

CSE-381: Systems 2

Exercise #10

Max Points: 20

Objective: The objective of this exercise is to explore the use of:

1. Review relative and absolute paths
2. Observing operations of soft and hard links
3. Understand internals of a file system

First, you should save/rename the lab procedure document using the naming convention **MUId_Exercise10.docx** (example: raodm_Exercise10.docx) before starting to work on this exercise. You may discuss any questions you may have with your instructor.

Name: Ce Zhang

Part #1: Understanding and working with file path [5 points]

Estimated time to complete: 15 minutes



Without a good understanding of paths, you will eventually be lost.

- Yours truly

Background

A *path* is the general form of the name of a file or directory. A path specifies a unique location in a file system. A path points to a file system location by following the directory tree hierarchy expressed in a string of characters in which path components, separated by a delimiting character, represent each directory. The delimiting character is most commonly the slash ("/"), the backslash character ("\"), or colon (":"). Paths are used extensively in computer science to represent the directory/file relationships common in modern operating systems, and are essential in the construction of Uniform Resource Locators (URLs). Paths used in operating systems are broadly classified into the following two categories:

- **Relative paths:** Relative paths refer to a file based on the present working directory from where a program is executed. Relative paths have one of the following forms:
 - Filename.txt
 - Dir1/Filename.txt
 - ./Dir1/Filename.txt (where . is the pwd)
 - ../../Dir1/Dir2/Filename.txt (where .. is parent directory)
- **Absolute paths:** A full path or absolute path is a path that points to the same location on one file system regardless of the present working directory or combined paths. It is usually written in reference to a root directory. Absolute paths on Linux always start with the / (root) path:

- o /home/raodm/Filename.txt
- o /usr/bin
- o /proc/cpuinfo

Complete the following table:

The first entry has been completed to illustrate an example.

Absolute path	Relative path	
	From your home directory	From /usr/share/
/usr/bin/emacs	../../usr/bin/emacs	../bin/emacs
~/	./	../../home/zhangc29
/bin/ls	../../bin/ls	../../bin/ls
/proc/cpuinfo	../../proc/cpuinfo	../../proc/cpuinfo

Part #2: Working with links [5 points]

Estimated time to complete: 15 minutes

Background

Most modern operating systems encourage the use of *links* or references to files to facilitate organization and streamlining applications without increasing storage requirements. Operating systems support a variety of links, with the following two links being the most common ones:

- **Hard link:** A hard link is a file system entry that associates a name with a file on a file system. A directory is itself a special kind of file that be hard linked. The term hard link is used in conjunction with file systems that allow multiple hard links to be created for the same file.
- **Symbolic link:** In computing, a symbolic link (aka *symlink* or soft link) is a special type of file that contains a reference to another file or directory in the form of an absolute or relative path and that affects pathname resolution.

Note that symbolic links are different from hard links. Hard links do not link paths on different volumes or file systems, whereas symbolic links may point to any file or directory irrespective of the volumes on which the link and target reside. Hard links always refer to an existing file, whereas symbolic links may contain an arbitrary path that does not point to anything.

Creating and manipulating links

Observe and record the operation of hard links and symlinks on a standard Linux file system using the following procedure from a bash shell (aka Terminal):

1. **Create temporary work directory:** Create a temporary work directory (say, named temp) and change your PWD to it. Record the disk usage of the directory using the command (do not forget the dot at the end):

```
$ du -sh .
```

2. Remember to use this command to measure disk usage in other parts of the experiment):

Initial disk usage:

35m

3. **Create a dummy data file:** For this experiment we will use a dummy data file for testing. Create a dummy file using the following command:

```
$ dd if=/dev/zero of=zero.dat count=20
```

Next use the `ls -lh` command to view the file created. Record the disk usage (it should be a couple KB over the size of the file due to block size of 4KB):

Disk usage without links:

10240 bytes

4. **Look into inode details:** You may look into inode details of a given file using the following `stat` command and record output below:

```
$ stat zero.dat
```

```
zhangc29@os1:~$ stat zero.dat
  File: zero.dat
  Size: 10240          Blocks: 24          IO Block: 4096   regular file
Device: fd00h/64768d Inode: 1447413       Links: 1
Access: (0644/-rw-r--r--)  Uid: (1592179/zhangc29)   Gid: ( 101/  uidd)
Access: 2019-11-04 13:00:41.692145707 -0500
Modify: 2019-11-04 13:00:41.692145707 -0500
Change: 2019-11-04 13:00:41.692145707 -0500
 Birth: -
```

From the above output from `stat` command, answer the following questions – **Don't forget unit of bytes** :

What is the "IO Block" size:

4096

How many "Blocks" are occupied:

24

Note: This is 512 byte blocks!

How much disk space is used by the blocks
(i.e., 512 * Blocks):

24*512= 12288 bytes

What is the actual file "Size":

10240bytes

How much disk space is unused or
wasted:

12288-10240=2048bytes

5. **Create a hard link to the file:** Create a hard link to the file using the command:

```
$ ln zero.dat zero_link1.dat
```

Actual disk usage with one hard link
(du -sh .):

35M

Note that the disk usage did not change significantly. Using the `ls -lh` command estimate how much disk space should have been used by the files:

Estimated disk usage with one hard link:

10k

6. **Look at inodes:** Use the `ls -li` command to list the files with their inode numbers. For links you should notice the same inode numbers. Record the output of `ls -li` command below:

```
zhangc29@os1:~$ ls -li
total 2800
1447413 -rw-r--r--  2 zhangc29 uidd   10240 Nov  4 13:00 zero.dat
1447413 -rw-r--r--  2 zhangc29 uidd   10240 Nov  4 13:00 zero_link1.dat
```

7. **Create another hard link to the file:** Just to ensure you understand links, create another hard link to the file using the command `ln zero.dat zero_link2.dat`

Disk usage with two hard links
(du -sh .):

35

Note that the disk usage did not change significantly. Using the `ls -lh` command estimate how much disk space should have been used by the files? In the space below record your inference as to why the disk space did not change:

Links are just inode entries. When the directory was created 4K, which is created for the initial set for the inodes. The space will not increase if we create the link. However, if many are created, then the space is increasing and which is good for bigger usage.

8. **Create a soft link to a file:** Create a symlink to the file using the command:

```
$ ln -s zero_link2.dat zero_symlink.dat
```

Disk usage with one soft link (`du -sh .`): **35m**

Using the `ls -lh` command view the difference between how hard link and symbolic links are displayed. Record the output below for your own future reference:

```
total 2812
1447413 -rw-r--r--  3 zhangc29 uuidd 10240 Nov  4 13:00 zero.dat
1447413 -rw-r--r--  3 zhangc29 uuidd 10240 Nov  4 13:00 zero_link1.dat
1447413 -rw-r--r--  3 zhangc29 uuidd 10240 Nov  4 13:00 zero_link2.dat
1448457 lrwxrwxrwx  1 zhangc29 uuidd   14 Nov  4 13:20 zero_symlink.dat ->
zero_link2.dat
```

9. **Observe inode behavior:** Using the `ls -li` command from earlier step, record the inode information for the files in the space below:

```
1447413    zero.dat
1447413    zero_link1.dat
1448457    zero_symlink.dat
```

Briefly describe why the inode number for the symlink is different?

The hard links is for the same issues on disk and same set of blocks. But symlink usea different file entry and needs a different inode to store information about the symlink

10. **Break the symbolic link:** Delete the file `zero_link2.dat` orphaning the symbolic link created earlier. Notice how the symbolic link exists but is broken/hanging without referring to an actual file. In addition, note that the disk usage does not change. Check the operation of the links using the following commands:

```
$ wc -c zero_symlink.dat
$ wc -c zero_link1.dat
```

11. **Restore the symbolic link:** Recreate the link to the file `zero_link2.dat` Notice how the symbolic link once again starts operating correctly.

12. **Remove a hard link:** Delete the file `zero.dat` file. Using the `ls -lh` command note that the hard linked file `zero_link1.dat` continues to exist but its link count has changed to 2 (from 3). However, note that although the original file has been deleted, the `zero_link1.dat` continues to contain the data in it via the following command:

```
$ wc -c zero_link1.dat
```

13. **Clean up:** You may clean up the temporary directory you have created.

Part #3: Review file permissions for privacy and security [5 points]

Estimated time to complete: 10 minutes

Background: In Linux, each directory & file has 3-sets read-write-execute (`rwX`) permissions (total of 9 characters) associated with it for:

- The first 3 are for `user` -- *i.e.*, the person who created/owns the file
- The second 3 are for the `group` associated with a file.
- The last 3 are for `others` -- *i.e.*, users who are not the owner and not part of the group

These permissions can be modified to balance privacy (*i.e.*, access to information) versus security (limiting type of access to a select subset of users) using the `chmod` command as shown below:

```
$ chmod u+rwX,g+rx-w,o+r-wX info.txt
```

Where, owner gets `rwX`, group (users) gets `rx` but not `w` (due to minus) and other (users) have only `r`.

Exercise: Using the `chmod` command, setup suitable file permissions to meet the policy scenario described below:

Three users on a Linux machine are setup such that Alice and Bob are in the same Linux group but not Eve. Alice has created three files, that need to be shared with the other two users with the following permissions:

File	Alice	Bob	Eve
one.txt	Read & write only	No access	Read only
two.txt	Read only	Read & execute only	No access
three.txt	Read & execute only	Read & write	Execute only

Show the series of Linux commands that Alice must execute at the shell prompt to setup the above permissions for the various file(s). Do make any assumptions about existing set of permissions for the files.

```

$ chmod u+rw-x,g-rwx,o+r-wx one.txt
$ chmod u+r-wx,g+rx-w,o-rwx two.txt
$ chmod u+rx-w,g+rw-x,o-rw+x three.txtAlternative syntax:
$ chmod 604 one.txt
$ chmod 450 two.txt
$ chmod 561 three.txt

```

Part #4: Understanding internals of a File System [5 points]*Estimated time to complete: 15 minutes***Background**

Most file systems use a list of blocks to store the data associated with a file. The list of blocks is stored on the file system in a special location. In this part of the exercise, we will use an example of a file system called File Allocation Table (FAT) to understand how blocks operate.

FAT file system internal details:

Use the following FAT information when answering the questions further below

i. BPB**Sector/block Size = 16 bytes****ii. Root Directory Entries:**

<i>File Name</i>	<i>File Size (bytes)</i>	<i>Starting Block #</i>
FILE1.TXT	39	2
FILE2.TXT	75	6
FILE3.TXT	23	8

iii. FAT block Entries (chain of block numbers):*Logical block #*

0x000	0xFFF	0x000	0x004	0x000	0x00B
0x005	0x003	0x007	0x008	0x00C	0x000
0x00A	0x000	0xFFF	0x00D	0xFFF	0x000

Note the following interpretation applies to above block # values:

<i>Entry Value</i>	<i>Interpretation</i>
0x000	Free Block
0x002 to 0xFEFF	Used and points to next block in FAT chain.
0xFF8 to 0xFFFF	Last block for a file (end of chain)

iv. Data content for each Block:

<i>Cluster #</i>	<i>Data</i>
2	This is a simple
3	data that is on.
4	text file that i
5	the data is all,
6	Hello, this is a

7	Simple manual th
8	at is used to sh
9	the tv was on so
a	i changed it too
b	S silly example.
c	ow case working□
d	of fat12 system.
e	and show it is a

1. For each one of the files below, indicate the chain of data clusters to be read to obtain the complete data. The first entry is already completed to illustrate an example.

<i>File Name</i>	<i>Block chain to read</i>
FILE3.TXT	0x8 (starting entry), 0xc (Entry in FAT corresponding to block 0x8). The entry for 0xc is 0dx , and its entry 0xff which signifies end of chain and that block is not included. So the sequence of blocks to be read is: 0x8, 0xc, 0xd
FILE1.TXT	0x2, 0x4, 0xb
FILE2.TXT	0x6, 0x7, 0x8, 0xc, 0xd

2. For each one of the files below, write the actual content that will be read from the file system. The first entry is already completed to illustrate an example.

<i>File Name</i>	<i>Actual data read (don't miss out spaces)</i>
FILE3.TXT	Copy-paste of contents in each block for the file identified earlier: at is used to show case Note: Remaining bytes in block 0xc are ignored because file size is only 23 bytes!
FILE1.TXT	This is a simplettext file that iS silly (note that remaining bytes are not read because file size is only 39 bytes)
FILE2.TXT	Hello, this is aSimple manual that is used to show case working of fat12 sy (note that remaining bytes are not read because file size is only 75 bytes)

Part #4: Submit files to Canvas

Upload the following files:

1. This document saved as a PDF with the convention *MUId_Exercise10.pdf*.