1. What an OS is

- The OS is a **middle layer** between you/your apps and the hardware.
- It **manages resources**: CPU, RAM, disk, network, and devices.
- It gives you a **place to work** (desktop, terminal), plus **security** so apps cannot easily break or spy on each other.
- It **hides hardware details**, so apps just say “open file”, “send data”, etc., without knowing how the hardware really does it.

2. Drivers, bare metal, and hardware

- Modern OSs must run on **many kinds of hardware** (different CPUs, GPUs, Wi-Fi cards, sound cards).
- **Device drivers** are small programs that know how to talk to one specific device (printer, GPU, etc.), so the kernel does not need to know every device in detail.
- In Linux, 60% of the kernel code is drivers because there are so many devices.
- Long ago, some PCs ran programs **directly on hardware** (“bare metal”), without a full OS; this is possible but becomes impractical for complex systems like laptops and desktops.

3. POSIX and families of OS

- **POSIX** is a **standard rulebook** for Unix-like systems so that the same program can run on different Unix-style OSs with few changes.
- It has:
  - A **core** part for basic things (files, processes, signals, shell tools).
  - A **real-time** part (often called 1b) for time-critical systems.
  - A **threads** part (often called 1c) for portable multithreading APIs.
- `nix` means **Unix-like** systems: Unix, Linux, FreeBSD, Solaris, macOS (which comes from BSD).
- Windows is **not** Unix, but it still has similar concepts (kernel, processes, memory, files).

4. Short OS history and sizes

- Early operating systems appeared in the **1950s**, mainly on big mainframe computers, to let many jobs and users share the same machine.
- Over time they added **multi-user** and **multitasking** so several users and many programs can run safely together.
- OS size examples (just to feel the scale):
  - **FreeRTOS** (tiny embedded OS): 9k lines of code.
  - Old **Unix** (early 1980s): 20K lines.

    - **Linux kernel today**: 30M lines.
    - Full systems like **Ubuntu** or **Windows 10**: 50M lines of code overall.

5. Debian, Ubuntu, and Windows

- **Debian** (started 1993) is known for being **very stable and secure**.
- Debian has 3 branches:
    - **stable**: very tested, safer, fewer changes.
    - **testing**: next stable, newer but can have some issues.
    - **unstable**: where new packages first land, can break more often.
- One of Debian’s future versions is named **Trixie**, and Debian is the base for many other systems: **Ubuntu**, **Linux Mint**, **Kali**, etc.
- **Ubuntu**:
    - Is built on Debian.
    - Releases twice a year, with **LTS (long term support)** versions every 2 years.
    - Uses the **GNOME** desktop by default.
    - Aims to be more **user-friendly, feature-rich** and “ready for the desktop” than pure Debian.
- **Windows** started as a **16-bit GUI shell on top of MS-DOS**.
- Later Microsoft created **Windows NT**, a newer 32-bit (and then 64-bit) design with a different, more modern kernel.
- **Windows ME** was the last of the old DOS-based line; **NT 3.1** was the early version of the modern NT family.

6. Kernel and user space (and GNU/Linux)

- The **kernel** is the **core** of the OS, like the engine of a car:
    - Talks to hardware.
    - Manages CPU time, memory, devices, and files at a low level.
- **User space** is everything on top:
    - Shell, desktop environment, tools, applications.
- The border is not always perfectly clear in all systems, but the idea is: **kernel = core power**, **user space = normal programs**.
- The **GNU project** (from 1983, Richard Stallman) created many user-space tools (compiler, shell, basic utilities).
- **Linus Torvalds** created the **Linux kernel** in 1991.
- Together, GNU tools + Linux kernel make what is often called **GNU/Linux** (most people just say “Linux”).

7. Processes, threads, CPU, and memory

- A **process** is a running program (for example, your browser).

- A process can have **multiple threads**, which are smaller “paths of execution” inside it (for example, different tabs or background tasks).
- The OS:
  - Creates and destroys processes.
  - Decides which process/thread **uses the CPU next** (scheduling).
  - **Switches** between them very fast (context switches) so many programs appear to run at the same time (multitasking).
- **Memory management** (RAM):
  - RAM is limited, so the kernel gives and takes memory for each process (allocation/deallocation).
  - The **MMU** hardware plus the kernel gives each process its own **virtual address space**, so processes cannot easily overwrite each other.
  - When RAM is full, the OS can move some less-used data to disk (**paging** / **swap**) to keep the system running.

8. Storage, disks, and file systems

- The kernel manages **disks and SSDs** as **block devices**.
- It understands:
  - **Partition tables** (how the disk is split into parts).
  - **RAID** (using several disks together for speed or safety).
  - **Mounting/unmounting** file systems so they appear as folders in a single tree.
- Different **file systems** exist (FAT, EXT, NTFS, etc.), each with its own on-disk format.
- The kernel can load modules/drivers for each file system type, but offers one **Virtual File System (VFS)** interface so programs just use generic “open/read/write file” calls and don’t care about the underlying type.