Lab 2.2. File System

Objectives

In this lab, we will review the basic system functions to manage a file system: file and directory creation, file descriptor duplication, file status and information, and file locking.

Contents

Environment Preparation

This lab only requires the development environment (compiler, editor and debugger). These tools are available in the ASOR VMs as well as in the physical machine of the lab.

Any editor can be used (`vi`, `nano`, `xedit`…). Also, it is also possible to use C or C++ language (`gcc` or `g++` compiler, respectively). If several files have to be compiled, the use of a tool like `make` is recommended. Finally, the recommended debugger is `gdb`. The use of IDEs like Eclipse is **not allowed**.

File Creation and Attributes

An inode stores different attributes of a file, like owner, access permissions, size or access, modification and creation times. In this section, we will see the most important system calls and commands to get and set these attributes.

Exercise 1. The main tool to check the content and basic attributes of a file is `ls`. Consult the `man` page and study options `-a -l -d -h -i -R -1 -F` and `--color`. Analyze the meaning of the output on each case.

Exercise 2. File permissions are `<type><rwx_owner><rwx_group><rwx_other>`:

- `type` can be `-` (regular file), `d` (directory), `l` (link), `c` (character device), `b` (block device), `p` (FIFO/pipe) or `s` (socket).
- `rwx` means read (4), write (2) and execution/access (1), respectively.

Check the permissions of some directories (with `ls -ld`).

Exercise 3. Permissions can be set using octal or symbolic notation. For example, try the following commands (which are equivalent):

- `chmod 540 file`
- `chmod u+rx,g+r-wx,o-wxr file`

How can we set permissions `rw-r--r-x` in both ways?

Exercise 4. Create a directory and remove execution/access permission for owner, group and other. Try to change to that directory.

Exercise 5. Write a program that, using the `open` call, creates a file with permissions `rw-r--r-x`. Check the result and file characteristics with the `ls` command.

Ejercicio 6. Default permissions when a file is created are derived from the user mask (*umask*). The `umask` command can be used to get and set this mask. Using this command, set the mask in a way that new files don't have write permissions for the group, and none for others. Check the behaviour with commands `touch` and `ls`.

Ejercicio 7. Modify exercise 5 in order to modify the user mask as in previous exercise before the file is created. Once the file is created, the original mask must be restored. Check the result with the `ls` command.

Ejercicio 8. The `ls` command shows the inode number with option `-i`. Other information from the inode can be obtained using the `stat` command. Consult the options of the command and check its behaviour.

Ejercicio 9. Write a program that emulates the behaviour of the `stat` command, showing:

- *major* and *minor* numbers of a device
- inode number of the file
- file type (directory, link or regular file)
- the last time the file was accessed. What is the difference between `st_mtime` and `st_ctime`?

Exercise 10. Links are created with command `ln`:

- Create a symbolic link (with option `-s`) to a regular file and to a directory. Check the result with `ls -l` and `ls -i`. Determine the inode of each file.
- Repeat the exercise with hard links. Determine the inode of each file and its properties with command `stat` (in particular, check the number of links).
- What happens if one of the hard links is removed? What happens if one of the symbolic links is removed? What if the original file is removed?

Exercise 11. `link(2)` and `symlink(2)` calls create hard and symbolic links, respectively. Write a program that receives a file path as an argument. If the path is a regular file, it creates a symbolic link and a hard link (with the same name, ending with `.sym` y `.hard`). Check the result with the `ls` command.

Redirections and File Descriptor Duplication

The shell provides operators (>, >&, >>) to redirect files (see the proposes exercises in the optional lab). This functionality is implemented using the `dup(2)` and `dup2(2)` system calls.

Exercise 1. Write a program that redirects the standard output to a file specified as a first argument. Test it by writing several lines to the standard output.

Exercise 2. Modify the previous program so that it also writes in the file the standard error output. Test the program including several sentences writing to both streams. Is there any difference if the redirections are done in different order? Why is "`ls > dirlist 2>&1`" not the same as "`ls 2>&1 > dirlist`"?

Exercise 3 **(Optional).** With `fcntl(2)` it is also possible to duplicate file descriptors. Study which options should be used to duplicate a descriptor.

File Locking

The file system provides an advisory file locking system. These functionality can be accessed through `flock(2)` or `fcntl(2)` (and even `lockf(3)`). In this section, we will use only `fcntl(2)`.

Exercise 1. The status and use of file locks in the system can be checked in `/proc/locks`. Analyze the contents of this file.

Exercise 2. Write a program that gets and shows the status of a lock on a file. The program will show the status of the lock (locked or unlocked), and:

- If it is unlocked, the program will set a write lock and will write the current time. After that, it will suspend for 30 seconds (using `sleep`) and, then, it will unlock the file lock.
- If it is locked, the program will exit.

The program should not modify the file if it doesn't own the lock.

Exercise 3 **(Optional).** Command `flock` provides access to locking functionality. Consult the `man` page and learn how this command works.

Project: Extended `ls` Command

Write a program meeting the following specifications:

- The program has a unique argument, which is the path to a directory. The program must check that the argument is correct.
- The program will walk through the directory entries, and:
    - If the entry is a non-executable regular file, it will write its name.
    - If the entry is an executable file, it will write the name followed by `'*'`.
    - If the entry is a directory, it will write its name followed by `'/'`.
    - If the entry is a symbolic link, it will write the name followed by `"->"` and the name of the file it points to. Use the `readlink(2)` function, and appropriately set the size of the buffer.
- At the end of the list, the program will write the total size of the files (not directories) in Kilobytes.