## Operating system

An Operating System (OS) acts as an interface connecting a computer user with the computer's hardware. An operating system falls under the category of system software that performs all the fundamental tasks like file management, memory handling, process management, handling the input/output, and governing and managing the peripheral devices like disk drives, networking hardware, printers, etc. Some well-liked Operating Systems are Linux, Windows, OS X, Solaris, OS/400, Chrome OS, etc.

## Feature of operating system

Here is a list of some significant functions of an Operating System, which is found common in almost all operating systems:

1. Resource management: Operating systems manage the computer's resources, such as its memory, processor, and storage, and allocate them to different tasks as needed.
2. Memory management: Operating systems manage the computer's memory and ensure that each program or process has access to the memory it needs to run.

3. Process management: Operating systems create and manage processes, which are units of work executed by the computer.
4. File management: Operating systems manage the files on the computer, including organizing them and providing access for different programs and users.
5. Security: Operating systems include security features to protect the computer from unauthorized access and viruses.
6. User interface: Operating systems provide an interface for users to interact with the computer, such as through a graphical user interface (GUI) or command-line interface (CLI).
7. Networking: Many operating systems include support for networking, allowing the computer to communicate and exchange data with other devices over a network, such as the internet or a local area network (LAN).
8. Device management: Operating systems manage the devices connected to the computer, such as printers, keyboards, and storage devices.
9. Power management: Operating systems include features to manage the power usage of the computer and conserve energy when possible.
10. Software installation and updates: Operating systems provide a mechanism for installing and updating software applications.

These are just a few examples of features commonly found in operating systems. The specific features of an operating system depend on the particular system and its intended use.

## Objective of operating system

An operating system consists of unique programs that control the execution of software. The OS acts as an intermediary between applications and hardware components. OS can be thought of as having three objectives. These are:

- Convenience: It makes a computer more suitable to use.
- Efficiency: It provides the computer system resources with efficiency and in easy to use format.
- Ability to develop: It should be built in such a way that it permits the efficient development, testing, and installation of new system functions without interfering with service.

At an upper level of any computer architecture, a computer is supposed to have a processor, memory, and some I/O components, with one or more quantities of each type. These components are interrelated and connected in a way to achieve the significant function of the computer, which is to execute programs.

There are four key structural elements of any computer. These are:

- Processor: It controls the processes within the computer and carries out its data processing functions. When there is only one processor available, it is in combination termed as the central processing unit (CPU), which you must be familiar with.
- Main memory: It stores data and programs within it. This memory is typically volatile and is also called primary memory. This is because when the computer is shut down, the contents within the memory get lost. In contrast, the contents of disk memory are kept hold of even when the computer system is turned off, which you call a shutting down of the Operating system or computer. Main memory is also termed real memory.
- Input/output (I/O) devices: This moves the data within the computer to its peripheral external environment. The external environment is supposed to have a variety of devices, including secondary memory devices (e.g., pen drives, CDs, etc.), communications equipment (such as LAN cable), terminals, etc.
- System bus: It provides communication between processors, main memory, and I/O modules.

These are some of the essential elements of a computer system. There may be other components, depending on the specific design and configuration of the system.

An Operating System supplies different kinds of services to both the users and to the programs as well. It also provides application programs (that run within an Operating system) an environment to execute it freely. It provides users the services run various programs in a convenient manner.

Here is a list of common services offered by an almost all operating systems:

- User Interface
- Program Execution
- File system manipulation
- Input / Output Operations
- Communication
- Resource Allocation
- Error Detection
- Accounting
- Security and protection

This chapter will give a brief description of what services an operating system usually provide to users and those programs that are and will be running within it.

## User interface of operating system

Usually Operating system comes in three forms or types. Depending on the interface their types have been further subdivided. These are:

- Command line interface
- Batch based interface
- Graphical User Interface

Let's get to know in brief about each of them. The command line interface (CLI) usually deals with using text commands and a technique for entering those commands. The batch interface (BI): commands and directives are used to manage those commands that are entered into files and those files get executed. Another type is the graphical user interface (GUI): which is a window system with a pointing device (like mouse or trackball) to point to the I/O, choose from menus driven interface and to make choices viewing from a number of lists and a keyboard to entry the texts.

## Program Execution in Operating System

The operating system must have the capability to load a program into memory and execute that program. Furthermore, the program must be able to end its execution, either normally or abnormally / forcefully.

### File System Manipulation in Operating System

Programs need has to be read and then write them as files and directories. File handling portion of operating system also allows users to create and delete files by specific name along with extension, search for a given file and / or list file information. Some programs comprise of permissions management for allowing or denying access to files or directories based on file ownership.

## I/O operations in Operating System

A program which is currently executing may require I/O, which may involve file or other I/O device. For efficiency and protection, users cannot directly govern the I/O devices. So, the OS provide a means to do I/O Input / Output operation which means read or write operation with any file.

## Communication System of Operating System

Process needs to swap over information with other process. Processes executing on same computer system or on different computer systems can communicate using operating system support. Communication between two processes can be done using shared memory or via message passing.

## Resource Allocation of Operating System

When multiple jobs running concurrently,  resources must need to be allocated to each of them. Resources can be CPU cycles, main memory storage, file storage and I/O devices. CPU scheduling routines are used here to establish how best the CPU can be used.

## Error Detection

Errors may occur within CPU, memory hardware, I/O devices and in the user program. For each type of error, the OS takes

adequate action for ensuring correct and consistent computing.

### Accounting

This service of the operating system keeps track of which users are using how much and what kinds of computer resources have been used for accounting or simply to accumulate usage statistics.

### Protection and Security

Protection includes in ensuring all access to system resources in a controlled manner. For making a system secure, the user needs to authenticate him or her to the system before using (usually via login ID and password)

In this chapter, you will learn about the general structure of functionalities and properties that the operating system provides for a typical computer system. A computer system may be organized in a number of different ways, which you can categorize roughly according to the number of general-purpose processors used.

You will learn about these properties one by one given below:

## Single Processor Systems

Most systems at least contain a single processor. The variation of single-processor systems may be unexpected, but since these systems can range from PDA's to mainframe computers. On a single-processor

system, there is one main CPU able to execute a general-purpose instruction layout, including different instructions from users and then process them. Almost every system has special-purpose processors within them. They may come in the appearance of device-specific processors, such as disks, keyboards, graphics controllers, etc.; or on mainframes, they may come in the form of more general-purpose processors.

## Batch Processing

Batch processing is a method where an operating system gathers different programs along with the data together in a batch before starting the process. An operating system does the below-mentioned activities to perform batch related processing:

- The operating system identifies a job or sets of jobs that are further assigned to a sequence of commands, programs, and data within a single unit.
- The operating system maintains the lists of number jobs in memory. It then executes them one by one based on some scheduling algorithm.
- Most of the jobs In operating system are processed in the order they have been submitted, i.e., first come first serve (FCFS) manner.

## Multiprogramming Property of Operating System

Multi-programmed structure or mechanism provides an environment where a variety of system resources like

memory, CPU, and various peripheral devices gets utilized efficiently, but they do not offer for user interaction with the computer system. Time-sharing which is also called multitasking is a logical extension or enhancement of the term multiprogramming.

In time-sharing systems, the processor is given multiple tasks by switching among them, but the switches take place so frequently that the users can work together with each and every program while it is running and it seems that all of the programs are running simultaneously. Time-sharing needs an interactive computer structure that allows direct communication between the user and the system.

**Multiprogramming in Operating System**

A single user cannot keep either the processor or the Input / Output devices busy every time. The concept of multiprogramming is implemented to amplify CPU utilization by managing jobs so that the CPU always has at least one job to execute.

Sharing the processor, when multiple programs reside in memory at a single time, is termed as multi-programming. Multi-programming takes for granted a single shared processor for one or more tasks.

**Clustered Systems in OS**

Another type of multiple - CPU concept is the clustered structured system. Like multi-processing, clustered systems collect together several CPUs to achieve the better computational job. Clustered systems vary from multiprocessor systems, but in the same time, they are composed of two or multiple individual systems that are coupled as a single unit.

During the olden days, computer systems allowed only one program to be executed at one time. This is why that program had complete power of the system and had access to all or most of the system's resources. In contrast, nowadays, current-day computer systems let multiple programs to be loaded into memory and execute them concurrently. This massive change and development required rigid control and more compartmentalization in various programs.

The more fused or complex the operating system is, the more it is expected to do on behalf of its users. Even though its main concern is the execution of user programs, it also requires taking care of various system tasks which are better left outside the kernel itself. So a system must consist of a set of processes: operating system processes, executing different system code and user processes which will be executing user code. In this chapter, you will learn about the processes that are being used and managed by the operating system.

**What are the processes?**

A process is mainly a program in execution where the execution of a process must progress in sequential order or based on some priority or algorithms. In other words, it is an entity that represents the fundamental working that has been assigned to a system.

When a program gets loaded into the memory, it is said to as a process. This processing can be categorized into four sections. These are:

- Heap
- Stack
- Data
- Text

**Process Concept**

There's a question which arises while discussing operating systems that involves when to call all the activities of the CPU. Even on a single-user operating system like Microsoft Windows, a user may be capable of running more than a few programs at one time like MS Word processor, different web browser(s) and an e-mail messenger. Even when the user can execute only one program at a time, the operating system might require maintaining its internal programmed activities like memory management. In these respects, all such activities are similar, so we call all of them as 'processes.'

Again another term - "job" and process are used roughly replacing each other. Much of the operating - system

theory and terminology was developed during a time when the main action of operating systems was job processing; so the term job became famous gradually. It would be confusing to avoid the use of commonly accepted terms which include the word job like 'job scheduling.'

**Process state of Operating System**

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. Each process may be in one of the following states:

- New: In this state, the process is being created.
- Running: In this state, instructions are being executed.
- Waiting: In this state, the process is waiting for the different event to occur like I/O completion or treatment of a signal.
- Ready: In this state, the process waits to assign a processor.
- Terminated: In this state, the process has finished executing.

The process model that has been discussed in previous tutorials described that a process was an executable program that is having a single thread of control. The majority of the modern operating systems now offer features enabling a process for containing multiple

threads of control. In this tutorial, there are many concepts associated with multithreaded computer structures. There are many issues related to multithreaded programming and how it brings effect on the design of any operating systems. Then you will learn about how the Windows XP and Linux OS maintain threads at the kernel level.

**What is a thread?**

A thread is a stream of execution throughout the process code having its program counter which keeps track of lists of instruction to execute next, system registers which bind its current working variables. Threads are also termed as lightweight process. A thread uses parallelism which provides a way to improve application performance.

## Major Types of Threads

Let us take an example where a web browser may have one thread to display images or text while another thread retrieves data from the network. Another example can be a word processor that may have a thread for displaying the UI or graphics while a new thread for responding to keystrokes received from the user and another thread is to perform spelling and grammar checking in the background. In some cases, a single application may be required to perform several similar tasks.

## Advantages / Benefits of Threads in Operating System

The advantages of multithreaded programming can be categorized into four major headings -

- Responsiveness: Multithreading is an interactive concept for an application which may allow a program to continue running even when a part of it is blocked or is carrying a lengthy operation, which increases responsiveness to the user.
- Resource sharing: Mostly threads share the memory and the resources of any process to which they fit in. The advantage of sharing code is that it allows any application to have multiple different threads of activity inside the same address space.
- Economy: In OS, allocation of memory and resources for process creation seems costly. Because threads can distribute resources of any process to which they belong, it became more economical to create and develop context-switch threads.
- Utilization of multiprocessor architectures: The advantages of multithreading can be greatly amplified in a multiprocessor architecture, where there exist threads which may run in parallel on diverse processors.

## Multithreading Models

All the threads must have a relationship between them (i.e., user threads and kernel threads). Here is a list which

tells the three common ways of establishing this relationship.

- Many-to-One Model: In the many-to-one model plots several user-level threads to a single kernel thread.
- One-to-One Model: In the one-to-one model maps every particular user thread to a kernel thread and provides more concurrency compare to many-to-one model.
- Many-to-Many Model: In the many-to-many model, many user-level threads get mapped to a smaller or equal quantity of kernel threads. The number of kernel threads might be exact to either a particular application or to a particular machine.

CPU scheduling is the foundation or starting concept of multi-programmed operating systems (OSs). By toggling the CPU with different processes, the operating system can make the computer and its processing power more productive. In this tutorial, you will learn about the introductory basic of CPU-scheduling concepts.

**What is CPU / Process Scheduling**

The CPU scheduling is the action done by the process manager to handle the elimination of the running process within the CPU and the inclusion of another process by certain specific strategies.

## Reason Behind the Use of CPU Scheduling

In a single-processor system, only one job can be processed at a time; rest of the job must wait until the CPU gets free and can be rescheduled. The aim of multiprogramming is to have some process to run at all times, for maximizing CPU utilization. The idea is simple. In this case, the process gets executed until it must wait, normally for the completion of some I/O request.

In a simple operating system, the CPU then just stands idle. All this waiting time is wasted; no fruitful work can be performed. With multiprogramming, you can use this time to process other jobs productively.

## CPU-I/O Burst Cycle

The success of CPU scheduling varies on an experiential property of processes: Process execution holds a cycle of CPU execution and Input / Output wait. Processes get to swap between these two states. Process execution begins with a burst of CPU. That is followed by an Input / Output burst and goes after by one more CPU burst, then one more Input / Output burst, and it continues. Eventually, the final or last CPU burst finish with a system request for terminating execution.

## CPU Schedulers

Whenever the CPU gets idle, the operating system (OS) has to select one of the processes in the ready queue for execution. The selection process is performed by the short-term scheduler (also known as CPU scheduler). The

scheduler picks up a process from the processes in memory which are ready to be executed and allocate the CPU with that process.

### Preemptive Scheduling

CPU scheduling choices may take place under the following four conditions:

- When a process toggles from the running state to its waiting state
- When a process toggles from the running state to its ready state (an example can be when an interrupt occurs)
- When a process toggles from the waiting state to its ready state (for example, at the completion of Input / Output)
- When a process terminates (example when execution ends)

## Necessary Conditions and Preventions for Deadlock

A deadlock state can occur when the following four circumstances hold simultaneously within a system:

- Mutual exclusion: At least there should be one resource that has to be held in a non-sharable manner; i.e., only a single process at a time can utilize the resource. If other process demands that resource, the requesting process must be postponed until the resource gets released.

- Hold and wait: A job must be holding at least one single resource and waiting to obtain supplementary resources which are currently being held by several other processes.
- No preemption: Resources can't be anticipated; i.e., a resource can get released only willingly by the process holding it, then after that, the process has completed its task.
- Circular wait: The circular - wait situation implies the hold-and-wait state or condition, and hence all the four conditions are not completely independent. They are interconnected among each other.

### Methods for Handling Deadlocks

Normally you can deal with the deadlock issues and situations in one of the three ways mentioned below:

- You can employ a protocol for preventing or avoiding deadlocks, and ensure that the system will never go into a deadlock state.
- You can let the system to enter any deadlock condition, detect it, and then recover.
- You can overlook the issue altogether and assume that deadlocks never occur within the system.

But is recommended to deal with deadlock, from the 1st option

## Basic hardware

main memory and different registers built inside the processor itself are the only primary storage that the CPU can have the right to use directly by accessing. There are some machine instructions which take

memory addresses as arguments or values, but none of them take disk addresses. So, any instructions in implementation and any data which is used by the instructions should have to be in one of these direct accessing storage devices. When the data are not in memory, they have to be moved there before the CPL can work on them.

Registers which are built into the CPU are accessible within one single cycle of the CPU clock. Most CPUs' can interpret those instructions and carry out simple operations on register contents at the rate of 1 or more process per clock tick. The same may not be said for main memory, which gets accessed via a transaction on the memory bus.

Address binding

Usually, a program inhabits on a disk in a binary executable form of a file. For executing, the program must be fetched into memory and positioned within a process (list in the queue). Depending on the usage of memory management, the process may get moved between disk and memory at the time of its execution. The processes on the disk are then waiting to be brought into main memory for implementing form the input queue. The normal method is to choose any one of the processes in the input queue and to load that process into the memory.

As the process gets executed, it is able now to access instructions and data from memory. Ultimately, the process expires, and its memory space is declared as available/free. Most systems let user process to exist in any part of the physical memory. Therefore, even if the address space of the computer begins at 00000, the first address of the user process need not have to be 00000. This approach can affect the addresses which the user program can use.

Normally, the binding of instructions and data onto memory addresses can be done at any of the step given below:

- Compile time: Compile time is the phase where the process will reside in memory and eventually absolute code can be generated.
- Load time: At compile time, when the process will reside in memory, the compiler must generate relocatable code. In that case, final binding gets delayed until load time.
- Execution time: Execution time is the time that a program or instruction takes for executing a particular task.

**A file operation**

A file is an abstract data type. For defining a file properly, we need to consider the operations that can be performed on files. The operating system can provide

system calls to create, write, read, reposition, delete, and truncate files. There are six basic file operations within an Operating system. These are:

- Creating a file: There are two steps necessary for creating a file. First, space in the file system must be found for the file. We discuss how to allocate space for the file. Second, an entry for the new file must be made in the directory.
- Writing a file: To write to a file, you make a system call specify about both the name of the file along with the information to be written to the file.
- Reading a file: To read from a file, you use a system call which specifies the name of the file and where within memory the next block of the file should be placed.
- Repositioning inside a file: The directory is then searched for the suitable entry, and the 'current-file-position' pointer is relocating to a given value. Relocating within a file need not require any actual I/O. This file operation is also termed as 'file seek.'
- Deleting a file: For deleting a file, you have to search the directory for the specific file. Deleting that file or directory release all file space so that other files can re-use that space.
- Truncating a file: The user may wish for erasing the contents of a file but keep the attributes same. Rather than deleting the file and then recreate it, this utility allows all attributes to remain unchanged

except the file length and let the user add or edit the file content.