Computer Architecture and Technology Area

Universidad Carlos III de Madrid

OPERATING SYSTEMS

Lab 1. System Calls

Bachelor's Degree in Computer Science & Engineering
Bachelor's Degree in Applied Mathematics & Computing
Dual Bachelor in Computer Science & Engineering & Business Administration

Year 2021/2022

Contents

1 Lab Statement

This lab allows the student to familiarize with Operating System calls (specially related to the file system management) following the POSIX standard. Unix allows you to make calls directly to the Operating System from a program implemented in a high level language, in particular, C language.

Most of input/output (I/O) operations in Unix can be done using uniquely five calls: `open, read, write, lseek and close`.

For the Operating System kernel, all files opened are identified using *file descriptors*. A file descriptor is a non negative integer. When we open a file that already exists, the kernel returns a file descriptor to the process. When we want to read or write from/to a file, we identify the file with the file descriptor that was returned by the open call.

Each open file has a current *read/write position* ("**current file offset**"). It is represented by a non negative integer that measures the number of bytes from the beginning of the file. The read and write operations normally start at the current position and create an increment in that position equal to the number of bytes read or written. By default, this position is initialized to 0 when a file is opened, unless the option `O_APPEND` is specified. The current position `current_offset`) of an open file can be changed explicitly using the system call `lseek`.

To manipulate directories, you can use the system calls *opendir, readdir y closedir*. An open directory is identified with a directory descriptor, which is a pointer of type DIR (*DIR**). When we open a directory with *opendir*, the kernel returns a directory descriptor from which the different entries to that directory can be read using the calls to the function *readdir*. The call readdir returns a directory entry in a pointer to a structure of type dirent (*struct dirent**). Such structure will contain the fields corresponding to that entry such as the name of the entry, or the type (if it is a normal file , another directory, symbolic links, etc.). Repeated calls to the function *readdir* will be returning the next entries in an open directory.

1.1 Lab Description

In this lab you will be implementing three C programs which use the system calls that were previously described. These programs will be **mycat**, **myls** and **mysize**. For this purpose you will have the following files with initial code *mycat.c*, *myls.c* and *mysize.c*.

1.1.1 mycat

The first program, **mycat** will open the file specified by argument and will show its content through standard output (the console) using the calls **open**, **read**, **write** and **close**. For this purpose:

- It will open the file specified as parameter with **open**.
- It will read the contents of the file using an intermediate buffer of **1024 bytes** (**read**).

- It will write (**write**) the content of the buffer in the standard output. Use the constant *STDOUT_FILENO* as value of the descriptor to write to the standard output.
- Finally, it will close the descriptor using **close**.

```
$ ./mycat p1_tests/f1.txt
Name1    M    32    09834320    24500
Name2    F    35    32478973    27456
Name3    M    53    98435834    45000
```

- **Usage:** `./mycat <path_input_file>`
- **Requirements:**
  - The program must show the whole **contents of the file.**
  - The program must return **-1** if no argument was passed.
  - The program must return **-1** if there was an error when opening the file (e.g. the file does not exist).
- **Test suggestion:**[1] Check that the output of the program over one file corresponds with the one offered by the command ***cat*** (no extra arguments) over that same file. For this purpose, it is recommended to compare the two outputs with the command ***diff***.

1.1.2 myls

The second program ***myls***, will open a directory passed as parameter (or the current directory if no directory is specified) and print on the screen all the entries that this directory contains, **one per line**. This program will:

- Obtain the specified directory from the arguments to the program or obtain the current directory using the call *getcwd*. Use the constant `PATH_MAX` as maximum size that can have the path of the current directory.
- Open the directory using ***opendir***.
- Then, it will read in each of the entries of the directory using ***readdir*** and print the name of the entry using ***printf***.
- Finally, it will close the descriptor of the directory through the call ***closedir***.

```
$ ./myls p1_tests/
dirC
f1.txt
dirA
f2.txt
.
..
```

[1] To fulfil this test is no guarantee of having the maximum mark in this exercise. It is just a suggestion so the students can check the general execution of their program. The students must also meet the other requirements of the program, write an adequate code, comment it, test extreme cases, and, in general, meet the other demands described in this statement.

- **Usage 1:** `./myls <directory>`
- **Usage 2:** `./myls`
- **Requirements**
  - The program must list **all entries** of the directory , in the order in which the call to *readdir* returns them, and showing each entry in one line.
  - The program must list the entries of the directory passed as parameter (usage 1), or from the current directory if no parameter was passed (usage 2).
  - **myls** must show the current directory (.) and the parent directory (..).
  - The program must return **-1** if an error happened while opening the directory (e.g, the directory does not exist).
- **Test suggestion:**[2] Check that the output of the program over a directory corresponds with the one obtained with the command `ls -f -l` over that same directory. For this purpose, it is recommended to compare the two outputs with the command ***diff***.

1.1.3 mysize

The third program, **mysize**, will obtain the current directory and will list all regular files that it contains as well as its size in bytes. For this purpose:

- It will obtain the current directory using the call *getcwd*.
- Use the constant `PATH_MAX` as maximum size of the path of the current directory.
- Open the file using ***opendir***.
- Then, it will read the entries of the directory using ***readdir***.
  - If the entry is a regular file (field *d_type* form the structure *dirent* equal to the constant `DT_REG`).
    1. Open the file using ***open***.
    2. Move the file pointer to the end of the file and obtain its value with ***lseek***.
    3. Close the file with ***close***.
    4. **Print the name of the file (field *d_name* of the structure dirent), followed by 4 blank spaces, and the size obtained by *lseek*, ending with an End Of Line character.**
- This procedure will be repeated for every entry in the directory.
- Finally it will close the directory descriptor with ***closedir***.

[2] To fulfil this test is no guarantee of having the maximum mark in this exercise. It is just a suggestion so the students can check the general execution of their program. The students must also meet the other requirements of the program, write an adequate code, comment it, test extreme cases, and, in general, meet the other demands described in this statement.

```
$ cd p1_tests/
$ ../mysize
f1.txt      87
f2.txt      87
```

- **Usage:** `./mysize`
- **Requirements:**
  - The program must show the name and size of all the regular files of the directory, in the order in which the call ***readdir*** returns them, and showing the data of each file in one line.
  - The program will only show the data from regular files.
  - The program will show the data using the following format:
    <name><4_blank_spaces><size>
  - The program must return **-1** if there was an error when opening the file.

1.1.4 Initial Code

In order to facilitate the realization of this assignment an initial code is provided in the file p1_system_calls_2022.zip. To extract its contents you can use the unzip command:

```
unzip p1_system_calls_2022.zip
```

As a result, you will find a new directory p1_system_calls/, onto which you must code the different programs. Inside this directory you will find:

- **Makefile**
  Do NOT modify this file. File used by the `make` tool to compile all programs. Use *make* to compile the programs and *make clean* to remove the compiled files.
- **mycat.c**
  This file must be modified. C Source file to code **mycat**.
- **myls.c**
  This file must be modified. C Source file to code **myls**
- **mysize.c**
  This file must be modified. C Source file to code **mysize**
- **authors.txt**
  This file must be modified. `txt` file where to include the authors of the practice.
- **checker_os_p1.py**
  Do NOT modify this file. Corrector provided for practice.
- **p1_tests/**
  This directory contains example files and directories to be able to execute and test your programs.

1.1.5 Program tester

The **python (Version 3)** script **checker_os_p1.py** is given to verify that the student submission follows the format conventions (it has the correct names and it is well compressed) and run some functionality tests, printing on screen the grade obtained with the provided code. The tester must be executed in the Linux computers of the the Virtual Aulas of the university.

The command to execute the tester is the following:

```
python3 checker_os_p1.py <submitted_file.zip>
```

Being submitted_file.zip the file that it is going to be delivered in Aula Global (see next section). Example:

```
$ python3 checker_os_p1.py ssoo_p1_100254896_100047014.zip
```

The format tester will print on the console messages stating whether the required format is correct or not.

2 Assignment submission

2.1 Deadline and method

The deadline for the submission of the internship in AULA GLOBAL will be **March 11th, 2022 (until 23:55h)**.

2.2 Submission

The submission must be done using Aula Global using the links available in the first assignment section and **by a single member of the group**. **The submission must be done separately for the code and report**. The report will be submitted through the TURNITIN tool.

2.3 Files to be submitted

You must submit the code in a zip compressed file with name:

os_p1_AAAAAAAAA_BBBBBBBBB_CCCCCCCC.zip

Where A...A, B...B, and C...C are the student identification numbers of the group. A maximum of 3 persons is allowed per group, if the assignment has a single author, the file must be named **os_p1_AAAAAAA.zip**. **The zip file will be delivered in the deliverer corresponding to the code of the practice.** The file to be submitted must contain:

- **Makefile**
- **mycat.c**
- **myls.c**
- **mysize.c**

- **authors.txt: Text file with the information of the authors in different lines, with the following format (csv): NIA, Surname, Name**

The report must be submitted in a PDF file. Notice that only PDF files will be reviewed and marked. The file must be named:

os_p1_AAAAAA_BBBBBB_CCCCCC.pdf

The report must contain at minimum:

- **Description of the code** detailing the main functions it is composed of. Do not include any source code in the report.
- **Tests cases** used and the obtained results. All test cases must be accompanied by a description with the motivation behind the tests. In this respect, there are three clarifications to take into account:
  1. Avoid duplicated tests that target the same code paths with equivalent input parameters.
  2. Passing a single test does guarantee the maximum marks. This section will be marked according to the level of test coverage of each program, nor the number of tests per program.
  3. Compiling without warnings does not guarantee that the program fulfills the requirements.
- **Conclusions**, describe the main problems found and how they have been solved. Additionally, you can include any personal conclusions from the completion of this assignment.

Additionally, marks will be given attending to the quality of the **report**. Consider that a minimum report:

- Must contain a title page with the name of the authors and their student identification numbers.
- Must contain a table of contents.
- Every page except the title page must be numbered.
- Text must be justified

The PDF file must be submitted using the TURNITIN link. Do not neglect the quality of the report as it is a significant part of the grade of each assignment.

NOTE: It is possible to submit the lab code as many times as you wish within the deadline, being the last submission the final version. **THE LAB REPORT CAN ONLY BE SUBMITTED ONCE ON TURNITIN.**

3 Rules

1. Programs that do not compile or do not satisfy the requirements will receive a mark of **zero**.
2. **Programs that use library functions (`fopen`, `fread`, `fwrite`, `etc.`) or similar, instead of system calls, will receive a grade of zero**.
3. All programs should compile without reporting any warnings.
4. Programs without comments will receive a **very low grade**.
5. The assignment must be submitted using the available links in Aula Global. Submitting the assignments by mail is not allowed without prior authorization.
6. The programs implemented must work in a virtual machine running Ubuntu Linux or in the Virtual Aulas provided by the informatics lab at the university platform. It is the student responsibility to be sure that the delivered code works correctly in those places.
7. It is mandatory to follow the input and output formats indicated in each program implemented. In case this is not fulfilled there will be a penalization to the mark obtained.
8. It is mandatory to implement error handling methods in each of the programs.
9. Students are expected to submit original work. In case plagiarism is detected between two assignments both groups will fail the continuous evaluation. Additional administrative charges of academic misconduct may be filled.

Failing to follow these rules will be translated into zero marks in the affected programs.

4 Appendix – System calls

A system call allows user programs to request services from the operating system. In this sense, system calls can be seen as the interface between the user and kernel spaces. In order to invoke a system call it is necessary to employ the functions offered by the underlying operating system. This section overviews a subset of system calls offered by Linux operating systems that can be invoked in a C program. As any other function, the typical syntax of a system calls follows:

```
status = function (arg1, arg2,.....);
```

4.1 I/O system calls

```
int open(const char * path, int flag, ...)
```

The file name specified by `path` is opened for reading and/or writing, as specified by the argument `flag`; the file descriptor is returned to the calling process.

More information: `man 2 open`

```
int close(int fildes)
```

The close() call deletes a descriptor from the per-process object reference table.

More information: `man 2 close`

```
ssize_t read(int fildes, void * buf, size_t nbyte)
```

Read() attempts to read `nbyte` bytes of data from the object referenced by the descriptor `fildes` into the buffer pointed to by `buf`.

More information: `man 2 read`

```
ssize_t write(int fildes, const void * buf, size_t nbyte)
```

Write() attempts to write `nbyte` of data to the object referenced by the descriptor `fildes` from the buffer pointed to by `buf`.

More information: `man 2 write`

```
off_t lseek(int fildes, off_t offset, int whence)
```

The lseek() function repositions the offset of the file descriptor `fildes` to the argument offset, according to the directive whence. The argument `fildes` must be an open file descriptor. Lseek() repositions the file pointer `fildes` as follows:

- If whence is `SEEK_SET`, the offset is set to offset bytes.
- If whence is `SEEK_CUR`, the offset is set to its current location plus offset bytes.
- If whence is `SEEK_END`, the offset is set to the size of the file plus offset bytes.

More information: `man 2 lseek`

4.2 File related system calls

```
DIR * opendir(const char * dirname)
```

The `opendir()` function opens the directory named by `dirname`, associates a directory stream with it, and returns a pointer to be used to identify the directory stream in subsequent operations.

More information: `man opendir`

```
struct dirent * readdir(DIR * dirp)
```

The `readdir()` function returns a pointer to the next directory entry. It returns NULL upon reaching the end of the directory or detecting an invalid `seekdir()` operation. The *dirent* structure contains a field *d_name* (*char * d_name*) with the filename and a *d_type* field (*unsigned char d_type*) with the type of file.

More information: `man readdir`

```
int closedir(DIR * dirp)
```

The `closedir()` function closes the named directory stream and frees the structure associated with the *dirp* pointer, returning 0 on success.

More information: `man closedir`

4.3 Manual (man function)

man is a command that formats and displays the online manual pages of the different commands, libraries and functions of the operating system. If a section is specified, man only shows information about name in that section. Syntax:

```
man [section] open
```

A man page includes the synopsis, the description, the return values, example usage, bug information, etc. about a name. The utilization of man is recommended for the realization of all lab assignments. **To exit a man page, press q**.

5 Bibliography

- El lenguaje de programación C: diseño e implementación de programas Félix García, Jesús Carretero, Javier Fernández y Alejandro Calderón. Prentice-Hall, 2002.
- The UNIX System S.R. Bourne Addison-Wesley, 1983.
- Advanced UNIX Programming M.J. Rochkind Prentice-Hall, 1985.
- Sistemas Operativos: Una visión aplicada Jesús Carretero, Félix García, Pedro de Miguel y Fernando Pérez. McGraw-Hill, 2001.
- Programming Utilities and Libraries SUN Microsystems, 1990.
- Unix man pages (`man function`)