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 2022/2023

Índice

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 means of 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 when a file is opened, unless the `O_APPEND` option is specified. The current position (`current_offset`) of an open file can be changed explicitly using the `lseek` system call.

To manipulate directories, the system calls `opendir, readdir and closedir` can be used. An open directory is identified with a directory descriptor, which is a pointer to a type DIR (*DIR**). When we open a directory with `opendir`, the kernel returns a directory descriptor, on which the entries of that directory can be read through calls to the `readdir` function. The `readdir` call returns a directory entry in a pointer to a dirent structure (*struct dirent*). This structure will contain the fields corresponding to that entry such as the entry name, or the type (if it is a normal file, if it is another directory, symbolic links, etc.). Successive calls to the `readdir` function will return successive entries of an open directory.

1.1 Lab Description

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

1.1.1 mywc

The first program, **mywc**, will open a file specified as an argument, **count the number of lines, words, and bytes of it**, and show these through the standard output (the console) using the appropriate system calls. To do this:

- It will open the file passed as a parameter.
- It will read the contents of the file **byte by byte**.

- It will update the counters based on the bytes read. It is understood that **two lines are separated by the character '\n'**, while **two words can be separated by the characters ' ' (blank space) or '\t'**. In addition, there will be no more than one successive space or line break.
- It will show the results on the console, followed by the file name. Separating each value from the next with a blank space.
- Finally, it will close the file.

```
$ ./mywc p1_tests/f1.txt
 2 15 85 p1_tests/f1.txt
```

- **Usage:** `./mywc <input file>`
- **Requirements:**
  - The program must show the **number of lines, words, and bytes on the console, followed by the name of the read file**.
  - The program will show the data in the following format:
    `<lines><space><words><space><bytes><space><file_name>`
  - The program must return -1 if no input argument has been passed.
  - The program must return -1 if there was an error opening the file (e.g. the file does not exist).
  - The program must return 0 if everything worked correctly.
- **Suggestion of test:**[1] Check that the output of the program on a file matches the output of the ***wc*** command (without arguments) on that same file. To do this, it is recommended to compare the two outputs with the ***diff*** command.

1.1.2 myenv

The second program, **myenv**, will open a file "env.txt" that **contains the environment variables** stored. Then, it will search in the file all the lines that have an occurrence of the variable that is received as an argument and will write them in the output file, passed by arguments, **printing one entry per line**. To do this:

- It will open the file "env.txt" provided (./env.txt).
- It will open the output file passed by arguments.
- It will read the lines one by one and search for the entry with the variable indicated by arguments.
- It will save these lines in the order of appearance in the output file.

[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, make the appropriate code, comment it, test extreme cases, and generally meet the other conditions described in the lab statement.

- Finally, it will close both files.

```
$ ./myenv PATH ./getenv.out
PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games
```

- **Usage:** `./myenv <env> <out_file>`
- **Requirements:**
  - The program must write **all the entries** with the variable in the order in which they are in the input file (`env.txt`) **located in the same directory as `myenv`**
  - The program must separate each entry by **a new line** (\n).
  - The program must create the out_file file empty if it does not find any line with the variable.
  - The program must return -1 if no input argument has been passed.
  - The program must return -1 if there was an error opening the file "env.txt" (e.g. the file does not exist).
  - The program must return 0 if everything worked correctly.

1.1.3 myls

The third program, ***myls***, will open a directory specified as an argument (or the current directory if no directory is specified as an argument), and will display on the screen the name of all the entries of that directory, **printing one entry per line**. To do this:

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

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

- **Usage 1:** `./myls <directory to list>`
- **Usage 2:** `./myls`
- **Requirements:**
  - The program must list **all the entries** of the directory, in the order in which the call to *readdir* returns them, and display each entry on a line.
  - The program must list the entries of the directory passed as a parameter (usage 1) or the current directory if they have not passed any parameter (usage 2).
  - **myls** must show the current directory (.) and the parent directory (..).
  - The program must return **-1** if there was an error opening the directory (e.g. the directory does not exist).
- **Suggestion of test:**[2] Check that the output of the program on a directory matches that of the **ls -f -l** command on that same directory: **ls -f -l <dir to list>**. To do this, it is recommended to compare the two outputs with the ***diff*** command.

1.1.4 Support Code

The **p1_syscall_2023.zip** file is made available to support the development of this lab. To extract its content, run the following:

```
unzip p1_syscall_2023.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.
- **mywc.c**
  Must be modified. C source file where students must code the **mywc** program.

[2]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.

- **myenv.c**
  Must be modified. C source file where students must code the **myenv** program.
- **myls.c**
  Must be modified. C source file where students must code the **myls** program.
- **authors.txt**
  Must be modified. `txt` file where to include the authors of the practice.
- **checker_os_p1.py**
  DO NOT modify. Corrector provided for practice.
- **p1_tests/**
  This directory contains example files and directories to execute and test the 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 the screen a tentative grade obtained with the provided code. The tester must be executed in the Linux computers of the Virtual Classrooms of the university.

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

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

```
$ python3 checker_os_p1.py os_p1_100254896_100047014.zip
```

The corrector 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 lab in AULA GLOBAL will be **March 12th, 2023 at 23:55h**

2.2 Submission Procedure

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

2.3 Files to Deliver

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

os_p1_AAAAAAAAA_BBBBBBBBB_CCCCCCCC.zip

Where A...A, B...B and C...C are the NIAs of the members of the group. In case of doing the practice alone, the format will be **os_p1_AAAAAAAAA.zip**. **The zip file will be delivered in the corresponding deliverer for the practice code**. The file must contain:

- **Makefile**
- **mywc.c**
- **myenv.c**
- **myls.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 corrector, it is recommended to use the following command:

```
zip os_p1_AAA_BBB_CCC.zip Makefile mywc.c myenv.c myls.c authors.txt
```

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

os_p1_AAAAAAAAA_BBBBBBBBB_CCCCCCC.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. In this respect, there are three clarifications to take into account:
  1. That a program compiles correctly and without warnings is not a guarantee that it will work correctly.
  2. Avoid duplicated tests that target the same code paths with equivalent input parameters.
  3. 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. It is better to have a few tests that evaluate different cases 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 text of the report must be justified.

The pdf file will be delivered in the corresponding deliverer for the practice report (TURNITIN deliverer).

NOTE: It is possible to deliver the practice code as often as you want within the delivery period, with the last delivery being considered the definitive version. **THE PRACTICE REPORT CAN ONLY BE DELIVERED ONCE THROUGH 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. 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 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.

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 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 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`)