CS 35L: Software Construction Lab

Lab 6

Hengda Shi

Week 1 Lecture 2

Announcements

- Check “Assignment Paraphrases and Hints” under week 1 on CCLE for tips
- Lab and Homework answers should be submitted together on CCLE
- Please use lnxsrv06, lnxsrv07, lnxsrv09, or lnxsrv10
- Final time and location is the same for all sections
- Homework Exercise 1.2.2: change to “Delete the 19th line”

Environment

- Environment is defined by environment variables in Unix-like OS
- Example of setting environment variable
  - $ TEST=”cs35l” # set variable TEST with string “cs35l”, note that no spaces around =
  - $ echo $TEST # print out TEST variable
- Other preset environment variables include $HOME, $PATH
- $HOME stores the home directory of the current user
- $PATH stores the path that shell will search for when a command is invoked
- For example, when `$ ls` is invoked, shell will search through paths in variable $PATH one by one until a executable named `ls` is found

Environment (cont’d)

- When we login into system, shell will undergoes a phase called initialization to set up the environment. It will search for `/etc/profile` and `~/.profile` and set environment variables based on file contents.
- After setting all variables, the shell prompt `$` will be displayed
- export command is used to update environment variables for the current shell session
- `$ export PATH=“/usr/local/cs/bin:$PATH”` will append the path to the PATH variable so that the shell program can search for executable in the path “/usr/local/cs/bin” before any other paths
- Put above command to `~/.profile` will append the directory to path permanently as long as `~/.profile` exists

More on I/O redirection

- Three standard file descriptors
  - stdin (standard input), stdout (standard output), stderr (standard error)
  - stdin is 0, stdout is 1, stderr is 2
  - `$ program < file` # read input from file to program (stdin)
    - Note: $ echo < file1 will not work because echo command does not read stdin
  - `$ program > file` or `$ program 1> file` # write output from program to file (stdout)
  - `$ program >> file` or `$ program 1>> file` # append output from program to file (stdout)
  - `$ program 2> file` # write error from program to file (stderr)
  - `$ program 2>> file` # append error from program to file (stderr)
  - `$ program &> file` # write both output and error from program to file (stdout and stderr)
  - `$ command 2> /dev/null` will discard error messages outputted to the shell because everything going to `/dev/null` will be discarded
  - Manual page can be found using `$ man stdin`

More on Linux Wildcard

- Linux wildcard can be used without double quotation for commands like ls, rm
- Double quotation must be added if linux wildcard is used in find command
  - For example, `$ find . -name f*` will fail because it has no double quotation around f*
  - `$ find . -name “f*”` is correct in this case

More on file permission (special file modes)

- setuid - executes with owner permission
  - e.g. -rwsr-xr-x (`$ ls -l /usr/bin/passwd`)
  - S for non-executable file, s for executable file
  - Example: passwd command is owned by root user, this program is set with setuid so that any user can change their own password
  - `$ chmod u+s program`
- setgid - executes with group permission
  - -rw-r-sr-x
  - `$ chmod g+s program`
- sticky bit - only owner or root can delete or rename file
  - e.g. drwxrwxrwt (`$ ls -ld /tmp`)
  - T for non-executable file, t for executable file
  - `$ chmod +t program`

Link revisited

- Every file will have some metadata such as file permission, owner, group, last modified date. Those information are stored in a data structure called **inode**.

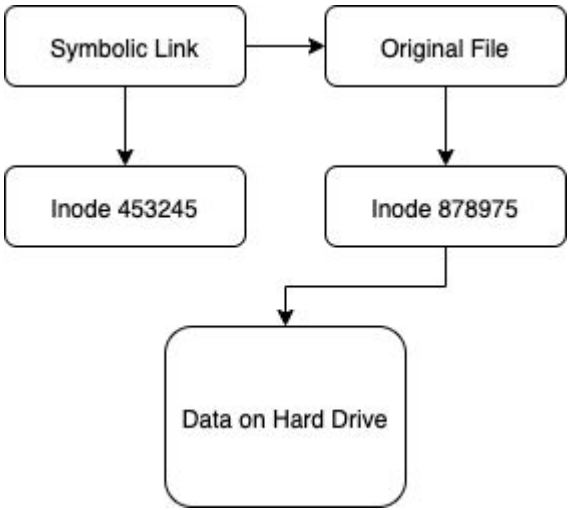
- All inodes are indexed to a inode table. A lightweight version is stored in memory, and the full version is stored on disk

Link revisited (cont’d)

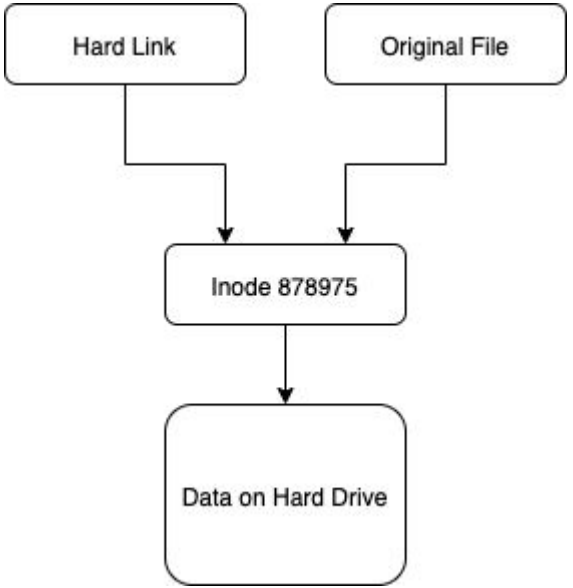
- However, file names are not stored in the inode, they are stored in the directory. We talked about the concept of “Everything is a file”. In the directory, it contains the file names and their corresponding inodes
- Effectively, a file is actually stored in three different locations - directory, inode, and data block which holds the actual data of the file
- To view inode information of a file, `$ stat filename`
- To find inode number of a file, `$ ls -i filename` or `$ stat -c %i filename`

Symbolic (Soft) Link vs. Hard Link

Soft Link



Hard Link



Symbolic (Soft) Link vs. Hard Link (cont'd)

- As we can see in previous slides, both links will not create an additional copy of the data, then why do we have two different types of links?

Soft Link

- can cross the file system
- allows you to link between directories
- has different inode number and file permissions than original file
- permissions will not be updated
- has only the path of the original file, not the contents

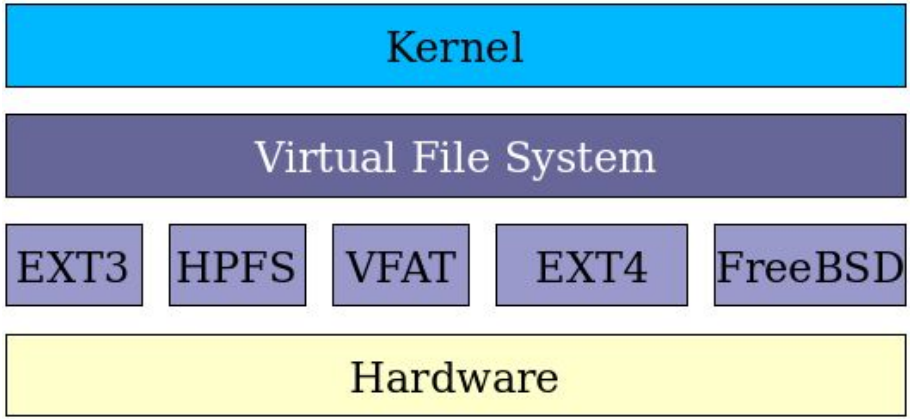
Hard Link

- can't cross the file system boundaries (i.e. A hardlink can only work on the same file system)
- can't link directories
- has the same inode number and permissions of original file
- permissions will be updated if we change the permissions of source file
- has the actual contents of original file, so that you still can view the contents, even if the original file moved or removed

ref: http://bit.ly/2N97j81

File System

- We mentioned that hard links cannot across file system boundaries, but what does that actually mean?
- Linux supports almost 100 types of file systems, and each of them has different policy on metadata location, data access policy, etc.
- To check all filesystems mounted, use `$ df -Th`



ref: https://red.ht/37QEO6Z

Directory Structure

- Directory structure is a mapping from file representations to inodes
- The mapping can be many-to-one, which means that multiple file representations can be mapped to the same inode
- Linux maintains a unified directory structure compared to Windows which assign a drive letter to different partitions, like C:, D:
- In Linux, devices or different partitions mounted will still be under the root directory

Text editors

vim and Emacs

vi editor

- Open a file - vi <filename> or vim <filename>
- Close a file - :q
- Save a file - :w
- Save and close a file - :wq
- Enter edit mode: i
- Exit edit mode: esc

Emacs editor (will be using Emacs for this course)

- Almost like a Windows text editor, but much more powerful
- Run emacs on the linux server
- C-h r (manual) and C-h t (tutorial)
- All emacs commands start with ‘C’ or ‘M’
- ‘C’ = ctrl; ‘M’ = alt (Windows)/ option (Mac)
- start emacs: `$ emacs <filename>`
- Exit emacs: `C-x C-c`

Basic emacs editing

- Insert text by simply typing it
- Undo by typing C-x u
- Save changes by typing C-x C-s
- Copy, cut, paste
  - C-space (starts selecting region)
  - M-w (copy a region)
  - C-w (cuts a region)
  - C-k (kill a line)
  - C-y (yank/paste)

Moving around

Keystrokes	Action
C-p	Up one line
C-n	Down one line
C-f	Forward one character
C-b	Backward one character
C-a	Beginning of line
C-e	End of line
C-v	Down one page
M-v	Up one page
M-f	Forward one word
M-b	Backward one word
M-<	Beginning of buffer
M->	End of buffer
C-g	Quit current operation

More emacs commands

- Search – C-s
- Replace – M-%
- Accessing menu – F10
- Switch buffer – C-x b
- Switch current window – C-x o
- Kill the current window – C-x 0 (zero)
- View typed commands (view keystrokes) - C-h l
- Search for commands - C-h a
- Help – C-h

Directory edit (dired) (C-x d)

- Creates an Emacs buffer containing list of directory contents
- Allows you to operate on files
- Allows you to navigate file system
- + - new directory, C-x C-f new file in directory, g - refresh dired buffer
- ! - run shell command
- https://www.gnu.org/software/emacs/refcards/pdf/dired-ref.pdf

EMACS tutorial links

- http://bit.ly/2CQy3H8 (some basic commands)
- http://stanford.io/2CTWNyl

Task 1

- Create a cpp file (print “hello”) using emacs
- Run the file within emacs
- Edit the file (print “hello again”) using emacs and save it as a new file
- Run the new file within emacs
- Hint: use C-h a to search how to compile and run shell command in emacs

cpp program

```
#include<iostream>
using namespace std;
int main() {
        cout << “Hello” << endl;
        return 0;
}
```

Compiling instruction:

```
$ g++ -o filename filename.cpp
$ ./filename
```

Task 2

- Create 2 .txt files (insert some lines) using emacs
- Find the difference between both of them using a linux command