Operating Systems

Grado en Informática 2024/2025

Lab Assignment 1: File systems

We continue to code the shell we started in the first lab assignment. We'll add the following commands. Check the supplied shell to check the workings and sintax for the commands. You can use the help command or "command -?" to get help):

makefile	creates a file
makedir	creates a directory
listfile	gives information on files or directories
cwd	prints current working directory
listdir	lists directories contents
reclist	lists directories recursively (subdirectories after)
revlist	lists directories recursively (subdirectories before)
erase	deletes files and/or empty directories
delrec	deletes files and/pr non empty directories recursively

IMPORTANT:

- Information on the system calls and library functions needed to code these programs is available through man: (open, opendir, readdir, lstat, unlink, mkdir, rmdir, realpath, readlink . . .) (**should you choose to use ftw or nftw, be assured you'll be questioned about how does it work!!!**)
- A reference shell is provided (for various platforms) for students to check how the shell should perform. Ths program should be checked to find out the syntax of the various commands
- An additional C file (ayudaP1.c) is provided with some useful functions
- To check what type of filesystem object a name is, one of the *stat* system calls must be used. **DO NOT USE THE FIELD d_type IN THE DIRECTORY ENTRY**
- The program should compile cleanly (produce no warnings even when compiling with gcc -Wall)
- **NO RUNTIME ERROR WILL BE ALLOWED** (segmentation, bus error . . .). Programs with runtime errors will yield no score.
- These programs can have no memory leaks (you can use valgrind to check)
- When the program cannot perform its task (for whatever reason, for example, lack of privileges) it should inform the user (See errors section)
- All input and output is done through the standard input and output

ERRORS:

When a system call cannot perform (for whatever reason) the task it was asked to do, it returns a special value (usually -1), and sets an external integer variable (errno) to an error code. The man page of the system call explains the reason why the system call produced such an error code.

A generic message explaning the error can be obtained with any of these methods:

- the **perror()** function prints that message to the standard error (the screen, if the standard error has not been redirected)
- the **strerror()** function returns a string with the error description if we supply it with the error code
- the external array of pointers, `extern char * sys_errlist[]`, contains the error descriptions indexed by error number so that `sys_errlist[errno]` has a description of the error associated with errno

WORK SUBMISSION

- Work must be done in pairs.
- **campusvirtual.udc.gal** will be used to submit the source code: a zipfile containing ONLY a directory named **P1 (capitals)** where all the source files of the lab assignment reside. The name of the zip file should be **p1.zip (lowercase)**
- The name of the main program will be p1.c (lowercase), Program must be able to be compiled with **gcc p1.c**, Optionally a Makefile can be supplied so that all of the source code can be compiled with just make. Should that be the case, the compiled program should be called p1 and placed on the same P1 directory (no build subdirs or similar)
- **ONLY ONE OF THE MEMBERS OF THE GROUP** will submit the source code. The names and logins of all the members of the group should appear in the source code of the main programs (at the top of the file)
- Works submited not conforming to these rules will be disregarded.
- DEADLINE: 23:00, Friday October the 25th, 2024