CAB403
Systems Programming

Semester 2, 2025

Course Notes

Coordinator
Timothy Chappell

INDEX

1. DEFINITIONS & REVIEW

The basic unit of storage in a computer is the **bit**, which can hold a value of either `0` or `1`. All other forms of data—such as numbers, characters, images, and audio—are built from collections of bits.

A **byte** consists of **8 bits** and is the smallest convenient unit of storage in most systems. Most computers can manipulate bytes directly but may not support operations on single bits.

A less common but important unit is the word, which is the native data size of a computer's architecture. A word typically consists of one or more bytes, and many CPU operations are performed in word-sized units.

> **Example: On a 64-bit system, a word is 8 bytes (64 bits), and memory and registers are optimised for this word size.**

Unit	Abbreviation	Size in Bytes
Kilobyte	KB	1,024 bytes (2^{10})
Megabyte	MB	1,048,576 bytes (2^{20})
Gigabyte	GB	1,073,741,824 bytes (2^{30})
Terabyte	TB	1,099,511,627,776 bytes (2^{40})
Petabyte	PB	1,125,899,906,842,624 bytes (2^{50})

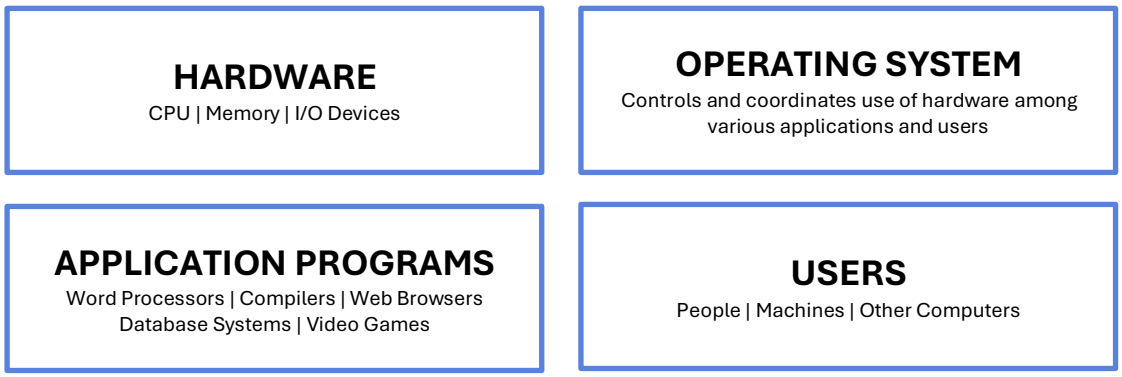
2. OPERATING SYSTEMS

An operating system acts as an intermediary between a user of a computer and the computer's hardware. Its goals are to:

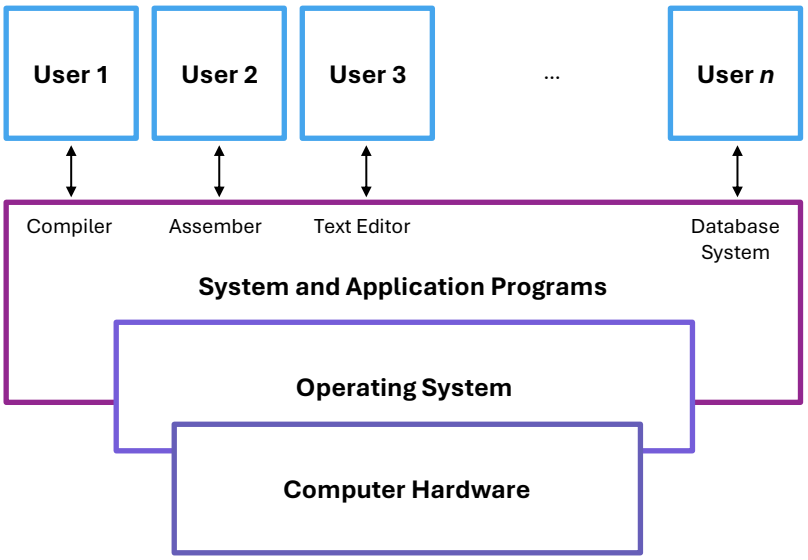
1. Execute user programs and make solving user problems easier.
2. Make the computer system convenient to use.
3. Use the computer hardware in an efficient manner.

2.1. Four Components of a Computer System

A computer system can be divided into four components:



And they have the following interactions with each other:



2.2. What Operating Systems Do

Operating systems perform a variety of functions depending on the type of system and the needs of its users. While everyday users typically value ease of use and convenience, other environments place more emphasis on efficient resource management and task optimisation.

1. **General users prefer:**
   (a) Convenience
   (b) Ease of use

(c) Little concern for how resources are allocated

2. **Shared systems (e.g. mainframes, minicomputers):**
   (a) Must keep multiple users satisfied
   (b) Prioritise fair and efficient resource utilisation
3. **Dedicated systems (e.g. workstations):**
   (a) Offer dedicated local resources
   (b) Often access shared server resources
4. **Handheld devices:**
   (a) Limited resources (CPU, memory)
   (b) Optimised for usability and battery life
5. **Embedded systems (e.g. in cars or appliances):**
   (a) Minimal or no user interface
   (b) Perform specific tasks automatically and reliably

2.3. Defining the Operating System

An operating system (**OS**) is broadly understood as both a **resource allocator** and a **control program**, though there is no single, universally accepted definition. Its core role is to manage system resources efficiently and ensure programs run safely and correctly.

The exact definition of an OS varies, but a practical approximation is:

Everything a vendor includes when delivering an operating system[1]

The **kernel** is the core of the operating system and the only program that is always running once the system is booted. It operates in a privileged mode, with full access to the hardware, and is responsible for managing the system at the most fundamental level. Its key responsibilities include:

- **Process management** – scheduling and switching between processes
- **Memory management** – allocating and freeing memory
- **Device management** – communicating with hardware via drivers
- **System calls and interrupt handling** – providing an interface between applications and hardware

All other software interacts with hardware only through the kernel, making it the essential bridge between system resources and user programs.

Other components of the operating system run on top of the kernel. These include system programs, which provide utilities and services to manage, configure, and monitor the system. Examples include terminal shells, file explorers, and background services (daemons). These tools help users interact with the OS and carry out basic tasks.

[1] Which can differ significantly between platforms (Linux vs MacOS vs Windows)

There are also application programs, which are user-installed software designed for specific purposes such as web browsing, document editing, or media playback. These run in user mode, meaning they don't have direct access to hardware and must rely on the OS and system programs to perform actions like reading files, connecting to networks, or managing memory.

2.4. Computer and OS Startup: The Bootstrap Program

When a computer is powered on or rebooted, a **bootstrap[2] program** is the first code to run. This program is responsible for getting the system ready to load the operating system.

- It is typically stored in ROM or EPROM and is considered part of the system's firmware.
- Because it is stored in non-volatile memory, it remains intact even when the computer is powered off.

The bootstrap program **performs initial hardware checks and sets up essential components before loading the operating system kernel into memory and starting its execution**. This process ensures the system is properly initialised and ready for use.

2.5. Computer System Organisation & Operation

A computer system is composed of one or more CPUs, device controllers, and shared memory, all connected via a common **bus**[3]. These components operate together to manage processing and input/output (I/O) tasks efficiently.

- Device controllers manage specific types of I/O devices (e.g. storage, keyboard, display).
- Each controller has a local buffer to temporarily hold data during transfers.
- The CPU moves data between main memory and the controllers' buffers.

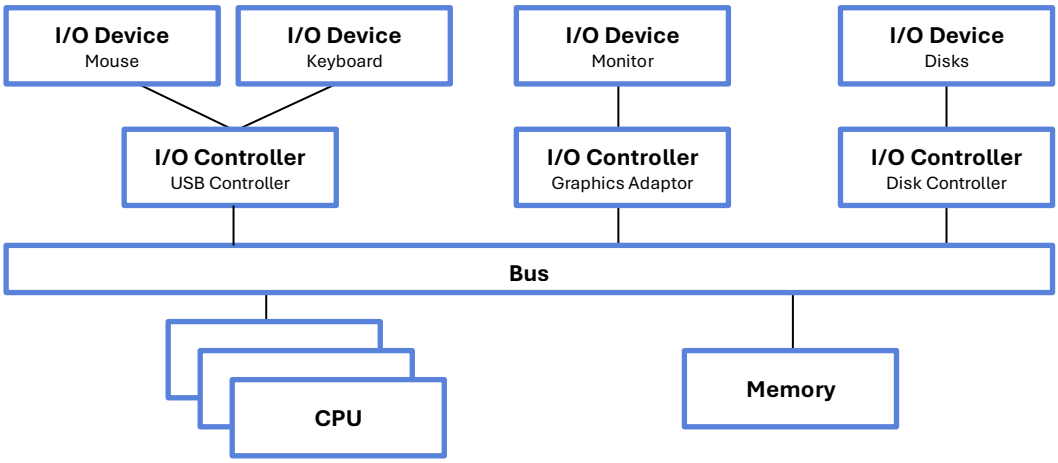


Figure 1 Computer System Organisation

CPU and I/O operations can occur concurrently, allowing the system to perform multiple tasks at once. When data needs to be read or written:

- The CPU issues a command to the relevant device controller.
- The device transfers data between the hardware and its local buffer.
- Once the operation completes, the controller sends an interrupt to notify the CPU.

[2] The bootstrap program is also sometimes called the bootloader. It may involve multiple stages, especially in modern systems where a primary loader (e.g. UEFI) hands off to a secondary bootloader (e.g. GRUB) before loading the OS kernel.

[3] Modern systems may include multiple buses—such as address, data, and control buses—for greater performance and modularity

This interrupt-driven model allows the CPU to continue other processing tasks while waiting for I/O to finish, improving system efficiency and responsiveness[4].

2.6. Interrupts & Handling Interrupts

Interrupts are a key mechanism in modern operating systems, allowing the system to respond efficiently to events. When an interrupt occurs, control is transferred to a specific piece of code known as an **interrupt service routine** (**ISR**). This transfer is typically done through an interrupt vector, which is a table containing the addresses of all available ISRs[5].

To ensure the interrupted program can resume correctly, the **interrupt architecture must save the address of the interrupted instruction**. This allows the CPU to return to its exact point of execution after the interrupt has been handled.

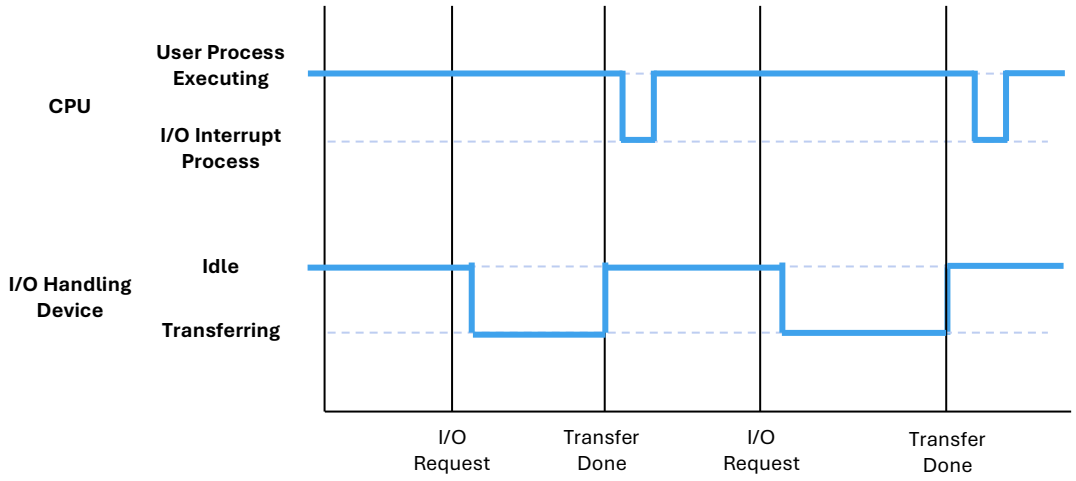
Some interrupts are generated by software rather than hardware. These are known as **traps or exceptions**, and they occur in response to errors (e.g. divide-by-zero) or user actions (e.g. system calls)[6]. Regardless of origin, modern operating systems are fundamentally interrupt driven, meaning they rely on interrupts to manage events and resources efficiently.

Interrupt Handling Process

When an interrupt occurs, the operating system must quickly respond while preserving the program's state:

1. The OS saves the current state of the CPU, including all registers and the program counter.
2. It determines the type of interrupt using one of two common methods:
   (a) **Polling** – the CPU checks each device in sequence to identify the source[7].
   (b) **Vectored interrupt system** – the interrupting device sends its own identifier, allowing immediate lookup of the correct ISR.
3. Control is passed to the appropriate interrupt handler, a code segment tailored to deal with that specific event.

After the ISR completes, the saved CPU state is restored, and execution resumes as if uninterrupted.



[4] Some architectures use Direct Memory Access (DMA) to allow certain devices to bypass the CPU and transfer data directly to memory

[5] The interrupt vector is typically stored in a fixed memory location and indexed based on the interrupt type or number

[6] Traps are synchronous software-generated interrupts that occur as part of a program's control flow, whereas hardware interrupts are asynchronous

[7] Polling is simpler to implement but less efficient than vectored systems, especially in systems with many I/O devices

2.7. I/O Structure

Input/output (I/O) operations in a computer system can be handled in several ways, depending on how control is returned to the user program. In early or simpler systems, control returns only after I/O completion, meaning the user program must wait until the operation finishes.

- This is often implemented using a wait instruction, which causes the CPU to idle until an interrupt signals that the I/O has completed.
- Alternatively, some systems use a busy-wait loop, where the CPU continually checks for I/O completion—a method that can waste CPU cycles and create contention for memory access[8].
- In such systems, only one I/O request can be active at a time, preventing simultaneous I/O processing.

In more advanced systems, control returns to the user program immediately after the I/O request starts, allowing the CPU to perform other tasks while waiting for I/O completion[9].

- The user program can issue a system call to the operating system when it needs to check or wait for I/O completion.
- The operating system uses a device status table to manage I/O devices. Each entry in this table includes:
  - » The device type
  - » The hardware address
  - » The current status or state (e.g. idle, busy, error)

When an I/O operation is initiated, the OS indexes into this table to retrieve the relevant device entry and updates it to reflect the pending operation and its associated interrupt handling routine[10].

2.8. Direct Memory Access (DMA) & Storage Structure

Direct Memory Access (**DMA**) is used by high-speed I/O devices that can transfer data nearly as fast as the system's main memory. With DMA, the device controller transfers entire blocks of data directly between its buffer and main memory, bypassing the CPU during the data transfer itself. The CPU is only involved at the start and end of the transfer, which significantly reduces overhead.

- Only one interrupt is generated per block, rather than one per byte.
- This improves efficiency for large data transfers such as disk reads or network packets.

Storage Structure

Storage in a computer system consists of several layers, each with different properties.

1. Main memory (RAM):
   - (a) The only large storage directly accessible by the CPU
   - (b) Provides random access to data
   - (c) Volatile – contents are lost when power is removed
2. Secondary storage:

[8] Busy-waiting ties up the CPU in a loop, reducing overall efficiency—especially in systems with multiple active processes.

[9] This approach enables asynchronous I/O, where multiple operations can be pending simultaneously, improving system responsiveness and throughput.

[10] The device status table acts as a central record for tracking I/O devices, helping the OS coordinate I/O requests and handle interrupts appropriately.

(a) Provides large, non-volatile capacity

(b) Slower than main memory but essential for persistent data storage

Common Secondary Storage Types

Storage Type	Characteristics
Magnetic disks	- Use rigid platters coated in magnetic material - Data is organised into tracks and sectors - Disk controller manages interaction with the system
Solid state drives (SSDs)	- Faster than magnetic disks, use flash-based technology - Non-volatile, no moving parts, lower latency - Becoming more widely adopted due to speed and reliability

Storage Hierarchy

Storage is arranged in a hierarchy, balancing speed, cost, and volatility. The closer the storage is to the CPU, the faster and more expensive it typically is, and the smaller in capacity.

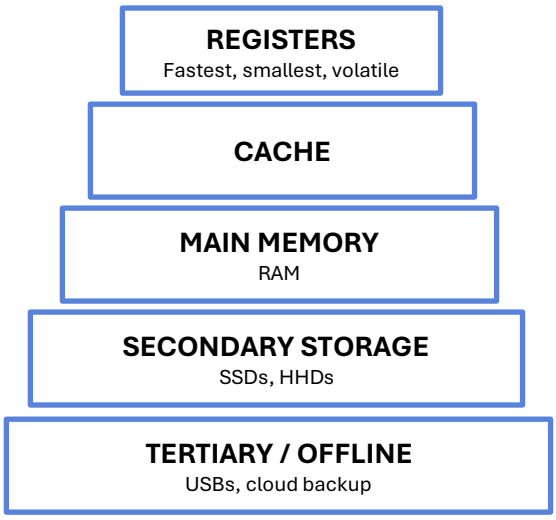


Figure 2 General Storage Hierarchy (Top to Bottom)

- **Caching**: Frequently accessed data is copied into faster storage (e.g. RAM acts as a cache for disk).
- **Device driver**: Software component that manages communication between the device controller and the kernel, offering a consistent interface regardless of hardware.

Intermediate Storage Types

Between secondary and tertiary storage lie **magnetic disks**, **optical disks**, and **magnetic tapes**, each with unique characteristics and historical importance.

- Magnetic disks (e.g. hard disk drives or HDDs) remain widely used for large-capacity storage, especially in desktops and servers.
- Optical disks (e.g. CDs, DVDs, Blu-rays) use lasers to read/write data and were once common for media distribution and backup. However, they are now rarely used in modern systems due to low capacity and slower speeds.

- Magnetic tapes offer very high capacity and reliability for long-term archival storage. While not common in consumer tech, they are still used in enterprise and government environments for backup and data preservation.

These storage types, while slower than SSDs, are still relevant in contexts where cost per byte and longevity are prioritised over speed.

Caching

Caching is a key performance optimisation technique used throughout computer systems, from hardware to the operating system level. The main idea is to temporarily copy frequently accessed information from slower storage into faster storage (the cache) to reduce access time.

- When data is needed, the cache is checked first:
  - » If the data is found (cache hit), it is used directly from the fast cache.
  - » If not (cache miss), the data is fetched from slower storage, copied into the cache, and then used.
- Caches are typically much smaller than the storage they mirror (e.g. CPU cache vs main memory, RAM vs disk).
- Caching occurs at many levels, such as:
  - » CPU caches (L1, L2, L3)
  - » Operating system page cache
  - » Browser cache and application-level caches

Efficient cache management is critical for performance. Key considerations include:

- Cache size – how much data the cache can hold
- Replacement policy – how the system decides which data to discard when the cache is full (e.g. Least Recently Used (LRU), First-In First-Out (FIFO))

Well-designed caching significantly reduces access time and improves overall system responsiveness.

2.9. Computer System Architecture

Most modern computer systems, from small devices like PDAs to large mainframes, use a single general-purpose processor. However, it is increasingly common for systems to include additional special-purpose processors, such as those for graphics, encryption, or signal processing.

Multiprocessor systems—also called parallel systems or tightly coupled systems—are becoming more widespread due to their performance and reliability benefits.

Advantages of Multiprocessor Systems

1. Increased throughput – Multiple processors can execute tasks simultaneously, improving overall performance.
2. Economy of scale – Shared resources (e.g. memory, power supply) reduce per-processor cost.
3. Increased reliability – If one processor fails, others can take over (graceful degradation or fault tolerance).

Types of Multiprocessing

Asymmetric Multiprocessing (**AMP**) is where a master processor controls the systems and others perform assigned tasks; this simplifies the system design and task allocation. **Symmetric Multiprocessing** (**SMP**), on the other hand, is when all processors are peers and share control equally. This is more common in modern systems due to better scalability and flexibility.

Von Neumann Architecture

The von Neumann architecture is a fundamental design model where a computer stores both data and instructions in the same memory space and processes them using a single control unit. This model underpins most modern computing systems, from embedded microcontrollers to full-scale general-purpose computers. While the core idea remains the same, its implementation varies depending on context—each diagram illustrates a different application of this architecture.

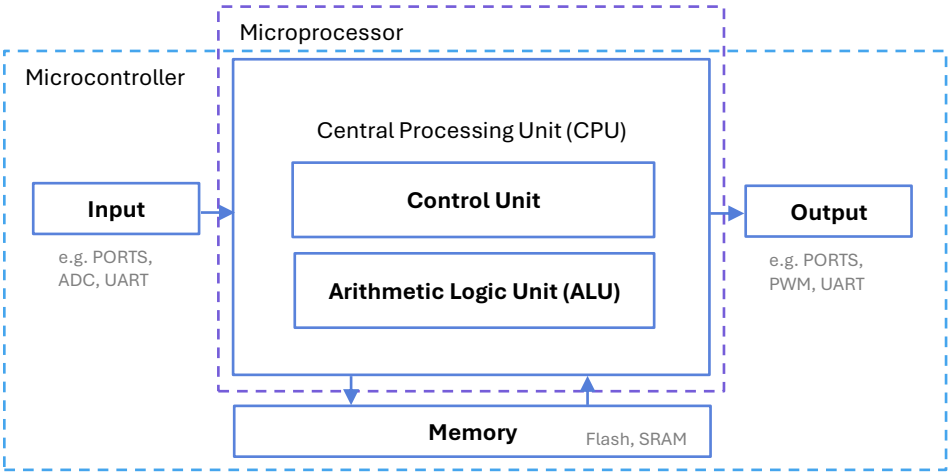


Figure 3 Embedded View – Microcontrollers and Integration

The first diagram illustrates von Neumann architecture as implemented in microcontrollers[11], commonly used in embedded systems.

- A microcontroller contains a microprocessor (CPU with Control Unit and ALU) along with on-chip memory (e.g. Flash, SRAM) and integrated I/O peripherals like ADCs or UARTs.
- This design allows all components—processing, storage, and I/O—to be tightly packed onto a single chip, making it ideal for compact, low-power applications such as IoT[12] devices or industrial automation.
- It shows how von Neumann's shared memory model can be physically realised in small-scale, efficient hardware.

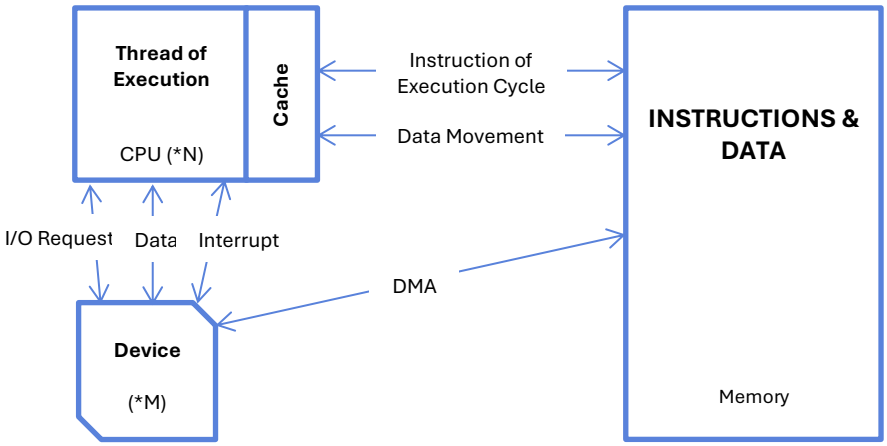


Figure 4 System-Level View – General-Purpose Computers with DMA

[11] From *CAB202 Microprocessors & Digital Systems*

[12] Internet of Things

The second diagram shifts to a system-level perspective, focusing on how general-purpose computers operate using the von Neumann model.

- It includes a CPU and cache interacting with main memory, with instructions and data stored together.
- External I/O devices communicate via interrupts, I/O requests, or Direct Memory Access (DMA)—a method that allows data to be transferred directly between devices and memory, bypassing the CPU for efficiency.
- This diagram is better suited to understanding data flow, instruction cycles, and performance considerations in PCs, servers, and operating systems.

Symmetric Multiprocessing Architecture & Dual Core Design

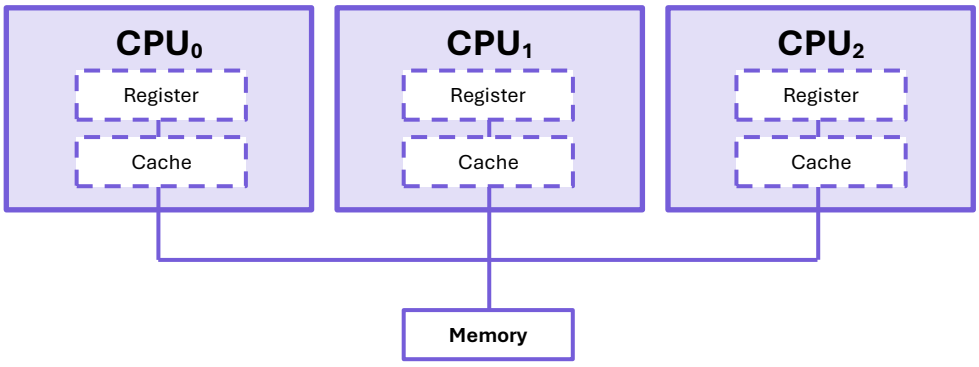


Figure 5 Separate CPUs in a Multiprocessor System

Many modern processors feature a **dual-core** or **multicore** design, where multiple processing cores are integrated on a single chip. This allows parallel execution within the same processor, improving efficiency and performance without increasing physical space or power consumption.

Architectural variations affect how processors access memory:

- **Uniform Memory Access (UMA):** All processors share the same memory and have equal access time. This is common in smaller multiprocessor systems.
- **Non-Uniform Memory Access (NUMA):** Each processor has its own local memory, which it accesses faster than non-local memory. This model is used in larger systems to improve scalability and reduce memory latency.[13]

Larger-scale systems may include blade servers, where a single chassis houses multiple independent systems, each with its own CPU, memory, and I/O. These are commonly used in enterprise environments for high-density computing.

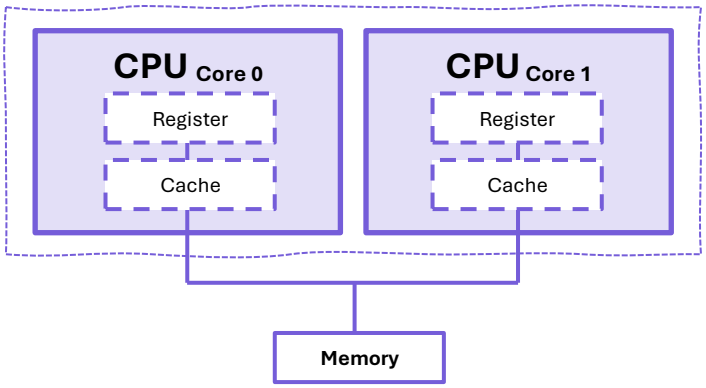


Figure 6 Dual-Core Processor (Multicore)

[13] In NUMA systems, software and the operating system must be aware of memory locality to fully optimise performance.

Blade servers are a type of modular server architecture where multiple independent server units (blades) are installed vertically into a single shared chassis. Each blade functions as a fully independent computer, with its own CPU, memory, and storage. The chassis provides shared power, cooling, and sometimes networking.

Clustered Systems

Clustered systems consist of **multiple independent computers (nodes)** working together to perform tasks, often appearing as a single system to users. Unlike multiprocessor systems—which have multiple CPUs within one machine—clustered systems involve multiple machines connected through a network, usually sharing storage via a Storage Area Network (**SAN**)[14].

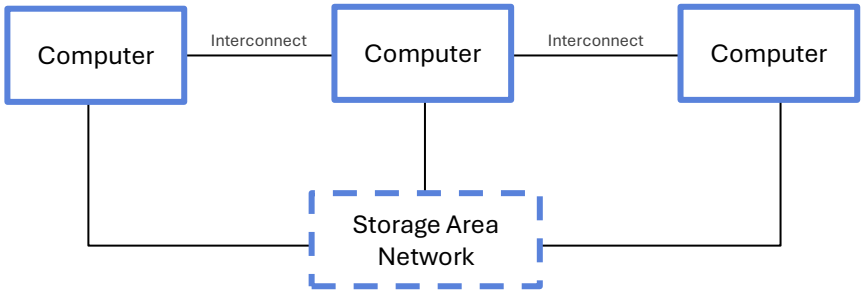


Figure 7 Clustered System

Clustered systems are commonly used to provide high availability. If one node fails, others can take over, minimising downtime:

- **Asymmetric clustering**: One node is active while another remains in hot standby, ready to take over if the active node fails.
- **Symmetric clustering**: Multiple nodes actively run applications and monitor each other, offering both load balancing and failover support.

Clusters are also used for high-performance computing (**HPC**), where tasks are split across nodes and processed in parallel. In such cases, applications must be written to support parallelisation to fully utilise the cluster.

To manage shared resources safely, some clustered systems use a Distributed Lock Manager (**DLM**)[15] to prevent conflicts when multiple nodes try to access the same resource.

2.10. Operating System Structure & Operations

Modern operating systems are designed to manage resources efficiently and ensure responsiveness for users and applications. Two key structural approaches—**multiprogramming** and **timesharing**—support these goals.

Multiprogramming and Timesharing

A single user cannot keep the CPU and I/O devices busy all the time. Multiprogramming addresses this by keeping multiple jobs (code and data) in memory at one. When one job is waiting (e.g. for I/O), the CPU switches to another, ensuring that system resources are always used.

- A **subset** of jobs is kept in memory, and **job scheduling** determines which runs next.

[14] A Storage Area Network (SAN) is a high-speed network that provides shared access to block-level storage, often used to enable all cluster nodes to read/write from the same storage pool.

[15] A Distributed Lock Manager ensures that only one node can access or modify a shared resource at a time, preventing data corruption and ensuring consistency across the cluster.

- **Timesharing** or (**multitasking**) extends this model by switching between jobs rapidly enough that users can interact with each one in real time.
- This requires fast context switching and aims for response times under 1 second.
- Each user has **at least one active process** in memory; if many jobs are ready at once, the OS uses **CPU scheduling** to manage execution order.
- When processes exceed available memory, the OS may perform **swapping** to move them in and out of memory.
- Virtual memory enables execution of processes even when they are not fully loaded into physical memory, improving flexibility and system utilisation[16].

OS Operation Modes and Interrupt Handling

Operating systems are **interrupt-driven**, responding to signals from **hardware** or **software traps**. Hardware interrupts might come from I/O devices, while software traps can result from errors (e.g. divide by zero) or system service requests.

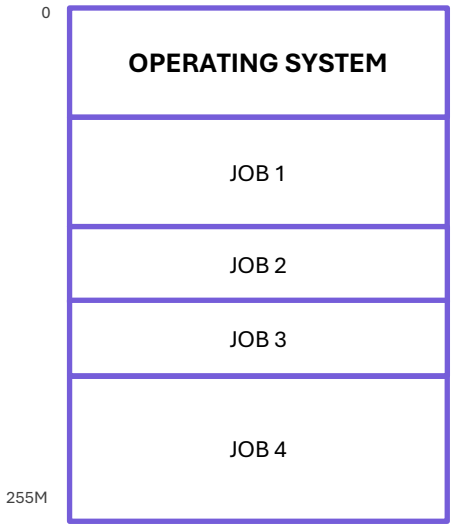


Figure 8 Memory Layout for Multiprogrammed System

- To protect itself and other components, the OS relies on **dual-mode operation:**
  - » **User Mode:** Normal application execution
  - » **Kernel Mode:** Privileged operations with full system access.
- A **mode bit**[17] provided by hardware distinguishes these modes.
- **Privileged instructions** (e.g. accessing I/O, modifying memory maps) can only be executed in **kernel mode**[18]**.**
- When a **system call** is made (e.g. to open a file), the OS switches from user to kernel mode; when the call completes, it returns to user mode.

[16] Virtual memory creates the illusion of a large, continuous memory space by using disk storage to extend physical RAM. It allows for larger and more flexible multitasking.

[17] The mode bit is a hardware-supported flag that indicates the current execution mode of the CPU—user mode or kernel mode. When set to user mode, the system restricts access to critical instructions and resources. When switched to kernel mode (e.g. during a system call or interrupt), the OS gains full access to protected operations. This mechanism is essential for enforcing separation between user-level and system-level code.

[18] This protection prevents user programs from directly interfering with the OS or other programs, ensuring system stability and security.

Modern CPUs may support additional modes, such as **virtual machine manager (VMM)** mode, which enables the safe execution of guest operating systems within a host system.

Transitioning from User to Kernel Mode

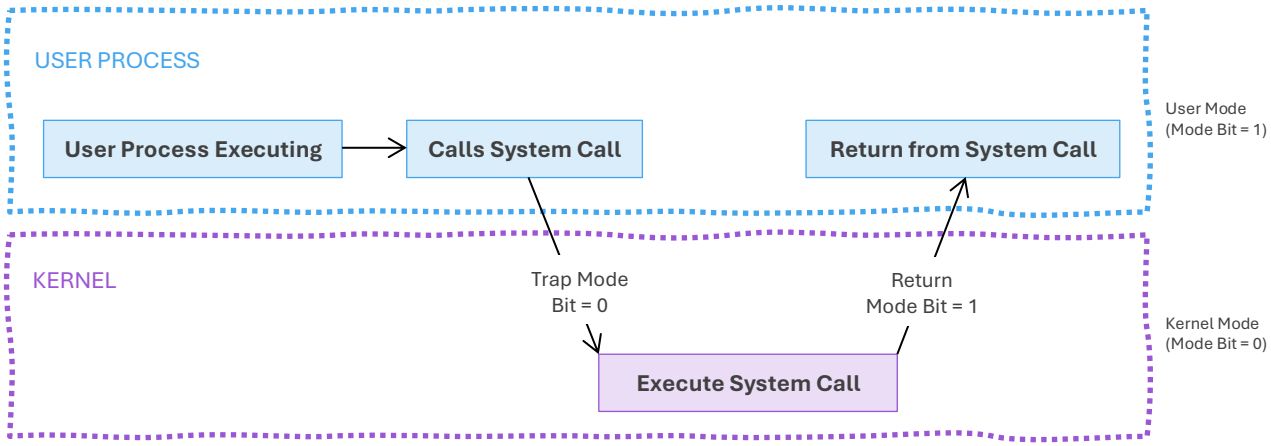


Figure 9 Diagram of Transition from User to Kernel Mode

To maintain control over system resources and prevent and single process from hogging the CPU or running indefinitely, operating systems use timers as a safeguard.

1. Before a process is scheduled, the OS sets a timer to trigger an interrupt after a specified time slice.
2. The OS decrements a counter during execution.
3. When the counter reaches zero, a timer interrupt is generated, causing a transition from user mode to kernel mode. This allows the OS to regain control, perform necessary actions (e.g. context switch), or terminate a misbehaving process.

This timer-based control ensures fair CPU sharing and is a key component of pre-emptive multitasking.