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 **interrup** 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.
- **Impedence 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	• **Multiplexes** many running programs • Taking turns until the timer hits zero, then interrups
Memory	Prevent programs from accessing memory outside its own chunk
IO	• Make IO instructions privilleged. • 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 privillege of a user/app

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

Definitions

- **Privillege escalation** allow user to gain permission it doesn't have
- **Principle of least privillege**: 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

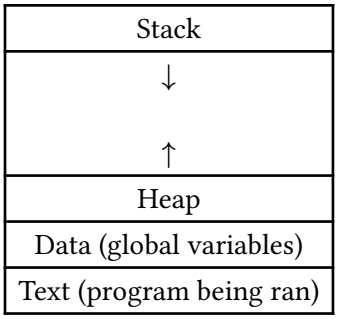
Only the operating system can intercept *Ctrl-Alt-Del*, so you know the login screen is real.

Processes

Definitions
- A **program** is a file on disk.
- A **process** is a program in execution: instances of the same program can be ran side by side.

A process is a unit of **protection** and **resource allocation**: kept isolated from each other.

- Each process executes on a **virtual memory**



- And has a one or more **threads of execution**.

  Each thread has
  - Program counter for current instruction.
  - Stack for temporary variables, parameters and return addresses.

Definition
Process control block is the data structure representing a process.
- Process ID
- Current process state (waiting/running)
- CPU scheduling information (priorities/scheduled tasks)
- Memory management info
- IO status + accounting (e.g. CPU seconds used) info

A **process context** is the machine environment while the process is running.

Definition
A **thread** represents an individual execution context.

Each thread has an associated **thread context block** with context and scheduling info, which determines which thread to run.

Process States
Context switching:
1. Save context of currently running process
2. Restores context of process being resumed

How much time this takes depends on hardware support.

The process starts at ready state
- Ready: waiting for scheduler to run it
- Running: self explainatory, interrupts while running puts the program in ready
- Waiting: waiting for IO or event
- Terminated: self explainatory

Processes can be organised into a **tree**

Method	Description
Fork (Unix)	Clones parent into a child process, then **execve** can replace the program of child.
CreateProcess (NT)	Requires the name of the program to be executed to be specified.

Processes are terminated because
- Process performed illegal operation (e.g. access memory without authorisation/executing a privileged instruction)
- Parent terminates child (termination cascades to childrens)
- Process finished execution
  - A **zombie** process is requested to be terminated but hasn't yet exit.
  - An **orphan** process remains running after the parent program exits.

Interprocess Communication

Requires between the processes:
- Syntax and semantics agreed upon
- Synchronisation: data transfer takes place according to agreed rules

Message passing: sends message to each other, mediated by the kernel

Shared memory: Establishes some part of memory both processes can access, this requires the usual memory protections to be removed.

Definition

Signals are simple messages: async notifications on a process.

E.g. SIGTERM (ignorable) and SIGKILL (unignoreable)
