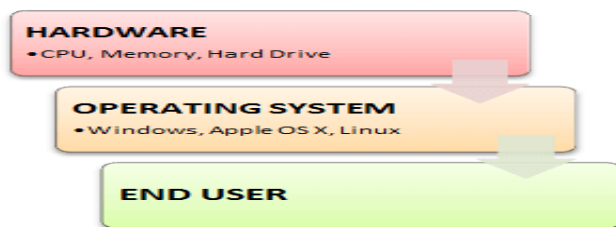


Operating System (OS)

An Operating System (OS) is a System Software that provides 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.

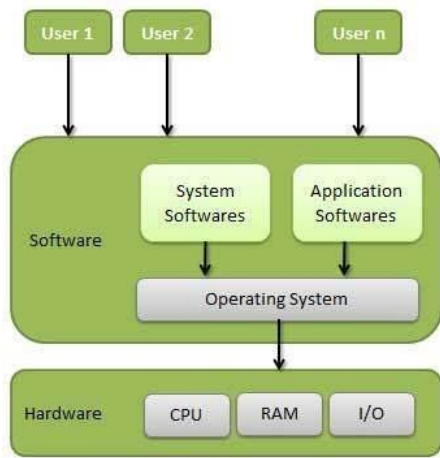
Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

The OS helps you to communicate with the computer without knowing how to speak the computer's language. It is not possible for the user to use any computer or mobile device without having an operating system.



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 by acting as Resource Manager.



Following are some of important functions of an operating System.

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

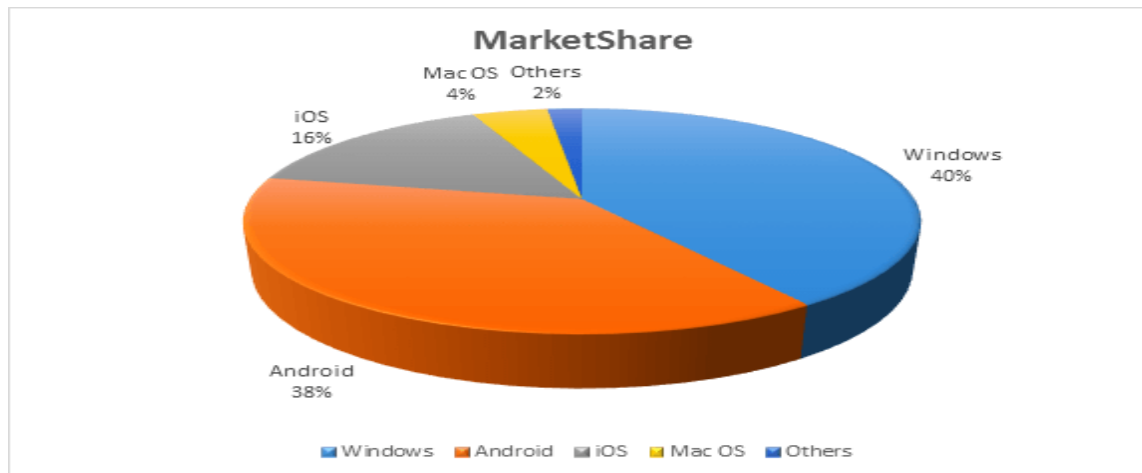
History of OS

- Operating systems were first developed in the late 1950s to manage tape storage
- The General Motors Research Lab implemented the first OS in the early 1950s for their IBM 701
- In the mid-1960s, operating systems started to use disks
- In the late 1960s, the first version of the Unix OS was developed
- The first OS built by Microsoft was DOS. It was built in 1981 by purchasing the

86-DOS software from a Seattle company

- The present-day popular OS Windows first came to existence in 1985 when a GUI was created and paired with MS-DOS.

Examples of Operating System with Market Share



Market Share of Operating Systems

Following are the Operating System examples with the latest Market Share

OS Name	Share
Windows	40.34
Android	37.95
iOS	15.44
Mac OS	4.34
Linux	0.95
Chrome OS	0.14
Windows Phone OS	0.06

Types of Operating System (OS)

Following are the popular types of OS (Operating System):

- Batch Operating System
- Multitasking/Time Sharing OS
- Multiprocessing OS
- Real Time OS
- Distributed OS
- NetworkOS
- Mobile OS

1. Batch Operating System

Some computer processes are very lengthy and time-consuming. To speed the same process, a job with a similar type of needs are batched together and run as a group.

The user of a batch operating system never directly interacts with the computer. In this type of OS, every user prepares his or her job on an offline device like a punch card and submit it to the computer operator.

2. Multi-Tasking/Time-sharing Operating systems

Time-sharing operating system enables people located at a different terminal(shell) to use a single computer system at the same time. The processor time (CPU) which is shared among multiple users is termed as time sharing.

3. Real time OS

A real time operating system time interval to process and respond to inputs is very small. Examples: Military Software Systems, Space Software Systems are the Real time OS example.

4. Distributed OperatingSystem

Distributed systems use many processors located in different machines to provide very fast computationtoitsusers.

5. Network Operating System

Network Operating System runs on a server. It provides the capability to serve to managedata,user,groups,security,application,andothernetworkingfunctions.

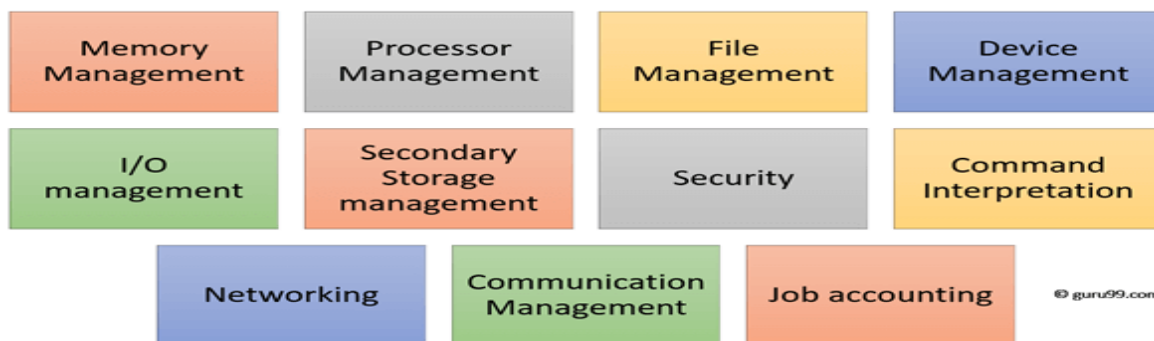
6. Mobile OS

Mobile operating systems are those OS which is especially that are designed to power smartphones, tablets, and wearables devices.

Some most famous mobile operating systems are Android and iOS, but others include BlackBerry, Web, and watchOS.

FunctionsofOperatingSystem

Some typical operating system functions may include managing memory, files, processes, I/O system & devices, security, etc. Below are the main functions of OperatingSystem:



Functions of Operating System

In an operating system software performs each of the function:

1. **Process management:** Process management helps OS to create and delete processes. It also provides mechanisms for synchronization and communication among processes.
2. **Memory management:** Memory management module performs the task of

allocation and de-allocation of memory space to programs in need of this resources.

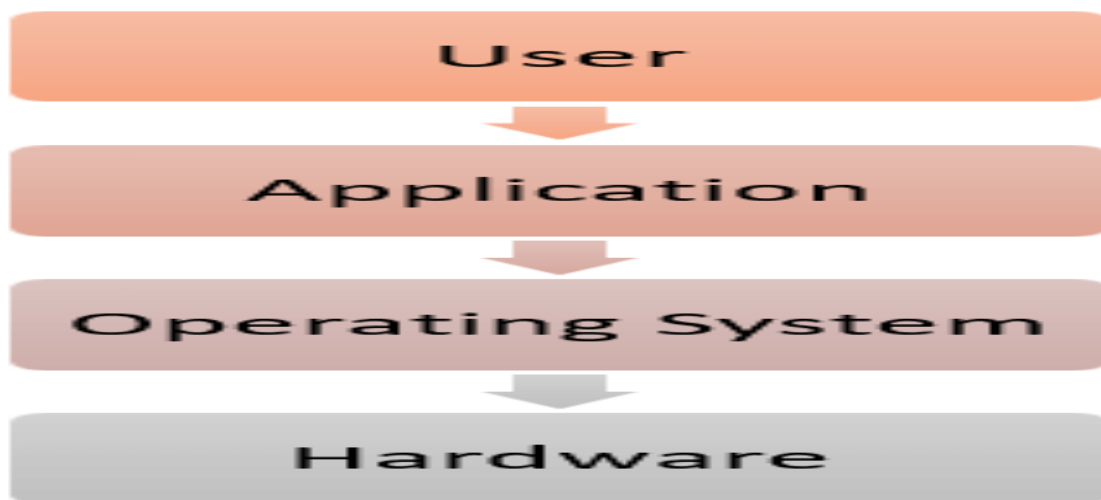
3. **File management:** It manages all the file-related activities such as organization storage, retrieval, naming, sharing, and protection of files.
4. **Device Management:** Device management keeps tracks of all devices. This module also responsible for this task is known as the I/O controller. It also performs the task of allocation and de-allocation of the devices.
5. **I/O System Management:** One of the main objects of any OS is to hide the peculiarities of that hardware devices from the user.
6. **Secondary-Storage Management:** Systems have several levels of storage which includes primary storage, secondary storage, and cache storage. Instructions and data must be stored in primary storage or cache so that a running program can reference it.
7. **Security:** Security module protects the data and information of a computer system against malware threat and authorized access.
8. **Command interpretation:** This module is interpreting commands given by the and acting system resources to process that commands.
9. **Networking:** A distributed system is a group of processors which do not share memory, hardware devices, or a clock. The processors communicate with one another through the network.
10. **Job accounting:** Keeping track of time & resource used by various job and users.
11. **Communication management:** Coordination and assignment of compilers,

interpreters, and another software resource of the various users of the computer systems.

Features of Operating System (OS)

Here is a list important feature of OS:

- Protected and supervisor mode
- Allows disk access and file systems Device drivers Networking Security
- Program Execution
- Memory management Virtual Memory Multitasking
- Handling I/O operations
- Manipulation of the filesystem
- Error Detection and handling
- Resource allocation
- Information and Resource Protection



Advantage of Operating System

- Allows you to hide details of hardware by creating an abstraction

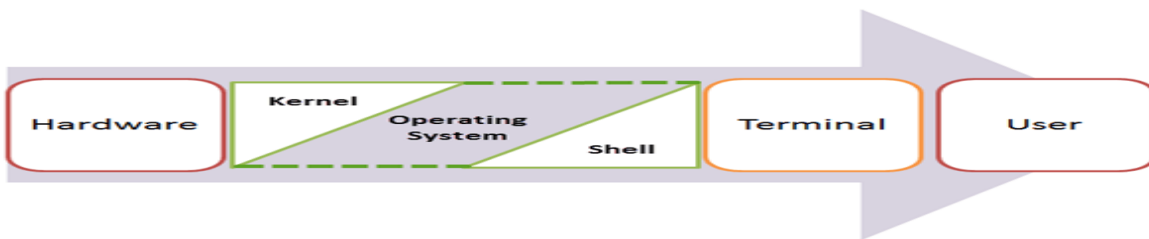
- Easy to use with a GUI
- Offers an environment in which a user may execute programs/applications
- The operating system must make sure that the computer system convenient to use
- Operating System acts as an intermediary among applications and the hardware components
- It provides the computer system resources with easy to use format
- Acts as an intermediary between all hardware's and software's of the system

Disadvantages of Operating System

- If any issue occurs in OS, you may lose all the contents which have been stored in your system
- Operating system's software is quite expensive for small size organization which adds burden on them. Example Windows
- It is never entirely secure as a threat can occur at any time

What is Kernel in Operating System?

The kernel is the central component of computer operating systems. The only job performed by the kernel is to manage the communication between the software and the hardware. A Kernel is at the nucleus of a computer. It makes the communication between the hardware and software possible. While the Kernel is the innermost part of an operating system, a shell is the outermost one.



Introduction to Kernel

Features of Kernel

- Low-level scheduling of processes
- Inter-process communication
- Process synchronization
- Context switching

Types of Kernel

There are many types of kernels that exist, but among them, the two most popular kernels are:

1. Monolithic

A monolithic kernel is a single code or block of the program. It provides all the required services offered by the operating system. It is a simplistic design which creates a distinct communication layer between the hardware and software.

2. Microkernels

Microkernel manages all system resources. In this type of kernel, services are implemented in different address space. The user services are stored in user

address space, and kernel services are stored under kernel address space. So, it helps to reduce the size of both the kernel and operating system.

Difference between Firmware and Operating System

Below are the Key Differences between Firmware and Operating System:

Firmware	OperatingSystem
Define Firmware: Firmware is one kind of programming that is embedded on a chip in the device which controls that specific device.	Define Operating System: OS provides functionality over and above that which is provided by the firmware.
Firmware is programs that been encoded by the manufacture of the IC or something and cannot be changed.	OS is a program that can be installed by the user and can be changed.
It is stored on non-volatile memory.	OS is stored on the hard drive.

Difference between 32-Bit and 64-Bit Operating System

Parameters	32. Bit	64. Bit
Architecture and Software	Allow 32 bit of data processing simultaneously	Allow 64 bit of data processing simultaneously

Compatibility	32-bit applications require 32-bit OS and CPUs.	64-bit applications require a 64-bit OS and CPU.
Systems Available	All versions of Windows 8, Windows 7, Windows Vista, and Windows XP, Linux, etc.	Windows XP Professional, Vista, 7, Mac OS X and Linux.
Memory Limits	32-bit systems are limited to 3.2 GB of RAM.	64-bit systems allow a maximum 17 Billion GB of RAM.

Generations of Operating Systems

Operating systems are there from the very first computer generation and they keep evolving with time. In this chapter, we will discuss some of the important types of operating systems which are most commonly used.

1. Batch operating system

The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.

The problems with Batch Systems are as follows –

- Lack of interaction between the user and the job.
- CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.
- Difficult to provide the desired priority.

2. Time-sharing operating systems

Time-sharing is a technique which enables many people, located at various terminals, to use a

particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of Multiprogrammed batch systems, the objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.

Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation. That is, if n users are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most.

The operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Advantages of Timesharing operating systems are as follows –

- Provides the advantage of quick response.
- Avoids duplication of software.
- Reduces CPU idle time.

Disadvantages of Time-sharing operating systems are as follows –

- Problem of reliability.
- Question of security and integrity of user programs and data.
- Problem of data communication.

3. Distributed operating System

Distributed systems use multiple central processors to serve multiple real-time applications and multiple users. Data processing jobs are distributed among the processors accordingly.

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as **loosely coupled systems** or

distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers, and so on.

The advantages of distributed systems are as follows –

- With resource sharing facility, a user at one site may be able to use the resources available at another.
- Speedup the exchange of data with one another via electronic mail.
- If one site fails in a distributed system, the remaining sites can potentially continue operating.
- Better service to the customers.
- Reduction of the load on the host computer.
- Reduction of delays in data processing.

4. Network operating System

A Network Operating System runs on a server and provides the server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.

Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

The advantages of network operating systems are as follows –

- Centralized servers are highly stable.
- Security is server managed.
- Upgrades to new technologies and hardware can be easily integrated into the system.
- Remote access to servers is possible from different locations and types of systems.

The disadvantages of network operating systems are as follows –

- High cost of buying and running a server.
- Dependency on a central location for most operations.
- Regular maintenance and updates are required.

5. Real Time operating System

A real-time system is defined as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. The time taken by the system to respond to an input and display of required updated information is termed as the **response time**. So in this method, the response time is very less as compared to online processing.

Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail. For example, Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

There are two types of real-time operating systems.

A. Hard real-time systems

Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems, secondary storage is limited or missing and the data is stored in ROM. In these systems, virtual memory is almost never found.

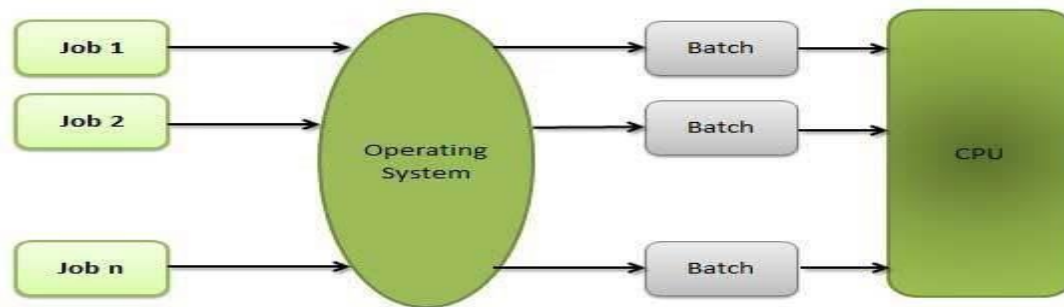
B. Soft real-time systems

Soft real-time systems are less restrictive. A critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems. For example, multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers, etc.

Batch processing

Batch processing is a technique in which an Operating System collects the programs and data together in a batch before processing starts. An operating system does the following activities related to batch processing –

- The OS defines a job which has predefined sequence of commands, programs and data as a single unit.
- The OS keeps a number a jobs in memory and executes them without any manual information.
- Jobs are processed in the order of submission, i.e., first come first served fashion.
- When a job completes its execution, its memory is released and the output for the job gets copied into an output spool for later printing or processing.



Advantages

- Batch processing takes much of the work of the operator to the computer.
- Increased performance as a new job get started as soon as the previous job is finished, without any manual intervention.

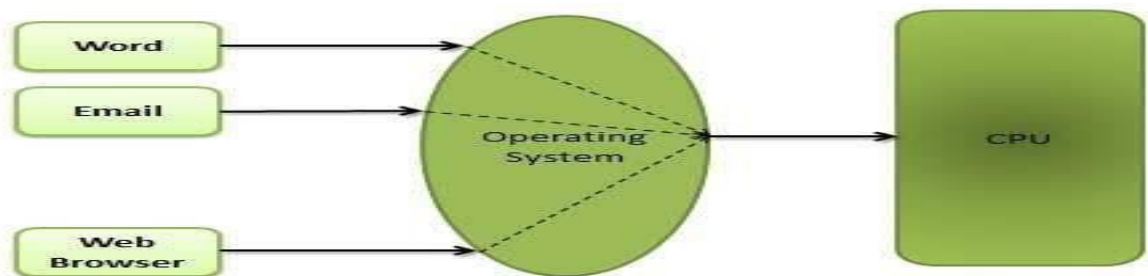
Disadvantages

- Difficult to debug program.
- A job could enter an infinite loop.
- Due to lack of protection scheme, one batch job can affect pending jobs.

Multitasking

Multitasking is when multiple jobs are executed by the CPU simultaneously by switching between them. Switches occur so frequently that the users may interact with each program while it is running. An OS does the following activities related to multitasking –

- The user gives instructions to the operating system or to a program directly, and receives an immediate response.
- The OS handles multitasking in the way that it can handle multiple operations/executes multiple programs at a time.
- Multitasking Operating Systems are also known as Time-sharing systems.
- These Operating Systems were developed to provide interactive use of a computer system at a reasonable cost.
- A time-shared operating system uses the concept of CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared CPU.
- Each user has at least one separate program in memory.



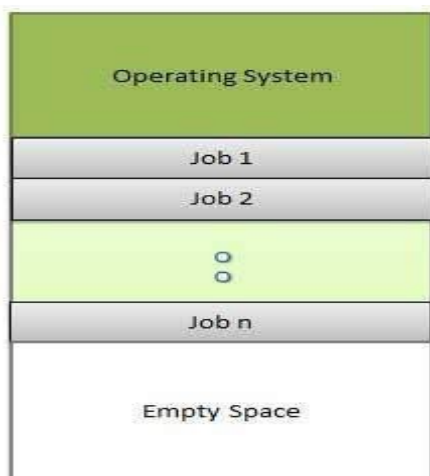
- A program that is loaded into memory and is executing is commonly referred to as a **process**.
- When a process executes, it typically executes for only a very short time before it either finishes or needs to perform I/O.
- Since interactive I/O typically runs at slower speeds, it may take a long time to complete. During this time, a CPU can be utilized by another process.

- The operating system allows the users to share the computer simultaneously.
Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user.
- As the system switches CPU rapidly from one user/program to the next, each user is given the impression that he/she has his/her own CPU, whereas actually one CPU is being shared among many users.

Multiprogramming

Sharing the processor, when two or more programs reside in memory at the same time, is referred as **multiprogramming**. Multiprogramming assumes a single shared processor. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.

The following figure shows the memory layout for a multiprogramming system.



An OS does the following activities related to multiprogramming.

- The operating system keeps several jobs in memory at a time.
- This set of jobs is a subset of the jobs kept in the job pool.
- The operating system picks and begins to execute one of the jobs in the memory.
- Multiprogramming operating systems monitor the state of all active programs and system resources using memory management programs to ensure that the CPU is never idle, unless there are no jobs to process.

Advantages

- High and efficient CPU utilization.
- User feels that many programs are allotted CPU almost simultaneously.

Disadvantages

- CPU scheduling is required.
- To accommodate many jobs in memory, memory management is required.

Interactivity

Interactivity refers to the ability of users to interact with a computer system. An Operating system does the following activities related to interactivity –

- Provides the user an interface to interact with the system.
- Manages input devices to take inputs from the user. For example, keyboard.
- Manages output devices to show outputs to the user. For example, Monitor.

The response time of the OS needs to be short, since the user submits and waits for the result.

Real Time System

Real-time systems are usually dedicated, embedded systems. An operating system does the following activities related to real-time system activity.

- In such systems, Operating Systems typically read from and react to sensor data.
- The Operating system must guarantee response to events within fixed periods of time to ensure correct performance.

Distributed Environment

A distributed environment refers to multiple independent CPUs or processors in a computer system. An operating system does the following activities related to distributed

environment –

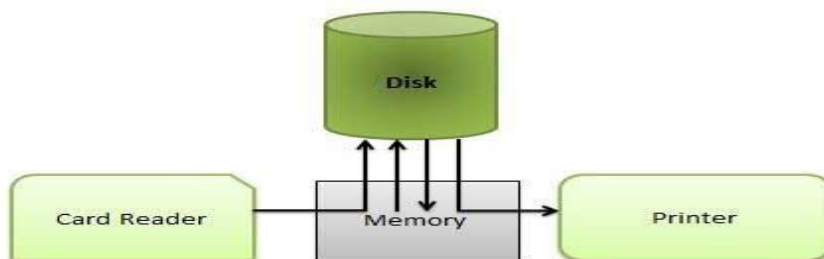
- The OS distributes computation logics among several physical processors.
- The processors do not share memory or a clock. Instead, each processor has its own local memory.
- The OS manages the communications between the processors. They communicate with each other through various communication lines.

Spooling

Spooling is an acronym for simultaneous peripheral operations on line. Spooling refers to putting data of various I/O jobs in a buffer. This buffer is a special area in memory or hard disk which is accessible to I/O devices.

An operating system does the following activities related to distributed environment –

- Handles I/O device data spooling as devices have different data access rates.
- Maintains the spooling buffer which provides a waiting station where data can rest while the slower device catches up.
- Maintains parallel computation because of spooling process as a computer can perform I/O in parallel fashion. It becomes possible to have the computer read data from a tape, write data to disk and to write out to a tape printer while it is doing its computing task.



Advantages

- The spooling operation uses a disk as a very large buffer.

- Spooling is capable of overlapping I/O operation for one job with processor operations for another job.

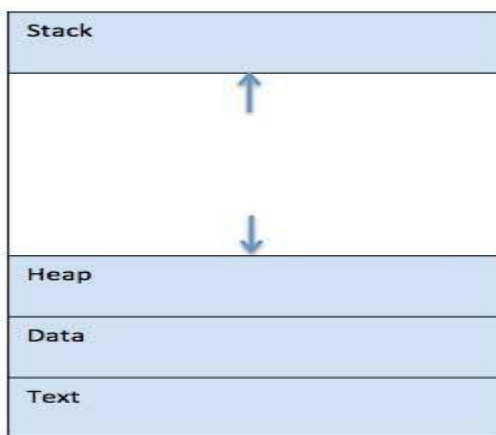
Process

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 –



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

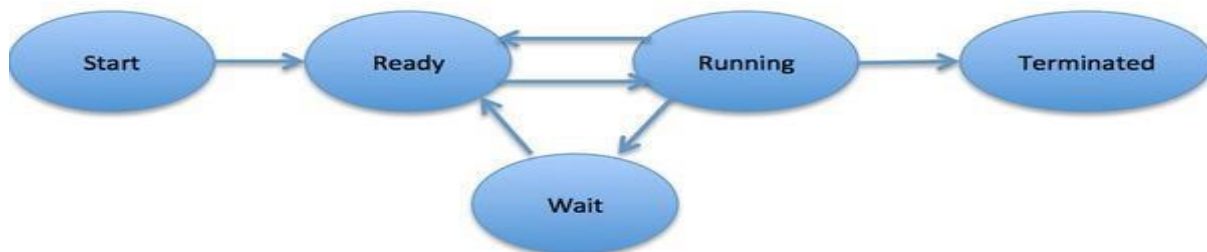
When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized. In general, a process can have one of the following five states at a time.

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 STATE

Processes can be any of the following states :

- **New** - The process is in the stage of being created.
- **Ready** - The process has all the resources available that it needs to run, but the CPU is not currently working on this process's instructions.
- **Running** - The CPU is working on this process's instructions.
- **Waiting** - The process cannot run at the moment, because it is waiting for some resource to become available or for some event to occur.
- **Terminated** - The process has completed.



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.

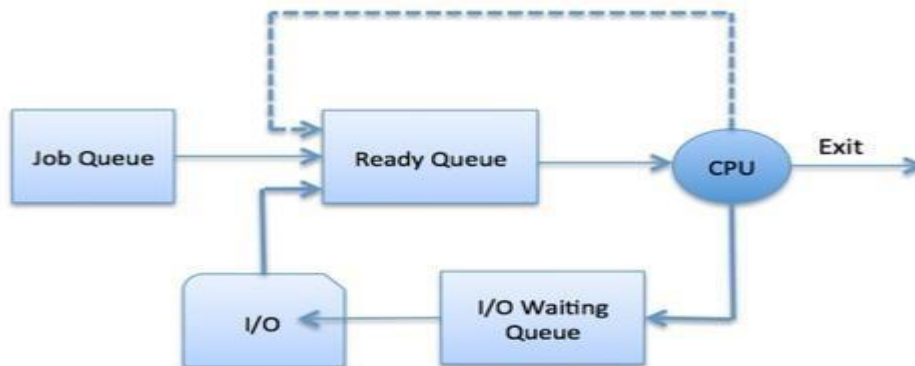
Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Process Scheduling Queues

The OS maintains all 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

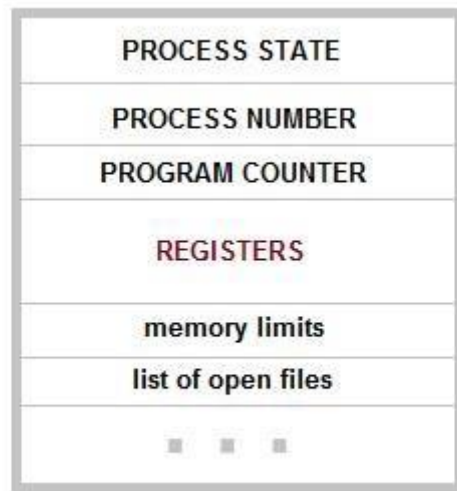
S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term

			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.

PROCESS CONTROL BLOCK

There is a Process Control Block for each process, enclosing all the information about the process. It is a data structure, which contains the following :

- Process State - It can be running, waiting etc.
- Process ID and parent process ID.
- CPU registers and Program Counter. **Program Counter** holds the address of the next instruction to be executed for that process.
- CPU Scheduling information - Such as priority information and pointers to scheduling queues.
- Memory Management information - Eg. page tables or segment tables.
- Accounting information - user and kernel CPU time consumed, account numbers, limits, etc.
- I/O Status information - Devices allocated, open file tables, etc.



Process Scheduling

The act of determining which process in the ready state should be moved to the running state is known as Process Scheduling.

The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs. For achieving this, the scheduler must apply appropriate rules for swapping processes IN and OUT of CPU.

Schedulers fall into one of the two general categories :

- **Non pre-emptive scheduling.** When the currently executing process gives up the CPU voluntarily.
- **Pre-emptive scheduling.** When the operating system decides to favour another process, pre-empting the currently executing process.

Scheduling Queues

- All processes when enters into the system are stored in the **job queue**.
- Processes in the Ready state are placed in the **ready queue**.
- Processes waiting for a device to become available are placed in **device queues**. There are unique device queues for each I/O device available.

Types of Schedulers

There are three types of schedulers available :

1. Long Term Scheduler :

Long term scheduler runs less frequently. Long Term Schedulers decide which program must get into the job queue. From the job queue, the Job Processor, selects processes and loads them into the memory for execution. Primary aim of the Job Scheduler is to maintain a good degree of Multiprogramming. An optimal degree of Multiprogramming means the average rate of process creation is equal to the average departure rate of processes from the execution memory.

2. Short Term Scheduler :

This is also known as CPU Scheduler and runs very frequently. The primary aim of this scheduler is to enhance CPU performance and increase process execution rate.

3. Medium Term Scheduler :

During extra load, this scheduler picks out big processes from the ready queue for some time, to allow smaller processes to execute, thereby reducing the number of processes in the ready queue.

Operations on Process

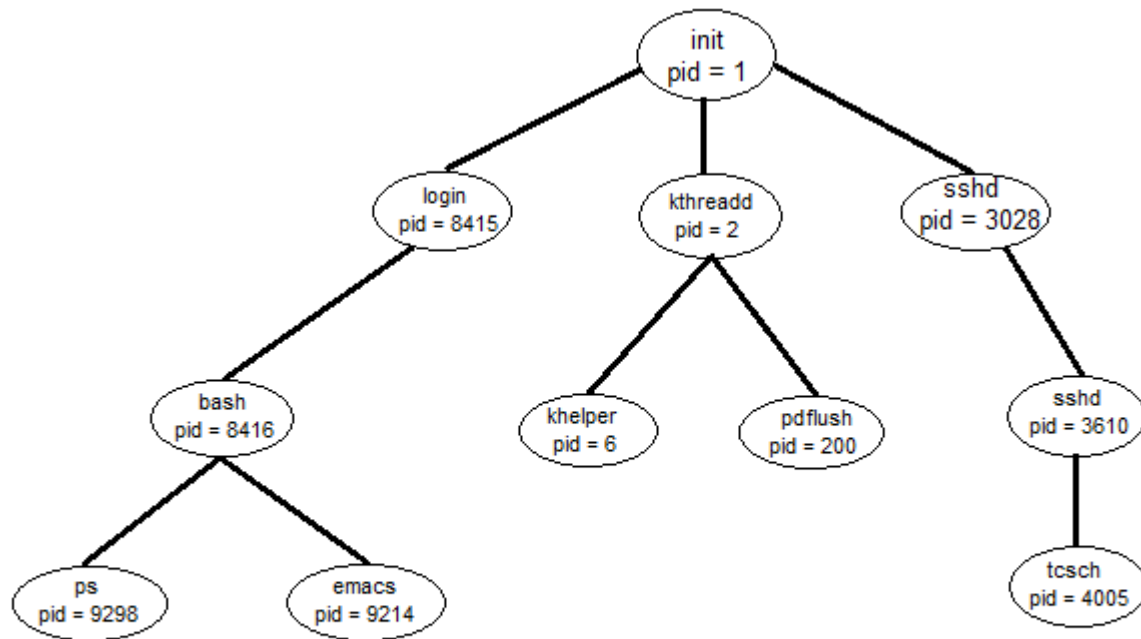
Process Creation

Through appropriate system calls, such as fork or spawn, processes may create other processes. The process which creates other process, is termed the **parent** of the other process, while the created sub-process is termed its **child**.

Each process is given an integer identifier, termed as process identifier, or PID. The parent PID (PPID) is also stored for each process.

On a typical UNIX systems the process scheduler is termed as **sched**, and is given PID 0. The first thing done by it at system start-up time is to launch **init**, which gives that process PID

1. Further Init launches all the system daemons and user logins, and becomes the ultimate parent of all other processes.



A Tree of processes on a typical Linux system

A child process may receive some amount of shared resources with its parent depending on system implementation. To prevent runaway children from consuming all of a certain system resource, child processes may or may not be limited to a subset of the resources originally allocated to the parent.

There are two options for the parent process after creating the child :

- Wait for the child process to terminate before proceeding. Parent process makes a `wait()` system call, for either a specific child process or for any particular child process, which causes the parent process to block until the `wait()` returns. UNIX shells normally wait for their children to complete before issuing a new prompt.
- Run concurrently with the child, continuing to process without waiting. When a UNIX shell runs a process as a background task, this is the operation seen. It is also possible for the parent to run for a while, and then wait for the child later, which might occur in a sort of a parallel processing operation.

Process Termination

By making the `exit`(system call), typically returning an int, processes may request their own termination. This int is passed along to the parent if it is doing a `wait()`, and is typically zero on successful completion and some non-zero code in the event of any problem.

Processes may also be terminated by the system for a variety of reasons, including :

- The inability of the system to deliver the necessary system resources.
- In response to a KILL command or other unhandled process interrupts.
- A parent may kill its children if the task assigned to them is no longer needed i.e. if the need of having a child terminates.
- If the parent exits, the system may or may not allow the child to continue without a parent (In UNIX systems, orphaned processes are generally inherited by `init`, which then proceeds to kill them.)

When a process ends, all of its system resources are freed up, open files flushed and closed, etc. The process termination status and execution times are returned to the parent if the parent is waiting for the child to terminate, or eventually returned to `init` if the process already became an orphan.

The processes which are trying to terminate but cannot do so because their parent is not waiting for them are termed **zombies**. These are eventually inherited by `init` as orphans and killed off.

CPU Scheduling

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

Scheduling Criteria

There are many different criterias to check when considering the "best" scheduling algorithm :

- **CPU utilization:** To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time(Ideally 100% of the time). Considering a real

system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

- **Throughput:** It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.
- **Turnaround time:** It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process(Wall clock time).
- **Waiting time:** The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.
- **Load average:** It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.
- **Response time:** Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

Scheduling Algorithms

Scheduling algorithms or scheduling policies are mainly used for short-term scheduling. Scheduling algorithms decide which of the processes in the ready queue is to be allocated to the CPU is basis on the type of scheduling policy and whether that policy is either preemptive or non-preemptive.

Preemptive scheduling: The preemptive scheduling is prioritized. The highest priority process should always be the process that is currently utilized.

Non-Preemptive scheduling: When a process enters the state of running, the state of that process is not deleted from the scheduler until it finishes its service time.

Scheduling Algorithms

We'll discuss four major scheduling algorithms here which are following :

1. First Come First Serve(FCFS) Scheduling
2. Shortest-Job-First(SJF) Scheduling
3. Priority Scheduling
4. Round Robin(RR) Scheduling
5. Multilevel Queue Scheduling

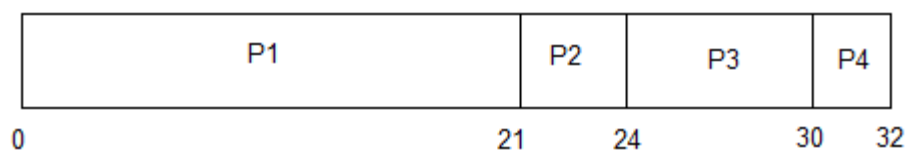
First Come First Serve(FCFS) Scheduling

- Jobs are executed on first come, first serve basis.
- Easy to understand and implement.
- Poor in performance as average wait time is high.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2



The average waiting time will be = $(0 + 21 + 24 + 30) / 4 = \underline{18.75 \text{ ms}}$



This is the GANTT chart for the above processes

Shortest-Job-First(SJF) Scheduling

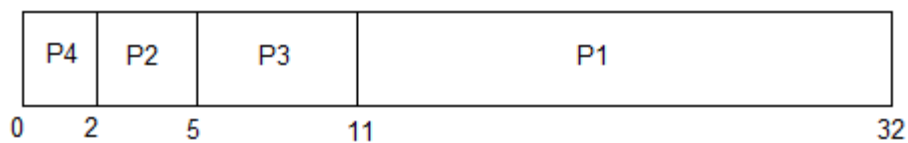
- Best approach to minimize waiting time.
- Actual time taken by the process is already known to processor.
- Impossible to implement.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2



In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following :

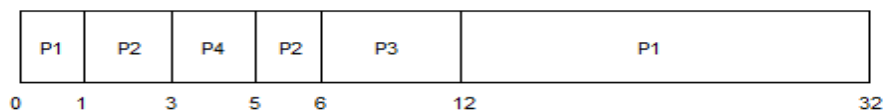


Now, the average waiting time will be = $(0 + 2 + 5 + 11)/4 = 4.5$ ms

In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with short burst time arrives, the existing process is preempted.

PROCESS	BURST TIME	ARRIVAL TIME
P1	21	0
P2	3	1
P3	6	2
P4	2	3

The GANTT chart for Preemptive Shortest Job First Scheduling will be,



The average waiting time will be, $((5-3) + (6-2) + (12-1))/4 = 4.25$ ms

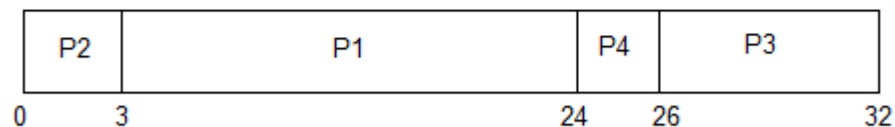
The average waiting time for preemptive shortest job first scheduling is less than both, non-preemptive SJF scheduling and FCFS scheduling.

Priority Scheduling

- Priority is assigned for each process.
- Process with highest priority is executed first and so on.
- Processes with same priority are executed in FCFS manner.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

PROCESS	BURST TIME	PRIORITY
P1	21	2
P2	3	1
P3	6	4
P4	2	3

The GANTT chart for following processes based on Priority scheduling will be,



The average waiting time will be, $(0 + 3 + 24 + 26) / 4 = \underline{13.25 \text{ ms}}$

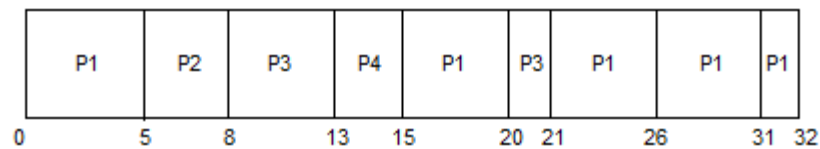
Round Robin(RR) Scheduling

- A fixed time is allotted to each process, called **quantum**, for execution.
- Once a process is executed for given time period that process is preempted and other process executes for given time period.
- Context switching is used to save states of preempted processes.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2



The GANTT chart for round robin scheduling will be,



The average waiting time will be, 11 ms.

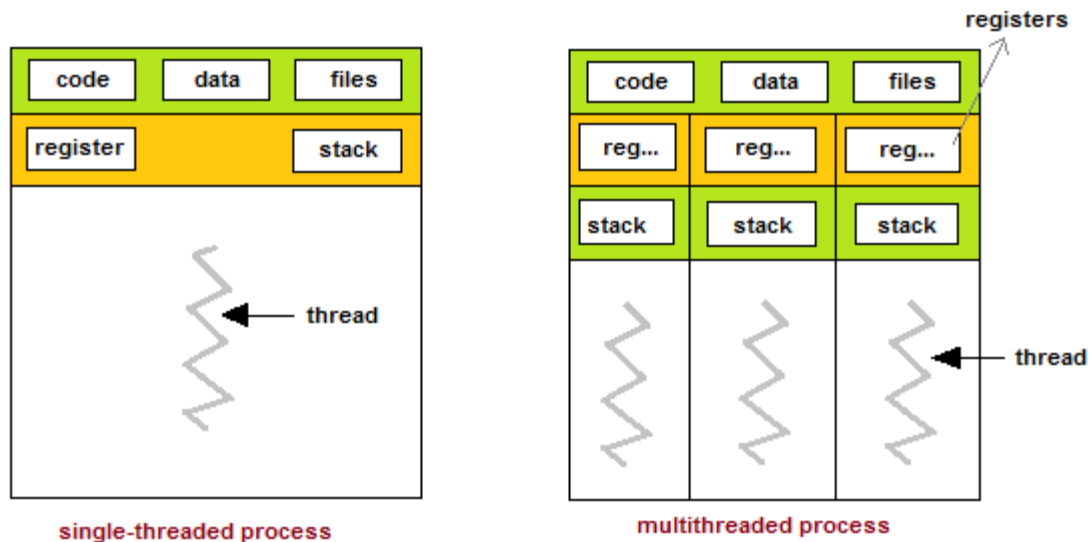
Multilevel Queue Scheduling

- Multiple queues are maintained for processes.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

Threads

Thread is an execution unit which consists of its own program counter, a stack, and a set of registers. Threads are also known as Lightweight processes. Threads are popular way to improve application through parallelism. The CPU switches rapidly back and forth among the threads giving illusion that the threads are running in parallel.

As each thread has its own independent resource for process execution, multiple processes can be executed parallelly by increasing number of threads.



Types of Thread

There are two types of threads :

- User Threads
- Kernel Threads

User threads, are above the kernel and without kernel support. These are the threads that application programmers use in their programs.

Kernel threads are supported within the kernel of the OS itself. All modern OSs support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.

Multithreading Models

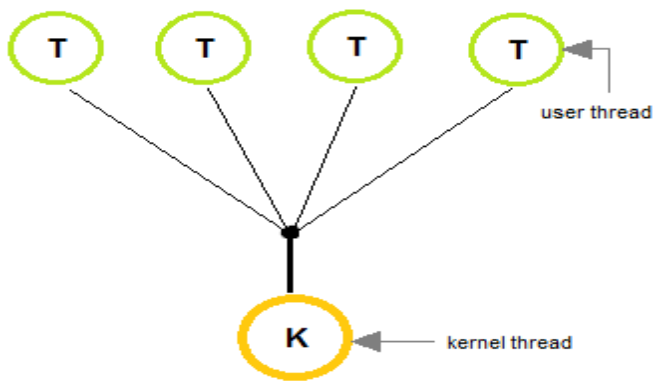
The user threads must be mapped to kernel threads, by one of the following strategies.

- Many-To-One Model
- One-To-One Model
- Many-To-Many Model

Many-To-One Model

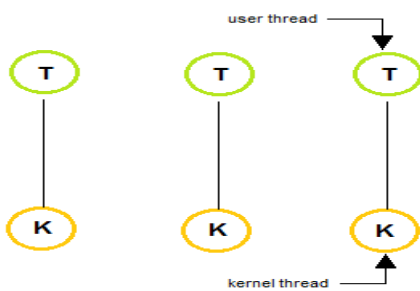
In the many-to-one model, many user-level threads are all mapped onto a single kernel thread.

- Thread management is handled by the thread library in user space, which is efficient in nature.



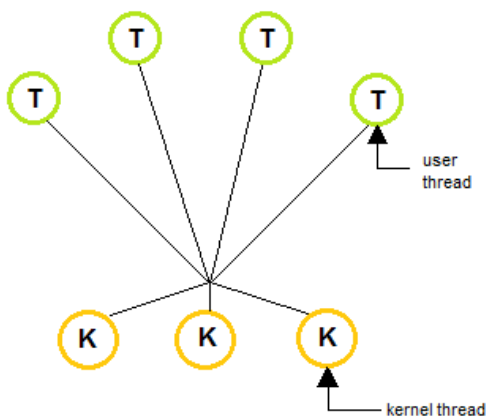
One-To-One Model

- The one-to-one model creates a separate kernel thread to handle each and every user thread.
- Most implementations of this model place a limit on how many threads can be created.
- Linux and Windows from 95 to XP implement the one-to-one model for threads.



Many-To-Many Model

- The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads, combining the best features of the one-to-one and many-to-one models.
- Users can create any number of the threads.
- Blocking the kernel system calls does not block the entire process.
- Processes can be split across multiple processors.



Thread Libraries

Thread libraries provide programmers with API for creating and managing of threads.

Thread libraries may be implemented either in user space or in kernel space. The user space involves API functions implemented solely within user space, with no kernel support. The kernel space involves system calls, and requires a kernel with thread library support.

Benefits of Multithreading

1. Responsiveness
2. Resource sharing, hence allowing better utilization of resources.
3. Economy. Creating and managing threads becomes easier.
4. Scalability. One thread runs on one CPU. In Multithreaded processes, threads can be distributed over a series of processors to scale.

5. Context Switching is smooth. Context switching refers to the procedure followed by CPU to change from one task to another.

Multithreading Issues

1. **Thread Cancellation.**

Thread cancellation means terminating a thread before it has finished working. There can be two approaches for this, one is **Asynchronous cancellation**, which terminates the target thread immediately. The other is **Deferred cancellation** allows the target thread to periodically check if it should be cancelled.

2. **Signal Handling.**

Signals are used in UNIX systems to notify a process that a particular event has occurred. Now in when a Multithreaded process receives a signal, to which thread it must be delivered? It can be delivered to all, or a single thread.

3. **fork() System Call.**

fork() is a system call executed in the kernel through which a process creates a copy of itself. Now the problem in Multithreaded process is, if one thread forks, will the entire process be copied or not?

4. **Security Issues** because of extensive sharing of resources between multiple threads.

There are many other issues that you might face in a multithreaded process, but there are appropriate solutions available for them. Pointing out some issues here was just to study both sides of the coin.

Process Synchronization

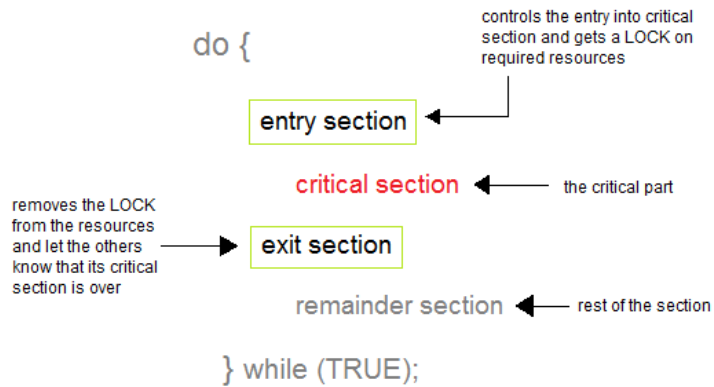
Process Synchronization means sharing system resources by processes in a such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.

Process Synchronization was introduced to handle problems that arose while multiple

process executions. Some of the problems are discussed below.

Critical Section Problem

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.



Solution to Critical Section Problem

A solution to the critical section problem must satisfy the following three conditions :

1. **Mutual Exclusion**

Out of a group of cooperating processes, only one process can be in its critical section at a given point of time.

2. **Progress**

If no process is in its critical section, and if one or more threads want to execute their critical section then any one of these threads must be allowed to get into its critical section.

3. **Bounded Waiting**

After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request is granted. So after the limit is reached, system must grant the process permission to get into its critical section.

Synchronization Hardware

Many systems provide hardware support for critical section code. The critical section problem could be solved easily in a single-processor environment if we could disallow interrupts to occur while a shared variable or resource is being modified.

In this manner, we could be sure that the current sequence of instructions would be allowed to execute in order without pre-emption. Unfortunately, this solution is not feasible in a multiprocessor environment.

Disabling interrupt on a multiprocessor environment can be time consuming as the message is passed to all the processors.

This message transmission lag, delays entry of threads into critical section and the system efficiency decreases.

Mutex Locks

As the synchronization hardware solution is not easy to implement for everyone, a strict software approach called Mutex Locks was introduced. In this approach, in the entry section of code, a

LOCK is acquired over the critical resources modified and used inside critical section, and in the exit section that LOCK is released.

As the resource is locked while a process executes its critical section hence no other process can access it.

Semaphores

In 1965, Dijkstra proposed a new and very significant technique for managing concurrent processes by using the value of a simple integer variable to synchronize the progress of interacting processes. This integer variable is called **semaphore**. So it is basically a synchronizing tool and is accessed only through two low standard atomic operations, wait and signal designated by P() and V() respectively.

The classical definition of wait and signal are :

- Wait : decrement the value of its argument S as soon as it would become non-negative.
- Signal : increment the value of its argument, S as an individual operation.

Properties of Semaphores

1. Simple
2. Works with many processes
3. Can have many different critical sections with different semaphores
4. Each critical section has unique access semaphores
5. Can permit multiple processes into the critical section at once, if desirable.

Types of Semaphores

Semaphores are mainly of two types:

1. Binary Semaphore

It is a special form of semaphore used for implementing mutual exclusion, hence it is often called *Mutex*. A binary semaphore is initialized to 1 and only takes the value 0 and 1 during execution of a program.

2. Counting Semaphores

These are used to implement bounded concurrency.

Limitations of Semaphores

1. Priority Inversion is a big limitation of semaphores.
2. Their use is not enforced, but is by convention only.
3. With improper use, a process may block indefinitely. Such a situation is called Deadlock. We will be studying deadlocks in details in coming lessons.

Classical Problem of Synchronization

Following are some of the classical problems faced while process synchronization in systems where cooperating processes are present.

Bounded Buffer Problem

- This problem is generalised in terms of the Producer-Consumer problem.
- Solution to this problem is, creating two counting semaphores "full" and "empty" to keep track of the current number of full and empty buffers respectively.

The Readers Writers Problem

- In this problem there are some processes (called readers) that only read the shared data, and never change it, and there are other processes (called writers) who may change the data in addition to reading or instead of reading it.
- There are various types of the readers-writers problem, most centred on relative priorities of readers and writers

Dining Philosophers Problem

- The dining philosopher's problem involves the allocation of limited resources from a group of processes in a deadlock-free and starvation-free manner.
- There are five philosophers sitting around a table, in which there are five chopsticks kept beside them and a bowl of rice in the centre. When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place.

Multiple Processors Scheduling in Operating System

Multiple processor scheduling or multiprocessor scheduling focuses on designing the system's scheduling function, which consists of more than one processor. Multiple CPUs share the load (load sharing) in multiprocessor scheduling so that various processes run simultaneously. In general, multiprocessor scheduling is complex as compared to single processor scheduling. In the multiprocessor scheduling, there are many processors, and they are identical, and we can run any process at any time.

The multiple CPUs in the system are in close communication, which shares a common bus, memory, and other peripheral devices. So we can say that the system is tightly coupled. These systems are used when we want to process a bulk amount of data, and these systems are mainly used in satellite, weather forecasting, etc.

There are cases when the processors are identical, i.e., homogenous, in terms of their functionality in multiple-processor scheduling. We can use any processor available to run any process in the queue.

Multiprocessor systems may be **heterogeneous** (different kinds of CPUs) or **homogenous** (the same CPU). There may be special scheduling constraints, such as devices connected via a private bus to only one CPU.

There is no policy or rule which can be declared as the best scheduling solution to a system with a single processor. Similarly, there is no best scheduling solution for a system with multiple processors as well.

Approaches to Multiple Processor Scheduling

There are two approaches to multiple processor scheduling in the operating system: Symmetric Multiprocessing and Asymmetric Multiprocessing.



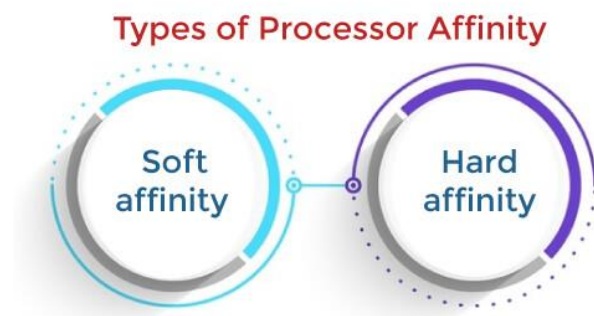
1. **Symmetric Multiprocessing:** It is used where each processor is **self-scheduling**. All processes may be in a common ready queue, or each processor may have its private queue for ready processes. The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute.

2. **Asymmetric Multiprocessing:** It is used when all the scheduling decisions and I/O processing are handled by a single processor called the **Master Server**. The other processors execute only the **user code**. This is simple and reduces the need for data sharing, and this entire scenario is called Asymmetric Multiprocessing.

Processor Affinity

Processor Affinity means a process has an **affinity** for the processor on which it is currently running. When a process runs on a specific processor, there are certain effects on the cache memory. The data most recently accessed by the process populate the cache for the processor. As a result, successive memory access by the process is often satisfied in the cache memory.

Now, suppose the process migrates to another processor. In that case, the contents of the cache memory must be invalidated for the first processor, and the cache for the second processor must be repopulated. Because of the high cost of invalidating and repopulating caches, most SMP (symmetric multiprocessing) systems try to avoid migrating processes from one processor to another and keep a process running on the same processor. This is known as processor affinity. There are two types of processor affinity, such as:



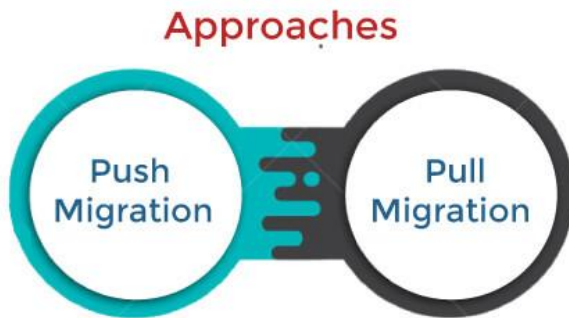
1. **Soft Affinity:** When an operating system has a policy of keeping a process running on the same processor but not guaranteeing it will do so, this situation is called soft affinity.
2. **Hard Affinity:** Hard Affinity allows a process to specify a subset of processors on which it may run. Some Linux systems implement soft affinity and provide system calls like ***sched_setaffinity()*** that also support hard affinity.

Load Balancing

Load Balancing is the phenomenon that keeps the workload evenly distributed across all processors in an SMP system. Load balancing is necessary only on systems where each processor has its own private queue of a process that is eligible to execute.

Load balancing is unnecessary because it immediately extracts a runnable process from the common run queue once a processor becomes idle. On SMP (symmetric

multiprocessing), it is important to keep the workload balanced among all processors to utilize the benefits of having more than one processor fully. One or more processors will sit idle while other processors have high workloads along with lists of processors awaiting the CPU. There are two general approaches to load balancing:



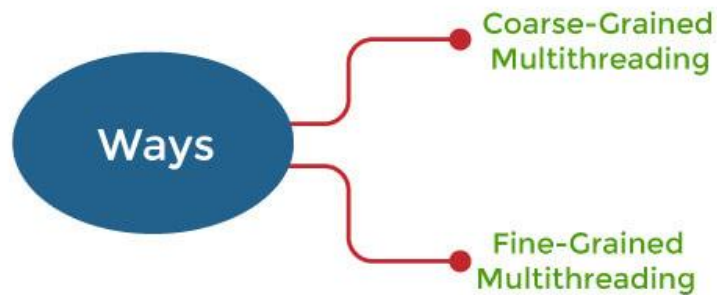
1. **Push Migration:** In push migration, a task routinely checks the load on each processor. If it finds an imbalance, it evenly distributes the load on each processor by moving the processes from overloaded to idle or less busy processors.
2. **Pull Migration:** Pull Migration occurs when an idle processor pulls a waiting task from a busy processor for its execution.

Multi-core Processors

In multi-core processors, multiple processor cores are placed on the same physical chip. Each core has a register set to maintain its architectural state and thus appears to the operating system as a separate physical processor. **SMP systems** that use multi-core processors are faster and consume less power than systems in which each processor has its own physical chip.

However, multi-core processors may complicate the scheduling problems. When the processor accesses memory, it spends a significant amount of time waiting for the data to become available. This situation is called a **Memory stall**. It occurs for various reasons, such as cache miss, which is accessing the data that is not in the cache memory.

In such cases, the processor can spend upto 50% of its time waiting for data to become available from memory. To solve this problem, recent hardware designs have implemented multithreaded processor cores in which two or more hardware threads are assigned to each core. Therefore if one thread stalls while waiting for the memory, the core can switch to another thread. There are two ways to multithread a processor:



1. **Coarse-Grained Multithreading:** A thread executes on a processor until a long latency event such as a memory stall occurs in coarse-grained multithreading. Because of the delay caused by the long latency event, the processor must switch to another thread to begin execution. The cost of switching between threads is high as the instruction pipeline must be terminated before the other thread can begin execution on the processor core. Once this new thread begins execution, it begins filling the pipeline with its instructions.
2. **Fine-Grained Multithreading:** This multithreading switches between threads at a much finer level, mainly at the boundary of an instruction cycle. The architectural design of fine-grained systems includes logic for thread switching, and as a result, the cost of switching between threads is small.

Symmetric Multiprocessor

Symmetric Multiprocessors (SMP) is the third model. There is one copy of the OS in memory in this model, but any central processing unit can run it. Now, when a system call is made, the central processing unit on which the system call was made traps the kernel and processed that system call. This model balances processes and memory dynamically. This approach uses Symmetric Multiprocessing, where each processor is self-scheduling.

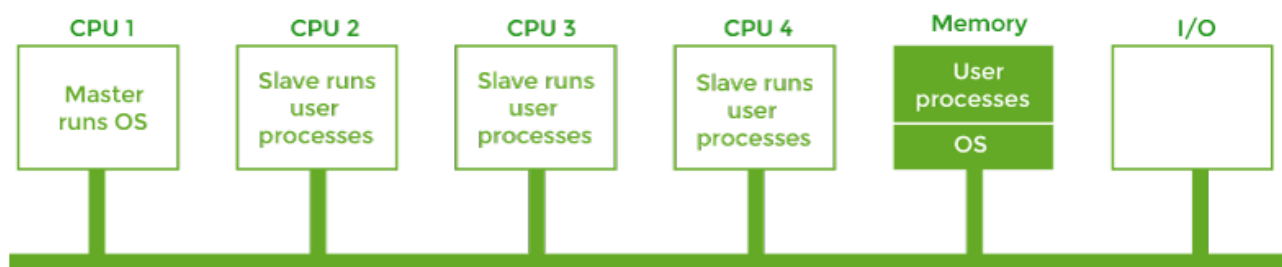
The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute. In this system, this is possible that all the process may be in a common ready queue or each processor may have its private queue for the ready process. There are mainly three sources of contention that can be found in a multiprocessor operating system.

- **Locking system:** As we know that the resources are shared in the multiprocessor system, there is a need to protect these resources for safe access among the multiple processors. The main purpose of the locking scheme is to serialize access of the resources by the multiple processors.
- **Shared data:** When the multiple processors access the same data at the same time, then there may be a chance of inconsistency of data, so to protect this, we have to use some protocols or locking schemes.

- **Cache coherence:** It is the shared resource data that is stored in multiple local caches. Suppose two clients have a cached copy of memory and one client change the memory block. The other client could be left with an invalid cache without notification of the change, so this conflict can be resolved by maintaining a coherent view of the data.

Master-Slave Multiprocessor

In this multiprocessor model, there is a single data structure that keeps track of the ready processes. In this model, one central processing unit works as a master and another as a slave. All the processors are handled by a single processor, which is called the master server.



The master server runs the operating system process, and the slave server runs the user processes. The memory and input-output devices are shared among all the processors, and all the processors are connected to a common bus. This system is simple and reduces data sharing, so this system is called **Asymmetric multiprocessing**.

Virtualization and Threading

In this type of **multiple processor** scheduling, even a single CPU system acts as a multiple processor system. In a system with virtualization, the virtualization presents one or more virtual CPUs to each of the virtual machines running on the system. It then schedules the use of physical CPUs among the virtual machines.

- Most virtualized environments have one host operating system and many guest operating systems, and the host operating system creates and manages the virtual machines.
- Each virtual machine has a guest operating system installed, and applications run within that guest.
- Each guest operating system may be assigned for specific use cases, applications, or users, including time-sharing or real-time operation.
- Any guest operating-system scheduling algorithm that assumes a certain amount of progress in a given amount of time will be negatively impacted by the virtualization.

- A time-sharing operating system tries to allot 100 milliseconds to each time slice to give users a reasonable response time. A given 100 millisecond time slice may take much more than 100 milliseconds of virtual CPU time. Depending on how busy the system is, the time slice may take a second or more, which results in a very poor response time for users logged into that virtual machine.
- The net effect of such scheduling layering is that individual virtualized operating systems receive only a portion of the available CPU cycles, even though they believe they are receiving all cycles and scheduling all of those cycles. The time-of-day clocks in virtual machines are often incorrect because timers take no longer to trigger than they would on dedicated CPUs.
- Virtualizations can thus undo the good scheduling algorithm efforts of the operating systems within virtual machines.

Introduction to Deadlock

Every process needs some resources to complete its execution. However, the resource is granted in a sequential order.

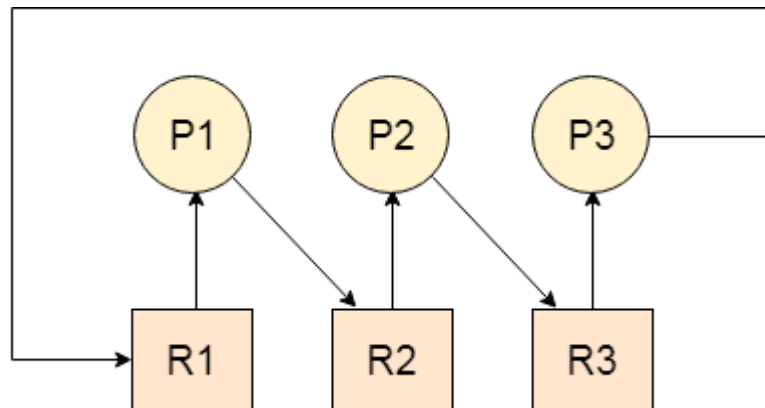
1. The process requests for some resource.
2. OS grant the resource if it is available otherwise let the process waits.
3. The process uses it and release on the completion.

A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.

Let us assume that there are three processes P1, P2 and P3. There are three different resources R1, R2 and R3. R1 is assigned to P1, R2 is assigned to P2 and R3 is assigned to P3.

After some time, P1 demands for R2 which is being used by P2. P1 halts its execution since it can't complete without R2. P2 also demands for R3 which is being used by P3. P2 also stops its execution because it can't continue without R3. P3 also demands for R1 which is being used by P1 therefore P3 also stops its execution.

In this scenario, a cycle is being formed among the three processes. None of the process is progressing and they are all waiting. The computer becomes unresponsive since all the processes got blocked.



Difference between Starvation and Deadlock

Sr.	Deadlock	Starvation
1	Deadlock is a situation where no process got blocked and no process proceeds	Starvation is a situation where the low priority process got blocked and the high priority processes proceed.
2	Deadlock is an infinite waiting.	Starvation is a long waiting but not infinite.
3	Every Deadlock is always a starvation.	Every starvation need not be deadlock.
4	The requested resource is blocked by the other process.	The requested resource is continuously be used by the higher priority processes.
5	Deadlock happens when Mutual exclusion, hold and wait, No preemption and circular wait occurs simultaneously.	It occurs due to the uncontrolled priority and resource management.

Necessary conditions for Deadlocks

1. **Mutual Exclusion**

A resource can only be shared in mutually exclusive manner. It implies, if two process cannot use the same resource at the same time.

2. **Hold and Wait**

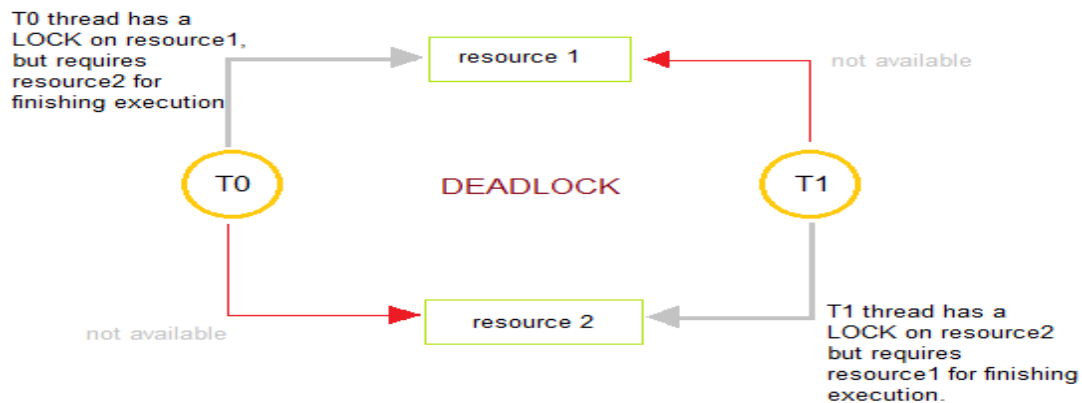
A process waits for some resources while holding another resource at the same time.

3. **No preemption**

The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.

4. **Circular Wait**

All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.



How to avoid Deadlocks

Deadlocks can be avoided by avoiding at least one of the four conditions, because all this four conditions are required simultaneously to cause deadlock.

1. **Mutual Exclusion**

Resources shared such as read-only files do not lead to deadlocks but resources, such as printers and tape drives, requires exclusive access by a single process.

2. **Hold and Wait**

In this condition processes must be prevented from holding one or more resources while simultaneously waiting for one or more others.

3. **No Preemption**

Preemption of process resource allocations can avoid the condition of deadlocks, where ever possible.

4. **Circular Wait**

Circular wait can be avoided if we number all resources, and require that processes request resources only in strictly increasing(or decreasing) order.

Handling Deadlock

The above points focus on preventing deadlocks. But what to do once a deadlock has occurred.

Following three strategies can be used to remove deadlock after its occurrence.

1. **Preemption**

We can take a resource from one process and give it to other. This will resolve the deadlock situation, but sometimes it does causes problems.

2. **Rollback**

In situations where deadlock is a real possibility, the system can periodically make a record of the state of each process and when deadlock occurs, roll everything back to the last checkpoint, and restart, but allocating resources differently so that deadlock does not occur.

3. **Kill one or more processes**

This is the simplest way, but it works.

What is a Livelock?

There is a variant of deadlock called livelock. This is a situation in which two or more processes continuously change their state in response to changes in the other process(es) without doing any useful work. This is similar to deadlock in that no progress is made but differs in that neither process is blocked or waiting for anything.

A human example of livelock would be two people who meet face-to-face in a corridor and each moves aside to let the other pass, but they end up swaying from side to side without making any progress because they always move the same way at the same time.

Memory Management

Main Memory refers to a physical memory that is the internal memory to the computer. The word main is used to distinguish it from external mass storage devices such as disk drives. Main memory is also known as RAM. The computer is able to change only data that is in main memory. Therefore, every program we execute and every file we access must be copied from a storage device into main memory.

All the programs are loaded in the main memory for execution. Sometimes complete program is loaded into the memory, but some times a certain part or routine of the program is loaded into the main memory only when it is called by the program, this mechanism is called **Dynamic Loading**, this enhance the performance.

Also, at times one program is dependent on some other program. In such a case, rather than loading all the dependent programs, CPU links the dependent programs to the main executing program when its required. This mechanism is known as **Dynamic Linking**.

Swapping

A process needs to be in memory for execution. But sometimes there is not enough main memory to hold all the currently active processes in a timesharing system. So, excess process are kept on disk and brought in to run dynamically. Swapping is the process of bringing in each process in main memory, running it for a while and then putting it back to the disk.

Contiguous Memory Allocation

In contiguous memory allocation each process is contained in a single contiguous block of memory. Memory is divided into several fixed size partitions. Each partition contains exactly one process. When a partition is free, a process is selected from the input queue and loaded into it. The free blocks of memory are known as *holes*. The set of holes is searched to determine which hole is best to allocate.

Memory Protection

Memory protection is a phenomenon by which we control memory access rights on a computer. The main aim of it is to prevent a process from accessing memory that has not been allocated to

it. Hence prevents a bug within a process from affecting other processes, or the operating system itself, and instead results in a segmentation fault or storage violation exception being sent to the disturbing process, generally killing of process.

Memory Allocation

Memory allocation is a process by which computer programs are assigned memory or space. It is of three types :

1. **First Fit**
The first hole that is big enough is allocated to program.
2. **Best Fit**
The smallest hole that is big enough is allocated to program.
3. **Worst Fit**
The largest hole that is big enough is allocated to program.

Fragmentation

Fragmentation occurs in a dynamic memory allocation system when most of the free blocks are too small to satisfy any request. It is generally termed as inability to use the available memory.

In such situation processes are loaded and removed from the memory. As a result of this, free holes exists to satisfy a request but is non contiguous i.e. the memory is fragmented into large no. Of small holes. This phenomenon is known as **External Fragmentation**.

Also, at times the physical memory is broken into fixed size blocks and memory is allocated in unit of block sizes. The memory allocated to a space may be slightly larger than the requested memory. The difference between allocated and required memory is known as **Internal fragmentation** i.e. the memory that is internal to a partition but is of no use.

Paging

A solution to fragmentation problem is Paging. Paging is a memory management mechanism that allows the physical address space of a process to be non-contiguous. Here physical memory is divided into blocks of equal size called **Pages**. The pages belonging to a certain process are loaded into available memory frames.

Page Table

A Page Table is the data structure used by a virtual memory system in a computer operating system to store the mapping between *virtual address* and *physical addresses*.

Virtual address is also known as Logical address and is generated by the CPU. While Physical address is the address that actually exists on memory.

Segmentation

Segmentation is another memory management scheme that supports the user-view of memory. Segmentation allows breaking of the virtual address space of a single process into segments that may be placed in non-contiguous areas of physical memory.

Segmentation with Paging

Both paging and segmentation have their advantages and disadvantages, it is better to combine these two schemes to improve on each. The combined scheme is known as 'Page the Elements'. Each segment in this scheme is divided into pages and each segment is maintained in a page table. So the logical address is divided into following 3 parts :

- Segment numbers(S)
- Page number (P)
- The displacement or offset number (D)

Virtual Memory

Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there.

In real scenarios, most processes never need all their pages at once, for following reasons :

- Error handling code is not needed unless that specific error occurs, some of which are quite rare.
- Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
- Certain features of certain programs are rarely used.

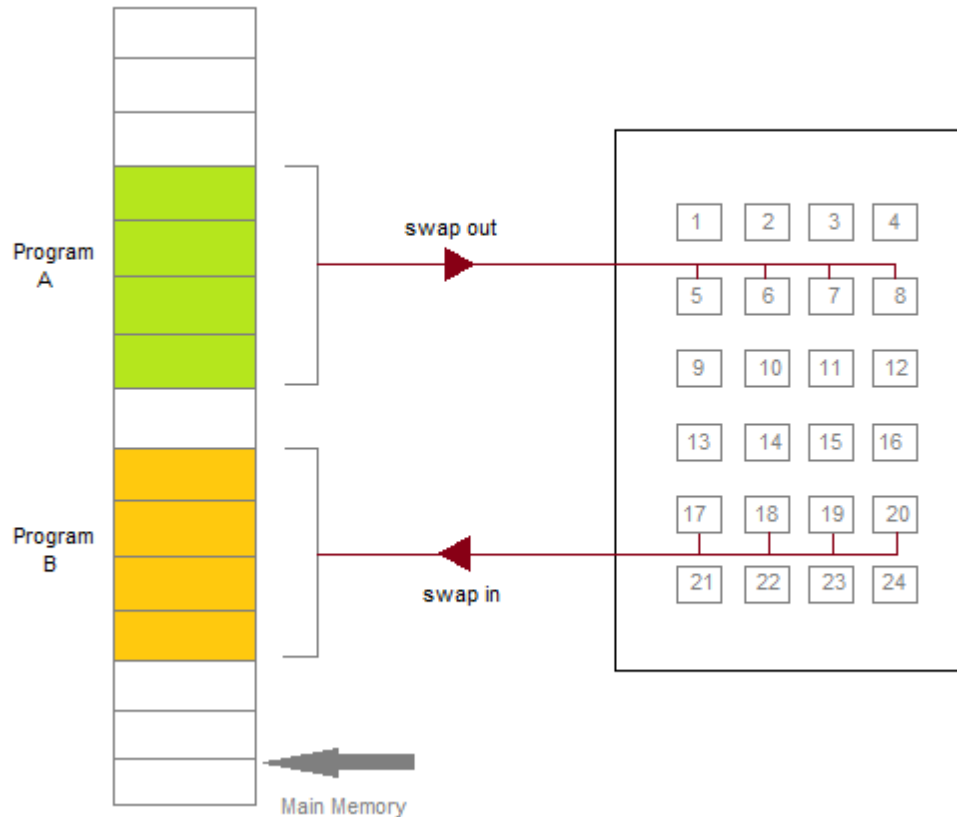
Benefits of having Virtual Memory :

1. Large programs can be written, as virtual space available is huge compared to physical memory.
2. Less I/O required, leads to faster and easy swapping of processes.
3. More physical memory available, as programs are stored on virtual memory, so they

occupy very less space on actual physical memory.

Demand Paging

The basic idea behind demand paging is that when a process is swapped in, its pages are not swapped in all at once. Rather they are swapped in only when the process needs them(On demand). This is termed as lazy swapper, although a pager is a more accurate term.



Initially only those pages are loaded which will be required the process immediately.

The pages that are not moved into the memory, are marked as invalid in the page table. For an invalid entry the rest of the table is empty. In case of pages that are loaded in the memory, they are marked as valid along with the information about where to find the swapped out page.

When the process requires any of the page that is not loaded into the memory, a page fault trap is triggered and following steps are followed,

1. The memory address which is requested by the process is first checked, to verify the request made by the process.
2. If its found to be invalid, the process is terminated.
3. In case the request by the process is valid, a free frame is located, possibly from a free-frame list, where the required page will be moved.
4. A new operation is scheduled to move the necessary page from disk to the specified memory location. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)
5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to valid.
6. The instruction that caused the page fault must now be restarted from the beginning.

There are cases when no pages are loaded into the memory initially, pages are only loaded when demanded by the process by generating page faults. This is called **Pure Demand Paging**.

The only major issue with Demand Paging is, after a new page is loaded, the process starts execution from the beginning. Its is not a big issue for small programs, but for larger programs it affects performance drastically.

Page Replacement

As studied in Demand Paging, only certain pages of a process are loaded initially into the memory. This allows us to get more number of processes into the memory at the same time. but what happens when a process requests for more pages and no free memory is available to bring them in. Following steps can be taken to deal with this problem :

1. Put the process in the wait queue, until any other process finishes its execution thereby freeing frames.
2. Or, remove some other process completely from the memory to free frames.
3. Or, find some pages that are not being used right now, move them to the disk to get free frames. This technique is called **Page replacement** and is most commonly used. We have some great algorithms to carry on page replacement efficiently.

Basic Page Replacement

- Find the location of the page requested by ongoing process on the disk.
- Find a free frame. If there is a free frame, use it. If there is no free frame, use a page-replacement algorithm to select any existing frame to be replaced, such frame is known as **victim frame**.
- Write the victim frame to disk. Change all related page tables to indicate that this page is no longer in memory.
- Move the required page and store it in the frame. Adjust all related page and frame tables to indicate the change.
- Restart the process that was waiting for this page.

FIFO Page Replacement

- A very simple way of Page replacement is FIFO (First in First Out)
- As new pages are requested and are swapped in, they are added to tail of a queue and the page which is at the head becomes the victim.
- Its not an effective way of page replacement but can be used for small systems.

LRU Page Replacement

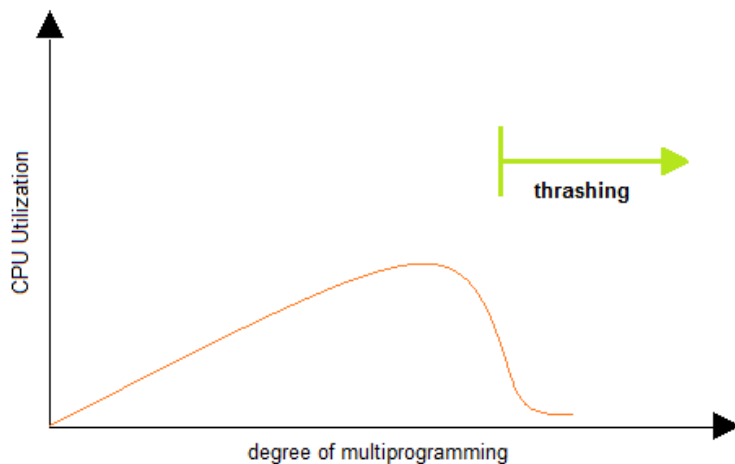
Below is a video, which will explain LRU Page replacement algorithm in details with an example.

Thrashing

A process that is spending more time paging than executing is said to be thrashing. In other words it means, that the process doesn't have enough frames to hold all the pages for its

execution, so it is swapping pages in and out very frequently to keep executing. Sometimes, the pages which will be required in the near future have to be swapped out.

Initially when the CPU utilization is low, the process scheduling mechanism, to increase the level of multiprogramming loads multiple processes into the memory at the same time, allocating a limited amount of frames to each process. As the memory fills up, process starts to spend a lot of time for the required pages to be swapped in, again leading to low CPU utilization because most of the processes are waiting for pages. Hence the scheduler loads more processes to increase CPU utilization, as this continues at a point of time the complete system comes to a stop.



To prevent thrashing we must provide processes with as many frames as they really need "right now".

File System

A file can be "free formed", indexed or structured collection of related bytes having meaning only to the one who created it. Or in other words an entry in a directory is the file. The file may have attributes like name, creator, date, type, permissions etc.

File Structure

A file has various kinds of structure. Some of them can be :

- **Simple Record Structure** with lines of fixed or variable lengths.
- **Complex Structures** like formatted document or reloadable load files.
- **No Definite Structure** like sequence of words and bytes etc.

Attributes of a File

Following are some of the attributes of a file :

- **Name** . It is the only information which is in human-readable form.
- **Identifier**. The file is identified by a unique tag(number) within file system.
- **Type**. It is needed for systems that support different types of files.
- **Location**. Pointer to file location on device.
- **Size**. The current size of the file.
- **Protection**. This controls and assigns the power of reading, writing, executing.
- **Time, date, and user identification**. This is the data for protection, security, and usage monitoring.

File Access Methods

The way that files are accessed and read into memory is determined by Access methods. Usually a single access method is supported by systems while there are OS's that support multiple access methods.

Sequential Access

- Data is accessed one record right after another in an order.
- Read command cause a pointer to be moved ahead by one.
- Write command allocate space for the record and move the pointer to the new End Of File.
- Such a method is reasonable for tape.

Direct Access

- This method is useful for disks.
- The file is viewed as a numbered sequence of blocks or records.
- There are no restrictions on which blocks are read/written, it can be done in any order.
- User now says "read n" rather than "read next".
- "n" is a number relative to the beginning of file, not relative to an absolute physical disk location.

Indexed Sequential Access

- It is built on top of Sequential access.
- It uses an Index to control the pointer while accessing files.

What is a Directory?

Information about files is maintained by Directories. A directory can contain multiple files. It can even have directories inside of them. In Windows we also call these directories as folders.

Following is the information maintained in a directory :

- **Name** : The name visible to user.
- **Type** : Type of the directory.
- **Location** : Device and location on the device where the file header is located.
- **Size** : Number of bytes/words/blocks in the file.
- **Position** : Current next-read/next-write pointers.
- **Protection** : Access control on read/write/execute/delete.
- **Usage** : Time of creation, access, modification etc.
- **Mounting** : When the root of one file system is "grafted" into the existing tree of another file system its called Mounting.