Unit-1

Definition of Operating Systems

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.

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

Types of Operating System

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.
-

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.

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.

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.

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.

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.

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.

Operating System - Services

Following are a few common services provided by an operating system −

- Program execution
- I/O operations
- File System manipulation
- Communication
- Error Detection

- Resource Allocation
- Protection

Program execution

Operating systems handle many kinds of activities from user programs to system programs like printer spooler, name servers, file server, etc. Each of these activities is encapsulated as a process.

A process includes the complete execution context (code to execute, data to manipulate, registers, OS resources in use). Following are the major activities of an operating system with respect to program management −

- Loads a program into memory.
- Executes the program.
- Handles program's execution.
- Provides a mechanism for process synchronization.
- Provides a mechanism for process communication.
- Provides a mechanism for deadlock handling.

I/O Operation

An I/O subsystem comprises of I/O devices and their corresponding driver software. Drivers hide the peculiarities of specific hardware devices from the users.

An Operating System manages the communication between user and device drivers.

- I/O operation means read or write operation with any file or any specific I/O device.
- Operating system provides the access to the required I/O device when required.

File system manipulation

A file represents a collection of related information. Computers can store files on the disk (secondary storage), for long-term storage purpose. Examples of storage media include magnetic tape, magnetic disk and optical disk drives like CD, DVD. Each of these media has its own properties like speed, capacity, data transfer rate and data access methods.

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions. Following are the major activities of an operating system with respect to file management −

- Program needs to read a file or write a file.
- The operating system gives the permission to the program for operation on file.
- Permission varies from read-only, read-write, denied and so on.
- Operating System provides an interface to the user to create/delete files.
- Operating System provides an interface to the user to create/delete directories.
- Operating System provides an interface to create the backup of file system.

Communication

In case of distributed systems which are a collection of processors that do not share memory, peripheral devices, or a clock, the operating system manages communications between all the processes. Multiple processes communicate with one another through communication lines in the network.

The OS handles routing and connection strategies, and the problems of contention and security. Following are the major activities of an operating system with respect to communication −

- Two processes often require data to be transferred between them
- Both the processes can be on one computer or on different computers, but are connected through a computer network.
- Communication may be implemented by two methods, either by Shared Memory or by Message Passing.

Error handling

Errors can occur anytime and anywhere. An error may occur in CPU, in I/O devices or in the memory hardware. Following are the major activities of an operating system with respect to error handling −

- The OS constantly checks for possible errors.
- The OS takes an appropriate action to ensure correct and consistent computing.

Resource Management

In case of multi-user or multi-tasking environment, resources such as main memory, CPU cycles and files storage are to be allocated to each user or job. Following are the major activities of an operating system with respect to resource management −

- The OS manages all kinds of resources using schedulers.
- CPU scheduling algorithms are used for better utilization of CPU.

Protection

Considering a computer system having multiple users and concurrent execution of multiple processes, the various processes must be protected from each other's activities.

Protection refers to a mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system. Following are the major activities of an operating system with respect to protection −

- The OS ensures that all access to system resources is controlled.
- The OS ensures that external I/O devices are protected from invalid access attempts.
- The OS provides authentication features for each user by means of passwords.

User operating system interface:-

GUI

A **GUI** (graphical user interface) is a system of interactive visual components for computer software. A GUI displays objects that convey information, and represent actions that can be taken by the user. The objects change color, size, or visibility when the user interacts with them.GUI objects include icons, cursors, and buttons. These graphical elements are sometimes enhanced with sounds, or visual effects like transparency and drop shadows.A GUI is considered to be more user-friendly than a text-based command-line interface, such as MS-DOS, or the shell of Unix-like operating systems.The GUI was first developed at Xerox PARC by Alan Kay, Douglas Engelbart, and a group of other researchers in 1981. Later, Apple introduced the Lisa computer with a GUI on January 19, 1983.

GUI overview

Below is a picture of the Windows 7 desktop and an example of a GUI.

CUI

Short for **character user interface** or **command-line user interface**, **CUI** is a way for users to interact with computer programs. It works by allowing the user (client) to issue commands as one or more lines of text (referred to as command lines) to a program. Good examples CUIs are MS-DOS and the Windows Command Prompt. One of the CUI's uses is that it provides an easy way to implement programming scripts.

Other examples of command lines

There are other command lines in addition to the ones mentioned above, namely, Terminal, and the Linux command line.

History

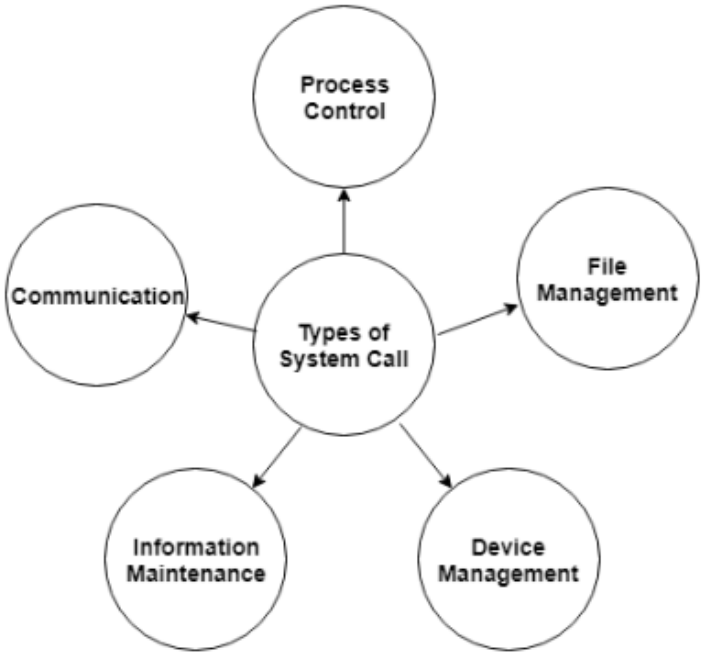
The command-line user interface was the primary method of communicating with a computer from the first machines and through the 1980s. Although it may still be accessed in today's operating systems, it is utilized far less due to the ease of use and familiarity of the GUI (graphical user interface). The CUI, however, is still preferred by many advanced end users as its features provide them with more comprehensive control over an operating system's functions.

The interface between a process and an operating system is provided by system calls. In general, system calls are available as assembly language instructions. They are also included in the manuals used by the assembly level programmers.

System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.

Types of System Calls

There are mainly five types of system calls. These are explained in detail as follows −



Here are the types of system calls −

Process Control

These system calls deal with processes such as process creation, process termination etc.

File Management

These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.

Device Management

These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.

Information Maintenance

These system calls handle information and its transfer between the operating system and the user program.

Communication

These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection.

Some of the examples of all the above types of system calls in Windows and Unix are given as follows −

Types of System Calls	Windows	Linux
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Management	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()

There are many different system calls as shown above. Details of some of those system calls are as follows −

wait()

In some systems, a process may wait for another process to complete its execution. This happens when a parent process creates a child process and the execution of the parent process is suspended until the child process executes. The suspending of the parent process occurs with a wait() system call. When the child process completes execution, the control is returned back to the parent process.

exec()

This system call runs an executable file in the context of an already running process. It replaces the previous executable file. This is known as an overlay. The original process identifier remains since a new process is not created but data, heap, stack etc. of the process are replaced by the new process.

fork()

Processes use the fork() system call to create processes that are a copy of themselves. This is one of the major methods of process creation in operating systems. When a parent process creates a child process and the execution of the parent process is suspended until the child process executes. When the child process completes execution, the control is returned back to the parent process.

exit()

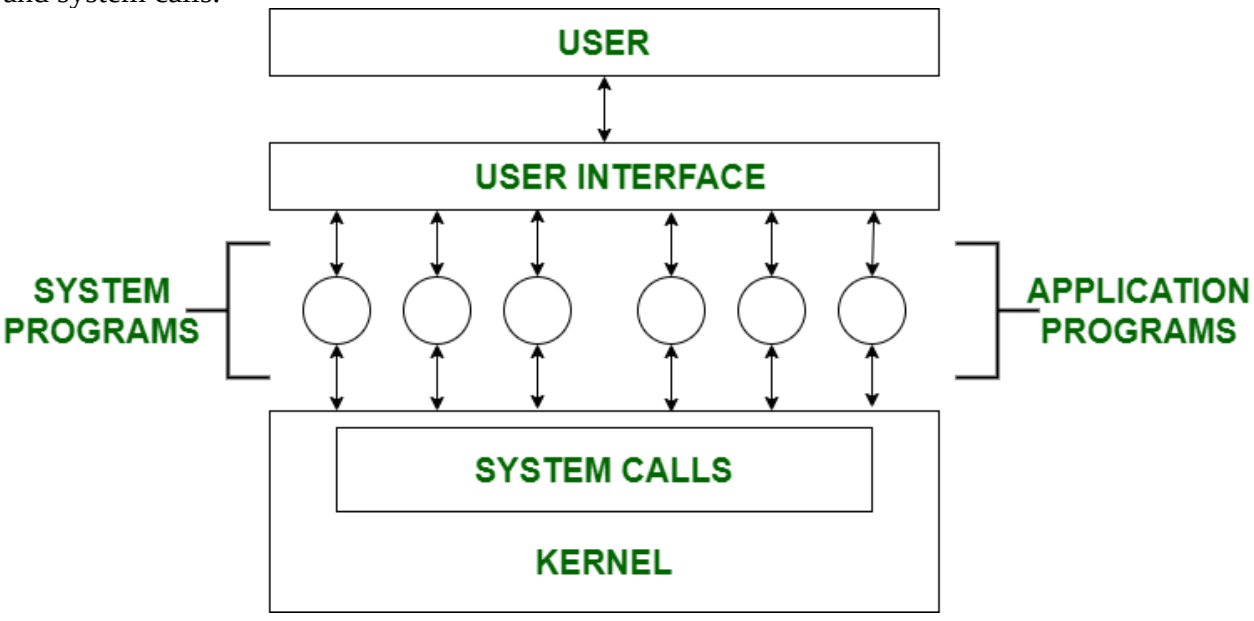
The exit() system call is used by a program to terminate its execution. In a multithreaded environment, this means that the thread execution is complete. The operating system reclaims resources that were used by the process after the exit() system call.

kill()

The kill() system call is used by the operating system to send a termination signal to a process that urges the process to exit.However, kill system call does not necessary mean killing the process and can have various meanings.

System Programs in Operating System

System Programming can be defined as act of building Systems Software using System Programming Languages. According to Computer Hierarchy, one which comes at last is Hardware. Then it is Operating System, System Programs, and finally Application Programs. Program Development and Execution can be done conveniently in System Programs. Some of System Programs are simply user interfaces, others are complex. It traditionally lies between user interface and system calls.



So here, user can only view up-to-the System Programs he can't see System Calls.

System Programs can be divided into these categories :

1. **File Management** – A file is a collection of specific information stored in memory of computer system. File

management is defined as process of manipulating files in computer system, it management includes process of creating, modifying and deleting files.

- It helps to create new files in computer system and placing them at specific locations.
- It helps in easily and quickly locating these files in computer system.
- It makes process of sharing of files among different users very easy and user friendly.
- It helps to stores files in separate folders known as directories.
- These directories help users to search file quickly or to manage files according to their types or uses.
- It helps user to modify data of files or to modify he name of file in directories.

2. **Status Information –**
   Information like date, time amount of available memory, or disk space is asked by some of users. Others providing detailed performance, logging and debugging information which is more complex. All this information is formatted and displayed on output devices or printed. Terminal or other output devices or files or a window of GUI is used for showing output of programs.

3. **File Modification –**
   For modifying contents of files we use this. For Files stored on disks or other storage devices we used different types of editors. For searching contents of files or perform transformations of files we use special commands.

4. **Programming-Language support –**
   For common programming languages we use Compilers, Assemblers, Debuggers and interpreters which are already provided to user. It provides all support to users. We can run any programming languages. All languages of importance are already provided.

5. **Program Loading and Execution –**
   When program is ready after Assembling and compilation, it must be loaded into memory for execution. A loader is part of an operating system that is responsible for loading programs and libraries. It is one of essential stages for starting a program. Loaders, relocatable loaders, linkage editors and Overlay loaders are provided by system.

6. **Communications –**
   Virtual connections among processes, users and computer systems are provided by programs. User can send messages to other user on their screen, User can send e-mail, browsing on web pages, remote login, transformation of files from one user to another.

Some examples of system programs in O.S. are –

- Windows 10
- Mac OS X
- Ubuntu

- Linux
- Unix
- Android
- Anti-virus
- Disk formatting
- Computer language translators

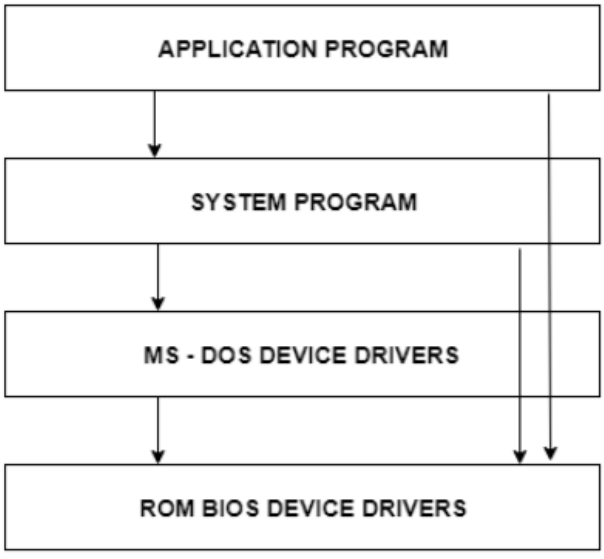
Operating System Structure

An operating system is a construct that allows the user application programs to interact with the system hardware. Since the operating system is such a complex structure, it should be created with utmost care so it can be used and modified easily. An easy way to do this is to create the operating system in parts. Each of these parts should be well defined with clear inputs, outputs and functions.

Simple Structure

There are many operating systems that have a rather simple structure. These started as small systems and rapidly expanded much further than their scope. A common example of this is MS-DOS. It was designed simply for a niche amount for people. There was no indication that it would become so popular.

An image to illustrate the structure of MS-DOS is as follows −



MS-DOS STRUCTURE

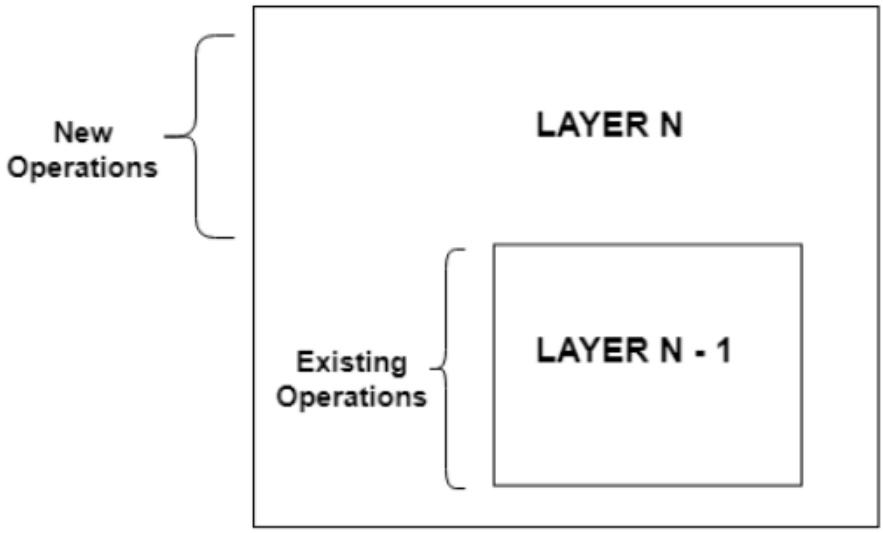
It is better that operating systems have a modular structure, unlike MS-DOS. That would lead to greater control over the computer system and its various applications. The modular structure would

also allow the programmers to hide information as required and implement internal routines as they see fit without changing the outer specifications.

Layered Structure

One way to achieve modularity in the operating system is the layered approach. In this, the bottom layer is the hardware and the topmost layer is the user interface.

An image demonstrating the layered approach is as follows −



Layered Structure of Operating System

As seen from the image, each upper layer is built on the bottom layer. All the layers hide some structures, operations etc from their upper layers.

One problem with the layered structure is that each layer needs to be carefully defined. This is necessary because the upper layers can only use the functionalities of the layers below them.

virtual machine (V

A virtual machine (VM) is a virtual environment that functions as a virtual computer system with its own CPU, memory, network interface, and storage, created on a physical hardware system (located off- or on-premises). Software called a hypervisor separates the machine's resources from the hardware and provisions them appropriately so they can be used by the VM.

The physical machines, equipped with a hypervisor such as Kernel-based Virtual Machine (KVM), is called the host machine, host computer, host operating system, or simply *host*. The many VMs that use its resources are guest machines, guest computers, guest operating systems, or simply *guests*. The hypervisor treats compute resources—like CPU, memory, and storage—as a pool of resources that can easily be relocated between existing guests or to new virtual machines.

VMs are isolated from the rest of the system, and multiple VMs can exist on a single piece of hardware, like a server. They can be moved between host servers depending on demand or to use resources more efficiently.

VMs allow multiple different operating systems to run simultaneously on a single computer—like a Linux® distro on a MacOS laptop. Each operating system runs in the same way an operating system or application normally would on the host hardware, so the end user experience emulated within the VM is nearly identical to a real-time operating system experience running on a physical machine.

Benefits of virtual machine:

Personalization:

There is great flexibility and personalization with virtual machines. No two environments have to be exactly the same; finance can have a different virtual environment than that of marketing. Likewise, depending on the user and their title, a virtual machine can be configured to maintain user defined settings for each session or can be configured to reset those settings at the beginning of each session.

Virtualization:

Virtualization involves using software to creating a "virtual", non-physical form or copy of something. This includes the reproduction of hardware in a digitally accessible manner. VMs can transform a workplace into a more mobile, flexible workplace where traditional hardware is replaced with virtual environments, infrastructure, and storage. Employees can access these virtual environments from anywhere at any time, allowing businesses to forgo the upfront costs of expensive hardware.

Backup by VM at a time:

With virtual machines, businesses can backup or store data one VM at a time. Instead of backing up individual files, the entire virtual machine can be backed up altogether. Remember, the VM is not locally hosted on the computer itself; instead all of the data and files used within the VM are accessed over a network like the internet. This model allows businesses to replicate or backup entire VMs at a time.

Easy Recover/Failover:

Since virtual machines can be backed up one VM at a time, the recovery/failover process is made much easier. Businesses not only can seamlessly failover to VMs, but can restore those VM's quickly and efficiently, making file and application access simple. This can help reduce downtime in the face of error or disaster.

Run multiple Operating Systems:

VM's allow users to work in multiple environments. For example, Microsoft Office users can run a Mac operating system or vice versa thanks to virtual machines. It's not just about running multiple operating systems, either. Users can utilize certain applications within the VM without having to provision it on their local computers. This saves IT professionals time, allowing them to work on patching, updating and testing the application, or even focus on other core competency projects.

OS and Application Updates:

Software updates are performed by the service provider that administers the VM. If there are new updates to the operating system, the service provider also manages those updates so that the end user's environment is always up to date. Applications that your employees use within the VM are also updated by the service provider, not your own IT team. This is what is referred to as Desktop as a Service and can save your IT team hours of monotonous updates.

Unit-2

Process Management (Principles and Brief Concept)

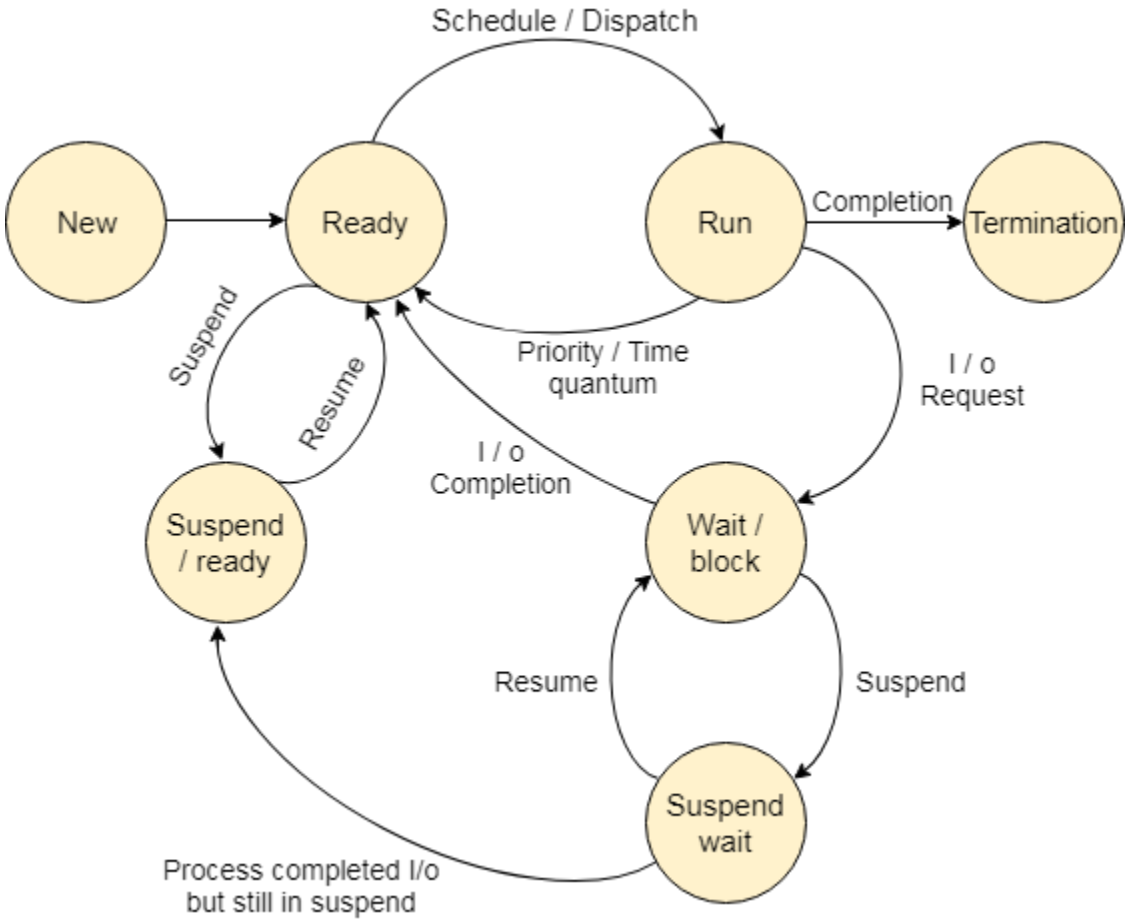
Process

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.

Process States

State Diagram



The process, from its creation to completion, passes through various states. The minimum number of states is five.

The names of the states are not standardized although the process may be in one of the following states during execution.

1. New

A program which is going to be picked up by the OS into the main memory is called a new process.

2. Ready

Whenever a process is created, it directly enters in the ready state, in which, it waits for the CPU to be assigned. The OS picks the new processes from the secondary memory and put all of them in the main memory.

The processes which are ready for the execution and reside in the main memory are called ready state processes. There can be many processes present in the ready state.

3. Running

One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm. Hence, if we have only one CPU in our system, the number of running processes for a particular time will always be one. If we have n processors in the system then we can have n processes running simultaneously.

4. Block or wait

From the Running state, a process can make the transition to the block or wait state depending upon the scheduling algorithm or the intrinsic behavior of the process.

When a process waits for a certain resource to be assigned or for the input from the user then the OS move this process to the block or wait state and assigns the CPU to the other processes.

5. Completion or termination

When a process finishes its execution, it comes in the termination state. All the context of the process (Process Control Block) will also be deleted the process will be terminated by the Operating system.

6. Suspend ready

A process in the ready state, which is moved to secondary memory from the main memory due to lack of the resources (mainly primary memory) is called in the suspend ready state.

If the main memory is full and a higher priority process comes for the execution then the OS have to make the room for the process in the main memory by throwing the lower priority process out into the secondary memory. The suspend ready processes remain in the secondary memory until the main memory gets available.

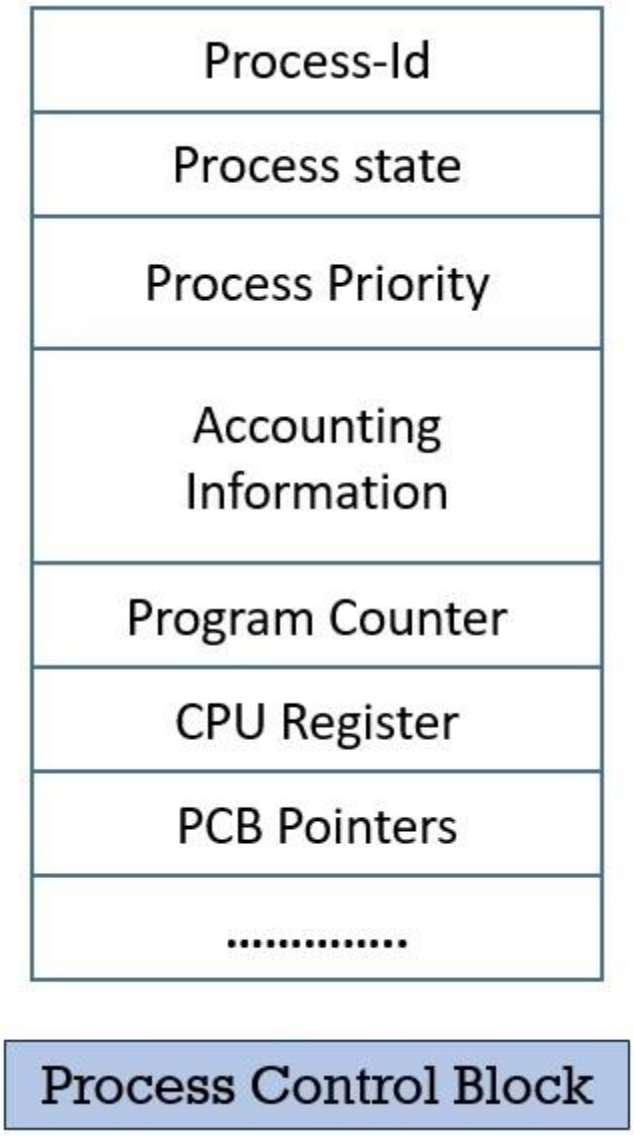
7. Suspend wait

Instead of removing the process from the ready queue, it's better to remove the blocked process which is waiting for some resources in the main memory. Since it is already waiting for some resource to get available hence it is better if it waits in the secondary memory and make room for the higher priority process. These processes complete their execution once the main memory gets available and their wait is finished.

Process Control Block (PCB)

For each process, the operating system maintains the **data structure**, which keeps the complete information about that process. This record or data structure is called **Process Control Block (PCB).**

Whenever a user creates a process, the operating system creates the corresponding PCB for that process. These PCBs of the processes are stored in the memory that is reserved for the operating system.



Process Control Block

The process control block has many fields that store the relative information about that process as you can see in the above figure. PCB contains Process-Id, Process State, Process Priority, Accounting Information, Program Counter, and also some other information which helps in controlling the operations of the process.

Here we will discuss the Process Control Block and its fields. So let us start.

Process-id

Whenever a new process is created by the user, the operating system allots a number to that process. This number becomes the unique identification of that process and it also helps in distinguishing that process from all other processes existing in the system. This number is also called as **process-id** of the process.

As we know, the operating system sets a limit on the maximum number of the processes it can deal with at a time. So, let us suppose that there are **n** number of the processes in the system. Now, the process-id will take on the values between 0 to n-1.

The operating system will allocate the value **0** to the first process that arrives in the system, number **1** to the next process and continues till **n-1**. At this point when the n-1 value is allocated to some process, and a new process arrives, the operating system wraps around and allocates value **0** to the newly arrived process. Considering that the process with process-id **0**, would have terminated.

Process-ids are not necessarily allocated in the ascending fashion. There is also another scheme to generated process-ids. Let us suppose, a PCB of a process requires **x** number of bytes and there are **n** number of processes in the system.

So, the operating system will reserve **nx** number of bytes for all the PCBs. And number the PCBs from **0 to n-1.** Now, when a process is created a fee PCB slot is allocated to that process and the PCB number itself becomes the process-id of the process.

In this case, the operating system has to maintain the chain of free PCBs. If the chain is empty no new process will get created.

Process State

A process in its lifetime undergoes different states. Like, a process may be in **waiting state, running state, ready state, blocked state, halted state**, and so on.

The PCBs field, **process state** holds the current state of the respective process. For example, if the process is currently executing. So, the process state will hold the **running** state for that process.

The information in the process state field is kept in the codified fashion.

Process Priority

The priority of the process is a **numeric** value, lesser the value, greater is the priority of that process. The priority of the process can be assigned externally by the user or by the operating system itself.

The process is assigned the priority at the time of its creation. The priority of the process may get changed over its lifetime depending on the various parameter. The parameters for changing the priority of the process can be the age of that process, the resources it consumed and so on.

Process Accounting Information

This field of PCB gives the account/description of the resources used by that process. Like, the amount of CPU time, real-time used, connect time.

Program Counter

The program counter is the pointer to an instruction in the program or code that is to be executed next. This field contains the address of the instruction that will be executed next in the process.

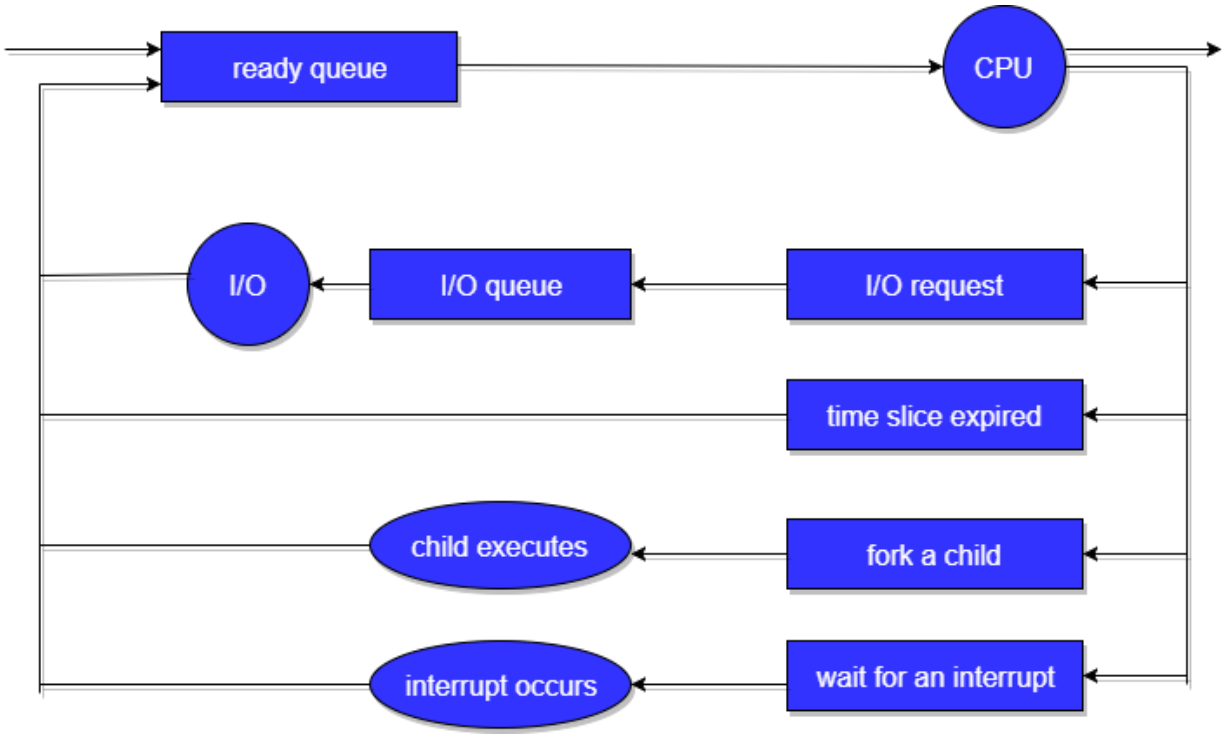
Scheduling Queues

What are Scheduling Queues?

- All processes, upon entering 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 available for each I/O device.

A new process is initially put in the **Ready queue**. It waits in the ready queue until it is selected for execution(or dispatched). Once the process is assigned to the CPU and is executing, one of the following several events can occur:

- The process could issue an I/O request, and then be placed in the **I/O queue**.
- The process could create a new subprocess and wait for its termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.



In the first two cases, the process eventually switches from the waiting state to the ready state, and is then put back in the ready queue. A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.

Job scheduler

A **job scheduler** is a computer application for controlling unattended background program execution of jobs. This is commonly called **batch scheduling**, as execution of non-interactive jobs is often called batch processing, though traditional *job* and *batch* are distinguished and contrasted; see that page for details. Other synonyms include **batch system**, **distributed resource management system** (**DRMS**), **distributed resource manager** (**DRM**), and, commonly today, **workload automation** (**WLA**). The data structure of jobs to run is known as the job queue.

Modern job schedulers typically provide a graphical user interface and a single point of control for definition and monitoring of background executions in a distributed network of computers. Increasingly, job schedulers are required to orchestrate the integration of real-time business activities with traditional background IT processing across different operating system platforms and business application environments.

Process scheduler

A scheduler is a type of system software that allows you to handle process scheduling.

There are mainly three types of Process Schedulers:

1. Long Term

2. Short Term
3. Medium Term

Long Term Scheduler

Long term scheduler is also known as a **job scheduler**. This scheduler regulates the program and select process from the queue and loads them into memory for execution. It also regulates the degree of multi-programing.

However, the main goal of this type of scheduler is to offer a balanced mix of jobs, like Processor, I/O jobs., that allows managing multiprogramming.

Medium Term Scheduler

Medium-term scheduling is an important part of **swapping**. It enables you to handle the swapped out-processes. In this scheduler, a running process can become suspended, which makes an I/O request.

A running process can become suspended if it makes an I/O request. A suspended processes can't make any progress towards completion. In order to remove the process from memory and make space for other processes, the suspended process should be moved to secondary storage.

Short Term Scheduler

Short term scheduling is also known as **CPU scheduler**. The main goal of this scheduler is to boost the system performance according to set criteria. This helps you to select from a group of processes that are ready to execute and allocates CPU to one of them. The dispatcher gives control of the CPU to the process selected by the short term scheduler.

Difference between Schedulers

Long-Term Vs. Short Term Vs. Medium-Term

Long-Term	**Short-Term**	**Medium-Term**
Long term is also known as a job scheduler	Short term is also known as CPU scheduler	Medium-term is also called swapping scheduler.
It is either absent or minimal in a time-sharing system.	It is insignificant in the time-sharing order.	This scheduler is an element of Time-sharing systems.
Speed is less compared to the short term scheduler.	Speed is the fastest compared to the short-term and medium-term scheduler.	It offers medium speed.

Long-Term	Short-Term	Medium-Term
Allow you to select processes from the loads and pool back into the memory	It only selects processes that is in a ready state of the execution.	It helps you to send process back to memory.
Offers full control	Offers less control	Reduce the level of multiprogramming.

Context Switch

A **context switch** occurs when a computer's CPU **switches from one process** or thread **to a different process** or thread.

- Context switching allows for one CPU to **handle** numerous processes or threads **without** the **need** for additional **processors**.
- A context switch 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.
- Any operating system that allows for multitasking relies heavily on the use of context switching to **allow different processes to run at the same time**.

Typically, there are three situations that a context switch is necessary, as shown below.

- **Multitasking** – When the CPU needs to switch processes in and out of memory, so that more than one process can be running.
- **Kernel/User Switch** – When switching between user mode to kernel mode, it may be used (but isn't always necessary).
- **Interrupts** – When the CPU is interrupted to return data from a disk read.

The steps in a full process switch are:

1. **Save the context** of the processor, including program counter and other registers.

2. **Update the process control block** of the process that is currently in the Running state. This includes changing the state of the process to one of the other states (Ready; Blocked; Ready/Suspend; or Exit). Other relevant fields must also be updated, including the reason for leaving the Running state and accounting information.

3. **Move** the **process control block** of this process to the **appropriate queue** (Ready; Blocked on Event *i ; Ready/Suspend).*

4. **Select another process** for execution.
5. Update the process control block of the process selected. This includes changing the state of this process to Running.

6. Update memory management data structures. This may be required, depending on how address translation is managed.

7. **Restore the context** of the processor to that which existed at the time the selected process was last switched out of the Running state, by loading in the previous values of the program counter and other registers.

Different Operations on Processes

There are many operations that can be performed on processes. Some of these are process creation, process preemption, process blocking, and process termination. These are given in detail as follows −

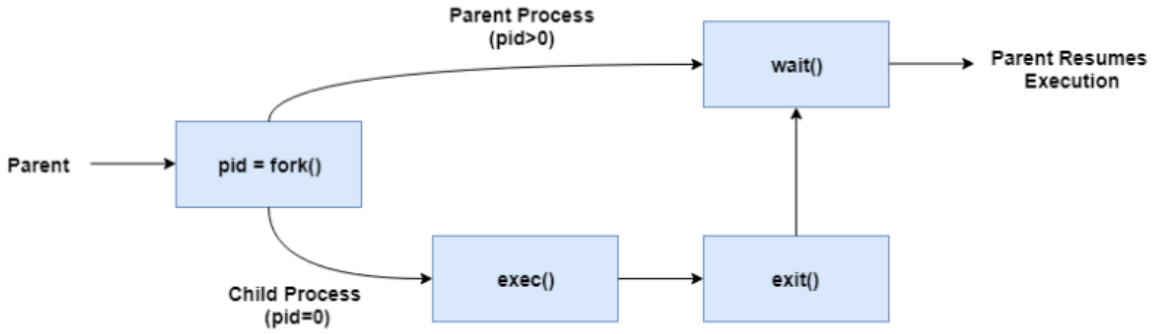
Process Creation

Processes need to be created in the system for different operations. This can be done by the following events −

- User request for process creation
- System initialization
- Execution of a process creation system call by a running process
- Batch job initialization

A process may be created by another process using fork(). The creating process is called the parent process and the created process is the child process. A child process can have only one parent but a parent process may have many children. Both the parent and child processes have the same memory image, open files, and environment strings. However, they have distinct address spaces.

A diagram that demonstrates process creation using fork() is as follows −

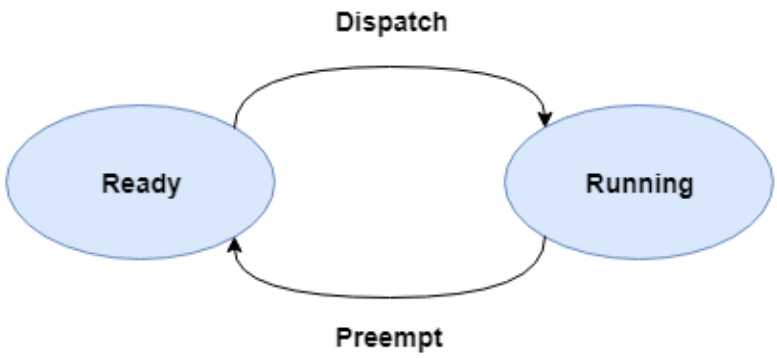


Process Creation Using fork()

Process Preemption

An interrupt mechanism is used in preemption that suspends the process executing currently and the next process to execute is determined by the short-term scheduler. Preemption makes sure that all processes get some CPU time for execution.

A diagram that demonstrates process preemption is as follows −

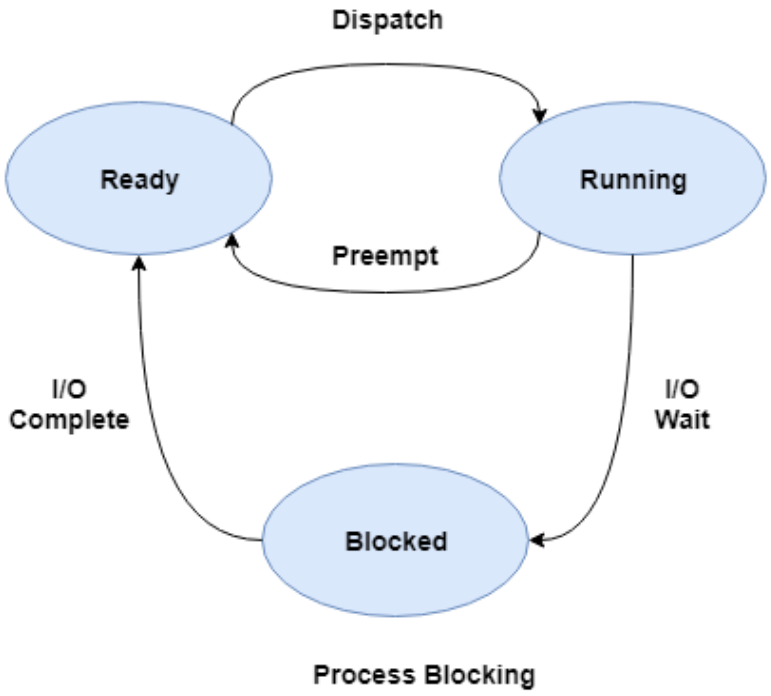


Process Preemption

Process Blocking

The process is blocked if it is waiting for some event to occur. This event may be I/O as the I/O events are executed in the main memory and don't require the processor. After the event is complete, the process again goes to the ready state.

A diagram that demonstrates process blocking is as follows −



Process Termination

After the process has completed the execution of its last instruction, it is terminated. The resources held by a process are released after it is terminated.

A child process can be terminated by its parent process if its task is no longer relevant. The child process sends its status information to the parent process before it terminates. Also, when a parent process is terminated, its child processes are terminated as well as the child processes cannot run if the parent processes are terminated.

Inter Process Communication (IPC)

A process can be of two types:

- Independent process.
- Co-operating process.

An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently, in reality, there are many situations when co-operative nature can be utilised for increasing computational speed, convenience and modularity. Inter process communication (IPC) is a mechanism which allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other through both:

1. Shared Memory
2. Message passing

The Figure 1 below shows a basic structure of communication between processes via the shared memory method and via the message passing method.

An operating system can implement both method of communication. First, we will discuss the shared memory methods of communication and then message passing. Communication between processes using shared memory requires processes to share some variable and it completely depends on how programmer will implement it. One way of communication using shared memory can be imagined like this: Suppose process1 and process2 are executing simultaneously and they share some resources or use some information from another process. Process1 generate information about certain computations or resources being used and keeps it as a record in shared memory. When process2 needs to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly. Processes can use shared memory for extracting information as a record from another process as well as for delivering any specific information to other processes.
Let's discuss an example of communication between processes using shared memory method.

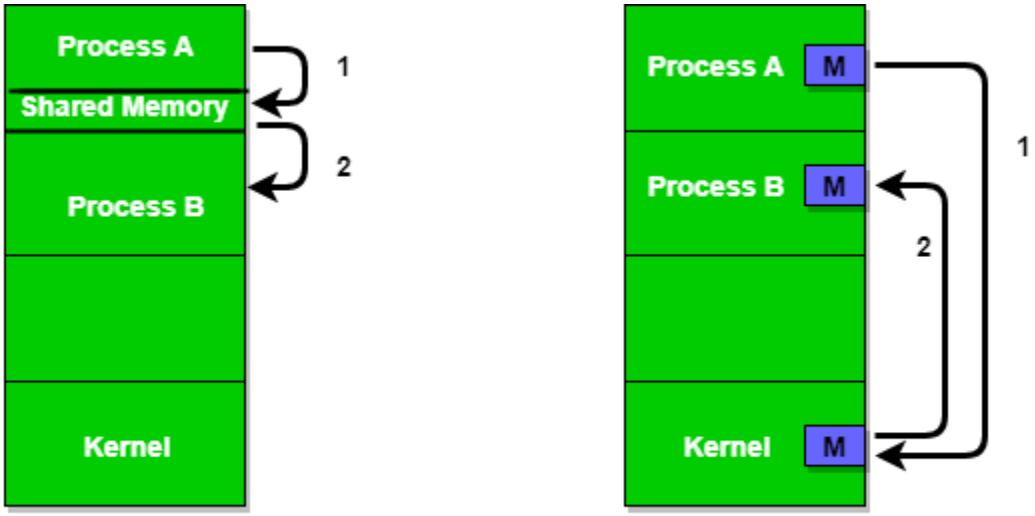


Figure 1 - Shared Memory and Message Passing

i) Shared Memory Method

Ex: Producer-Consumer problem

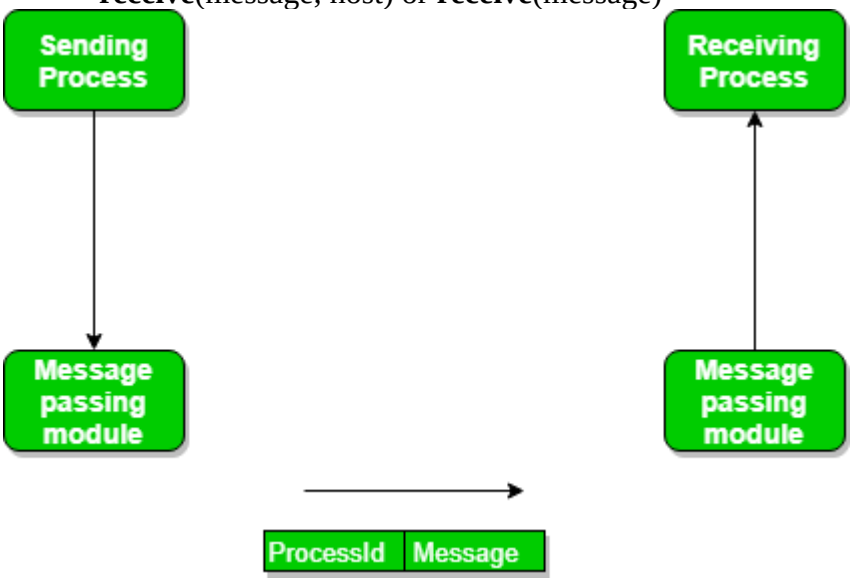
There are two processes: Producer and Consumer. Producer produces some item and Consumer consumes that item. The two processes share a common space or memory location known as a buffer where the item produced by Producer is stored and from which the Consumer consumes the item, if needed. There are two versions of this problem: the first one is known as unbounded buffer

problem in which Producer can keep on producing items and there is no limit on the size of the buffer, the second one is known as the bounded buffer problem in which Producer can produce up to a certain number of items before it starts waiting for Consumer to consume it. We will discuss the bounded buffer problem. First, the Producer and the Consumer will share some common memory, then producer will start producing items. If the total produced item is equal to the size of buffer, producer will wait to get it consumed by the Consumer. Similarly, the consumer will first check for the availability of the item. If no item is available, Consumer will wait for Producer to produce it. If there are items available, Consumer will consume it.

Messaging Passing Method

Now, We will start our discussion of the communication between processes via message passing. In this method, processes communicate with each other without using any kind of shared memory. If two processes p1 and p2 want to communicate with each other, they proceed as follows:

- Establish a communication link (if a link already exists, no need to establish it again.)
- Start exchanging messages using basic primitives.
  We need at least two primitives:
  – **send**(message, destinaion) or **send**(message)
  – **receive**(message, host) or **receive**(message)



The message size can be of fixed size or of variable size. If it is of fixed size, it is easy for an OS designer but complicated for a programmer and if it is of variable size then it is easy for a programmer but complicated for the OS designer. A standard message can have two parts: **header and body.**

The **header part** is used for storing message type, destination id, source id, message length, and control information. The control information contains information like what to do if runs out of buffer space, sequence number, priority. Generally, message is sent using FIFO style.

Message Passing through Communication Link.

Direct and Indirect Communication link

Now, We will start our discussion about the methods of implementing communication link. While implementing the link, there are some questions which need to be kept in mind like :

1. How are links established?
2. Can a link be associated with more than two processes?
3. How many links can there be between every pair of communicating processes?
4. What is the capacity of a link? Is the size of a message that the link can accommodate fixed or variable?
5. Is a link unidirectional or bi-directional?

A link has some capacity that determines the number of messages that can reside in it temporarily for which every link has a queue associated with it which can be of zero capacity, bounded capacity, or unbounded capacity. In zero capacity, the sender waits until the receiver informs the sender that it has received the message. In non-zero capacity cases, a process does not know whether a message has been received or not after the send operation. For this, the sender must communicate with the receiver explicitly. Implementation of the link depends on the situation, it can be either a direct communication link or an in-directed communication link.
Direct Communication links are implemented when the processes uses a specific process identifier for the communication, but it is hard to identify the sender ahead of time.
For example: the print server.

In-direct Communication is done via a shared mailbox (port), which consists of a queue of messages. The sender keeps the message in mailbox and the receiver picks them up.

Preemptive and Non-Preemptive Scheduling

1. Preemptive Scheduling:

Preemptive scheduling is used when a process switches from running state to ready state or from waiting state to ready state. The resources (mainly CPU cycles) are allocated to the process for the limited amount of time and then is taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining. That process stays in ready queue till it gets next chance to execute.

Algorithms based on preemptive scheduling are: Round Robin (RR),Shortest Remaining Time First (SRTF), Priority (preemptive version), etc.

Process	Arrival Time	CPU Burst Time (in millisec.)
P0	3	2
P1	2	4
P2	0	6
P3	1	4

Preemptive Scheduling

2. Non-Preemptive Scheduling:

Non-preemptive Scheduling is used when a process terminates, or a process switches from running to waiting state. In this scheduling, once the resources (CPU cycles) is allocated to a process, the process holds the CPU till it gets terminated or it reaches a waiting state. In case of non-preemptive scheduling does not interrupt a process running CPU in middle of the execution. Instead, it waits till the process complete its CPU burst time and then it can allocate the CPU to another process.

Algorithms based on non-preemptive scheduling are: Shortest Job First (SJF basically non preemptive) and Priority (non preemptive version), etc.

Process	Arrival Time	CPU Burst Time (in millisec.)
P0	3	2
P1	2	4
P2	0	6
P3	1	4

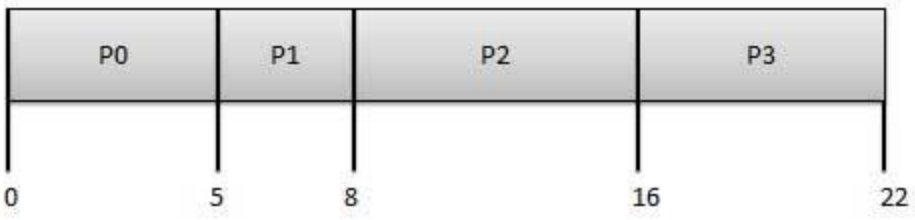
0 6 10 14 16

Non-Preemtive Scheduling

First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, 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-preemptive, 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 processer 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

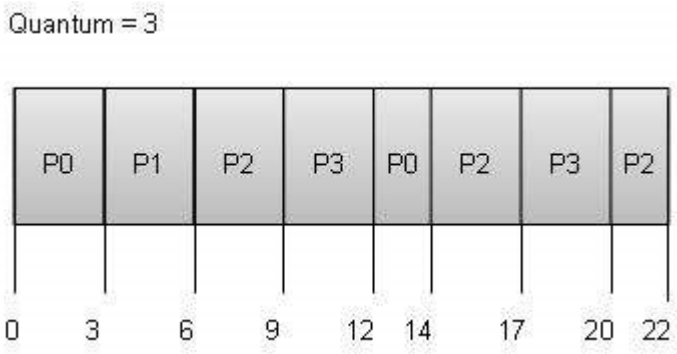
Waiting time of each process is as follows −

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

Round Robin Scheduling

- Round Robin is the preemptive 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 preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.



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

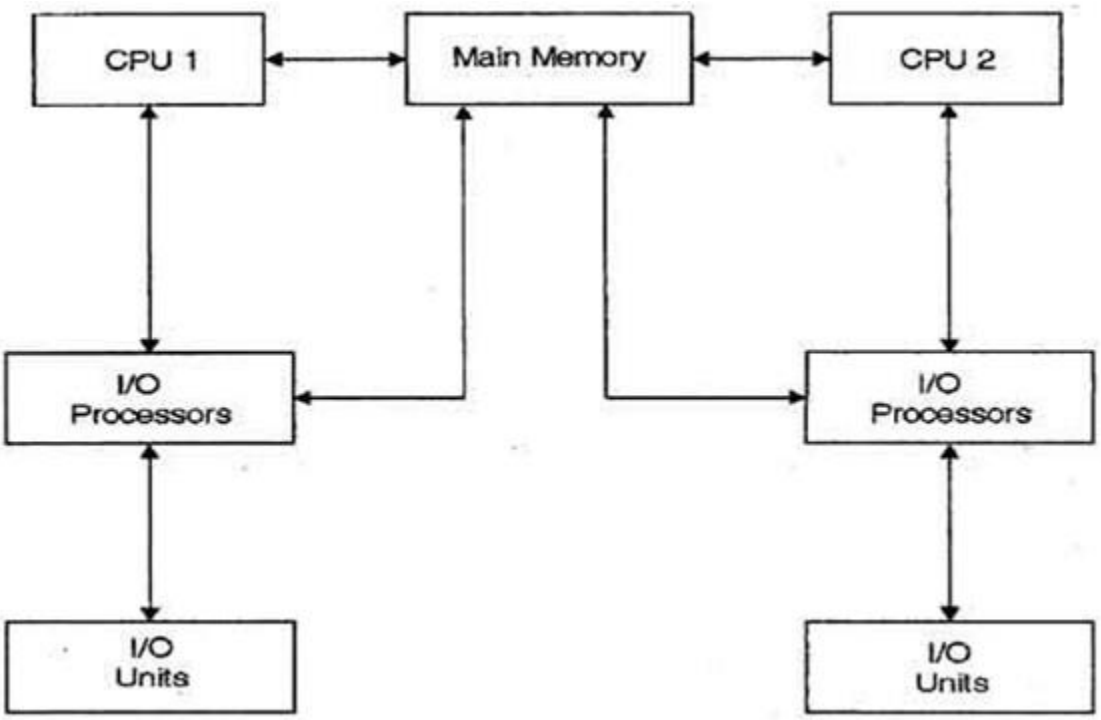
Multiple-Processor Scheduling

Multiprocessor Operating system

A multiprocessor system consists of several processors which share memory. In the multiprocessor, there is more than one processor in the system. The reason we use multiprocessor is that sometimes load on the processor is very high but input output on other function is not required. This type of operating system is more reliable as even if on processor goes down the other can still continues to work. This system is relatively cheap because we are only having the copies of processor but other devices like input-output and Memory are shared. In the multiprocessor system all the processor operate under the single operating system. Multiplicity of the processor and how the processors work together are transparent to the other.

In this, the user does not know in which processor their process work. A process is divided into several small processes and they work independently on the different processor. A system can be both multi-programmed by having multiple programs running at the same time and multiprocessing by having more than one physical and the processor.

In this diagram, there are more than 1 CPU and they shared a common memory.



Multiprocessing scheduling

In the multiprocessor scheduling, there are multiple CPU's which share the load so that various process run simultaneously. In general, the 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 CPU's in the system are in the close communication which shares a common bus, memory and other peripheral devices. So we can say that the system is a tightly coupled system. These systems are used when we want to process a bulk amount of data. These systems are mainly used in satellite, weather forecasting etc.

Multiprocessing system work on the concept of symmetric multiprocessing model. In this system, each processor work on the identical copy of the operating system and these copies communicate with each other. We the help of this system we can save money because of other devices like peripherals. Power supplies and other devices are shared. The most important thing is that we can do more work in a short period of time. If one system fails in the multiprocessor system the whole system will not halt only the speed of the processor will be slow down. The whole performance of the multiprocessing system is managed by the operating system . operating system assigns different task to the different processor in the system. In the multiprocessing system, the process is broken into the thread which they can be run independently. These type of system allow the threads to run on more than one processor simultaneously. In these systems the various process in

the parallel so this is called parallel processor. Parallel processing is the ability of the CPU to run various process simultaneously. In the multiprocessing system, there is dynamically sharing of resources among the various processors.

Multiprocessor operating system is a kind of regular OS which handles many systems calls at the same time, do memory management, provide file management also the input-output devices.

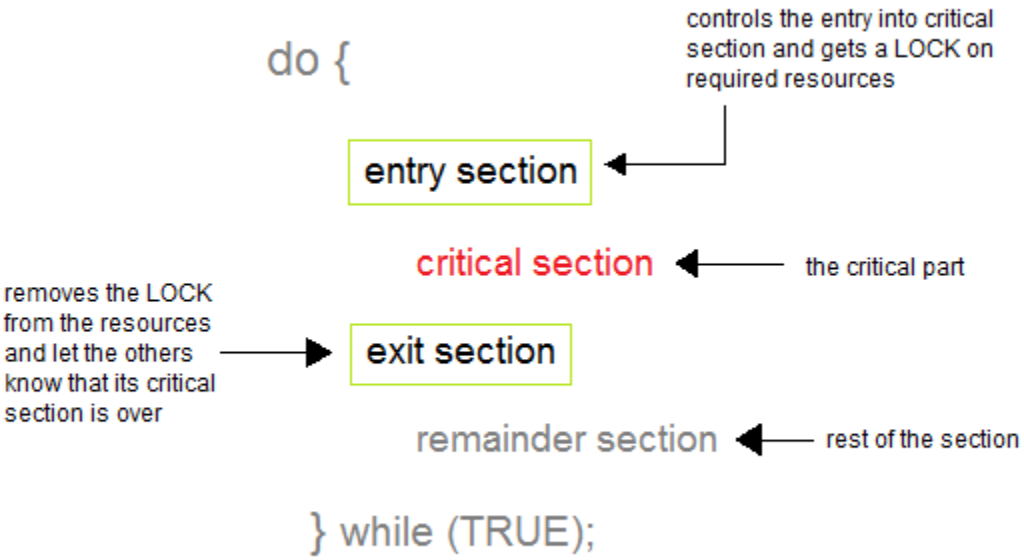
Process Synchronization

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.

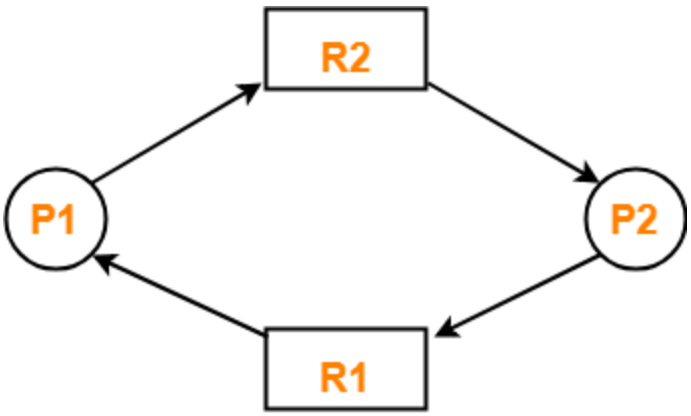
Unit-3

Deadlock in OS-

Deadlock is a situation where-

- The execution of two or more processes is blocked because each process holds some resource and waits for another resource held by some other process.

Example-



Example of a deadlock

Here

- Process P1 holds resource R1 and waits for resource R2 which is held by process P2.
- Process P2 holds resource R2 and waits for resource R1 which is held by process P1.

- None of the two processes can complete and release their resource.
- Thus, both the processes keep waiting infinitely.

Conditions For Deadlock-

There are following 4 necessary conditions for the occurrence of deadlock-

1. Mutual Exclusion
2. Hold and Wait
3. No preemption
4. Circular wait

1. Mutual Exclusion-

By this condition,

- There must exist at least one resource in the system which can be used by only one process at a time.
- If there exists no such resource, then deadlock will never occur.
- Printer is an example of a resource that can be used by only one process at a time.

2. Hold and Wait-

By this condition,

- There must exist a process which holds some resource and waits for another resource held by some other process.

3. No Preemption-

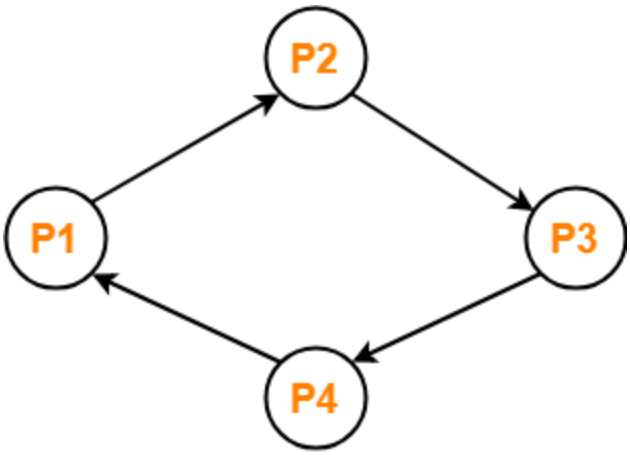
By this condition,

- Once the resource has been allocated to the process, it can not be preempted.
- It means resource can not be snatched forcefully from one process and given to the other process.
- The process must release the resource voluntarily by itself.

4. Circular Wait-

By this condition,

- All the processes must wait for the resource in a cyclic manner where the last process waits for the resource held by the first process.



Circular Wait

Here,

- Process P1 waits for a resource held by process P2.
- Process P2 waits for a resource held by process P3.
- Process P3 waits for a resource held by process P4.
- Process P4 waits for a resource held by process P1.

Methods for Handling Deadlocks

Deadlock detection, deadlock prevention and deadlock avoidance are the main methods for handling deadlocks. Details about these are given as follows −

Deadlock Detection

Deadlock can be detected by the resource scheduler as it keeps track of all the resources that are allocated to different processes. After a deadlock is detected, it can be handed using the given methods −

- All the processes that are involved in the deadlock are terminated. This approach is not that useful as all the progress made by the processes is destroyed.
- Resources can be preempted from some processes and given to others until the deadlock situation is resolved.

Deadlock Prevention

It is important to prevent a deadlock before it can occur. So, the system checks each transaction before it is executed to make sure it does not lead to deadlock. If there is even a slight possibility that a transaction may lead to deadlock, it is never allowed to execute.

Some deadlock prevention schemes that use timestamps in order to make sure that a deadlock does not occur are given as follows −

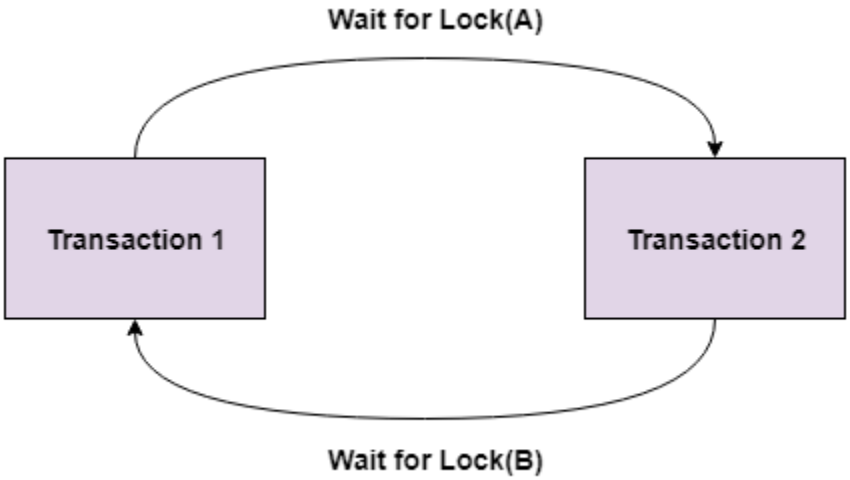
- **Wait - Die Scheme**
- In the wait - die scheme, if a transaction T1 requests for a resource that is held by transaction T2, one of the following two scenarios may occur −
    - TS(T1) < TS(T2) - If T1 is older than T2 i.e T1 came in the system earlier than T2, then it is allowed to wait for the resource which will be free when T2 has completed its execution.
    - TS(T1) > TS(T2) - If T1 is younger than T2 i.e T1 came in the system after T2, then T1 is killed. It is restarted later with the same timestamp.
- **Wound - Wait Scheme**
- In the wound - wait scheme, if a transaction T1 requests for a resource that is held by transaction T2, one of the following two possibilities may occur −
    - TS(T1) < TS(T2) - If T1 is older than T2 i.e T1 came in the system earlier than T2, then it is allowed to roll back T2 or wound T2. Then T1 takes the resource and completes its execution. T2 is later restarted with the same timestamp.
    - TS(T1) > TS(T2) - If T1 is younger than T2 i.e T1 came in the system after T2, then it is allowed to wait for the resource which will be free when T2 has completed its execution.

Deadlock Avoidance

It is better to avoid a deadlock rather than take measures after the deadlock has occurred. The wait for graph can be used for deadlock avoidance. This is however only useful for smaller databases as it can get quite complex in larger databases.

Wait for graph

The wait for graph shows the relationship between the resources and transactions. If a transaction requests a resource or if it already holds a resource, it is visible as an edge on the wait for graph. If the wait for graph contains a cycle, then there may be a deadlock in the system, otherwise not.



Ostrich Algorithm

The ostrich algorithm means that the deadlock is simply ignored and it is assumed that it will never occur. This is done because in some systems the cost of handling the deadlock is much higher than simply ignoring it as it occurs very rarely. So, it is simply assumed that the deadlock will never occur and the system is rebooted if it occurs by any chance.

Recovery from Deadlock

When a Deadlock Detection Algorithm determines that a deadlock has occurred in the system, the system must recover from that deadlock. There are two approaches of breaking a Deadlock:

1. Process Termination:

To eliminate the deadlock, we can simply kill one or more processes. For this, we use two methods:

- **(a). Abort all the Deadlocked Processes:**
  Aborting all the processes will certainly break the deadlock, but with a great expenses. The deadlocked processes may have computed for a long time and the result of those partial computations must be discarded and there is a probability to recalculate them later.
- **(b). Abort one process at a time untill deadlock is eliminated:**
  Abort one deadlocked process at a time, untill deadlock cycle is eliminated from the system. Due to this method, there may be considerable overhead, because after aborting each process, we have to run deadlock detection algorithm to check whether any processes are still deadlocked.

2. Resource Preemption:

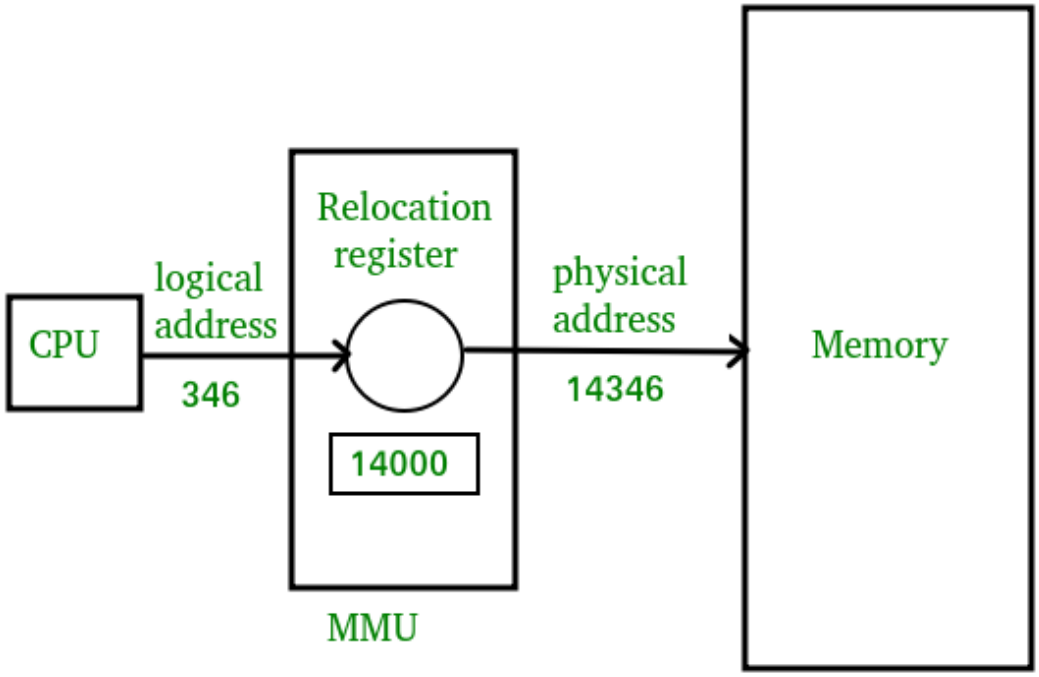
To eliminate deadlocks using resource preemption, we preepmt some resources from processes and give those resources to other processes. This method will raise three issues –

- **(a). Selecting a victim:**
  We must determine which resources and which processes are to be preempted and also the order to minimize the cost.
- **(b). Rollback:**
  We must determine what should be done with the process from which resources are preempted. One simple idea is total rollback. That means abort the process and restart it.
- **(c). Starvation:**
  In a system, it may happen that same process is always picked as a victim. As a result, that process will never complete its designated task. This situation is called **Starvation** and must be avoided. One solution is that a process must be picked as a victim only a finite number of times.

Unit-4

Logical and Physical Address

Logical Address is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically, therefore, it is also known as Virtual Address. This address is used as a reference to access the physical memory location by CPU. The term Logical Address Space is used for the set of all logical addresses generated by a program's perspective. The hardware device called Memory-Management Unit is used for mapping logical address to its corresponding physical address.
Physical Address identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, the logical address must be mapped to the physical address by MMU before they are used. The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.



Mapping virtual-address to physical-addresses

Differences Between Logical and Physical Address in Operating System

1. The basic difference between Logical and physical address is that Logical address is generated by CPU in perspective of a program whereas the physical address is a location that exists in the memory unit.
2. Logical Address Space is the set of all logical addresses generated by CPU for a program whereas the set of all physical address mapped to corresponding logical addresses is called Physical Address Space.
3. The logical address does not exist physically in the memory whereas physical address is a location in the memory that can be accessed physically.
4. Identical logical addresses are generated by Compile-time and Load time address binding methods whereas they differs from each other in run-time address binding method. Please refer this for details.
5. The logical address is generated by the CPU while the program is running whereas the physical address is computed by the Memory Management Unit (MMU).

Comparison Chart:

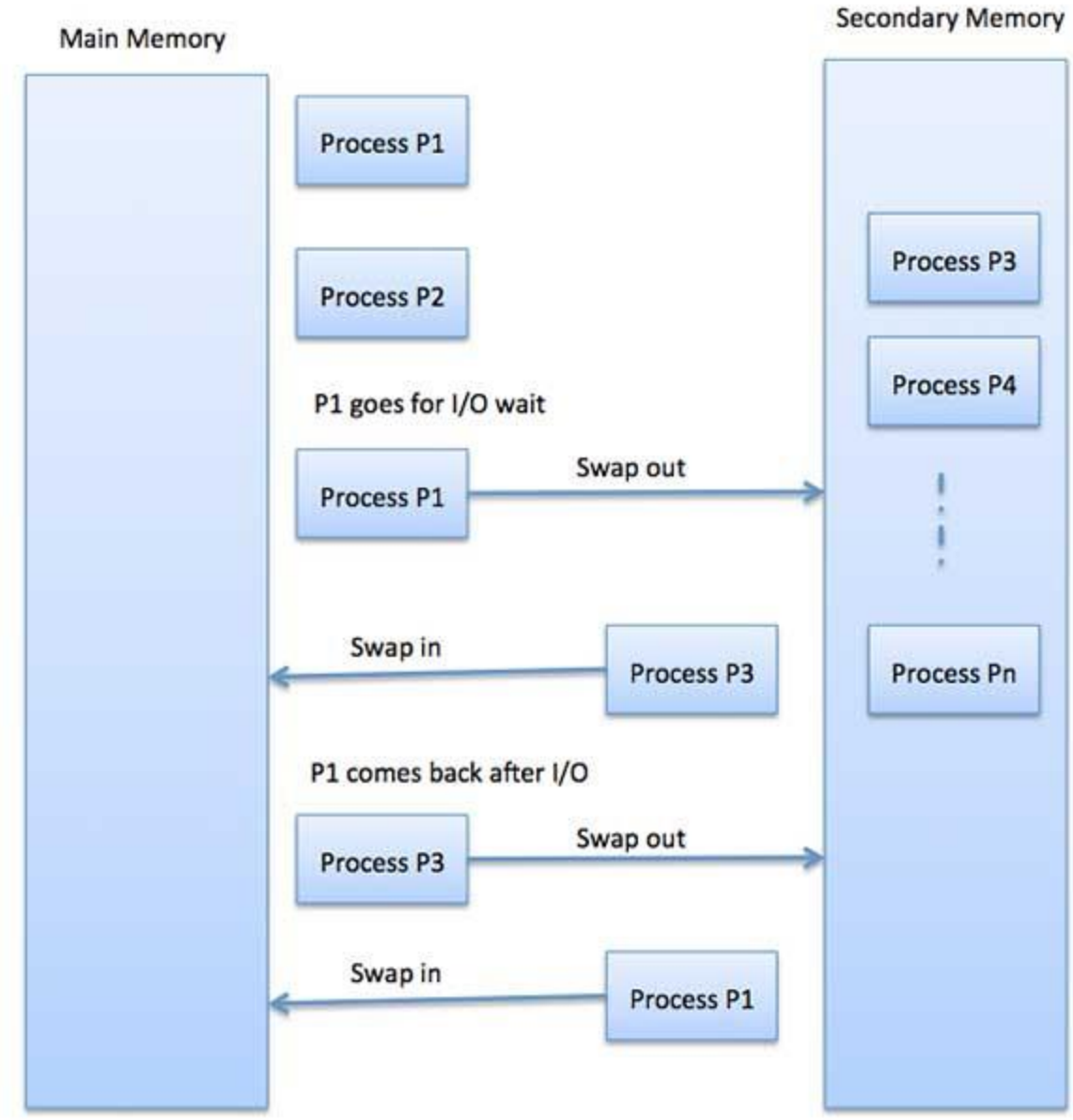
PARAMENTER	LOGICAL ADDRESS	PHYSICAL ADDRESS
Basic	generated by CPU	location in a memory unit

PARAMENTER	LOGICAL ADDRESS	PHYSICAL ADDRESS
Address Space	Logical Address Space is set of all logical addresses generated by CPU in reference to a program.	Physical Address is set of all physical addresses mapped to the corresponding logical addresses.
Visibility	User can view the logical address of a program.	User can never view physical address of program.
Generation	generated by the CPU	Computed by MMU
Access	The user can use the logical address to access the physical address.	The user can indirectly access physical address but not directly.

Swapping

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason **Swapping is also known as a technique for memory compaction**.



The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Let us assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to or from memory will take

2048KB / 1024KB per second
= 2 seconds
= 2000 milliseconds

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.

Memory Allocation

Main memory usually has two partitions −

- **Low Memory** − Operating system resides in this memory.
- **High Memory** − User processes are held in high memory.

Operating system uses the following memory allocation mechanism.

S.N.	Memory Allocation & Description
1	**Single-partition allocation** In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.
2	**Multiple-partition allocation** In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

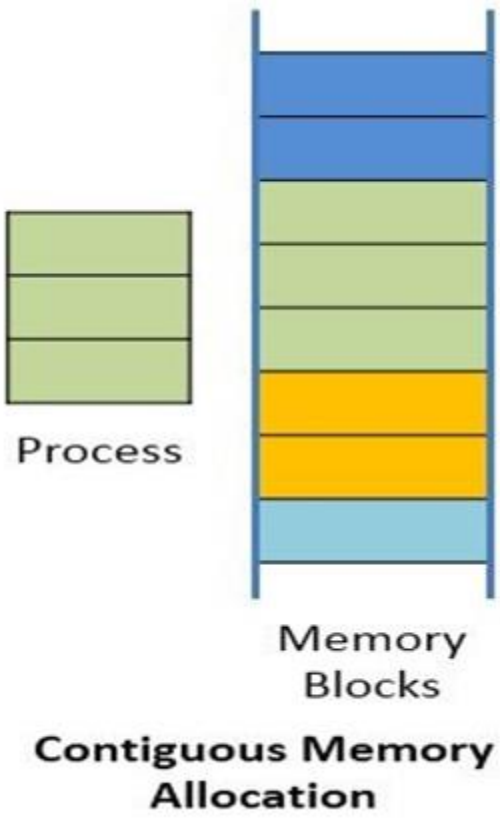
Contiguous Memory allocation

In **contiguous memory allocation**, all the available memory space remain together in one place. It means freely available memory partitions are not scattered here and there across the whole memory space.

In the **contiguous memory allocation**, both the operating system and the user must reside in the main memory. The main memory is divided into two portions one portion is for the operating and other is for the user program.

In the **contiguous memory allocation** when any user process request for the memory a single section of the contiguous memory block is given to that process according to its need. We can achieve contiguous memory allocation by dividing memory into the fixed-sized partition.

A single process is allocated in that fixed sized single partition. But this will increase the degree of multiprogramming means more than one process in the main memory that bounds the number of fixed partition done in memory. Internal fragmentation increases because of the contiguous memory allocation.



→ Fixed sized partition

In the fixed sized partition the system divides memory into fixed size partition (may or may not be of the same size) here entire partition is allowed to a process and if there is some wastage inside the partition is allocated to a process and if there is some wastage inside the partition then it is called internal fragmentation.

Advantage: Management or book keeping is easy.

Disadvantage: Internal fragmentation

→ Variable size partition

In the variable size partition, the memory is treated as one unit and space allocated to a process is exactly the same as required and the leftover space can be reused again.

Advantage: There is no internal fragmentation.

Disadvantage: Management is very difficult as memory is becoming purely fragmented after some time.

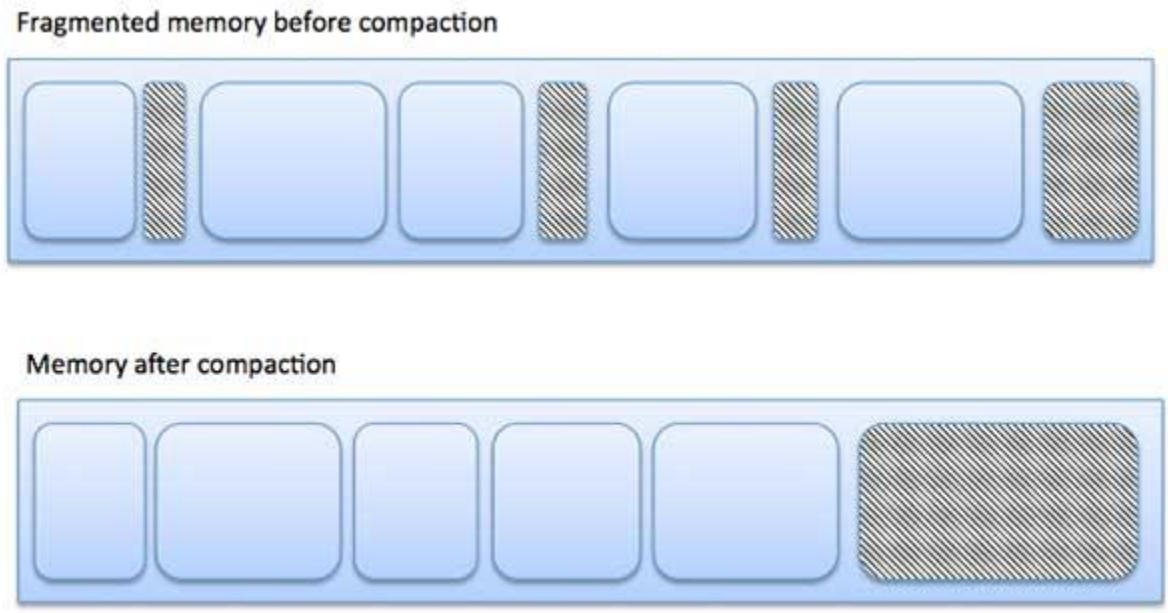
Fragmentation

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types −

S.N.	Fragmentation & Description
1	**External fragmentation** Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
2	**Internal fragmentation** Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory −

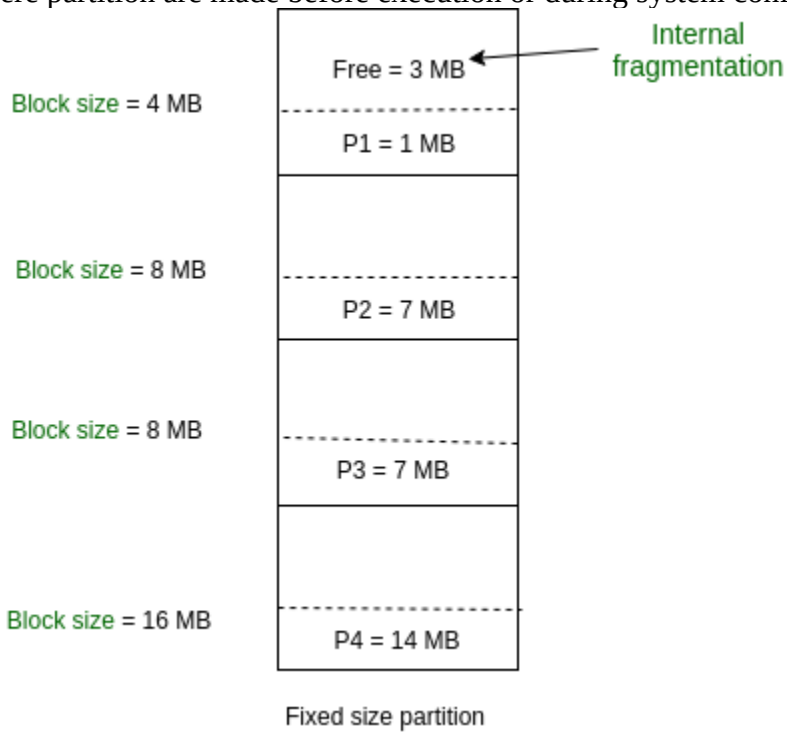


External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

Fixed Partitioning:

This is the oldest and simplest technique used to put more than one processes in the main memory. In this partitioning, number of partitions (non-overlapping) in RAM are **fixed but size** of each partition may or **may not be same**. As it is **contiguous** allocation, hence no spanning is allowed. Here partition are made before execution or during system configure.



Fixed size partition

As illustrated in above figure, first process is only consuming 1MB out of 4MB in the main memory.
Hence, Internal Fragmentation in first block is (4-1) = 3MB.
Sum of Internal Fragmentation in every block = (4-1)+(8-7)+(8-7)+(16-14)= 3+1+1+2 = 7MB.

Suppose process P5 of size 7MB comes. But this process cannot be accommodated inspite of available free space because of contiguous allocation (as spanning is not allowed). Hence, 7MB becomes part of External Fragmentation.

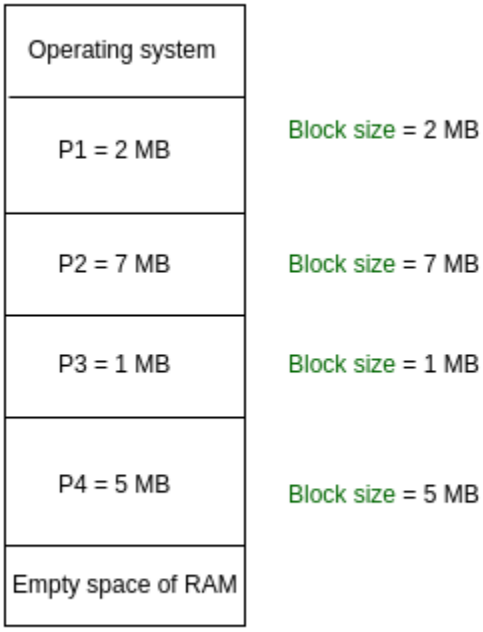
Variable Partitioning –

It is a part of Contiguous allocation technique. It is used to alleviate the problem faced by Fixed Partitioning. In contrast with fixed partitioning, partitions are not made before the execution or during system configure. Various **features** associated with variable Partitioning-

1. Initially RAM is empty and partitions are made during the run-time according to process's need instead of partitioning during system configure.
2. The size of partition will be equal to incoming process.

3. The partition size varies according to the need of the process so that the internal fragmentation can be avoided to ensure efficient utilisation of RAM.
4. Number of partitions in RAM is not fixed and depends on the number of incoming process and Main Memory's size.

Dynamic partitioning



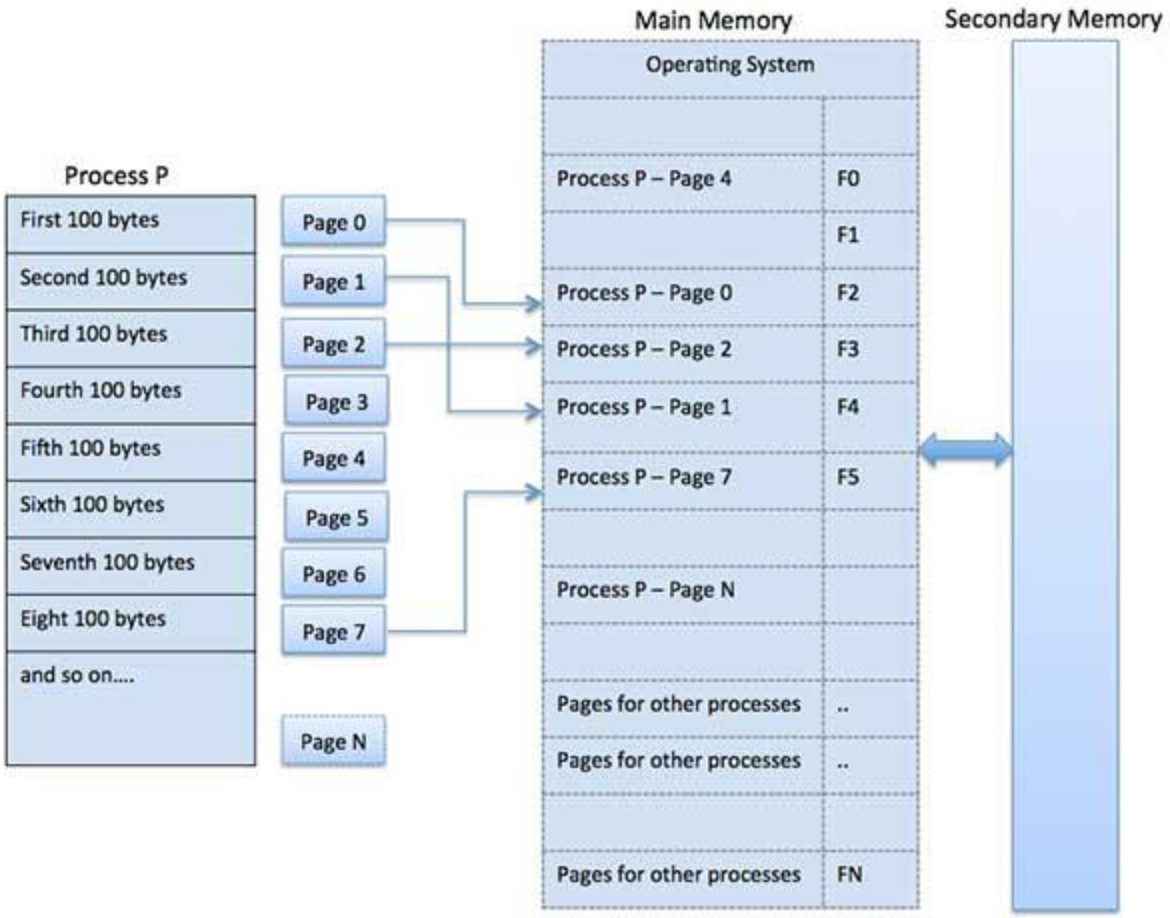
Partition size = process size
So, no internal Fragmentation

Paging

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.



Address Translation

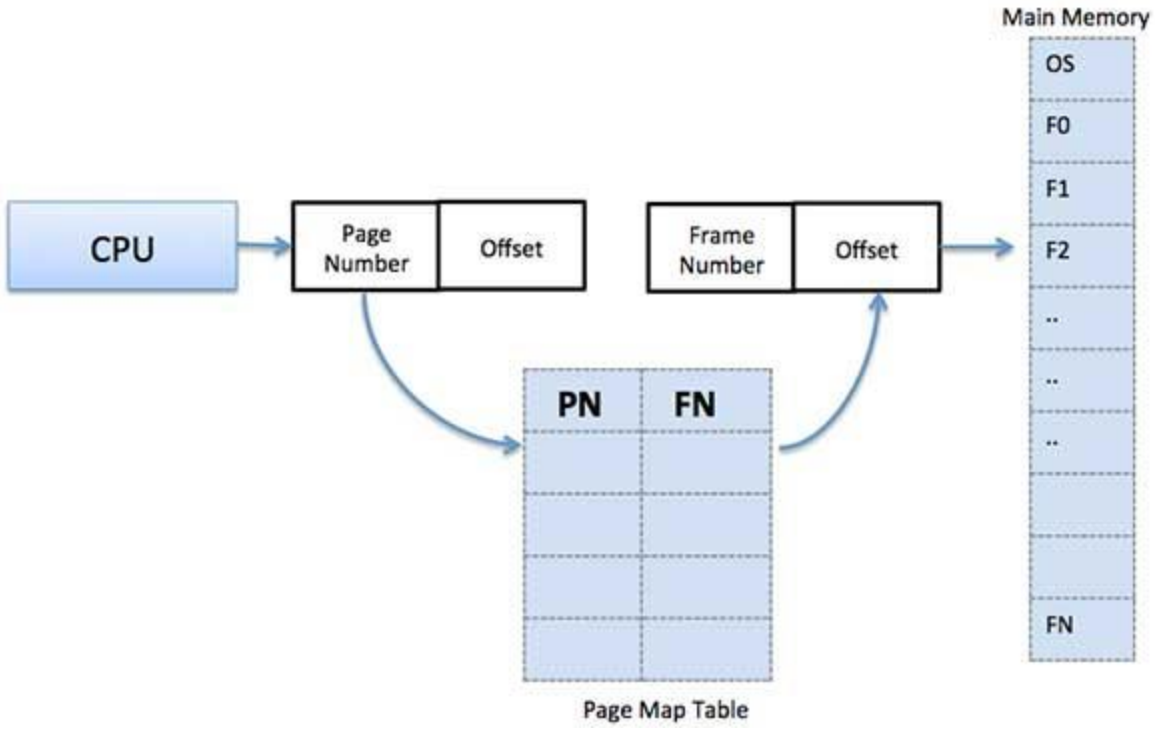
Page address is called **logical address** and represented by **page number** and the **offset**.

Logical Address = Page number + page offset

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

Physical Address = Frame number + page offset

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.



When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging −

- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

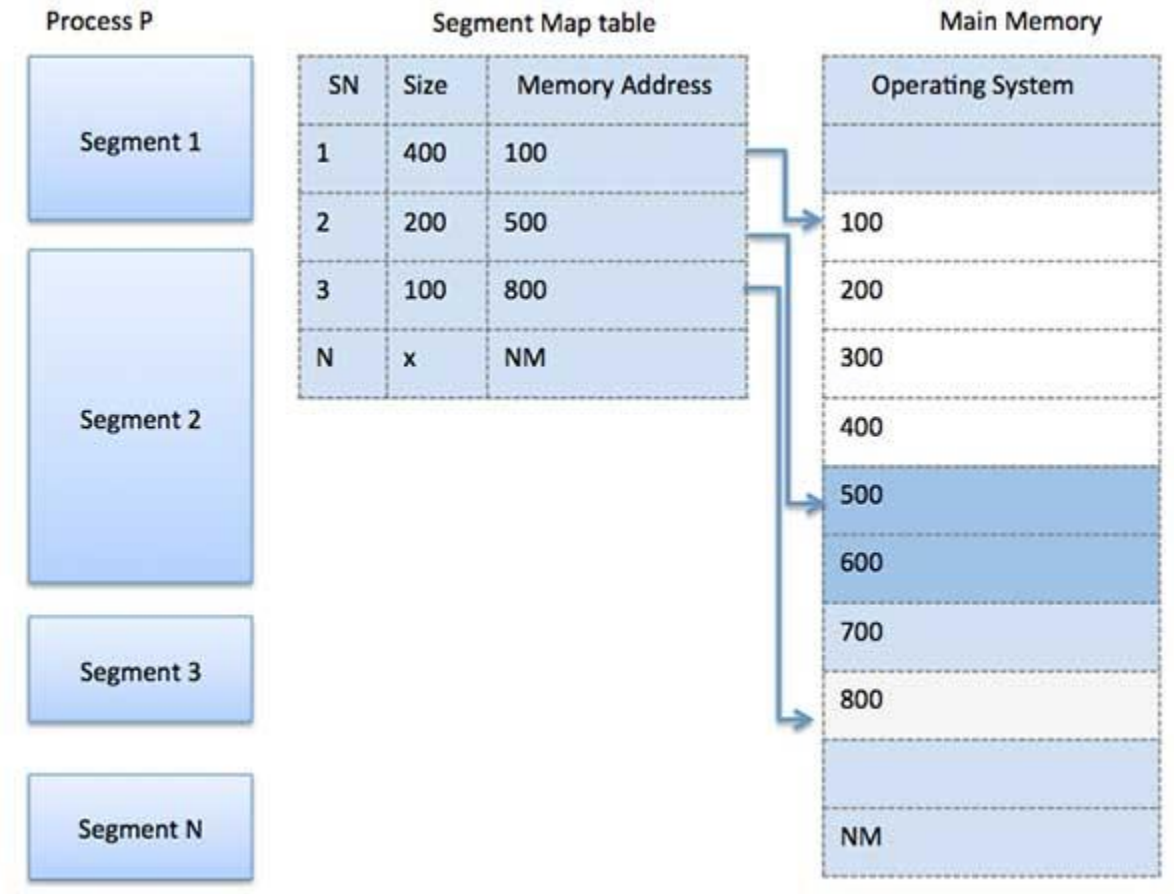
Segmentation

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.



Virtual Memory

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of main memory. The addresses a program may use to reference memory are

distinguished from the addresses the memory system uses to identify physical storage sites, and program generated addresses are translated automatically to the corresponding machine addresses. The size of virtual storage is limited by the addressing scheme of the computer system and amount of secondary memory is available not by the actual number of the main storage locations.

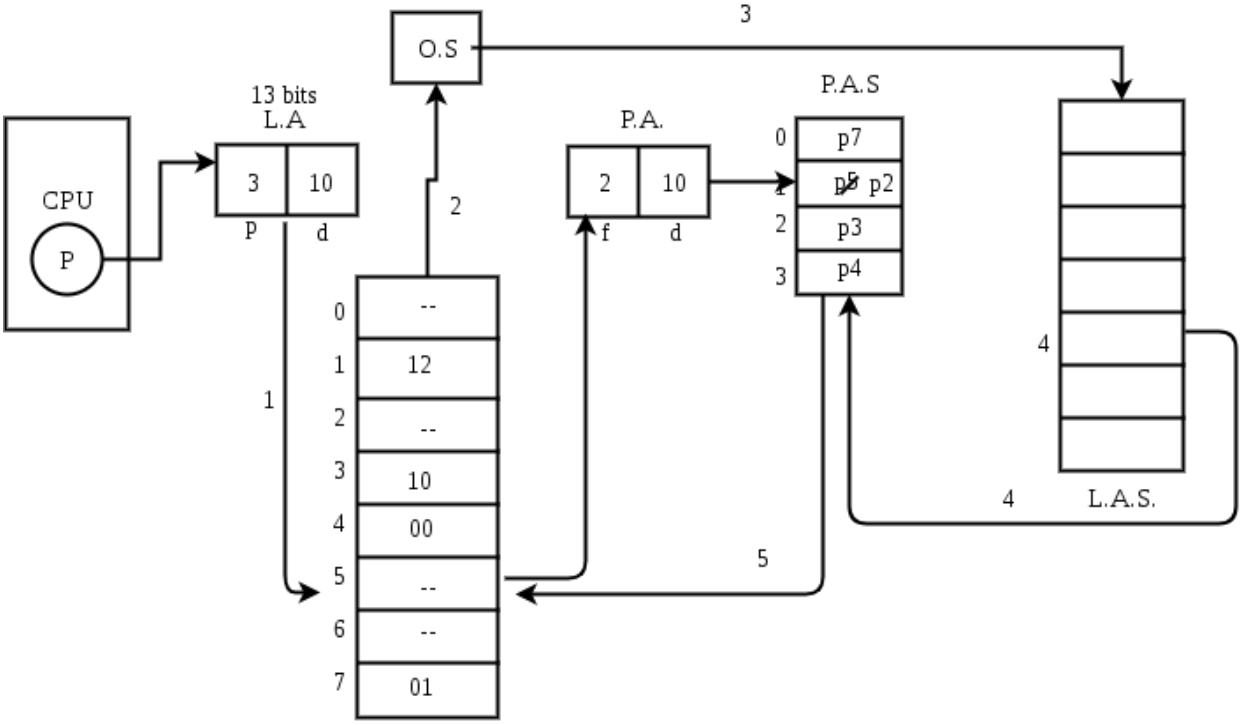
It is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.

1. All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of main memory such that it occupies different places in main memory at different times during the course of execution.
2. A process may be broken into number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time address translation and use of page or segment table permits this.

If these characteristics are present then, it is not necessary that all the pages or segments are present in the main memory during execution. This means that the required pages need to be loaded into memory whenever required. Virtual memory is implemented using Demand Paging or Demand Segmentation.

Demand Paging :
The process of loading the page into memory on demand (whenever page fault occurs) is known as demand paging.

The process includes the following steps :



Page Table

1. If CPU try to refer a page that is currently not available in the main memory, it generates an interrupt indicating memory access fault.
2. The OS puts the interrupted process in a blocking state. For the execution to proceed the OS must bring the required page into the memory.
3. The OS will search for the required page in the logical address space.
4. The required page will be brought from logical address space to physical address space. The page replacement algorithms are used for the decision making of replacing the page in physical address space.
5. The page table will updated accordingly.
6. The signal will be sent to the CPU to continue the program execution and it will place the process back into ready state.

Hence whenever a page fault occurs these steps are followed by the operating system and the required page is brought into memory.

Unit-5

Types of devices

The OS peripheral devices can be categorized into 3: Dedicated, Shared, and Virtual. The differences among them are the functions of the characteristics of the devices as well as how they are managed by the Device Manager.

Dedicated devices:-

Such type of devices in the **device management in operating system** are dedicated or assigned to only one job at a time until that job releases them. Devices like printers, tape drivers, plotters etc. demand such allocation scheme since it would be awkward if several users share them at the same point of time. The disadvantages of such kind f devices s the inefficiency resulting from the allocation of the device to a single user for the entire duration of job execution even though the device is not put to use 100% of the time.
Shared devices:-

These devices can be allocated o several processes. Disk-DASD can be shared among several processes at the same time by interleaving their requests. The interleaving is carefully controlled by the Device Manager and all issues must be resolved on the basis of predetermined policies.

Virtual Devices:-

These devices are the combination of the first two types and they are dedicated devices which are transformed into shared devices. For example, a printer converted into a shareable device via spooling program which re-routes all the print requests to a disk. A print job is not sent straight to the printer, instead, it goes to the disk(spool)until it is fully prepared with all the necessary sequences and formatting, then it goes to the printers. This technique can transform one printer into several virtual printers which leads to better performance and use.

Input/output device

Alternatively referred to as an **IO device**, an **input/output device** is any hardware used by a human operator or other systems to communicate with a computer. As the name suggests, input/output devices are capable of sending data (output) to a computer and receiving data from a computer (input).

Examples of input/output devices

- CD-R/RW, DVD, and Blu-ray drive
- Digital camera
- Floppy diskette drive

- Hard drives
- Modem
- Network adapter
- SD Card
- Touch screen
- USB thumb drives

Storage device

Alternatively referred to as **digital storage**, **storage**, **storage media**, or **storage medium**, a **storage device** is any hardware capable of holding information either temporarily or permanently. The picture shows an example of a Drobo, an external secondary storage device.There are two types of storage devices used with computers: a primary storage device, such as RAM, and a secondary storage device, such as a hard drive. Secondary storage can be removable, internal, or external.

Examples of computer storage

Magnetic storage devices

Today, magnetic storage is one of the most common types of storage used with computers. This technology is found mostly on extremely large HDDs or hybrid hard drives.

- Floppy diskette
- Hard drive
- Magnetic strip

- SuperDisk
- Tape cassette
- Zip diskette

Optical storage devices

Another common type of storage is optical storage, which uses lasers and lights as its method of reading and writing data.

- Blu-ray disc
- CD-ROM disc
- CD-R and CD-RW disc.
- DVD-R, DVD+R, DVD-RW, and DVD+RW disc.

Flash memory devices

Flash memory has replaced most magnetic and optical media as it becomes cheaper because it is the more efficient and reliable solution.

- USB flash drive, jump drive, or thumb drive.
- CF (CompactFlash)
- M.2
- Memory card
- MMC
- NVMe
- SDHC Card
- SmartMedia Card
- Sony Memory Stick
- SD card
- SSD
- xD-Picture Card

Online and cloud

Storing data online and in cloud storage is becoming popular as people need to access their data from more than one device.

- Cloud storage
- Network media

Paper storage

Early computers had no method of using any of the technologies above for storing information and had to rely on paper. Today, these forms of storage are rarely used or found. In the picture is an example of a woman entering data to a punch card using a punch card machine.

- OMR
- Punch card

Note

A hard copy is considered a form of paper storage, although it cannot be easily used to input data back into a computer without the aid of OCR.

Buffering

I/O is the process of transferring data between a program and an external device. The process of optimizing I/O consists primarily of making the best possible use of the slowest part of the path between the program and the device.

The slowest part is usually the physical channel, which is often slower than the CPU or a memory-to-memory data transfer. The time spent in I/O processing overhead can reduce the amount of time that a channel can be used, thereby reducing the effective transfer rate. The biggest factor in maximizing this channel speed is often the reduction of I/O processing overhead.

A *buffer* is a temporary storage location for data while the data is being transferred. A buffer is often used for the following purposes:

- Small I/O requests can be collected into a buffer, and the overhead of making many relatively expensive system calls can be greatly reduced.A collection buffer of this type can be sized and handled so that the actual physical I/O requests made to the operating system match the physical characteristics of the device being used. For example, a 42-sector buffer, when read or written, transfers a track of data between the buffer and the DD-49 disk; a track is a very efficient transfer size.
- Many data file structures, such as the f77 and cos file structures, contain control words. During the write process, a buffer can be used as a work area where control words can be inserted into the data stream (a process called *blocking*). The blocked data is then written to the device. During the read process, the same buffer work area can be used to examine and remove these control words before passing the data on to the user (*deblocking*).
- When data access is random, the same data may be requested many times. A *cache* is a buffer that keeps old requests in the buffer in case these requests are needed again. A cache that is sufficiently large and/or efficient can avoid a large part of the physical I/O by having the data ready in a buffer. When the data is often found in the cache buffer, it is referred to as having a high *hit rate*. For example, if the entire file fits in the cache and the file is present in the cache, no more physical requests are required to perform the I/O. In this case, the hit rate is 100%.
- Running the disks and the CPU in parallel often improves performance; therefore, it is useful to keep the CPU busy while data is being moved. To do this when writing, data can be transferred to the buffer at memory-to-memory copy speed and an asynchronous I/O request can be made. The control is then immediately returned to the program, which continues to execute as if the I/O were complete (a process called *write-behind*). A similar process can be used while reading; in this process, data is read into a buffer before the actual request is issued for it. When it is needed, it is already in the buffer and can be transferred to the user at very high speed. This is another form or use of a cache.

Buffers are used extensively on UNICOS and UNICOS/mk systems. Some of the disk controllers have built-in buffers. The kernel has a cache of buffers called the *system cache* that it uses for various I/O functions on a system-wide basis. The Cray IOS uses buffers to enhance I/O performance. The UNICOS logical device cache (ldcache) is a buffering scheme that uses a part of the solid-state storage device (SSD) or buffer memory resident (BMR) in the IOS as a large buffer that is associated with a particular file system. The library routines also use buffers.

The I/O path is divided into two parts. One part includes the user data area, the library buffer, and the system cache. The second part is referred to as the *logical device*, which includes the ultimate I/O device and all of the buffering, caching, and processing associated with that device. This includes any caching in the disk controller and the operating system.

Users can directly or indirectly control some buffers. These include most library buffers and, to some extent, system cache and

ldcache

. Some buffering, such as that performed in the IOS, or the disk controllers, is not under user control.

A *well-formed request* refers to I/O requests that meet the criteria for UNICOS systems; a well-formed request for a disk file requires the following:

- The size of the request must be a multiple of the sector size in bytes. For most disk devices, this will be 4096 bytes.
- The data that will be transferred must be located on a word boundary.

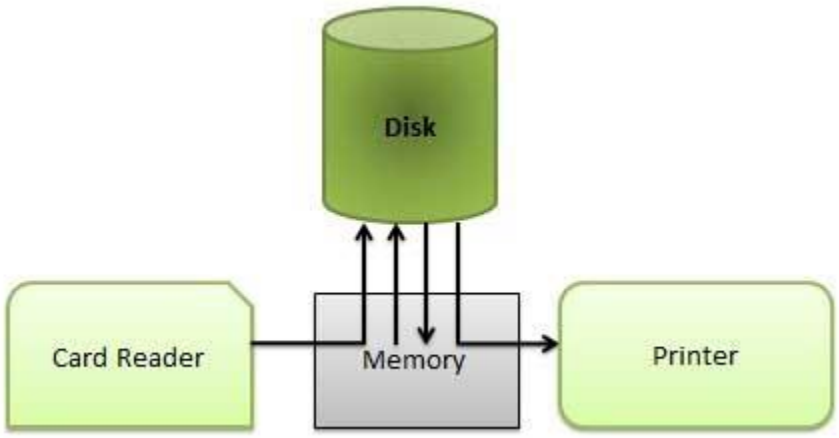
The file must be positioned on a sector boundary. This will be a 4096-byte sector boundary for most disks.

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.