MCA 104:OPERATING SYSTEM

UNIT – I

INTRODUCTION: Operating Systems Overview, Operating system functions, Overview of Computer Operating systems, Protection and Security, Distributed Systems, Special purpose systems, Operating System structures system components, Operating system services, system calls, system programs, system structure, Virtual machines, System Design and Implementation, System Generation.

PROCESS MANAGEMENT: Process Concept, Process Scheduling, Operations on Processes, Inter process Communication, Communication in Client – Server Systems

UNIT – II

CPU SCHEDULING: Basic Concepts, Scheduling Criteria, Scheduling Algorithms, Multiple-Processor Scheduling, Real-time Scheduling, Algorithm Evaluation

PROCESS SYNCHRONIZATION: Background, The Critical-Section Problem, Peterson's Solution, Synchronization Hardware, Semaphores, Classic Problems of Synchronization, Monitors, Atomic Transactions

DEADLOCKS: System Model, Deadlock Characterization, Methods for handling Deadlocks, Deadlock Prevention, Deadlock Avoidance, Deadlock Detection, Recovery from Deadlock

UNIT – III

MEMORY MANAGEMENT: Background, Swapping, Contiguous Memory Allocation, paging, Structure of the Page Table, Segmentation, Segmentation with Paging.

VIRTUAL MEMORY: Background, Demand Paging, Process Creation, Page Replacement, Allocation of Frames, Thrashing.

File System Interface: File Concept, Access Methods, Directory Structure, File – System Mounting, File Sharing, Protection.

UNIT – IV

FILE SYSTEM IMPLEMENTATION: File System Structure, File System Implementation, Directory Implementation, Allocation Methods, Free Space Management.

MASS - STORAGE STRUCTURE: Disk Structure, Disk Scheduling, Disk Management, Swap- Space management, RAID Structure, Stable – Storage Implementations, Tertiary Storage Structure

UNIT – V

SECURITY : User Authentication, program threads, system theats, security system facilities.

LINUX SYSTEMS: History, design principles, kernel modules, Management scheduling, scheduling, memory management and file system.

TEXTBOOK:

1. Operating System Principles by Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Seventh Edition.

LECTURE NOTES

UNIT-I

1). Overview of Operating Systems (or) Introduction to 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.

Definition of OS :-

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.

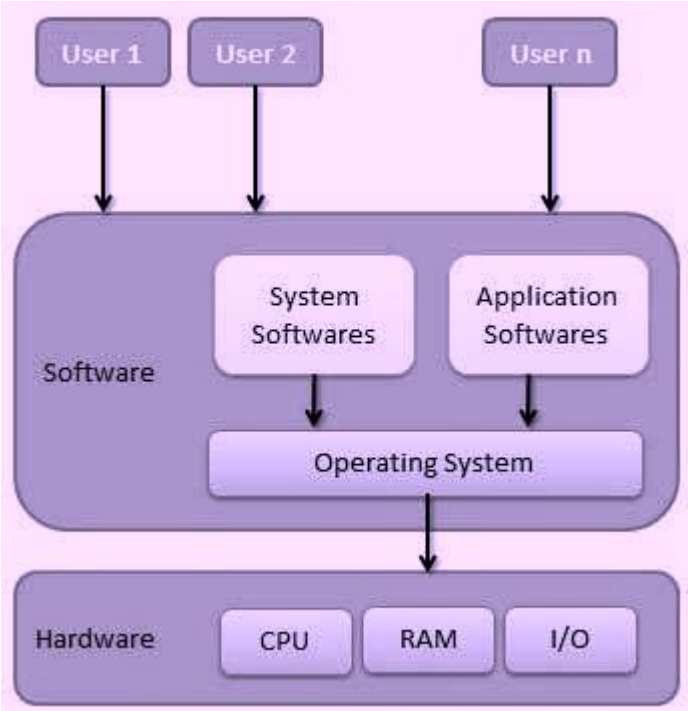


Figure : Operating System

2). Functions of Operating System.

Following are some of the important functions of an operating System.

1. Memory Management
2. Processor Management
3. Device Management
4. File Management
5. Security

6. Control over system performance
7. Job accounting
8. Error detecting aids
9. Coordination between other software and users

1. Memory Management :-

Memory management refers to management of Primary Memory or Main Memory. For a program to be executed, it must in the main memory. An Operating System does the following activities for memory management :

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

2. Processor Management:-

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

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

3. Device Management :-

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

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

4. **File Management :-**

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions. An Operating System does the following activities for file management :

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

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

3). Protection and Security

Protection and security requires that computer resources such as CPU, softwares, memory etc. are protected. This extends to the operating system as well as the data in the system. This can be done by ensuring integrity, confidentiality and availability in the operating system. The system must be protect against unauthorized access, viruses, worms etc.

Threats to Protection and Security:-

A threat is a program that is malicious in nature and leads to harmful effects for the system. Some of the common threats that occur in a system are given below:

Virus :- Viruses are generally small snippets of code embedded in a system. They are very dangerous and can corrupt files, destroy data, crash systems etc.

Trojan Horse :-A Trojan horse can secretly access the login details of a system. Then a malicious user can use these to enter the system.

Trap Door :-A trap door is a security breach that may be present in a system without the knowledge of the users. It can be exploited to harm the data or files in a system by malicious people.

Worm :-A worm can destroy a system by using its resources to extreme levels. It can generate multiple copies which claim all the resources and don't allow any other processes to access them. A worm can shut down a whole network in this way.

Protection and Security Methods :-

The different methods that may provide protection and security for different computer systems are given below:

Authentication:-

The operating system makes sure that all the users are authenticated before they access the system. The different ways to make sure that the users are authentic are:

- Username/ Password

  Each user has a distinct username and password combination and they need to enter it correctly before they can access the system.

- User Key/ User Card

  The users need to punch a card into the card slot or use their individual key on a keypad to access the system.

- User Attribute Identification

  Different user attribute identifications that can be used are fingerprint, eye retina etc. These are unique for each user and are compared with the existing samples in the database. The user can only access the system if there is a match.

One Time Password:-

These passwords provide a lot of security for authentication purposes. A one time password can be generated exclusively for a login every time a user wants to enter the system. It cannot be used more than once.

4). Special-Purpose Systems - Distributed Systems

There are different classes of computer systems whose functions are more limited and whose objective is to deal with limited computation domains. Some of them are given below.

1. Real-Time Embedded Systems:- Embedded computers are the most prevalent form of computers in existence. These devices are found everywhere, from car engines and manufacturing robots to VCRs and microwave ovens. They tend to have very specific tasks. The systems they run on are usually primitive, and so the operating systems provide limited features. Usually, they have little or no user interface, preferring to spend their time monitoring and managing hardware devices, such as automobile engines and robotic arms.

Embedded systems almost always run real-time operating systems. A real-time system is used when rigid time requirements have been placed on the operation of a processor or the flow of data; thus, it is often used as a control device in a dedicated application. Sensors bring data to the computer. The computer must analyze the data and possibly adjust controls to modify the sensor inputs. Systems that control scientific experiments, medical imaging systems, industrial control systems, and certain display systems are real-time systems.

2. Multimedia Systems:- Most operating systems are designed to handle conventional data such as text files, programs, word-processing documents, and spreadsheets. However, a recent trend in technology is the incorporation of multimedia data into computer systems. Multimedia data consist of audio and video files as well as conventional files. These data differ from conventional data in that multimedia data—such as frames of video—must be delivered (streamed) according to certain time restrictions (for example, 30 frames per second). Multimedia describes a wide range of applications that are in popular use today. These include audio files such as MP3 DVD movies, video conferencing, and short video clips of movie previews or news stories downloaded over the Internet. Multimedia applications may also include live webcasts (broadcasting over the World Wide Web) of speeches or sporting events.

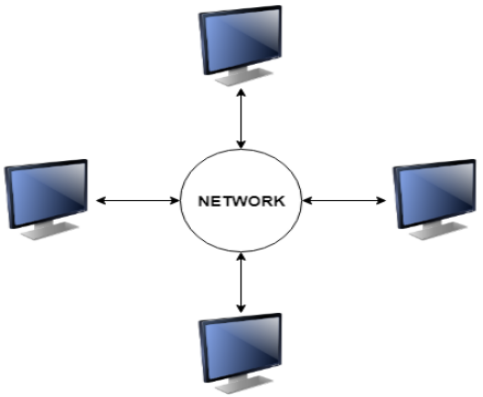
3.Handheld Systems:- Handheld systems include personal digital assistants (PDAs), such as Palm and Pocket-PCs, and cellular telephones, many of which use special-purpose embedded operating systems. Developers of handheld systems and applications face many challenges, most of which are due to the limited size of such devices. For example, a PDA is typically about 5 inches in

height and 3 inches in width. Because of their size, most handheld devices have a small amount of memory, slow processors, and small display screens.

Distributed Systems:-

A distributed system contains multiple nodes that are physically separate but linked together using the network. All the nodes in this system communicate with each other and handle processes in tandem. Each of these nodes contains a small part of the distributed operating system software.

A diagram to better explain the distributed system is given below:



DISTRIBUTED OPERATING SYSTEM

Types of Distributed Systems:- The nodes in the distributed systems can be arranged in the form of client/server systems or peer to peer systems:

- **Client/Server Systems: -** In client server systems, the client requests a resource and the server provides that resource. A server may serve multiple clients at the same time while a client is in contact with only one server. Both the client and server usually communicate via a computer network and so they are a part of distributed systems.

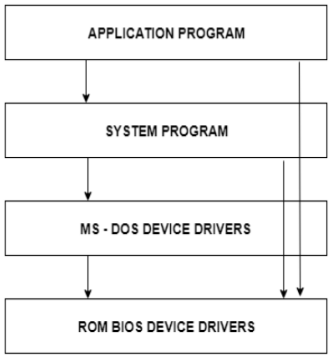
- **Peer to Peer Systems**: - The peer to peer systems contains nodes that are equal participants in data sharing. All the tasks are equally divided between all the nodes. The nodes interact with each other as required as share resources. This is done with the help of a network.

5). Operating System Structures

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 simple structure. These started as small systems and rapidly expanded much further than their scope. A common example of this is MS-DOS. The following figure represents the structure of MS-DOS.

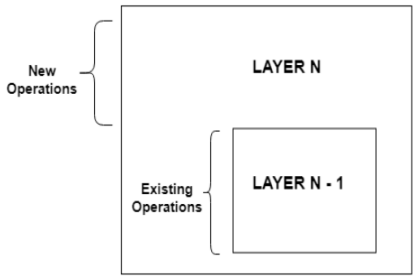


MS-DOS STRUCTURE

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 can fit without changing the outer specifications.

Layered Structure :-

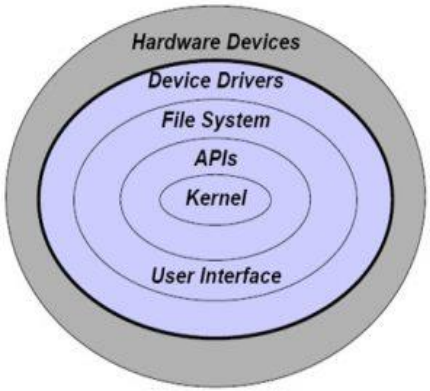
One way to achieve modularity in the operating system is the layered approach. In this, the bottom layer is the hardware and the topmost layer is the user interface. The following figure represents the layered approach.



Layered Structure of Operating System

As seen from the above figure, 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.

6).Componenets of an operating system



The components of an operating system play a key role to make a variety of computer system parts work together. The operating components are given below.

Kernel :-The kernel in the OS provides the basic level of control on all the computer peripherals.

Process Execution:-The OS gives an interface between the hardware as well as an application program so that the program can connect through the hardware device by simply following procedures & principles configured into the OS. The program execution mainly includes a process created through an OS kernel that uses memory space as well as different types of other resources.

Interrupt:- An interrupt is nothing but one kind of signal between a device as well as a computer system otherwise from a program in the computer that requires the OS to leave and decide accurately what to do subsequently. Whenever an interrupt signal is received, then the hardware of the computer puts on hold automatically whatever computer program is running presently, keeps its status & runs a computer program which is connected previously with the interrupt.

Memory Management:-The functionality of an OS is nothing but memory management which manages main memory & moves processes backward and forward between disk & main memory during implementation. It verifies how much memory can be allocated to processes and also makes a decision to know which process will obtain memory at what time.

Multitasking:-It describes the working of several independent computer programs on a similar computer system. Multitasking in an OS allows an operator to execute one or more computer tasks at a time. Since many computers can perform one or two tasks at a time, usually this can be done with the help of time-sharing, where each program uses the time of a computer to execute.

Networking:-Networking can be defined as when the processor interacts with each other through communication lines. The design of communication-network must consider routing, connection methods, safety, the problems of opinion & security.

Security:-If a computer has numerous individuals to allow the immediate process of various processes, then the many processes have to be protected from other activities.

User Interface:-A GUI or user interface (UI) is the part of an OS that permits an operator to get the information. A user interface based on text displays the text as well as its commands which are typed over a command line with the help of a keyboard.The UI of any application can be classified into two types namely GUI (graphical UI) & CLI (command line user interface).

7). Services of an operating system

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

Program execution :- 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 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 :- 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.

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.

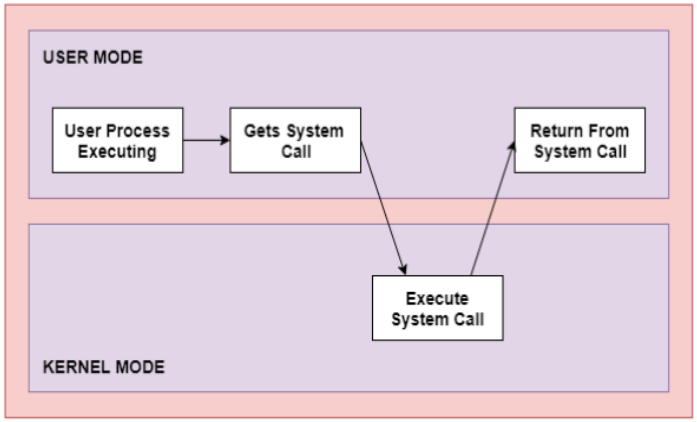
Protection :- Following are the major activities of an operating system with respect to protection −

- 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.

8). System calls

The interface between a process and an operating system is provided by system calls. 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.

The following figure representing the execution of the system call



In general, system calls are required in the following situations −

- If a file system requires the creation or deletion of files. Reading and writing from files also require a system call.
- Creation and management of new processes.
- Network connections also require system calls. This includes sending and receiving packets.

- Access to a hardware devices such as a printer, scanner etc. requires a system call.

Types of System Calls :- The following are the different 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 :

open() :- The open() system call is used to provide access to a file in a file system.

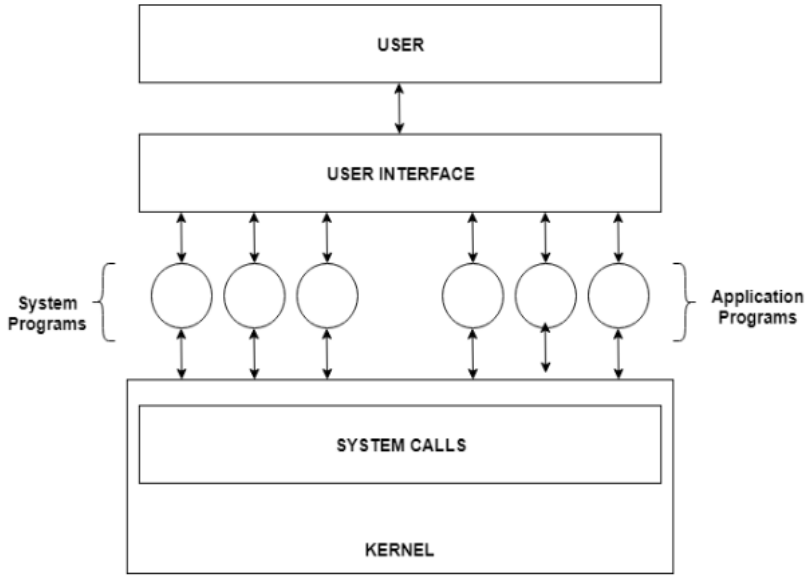
read() :- The read() system call is used to access data from a file that is stored in the file system.

write() :- The write() system calls writes the data from a user buffer into a device such as a file.

close() :- The close() system call is used to terminate access to a file system

9). System programs

System programs provide an environment where programs can be developed and executed. The system program serves as a part of the operating system. It traditionally lies between the user interface and the system calls. The following figure describes system programs in the operating system hierarchy is as follows :



Types of System Programs :- The following are the different types of system programs :

- **Status Information :-** The status information system programs provide required data on the current or past status of the system. This may include the system date, system time, available memory in system, disk space, logged in users etc.
- **Communications :-** These system programs are needed for system communications such as web browsers. Web browsers allow systems to communicate and access information from the network as required.
- **File Manipulation :-** These system programs are used to manipulate system files. This can be done using various commands like create, delete, copy, rename, print etc. These commands can create files, delete files, copy the contents of one file into another, rename files, print them etc.
- **Program Loading and Execution :-** The system programs that deal with program loading and execution make sure that programs can be loaded into memory and executed correctly. Loaders and Linkers are a prime example of this type of system programs.
- **File Modification :-** System programs that are used for file modification basically change the data in the file or modify it in some other way. Text editors are a big example of file modification system programs.

- **Application Programs** :- Application programs can perform a wide range of services as per the needs of the users. These include programs for database systems, word processors, plotting tools, spreadsheets, games, scientific applications etc.
- **Programming Language Support** :- These system programs provide additional support features for different programming languages. Some examples of these are compilers, debuggers etc. These compile a program and make sure it is error free respectively.

10). Virtual Machines

Virtual Machine abstracts the hardware of our personal computer such as CPU, disk drives, memory, NIC (Network Interface Card) etc, into many different execution environments as per our requirements, hence giving us a feel that each execution environment is a single computer. For example, VirtualBox.

When we run different processes on an operating system, it creates an illusion that each process is running on a different processor having its own virtual memory, with the help of CPU scheduling and virtual-memory techniques. There are additional features of a process that cannot be provided by the hardware alone like system calls and a file system. The virtual machine approach does not provide these additional functionalities but it only provides an interface that is same as basic hardware. Each process is provided with a virtual copy of the underlying computer system.

We can create a virtual machine for several reasons, all of which are fundamentally related to the ability to share the same basic hardware yet can also support different execution Environments, i.e., different operating systems simultaneously.

The main drawback with the virtual-machine approach involves disk systems. Let us suppose that the physical machine has only three disk drives but wants to support seven virtual machines. Obviously, it cannot allocate a disk drive to each virtual machine, because virtual-machine software itself will need substantial disk space to provide virtual memory. The solution is to provide virtual disks.

Users are thus given their own virtual machines. After which they can run any of the operating systems or software packages that are available on the underlying machine. The virtual-machine software is concerned with multi-programming multiple virtual machines onto a physical machine, but it does not need to consider any user-support software. This arrangement can provide a useful way to divide the problem of designing a multi-user interactive system, into two smaller pieces.

Advantages:

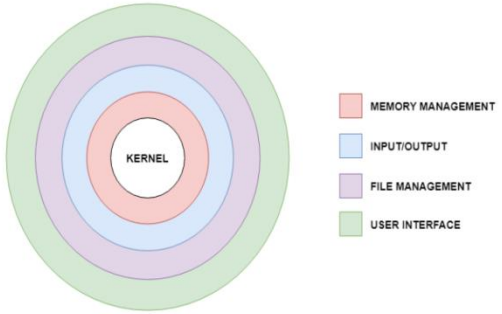
1. There are no protection problems because each virtual machine is completely isolated from all other virtual machines.
2. Virtual machine can provide an instruction set architecture that differs from real computers.
3. Easy maintenance, availability and convenient recovery.

Disadvantages:

1. When multiple virtual machines are simultaneously running on a host computer, one virtual machine can be affected by other running virtual machines, depending on the workload.
2. Virtual machines are not as efficient as a real one when accessing the hardware.

11). Operating System Design and Implementation

An operating system is a construct that allows the user application programs to interact with the system hardware. Operating system by itself does not provide any function but it provides an atmosphere in which different application programs can do useful work. There are many problems that can occur while designing and implementing an operating system.



Layered Operating System Design

Operating System Design Goals:-

It is quite complicated to define all the goals and specifications of the operating system while designing it. The design changes depending on the type of the operating system i.e if it is batch system, time shared system, single user system, multi user system, distributed system etc.There are basically two types of goals while designing an operating system. These are:

- **User Goals**:- The operating system should be convenient, easy to use, reliable, safe and fast according to the users.
- **System Goals:-** The operating system should be easy to design, implement and maintain. These are specifications required by those who create, maintain and operate the operating system. But there is not specific method to achieve these goals as well.

Operating System Implementation:-The operating system needs to be implemented after it is designed. Earlier they were written in assembly language but now higher level languages are used. The first system not written in assembly language was the Master Control Program (MCP) for Burroughs Computers.

Advantages of Higher Level Language

There are multiple advantages to implementing an operating system using a higher level language such as: the code is written more fast, it is compact and also easier to debug and understand. Also, the operating system can be easily moved from one hardware to another if it is written in a high level language.

Disadvantages of Higher Level Language

Using high level language for implementing an operating system leads to a loss in speed and increase in storage requirements. However in modern systems only a small amount of code is needed for high performance, such as the CPU scheduler and memory manager.

12). Operating System Generations

Operating Systems have evolved over the years. There are four generations of operating systems. These can be described as follows :

1.The First Generation (1945 - 1955): Vacuum Tubes and Plug boards:-

Digital computers were not constructed until the second world war. Calculating engines with mechanical relays were built at that time. However, the mechanical relays were very slow and were later replaced with vacuum tubes. These machines were enormous but were still very slow.

These early computers were designed, built and maintained by a single group of people. Programming languages were unknown and there were no operating systems so all the programming was done in machine language. All the problems were simple numerical calculations.

By the 1950's punch cards were introduced and this improved the computer system. Instead of using plug boards, programs were written on cards and read into the system.

2.The Second Generation (1955 - 1965): Transistors and Batch Systems:-

Transistors led to the development of the computer systems that could be manufactured and sold to paying customers. These machines were known as mainframes and were locked in air-conditioned computer rooms with staff to operate them.

The Batch System was introduced to reduce the wasted time in the computer. A tray full of jobs was collected in the input room and read into the magnetic tape. After that, the tape was rewound and mounted on a tape drive. Then the batch operating system was loaded in which read the first job from the tape and run it. The output was written on the second tape. After the whole batch was done, the input and output tapes were removed and the output tape was printed.

3.The Third Generation (1965 - 1980): Integrated Circuits and Multiprogramming:-

Until the 1960's, there were two types of computer systems i.e the scientific and the commercial computers. These were combined by IBM in the System/360. This used integrated circuits and provided a major price and performance advantage over the second generation systems.

The third generation operating systems also introduced multiprogramming. This meant that the processor was not idle while a job was completing its I/O operation. Another job was scheduled on the processor so that its time would not be wasted.

4.The Fourth Generation (1980 - Present): Personal Computers:-

Personal Computers were easy to create with the development of large-scale integrated circuits. These were chips containing thousands of transistors on a square centimetre of silicon. Because of these, microcomputers were much cheaper than minicomputers and that made it possible for a single individual to own one of them.

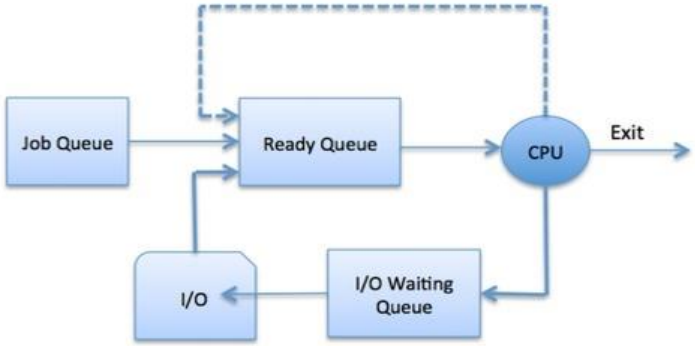
The advent of personal computers also led to the growth of networks. This created network operating systems and distributed operating systems. The users were aware of a network while using a network operating system and could log in to remote machines and copy files from one machine to another.

13). Process concept and Process scheduling

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy. Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Process Scheduling Queues:- The 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.



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

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

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

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

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

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

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

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

14). Operations on process

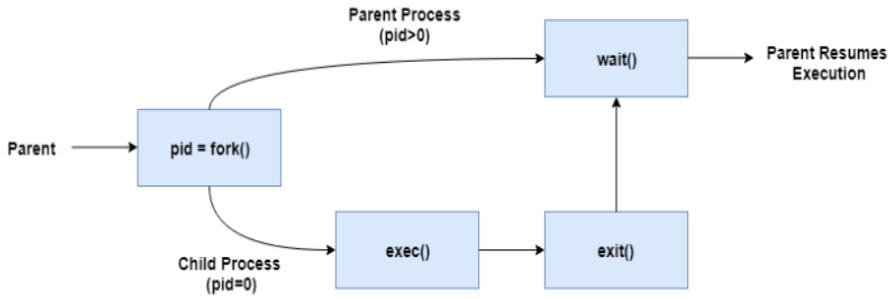
There are many operations that can be performed on processes. Some of these are process creation, process pre-emption, process blocking, and process termination.

1.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.

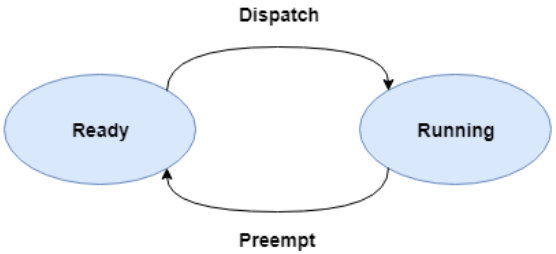
A diagram that demonstrates process creation using fork() is as follows :



Process Creation Using fork()

2.<u>Process Preemption</u>:-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.

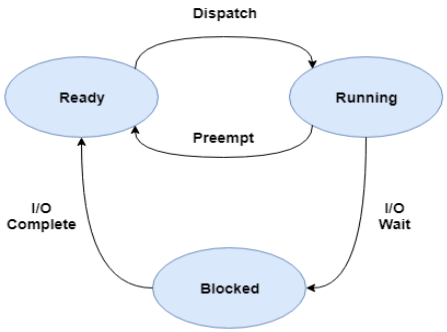
A diagram that demonstrates process preemption is as follows:



Process Preemption

3.<u>Process Blocking</u>:- 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 :



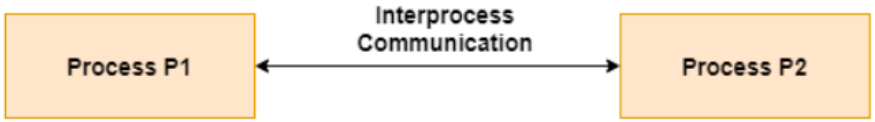
Process Blocking

4.<u>Process Termination</u>:- 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.

15). Inter Process communication (IPC)

Inter 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 the transferring of data from one process to another.

A diagram that illustrates inter process communication is as follows :

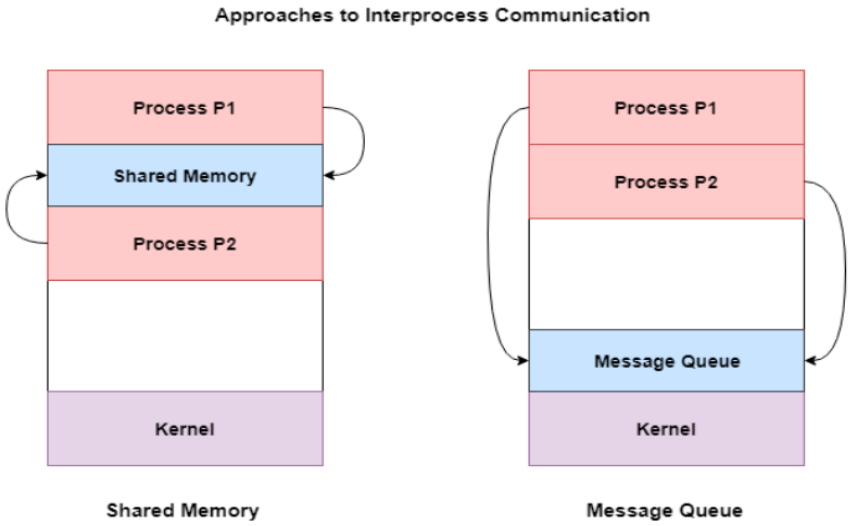


Approaches to Inter process Communication:-

The different approaches to implement inter process communication are given as follows:

- **Pipe :-** A pipe is a data channel that is unidirectional. Two pipes can be used to create a two-way data channel between two processes. This uses standard input and output methods. Pipes are used in Windows operating systems.
- **Socket :-**The socket is the endpoint for sending or receiving data in a network. This is true for data sent between processes on the same computer or data sent between different computers on the same network. Most of the operating systems use sockets for inter process communication.
- **File :-**A file is a data record that may be stored on a disk or acquired on demand by a file server. Multiple processes can access a file as required. All operating systems use files for data storage.
- **Signal :-**Signals are useful in inter process communication in a limited way. They are system messages that are sent from one process to another. Normally, signals are not used to transfer data but are used for remote commands between processes.
- **Shared Memory :-**Shared memory is the memory that can be simultaneously accessed by multiple processes. This is done so that the processes can communicate with each other. All Windows operating systems use shared memory.
- **Message Queue :-**Multiple processes can read and write data to the message queue without being connected to each other. Messages are stored in the queue until their recipient retrieves them. Message queues are quite useful for inter process communication and are used by most operating systems.

A diagram that demonstrates message queue and shared memory methods of inter process communication is as follows :



16). Communication in Client – Server Systems

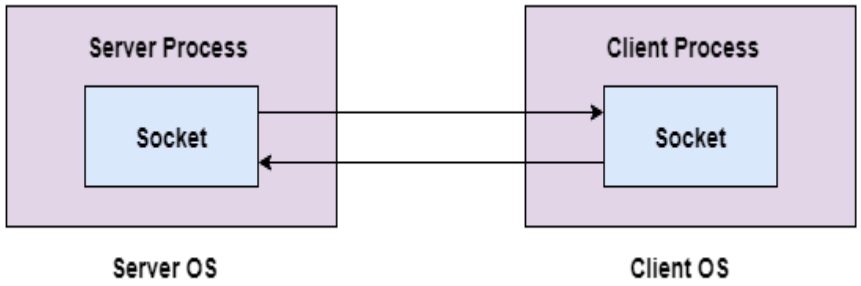
Client/Server communication involves two components, namely a client and a server. There are usually multiple clients in communication with a single server. The clients send requests to the server and the server responds to the client requests. There are three main methods to client/server communication and these are given below

1. Sockets:-

Sockets facilitate communication between two processes on the same machine or different machines. They are used in a client/server framework and consist of the IP address and port number. Many application protocols use sockets for data connection and data transfer between a client and a server.

Socket communication is quite low-level as sockets only transfer an unstructured byte stream across processes. The structure on the byte stream is imposed by the client and server applications.

A diagram that illustrates sockets is as follows:

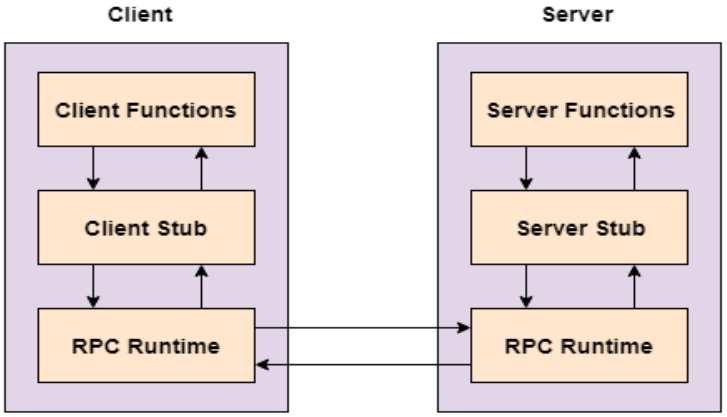


2. Remote Procedure Calls(RPC):-

These are inter process communication techniques that are used for client-server based applications. A remote procedure call is also known as a subroutine call or a function call.

A client has a request that the RPC translates and sends to the server. This request may be a procedure or a function call to a remote server. When the server receives the request, it sends the required response back to the client.

A diagram that illustrates remote procedure calls is given as follows −



3.Pipes:-

These are communication methods that contain two end points. Data is entered from one end of the pipe by a process and consumed from the other end by the other process.

The two different types of pipes are ordinary pipes and named pipes. Ordinary pipes only allow one way communication. For two way communication, two pipes are required. Ordinary pipes have a parent child relationship between the processes as the pipes can only be accessed by processes that created or inherited them.

Named pipes are more powerful than ordinary pipes and allow two way communication. These pipes exist even after the processes using them have terminated. They need to be explicitly deleted when not required anymore.

A diagram that demonstrates pipes are given as follows :

Short Questions:

1.What is operating system?

2.What are the functions of operating system?

3.What is Operating system structure?

4.What are the components of an operating system?

Long Questions:

1.What is System calls and what are the types of System calls?

2.what are System programs and types of system programs?

3.Explain about operating system design and implementation in detail?

4.Explain about Inter process Communication in detail?

UNIT-II

1). CPU Scheduling in Operating System

CPU scheduling is a process that 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.

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.
2. When a process switches from the **running** state to the **ready** state .
3. When a process switches from the **waiting** state to the **ready** state.
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**.

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.

CPU Scheduling: Scheduling Criteria

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

- **CPU Utilization**:- To make out the best use of the CPU and not to waste any CPU cycle, the CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)
- Throughput:- It is the total number of processes completed per unit of time. This may range from 10/second to 1/hour depending on the specific processes.
- Turn around Time:- It is the amount of time taken to execute a particular process, i.e. The interval from the time of submission of the process to the time of completion of the process.
- Waiting Time:- The amount of time a process has been waiting in the ready queue to acquire get control on the CPU.
- Load Average:- It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.
- Response Time:- Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution (final response).

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

2).CPU Scheduling Algorithms

To decide which process to execute first and which process to execute last to achieve maximum CPU utilization, computer scientists have defined some algorithms, they are:

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

1) First Come First Serve (FCFS) Scheduling

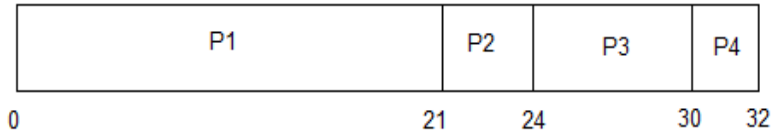
In the "First come first serve" scheduling algorithm, as the name suggests, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.

For every scheduling algorithm, **Average waiting time** is a crucial parameter to judge its performance. Lower the Average Waiting Time, better the scheduling algorithm.

Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in the same order, with **Arrival Time** `0`, and given **Burst Time**, let's find the average waiting time using the FCFS scheduling algorithm.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The average waiting time will be = (0 + 21 + 24 + 30)/4 = 18.75 ms



This is the GANTT chart for the above processes

The average waiting time will be `18.75 ms`

For the above given processes, first **P1** will be provided with the CPU resources,

- Hence, waiting time for **P1** will be `0`
- **P1** requires `21 ms` for completion, hence waiting time for **P2** will be `21 ms`
- Similarly, waiting time for process **P3** will be execution time of **P1** + execution time for **P2**, which will be `(21 + 3) ms` = `24 ms`.

- For process **P4** it will be the sum of execution times of **P1**, **P2** and **P3**.

The **GANTT chart** above perfectly represents the waiting time for each process.

2) Shortest Job First(SJF) Scheduling

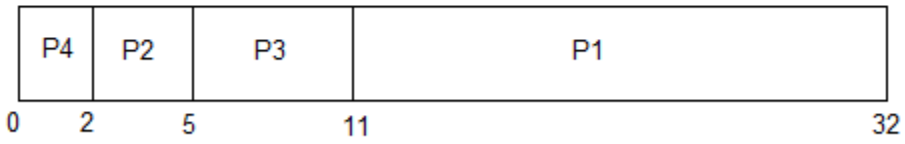
Shortest Job First scheduling works on the process with the shortest **burst time** or **duration** first.

Consider the below processes available in the ready queue for execution, with **arrival time** as 0 for all and given **burst times**.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

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

Hence the GANTT chart will be following :



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

As we can see in the **GANTT chart** above, the process **P4** will be picked up first as it has the shortest burst time, then **P2**, followed by **P3** and at last **P1**.

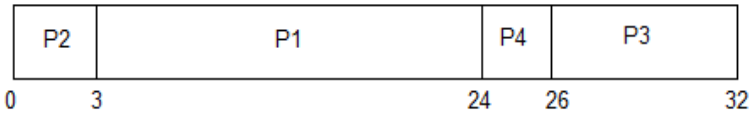
3) Priority CPU Scheduling

In priority scheduling the scheduling is done on the basis of priority of the process where the process which is most urgent is processed first, followed by the ones with lesser priority in order. Processes with same priority are executed in FCFS manner.

Consider the below table for processes with their respective CPU burst times and the priorities.

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

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



The average waiting time will be, (0 + 3 + 24 + 26)/4 = 13.25 ms

As we can see in the GANTT chart that the processes are given CPU time just on the basis of the priorities. But in case of priority scheduling if new higher priority processes keeps coming in the ready queue then the processes waiting in the ready queue with lower priority may have to wait for long durations before getting the CPU for execution.

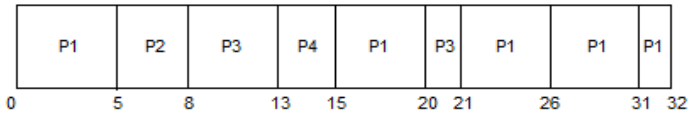
4) Round Robin Scheduling

Round Robin (RR) scheduling algorithm is mainly designed for time-sharing systems. This algorithm is similar to FCFS scheduling, but in Round Robin (RR) scheduling, pre-emption is added which enables the system to switch between processes.

- A fixed time is allotted to each process, called a **quantum**, for execution.
- Once a process is executed for the given time period that process is pre-empted and another process executes for the given time period.
- Context switching is used to save states of pre-empted processes.
- This algorithm is simple and easy to implement and the most important is thing is this algorithm is starvation-free as all processes get a fair share of CPU.
- Round Robin Scheduling algorithm resides under the category of **Pre-emptive** Algorithms.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The GANTT chart for round robin scheduling will be,



The average waiting time will be, 11 ms.

5) Multilevel Queue Scheduling Algorithm

A multi-level queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.

For example, separate queues might be used for foreground and background processes. The foreground queue might be scheduled by the Round Robin algorithm, while the background queue is scheduled by an FCFS algorithm.

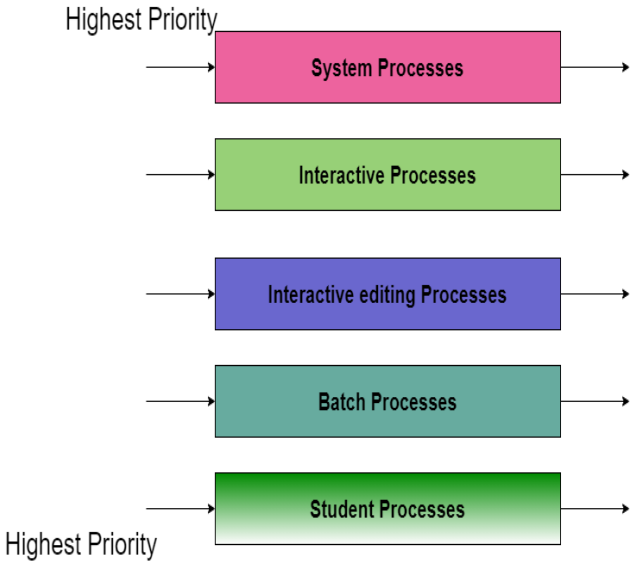
Consider an example of a multilevel queue-scheduling algorithm with five queues:

1. System Processes
2. Interactive Processes
3. Interactive Editing Processes
4. Batch Processes

5. Student Processes

Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process will be pre-empted.

In this case, if there are no processes on the higher priority queue only then the processes on the low priority queues will run. For **Example:** Once processes on the system queue, the Interactive queue, and Interactive editing queue become empty, only then the processes on the batch queue will run.

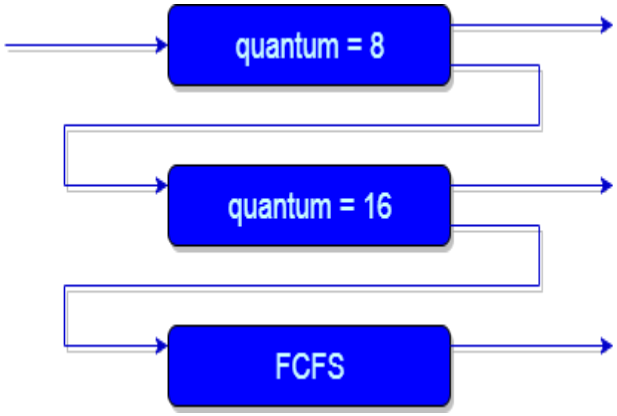


6) Multilevel Feedback Queue Scheduling

In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. In multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue.

In general, a multilevel feedback queue scheduler is defined by the following parameters:

- The number of queues.
- The scheduling algorithm for each queue.
- The method used to determine when to upgrade a process to a higher-priority queue.
- The method used to determine when to demote a process to a lower-priority queue.
- The method used to determine which queue a process will enter when that process needs service.



An example of a multilevel feedback queue can be seen in the above figure.

3).Multiple-Processor Scheduling in Operating System

In multiple-processor scheduling **multiple CPU's** are available and hence **Load Sharing** becomes possible. However multiple processor scheduling is more **complex** as compared to single processor scheduling.

Approaches to Multiple-Processor Scheduling :–

One approach is when all the scheduling decisions and I/O processing are handled by a single processor which is called the **Master Server** and the other processors executes only the **user code**. This is simple and reduces the need of data sharing. This entire scenario is called **Asymmetric Multiprocessing**.

A second approach uses **Symmetric Multiprocessing** where each processor is **self scheduling**. All processes may be in a common ready queue or each processor may have its own private queue for ready processes. The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute.

4).Algorithm Evaluation

There are many scheduling algorithms, each with its own parameters. As a result, selecting an algorithm can be difficult. The first problem is defining the criteria to be used in selecting an algorithm. Criteria are often defined in terms of CPU utilization, response time, or throughput. To select an algorithm, we must first define the relative importance of these measures. Our criteria may include several measures, such as:

- Maximizing CPU utilization under the constraint that the maximum response time is 1 second
- Maximizing throughput such that turnaround time is (on average) linearly proportional to total execution time Once the selection criteria have been defined, we want to evaluate the algorithms under consideration.We next describe the various evaluation methods we can use.

- **Deterministic Modeling :-**

One major class of evaluation methods is analytic evaluation. Analytic evaluation uses the given algorithm and the system workload to produce a formula or number that evaluates the performance of the algorithm for that workload. One type of analytic evaluation is deterministic modelling. This method takes a particular predetermined workload and defines the performance of each algorithm for that workload.

Deterministic modelling is simple and fast. It gives us exact numbers, allowing us to compare the algorithms. However, it requires exact numbers for input, and its answers apply only to those cases. The main uses of deterministic modelling are in describing scheduling algorithms and providing examples.

In cases where we are running the same program over and over again and can measure the program's processing requirements exactly, we may be able to use deterministic modelling to select a scheduling algorithm. Furthermore, over a set of examples, deterministic modelling may indicate trends that can then be analyzed and proved separately.

- **Queuing Models :-**

On many systems, the processes that are run vary from day to day, so there is no static set of processes (or times) to use for deterministic modelling. What can be determined; however, is the distribution of CPU and I/O bursts. These distributions can be measured and then approximated or simply estimated. The result is a mathematical formula describing the probability of a particular CPU burst. Commonly, this distribution is exponential and is described by its mean. Similarly, we can describe the distribution of times when processes arrive in the system (the arrival-time distribution). From these two distributions, it is possible to compute the average throughput, utilization, waiting time, and so on for most algorithms. The computer system is described as a network of servers.

Each server has a queue of waiting processes. The CPU is a server with its ready queue, as is the I/O system with its device queues. Knowing arrival rates and service rates, we can compute utilization, average queue length, average wait time, and so on. This area of study is called queueing-network analysis.

- **Simulations :-**

To get a more accurate evaluation of scheduling algorithms, we can use simulations. Running simulations involves programming a model of the computer system. Software data structures represent the major components of the system. The simulator has a variable representing a clock; as this variable's value is increased, the simulator modifies the system state to reflect the activities of the devices, the processes, and the scheduler. As the simulation executes, statistics that indicate algorithm performance are gathered and printed. The data to drive the simulation can be generated in several ways. The most common method uses a random-number generator, which is programmed to generate processes, CPU burst times, arrivals, departures, and so on, according to probability distributions.

5).Process Synchronization

Generally process is categorized into two types on the basis of synchronization and these are given below:

- Independent Process
- Cooperative Process

Independent Processes :-Two processes are said to be independent if the execution of one process does not affect the execution of another process.

Cooperative Processes :- Two processes are said to be cooperative if the execution of one process affects the execution of another process. These processes need to be synchronized so that the order of execution can be guaranteed.

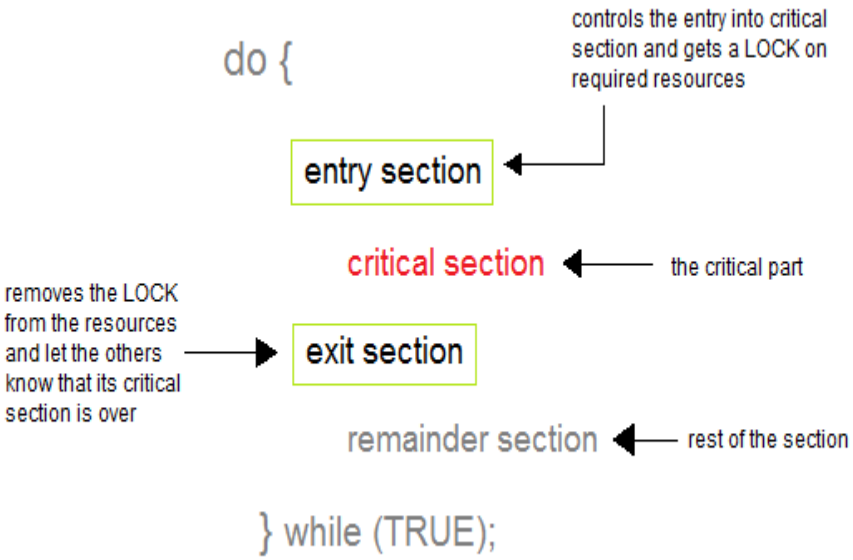
Process synchronization is the task phenomenon of coordinating the execution of processes in such a way that no two processes can have access to the same shared data and resources.

- It is a procedure that is involved in order to preserve the appropriate order of execution of cooperative processes.
- In order to synchronize the processes, there are various synchronization mechanisms.
- Process Synchronization is mainly needed in a multi-process system when multiple processes are running together, and more than one process try to gain access to the same shared resource or any data at the same time.

Race Condition :-At the time when more than one process is either executing the same code or accessing the same memory or any shared variable; In that condition, there is a possibility that the output or the value of the shared variable is wrong so for that purpose all the processes are doing the race to say that my output is correct. This condition is commonly known as **a race condition.**

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. The entry to the critical section is mainly handled by `wait()` function while the exit from the critical section is controlled by the **`signal()`** function.



Entry Section :- In this section mainly the process requests for its entry in the critical section.

Exit Section :- This section is followed by the critical section.

<u>The solution to the Critical Section Problem</u> :-

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

- **Mutual Exclusion: -** Out of a group of cooperating processes, only one process can be in its critical section at a given point of time.
- **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.
- **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, the system must grant the process permission to get into its critical section.

6).Solutions for the Critical Section

The critical section plays an important role in Process Synchronization so that the problem must be solved. Some widely used methods to solve the critical section problem are as follows:

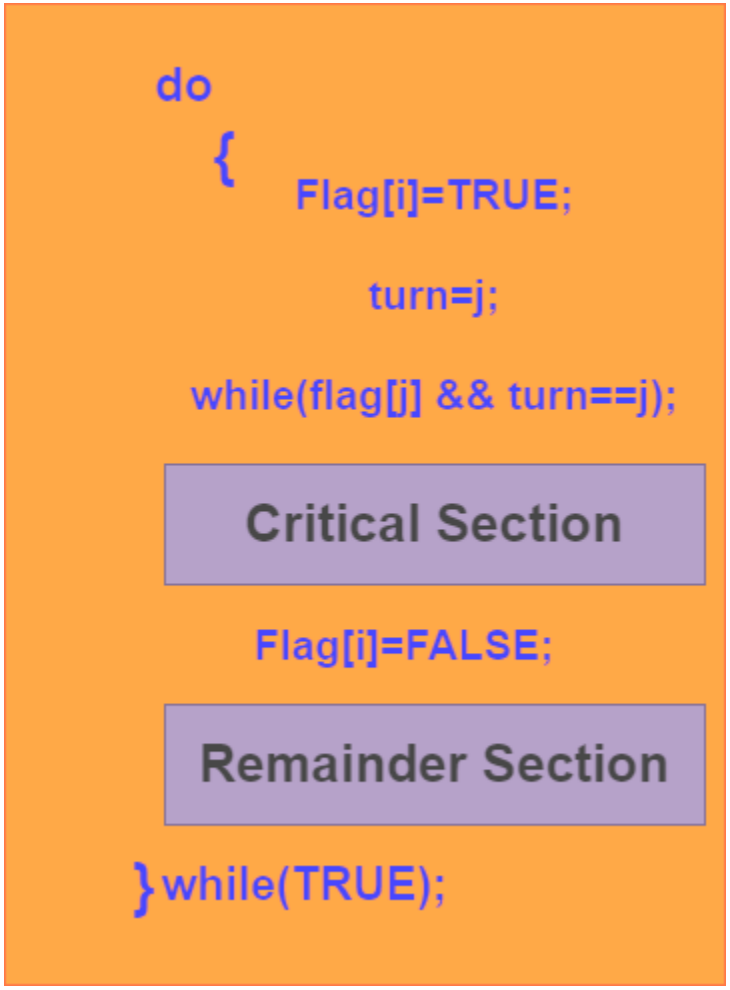
1.Peterson's Solution

This is widely used and software-based solution to critical section problems. Peterson's solution was developed by a computer scientist Peterson that's why it is named so.

With the help of this solution whenever a process is executing in any critical state, then the other process only executes the rest of the code, and vice-versa can happen. This method also helps to make sure of the thing that only a single process can run in the critical section at a specific time.

This solution preserves all three conditions:

- Mutual Exclusion is comforted as at any time only one process can access the critical section.
- Progress is also comforted, as a process that is outside the critical section is unable to block other processes from entering into the critical section.
- Bounded Waiting is assured as every process gets a fair chance to enter the Critical section.



The above shows the structure of process **Pi** in **Peterson's solution.**

- Suppose there are **N processes (P1, P2, ... PN)** and as at some point of time every process requires to enter in the **Critical Section**
- A **FLAG[]** array of size N is maintained here which is by default false. Whenever a process requires to enter in the critical section, it has to set its flag as true. Example: If Pi wants to enter it will set **FLAG[i]=TRUE.**

- Another variable is called **TURN** and is used to indicate the process number that is currently waiting to enter into the critical section.
- The process that enters into the critical section while exiting would change the **TURN** to another number from the list of processes that are ready.
- Example: If the turn is 3 then P3 enters the Critical section and while exiting turn=4 and therefore P4 breaks out of the wait loop.

2.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.

Disabling interrupt on a multiprocessor environment can be time-consuming as the message is passed to all the processors.This message transmission lag delays the entry of threads into the critical section, and the system efficiency decreases.

7).Semaphores

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

The definitions of wait and signal are as follows:

Wait :-The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

```
wait(S)
{
   while (S<=0);

   S--;
}
```

Signal :-The signal operation increments the value of its argument S.

```
signal(S)
{
```

```
    S++;
}
```

Types of Semaphores :-

There are two main types of semaphores : counting semaphores and binary semaphores.

Counting Semaphores :- These are integer value semaphores and have an unrestricted value domain. These semaphores are used to coordinate the resource access, where the semaphore count is the number of available resources. If the resources are added, semaphore count automatically incremented and if the resources are removed, the count is decremented.

Binary Semaphores :- The binary semaphores are like counting semaphores but their value is restricted to 0 and 1. The wait operation only works when the semaphore is 1 and the signal operation succeeds when semaphore is 0. It is sometimes easier to implement binary semaphores than counting semaphores.

Advantages of Semaphores :- Some of the advantages of semaphores are as follows:

- Semaphores allow only one process into the critical section. They follow the mutual exclusion principle strictly and are much more efficient than some other methods of synchronization.
- There is no resource wastage because of busy waiting in semaphores as processor time is not wasted unnecessarily to check if a condition is fulfilled to allow a process to access the critical section.
- Semaphores are implemented in the machine independent code of the microkernel. So they are machine independent.

Disadvantages of Semaphores :- Some of the disadvantages of semaphores are as follows :

- Semaphores are complicated so the wait and signal operations must be implemented in the correct order to prevent deadlocks.
- Semaphores are impractical for last scale use as their use leads to loss of modularity. This happens because the wait and signal operations prevent the creation of a structured layout for the system.
- Semaphores may lead to a priority inversion where low priority processes may access the critical section first and high priority processes later.

8).Classical problems of Synchronization

The following problems of synchronization are considered as classical problems. These problems are used for testing nearly every newly proposed synchronization scheme.

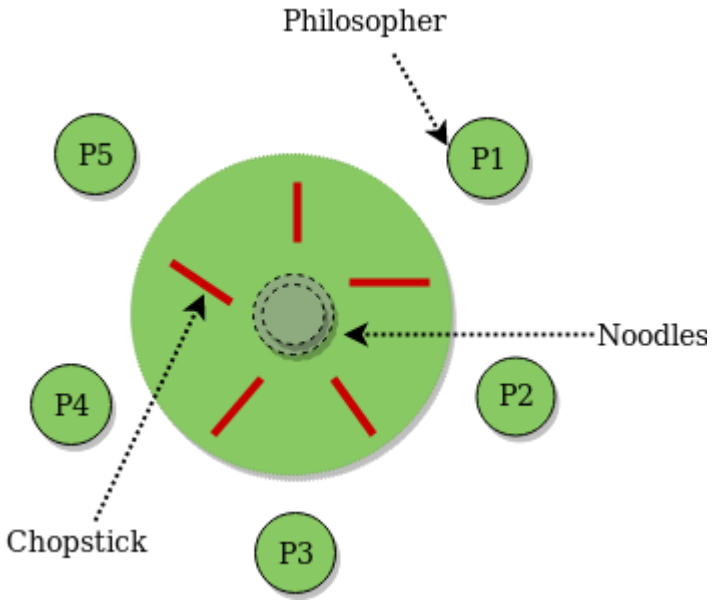
1. Bounded-buffer (or Producer-Consumer) Problem
2. Dining-Philosophers Problem
3. Readers and Writers Problem

1.**Bounded-buffer (or Producer-Consumer) Problem**

Bounded Buffer problem is also called producer consumer problem. This problem is generalized in terms of the Producer-Consumer problem. Solution to this problem is, creating two counting semaphores "full" and "empty" to keep track of the current number of full and empty buffers respectively. Producers produce a product and consumers consume the product, but both use of one of the containers each time.

2. **Dining-Philosophers Problem**

The Dining Philosopher Problem states that 'K' philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both. This problem involves the allocation of limited resources to a group of processes in a deadlock-free and starvation-free manner.



3.**Readers and Writers Problem:**

Suppose that a database is to be shared among several concurrent processes. Some of these processes may want only to read the database, whereas others may want to update (that is, to read and write) the database. We distinguish between these two types of processes by referring to the former as readers and to the latter as writers. Precisely in OS we call this situation as the readers-writers problem. Problem parameters:

- One set of data is shared among a number of processes.
- Once a writer is ready, it performs its write. Only one writer may write at a time.
- If a process is writing, no other process can read it.
- If at least one reader is reading, no other process can write.
- Readers may not write and only read.

9).Monitors

Monitors are a synchronization construct that were created to overcome the problems caused by semaphores such as timing errors. Monitors are abstract data types and contain shared data variables and procedures. The shared data variables cannot be directly accessed by a process and procedures are required to allow a single process to access the shared data variables at a time.This is demonstrated as follows:

```
monitor monitorName
{
   data variables;

   Procedure P1(....)
   {

   }

   Procedure P2(....)
   {

   }

   Procedure Pn(....)
   {

   }

   Initialization Code(....)
   {

   }
}
```

Only one process can be active in a monitor at a time. Other processes that need to access the shared variables in a monitor have to line up in a queue and are only provided access when the previous process releases the shared variables.

10).Atomic Transactions

The mutual exclusion of critical sections ensures that the critical sections are executed atomically. That is, if two critical sections are executed concurrently, the result is equivalent to their sequential execution in some unknown order. Although this property is useful in many application domains, in many cases we would like to make sure that a critical section forms a single logical unit of work that either is performed in its entirety or is not performed at all. An example is funds transfer, in which one

account is debited and another is credited. Clearly, it is essential for data consistency either that both the credit and debit occur or that neither occurs.

Consistency of data, along with storage and retrieval of data, is a concern often associated with database systems. Recently, there has been an upsurge of interest in using database-systems techniques in operating systems. Operating systems can be viewed as manipulators of data; as such, they can benefit from the advanced techniques and models available from database research. For instance, many of the ad hoc techniques used in operating systems to manage files could be more flexible and powerful if more formal database methods were used in their place.

System Model:- A collection of instructions (or operations) that performs a single logicalfunction is called a transaction. A major issue in processing transactions is the preservation of atomicity despite the possibility of failures within the computer system.

A commit operation signifies that the transaction has terminated its execution successfully, whereas an abort operation signifies that the transaction has ended its normal execution due to some logical error or a system failure. If a terminated transaction has completed its execution successfully, it is committed; otherwise, it is aborted.

Since an aborted transaction may already have modified the data that it has accessed, the state of these data may not be the same as it would have been if the transaction had executed atomically. So that atomicity is ensured, an aborted transaction must have no effect on the state of the data that it has already modified. Thus, the state of the data accessed by an aborted transaction must be restored to what it was just before the transaction started executing. We say that such a transaction has been rolled back. It is part of the responsibility of the system to ensure this property.

To determine how the system should ensure atomicity, we need first to identify the properties of devices used for storing the various data accessed by the transactions. Various types of storage media are distinguished by their relative speed, capacity, and resilience to failure.

• **Volatile storage**:- Information residing in volatile storage does not usually survive system crashes. Examples of such storage are main and cache memory. Access to volatile storage is extremely fast, both because of the speed of the memory access itself and because it is possible to access directly any data item in volatile storage.

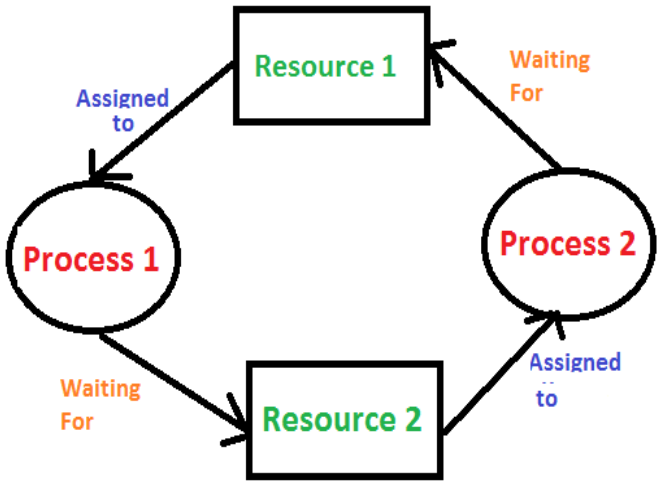
•**Non-volatile storage:-** Information residing in non-volatile storage usually survives system crashes. Examples of media for such storage are disks and magnetic tapes. Disks are more reliable than main memory but less reliable than magnetic tapes. Both disks and tapes, however, are subject to failure, which may result in loss of information. Currently, non-volatile storage is slower than volatile storage by several orders of magnitude, because disk and tape devices are electromechanical and require physical motion to access data.

• **Stable storage**:- Information residing in stable storage is never lost (never should be taken with a grain of salt, since theoretically such absolutes cannot be guaranteed). To implement an approximation of such storage, we need to replicate information in several non-volatile storage caches (usually disk) with independent failure modes and to update the information in a controlled manner.

11).Deadlocks and its characteristics

A process in operating systems uses different resources and uses resources in the following way.
1) Requests a resource
2) Use the resource
3) Releases the resource

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



Deadlock can arise if the **following four conditions hold simultaneously (Necessary Conditions).**

1. **Mutual Exclusion:-** One or more than one resource are non-shareable (Only one process can use at a time)
2. **Hold and Wait:-** A process is holding at least one resource and waiting for resources.
3. **No Preemption:-** A resource cannot be taken from a process unless the process releases the resource.
4. **Circular Wait:-** A set of processes are waiting for each other in circular form.

12).Deadlock handling techniques (or) Methods to handle deadlocks.

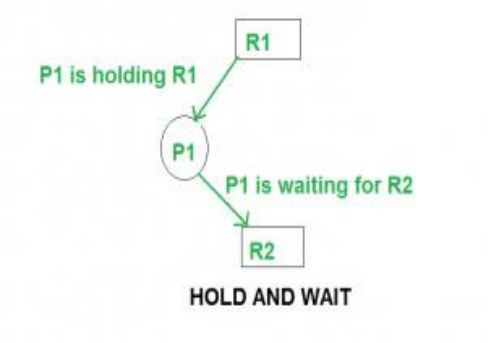
Deadlocks can be handled by using the following three techniques.

1. **Deadlock Prevention**
2. **Deadlock Avoidance**
3. **Deadlock Detection and Recovery**

1. **Deadlock Prevention :-**

We can prevent Deadlock by eliminating any of the following four conditions.

- **Eliminate Mutual Exclusion :-**It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.
- **Eliminate Hold and wait** :-Allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization. for example, if a process requires printer at a later time and we have allocated printer before the start of its execution printer will remain blocked till it has completed its execution. The process will make a new request for resources after releasing the current set of resources. This solution may lead to starvation.



- **Eliminate No Preemption** :-Preempt resources from the process when resources required by other high priority processes.
- **Eliminate Circular Wait** :-Each resource will be assigned with a numerical number. A process can request the resources increasing/decreasing. order of numbering.
  For Example, if P1 process is allocated R5 resources, now next time if P1 ask for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.

2.<u>Deadlock Avoidance</u> :-

Deadlock avoidance can be done with Banker's Algorithm.

Banker's Algorithm :- Bankers's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it checks for the safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it doesn't allow the request made by the process.

Inputs to Banker's Algorithm :-

1. Max need of resources by each process.
2. Currently, allocated resources by each process.
3. Max free available resources in the system.

The request will only be granted under the below condition :-

1. If the request made by the process is less than equal to max need to that process.
2. If the request made by the process is less than equal to the freely available resource in the system.

Example:

Total resources in system:

A	B	C	D
6	5	7	6

Available system resources are:

A	B	C	D
3	1	1	2

Processes (currently allocated resources):

	A	B	C	D
P1	1	2	2	1
P2	1	0	3	3
P3	1	2	1	0

Processes (maximum resources):

	A	B	C	D
P1	3	3	2	2
P2	1	2	3	4
P3	1	3	5	0

Need = maximum resources - currently allocated resources.

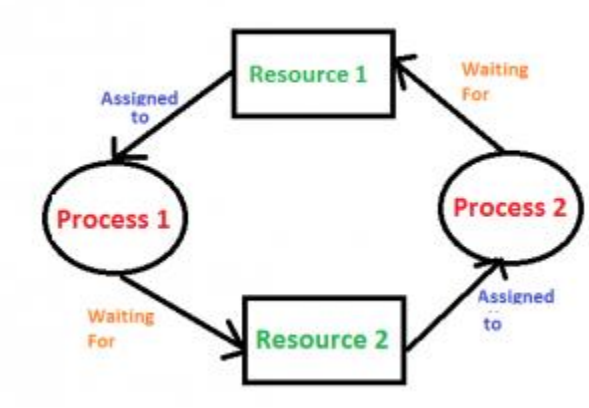
Processes (need resources):

	A	B	C	D
P1	2	1	0	1
P2	0	2	0	1
P3	0	1	4	0

3.Deadlock Detection And Recovery :-

Deadlock Detection:-

1. If resources have single instance :-
   In this case for Deadlock detection we can run an algorithm to check for cycle in the Resource Allocation Graph. Presence of cycle in the graph is the sufficient condition for deadlock.



   In the above diagram, resource 1 and resource 2 have single instances. There is a cycle R1 → P1 → R2 → P2. So, Deadlock is Confirmed.

2. If there are multiple instances of resources:-
   Detection of the cycle is necessary but not sufficient condition for deadlock detection, in this case, the system may or may not be in deadlock varies according to different situations.

Deadlock Recovery:-

A traditional operating system such as Windows doesn't deal with deadlock recovery as it is time and space consuming process. Real-time operating systems use Deadlock recovery. The following are the two different deadlock r**ecovery methods:**

1. **Killing the process :-** killing all the process involved in the deadlock. Killing process one by one. After killing each process check for deadlock again keep repeating the process till system recover from deadlock.
2. **Resource Preemption :-** Resources are preempted from the processes involved in the deadlock, preempted resources are allocated to other processes so that there is a possibility of recovering the system from deadlock. In this case, the system goes into starvation.

Short questions:

1.Explain about CPU scheduling in Operating system?

2.Explain First come first serve algorithm with an example?

3.Explain about Round robin Scheduling?

4.Explain about Monitors?

Long Questions:

1.Explain about Deadlock and its Characteristics?

2.Explain about deadlock avoidance and its prevention?

3.Explain about process Synchronization?

4.Explain about semaphores and types of semaphores?

UNIT-III

1). Introduction to Memory Management

Address uniquely identifies a location in the memory. We have two types of addresses that are logical address and physical address. The logical address is a virtual address and can be viewed by the user. The user can't view the physical address directly.

Logical Address:-

Address generated by **CPU** while a program is running is referred as **Logical Address**. The logical address is virtual as it does not exist physically. Hence, it is also called as **Virtual Address**. This address is used as a reference to access the physical memory location. The set of all logical addresses generated by a programs perspective is called **Logical Address Space**. The logical address is mapped to its corresponding physical address by a hardware device called **Memory-Management Unit(MMU)**.

Physical Address:-

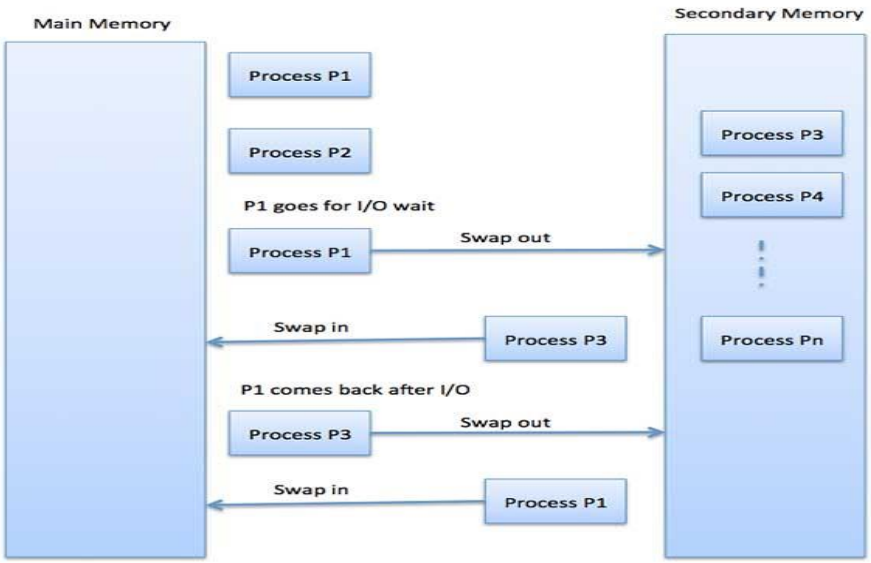
Physical Address identifies a physical location in a memory. MMU computes the physical address for the corresponding logical address. The user never deals with the physical address. Instead, the physical address is accessed by its corresponding logical address by the user. The user program generates the logical address and thinks that the program is running in this logical address. But the program needs physical memory for its execution. Hence, the logical address must be mapped to the physical address before they are used.

The logical address is mapped to the physical address using a hardware called **Memory-Management Unit**. The set of all physical addresses corresponding to the logical addresses in a Logical address space is called **Physical Address Space**.

2). Swapping

Swapping is a mechanism in which a process can be swapped(or moved) temporarily out of main memory 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

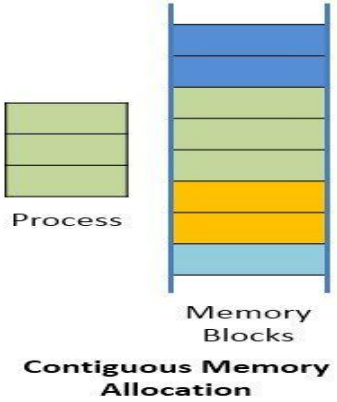
```
2048KB / 1024KB per second
= 2 seconds
= 2000 milliseconds.
```

3). Contiguous Memory Allocation

The operating system and the user's processes both must be accommodated in the main memory. Hence the main memory is **divided into two** partitions: at one partition the operating system resides and at other the user processes reside. In usual

conditions, the several user processes must reside in the memory at the same time, and therefore, it is important to consider the allocation of memory to the processes.

The Contiguous memory allocation is one of the methods of memory allocation. In contiguous memory allocation, when a process requests for the memory, a **single contiguous section of memory blocks** is assigned to the process according to its requirement.



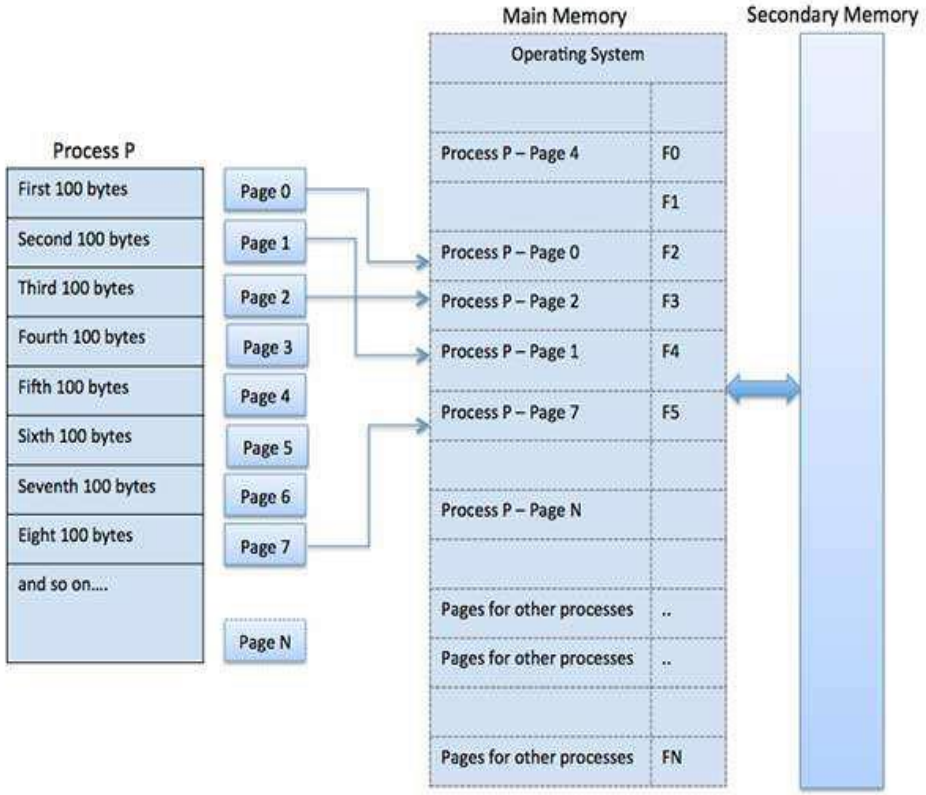
Contiguous Memory Allocation

The contiguous memory allocation can be achieved by dividing the memory into the **fixed-sized partition** and allocate each partition to a single process only. But this will cause the degree of multiprogramming, bounding to the number of fixed partition done in the memory. The contiguous memory allocation also leads to the **internal fragmentation**. Like, if a fixed sized memory block allocated to a process is slightly larger than its requirement then the left over memory space in the block is called internal fragmentation. When the process residing in the partition terminates the partition becomes available for another process.

In the variable partitioning scheme, the operating system maintains a **table** which indicates, which partition of the memory is free and which occupied by the processes.

4). Paging

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. 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:-

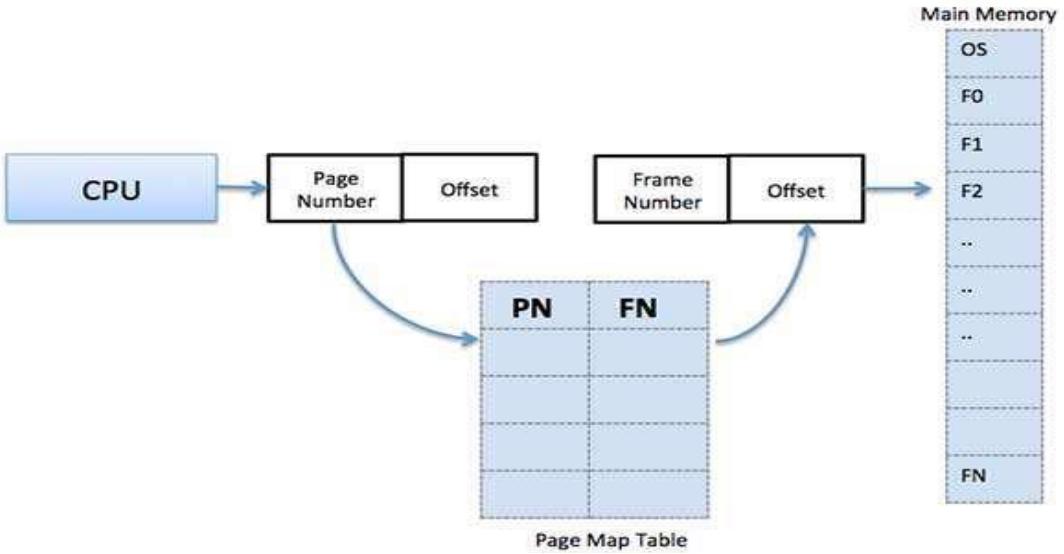
Page address is called **logical address** and represented by **page number** and the **offset**.

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 + frame 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:-

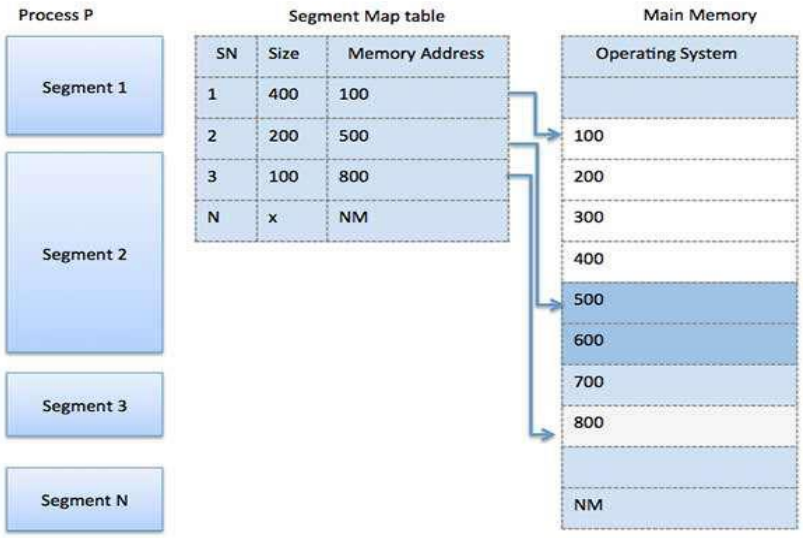
- Paging reduces external fragmentation, but still suffers from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not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

small RAM.

5). 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.



6). Virtual memory

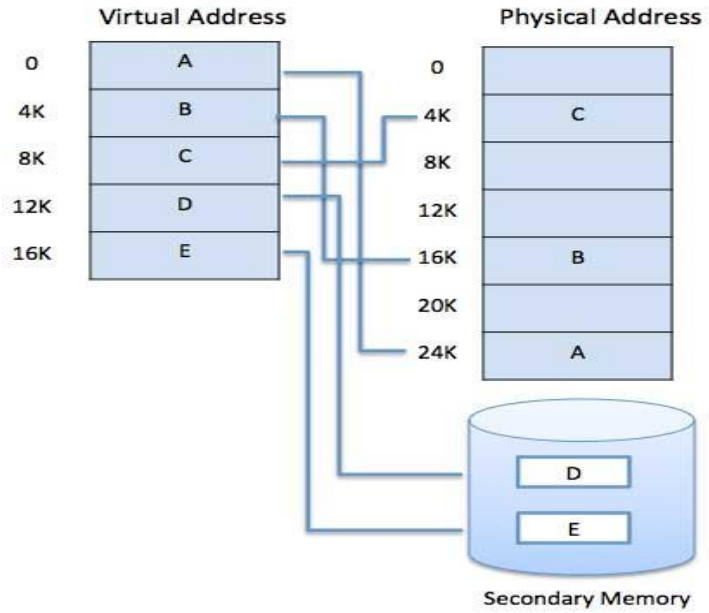
A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.

The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when entire program is not required to be loaded fully in main memory.

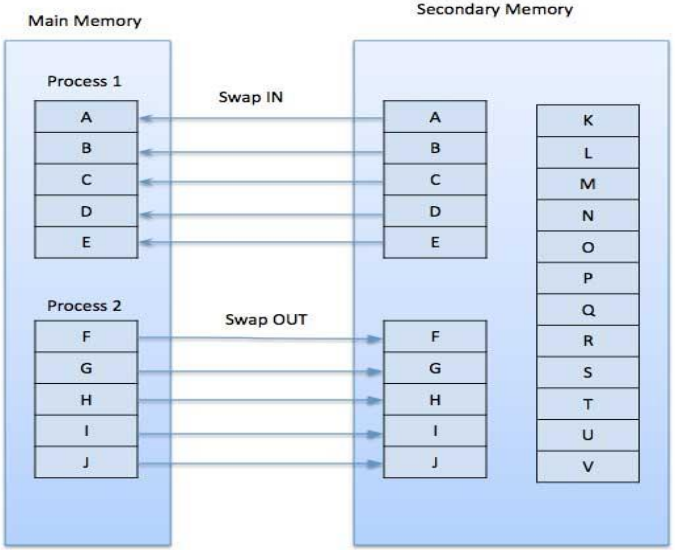
- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available.
- Each user program could take less physical memory; more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below:



Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

7).



Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. a context switch occurs, the operating system does not copy any old program's pages out to the disk or any of the new program's into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's as they are referenced.

When
of the
pages

pages

While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Advantages :- Following are the advantages of Demand Paging :

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages :-

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

8). Page Replacement Algorithms

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in.

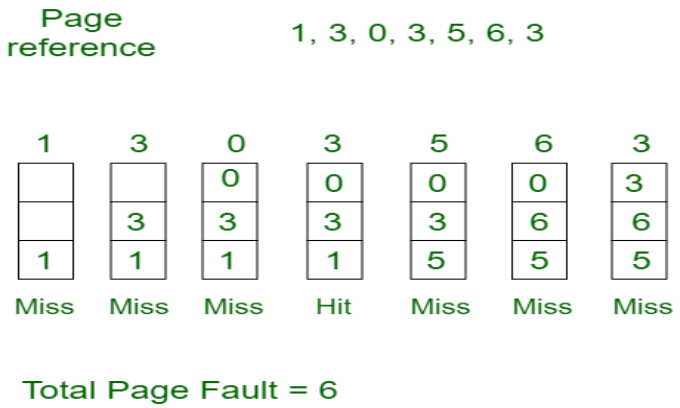
Page Fault:– A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

Page Replacement Algorithms:- The following are the different page replacement algorithms.

1.**First In First Out (FIFO): –** This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced then the page in the front of the queue is selected for removal.

For example, consider page reference string 1, 3, 0, 3, 5, 6 with 3 page frames and how many number of page faults will occur is given below.

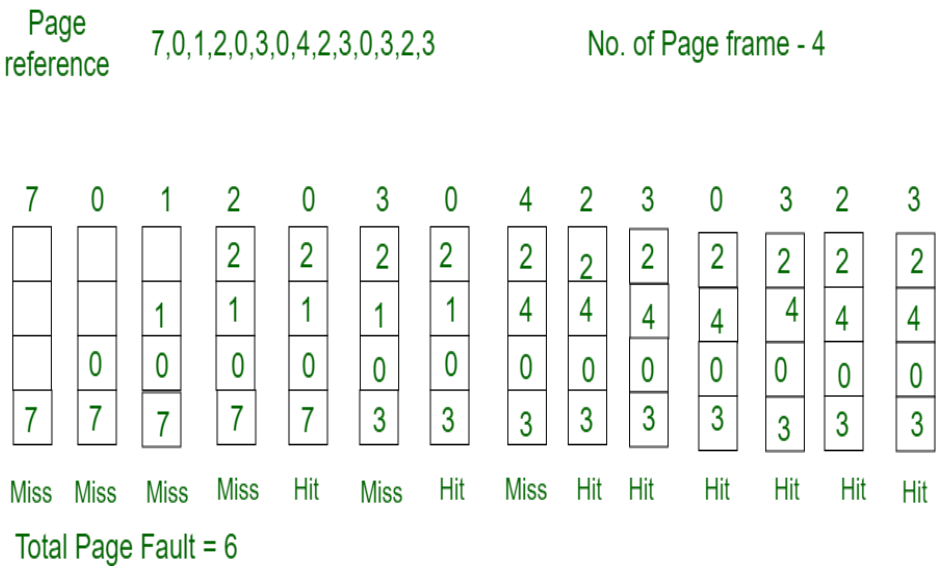


Initially all slots are empty, so when 1,3,0 came they are allocated to the empty slots —>**3 Page Faults.**
when 3 comes, it is already in memory so —> **0 Page Faults.**
Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.**
6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 Page Fault.**
Finally when 3 come it is not available so it replaces 0 —> **1 page fault.**

2.**Optimal Page replacement**:**–** In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

For example, consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frames and how many number of page faults will occur is given below.

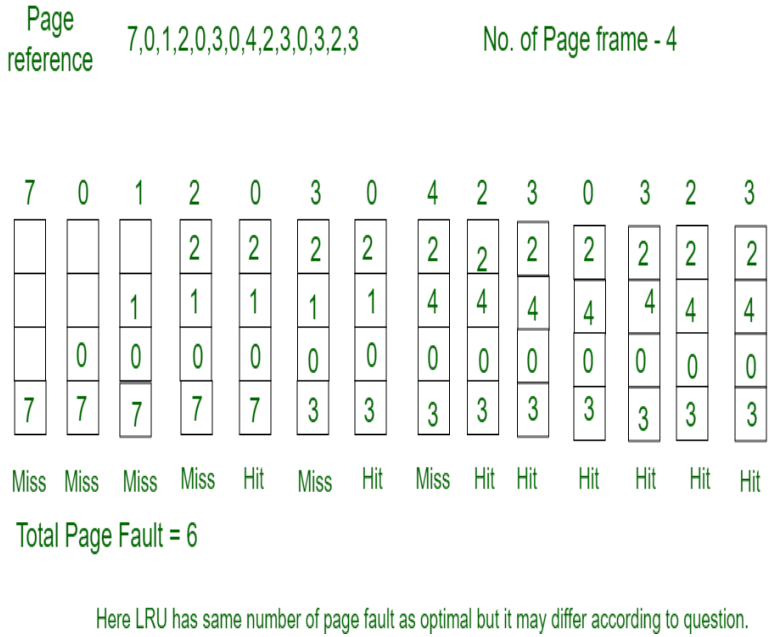
.



Initially all slots are empty, so when 7,0,1,2 are allocated to the empty slots —> **4 Page faults**
0 is already there so —> **0 Page fault.**
when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.**
0 is already there so —> **0 Page fault.**.
4 will takes place of 1 —> **1 Page Fault.**

Now for the further page reference string —> **0 Page fault** because they are already available in the memory.
Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests.

3.**Least Recently Used:–** In this algorithm page will be replaced which is least recently used.
For example, consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames and how many number of page faults will occur is given below.



Here LRU has same number of page fault as optimal but it may differ according to question.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already their so —> **0 Page fault.**
when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**
0 is already in memory so —> **0 Page fault**.
4 will takes place of 1 —> **1 Page Fault**
Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

9). Allocation of Frames

An important aspect of operating systems, virtual memory is implemented using demand paging. Demand paging necessitates the development of a page-replacement algorithm and a **frame allocation algorithm**. Frame allocation algorithms are used if we have multiple processes; it helps to decide how many frames to allocate to each process.

Frame allocation algorithms :– The two algorithms commonly used to allocate frames to a process are:

1. **Equal allocation:-** In a system with x frames and y processes, each process gets equal number of frames, i.e. x/y. For instance, if the system has 48 frames and 9 processes, each process will get 5 frames. The three frames which are not allocated to any process can be used as a free-frame buffer pool.

   Disadvantage:- In systems with processes of varying sizes, it does not make much sense to give each process equal frames. Allocation of a large number of frames to a small process will eventually lead to the wastage of a large number of allocated unused frames.

2. **Proportional allocation:-** Frames are allocated to each process according to the process size.
   For a process p_i of size s_i, the number of allocated frames is $\mathbf{a_i = (s_i/S)*m}$, where S is the sum of the sizes of all the processes and m is the number of frames in the system. For instance, in a system with 62 frames, if there is a process of 10KB and another process of 127KB, then the first process will be allocated (10/137)*62 = 4 frames and the other process will get (127/137)*62 = 57 frames.

   Advantage:-All the processes share the available frames according to their needs, rather than equally.

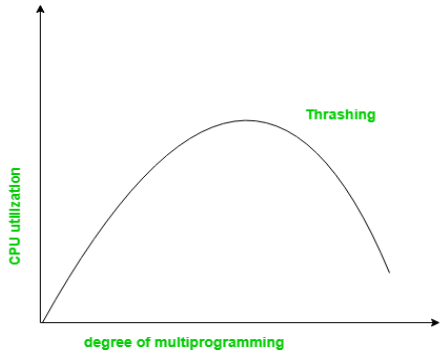
Global vs Local Allocation: –

The number of frames allocated to a process can also dynamically change depending on whether you have used **global replacement** or **local replacement** for replacing pages in case of a page fault.

1. **Local replacement:** When a process needs a page which is not in the memory, it can bring in the new page and allocate it a frame from its own set of allocated frames only.
2. **Global replacement:** When a process needs a page which is not in the memory, it can bring in the new page and allocate it a frame from the set of all frames, even if that frame is currently allocated to some other process; that is, one process can take a frame from another.

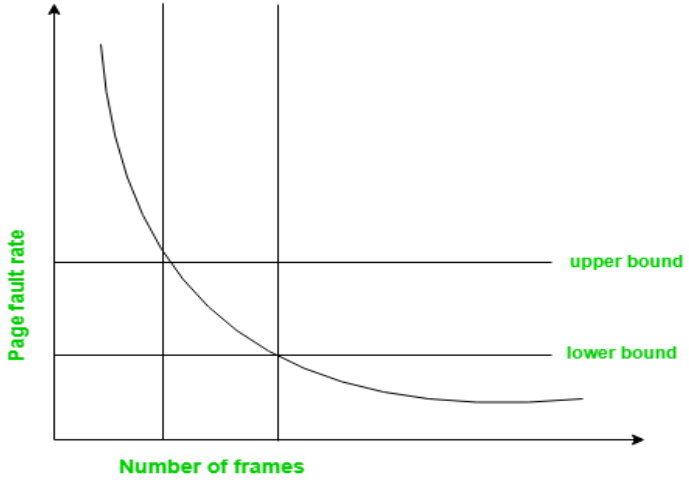
10).Thrashing

Thrashing is a condition or a situation when the system is spending a major portion of its time in servicing the page faults, but the actual processing done is very negligible.



The basic concept involved is that if a process is allocated too few frames, then there will be too many and too frequent page faults. As a result, no useful work would be done by the CPU and the CPU utilization would fall drastically. The long-term scheduler would then try to improve the CPU utilization by loading some more processes into the memory thereby increasing the degree of multi programming.

A more direct approach to handle thrashing is the one that uses Page-Fault Frequency concept.



The problem associated with Thrashing is the high page fault rate and thus, the concept here is to control the page fault rate. If the page fault rate is too high, it indicates that the process has too few frames allocated to it. On the contrary, a low page fault rate indicates that the process has too many frames.

Upper and lower limits can be established on the desired page fault rate as shown in the diagram.
If the page fault rate falls below the lower limit, frames can be removed from the process. Similarly, if the page faults rate exceeds the upper limit, more number of frames can be allocated to the process. If the page fault rate is high with no free frames, then some of the processes can be suspended and frames allocated to them can be reallocated to other processes. The suspended processes can then be restarted later.

6). Introduction to files or concept of a file.

A **file** is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. A **File Structure** should be according to a required format that the operating system can understand.

- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.
- An object file is a sequence of bytes organized into blocks that are understandable by the machine.
- When operating system defines different file structures, it also contains the code to support these file structure. Unix, MS-DOS support minimum number of file structure.

Types of Files :-Operating system like MS-DOS and UNIX have the following types of files:

Ordinary files

- These are the files that contain user information.
- These may have text, databases or executable program.
- The user can apply various operations on such files like add, modify, delete or even remove the entire file.

Directory files

- These files contain list of file names and other information related to these files.

Special files

- These files are also known as device files.
- These files represent physical device like disks, terminals, printers, networks, tape drive etc.

These files are of two types :

- **Character special files** :− data is handled character by character as in case of terminals or printers.
- **Block special files** :− data is handled in blocks as in the case of disks and tapes.

7). File Access Mechanisms

File access mechanism refers to the manner in which the records of a file may be accessed. The following are the different ways to access files:

- Sequential access
- Direct/Random access
- Indexed sequential access

Sequential access :-

A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other. This access method is the most primitive one. Example: Compilers usually access files in this fashion.

Direct/Random access :-

- Random access file organization provides, accessing the records directly.
- Each record has its own address on the file by the help of which it can be directly accessed for reading or writing.
- The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.

Indexed sequential access :-

- This mechanism is built up on base of sequential access.

- An index is created for each file which contains pointers to various blocks.
- Index is searched sequentially and its pointer is used to access the file directly.

8). Directory Structure or File Directories

Collection of files is a file directory. The directory contains information about the files, including attributes, location and ownership. Much of this information, especially that is concerned with storage, is managed by the operating system. The directory is itself a file, accessible by various file management routines.

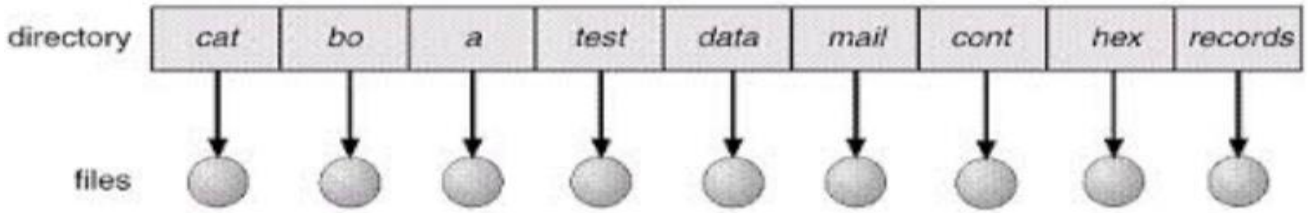
Information contained in a device directory are:

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed
- Date last updated
- Owner id
- Protection information

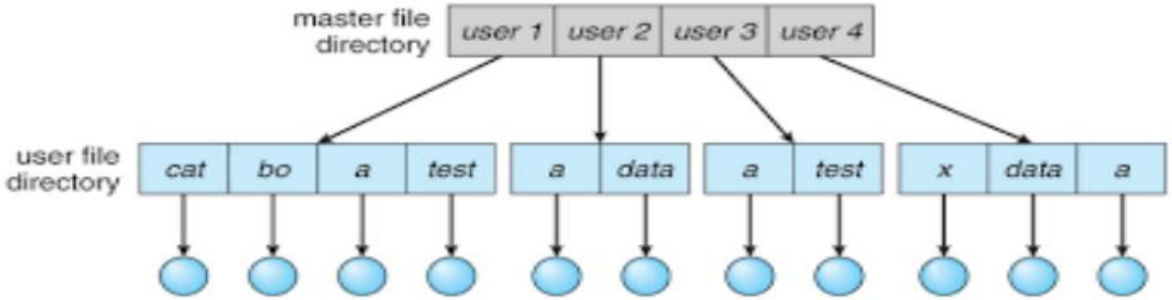
Operations performed on directory are:

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

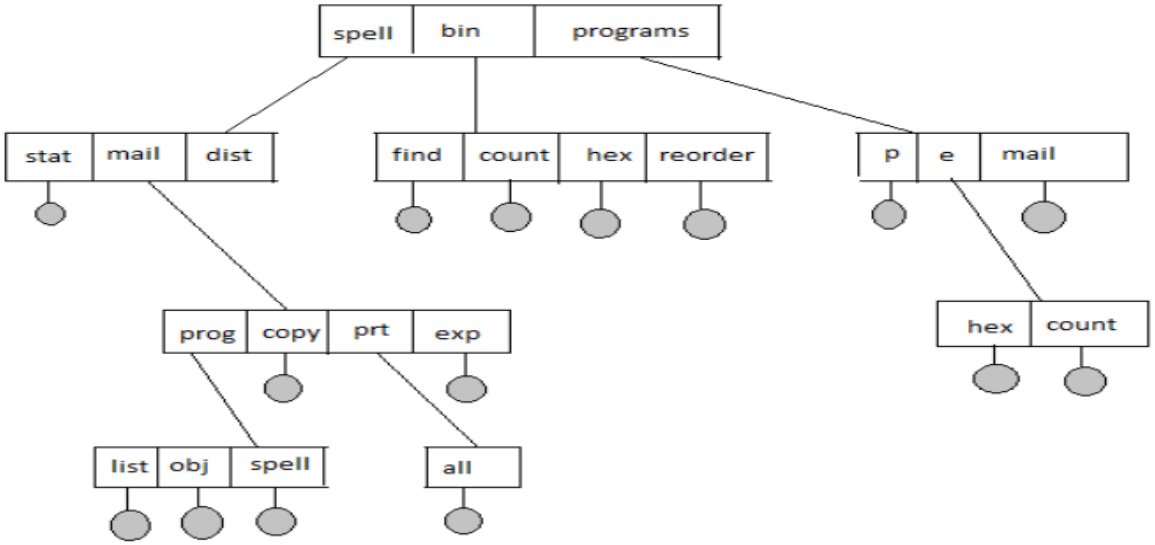
SINGLE-LEVEL DIRECTORY:- In this a single directory is maintained for all the users.



TWO-LEVEL DIRECTORY:- In this separate directories for each user is maintained.



TREE-STRUCTURED DIRECTORY :- Directory is maintained in the form of a tree. Searching is efficient and also there is grouping capability.



9). File Sharing and Protection.

Sharing of files on multi-user systems is desirable. Sharing may be done through a protection scheme. On distributed systems, files may be shared across a network. Network File System (NFS) is a common distributed file-sharing method. **User Ids** are used to identify users, allowing permissions and protections to be per-user. **Group IDs** allow users to be in groups, permitting group access rights.

In computer systems, a lot of user's information is stored, the objective of the operating system is to keep safe the data of the user from the improper access to the system. **Protection** can be provided in number of ways. For a single laptop system, we might provide protection by locking the computer in a desk drawer or file cabinet. For multi-user systems, different mechanisms are used for the protection.

Types of Access :-The files which have direct access of the any user have the need of protection. The files which are not accessible to other users doesn't require any kind of protection. The mechanism of the protection provide the facility of the controlled access by just limiting the types of access to the file. Access can be given or not given to any user depends on several factors, one of which is the type of access required. Several different types of operations can be controlled:

- **Read :–** Reading from a file.

- **Write:–** Writing or rewriting the file.
- **Execute:–**Loading the file and after loading the execution process starts.
- **Append:–**Writing the new information to the already existing file, editing must be end at the end of the existing file.
- **Delete :–**Deleting the file which is of no use and using its space for the another data.
- **List :–**List the name and attributes of the file.

Operations like renaming, editing the existing file, copying; these can also be controlled.

Short Questions:

1. **Write a short note on memory management?**
2. **Explain about paging?**
3. **Explain about virtual memory?**
4. **Explain about Thrashing?**

Long Questions:

1.Explain about page replacement Algorithms?

2.Describe about Directory structure?

3.Explain about file sharing and file protection?

4. Explain about demand paging?

UNIT – IV

1). File-System Structure.

A file is a collection of related information. The file system resides on secondary storage and provides an efficient and convenient access to the disk by allowing data to be stored, located, and retrieved. File system organized into many layers which is shown in the following figure:

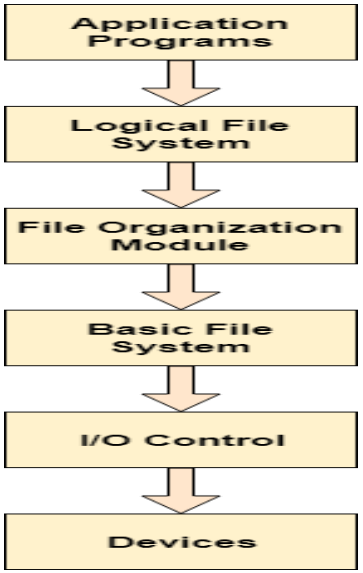


Fig : File System Structure

- **I/O Control :–** Device drivers acts as interface between devices and OS, they help to transfer data between disk and main memory.
- **Basic file system:–** It issues general commands to device drivers to read and write physical blocks on disk. It manages the memory buffers and caches. A block in buffer can hold the contents of the disk block and cache stores frequently used file system metadata.

- **File organization Module:–**It has information about files, location of files and their logical and physical blocks. Physical blocks do not match with logical numbers of logical block numbered from 0 to N. It also has a free space which tracks unallocated blocks.
- **Logical file system:–**It manages metadata about a file i.e. includes all details about a file except the actual contents of file. It also maintains details via file control blocks. File control block (FCB) has information about a file – owner, size, permissions, and location of file contents.

2). File-System Implementation

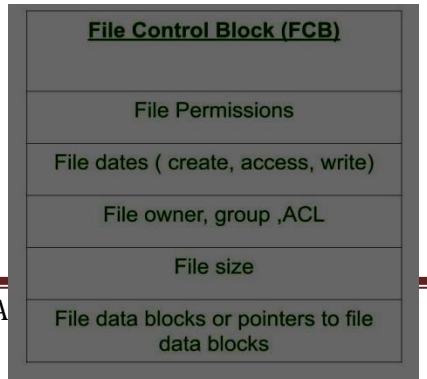
File system can be implemented by using following two data structures:

1. On-disk Structures
2. In-Memory Structure

1. On-disk Structures:–

Generally they contain information about total number of disk blocks, free disk blocks, location of them and etc. Below given are different on-disk structures:

- **Boot Control Block:–** It is usually the first block of volume and it contains information needed to boot an operating system. In UNIX it is called boot block and in NTFS it is called as partition boot sector.
- **Volume Control Block:–** It has information about a particular partition ex:- free block count, block size and block pointers etc. In UNIX it is called super block and in NTFS it is stored in master file table.
- **Directory Structure:–** They store file names and associated inode numbers. In UNIX,it includes file names and associated file names and in NTFS, it is stored in master file table.
- **Per-File FCB:–** It contains details about files and it has a unique identifier number to allow association with directory entry. In NTFS it is stored in master file table.

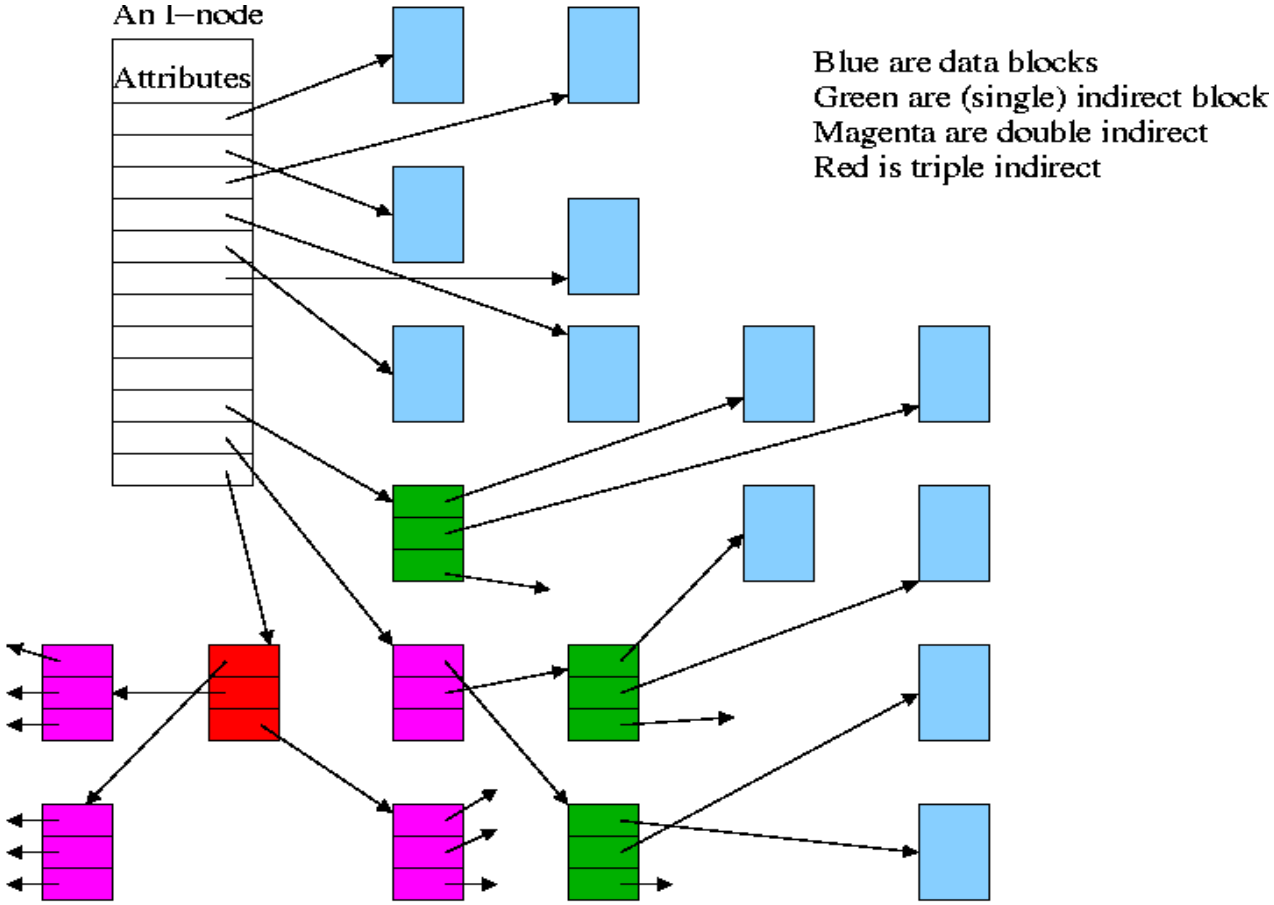


2. <u>In-Memory Structure</u>:- They are maintained in main-memory and these are helpful for file system management for caching. Several in-memory structures given below :

- **Mount Table:–** It contains information about each mounted volume.
- **Directory-Structure cache:–**This cache holds the directory information of recently accessed directories.
- **System wide open-file table:–**It contains the copy of FCB of each open file.
- **Per-process open-file table:–**It contains information opened by that particular process and it maps with appropriate system wide open-file.

3). Directory implementation.

The selection of directory-allocation and directory-management algorithms significantly affects the efficiency, performance, and reliability of the file system.



Linear List :-

- The simplest method of implementing a directory is to use a linear list of file names with pointers to the data blocks. This method is simple to program but time-consuming to execute. To create a new file., we must first search the directory to be sure that no existing file has the same name. Then, we add a new entry at the end of the directory. To delete a file, we search the directory for the named file, then release the space allocated to it.
- To reuse the directory entry, we can do one of several things. We can mark the entry as unused (by assigning it a special name), or we can attach it to a list of free directory entries. A third alternative is to copy the last entry in the directory into

the freed location and to decrease the length of the directory. A linked list can also be used to decrease the time required to delete a file.

- The real disadvantage of a linear list of directory entries is that finding a file requires a linear search. Directory information is used frequently, and users will notice if access to it is slow. In fact, many operating systems implement a software cache to store the most recently used directory information. A cache hit avoids the need to constantly reread the information from disk. A sorted list allows a binary search and decreases the average search time.
- However, the requirement that the list be kept sorted may complicate creating and deleting files, since we may have to move substantial amounts of directory information to maintain a sorted directory. A more sophisticated tree data structure, such as a B-tree, might help here. An advantage of the sorted list is that a sorted directory listing can be produced without a separate sort step.

Hash Table :-

Another data structure used for a file directory is a hash table. With this method, a linear list stores the directory entries, but a hash data structure is also used. The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list. Therefore, it can greatly decrease the directory search time. Insertion and deletion are also fairly straightforward, although some provision must be made for collisions—situations in which two file names hash to the same location.

4). File Allocation Methods.

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

The main idea behind these methods is to provide:

- Efficient disk space utilization.
- Fast access to the file blocks.

All the three methods have their own advantages and disadvantages and these are given below:

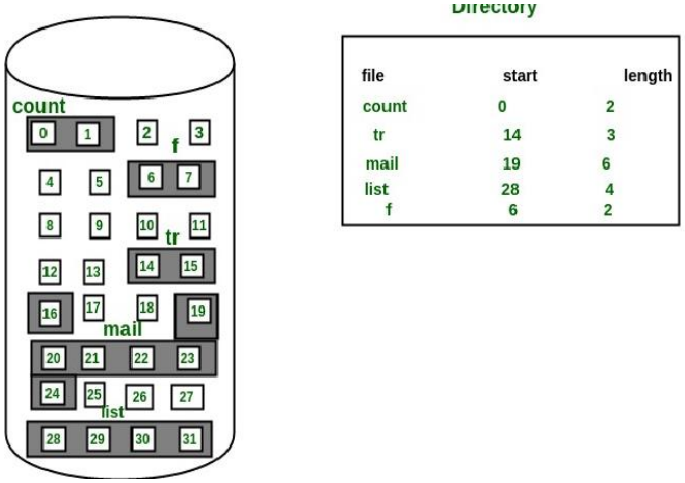
1. Contiguous Allocation:-

In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block 'b' as the starting location, then the blocks assigned to the file will be: *b, b+1, b+2,……b+n-1*. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.

The directory entry for a file with contiguous allocation contains:

- Address of starting block
- Length of the allocated portion.

The *file 'mail'* in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies *19, 20, 21, 22, 23, 24* blocks.



Advantages:

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the ‘k’ th block of the file which starts at block ‘b’ can easily be obtained as (b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

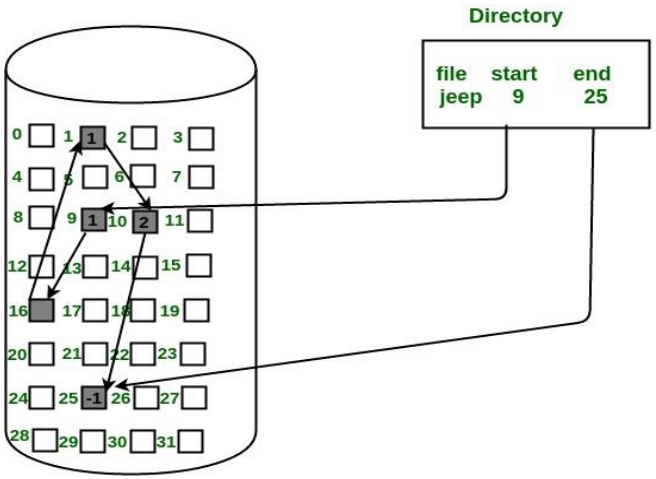
Disadvantages:

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

2. Linked List Allocation

In this scheme, each file is a linked list of disk blocks which **need not be** contiguous. The disk blocks can be scattered anywhere on the disk.

The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file. The file ‘jeep’ in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.



Advantages:

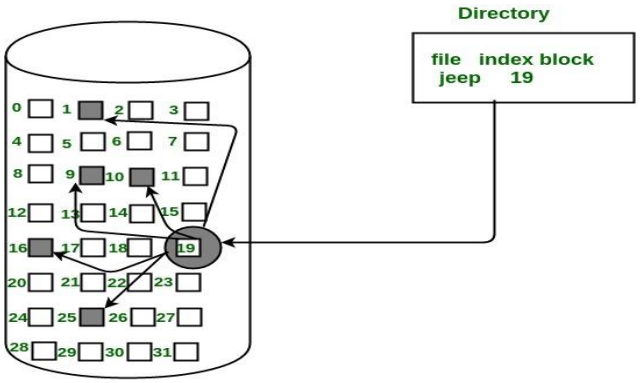
- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous block of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages:

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We cannot directly access the blocks of a file. A block ‘k’ of a file can be accessed by traversing ‘k’ blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation obtain some extra overhead.

3. Indexed Allocation

In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file. Each file has its own index block. The ‘i’ th entry in the index block contains the disk address of the ‘i’ th file block. The directory entry contains the address of the index block as shown in the image:



Advantages:

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantage:

- The pointer overhead for indexed allocation is greater than linked allocation.

5). Free space management in Operating System.

The system keeps track of the free disk blocks for allocating space to files when they are created. Also, to reuse the space released from deleting the files, free space management becomes crucial. The system maintains a free space list which keeps track of the disk blocks that are not allocated to some file or directory. The free space list can be implemented by using the following methods:

1. Bitmap or Bit vector: –

A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: *0 indicates that the block is allocated* and 1 indicates a free block.
The given instance of disk blocks on the disk in the following *Figure 1* (where green blocks are allocated) can be represented by a bitmap of 16 bits as: **0000111000000110**.

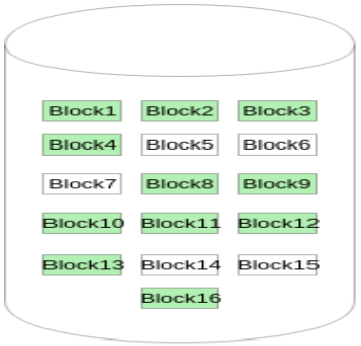


Figure - 1

Advantages:–

- Simple to understand.
- Finding the first free block is efficient. It requires scanning the words (a group of 8 bits) in a bitmap for a non-zero word. (A 0-valued word has all bits 0). The first free block is then found by scanning for the first 1 bit in the non-zero word.

2. Linked List :–

In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.

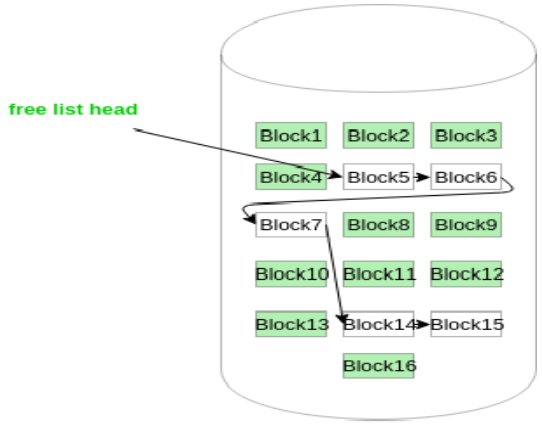


Figure - 2

In *Figure-2*, the free space list head points to Block 5 which points to Block 6, the next free block and so on. The last free block would contain a null pointer indicating the end of free list.

A drawback of this method is the I/O required for free space list traversal.

3.Grouping :–

This approach stores the address of the free blocks in the first free block. The first free block stores the address of some, say 'n' free blocks. Out of these 'n' blocks, the first 'n-1' blocks are actually free and the last block contains the address of next free 'n' blocks.

An **advantage** of this approach is that the addresses of a group of free disk blocks can be found easily.

4. Counting: –

This approach stores the address of the first free disk block and a number 'n' of free contiguous disk blocks that follow the first block. Every entry in the list would contain: Address of first free disk block and a number 'n'. For example, *in Figure-1*, the first entry of the free space list would be: ([Address of Block 5], 2), because 2 contiguous free blocks follow block 5.

6). overview of Mass Storage Structure (or) magnetic disk structure.

In modern computers, most of the mass storage or secondary storage is in the form of magnetic disks. Hence, knowing the structure of a magnetic disk is necessary to understand how the data in the disk is accessed by the computer.

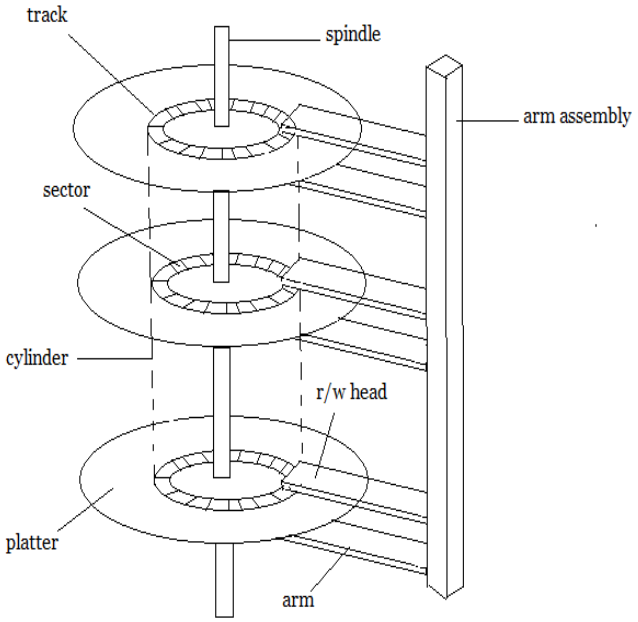


Fig : Structure of a magnetic disk

A magnetic disk contains several **platters**. Each platter is divided into circular shaped **tracks**. The length of the tracks near the centre is less than the length of the tracks farther from the centre. Each track is further divided into **sectors**, as shown in the figure.

Tracks of the same distance from centre form a cylinder. A read-write head is used to read data from a sector of the magnetic disk.

The speed of the disk is measured as two parts:

- **Transfer rate:** This is the rate at which the data moves from disk to the computer.
- **Random access time:** It is the sum of the seek time and rotational latency.

Seek time is the time taken by the arm to move to the required track. **Rotational latency** is defined as the time taken by the arm to reach the required sector in the track.

Even though the disk is arranged as sectors and tracks physically, the data is logically arranged and addressed as an array of blocks of fixed size. The size of a block can be **512** or **1024** bytes. Each logical block is mapped with a sector on the disk, sequentially. In this way, each sector in the disk will have a logical address.

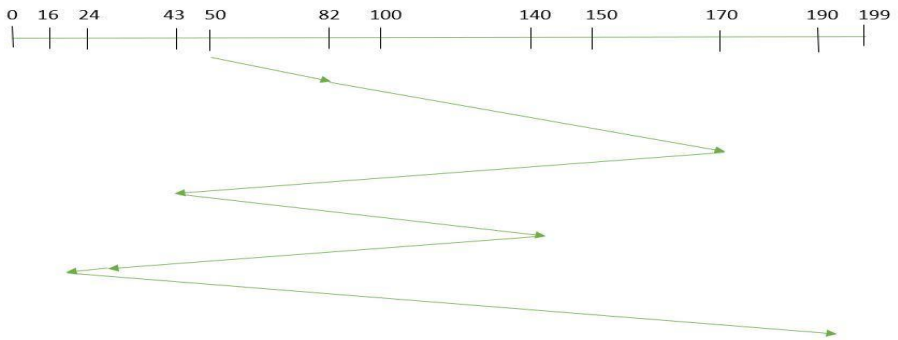
7). Disk Scheduling Algorithms.

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling. Disk scheduling is important because, multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.

The following are the different Disk Scheduling Algorithms:

1.**FCFS:**- FCFS(First Come First Serve) is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.

Example:-Suppose the order of request is : (82,170,43,140,24,16,190)
And current position of Read/Write head is : 50



So, total seek time=(82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16)
=642

Advantages:-

- Every request gets a fair chance
- No indefinite postponement

Disadvantages:-

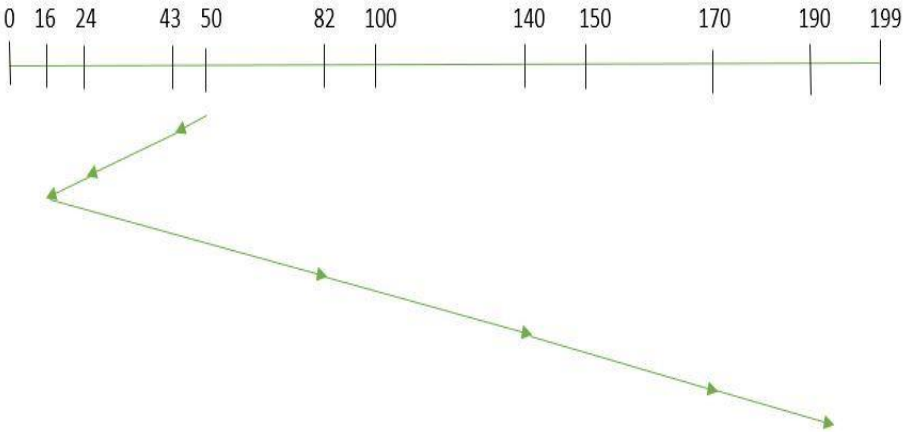
- Does not try to optimize seek time
- May not provide the best possible service

2.**SSTF:**- In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system.

Example:-

Suppose the order of request is- (82,170,43,140,24,16,190)

And current position of Read/Write head is : 50



So, total seek time=(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-40)+(190-170)

=208

Advantages:-

- Average Response Time decreases
- Throughput increases

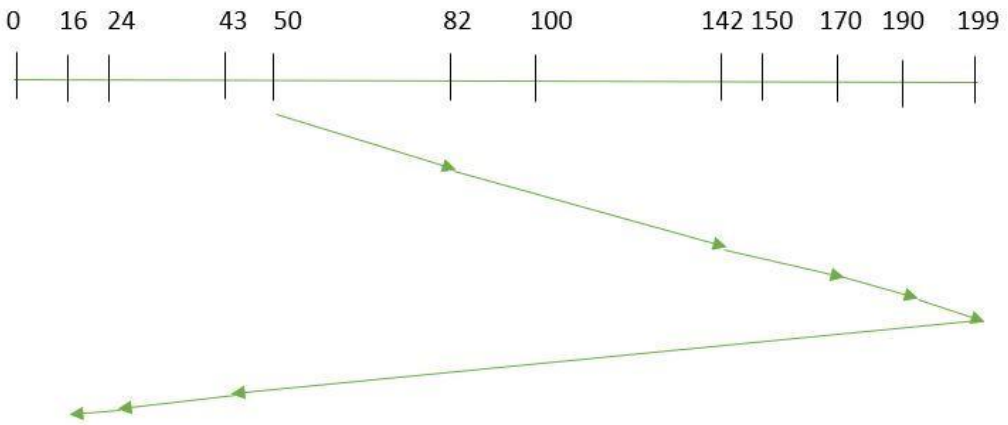
Disadvantage:-

- Overhead to calculate seek time in advance

3.**SCAN:**- In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and hence also known as **elevator algorithm.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**



So, total seek time =(199-50)+(199-16)
=332

Advantages:-

- High throughput
- Best Average response time

Disadvantage:-

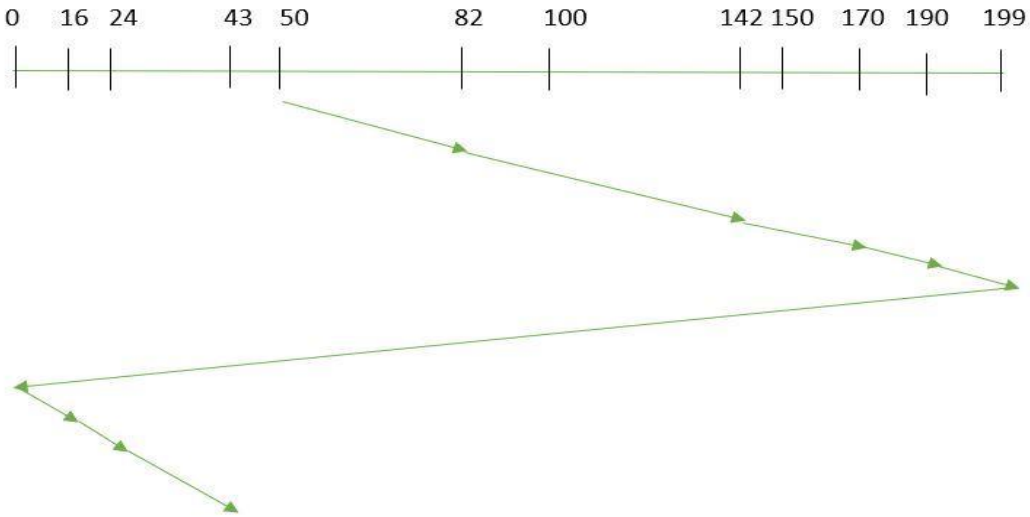
- Long waiting time for requests for locations just visited by disk arm

4.CSCAN:- In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.

These situations are avoided in *CSCAN* algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

Example:-

Suppose the requests to be addressed are : 82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**



So, total Seek time =(199-50)+(199-0)+(43-0)
=391

Advantage:-

- Provides more uniform wait time compared to SCAN

8). Disk Management.

Disk Formatting

- Before a disk can be used, it has to be ***low-level formatted***, which means laying down all of the headers and trailers marking the beginning and ends of each sector. Included in the header and trailer are the linear sector numbers, and ***error-correcting codes, ECC,*** which allow damaged sectors to not only be detected, but in many cases for the damaged data to be recovered (depending on the extent of the damage). Sector sizes are traditionally 512 bytes, but may be larger, particularly in larger drives.
- After partitioning, then the file systems must be ***logically formatted,*** which involves laying down the master directory information (FAT table or inode structure), initializing free lists, and creating at least the root directory of the file system. Disk partitions which are to be used as raw devices are not logically formatted. This saves the overhead and disk space of the file system structure, but requires that the application program to manage its own disk storage requirements.

Boot Block

- Computer ROM contains a ***bootstrap*** program (OS independent) with just enough code to find the first sector on the first hard drive on the first controller, load that sector into memory, and transfer control over to it.
- The first sector on the hard drive is known as the ***Master Boot Record, MBR,*** and contains a very small amount of code in addition to the ***partition table.*** The partition table documents how the disk is partitioned into logical disks, and indicates specifically which partition is the ***active*** or ***boot*** partition.
- The boot program then looks to the active partition to find an operating system, possibly loading up a slightly larger / more advanced boot program along the way.

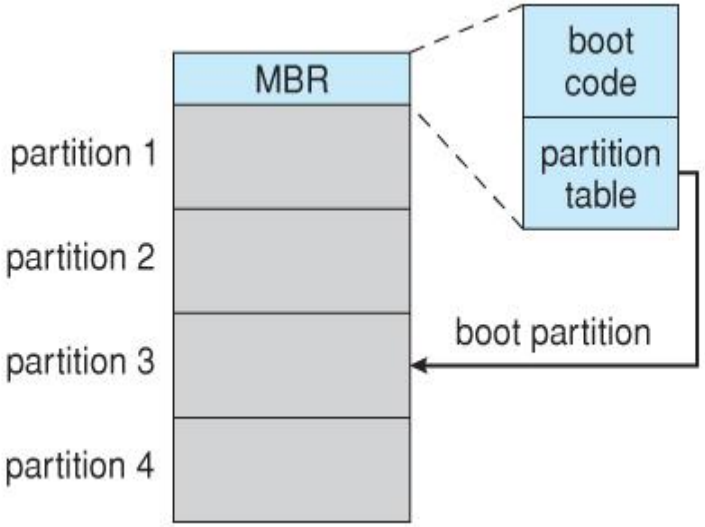


Figure : Booting from disk in Windows 2000.

Bad Blocks

- No disk can be manufactured to 100% perfection, and all physical objects wear out over time. For these reasons all disks are shipped with a few bad blocks, and additional blocks can be expected to go bad slowly over time. If a large number of blocks go bad then the entire disk will need to be replaced, but a few here and there can be handled through other means.
- In the old days, bad blocks had to be checked for manually. Formatting of the disk or running certain disk-analysis tools would identify bad blocks, and attempt to read the data off of them one last time through repeated tries. Then the bad blocks would be mapped out and taken out of future service. Sometimes the data could be recovered, and sometimes it was lost forever.
- Modern disk controllers make much better use of the error-correcting codes, so that bad blocks can be detected earlier and the data usually recovered. (Recall that blocks are tested with every write as well as with every read, so often errors can be detected before the write operation is complete, and the data simply written to a different sector instead.

9). Swap-Space Management.

- Modern systems typically swap out pages as needed, rather than swapping out entire processes. Hence the swapping system is part of the virtual memory management system.
- Managing swap space is obviously an important task for modern OS.

Swap-Space Use

- The amount of swap space needed by an OS varies greatly according to how it is used. Some systems require an amount equal to physical RAM; some want a multiple of that; some want an amount equal to the amount by which virtual memory exceeds physical RAM, and some systems use little or none at all.
- Some systems support multiple swap spaces on separate disks in order to speed up the virtual memory system.

Swap-Space Location

Swap space can be physically located in one of two locations:

- As a large file which is part of the regular file system. This is easy to implement, but inefficient. Not only must the swap space be accessed through the directory system, the file is also subject to fragmentation issues. Caching the block location helps in finding the physical blocks, but that is not a complete fix.
- As a raw partition, possibly on a separate or little-used disk. This allows the OS more control over swap space management, which is usually faster and more efficient. Fragmentation of swap space is generally not a big issue, as the space is re-initialized every time the system is rebooted. The downside of keeping swap space on a raw partition is that it can only be grown by repartitioning the hard drive.

Swap-Space Management: An Example

- Historically operating systems swapped out entire processes as needed. Modern systems swap out only individual pages, and only as needed. For example process code blocks and other blocks that have not been changed since they were originally loaded are normally just freed from the virtual memory system rather than copying them to swap space, because it is faster to go find them again in the file system and read them back in from there than to write them out to swap space and then read them back.

- In the mapping system shown below for Linux systems, a map of swap space is kept in memory, where each entry corresponds to a 4K block in the swap space. Zeros indicate free slots and non-zeros refer to how many processes have a mapping to that particular block (>1 for shared pages only.)

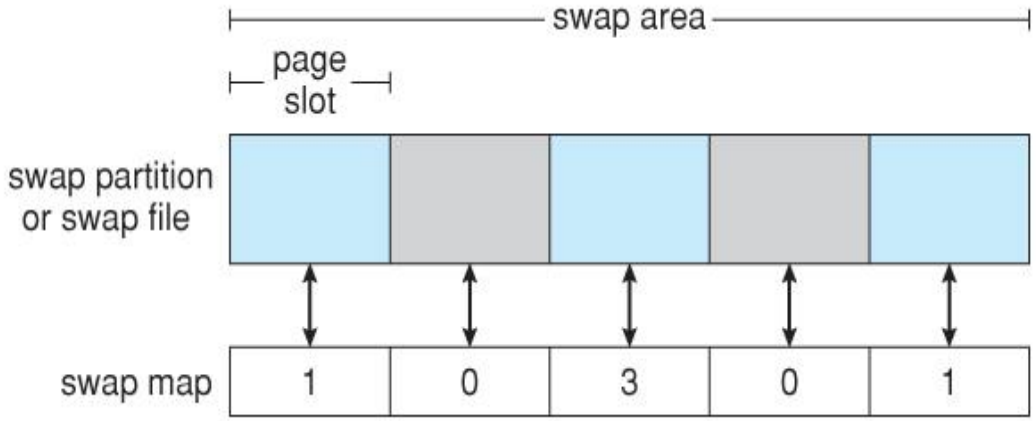


Figure : The data structures for swapping on Linux systems.

10) .RAID (Redundant Array of Independent Disks) Structure

- The general idea behind RAID is to employ a group of hard drives together with some form of duplication, either to increase reliability or to speed up operations, (or sometimes both.)
- **RAID** originally stood for **Redundant Array of Inexpensive Disks,** and was designed to use a bunch of cheap small disks in place of one or two larger more expensive ones. Today RAID systems employ large possibly expensive disks as their components, switching the definition to **Independent** disks.

Improvement of Reliability via Redundancy

- The more disks a system has, the greater the likelihood that one of them will go bad at any given time. Hence increasing disks on a system actually **decreases** the **Mean Time To Failure, MTTF** of the system.
- If, however, the same data was copied onto multiple disks, then the data would not be lost unless **both** (or all) copies of the data were damaged simultaneously, which is a **much** lower probability than for a single disk going bad. More specifically, the second disk would have to go bad before the first disk was repaired, which brings the **Mean Time To Repair** into play.

- This is the basic idea behind disk **mirroring**, in which a system contains identical data on two or more disks.

Improvement in Performance via Parallelism

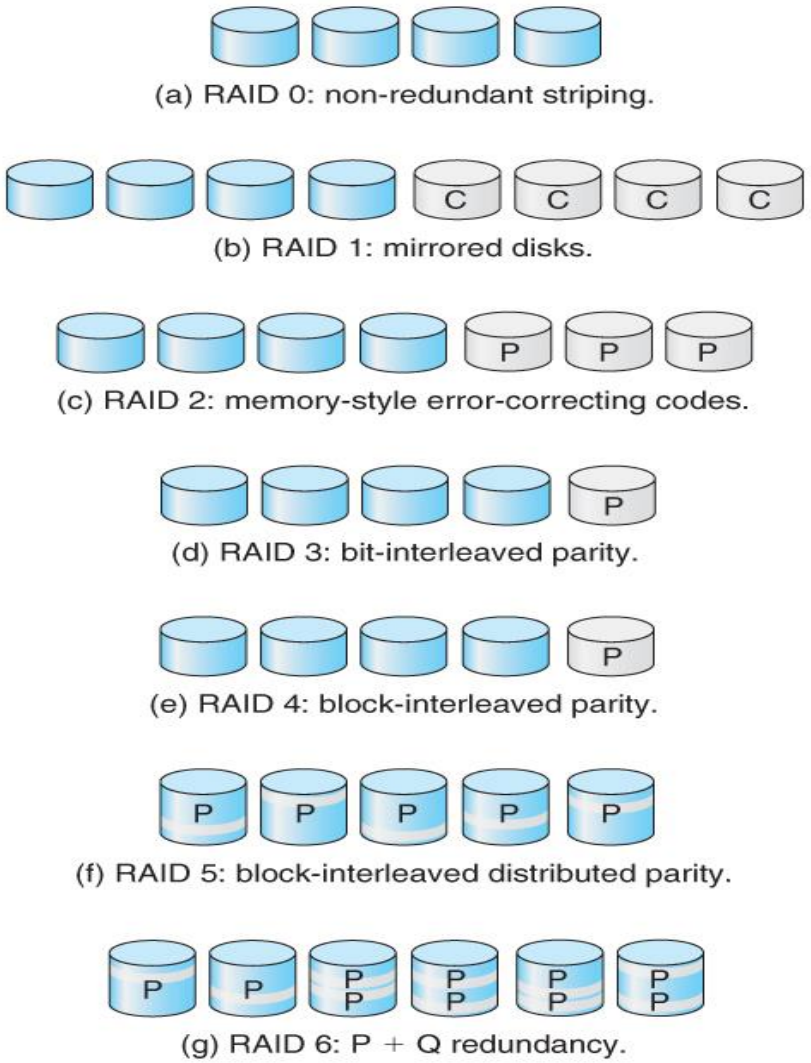
- There is also a performance benefit to mirroring, particularly with respect to reads. Since every block of data is duplicated on multiple disks, read operations can be satisfied from any available copy, and multiple disks can be reading different data blocks simultaneously in parallel.
- Another way of improving disk access time is with **striping**, which basically means spreading data out across multiple disks that can be accessed simultaneously.
    - With **bit-level striping** the bits of each byte are striped across multiple disks.
    - **Block-level striping** spreads a file system across multiple disks on a block-by-block basis, so if block N were located on disk 0, then block N + 1 would be on disk 1, and so on.

RAID Levels

- Mirroring provides reliability but is expensive; Striping improves performance, but does not improve reliability. Accordingly there are a number of different schemes that combine the principals of mirroring and striping in different ways, in order to balance reliability versus performance versus cost. These are described by different **RAID levels**, as follows: (In the diagram that follows, "C" indicates a copy, and "P" indicates parity, i.e. checksum bits.)
- **Raid Level 0 -** This level includes striping only, with no mirroring.
- **Raid Level 1 -** This level includes mirroring only, no striping.
- **Raid Level 2 -** This level stores error-correcting codes on additional disks, allowing for any damaged data to be reconstructed by subtraction from the remaining undamaged data. This scheme requires only three extra disks to protect 4 disks worth of data, as opposed to full mirroring.
- **Raid Level 3 -** This level is similar to level 2, except that it takes advantage of the fact that each disk is still doing its own error-detection, so that when an error occurs, there is no question about which disk in the array has the bad data. As a result a single parity bit is all that is needed to recover the lost data from an array of disks. Level 3 also includes striping, which improves performance. The downside with the parity approach is that every disk must take part in every disk access, and the parity bits must be constantly calculated and checked, reducing performance.
- **Raid Level 4 -** This level is similar to level 3, employing block-level striping instead of bit-level striping. The benefits are that multiple blocks can be read independently, and changes to a block only require writing two blocks (data and parity) rather than involving all disks.
- **Raid Level 5 -** This level is similar to level 4, except the parity blocks are distributed over all disks, thereby more evenly balancing the load on the system. For any given block on the disk(s), one of the disks will hold the parity information for that

block and the other N-1 disks will hold the data. But the same disk cannot hold both data and parity for the same block, as both would be lost in the event of a disk crash.

- **Raid Level 6 -** This level extends raid level 5 by storing multiple bits of error-recovery codes, for each bit position of data, rather than a single parity bit. In the example shown below 2 bits of ECC are stored for every 4 bits of data, allowing data recovery in the face of up to two simultaneous disk failures.



(a) RAID 0: non-redundant striping.

(b) RAID 1: mirrored disks.

(c) RAID 2: memory-style error-correcting codes.

(d) RAID 3: bit-interleaved parity.

(e) RAID 4: block-interleaved parity.

(f) RAID 5: block-interleaved distributed parity.

(g) RAID 6: P + Q redundancy.

Figure : RAID levels.

11). Explain about Stable-Storage Implementation.

- The concept of stable storage involves a storage medium in which data is ***never*** lost, even in the case of equipment failure in the middle of a write operation.
- To implement this requires two (or more) copies of the data, with separate failure modes.
- An attempted disk write results in one of three possible outcomes:
    1. The data is successfully and completely written.
    2. The data is partially written, but not completely. The last block written may be garbled.
    3. No writing takes place at all.
- Whenever an equipment failure occurs during a write, the system must detect it, and return the system back to a consistent state. To do this requires two physical blocks for every logical block, and the following procedure:
    1. Write the data to the first physical block.
    2. After step1 had completed, then write the data to the second physical block.
    3. Declare the operation complete only after both physical writes have completed successfully.
- During recovery the pair of blocks is examined.
    - If both blocks are identical and there is no sign of damage, then no further action is necessary.
    - If one block contains a detectable error but the other does not, then the damaged block is replaced with the good copy. This will either undo the operation or complete the operation, depending on which block is damaged and which is undamaged.
    - If neither block shows damage but the data in the blocks differ, then replace the data in the first block with the data in the second block.

12). Explain about Tertiary-Storage Devices.

Removable Disks

- Removable magnetic disks (e.g. floppies) can be nearly as fast as hard drives, but are at greater risk for damage due to scratches. Variations of removable magnetic disks up to a GB or more in capacity have been developed.

- A **magneto-optical** disk uses a magnetic disk covered in a clear plastic coating that protects the surface.
  - The heads sit a considerable distance away from the magnetic surface, and as a result do not have enough magnetic strength to switch bits at normal room temperature**.**
  - For writing, a laser is used to heat up a specific spot on the disk, to a temperature at which the weak magnetic field of the write head is able to flip the bits.
  - For reading, a laser is shined at the disk, and the Kerr effect causes the polarization of the light to become rotated either clockwise or counter-clockwise depending on the orientation of the magnetic field.
- **Optical disks** do not use magnetism at all, but instead use special materials that can be altered (by lasers) to have relatively light or dark spots.
  - For example the **phase-change disk** has a material that can be frozen into either a crystalline or an amorphous state, the latter of which is less transparent and reflects less light when a laser is bounced off a reflective surface under the material.
    - Three powers of lasers are used with phase-change disks: (1) a low power laser is used to read the disk, without affecting the materials. (2) A medium power erases the disk, by melting and re-freezing the medium into a crystalline state, and (3) a high power writes to the disk by melting the medium and re-freezing it into the amorphous state.
    - The most common examples of these disks are re-writable CD-RWs and DVD-RWs.
- An alternative to the disks described above are **Write-Once Read-Many, WORM** drives.
  - The original version of WORM drives involved a thin layer of aluminium sandwiched between two protective layers of glass or plastic.
    - Holes were burned in the aluminium to write bits.
    - Because the holes could not be filled back in, there was no way to re-write to the disk. (Although data could be erased by burning more holes.)
  - WORM drives have important legal ramifications for data that must be stored for a very long time and must be provable in court as unaltered since it was originally written. (Such as long-term storage of medical records.)
  - Modern CD-R and DVD-R disks are examples of WORM drives that use organic polymer inks instead of an aluminium layer.
- Read-only disks are similar to WORM disks, except the bits are pressed onto the disk at the factory, rather than being burned on one by one.

- Tape drives typically cost more than disk drives, but the cost per MB of the tapes themselves is lower.
- Tapes are typically used today for backups, and for enormous volumes of data stored by certain scientific establishments. For e.g. NASA's archive of space probe and satellite imagery, which is currently being downloaded from numerous sources faster than anyone can actually look at it.
- Robotic tape changers move tapes from drives to archival tape libraries upon demand.

Short Questions:

1.Explain about File-System Structure?

2.Explain about File System Implementation?

3.Explain about File allocation methods?

4.Explain about directory implementation?

Long Questions:

1.Explain about Free space management in operating system?

2.Explain about disk scheduling algorithm?

3.Explain about RAID(Redundant Array of Independent Disks) Structure?

4. Explain about Tertiary-Storage Devices.

UNIT-V

Topics:

- **User Authentication**
- **Program threats**

- **System threats**
- **Security system facilities**

Introduction to Security

Security refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system. If a computer program is run by an unauthorized user, then he/she may cause severe damage to computer or data stored in it. So a computer system must be protected against unauthorized access, malicious access to system memory, viruses, worms etc.

(14.1) User Authentication:

A major security problem for operating system is **authentication.** The protection system depends on an ability to identify the programs and processes currently executing.

- Authentication refers to identifying each user of the system and associating the executing programs with those users. It is the responsibility of the Operating System to create a protection system which ensures that a user who is running a particular program is authentic.

Operating Systems generally identifies/authenticates users using following **three ways**:

- User possession (a key or card)
- User knowledge (a user identifier and password),
- User attribute (fingerprint, retina pattern, or signature)
- **Username / Password** – User need to enter a registered username and password with Operating system tologin into the system.
- The most common approach to authenticating a user identity is the use of passwords.
- If the password is correct, access is granted. Different passwords may be associated with different accessrights.
- For example, different passwords may be used for each of the following file operations: reading, appending, and updating
- **Password Vulnerabilities-** Passwords are extremely common because they are easy to understand and use. Unfortunately, passwords can often be guessed, accidentally exposed, sniffed, or illegally transferred form an authorized user to an unauthorized one, as we show next.
- **Encrypted Passwords-** One problem with all these approaches is the difficulty of keeping the password secret within the computer. The Unix system uses encryption to avoid the necessity of keeping its password list secret.
- **One Time Passwords-**To avoid the problems of password sniffing and shoulder surfing, a system could usea set of **paired password**. When a session begins, the system randomly selects and presents one part of a password pair ,the user must supply the other parts.
- The user uses the keypad to enter the shared secret, also known as a **Personal identification number (PIN).** The display shows the one- time password.
- Another variation on one time passwords is the use of a **code book**, or **one-time pad**, which is a list of single -use password.

- **Biometrics-**There are many other variations on the use passwords for authentication. Palm or hand –readers are common to secure physical access, for example access to a data center. These readers match stored parameters against what is being read from their hand –reader pad.

One Time passwords

One-time passwords provide additional security along with normal authentication. In One-Time Password system, a unique password is required every time user tries to login into the system. Once a one-time password is used, then it cannot be used again. One-time password are implemented in various ways.

- **Random numbers** − Users are provided cards having numbers printed along with corresponding alphabets.System asks for numbers corresponding to few alphabets randomly chosen.
- **Secret key** − User are provided a hardware device which can create a secret id mapped with user id. Systemasks for such secret id which is to be generated every time prior to login.
- **Network password** − Some commercial applications send one-time passwords to user on registered mobile/email which is required to be entered prior to login.

(14.2) Program Threats:

Operating system's processes and kernel do the designated task as instructed. If a user program made these process do malicious tasks, then it is known as **Program Threats**. One of the common example of program threat is a program installed in a computer which can store and send user credentials via network to some hacker. Following is the list of some well-known program threats.

- **Trojan Horse** − Such program traps user login credentials and stores them to send to malicious user whocan later on login to computer and can access system resources.
- **Trap Door** − If a program which is designed to work as required, have a security hole in its code and performillegal action without knowledge of user then it is called to have a trap door.
- **Stack and Buffer Overflow-** The stack or overflow attack is the most common way for an attacker outsideof the system, on a network or dial up connection to gain unauthorized access to the target system.

1. Overflow an input field, command line argument, or input buffer, for example, on a network daemon,until it writes into the stack.

2. Overflow the current return address on the stack with the address of the exploit code loaded.
3. Write a simple set of code for the next space in the stack that includes the commands that the attackerwishes to execute, for instance, spawn a shell.

(14.3) System Threats:

- System threats refers to misuse of system services and network connections to put user in trouble.
- System threats can be used to launch program threats on a complete network called as program attack.
- System threats creates such an environment that operating system resources/ user files are misused.
- Following is the list of some well-known system threats.

Worm – Worm is a process which can choked down a system performance by using system resources to extreme levels. A Worm process generates its multiple copies where each copy uses system resources, prevents all other processes to get required resources. Worm processes can even shut down an entire network.

- This worm consisted of two parts:
  1. A small program called a ***grappling hook,*** which was deposited on the target system through one of three vulnerabilities, and
  2. The main worm program, which was transferred onto the target system and launched by the grappling hook program.

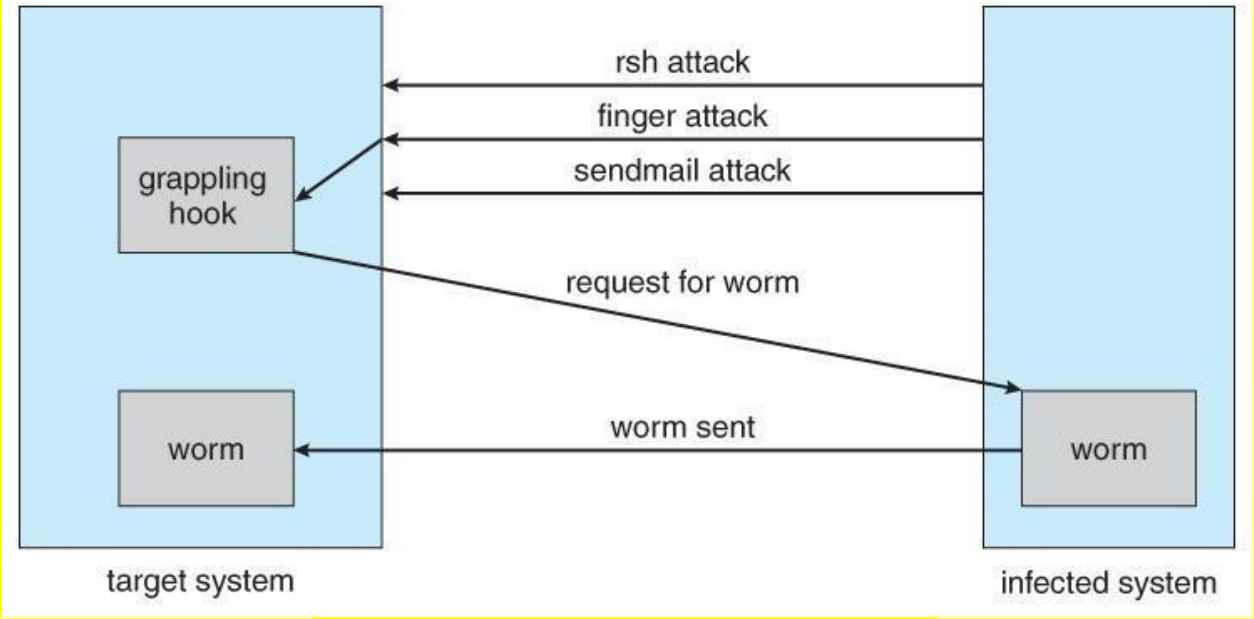


Figure 15.6 - The Morris Internet worm.

Virus – Virus as name suggest can replicate themselves on computer system. They are highly dangerous and can modify/delete user files, crash systems. A virus is generatlly a small code embedded in a program. As user accesses the program, the virus starts getting embedded in other files/ programs and can make system unusable for user

Denial of Service – The last attack category denial of service, does not involves gaining information or stealing resources, but rather disabling legitimate use of a system or facility. an intruder could delete all the files on a system.

For example. Most denial of service attacks involve system that the attacker has not penetrated. Indeed, launching an attack that prevents legitimate use is frequently easier than breaking into a machine or facility

(14.4) Securing Systems and Facilities:

Securing system and facilities is intimately linked to intrusion detection. Both techniques need to work together to assure that a system is secure and that, if a security breach happens, it is detected.

One method of improving system security is periodically to scan the system for security holes. These scans can be done when the computer has relatively little use, so they will have less effect than logging. Such a scan can check a variety of aspects of the system:

- Short or easy –to-guess passwords
- Unauthorized privileged programs, such as setuid programs
- Unauthorized programs in system directories
- Unexpected long-running processes
- Improper directory protections on both user and system directories
- Improper protections on system data files, such as the password file, device drivers, or even the operating-system kernel itself
- Dangerous entries in the program search path
- Changes to system programs detected with checksum values
- Unexpected or hidden network daemons

Any problems found by a security scan can either be fixed automatically or reported to the managers of the system.

- Firewalls are devices (or sometimes software) that sit on the border between two security domains and monitor/log activity between them, sometimes restricting the traffic that can pass between them based on certain criteria.
- For example a firewall router may allow HTTP: requests to pass through to a web server inside a company domain while not allowing telnet, ssh, or other traffic to pass through.

- A common architecture is to establish a de-militarized zone, DMZ, which sort of sits "between" the companydomain and the outside world, as shown below.
- Company computers can reach either the DMZ or the outside world, but outside computers can only reachthe DMZ.
- Perhaps most importantly, the DMZ cannot reach any of the other company computers, so even if the DMZ is breached, the attacker cannot get to the rest of the company network.

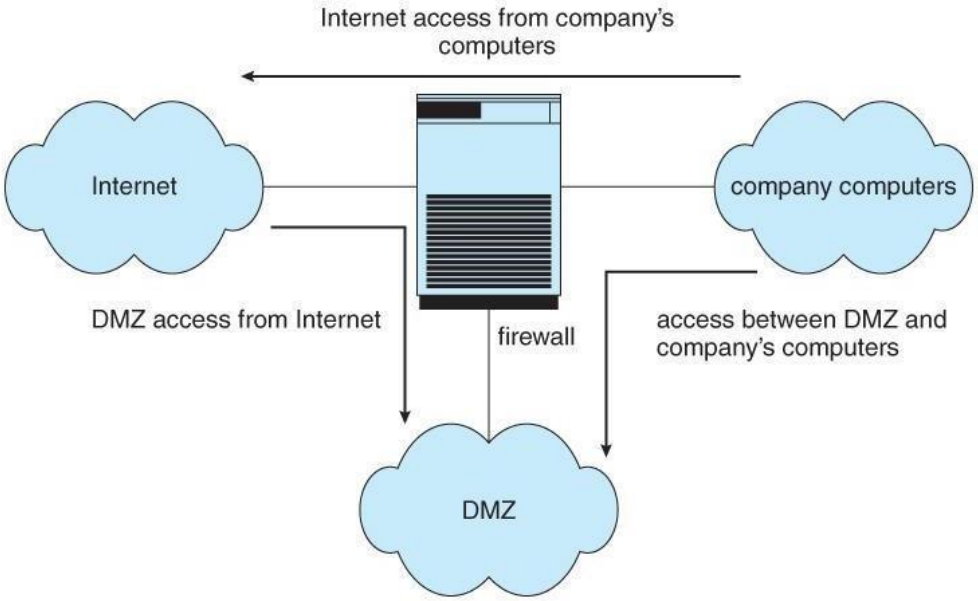


Figure - Domain separation via firewall.

Firewalls themselves need to be resistant to attacks, and unfortunately have several vulnerabilities:

- ***Tunneling,*** which involves encapsulating forbidden traffic inside of packets that are allowed.
- Denial of service attacks addressed at the firewall itself.
- Spoofing, in which an unauthorized host sends packets to the firewall with the return address of an authorizedhost.

In addition to the common firewalls protecting a company internal network from the outside world, there are also some specialized forms of firewalls that have been recently developed:

- A ***personal firewall*** is a software layer that protects an individual computer. It may be a part of the operating system or a separate software package.
- An ***application proxy firewall*** understands the protocols of a particular service and acts as a stand-in (andrelay) for the particular service.
- For example, and SMTP proxy firewall would accept SMTP requests from the outside world, examine themfor security concerns, and forward only the "safe" ones on to the real SMTP server behind the firewall.
- ***XML firewalls*** examine XML packets only, and reject ill-formed packets. Similar firewalls exist for otherspecific protocols.
- ***System call firewalls*** guard the boundary between user mode and system mode, and reject any system callsthat violate security policies.

LINUX SYSTEM

TOPICS:

- **History**
- **Design Principles**
- **Kernel Modules**
- **Process Management**
- **Scheduling**
- **Memory Management**
- **File System**

History:

Linux is a modern, free operating system based on UNIX standards

- First developed as a small but self-contained kernel in 1991 by Linus Torvalds, with the major design goal ofUNIX compatibility
- Its history has been one of collaboration by many users from all around the world, corresponding almostexclusively over the Internet
- It has been designed to run efficiently and reliably on common PC hardware, but also runs on a variety ofother platforms
- The core Linux operating system kernel is entirely original, but it can run much existing free UNIX software,resulting in an entire UNIX-compatible operating system free from proprietary code
- Many, varying Linux Distributions including the kernel, applications, and management tools

Linux is one of popular version of UNIX operating System. It is open source as its source code is freely available. It is free to use. Linux was designed considering UNIX compatibility. Its functionality list is quite similar to that of UNIX.

The Linux Kernel:

Version 0.01 was released at May 1991

- no networking
- ran only on 80386-compatible Intel processors and on PC hardware
- extremely limited device-drive support • supported only the Minix file system
- Linux 1.0 (March 1994) included these new features:
- support UNIX's standard TCP/IP networking protocols
- BSD-compatible socket interface for networking programming
- device-driver support for running IP over an Ethernet
- enhanced file system
- support for a range of SCSI controllers for high-performance disk access

- extra hardware support

Version 1.2 (March 1995) was the final PC-only Linux kernel.

Version 2.0 was released in June 1996

- support for multiple architectures, including a fully 64-bit native Alpha port
- support for multiprocessor architectures
- improved memory-management code
- improved TCP/IP performance
- support for internal kernel threads
- standardized configuration interface

2.4 and 2.6 increased SMP support

- added journaling file system
- preemptive kernel
- 64-bit memory support

The Linux System

- Linux uses many free tools developed as part of
- Berkeley's BSD operating system • socket interface
- networking tools (e.g., traceroute…)MIT's X Window System
- Free Software Foundation's GNU project
- bin-utilities,
- Linux used to developed by individual, now also big cooperators
- IBM, Intel, Red hat, Marvell, Microsoft.
- Main Linux repository: www.kernel.org

Linux Distributions

- Standard, precompiled sets of packages, or distributions
- include the basic Linux system
- system installation and management utilities
- ready-to-install packages of common UNIX tools
- Popular Linux distributions

- Ubuntu, Fedora, Debian, Open Suse,

Linux Licensing •

- Linux kernel is distributed under GNU General Public License (GPL)
- GPL is defined by the Free Software Foundation
- GPL implications: •
- anyone using Linux, or creating their own derivative of Linux, may not make the derived (public) product proprietary •
- software released under GPL may not be redistributed as binary-only • LGPL: Lesser GPL •
- allow non-(L)GPL software to link to LGPL licensed software

Components of Linux System

Linux Operating System has primarily three components

- **Kernel** – Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.
- **System Library** – System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implement most of the functionalities of the operating system and do not requires kernel module's code access rights.
- **System Utility** – System Utility programs are responsible to do specialized, individual level tasks.

Kernel Mode vs User Mode

Kernel component code executes in a special privileged mode called **kernel mode** with full access to all resources of the computer. This code represents a single process, executes in single address space and do not require any context switch and hence is very efficient and fast. Kernel runs each processes and provides system services to processes, provides protected access to hardware to processes.

Support code which is not required to run in kernel mode is in System Library. User programs and other system programs works in **User Mode** which has no access to system hardware and kernel code. User programs/ utilities

use System libraries to access Kernel functions to get system's low level tasks.

Basic Features

Following are some of the important features of Linux Operating System.

- **Portable** − Portability means software can works on different types of hardware in same way. Linux kernel and application programs supports their installation on any kind of hardware platform.
- **Open Source** − Linux source code is freely available and it is community based development project. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- **Multi-User** − Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.
- **Multiprogramming** − Linux is a multiprogramming system means multiple applications can run at same time.
- **Hierarchical File System** − Linux provides a standard file structure in which system files/ user files are arranged.
- **Shell** − Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs. etc.
- **Security** − Linux provides user security using authentication features like password protection/ controlledaccess to specific files/ encryption of data.

Architecture

The following illustration shows the architecture of a Linux system −

The architecture of a Linux System consists of the following layers −

- **Hardware layer** − Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).
- **Kernel** − It is the core component of Operating System, interacts directly with hardware, provides low levelservices to upper layer components.
- **Shell** − An interface to kernel, hiding complexity of kernel's functions from users. The shell takes commandsfrom the

user and executes kernel's functions.

- **Utilities** – Utility programs that provide the user most of the functionalities of an operating systems.

Linux History

- **Evolution of Computer:** In earlier days, computers were as big as houses or parks. So you can imagine howdifficult it was to operate them.
- Moreover, every computer has a different operating system which made it completely worse to operate onthem.
- Every software was designed for a specific purpose and was unable to operate on other computer.
- It was extremely costly and normal people neither can afford it nor can understand it.
- **Evolution of Unix**: In 1969, a team of developers of Bell Labs started a project to make a common softwarefor all the computers and named it as 'Unix'.
- It was simple and elegant, used 'C' language instead of assembly language and its code was recyclable.
- As it was recyclable, a part of its code now commonly called 'kernel' was used to develop the operatingsystem and other functions and could be used on different systems. Also its source code was open source.

Initially, Unix was only found in large organizations like government, university, or larger financial corporations with mainframes and minicomputers (PC is a microcomputer).

Unix Expansion:

- In eighties, many organizations like IBM, HP and dozen other companies started creating their own Unix. Itresult in a mess of Unix dialects.
- Then in 1983, Richard Stallman developed GNU project with the goal to make it freely available Unix likeoperating system and to be used by everyone.
- But his project failed in gaining popularity. Many other Unix like operating system came into existence butnone of them was able to gain popularity.

Evolution of Linux:

- In 1991, Linus Torvalds a student at the university of Helsinki, Finland, thought to have a freely availableacademic version of Unix started writing its own code. Later this project became the Linux kernel.
- He wrote this program specially for his own PC as he wanted to use Unix 386 Intel computer but couldn'tafford it.
- He did it on MINIX using GNU C compiler. GNU C compiler is still the main choice to compile Linux code butother compilers are also used like Intel C compiler.
- He started it just for fun but ended up with such a large project. Firstly he wanted to name it as 'Freax' butlater it became 'Linux'.
- He published the Linux kernel under his own license and was restricted to use as commercially. Linux uses most of its tools from GNU software and are under GNU copyright. In 1992, he released the kernel under GNU General Public License.

Linux Today

- Today, supercomputers, smart phones, desktop, web servers, tablet, laptops and home appliances like washing machines, DVD players, routers, modems, cars, refrigerators, etc use Linux OS.

Design Principles:

- Linux is a multiuser, multitasking system with a full set of UNIX-compatible tools.
- Its file system adheres to traditional UNIX semantics, and it fully implements the standard UNIX networkingmodel.
- Linux is designed to be compliant with the relevant POSIX documents; at least two Linux distributions haveachieved official POSIX certification.
- The Linux programming interface adheres to the SVR4 UNIX semantics, rather than to BSD behavior.
- As PCs became more powerful and as memory and hard disks became cheaper, the original, minimalist Linuxkernels grew to implement more UNIX functionality.
- Speed and efficiency are still important design goals, but much recent and current work on Linux has concentrated on a third major design goal: standardization. One of the prices paid for the diversity of UNIX implementations currently available is that source code written for one may not necessarily compile or run correctly on another.

- Even when the same system calls are present on two different UNIX systems, they do not necessarily behave in exactly the same way.
- Linux is designed to be compliant with the relevant POSIX documents; at least two Linux distributions have achieved official POSIX certification.
- 1) Components of a Linux System
- The Linux system has three main bodies of code, in sequence with, most conventional UNIX implementations.
- **THE KERNEL**: "The kernel" is in charge for maintaining all the vital abstractions of the operating system, together with such things as virtual memory and processes. The Linux kernel forms the central part of Linux operating system. It provides all the functionality compulsory to run processes, and it also provides "system services" to give arbitrated and sheltered or protected access to hardware resources. The kernel implements every feature that is required to be eligible as an operating system.
- **THE SYSTEM LIBRARIES:** "the system libraries" describe a typical set of functions through which applications can interrelate through the kernel. And which apply much of the operating system functionalitythat does not require the full rights or privileges of kernel code.
- **THE SYSTEM UTILITIES:** "the system utilities" are the programs that execute individual, particular and specialized managing tasks.
- Some of the system utilities may be invoked just once to initialize and configure some features of the system; others (known as daemons in UNIX language) may run enduringly, conducting such tasks as responding to inward or incoming network connections, accepting logon requests terminals or updating log records and files.
- The whole kernel code executes in the privileged mode of processor along with the full access to all the physical resources of the computer. This privileged mode in Linux is referred as "kernel mode", equal to the monitor mode.
- In Linux user-mode code is not built into the kernel. Any operating-system-support code that does not require to execute in kernel mode is located into the system libraries as an alternative.
- Because all kernel code and data structures are kept in a single address space, no context switches are necessary when a process calls an operating-system function or when a hardware interrupt is delivered.

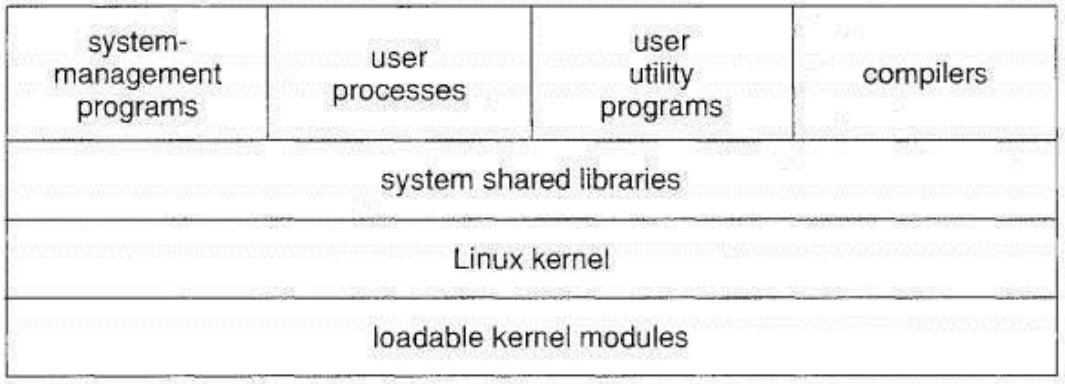


Fig: Components of the Linux system.

- This single address space contains not only the core scheduling and virtual memory code but all kernel code, including all device drivers, file systems, and networking code.
- Even though all the kernel components share this same melting pot, there is still room for modularity.
- The Linux kernel forms the core of the Linux operating system.
- The system libraries provide many types of functionality. At the simplest level, they allow applications to make kernel-system service requests. Making a system call involves transferring control from unprivileged user mode to privileged kernel mode; the details of this transfer vary from architecture to architecture. The libraries take care of collecting the system-call arguments and, if necessary, arranging those arguments in the special form necessary to make the system call.
- The libraries may also provide more complex versions of the basic system calls. For example, the C language's buffered file-handling functions are all implemented in the system libraries, providing more advanced control of file I/ 0 than the basic kernel system calls.
- The LINUX system includes a wide variety of user-mode programs-both system utilities and user utilities.
- The system utilities include all the programs necessary to initialize the system, such as those to configure network devices and to load kernel modules. Continually running server programs also count as system utilities; such programs handle user login requests, incoming network connections, and the printer queues.

Kernel module:

A **kernel module** is a binary image containing code and data structures that runs in the UNIX kernel. It has the following characteristics:

- Is statically loaded as part of /vmunix or dynamically loaded into memory
- Runs in kernel mode
- Has a file name ending with the extension .mod
- Contains a well-defined routine that executes first to initialize the module
- May be a device driver when it performs any one of these additional tasks:
    - Handles interrupts from hardware devices
    - Accepts I/O requests from applications

The kernel contains many modules, some of which are device drivers. In this book, a kernel module is defined more broadly than a device driver because it can be used to perform a variety of functions, including:

- Management functions
- Common functions shared by other modules

- Kernel code that can be compiled, loaded, and unloaded independently
- it allows a Linux system to be set up with standard minimal kernel
- other components loaded as modules
- typically to implement device drivers, file systems, or networking protocols
- ***Three components to Linux module support:***
- **module management** • load/unload the module
- resolve symbols (similar to a linker)
- ***driver registration***
- kernel define an interface, module implement the interface
- module registers to the kernel, kernel maintain a list of loaded modules
- ***conflict resolution***
- resource conflicts

- Tools to support kernel modules: lsmod, rmmod, modprobe

Module Management:

- Supports loading modules into memory and letting them talk to the rest of the kernel
- Module loading is split into two separate sections:
- Managing sections of module code in kernel memory
- Handling symbols that modules are allowed to reference
- The module requestor manages loading requested, but currently unloaded, modules; it also regularly queries the kernel to see whether a dynamically loaded module is still in use, and will unload it when it is no longer actively needed.

Driver Registration:

- Allows modules to tell the rest of the kernel that a new driver has become available
- The kernel maintains dynamic tables of all known drivers, and provides a set of routines to allow drivers to beadded to or removed from these tables at any time
- Registration tables include the following items:
  - Device drivers
  - File systems
  - Network protocols
  - Binary format

Device drivers : These drivers include character devices (such as printers terminals, or mice), block devices (including all disk drivers), and network interface devices.

File system: The file system may be anything that implements Linux virtual-file -system calling routines.

Network protocol: A module may implement an entire networking protocol, such as IPX, or simply a new set of packet-filtering rules for a network firewall.

Binary format: This format specifies a way of recognizing, and loading, a new type of executable file.

Conflict Resolution :

- A mechanism that allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another driver.
- The conflict resolution module aims to:
- Prevent modules from clashing over access to hardware resources
- Prevent auto probes from interfering with existing device drivers
- Resolve conflicts with multiple drivers trying to access the same hardware:
- 1. Kernel maintains list of allocated HW resources
- 2. Driver reserves resources with kernel database first
- 3. Reservation request rejected if resource not available

Process Management :

- A process is the basic context within which all user-requested activity is serviced within the operating system.
- ***The Fork/ Exec Process Model:***
- UNIX process management separates the creation of processes and the running of a new program into twodistinct operations.
- The **fork()** system call creates a new process
- A new program is run after a call to **exec()**
- Under UNIX, a process encompasses all the information that the operating system must maintain to track thecontext of a single execution of a single program
- Under Linux, process properties fall into three groups: the process ’s identity, environment, and context.

Process Identity:

Process ID (PID) - The unique identifier for the process; used to specify processes to the operating system when an

application makes a system call to signal, modify, or wait for another process

Credentials - Each process must have an associated user ID and one or more group IDs that determine the process's rights to access system resources and files

Personality - Not traditionally found on UNIX systems, but under Linux each process has an associated personality identifier that can slightly modify the semantics of certain system calls.

Used primarily by emulation libraries to request that system calls be compatible with certain specific flavors of UNIX.

Namespace – Specific view of file system hierarchy.

- Most processes share common namespace and operate on a shared file-system hierarchy
- But each can have unique file-system hierarchy with its own root directory and set of mounted file systems.

Process Environment: The process's environment is inherited from its parent, and is composed of two null-terminated vectors:

- The **argument vector** lists the command-line arguments used to invoke the running program; conventionallystarts with the name of the program itself.
- The **environment vector** is a list of "NAME=VALUE" pairs that associates named environment variables with arbitrary textual values.
- Passing environment variables among processes and inheriting variables by a process's children are flexiblemeans of passing information to components of the user-mode system software.
- The environment-variable mechanism provides a customization of the operating system that can be set on aper-process basis, rather than being configured for the system as a whole.

Process Context:

- The (constantly changing) state of a running program at any point in time
- The **scheduling context** is the most important part of the process context; it is the information that the scheduler needs to suspend and restart the process.
- The kernel maintains **accounting** information about the resources currently being consumed by eachprocess, and the total resources consumed by the process in its lifetime so far.

- The **file table** is an array of pointers to kernel file structures.
- When making file I/O system calls, processes s refer to files by their index into this table, the **file descriptor (fd)**
- Whereas the file table lists the existing open files, the file-system context applies to requests to open newfiles.
- The current root and default directories to be used for new file searches are stored here
- The **signal-handler table** defines the routine in the process' s address space to be called when specificsignals arrive.
- The **virtual-memory context** of a process describes the full contents of the its private address space.

Processes and Threads

Linux uses the same internal representation for processes and threads; a thread is simply a new process that happens to share the same address space as its parent λ Both are called tasks by Linux ν A distinction is only made when a new thread is created by the clone() system call λ fork() creates a new task with its own entirely new task context λ clone() creates a new task with its own identity, but that is allowed to share the data structures of its parent ν Using clone() gives an application fine-grained control over exactly what is shared between two threads

flag	meaning
FS	is
VM	The same memory space is
GHAND	
LES	The set of open files is

Scheduling

- The job of allocating CPU time to different tasks within an operating system
- While scheduling is normally thought of as the running and interrupting of processes, in Linux, scheduling also includesthe running of the various kerneltasks

CFS

- Running kernel tasks encompasses both tasks that are requested by a runningprocess and tasks that execute internally on behalf of a device driver
- As of 2.5, new scheduling algorithm – preemptive, priority-based, known as *O*(1)
  - Real-time range
  - nice value
  - Had challenges with interactive performance
  - 2.6 introduced **Completely Fair Scheduler** (**CFS**)

- Eliminates traditional, common idea of time slice
- Instead all tasks allocated portion of processor'stime
- CFS calculates how long a process should run as afunction of total number of tasks
- *N* runnable tasks means each gets 1/*N* of processor'stime
- Then weights each task with its nice value
- Smaller nice value -> higher weight (higher priority)
- Then each task run with for time proportional to task's weight divided by total weight of all runnable tasks
- Configurable variable **target latency** is desired interval during which each task should run at least once
- Consider simple case of 2 runnable tasks with equal weightand target latency of 10ms – each then runs for 5ms

- If 10 runnable tasks, each runs for 1ms
- **Minimum granularity** ensures each run has reasonable amount of time (which actually violates fairness idea)

Kernel Synchronization

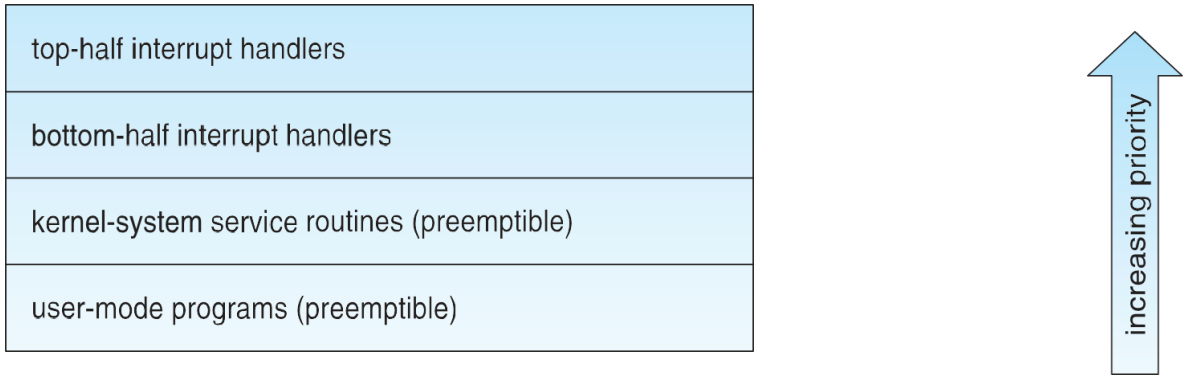
- A request for kernel-mode execution can occur in two ways:
  - A running program may request an operating system service, either explicitly via a system call, or implicitly, for example, when a page fault occurs
  - A device driver may deliver a hardware interrupt that causes the CPU to start executing a kernel-defined handler for that interrupt
- Kernel synchronization requires a framework that will allow the kernel's critical sections to run without interruption by another critical section
  - Linux uses two techniques to protect critical sections:
    1. Normal kernel code is nonpreemptible (until 2.6)
       - when a time interrupt is received while a process is executing a kernel system service routine, the kernel's **need_resched** flag is set so that the scheduler will run once the system call has completed and control is about to be returned to user mode
    2. The second technique applies to critical sections that occur in an interrupt service routines
       - By using the processor's interrupt control hardware to disable interrupts during a critical section, the kernel guarantees that it can proceed without the risk of concurrentaccess of shared data structures
  - Provides spin locks, semaphores, and reader-writer versions of both
    - Behavior modified if on single processor or multi:

single processor	multiple processors
Disable kernel preem	Acqre spin lock.
Enable kernel preem	spin lock.

- To avoid performance penalties, Linux's kernel uses a synchronization architecture that allows long critical sectionsto run without having interrupts disabled for the critical section's entire duration
- Interrupt service routines are separated into a *top half* and a *bottom half*
  - The top half is a normal interrupt service routine, and runs with recursive interrupts disabled
  - The bottom half is run, with all interrupts enabled, by a miniature scheduler that ensures that bottom halves never interrupt themselves
  - This architecture is completed by a mechanism for disabling selected bottom halves while executing normal, foreground kernel code

Interrupt Protection Levels

- each level may be inturrped by Each level may be interrupted by code running
- at a higher level, but will never be interrupted by code running at the same or a



lower level

- User processes can always be preempted by another process when a time-sharing scheduling interrupt occurs

Symmetric Multiprocessing

- Linux 2.0 was the first Linux kernel to support **SMP** hardware; separate processes or threads can execute in parallel on separate processors
- Until version 2.2, to preserve the kernel's nonpreemptible synchronization requirements, SMP imposes the restriction, via a single kernel spinlock, that only one processor at a time may execute kernel-mode code
- Later releases implement more scalability by splitting single spinlock into multiple locks, each protecting a small subset of kernel data structures
- Version 3.0 adds even more fine-grained locking, processor affinity, and load-balancing

Memory Management

- Linux's physical memory-management system deals with allocating and freeing pages, groups of pages, and smallblocks of memory
- It has additional mechanisms for handling virtualmemory, memory mapped into the address space of running processes
- Splits memory into four different **zones** due to hardware characteristics
  - Architecture specific, for example on x86:

zone	physical memory
ONE Z DMA	< 6 MB
ONE Z	6 896 MB
ONE ZHI GHMEM	> 896 MB

Managing Physical Memory

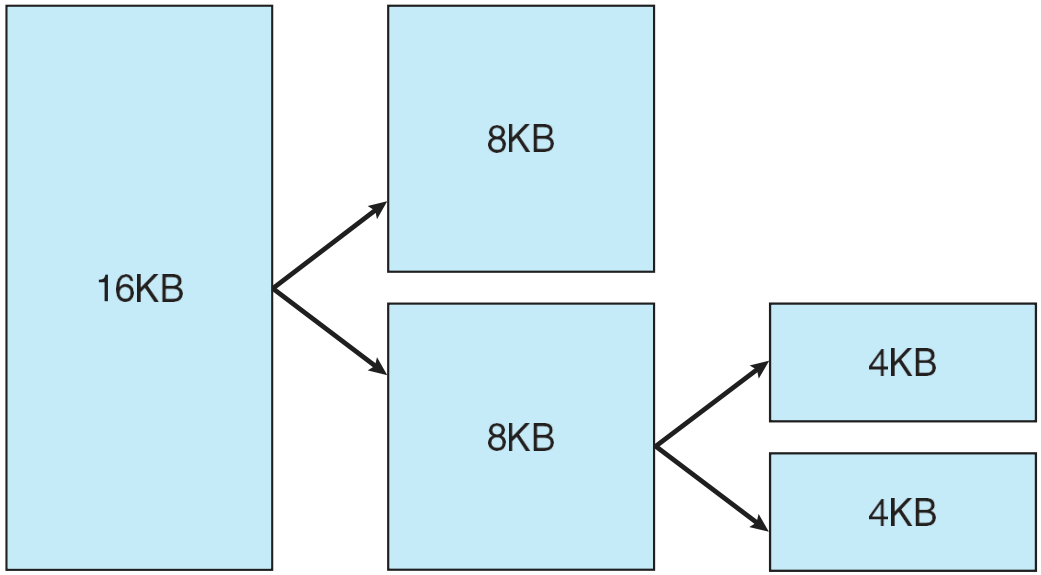
- The page allocator allocates and frees all physical pages; it can allocate ranges of physically-contiguous pages on request
- The allocator uses a buddy-heap algorithm to keep track of available physical pages
- Each allocatable memory region is paired with an adjacent partner
- Whenever two allocated partner regions are both freed up they are combined to form a larger region
- If a small memory request cannot be satisfied by allocating an existing small free region, then a larger free region will be subdivided into two partners to satisfy the request

- Memory allocations in the Linux kernel occur either statically (drivers reserve a contiguous area of memory during system boot time) or dynamically (via the page allocator)
  - Also uses **slab allocator** for kernel memory
  - **Page cache** and virtual memory system also manage

Physical memory

- Page cache is kernel's main cache for files and main mechanism for I/O to block devices
- Page cache stores entire pages of file contents for local and network file I/O

Splitting of Memory in a Buddy Heap



Virtual Memory

- The VM system maintains the address space visible to each process: It creates pages of virtual memory on demand, and manages the loading of those pages from disk or their swappingback out to disk as required.
- The VM manager maintains two separate views of a process's address space:
- A logical view describing instructions concerning the layout of the address space
- The address space consists of a set of non-overlapping regions, each representing a continuous, page-aligned subset of the address space
- A physical view of each address space which is stored in the hardware page tables for the process
- Virtual memory regions are characterized by:
- The backing store, which describes from where the pages for a regioncome; regions are usually backed by a file or by nothing (**demand- zero memory**)
- The region's reaction to writes (page sharing or copy-on- write
- The kernel creates a new virtual address space
- When a process runs a new program with the exec() system call
- Upon creation of a new process by the fork()system call
- On executing a new program, the process is given a new, completely empty virtual-address space; the program-loading routines populate the address space with virtual-memory regions
- Creating a new process with fork() involves creating a completecopy of the existing process's virtual address space
- The kernel copies the parent process's VMA descriptors, then creates a new set of page tables for the child
- The parent's page tables are copied directly into the child's, with the reference count of each page covered being incremented
- After the fork, the parent and child share the same physical pages of memory intheir address spaces

Swapping and Paging

The VM paging system relocates pages of memory from physical memory out to disk when the memory is neededfor something else

The VM paging system can be divided into two sections:

- The **pageout-policy** algorithm decides which pages to write out to disk, and when
- The **paging mechanism** actually carries out the transfer,and pages data backinto physical memory as needed
- Can page out to either swap device or normal files

- Bitmap used to track used blocks in swap space keptin physical memory
- Allocator uses next-fit algorithm to try to write contiguous runs

Kernel Virtual Memory:

- The Linux kernel reserves a constant, architecture-dependent region of the virtual address space of every process for its own internal use.
- This kernel virtual-memory area contains two regions:
- A static area that contains page table references to every available physical page of memory in the system, so that there is a simple translation from physical to virtual addresses when running kernel code.
- The reminder of the reserved section is not reserved for any specific purpose; its page-table entries can be modified to point to any other areas of memory.

Executing and Loading User Programs

- Linux maintains a table of functions for loading programs; it gives each function the opportunity to try loading the given file when an exec system call is made
- The registration of multiple loader routines allows Linux to support both the ELF and a out binary formats
- Initially, binary-file pages are mapped into virtual memory
- Only when a program tries to access a given page will a page fault resultin that page being loaded into physical memory
- An ELF-format binary file consists of a header followed by several page-aligned sections
- The ELF loader works by reading the header and mapping the sections ofthe file into separate regions of virtual memory

Static and Dynamic Linking:

- A program whose necessary library functions are embedded directly in the program's executable binary file is statically linked to its libraries
- The main disadvantage of static linkage is that every program generated must contain copies of exactly the same common system library functions
- Dynamic linking is more efficient in terms of both physical memory and disk-space usage because it loads the system libraries into memory only once
- Linux implements dynamic linking in user mode through special linker library
- Every dynamically linked program contains small statically linked function called when process starts
- Maps the link library into memory
- Link library determines dynamic libraries required by process and names of variables and functions needed
- Maps libraries into middle of virtual memory and resolves references to symbols contained in the libraries
- Shared libraries compiled to be position-independent code (PIC) so can be loaded anywhere
- To the user, Linux's file system appears as a hierarchical directory tree obeying UNIX semantics
- Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the virtual file system (VFS)
- The Linux VFS is designed around object-oriented principles and is composed of four components:
- A set of definitions that define what a file object is allowed to look likeThe
- inode object structure represent an individual file
- The file object represents an open file
- The superblock object represents an entire file system
- A dentry object represents an individual directory entry
- To the user, Linux's file system appears as a hierarchical directory tree obeying UNIX semantics
- Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the virtual file system (VFS)
- The Linux VFS is designed around object-oriented principles and layer of software to manipulate those objects with a set of operations on the objects

- For example for the file object operations include (from struct file_operations in /usr/include/linux/fs.h

  int open(. . .) – Open a file
  ssize t read(. . .) — Read from a file
  ssize t write(. . .) – Write to a file
  int mmap(. . .) – Memory-map a file

The Linux ext3 File System:

- ext3 is standard on disk file system for Linux
- Uses a mechanism similar to that of BSD Fast File System (FFS) for locatingdata blocks belonging to a specific file
- Supersedes older extfs, ext2 file systems
- Work underway on ext4 adding features like extents
- Of course, many other file system choices with Linux distros.
- The main differences between ext2fs and FFS concern their disk allocation policies
- In ffs, the disk is allocated to files in blocks of 8Kb, with blocks being subdivided into fragments of 1Kb to store small files or partially filled blocks at the end of a file
- ext3 does not use fragments; it performs its allocations in smaller units
- The default block size on ext3 varies as a function of total size of file system with support for 1, 2, 4 and 8 KB blocks
- ext3 uses cluster allocation policies designed to place logically adjacent blocks of a file into physically adjacent blocks on disk, so that it can submit an I/O request for several disk blocks as a single operation on a block group
- Maintains bit map of free blocks in a block group, searches for free byte to allocate at least 8 blocks at a time

Journaling:ext3 implements journaling, with file system updates first written to a log file in the form of transactions Once in log file, considered committed Over time, log file transactions replayed over file system to put changes in place On system crash, some transactions might be in journal but not yet placed into file system Must be completed once system recovers No other consistency checking is needed after a crash (much faster than older methods) Improves write performance on hard disks by turning random I/O into sequential I/O.

The Linux Proc File System:

The proc file system does not store data, rather, its contents are computed on demand according to user file I/O requests

proc must implement a directory structure, and the file contents within; it must then define a unique and persistent inode number for each directory and files it contains

It uses this inode number to identify just what operation is required when a user tries to read from a particular file inode or perform a lookup in a particular directory inode

When data is read from one of these files, proc collects the appropriate information, formats it into text form and places it into the requesting process's read buffer

Input and Output:

The Linux device-oriented file system accesses disk storage through two caches: Data is cached in the page cache, which is unified with the virtual memory system

- Metadata is cached in the buffer cache, a separate cache indexed by the physical disk block
- Linux splits all devices into three classes:
- block devices allow random access to completely independent, fixed size blocksof data
- character devices include most other devices; they don't need to support the functionality of regular files
- network devices are interfaced via the kernel's networking subsystem

  Block Devices: Provide the main interface to all disk devices in a system
  - The block buffer cache serves two main purposes
  - it acts as a pool of buffers for active I/O. it serves as a cache forcompleted I/O.
  - The request manager manages the reading and writing of buffer contents to and from a block device driver .
  - Kernel 2.6 introduced Completely Fair Queueing (CFQ)
  - Now the default scheduler.Fundamentally different from elevator algorithms .Maintains set of lists, one for each process by default.
  - Uses C-SCAN algorithm, with round robin between all outstanding I/Ofrom all processes . Four blocks from each process put on at once

Device-Driver Block Structure:

Character Devices: A device driver which does not offer random access to fixed blocks of data .

A character device driver must register a set of functions which implement the driver's various file I/O operations . The kernel performs almost no preprocessing of a file read or write request to a character device, but simply passes on the request to the device .The main exception to this rule is the special subset of character device drivers which implement terminal devices, for which the kernel maintains a standard interface.

Line discipline is an interpreter for the information from the terminal device . The most common line discipline is tty discipline, which glues the terminal's data stream onto standard input and output streams of user's running processes, allowing processes to communicate directly with the user's terminal .

Several processes may be running simultaneously, tty line discipline responsible for attaching and detaching terminal's input and output from various processes connected to it as processes are suspended or awakened by user. Other line disciplines also are implemented have nothing to do with I/O to user process – i.e. PPP and SLIP networking protocols.

Interprocess Communication: Like UNIX, Linux informs processes that an event has occurred via signals .There is a limited number of signals, and they cannot carry information: Only the fact that a signal occurred is available to a process .The Linux kernel does not use signals to communicate with processes with are running in kernel mode, rather, communication within the kernel is accomplished via scheduling states and wait_queue structures .Also implements System V Unix semaphores . Process can wait for a signal or a semaphore .Semaphores scale better.Operations on multiple semaphores can be atomic.

Passing Data Between Processes:

The pipe mechanism allows a child process to inherit a communication channel to its parent, data written to one end of the pipe can be read a the other .

Shared memory offers an extremely fast way of communicating; any data written by one process to a shared memory region can be read immediately by any other process that has mapped that region into its address space .

To obtain synchronization, however, shared memory must be used in conjunction with another Inter process communication mechanism.

Network Structure:

Networking is a key area of functionality for Linux .It supports the standard Internet protocols for UNIX to UNIX communications .It also implements protocols native to non-UNIX operating systems, in particular, protocols used on PC networks, such as

Appletalk and IPX . Internally, networking in the Linux kernel is implemented by three layers of software:

The socket interface Protocol drivers Network device drivers Most important set of protocols in the Linux networking system is the internet protocol suite .

It implements routing between different hosts anywhere on the network . On top of the routing protocol are built the UDP, TCP and ICMP protocols Packets also pass to firewall management for filtering based on firewall chains of rules.

Short Questions:

1.Explain about system threats?

2.Explain about securing systems and facilities?

3.Explain about kernel modules?

4.Explain about components of linux System?

Long Questions:

1.Explain about kernel synchronization?

2.Explain about kernel virtual memory?

3.Explain about process and threads?

4.Explain about process Management?

Question papers for Sample:

S.V.U. COLLEGE OF COMMERCE MANAGEMENT AND COMPUTER SCIENCE :: TIRUPATHI

Time:2hours INTERNAL EXAMINATIONS -I MAX MARKS:30

MCA 104 -OPERATING SYSTEM

Section-A

Answer any five from the following 5*2=10M

1.**What is operating system?**

2.What are the functions of operating system?

3.What is Operating system structure?

4.What are the components of an operating system?

5.Explain about CPU scheduling in Operating system?

6.Explain First come first serve algorithm with an example?

7.Explain about Round robin Scheduling?

8.Explain about Monitors?

2*10=20M

Long Questions:

Answer any one question from the each section

Section-1

9.Explain about Deadlock and its Characteristics?

or

10.Explain about deadlock avoidance and its prevention

Section-2

11.What is System calls and what are the types of System calls?

or

12.what are System programs and types of system programs?

S.V.U. COLLEGE OF COMMERCE MANAGEMENT AND COMPUTER SCIENCE :: TIRUPATHI

Time:2hours INTERNAL EXAMINATIONS -II MAX MARKS:30

MCA 104 -OPERATING SYSTEM

Section-A

Answer any five from the following 5*2=10M

1. **Write a short note on memory management?**

2. **Explain about paging?**
3. **Explain about virtual memory?**
4. **Explain about Thrashing?**
5. **Explain about File-System Structure?**
6. **Explain about File System Implementatio**
7. **Explain about File allocation methods?**
8. **Explain about directory implementation?**

Long Questions: **2*10=20M**

Answer any one question from the each section

Section-1

9.Explain about Free space management in operating system?

or

10.Explain about disk scheduling algorithm?

Section -2

11.Explain about file sharing and file protection?

12. Explain about demand paging?

MASTER OF COMPUTER APPLICATIONS SEMESTER EXAMINATION

Paper MCA 104-OPERATING SYSTEM

(Under C.B.S.C Revised Regulations w.e.f.2021-2023)

(Common paper to University and all Affiliated Colleges)

Time:3 hours **Max.Marks:70**

PART-A

(Compulsory)

Answer any five of the following questions each question carries 2 marks(5*2=10)

1. a) What are the functions of operating system?

b) .Explain about CPU scheduling in Operating system?

c)Explain First come first serve algorithm with an example?

d)Explain about Round robin Scheduling?

e) Explain about directory implementation?

f) . Explain about File allocation methods

g) Explain about paging?

h) Explain about File-System Structure?

i) .Explain about system threats?

j).Explain about securing systems and facilities

PART-B

Answer any ONE full question from each unit

Each question carries 12 Marks (5*12=60M)

UNIT-I

2.Explain about Deadlock and its Characteristics?

or

3.Explain about deadlock avoidance and its prevention

UNIT-2

4.What is System calls and what are the types of System calls?

or

5.what are System programs and types of system programs?

UNIT-3

6. Explain about Free space management in operating system?

(or)

7. Explain about disk scheduling algorithm?

UNIT-4

8. Explain about file sharing and file protection?

(or)

9. Explain about demand paging?

UNIT-5

10.Explain about process Management?

(or)

11. Explain about kernel synchronization?