**Chapter 1: Introduction**

Table of Contents

[User View VS System View 3](#_Toc83032853)

[Types of Programs 4](#_Toc83032854)

[Computer System Operation 5](#_Toc83032855)

[Interrupts 6](#_Toc83032856)

[Storage Structure 7](#_Toc83032857)

[I/O Structure 8](#_Toc83032858)

[Device Controllers and Device Drivers 8](#_Toc83032859)

[I/O Operations 9](#_Toc83032860)

[Interrupt Driven I/O 9](#_Toc83032861)

[Direct Memory Access 10](#_Toc83032862)

[Computer System Architecture 11](#_Toc83032863)

[Single-Processor Systems 11](#_Toc83032864)

[Multi-Processor Systems 11](#_Toc83032865)

[Clustered Systems 12](#_Toc83032866)

[OS Structures 14](#_Toc83032867)

[OS Operations 15](#_Toc83032868)

An operating system is an **intermediary** between the user and the hardware. It has two jobs:

1. Control hardware
2. Co-ordinate resources

Here, resources refer to things like the processor cycles, memory, etc.

## User View VS System View

The operating system can be seen from two perspectives, the **user**’s perspective and the **system**’s perspective.

From the user’s perspective, the operating system can be divided into four categories:

1. **Single user** – The operating system is monopolized, which leads to high performance.
2. **Multiple users** on the same **PC** – Maximum resource utilization must be achieved.
3. **Multiple users** on the same **network** – Maximum resource utilization must be achieved.
4. **No users** – Some operating systems work on their own. Home devices and automobiles may have numeric keypads or indicator lights, but most people will fail to even recognize that they are computers due to the lack of user intervention.

From the system’s perspective, the operating system has a few duties:

* It acts as a **resource allocator**
* It **distributes resources** efficiently
* It acts as a **control program**, e.g. by preventing specific users from performing certain actions
* It **executes programs** and **prevents activities** that are harmful to the system

## Types of Programs

**Kernel** - An OS is actually a combination of several programs. A few parts of it are running all the time, e.g. file explorer. The combination of these parts is called the kernel.

**System Programs** – These are programs that are associated with the OS, but are not necessarily part of the Kernel, meaning they are not constantly running or can be turned off. Windows Defender is a system program.

**Application Programs** – These are any programs that are not associated with the OS.

**Middleware** – These are software frameworks that provide additional features for developers. For example, if we build some software that uses an API, the API is middleware.

**Bootstrap Program** – Also called the bootloader, this program acts like a starter. It resides in the ROM and loads the OS from the ROM to the RAM. The Bootstrap program is thus the starting point of all operations on a computer.

## Computer System Operation

A modern computer has **1 or more CPUs** that perform some operations. To perform these operations, it will need to interact with **external hardware**. The CPU does this with the help of **device controllers**.

A device controller gives access to **shared memory** through a **common BUS**.

The CPU and device controllers can execute in **parallel**, competing for memory cycles. To make sure this works properly, a **memory controller** is required, which synchronizes access to shared memory.

## Interrupts

An interrupt is a **notification** to the system by one program requesting control while a different program is running.

Interrupts can be generated by hardware or by software. A **hardware interrupt** is generated by sending a **signal** through the system bus. A **software interrupt** is generated using a special process called a **system call** or a monitor call.

When an interrupt is detected, the CPU must stop the current task and store the address of the next instruction that was supposed to be executed.

Next, the **Interrupt Service Routine** (ISR) is started. Each device has a unique ISR, identified by a unique number associated with the device. This ISR is responsible for processing the interrupt.

Once the ISR is finished processing the interrupt, it **stores the result** of the operation and **returns control** to the OS, which starts the previous program back where it left off.

## Storage Structure

Every program that is running needs to first be loaded onto the **RAM**. Each instruction of the program is then taken into **Internal Registers** (IRs) or cache memories, from where they are operated on.

The **LOAD** instruction moves data from the RAM to the IRs. The **STORE** instruction moves data from the IRs to the RAM.

For example, consider a simple mathematical operation where two numbers are being added. The instruction for the entire mathematical operation must be loaded from memory and stored on an IR.

Next, the instruction is decoded and the CPU realizes that some addition must be performed on two numbers. These two numbers must then be loaded from memory and stored in IRs as well.

Finally, the mathematical operation can be done on the numbers and the result stored in an IR. From here, the results may be sent back to memory.

In all of this, memory refers to RAM. This raises the question of why we do not simply use RAM directly instead of having IRs in between. RAM is faster after all.

The problem is that the RAM is **volatile**. It is constantly changing and we will most likely lose data if we try to store it there. It is also **too small** for most purposes. This means we need things like the ROM for important programs like the OS.

## I/O Structure

The OS tends to have a lot of code dedicated to I/O. This is for two reasons:

1. It ensures reliability and system performance.
2. It addresses a varying nature of I/O devices.

### Device Controllers and Device Drivers

Any computer has **CPUs** and **multiple device controllers**.

Device controllers are **hardware** dedicated to dealing with different **devices**. They have some **local buffer** and some special **registers**. The device controller is responsible for **data transfer** between the device and the local buffer.

For every device controller, there is also a **device driver**. Whenever we attach a new device to the computer, the corresponding device driver is installed. The device driver is **software** that acts as the **interpreter** between the OS and the device controller.

## I/O Operations

There are two types of I/O Operations, **Interrupt Driven I/O** and **Direct Memory Access (DMA)**.

### Interrupt Driven I/O

The steps for Interrupt Driven I/O are:

1. The device driver loads the appropriate **registers**. The registers are located in the device controller.
2. The device controller will **examine** the registers and determine what **action** to take.
3. The device controller starts **data transfer** from the device to the local buffer.
4. The device controller **interrupts** the device driver about the completion of the transfer.
5. The device driver **transfers control** to the OS so that it can retrieve the output of previous instructions.

### Direct Memory Access

For a small amount of data, the Interrupt Driven approach is fine, but for a large amount of data, this will cause a **huge number of interrupts**. Instead, we use DMA. The steps for DMA are:

1. Once the required buffers, pointers and counters for the I/O action has been set up, the device controller transfers an entire **block of data** between the local buffer and the RAM. This does not involve any intervention by the CPU.
2. An interrupt is generated only **once**, to indicate the completion of the transfer of the memory block. Instead of generating interrupts for each byte, we only generate one for the **entire memory block**.
3. The device driver still exists and interprets the data **after** it has been transferred to the RAM.

## Computer System Architecture

### Single-Processor Systems

A single-processor system has a **single CPU**. It may have other, **special-purpose** processors, but those processors do not execute **user tasks**.

### Multi-Processor Systems

A multi-processor system has **multiple CPUs** that are able to execute user tasks. They **share** the BUS, clock cycles, memory and peripherals.

#### Multi-Processor System Advantages

A multi-processor system provides several advantages:

1. It has **increased throughput**.
2. It provides an **economy of scale**. The fact that memory and other resources are **shared** means the overall system cost is lower than it would be for multiple single-processor systems.
3. It has **increased reliability**, since if a single processor fails, the others can **continue serving**.

Related to increased reliability is the concept of **Graceful Degradation**. This is the ability to provide services proportional to the available hardware. Over time, as the hardware degrades in the multi-processor system, its ability will begin to degrade, which will give the user time to shift to a new system. A single-processor system would fail all at once.

Some systems even go beyond this, providing **Fault Tolerance**. This is the ability to **detect faults** and **diagnose** them. It requires **additional hardware** to be available as well as some **software** components.

#### Multi-Processor System Types

Multi-processor systems can be of two types, **asymmetric** or **symmetric**.

Asymmetric systems have **boss-worker relationships**. One processor acts as the boss, **assigning work** to the other, worker processors.

Symmetric systems have all the processors working on their **own tasks**. They have their own sets of cache and registers and all of them are able to perform all the tasks required by the OS.

### Clustered Systems

A clustered system is a **combination** of one or more individual systems that are said to be **loosely coupled**. They may share some storage and could be linked closely via LAN.

There are two types of clustered systems, **asymmetric** systems and **symmetric** systems.

In an asymmetric clustered system, one system is in **hot-standby mode**. This means that it does not run applications, like the other systems, or **active nodes**, do. Instead, it **monitors** the tasks of the other nodes. If one of the current active nodes **fails** for any reason, the system that was in hot-standby mode **becomes an active node**. Once this is done, it is possible that some other node goes into **hot-standby mode**.

In a symmetric clustered system, **all the nodes are monitoring** each other all the time. If any node fails, the entire system becomes aware of it immediately and can react appropriately, perhaps by notifying users that the server is down and is being repaired.

## OS Structures

All modern CPUs have **multiple cores**, which means all operating systems have the ability to **multi-program**. This increases the **CPU utilization** by organizing the tasks in a way such that the CPU **never remains idle**. The OS has several tasks simultaneously kept in a **job pool** in the **ROM**. These tasks are awaiting allocation of **RAM**.

## OS Operations

When any event occurs, the OS needs to be informed. This is done with **interrupts**, either hardware generated ones or software generated ones. Software generated interrupts are called **traps**.

The OS also needs to ensure that every process does not get access to all parts of the computer, which could harm the computer. This is done via two approaches, **dual and multi-mode operations** and **timers**.

In dual and multi-mode operations, the OS just keeps a **mode bit**. If the mode bit is set to **0**, this means the **kernel mode** is active and elevated actions are allowed. If the mode bit is set to **1**, this means the **user mode** is active and certain actions cannot be executed.

Under the timer approach, every program can execute for a **fixed time**. Once that time has passed, the program is set aside and a different process is executed.