

UNIT-1

Operating System - Overview

An **Operating System** (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

An operating system is software that enables applications to interact with a computer's hardware. The software that contains the core components of the operating system is called the **kernel**.

The primary purposes of an **Operating System** are to enable applications (software) to interact with a computer's hardware and to manage a system's hardware and software resources.

Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc. Today, Operating systems is found almost in every device like mobile phones, personal computers, mainframe computers, automobiles, TV, Toys etc.

Definitions

- ❖ An Operating System is the low-level software that supports a computer's basic functions, such as scheduling tasks and controlling peripherals.

We can refine this definition:

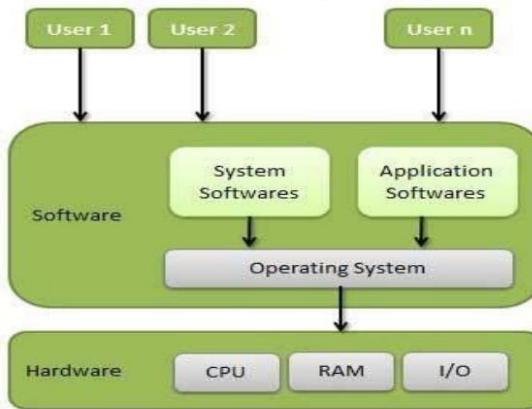
- ❖ An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.

Following is another definition:

- ❖ An operating system (OS) is system software that manages computer hardware, software resources, and provides common services for computer programs.

Architecture

We can draw a generic architecture diagram of an Operating System which is as follows:



Operating System Generations

Operating systems have been evolving over the years. We can categorise this evaluation based on different generations which is briefed below:

0th Generation

The term 0th generation is used to refer to the period of development of computing when Charles Babbage invented the Analytical Engine and later John Atanasoff created a computer

in 1940. The hardware component technology of this period was electronic vacuum tubes. There was no Operating System available for this generation computer and computer programs were written in machine language. These computers in this generation were inefficient and dependent on the varying competencies of the individual programmer as operators.

First Generation (1951-1956)

The first generation marked the beginning of commercial computing including the introduction of Eckert and Mauchly's UNIVAC I in early 1951, and a bit later, the IBM 701.

System operation was performed with the help of expert operators and without the benefit of an operating system for a time though programs began to be written in higher level, procedure-oriented languages, and thus the operator's routine expanded. Later mono-programmed operating system was developed, which eliminated some of the human intervention in running job and provided programmers with a number of desirable functions. These systems still continued to operate under the control of a human operator who used to follow a number of steps to execute a program. Programming language like FORTRAN was developed by John W. Backus in 1956.

Second Generation (1956-1964)

The second generation of computer hardware was most notably characterised by transistors replacing vacuum tubes as the hardware component technology. The first operating system GMOS was developed by the IBM computer. GMOS was based on single stream batch processing system, because it collects all similar jobs in groups or batches and then submits the jobs to the operating system using a punch card to complete all jobs in a machine. Operating system is cleaned after completing one job and then continues to read and initiates the next job in punch card.

Researchers began to experiment with multiprogramming and multiprocessing in their computing services called the time-sharing system. A noteworthy example is the Compatible Time-Sharing System (CTSS), developed at MIT during the early 1960s.

Third Generation (1964-1979)

The third generation officially began in April 1964 with IBM's announcement of its System/360 family of computers. Hardware technology began to use integrated circuits (ICs) which yielded significant advantages in both speed and economy.

Operating system development continued with the introduction and widespread adoption of multiprogramming. The idea of taking fuller advantage of the computer's data channel I/O capabilities continued to develop.

Another progress which leads to developing of personal computers in fourth generation is a new development of minicomputers with DEC PDP-1. The third generation was an exciting time, indeed, for the development of both computer hardware and the accompanying operating system.

Fourth Generation (1979 – Present)

The fourth generation is characterised by the appearance of the personal computer and the workstation. The component technology of the third generation, was replaced by very large-scale integration (VLSI). Many Operating Systems which we are using today like Windows, Linux, MacOS etc developed in the fourth generation.

Following is some of important functions of an operating System.

- Memory Management
- Processor Management
- Device Management

- File Management
- Network Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

Functions of Operating System

An Operating System acts as a communication interface between the user and computer hardware. Its purpose is to provide a platform on which a user can execute programs conveniently and efficiently. An operating system is software that manages the allocation of Computer Hardware. The coordination of the hardware must be appropriate to ensure the computer system's correct operation and to prevent user programs from interfering with it. The main goal of the Operating System is to make the computer environment more convenient to use and the Secondary goal is to use the resources most efficiently. In this article we will see functions of operating system in detail.

Why Operating Systems Used?

Operating System is used as a communication channel between the computer hardware and the user. It works as an intermediate between System Hardware and End-User. Operating System handles the following responsibilities:

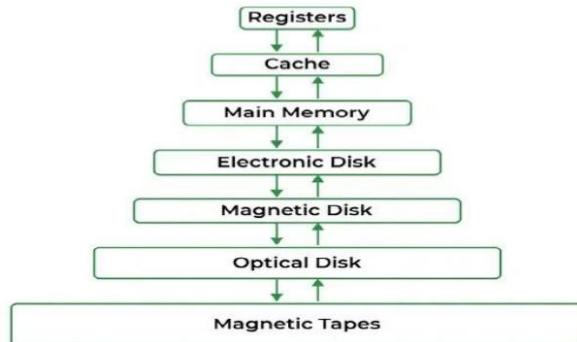
- It controls all the computer resources.
- It provides valuable services to user programs.
- It coordinates the execution of user programs.
- It provides resources for user programs.
- It provides an interface (virtual machine) to the user.
- It hides the complexity of software.
- It supports multiple execution modes.
- It monitors the execution of user programs to prevent errors.

Memory Management

Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.

Main memory provides a fast storage that can be accessed directly by the CPU. For a program to be executed, it must be in the main memory. An Operating System does the following activities for memory management –

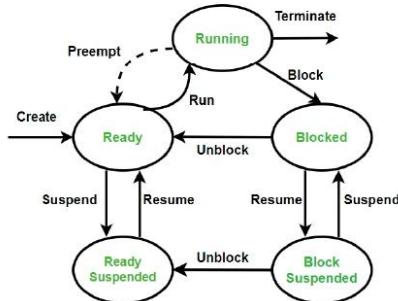
- Keeps tracks of primary memory, i.e., what part of it are in use by whom, what part are not in use.
- In multiprogramming, the OS decides which process will get memory when and how much.
- Allocates the memory when a process requests it to do so.
- De-allocates the memory when a process no longer needs it or has been terminated.



Processor Management

In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called **process scheduling**. An Operating System does the following activities for processor management –

- Keeps tracks of processor and status of process. The program responsible for this task is known as **traffic controller**.
- Allocates the processor (CPU) to a process.
- De-allocates processor when a process is no longer required.



Device Management

An Operating System manages device communication via their respective drivers. It does the following activities for device management –

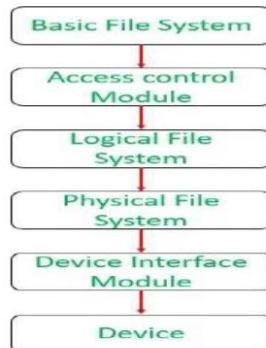
- Keeps tracks of all devices. Program responsible for this task is known as the **I/O controller**.
- Decides which process gets the device when and for how much time.
- Allocates the device in the efficient way.
- De-allocates devices.

File Management

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.

An Operating System does the following activities for file management –

- Keeps track of information, location, uses, status etc. The collective facilities are often known as **file system**.
- Decides who gets the resources.
- Allocates the resources.
- De-allocates the resources.



Other Important Activities

Following is some of the important activities that an Operating System performs –

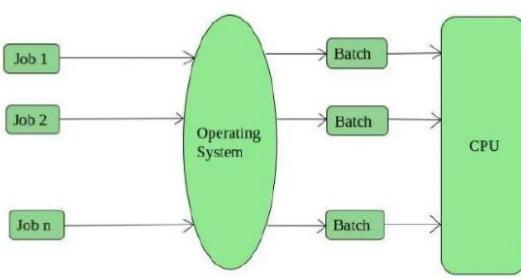
- **Security** – By means of password and similar other techniques, it prevents unauthorized access to programs and data.
- **Control over system performance** – Recording delays between request for a service and response from the system.
- **Job accounting** – Keeping track of time and resources used by various jobs and users.
- **Error detecting aids** – Production of dumps, traces, error messages, and other debugging and error detecting aids.
- **Coordination between other software's and users** – Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

Types of Operating Systems

Operating Systems can be categorized according to different criteria like whether an operating system is for mobile devices (examples Android and iOS) or desktop (examples Windows and Linux). In this article, we are going to classify based on functionalities an operating system provides.

1. Batch Operating System

This type of operating system does not interact with the computer directly. There is an operator which takes similar jobs having the same requirements and groups them into batches. It is the responsibility of the operator to sort jobs with similar needs. Batch Operating System is designed to manage and execute a large number of jobs efficiently by processing them in groups.



Batch Operating System

Advantages of Batch Operating System

- Multiple users can share the batch systems.
- The idle time for the batch system is very less.
- It is easy to manage large work repeatedly in batch systems.

Disadvantages of Batch Operating System

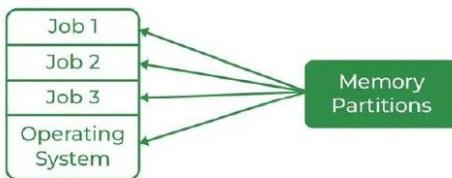
- CPU is not used efficiently. When the current process is doing IO, CPU is free and could be utilized by other processes waiting.
- The other jobs will have to wait for an unknown time if any job fails.
- In batch operating system, average response time increases as all processes are processed one by one.

Examples of Batch Operating Systems: Payroll Systems, Bank Statements, etc.

2. Multi-Programming Operating System

Multiprogramming Operating Systems can be simply illustrated as more than one program is present in the main memory and any one of them can be kept in execution. This is basically used for better utilization of resources.

Multiprogramming



Multi Programming

Advantages of Multi-Programming Operating System

- CPU is better utilized and overall performance of the system improves.
- It helps in reducing the response time.

Multi-Tasking/Time-sharing Operating systems

It is a type of Multiprogramming system with every process running in round robin manner. Each task is given some time to execute so that all the tasks work smoothly. Each user gets the time of the CPU as they use a single system. These systems are also known as Multitasking

Systems. The task can be from a single user or different users also. The time that each task gets to execute is called quantum. After this time interval is over OS switches over to the next task.

Advantages of Time-Sharing OS

- Each task gets an equal opportunity.
- Fewer chances of duplication of software.
- CPU idle time can be reduced.
- Resource Sharing: Time-sharing systems allow multiple users to share hardware resources such as the CPU, memory, and peripherals, reducing the cost of hardware and increasing efficiency.
- Improved Productivity: Time-sharing allows users to work concurrently, thereby reducing the waiting time for their turn to use the computer. This increased productivity translates to more work getting done in less time.
- Improved User Experience: Time-sharing provides an interactive environment that allows users to communicate with the computer in real time, providing a better user experience than batch processing.

Disadvantages of Time-Sharing OS

- Reliability problem.
- One must have to take care of the security and integrity of user programs and data.
- Data communication problem.
- High Overhead: Time-sharing systems have a higher overhead than other operating systems due to the need for scheduling, context switching, and other overheads that come with supporting multiple users.
- Complexity: Time-sharing systems are complex and require advanced software to manage multiple users simultaneously. This complexity increases the chance of bugs and errors.
- Security Risks: With multiple users sharing resources, the risk of security breaches increases. Time-sharing systems require careful management of user access, authentication, and authorization to ensure the security of data and software.

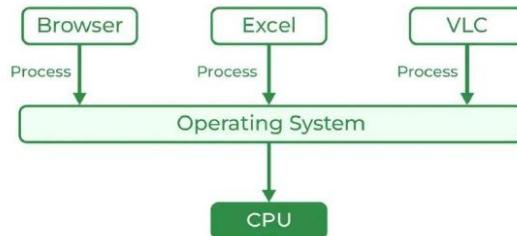
Examples of Time-Sharing OS with explanation

- **IBM VM/CMS:** IBM VM/CMS is a time-sharing operating system that was first introduced in 1972. It is still in use today, providing a virtual machine environment that allows multiple users to run their own instances of operating systems and applications.
- **TSO (Time Sharing Option) :** TSO is a time-sharing operating system that was first introduced in the 1960s by IBM for the IBM System/360 mainframe computer. It allowed multiple users to access the same computer simultaneously, running their own applications.
- **Windows Terminal Services:** Windows Terminal Services is a time-sharing operating system that allows multiple users to access a Windows server remotely. Users can run their own applications and access shared resources, such as printers and network storage, in real-time.

3. Multi-Processing Operating System

Multi-Processing Operating System is a type of Operating System in which more than one CPU is used for the execution of resources. It betters the throughput of the System.

Multitasking



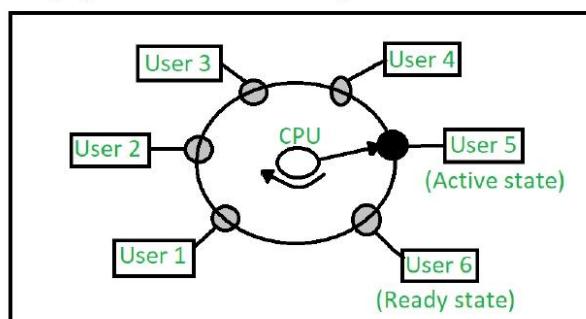
Multiprocessing Operating System

Advantages of Multi-Processing Operating System

- It increases the throughput of the system as processes can be parallelized.
- As it has several processors, so, if one processor fails, we can proceed with another processor.

4. Multi User Operating Systems

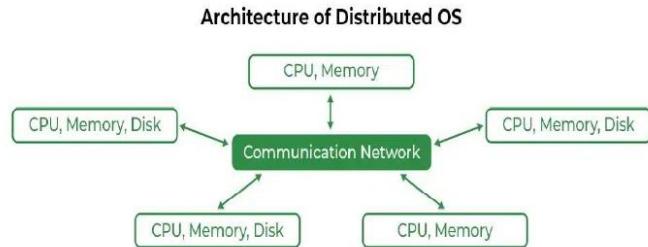
These systems allow multiple users to be active at the same time. This system can be either multiprocessor or single processor with interleaving.



Time-Sharing OS

5. Distributed Operating System

These types of operating system is a recent advancement in the world of computer technology and are being widely accepted all over the world and, that too, at a great pace. Various autonomous interconnected computers communicate with each other using a shared communication network. Independent systems possess their own memory unit and CPU. These are referred to as loosely coupled systems or distributed systems . These systems' processors differ in size and function. The major benefit of working with these types of the operating system is that it is always possible that one user can access the files or software which are not actually present on his system but some other system connected within this network i.e., remote access is enabled within the devices connected in that network.



Distributed OS

Advantages of Distributed Operating System

- Failure of one will not affect the other network communication, as all systems are independent of each other.
- Electronic mail increases the data exchange speed.
- Since resources are being shared, computation is highly fast and durable.
- Load on host computer reduces.
- These systems are easily scalable as many systems can be easily added to the network.
- Delay in data processing reduces.

Disadvantages of Distributed Operating System

- Failure of the main network will stop the entire communication.
- To establish distributed systems the language is used not well-defined yet.
- These types of systems are not readily available as they are very expensive. Not only that the underlying software is highly complex and not understood well yet.

Examples of Distributed Operating Systems are LOCUS, etc.

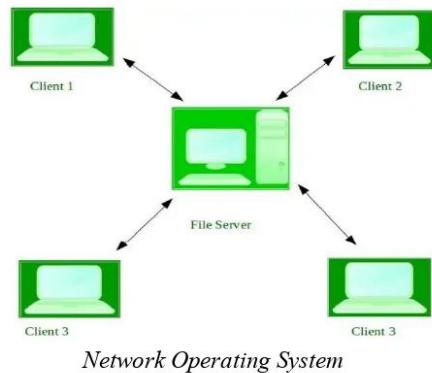
Issues With Distributed Operating Systems

- Networking causes delays in the transfer of data between nodes of a distributed system. Such delays may lead to an inconsistent view of data located in different nodes, and make it difficult to know the chronological order in which events occurred in the system.
- Control functions like scheduling, resource allocation, and deadlock detection have to be performed in several nodes to achieve computation speedup and provide reliable operation when computers or networking components fail.
- Messages exchanged by processes present in different nodes may travel over public networks and pass through computer systems that are not controlled by the distributed operating system. An intruder may exploit this feature to tamper with messages, or create fake messages to fool the authentication procedure and masquerade as a user of the system.

6. Network Operating System

These systems run on a server and provide the capability to manage data, users, groups, security, applications, and other networking functions. These types of operating systems allow shared access to files, printers, security, applications, and other networking functions over a small private network. One more important aspect of Network Operating Systems is that all the users are well aware of the underlying configuration, of all other users within the network, their individual connections, etc. and that's why these computers are popularly known as tightly coupled systems .

Fundamentals of Operating system



Network Operating System

Advantages of Network Operating System

- Highly stable centralized servers.
- Security concerns are handled through servers.
- New technologies and hardware up-gradation are easily integrated into the system.
- Server access is possible remotely from different locations and types of systems.

Disadvantages of Network Operating System

- Servers are costly.
- User has to depend on a central location for most operations.
- Maintenance and updates are required regularly.

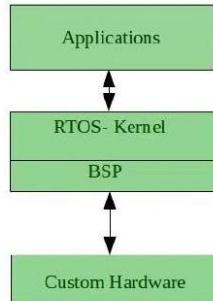
Examples of Network Operating Systems are Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, BSD, etc.

7. Real-Time Operating System

These types of OSs serve real-time systems. The time interval required to process and respond to inputs is very small. This time interval is called **response time**. **Real-time systems** are used when there are time requirements that are very strict like missile systems, air traffic control systems, robots, etc.

Types of Real-Time Operating Systems

- **Hard Real-Time Systems:** Hard Real-Time OSs are meant for applications where time constraints are very strict and even the shortest possible delay is not acceptable. These systems are built for saving life like automatic parachutes or airbags which are required to be readily available in case of an accident. Virtual memory is rarely found in these systems.
- **Soft Real-Time Systems:** These OSs are for applications where time-constraint is less strict.



Real-Time Operating System

Advantages of RTOS

- **Maximum Consumption:** Maximum utilization of devices and systems, thus more output from all the resources.
- **Task Shifting:** The time assigned for shifting tasks in these systems is very less. For example, in older systems, it takes about 10 microseconds in shifting from one task to another, and in the latest systems, it takes 3 microseconds.
- **Focus on Application:** Focus on running applications and less importance on applications that are in the queue.
- Real-time **operating system in the embedded system:** Since the size of programs is small, RTOS can also be used in embedded systems like in transport and others.
- **Error Free:** These types of systems are error-free.
- **Memory Allocation:** Memory allocation is best managed in these types of systems.

Disadvantages of RTOS

- **Limited Tasks:** Very few tasks run at the same time and their concentration is very less on a few applications to avoid errors.
- **Use heavy system resources:** Sometimes the system resources are not so good and they are expensive as well.
- **Complex Algorithms:** The algorithms are very complex and difficult for the designer to write on.
- **Device driver and interrupt signals:** It needs specific device drivers and interrupts signal to respond earliest to interrupts.
- **Thread Priority:** It is not good to set thread priority as these systems are very less prone to switching tasks.

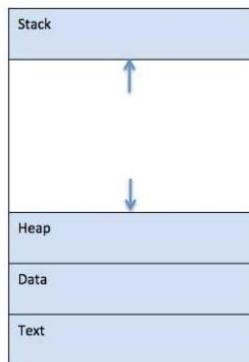
Examples of Real-Time Operating Systems are Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

Concept of Process:

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

“A process is defined as an entity which represents the basic unit of work to be implemented in the system.”

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program. When a program is loaded into the memory and it becomes a process, it can be divided into four sections – stack, heap, text and data. The following image shows a simplified layout of a process inside main memory –



S.N.	Component & Description
1	Stack The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2	Heap This is dynamically allocated memory to a process during its run time.
3	Text This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4	Data This section contains the global and static variables.

Program

A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. For example, here is a simple program written in C programming language –

```
#include <stdio.h>

int main() {
    printf("Hello, World! \n");
    return 0;
}
```

A computer program is a collection of instructions that performs a specific task when executed by a computer. When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program.

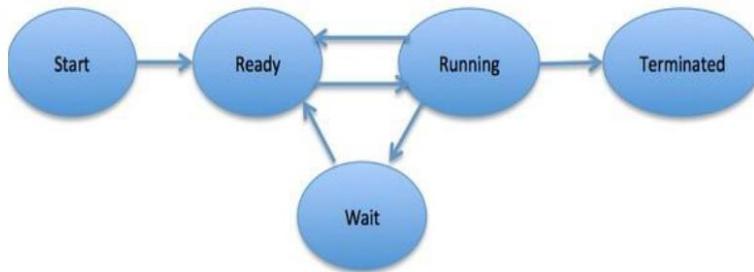
A part of a computer program that performs a well-defined task is known as an **algorithm**. A collection of computer programs, libraries and related data are referred to as a **software**.

Process Life Cycle

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

In general, a process can have one of the following five states at a time.

S.N.	State & Description
1	Start This is the initial state when a process is first started/created.
2	Ready The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after Start state or while running it by but interrupted by the scheduler to assign CPU to some other process.
3	Running Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.
4	Waiting Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
5	Terminated or Exit Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.



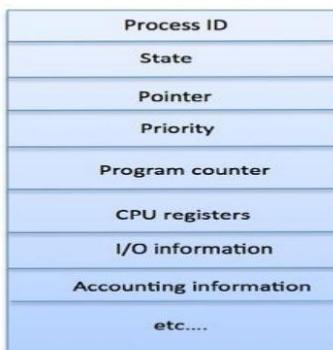
Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table –

S.N.	Information & Description
1	Process State The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2	Process privileges This is required to allow/disallow access to system resources.
3	Process ID Unique identification for each of the process in the operating system.
4	Pointer A pointer to parent process.

5	Program Counter Program Counter is a pointer to the address of the next instruction to be executed for this process.
6	CPU registers Various CPU registers where process need to be stored for execution for running state.
7	CPU Scheduling Information Process priority and other scheduling information which is required to schedule the process.
8	Memory management information This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9	Accounting information This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	IO status information This includes a list of I/O devices allocated to the process.

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB –



The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

What is Thread?

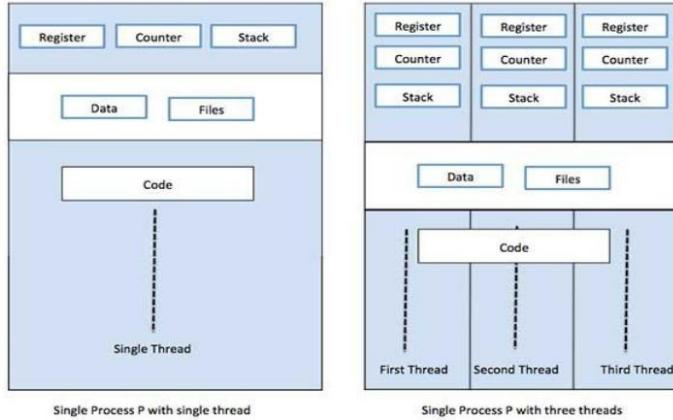
A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

A thread is also called a **lightweight process**. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for

parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.



Difference between Process and Thread

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

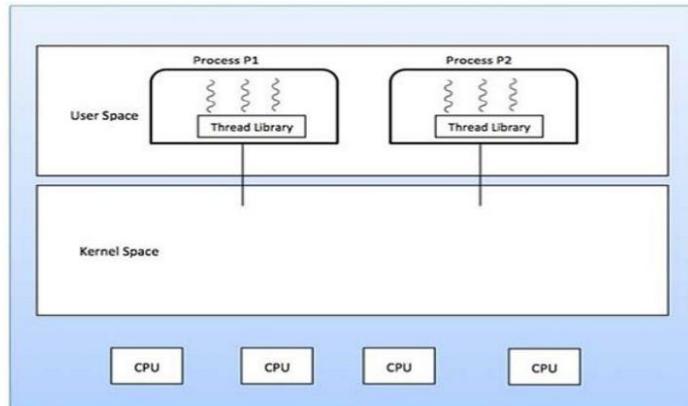
Types of Thread

Threads are implemented in following two ways –

- **User Level Threads** – User managed threads.
- **Kernel Level Threads** – Operating System managed threads acting on kernel, an operating system core.

User Level Threads

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.



Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

Kernel Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individuals' threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

Multithreading Models

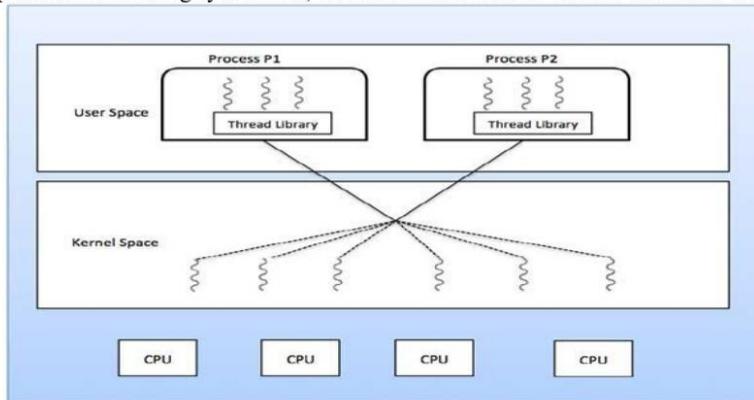
Some operating system provides a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

- Many to many relationships.
- Many to one relationship.
- One to one relationship.

Many to Many Model

The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.

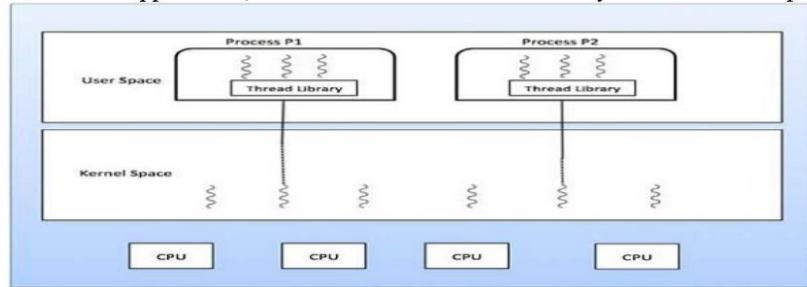
The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine. This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.



Many to One Model

Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

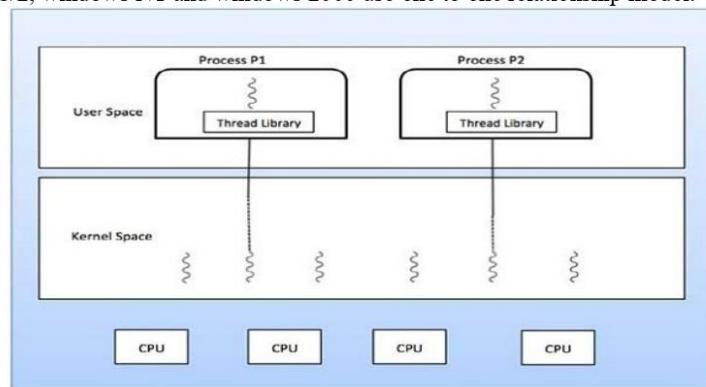
If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.



One to One Model

There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model. It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.



Difference between User-Level & Kernel-Level Thread

S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

Operating System - Process Scheduling

Definition

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Categories of Scheduling

There are two categories of scheduling:

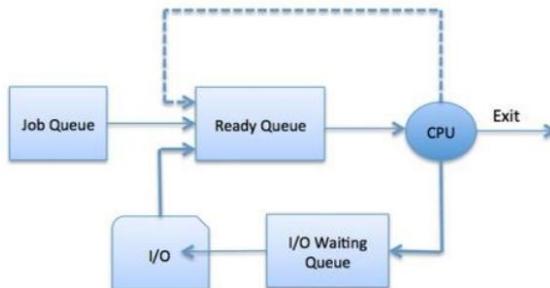
1. **Non-pre-emptive:** Here the resource can't be taken from a process until the process completes execution. The switching of resources occurs when the running process terminates and moves to a waiting state.
2. **Pre-emptive:** Here the OS allocates the resources to a process for a fixed amount of time. During resource allocation, the process switches from running state to ready state or from waiting state to ready state. This switching occurs as the CPU may give priority to other processes and replace the process with higher priority with the running process.

Process Scheduling Queues

The OS maintains all Process Control Blocks (PCBs) in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues –

- **Job queue** – This queue keeps all the processes in the system.
- **Ready queue** – This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** – The processes which are blocked due to unavailability of an I/O device constitute this queue.



The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

Schedulers

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types –

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long-term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

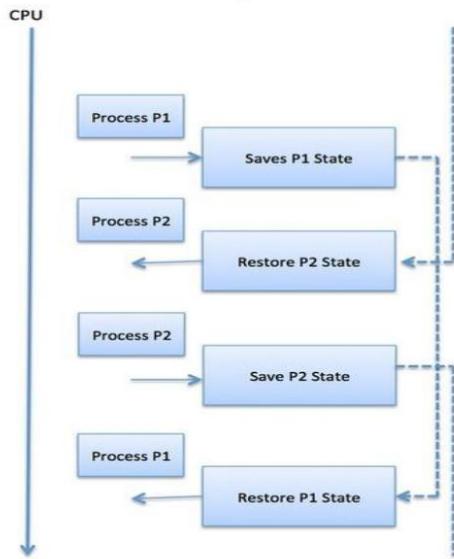
Comparison among Scheduler

S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short- and long-term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time-sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

Context Switching

A context switching is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.



Context switches are computationally intensive since register and memory state must be saved and restored. To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers. When the process is switched, the following information is stored for later use.

- Program Counter
- Scheduling information
- Base and limit register value
- Currently used register
- Changed State
- I/O State information
- Accounting information

Scheduling algorithms-

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which we are going to discuss in this chapter –

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin (RR) Scheduling

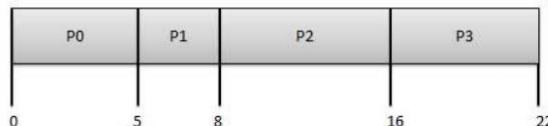
These algorithms are either **non-pre-emptive** or **pre-emptive**. Non-pre-emptive algorithms are designed so that once a process enters the running state, it cannot be pre-empted until it completes its allotted time, whereas the pre-emptive scheduling is based

on priority where a scheduler may pre-empt a low priority running process anytime when a high priority process enters into a ready state.

First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-pre-emptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process is as follows –

Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

Average Wait Time: $(0+4+6+13) / 4 = 5.75$

Shortest Job Next (SJN)

- This is also known as **shortest job first**, or SJF
- This is a non-pre-emptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processor should know in advance how much time process will take.

Given: Table of processes, and their Arrival time, Execution time

Process	Arrival Time	Execution Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	14
P3	3	6	8

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8



Waiting time of each process is as follows –

Process	Waiting Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	14 - 2 = 12
P3	8 - 3 = 5

Average Wait Time: $(0 + 4 + 12 + 5)/4 = 21 / 4 = 5.25$

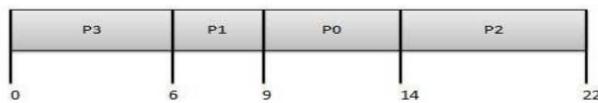
Priority Based Scheduling

- Priority scheduling is a non-pre-emptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Given: Table of processes, and their Arrival time, Execution time, and priority. Here we are considering 1 is the lowest priority.

Process	Arrival Time	Execution Time	Priority	Service Time
P0	0	5	1	0
P1	1	3	2	11
P2	2	8	1	14
P3	3	6	3	5

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Waiting time of each process is as follows –

Process	Waiting Time
P0	0 - 0 = 0
P1	11 - 1 = 10
P2	14 - 2 = 12
P3	5 - 3 = 2

Average Wait Time: $(0 + 10 + 12 + 2)/4 = 24 / 4 = 6$

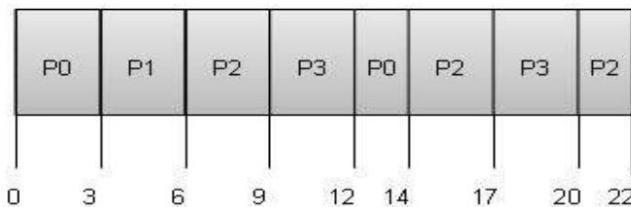
Shortest Remaining Time

- Shortest remaining time (SRT) is the pre-emptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be pre-empted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

Round Robin Scheduling

- Round Robin is the pre-emptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is pre-empted and other process executes for a given time period.
- Context switching is used to save states of pre-empted processes.

Quantum = 3



Wait time of each process is as follows –

Process	Wait Time : Service Time - Arrival Time
P0	$(0 - 0) + (12 - 3) = 9$
P1	$(3 - 1) = 2$
P2	$(6 - 2) + (14 - 9) + (20 - 17) = 12$
P3	$(9 - 3) + (17 - 12) = 11$

Average Wait Time: $(9+2+12+11) / 4 = 8.5$