Linux Programming

* What is an operating system:

. OS is the interface between computer user and computer hardware.

. It is a resource manager.

. It manages all the resources like files, folders etc.

* Linux is the best-known & most used OS.

- Open source
- Highly secure
- Multi user / multi programming → many can access in same time & do many tasks
- Most reliable → GUI & CUI
- Portable → no seperate hardware required, can be installed in any system.
- Worry-free OS

Ex: Redhat, Suse, ubuntu
(Redhat, Suse: enterprise (paid))

* Kernel → primary components of operating system.

Shell → secures the hardware & kernel. It receives the request, checks the symantix & sends to kernel

* Linux was created in 1991 by Linus Torvalds a then student at the University of Helsinki (Unix was earlier name)

developed by Dennis & ken Thomson in UT&T lab 1971

* GPL → General ~~purpose~~ public license

It is given to users to freely use the software for study, share & modify purpose

* Commands:

- date ——→ Today's day, date, time, yr
- cal ——→ current month calender
- cal june 2020 → june month calender
- cal 2020 ——→ whole calender
- cal 06 2020 ——→
- echo Hello all → O/P: Hello all
- echo Today's date is ——→ Today's date is
- echo Today's date is (date) ——→ error
- echo Today's date is $(date) ——→ Today's date is ——————
- pwd ——→ working directory
- ls ——→ list / file
- ls -l ——→ all the details of the file or all the files in directory
- ls -r ——→ " " " " " " " " " " " " " "
  in reverse order
- ls -t ——→ . . . . . according to the time they have been saved

File handling utilities
creating a file
moving " "
deleting " "

Unit I

* File Handling Utilities:

The Linux commands which help us to

create
delete
rename
move
copy
edit

and perform other related activities on Linux files.

* Commands:

- clear ——→ clears the screen
- man ——→ manual for any utility. It will display

all the details of that command.

- ls -l → blue files are directories
  white files are regular files
  green files are special files like obj file etc.
- history → all the commands typed so far along with history command.
- rmdir → remove the directory or many at a time
- rm → " " file / files (if given two or more names)
- cat → we can see the contents of a file (concatenation)
- touch → to create a file
- cat >>file1 → allows to write content in file
  Then to come out from it, press Ctrl+D
- vi → allows to write content in file
  So type 'i' to convert into insert mode, to come out Esc, colon, w, q
- mv → to rename file
  Ex: mv oldname newname
- cp → copy contents from one file to another
  Ex: cp file2 file3
- cat -n file2 → gives no.s to the contents of file2
- mkdir → one or more directories can be created one by one or at a time (no commas)
- cd → current directory
- mkdir -p d1/d2/d3/d4 → in d1 directory, d2 directory is created, in d2 directory d3 is created .....

- cc → used to compile c files
- ./a.out → displays the output for ↓
- chown → can change owner of file
- chgrp → change grp.

} both can be done at a time
Ex: chown root : root f11

* Ownership of Linux files:

Every file & directory on our Linux system is assigned 3 types of owner.

→ User:

A user is the owner of the file. By default, the person who created a file becomes its owner.

→ Group:

A user grp can contain multiple users. All users belonging to a group will have same access permissions to the file.

→ Other:

Any other user who has access to a file. It means everybody else. This person has neither created the file nor a grp user who could own the file.

Ex: person accessing a website.

* Permissions:

- read → authority to read & open a file
- write → modify contents (add, remove, rename files)
- execute → .exe (Windows). In Linux we cannot run a program unless the execute permission is set.

* ls - l

user grp others

-rw-rw-r-- 1 home home 0 2012-08-30 19:06 MyFile

root

file type

- implies we have selected a file

r = read permission

w = write "

x = execute "

- = no permission

* chmod command: (If u don't want ppl to see images of urs)

We can use 'change mode' command to set permissions on a file/directory for the owner, grp & the world.

Syntax: chmod permissions filename.

r - 4
w - 2
x - 3

There are 2 ways to use the command:

- Absolute or Numeric mode → 4 = read, 2 = write, 1 = execute
  → permissions written in octal mode
- Symbolic mode → permissions written in characters

* Absolute mode:

No.	permission	symbol
0	no permission	---
1	execute	--x
2	write	-w-
3	execute + write	-wx
4	read	r--
5	read + execute	r-x
6	" + write	rw-
7	" " + execute	rwx

Ex: chmod 764 filename

- owner can r, w, x
- grp can r, x, w
- world can r

* Symbolic mode:

You can modify permissions of a specific owner.

Operator	Description
+	adds a permission to a file/directory
-	removes the permission
=	sets the permissions & overrides the previous permissions

User	Denotations
U	user/owner
G	group
O	other
a	all users

Ex: chmod u-rwx f11

chmod o-rw+x, u+x f11

* cat f11
cat f1
2
1
3
4
5
clear
sort f1 ⟶ A/D sorting
sort -r f1 ⟶ A/D reverse order
1
2
3
4
5
clear

* uniq f1 ⟶ It ~~sorts~~ removes duplicate values
* sort f1 | uniq ⟶ It sorts & " " "
* head -5 f1 ⟶ only 5 lines will be displayed.
* tail f1 ⟶ lines from bottom displayed
* wc f1 ⟶ filters data & gives details
* wc -l f1 (no. of lines in file)
  wc -c f1 (no. of characters in file)
  wc -w f1 (no. of words in file)
* cat >f1.txt (new lines)
-
-
-
-
Ctrl+D (comes out)

* cat >>f1.txt (append)

* wc f1.txt (character count)

* wc -cw f1.txt (character & word
  -cl
  -wl

* n| f1.txt
1. ____
2. ____
3. ____

* tac f1.txt
(all lines are reversed)

* echo "hi" | tr 'a-z' 'A-Z'
HI

* more f1.txt
  less f1.txt

* Process: foreground / background

A program under execution.

. fg (~~behind the~~ running in front) (The current file we use) ex: chrome

. bg (behind running) (like opened files) ex: powerpt, notepad

. top (ex: clg) (all running processes on Linux)

. ps (ex: cse dept) (checks all the processes running under a user) process status

. kill (terminate a process)

. nice (multiple process happen at a time, but few are imp so they have some priority order)

. who (used to know who is currently logged on to the system)

An instance of a program is called process. Any command given to Linux machine is a process

* ps

ps -f
ps 48
nice +20
nice -20
top
kill 141 (process id)

* who [-option]

* w:
same as who but gives detailed o/p of user activities

Syntax: $w [-options]

options: -H, -W

* Disk Utilities:

. du:
It displays the disk space used by the specified file/directory & it displays disk usage of the current directory.

Syntax: $du [-option]

options:

-a : (all) count of all files, not just for directory

-b: (byte) it prints the size in bytes

. df:
It displays empty disk space

Syntax: $df

* Network Utilities:

- who am i: It displays from which mail you are using the terminal system.

Syntax: whoami

- ifconfig: It displays interface MAC address, IP address etc of system.

Syntax: ifconfig

- wall: It is used to give any message to all users connected to our system.

Syntax: wall msg

- telnet: Used to connect with host system.

Syntax: telnet [hostname / IP address]

- ping: When something is sent from source to destination, we should know if it is sent / connected properly

Syntax: ping

- rlogin: remote login (server is very far from us system but still we can login to remote systems)

Syntax: rlogin

Text processing commands (cont.)

* cat > file1

1. vjit.ac.in 2. ibm.com 3. jntu.ac.in

fmt -w 1 file1 → (horizontal to vertical)

1. vjit.ac.in
2. ibm.com
3. jntu.ac

* sort f5 | lpr ⟶ sends for printing purpose.
* csplit f5 3 ⟶ file will be split into 2 where first file contains 3 lines, remaining in file 2. (xx00, xx01)

* csplit -f abc f5 5 → names will be given to splitted files (abc00, abc01)

* comm f4 f5 → gives common & different things in both files. but both files should be in sorted order.

```
sort f4 > f11
sort f5 > f21
comm f4 f5
```

o/p (Nothing common)

* diff
* cmp
* sed
* awk

* Sed: (used for substitution / find & replace)

- SED command in UNIX stands for stream editor & it can perform lots of functions on file like searching, find & replace, insertion & deletion.
- By using SED you can edit files even without opening it, which is much quicker way to find & replace something in file, than first opening that file in VI editor & then changing it.
- SED command in UNIX supports regular expression which allows it perform complex matching.

Syntax:

```
sed options... [SCRIPT][INPUTFILE...]
```

Ex:

```
cat > geekfile.txt
```

1. Replacing / substituting string: (changing word unix to linux)

```
sed 's/unix/linux/' geekfile.txt
```

By default the sed command replaces the first occurance of the pattern in each line & it won't replace the second, third ... occurence in the line.

2. Replacing the n^{th} occurence of a pattern in a line:

```
sed 's/unix/linux/2' geekfile.txt
```

O/P:

unix is great os. linux is opensource. unix is free os

—

—

—

3. Replacing all the occurence of the pattern in a line:

```
sed 's/unix/linux/g' geekfile.txt
```

(g → global replacement)

O/P: ~~linux~~ linux is grt os. linux is opensource. linux is free os.

—

—

—

4. Replacing from n^{th} occurence to all occurences in a line:

```
sed 's/unix/linux/3g' geekfile.txt
```

(3g → $3^{rd}, 4^{th}, 5^{th}$ .... unix will change to linux)

5. Duplicate the replaced with /p flag:

(/p → print the replaced line twice)

```
sed 's/unix/linux/p' geekfile.txt
```

6. Print only the replaced lines:

```
sed -n 's/unix/linux/p' geekfile.txt
```

7. Replacing string on a range of lines:

1,$ last line

```
sed '1,3 s/unix/linux/' geekfile.txt
```

only these lines will be replaced.

8. Deleting lines from a particular file:

- To delete particular line 'n'

```
sed 'nd' filename.txt
```

- To delete last line

```
sed '$d' filename.txt
```

- To delete line from range x to y

```
sed 'x,yd' filename.txt
```

9. To delete from n^{th} to last line:

```
sed 'nth,$d' filename.txt
```

10. To delete pattern matching line:

```
sed '/pattern/d' filename.txt
```

11. Inserts blank line:

```
sed '/unix/G' stream.txt
```

12. prints required line:

```
sed -n 'np' stream.txt
sed -n '1,3p' stream.txt
sed -n '2,+3p' stream.txt
```

* awk:

Alfred Aho, Peter Weinberger & Brian Kernighan

1977

AT & T Bell laboratories

awk - finds & replaces text, database sort/validate/index.

- Awk command is for processing text.
- It is a scripting language used for manipulating data & generating reports.
- The awk command programming language requires no compiling, and allows the user to use variables, numeric functions, string functions & logical operators.

* awk operations:

a) Scans a file line by line

b) Splits each I/P line into fields

c) Compares I/P file/fields to pattern

d) Performs action(s) on matched lines.

* syntax: awk options 'selection_criteria{action}' input-file > output-fi

Ex: write a file employee.txt ⟶ name designation salary

awk '{print}' employee.txt

awk '/manager/{print}' employee.txt (lines having manager word)

awk '{print $1, $4}' employee.txt (1st line to 4th line)

o/p of this command is I/P for awk

ls -l | awk '{print $2}' (file name isn't mentioned)

awk '{print NR"-"$1}' employee.txt (gives no.s to lines & then displays that no. line)

awk '{print $1, $(NF-1)}' sample.txt (when we don't no no. of lines)

↓
last line

awk 'NF > 0' sample.txt (Don't display blank lines)

awk 'END{print NR}' sample.txt

2. Shell Programming with BASH (Bourn Again Shell)

- Shell is a user program or an environment provided for user interaction.
- It is a command interpreter which reads from standard I/P / file & executes them

User Requests

↓

Shell

↓

Kernel

↓

Hardware

- Kinds of Shells:

Bourne Shell
C "
Korn "
Bash "
Tcsh "

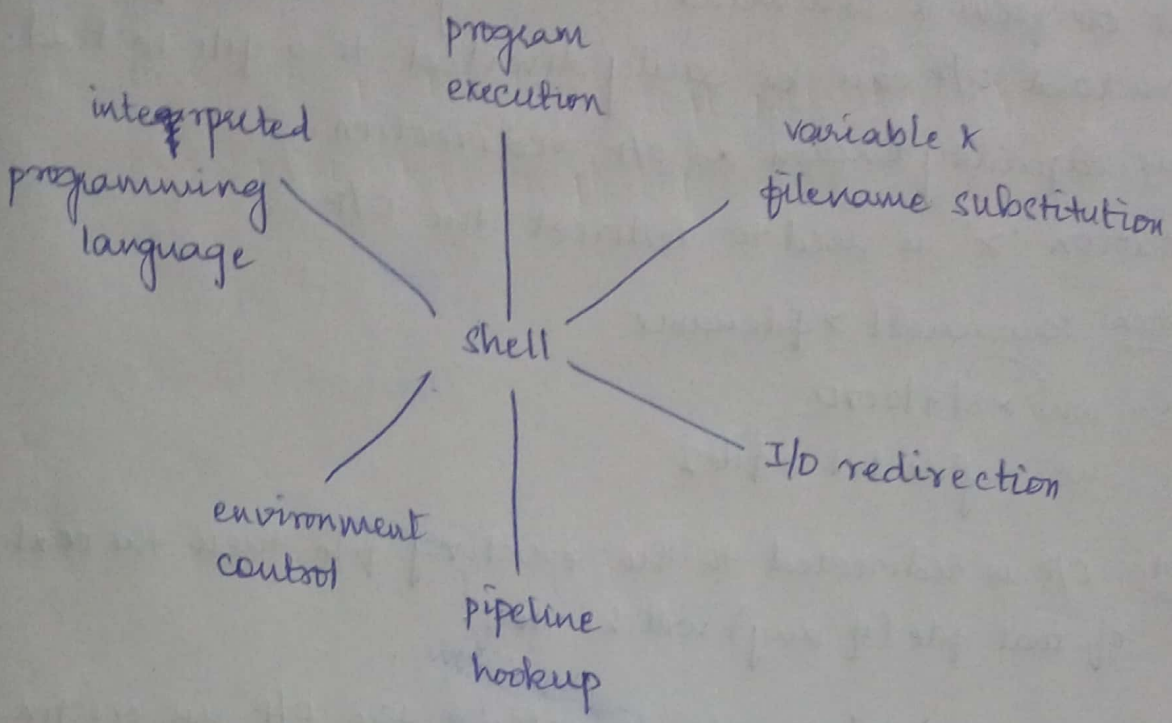
- $SHELL (or) $0
  echo $SHELL (or) echo $0

O/P: /bin/bash

- cat /etc/shells

O/P: /bin/sh
/bin/bash
/usr/bin/sh
:

Shell responsibilities:



* Pipe & Redirection:

- A pipe is a form of redirection that is used in Linux, transfer of standard o/p to some other destination.
- Use to send the o/p of one command/program to another command/program/process for further processing.
- We can do so by using the pipe character '|'.
- Pipe is used to combine two or more commands, and o/p of one command acts as I/p to another command & this command's o/p may act as I/p to the next command & so on.
- It can also be visualize as a temporary connection b/w two or more commands/programs.

Syntax:

Command1 | command2 ------

Ex: ls -l | more ls | sort | lpr cat myfile.txt | head -7 | tail -5

ls | sort sort myfile.txt | uniq

* O/P Redirection:

- The o/p from a command normally intended for standard o/p can be easily diverted to a file instead. This capability known as o/p redirection.
- Notation '>' is used to redirect the o/p

Syntax: command1 > filename

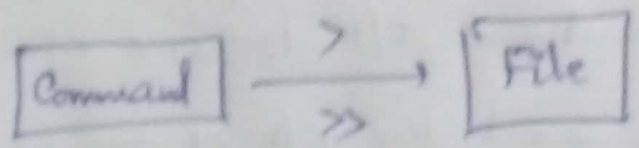
ex: who > 3/9/2020

wc myfile.txt > file2

Note: o/p is redirected to the existing file then the content of that file (if any) will be lost.

- We can use '>>' operator to append the o/p in existing file

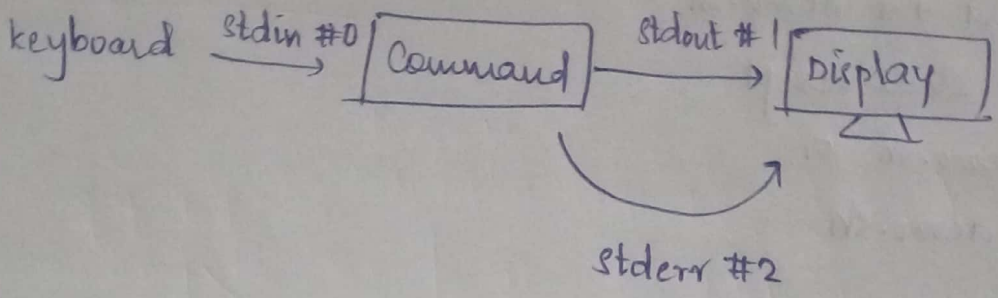
```
echo line1 > users
cat users
o/p: line1
echo line2 >> users
cat users
o/p: line1
     line2
```



Pipe buffer → temporary storage

* I/P Redirection:

```
wc -l users
#1 users
wc -l < users
#2
```



* wc -w<<eof
> this is here document
> hello
> h r u
> eof
8

* sort <<r → ends
> x
> g
> t
> a
> c
> r

* echo <<6 → ends
welcome to here document
6

* Shell Scripts:

1. Create a new file demo.sh using a text editor cat or vi
   nano demo.sh

2. Add the code
   #!/bin/bash ——→ bash shell
   echo "Hello World"

3. Set the script executable permission by running chmod

chmod +x demo.sh

4. Execute shell script

./demo.sh or

sh demo.sh

* Linux Metacharacters:

- A metacharacter is any character that has a special meaning.
- Linux has a fair no. of these metacharacters.

Symbol	Meaning
>	O/P redirection
>>	O/P redirection (append)
<	I/P "
*	File substition wildcard; zero/more characters
?	" " " ; one character
[]	" " " ; any char b/w brackets
'cmd'	Command Substitution
$(cmd)	" "
|	The pipe
;	Command sequence
||	or conditional execution
&&	and " "
()	group commands
&	run command in background
#	comment
$	expand the value of a variable
\	prevent or escape interpretation of next char
<<	I/P redirection

fills the gaps

ls 1?s? ← ?

displays first last

Ex: echo "\$9800"

O/P: $9,800

* File Name Substition:

File name subs. is a feature which allows special characters & pattern to substituted with file names in the current directory

The shell performs substitution when it encounters an expression that contains one or more special characters.

?	match any single character
*	match zero or more characters, including null
[abc]	match any characters b/w the brackets
[x-x]	match any characters in range x to x
[a-ce-g]	match any characters in range a to c, e to g
[!abc]	match any characters not b/w brackets
[!x-x]	match any characters not in range x to x
.	strings starting with '.' must be explicitly matched
?(pattern-list)	matches zero or one occurrence of any pattern
*(pattern-list)	match zero or more occurences of any.

* Command Substitution:

Command subs. is the mechanism by which the shell performs a given set of commands & then substitutes their O/P in the place of the commands.

```
DATE = 'date'
echo = "Date is $DATE"
USERS = 'who | wc -l'
echo = "Logged in user are $USERS"
UP = 'date ; uptime'
echo = "Uptime is $UP"
```

O/P:

Date is MON Sep 11 34:59:57 MST 2090

Logged in user are 1

Uptime is MON Sep 11 34:59:57 MST 2020 05:59:57 ....

Ex: (a) $a = 10

$ echo -e "Value of a is $a\n"

Same we can write as script

#! /bin/sh

a = 10

echo -e "Value of a is $a\n"

(b) echo "Computer name is $(hostname)"

(or)

echo "Computer name is `hostname`"

(c) var = $(cat "filename" 2>/dev/null)

* Shell variables:

- The name of a variable can contain only letters, no.s or the underscore
- Shell variables will have their names in uppercase

Setting & unsetting variables:

$ unset variable_name

* variables defined by user → local variables

System → global "

- ls ? → fetches all the files with one character name
- ls ?? → 2 characters

ls *.py = all files with .py

test 2 -gt 5 echo y

o/p: y

test 5 -gt 2 echo y

o/p: No o/p because false

test 5 -lt 7 echo yes || echo no

o/p: Yes

test 5 -gt 7 echo yes || echo no

o/p: No

* quotes:

single '
double "
back `

" " → normal quotes, commands inside it can be executed using $

ex: echo "date is $date"

o/p: date is ----

' ' → commands won't be executed

ex: echo 'date is $date'

o/p: date is $date

\ \ → used inside single & double quotes or without both of them
→ used instead of $ to execute commands

ex: echo "date is `date`"

o/p: date is ----

* Arithmetics in Shell:

The expr command is used to evaluate expressions & print out values

Ex: `expr 5 = 3`
`expr 8 != 5`
`expr 8 \< 5`

`expr 3 + 5` (space)
`expr 15 % 3`
:

The easiest way to do basic math on Linux CLI is using double paranthesis.

```
ADD = $((1+2))
echo $ADD
MUL = $((ADD *2))
echo $MUL
```

* if stmt:

Syntax:

```
if [condition]
then
<execute this>
else
<execute this>
fi
```

Ex:

```
cat if-stmt.sh          → shell script file
#!/bin/bash             → bash shell not mandatory
if [$1 -lt 100]         → $1: variable
then
echo "Your no. is smaller than 100"
else
echo "Your no. is greater than 100"
fi
```

O/P: `sh if_stmt.sh 34`
Your no. is smaller than 100

In place of scanf we use "read".

* For loop:

```
for i in <values>
do
<execute this>
```

done

*Ex:

```
for ((i=0; i<=5; i++))
do
echo "welcome to for loop $i times"
done
```

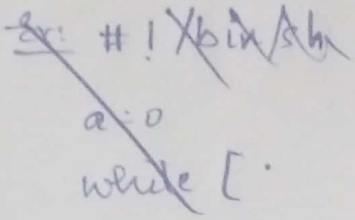
*while loop:

Ex: cat >while-loop.sh

```
#!/bin/bash
count=0
while [$count -lt 3]
do
echo Count is $count
count=$(expr $count +1)
done
```

O/P: sh while-loop.sh

```
Count is 0
Count is 1
Count is 2
```



3. Files & Directories

- All data in Linux is organised into files
- All files are organised into directories
- These directories are organised into a tree-like structure called the filesystem.

* Basic Types of files:

- **Ordinary files:**

It is a file on the system that contains data, text, or program instructions. In this tutorial, you look at working with ordinary files.

→ Readable files
→ Binary "
→ Image "
→ Compressed " & so on

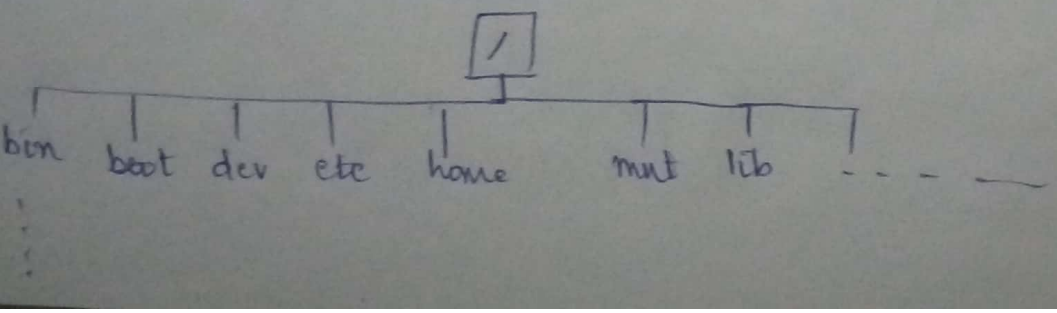
- **Directories:**

They store both special & ordinary files. For users familiar with Windows or Mac OS, Unix directories are equivalent to folders.

- **Special files:**

Provide access to hardware such as hard drives, CD-ROM drives, modems & Ethernet adapters. Other special files are similar to aliases or shortcuts & enable you to access a single file using diff. names.

* Linux file system structure:



* Types of files:

- Regular file (-)
- Directory file (d)

* Special files:

- Block file (b) - These files are hardware files most of them are present in /dev
- Character device file (c) - Provides a serial stream of I/P or O/P
- Named pipe file or just a pipe file (p)
- Symbolic link file (l)
- Socket file (s)

* Linux file tree:

- /(root): All files & directory appears (starts) under the root directory
- Only root user has the right to write under this directory.
- /bin: essential command binaries that need to be available in single user mode; for all users, cat, ls, cp ...
- /boot: boot loaders files like kernels, initrd
- /dev: device files
- /etc: host specific system configuration
- /lib: Essential shared libraries & kernel modules
- /media: Mount pt for removable media
- /mnt: mount point for mounting a filesystem temporarily
- /opt: Add-on application software packages
- /sbin: essential system binaries
- /srv: data for services provided by this system
- /tmp: temp. files
- /usr: secondary hierarchy
- /var

* Inodes:

The index node (inode) is a data structure that stores various information about a file in Linux.

- Mode
- Ownership
- Group
- File type
- File size
- no. of links

Each inode is identified by an integer no., assigned to a file when it is created.

$ ls -il

$ ls -i myfile.txt

Inode doesn't store the content of the file & filename.

* What happens with inode no. when:

- copy → many copies created
- move or
- delete a file on the filesystem

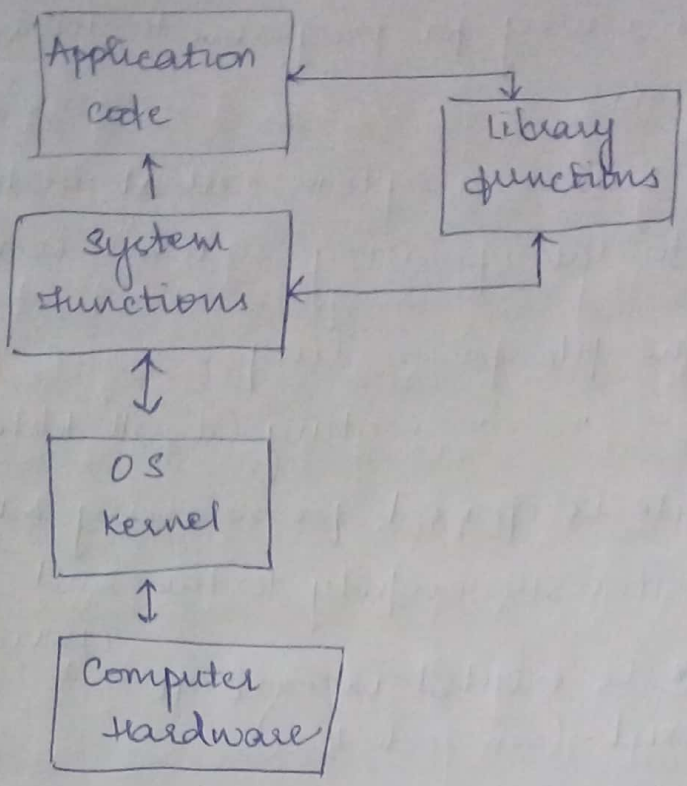
"An inode is a data structure that stores all the info abt a file except its name & its actual data".

In case of inodes are full, we need to remove unused files from the filesystem to make Inode free. There is no option to increase/decrease inodes on disk. It only created during the creation of filesystem on

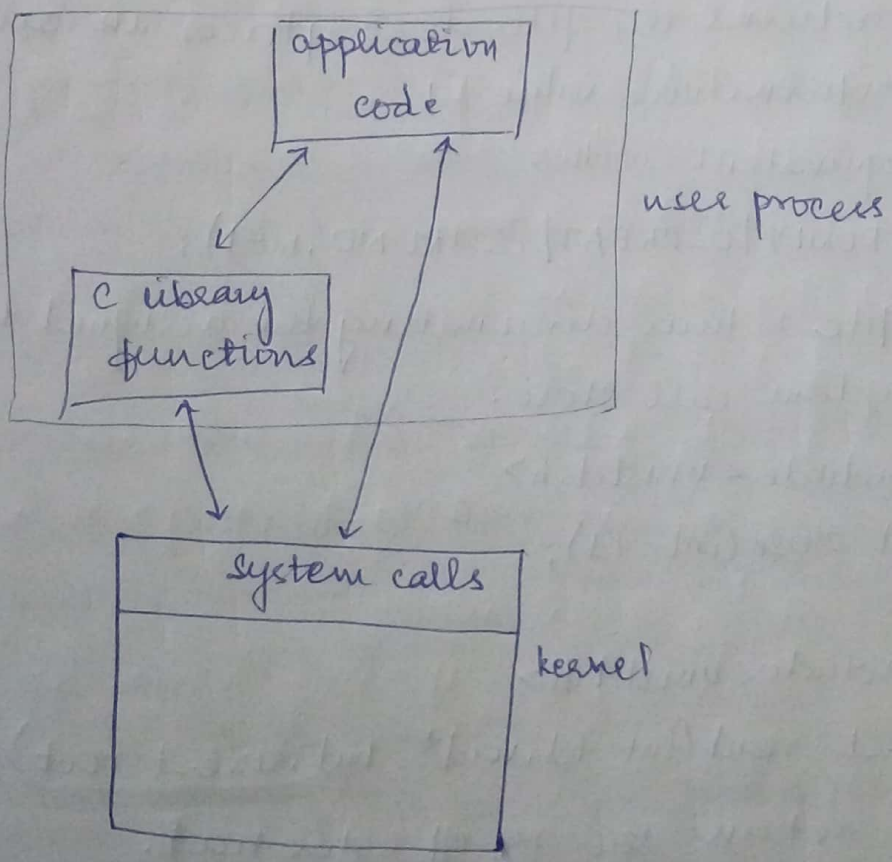
* Library functions kernel support for files:

Library functions can be of two types:

- Functions which do not call any system call.
- Functions that make a system call.



* Diff. b/w C library functions & system calls:



* System calls for file I/O operations:

- Open
- Create
- Read
- Write
- Close

- The system call is a way for programs to interact with the operating system.
- When the program makes a system call at that time it makes a request to the operating system's kernel.
- O_RDONLY : opens the file for reading
- O_WRONLY : " " " " writing (it will delete existing file content)
- O_RDWR : The file is opened for reading & writing
- O_APPEND : It writes successively to the end of the file (previous content present)
- O_CREAT : The file is created in case it didn't exist. (can lock this)
- O_TRUNC : If the file exists all of its content will be deleted.
- The function returns the file descriptor or in case of an error it returns the value -1

This call is equivalent with:

open (path, O_WRONLY | O_CREAT | O_TRUNC, mod);

- For closing a file & thus eliminating the assigned descriptor we use the system call close.

```
#include <unistd.h>      // unix standard
int close (int fd);      // file descriptor
```

*

```
#include <unistd.h>
ssize_t read (int fd, void* buf, size_t noct);
```

- The function returns the no. of bytes read.

```
#include <unistd.h>
ssize_t write (int fd, const void* buf, size_t noct);
```

- The function returns the no. of bytes written -1 in case an error occured.

* File attributes:

- Linux maintains a set of common attributes for each file in a file system.
- The attributes of a file can be determined using stat & fstat system calls and here we are using a header #include <sys/stat.h>
- The following are file attributes

1. File type (-) Type of file
2. Access permission (rw-r?r?) file access for owner, grp, other
3. Hard link count (1) no of hard links in a file
4. UID (lab2) user ID / owner ID
5. GID (user) file group ID
6. File size (50) size of file in bytes
7. Last access time / Last modify time / Last change time (Feb 9 13:19)
8. Inode no. (system inode no. of a file)
9. File system ID (test) file system ID where the file is stored

* Hard link vs Symbolic Link:

Hard link	Symbolic link
• does not create a new inode	• create a new inode
• cannot link directories, unless this is done by the root	• can link directories
• cannot link files across file systems	• can link files across file systems
• Increase the hard link count of the linked inode.	• Does not change the hard link count of the linked inode.

* File table:

- The set of open files is represented by a file table that is shared by all processes.
- Each file table entry consists of (for our purposes) the current file position, a reference count of the number of descriptor entries that currently point to it & ~~add~~ a pointer to an entry in the inode table.
- Closing a descriptor decrements the reference count in the associated file table entry. The kernel will not delete the file table entry until its reference count is zero.

* Some useful library functions:

Function	Description
abs	integer absolute value
ctime	convert date & time
fopen	open a stream
printf	formatted O/p
fputc	put character or word on a stream
getwd	get current working directory path name
Strcat	string concatenation
ctype	character classification & conversion macros & functions
mktemp	make a unique file name
puts	put a string on a stream
Sleep	Suspend execution for interval
stdio	Standard buffered I/O package

* Some standard I/O functions:

fopen(), fclose(), fflush(), fseek(), fgetc(), getc(), getchar(), fputc(), putc(), putchar(), fgets(), gets()

* Directories: creating, removing & changing:

Standard libraries & system calls provide complete control over the creation & maintainance of files & directories.

• mkdir():

The mkdir() system call is used to create the directories. It creates a new & empty directory.

Syntax: The function prototype of mkdir() is

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <error.h>
#include <unistd.h>
int mkdir (const char* pathname, int mode);
```

• rmdir():

The rmdir() system call is used to remove the directories.

Syntax:
```
#include <unistd.h>
int rmdir (const char* pathname);
```

• chdir():

The chdir() system call is used to change current working directory of calling process.

Syntax: The function prototype of chdir() is

```
#include <unistd.h>
int chdir (const char* pathname);
```

• chmod():

The chmod() i.e change mode system call helps to modify the permission flag.

Syntax: The function prototype of chmod() is

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
"       <error.h>
"       <unistd.h>
int chmod (const char* filename, int mode);
```

• chown():

This system call altees the owner or grp of a file. chown() system call helps in modifying the owner & group ID's of a file. In this on success, it returns a 0, on error returns -1.

Syntax: The function prototype of chown() is

```
#include <sys/types.h>
"       <sys/stat.h>
"       <fcntl.h>
"       <error.h>
"       <unistd.h>
int chown(const char* filename, uid_t owner-id, git_t
    groupid);
```