Computer Architecture and Technology Area (ARCOS)

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's in Computer Science & Engineering & Business Administration

Academic Year 2024/2025

Index

1 Lab Statement

This lab allows the student to become familiar with the OS system calls (in particular, the file system) following the POSIX standard. Unix allows system calls to be made directly from a program written in a high-level language, particularly C. Most file input/output (I/O) operations in Unix can be performed using only five calls: open, read, write, lseek and close.

For the operating system kernel, all open files are identified by file descriptors. A file descriptor is a non-negative integer. When we open (open) an existing file, the kernel returns a file descriptor to the process. When we want to read or write to/from a file, we identify the file with the file descriptor returned by the previously described 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. Read and write operations usually start at the current position and cause an increase in that position, equal to the number of bytes read or written. By default, this position is initialized to 0 (zero) when a file is opened, unless the **O_APPEND** option is specified. An open file's current position (current_offset) can be changed explicitly using the lseek system call.

1.1 Lab Description

The aim is to implement two C programs that use the system calls previously described. These programs will be **crear** and **combine**. For this, you will have the corresponding code files *crear.c* and *combine.c*.

1.1.1 crear

The first program, **crear**, will create a file with a specific protection mode. Arguments of the program are the name of the file and the specific creation mode (in octal). It has to create the file with the indicated mode (octal). In case of an error, a warning must be displayed via the error output.

As the operating system may have defined a default creation mask, so before creating the file you must remove the creation mask (keeping copy). After creating the file, you have to restore the previous mask.

Usage: `./crear <fichero> <modo>`
`E.g.: ./crear mydat.txt 755`

- **Requirements:**

- The program must return -1 if no input argument or a wrong number of arguments is passed.
- The program must return -1 if there was an error creating the file (e.g. the file already exist).
- The program must return 0 if everything was correctly.

- **Suggestion of test:**
  - Check that the file has been created well using "ls -l" command in the directory. You should see the file name with the permissions requested in rwxrwxrwx mode.

1.1.2 combine

The second program, ***combine***, will merge data from two input files, including records from students in a course, storing the data combination in a new third file passed by parameter. This output file must be ordered by note in ascending order. **NOTE:** the third file is not a text file. The data is written as binary data.
The data structure including students' data is like the following:

```
struct alumno{
      char nombre[50];
      int nota;
      int convocatoria;
};
```

There will not be repeated student entries in the files. Also, the maximum number of allowed students is 100. If more than 100 are detected, print an error and finish the execution.
Moreover, the program must create another file, called "`estadisticas.csv`" (values separated with ";") including a statistic of students' scores in the merged file. You must make statistics of students' scores with the following classification:

- M: the score is 10.
- S: the score is 9.
- N: the score is 8 or 7.
- A: the score is 6 or 5.
- F: the score is lower than 5.

Each category must include: tag of the category, number of students in this category, and percent of students in this category (with 2 decimals). Example:
M;54;23.50%
S;32;15.55%

- **Usage 1:** ./combine <course file 1> <course file 2> <output file >

Requirements:

- The program must return -1 if no input argument or a wrong number of arguments is passed.
- The program must return **-1** if there was an error opening the input files (e.g. an input file does not exist).

Suggestion of test:[1] Check that the output of the program in the statistics file match the merged file output.

Note: The library string.h contains the function strcmp. You can use it to compare two strings.

1.1.3 Support Code

The **p1_syscall_ENG_aula_global.zip** file is made available to support the development of this lab. To extract its content, run the following command:

```
unzip p1_syscall_ENG_aula_global.zip
```

After extracting its content, the p1-syscall directory is created, where the lab must be developed. The following files are included in this directory:

- **Makefile**
  DO NOT modify. Source file for the make tool. It automatically recompiles the source files that are modified. Use *make* to compile the programs, and *make clean* to delete the compiled files.
- **crear.c**
  Must be modified. C source file where students must code the **crear** program.
- **combine.c**
  Must be modified. C source file where students must code the **combine** program.
- **authors.txt**
  Must be modified. txt format file where the authors of this practice will be included. One line per author
  Important: format should be:
  `NIA, Surname, Name`

1.1.4 Program tester

The **python (Version 3)** script **probador-ssoo-p1.py** is given to the student to verify that the submission follows the format conventions (the files have the correct names and the ZIP is well

[1] Meeting this test is not a guarantee of getting the maximum grade in the exercise. It is only a suggestion for students to check the general operation of their program. Students must also meet the other requirements of the program, perform the appropriate code, comment it, test extreme cases, and generally meet the other conditions described in the lab statement.

compressed) and run some functionality tests, printing on the screen a tentative grade obtained with the provided code. The tester must be executed in Ubuntu Linux Virtual Machines or in the Virtual Classrooms provided by the Informatic Department laboratory. In order to use the tester correctly, the ZIP should be extracted into the project folder. To run the program tester, use the following command:

$ *python* **probador-ssoo-p1.py** *<submission.zip>*

Where submission.zip is the file that will be delivered to Aula Global (see next section). Example:

$ *python* **probador-ssoo-p1.py** *os_p1_100254896_100047014.zip*

The program tester will print messages on the screen indicating whether the format is or is not correct.

2 Lab Submission

2.1 Delivery Deadline

The deadline for the submission of the laboratory in AULA GLOBAL will be **March 3th, 2025 at 23:55h**

2.2 Submission Procedure

The delivery of the laboratory must be done electronically and by **a single member of the group**. In AULA GLOBAL, a link will be enabled to deliver the lab. Specifically, **a link will be enabled for the lab code, and another with TURNITIN for the lab report**.

2.3 Files to Deliver

A compressed file in zip format must be delivered with the name:

Os_p1_NIA1_NIA2_NIA3.zip

With the NIA of the group members. In case of doing the practice alone, the format will be **os-p1_NIA.zip**. **The zip file will be delivered in the corresponding link for the lab code**. The file must contain:

- **Makefile**
- **crear.c**
- **combine.c**
- **authors.txt: Text file in CSV format with one author per line.**
  - The format is: NIA, Surnames, Name

NOTE

To compress these files and be processed correctly by the provided program tester, it is recommended to use the following command:

$ *zip os-p1-NIA1-NIA2-NIA3.zip Makefile crear.c combine.c authors.txt*

The report will be delivered in PDF format in a file called:

os-p1-NIA1-NIA2-NIA3.pdf

Only reports in pdf format will be corrected and graded. They must contain at least the following sections:

- **Description of the code** detailing the main functions implemented. DO NOT include any source code in this section of the report. Any code will be automatically ignored.
- **Test cases** used and results obtained from their execution: Higher scores will be given to advanced tests, extreme cases, and in general, to those tests that guarantee the correct operation of the functions in all cases. There are three clarifications to take into account:
  1. If a program compiles correctly and without warnings is not a guarantee that it will work correctly.
  2. Avoid duplicating tests that assess the same program flows. The score in this section is not measured by the number of tests, but by the degree of test coverage. Few tests that evaluate different cases are better than many tests that always evaluate the same case.
- **Conclusions**, problems found, how they have been solved, and personal opinions.

The following aspects related to the **presentation** of the lab report will also be scored:

- It must contain a cover page, with the authors and their NIAs.
- It must contain an index of contents.
- The report must have page numbers on all pages (except the cover).

The pdf file will be delivered in the corresponding deliverer for the practice report (TURNITIN deliverer). THE LAB REPORT CAN ONLY BE DELIVERED ONCE THROUGH TURNITIN.

NOTE: It is possible to deliver the lab code as often as you want within the delivery period, with the last delivery being considered the definitive version.

Important: If you use AI tools and several groups deliver the same code, if will be considered a copy.

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. It is also not allowed to use statements or functions such as goto or stat**.
3. 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.
4. All programs should compile without reporting any **warnings**.
5. The programs implemented must work in a virtual machine running Ubuntu Linux and 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.
6. Programs without comments will receive a **very low grade**.
7. The assignment must be submitted using the available links in Aula Global. Submitting the assignments by mail is not allowed without prior authorization.
8. 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.
9. It is mandatory to implement error handling methods, **beyond what is explicitly requested, in each of the programs**.
   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 Related 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 creat(const char *path, mode_t mode);*

The file name specified by path is created with protection mode. the file descriptor is returned to the calling process. More information: man 2 creat

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 _ SEEKSET, the offset is set to offset bytes.
- If whence is _ SEEKCUR, the offset is set to its current location plus offset bytes.
- If whence is _ SEEKEND, the offset is set to the size of the file plus offset bytes.

More information: man 2 lseek

4.2 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

- Introduction to C. https://www.w3schools.com/c/c_intro.php
- Language C portal. https://en.cppreference.com/w/c
- Michael Kerrisk. The Linux Programming Interface. 2010. No Starch Press, ISBN 978-1-59327-220-3.
- Advanced UNIX Programming M.J. Rochkind Prentice-Hall, 1985.
- Programming Utilities: https://www.geeksforgeeks.org/introduction-of-system-call/
- Linux man pages (man function)