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

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

## 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.