

# 1. Interrupts and syscalls

- The OS is **interrupt-driven**: it mostly waits, then reacts when something happens (key press, network packet, timer tick).
- **User** → **kernel**: programs call special functions called **system calls** (syscalls) to ask the kernel for things (read file, create process, etc.).
- **Kernel** → **user**: when the kernel is done, it **returns** back to the program.
- **Interrupts** can come from:
  - **Hardware** (e.g. keyboard, disk, network card).
  - An internal **timer** that fires regularly (often measured by “HZ” and “jiffies”).

# 2. Security: memory and permissions

- The kernel controls memory access with the **MMU** (Memory Management Unit).
- The MMU maps **virtual addresses** (what programs see) to **physical addresses** (real RAM) and can mark regions as read / write / execute.
- The OS protects files and processes with:
  - File **permissions**, **owners**, and **ACLs** (Access Control Lists).
  - Extra security layers like **SELinux** that define detailed rules about what can talk to what.

# 3. Users, groups, and privileges

- The system can have **many users** at once (multi-user):
  - Real people.
  - System users (background services, daemons).
- In Unix-like systems:
  - Each user has a **uid** (user ID).
  - Each group has a **gid** (group ID).
- In Windows, similar ideas exist with **SIDs** (security IDs).
- Permissions depend on:
  - **Who** you are (uid).
  - **What groups** you belong to (gid).
- Special bits like **setuid** and **setgid** let a program temporarily run with more privileges (for example, run as root even if started by a normal user).

# 4. Types of OS environments

- **Desktop/Laptop**:
  - Windows, macOS, Linux.
  - Focus on **interactive use** with a graphical interface (GUI).

- **Server / Cloud:**
  - Mostly Linux.
  - Often no GUI; used for **batch processing**, web services, databases.
- **Mobile:**
  - Android, iOS.
  - Designed for touch, battery saving, mobile radios, app stores.
- **Embedded:**
  - FreeRTOS and similar tiny OSs.
  - Often **no GUI**, running on small chips in devices (routers, microwaves, cars).

## 5. Virtualisation and cloud

- **Virtualisation** lets you run one or more **guest OSs** inside a **host** using a **hypervisor**:
  - Example: Linux host, several virtual Windows and Linux guests.
  - Lets different OSs share the same physical machine.
- It can use:
  - Full **emulation** (pretend hardware).
  - **Paravirtualisation** (guest knows it's virtual and cooperates for speed).
- **Cloud computing** builds on this idea:
  - **Private cloud**: inside a company.
  - **Public cloud**: services like AWS, Azure, GCP.
- Cloud service layers:
  - **IaaS**: Infrastructure as a Service – you get virtual hardware (VMs, disks, networks).
  - **PaaS**: Platform as a Service – hardware + runtime stack (databases, runtimes, etc.).
  - **SaaS**: Software as a Service – you just use the application (e.g. web mail).
- Main benefits: **flexibility** (spin things up/down) and **scalability** (handle more users by adding resources).

## 6. Embedded real-time systems

- Embedded real-time systems run on **small hardware** with limited CPU, RAM, and flash.
- They are often **deterministic**: operations must finish within strict **deadlines**.
  - **Hard real-time**: missing a deadline is unacceptable (e.g. brakes in a car).
- They use OSs with a **small footprint** and only the features they truly need.

## 7. Open vs closed, free vs open source

- **Closed-source OS** (like Windows):
  - Source code is **not public**.
  - You cannot easily study or change the internals.

- **Open-source OS** (like Linux, FreeRTOS):
  - Source code is public; you can **read, modify, and share** within license rules.
  - Great for learning how an OS works.
- This course focuses on **Linux** because of that openness.

## “Free software” vs “open source”

- **Free software** (FSF sense) means **freedom**, not price: “free as in freedom, not beer”.
- **GPL**:
  - Ensures users keep freedoms to use, study, modify, and share.
  - Requires that derived programs also share their source (copyleft).
  - Critics say this can **restrict developers** who want to keep changes closed.
- **Permissive licenses** (MIT, Apache, BSD):
  - Allow reuse in almost any project, including closed-source.
  - Fewer obligations, more developer freedom, but less guarantee that improvements come back.

## FOSS pros and cons

- **Advantages**:
  - Transparency (you can inspect code).
  - Community development and review.
  - No single vendor lock-in.
- **Drawbacks**:
  - **Forking** can cause fragmentation (many similar but slightly different versions).
  - Volunteers may prefer **fun parts** over boring maintenance.
  - Docs can be **missing or outdated**.
  - Funding often depends on **vested interests** or sponsors.

## 8. GPL, LGPL, and linking

- With the **GPL**, when you create a **derived work** (e.g. link your code to a GPL library or modify GPL code), you must:
  - Keep the **GPL license**.
  - Release the **source code** of the derived work to users.
- **Copyright** stays with the original authors, but your derivative must respect GPL rules.
- **LGPL** is more relaxed for libraries:
  - You can link against LGPL libraries in your program **without** having to release your whole program’s source.
  - You still have to follow some rules for the library itself (e.g. allow relinking).

## 9. Kernel data structures and C

- Inside the kernel, code is written mainly in **C**, using **pointers** heavily.
- Common data structures:
  - **Linked lists**.
  - **Stacks** and **queues**.
  - **Binary trees** (including red-black trees).
  - **Hash tables**.
  - **Bitmaps** (compact way to track free/used items).
- Linux provides helper headers like `<linux/list.h>` for linked lists and `<linux/rbtree.h>` for red-black trees.

## 10. Kernel memory allocation (kmalloc)

- Kernel code can request memory dynamically using `kmalloc()` and free it with `kfree()`.
- `kmalloc()` typically uses a **slab allocator** to manage small chunks efficiently and give **physically contiguous** memory.
- For devices that need special memory (like DMA – direct memory access), there are options to allocate suitable buffers.
- If allocated memory is never freed, the kernel can suffer **memory leaks**, which slowly eat available RAM and can destabilise the system.