

Day 1 – Phase 1: System Update & Directory Setup

Tasks:

- Refresh package lists and upgrade the system.

```
salma2002@MSI:~$ sudo apt upgrade
[sudo] password for salma2002:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following package was automatically installed and is no longer required:
  libllvm17t64
Use 'sudo apt autoremove' to remove it.
The following upgrades have been deferred due to phasing:
  base-files libegl-mesa0 libgbm-dev libgbm1
  libgl1-amber-dri libgl1-mesa-dri libglx-mesa0
  mesa-libgallium mesa-vulkan-drivers motd-news-config
  snapd wsl-setup
0 upgraded, 0 newly installed, 0 to remove and 12 not upgraded.
```

- Verify system details: kernel version, user, time.

```
0 upgraded, 0 newly installed, 0 to remove and 12 not upgraded.
salma2002@MSI:~$ uname -a
Linux MSI 6.6.87.2-microsoft-standard-WSL2 #1 SMP PREEMPT_DYNAMIC Thu Jun  5 18:30:46 UTC 2025 x86_64 x86_64 x86_64 GNU/Linux
```

- Create /home//iot_logger with subdirectories: logs, scripts, data.

```
salma2002@MSI:~$ cd /home
salma2002@MSI:/home$ ls
salma  salma2  salma2002
salma2002@MSI:/home$ cd /salma2002 && mkdir iot_logger
-bash: cd: /salma2002: No such file or directory
salma2002@MSI:/home$ cd salma2002 && mkdir iot_logger
salma2002@MSI:~$ cd /home/salma2002 && mkdir iot_logger
mkdir: cannot create directory 'iot_logger': File exists
salma2002@MSI:~$ cd /home/salma2002
salma2002@MSI:~$ ls
Desktop  Documents  Downloads  iot_logger  yes  yes.pub
salma2002@MSI:~$ cd iot_logger && mkdir logs scripts data
salma2002@MSI:~/iot_logger$ ls
data  logs  scripts
salma2002@MSI:~/iot_logger$
```

Open-Ended Questions:

- Draw or describe the Linux architecture layers (hardware → kernel → shell → user space). Where do system calls fit?

Linux Architecture Layers

Linux is structured into several layers that work together, starting from the physical hardware to the applications you interact with:

1. Hardware

At the base of the Linux system is the hardware layer, which includes all physical components of a computer such as the CPU, RAM, hard disk, motherboard, network devices, and peripherals. This layer provides the foundation for all higher layers.

2. Kernel

The kernel sits directly above the hardware. It is the core component of the Linux operating system, managing critical tasks like process scheduling, memory allocation, disk operations, and hardware communication.

- Linux uses a monolithic kernel, meaning it bundles device drivers, file system management, and system calls into one large binary.
- System calls act as the interface between user applications and the kernel, enabling programs to request services such as file access, process control, or network communication.

3. Shell

The shell is a user interface that enables interaction with the kernel. Typically, this takes place through a command-line interface (CLI).

- Users enter commands, which the shell interprets and passes to the kernel.
- Popular shells include Bash, C Shell (csh), and Z Shell (zsh), each offering different syntax and features.

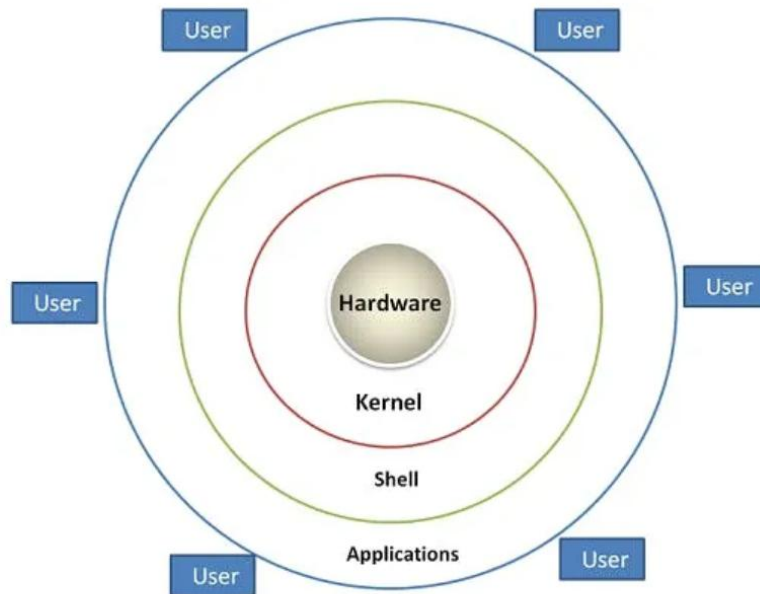
4. Applications

At the top layer are applications — the software tools and programs that users directly work with, such as text editors, browsers, or file managers.

- Applications depend on the kernel for hardware access and often use the shell for command execution.
- For example, running a command to open a file involves the shell parsing the command, the kernel retrieving data from storage, and the application displaying it to the user.

Summary: Together, the hardware, kernel, shell, and applications create a layered structure where system calls serve as the critical bridge between user space and the kernel.

Reference: [*Medium: Linux Architecture*](#)



- Explain the purpose of these directories: `/`, `/bin`, `/sbin`, `/usr`, `/etc`, `/var`.

Linux uses a hierarchical file system where each directory serves a distinct role in system operation and organization. Below are the purposes of important directories:

- **/ (Root Directory):**
The top-level directory from which the entire Linux file system hierarchy starts. Every other directory and file branches from here.
- **/bin:**
Contains essential binary executables needed for basic system operations. These commands (like `ls`, `cp`, `mv`) are required for both normal users and during system startup.
- **/sbin:**
Stores system binaries mainly used for administrative tasks and system management (e.g., `ifconfig`, `reboot`). These tools are typically executed by the root user.
- **/usr:**
Holds user-related programs, utilities, libraries, and documentation that are not essential for booting but are necessary for day-to-day operations.

- **/etc:**
Contains configuration files for system and application settings. These are usually plain text files, often with a .conf extension, that define how services and software behave.
- **/var:**
Designed for files that change frequently during normal system use. Examples include system logs, cache data, spool files, and temporary data generated by applications.
- **/root:**
The home directory of the root (superuser) account, separate from the general /home directory used for regular users.
- **/mnt:**
A temporary mount point used for attaching external file systems, such as USB drives, network shares, or other storage devices.

Reference: [Baeldung: Linux Directories](#)

• **Why does Linux treat everything as a file? Explain the difference between a program and a process.**

Linux follows the philosophy that everything is represented as a file, whether it's data on disk, a hardware device, or even a network socket. This design choice provides several benefits:

- **Convenience:** Programs can interact with hardware in the same way they interact with files. For example, sending commands to a mouse or keyboard can be done through simple read/write operations. Permissions can also be applied consistently, just as they are with regular files.
- **Unification:** Instead of having different mechanisms for handling files, devices, sockets, or processes, Linux uses a single unified interface — the file abstraction.
- **Consistency:** Regardless of what you are accessing (a hard disk, USB, keyboard, or memory info), the same basic operations apply: open, read, write, and close.
- **Flexibility:** Because everything is treated like a file, it becomes easy to connect different system components. For example, input from a device can be redirected directly into another program using simple file-like operations.

Reference: [Stack Overflow – Why Everything is a File](#)

Difference Between a Program and a Process

1. Program

- A program is a set of instructions stored on disk, usually in the form of an executable file.
- It is a passive entity — simply code waiting to be run.
- Example: `/bin/ls` is a program that lists directory contents.

2. Process

- A process is an active instance of a program in execution.
- It has its own memory space, CPU state, environment variables, and open file descriptors.
- Example: When you type `ls` in the terminal, the kernel loads the `/bin/ls` program into memory and runs it as a process.

Key Distinction:

- A program is static (code on disk).
- A process is dynamic (a running execution of that code).