BACHELOR'S DEGREE IN COMPUTER SCIENCE AND ENGINEERING

YEAR 2018/2019

# LAB 1. SYSTEM CALLS

**OPERATING SYSTEMS**

Marina Torelli Postigo 100383479 G88
Sergio Cruzado Muñoz 100346032 G89
María del Carmen Díaz de Mera Gómez-Limón 100383384 G88

# Table of contents

# 1. Mycat

## 1.1. Description of the code

The program consists only of the main function, which receives the name of a text file in argv[1], so argc will always equal 2.

First it gets the file descriptor, a positive integer using the open syscall using the O_RDONLY flag, as we only need to read the contents of the file and not to write in it. It checks for errors and if one occurs in this step then the function returns -1.

Then it will read the file in blocks of 1024 bytes and write the contents in the buffer. The syscall read returns the number of bytes read, so the while loop stops when it returns 0 (when it has read all the file) or when it returns -1 (there was an error). After reading nBytes it will call the write syscall, which will write the contents of the buffer and return the number of bytes read, which will have to equal the number of bytes read. Otherwise an error occurred. We use the constant STODUT_FILENOT as specified by the statement.

When the while loop ends, we check that it did not detect any errors in read. Then the file is closed, and it is checked that there are not any errors. Otherwise the syscall returns -1.

In all the cases where there is any kind of error the program returns -1, if everything goes as expected it returns 0.

## 1.2. Tests

| Objective | Expected Output | Procedures |
|---|---|---|
| General functionality of the program with a normal text file | Full text in same format as cat command | We use one of the given files for tests in the directory p1_tests/<br>./mycat p1_tests/f1.txt |
| Functionality with an empty text file | Nothing | We create an empty text file with the name empty_file.txt<br>./mycat p1_tests/empty_fie.txt |
| Functionality with a text file with more bytes than the buffer (1024 bytes) | Full text in same format as cat | We can use the website lipsum.com/ to create a file with more than 1024 bytes called Lorem Ipsum<br>./mycat p1_tests/Lorem Ipsum.txt |
| Check that the program doesn't work without an input | "The input isn't valid" | We call the program with no input<br>./mycat |
| Check that the program doesn't work with more than one input | "The input isn't valid" | We follow the program with two different text files as inputs<br>./mycat p1_tests/f1.txt p1_tests/f2.txt |
| Check that the program doesn't work when the input is a directory | "Error in reading: Is a directory" | We follow the program with the directory p1_tests/<br>./mycat p1_tests/ |
| Check that the program detects non-existing files | "Error in opening the file: No such file or directory" | We use as input the name of a file that doesn't exist<br>./mycat file.txt |

When running these tests, the expected output was obtained in all of them.

This means all test were successful and the program woks properly for all these different cases of execution.

# 2. Myls

## 2.1. Description of the code

The program only consists of the main program. The program either receives one or no inputs so the value of argc can only be 1 or 2, otherwise the program is finished and it returns -1.

If the value of argv[1] isn't NULL (or alternatively, argc==2), it is considered as the received input the path of a directory, and the opendir syscall is directly used. Otherwise, when no input has been received, the path of the current directory is obtained using the getcwd syscall. If there are not any errors (getcwd returns NULL), the path of the current directory is kept in the buffer, which has been created using the constant PATH_MAX. Once done, the opendir function is called using the contents of the buffer as input.

It is checked that no errors occurred when opening the directory, when the opendir syscall returns NULL and then it proceeds to read the entries of the directory. Before using the readdir syscall, the value of errno (which we can access with the header <errno.h>) is stored. This is done because if the readdir function returns NULL it can either mean that it finished reading the entries or that there was an error. The only way of knowing if it was an error is checking that the value of errno is not changed.

A while is used to call the readdir syscall until it returns NULL and then print the name of the current entry. It is stored in the field d_name of the dirent structure returned by the readdir syscall.

When the while finishes, we compare the current value of errno with the one previously stored to check for errors in readdir and then close the directory using the closedir syscall, checking that it doesn't return -1 in case of error.

## 2.2. Tests

| Objective | Expected Output | Procedures |
|---|---|---|
| General functionality with a different directory than the current one. | .<br>..<br>All contents of the directory listed | ./myls p1_tests/ |
| General functionality with current directory. | .<br>..<br>All contents of the directory listed | We call the program with no inputs.<br>./myls |
| Check functionality with an empty directory. | .<br>.. | We use one of the empty directories in the p1_tests/ folder.<br>./myls p1_tests/dirA |
| Check functionality with a non-existing directory. | “Error in opening directory: No such file or directory” | We use as input a directory that doesn’t exist.<br>./myls directory |
| Check that the program doesn’t work with more than one input. | “The input isn’t valid” | We follow the name of the program with two different directories.<br>./myls p1_tests/dirC p1_tests/dirA |
| Check that the program detects when the input isn’t a directory | “Error in opening directory: Not a directory” | We use as input the name of a text file for example<br>./myls p1_tests/f1.txt |

When running these tests, the expected output was obtained in all of them.

This means all test were successful and the program woks properly for all these different cases of execution.

# 3. Mysize

## 3.1. Description of the code

The program consists only of the main function. As it doesn't receive any input, the value of argc should always be one, in any other case, the program returns -1.

First, the program gets the path of the current working directory using the getcwd syscall in the same way as myls. If the function doesn't return NULL we get the path in the buffer which we use as input to opendir. We check again for errors, which happens when the opendir function returns NULL.

Similarly to myls we store the value of errno and use a while to call the readdir function until it returns NULL. We check each entry to look for the regular files, when the value of the field d_type of the dirent structure equals the constant DT_REG. If it is a regular file, we use the open syscall to get the file descriptor, using the flag O_RDONLY as in mycat. We check that the function didn't return -1 and there was an error and proceed to get the size. We can get the size as the value returned from lseek when we move the pointer to the end of the file (SEEK_END with an offset=0) so the function returns its position measured in bytes. After getting the size we close again the file checking for errors in case the syscall close returns -1 and we print the name of the file and its size separated by a tab.

When the while loop ends we check if there was any error in readdir comparing the current value of errno with the one previously stored. Then we use the closedir syscall, checking again that it doesn't return -1 and there was an error.

In the next section there are shown several tests which have been performed for this program and they were all successful in obtaining the expected correct output.

### 3.2. Tests

| Objective | Expected Output | Procedures |
|---|---|---|
| General functionality | File1 Size1<br>File2 Size2<br>... | ./mysize |
| Check functionality with an empty directory | Nothing | We use for example the dirC directory inside p1_tests when the program is in the P1_material folder<br>../../mysize |
| Check functionality with non-regular files (like other directories) in the directory | Only files listed, not directories | We use the p1_tests folder which contains both directories and files when the program is in the P1_material folder<br>../mysize |
| Check functionality with an empty file in the directory | empty_file.txt 0 | We add an empty text file in the p1_tests folder and call again the program<br>../mysize |
| Check that the program doesn't accept any input | "The input isn't valid" | We follow the program with an input<br>./mysize p1_tests |

## 4. Conclusion

In the beginning we had some trouble understanding how all the inputs for the syscalls worked and what they were for but making use of the manual for the terminal, where everything was well explained, we were able to properly understand how they worked.

The programs themselves did not have great difficulty, as they have a direct implementation, which one could program easily just in the main. In addition, they did not take a lot of time. It took more time to write the report and to do the test cases, all while checking that everything worked as it should.