Computer Architecture and Technology Area (ARCOS)

Universidad Carlos III de Madrid

Operating Systems

Laboratory 1. System calls

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

Academic Year 2025/2026

Index

1 Laboratory Statement

This laboratory allows students to become familiar with system calls to the operating system under the POSIX standard. Unix allows system calls to be made directly from a high-level language program, particularly in C. Most file input/output (I/O) in Unix can be performed using only a few calls: *open, creat, read, write, lseek*, and *close*. The same applies to directories: *opendir, readdir*, etc.

For the operating system kernel, all open files are identified by file descriptors, which are non-negative integers. 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 using the descriptor returned by the previously described system call.

Each open file has a **current file offset**, represented by a non-negative integer that measures the number of bytes from the beginning of the file. Read and write operations normally begin at the current position and increase it by the number of bytes read or written. By default, this position is initialized to 0 when a file is opened, unless the O_APPEND option is specified. The current offset can be explicitly changed using the *lseek* system call.

On the other hand, to work with directory information (instead of files), there are also system calls designed for that purpose. As with files, directories have an internal pointer for their entries, and they can be traversed to obtain different data. This allows traversal of the filesystem tree without external libraries or components.

More information on these functions to perform system calls can be found in the Appendix section.

1.1 Description of the laboratory

You must implement two programs in C: **mycalc** and **mydu**, using the system calls described above and the provided source files mycalc.c and mydu.c.

1.1.1 mycalc

The first program, **mycalc**, acts as a simple calculator. It has two modes: **Interactive Calculator** and **History** modes.

The program in **Interactive Calculator** mode receives 3 arguments:

1. an integer,
2. a character defining an operation, and
3. a second integer.

The program must compute the requested operation and display the result on screen. The allowed operations are: addition (+), subtraction (-), multiplication (x), and division (/).

Every successfully executed operation must be stored in a log file named "**mycalc.log"**, with one operation and its result per line.

The program in **History** mode receives 2 arguments:

1. "-b" to indicate history mode and
2. an integer identifying the line number of the operation to retrieve from the log.

This functionality must read the corresponding line from the log file and display it on the screen. No recalculation is needed – only display.

Usage 1 (Interactive Calculator): `./mycalc <num1> <operation> <num2>`
Usage 2 (History): `./mycalc -b <operation number>`

- **Requirements:**
  - The program must not use "**stdio.h**"; only system calls may be used.
  - Errors must be printed to standard error.
  - Errors must not be written to the log.
  - Correct information must be printed to standard output.
  - The program must return -1 on any error.
  - The program must return -1 if the operation symbol is invalid.
  - The program must return -1 if the history flag is incorrect.
  - The program must return 0 if everything worked correctly.
- **Test suggestion:** Verify that messages are displayed by the correct output. Verify that the line indicated in history mode obtains the desired line.

Example of program output in calculator mode:

```
Shell
./mycalc 5242 x 3442
Operation: 5242 x 3442 = 18042964
./mycalc 5242 x
Usage (1): ./mycalc <num1> <operation (+|-|x|/)> <num2>
Usage (2): ./mycalc -b <num operation>
./mycalc 8374 + 1121
Operation: 8374 + 1121 = 9495
./mycalc 8374 / 0
Error: Division by zero
```

Considering the next **example of the log file**:

```
Shell
Operation: 5242 x 3442 = 18042964
Operation: 8374 + 1121 = 9495
Operation: 3 x 21 = 63
Operation: 3 / 21 = 0
Operation: 21 / 3 = 7
Operation: 5 x 4 = 20
Operation: 5 x 34 = 170
```

The example output of the program in historical mode is obtained:

```
Shell
./mycalc -b 1
Line 1: Operation: 5242 x 3442 = 18042964

./mycalc -b 4
Line 4: Operation 3 / 21 = 0

./mycalc -b -1
Error: Invalid line number

./mycalc -b 10
Error: Invalid line number
```

1.1.2 mydu

The second program, **mydu**, calculates the size of a directory and all its contents recursively. This means that if the directory contains subdirectories (and those contain more, etc.), the total size (in **KB**) must include all levels.

The program has three modes:

1. Calculate the size of a directory passed as an argument.
2. Calculate the size of the current directory if no argument is provided.
3. Read a binary file containing the history of analyzed directories. Every time "./mydu [directory]" is executed, the results printed to the screen must also be written to a binary file named **mydu.bin**. It is recommended to create a structure with two fields (size and path) and write that structure to the file. Using the flag "-b" prints the contents of the binary file in a readable format.

NOTE

For this exercise, library functions such as "printf" and "perror" are allowed to display messages on the screen, both through standard output and error output, but writing/reading files must be done using system calls (open, read, write, lseek, close).

Usage 1: `./mydu [<directory>]`
Usage 2: `./mydu [-b]`

- **Requirements:**
  - The program must return **-1** on any error.
  - Error messages must be printed to standard error, while correct information must be printed to standard output.
  - If correct, the program must return **0**.
  - Listed directories with size must show the full path from the indicated directory (do not list only the file name).
  - Regular files are NOT valid arguments. If a file is detected, print its name and an error message.
  - The binary file must be named "mydu.bin"
- **Test suggestion:** Check that the calculation performed is the same as that provided by the "du" command in the terminal on the same directory. The final tests must be performed on the Guernika server at the UC3M Computer Science Laboratory (https://www.lab.inf.uc3m.es/servicios/aulas-virtuales-del-laboratorio/).

Example Directory Structure:

```
folder1
     |_ file(s)
     |_ folder2
          |_ file(s)
     |_ folder3
          |_ file(s)
```

Example output on Guernika

Shell

```
./mydu
3744   folder1/folder2
3168   folder1/folder3
8352   folder1
8376    .
```

```
./mydu .
3744   folder1/folder2
3168   folder1/folder3
8352   folder1
8412    .
# Note that the size of "." has increased because the binary file "mydu.bin" has
been created after the first run.

./mydu folder1/folder2
3744   folder1/folder2

./mydu mydu.bin
mydu.bin: It is not a directory
```

Example output in reading mode of the binary file (after the previous process):

```
Shell
./mydu -b

--- Contents of binary file ---
3744    ./folder1/folder2
3168    ./folder1/folder3
8352    ./folder1
8376    .
3744    ./folder1/folder2
3168    ./folder1/folder3
8352    ./folder1
8412    .
3744    folder1/folder2
```

1.1.3 Supporting source code

To facilitate this exercise, the file **ssoo_p1_calls_2026.zip** is available, containing supporting source code. To extract its contents, run the following command:

```
unzip ssoo_p1_calls_2026.zip
```

When extracting its contents, the p1_calls/ directory is created, where the exercise should be carried out. The following files will be included in this directory:

- Makefile

  This file should <u>NOT</u> be modified. Source file for the make tool. It is used to automatically recompile only those source files that have been modified. Use “make” to compile the programs, and “make clean” to delete the compiled files.

- mycalc.c
  Must be modified. C source file where students must code the **mycalc** program.

- mydu.c
  Must be modified. C source file where students must code the **mydu** program.

- authors.txt
  Must be modified. Text file (.txt) containing the authors of the laboratory (one per line). Format (without accents):
  NIA1,Surname(s),Name(s)
  NIA2,Surname(s),Name(s)
  NIA3,Surname(s),Name(s)

```
Textproto
Valid example:
100499455,Surname1 Surname2,Name
100499455,Surname1 Surname2,Name1 Name2
```

```
Textproto
Invalid example:
,Surname1 Surname2,Name1 (Not include NIA)
(Whitespace)
Surname1 Surname2,Compound name,100499455
499455,Surname1 Surname2,Name (Incomplete NIA)
```

1.1.4 Provided Submission Checker

Students are provided with a Python script (version 3) called "**checker_ssoo_p1.py**" that verifies that the format of the laboratory assignment is correct. The script checks that the assignment follows the naming specifications, is properly compressed, includes everything specified, and **compiles correctly**.

Requirements
The following software must be installed in the environment where the checker will be executed:

- Python 3
- Make
- Unzip

These dependencies are already installed on the Guernika server of the UC3M Computer Science Laboratory (https://www.lab.inf.uc3m.es/servicios/aulas-virtuales-del-laboratorio/). Otherwise, you will need to run the following command on your computer if you are working with a virtual machine, WSL, or Docker:

```
Shell
sudo apt-get update && apt-get install -y make unzip
```

How to use

It is assumed that the command "python3" exists. If you are using an alias, replace "python3" with "python" in the following command to run the checker:

```
Shell
python3 checker_ssoo_p1.py <deliverable.zip>
```

Where **deliverable.zip** is the file to be submitted via Aula Global (see the Submission section).

For example:

```
Shell
python3 checker_ssoo_p1.py ssoo_p1_100254896_100047014.zip
```

The checker prints messages on the screen showing if the format is correct or not.

NOTE

The use of this checker is mandatory before submitting your work, as it verifies that the programs have been compiled correctly. During grading, any submission that does not compile will receive a direct score of 0 points.

1.2 Consideration on the Use of AI

Although the use of AI tools is permitted, it is strongly recommended that you avoid using them for purely educational reasons. It should be noted that, although the use of such tools will not be checked, there are two reasons why it is best not to rely on them in this course:

1. Learning to program in C/C++ and other languages is done through programming. If you delegate that task, it will be difficult to write a program later without help.
2. This course includes a midterm exam and a final exam, which historically require students to write programs in C on paper. If you do not have sufficient proficiency, you will not be able to complete these tests correctly.

2 Submission

The practices will be carried out in teams of **maximum 3 members** belonging to the **same small group**.

2.1 Submission Deadline

The deadline for submitting the assignment to AULA GLOBAL is **March 6, 2026, at 11:55 p.m**.

2.2 Compilation and Testing

This laboratory can be carried out using any of the alternatives identified in the course: native Linux, virtual machine, WSL, or Docker containers. **However, for the final submission, compilation, and testing must be performed on the Guernika server in the UC3M Computer Science Laboratory** (https://www.lab.inf.uc3m.es/servicios/aulas-virtuales-del-laboratorio/). The website includes a section for opening an account to access the server.

Once you have an account, testing and final compilation must be carried out in that environment, as this is where corrections will be made. Any assignment that has not been compiled and tested in Guernika has the risk of containing errors that may prevent teachers from evaluating it correctly.

2.3 Submission Procedure

Assignments must be submitted by **a single member of the group**. Links will be provided on AULA GLOBAL through which assignments can be submitted. Specifically, **one link will be provided for the assignment code, and another TURNITIN link for the assignment report**.

2.4 Files to Submit

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

ssoo_p1_NIA1_NIA2_NIA3.zip

With the NIAs of the group members. If the assignment is done individually, the format will be **ssoo_p1_NIA.zip. The zip file will be submitted to the deliverer corresponding to the assignment code.**

The file must contain:

- **Makefile**
- **mydu.c**

- **mycalc.c**
- **authors.txt: Text file (.txt) containing the authors of the laboratory (one per line). Format (without accents) as indicated above.**

NOTE

To compress these files and ensure they are processed correctly by the provided tester, we recommend using the following command:

```
zip ssoo_p1_NIA1_NIA2_NIA3.zip Makefile mycalc.c mydu.c authors.txt
```

The report shall be submitted in PDF format in a file named:

ssoo_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 source code for the exercise in this section. Any code will be automatically ignored.
- **Battery of tests** used and results obtained. Higher scores will be given to advanced tests, extreme cases, and, in general, to those tests that guarantee the correct functioning of the laboratory in all cases. Keep in mind:
  1. The fact that a program compiles correctly and without warnings is no guarantee that it will work correctly.
  2. Avoid duplicate tests that evaluate the same program flows. The score for this section is not measured based on the number of tests, but on the degree of coverage of the tests. It is better to have a few tests that evaluate different cases than many tests that always evaluate the same case.
- **Conclusions** and problems encountered, how they were solved, and personal opinions.

The following aspects related to the presentation of the laboratory will also be scored:

- It must contain a cover page with the authors of the laboratory and their NIA numbers.
- It must contain a table of contents.
- The report must have page numbers on all pages (except the cover page).

The PDF file will be submitted via the submission link corresponding to the laboratory report (TURNITIN submission system).

NOTE: You may submit the laboratory code as many times as you wish within the submission period, with the last submission being the final version.

Important: If AI tools are used and the solutions of several groups coincide, it will be considered plagiarism.

THE LABORATORY REPORT MAY ONLY BE SUBMITTED ONCE THROUGH TURNITIN.

3 Rules

1. Practical assignments that **do not compile** or do not comply with the functionality and requirements set out will receive a grade of 0.

2. **Programs that use library functions (fopen, fread, fwrite, etc.) or similar, instead of system calls, will receive a grade of 0. The use of statements or functions such as goto is also not permitted.**

3. Special attention will be paid to detecting copied functionality between two assignments. If common implementations are found in two assignments, the students involved (copiers and those copied) will lose the grades obtained through continuous assessment.

4. Programs must compile without **warnings**.

5. Programs must run on a Linux system; assignments for Windows systems are not permitted. In addition, to ensure that the assignment works correctly, its compilation and execution must be checked on the Guernika server (https://www.lab.inf.uc3m.es/servicios/aulas-virtuales-del-laboratorio/). If the code submitted does not compile or does not work on this platform, the implementation will not be considered correct.

6. A program without comments will receive a **very low grade.**

7. The assignment will be submitted through Aula Global, as detailed in the Submission section of this document. Submission by email without prior authorization is not permitted.

8. The input and output format indicated in each program to be implemented must be respected at all times.

9. An error handling must be performed on each of the programs, **more thoroughly than explicitly requested in each section.**

Submitted programs that do not follow these guidelines will not be considered approved.

4 Appendix (System Calls)

System calls provide the interface between the operating system and a running program. UNIX allows system calls to be made directly from a program written in a high-level language, particularly C, in which case the calls resemble function calls, as if they were defined in a standard library. The general format of a system call is:

status = **standard_function** (arg1, arg2,.....)

If a call is unsuccessful, it would return the value -1 in the status variable. The error number is placed in the global variable errno, which we can use to find the association between the error and what actually happened in the errno.h file (located in the path: /usr/src. In Linux: /usr/src/linux/include/asm/errno.h).

4.1 File-related System Calls

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

Opens or creates a file specified by path. The opened file can be used for reading, writing, or both, depending on what is specified by flag. Returns a file descriptor that can be used for reading or writing to the file.
More help at: man 2 open

*int **close**(int fildes)*

Close a file previously opened associated with the descriptor file.
If n = -1 ⟶ Error at close file.
More help at: man 2 close

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

Attempts to read a file (whose file descriptor was obtained by opening it) as many bytes as indicated by *nbyte,* placing the information read from the memory address buffer. Returns the number of bytes read (which may be less than or equal to *nbyte*).
If returns = 0 ⟶ End of file (EOF).
If returns = -1 ⟶ Error at read.
More help at: man 2 read

ssize_t ***write****(int fildes, const void * buf, size t nbyte)*

Attempts to write as many bytes as specified by *nbyte* to a file (whose file descriptor files were obtained by opening it), taking them from the memory address specified in the buffer. Returns the number of bytes that were actually written (which may be less than or equal to *nbyte*).

If returns = -1 ⟶ Error at write.
Each *write* or *read* automatically updates the current position of the file, which is used to determine the position in the file for the next *read* or *write.*
More help at: man 2 write.

off_t ***lseek****(int fildes, off t offset, int whence)*

Modifies the value of the file descriptor pointer in the file to the explicit position at *offset* from the reference imposed in *whence*, so that *write* or *read* calls can be initiated anywhere in the file

- If returns = -1 ⟶ Error positioning.

whence parameter could take the following values:

- SEEK_SET ⟶ from the beginning of the file.
- SEEK_CUR ⟶ from the current position.
- SEEK_END ⟶ from the end of the file.

The *offset* parameter is expressed in bytes and takes a positive or negative value. For example:

```
Shell
abcdefghi
```

```
C/C++
// File descriptor “fd” of the file opened is 5.
lseek(5,4,SEEK_SET) // Moving forward 4 bytes, the next reading would be “e”.
```

More help at: man 2 lseek

4.2 Manual (man function)

man is the system manual pager, i.e., it allows you to search for information about a program, utility, or function. See the following example:

man [section] open

A manual page has several parts. These are labelled as NAME, SYNOPSIS, DESCRIPTION, OPTIONS, FILES, etc. The SYNOPSIS label lists the libraries (identified by the #include directive) that must be included in the user's C program to use the corresponding functions. **To exit from the man manual page, just press 'q'.** The common usage to use man are the following:

- **man section element**: Shows the page of the element available in the manual section.
- **man -a element**: Presents, sequentially, all pages of the element available in the manual. Between pages, the user can choose whether to go to the next page or exit.
- **man -k keyword**: Searches the keyword between the short descriptions and pages of the manual and presents it for every match.

5 Bibliography

- El lenguaje de programación C: diseño e implementación de programas. Félix García, Jesús Carretero, Javier Fernández y Alejandro Calderón. Prentice-Hall, 2002.
- N. Matthew and R. Stones. Programación Linux. Anaya Multimedia. 2008. ISBN: 978-8441524422
- Lenguaje C. https://en.cppreference.com/w/c
- POSIX. https://pubs.opengroup.org/onlinepubs/9799919799/functions/contents.html
- Sistemas Operativos: Una visión aplicada. Jesús Carretero, Félix García, Pedro de Miguel y Fernando Pérez. McGraw-Hill, 2001.
- Unix man pages (man function)