# Lab 3: UNIX File System

Operating Systems
Spring 2003

### Lab Assistant

Alexandre David
www.docs.uu.se/~adavid/courses/os/osvt2003.html
email: adavid@it.uu.se
room: 1237
postbox: 26 (4th floor, building 1)
phone: 018 - 471 7341

**Questions** are numbered **Qx** and **programs Px**. Programs are to be handed with their description, see section 6. When you answer, give the section number and the question number, and use **sentences** to answer. Answers of the type "yes" "3" "no" will be rejected. Answers have to be given on the special answer sheet included in the lab package. You should have unpacked the "lab package" now (from lab1), it contains a directory `lab3` for this lab. **Go inside this directory first.**

## 1 Introduction

In this assignment you will study the UNIX file system. In particular, you will learn how the file system is structured, which data types are used to represent the different objects, and which functions and system calls you can use to work with the file system in your code.

## 2 Getting Started

Read the manual pages for `df(1)`, `ls(1)` and `ln(1)`.

### 2.1 `df`

The `df` command displays the amount of disk space occupied by mounted or unmounted file systems, directories, or mounted resources, the amount of used and available space, and how much of the file system's total capacity has been used.

The root file system `/` is always available on a machine while other parts can be integrated (mounted) into the file system.

By giving `df` a directory or file as argument you can determine, on which machine the argument *actually* is located.

**Q1:** Try to determine on what machine and under what directories the following directories are mounted:

- `/stud`
- `/stud/docs`
- Your home directory.
- `/usr/hacks`
- `/usr/hacks/share`

You may see `automount` which means that the directory is mounted only when needed so try `ls` on that directory and the automounter will mount it. You should be able to continue after that.

## 2.2 `ls`

`ls` has many options for displaying information about files and directories in various formats. Use `ls`, with options where appropriate, to determine the following:

- `ls -l` reports a total at the top.
  **Q2:** what does the number indicate?
  Try `ls` with some different flags before you decide on an answer. Hint: it is *not* the number of files and the answer *is* available in the manual page.
- In `filesys/test/alpha/`, located in the course directory `/stud/docs/kurs/os`, there are several identical files.
  **Q3:** which of them are in fact separate and unique (and which are not)? Explain your answer.
- Create an empty directory in your work directory. Type `ls -l` and look at the entry.
  **Q4:** why is there a "2" after the permissions, what is the meaning of the number 2 in this context. (i.e. what are the two items?)
  Hint: it is *not* the number of subdirectories.
- Copy a file to the new directory, and see what happens to the "2".
  **Q5:** Has it changed? Why or why not?
- Create a directory in the new directory (i.e. you should now have a directory containing a file and an empty directory). Check the number again.
  **Q6:** explain what has happened.

## 2.3 `ln`

Use `ln` to create links and investigate the behaviour of the links in different situations:

- Change to the new directory and use `ln -s` to create a *symbolic link* to the file.
  **Q7:** does the original file show any indication of the symbolic link?
  Do the same for the directory and see if there is any indication. Try to make a symbolic link to a file that does not exist, or make a symbolic link and then remove the file.
  **Q8:** what happens?
- Type `ln -s gurka gurka`.
  **Q9:** explain what happens when you try to read the file `gurka` you just created.
- Now make a *hard link* to a file (i.e. `ln` without `-s`).
  **Q10:** what happens now?

- **Q11:** What happens when you change the permissions of the original file with `chmod`, or update its modification time with `touch`?
- **Q12:** Apart from the names, how can you tell which was the original file, and which is the link you just created?
- **Q13:** What happens if you remove the original?
- **Q14:** What happens if you try to make a hard link to a directory? To a non-existent file?

After all this, you should understant the difference between hard and symbolic links.

## 3 Inodes

In this part of the assignment you will learn about the `stat(2)` system call and the `stat` structure that it returns. The `stat` structure contains a number of fields with information from a file's inode. Read the manual page for `stat(2)`. Observe that there are several `stat` functions, use the right one in the right place. Ie. make sure that you report information about the file that *holds* a symbolic link and not the file (if any) that the symbolic link points to.

**P1:** write a program that, given the name of a file or directory on the command line, reports the following information from the corresponding inode:

- mode (permissions)
- number of links
- owner's name
- group name
- size in bytes
- size in blocks
- last modification time
- name

In addition, the type of the object should be indicated as follows: if the file is a symbolic link, the file "pointed to" by the link should appear following the name, such as with `ls -l`, see `readlink(2)`. Note that `readlink()` does not terminate the returned string properly, you will need to do this yourself, (make sure that your link printouts look similiar to `ls -l`). As well, a character should be added to the end of the filename to indicate if it is a directory or an executable file; one of `'/'` or `'*'` as described for `ls -F`.

If you write a function that, when given a filename, reports the information as described, then you can easily reuse it for the next part of the assignment. The information should preferably be presented entirely on one line and in strict columns (such as `ls -l` does).

You will need `getpwuid(3C)` and `getgrgid(3C)` in order to convert numerical ids to strings, (use `ls -l`, `ls -n` and `ls -nl` to see the difference), as well as `localtime(3C)` and `strftime(3C)` to convert time structures into a readable format.

It is not necessary to print the mode symbolically like `ls` does (e.g. `-rwxr-xr-x`). But, if you choose to print it numerically, use octal, *not* decimal, since the octal representation is closer to the symbolic one (why?). (For instance, the symbolic mode given above would be 0755 octal). See the manual page for `printf(3S)` on how to print an octal number. In addition, you only need to print the lower 9 bits of the mode (i.e. use C:s bitwise AND operator like: `mode & 0777`). You can find proper bit masks in `/usr/include/sys/stat.h`

To examine if a certain bit in the mode field is set see the manual pages for `stat(5)` and `mknod(2)`. The constants defined in `stat(5)` are used like: `mode & FLAG` and evaluates to a non-zero value if the test is true. The macro function `S_ISLNK(mode)` is for some reason not mentioned in the `stat(5)` manual pages. It tests for a symbolic link and evaluates to a non-zero value if the test is true and the right `stat` function has been used.

For information about using command line arguments in your program, see the document on giving arguments to programs in C, that is given in the course WWW home page.

**Use a function, which given a file name, prints information on it.** This is to help you for the next exercise.

# 4 Traversing directories

In this section you will learn how to read directories and traverse the file system.

**P2:** the assignment is to write a program that traverses the file system from a starting point provided on the command line, similar to `ls -lRa`.

Begin (as always) by reading some of the appropriate manual pages. The major functions you will need are `opendir(3C)`, `readdir(3C)`, `closedir(3C)`, `chdir(2)`, `getcwd(3C)` and `rewinddir(3C)`.

The `struct dirent` mentioned on the manual page for `readdir(3C)` is documented in the `dirent(4)` manual page. The `dirent` structure contains a number of fields with information from an entry in a directory.

In `filesys/test` there are a number of subdirectories and files of different types. Change to that directory and run `ls -lagFR` to get an idea of the kind of output you should expect from your program. The test run you hand in for the assignment should include at least the results of running your program in this directory.

A natural structure for the program is a function that traverses the list of files in a *single* directory. When it reaches a directory in the list, it can call itself recursively. To start, just invoke the function with the name of the starting directory.

You will need to deal with the possibility that you may not have the proper permissions to search or enter certain directories. There are also some other error-like situations that need to be handled properly. In none of these cases should the program need to exit, although it may need to take some special action. See what `-ls` does for example. Notice that there is a hidden file that your program should to find without crashing. It is called "you_get_it".

**Pay attention to null pointers and possible buffer overflow!!!!!!!!!!** The assignment is not a C-programming assignment and is focused on OS. However it is very important to pay attention to these extremely sensitive issues in programming.

# 5 Important

- *Always* check return values. Most system calls return a value to indicate if they were successful, and if not, the reason for failure. They do fail sometimes! In these cases you should use `perror(3C)` to print a message describing the failure.
- `perror(3C)` is a special message function that understands the error values returned by most system calls. Use it! But note also that `perror(3C)` is not a general purpose function, it will only indicate the status of system calls and *some* C library functions. In other cases, use `fprintf(stderr,...)` to report errors.
- See in the headers of the different manual pages which header files you should include. Compile with: `make` and change the `Makefile` to add targets for the different programs if you want to.

## 6 Lab Report

A proper report is to be handed in for the lab. It should include at least the following:

- The cover page provided at the end of this description.
- Answers to the given questions.
- A description of your implementation. Explain in abstract terms (not program syntax) how it works and why you chose your particular solution to the problem. Use diagrams if you like.
- Pointers to readable source files, executable files, and the shell commands used to start the programs.
- Listings of well-commented and well-structured programs and informative listings of test runs. The test run shall be on the directories in `/stud/docs/kurs/os/filesys/test`. Use the command `a2ps(1)` to get nice printouts.

The assignment must be handed before June 1st, 2003.

## 7 FAQ

**How to print?** `lp -d`*printer_name file_name* sends a file to the printer and you get a simple printing. A better way to print C files is to use `a2ps -P`*printer_name file_name* which is nicer.

**What is the printer's name?** The printer's name is `pr`*number* where *number* is the room number where you are sitting.

**How to save execution trace?** If you have a program that just prints out information and does not use any input from the user while running (like `ls`) then you can use *your_program your_options* `> trace.log` and you will get your trace in the file `trace.log`. If you want to take a sample and still see the printing, use the mouse: select with the left button, and copy in emacs with the middle button.

**How to get rid of warnings?** Use proper includes in your C program. Files are given at the beginning of man pages. Check bad constructions and arguments of formated strings as used by printf.

**Why can I have a warning and get the program compiled and linked?** If the compiler does not find a function declaration then it takes a default declaration, which is a function which returns an integer. If you are lucky, this may match your function.

**Why to compile with -Wall?** -Wall means "warning all" and this gives you warnings on suspect constructions like `if (a=3) { ... }` or `printf("n=%d");` which are programming errors though valid with respect to the C syntax. It is useful to get rid of warnings!

# Lab 3: UNIX File System

Operating Systems
Spring 2003

## Lab Assistant

Alexandre David
www.docs.uu.se/~adavid/courses/os/osvt2003.html
email: adavid@it.uu.se
room: 1237
postbox: 26 (4th floor, building 1)
phone: 018 - 471 7341

I/We have solved these labs independently, and each of us has actively participated in the development of all of the lab solutions.

| Lab solver 1 | Lab solver 2 |
|---|---|
| ............................................<br>Signature | ............................................<br>Signature |
| ............................................<br>Name | ............................................<br>Name |
| ............................................<br>Personal number | ............................................<br>Personal number |
| ............................................<br>Username | ............................................<br>Username |

..........................................................................................
Path to source files