UNIX Tutorial Sheet #1

Topics Covered:

1. FILE STRUCTURE
2. ACCESS PERMISSIONS FOR FILES/DIRECTORIES
3. UNIX COMMANDS FOR FILES/DIRECTORIES
4. OPEN SYSTEM CALL
5. CREAT SYSTEM CALL
6. CLOSE SYSTEM CALL
7. SOLVED EXERCISE

File System is the abstraction used by Kernel to represent and organize the storage resources. A unix file system organize the files in a tree stricture of any arbitrary depth. Files in UNIX is a broad term which applies not just to data files but the term files can also refer to devices , sockets , pipes etc. Root of the tree is named by '/' character. Tree Structure looks like

```
                        /
                        |
    ---------------------------------------------
     /       |       |        |        \
   Etc     bin     usr      tmp       dev
            |                          |
       -------------              --------------
        /        \                  /          \
      ls …… csh                 ucb …..... lib
```

In many system user files are subdirectories also like under root director bin is the subdirectory. There are two paths of any file i.e. Relative path or absolute path. Relative path is path of file/directory from its parent director only like bin/ush is the relative path of file csh and root/bin/csh is the absolute path of file csh i.e path from root directory to the file/directory.

There are four types of files in UNIX File System:

1. **Ordinary File:** An ordinary file may contain text, a program, or other data. It can be either an ASCII file, with each of its bytes being in the numerical range 0 to 127, i.e. in the 7-bit range, or a binary file, whose bytes can be of all possible values 0 to 255, in the 8-bit range.

2. **Directory Files**: Suppose that in the directory x I have a, b and c, and that b is a directory, containing files u and v. Then b can be viewed not only as a directory, containing further files, but also as a file itself. The file b consists of information about the directory b; i.e. the file b has information stating that the directory b has files u and v, how large they are, when they were last modified, etc.
3. **Device Files:** In Unix, physical devices like printers, terminals etc. are represented as "files." The same read() and write() functions used to read and write real files can also be used to read from and write to these devices.
4. **Link Files:** Suppose we have a file X, and type

   ```
   ln X Y
   ```

   If we then run ls, it will appear that a new file, Y, has been created, as a copy of X, as if we had typed

   ```
   cp X Y
   ```

   However, the difference is the cp does create a new file, while ln merely gives an alternate name to an old file. If we make Y using ln, then Y is merely a new name for the same physical file X.

UNIX Commands

UNIX Command to obtain information about all the files in a given directory:

```
ls  -al
```

where ls is to list all the files in the present working directory, 'a' (all) and 'l' (long) are the options which give more details with files.

Example:

```
 ls -al
drwxr-xr-x    6   ecs4005   1024    Apr 22    13:30   ./
drwxr-xr-x    74  root      1536    Mar 24    12:51   ../
-rw-------    1   ecs4005   188     Apr 13    15:53   .login
-rw-------    1   ecs4005     6     Mar 24    11:29   .logout
-rw-------    1   ecs4005   253     Apr 10    12:50   .xinitrc
-rw-r--r--    1   ecs4005   516     Apr 10    13:00   .twmrc
-rw-r--r--    1   ecs4005   1600    Apr 22    10:59   test2.out
```

First column gives information about access permissions to file/directory (First character in first column is either 'd' or '-' where 'd' stand for directory and '-' stands for files.)
Second column tells about number of files in a directory in case of directory otherwise it is 1.
Third column is the owner of file of directory.
Fourth column gives the information about size of file/directory in bytes.
Fifth column gives information about date of last modification.
Sixth column is the name of file.

Example:

Type command `ls -lr /etc/i* to see the result.`

Explanation: In this, the "l" and "r" options of the ls command are invoked together. The l option calls for a long output, and the r option causes ls to operate recursively, moving down directory trees. The last part of the example, "/etc/i*", directs the ls command to list files and directories in the /etc directory, that begin with the letter i. The wildcard character, "*", matches any characters.

File Access Controls:

First column, i.e. of 10 characters long, in the above information gives the complete details about the access permission and restrictions on the corresponding file/directory.

Position 1: It can be ‘d’ for directory file, ‘-’ for ordinary file, ‘l’ for link file or symbolic link.

Position 2 to 4: These 3 characters give the information about the permission for the owner of corresponding files. These are ‘rwx’ i.e ‘read write execute’. If read permission is given to the owner then first character i.e character at position 2 will be r otherwise it will be ‘-’. If write permission is given to the owner then second character i.e character at position 3 will be w otherwise it will be ‘-’. And if execute permission is given to the owner then third character i.e character at position 4 will be x otherwise it will be ‘-’.

Position 5 to 7: These are the permissions for other users in the same group. i.e. ‘rwx’ permissions for others.

Position 8 to 10: These are the ‘rwx’ permissions for other users in the different group.

If you want to access any file in a directory then you need to have execute permission on that corresponding directory.

Command to change the access permission on any file:

chmod (i.e. change mode) is the command to change the file permissions.

There are two ways the change the permissions.

1. **Octal number:** Access permission for each type of user is denoted by 3 characters ‘rwx’. If any permission is given to any user then it is 1 otherwise it is 0. Eg. 101 on ‘abc’ file for owner it means read and execute permission are given to owner of ‘abc’ file but owner doesn’t have write permission on the file. 101 in octal number is 5.
   Maximum number anyone can form with 3 bits is 7.
   Example:
   ```
   Chmod 755 abc.exe
   ```
   It means on abc file owner has access permission i.e. 7 means 111 (all read, write, execute). All others have access permission i.e. 5 means 101 (read and execute but not write)

2. **Symbolic:**
   ```
   chmod ugo+rw .login
   ```

This command would add read and write permission for all users to the .login file of the person. Where u user (i.e. owner), g group, o others, + add permission, - remove permission, r read, w write, x execute

Command to create the file:
Files can be created using any text editor like vi, pico editor etc or with unix commands.
Using vi editor : vi filename
this will create a file or file can be edited with command

Unix command to create a file is cat (stands for concatenation). Symbol used with cat command is '>' (redirecting the output to file written after this symbol)

Example 1:

```
cat > unix
```

where unix is the file name which you want to create. The command cat generally reads in a file and displays it to standard output. When there is no filename directly following the command, cat treats standard input as a file. The ">" symbol will redirect the output from cat into the abc file you specify. cat will keep reading and writing each line you type until it encounters an endof-file character. By typing CTRL-D on a line by itself, you generate an end-of-file character. It will stop when it sees this character.

Note: cat command is used for reading a file and appending a file also.
Example 2:

```
cat abc def > final
```

here contents of file abc and def will be redirected to file final. It will overwrite the final file.
Example 3:

```
cat abc >> final
```

where >> is the double redirection symbol. Above command will append the contents of abc file to the end of final file. Contents of final file will not be overwritten.

To create the directory:

```
mkdir study
```

it will create the directory with name study under the present directory.

To Display the contents of a file:
In Unix, contents of file can be displayed by more than one way.

1. **cat command:**

Example:

```
cat abc
```

It will display complete contents of file at a time even though file is very long. You need to use scroll up to see the contents. Otherwise You can control the flow of text by using CTRL-S and CTRL-Q. CTRL-S stops the flow of text and CTRL-Q restarts it.

2. **more command:**

Example:

```
more abc
```

It will display only one screen of information at a time. It waits for you to press spacebar or enter to display more contents. It will display at bottom that how many percentage of file has been displayed till now.

3. **less command:**

Some unix system supports less commands also.

Example:

```
less abc
```

it allows backward and forward movement in the file. It does not read the entire input file before starting. You can use page up and page down button to see the contents.

The head and tail commands

The head command allows you to see the top part of a file. You may specify the number of lines you want, or default to ten lines.

Example: `head -15 /etc/rc`

to see the first fifteen lines of the /etc/rc file.

The tail command works like head, except that it shows the last lines of of file.

Example: `tail /etc/rc`

to see the last ten lines of the file /etc/rc. Because we did not specify the number of lines as an option, the tail command defaulted to ten lines.

To find the directory in which you are at present at to change the directory:

Type command `pwd`

It will give you the name of directory in which you are working at present. pwd is present working directory.

To change your working directory, if you want to move into directory name:

```
cd name  (this is the relative path of directory name)
```

if you want to give absolute path of directory:

```
cd ~/name (complete path using ~)
```

otherwise

```
cd home/details/name (complete path)
```

To copy the file or directory:
cp command copies the file or directory.
Example:

```
cp file1 file2
```

it will copy file1 to file2. If file2 doesn’t exist then it will create new file with name file2 otherwise it will overwrite the contents of file2.

```
cp file1 file2
```

it will copy file and prompt you before overwriting.

```
cp -i /home/dvader/notes/meeting1 .
```

it will copy file meeting1 present in directory /home/dvader/notes to the current directory, where . (period) indicated the current directory as destination.

To move a file from one directory into another directory and to rename a file:

```
mv abc study
```

it will move file abc into directory study.

```
mv abc study/os/chapter1
```

it will move file abc into directory chapter1 which is under study/os

```
mv abc test1
```

it will rename file with name abc with new name test1

NOTE: mv command is used for renaming a file and moving a file into another directory both. First argument will always be file name wither in moving a file or renaming a file, but if second argument in mv command is file name i.e. test1 then mv is renaming a file and if second argument is directory then mv is moving a file into directory.

To remove a file or directory:

```
rm abc
```

it will delete the file named abc.

```
rm –i abc
```

it will delete the file named abc and prompt you before deleting that are you sure you want to delete it.

```
rmdir study
```

it will delete the directory with named study (make sure study is the empty directory)

```
rmdir –i study
```
it will delete the directory named study and prompt you before deleting that are you sure you want to delete it.

Encoding and decoding the files:
```
compress abc
```
it will compress (encode) the file named abc and replace it with abc.z
```
uncompress abc.z
```
it will decode the file abc.z and replace it with abc
```
zcat abc.z
```
it will display the contents of compressed file.

Counting Words, Line in a file:
```
wc abc
```
output of above command is 6   22   93 tmp
it means file abc has 6 lines, 22 words and 93 characters.

```
wc –l abc
```
it will count only lines in file abc.

```
wc –w abc
```
it will count only words in file abc.

```
wc –c abc
```
it will count only characters in file abc.

File System:

A unix file system organize the files in a tree stricture of any arbitrary depth. Files in UNIX is a broad term which applies not just to data files but the term files can also refer to devices , sockets , pipes etc. Root of the tree is named by ‘/’ character. The attributes of a file are such things as type of file—regular file, directory—the size of the file, the owner of the file, permissions for the file—whether other users may access this file—and when the file was last modified. The `stat` and `fstat` functions return a structure of information containing all the attributes of a file (we will discuss in detail later).

File Descriptor:

File descriptors are normally small non-negative integers that the kernel uses to identify the files being accessed by a particular process. Whenever it opens an existing file or creates a new file, the kernel returns a file descriptor that we use when we want to read or write the file. These file descriptor numbers can vary from 0 to some N. The actual value of N is UNIX version dependant but on most versions it is 1024. The first three file descriptors need not be explicitly created (Standard Input, Standard Output, and Standard Error). These constants are defined in the **`<unistd.h>`** header.These three are already open and have pre-defined meanings as follows:

File Descriptor 0 : Standard Input (STDIN_FILENO)
File Descriptor 1 : Standard Output (STDOUT_FILENO)
File Descriptor 2: Standard Error Output (STDERR_FILENO)

By default all of these file descriptors refer to the terminal or console (Recall according to UNIX, a terminal is also a file). The names (like STDIN_FILENO) are more convenient to use than their corresponding numbers (like 0). Usually the standard input and output is the console.

To understand the difference between Standard Output and Standard Error Output, Execute the program below:

```
# include<unistd.h>
# include<fcntl.h>
int main()
{
int i=0;
int fd;

for(i=0;i<1024;i++)
{
        fd=open("input.txt",O_RDWR);
        if(fd==-1)
        {
        write(STDOUT_FILENO,"ERROR",6);
        break;
        }
}
return 0;
}
```

The exact working of the program is not relevant at this point. Compile this program and run it as follows:

```
./a.out
./a.out > out.txt
```

In the first case, the contents are displayed on screen. In the second case, the contents that were to be displayed on the screen are instead pushed into a newly created file called “out.txt”.
Now replace the write () in the program as follows:

write (STDERR_FILENO,"ERROR",6);
After replacement with STDERR_FILENO in both cases of execution the program would always show the output on the screen (console) only.
When we write to the Standard Error Output the output would never get re-directed or lost in any other way (by use of pipes).
As a side note, we conclude there is an error in the above program because we are trying to get file descriptors more than 1024 number of times.

I/O System Calls:

Open function:

#include <fcntl.h>
int open(const char *pathname, int flag, mode_t mode);

This function is defined in header file <fcntl.h>. If file is opened successfully, file descriptor will be returned but is any error occurred then it will return value -1.
open() system call opens an existing file or can be used to create a new file and then open it, if the file does not exist.

Here is this function The pathname is the name of the file to open or create. This function has options, which are specified by the flag argument. These are:
`O_RDONLY` Open for reading only.
`O_WRONLY` Open for writing only.
`O_RDWR` Open for reading and writing.

We can combine the options by ORing together these.

Most implementations define O_RDONLY as 0, O_WRONLY as 1, and O_RDWR as 2. We can use these constants also as argument in function call.

One and atmost one of these three options must be specified.
Some optional options are:

1. O_APPEND Append to the end of file on each write.
2. `O_CREAT` Create the file if it doesn't exist.
3. `O_EXCL` Generate an error if `O_CREAT` is also specified and the file already exists.

4. `O_TRUNC` If the file exists and if it is successfully opened for either write-only or read–write, truncate its length to 0.

There are access permission on every file for owner of file, group of users and for other users (read, write execute), already discussed earlier in this lab sheet.
In Unix its representation is different. General notation is S_I**pwww** where **p** is the permission (R, W, X) and **www** is for whom (USR, GRP or OTH) . The octal value 755 can be symbolically represented as:

S_IRUSR|S_IWUSR|S_IXUSR|S_IRGRP|S_IXGRP|S_IROTH|S_IXOTH

Now the open() call would look like,

```
int fd;
fd=open("input.txt",
O_WRONLY|O_CREAT|O_TRUNC,
S_IRUSR|S_IWUSR|S_IXUSR|S_IRGRP|S_IXGRP|S_IROTH|S_IXOTH);
```

The above call creates a new file with no data in it. If the file exists, the data is deleted completely (O_TRUNC). If the file does not exist, it is created as new (O_CREAT).Since the file is newly created, it's opened in a write-only mode (O_WRONLY). And the permissions are set to be 755 as described above.

The combination of O_WRONLY|O_CREAT|O_TRUNC is, in fact, so common, that a dedicated system call exists just for that combination – the creat() system call.

Creat Function:

A new file can also be created by calling the **creat** function.

#include <fcntl.h>
int creat(const char *pathname, mode_t mode);

If file is created successfully then file descriptor for write only mode will be returned otherwise if any error occurs -1 will be returned.
This function is equivalent to open (pathname, O_WRONLY | O_CREAT | O_TRUNC, mode);

Close Function:

An open file is closed by calling the **close** function.

#include <unistd.h>
int close(int filedes);

Exercise:

Show the kernel datastructures (file descriptor table, file table, inode table) after the following commands has executed.

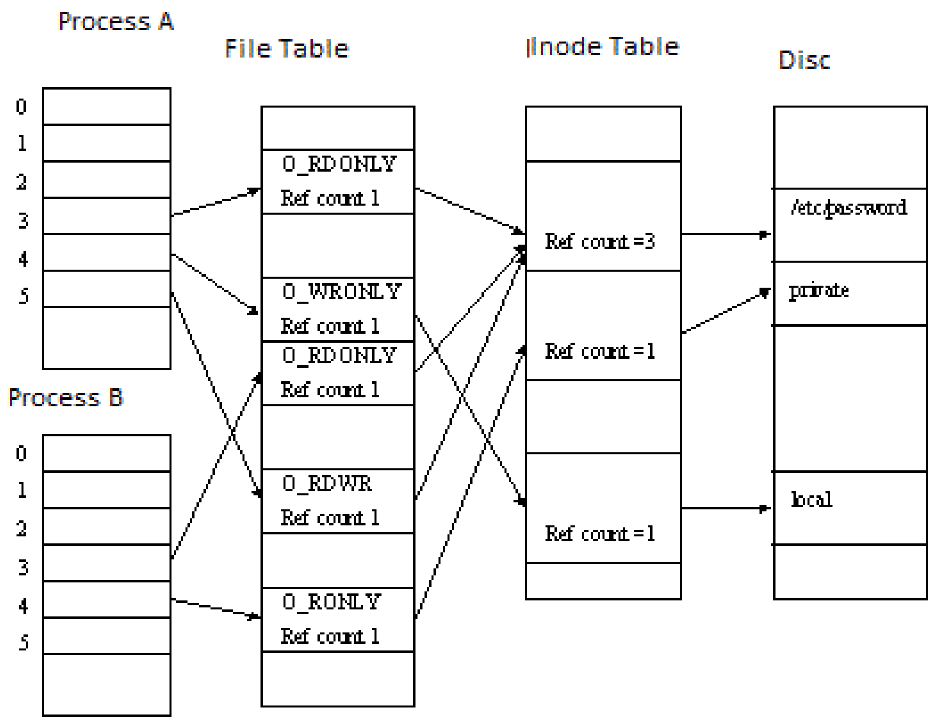
Process A:

```
fd= open("/etc/passwd",O_RDONLY);
fd2= open("local",O_WRONLY);
fd3= open("/etc/passwd",O_RDWR);
```

Process B:

```
fd1= open("/etc/passwd",O_RDONLY);
fd2= open("private",O_RDONLY);
```

Solution:


Process A
File Table
Inode Table
Disc
0
1
2
3
4
5
O_RDONLY
Ref count 1
O_WRONLY
Ref count 1
O_RDONLY
Ref count 1
O_RDWR
Ref count 1
O_RONLY
Ref count 1
Ref count =3
Ref count =1
Ref count =1
/etc/password
private
local
Process B
0
1
2
3
4
5