Course Name : eDAC MAY-2021 (Pune & Karad)
Module Name : Operating System Concepts with Linux

16 Class Room Hrs + 8 Lab Hrs = 24 Hrs.

OS DAY-01:

- there are four steps to learn an OS subject:
1. step1: "end user" -- linux commands - user commands

2. step2: "admin user" -- admin commands, shell script programming & installations.

+ Responsibilites of System Administrator:
- installing and configuring softwares, hardware and networks.
- monitoring system performance and troubleshooting issues.
- ensuring security and efficiency of IT infrastructure.

3. step3: "programmer user" -- system call programming

4. step4: "design/internals" -- to learn os internals/architecture

+ Introduction to an OS:

Q. Why there is a need of an OS?

Q. What is Computer?
- Computer is a machine/hardware/digital device mainly contains: Processor/ CPU, Memory Devices, I/O Devices etc....

- Basic Functions of Computer:
1. data storage
2. data processing
3. data movement
4. control
- by using computer machine various/different tasks can be performed efficiently and accurately.

- As any user cannot directly interacts with computer hardware, so there is a need of some interface between user & hardware, and to provide this interface is the job of an OS.

Q. What is a software?
- software is a collection of programs.

Q. What is a Program?
Program is a finite set of instructions written in any programming langauge (low level / high level), given to the machine to do specific task.

- There are 3 types of programs:
1. System Programs: programs which are in built into an OS OR part of an OS.
e.g. Kernel, device driver, memory manager, loader, dispatcher etc...

2. Application Programs: programs which are used for specific purpose and which are not a part of an OS
e.g. MS Office, zoom, notepad, google chrome, calculator, games, task manager etc....

3. User Programs: programs defined by programmers
C, Java, C++ etc...

- It is not possible for any user to directly interacts with an OS, and hence an OS provides 2 kinds of interfaces for the user in the form of program only this program is referred as a **shell.**

Shell – it is an application program through which user can interacts with an OS/Kernel.

- There are 2 types of Shell:
1. CUI (Command User Interface) Shell/CLI (Command Line Interface):
- in this type of interface user can interacts with an OS by means of entering commands in a text format through shell.
CUI shell accepts command from the user in a text format and execute it from the kernel or get done task from the kernel/OS.

Linux: terminal – bsh, bash, csh, ksh etc...
Windows: command prompt – cmd.exe, powershell
MS DOS: command .com

2. GUI (Graphical User Interface) Shell
- in this type of interface user can interacts with an OS by means of making an events like left click, right click, double click, menu bar, click on buttons, etc....

Windows: explorer.exe (goto task manager -> search for process named as explorer.exe -> right click on it say end task => GUI will be disappeared).
run task : explorer.exe OR restart
Linux: GUI Shell Programs
GNOME: GNU Network Object Model Environment
KDE: Kommon Desktop Environment

- We can install multiple OS's on single machine/HDD, but at a time only one OS can be active.
- VM Ware => Host OS & Guest OS
- bash, cmd.exe -> these application programs which either comes with an OS or can be installed later => **utility programs**

vs code editor – source code editor => editor specially used for programming

- **Eclipse is an IDE: Integrated Developement Environment,** written mostly in Java and its primary use is for developing java applications, but it may be used to develope applications in other programming languages via plug-ins, including Ada, ABAP, C, C++, C#, COBOL, D, Fortran, JavaScript, Perl, PHP, Python, R, Ruby etc...

- An IDE is **"application software"** which is a collection of tools/programs like an **editor(source code editor), compiler, linker, assembler, debugger** etc... required for faster software developement.
- Any IDE normally consists of a **source code editor, build automation tools, and a debugger.**
- Most of the IDEs have intelligent code completion.
- Some IDEs such as **netbeans** and **eclipse**, contains a compiler, interpreter, or both.
- IDE is in constrast with vi editor, gcc, ld etc....

e.g. eclipse, netbeans, code blocks, visual studio, borland turbo c, turbo c++, android studio, anjuta, Dev C++, CodeLite, QT Creator etc...

 # Compilation Flow:

+ editor : it is an application program used for text editing also can be used to write a source code.
e.g. notepad, gedit, vi editor, eclipse source code editor program, vs code editor etc...

+ "preprocessor": it is an application program gets executes before compilation and performs two tasks:
1. removes all comments from the source code, and
2. executes all pre-processor directives like #include, #define, #ifndef, #ifdef, #elif, #endif, #pragma etc... conditional compilation preprocessor directive i.e. header guards.

e.g. **"cpp" - c preprocessor, M4 macro processor in a UNIX like systems.**
- the output of preprocessor is an **intermediate code,** as this file gets created with the combination of header file and source file, size of this file gets increases and hence it is also called as an **expanded source code**.

- command to create an intermediate code/file from source code/file:
    $gcc -E -o program.i program.c ==> program.i

+ "compiler": compiler is an application program which converts high level programming language code (i.e. human understandable language code) into the low level programming language code (i.e. machine understandable language code), in our example from C programming language into an assembley language.

- output of a compiler is an **"assembly language code".**
- e.g. **GCC: GNU Compiler's Collection**, originally named as **GNU C Compiler**.
- GNU: GNU's Not UNIX/GNU is Not UNIX which is a recursive acronym.
- GNU: it is an Open Source Project by OSF (Open Source Foundation).
- Linux is also a GNU Project.
- Now a days GCC can be used for compilation of C++, FORTRAN, Objective C, Objective C++, Ada etc... programming languages.
- we need to use front ends of "gcc".
- Example: Borland's Turbo C, Turbo C++, Microsoft Visual C, etc...
- Compiler does **tokanization, syntax checking, code analysis, code optimization, parsing** etc...

"Compiler's Construction" -- By Aho, Ullman -- from AT&T Bells Labs.

- command to create an assembly language code from the source code:
    $gcc -S program.c ==> program.s

+ assembler: it is an application program which converts assembly language code into the machine language code i.e. **object code.**
e.g. Linux: **"asm"**
    Windows : Microsoft Assembler : **"masm"**
    Turbo Assembler : **"tasm"**
    etc...

- declarations of library functions exists in a **system header files** e.g. stdio.h, string.h, stdlib.h etc... , whereas definitions of all library functions are exists in a "lib" folder in a **"precompiled object module"** format.

+ linker: it is an application program which links object file(s) in a project with precompiled object modules of library functions and creates a "single" executable file.
e.g. **"ld"** -- link editor in Linux.

"build = compilation + linking"
compilation --> to create object code from source code.
linking --> linking of all object files with precompiled object modules by the linker and creates single executable file.

- command to create an executable file from object file:
    $gcc -o program.out program.o ==> program.out

- command to execute a program:
    $./program.out

program.c ==> [preprocessor] ==> program.i ==> [compiler] ==> program.s/.asm ==> [assembler] ==> program.o/.obj ==> [linker] ==> program.out/.exe

What is a Process?
- process is program in execution
- running program is called as a process
- when a program gets loaded into the main memory
- program is a passive entity, whereas a process is an active entity.

High level: instructions written in this programming languages can be understandable diretctly by the programmer user.
C, C++, Java, Python etc....

Low level : instructions written in this programming languages can be either understandable diretctly by the machine or close to the machine.
e.g. assembly language

- In C programming language there are some features like pointers, register storage class etc... by using which we can go close to the machine and hence C programming lanaguage is also called as **middle level.**

Scenario-1:
Machine -1: Linux : program.c
Machine-2 : Windows : program.c => build (compile + link) & execute => YES

Portability: program written in C on one machine/platform can be compile and execute on any other machine/platform.

Scenario-2:
Machine -1: Linux : program.c => build (compile + link) => program.out (executable code)

Machine-2 : Windows : program.out (executable code) => execute => **NO**

function or method or routine or procedure or algorithms or events/operations

Q. Why an executable file/code created on one machine/platform cannot be execute on any other machine platform?

- In Linux, file format of an executable file is **"ELF": Executable & Linkable Format (Formerly named as Extensible Linking Format)**, whereas in Windows, file format of an executable file is **"PE": Portable Executable.**
- An ELF is a common standard file format for **an executable files, object codes, shared libraries**, and **core dumps**.
- **core dump** is also called as **crash dump, memory dump, or system dump** consists of the recorded state of the working memory of a computer program at a specific time, generally when the program has crashed or otherwise terminated abnormally.
- An executable file is a program only, that contains set of instructions with extra information.
- In Linux, ELF is a **"sectioned binary"**, i.e. executable file in linux is divided logically into different sections, and each section contains specific data.
- An elf file mainly it contans: **exe header/primary header/elf header, data section, bss section i.e. block started by symbol section, rodata i.e. read only data section, code/text section, symbol table** etc....

1. "exe header/primary header": it contains info to starts an execution of a program (executable file), mainly it contains:
i. Magic Number: it is a constant number generated by the compiler which is file format specific.
- In ELF magic number starts with **"7f E L F"** -- ASCII values of letters E, L & F in its equivalent hexadecimal format.
- In PE -- magic number starts with ASCII values of letters M Z in its equivalent hexadecimal format, as **"Mark Zbikowski"** architect of Windows operating system at Microsoft.
ii. addr of an entry point function: it contains an addr of **_start( ) routine** in which addr of function can be kept from which an execution to be started. Bydefault it contains an addr of **main( ) function** and it can be changed as well.
iii. info about remaining sections (metadata -- data about data)
etc....

2. "data section": it contains initialized global & static variables.

3. "bss section (bock started by symbol)": it contains uninitialized global & static variables.

4. "rodata section (read only data section)": it contains constants & string literals.
e.g.
constants => 1000, 100L, 0x42, 012, 'A', 4.5f, 10.20 etc...
string literals => char *str = "sunbeam infotech";
5. "code/text section": it contains executable instructions

6. "symbol table": it contains info about symbols i.e. functions and their variables in tabular format.
etc...

+ Magic Number:
- Magic number are the first few bytes of a file which are unique to a particular file type. These unique bytes are reffered as **magic number**, also sometimes reffered as **file signature.**
- These bits/bytes can be used by the system **to differentiate between and recognize different files without file extension (specially in Linux/UNIX based OS).**

```
sunbeam@sachin-lenovo:~/feb2021/eDAC/OS/project$ size program.out
   text    data     bss     dec     hex filename
   1754     616       8    2378     94a program.out
```

intialized global & static variables:
```
sunbeam@sachin-lenovo:~/feb2021/eDAC/OS/project$ size program.out
   text    data     bss     dec     hex filename
   1754     624       8    2386     952 program.out
```

unintialized global & static variables:
```
sunbeam@sachin-lenovo:~/feb2021/eDAC/OS/project$ size program.out
   text    data     bss     dec     hex filename
   1754     616      16    2386     952 program.out
```

- loader: it is a system program (i.e. inbuilt program of an OS), which loads an executable file from HDD into the main memory.

- When we execute a program, loader first verifies file format of an executable file, if file format matches then it checks magic number, and if file format as well magic number both matches then only it loads an executable file from hdd into the main memory => an execution of a program is started OR process has been submitted.

 # What is an OS?
- It is a **system software** (i.e. collection of system programs), which acts as an **interface between user and computer hardware** (i.e. processor , memory devices & io devices etc...).
- An OS also acts as an **interface between programs (user & application programs) and computer hardware.**
- As an OS allocates required resources like **CPU time, main memory, i/o devices access** etc... to running programs, it is also called as **"resource allocator".**
- As an OS manages limited avaialable resources among all running programs, it is also called as a **"resource manager"**.
- An OS provides environment/platform to all types of programs to complete their execution.
- An OS controls execution of all programs, as well as it controls all h/w devices connected to the computer system, so it is also called as a **"control program".**

- An OS is a **software** (i.e. collection of hundreds of system programs and application programs in a binary format) which comes in either CD/DVD/PD, mainly having three components:
OS = Kernel + Utility Programs + Application Programs.

1. Kernel: it is a core part/program of an OS that runs continuosly into the main memory does basic minimal functionalities of it.
i.e. Kernel = OS - without kernel OS is nothing
- **"Kernel is OS OR OS is Kernel"**.

2. Utility Programs/Softwares:
- e.g. disk manager, task manager, anti-virus software, explorer.exe, backup software etc...

3. Application Programs/Softwares:
- e.g. MS Office, vi editor, etc....

- Linux Kernel – its source code is freely available onto the internet => **open source OS, specially designed for R&D.**
- Windows Kernel source code is not freely available – commercial OS specially designed for business.

+ Total there are 8 functions of an OS:
kernel /compulsory / core functionalities/basic minimal functionalties:
1. Process Management
2. Memory Management
3. File & IO Management
4. Hardware Abstraction
5. CPU Scheduling

---

extra utility/optionl functionalities
6. User Interfacing
7. Networking
8. Protection & Security

Installation of an OS:
- To install an OS onto the computer machine, is nothing but to store OS software (i.e. collection of thousands of system programs & application programs which are in a binary format/object code format) onto the **HDD.**

Booting:
- If any OS wants to becomes active, atleast its core program i.e. kernel must be loaded intially from HDD into the main memory, and process of loading of kernel of an OS from HDD into the main memory is called as a **booting** and it is done by **bootstrap program.**

- bootstrap program is in HDD, which is invoked by **bootloader program.**
- bootloader program resides into the HDD

- While booting kernel of an OS gets loaded into the main memory and runs continuosly into it does basic minimal functionalities, and kernel remains present inside the main memory till we do not shutdown the system.

core computer system: components which are mounted/attached onto the motherboard onto the slots is referred as core computer system.
e.g. CPU, Main Memory, Cache Memory, ROM etc...

peripheral devices/peripherals/external devices: devices which are connected to the motherboard externally through ports.
e.g. keyboard, monitor, mouse, hdd etc...

- bootable device/partition: if any **storage device/partition** contains one special program called as **bootstrap program** in its first sector i.e. in a **boot sector (first 512 bytes in a storage device/partition)**, then such a device/partition is referred as bootable device/partition.

- if boot sector of any storage device do not contains bootstrap program then it will be referred as non-bootable.

There are 2 stpes of booting:
1. machine (h/w) boot:
step-1: when we swicthed on power supply, current gets passed to the **motherboard** on which **ROM Memory** is there, which contains one micro-program called as **BIOS (Basic Input Output System) gets executes first.**

step-2: first step of BIOS is **POST (Power On Self Test)**, under POST it checks wheather all peripheral devices are connected properly or not and their working status.

step-3: after POST, BIOS program invokes **bootstrap loader program**, which searches for available bootable devices in a computer system, and it selects only one bootable device at a time as per the priority decided in a BIOS settings. (bydefault it selects HDD as a bootable device).

2. system boot:
step-4: after selection of HDD as a bootable device, **bootloader program** which is present inside it gets invokes, and it displays list of names operating systems installed onto the HDD, from which user need to select any one OS at a time.

step-5: upon selection of any OS, **bootstrap program** of that OS gets invokes and it locates kernel into the HDD and load it into the main memory.

- We can install multiple OS's on single HDD, but to do this we need to divide HDD logically into the partitions and on each partition we can install one OS.

OS DAY-02:
- UNIX
- Windows
- Linux
- MAC OS X
- Android
- Solaris
- Symbian
- Palm OS
- Google OS
- Kali Linux
etc....

UNIX (UNICS): Uniplexed Information & Computing Services/System

Q. Why UNIX?
- OS Concepts with Linux
- Linux is UNIX like OS i.e. Linux OS was developed by getting inspired from UNIX OS.
- Whatever is there in UNIX, everything is there into the Linux.
- **UNIX OS is treated as mother of all modern OS's** like : Linux, Windows, MAC OS X, iOS etc..., as all these OS's follows system arch design of UNIX.
- UNIX was specially designed for developers by the developers.
- UNIX was developed @AT&T Bell Labs in US in the decade of 1970's by Ken Thompson, Denies Ritchie & team.
- In UNIX first time, **multi-tasking, mult-threading & multi-user** feateures has been introduced.
- UNIX is the most secured OS and hence Linux
- Filesystem of UNIX is most secured, files of one user can be protected from another users by giving perms.
- UNIX & Linux mostly used for Server machines.

- Linux OS was developed by **Linus Torvalds in University Helsinki, in the decade of 1990's, and first linux kernel (vimlinuz) version 0.01 was released in the year 1991.**
- Linux is an open source OS: source code of linux kernel is freely available onto the internet, so anyone can download it and can add its own utilities and come up with their of distro of Linux
- Ubuntu Linux
- Fedora Linux
- Centos Linux – Embeded Developement
- RedHat Linux
etc...

- Android (Java) is Linux based OS used in mobile smart phones.

+ System architecture of UNIX
- Kernel acts as an interface between user programs & computer hardware.
- Human Body System:
    - respiratory system
    - digestive system
    - nervous system
    - reproduction system
    etc...

- for functioning of whole human body system all subsystems works together, similarly, in any OS, there are diff subsystems which works together for functioning whole OS.
- An OS contains following subsystems:
 - file subsystem
 - process control subsystem: ipc, memory management & scheduling
 - system call interface block
 - buffer chache
 - character devices & block devices block
 - device drivers
 - hardware abstraction block
etc...

- there are 2 major subsystem of any OS:
1. process control subsystem
2. file subsystem.

- file & process are two very important concepts of any OS.
- In UNIX, “file has space and process has life”
- In UNIX, whatever that can be stored is considered as a file, whereas whatever is active is conisedered as a process.

- KBD => hardware / input device => file
- Monitor => hardware / output device => file
- HDD => hardware / memory device => file

UNIX catagorised deviced into two catagories:
1. character devices: devices from which data gets tranferes char by char i.e. byte by byte
UNIX treats character devices as a character special device files (c).
e.g. keyboard, monitor, printer, mouse etc....

2. block devices: devices from which data gets tranferes block by block i.e. sector by sector (usually size of 1 sector = 512 bytes).
UNIX treats block devices as a block special device files (b)

e.g. all storage devices like HDD, PD, CD, DVD etc...

When we copy data from HDD to PD
- UNIX treats this as a tranfer of data from one file to another file.

- In UNIX / Linux there are 7 types of file:
1. regular file (-) : e.g. text files, image files, audio files, video files, source files etc...

2. directory file (d) : UNIX treats all directories are as well dir file whose contents are names of files & sub dir's in it.

3. character special device file (c): all character devices

4. block special device file (b): all storage devices

5. linkable file (l) : it is used to store link of existing file
its a symbolic link (like in windows – shotcut of any file which is onto the desktop).

6. pipe file (p): this type of file is used in inter process communication in pipe command & pipe( ) system call.

7. socket file (s): this type of file is used in inter process communication across the systems.

Human Body + Soul => Active => Process
Human Body - Soul => Passive => Dead Body => File

- file has space & process has life

- device driver: it is system program/s which enables the kernel/system to communicates with that particular hardware/device.

- buffer cache – it is a purely software technique, in which kernel used portion of the main memory to keep/store most recently accessed disk contents to achieve max throughput in min hardware movement.

- hardware control block is used by the kernel for hardware abstraction.

- L1 cache & L2 cache (level -1 & level-2 cache) its hardware, cache memory used to reduce speed mismatch between the CPU and cache memory.
- L1 cache => data cache
- L2 cache => instruction cache
- In modern CPU's cache memory is inbuilt into it.

- file & io management, process management, ipc, scheduling, memory management, hardware abstraction are services/functionalities of the kernel.

Kernel = Program

functions: system calls:
fork( )
open( )
read( )
write( )
.
.
.

Linked List: slll.c

main( ); => client

services:
create_node( );
add_last( );
add_first( );
add_at_pos( )
delete_last( );
delete_first( );

- system calls: are the functions defined in C, C++ & assembly language which provides an interface to the services made available by the kernel for user (programmer user).
System call interface block => system calls API's : Application Programming Interface

- If any user (programmer user) wants to use services made available by the kernel in his/her program, then either it can be used directly by means of giving call to system call API's or can be used indirectly by means of giving call to library functions from inside which there is call given to system call API's only.

Wrapper functions => are used just for calling original fuctions.

file handling:
C Programming:
fopen( ) lib function internally makes call to open( ) sys call api, which internally makes call to _open( ) sys call.

fopen( ) => open( ) sys call api : to open a file / to create a new file

fwrite( )/fprintf( )/printf( )/fputs( )/fputc( ) => **write( )** sys call api => **_write( )** system call.

fread( )/fscanf( )/scanf( )/fgets( )/fgetc( ) => **read( ) sys call api** => **_read( )** sys call.

- In UNIX there are 64 system calls
- In Linux there are 300 system calls
- In Windows there are more than 3000 system calls

In UNIX => fork( ) : to create a new process / child process
In Linux => fork( ) / clone( ) / vfork( ) : to create a new process / child process

In Windows => CreateProcess( ) : to create a new process / child process

- Irrespective of an OS there are 6 catagories of system calls:
1. file manipulation/operations system calls: e.g. open( ), write( ), read( ), close( ), lseek( ) etc...

2. device control system calls: e.g. open( ), write( ), read( ), close( ), ioctl( ) etc...

3. process control system calls: e.g. fork( ), _exit( ), wait( ) etc...

4. accounting information system calls: e.g. getpid( ), getppid( ), stat( ) etc...

5. inter process communication system calls: e.g. pipe( ), signal( ), kill( ) etc...

6. protection & security system calls (filesystem level) : chmod( ), chown( ) etc...

```
//user defined code/user program
#include<stdio.h>
int  main(void)
{
        int n1, n2;
        int res;

        printf("enter n1 & n2: ");//write( ) sys call – system defined code
        scanf("%d %d", &n1, &n2);//read( ) sys call - system defined code

        res = n1 + n2;//arithmetic expression
        printf("res = %d\n", res);//write( ) sys call – system defined code

        return  0;//successfull termination
}
```

sum.c => build => sum.out => execute

- Whenever system call gets called the CPU switches from user defined code to the system defined code, and hence system calls are also called as **software interrupts/trap.**

What is an interrupt?
- an interrupt is a signal which recieved by the CPU from any io device due to which it stops an execution of one job/process and starts executing another job/process => **hardware interrupt.**

- throughout an execution of any program, the CPU switches between user defined code and system defined code, and hence we can say system runs in a two modes: kernel mode & user mode, and this mode of operation of the system is called as **dual mode operation/dual mode protection.**

1. kernel mode: when the CPU executes system defined code instructions, we can say system runs in a kernel mode/system mode.
- kernel mode is also called as **system mode/protected mode/priveledged mode/monitor mode.**

2. user mode: when the CPU executes user defined code instructions, we can say system runs in a user mode.
- user mode is also called as **non-priveledged mode**

- The CPU can differentiate between user defined code instructions and system defined code instructions by using one bit which is onto the CPU called **mode bit** maintained by an OS.
mode bit = 0 => kernel mode
mode bit = 1 => user mode

- Whenever system call occurs, an OS clears the mode bit (to make the value of bit as 0), after completion of system call value of mode bit will be set (to make the value of mode bit as 1).
- The CPU can have access of hardware devices only in a kernel mode, access h/w devices is restircted for the CPU in a user mode, protection of h/w devices can be achieved.

user written one program => there is logic to crash hdd

the CPU is currently executing user defined code => mode bit = 1 => user mode.

=> There are major subsystems of any OS:
1. Process Control Subsystem
2. File Subsystem

Linux:
- When we install Linux OS, we need to create two partitions:

1. swap partition – is used by linux system as a swap arear in which inactive running programs temporarily.

- **conventionally** the size of swap partition/swap area should be doubles the size of main memory.
- if the size RAM = 4 GB => size of swap partition should be 8 GB.
- if the size RAM = 2 GB => size of swap partition should be 4 GB.
- if the size RAM = 8 GB => size of swap partition should be 16 GB.
.
.
.

- linux system use swap partition which is a **portion of HDD** as an extension of the main memory in which **inactive running programs** can be kept temporarily.

2. data partition "/" root (/) partition – in which linux system keeps/stores all data in an organized manner => **filesystem** i.e. **linux filesystem** follows heirarchical structure (inverted tree like structure).

(/) root dir : starting point/dir of linux filesystem structure.
- linux filesystem structure starts from root (/) dir.

- root dir i.e. "/" dir contains sub-dir's like: "/boot", "/bin", "/sbin", "/etc", "/lib", "/usr", "/home", "/root" "/mnt", "/dev" etc....

1. **"/boot":** contains static files of the boot loader, kernel, initrd (initial ramdisk).
- it contains information about booting the system
- it contains linux kernel by the name **vmlinuz.**

2. **"/bin":** contains user commands in a binary format, e.g. commands like ls, cat, cp, mv etc...

3. **"/sbin":** contains admin/system commands in a binary format, e.g. lscpu, adduser, deluser, mkfs etc...

4. **"/home":** contains home dir's of all users for multi-user system.
- for any user on its creation of an account by default subdir by the name of the user got created by the operating system, in which that user can store data, user can have read, write as well execute perms in that dir.

5. **"/root": root** is a home dir for root user

6. **"/etc":** contains host specific system configuration files - it is like a **"control panel"** in windows.
- in computing configuration files (or config files) are files used to configure the parameters and initial settings for some computer programs. They are used for user applications, server processes and operating system settings.
- e.g. **YaST: Yet another Setup Tool** (Linux operating system setup and configuration tool) or **debconf** (for performing system-wide configuration taskson UNIX-like operating systems).
"/etc/opt": contains configuration files for **add-on-packages** that are stored in /opt.
7. **"/dev":** contains device files

8. **"/lib":** contains shared libraries required for /bin and /sbin.

9. **"/mnt":** contains temporarily mounted filesystems

10. **"/media":** mount point for removable media such as CDROMs.

11. **"/opt":** contains optional application software packages.

12. **"/proc":** virtual filesystem provides process and kernel information as a files. In Linux , corresponds to a **procfs** mount. Generally automatically generated and populated by the system, on the fly.

- In Linux filesystem hierarchy, each file is uniquely identified by its path
path : it is a unique location of any file in a filesystem structure represented/denoted in a **string format** and file names/sub dir name are seperated by **delimiter char (delimiter).**

- there are 3 delimiter chars:
1. slash (/) - used in UNIX based OS's/Linux
2. backslash (\ ) - used in Windows
3. colon (:)

example:
"/" => root (/) dir
"/bin" =>
"/home/sunbeam/feb2021/eDAC"

"C:\sachin\edac"

"https://us06web.zoom.us/support/down4j"

- any file in a linux filesystem can be accessed by using 2 ways:
and hence there are two types of path:
1. absoulte path: it is a full path of any file which starts from root (/) dir.
i.e. path of the file with respect to root (/) dir.

2. relative path: path of the file with respect to current directory.

- whenever we create new dir, bydefault two hidden entries gets created inside a dir file.
. (single dot) => current dir
.. (two dots) => parent dir

- root (/) dir is a parent of itself
inode =>

- In Linux filename which starts with dot (.) [the period] is considered as a hidden file.

Basic Linux Commands:

Command Name: "pwd" - print/present working directory
- this command displays an absolute path (i.e. full path) of the present/current working directory
- pwd command internally refers the value of shell variable by the name "PWD".

Command Name: "cd" – to change current working directory
$cd <dirpath> -- change dir to the dirpath
$cd / -- goto the root directory -- "/" slash -- is user short hand variable
$cd ~ -- goto the user's home dir -- "~" tild symbol -- is user short hand variable
$cd -- goto the user's home dir
$cd - -- goto the prev directory
$cd . -- remains in current dir
$cd .. -- goto the parent dir of the current working dir

Command Name: "ls" - to display contents of the directory/list directory contents.
- contents of the directory files are nothing but name of files and sub-dir's.
$ls <dirpath> -- it displays contents of the dir by the name "dirpath".
$ls -- bydefault it displays contents of the current directory columnwise
$ls -l -- its displays contents of the dir in a listing format.

total n
n = total no. of data blocks allocated for the files and sub dir's in a given dir
feild-1: first char of feild 1 denotes type the file next 3 chars of field 1 denotes access perms for user/owner
next 3 chars of field 1 denotes access perms for group members last 3 chars of field 1 denotes access perms for group others.
feild-2: denotes no. of links exists for that file for regular file bydefualt no. of links i.e. link count = 1
for directory file bydefualt no. of links i.e. link count = 2.
feild-3 : denotes name of an user/owner
feild-4 : denotes name of group
feild-5 : denotes name of size of the file
feild-6 : denotes time stamps - date of creation and last modified date & time in ISO format.
feild-7 : name of the file
- In UNIX/Linux filename starts with (".") is considered as a **hidden file**
- the period (".")

options/flags/args description:
-h : displays size of the files in human redable format
-l : displays contents of the dir in a listing format
-a : display all contents of the dir (including hidden files)
-A : display all contents of the dir (including hidden files but excluding entries for "." & "..").

-i : display inode number of the files inode number is an unique idenfier of the file.
etc..... explore more options from man pages

Command Name: "cat" -
- cat command is used for concatenation of contents of file/files and print it onto standard output.
- cat command can be used to display contents of the regular file.

Command Name: "chmod" - used to change file mode bits
- to change (i.e. to assign/ to remove) access permission for user/owner, group members and others.

+x : to assign execute perm to all
+r : to assign read perm to all
+w : to assign write perm to all
u+r: to assign read perm to user/owner
u+w: to assign write perm to user/owner
u+x: to assign excecute perm to user/owner
g+r: to assign read perm to group members
-x : to remove execute perm from all
-r : to remove read perm from all
-w : to remove write perm from all
u-r: to remove read perm from user/owner
u-w: to remove write perm from user/owner
u-x: to remove excecute perm from user/owner
g-r: to remove read perm from group members
.
.
.

hexadecimal values:
r -- read -- 4
w -- write -- 2
x -- execute -- 1
.
.
.

Command Name: "cp" - to copy file/files

$cp <source> <destination>

Command Name: "mv" - command used to move file/s from one location to another location.