Computer Architecture and Technology Area

Carlos III University of Madrid

OPERATING SYSTEMS

Lab 3. Operating System calls

BACHELOR'S DEGREE IN COMPUTER SCIENCE AND ENGINEERING

Index

1. Lab Statement

This lab allows the student to familiarize with Operating System calls (especially, file system) following the POSIX standard. UNIX allows you to make calls directly to the Operating System from a program implemented in a high level language, which for his lab will be C.

The majority of input/output (I/O) operations in UNIX can be done using uniquely five calls: `open, read, write, lseek y 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 which 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 opened 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 ***filter***, ***dircount*** and ***mysize***. For this purpose you will have the following files with initial code *filter.c*, *dircount.c* and *mysize.c* and a compilation file *Makefile*.

filter

You have to implement a program named “filter”. This program will receive from the user two parameters which will correspond to (1) the type of filter that is going to be applied and (2) the file that is going to be processed. Then, the program action will be to read the specified file and processed. In this assignment, the filter will replace each lowercase letter by the equivalent uppercase letter. After the filter processing the final result will be displayed on the screen (standard output). More specifically, the program has to complete the following tasks:

- Check the input arguments. The use of strcmp() function is recommended.
- Open the file provided as input argument.
- Read the file
- Character replacement if the filtering is activated. The use of function toupper() is recommended.
- Write the resulting processed file on the standard output (file descriptor 1).
- Close the file

```
$./filter –nofilter testfile.txt
Hola Mundo
$./filter –M testfile.txt
HOLA MUNDO
```

Use: **`./filter`** *`<filter> <input file>`*

Filter:

- **"-M"** converts lowercase into uppercase letters.
- **"-nofilter"** does not convert lowercase into uppercase letters.
-

Requirements :

- The program has to show all the file contents.
- If the number of input arguments is incorrect, the program returns -1 and shows no information on screen.
- If the filter type is incorrect, the program returns -1 and shows no information on screen.
- If the filter type is correct, the program has to apply the corresponding filter.
- It must check the result of all the system calls (open, read and close files). In case of error the program will end with "-1".

dircount

The second program ***dircount*** opens a directory passed as parameter (or the current directory if no directory name is provided) and prints on the screen all the entries that this directory contains.

This program will:

- Open the directory passed as parameter or using the system call getcwd() to obtain the name of the current directory.
- Then, it will read in turn the entries of the directory.
- For each entry, it will print in the standard output the name of the entry, one in each line. It will increase a local counter of file or directory, according the entry type.
- It will display on screen the values of the file and directory counters according the following format:
  Dirs:<blank character><number of directories>
  Files:<blank character><number of files>
- Finally, it will close the directory descriptor.

```
./dircount dir

.

..

dir1

file1

file2


Dirs: 3

Files: 2
```

Case use 1: `./dircount` *<directory>*
Case use 2: `./dircount`

Requirements:

- The program must be capable of exploring any directory and show its entries on the screen.
- The program has to display the provided directory name (Case use 1) or the current one (case use 2).
- The program must show in turn the name of each entry. One per line.
- The program must return -1 if an error occurred while opening the directory or the number of input arguments is incorrect.
- The program output has to fulfil the output format.

mysize

The third program obtains the current directory and lists al the regular files as well as their related size.

This program will:

- Get the current directory by means of getcwd()
- Open the current directory by means of opendir()
- Read each directory entry (by means of readdir)
  - If the entry is a regular file
    - Open the file by means of *open*()
    - Move to the end of file (by means of *lseek*) and get its size.
    - Close the file
    - Print the file name and its related size according the following format:

Name:<blank character><file name><blank character>size:<blank character><file size>

- Close the directory by means of closedir()

Example

```
$./mysize
name: mysize.c size: 2065
name: mysize size: 9032
name: Makefile size: 175
```

Use: ./mysize

Requirements:

- The program has to show the name and size of the regular files, following the specified format in the same order than the one provided by *readdir*.
- The program only shows the regular files.
- The program will return -1 in case of error (for example, unable to open the directory) and no output will be displayed.

1.2. *Initial code*

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

```
tar zxvf systemcalls.tgz
```

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

Makefile
File used by the make tool to compile all programs. **Do not modify this file.** Use *$ make* to compile the programs and *$ make clean* to remove the compiled files.

filter.c
Source file to code ***filter***.

dircount.c
Source file to code ***dircount***.

mysize.c
Source file to code ***mysize***.

2. Assignment submission

2.1. *Deadline*

3 weeks are recommended to make this project.

2.2. *Files to be submitted*

Submission the code in a zip compressed file with name ssoo_p3_AAAAAAAA_BBBBBBBB.zip where A…A and B…B are the student identification numbers of the group. We recommend a maximum of 2 persons per group, but it can be individual. The file to be submitted must contain:

- **Makefile**
- **filter.c**
- **discount.c**
- **mysize.c**

The report must be submitted in a PDF file. A minimum report must contain:

- **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.

- Avoid duplicated tests that target the same code paths with equivalent input parameters.
- 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.
- Compiling without warnings does not guarantee that the program fulfils the requirements.

- **Conclusions**, describing the main problems found and how they have been solved. Additionally, you can include any personal conclusions from the realization 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 an index.
- Every page except the title page must be numbered.
- Text must be justified.

Do not neglect the quality of the report as it is a significant part of the grade of each assignment.

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,.....);
```

I/O system calls

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

The file name specified by *path* is opened for reading and/or writing, as specified by the argument *oflag*; 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`

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`

Manual (man command).

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] name
```

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*.**

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`)