EXPERIMENT 1
Introduction to OS and LINUX

OBJECTIVE:

- To get familiarized with the basics of operating systems
- Learn the basic commands used in Linux

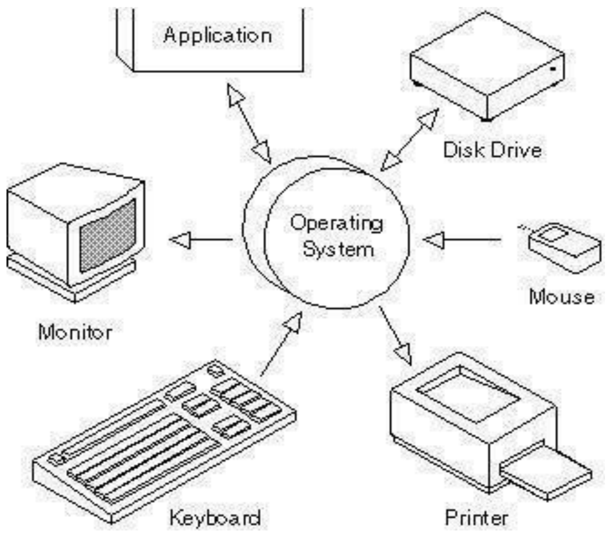
BACKGROUND:

On Computer Startup:

- Power-on self-test (POST) checks for errors
    - CPU
    - Memory
    - Basic input-output systems (BIOS)
- BIOS/firmware
    - Activates the computer's hard disk drives
- Bootstrap loader
    - First piece of the operating system
    - Has a single function to load the operating system into the memory

Operating System:

- What is Operating System?
  Supports computer's basic functions as shown in Figure 1.1
- What tasks an OS Perform?
    - Processor management
    - Memory management
    - Device management
    - Storage management
    - Application interface
    - User interface
- Types
    - Linux
    - Windows 8, Windows 7, Vista, XP
    - Mac



OS Basic Functions
Figure 1.1

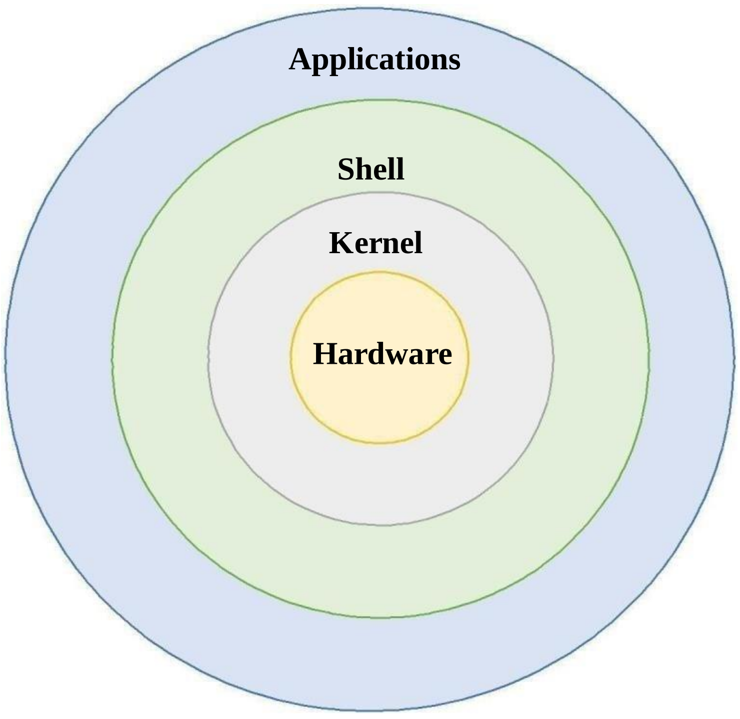
What is LINUX?

- A fully-networked 32/64-Bit Unix-like Operating System
    - Compilers Like C, C++
- Multi-user, Multitasking
- Coexists with other Operating Systems
- Includes the Source Code
- Open Source

Why is it Significant?

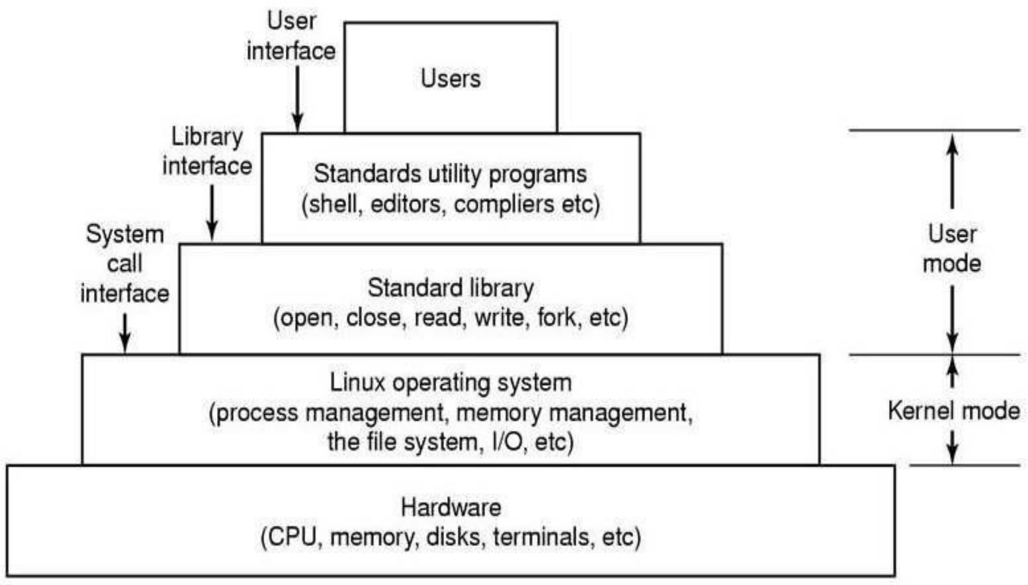
- Growing popularity
- Powerful
    - Runs on multiple hardware platforms
    - Users like its speed and stability
    - No requirement for latest hardware
    - It is free
    - Licensed under GPL (General Public License)

System Structure:
An operating system is a construct that allows the user application programs to interact with the system hardware. This is further divided in two layers i.e Kernel and Shell. Kernel deals with the Hardware and Shell deals with Applications as shown in Figure 1.2.



Structure of an Operating System
Figure 1.2

The Linux System:
Linux is the best-known and most-used open source operating system. As an operating system, Linux is software that sits underneath all of the other software on a computer, receiving requests from those programs and relaying these requests to the computer's hardware. Linux System Structure is shown in Figure 1.3.


User interface
Users
Library interface
Standards utility programs (shell, editors, compliers etc)
System call interface
Standard library (open, close, read, write, fork, etc)
Linux operating system (process management, memory management, the file system, I/O, etc)
Hardware (CPU, memory, disks, terminals, etc)
User mode
Kernel mode

Linux System Structure
Figure 1.3

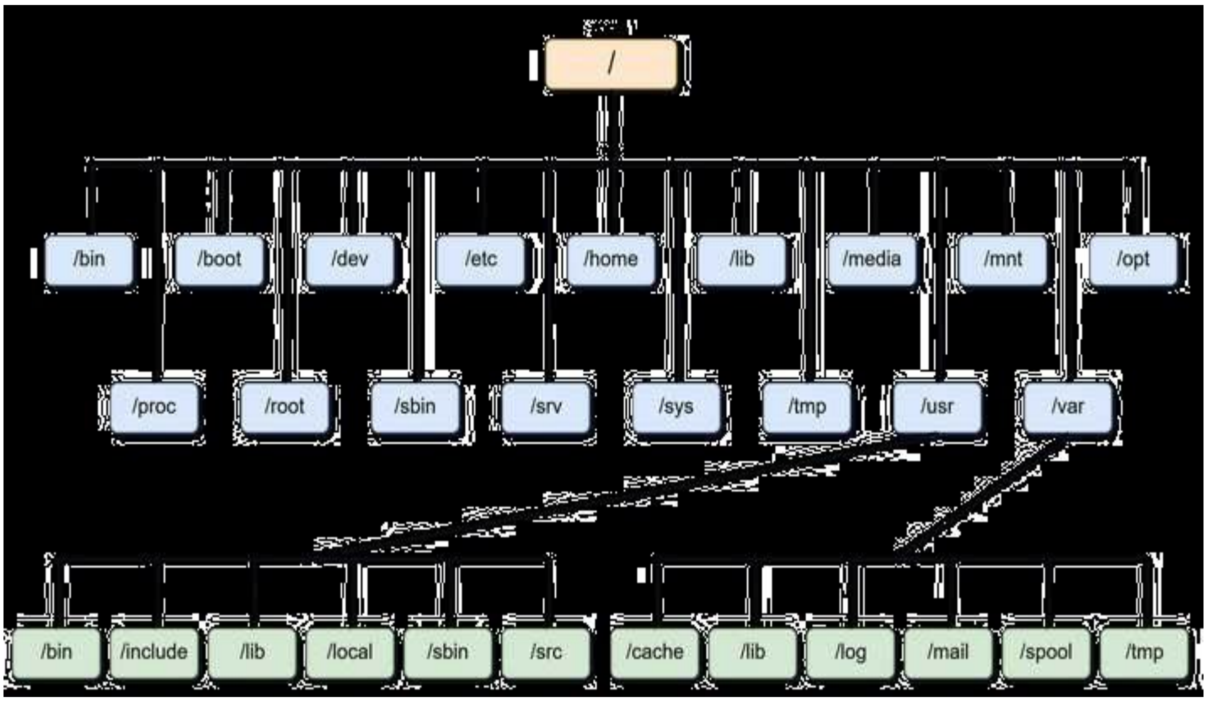
Linux Command Basics:

- To execute a command, type its name and arguments at the command line
- <command_name> <space> <options> <space> <arguments>

Editors:
Several Choices available:

vi	Standard UNIX editor
the	XEDIT like editor
xedit	X windows text editor
emacs	Extensible, Customizable Self-Documenting Display Editor
pico	Simple display-oriented text editor
nedit	X windows Motif text editor

The File system:
Unix uses a hierarchical file system structure, much like an upside-down tree, with root (/) at the base of the file system and all other directories spreading from there as can be seen from Figure 1.4. Linux differs from Windows in many ways. The comparison has been given in Table 1.5.



Linux File System
Figure 1.4

Windows	LINUX
• The directories in MS-DOS path are separated by '\' • File names are case insensitive. • Where DOS/Windows had various partitions and then directories under those partitions. • An executable is one with an extension of .exe, .com or .bat. • You can set attributes to make file read only, hidden	• Paths are separated by '/'. • File names are case sensitive. • There is only a single hierarchal directory structure (resembles a tree). Everything starts from the root directory, represented by '/', and then • expands into sub-directories. • Any file whose execute permission is turned on is executable • You can set permissions on a file

Table 1.5 (Comparison between Windows OS and Linux OS)

Special Files:

- **/home -** all users' home directories are stored here
- **/bin, /usr/bin -** system commands
- **/etc -** all sorts of configuration files
- **/var -** logs, spool directories etc
- **/dev -** device files
- **/proc -** special system files

Virtual Machine:

- What is virtual Machine?
  - VirtualBox and VMWare
- ISO files – Ubuntu ISO file
- Ubuntu installation on VirtualBox or VMware

Installation of Linux in Virtual Machine:
a. Install VMware on your Machines.
b. Get Latest ISO file of Ubuntu distribution according to your system architecture (32bit or 64bit) from following link http://www.ubuntu.com/download/desktop.
c. Install Ubuntu from this ISO image file as guest Operating system in VMware.

In-Lab Questions:

1. Run the following commands:

Command	Description
`pwd`	Display the current working directory.
`ls`	List files and directories.
`cd Desktop`	Change the current directory to Desktop.
`mkdir NewFolder`	Create a new directory named "NewFolder".
`rmdir NewFolder`	Remove the empty directory "NewFolder".
`cp file.txt /tmp`	Copy "file.txt" to the "/tmp" directory.
`mv file.txt newname.txt`	Rename "file.txt" to "newname.txt".
`rm oldfile.txt`	Remove the file "oldfile.txt".
`touch newfile.txt`	Create an empty file named "newfile.txt".
`nano myfile.txt`	Open the nano text editor for editing.
`cat myfile.txt`	Display the content of "myfile.txt".
`echo "Hello, Linux!"`	Display the message "Hello, Linux!".
`man ls`	Display the manual page for `ls`.
`chmod 755 script.sh`	Change the permissions of "script.sh".
`chown user:group myfile.txt`	Change the owner and group of "myfile.txt".
`ps aux`	Display information about active processes.
`kill 1234`	Terminate the process with PID 1234.
`top`	Display real-time system statistics.
`df -h`	Display disk space usage.
`du -h`	Display file and directory space usage.
`grep "pattern" file.txt`	Search for a pattern in "file.txt".
`find /home/user -name "*.txt"`	Search for ".txt" files in /home/user.
`tar -czvf archive.tar.gz folder/`	Create a compressed archive of "folder/".
`gzip file.txt`	Compress "file.txt" and replace it with "file.txt.gz".

2. Type and save the following code as hello.cpp. Now compile it to generate assembly code using $ g++ -S hello.cpp

```
#include <iostream>

int main() {
    // Print "Hello, World!" to the console
    std::cout << "Hello, World!" << std::endl;

    // Return 0 to indicate successful completion
    return 0;
}
```