Universidad Carlos III de Madrid


# OPERATING SYSTEMS

Bachelor's Degree in Computer Science & Engineering

*Lab 1: System Calls*

**Academic Year 2023/2024**

Inés Fuai Guillén Peña | 100495752 | 100495752@alumnos.uc3m.es | g89
Salvador Ayala Iglesias | 100495832 | 100495832@alumnos.uc3m.es | g89
Iván García Maestre | 100472753 | 100472753@alumnos.uc3m.es | g89

# TABLE OF CONTENTS

# gerror function

To check for errors, we have implemented a new function: ***gerror***. This function returns an integer and takes two arguments (*nParameter*: of int type, it represents the number of parameters passed, and *Parameter*: an array of pointers to characters, which are the program's arguments). Within the function, we declare two new variables: format (a char array of size 1000, initialized to 0) and size (an int, initialized to 0).

If no errors occurred during the implementation of this function, we return 0 and finish. On the other hand, if we encounter an issue, we need to determine the type of error in order to print the appropriate message corresponding to the error. For this, we create two conditional statements (if): if it is not a system error, we create a mandatory conditional in which nParameter is a negative int (that we will multiply by -1 later on to make it positive) to return a system error, as requested. In the case we have a system error, we create a for loop in which we will determine the type of error. Finally, we print the message with the type of error and return -1.

# mywc.c

## Definition:

Using the appropriate system calls and only the functions 'open', 'write', 'read', 'lseek', and 'close', we were instructed to implement a program called ***mywc*** whose main goal is to return the number of bytes, words, and lines that the file given as a parameter contains.

## Description of the code:

First, we include the necessary libraries: <stdio.h> (standard library for input and output), <string.h> (library which contains a set of functions for manipulating strings: copying, changing characters, comparing strings, etc.), <stdlib.h> (library for performing common operations: dynamic memory management, data type conversion, and random number generation), <fcntl.h> (library for performing operations on file descriptors), <unistd.h> (library for interacting with the Unix operating system: process management, files and directories, and interaction with the file system), <errno.h> (defines macros for values that are used for error reporting in the C library functions and defines the macro errno). We also define *SIZ_MAX*, which will be the size for storing the read data, *FALSE* with value 0 and *TRUE* with value !FALSE, which will be used as booleans in the main function.

Next, the main function is declared, which will return an integer and take two arguments (*argc*: represents the number of command-line arguments passed to the program, it is of integer type, and *argv*: array of pointers to characters, which are the

program's arguments). Then, within the main function, the following variables are declared: pfile (of type char, is a pointer in which we will save the content of the argument passed), fp (of type int, we will use it to check whether we are able to open the file or not), nlines, nwords, nbytes (all of them of type int, to store the requested data from the file and initialized to 0), and pline (of type char, is a pointer where we will save space memory). The arguments passed to the program must be two: the program's name and the file; if the number of arguments passed to the program is less than two, an error is thrown. Next, the file is opened; if there is any problem when opening it, an error is thrown.

If there are no problems, we declare two new variables: siz (of type int, initialized to 0) and word (of type int, initialized to FALSE). A while loop is created to read the data from the file (byte by byte) until the entire file is read. Inside the while loop, a for loop is created with a new variable: c (of type char, which will have the value of the character contained in the address we are reading). Each time we finish the for loop, the variable 'nbytes' (total bytes read from the file) will be increased. Within the for loop, we will also analyze which byte has just been read: if '\n' has been read, the value of the 'nlines' variable is incremented, and the variable 'bword' is set to FALSE; if any character different to ' ' or '\t' has been read and the variable 'bwords' is FALSE, we increment the variable 'nwords' and set 'bwords' to TRUE, meaning we have started reading a new word.

Before finishing the program, we set free the memory space we have used before ('pline'), close the file and print a message with the number of lines, number of words, number of bytes and the file's name. Finally, we return 0.

## Test-cases:

- **f1.txt:** Check when we have tabs between the words instead of spaces.

```
ines@orden08:~/lab1$ cat f1.txt
Name1   M       32      09834320        24500
Name2   F       35      32478973        27456
Name3   M       53      98435834        45000
```

```
ines@orden08:~/lab1$ mywc f1.txt
 3 15 78 f1.txt
ines@orden08:~/lab1$ wc f1.txt
 3 15 78 f1.txt
```

```
ines@orden08:~/lab1$  mywc f1.txt > d1
ines@orden08:~/lab1$ wc f1.txt > d2
ines@orden08:~/lab1$ diff d1 d2
ines@orden08:~/lab1$
```

- **test1.txt:** Check when we have lines without any character written.

```
ines@orden08:~/lab1$ cat test1.txt
hello world

this is a test to check if our function works correctly

_
_

bye!
```

```
ines@orden08:~/lab1$ mywc test1.txt
 8 16 80 test1.txt
ines@orden08:~/lab1$ wc test1.txt
 8 16 80 test1.txt
ines@orden08:~/lab1$ mywc test1.txt>d1
ines@orden08:~/lab1$ wc test1.txt>d2
ines@orden08:~/lab1$ diff d1 d2
ines@orden08:~/lab1$
```

- **test2.txt:** Check when the file is empty.

```
ines@orden08:~/lab1$ cat test2.txt
ines@orden08:~/lab1$
```

```
ines@orden08:~/lab1$ mywc test2.txt
0 0 0 test2.txt
ines@orden08:~/lab1$ wc test2.txt
0 0 0 test2.txt
ines@orden08:~/lab1$ mywc test2.txt > d1
ines@orden08:~/lab1$ wc test2.txt > d2
ines@orden08:~/lab1$ diff d1 d2
ines@orden08:~/lab1$
```

- **test3.tx:** Check when the file given has an incorrect format (i. e. the format name is incomplete)

```
ines@orden08:~/lab1$ mywc test3.tx
mywc: test3.tx: No such file or directory
ines@orden08:~/lab1$ wc test3.tx
wc: test3.tx: No such file or directory
ines@orden08:~/lab1$
```

# myls.c

Definition:

In this exercise, we were instructed to implement a program called ***myls*** whose main goal is to open a directory specified as an argument (or the current directory if no directory is specified as an argument) and display on the screen the name of all the entries of that directory, printing one entry per line.

## Description of the code:

First, we include the necessary libraries: <stdio.h> (standard library for input and output), <string.h> (library which contains a set of functions for manipulating strings: copying, changing characters, comparing strings, etc.), <stdlib.h> (library for performing common operations: dynamic memory management, data type conversion, and random number generation), <unistd.h> (library for interacting with the Unix operating system: process management, files and directories, and interaction with the file system), <errno.h> (defines macros for values that are used for error reporting in the C library functions and defines the macro errno), <dirent.h> (library used for directory access operations), <linux/limits.h> (library which contains symbolic names that represent standard values for limits on resources, in this case PATH_MAX has a value of 4096).

Next, the main function is declared, which will return an integer and take two arguments (*argc*: representing the number of command-line arguments passed to the program, it's of integer type and *argv*: an array of pointers to characters, which are the program's arguments). Then, within the main function, the following variables are declared: dir (of type DIR, is a pointer which will have the directory address passed as an argument), entry (of type char, is a pointer which points to a string that gives the name of a file in the directory), actual_path (of type char and size PATH_MAX, is an array initialized to 0 in which we will copy the directory passed as an argument).

First, we should check if we have received a parameter along with the function invocation. If not, we should take the directory upon which the function has been invoked (using *getcwd*). On the other hand, if a directory is passed, that will be the one we use for the resolution of the exercise.

Then, we check if we can open the directory: if we have not been able, we return an error; otherwise, a while loop is created to read all the directory entries (using the pointer 'entry') and we print the name of each one (accessing the name field of the structure). Before finishing, we check if the directory has been closed without issues; if there has been any, we return an error. If not, we return 0.

Test-cases:

- **Test when no directory name is given:** The function takes the current directory.

```
ines@orden08:~/lab1$ myls
mywc
mywc2
test1.txt
mk
..
k2
d1.txt
p2
f1.txt
new1.c
test2.txt
Makefile
f2.txt
d2.txt
d2
d1
t1
mywc2.c
.mywc.c.swo
file.txt
myls.c
p1
new1
t2
mywc.c.back
mywc.o
.
myls
new2.c
k1
test3.txt
mywc.c
myls.o
```

```
ines@orden08:~/lab1$ ls -f -1
mywc
mywc2
test1.txt
mk
..
k2
d1.txt
p2
f1.txt
new1.c
test2.txt
Makefile
f2.txt
d2.txt
d2
d1
t1
mywc2.c
.mywc.c.swo
file.txt
myls.c
p1
new1
t2
mywc.c.back
mywc.o
.
myls
new2.c
k1
test3.txt
mywc.c
myls.o
```

```
ines@orden08:~/lab1$ ls -f -1 > d1
ines@orden08:~/lab1$ myls > d2
ines@orden08:~/lab1$ diff d1 d2
ines@orden08:~/lab1$
```

- **Test when a directory is given**

```
ines@orden08:~/lab1$ myls dirA
..
f1.txt
f2.txt
.
myls.o
```

```
ines@orden08:~/lab1$ ls -1f dirA
..
f1.txt
f2.txt
.
myls.o
```

```
ines@orden08:~/lab1$ myls dirA > d1
ines@orden08:~/lab1$ ls -1f dirA > d2
ines@orden08:~/lab1$ diff d1 d2
ines@orden08:~/lab1$
```

- **Test when the directory does not exist**

```
ines@orden08:~/lab1$ myls dirX
myls: cannot access 'dirX': No such file or directory
ines@orden08:~/lab1$ ls -f -1 dirX
ls: cannot access 'dirX': No such file or directory
```

# myishere.c

## Definition:

In this exercise, we were instructed to implement a program called ***myishere*** whose main goal is to receive the name of a directory as a first argument and check if the file whose name is received as the second argument is in that directory.

## Description of the code:

This function follows the same structure as the previous exercise, with the addition of checking if the file is in the given directory.

First, we include the necessary libraries: <stdio.h> (standard library for input and output), <string.h> (library which contains a set of functions for manipulating strings: copying, changing characters, comparing strings, etc.), <stdlib.h> (library for performing common operations: dynamic memory management, data type conversion, and random number generation), <errno.h> (defines macros for values that are used for error reporting in the C library functions and defines the macro errno), <dirent.h> (library used for directory access operations), <linux/limits.h> (library which contains symbolic names that represent standard values for limits on resources, in this case PATH_MAX has a value of 4096), <sys/types.h> (library that defines a collection of *typedef* symbols and structures). We also define FALSE with value 0 and TRUE with value !FALSE, which will be used as booleans in the main function.

Next, the main function is declared, which will return an integer and take two arguments (*argc*: representing the number of command-line arguments passed to the program, it's of integer type and *argv*: an array of pointers to characters, which are the program's arguments). Then, within the main function, the following variables are declared: dir (of type DIR, is a pointer which will have the directory address passed as an argument), entry (of type char, is a pointer which points to a string that gives the name of a file in the directory), actual_path (of type char and size PATH_MAX, is an array initialized to 0 in which we will copy the directory passed as an argument), filename (of type char and size FILENAME_MAX, is an array initialized to 0 in which we will copy the file name passed as an argument).

First, we should check if we have received enough arguments, which must be three: the program's name, the directory and the file name; if the number of arguments passed to the program is less than three, an error is thrown. Next, the file is opened; if there is any problem when opening it, an error is thrown. If not, we declare a new variable (bfound: with initial value 0). Afterwards, a while loop is created to read all the directory entries and we check the name of each file name; if we find the name we are looking for, bfound is set to TRUE and we get out of the loop. Depending on whether we have found the desired file, we return one message or another; for this, we use the variable bfound.

Before finishing, we check if the directory has been closed without issues; if there has been any, we return an error. If not, we return 0 and finish.

Test-cases:

- **Test when the directory does not exist**

```
ines@orden08:~/lab1$ myishere
myishere: Not enough arguments
```

```
ines@orden08:~/lab1$ myishere dirA1 test2.txt
myishere: dirA1: test2.txt: No such file or directory
```

- **Test when the directory exists but the file does not exist**

```
ines@orden08:~/lab1$ myishere dirA test2.txt
File test2.txt is not in directory dirA
```

```
ines@orden08:~/lab1$ ls dirA
f1.txt  f2.txt  myls.o
```

- **Test when the file is in the directory given as a parameter**

```
ines@orden08:~/lab1$ myishere dirA f1.txt
File f1.txt is in directory dirA
```

# Conclusions and problems encountered

While working on the first exercise, ***mywc***, our main challenge was adapting to the syntax and usage of the C language. When it came to implementing the exercise functions, we did not face significant challenges.

For ***myls***, the main challenge was becoming familiar with managing directories and files.

With ***myishere***, since it was very similar to *myls*, there were no issues.

The most difficult part was getting started because, even though we knew some theory and a bit of C, we struggled with using the console, compiling, running files and understanding the correct functioning of pointers and structures.
Now, we're pleased to say that we have a better understanding of this language, environment, and operating system. We've also become familiar with Linux and its command operations.