OPERATING SYSTEM

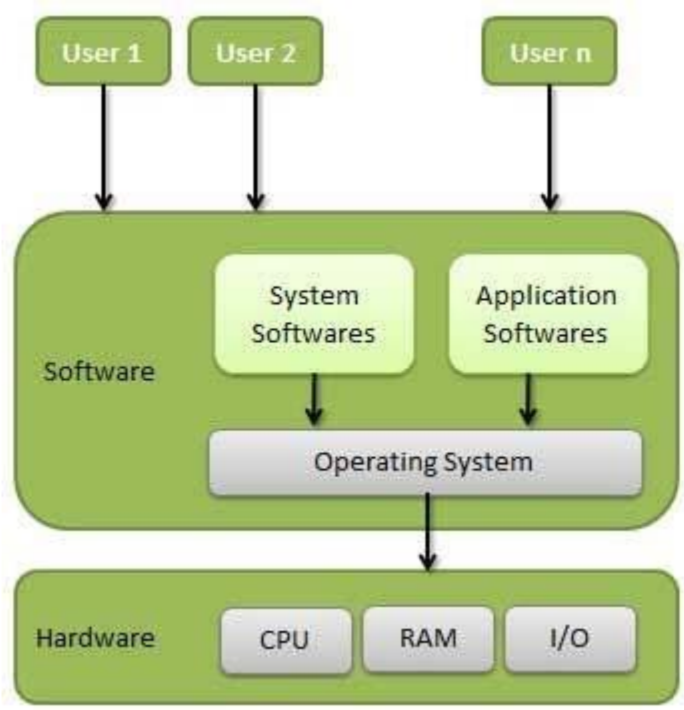
Overview of operating System

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.

1. *Definition of operating system*

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



Following are some of important functions of an operating System.

- Memory Management
- Processor Management
- Device Management
- File Management

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

2.Types of Operating system

Types of operating systems which are most commonly used are as follows .

2.1.Batch operating system

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

The problems with Batch Systems are as follows −

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

2.2Time-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.

2.3.Distributed operating System

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

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as **loosely coupled systems** or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers, and so on.

The advantages of distributed systems are as follows −

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

2.4.Network operating System

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

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

The advantages of network operating systems are as follows −

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

The disadvantages of network operating systems are as follows −

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

2.5.Real Time operating System

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

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

There are two types of real-time operating systems.

2.5.1.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.

2.5.2.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.

3.Operating system Services

An Operating System provides services to both the users and to the programs.

- It provides programs an environment to execute.
- It provides users the services to execute the programs in a convenient manner.

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

3.1.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.

3.2.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.

3.3.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.

3.4.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.

3.5.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.

3.6.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.

3.7.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.

4.User Operating System Interface

4.1.User Operating System Interface -CLI

Command Line Interface (CLI) or command interpreter allows direct command entry sometimes implemented in kernel, sometimes by systems program sometimes multiple flavors implemented –shells primarily fetches a command from user and executes it. Sometimes commands built-in, sometimes just names of programs

4.2.User Operating System Interface -GUI

User-friendly desktop metaphor interface usually mouse, keyboard, and monitor icons represent files, programs, actions, etc various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a folder) invented at Xerox PARC

Many systems now include both CLI and GUI interfaces

Microsoft Windows is GUI with CLI ―command‖ shell

Apple Mac OS X as ―Aqua‖ GUI interface with UNIX kernel underneath and shells available

Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)

5.System Calls

In computing, a **system call** is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to **interact with the operating system**. A computer program makes a system call when it makes a request to the operating system's kernel. System call **provides** the services of the operating system to the user programs via Application Program Interface(API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system. System calls are the only entry points into the kernel system. All programs needing resources must use system calls.
Services Provided by System Calls :

1. Process creation and management
2. Main memory management
3. File Access, Directory and File system management
4. Device handling(I/O)
5. Protection
6. Networking, etc.

5.1.Types of System Calls :

There are 5 different categories of system calls –

1. Process control: end, abort, create, terminate, allocate and free memory.
2. File management**:** create, open, close, delete, read file etc.
3. Device management
4. Information maintenance
5. Communication

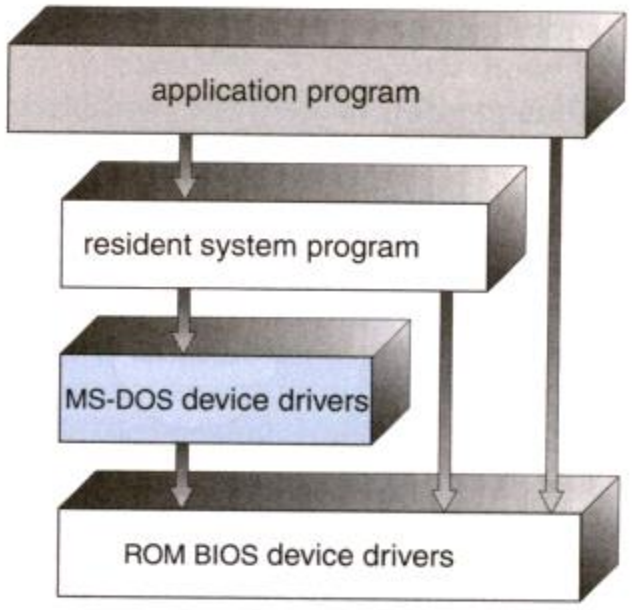
Examples of Windows and Unix System Calls –

	WINDOWS	**UNIX**
	CreateProcess()	fork()
	ExitProcess()	exit()
Process Control	WaitForSingleObject()	wait()
	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
File Manipulation	CloseHandle()	close()
	SetConsoleMode()	ioctl()
	ReadConsole()	read()
Device Manipulation	WriteConsole()	write()
	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
Information Maintenance	Sleep()	sleep()
	CreatePipe()	pipe()
	CreateFileMapping()	shmget()
Communication	MapViewOfFile()	mmap()

	SetFileSecurity()
Protection	InitlializeSecurityDescriptor()

6.Operating System structure

6.1Simple Structure

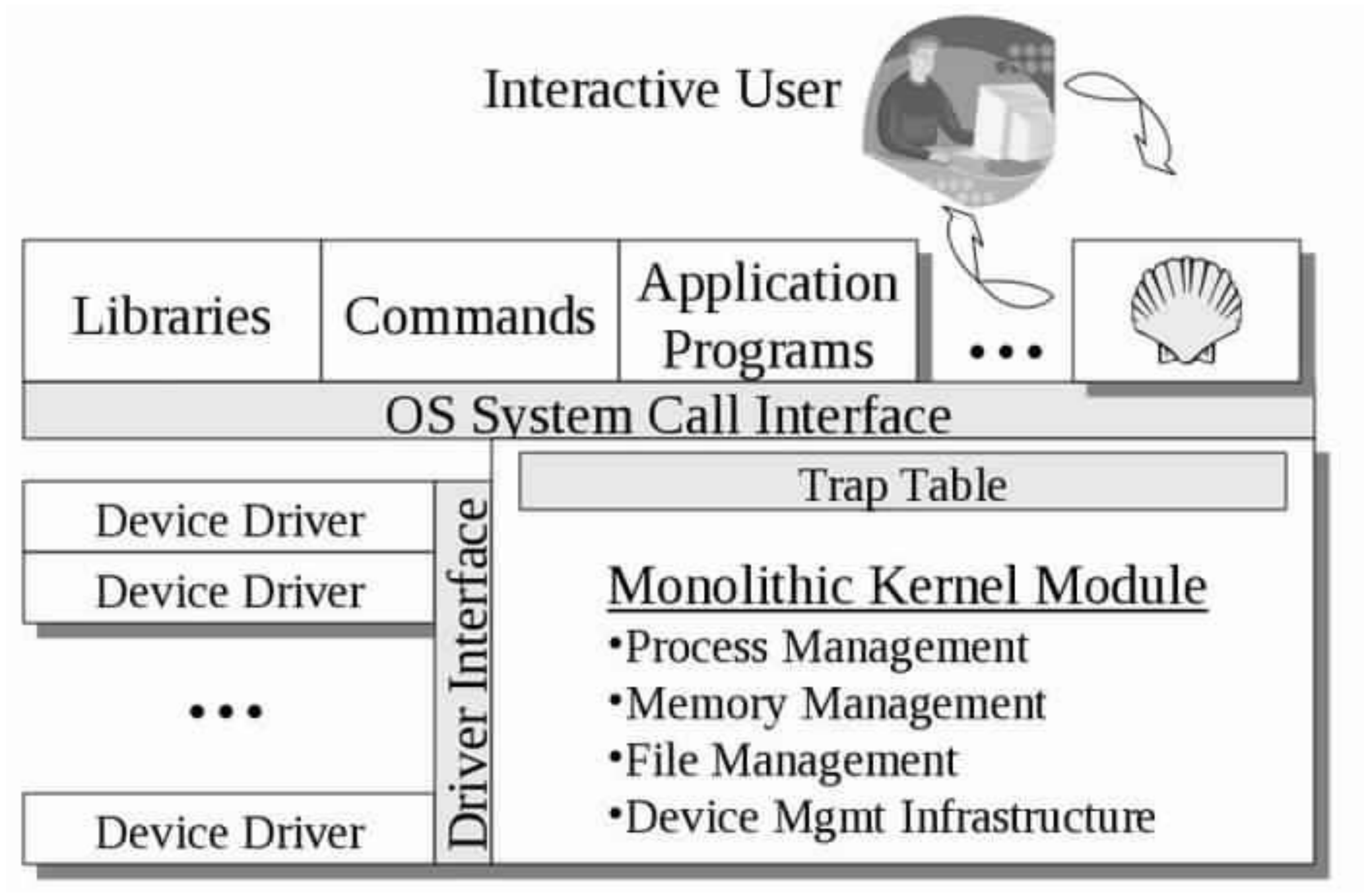


In MS-DOS, applications may bypass the operating system.

- Operating systems such as MS-DOS and the original UNIX did not have well-defined structures.
- There was no CPU Execution Mode (user and kernel), and so errors in applications could cause the whole system to crash.

6.2Monolithic approach

- Functionality of the OS is invoked with simple function calls within the kernel, which is one large program.
- Device drivers are loaded into the running kernel and become part of the kernel.

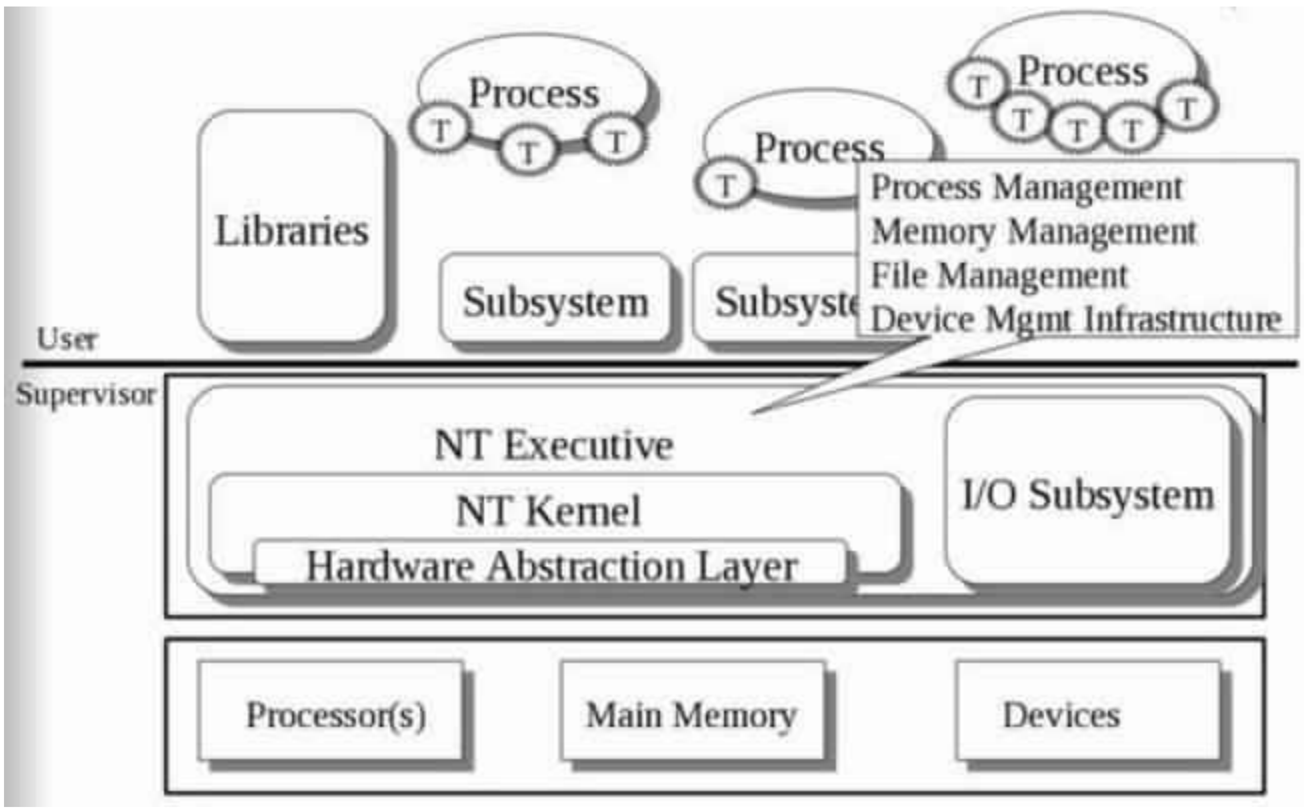


A monolithic kernel, such as Linux and other Unix systems.

6.3.Layered Approach

This approach breaks up the operating system into different layers.

- This allows implementers to change the inner workings, and increases modularity.
- As long as the external interface of the routines don't change, developers have more freedom to change the inner workings of the routines.
- With the layered approach, the bottom layer is the hardware, while the highest layer is the user interface.
    - The main *advantage* is simplicity of construction and debugging.
    - The main *difficulty* is defining the various layers.
    - The main *disadvantage* is that the OS tends to be less efficient than other implementations.



The Microsoft Windows NT Operating System. The lowest level is a monolithic kernel, but many OS components are at a higher level, but still part of the OS.

7.Virtual machine

A virtual machine (VM) is a software program or operating system that not only exhibits the behavior of a separate computer, but is also capable of performing tasks such as running applications and programs like a separate computer. A virtual machine, usually known as a guest is created within another computing environment referred as a "host." Multiple virtual machines can exist within a single host at one time.

A virtual machine is also known as a guest.

Virtual machines are becoming more common with the evolution of virtualization technology. Virtual machines are often created to perform certain tasks that are different than tasks performed in a host environment.

Virtual machines are implemented by software emulation methods or hardware virtualization techniques. Depending on their use and level of correspondence to any physical computer, virtual machines can be divided into two categories:

7.1.System Virtual Machines: A system platform that supports the sharing of the host computer's physical resources between multiple virtual machines, each running with its own copy of the operating system. The virtualization technique is provided by a software layer known as a hypervisor, which can run either on bare hardware or on top of an operating system.

7.2.Process Virtual Machine: Designed to provide a platform-independent programming environment that masks the information of the underlying hardware or operating system and allows program execution to take place in the same way on any given platform.

Some of the advantages of a virtual machine include:

- Allows multiple operating system environments on a single physical computer without any intervention
- Virtual machines are widely available and are easy to manage and maintain.
- Offers application provisioning and disaster recovery options

Some of the drawbacks of virtual machines include:

- They are not as efficient as a physical computer because the hardware resources are distributed in an indirect way.
- Multiple VMs running on a single physical machine can deliver unstable performance

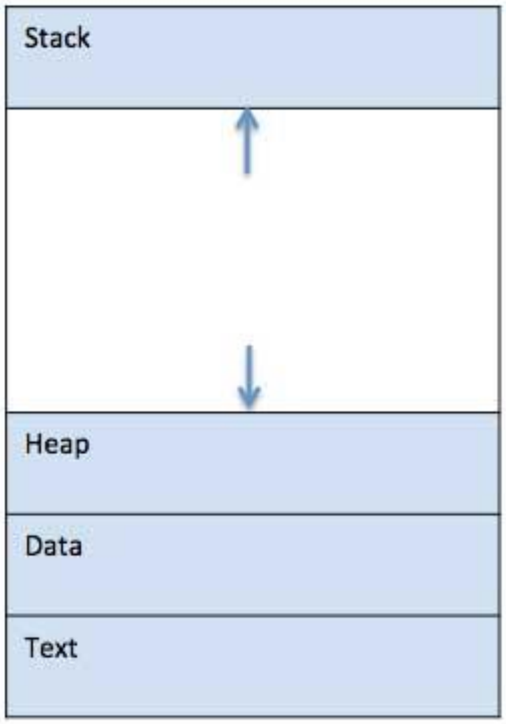
Process Management

1.Process

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.A process is defined as an entity which represents the basic unit of work to be implemented in the system.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory −



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

2.Program

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

```
#include <stdio.h>


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

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

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

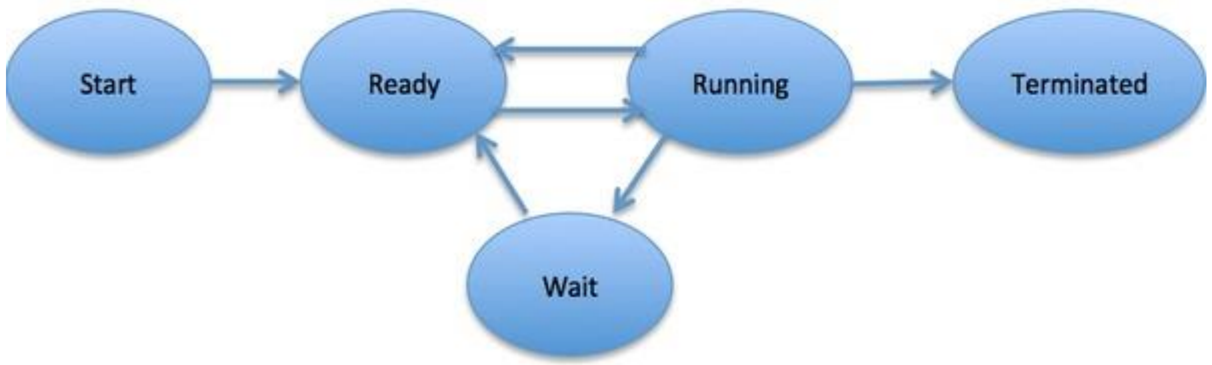
3.Process Life Cycle

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

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

S.N.	State & Description
1	**Start** This is the initial state when a process is first started/created.
2	**Ready** The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process.

3	**Running** Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.
4	**Waiting** Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
5	**Terminated or Exit** Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.



4.Process Control Block (PCB)

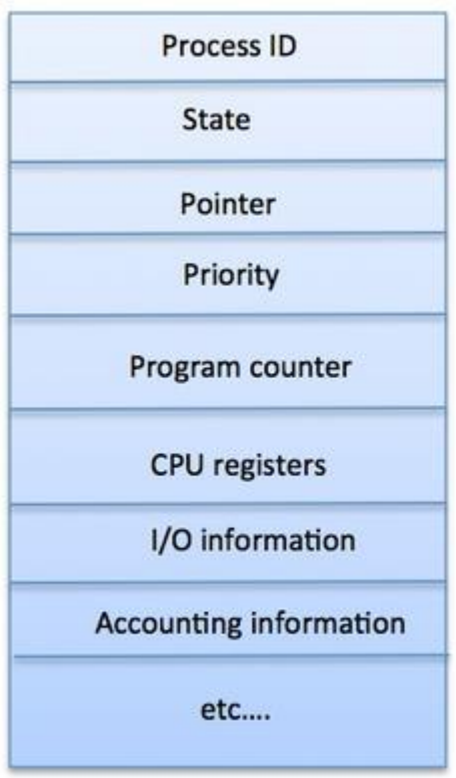
A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table −

S.N.	Information & Description
1	**Process State** The current state of the process i.e., whether it is ready, running, waiting, or whatever.

2	**Process privileges** This is required to allow/disallow access to system resources.
3	**Process ID** Unique identification for each of the process in the operating system.
4	**Pointer** A pointer to parent process.
5	**Program Counter** Program Counter is a pointer to the address of the next instruction to be executed for this process.
6	**CPU registers** Various CPU registers where process need to be stored for execution for running state.
7	**CPU Scheduling Information** Process priority and other scheduling information which is required to schedule the process.
8	**Memory management information** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9	**Accounting information** This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	**IO status information**

	This includes a list of I/O devices allocated to the process.

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



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

5.Messaging Passing

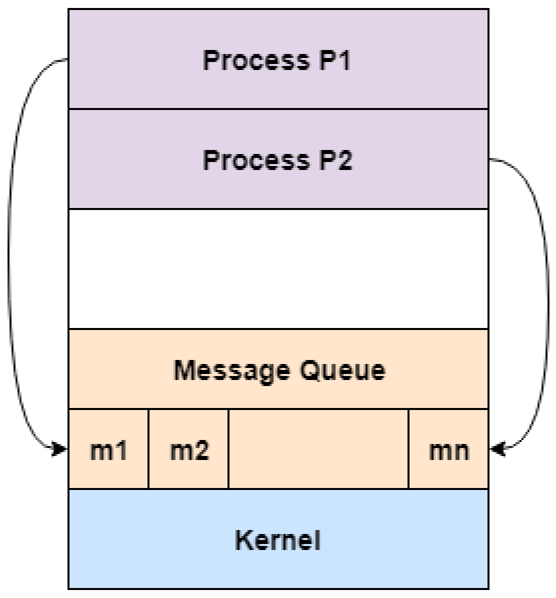
The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

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

Process communication is the mechanism provided by the operating system that allows processes to communicate with each other. This communication could involve a process letting another process know that some event has occurred or transferring of data from one process to another. One of the models of process communication is the message passing model.

Message passing model allows multiple processes to read and write data to the message queue without being connected to each other. Messages are stored on the queue until their recipient retrieves them. Message queues are quite useful for interprocess communication and are used by most operating systems.

A diagram that demonstrates message passing model of process communication is given as follows:



Message Passing Model

In the above diagram, both the processes P1 and P2 can access the message queue and store and retrieve data.

5.1.Advantages of message passing

Some of the advantages of message passing model are given as follows:

1. The message passing model is much easier to implement than the shared memory model.

2. It is easier to build parallel hardware using message passing model as it is quite tolerant of higher communication latencies.

5.2.Disadvantages of messaging passing model

The message passing model has slower communication than the shared memory model because the connection setup takes time.

6.CPU Scheduling

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

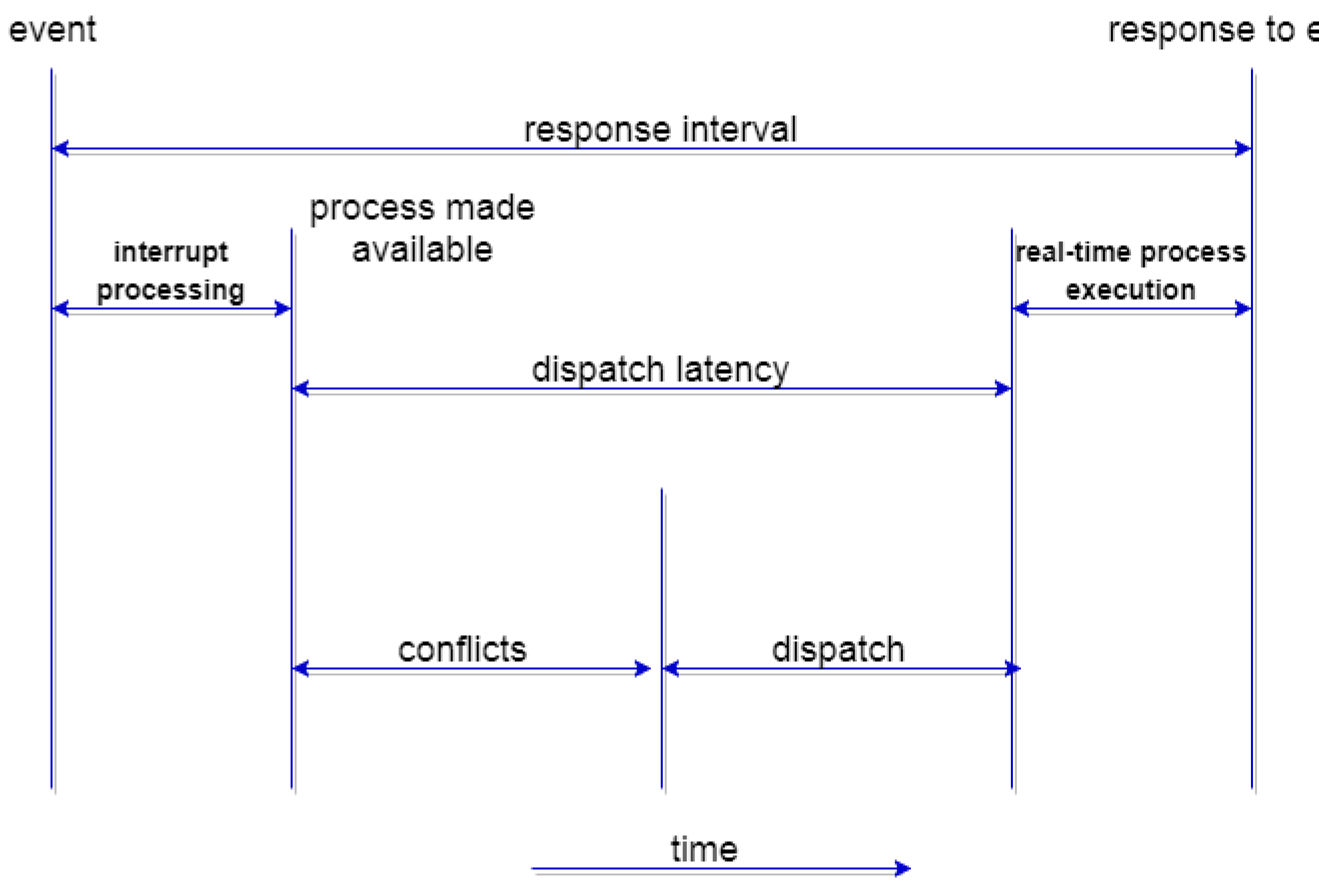
Whenever the CPU becomes idle, the operating system must select one of the processes in the **ready queue** to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

6.1.CPU Scheduling: Dispatcher

Another component involved in the CPU scheduling function is the **Dispatcher**. The dispatcher is the module that gives control of the CPU to the process selected by the **short-term scheduler**. This function involves:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program from where it left last time.

The dispatcher should be as fast as possible, given that it is invoked during every process switch. The time taken by the dispatcher to stop one process and start another process is known as the **Dispatch Latency**. Dispatch Latency can be explained using the below figure:



6.2.Types of CPU Scheduling

CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the **running** state to the **waiting** state(for I/O request or invocation of wait for the termination of one of the child processes).
2. When a process switches from the **running** state to the **ready** state (for example, when an interrupt occurs).
3. When a process switches from the **waiting** state to the **ready** state(for example, completion of I/O).
4. When a process **terminates**.

In circumstances 1 and 4, there is no choice in terms of scheduling. A new process(if one exists in the ready queue) must be selected for execution. There is a choice, however in circumstances 2 and 3.

When Scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is **non-preemptive**; otherwise the scheduling scheme is **preemptive**.

6.2.1.Non-Preemptive Scheduling

Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

This scheduling method is used by the Microsoft Windows 3.1 and by the Apple Macintosh operating systems.

It is the only method that can be used on certain hardware platforms, because It does not require the special hardware(for example: a timer) needed for preemptive scheduling.

6.2.2.Preemptive Scheduling

In this type of Scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

7.CPU Scheduling: Scheduling Criteria

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

7.1.CPU Utilization

To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

7.2.Throughput

It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

7.3.Turnaround Time

*I*t is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process(Wall clock time).

7.4.Waiting Time

The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

7.5.Load Average

It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

7.6.Response Time

Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).

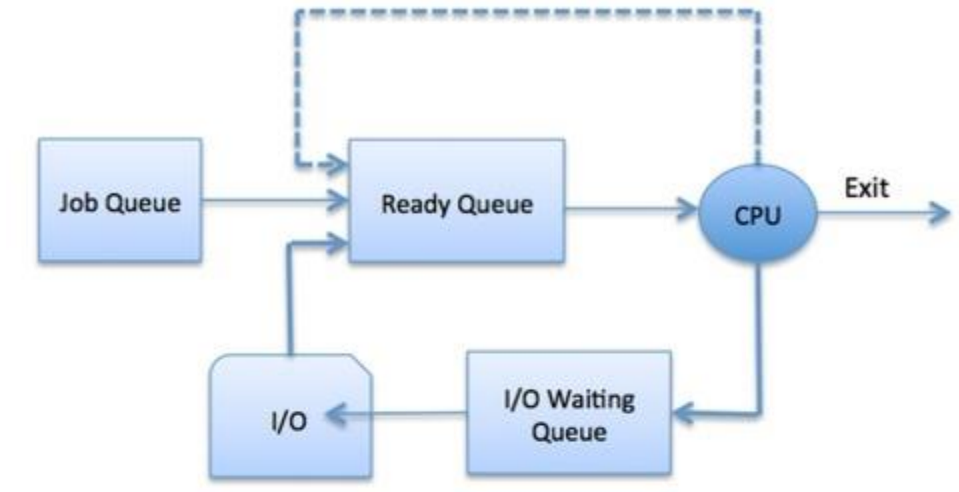
In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

8.Process Scheduling Queues

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

The Operating System maintains the following important process scheduling queues −

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



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

8.1.Two-State Process Model

Two-state process model refers to running and non-running states which are described below −

S.N.	State & Description
1	**Running** When a new process is created, it enters into the system as in the running state.
2	**Not Running** Processes that are not running are kept in queue, waiting for their turn to execute. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list. Use of dispatcher is as follows. When a process is interrupted, that process is transferred in the waiting queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then selects a process from the queue to execute.

9.Schedulers

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

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

9.1.Long Term Scheduler

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

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

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

9.2.Short Term Scheduler

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

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

9.3.Medium Term Scheduler

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

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

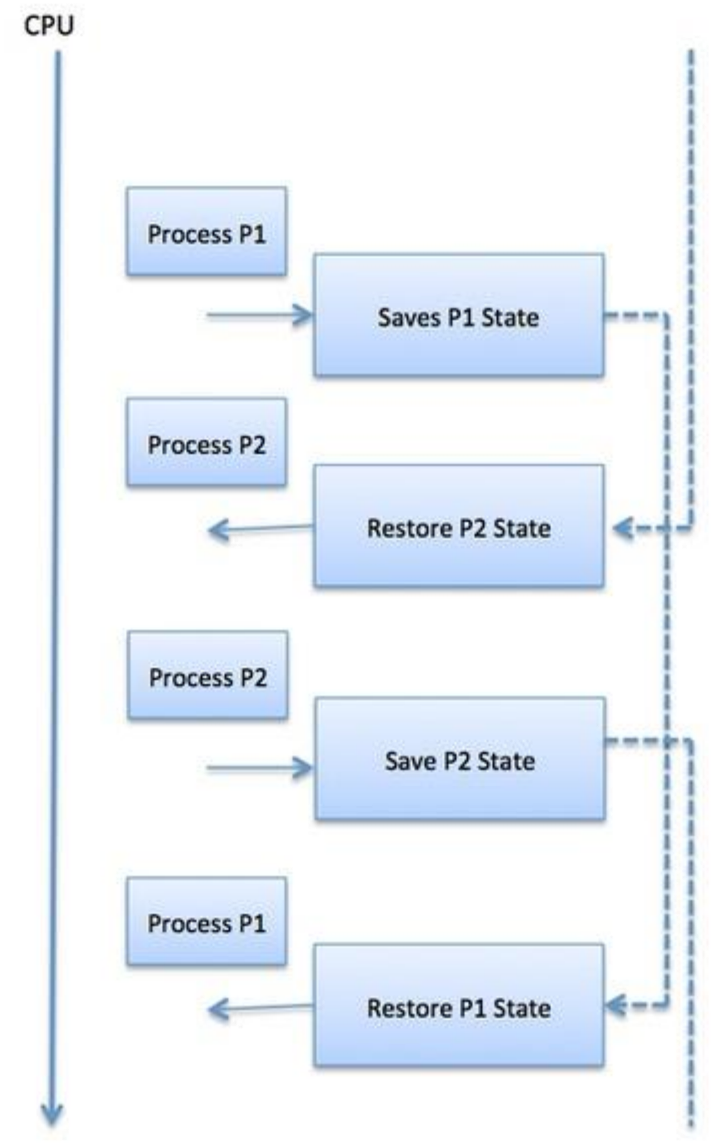
Comparison among Scheduler

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

10.Context Switching

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. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

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



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

- Program Counter
- Scheduling information
- Base and limit register value
- Currently used register
- Changed State

- I/O State information
- Accounting information

10.Interprocess Communication

Interprocess communication (IPC) is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an operating system. This allows a program to handle many user requests at the same time. Since even a single user request may result in multiple processes running in the operating system on the user's behalf, the processes need to communicate with each other. The IPC interfaces make this possible. Each IPC method has its own advantages and limitations so it is not unusual for a single program to use all of the IPC methods.

11.Shared Memory System

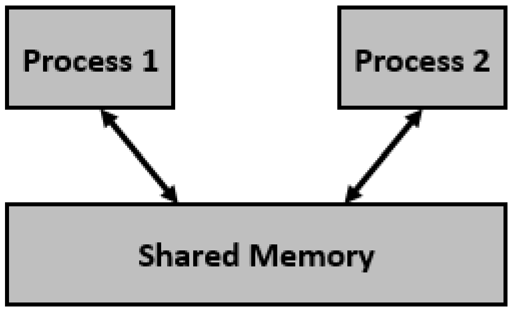
Shared memory is a memory shared between two or more processes. However, why do we need to share memory or some other means of communication?

To reiterate, each process has its own address space, if any process wants to communicate with some information from its own address space to other processes, then it is only possible with IPC (inter process communication) techniques. As we are already aware, communication can be between related or unrelated processes.

Usually, inter-related process communication is performed using Pipes or Named Pipes. Unrelated processes (say one process running in one terminal and another process in another terminal) communication can be performed using Named Pipes or through popular IPC techniques of Shared Memory and Message Queues.

We have seen the IPC techniques of Pipes and Named pipes and now it is time to know the remaining IPC techniques viz., Shared Memory, Message Queues, Semaphores, Signals, and Memory Mapping.

In this chapter, we will know all about shared memory.



We know that to communicate between two or more processes, we use shared memory but before using the shared memory what needs to be done with the system calls, let us see this −

- Create the shared memory segment or use an already created shared memory segment (shmget())
- Attach the process to the already created shared memory segment (shmat())
- Detach the process from the already attached shared memory segment (shmdt())
- Control operations on the shared memory segment (shmctl())

12.Scheduling Algorithm

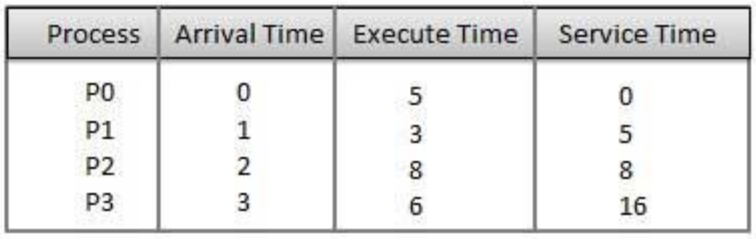
A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which we are going to discuss in this chapter −

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

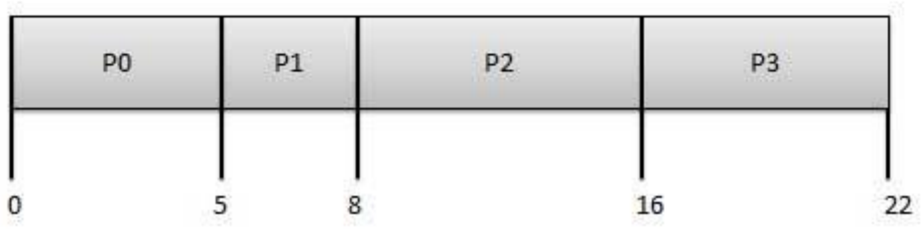
These algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

12.1.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



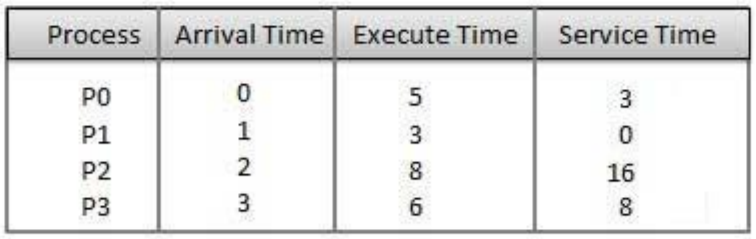
Wait time of each process is as follows −

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

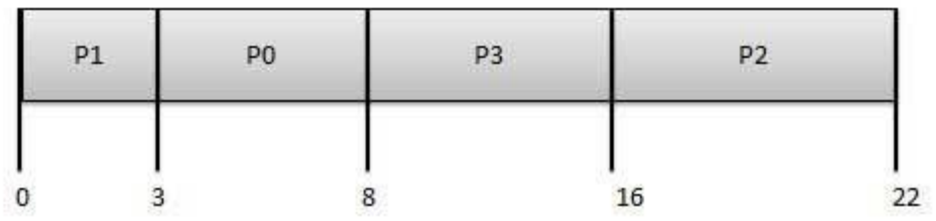
Average Wait Time: (0+4+6+13) / 4 = 5.75

12.2.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.



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



Wait time of each process is as follows −

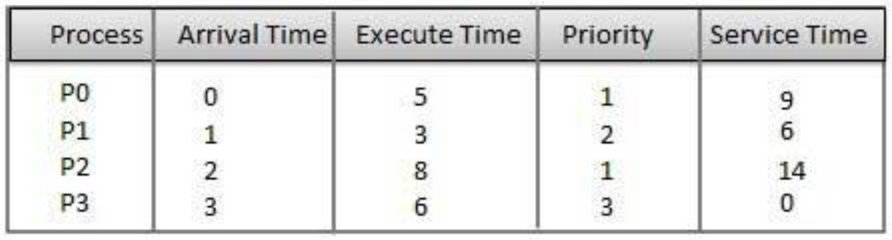
Process	Wait Time : Service Time - Arrival Time
P0	3 - 0 = 3
P1	0 - 0 = 0
P2	16 - 2 = 14
P3	8 - 3 = 5

Average Wait Time: (3+0+14+5) / 4 = 5.50

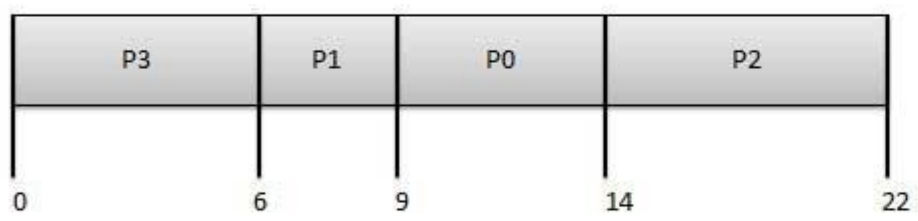
12.3.Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.

- Priority can be decided based on memory requirements, time requirements or any other resource requirement.



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



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	9 - 0 = 9
P1	6 - 1 = 5
P2	14 - 2 = 12
P3	0 - 0 = 0

Average Wait Time: (9+5+12+0) / 4 = 6.5

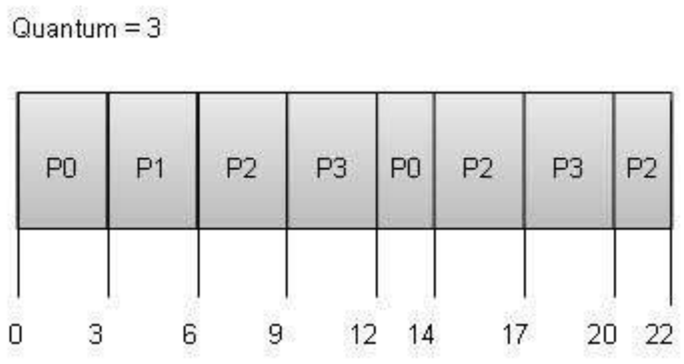
12.4.Shortest Remaining Time

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.

- It is often used in batch environments where short jobs need to give preference.

12.5.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

12.6.Multiple-Level Queues Scheduling

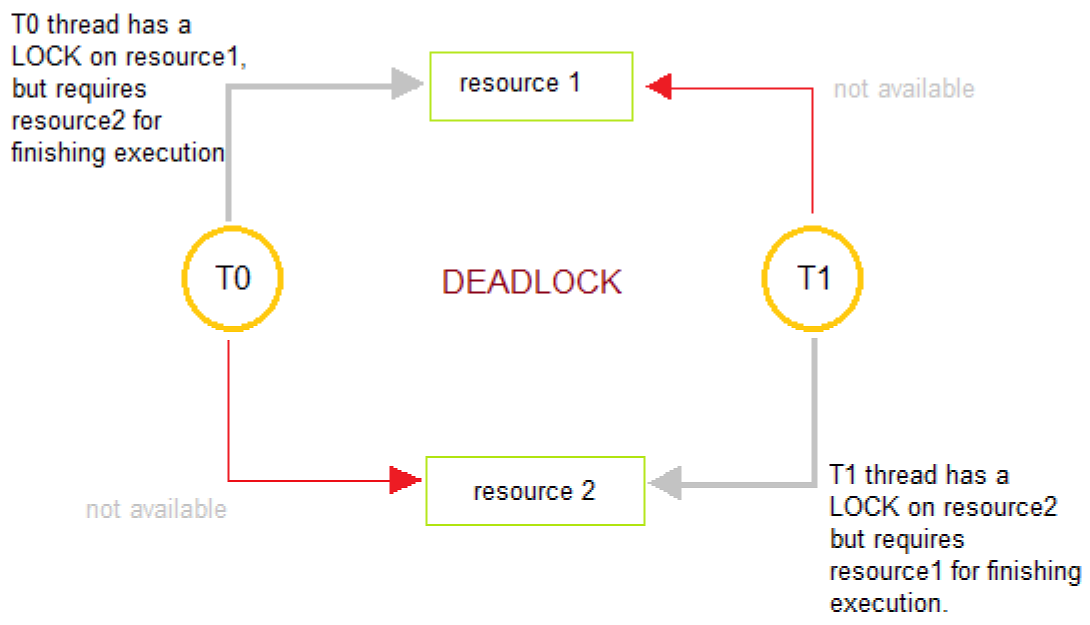
Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

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

For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

Deadlock

Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.


T0 thread has a LOCK on resource1, but requires resource2 for finishing execution
resource 1
not available
T0
DEADLOCK
T1
not available
resource 2
T1 thread has a LOCK on resource2 but requires resource1 for finishing execution.

1.Condition for deadlock

A deadlock situation on a resource can arise if and only if all of the following conditions hold simultaneously in a system

> *1.1.Mutual exclusion:* At least one resource must be held in a non-shareable mode. Otherwise, the processes would not be prevented from using the resource when necessary. Only one process can use the resource at any given instant of time.

1.2.Hold and wait or *resource holding:* a process is currently holding at least one resource and requesting additional resources which are being held by other processes.

1.3.No preemption: a resource can be released only voluntarily by the process holding it.

1.4.Circular wait: each process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource. In general, there is a set of waiting processes, $P = \{P_1, P_2, \ldots, P_N\}$, such that P_1 is waiting for a resource held by P_2, P_2 is waiting for a resource held by P_3 and so on until P_N is waiting for a resource held by P_1.

2.Deadlock avoidance

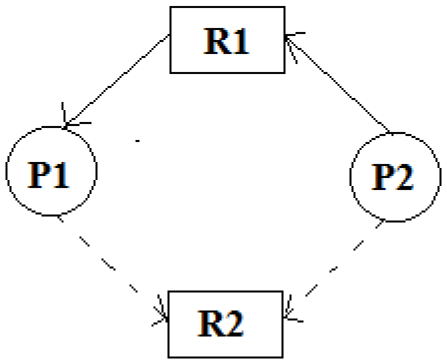
As you saw already, most prevention algorithms have poor resource utilization, and hence result in reduced throughputs. Instead, we can try to avoid deadlocks by making use prior knowledge about the usage of resources by processes including resources available, resources allocated, future requests and future releases by processes. Most deadlock avoidance algorithms need every process to tell in advance the maximum number of

resources of each type that it may need. Based on all these info we may decide if a process should wait for a resource or not, and thus avoid chances for circular wait.

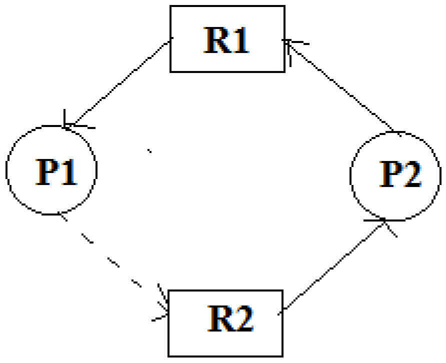
If a system is already in a safe state, we can try to stay away from an unsafe state and avoid deadlock. Deadlocks cannot be avoided in an unsafe state. A system can be considered to be in safe state if it is not in a state of deadlock and can allocate resources upto the maximum available. A safe sequence of processes and allocation of resources ensures a safe state. Deadlock avoidance algorithms try not to allocate resources to a process if it will make the system in an unsafe state. Since resource allocation is not done right away in some cases, deadlock avoidance algorithms also suffer from low resource utilization problem.

A resource allocation graph is generally used to avoid deadlocks. If there are no cycles in the resource allocation graph, then there are no deadlocks. If there are cycles, there may be a deadlock. If there is only one instance of every resource, then a cycle implies a deadlock. Vertices of the resource allocation graph are resources and processes. The resource allocation graph has request edges and assignment edges. An edge from a process to resource is a request edge and an edge from a resource to process is an allocation edge. A calm edge denotes that a request may be made in future and is represented as a dashed line. Based on calm edges we can see if there is a chance for a cycle and then grant requests if the system will again be in a safe state.

Consider the image with calm edges as below:



If R2 is allocated to p2 and if P1 request for R2, there will be a deadlock.



The resource allocation graph is not much useful if there are multiple instances for a resource. In such a case, we can use Banker's algorithm. In this algorithm, every process must tell upfront the maximum resource of each type it need, subject to the maximum available instances for each type. Allocation of resources is made only, if the allocation ensures a safe state; else the processes need to wait. The Banker's algorithm can be divided into two parts: Safety algorithm if a system is in a safe state or not. The resource request algorithm make an assumption of allocation and see if the system will be in a safe state. If the new state is unsafe, the resources are not allocated and the data structures are restored to their previous state; in this case the processes must wait for the resource.

3.How to avoid Deadlocks

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

3.1.Mutual Exclusion

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

3.2.Hold and Wait

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

3.3.No Preemption

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

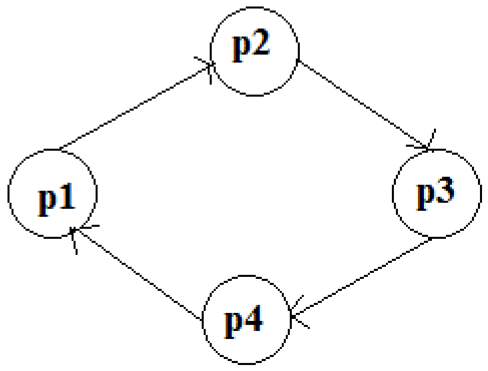
3.4.Circular Wait

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

4.Deadlock Detection

If deadlock prevention and avoidance are not done properly, as deadlock may occur and only things left to do is to detect the recover from the deadlock.

If all resource types has only single instance, then we can use a graph called wait-for-graph, which is a variant of resource allocation graph. Here, vertices represent processes and a directed edge from P1 to P2 indicate that P1 is waiting for a resource held by P2. Like in the case of resource allocation graph, a cycle in a wait-for-graph indicate a deadlock. So the system can maintain a wait-for-graph and check for cycles periodically to detect any deadlocks.



The wait-for-graph is not much useful if there are multiple instances for a resource, as a cycle may not imply a deadlock. In such a case, we can use an algorithm similar to Banker's algorithm to detect deadlock. We can see if further allocations can be made on not based on current allocations.

Memory management Function

1.Logical 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.

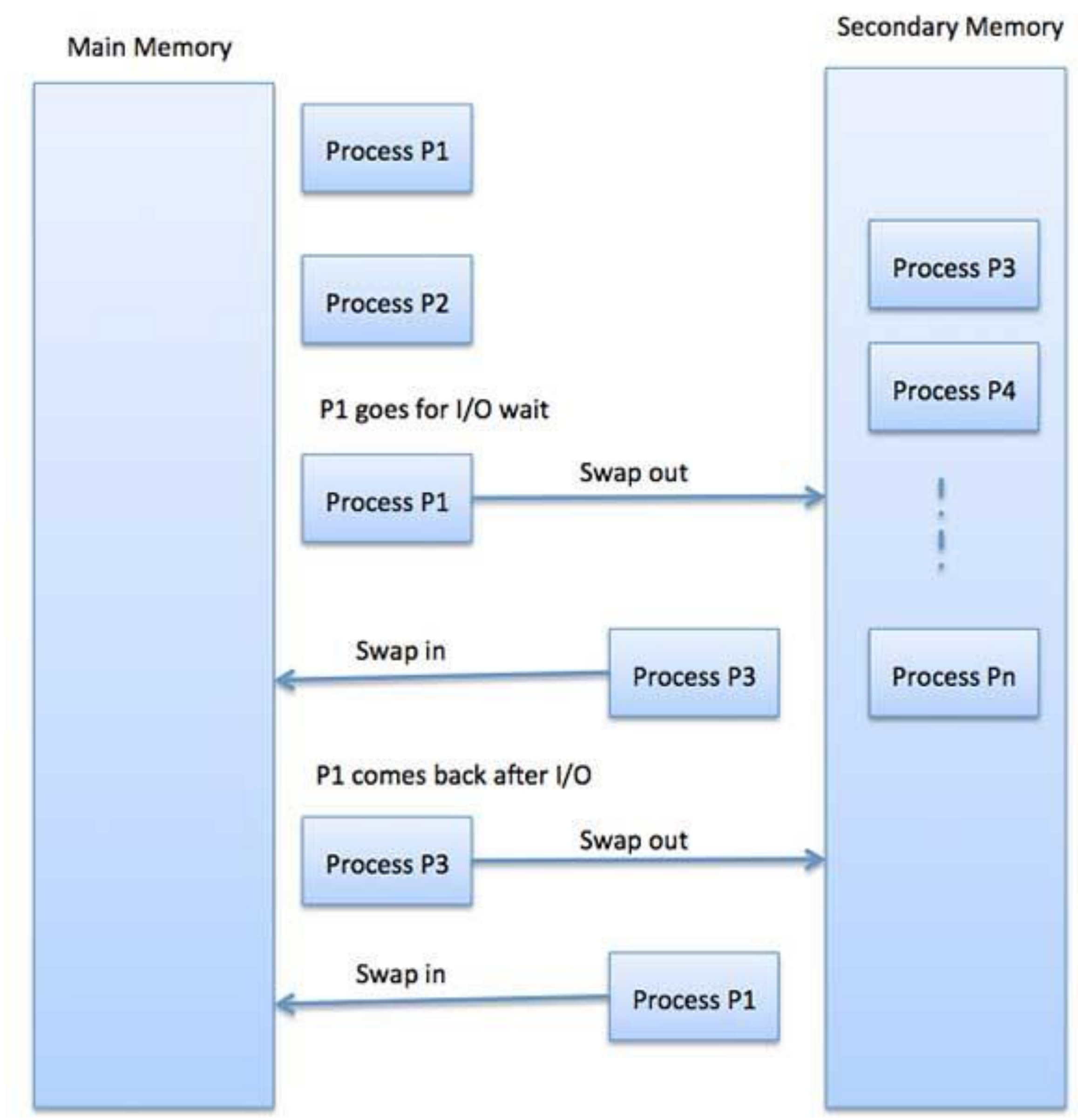
2.Physical Address

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.

3.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.



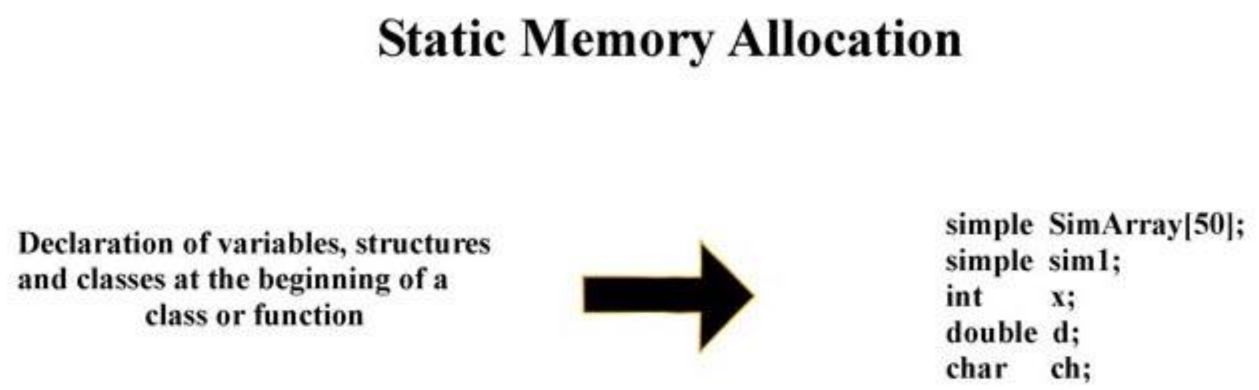
The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

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

3.Memory Allocation

Memory allocation is the process of setting aside sections of memory in a program to be used to store variables, and instances of structures and classes. There are two basic types of memory allocation:

3.1.Static Memory Allocation

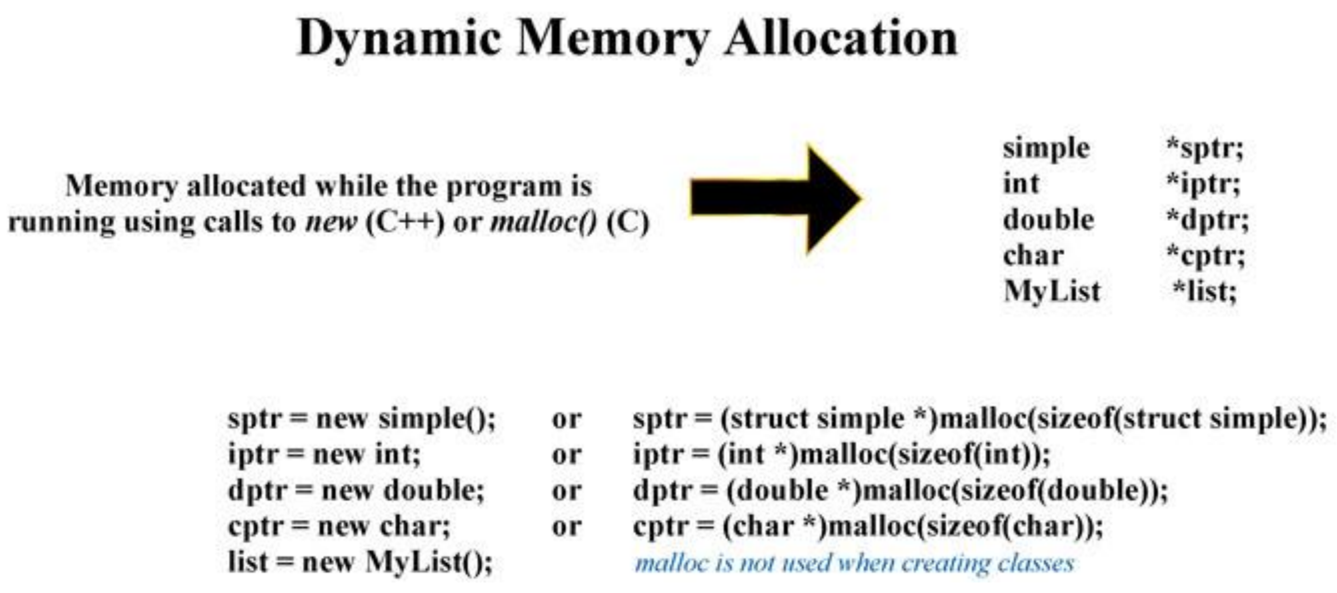


Static Memory Allocation

Declaration of variables, structures and classes at the beginning of a class or function

```
simple  SimArray[50];
simple  sim1;
int     x;
double  d;
char    ch;
```

When you declare a variable or an instance of a structure or class. The memory for that object is allocated by the operating system. The name you declare for the object can then be used to access that block of memory.
3.2.Dynamic Memory Allocation



Dynamic Memory Allocation

Memory allocated while the program is running using calls to *new* (C++) or *malloc()* (C)

```
simple     *sptr;
int        *iptr;
double     *dptr;
char       *cptr;
MyList     *list;
```

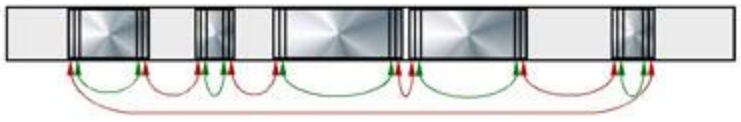
```
sptr = new simple();    or    sptr = (struct simple *)malloc(sizeof(struct simple));
iptr = new int;         or    iptr = (int *)malloc(sizeof(int));
dptr = new double;      or    dptr = (double *)malloc(sizeof(double));
cptr = new char;        or    cptr = (char *)malloc(sizeof(char));
list = new MyList();          malloc is not used when creating classes
```

When you use dynamic memory allocation you have the operating system designate a block of memory of the appropriate size while the program is running. This is done either with the new operator or with a call to the malloc function. The block of memory is allocated and a pointer to the block is returned. This is then stored in a pointer to the appropriate data type.

3.3.The Heap

The *Heap* is that portion of computer memory, allocated to a running application, where memory can be allocated for variables, class instances, etc. From a program's heap the OS allocates memory for dynamic use. Given a pointer to any one of the allocated blocks the OS can search in either direction to locate a block big enough to fill a dynamic memory allocation request.

Blocks of memory allocated on the heap are actually a special type of data structure consisting of (1) A pointer to the end of the previous block, (2) a pointer to the end of this block, (3) the allocated block of memory which can vary in size depending on its use, (4) a pointer to the beginning of this block, and (5) a pointer to the next block.



The Heap Layout.

Blank spaces between allocated blocks are where previously used blocks have been deallocated.

Depending on the type of operating system there are two possible algorithms that can be implemented in order to locate a block of memory in the heap to allocate:

- First-Fit - The needed amount of memory is allocated in the first block located in the heap that is big enough. This is faster but can result in greater fragmentation* of the heap.
- Best-Fit - All of the heap is searched and the needed amount of memory is allocated in the block where there is the least amount of memory left over. Slower but can result in less fragmentation*.

3.3.1.Heap Fragmentation

It is the condition that results when there is a lot of memory allocation and deallocation taking place in a program while it is running. With either memory allocation algorithm method a point may be reached where an attempt to allocate additional memory may fail. There may be enough total memory available in the heap, but it is just not all in one place. Some operating systems (Mac OS most noteably) have a special algorithm that reduces heap fragmentation through a process known as compacting the heap. That is allocated blocks in the heap are moved together and pointers to these blocks are changed to match the new locations. This can be done automatically without the programmer having to provide any special coding and completely invisible to the user.

3.3.2.Segementation

In Operating Systems, Segmentation is a memory management technique in which, the memory is divided into the variable size parts. Each part is known as segment which can be allocated to a process.

The details about each segment are stored in a table called as segment table. Segment table is stored in one (or many) of the segments.

Segment table contains mainly two information about segment:

1. Base: It is the base address of the segment
2. Limit: It is the length of the segment.

Why Segmentation is required?

Till now, we were using Paging as our main memory management technique. Paging is more close to Operating system rather than the User. It divides all the process into the form of pages regardless of the fact that a process can have some relative parts of functions which needs to be loaded in the same page.

Operating system doesn't care about the User's view of the process. It may divide the same function into different pages and those pages may or may not be loaded at the same time into the memory. It decreases the efficiency of the system.

It is better to have segmentation which divides the process into the segments. Each segment contain same type of functions such as main function can be included in one segment and the library functions can be included in the other segment,

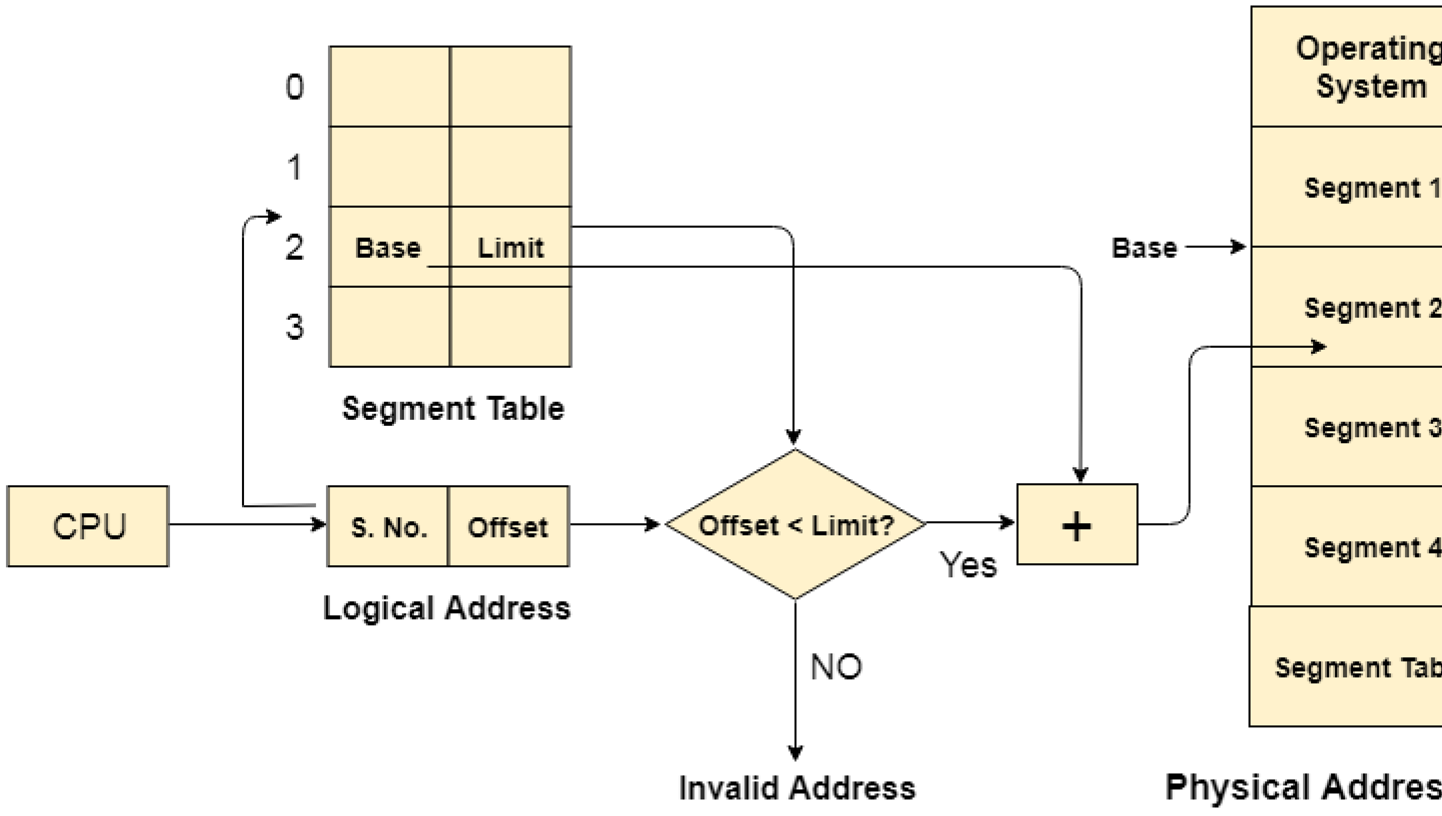
Translation of Logical address into physical address by segment table

CPU generates a logical address which contains two parts:

1. Segment Number
2. Offset

The Segment number is mapped to the segment table. The limit of the respective segment is compared with the offset. If the offset is less than the limit then the address is valid otherwise it throws an error as the address is invalid.

In the case of valid address, the base address of the segment is added to the offset to get the physical address of actual word in the main memory.



4.Advantages of Segmentation

1. No internal fragmentation
2. Average Segment Size is larger than the actual page size.
3. Less overhead
4. It is easier to relocate segments than entire address space.
5. The segment table is of lesser size as compare to the page table in paging.

5.Disadvantages

1. It can have external fragmentation.
2. it is difficult to allocate contiguous memory to variable sized partition.
3. Costly memory management algorithms.

6.Virtual Memory

Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory.

In this scheme, User can load the bigger size processes than the available main memory by having the illusion that the memory is available to load the process.

Instead of loading one big process in the main memory, the Operating System loads the different parts of more than one process in the main memory.

By doing this, the degree of multiprogramming will be increased and therefore, the CPU utilization will also be increased.

How Virtual Memory Works?

In modern word, virtual memory has become quite common these days. In this scheme, whenever some pages needs to be loaded in the main memory for the execution and the memory is not available for those many pages, then in that case, instead of stopping the pages from entering in the main memory, the OS search for the RAM area that are least used in the recent times or that are not referenced and copy that into the secondary memory to make the space for the new pages in the main memory.

Since all this procedure happens automatically, therefore it makes the computer feel like it is having the unlimited RAM.

7.Demand Paging

Demand Paging is a popular method of virtual memory management. In demand paging, the pages of a process which are least used, get stored in the secondary memory.

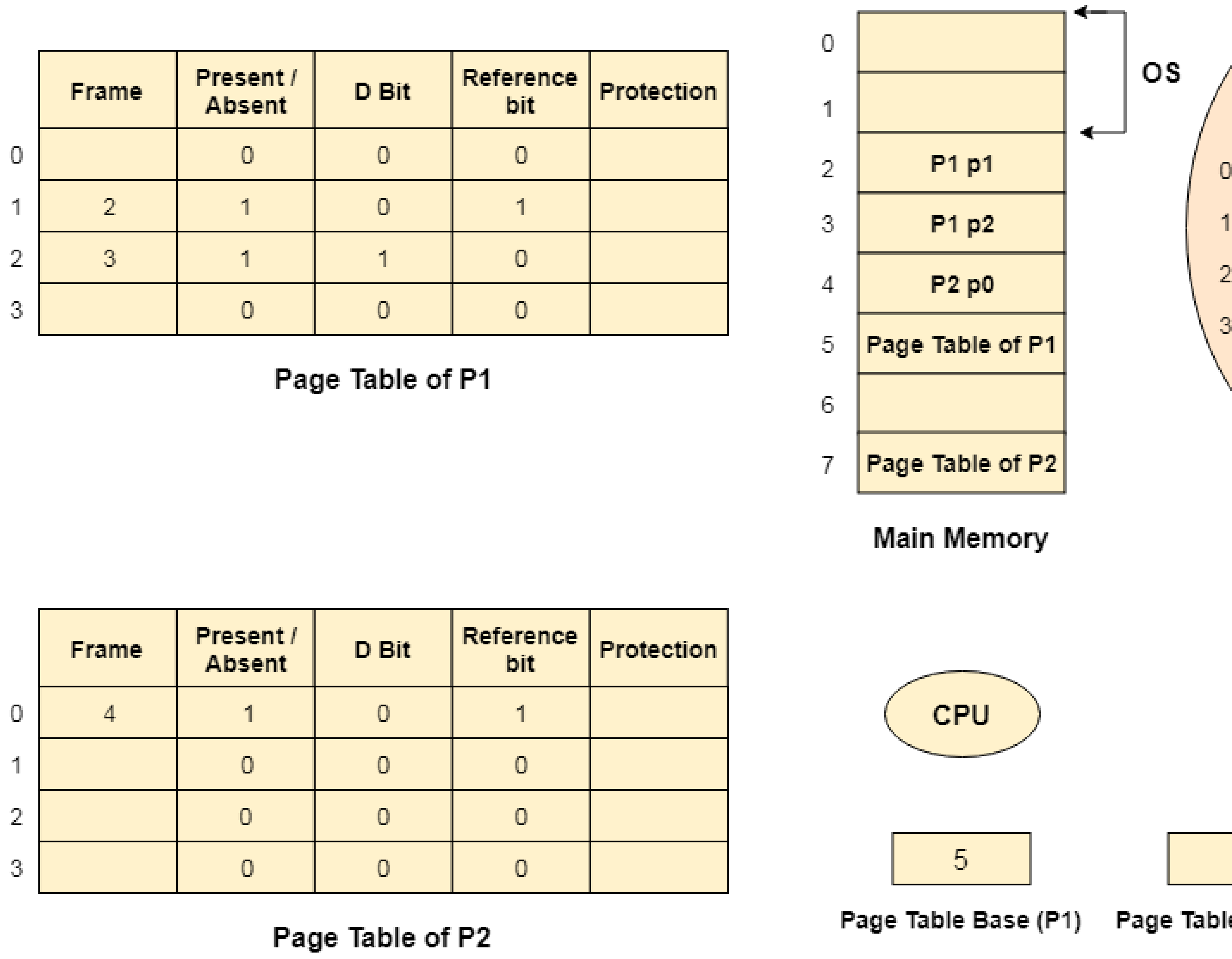
A page is copied to the main memory when its demand is made or page fault occurs. There are various page replacement algorithms which are used to determine the pages which will be replaced. We will discuss each one of them later in detail.

Snapshot of a virtual memory management system

Let us assume 2 processes, P1 and P2, contains 4 pages each. Each page size is 1 KB. The main memory contains 8 frame of 1 KB each. The OS resides in the first two partitions. In the third partition, 1st page of P1 is stored and the other frames are also shown as filled with the different pages of processes in the main memory.

The page tables of both the pages are 1 KB size each and therefore they can be fit in one frame each. The page tables of both the processes contain various information that is also shown in the image.

The CPU contains a register which contains the base address of page table that is 5 in the case of P1 and 7 in the case of P2. This page table base address will be added to the page number of the Logical address when it comes to accessing the actual corresponding entry.



	Frame	Present / Absent	D Bit	Reference bit	Protection
0		0	0	0	
1	2	1	0	1	
2	3	1	1	0	
3		0	0	0	

Page Table of P1

	Frame	Present / Absent	D Bit	Reference bit	Protection
0	4	1	0	1	
1		0	0	0	
2		0	0	0	
3		0	0	0	

Page Table of P2

8.Advantages of Virtual Memory

1. The degree of Multiprogramming will be increased.
2. User can run large application with less real RAM.
3. There is no need to buy more memory RAMs.

9.Disadvantages of Virtual Memory

1. The system becomes slower since swapping takes time.
2. It takes more time in switching between applications.
3. The user will have the lesser hard disk space for its use.

I/O management Function

1.Storage devices

• A storage device is used in the computers to store the data.

• Provides one of the core functions of the modern Computer

2.Types of Storage

There are four type of storage:

• Primary Storage

• Secondary Storage

• Tertiary Storage

• Off-line Storage

2.1.Primary storage devices

• Also known as main memory.

• Main memory is directly or indirectly connected to

the central processing unit via a memory bus.

• The CPU continuously reads instructions stored

there and executes them as required.

Example

– RAM

– ROM

Cache

2.2.Primary Storage

RAM

• It is called Random Access Memory because any of the data in RAM can be accessed just as fast as any of the other data.

• There are two types of RAM:

– DRAM (Dynamic Random Access Memory)

– SRAM (Static Random Access Memory)

Primary Storage

ROM

• This memory is used as the computerbegins to boot up.

• Small programs called firmware are often stored in ROM chips on hardware devices (like a BIOS chip), and they contain instructions the computer can use in performing some of the most basic operations required to operate hardware devices.

• ROM memory cannot be easily or quickly overwritten or modified.

3.Spooling

• Also known as main memory.

Spooling's name comes from the acronym for Simultaneous Peripheral Operation On-Line (SPOOL). Spooling waits until the entire operation is done before sending it to the output device or a network, and your likeliest encounter with spooling probably comes from sending document to a printer. If you've ever wondered why there's a delay between when you press "Print" and a document coming out of the printer, spooling is why – the computer processes the entire print job into a format the printer can handle and sends it down the serial bus to the printer. Spooling is also used for sending and receiving email.

4.Buffering

Spooling is great for some sorts of computer tasks, but it's not appropriate for others. Watching video on YouTube comes with the expectation that clicking "Play" will cause the video to begin playing immediately. For this to work, the website sends small parts of the video when the page loads, and then starts sending the next parts of the video when the "Play" button is clicked. These subsequent parts are queued up in a buffer so that the video plays smoothly even though it's not fully downloaded when you start it.

Linux Operating system

1.History of linux

Linux is the first truly free Unix-like operating system. The underlying GNU Project was launched in 1983 by *Richard Stallman* originally to develop a Unix-compatible operating system called GNU, intended to be entirely free software. Many programs and utilities were contributed by developers around the world, and by 1991 most of the components of the system were ready. Still missing was the kernel.

Linus Torvalds invented Linux itself. In 1991, Torvalds was a student at the University of Helsinki in Finland where he had been using Minix, a non-free Unix-like system, and began writing his own kernel. He started by developing device drivers and hard-drive access, and by September had a basic design that he called Version 0.01. This kernel, which is called Linux, was afterwards combined with the GNU system to produce a complete free operating system.

On October 5th, 1991, Torvalds sent a posting to the comp.os.minix newsgroup announcing the release of Version 0.02, a basic version that still needed Minix to operate, but which attracted considerable interest nevertheless. The kernel was then rapidly improved by Torvalds and a growing number of volunteers communicating over the*Internet*, and by December 19th a functional, stand-alone Unix-like Linux system was released as Version 0.11.

On January 5, 1992, Linux Version 0.12 was released, an improved, stable kernel. The next release was called Version 0.95, to reflect the fact that it was becoming a full-featured system. After that Linux became an underground phenomenon, with a growing group of distributed programmers that continue to debug, develop, and enhance the source code baseline to this day.

Torvalds released Version 0.11 under a freeware license of his own devising, but then released Version 0.12 under the well established GNU General Public License. More and more free software was created for Linux over the next several years.

Linux continued to be improved through the 1990's, and started to be used in large-scale applications like *web hosting*, networking, and database serving, proving ready for production use. Version 2.2, a major update to the Linux kernel, was officially released in January 1999. By the year 2000, most computer companies supported Linux in one way or another, recognizing a common standard that could finally reunify the fractured world of the *Unix Wars*. The next major release was V2.4 in January 2001, providing(among other improvements) compatibility with the upcoming generations of Intel's 64-bit Itanium computer processors.

Although Torvalds continued to function as the Linux kernel release manager, he avoided work at any of the many companies involved with Linux in order to avoid showing favoritism to any particular organization, and instead went to work for a company called Transmeta and helped develop mobile computing solutions, and made his home at the Open Source Development Labs (OSDL), which merged into The Linux Foundation.

Linux commands and Filters