Assignment 1

Iván Miguélez García 100383387

José Manuel Frías Salvachúa 100383533

Table of contents

* Description of the code
  + Clarifications
  + Mycat
    - Tests Mycat
  + Myls
    - Test Myls
  + Mysize
    - Test Mysize
* Conclusions

**Description of the code**

**Clarifications**

We have decided to use a buffer of 1024 bytes not only in mycat, as the pdf says, but in the other programs too.

Almost every function is inside an statement to track errors.

When it is said that a test works, it means that the output of the program is what is expected to return.

**Mycat**

The command uses functions open, read, write and close.

First of all we make the comprobation if there is a file indicated after the ./mycat command. If that space is empty we will receive NULL while trying to invoke it via arg[v], and we will directly return a -1 because we know our command won’t work without an specific file.

We then try to open the selected file. The “open” function will return an integer showing us the file identificator that will allow us to to read the file in posterior functions. For this operation we apply the read only policy because we don’t want to modify what's on the file, only see it’s information. If we couldn’t open the file for whatever reason, we will return an error message.

Once we have a file identificator we can proceed to the reading. We will use a while loop for this operation. This while loop consist on successive “read” functions. In each read function we will read N bytes of our file, store that bytes in a buffer and throw as output the number of bytes that were readen. If this number is below one we know we have reached the end of our file. Each time after reading, we will invoke a “write” operation, with the flag STDOUT\_FILENO, the buffer and the size as arguments. This will write in the console the content of the buffer. After this we would execute the loop again. If we find any problem while trying to write , we will print an error.

At last we close the file using it fd again.

**Tests**

1. Opening a file in the same directory. **Works**.

*./mycat f1.txt*

Expected: Printed the info inside the file

Received: Printed the info inside the file

1. Opening a file that doesn’t exists in the same directory. Works.

*./mycat f1.txt*

1. Opening a file in a different directory, given the path as argument. **Works**.

*./mycat p1\_test/f1.txt*

Expected: Printed the info inside the file

Received: Printed the info inside the file

1. Opening a file that doesn’t exists in a different directory given a path. **Works**.

*./mycat p1\_test/f1.txt*

Expected: Error opening

Received: Error opening file: No such file or directory

5.Using a directory as argument after the command

*./mycat p1\_tests*

Expected: Print nothing

Received: Print nothing

**Myls**

This commands depends on 4 functions: getcwd, opendir, readdir and closedir.

We will use the first one when the user input doesn’t include a directory after the command. In this case we will need a buffer that stores a string with the path needed to reach our current directory. We use as size for the buffer 1024 to follow the convention stated in the clarifications part of this report. After this, we copy the buffer in the argv[1]. This is done to avoid confusion in the code, because next functions will be calling this argument. Also, allows us to prevent a creation of a dummy variable. If in the user input there is a directory written after the command this step will be ignored.

Next, we try to open the directory. Opendir will find the pointer of the directory written in the arguments, and then store it in the dirp variable . We execute a control mechanism in case opendir can’t open the directory (directory passed in the argument doesn’t exist). If this is the case, NULL will be stored in a variable dirp, and we will print an error in the screen.

Now, we need to read the content of this directory so we call function readdir. This creates a struct pointer with the information of the files and directories that we will store in the variable extra. The while loop allows us to go through all this files until reaching a NULL state (indicating everything in the directory has been read). Inside the loop we look inside the structure received in readdir and look for a string of characters containing the name of the file or directory and store it in variables names. This variable is printed afterwards.

Once the while loop is finished, only thing left to do is close the directory. We call closedir function and store is integer result in variable ending. If this variable is less than 0 (-1), we know that there was an error while closing, and print this error in the screen.

**Tests**

1. Introducing a correct input. **Works**.

*./myls p1\_tests*

Expected: The list of the files in the directory

Received: The list of the files in the directory

1. Introducing just the command. **Works**.

*./myls*

Expected: The list of files in our current directory

Received: The list of files in our current directory

1. Introducing inexisting directory . **Works**.

*./myls adiughaeirughiaegvaerugpieo*

Expected: Error while opening

Received: Error while opening: No such file or directory

4. Introducing the name of a file after the command.

*./myls Makefile*

Expected: Error while opening

Received: Error while opening: Not a directory

**Mysize**

For this program we used functions getcwd, opendir, readdir, open, lseek, close and closedir.

At the beginning we decided to print an error if any parameter is introduced along with the execution of the program, as we are requested to print information about the directory we are in.

At the beginning, we decided to just ignore if a parameter was entered or not, but we came to the conclusion that it didn’t have much sense to introduce a directory as parameter and show the content of another one.

Then the code continues just like myls, only that it opens the directory we are in, so we don’t need to enter any parameter when executing. Then that directory is opened. As readdir returns an structure, it will be saved to use its information later. This function is inside a while loop because we need to check that each element is a regular file. If it is not a regular file, it will go to the next file. If a regular file is found, then the file is opened, and using lseek the size of the file is obtained.

After that, the file is closed, and the program prints on the console the name of the file and its size. When there are no more files to read, the program exits the loop and closes the directory, finishing the program after that.

**Tests**

1. Using the program in the directory were it is allocated. **Works**.

*./mysize*

Expected: Print the list of files and their size

Received: Print the list of files and their size

1. Using the program for a different directory were it is allocated. **Returns an error.**

*./mysize*