Operating Systems

First Assignment Report

Members: Carlos Iborra Llopis / Pablo Brasero Martínez / Marcos Caballero Cortés
NIAs: 100451170 / 100451247 / 100451047
Group: G89

Table of Contents

Code Description

Our code is composed of the following three functions: *mycat.c*, *myls.c,* and *mysize.c*.

mycat.c

The *mycat.c* file implements the functionalities of opening, reading, and writing a specific file in such a way that its content is printed on the screen without using *printf*.

First of all, we will check if we have passed something as an argument by console. Invoking *mycat.c* alone will produce an error on the screen and the termination of the process.
Secondly, we will proceed to read the file using the file passed as a parameter to open it through the open function. To guarantee the correct opening of the file, we carry out a check, in case of error, the execution is terminated.

Once we have the file reference, we proceed to implement its reading supported by a buffer to limit the reading to 1024 bytes per cycle. As we cannot guarantee that the file only occupies 1024 bytes, a loop will start where the data obtained in the first 1024 bytes will be written on the screen. In the case of not having reached the end of the file, we will continue with the following 1024 and so on until the reading-writing process is finished.

At the end of the function, if the file was read and written until the end of the file, we close the file and return 0.
Otherwise, if the program found an error throughout the execution, it will return - 1.

myls.c

In the *myls.c* file, the objective to be reached was to open a directory that is passed as a parameter, in case no directory is given, the program will use the directory in which the program is running. Then the program will print on the screen all the entries that the directory contains, remarking that only one entry will be printed per line. All entries of the directory are listed following the order in which *readdir* returns them.

As functions used for the program, *opendir*, *readdir*, *printf*, and *closedir* were the ones that we used the most in conjunction with *strcpy* and *getcwd*. Each one was used when it was needed for the program, first starting with *getcwd* and *strcpy* to get a directory and store it in a variable, the *opendir* was used to open the directory given, *readdir* was used to read the directory wanted to be read and lastly, *closedir* is used at the end of the program to close the directory selected in the program.

When it comes to printing the information, every entry is printed in a single line and a single dot (.) is used to show the current directory and a double dot (..) to show the parent directory.

If an error occurs when the program tries to open the directory (directory does not exist), -1 will be returned. Else, the program will return 0.

mysize.c

Lastly, in *mysize.c*, the objective is to obtain a certain directory and print all the names of the files inside the mentioned folder as well as their sizes, separated by 4 blank spaces. Following the format <*file*><*4 blank spaces*><*size*>.

In order to perform this, the program first declares all the libraries. Afterward, inside main, the program gets the path of the current directory, opens the directory, returns a DIR pointer to the beginning of the directory, and creates a dirent structure. Then it checks whether there has been an error opening the folder, if not, it creates a pointer to the previously created dirent structure representing the directory entry.

Next, we check with a while whether there are files left to be read or not, then we select a file, open it, check if there has occurred an error when opening it and we create two pointers, one at the start of the file and one at the end so that the size of the file will be the result of subtracting the end pointer minus the start pointer. So, after having both the name of the file and the size, we print them, remembering that there should be 4 blank spaces between the obtained data.

Finally, we close the opened file, check if there has occurred an error when closing the file and if not, we select the next directory entry (file) and repeat the process until there are no more files to be read in the folder.

Whenever an error occurs, the program returns -1, else if no error has occurred, the program reaches the end and returns 0.

Test Cases

mycat.c

For performing the test case for *mycat.c*, we compared our program output with the command *cat* output, and we noticed how they both printed the same, so at least our program does the exact same things like the command *cat*.

```
aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022$ ./mycat p1_tests/f1.txt
Name1   M       32      09834320        24500
Name2   F       35      32478973        27456
Name3   M       53      98435834        45000aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022$
aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022$ cat p1_tests/f1.txt
Name1   M       32      09834320        24500
Name2   F       35      32478973        27456
Name3   M       53      98435834        45000aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022$
```

Screenshot 1. Running *mycat.c* and *cat*, in p1_tests/f1.txt

myls.c

Similarly happens with *myls.c* program, as we decided to compare it to the command *ls -f -l* over the same directory. But here there is a big difference, as the *ls* command prints more information than ours as can be seen in the below screenshot. But the important part is that the things our program needed to perform happen to be the same as the *ls* command on the right part of the printed data.

```
aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022$ ./myls p1_tests/
.
f1.txt
..
dirA
dirC
f2.txt
aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022$ ls -f -l p1_tests/
total 24
drwxr-xr-x 4 aulavirtual aulavirtual 4096 feb  7 12:23 .
-rw-r--r-- 1 aulavirtual aulavirtual   79 feb  7 12:25 f1.txt
drwxr-xr-x 4 aulavirtual aulavirtual 4096 mar 11 17:22 ..
drwxr-xr-x 2 aulavirtual aulavirtual 4096 feb  7 03:23 dirA
drwxr-xr-x 2 aulavirtual aulavirtual 4096 feb  7 03:23 dirC
-rw-r--r-- 1 aulavirtual aulavirtual   79 feb  7 12:25 f2.txt
```

Screenshot 2. Running *myls.c* and *ls -f -l*, in p1_tests

mysize.c

Last but not least, as for the test case for *mysize.c* we decided to do something different as there is no command which does the same as *mysize.c* (which is what happened with *myls.c* and *mycat.c*). So instead, we did 3 test cases apart from the *p1_tests* files, the first one, checks what happens when running *mysize.c* in a folder with a file with a wrong extension (in screenshot example, *wrong.ttxtx*), the second one, checks what happens when running the program in a folder with an empty file (*empty.txt*), and the last one, checks what happens when we run *mysize.c* in an empty folder. The results can be seen in the below screenshots.

```
aulavirtual@Linuxv3042:~$ cd Escritorio/p1_system_calls_2022
aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022$ cd p1_tests
aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022/p1_tests$ ../mysize
f1.txt    79
f2.txt    79
```

Screenshot 3. Running *mysize.c* in *p1_tests*

```
aulavirtual@Linuxv3042:~$ cd Escritorio/p1_system_calls_2022
aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022$ cd wrongFile
aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022/wrongFile$ ../mysize
wrong.ttxtx    0
aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022/wrongFile$ cd
aulavirtual@Linuxv3042:~$ cd Escritorio/p1_system_calls_2022
aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022$ cd emptyFile
aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022/emptyFile$ ../mysize
empty.txt    0
aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022/emptyFile$ cd
aulavirtual@Linuxv3042:~$ cd Escritorio/p1_system_calls_2022
aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022$ cd emptyFolder
aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022/emptyFolder$ ../mysize
aulavirtual@Linuxv3042:~/Escritorio/p1_system_calls_2022/emptyFolder$
```

Screenshot 4. Running *mysize.c* in the three mentioned folders.

Conclusions

Encountered Problems

During this practice, we did not have any big troubles implementing the requirements to the functions. Most of the trouble came from us being used to programming in python, so the main problem was getting used to the use of the C language and its syntax.

Implementing our code in the Linux launcher was also a challenge because we had to adapt our knowledge to a new operating system with new functions and utilities. Another problem we found was working with the terminal, as we were neither used to executing commands through the terminal and this made us work harder and figure out which are the main commands and how they were used.

Lastly, we also had no trouble but doubts about why the programs we were given contain libraries that we are not using in any of the programs (<*sys/types.h*> and <*sys/stat.h*>), and other doubts like why we were told to do some things that later on when executing *Makefile* were needed to be erased as they raised problems (for example, functions that were not used).

Our Conclusion

To sum up, at the beginning that when we were taught the practice we realized we had to practice a little bit more in order to get used to the coding used in c, but within a week of learning, it became simpler and easier.

The biggest problem was understanding correctly how pointers and structures worked but other than that the practice didn't come up with too many complications, overall it was good introductory practice for C language, POSIX, and Linux systems.