

Operating Systems

The layers in a computer.

- **Hardware** provides basic computing resources.
- **OS** controls and coordinate resources.
- **Applications and users.**

Bus Hierarchy

- The **processor bus** is the widest and fastest for CPU to talk to cache.
- The **memory bus** talks to memory.
- The **PCI bus** communicate with devices.
- **Bridges** forwards a single from a bus to another.

Booting

1. Bootstrap program runs when computer powers on:

A small part of the CPU/board stores instructions to tell the CPU how to

- Access the memory.
- Initialise the bus and talk to devices.

2. Kernel

3. Normal operation of a computer: communicates with memory, IO devices.

Interrupts are how devices communicates to the CPU, when an interrupt occurs:

1. Store the program counter
2. Jumps to interrupt service routine, a **interrupt vector** contains address to all ISR.
3. The CPU resumes

An **interrupt** usually happens at an **instruction boundary**.

Definition

An **exception** is a software interrupt.

Storage

Definition

A **word** is the computer's smallest native unit of data.

Storage is organised in order of speed

1. Registers
2. Cache

Note

Cache is managed transparently by the computer.

3. Main memory
4. Storage / IO devices

Note

Each device needs an IO driver to provide a uniform interface between the controller and kernel.

Definitions

- **Jitter** is the variation in latency.
- **Impedance mismatch** happens when two computers operates at different speeds
- **Caching**: high performance storage to mask the performance cost of accessing slow stuff.
- **Buffering** is a memory between two components with small differences in bandwidth.
- A **bottleneck** is the most constrained resource in system.
- A **balanced system** is where all resources are bottlenecked.

Resource Management

Resource	Description
CPU	<ul style="list-style-type: none">• Multiplexes many running programs• Taking turns until the timer hits zero, then interrupts
Memory	Prevent programs from accessing memory outside its own chunk
IO	<ul style="list-style-type: none">• Make IO instructions privileged.• For devices accessed via memory, use memory protection mechanisms.

Protection

OS evolution:

1. Open shop: programming is in machine code
2. Batch systems: runs a set of programs in batch
3. Multiprogramming/time sharing: uses scheduling so PC remains responsive even if a program is in an infinite loop.

The CPU runs in two modes (distinguished with a hardware **mode bit**)

- **Kernel mode**: can interrupt programs in user mode
- **User mode**: cannot interrupt programs in kernel mode

This is CPU enforced, x86 CPUs have ring 0 to ring 3 instead.

1. PC resets to kernel mode
2. Kernel mode runs a program in user mode, CPU is now in user mode
3. User mode program calls an OS function from kernel mode, CPU is now in kernel mode
4. Go back to step 2 once completed

Kernels

The kernel does IO for user mode apps to prevent direct access.

OS functions are accessed through **system calls**

- Invoked by putting required parameters in the write place and **trapping**

The different ways of passing parameters are:

- Load into registers
- Place onto stack
- Block in memory, address to block placed in register
- OS contains vectors that enforces code run when mode switch happens

Note

An alternative is that the OS **emulates** for the application: checks every instruction before executing in some virtualisation system.

Syscalls are **language agnostic**, standard interface to OS services.

Microkernels

Microkernels tries to make the kernel minimum: putting most services in user mode.

- Use message passing to access **servers** (in user mode)
- But message passing requires context switching and is more expensive than trapping

Many OS have servers but also include a lot of functions in kernel due to performance concerns.

Virtualisation

- **Virtual machine**
 - Encapsulates the entire OS, multiple kernels in memory
 - Almost no chance programs from different VMs can interfere with each other
- **Containers**
 - Expose functionality in a kernel so each container acts as a separate entity even though they share the same kernel.

Security

To determine the privilege of a user/app

1. Identify the user/app (or the group the user is in)
2. Identify privilege

Definitions

- **Privilege escalation** allow user to gain permission it doesn't have
- **Principle of least privilege**: permission set to limit damage if abused

The OS should isolate apps from each other.

Definitions

- **Covert channels** leak information through side effects, e.g. electromagnetic waves of a wire.
- A **domain** is a set of **access rights**, domain limits access to objects
 - In UNIX a domain is a user ID, or an app on a phone.
- **Objects** are things to be accessed.
- An **access matrix** is a set of domain against objects

Since the matrix is sparse, instead store as

- **Access control list**: index by object
- **Capabilities**: by domain

ACL checks are done in software, frequent checks are cached to improve performance.

Authentication

- Password hashed with salt
- Multifactor authentication
- Failed accesses are logged

Only the operating system can intercept *Ctrl-Alt-Del*, so you know the login screen is real.
