Lecture 2: All about Linux

Linux History

The Unix Roots (1969 – 1980s)

- 1969: Unix was created at AT&T Bell Labs by Ken Thompson and Dennis Ritchie.
- It was a **multiuser, multitasking** OS written originally in assembly, later in **C** (which made it portable).
- Universities, research labs, and companies adopted it.
- AT&T put **licensing restrictions**, so Unix wasn't truly free.

The GNU Project (1983)

- **Richard Stallman** launched **GNU Project** in 1983.
- Goal: build a **free and open-source Unix-like OS**.
- GNU provided many key components:
  - GNU Compiler Collection (**GCC**)
  - GNU Shell (bash)
  - Core utilities (cp, ls, grep, etc.)
- But GNU lacked one thing: a **kernel** (their GNU Hurd kernel was delayed).

Birth of Linux Kernel (1991)

- **Linus Torvalds**, a Finnish student at University of Helsinki, started a personal project to build a small free operating system kernel.
- **August 25, 1991**: Linus announced his project on a Usenet group: "I'm doing a (free) operating system (just a hobby, won't be big and professional like GNU)."
- He released **Linux 0.01** in **September 1991**.
- It was inspired by **Minix** (a teaching OS created by Andrew S. Tanenbaum).

Linux + GNU = A Complete OS (1992–1994)

- The Linux kernel + GNU tools were combined, creating a **fully functional free OS**.
- This is why many people call it **GNU/Linux**.
- **1992**: Linux adopted the **GNU General Public License (GPL)**, ensuring it stayed free and open.
- **1994**: Linux 1.0 was released—first official stable kernel version.

Rise of Distributions (Mid-1990s)

- Developers began packaging Linux with software, making **Linux distributions (distros)**.
- Early distros: **Debian (1993)**, **Red Hat (1994)**.
- Distros made Linux easier to install and use.

Linux in Enterprises & Servers (Late 1990s – 2000s)

- **IBM, HP, Dell** adopted Linux for servers.
- Linux grew in **web hosting, supercomputers, and enterprise data centres**.
- Projects like **Apache (web server)** and **MySQL** ran on Linux, forming the famous **LAMP stack (Linux, Apache, MySQL, PHP/Perl/Python)**.

Linux Everywhere (2000s – Present)

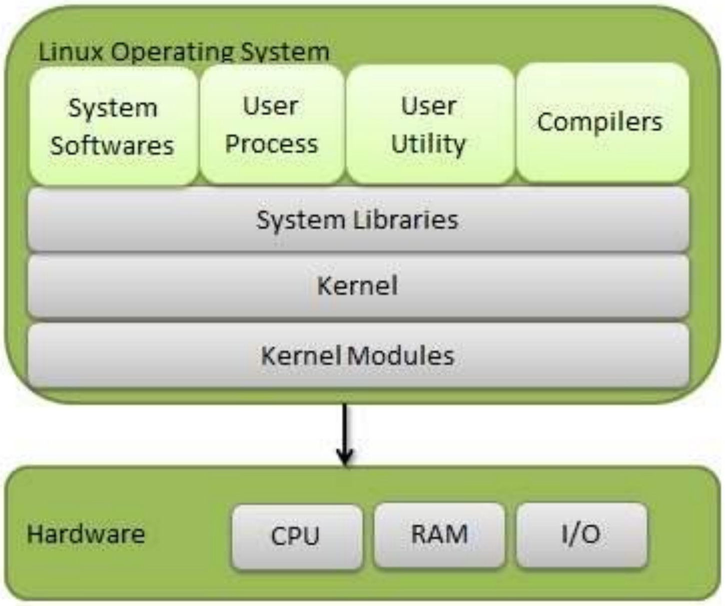
- **Supercomputers**: By 2010s, almost **100% of the world's fastest supercomputers** ran Linux.
- **Mobile**: **Android (2008)** is based on the Linux kernel → making Linux the world's most widely used OS on phones.
- **Cloud & Containers**: Linux became the backbone of **cloud computing, DevOps, Docker, Kubernetes**.
- **IoT & Embedded**: Routers, smart TVs, drones, cars → many run Linux.

Modern Linux

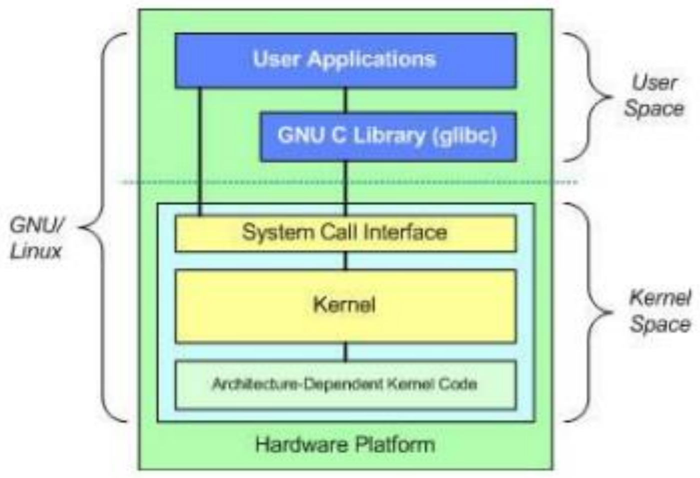
- Kernel maintained by **Linus Torvalds** with help from thousands of developers worldwide.
- **Linux Foundation** (est. 2000) coordinates development.
- Major distros today: **Ubuntu, Fedora, Debian, Red Hat, SUSE, Arch, Alpine**.
- Philosophy: **open-source, modular, community-driven innovation**.

Linux Architecture

Linux is an operating system's kernel. The kernel controls everything. It performs tasks that create and maintain the Linux environment. It receives instructions from Shell (used by the user to give commands to Linux) regarding hardware usage for desired tasks.



Linux Kernel:



Hardware Layer

- Physical components: CPU, RAM, storage, network cards, I/O devices.

Kernel Layer

- Core of Linux, runs in **privileged mode**.
- Directly controls hardware using device drivers.
- Provides abstraction to user programs.

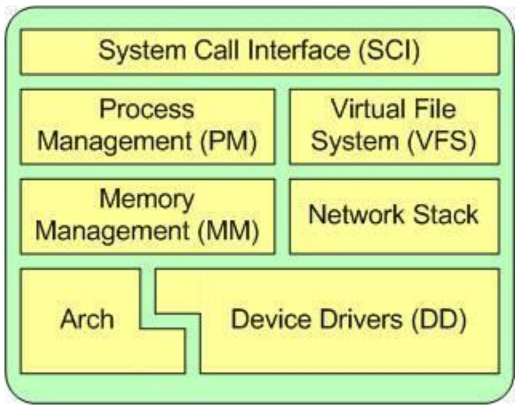
System Call Interface (SCI)

- Bridge between **user space** and **kernel space**.
- Applications request services (e.g., open file, create process) via system calls.

User Space

- **Shells:** CLI (bash, zsh, sh) or GUI.
- **Utilities:** ls, cp, grep, cat, etc.
- **Applications:** web browsers, compilers, text editors.

There is also the GNU C Library (glibc). This provides the system call interface that connects to the kernel and provides the mechanism to transition between the user-space application and the kernel (each has its own address space).



Detailed Kernel Components

The Linux **kernel** has 5 major subsystems:

a) Process Management

- Creates, schedules, and terminates processes.
- Keeps track of process states (running, waiting).
- Handles multitasking
- Tools: ps, top, kill.
- In the kernel, these are called threads and represent an individual virtualization of the processor (thread code, data, stack, and CPU registers).

b) Memory Management

- Handles allocation/deallocation of memory.
- Virtual memory: gives each process its own address space.
- Paging & Swapping.
- Maintains caches, buffers for performance.

c) File System Management

- Provides a **hierarchical directory structure**.
- Supports multiple file systems: ext4, XFS, Btrfs, FAT, NTFS.

- Uses **inodes** to store metadata.
- Everything (files, devices, sockets, processes) is treated as a **file**.

d) Device Management

- Uses **device drivers** to manage hardware.
- Provides a standard interface for apps (e.g., writing to /dev/tty for terminal).
- Devices are represented as files in /dev.

e) Networking

- Implements the full **TCP/IP stack**.
- Allows communication between processes, systems, and the internet.
- Supports firewalls (iptables, nftables).

User Space Components

a) Shell

- Interface between user and kernel.
- Types: Bourne shell (sh), Bourne Again shell (bash), Korn shell (ksh), Z shell (zsh).
- Accepts user commands and passes them to kernel via system calls.

b) System Utilities

- Basic tools: cp, mv, ls, grep, chmod, etc.
- Manage files, users, processes, networking.

c) Application Programs

- Software built on top of Linux (e.g., LibreOffice, Firefox, Python, Java).
- Runs in **user mode** (less privileged).

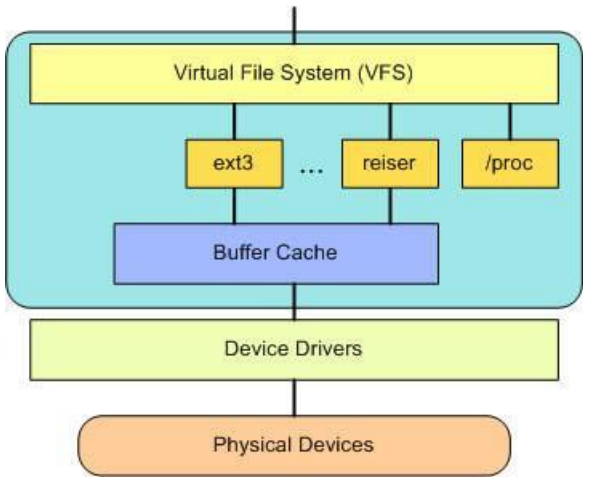
Modes of Operation

- **User Mode** → Applications run with limited access (can't touch hardware directly).
- **Kernel Mode** → Kernel runs with full access to hardware.
- System calls switch execution from user mode → kernel mode → back to user mode.

Linux Philosophy

- "Everything is a file" (devices, processes, configs).
- "Do one thing and do it well" (small tools combined with pipes).
- "Open-source, community-driven development."

Virtual File System:

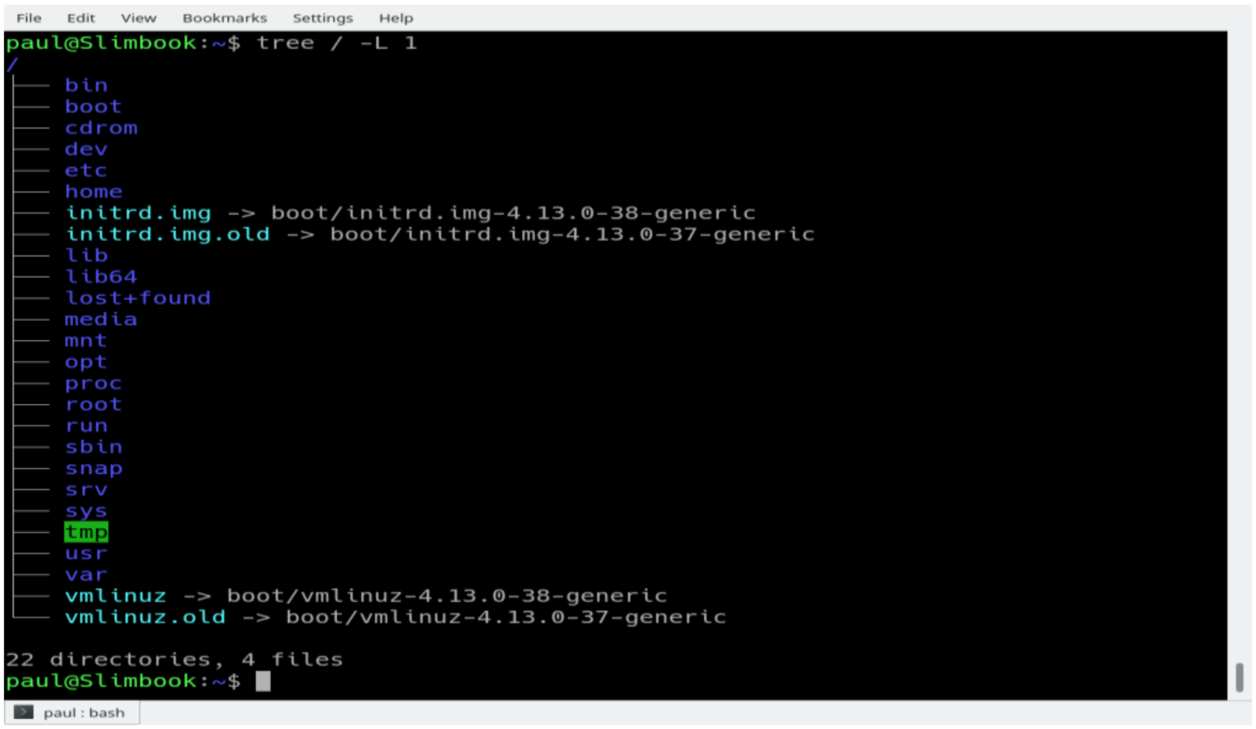


• Provides a common interface abstraction for file systems.

• At the top of the VFS is a common API abstraction of functions such as open, close, read, and write.

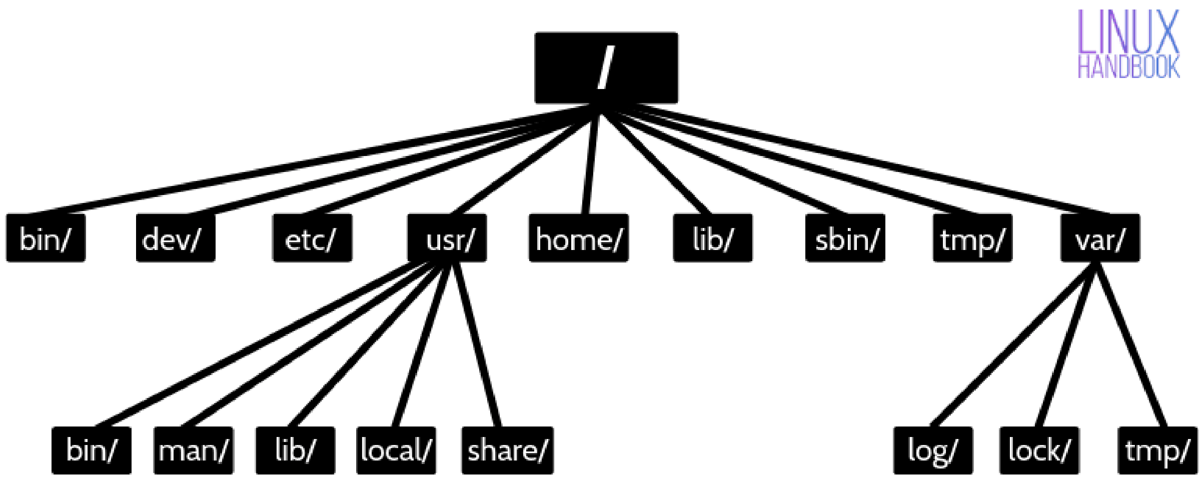
• At the bottom of the VFS are the file system abstractions that define how the upper layer functions are implemented. These are plug-ins for the given file system (of which over 50 exist). You can find the file system sources in ./linux/fs.

Linux File System:



```
paul@Slimbook:~$ tree / -L 1
/
├── bin
├── boot
├── cdrom
├── dev
├── etc
├── home
├── initrd.img -> boot/initrd.img-4.13.0-38-generic
├── initrd.img.old -> boot/initrd.img-4.13.0-37-generic
├── lib
├── lib64
├── lost+found
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin
├── snap
├── srv
├── sys
├── tmp
├── usr
├── var
├── vmlinuz -> boot/vmlinuz-4.13.0-38-generic
└── vmlinuz.old -> boot/vmlinuz-4.13.0-37-generic

22 directories, 4 files
paul@Slimbook:~$
```



/bin is the directory that contains binaries, that is, some of the applications and programs you can run.

• /boot directory contains files required for starting your system.

• /dev contains device files (plug in a webcam etc.)

• /etc: initially a dumping ground for sysadmins – now contains config files mostly (name, pwd, mount points etc.)

• /home: where you will find users' personal directories.

• /lib is where libraries live. Libraries are files containing code that your applications can use.

• /media directory is where external storage will be automatically mounted when you plug it in and try to access it.

• /mnt directory, however, is a bit of remnant from days gone by. This is where you would manually mount storage devices or partitions. It is not used very often nowadays.

• /opt directory is often where software you compile (that is, you build yourself from source code) sometimes lands (and in /usr/local or /usr/local/bin)

• /proc, like /dev is a virtual directory. It contains information about your computer, such as information about your CPU and the kernel your Linux system is running.

• /root is the home directory of the superuser (also known as the "Administrator") of the system.

• /run is another new directory. System processes use it to store temporary data

• /sbin is similar to /bin, but it contains applications that only the superuser (hence the initials) will need.

• /usr directory was where users' home directories were originally kept back in the early days of UNIX. However, now /home is where users kept their stuff as we saw above. These days, /usr contains a mish-mash of directories

• /srv directory contains data for servers.

• /sys is another virtual directory like /proc and /dev and also contains information from devices connected to your computer.

• /tmp contains temporary files,

• /var was originally given its name because its contents were deemed variable, in that it changed frequently – nowadays logs

User Groups and Permissions

Linux is a **multiuser operating system**. This means:

- Multiple people can use at same time (through terminals for example).
- Resources are protected using a **permissions system**.

Users in Linux

Every person (or process) using Linux is assigned a **user account**.

Types of Users

1. **Root User (Administrator)**
    - Username: root
    - Full privileges: install software, change configurations, access all files.
    - **Superuser**
    - Very powerful but dangerous
2. **Normal Users**
    - Created by the administrator.
    - Each has a **home directory** (e.g., /home/babar).
    - Limited access (cannot modify system files).
3. **System Users**
    - Created automatically for running services.
    - Examples: www-data (for web server), mail, daemon.
    - They do not log in like normal users.

User Identification

- Each user is assigned:
    - **Username** (e.g., tariq)
    - **UID (User ID)** (unique number, e.g., 1001)
    - **Default group (GID)**

- Stored in /etc/passwd.

Groups in Linux

A **group** is a collection of users.

Purpose: make permissions easier by managing sets of users together.

Types of Groups

1. **Primary Group**
    - Assigned when the user is created.
    - Every file the user creates belongs to this group by default.
2. **Secondary Groups**
    - Additional groups that a user can be a member of.
    - Example: a user may belong to the developers group as well as docker group.

Group Identification

- Each group has:
    - **Group name** (e.g., students)
    - **GID (Group ID)** (unique number, e.g., 1002)
- Stored in /etc/group.

File Ownership

Each file/directory in Linux has **two owners**:

1. **User (owner)** → who created the file.
2. **Group** → the group associated with the file.

Linux Permissions Model

Permissions define **what actions are allowed** on a file/directory.

Three Types of Permissions

1. **Read (r)**
   - File: can view contents.
   - Directory: can list files inside.
2. **Write (w)**
   - File: can modify or delete contents.
   - Directory: can create or remove files.
3. **Execute (x)**
   - File: can run/execute the file (if it's a program/script).
   - Directory: can enter (cd) into it.

Permission Categories

Each file has **three categories of permissions**:

1. **Owner (User)** → the file creator.
2. **Group** → other users in the same group.
3. **Others (World)** → everyone else.

Example

-rwxr-xr--

Breakdown:

- - → regular file (d = directory, l = link, etc.)
- rwx → Owner has **read, write, execute**.
- r-x → Group has **read and execute** only.
- r-- → Others have **read only**.

Numeric (Octal) Representation

Permissions can also be represented as numbers:

- **r = 4, w = 2, x = 1**.

Example:

- rwxr-xr-- = 755
  - Owner: 7 (4+2+1 = rwx)
  - Group: 5 (4+1 = r-x)
  - Others: 4 (4 = r--)

Common values:

- 777 → full access (rwx for all).
- 755 → owner full, group/others read+execute.
- 644 → owner read+write, group/others read only.

Special Permissions

Apart from the basic rwx, Linux has **three special bits**:

1. **SUID (Set User ID)**
   - Applied to executable files.
   - Program runs with the file owner's privileges.
   - Example: /usr/bin/passwd runs with root privileges.
2. **SGID (Set Group ID)**
   - Applied to executables: program runs with the group's privileges.
   - Applied to directories: files created inside inherit the directory's group.
3. **Sticky Bit**
   - Applied to directories (e.g., /tmp).
   - Users can only delete **their own files**, even if others have write permission.

Practical Example

Suppose a file has this:

-rw-r----- 1 babar students 1200 Sep 8 report.txt

- Owner: **babar**→ can read & write.
- Group: **students** → can read only.
- Others: no access.

Linux Shells

A **shell** is a program that acts as an **interface** between the user and the Linux kernel. It accepts **commands** from the user → translates them into **system calls** → kernel executes them.

Two types:

- **Command Line Interface (CLI)** shells
- **Graphical shells** (GNOME, KDE, etc.)

Types of Shells in Linux

Different shells exist, each with its own features:

1. **sh (Bourne Shell)**
   - Original Unix shell by Stephen Bourne (1977).
   - Very portable, simple scripting.
2. **bash (Bourne Again Shell)**
   - Most common in Linux.
   - Enhances sh with command history, tab completion, scripting features.
3. **csh (C Shell)**
   - Syntax similar to C programming.
   - Introduced features like aliases and command history.
4. **ksh (Korn Shell)**

  - Combination of Bourne shell + C shell features.
  - Popular for scripting in enterprises.
5. **zsh (Z Shell)**
  - Advanced shell with better completion, customization, and plugins.
  - Used in modern setups (macOS default).

Responsibilities of a Shell

- Command execution (ls, cat, etc.).
- I/O redirection (>, <, >>, 2>).
- Pipelines (|).
- Variable and environment management (PATH, HOME).
- Scripting (automating tasks).

Types of Linux File Systems

1. **ext2 (Second Extended File System)**
   - Early Linux FS, no journaling.
2. **ext3**
   - Added journaling (log system that gives faster recovery after crash).
3. **ext4**
   - Default in many distros.
   - Supports large files, backward compatible with ext2/3.
4. **XFS**
   - High-performance, scalable FS (good for servers).
5. **Btrfs (B-tree File System)**
   - Modern FS with snapshots, checksums, RAID support.
6. **Other Supported FS**
   - FAT32, NTFS (Windows compatibility), ISO9660 (CD/DVD).

Key Concepts

- **Inodes:**
    - Data structure storing metadata (permissions, owner, size, timestamps).
    - Does **not** store filename; directory maps names to inode numbers.
- **Mounting:**
    - Process of attaching a filesystem to the directory tree.
    - Example: USB drive mounted at /media/usb.