# CHAPTER 1

# Introduction to Operating Systems

## I.1. Introduction

In the 1940s, only a limited number of people could use computers, which were viewed as complicated machines aimed at assisting with operations. However, as the years went on, computers became more accessible and by the 1980s were widely in use.

During this revolution, the importance of interacting with computers through operating systems comes into the picture. Operating systems make it easy for users to interface with the machine, allow efficient scheduling of resources, and enable users to accomplish tasks without requiring knowledge of the intricate workings of the system.

Consequently, people can see how important operating systems are in making computers easy to use and available to more users.

## I.2. Computer Fundamentals

A computer is a powerful information-processing machine. It has two main categories of components: **hardware** and **software**.

Hardware is tangible (that we can touch and feel), consisting of three main components: a **central processing unit** (CPU or processor), **memory modules** (RAM, ROM, HDD, flash drives), **input/output devices** (Input devices like keyboard, mouse; output devices like monitor, printers).

Without software, a computer is just a useless piece of hardware. Thanks to this software, it can store, process and retrieve information, and be used for many other activities that justify its usefulness.

Software, on the other hand, is intangible but stored electronically. These are programs that run on a computer. Computer software can be divided roughly into two kinds: **system programs** (**software**), which manage the operation of the computer itself, and **application programs** (**software**)**,** which perform the actual work the user wants.

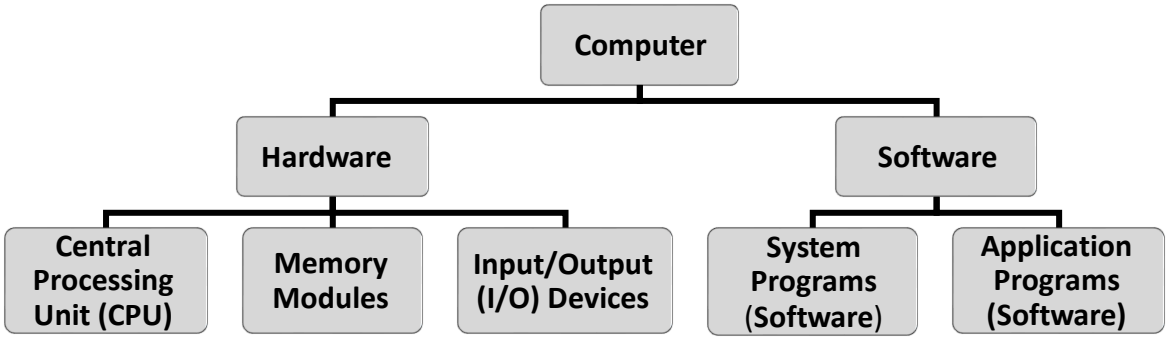



**Figure 1.1 : Different computer components**

## I.3. Computer Organization

Many years ago, it became abundantly clear that some way had to be found to shield programmers from the complexity of the hardware. The way that has evolved gradually is to put a layer of software on top of the bare hardware, manage all parts of the system, and present the user with an interface or **virtual machine** that is easier to understand and program. This layer of software is the operating system.

The location of the operating system is illustrated in Figure 1.2.

<table>
<tr><td>Banking System</td><td>Airline Reservation</td><td>Web Browser</td><td>Application Programs</td></tr>
<tr><td>Compilers</td><td>Editors</td><td>Command Interpreter</td><td rowspan="2">System Programs</td></tr>
<tr><td colspan="3">Operating System (OS)</td></tr>
<tr><td colspan="3">Machine Language</td><td rowspan="3">Hardware</td></tr>
<tr><td colspan="3">Microarchitecture</td></tr>
<tr><td colspan="3">Physical Devices</td></tr>
</table>

**Figure 1.2 : Abstract view of a computer**

At the bottom is the hardware, which, in many cases, is composed of two or more levels (or layers). The lowest level contains **physical devices**, i.e. integrated circuit chips, wires, power supplies, cathode ray tubes, and similar physical devices. How these are constructed and how they work is the province of the electrical engineer.

Next comes the **microarchitecture level**, in which the physical devices are grouped together to form functional units. Typically, this level contains some registers internal to the CPU (Central Processing Unit) and a data path containing an arithmetic logic unit. On some machines, the operation of the data path is controlled by software, called the **microprogram**.

Microprogram is usually located in ROM. It's basically an interpreter that looks up machine-language instructions such as ADD, MOV, and JMP and executes them one after the other. The Microprogram is not part of the hardware, but it is always described in computer manufacturers' manuals. This leads many people to consider it part of the "Machine". On other machines, the microprogram is embedded in the hardware and does not constitute a separate layer on its own ( It is controlled directly by hardware circuits).

The purpose of the data path is to execute some set of instructions. Therefore, the hardware and instructions visible to an assembly language programmer form the **ISA (Instruction Set Architecture)**. This level is often called **machine language**.

The machine language typically has between 50 and 300 instructions, mostly for moving data around the machine, doing arithmetic, and comparing values. In this level, the input/output devices are controlled by loading values into special device registers. For example, a disk can be commanded to read by loading the values of the disk address, main memory address, byte count, and direction (read or write) into its registers. In practice, many more parameters are needed, and

the status returned by the drive after an operation may be complex. Furthermore, for many I/O (Input/Output) devices, timing plays an important role in the programming.

A major function of the **operating system** is to hide all this complexity and give the programmer a more convenient set of instructions to work with. For example, READ BLOCK from FILE is conceptually much simpler than having to worry about the details of moving disk heads, waiting for them to settle down, and so on.

On top of the operating system is the rest of the system software. Here we find the command interpreter (shell), window systems, compilers, editors, and similar application-independent programs. It is important to realize that these programs are not part of the operating system, even though they are typically supplied preinstalled by the computer manufacturer, or in a package with the operating system if it is installed after purchase. This is a crucial, but subtle, point.

The operating system is (usually) that portion of the software that runs in **kernel mode** or **supervisor mode**. It is protected from user tampering by the hardware (ignoring for the moment some older or low-end microprocessors that do not have hardware protection at all). Compilers and editors run in **user mode**.

Finally, above the system programs come the application programs. These programs are purchased (or written by) the users to solve their particular problems, such as word processing, spreadsheets, engineering calculations, storing information in a database, and so on.

## I.4. Computer System

A computer system can be divided roughly into four components: the hardware, the operating system, the application programs, and the users (Figure 1.3).

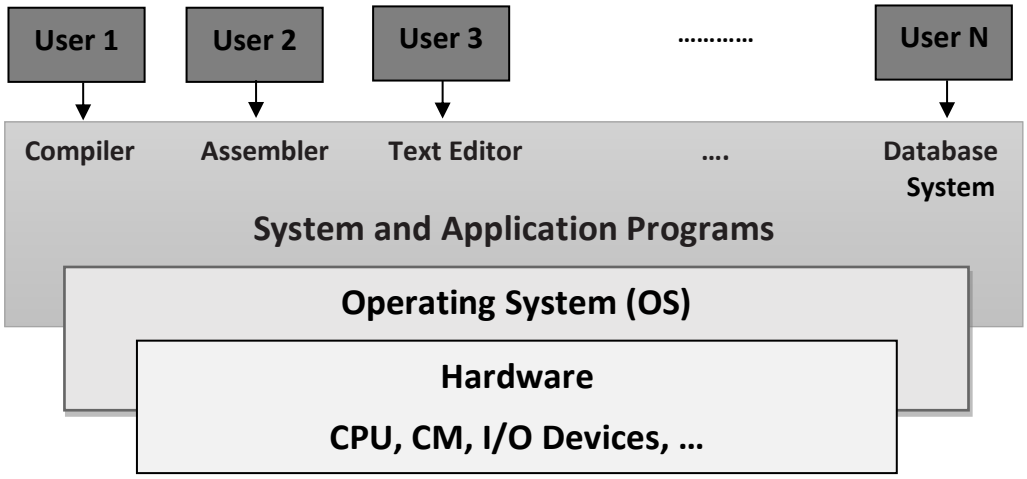



**Figure 1.3 : Abstract view of a computer system**

The most fundamental system program is the operating system, whose role is to control all the computer's resources and provide a base upon which application programs can be written.

An operating system (OS) is the core of system software that conceptually sits on top of the hardware and below the application software. It runs all the time if a present-day computer runs and interacts with the hardware and all application software (application programs as well as utilities) for the convenience of the users.

We can also view a computer system as consisting of hardware, software, and data. The operating system provides the means for proper use of these resources in the operation of the computer system.

The figure clearly shows that the operating system directly controls computer hardware resources. Other programs rely on facilities provided by the operating system to gain access to computer system resources. here are two ways one can interact with the operating system :

- By means of Operating **System Call** in a program
- Directly by means of **Operating System Commands**

## System Call

System calls provide the interface to a running program and the operating system. The user program receives operating system services through a set of system calls. Earlier these calls were available in assembly language instructions but nowadays these features are supported through high-level languages like C, Pascal, etc., which replace assembly language for system programming. The use of system calls in C or Pascal programs very much resembles pre-defined function or subroutine calls.

As an example of how system calls are used, let us consider a simple program to copy data from one file to another. In an interactive system, the following system calls will be generated by the operating system:

- Prompt messages for inputting two file names and reading them from a terminal.
- Open source and destination file.
- Prompt error messages in case the source file cannot be open because it is protected against access or the destination file cannot be created because there is already a file with this name.
- Read the source file.
- Write into the destination file.
- Display status information regarding various Read/Write error conditions. For example, the program may find that the end of the file has been reached or that there was a hardware failure. The write operation may encounter various errors, depending upon the output device (no more disk space, physical end of tape, printer out of paper and so on).
- Close both files after the entire file is copied.

As we can observe, a user program heavily uses the operating system. All interaction between the program and its environment must occur as the result of requests from the program to the operating system.

## Operating System Commands

Apart from system calls, users may interact with the operating system directly using operating system commands.

For example, if you want to list files or sub-directories in MS-DOS, you invoke "**dir**" command. In either case, the operating system acts as an interface between users and the hardware of a

computer system. The fundamental goal of computer systems is to solve user problems. Towards this goal, computer hardware is designed.

Since the bare hardware alone is not very easy to use, programs, (software) are developed. These programs require certain common operations, such as controlling peripheral devices. The command function of controlling and allocating resources is then brought together into one piece of software; the operating system.

To see what operating systems are and what operating systems do, let us consider how they have evolved over the years. By tracing that evolution, we can identify the common elements of operating systems and examine how and why they have developed as they have.

## I.5. What Operating Systems Do?

The Operating System is an intermediary between the machine hardware and application programs. It also controls the hardware and coordinates its use among the various application programs for the various users.

To understand more fully the operating system's role, we next explore operating systems from two viewpoints: that of the user and that of the system.

### User View

In most cases, a personal computer (PC) or laptop is used by a single user (Figure 1.4) at any given time. The user monopolizes the resources (hardware resources or software resources). The main task of the operating system is **to ensure ease of use of the resources**.

By contrast, in a multi-user environment (Figure 1.5), several users work on a single system (mainframe, minicomputer, workstation, or server) connected via their own terminals. Users share different computing resources and exchange information. In this case, **fair and equitable use of resources** is very important.

**Figure 1.4 : Single-User OS**

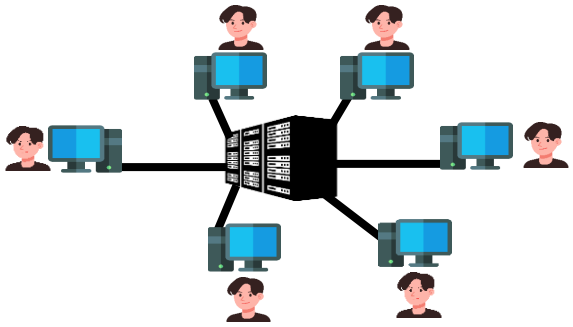


**Figure 1.5 : Multi-User OS**

The role of an OS is not only to offer ease of use to the users but also to ensure that every user gets a fair share of the resources and maximizes the overall performance of the system from the point of view of all users (e.g., maximum resource utilization, lowest overall CPU time, etc.). Therefore, their operating system is designed to compromise between individual usability and resource utilization.

However, there are few embedded systems in which the operating system does not interact with the user, or only very rarely. For example, household appliances such as refrigerators, washing machines, dishwashers, or car indicators.

### User System

As an OS can directly interact with the hardware of the system and the user programs cannot, the OS must ensure **resource allocation** to all users. These resources can be hardware resources like CPU time, memory blocks, I/O, and network devices as well as software resources like programs, file systems, etc.

From the computer's point of view, the operating system is the program most intimately involved with the hardware. In this context, we can view an operating system as a control program or resource allocator for a computer.

Therefore, the OS must keep the computer system in a coherent state, avoiding errors and incorrect use of the machine's programs. It must also apply a rigorous resource allocation strategy (hardware or software) to optimize utilization (system performance). It must also apply a security strategy to keep the computer system away from unauthorized access (execution mode, user groups, data encryption, etc.).

## I.6. What is an Operating System?

There are several possible definitions of operating systems.

**Definition 1.1**

An Operating System (OS) is the most important part of a computer system. It consists of a set of programs acting as an interface between the user and the hardware. Its main purpose is to make the computer system more convenient to use, and to ensure that it operates efficiently and optimally.

**Definition 1.2**

An operating system is a set of programs and sub-programs (functions) that manage hardware and software resources to coordinate computer operations. It is the intermediary between application programs and hardware: it intercepts requests from applications and transmits them to the various hardware resources (central memory, input/output peripherals, etc.).

It removes the hardware from the programmer's view and offers a pleasant view of the computer, transforming it into a **virtual machine** that can be easily manipulated by a simple user.

**Definition 1.3**

An Operating System (OS) is a set of programs that work together to manage the resources of a machine (computer). It must satisfy the following two (main) functions:

- **Share resources:** A system must share computer resources between several simultaneous users.
- **Presenting a virtual machine to the user:** The operating system must convert ordinary hardware into a virtual machine.

**Conclusion :** **Operating System = Allocation Program + Control Program.**

## I.7. Operating System Objectives

The Operating system has two main objectives:

**Presentation**

- The OS utilizes computer hardware efficiently.
- It makes the computer system more convenient for the user.
- The OS provides the user with a simpler and more pleasant abstraction than the underlying hardware, effectively acting as a virtual machine.

**Management**

- The OS controls and manages resources (both hardware and software) among the programs that utilize them.
- It assists user programs in their operations.
- The OS protects users in situations of shared usage.

## I.8. History and Evolution of Operating Systems

Historically, operating systems have been linked to the architecture of the computers on which they were implemented. The evolution of one follows the evolution of the other, and they develop in parallel. Development was motivated by the following objectives:

- Relieve the programmer of huge and tedious programming tasks, and allow him to concentrate on writing their application.
- Protect the system from users and false manipulations.
- Provide a simple, uniform and coherent view of the machine and its resources.

In this section, we describe the successive generations of computers and look at what their operating systems were like. We can distinguish four generations of systems: early systems, batch systems, multi-programmed systems and time-sharing systems.

### I.8.1. Early System (Vacuum Tubes and Plugboards)

This generation, which dates from 1945-1955, refers to the first systems, which were bulky, very fragile and very slow vacuum-tube machines. These machines, designed for machine-language programming on plugboards or punched cards, were used for simple calculations (sine and cosine tables). They ran without operating systems and were single-user systems. The user is responsible for building, programming and maintaining his program. Each user is allotted a fixed amount of time to use the computer, so he has to reserve the machine in advance for a time slot that could exceed the real time slot he needs to run its program. This led to poor processor utilization, as CPU time was wasted and not very active.

**Disadvantages**

- The time for setting up is much higher than the actual time of computation.
- There is wasted time waiting for the program to start.
- The exercise is extremely laborious and error-prone.
- Machine execution speed limited by the speed of the operator pressing buttons and feeding peripherals.

- No difference between: designers; builders; programmers; users; maintainers.

## I.8.2. Batch Processing Systems (Transistors)

The 1955-1965 period, exactly in the middle of the 1950s, saw the use of transistor machines that changed the picture radically, but were still expensive. Therefore, computers became reliable enough with the expectation that they would continue to work long enough to get some useful work done. These machines were designed for Fortran and Assembler programming. They used a Fortran loader and compiler, which constituted the first basic software. In this generation, a clear distinction was made between designers; builders; programmers; users; maintainers.

Batch processing is the sequencing of jobs according to the order of control cards, using a sequence monitor. The aim was to reduce the time wasted by idle processors between the execution of two jobs or programs (during this period, transistor machines with magnetic tape units were introduced, and computers evolved).

The main idea was to collect a set of jobs and then transfer them to a magnetic tape using an auxiliary computer (e.g. IBM 1401). This tape would then be rewound onto the tape drive of the main computer (e.g. IBM 7094) to execute the transcribed jobs using a special program (the ancestor of today's O.S. e.g. FMS: Fortran Monitor System, IBSYS). The results are retrieved on another tape for printing by an auxiliary computer. This situation is illustrated in Figure 1.6.

When the monitor encounters a control card indicating program execution, it loads the program and gives it control. When finished, the program gives control back to the sequence monitor. This continues with the next control card, and so on until all jobs have been completed. The structure of a submitted job is shown in Figure 1.7 :

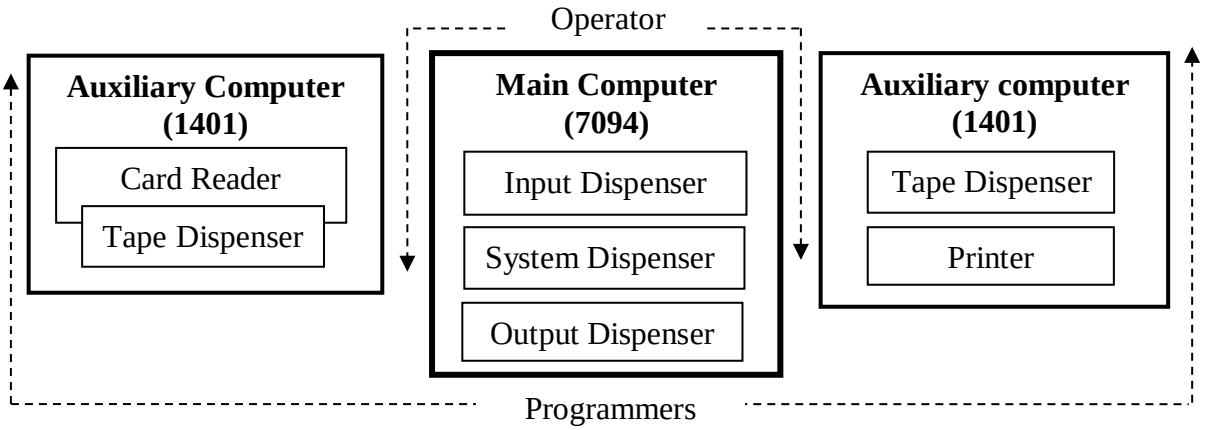



**Figure 1.6 : A batch system**

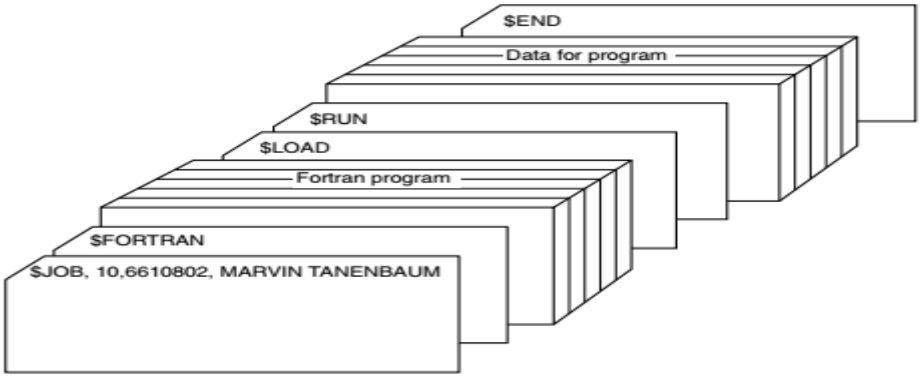



**Figure 1.7 : Structure of a typical FMS job.**

**Disadvantages**

- Time lost due to CPU busy during I/O operations (In effect, CPU is often idle, because the speeds of the mechanical I/O devices are slower than CPU).
- Lack of interaction between user and job during execution
- Difficult to provide the desired priority.
- Unfinished jobs are abandoned.

## I.8.3. Multiprogramming Systems

The years 1965-1980 saw the use of integrated circuit machines, which were less costly and more powerful, as well as the emergence of a family of compatible computers with a single operating system (IBM 360, 370, .etc., machines with the same architecture and instruction set).

The CPU's speed outpaces that of input/output devices, resulting in frequent waiting periods. Various techniques have been developed to enhance computer system performance, such as Direct Access Memory (DAM), buffering, and spooling (Simultaneous Peripheral Operation On Line). All these technologies allow the use of secondary memory instead of magnetic tape, leaving the CPU running only for computational tasks and exchanging I/O over a channel.

**DMA (Direct Memory Access)**

DMA (Direct Memory Access) chip which directly transfers the entire block of data from its own buffer to main memory without intervention by the CPU was a major development. While the CPU is executing, DMA can transfer data between high-speed input/output devices and main memory. By utilizing DMA, the CPU can delegate data transfer tasks to the DMA controller, which can directly access system memory and manage the data transfer independently. The DMA controller can transfer data in blocks or chunks without requiring constant intervention from the CPU for each data transfer operation.

**Buffering**

A buffer is an area or primary storage for holding data during I/O transfers, where the I/O transfer speed depends on many factors related to I/O Hardware but is normally unrelated to processor operation. On input, the data is placed in the buffer by an I/O channel when the transfer is complete the processor may access the data.

**Spooling (Simultaneous Peripheral Operation On-line)**

Spooling uses the disk as a huge buffer for reading as far ahead as possible on input devices and storing output files until the output devices can accept them. Spooling is now a standard feature of most O.S. Spooling allows the computation of one job can overlap with the I/O of another job; therefore, spooling can keep both CPU and the I/O devices working at much higher rates.

**Difference Between Spooling and Buffering in OS**

Spooling and buffering are the two ways by which I/O subsystems improve the performance and efficiency of the computer by using a storage space in main memory or on the disk.

The main aim of buffers is to match the speed of data streaming between a sender and receiver. There is a difference between Spooling and Buffering.

- The basic difference between Spooling and Buffering is that Spooling overlaps the I/O of one job with the execution of another job while the buffering overlaps I/O of one job with the execution of the same job.
- In buffering, there is a small separate area in the memory know as a buffer. But spooling can make use of the whole memory.
- Spooling is more efficient than buffering.

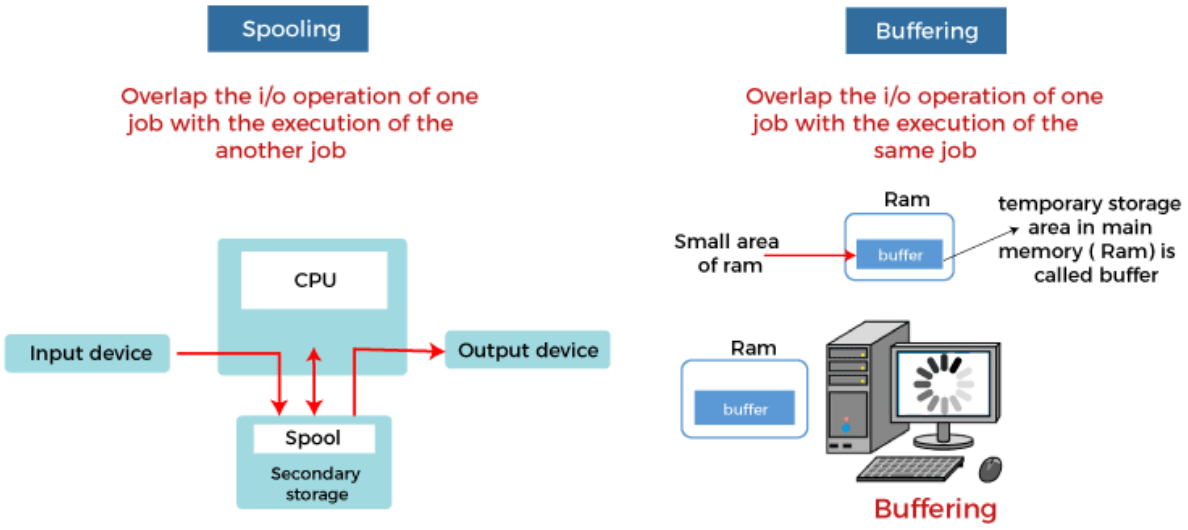



**Figure 1.8 : Spooling and Buffering in OS**

**Multiprogramming**

Buffering and spooling improve system performance by overlapping the input, output and computation of a single job, but both of them have their limitations. A single user cannot always keep CPU or I/O devices busy at all times.

Multiprogramming offers a more efficient approach to increase system performance. In order to increase the resource utilization, systems supporting multiprogramming approach allow more than one job (program) to utilize CPU time at any moment. More number of programs competing for system resources, better will be resource utilization.

A multi-programming system holds several jobs (tasks) in memory at the same time (Figure 1.9).

To overcome the disadvantages of batch processing, the idea was to maintain several jobs or tasks in memory, ready for execution, and to efficiently share machine resources between these jobs.

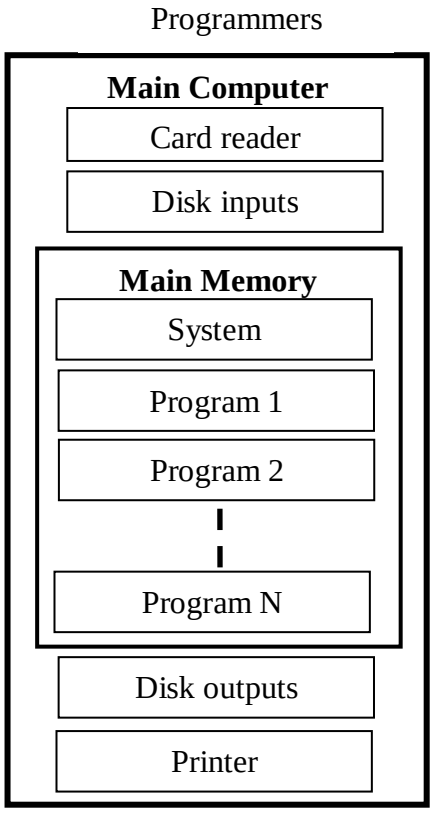
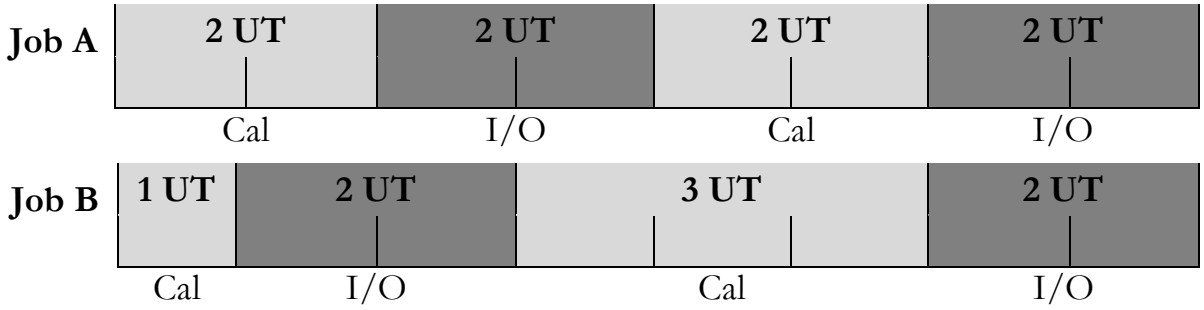



**Figure 1.9 : A multiprogramming system**

In effect, the processor is allocated to a job, and as soon as the job makes an I/O request, the processor is allocated to another job, thus eliminating waiting times for the processing unit in charge of I/O, called the I/O channel.

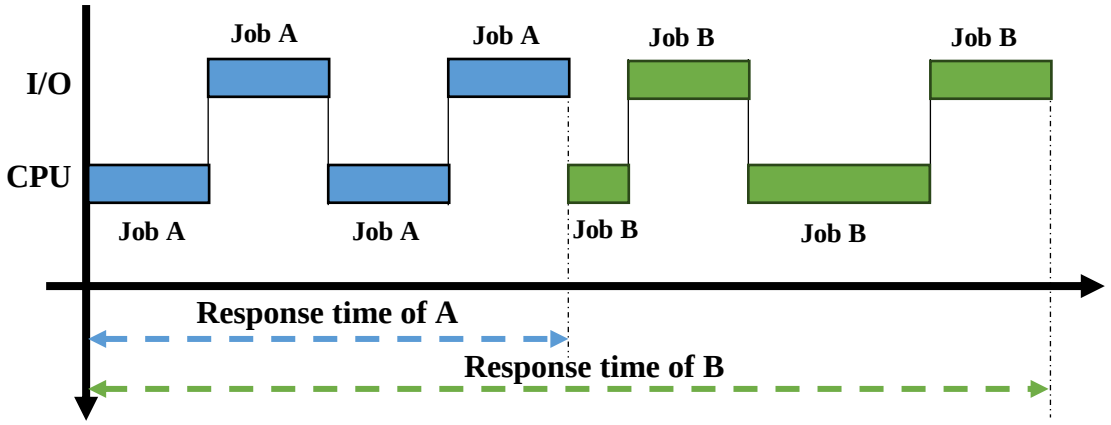
## Example

Consider the following two programs A and B:

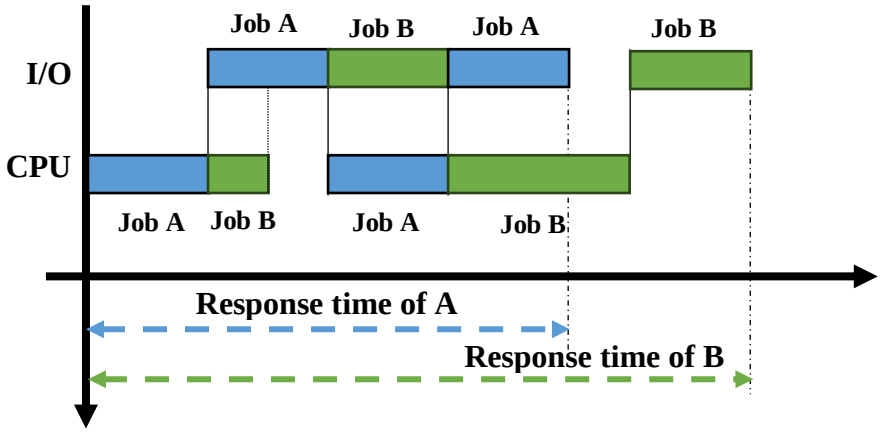



A single I/O device is assumed.

**Uni-programming system**




**Multi-programming system**




**Comparison of Uni-programming and Multi-programming**

| Uni-programming | Multi-programming |
|---|---|
| ✓ Poor use of resources (processor, memory, I/O, etc.).<br>✓ Very long job response times.<br>✓ Simple OS; only constraint : protect the resident part of the system from users. | ✓ Better resource load balancing.<br>✓ Better use of memory (minimizing free space).<br>✓ Possibility of improving response times for short jobs.<br>✓ Protect user programs from the actions of other users, and also protect the resident part of users. |

## I.8.4. Time Sharing Systems

Time-sharing is a logical extension of multiprogramming. It is a variant of the multiprogramming mode where CPU time is distributed in small slices, and it is called **time quantum**.

Time-sharing systems involve dividing the CPU between several tasks by quantum of time, so that they can be processed "**simultaneously**", enabling several users connected online to be served interactively at the same time.

The aim is to offer users direct interaction with the machine via conversation terminals and to allocate the processor to them successively for a quantum of time, so that each user has the impression of having the machine all to himself. They can also control the job they have submitted directly from the terminal (correct errors, recompile, resubmit the job, etc.).

These include CTSS (Compatible Time-Sharing System), MULTICS (MULTiplexed Information and Computing Service), an ancestor of Unix, and Unix, the most widely ported. In fact, most of today's systems are time-sharing.

The main difference among batch systems, multiprogramming and time-sharing systems is that in the case of batch systems and multiprogramming, the objective is to maximize processor use, whereas in time-sharing systems, the objective is to minimize response time.

**Disadvantages**

- It is more complex than a multiprogramming operating system
- The system must have memory management & protection since several jobs are kept in memory at the same time.
- Time-sharing system must also provide a file system, so disk management is required.
- It provides a mechanism for concurrent execution which requires complex CPU scheduling schemes

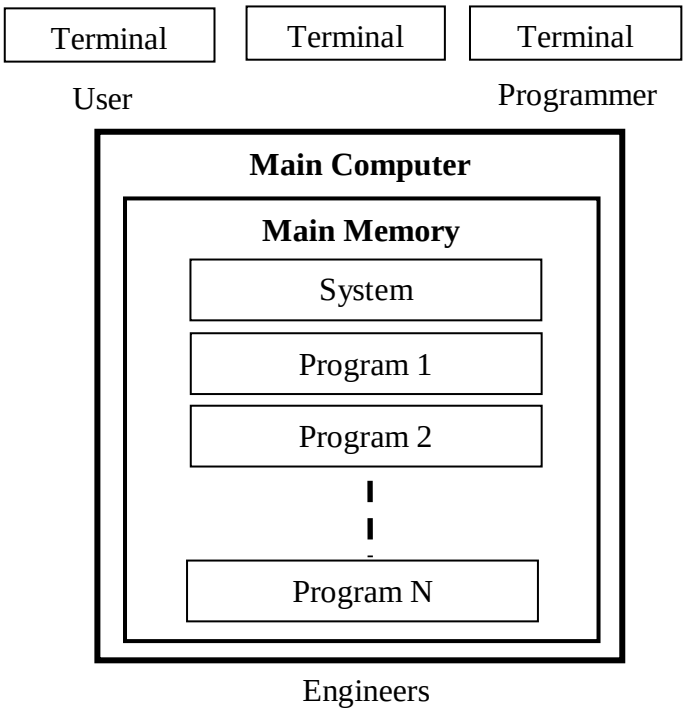



**Figure 1.10 : A time-sharing system**

## I.9. Operating System Functions

The operating system performs the following main functions:

### I.9.1. Resource Management

Operating systems work as the resource manager of a computer. These resources are processors, memory, file system and I/O devices. We briefly introduce here how these resources are managed by operating systems in general.

**Processor Management** : Manage processor allocation between different programs using a scheduling algorithm.

**Process Management** : Manage the execution of processes by allocating the resources needed for them to run smoothly. An operating system does the following tasks related to the process management:

- Creation and deletion of processes
- Scheduling of processes (or threads aka sub-processes) for CPU time
- Suspending and resuming processes
- Communication and coordination among several cooperating processes
- Resolving contention among competing processes

**Memory Management** : As part of memory management, OS does the following:

- Keeping track of memory addresses being accessed and by whom (process-id)
- Allocating and deallocating memory space to different processes
- Deciding which processes (or their parts) and their data need to be brought into and removed from memory

**File-System Management** : The information (code + data) is stored in the logical unit of files. A file is a sequence of records. Each file is a device-independent concept. However, each physical storage medium has different physical characteristics with diverse ways of storing and retrieving data. Operating systems provide an abstraction of files and map the logical files onto physical media. File system management, a part of an OS, also helps organize the files into directories (or folders) that users think are a logical collection of related files. Specifically, an operating system does the following jobs as part of file system management:

- Formatting media into file system type (e.g., DOS, Windows, Unix file system, etc.).
- Mapping files onto physical media.
- Creation, modification, and deletion of files.
- Creation, modification, organization, and deletion of directories (and sub-directories).
- Copying (backing up) files and directories from one media to another.

**Inputs/Outputs Management** : I/O devices are made of different physical materials, have different physical characteristics, and thus require different handling techniques. Computer performance depends on proper management and control of the I/O devices, and application programmers must be kept free of their low-level nitty-gritty, which can be diverse and complex.

Operating systems provide a general device driver for related categories of I/O devices and specific ones when necessary. The drivers interact with the device controllers and manage the devices. Operating systems provide users with simpler interfaces to interact with the devices. In most operating systems, I/O subsystem does the job, by specifically providing:

- A memory management module to buffer, cache, and spool data transfer
- A general device driver interface
- Specific device driver interfaces.

## I.9.2. Security and Protection

Several users can simultaneously use a computer in a multi-user, multi-program environment. A user program may inadvertently or intentionally (with malicious intent) access a resource and/or execute code it is not intended to use.

Every computer should have a policy and mechanism for restricting access to important resources. Security and protection are often seen as similar concepts. However, semantically, security refers to policies to safeguard a system from external attacks, while protection refers to the implementation of mechanisms to deal primarily with internal attacks.

Computer systems have different protection schemes that are implemented at various levels: some are at the hardware level and some are at the software (or operating system) level.

**Processing Mode** : At the hardware level, every processor supports at least two operating modes: **kernel** (or **supervisor**, **system**, or **privileged**) mode and **user** mode. A **mode-bit** is used to denote (say, kernel-mode 0 and user-mode 1) in which mode the processor is executing instructions.

In the kernel mode, a processor can execute all instructions like accessing all hardware and code from any user programs. But in used mode, the processor can only execute instructions from designated memory regions (user area) and cannot access hardware. Some processors support more than two operating modes (e.g., Intel has **4** protection rings or modes, and ARMv8 has **7** modes).

If the user program needs to access hardware or execute a code beyond its designated area, it needs to raise a service request to the operating system through a **system call (or syscall)**. Syscalls change the processor mode from user to kernel. The syscalls are services of the operating system.

**Address Space** : Every process is allocated memory that may not be physically contiguous and not entirely loaded in the RAM while the program is being run. However, the process sees conceptually (or virtually) the space as contiguous. It is referred to as virtual address space and such memory is called virtual memory. A fixed portion of the virtual address space of each process maps the kernel code and data - this portion is called **kernel space** or **system space** that can only be accessed in kernel mode.

Operating system kernel maintains some global data structures and some per-process objects. Two important per-process objects are the user area and kernel stack. User processes are not allowed to modify the objects and can only be modified in kernel mode. Therefore, user processes cannot access kernel space, if they need to execute some portion from kernel, they should use **syscall**.

**Execution Context** : Kernel functions may run for the need of the kernel itself or due to the request of the user programs. When they are run by the kernel for its own reasons (system-wide maintenance & managerial purpose initiated mostly by interrupts), it is called **system context**.

During this time, it usually does not need to access the user address space, user area, or kernel stack of the current process. However, when kernel functions are run by the kernel on behalf of the requesting process, it is called to be run in the **process context**. The kernel may access and modify the **process address space**, **user area**, and **kernel stack** of the current process.

### Remark

It is important to note that the term **kernel** comes in 3 different contexts:

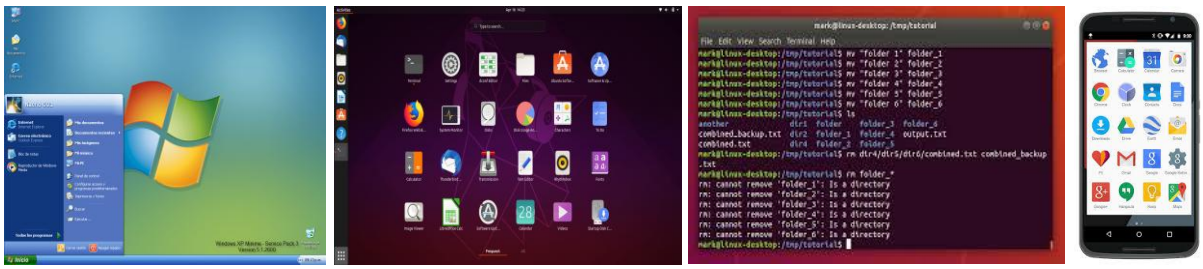
- Kernel can mean the core of operating system software (code + data)
- Hardware operating mode (kernel mode) and
- Virtual memory space (kernel space).

We must understand the meaning from the context.

## I.10. Operating System Services

Operating systems provide different services to the users and application programs. The services offer ease of using the computer. These services vary from one OS to another. Here, we list some of the general and common services:

**User Interfaces** : All OSs provide user interfaces (UIs). Modern operating systems typically feature graphical user interfaces (GUIs). Desktop computers allow users to navigate various work windows within the GUI using a mouse or touchpad, while smartphones, tablets, and high-end laptops utilize touchscreen interfaces that respond to finger touches. Additionally, PC-based operating systems offer command-line interfaces (CLIs), which accept input only through text commands.



**(a) GUI of Windows (b) GUI of Ubuntu (c) CLI of Ubuntu (d) Android GU**

**Figure 1.11 : User interfaces of different operating systems**

**Program Execution** : The system must load a program into memory and execute it. The program must be able to terminate its execution either normally or abnormally.

**I/O Operation** : No user program can directly handle I/O devices, but they often need to do so, (like reading from a file, an input device, or a network device and/or writing onto one or more of them), during their execution. Some special functions can be desired; therefore, operating systems facilitate this through different system calls (**syscall**).

**File System Manipulation** : Users can access and modify the contents of a file using either programs or user interfaces (UIs). Operating systems (OSs) provide the necessary mechanisms for creating, opening, modifying, and deleting files, as well as for organizing them within a hierarchy

of directories and subdirectories. Additionally, an OS allows users to create, modify, and delete folders or directories, along with their subfolders (subdirectories), through programs or UIs. Hence, the operating system must maintain each file correctly.

**Communication** : Processes can interact with each other either within a single machine or between multiple remote machines connected through a network. Operating systems offer communication mechanisms such as shared memory for processes on the same physical system, as well as message passing, which involves the operating system transferring packets of information between processes.

**Error Detection** : Errors can occur during program execution and are monitored by hardware components like the CPU (e.g., division by zero), memory (e.g., illegal memory access), or I/O devices (e.g., no pages in the printer). The operating system (OS) monitors the hardware, checking for exceptions and notifying the relevant programs through Interrupt Service Routines (ISRs). This allows the programs to take appropriate action. If necessary, the OS may terminate malfunctioning processes to ensure the smooth operation of other processes and the overall system.

**Resource Allocation** : In a single-user, single-process environment, all resources are monopolized by one process at a time. In contrast, a multiprogramming environment allows multiple processes to request the same resource, such as the CPU, concurrently. To manage this effectively, operating systems must implement scheduling algorithms (for example, CPU scheduling) so that each process can access the resources fairly. In this case, we need fairness to apply to all computer resources, including CPU, memory, and peripherals.

The complexity of resource management increases as we move from single-user, multiprogramming systems to distributed or clustered systems. In these cases, the operating system must act as a fair allocator of resources to all candidate processes and as an arbitrator to resolve conflicts and prevent the unfair monopolization of resources.

## I.11. Classification of Operating Systems

Operating systems can be classified according to different contexts: usage, services provided and hardware architecture.

### I.11.1. System Classification Based on Usage Constraints

Depending on the usage constraints, we distinguish:

- **Single-user/single-task systems**, where only one user uses the system at a time, and only one task can be executed at a time. This is the case with MS-DOS, for example.
- **Single-user/multitasking systems** where only one user at a time can run several tasks simultaneously, as is the case with Windows.
- **Multi-user/multitasking systems** where several users at a time can each execute several tasks simultaneously and share the same hardware resources, as is the case with Unix.

### I.11.2. System Classification by Service

Depending on the services provided, two categories can be distinguished:

- **Real-time systems**, which are used in specific fields (processes, robotics, nuclear power plants, etc.). These systems ensure short response times to critical tasks, and are reliable and fault-tolerant.
- **Transactional systems** are dedicated to the management of huge databases (reservation systems, banking systems, etc.) and guarantee inconsistency-free updates.

### I.11.3. System Classification by Architecture

According to their hardware architecture, operating systems are classified into two categories:

- **Single-processor systems**, which refer to multi-programmed, time-sharing systems. These systems need only one processor, which can perform pseudo-parallel calculations to keep tasks moving forward at the same time.
- **Multiprocessor (Parallel) systems** such as SunOS 4, SunOS 5 (Solaris 2) and Linux are capable of parallel processing by several processors. They are reliable and feature high processing capacity and short response times.

## I.12. Examples of Operating Systems

Several operating system variants have appeared since the 1950s, the most famous of which are :

### I.12.1. Microsoft Disk Operating System (MS-DOS)

MS-DOS is a single-user system, known since the early '80s, designed for microcomputers (the first IBM-PC), that Bill Gates's young company developed for IBM as Personal Computer - Disk Operating System (PC-DOS) and now associated with Windows.

In 1981, IBM licensed and marketed its PC-DOS rebranding of MS-DOS to run on IBM PCs. It was the main operating system for personal computers from 1981 to 1995 before graphical user interfaces (GUIs) like Microsoft Windows became popular. MS-DOS was released for x86 computers, went through eight major versions and was ultimately retired from all active support in 2006.

### I.12.2. Windows Operating System

The Windows operating system, developed by Microsoft in the mid-1980s, features a graphical interface. Its name is derived from the concept of using separate windows for each task execution. It is designed for several architectures (PC, workstation, laptop, client/server network, etc.).

It has gone through several versions, as for PC: 1.0, 2.0, 3.10, 3.11, 9x, 2000, Me, XP, Vista, Seven, 10, also: Windows Server, NT for multi-user and client/server architecture.

### I.12.3. UNIX Operating System

UNIX (Uniplexed Information and Computer Service) is an operating system that was initially developed in the 1969s by Ken Thompson, Dennis Ritchie, and others at AT&T Laboratories and has been continually enhanced ever since.

It is like a backbone for many modern operating systems like Ubuntu, Solaris, Kali Linux, Arch Linux, and also the POSIX standard. It is a stable, multi-user, multi-tasking system for servers,

desktops, and laptops. Additionally, UNIX is a family of time-sharing systems proposed for most architectures, used in certain industrial and research environments.

**History**

- **1960** : MULTICS by Bell Labs, GE and MIT, project abandoned
- **1966** : Bell Labs (AT&T) started an SE for internal use, developed by Ken Thompson and Dennis Ritchie. Ritchie also invented the C language, successor to Thompson's B language.
- **1969** : Birth first version of UNIX, named after UNICS at Bell Labs by Ken Thompson.
- **1973** : Complete rewriting of UNIX in C language.
- **1974** : Publication of UNIX in the C language by Thompson and Ritchie.
- **1974** : AT&T licenses their UNIX to universities, giving rise to BSD (Berkeley Software Distribution) thanks to the research work of UC Berkeley's CSRG (Computer Systems Research Group).
- **1978 - 1980** : AT&T introduces and allows other manufacturers to clone their UNIX (Ultrix DEC, BSD Sun, AIX IBM, AUX Apple, XENIX Microsoft, Irix SGI).
- **To 1980s** : AT & T markets System III, System IV and System V versions.
- **1991** : The first version of Linux, a clone of the MINIX system.

**Reasons for UNIX's Success**

- High-level language compared with other systems at the time.
- Adapted user interface.
- Open source -> source code can be improved by anyone, resulting in the appearance of several variants by different companies: System V by AT & T, XENIX by Microsoft, ULTRIX by DEC, for Linux -> Redhat, Coldera, ...
- Free versions are available.

**UNIX Kernel Services**

- Hardware event management.
- Resource management.
- Data structure management.
- Memory and processor management.
- Provides basic services.