integer and then we check if it is a valid input, that is, one between 1 and the number of commands. In order to apply the execvp, we need to extract

the command line we want from the struct and store it using memory allocation and then, in the final add a NULL. We find that execvp does not take & as an argument so we added a condition to just replace it by a null and apply concurrency later on.
Finally we create the process and apply the execvp in the child one. If the command is not concurrent, then we need to wait them to finish.