आईएफटीएम विश्वविद्यालय, मुरादाबाद, उत्तर प्रदेश

IFTM University, Moradabad, Uttar Pradesh

NAAC ACCREDITED

E-Content

IFTM University, Moradabad

MCACC-12

UNIT-1

OPERATING SYSTEM

The operating system is the most important program that runs on a computer. Every general purpose computer must have an operating system to run other programs and applications. Operating systems perform basic tasks, such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices such as disk drives and printers.

Definition:- An operating system is a system software which act as an interface between a user and computer hardware. It provides an environment in which a user may execute programs. A computer system can be divided in to four component.

Operating system as User Interface –

1. User
2. System and application programs
3. Operating system
4. Hardware

Every general-purpose computer consists of the hardware, operating system, system programs, and application programs. The hardware consists of memory, CPU, ALU, and I/O devices, peripheral device, and storage device. System program consists of compilers, loaders, editors, OS, etc. The application program consists of business programs, database programs.

{

“Hardware Provides basic computing resources (CPU, memory, I/O devices).

Operating System- Controls and coordinates the use of hardware among application programs.

Application Programs- Solve computing problems of users (compilers, database systems, video games, business programs such as banking software).

Users- People, machines, other computers”

}

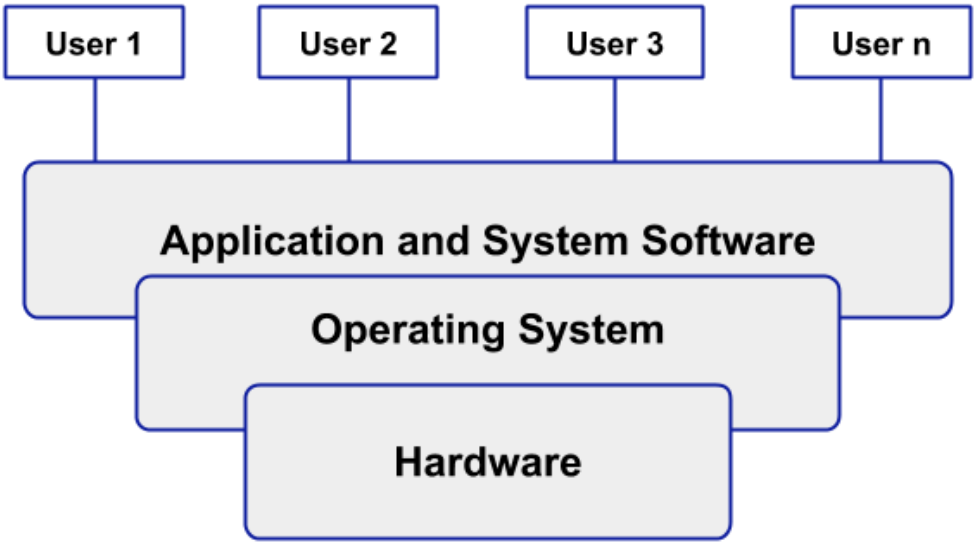


Fig1: Conceptual view of a computer system

Every computer must have an operating system to run other programs. The operating system coordinates the use of the hardware among the various system programs and application programs for various users. It simply provides an environment within which other programs can do useful work.

Goals of the Operating System:-

There are two types of goals of an Operating System i.e. Primary Goals and Secondary Goal.

- **Primary Goal:** The primary goal of an Operating System is to provide a user-friendly and convenient environment. We know that it is not compulsory to use the Operating System, but things become harder when the user has to perform all the process scheduling and converting the user code into machine code is also very difficult. So, we make the use of an Operating System to act as an intermediate between us and the hardware. All you need to do is give commands to the Operating System and the Operating System will do the rest for you. So, the Operating System should be convenient to use.
- **Secondary Goal:** The secondary goal of an Operating System is efficiency. The Operating System should perform all the management of resources in such a way that the resources are fully utilized and no resource should be held idle if some request to that resource is there at that instant of time.

So, in order to achieve the above primary and secondary goals, the Operating System performs a number of functions.

Evolution of Operating System:-

Evolution -Evolution mean the gradual development of something.

Evolution of operating system is divided into 5 phases

PHASE 0: IN THE BEGINNING (1940-1955):-

Phase 0: No operating system

- Computers are exotic experimental equipment.
- Program in machine language.
- Use plug boards to direct computer.
- No overlap between computation, I/O, think time, and response time.
- Programs manually loaded via card decks.

Phase 1 (1955-1970):-

- Make more efficient use of the computer
- computer: move the person away from the machine.
- User at console: one user at a time
- Batch monitor: load program, run, print
- OS becomes a batch monitor: a program that loads a user's
- If program failed, the OS record the contents of memory and saves it somewhere.
- Os/360 was introduced in 1963; worked in 1968.This Systems were enormously complicated. They were written in assembly code. No structured programming.

MODIFICATIONS:

- More efficient use of hardware.
- Efficiency increases because it processes the jobs as a batch collectively rather than individually.

Limitations

- No protection
- Difficult to debug!

Phase 2 (1970-1980):-

- Interactive timesharing: CTSS:
- Developed at MIT. One of the first timesharing systems. to let multiple users interact with the system at the same time
- Sacrifice CPU time to get better response time
- Users do debugging, editing, and email online.

MODIFICATIONS:

- Better utilization of resources.
- More than one user executes their tasks simultaneously.

LIMITATIONS

- Thrashing- Thrashing caused by many Factors including
- Swapping
- Inefficient queuing
- Performance very non-linear response with load

Phase 3: 1980-1990:-

- OS becomes a subroutine library
- One application at a time (MSDOS, Required high level protection and privacy for user data. First “mice”, “windows” Apple Lisa/ Macintosh: 1984 Xerox Star pp / “Look and Feel” suit 1988 Microsoft Windows: Win 1.0 (1985) .
-

Phase 4: (1990-2000):-

Networked Systems:

- Networking (Local Area Networking)
- Different machines share resources Printers, File Servers, Web Servers Client – Server Model Services: Computing File Storage

Modifications:

- Internet service providers (service between OS and apps)
- Information becomes a commodity.
- Advertising becomes a computer marketplace.

Limitations:-

- Eventually PCs become powerful: OS regains all the complexity of a “big” OS

- memory protection because of multiprogramming.

Phase 5: 2000??-???? Mobile:-

- Mobile and computer operating systems have CP/M, …)
- Gates approached Seattle Computer Products, bought 86-DOS, and created MS- DOS.
- GUI operating systems was developed first time in phase 3.

Modifications:

- OS becomes a subroutine library and command executive.
- finish quickly and run existing programs.

Limitations

complicated as compare to uni programming been developed in different ways and for different uses. Computer OS products are older and more familiar to larger groups of users. Through the last 20 years, the simple idea of a computer operating system has been continually built on and improved. Through this time, Microsoft Windows, Android and Apple's Mac OS have emerged as the two dominant operating system designs.

So many types of GUI operating systems are develop in phase 5 major types are:

OS system of mobiles.

window 95,

window 98,

window XP,

window crystal vista window 8,

window 10,

Android all Versions.

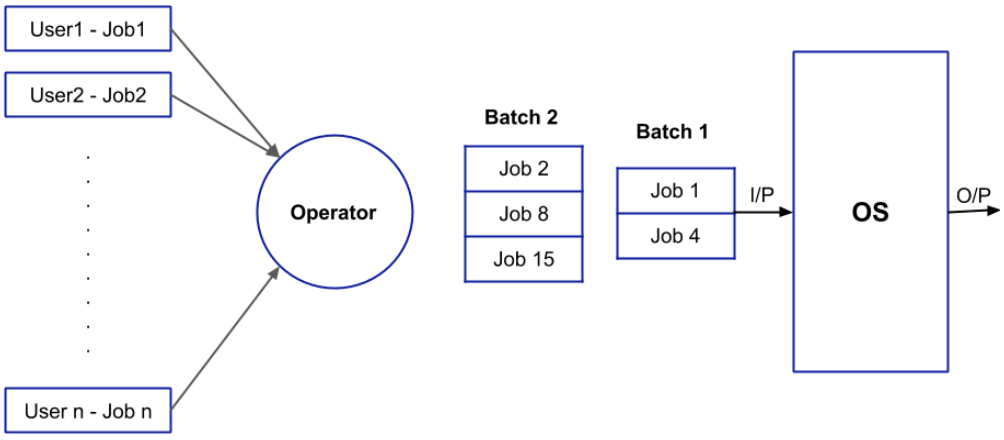
The designers and developers try to develop operating system and make it user friendly all GUI operating System is user friendly operating system. it is more easy for the user to use GUI OS as compared to Unix, Linux, Ms. Dos etc. because while using these OS user must familiar with its commands .

The goal in OS development is to make the machine convenient to use.

Types of an Operating System:-

1. Batch Operating System

In a Batch Operating System, the similar jobs are grouped together into batches with the help of some operator and these batches are executed one by one. For example, let us assume that we have 10 programs that need to be executed. Some programs are written in C++, some in C and rest in Java. Now, every time when we run these programmes individually then we will have to load the compiler of that particular language and then execute the code. But what if we make a batch of these 10 programmes. The benefit with this approach is that, for the C++ batch, you need to load the compiler only once. Similarly, for Java and C, the compiler needs to be loaded only once and the whole batch gets executed. The following image describes the working of a Batch Operating System.



Advantages:

1. The overall time taken by the system to execute all the programmes will be reduced.
2. The Batch Operating System can be shared between multiple users.

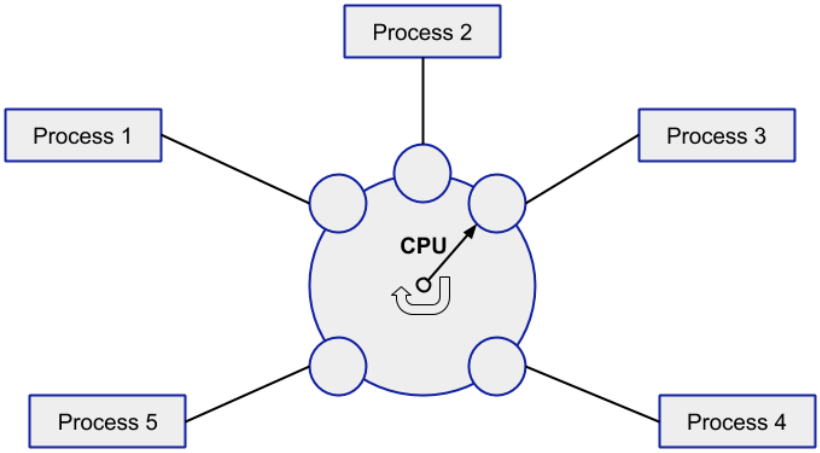
Disadvantages:

1. Manual interventions are required between two batches.
2. The CPU utilization is low because the time taken in loading and unloading of batches is very high as compared to execution time.

Examples of the batch operating system: transactions, payroll system, bank statements, reporting, integration, etc.

2. Time-Sharing or Multi-tasking Operating System

In a Multi-tasking Operating System, more than one processes are being executed at a particular time with the help of the time-sharing concept. So, in the time-sharing environment, we decide a time that is called time quantum and when the process starts its execution then the execution continues for only that amount of time and after that, other processes will be given chance for that amount of time only. In the next cycle, the first process will again come for its execution and it will be executed for that time quantum only and again next process will come. This process will continue. The following image describes the working of a Time-Sharing Operating System.



Advantages:

1. Since equal time quantum is given to each process, so each process gets equal opportunity to execute.
2. The CPU will be busy in most of the cases and this is good to have case.

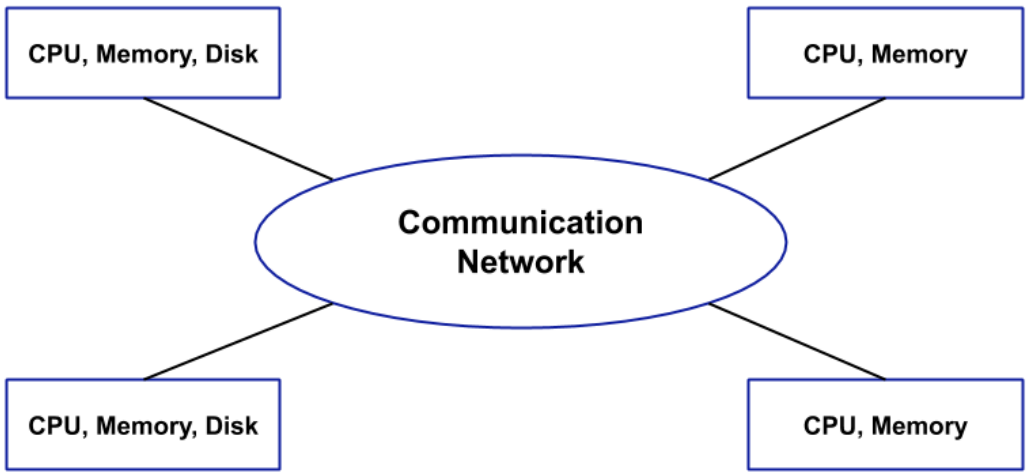
Disadvantages:

1. Process having higher priority will not get the chance to be executed first because the equal opportunity is given to each process.

   Examples of multitasking: eating and watching TV simultaneously, chatting during classes, eating chocolates while walking, talking on a phone while walking, etc.

3. Distributed Operating System

In a Distributed Operating System, we have various systems and all these systems have their own CPU, main memory, secondary memory, and resources. These systems are connected to each other using a shared communication network. Here, each system can perform its task individually. The best part about these Distributed Operating System is remote access i.e. one user can access the data of the other system and can work accordingly. So, remote access is possible in these distributed Operating Systems. The following image shows the working of a Distributed Operating System.



Advantages:

1. Since the systems are connected with each other so, the failure of one system can't stop the execution of processes because other systems can do the execution.
2. Resources are shared between each other.
3. The load on the host computer gets distributed and this, in turn, increases the efficiency.

Disadvantages:

1. Since the data is shared among all the computers, so to make the data secure and accessible to few computers, you need to put some extra efforts.
2. If there is a problem in the communication network then the whole communication will be broken.

Examples of distributed OS: intranets, the internet, sensors networks, etc.

4. Embedded Operating System

An Embedded Operating System is designed to perform a specific task for a particular device which is not a computer. For example, the software used in elevators is dedicated to the working of elevators only and nothing else. So, this can be an example of Embedded Operating System. The Embedded Operating System allows the access of device hardware to the software that is running on the top of the Operating System.

The block diagram of an embedded system consists of input devices, output devices, and memory.

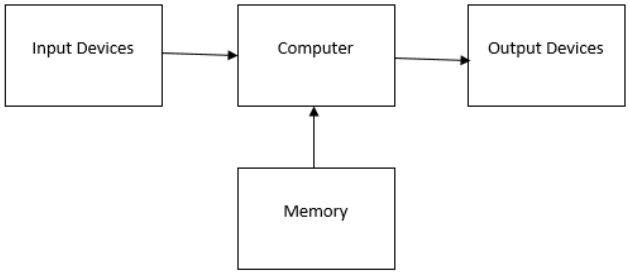


Fig The block diagram of an embedded

Input Devices: Input devices are used to send the data from the user to the system, here the user is the input. Some of the input devices are Keyboard, mouse, microphone, hard disk, sensors, switches, etc.

Output Devices: Out devices show the result to the humans in the form of text, image or sounds. Some of the output devices are printers, monitors, LCD, LED, motors, relays, buzzers, etc.

Memory: The memory is used to store the data. Some of the memory devices are SD card, EEPROM (Electrically Erasable Programmable Read-Only Memory), Flash memory. The memory devices used in the embedded system are Non-volatile RAM, volatile RAM, Dynamic Random Access Memory), etc.

Advantages:

1. Since it is dedicated to a particular job, so it is fast.
2. Low cost.

3. These consume less memory and other resources.

Disadvantages:

1. Only one job can be performed.
2. It is difficult to upgrade or is nearly scalable.

Some applications of the embedded operating system are shown in the below

- **Mobiles, Washing machines, Televisions, Microwave Ovens, Televisions, Computers, Laptops, Dishwashers, ATM's, Satellites, Vehicles**

5. Real-time Operating System

The Real-time Operating Systems are used in the situation where we are dealing with some real-time data. So, as soon as the data comes, the execution of the process should be done and there should be no delay i.e. no buffer delays should be there. Real-time OS is a time-sharing system that is based on the concept of clock interrupt. So, whenever you want to process a large number of request in a very short period of time, then you should use Real-time Operating System. For example, the details of the temperature of the petroleum industry are very crucial and this should be done in real-time and in a very short period of time. A small delay can result in a life-death situation. So, this is done with the help of Real-time Operating System. There are two types of Real-time Operating System:

1. **Hard Real-time:** In this type, a small delay can lead to drastic change. So, when the time constraint is very important then we use the Hard Real-time.
2. **Soft Real-time:** Here, the time constraint is not that important but here also we are dealing with some real-time data.

Advantages:

1. There is maximum utilization of devices and resources.
2. These systems are almost error-free.

Disadvantages:

1. The algorithms used in Real-time Operating System is very complex.
2. Specific device drivers are used for responding to the interrupts as soon as possible.

Example of real time OS:-

• Radar systems, network switching control systems, satellite monitoring systems, satellite launch-control and maneuvering mechanisms, global positioning systems all have their roots in RTOS.

• Now a days RTOS are increasingly finding use in strategic and military operations. These are used in guided missile launching units, track-and-trace spy satellites, etc.

6.Network operating System

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

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

The advantages of network operating systems are as follows −

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

The disadvantages of network operating systems are as follows −

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

Examples of network OS: Windows 2000, Linux, Microsoft windows, etc.

Operating System Structure:-

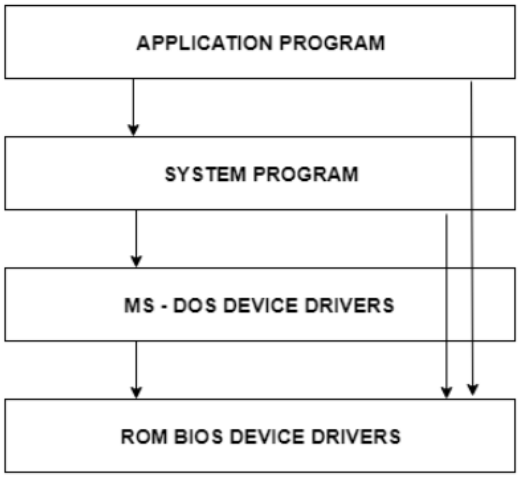
An operating system is a construct that allows the user application programs to interact with the system hardware. Since the operating system is such a complex structure, it should be created

with utmost care so it can be used and modified easily. An easy way to do this is to create the operating system in parts. Each of these parts should be well defined with clear inputs, outputs and functions.

1. Simple Structure(Monolithic)
2. Layered Structure or Layered Approach
3. Microkernel

Simple Structure(Monolithic):-

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



MS-DOS STRUCTURE

It is better that operating systems have a modular structure, unlike MS-DOS. That would lead to greater control over the computer system and its various applications. The modular structure would also allow the programmers to hide information as required and implement internal routines as they see fit without changing the outer specifications.

Unix is another OS that was inertialy limited by hardware functionality.

It consists two separate part

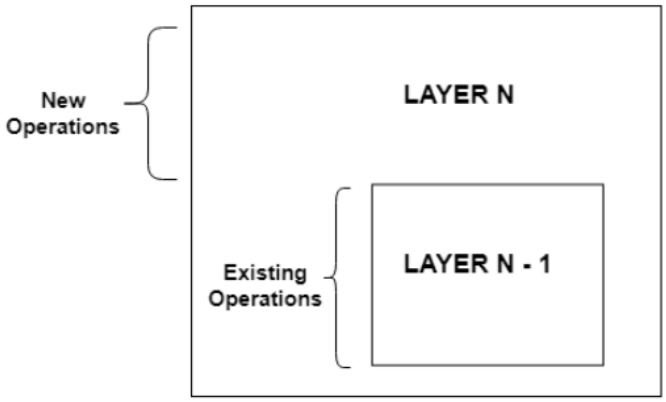
- The Kernal
- System programs

The kernel is further separated into a series of interface and device driver. In monolithic system it is possible to have at least a little structure so it is called simple structure.

2. Layered Structure or Layered Approach:-

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

An image demonstrating the layered approach is as follows –



Layered Structure of Operating System

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

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

Kernel-Based Approach

A kernel is a central component of an operating system. It acts as an interface between the user applications and the hardware. It manages the communication between the software (user level applications) and the hardware (CPU, disk memory, etc). The kernel provides functions like Process management, Device management, Memory management, Interrupt handling, I/O communication, File system, etc.

Note: Kernel and Kernel OS are different.

We can say that Linux is a kernel, not an operating system because it does not include applications like file-system utilities, windowing systems, and graphical desktops, system administrator commands, text editors, compilers, etc. So, various companies add such kind of applications over Linux kernel and provide their operating system like Ubuntu, Suse, CentOS, RedHat, etc.

Kernels may be classified mainly in three categories: -

1. Monolithic Kernel

2. Micro Kernel
3. Hybrid Kernel

1. Monolithic Kernel

Earlier in this type of kernel architecture, all the basic system services like process and memory management, interrupt handling, etc. were packaged into a single module in kernel space. This type of architecture led to some serious drawbacks like the huge size of the kernel, poor maintainability, which meant fixing bugs or addition of new features required recompilation of the whole kernel!

In modern-day approach to monolithic architecture, the kernel consists of different modules which can be dynamically loaded and unloaded. This modular approach allows easy extension of OS's capabilities. With this approach, maintainability of the kernel also has become easy as only the concerned module needs to be loaded and unloaded every time there is a change or bug fix. Also, because of this dynamic loading and unloading of the modules, stripping down the kernel for smaller platforms (like embedded devices, etc.) became easy too.

Linux follows the monolithic modular approach.

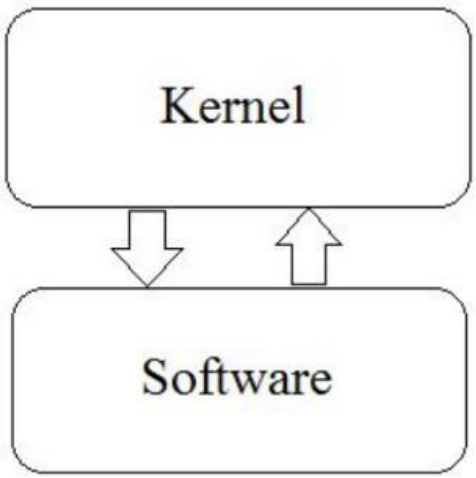


Fig: Monolithic kernel

2. Micro kernels:-

This architecture majorly solves the problem of the ever-growing size of kernel code which can't be controlled in the monolithic approach. This architecture offers some basic services like device driver management, a protocol stack, file system, etc. to run in user-space. This reduces the kernel code size and also increases the more security and stability of OS as there's bare minimum code running in the kernel.

This architecture also provides robustness. Suppose a network service crashes due to a buffer overflow, then only the networking service's memory would be corrupted, leaving the rest of the system still functional. In this architecture, all the basic OS services which are made part of user space run as servers which are used by other programs in the system through inter process communication (IPC). E.g. There are servers for device drivers, network protocol stacks, file systems, graphics, etc.

Micro-kernel servers are essentially daemon programs. Kernel grants some of them privileges to interact with parts of physical memory that are otherwise off-limits to most programs. This allows some servers, specific device drivers, to interact directly with the hardware. These servers are started at the system start-up.

So, the bare minimum that micro Kernel architecture recommends in kernel space -

- Managing memory protection
- Process scheduling
- Inter-process communication (IPC)

Apart from the above, all other basic services can be made part of user space and can be run in the form of servers.

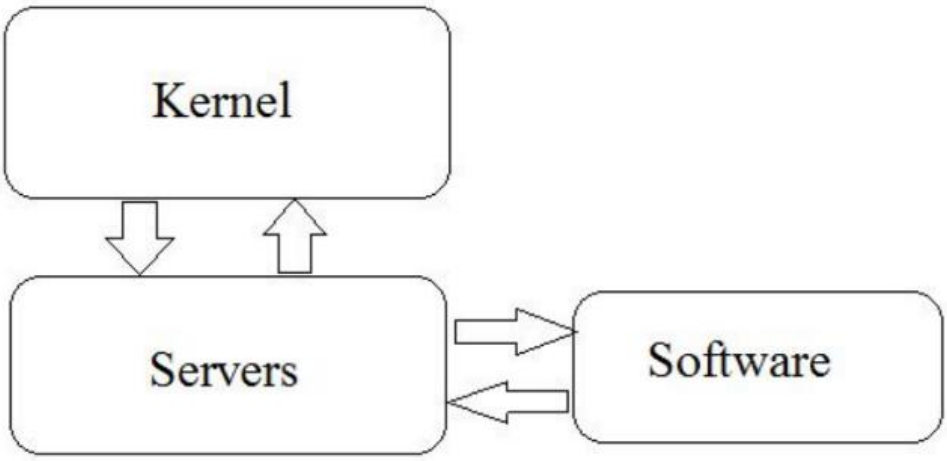


Fig: Microkernel

3. Hybrid Kernel

Hybrid Kernel combines the advantages of Monolithic kernel and Micro Kernel. This kernel takes some features from monolithic kernel like speed, simplicity of design, and modularity plus execution safety from the micro kernel.

Structure of a hybrid kernel is similar to the micro-kernel but it is implemented like a monolithic kernel. All operating system services like IPC (Inter-process communication), device drivers, etc. live in kernel space. So the benefits to the user space are reduced but it still has services like Unix-server, file server, and applications. Hybrid Kernel also doesn't have performance overhead for context switching and message passing. The best example of the Hybrid kernel is Microsoft Windows NT Kernel.

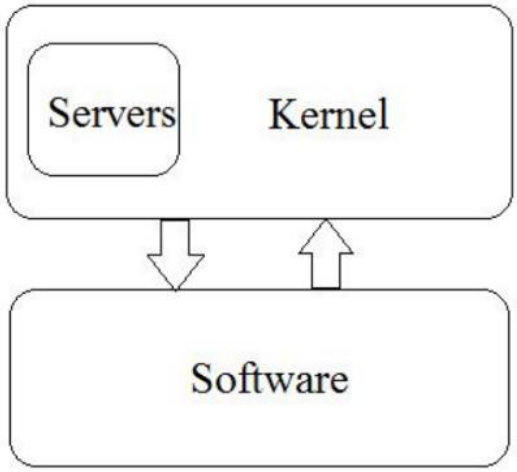


Fig: Hybrid kernel

Operating System Services:-

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

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

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

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

Program execution:-

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

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

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

I/O Operation:-

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

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

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

File system manipulation:-

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

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

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

Communication:-

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

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

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

Error handling:-

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

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

Resource Management:-

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

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

Protection:-

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

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

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

Components of 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 discussed below.

Kernel

The kernel in the OS provides the basic level of control on all the computer peripherals. In the operating system, the kernel is an essential component that loads firstly and remains within the main memory. So that memory accessibility can be managed for the programs within the RAM, it creates the programs to get access from the hardware resources. It resets the operating states of the CPU for the best operation at all times.

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

In the operating system, interrupts are essential because they give a reliable technique for the OS to communicate & react to their surroundings. 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. This tracks each & every memory position; until it is assigned to some process otherwise it is open. 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. Whenever memory is unallocated, then it tracks correspondingly to update the status. Memory management work can be divided into three important groups like memory management of hardware, OS and application memory management.

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.

Presently most of the operating systems maintain different networking techniques, hardware, & applications. This involves that computers that run on different operating systems could be included in a general network to share resources like data, computing, scanners, printers, which uses the connections of either wired otherwise wireless.

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. This system security mainly depends upon a variety of technologies that work effectively. Current operating systems give an entry to a number of resources, which are obtainable to work the software on the system, and to external devices like networks by means of the kernel. The operating system should be capable of distinguishing between demands which have to be allowed for progressing & others that don't need to be processed. Additionally, to permit or prohibit a security version, a computer system with a high level of protection also provides auditing options. So this will allow monitoring the requests from accessibility to resources

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 OS-based applications mainly provide a specific user interface for efficient communication. The main function of a user interface of an application is to get the inputs from the operator & to provide o/ps to the operator. But, the sorts of inputs received from the user interface as well as the o/p types offered by the user interface may change from application to application. The UI of any application can be classified into two types namely GUI (graphical UI) & CLI (command line user interface).

Thus, this is all about an overview of an operating system. The main components of an OS mainly include kernel, API or application program interface, user interface & file system, hardware devices and device drivers.

Virtual Machine

A **Virtual Machine** (VM) is a compute resource that uses software instead of a physical computer to run programs and deploy apps. One or more virtual "guest" machines run on a physical "host" machine. Each virtual machine runs its own operating system and functions separately from the other VMs, even when they are all running on the same host. This means that, for example, a virtual MacOS virtual machine can run on a physical PC. Virtual machine technology is used for many use cases across on-premises and cloud environments. More recently, public cloud services are using virtual machines to provide virtual application resources to multiple users at once, for even more cost efficient and flexible compute.

Virtual machines (VMs) allow a business to run an operating system that behaves like a completely separate computer in an app window on a desktop. VMs may be deployed to accommodate different levels of processing power needs, to run software that requires a different operating system, or to test applications in a safe, sandboxed environment.

Virtual machines have historically been used for server virtualization, which enables IT teams to consolidate their computing resources and improve efficiency. Additionally, virtual machines can perform specific tasks considered too risky to carry out in a host environment, such as accessing virus-infected data or testing operating systems. Since the virtual machine is separated from the rest of the system, the software inside the virtual machine cannot tamper with the host computer.

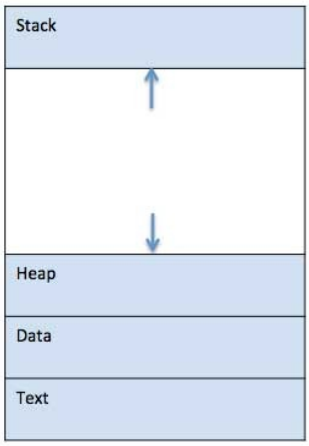
.

UNIT-2

Process:-

A program in execution is called process. The processer **(A *processor,* is a small chip that resides in *computers* and other electronic devices. Its basic job is to receive input and provide the appropriate output.)** has to manage several activities at one time. Each activity is correspond to one process. Process execution must progress in sequential fashion. The OS must maintain a data structure for each process, which describes the state and resource ownership of that process, and which enables the OS to exert control over each process.

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



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

Difference between Process and a Program:-

```
Main()
{
 int i, prod=1;
for(i=0;i<100;i++)
prod=prod*i;
}
```

- This is a program that contain one multiplication statement(prod=prod*i) but the process will execute 100 multiplication.
- Process is active entity.
- Program is passive entity.

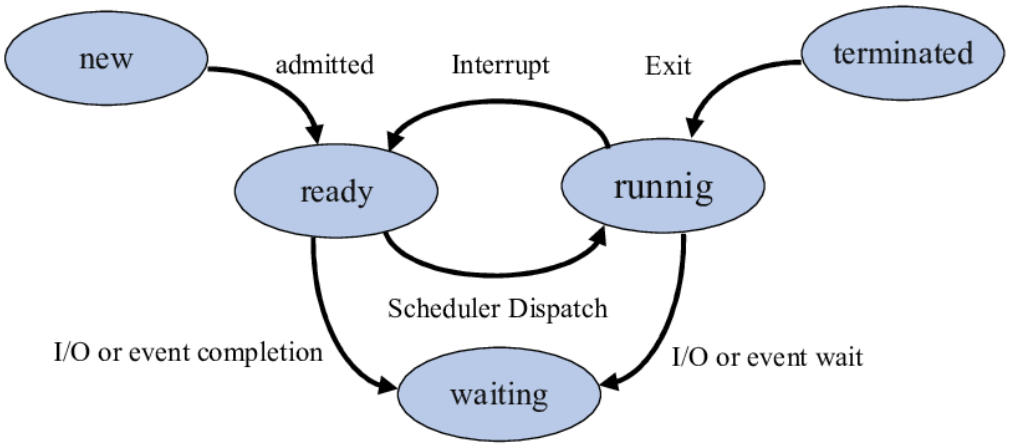
SR.NO.	PROGRAM	PROCESS
1.	Program contains a set of instructions designed to complete a specific task.	Process is an instance of an executing program.
2.	Program is a passive entity as it resides in the secondary memory.	Process is a active entity as it is created during execution and loaded into the main memory.
3.	Program exists at a single place and continues to exist until it is deleted.	Process exists for a limited span of time as it gets terminated after the completion of task.
4.	Program is a static entity.	Process is a dynamic entity.
5.	Program does not have any resource requirement, it only requires memory space for storing the instructions.	Process has a high resource requirement, it needs resources like CPU, memory address, I/O during its lifetime.
6.	Program does not have any control block.	Process has its own control block called Process Control Block.

Process States or Process Life Cycle:-

When a process comes in execution it change its states. The state of a process is defined in part by the current activity of that process. Each process may be one of the following state during the

time of execution.
New: In this state the process is being created.
Ready: In this state the process is to be assign a duty by the processor.
Waiting: In this state the process is waiting or blocked for some event to occur such as input/output completion.
Running: In this state Instructions are executed.
Terminated: In this state the process has finished the execution

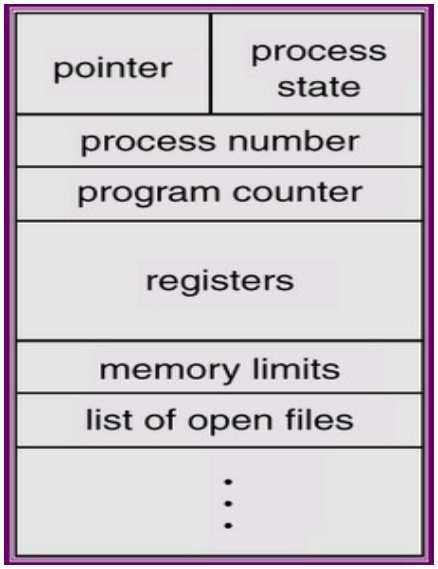
State Transition Diagram:-



Process Control Block (PCB):-

In OS each process is represented by a process control block or task control block. It is a data structure that physically represent a process in the memory of a computer system. It contains pieces of information associate with a specific process that include following:-

PCB



Pointer:- A pointer to parent process.

Process State:- It indicates the information about the state of process such as blocked ready running etc.

Process ID:- Each process is assigned a unique identification number, when it is entered into the system.

Program Counter:- It indicates the address of the next instruction to be executed.

CPU Registers:- It indicates the information about the contents of the CPU registers. The information of CPU registers must be saved when an interrupt occurs, so that the process can be continued correctly afterward. Registers hold the processed the result of calculations or addresses pointing to the memory locations of desired data.

CPU Scheduling Information:- It indicates the information needed for CPU scheduling such as process priority, pointers to scheduling queues and other scheduling parameters.

Memory Management Information:- It indicates the information needed for memory management such as value of the base and limit registers, page tables or segment tables, amount of memory units allocated to the process etc.

Accounting Information:- It indicates the information about process number, CPU used by the process time limits etc.

I/O Status Information:- It indicates the information about I/O devices allocated to the process a list of open files access rights of files opened and so on,

Link to Parent Process:- A new process can be created from existing process; the existing

process is called the parent process of the newly created process. The address of the PCB of parent process is stored.

Link to Child Process:- The addresses of the PCBs of the child processes in the main memory are stored.

Process Scheduling

- The two main objectives of the process scheduling system are to keep the CPU busy at all times and to deliver "acceptable" response times for all programs, particularly for interactive ones.
- The process scheduler must meet these objectives by implementing suitable policies for swapping processes in and out of the CPU.
- (Note that these objectives can be conflicting. In particular, every time the system steps in to swap processes it takes up time on the CPU to do so, which is thereby "lost" from doing any useful productive work.)

3.2.1 Scheduling Queues

- All processes are stored in the **job queue.**
- Processes in the Ready state are placed in the **ready queue.**
- Processes waiting for a device to become available or to deliver data are placed in **device queues**. There is generally a separate device queue for each device.
- Other queues may also be created and used as needed.

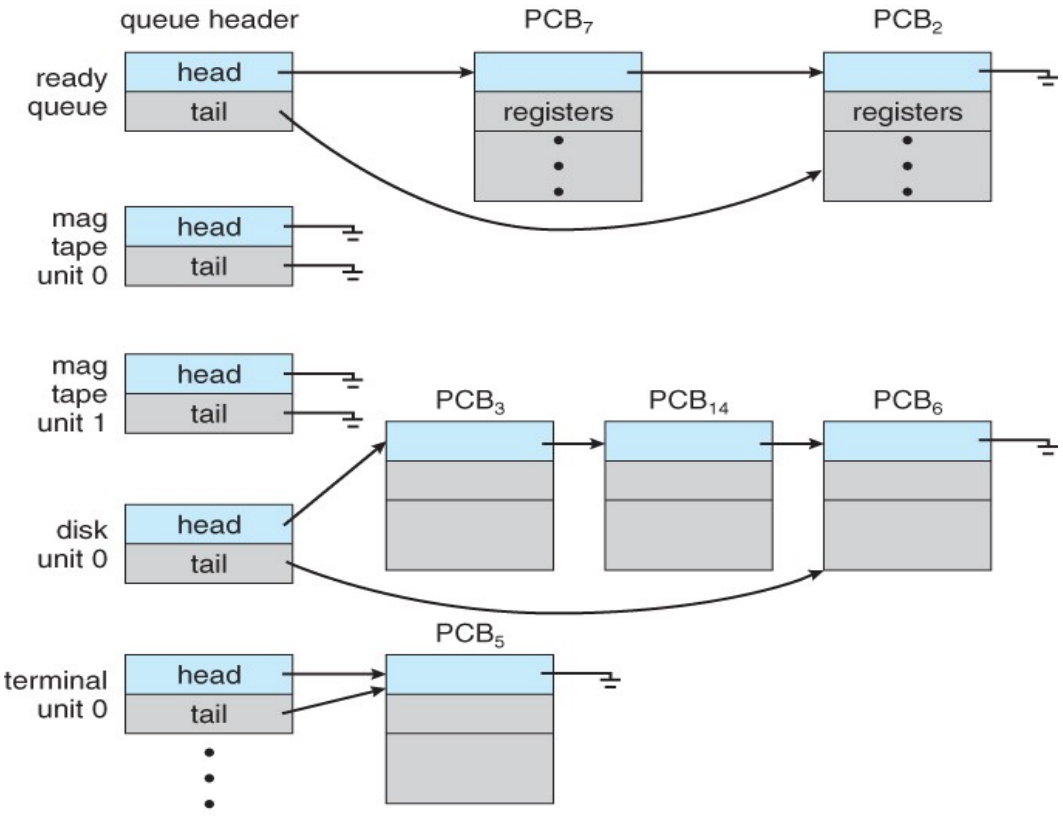


Figure 3.5 - The ready queue and various I/O device queues

Cooperating Process:-

Cooperating processes are those that can affect or are affected by other processes running on the system. Cooperating processes may share data with each other.

Reasons for needing cooperating processes

There may be many reasons for the requirement of cooperating processes. Some of these are given as follows −

- **Modularity**
  Modularity involves dividing complicated tasks into smaller subtasks. These subtasks can completed by different cooperating processes. This leads to faster and more efficient completion of the required tasks.
- **Information Sharing**
  Sharing of information between multiple processes can be accomplished using cooperating processes. This may include access to the same files. A mechanism is required so that the processes can access the files in parallel to each other.
- **Convenience**
  There are many tasks that a user needs to do such as compiling, printing, editing etc. It is convenient if these tasks can be managed by cooperating processes.

- **Computation Speedup**
  Subtasks of a single task can be performed parallely using cooperating processes. This increases the computation speedup as the task can be executed faster. However, this is only possible if the system has multiple processing elements.

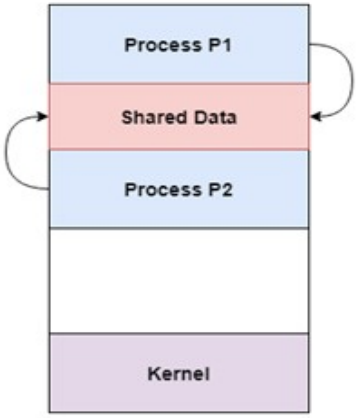
Methods of Cooperation

Cooperating processes can coordinate with each other using shared data or messages. Details about these are given as follows −

- **Cooperation by Sharing**

  The cooperating processes can cooperate with each other using shared data such as memory, variables, files, databases etc. Critical section is used to provide data integrity and writing is mutually exclusive to prevent inconsistent data.

  A diagram that demonstrates cooperation by sharing is given as follows −

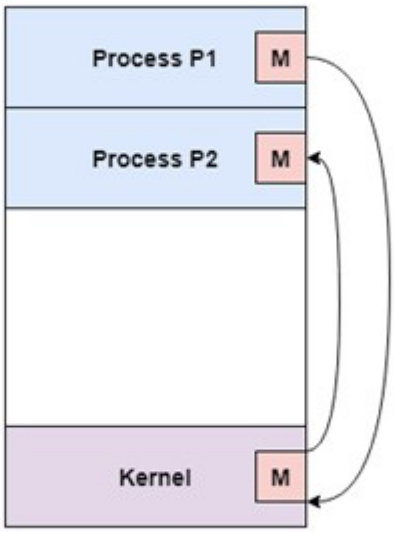


- In the above diagram, Process P1 and P2 can cooperate with each other using shared data such as memory, variables, files, databases etc.

- **Cooperation by Communication**

  The cooperating processes can cooperate with each other using messages. This may lead to deadlock if each process is waiting for a message from the other to perform a operation. Starvation is also possible if a process never receives a message.

  A diagram that demonstrates cooperation by communication is given as follows –



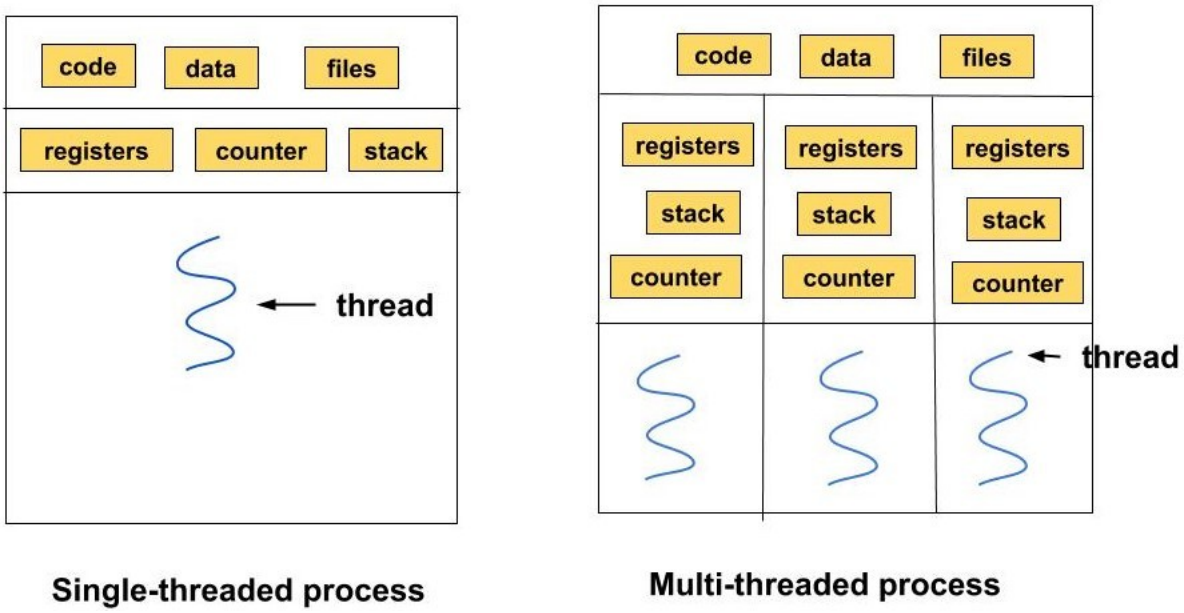
In the above diagram, Process P1 and P2 can cooperate with each other using messages to communicate.

Thread

A thread is an execution unit that has its own program counter, a stack and a set of registers that reside in a process. Threads can't exist outside any process. Also, each thread belongs to exactly one process. The information like code segment, files, and data segment can be shared by the different threads.

Threads are popularly used to improve the application through **parallelism**. Actually only one thread is executed at a time by the CPU, but the **CPU switches rapidly** between the threads to give an illusion that the threads are running parallelly.

Threads are also known as light-weight processes.



Single-threaded process

Multi-threaded process

The diagram above shows the single-threaded process and the multi-threaded process. A **single-threaded process** is a process with a single thread. A **multi-threaded process** is a process with multiple threads. As the diagram clearly shows that the multiple threads in it have its own registers, stack, and counter but they share the code and data segment.

Types of Thread

User-Level Thread

1. The user-level threads are managed by users and the kernel is not aware of it.
2. These threads are faster to create and manage.
3. The kernel manages them as if it was a single-threaded process.
4. It is implemented using user-level libraries and not by system calls. So, no call to the operating system is made when a thread switches the context.
5. Each process has its own private thread table to keep the track of the threads.

Kernel-Level Thread

1. The kernel knows about the thread and is supported by the OS.
2. The threads are created and implemented using system calls.
3. The thread table is not present here for each process. The kernel has a thread table to keep the track of all the threads present in the system.
4. Kernel-level threads are slower to create and manage as compared to user-level threads.

Advantages of threads

1. **Performance:** Threads improve the overall performance(throughput, computational speed, responsiveness) of a program.
2. **Resource sharing:** As the threads can share the memory and resources of any process it allows any application to perform multiple activities inside the same address space.
3. **Utilization of Multiple Processor Architecture:** The different threads can run parallel on the multiple processors hence, this enables the utilization of the processor to a large extent and efficiency.
4. **Reduced Context Switching Time:** The threads minimize the context switching time as in Thread Switching, the virtual memory space remains the same.
5. **Concurrency:** Thread provides concurrency within a process.
6. **Parallelism:** Parallel programming techniques are easier to implement.

Difference between process and thread

1. **Definition:** Process means a program that is currently under execution, whereas thread is an entity that resides within a process that can be scheduled for execution.
2. **Termination Time:** The processes take more time to terminate, whereas threads take less time to terminate.
3. **Creation Time:** The process creation time takes more time as compared to thread creation time.
4. **Context Switching Time:** Process context switching takes more time as compared to the thread context switching.
5. **Communication:** The communication between threads requires less time as compared to the communication between processes.
6. **Resources:** Processes are also called heavyweight processes as they use more resources. The threads are called light-weight processes as they share resources.
7. **Memory:** A Process is run in separate memory space, whereas threads run in shared memory space.
8. **Sharing Data:** Different processes have different copies of data, files, and codes whereas threads share the same copy of data, file and code segments.
9. **Example:** Opening a new browser (say Chrome, etc) is an example of creating a process. At this point, a new process will start to execute. On the contrary, opening multiple tabs in the browser is an example of creating the thread.

Inter Process Communication (IPC)

What is Inter Process Communication?
Inter process communication (IPC) is used for exchanging data between multiple threads in one or more processes or programs. The Processes may be running on single or multiple computers connected by a network. The full form of IPC is Inter-process communication.

It is a set of programming interface which allow a programmer to coordinate activities among various program processes which can run concurrently in an operating system. This allows a specific program to handle many user requests at the same time.

Since every single user request may result in multiple processes running in the operating system, the process may require to communicate with each other. Each IPC protocol approach has its own advantage and limitation, so it is not unusual for a single program to use all of the IPC methods.

Approaches for Inter-Process Communication

Here, are few important methods for interprocess communication:

Pipes

Pipe is widely used for communication between two related processes. This is a half-duplex method, so the first process communicates with the second process. However, in order to achieve a full-duplex, another pipe is needed.

Message Passing:

It is a mechanism for a process to communicate and synchronize. Using message passing, the process communicates with each other without resorting to shared variables.

IPC mechanism provides two operations:

- Send (message)- message size fixed or variable
- Received (message)

Message Queues:

A message queue is a linked list of messages stored within the kernel. It is identified by a message queue identifier. This method offers communication between single or multiple processes with full-duplex capacity.

Direct Communication:

In this type of inter-process communication process, should name each other explicitly. In this method, a link is established between one pair of communicating processes, and between each pair, only one link exists.

Indirect Communication:

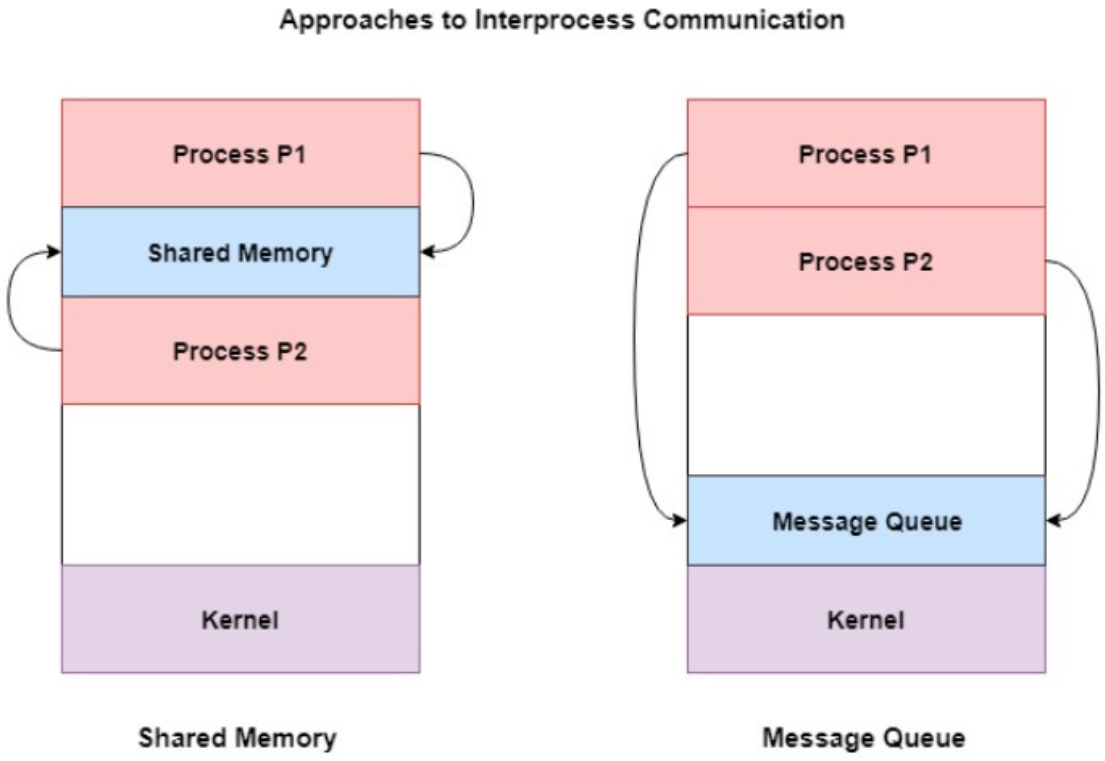
Indirect communication establishes like only when processes share a common mailbox each pair of processes sharing several communication links. A link can communicate with many processes. The link may be bi-directional or unidirectional.

Shared Memory:

Shared memory is a memory shared between two or more processes that are established using shared memory between all the processes. This type of memory requires to protected from each other by synchronizing access across all the processes.

FIFO:

Communication between two unrelated processes. It is a full-duplex method, which means that the first process can communicate with the second process, and the opposite can also happen.



CPU Scheduling:-

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

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

Schedulers:-

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

There are mainly three types of Process Schedulers:

1. Long Term
2. Short Term
3. Medium Term

Long Term Scheduler or Job Scheduler :-

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

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

Medium Term Scheduler:-

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

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

Short Term Scheduler:-

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

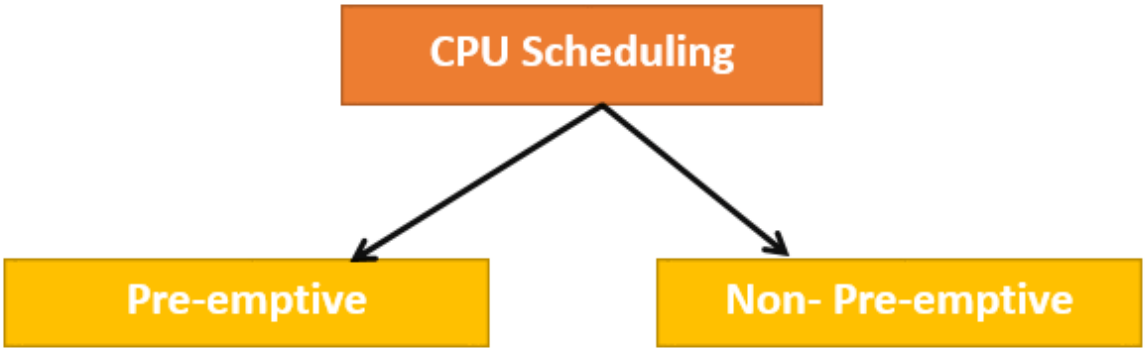
Difference between Schedulers:-

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

Long-Term	Short-Term	Medium-Term
Long term is also known as a job scheduler	Short term is also known as CPU scheduler	Medium-term is also called swapping scheduler.
It is either absent or minimal in a time-sharing system.	It is insignificant in the time-sharing order.	This scheduler is an element of Time-sharing systems.
Speed is less compared to the short term scheduler.	Speed is the fastest compared to the short-term and medium-term scheduler.	It offers medium speed.
Allow you to select processes from the loads and pool back into the memory	It only selects processes that is in a ready state of the execution.	It helps you to send process back to memory.
Offers full control	Offers less control	Reduce the level of multiprogramming.

Types of CPU Scheduling

Here are two kinds of Scheduling methods:



Preemptive Scheduling

In Preemptive Scheduling, the tasks are mostly assigned with their priorities. Sometimes it is important to run a task with a higher priority before another lower priority task, even if the lower priority task is still running. The lower priority task holds for some time and resumes when the higher priority task finishes its execution.

Non-Preemptive Scheduling

In this type of scheduling method, the CPU has been allocated to a specific process. The process that keeps the CPU busy will release the CPU either by switching context or terminating. It is the only method that can be used for various hardware platforms. That's because it doesn't need special hardware (for example, a timer) like preemptive scheduling.

When scheduling is Preemptive or Non-Preemptive?

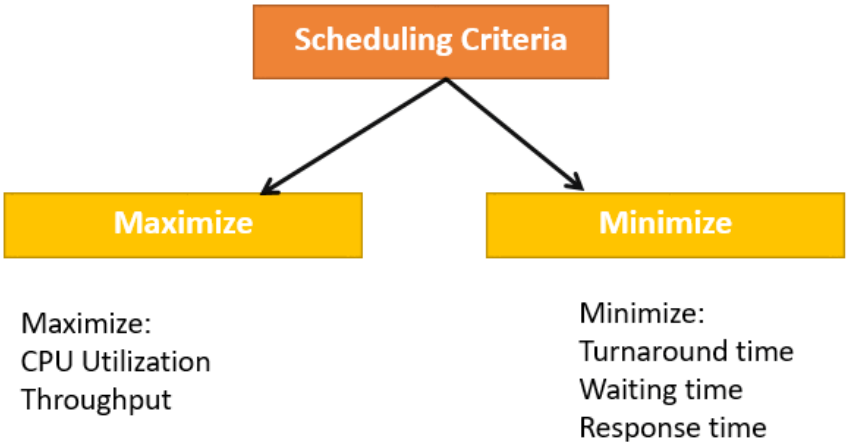
To determine if scheduling is preemptive or non-preemptive, consider these four parameters:

1. A process switches from the running to the waiting state.
2. Specific process switches from the running state to the ready state.
3. Specific process switches from the waiting state to the ready state.
4. Process finished its execution and terminated.

Only conditions 1 and 4 apply, the scheduling is called non- preemptive.
All other scheduling are preemptive.

CPU Scheduling Criteria

A CPU scheduling algorithm tries to maximize and minimize the following:



Maximize:

CPU utilization: CPU utilization is the main task in which the operating system needs to make sure that CPU remains as busy as possible. It can range from 0 to 100 percent. However, for the RTOS, it can be range from 40 percent for low-level and 90 percent for the high-level system.

Throughput: The number of processes that finish their execution per unit time is known Throughput. So, when the CPU is busy executing the process, at that time, work is being done, and the work completed per unit time is called Throughput.

Minimize:

Waiting time: Waiting time is an amount that specific process needs to wait in the ready queue.

Response time: It is an amount to time in which the request was submitted until the first response is produced.

Turnaround Time: Turnaround time is an amount of time to execute a specific process. It is the calculation of the total time spent waiting to get into the memory, waiting in the queue and, executing on the CPU. The period between the time of process submission to the completion time is the turnaround time.

Interval Timer

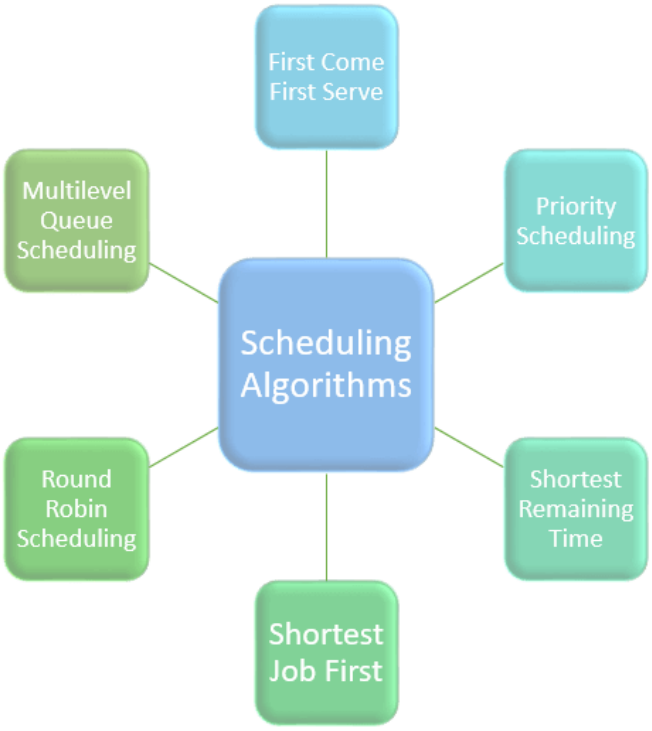
Timer interruption is a method that is closely related to preemption. When a certain process gets the CPU allocation, a timer may be set to a specified interval. Both timer interruption and preemption force a process to return the CPU before its CPU burst is complete.

Most of the multi-programmed operating system uses some form of a timer to prevent a process from tying up the system forever.

Types of CPU scheduling Algorithm

There are mainly six types of process scheduling algorithms

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



Scheduling Algorithms

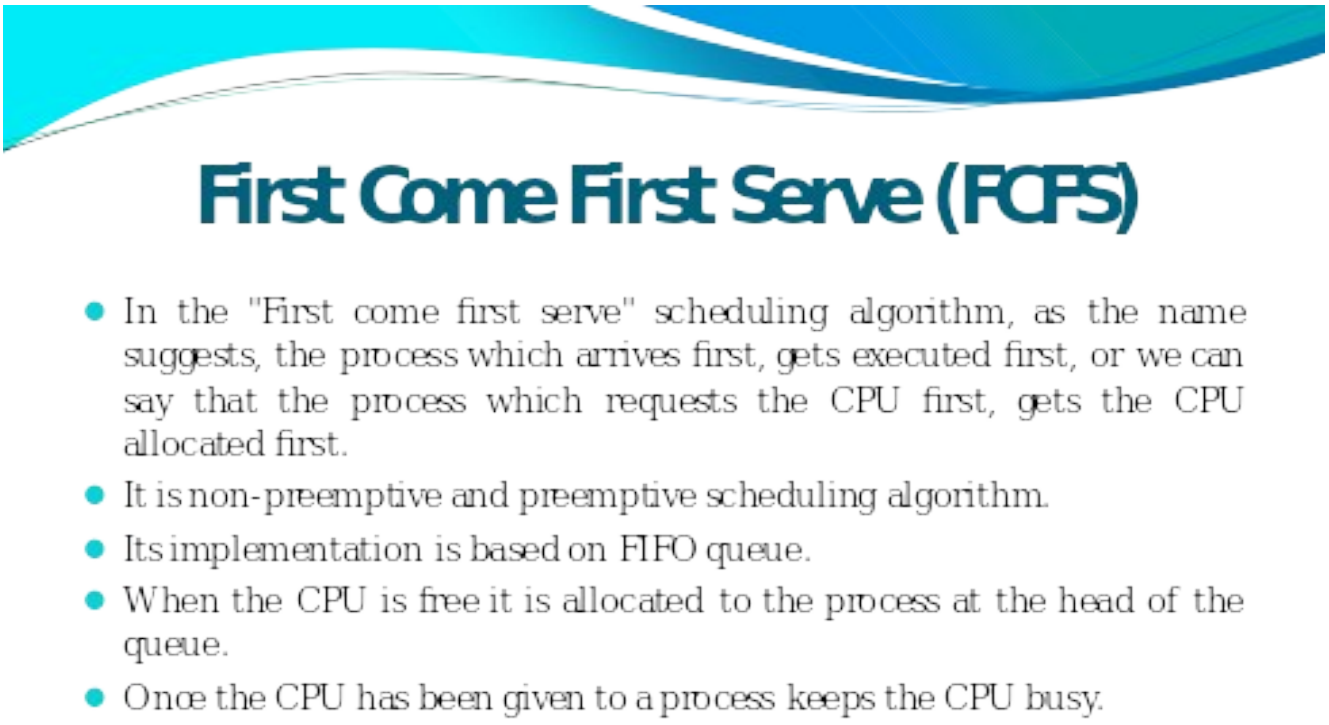
First Come First Serve

First Come First Serve is the full form of FCFS. It is the easiest and most simple CPU scheduling algorithm. In this type of algorithm, the process which requests the CPU gets the CPU allocation first. This scheduling method can be managed with a FIFO queue.

As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue. So, when CPU becomes free, it should be assigned to the process at the beginning of the queue.

Characteristics of FCFS method:

- It offers non-preemptive and pre-emptive scheduling algorithm.
- Jobs are always executed on a first-come, first-serve basis.
- It is easy to implement and use.
- However, this method is poor in performance, and the general wait time is quite high.



First Come First Serve (FCFS)

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

Wait time of each process is as follows –

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

Lets take an example:-

Q:-Draw the Gantt chart for FCFS at arrival time zero and burst-time is given in micro second calculate average waiting time and also calculate turn around time for the same process.

Process	Burst Time
P1	13
P2	08
P3	83

Solution:-

Gantt Chart:-

Waiting time for each process is-

Process	Waiting Time
P1	00
P2	13
P3	21

Problems with FCFS Scheduling:-

Below we have a few shortcomings or problems with the FCFS scheduling algorithm:

- It is **Non Pre-emptive** algorithm, which means the **process priority** doesn't matter. If a process with very least priority is being executed, more like **daily routine backup** process, which takes more time, and all of a sudden some other high priority process arrives, like **interrupt to avoid system crash**, the high priority process will have to wait, and hence in this case, the system will crash, just because of improper process scheduling.
- Not optimal Average Waiting Time.
- Resources utilization in parallel is not possible, which leads to **Convoy Effect**, and hence poor resource(CPU, I/O etc) utilization.

Shortest Remaining Time

The full form of SRT is Shortest remaining time. It is also known as SJF preemptive scheduling. In this method, the process will be allocated to the task, which is closest to its completion. This method prevents a newer ready state process from holding the completion of an older process.

Characteristics of SRT scheduling method:

- This method is mostly applied in batch environments where short jobs are required to be given preference.
- This is not an ideal method to implement it in a shared system where the required CPU time is unknown.
- Associate with each process as the length of its next CPU burst. So that operating system uses these lengths, which helps to schedule the process with the shortest possible time.

Shortest Job First

SJF is a full form of (Shortest job first) is a scheduling algorithm in which the process with the shortest execution time should be selected for execution next. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution.

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

- This is the best approach to minimize waiting time.

- This is used in Batch Systems.
- It is of two types:
    1. Non Pre-emptive
    2. Pre-emptive
- To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time.
- This scheduling algorithm is optimal if all the jobs/processes are available at the same time. (either Arrival time is `0` for all, or Arrival time is same for all.

Characteristics of SJF Scheduling

- It is associated with each job as a unit of time to complete.
- In this method, when the CPU is available, the next process or job with the shortest completion time will be executed first.
- It is Implemented with non-preemptive policy.
- This algorithm method is useful for batch-type processing, where waiting for jobs to complete is not critical.
- It improves job output by offering shorter jobs, which should be executed first, which mostly have a shorter turnaround time.

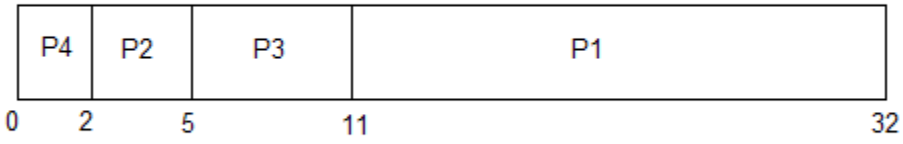
Non Pre-emptive Shortest Job First

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 you 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**.
We scheduled the same set of processes using the First come first serve algorithm in the previous tutorial, and got average waiting time to be `18.75 ms`, whereas with SJF, the average waiting time comes out `4.5 ms`.

Problem with Non Pre-emptive SJF

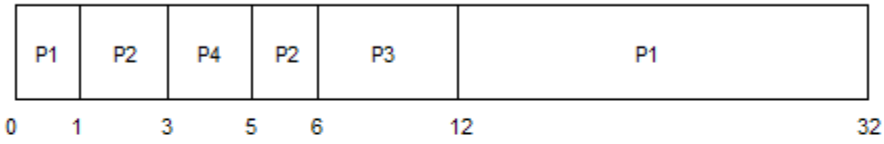
If the **arrival time** for processes are different, which means all the processes are not available in the ready queue at time `0`, and some jobs arrive after some time, in such situation, sometimes process with short burst time have to wait for the current process's execution to finish, because in Non Pre-emptive SJF, on arrival of a process with short duration, the existing job/process's execution is not halted/stopped to execute the short job first.
This leads to the problem of **Starvation**, where a shorter process has to wait for a long time until the current longer process gets executed. This happens if shorter jobs keep coming, but this can be solved using the concept of **aging**.

Pre-emptive Shortest Job First

In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with **short burst time** arrives, the existing process is preempted or removed from execution, and the shorter job is executed first.

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

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



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

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

As you can see in the **GANTT chart** above, as **P1** arrives first, hence it's execution starts immediately, but just after `1 ms`, process **P2** arrives with a **burst time** of `3 ms` which is less than the burst time of **P1**, hence the process **P1**(1 ms done, 20 ms left) is preemptied and process **P2** is executed.

As **P2** is getting executed, after `1 ms`, **P3** arrives, but it has a burst time greater than that of **P2**, hence execution of **P2** continues. But after another millisecond, **P4** arrives with a burst time of `2 ms`, as a result **P2**(2 ms done, 1 ms left) is preemptied and **P4** is executed.

After the completion of **P4**, process **P2** is picked up and finishes, then **P2** will get executed and at last **P1**.

The Pre-emptive SJF is also known as **Shortest Remaining Time First**, because at any given point of time, the job with the shortest remaining time is executed first.

Priority Based Scheduling

Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler selects the tasks to work as per the priority.

Priority scheduling also helps OS to involve priority assignments. The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis. Priority can be decided based on memory requirements, time requirements, etc.

In case of priority scheduling the priority is not always set as the inverse of the CPU burst time, rather it can be internally or externally set, but yes 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.

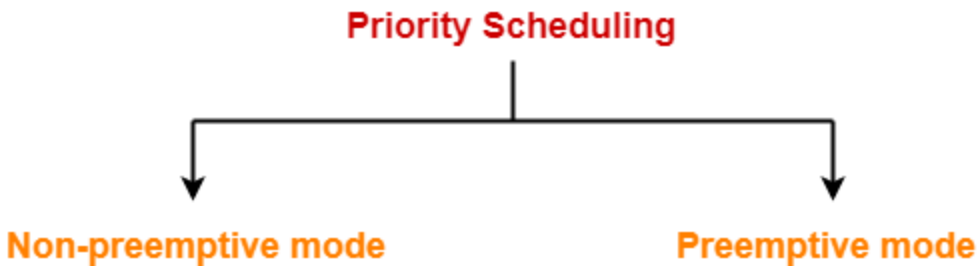
Processes with same priority are executed in FCFS manner.

The priority of process, when internally defined, can be decided based on **memory requirements**, **time limits** ,**number of open files**, **ratio of I/O burst to CPU burst** etc.

Whereas, external priorities are set based on criteria outside the operating system, like the importance of the process, funds paid for the computer resource use, makrte factor etc.

Types of Priority Scheduling Algorithm

Priority scheduling can be of two types:



1. **Preemptive Priority Scheduling**: If the new process arrived at the ready queue has a higher priority than the currently running process, the CPU is preempted, which means the processing of the current process is stoped and the incoming new process with higher priority gets the CPU for its execution.
2. **Non-Preemptive Priority Scheduling**: In case of non-preemptive priority scheduling algorithm if a new process arrives with a higher priority than the current running process, the incoming process is put at the head of the ready queue, which means after the execution of the current process it will be processed.

Advantages-

- It considers the priority of the processes and allows the important processes to run first.
- Priority scheduling in preemptive mode is best suited for real time operating system.

Disadvantages-

- Processes with lesser priority may starve for CPU.
- There is no idea of response time and waiting time.

Important Notes-

Note-01:

- The waiting time for the process having the highest priority will always be zero in preemptive mode.
- The waiting time for the process having the highest priority may not be zero in non-preemptive mode.

Note-02:

Priority scheduling in preemptive and non-preemptive mode behaves exactly same under following conditions-

- The arrival time of all the processes is same
- All the processes become available

PRACTICE PROBLEMS BASED ON PRIORITY SCHEDULING-

Problem-01:

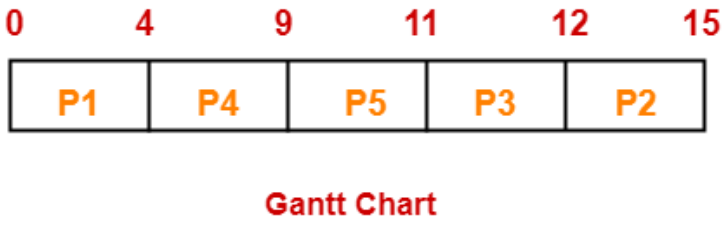
Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time	Priority
P1	0	4	2
P2	1	3	3
P3	2	1	4

P4	3	5	5
P5	4	2	5

If the CPU scheduling policy is priority non-preemptive, calculate the average waiting time and average turn around time. (Higher number represents higher priority)

Solution- Gantt Chart-



Gantt Chart

Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	4	4 – 0 = 4	4 – 4 = 0
P2	15	15 – 1 = 14	14 – 3 = 11
P3	12	12 – 2 = 10	10 – 1 = 9
P4	9	9 – 3 = 6	6 – 5 = 1
P5	11	11 – 4 = 7	7 – 2 = 5

- Average Turn Around time = (4 + 14 + 10 + 6 + 7) / 5 = 41 / 5 = 8.2 unit
- Average waiting time = (0 + 11 + 9 + 1 + 5) / 5 = 26 / 5 = 5.2 unit

Round-Robin Scheduling

Round robin is the oldest, simplest scheduling algorithm. The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turn. It is mostly used for scheduling algorithms in multitasking. This algorithm method helps for starvation free execution of processes.

Characteristics of Round-Robin Scheduling

- Round robin is a hybrid model which is clock-driven
- Time slice should be minimum, which is assigned for a specific task to be processed. However, it may vary for different processes.
- It is a real time system which responds to the event within a specific time limit.

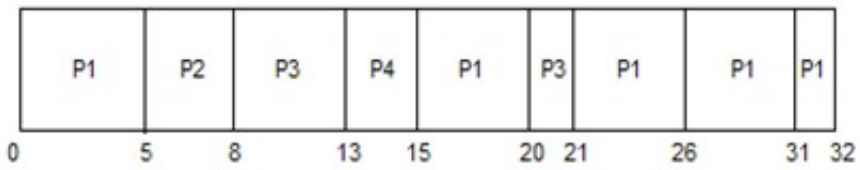
Key Points:-

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

Example:- In the given table Process(P1, P2, P3, P4) and burst time(21, 3, 6, 2) is given. The time quantum is 5 make a gantt chart and also calculate average waiting time.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The GANTT chart for round robin scheduling will be,

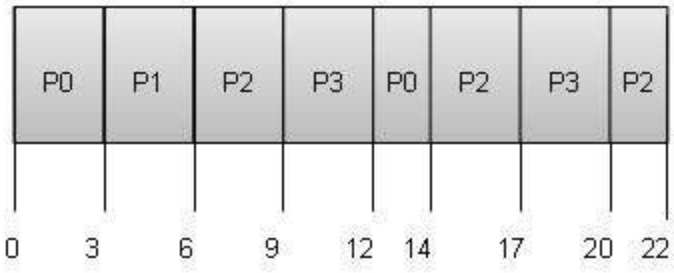


Process	Waiting Time
P1	(0-0) + (15-5) + (21-20) + (26-21) + (31-26) + (32-31) = 22
P2	(5-1) = 4
P3	(8-2) + (20-13) =13
p4	(13-3) = 10

Average Waiting Time= $\frac{22+4+13+10}{4} = 12.25$

Q2.

Quantum = 3



Wait time of each process is as follows −

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

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

Multiple-Level Queues Scheduling

This algorithm separates the ready queue into various separate queues. In this method, processes are assigned to a queue based on a specific property of the process, like the process priority, size of the memory, etc.

However, this is not an independent scheduling OS algorithm as it needs to use other types of algorithms in order to schedule the jobs.

Characteristic of Multiple-Level Queues Scheduling:

- Multiple queues should be maintained for processes with some characteristics.
- Every queue may have its separate scheduling algorithms.
- Priorities are given for each queue.

The Purpose of a Scheduling algorithm

Here are the reasons for using a scheduling algorithm:

- The CPU uses scheduling to improve its efficiency.

BCOM(H) 206

FUNDAMENTALS OF COMPUTERS

UNIT-2

Basics of MS-Word:-

Microsoft Word is a word processing program that was first developed by Mic
Since that time, Microsoft has released an abundance of updated versions, each
features and incorporating better technology than the one before it. The most cur
version of Microsoft Word is Office 365, but the software version of Microso
includes Word 2019. Microsoft Word is included in all of the Microsoft Office app
The most basic (and least expensive) suites also include Microsoft PowerPoint
Excel. Additional suites exist and include other Office programs, such as Microso
Skype for Business.

Since MS Word is one of the most used programs of the Office Suite, some bas
regarding its creation and development has been given below:

- Charles Simonyi, a developer and Richard Brodie, a software engineer, creators of MS Word
- This program was initially named “Multi-Tool Word” but later, was re Word
- It was introduced in 1983
- Word for Windows is available standalone or as a part of MS Office suite
- MS Word for Mac was introduced by Microsoft as Word 1.0 in 1985
- The extension for any word file is “.doc”

Microsoft Word used to make professional-quality documents, letters, reports, etc
a word processor developed by Microsoft. It has advanced features which allow
and edit your files and documents in the best possible way.

It helps you to allocate resources among competing processes.

- The maximum utilization of CPU can be obtained with multi-programming.
- The processes which are to be executed are in ready queue.
-

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. In multiple processor scheduling there are cases when the processors are identical i.e. HOMOGENEOUS, in terms of their functionality, we can use any processor available to run any process in the queue.

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.

Real-Time Systems

• Definition – Systems whose correctness depends on their temporal aspects as well as their functional aspects

• Performance measure – Timeliness on timing constraints (deadlines) – Speed/average case performance are less significant.

Real-time Systems

• Real-time monitoring systems

• Signal processing systems (e.g., radar)

• On-line transaction systems

• Multimedia (e.g., live video multicasting)

• Embedded control systems: – automotives – Robots – Aircrafts – Medical devices

Real-time Scheduling

Static table-driven approach - a preprocessed analysis determines the scheduling order of tasks to be run. Earliest deadline first may be used. Predictable, but a new analysis must be done if a new task is added to the system.

Static priority-driven, preemptive approach - an analysis determines the priority of the processes to be run in a regular scheduling system Easy to build on a non real-time system. Usually a rate-monotonic algorithm is used.

Dynamic planning-based approach - when a task is being added to the system, a check on the feasibility of adding the process is made. If it jeopardizes the other processes, it is not added.

Dynamic best effort approach - no analysis is made. Any process whose deadline is missed is aborted. Easy to implement and might have to be used if aperiodic tasks are used.

RT Examples

Figure shows an example of these different scheduling approaches with periodic tasks.

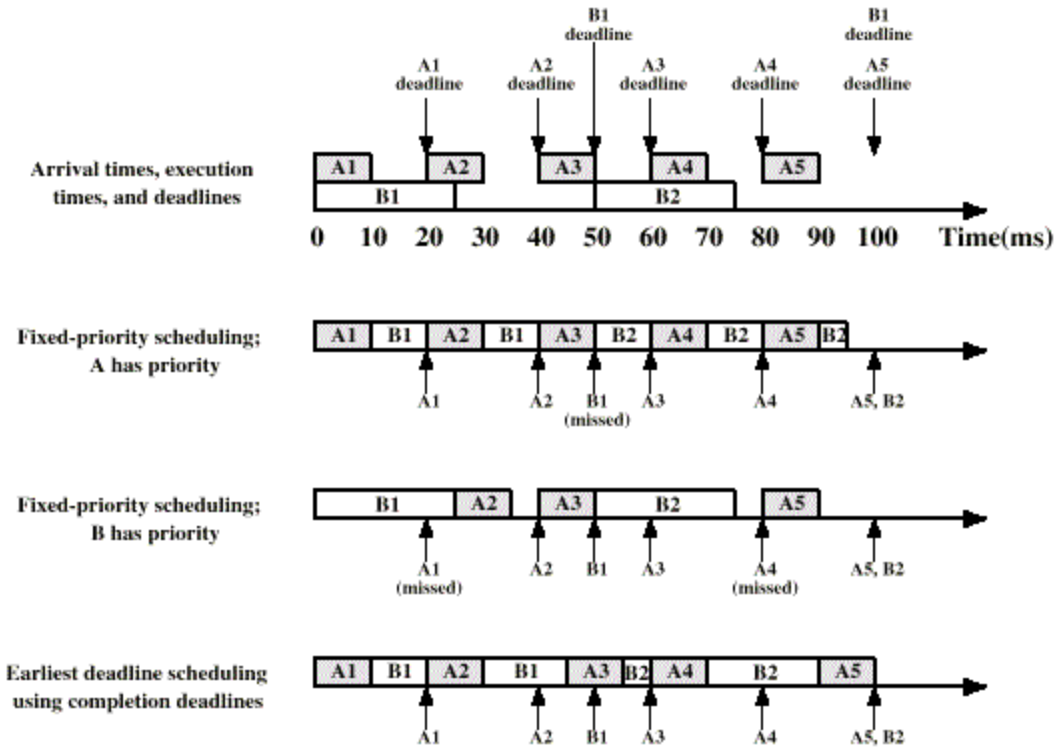


Figure 10.5 Scheduling of Periodic Real-time Tasks with Completion Deadlines

Deadline Scheduling

Figure 10.6 shows the scheduling of aperiodic real-time tasks.

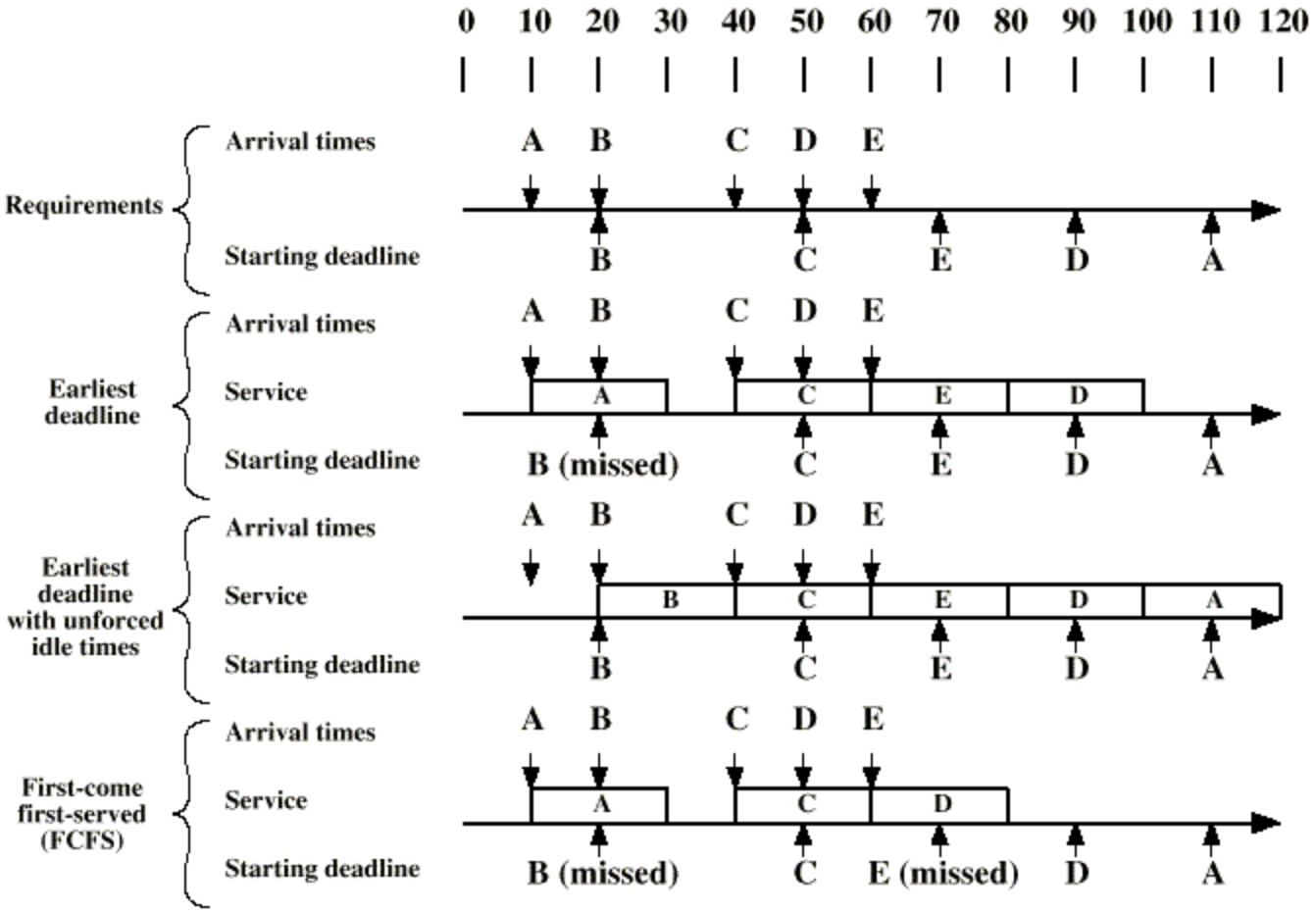


Figure 10.6 Scheduling of Aperiodic Real-time Tasks with Starting Deadlines

UNIT-3

Process Synchronization

Process Synchronization was introduced to handle problems that arose while multiple process executions.

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

It 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 processes try to gain access to the same shared resource or any data at the same time.

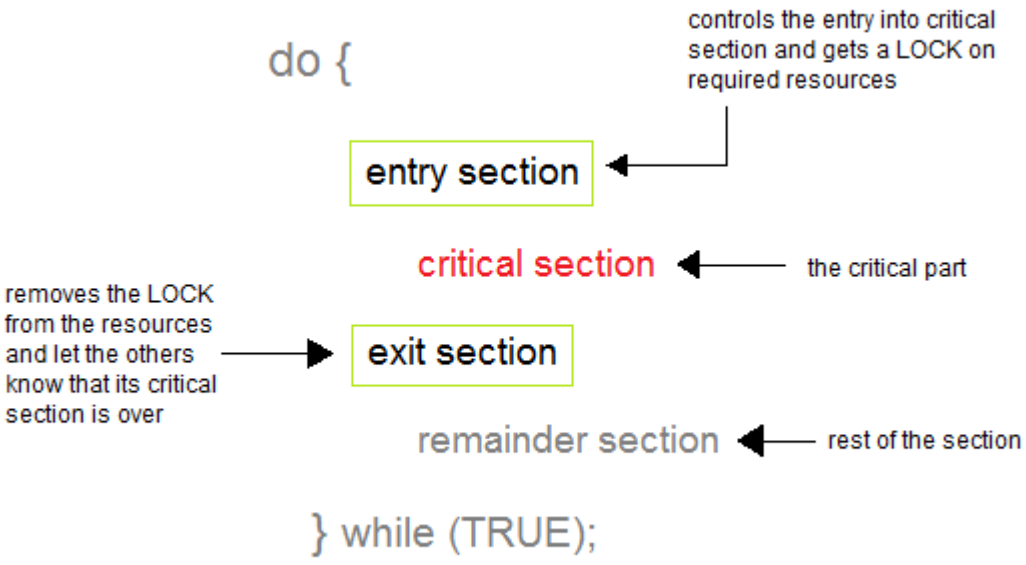
Race Condition

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.** As several

processes access and process the manipulations on the same data in a concurrent manner and due to which the outcome depends on the particular order in which the access of data takes place. Mainly this condition is a situation that may occur inside the **critical section**. Race condition in the critical section happens when the result of multiple thread execution differs according to the order in which the threads execute. But this condition is critical sections can be avoided if the critical section is treated as an atomic instruction. Proper thread synchronization using locks or atomic variables can also prevent race conditions.

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.

The solution to the Critical Section Problem

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

1. Mutual Exclusion

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

2. Progress

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

3. Bounded Waiting

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

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 method to solve the critical section problem are as follows:

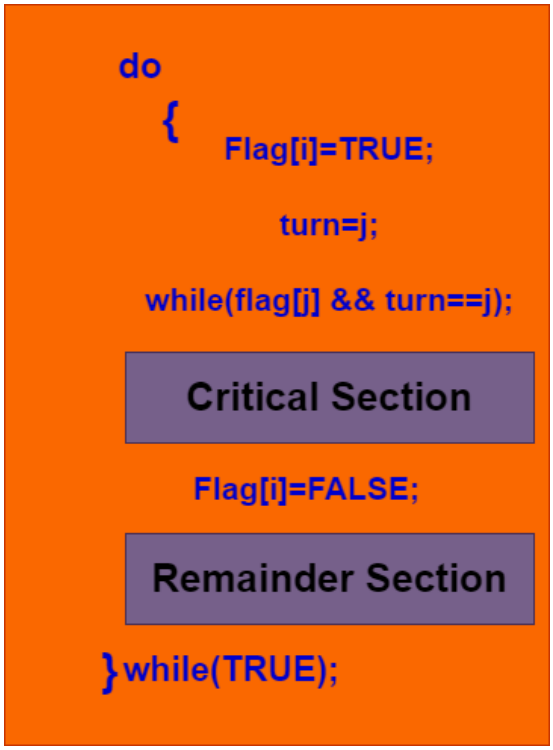
1.Peterson's Solution

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.

Synchronization Hardware

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

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

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

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

Mutex Locks

As the synchronization hardware solution is not easy to implement for everyone, a strict software approach called Mutex Locks was introduced. In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside the critical section, and in the exit section that LOCK is released.

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

Classical Problems of Synchronization

In this tutorial we will discuss about various classical problem of synchronization.

Semaphore can be used in other synchronization problems besides Mutual Exclusion.

Below are some of the classical problem depicting flaws of process synchronaization in systems where cooperating processes are present.

We will discuss the following three problems:

1. Bounded Buffer (Producer-Consumer) Problem

2. Dining Philosophers Problem
3. The Readers Writers Problem

Bounded Buffer Problem

Because the buffer pool has a maximum size, this problem is often called the **Bounded buffer problem**.

- This problem is generalized in terms of the **Producer Consumer problem**, where a **finite** buffer pool is used to exchange messages between producer and consumer processes.
- 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.
- In this Producers mainly produces a product and consumers consume the product, but both can use of one of the containers each time.
- The main complexity of this problem is that we must have to maintain the count for both empty and full containers that are available.

Dining Philosophers Problem

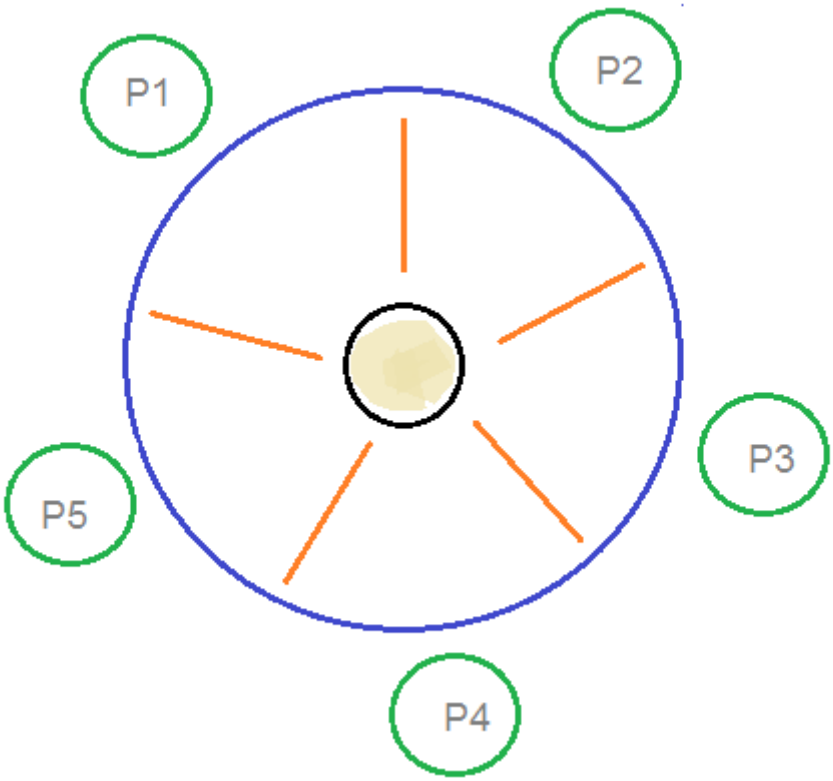
- The dining philosopher's problem involves the allocation of limited resources to a group of processes in a deadlock-free and starvation-free manner.
- There are five philosophers sitting around a table, in which there are five chopsticks/forks kept beside them and a bowl of rice in the centre, When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

Dining Philosophers Problem

The dining philosophers problem is another classic synchronization problem which is used to evaluate situations where there is a need of allocating multiple resources to multiple processes.

What is the Problem Statement?

Consider there are five philosophers sitting around a circular dining table. The dining table has five chopsticks and a bowl of rice in the middle as shown in the below figure.



Dining Philosophers Problem

At any instant, a philosopher is either eating or thinking. When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place.

Here's the Solution

From the problem statement, it is clear that a philosopher can think for an indefinite amount of time. But when a philosopher starts eating, he has to stop at some point of time. The philosopher is in an endless cycle of thinking and eating.

An array of five semaphores, stick[5], for each of the five chopsticks.

The code for each philosopher looks like:

```
while(TRUE)
```

```
{

   wait(stick[i]);

   /*

      mod is used because if i=5, next

      chopstick is 1 (dining table is circular)

   */

   wait(stick[(i+1) % 5]);



   /* eat */

   signal(stick[i]);



   signal(stick[(i+1) % 5]);

   /* think */

}
```

When a philosopher wants to eat the rice, he will wait for the chopstick at his left and picks up that chopstick. Then he waits for the right chopstick to be available, and then picks it too. After eating, he puts both the chopsticks down.

But if all five philosophers are hungry simultaneously, and each of them pickup one chopstick, then a deadlock situation occurs because they will be waiting for another chopstick forever. The possible solutions for this are:

- A philosopher must be allowed to pick up the chopsticks only if both the left and right chopsticks are available.
- Allow only four philosophers to sit at the table. That way, if all the four philosophers pick up four chopsticks, there will be one chopstick left on the table. So, one philosopher can start eating and eventually, two chopsticks will be available. In this way, deadlocks can be avoided.

The Readers Writers Problem

- In this problem there are some processes(called **readers**) that only read the shared data, and never change it, and there are other processes(called **writers**) who may change the data in addition to reading, or instead of reading it.
- There are various type of readers-writers problem, most centred on relative priorities of readers and writers.
- The main complexity with this problem occurs from allowing more than one reader to access the data at the same time.

Readers writer problem is another example of a classic synchronization problem. There are many variants of this problem, one of which is examined below.

The Problem Statement

There is a shared resource which should be accessed by multiple processes. There are two types of processes in this context. They are **reader** and **writer**. Any number of **readers** can read from the shared resource simultaneously, but only one **writer** can write to the shared resource. When a **writer** is writing data to the resource, no other process can access the resource. A **writer** cannot write to the resource if there are non zero number of readers accessing the resource at that time.

The Solution

From the above problem statement, it is evident that readers have higher priority than writer. If a writer wants to write to the resource, it must wait until there are no readers currently accessing that resource.

Here, we use one **mutex** m and a **semaphore** w. An integer variable read_count is used to maintain the number of readers currently accessing the resource. The variable read_count is initialized to 0. A value of 1 is given initially to m and w.

Instead of having the process to acquire lock on the shared resource, we use the mutex m to make the process to acquire and release lock whenever it is updating the read_count variable.

The code for the **writer** process looks like this:

```
while(TRUE)

{

   wait(w);



  /* perform the write operation */



  signal(w);

}
```

And, the code for the **reader** process looks like this:

```
while(TRUE)

{

   //acquire lock

   wait(m);

   read_count++;

   if(read_count == 1)
```

```
    wait(w);



  //release lock

  signal(m);



  /* perform the reading operation */



  // acquire lock

  wait(m);

  read_count--;

  if(read_count == 0)

    signal(w);



  // release lock

  signal(m);

}
```

Here is the Code uncoded(explained)

- As seen above in the code for the writer, the writer just waits on the **w** semaphore until it gets a chance to write to the resource.
- After performing the write operation, it increments **w** so that the next writer can access the resource.
- On the other hand, in the code for the reader, the lock is acquired whenever the **read_count** is updated by a process.
- When a reader wants to access the resource, first it increments the **read_count** value, then accesses the resource and then decrements the **read_count** value.
- The semaphore **w** is used by the first reader which enters the critical section and the last reader which exits the critical section.
- The reason for this is, when the first readers enters the critical section, the writer is blocked from the resource. Only new readers can access the resource now.
- Similarly, when the last reader exits the critical section, it signals the writer using the **w** semaphore because there are zero readers now and a writer can have the chance to access the resource.

Introduction to Semaphores

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

In very simple words, the **semaphore** is a variable that can hold only a non-negative Integer value, shared between all the threads, with operations **wait** and **signal**, which work as follow:

```
P(S): if S >= 1 then S := S - 1

    else <block and enqueue the process>;



V(S): if <some process is blocked on the queue>
```

```
    then <unblock a process>

   else S := S + 1;
```

The classical definitions of **wait** and **signal** are:

- **Wait**: This operation decrements the value of its argument S, as soon as it would become non-negative(greater than or equal to 1). This Operation mainly helps you to control the entry of a task into the critical section. In the case of the negative or zero value, no operation is executed. wait() operation was originally termed as P; so it is also known as **P(S) operation**. The definition of wait operation is as follows:

```
wait(S)

{

   while (S<=0);//no operation

   S--;

}
```

Note:

When one process modifies the value of a semaphore then, no other process can simultaneously modify that same semaphore's value. In the above case the integer value of S(S<=0) as well as the possible modification that is S-- must be executed without any interruption.

- **Signal**: Increments the value of its argument S, as there is no more process blocked on the queue. This Operation is mainly used to control the exit of a task from the critical section.signal() operation was originally termed as V; so it is also known as **V(S) operation**. The definition of signal operation is as follows:

```
signal(S)
{
S++;
}
```

Also, note that all the modifications to the integer value of semaphore in the wait() and signal() operations must be executed indivisibly.

Properties of Semaphores

1. It's simple and always have a non-negative integer value.
2. Works with many processes.
3. Can have many different critical sections with different semaphores.
4. Each critical section has unique access semaphores.
5. Can permit multiple processes into the critical section at once, if desirable.

We will now cover the types of semaphores in the Operating system;

Types of Semaphores

Semaphores are mainly of two types in Operating system:

1. **Binary Semaphore:**

   It is a special form of semaphore used for implementing mutual exclusion, hence it is often called a **Mutex**. A binary semaphore is initialized to 1 and only takes the values 0 and 1 during the execution of a program. In Binary Semaphore, the wait operation works only if the value of semaphore = 1, and the signal operation succeeds when the semaphore= 0. Binary Semaphores are easier to implement than counting semaphores.

2. **Counting Semaphores:**

   These are used to implement **bounded concurrency**. The Counting semaphores can range over an **unrestricted domain**. These can be used to control access to a given resource that consists of a finite number of Instances. Here the semaphore count is used to indicate the number of available resources. If the resources are added then the semaphore count automatically gets incremented and if the resources are removed, the count is decremented. Counting Semaphore has no mutual exclusion.

Example of Use

Here is a simple step-wise implementation involving declaration and usage of semaphore.

```
Shared var mutex: semaphore = 1;
Process i
   begin
   .
```

.

P(mutex);

execute CS;

V(mutex);

.

.

End;

Advantages of Semaphores

Benefits of using Semaphores are as given below:

- With the help of semaphores, there is a flexible management of resources.
- Semaphores are machine-independent and they should be run in the machine-independent code of the microkernel.
- Semaphores do not allow multiple processes to enter in the critical section.
- They allow more than one thread to access the critical section.
- As semaphores follow the mutual exclusion principle strictly and these are much more efficient than some other methods of synchronization.
- No wastage of resources in semaphores because of busy waiting in semaphores as processor time is not wasted unnecessarily to check if any condition is fulfilled in order to allow a process to access the critical section.

Disadvantages of Semaphores

- One of the biggest limitations is that semaphores may lead to priority inversion; where low priority processes may access the critical section first and high priority processes may access the critical section later.
- To avoid deadlocks in the semaphore, the Wait and Signal operations are required to be executed in the correct order.
- Using semaphores at a large scale is impractical; as their use leads to loss of modularity and this happens because the wait() and signal() operations prevent the creation of the structured layout for the system.
- Their use is not enforced but is by convention only.
- With improper use, a process may block indefinitely. Such a situation is called **Deadlock**.

Deadlock

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

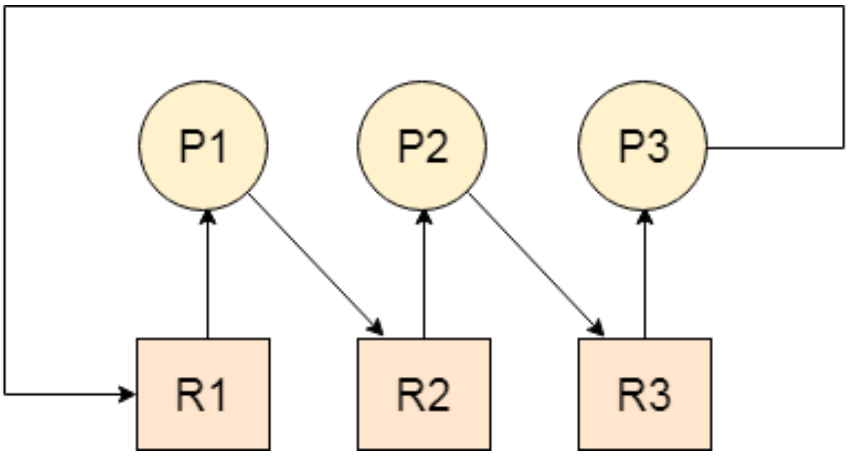
1. The process requests for some resource.
2. OS grant the resource if it is available otherwise let the process waits.
3. The process uses it and release on the completion.

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

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

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

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



System Model:-

- For the purposes of deadlock discussion, a system can be modeled as a collection of limited resources, which can be partitioned into different categories, to be allocated to a number of processes, each having different needs.
- Resource categories may include memory, printers, CPUs, open files, tape drives, CD-ROMS, etc.
- By definition, all the resources within a category are equivalent, and a request of this category can be equally satisfied by any one of the resources in that category. If this is not the case (i.e. if there is some difference between the resources within a category), then that category needs to be further divided into separate categories. For example, "printers" may need to be separated into "laser printers" and "color inkjet printers".
- Some categories may have a single resource.
- In normal operation a process must request a resource before using it, and release it when it is done, in the following sequence:
    1. **Request -** If the request cannot be immediately granted, then the process must wait until the resource(s) it needs become available. For example the system calls open( ), malloc( ), new( ), and request( ).
    2. **Use -** The process uses the resource, e.g. prints to the printer or reads from the file.
    3. **Release -** The process relinquishes the resource. so that it becomes available for other processes. For example, close( ), free( ), delete( ), and release( ).
- For all kernel-managed resources, the kernel keeps track of what resources are free and which are allocated, to which process they are allocated, and a queue of processes waiting for this resource to become available. Application-managed resources can be controlled using mutexes or wait( ) and signal( ) calls, (i.e. binary or counting semaphores.)
- A set of processes is deadlocked when every process in the set is waiting for a resource that is currently allocated to another process in the set (and which can only be released when that other waiting process makes progress.)

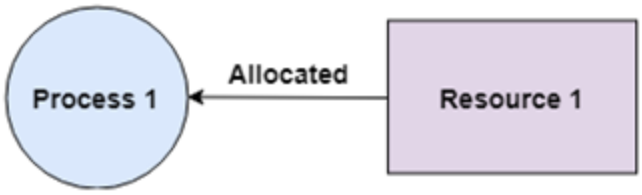
Deadlock Characterization:-

Necessary conditions for Deadlocks

1. **Mutual Exclusion**

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

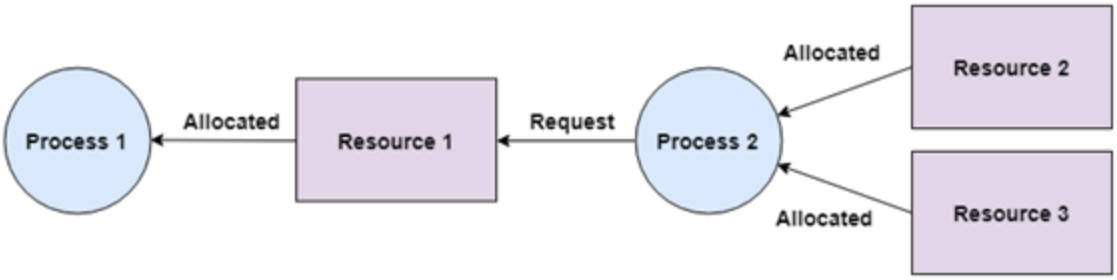
There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



2. **Hold and Wait**

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

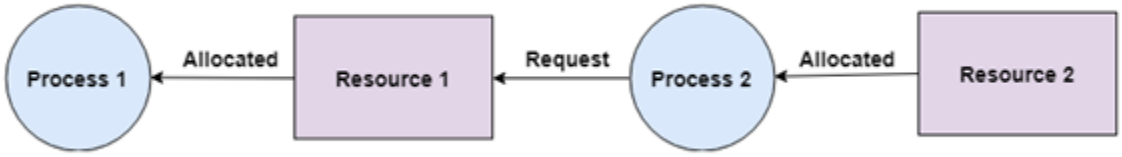
A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



3. **No preemption**

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

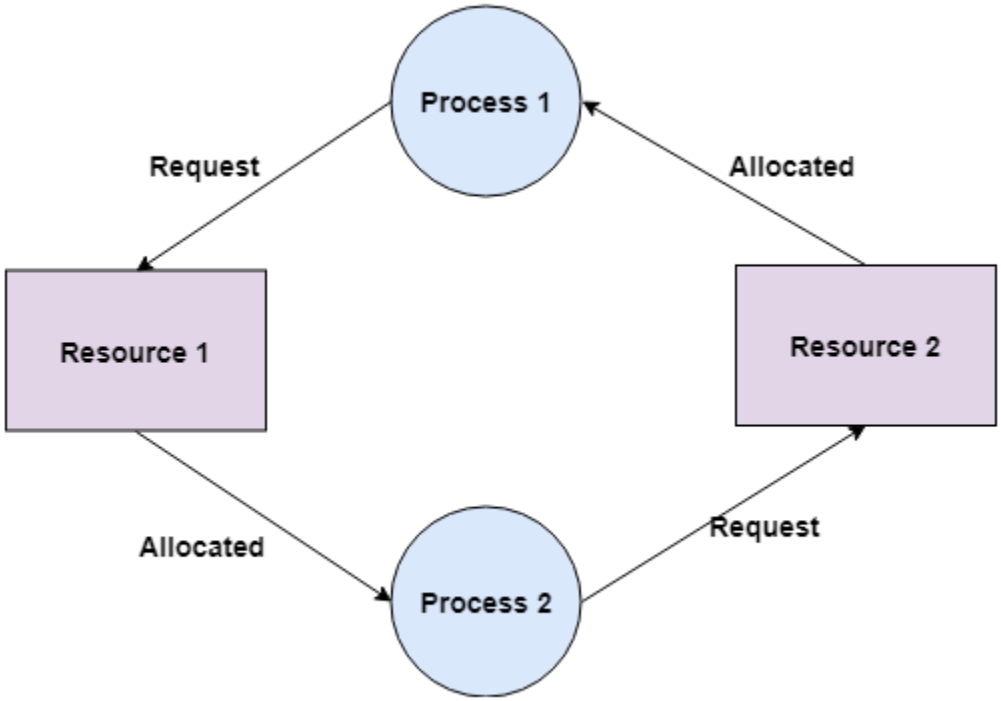
A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



4. **Circular Wait**

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

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



Methods for Handling Deadlocks

- Generally speaking there are three ways of handling deadlocks:

1. **Deadlock prevention or avoidance -** Do not allow the system to get into a deadlocked state. In order to avoid deadlocks, the system must have additional information about all processes. In particular, the system must know what resources a process will or may request in the future. (Ranging from a simple worst-case maximum to a complete resource request and release plan for each process, depending on the particular algorithm.)
2. **Deadlock detection and recovery -** Abort a process or preempt some resources when deadlocks are detected. Deadlock detection is fairly straightforward, but deadlock recovery requires either aborting processes or preempting resources, neither of which is an attractive alternative.
3. **Ignore the problem all together -** If deadlocks only occur once a year or so, it may be better to simply let them happen and reboot as necessary than to incur the constant overhead and system performance penalties associated with deadlock prevention or detection. This is the approach that both Windows and UNIX take. If deadlocks are neither prevented nor detected, then when a deadlock occurs the system will gradually slow down, as more and more processes become stuck waiting for resources currently held by the deadlock and by other waiting processes. Unfortunately this slowdown can be indistinguishable from a general system slowdown when a real-time process has heavy computing needs.

1. Deadlock Prevention

If we simulate deadlock with a table which is standing on its four legs then we can also simulate four legs with the four conditions which when occurs simultaneously, cause the deadlock.

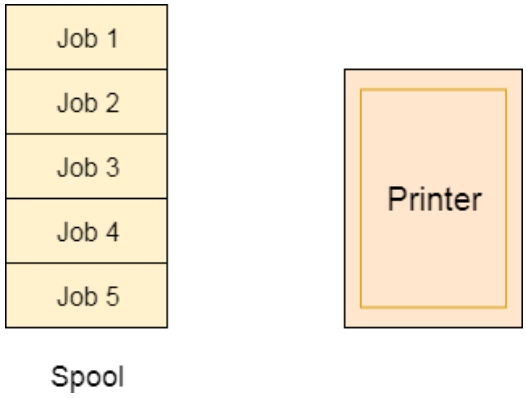
However, if we break one of the legs of the table then the table will fall definitely. The same happens with deadlock, if we can be able to violate one of the four necessary conditions and don't let them occur together then we can prevent the deadlock.

Let's see how we can prevent each of the conditions.

Mutual Exclusion

Spooling

For a device like printer, spooling can work. There is a memory associated with the printer which stores jobs from each of the process into it. Later, Printer collects all the jobs and print each one of them according to FCFS. By using this mechanism, the process doesn't have to wait for the printer and it can continue whatever it was doing. Later, it collects the output when it is produced.



Although, Spooling can be an effective approach to violate mutual exclusion but it suffers from two kinds of problems.

1. This cannot be applied to every resource.
2. After some point of time, there may arise a race condition between the processes to get space in that spool.

We cannot force a resource to be used by more than one process at the same time since it will not be fair enough and some serious problems may arise in the performance. Therefore, we cannot violate mutual exclusion for a process practically.

2. Hold and Wait

Hold and wait condition lies when a process holds a resource and waiting for some other resource to complete its task. Deadlock occurs because there can be more than one process which are holding one resource and waiting for other in the cyclic order.

However, we have to find out some mechanism by which a process either doesn't hold any resource or doesn't wait. That means, a process must be assigned all the necessary resources before the execution starts. A process must not wait for any resource once the execution has been started.

!(Hold and wait) = !hold or !wait (negation of hold and wait is, either you don't hold or you don't wait)

This can be implemented practically if a process declares all the resources initially. However, this sounds very practical but can't be done in the computer system because a process can't determine necessary resources initially.

Process is the set of instructions which are executed by the CPU. Each of the instruction may demand multiple resources at the multiple times. The need cannot be fixed by the OS.

The problem with the approach is:

1. Practically not possible.
2. Possibility of getting starved will be increases due to the fact that some process may hold a resource for a very long time.

3. No Preemption

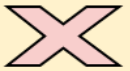
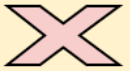
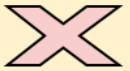
Deadlock arises due to the fact that a process can't be stopped once it starts. However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.

This is not a good approach at all since if we take a resource away which is being used by the process then all the work which it has done till now can become inconsistent.

Consider a printer is being used by any process. If we take the printer away from that process and assign it to some other process then all the data which has been printed can become inconsistent and ineffective and also the fact that the process can't start printing again from where it has left which causes performance inefficiency.

4. Circular Wait

To violate circular wait, we can assign a priority number to each of the resource. A process can't request for a lesser priority resource. This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed.

Condition	Approach	Is Practically Possible?
Mutual Exclusion	Spooling	
Hold and Wait	Request for all the resources initially	
No Preemption	Snatch all the resources	
Circular Wait	Assign priority to each resources and order resources numerically	

Among all the methods, violating Circular wait is the only approach that can be implemented practically.

Deadlock Avoidance:-

In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system. The state of the system will continuously be checked for safe and unsafe states.

In order to avoid deadlocks, the process must tell OS, the maximum number of resources a process can request to complete its execution.

The simplest and most useful approach states that the process should declare the maximum number of resources of each type it may ever need. The Deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition.

Safe and Unsafe States

The resource allocation state of a system can be defined by the instances of available and allocated resources, and the maximum instance of the resources demanded by the processes.

A state of a system recorded at some random time is shown below.

Resources Assigned

Process	Type 1	Type 2	Type 3	Type 4
A	3	0	2	2
B	0	0	1	1
C	1	1	1	0
D	2	1	4	0

Resources still needed

Process	Type 1	Type 2	Type 3	Type 4
A	1	1	0	0
B	0	1	1	2
C	1	2	1	0
D	2	1	1	2

E = (7 6 8 4)

P = (6 2 8 3)

A = (1 4 0 1)

Above tables and vector E, P and A describes the resource allocation state of a system. There are 4 processes and 4 types of the resources in a system. Table 1 shows the instances of each resource assigned to each process.

Table 2 shows the instances of the resources, each process still needs. Vector E is the representation of total instances of each resource in the system.

Vector P represents the instances of resources that have been assigned to processes. Vector A represents the number of resources that are not in use.

A state of the system is called safe if the system can allocate all the resources requested by all the processes without entering into deadlock.

If the system cannot fulfill the request of all processes then the state of the system is called unsafe.

The key of Deadlock avoidance approach is when the request is made for resources then the request must only be approved in the case if the resulting state is also a safe state.

Banker's Algorithm:-

Banker's algorithm is a **deadlock avoidance algorithm**. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not.
Consider there are `n` account holders in a bank and the sum of the money in all of their accounts is `S`. Every time a loan has to be granted by the bank, it subtracts the **loan amount** from the **total money** the bank has. Then it checks if that difference is greater than `S`. It is done because, only then, the bank would have enough money even if all the `n` account holders draw all their money at once.

Banker's algorithm works in a similar way in computers.

Whenever a new process is created, it must specify the maximum instances of each resource type that it needs, exactly.

Let us assume that there are `n` processes and `m` resource types. Some data structures that are used to implement the banker's algorithm are:

1. `Available`

It is an **array** of length `m`. It represents the number of available resources of each type. If `Available[j] = k`, then there are `k` instances available, of resource type `R(j)`.

2. `Max`

It is an `n x m` matrix which represents the maximum number of instances of each resource that a process can request. If `Max[i][j] = k`, then the process `P(i)` can request atmost `k` instances of resource type `R(j)`.

3. `Allocation`

It is an `n x m` matrix which represents the number of resources of each type currently allocated to each process. If `Allocation[i][j] = k`, then process `P(i)` is currently allocated `k` instances of resource type `R(j)`.

4. `Need`

It is an `n x m` matrix which indicates the remaining resource needs of each process. If `Need[i][j] = k`, then process `P(i)` may need `k` more instances of resource type `R(j)` to complete its task.

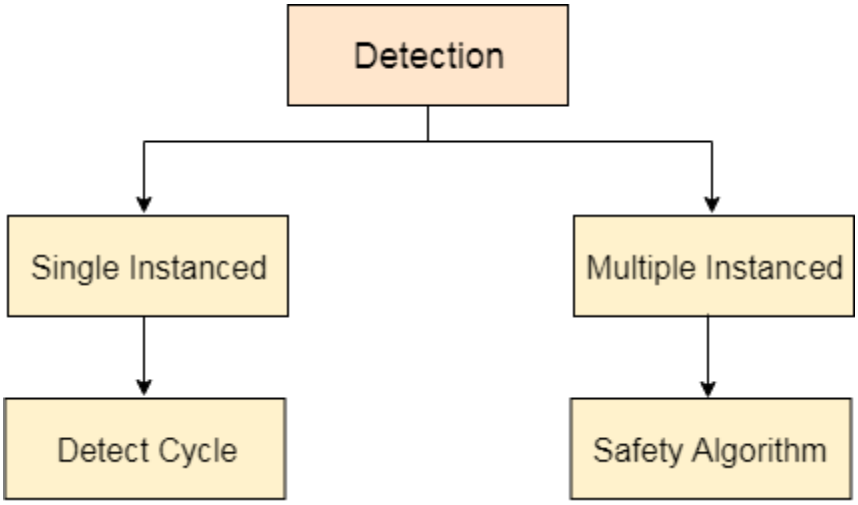
```
Need[i][j] = Max[i][j] - Allocation [i][j]
```

Deadlock Detection and Recovery

In this approach, The OS doesn't apply any mechanism to avoid or prevent the deadlocks. Therefore the system considers that the deadlock will definitely occur. In order to get rid of

deadlocks, The OS periodically checks the system for any deadlock. In case, it finds any of the deadlock then the OS will recover the system using some recovery techniques.

The main task of the OS is detecting the deadlocks. The OS can detect the deadlocks with the help of Resource allocation graph.



In single instanced resource types, if a cycle is being formed in the system then there will definitely be a deadlock. On the other hand, in multiple instanced resource type graph, detecting a cycle is not just enough. We have to apply the safety algorithm on the system by converting the resource allocation graph into the allocation matrix and request matrix.

In order to recover the system from deadlocks, either OS considers resources or processes

For Resource

Preempt the resource

We can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner. Well, choosing a resource which will be snatched is going to be a bit difficult.

Rollback to a safe state

System passes through various states to get into the deadlock state. The operating system can rollback the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state.

The moment, we get into deadlock, we will rollback all the allocations to get into the previous safe state.

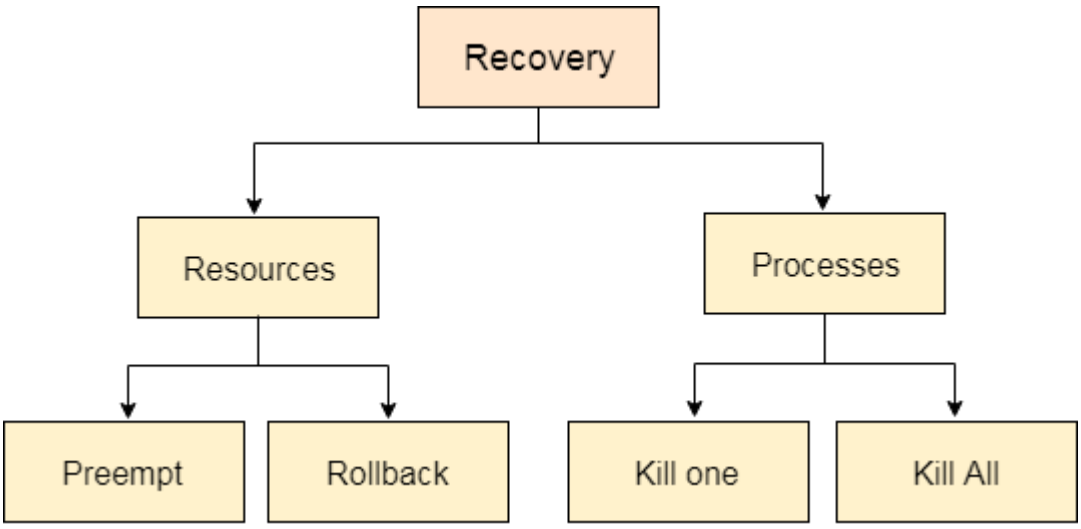
For Process

Kill a process

Killing a process can solve our problem but the bigger concern is to decide which process to kill. Generally, Operating system kills a process which has done least amount of work until now.

Kill all process

This is not a suggestible approach but can be implemented if the problem becomes very serious. Killing all process will lead to inefficiency in the system because all the processes will execute again from starting.



UNIT-4
OS

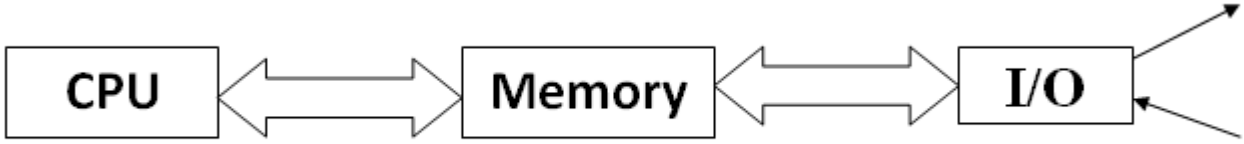
Computer Memory:-

Computer memory can be defined as a collection of some data represented in the binary format. A computer device that is capable to store any information or data temporally or permanently is called storage device.

Storage Management:-

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

The memory management method deals with allocation of finite amount of memory to the requesting process. In ready state it is necessary that the process should have access to the certain amount of memory. Memory is a long array of bytes. Each with its own address using read and write statement the CPU and input/output device interact with the memory.



One of the main tasks of an operating system is to manage the computer's memory. This includes many responsibilities, including

- Being aware of what parts of the memory are in use and which parts are not.

- Allocating memory to processes when they request it and de-allocating memory when a process releases its memory.
- Moving data from memory to disc, when the physical capacity becomes full, and vice versa.

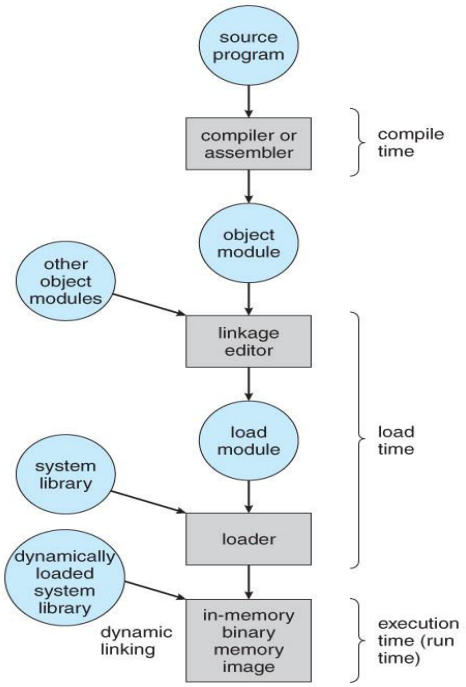
Address:- It is two types

1. **Logical address:-** Logical address are generated by the CPU. These address are defined by CPU while writing any program generated by CPU.
2. **Physical address:-** The address of memory area where the user program actually resides is called Physical address.

Address Binding Mechanism:- Memory consist of large array of words or bytes. Each with its own address. The processor is to select one of the process from the queue and load into memory as the process is executed. It access data and program from the memory.

User programs typically refer to memory addresses with symbolic names such as "i", "count", and "average Temperature". These symbolic names must be mapped or bound to physical memory addresses, which typically occurs in several stages:

Diagram shows the various stages of the binding processes and the units involved in each stage:



Multistep processing of a user program

Compile Time - If it is known at compile time where a program will reside in physical memory, then absolute code can be generated by the compiler, containing actual physical addresses. However if the load address changes at some later time, then the program will have to be recompiled. DOS .COM programs use compile time binding.

Load Time - If the location at which a program will be loaded is not known at compile time, then the compiler must generate relocatable code, which references addresses relative to the start of the program. If that starting address changes, then the program must be reloaded but not recompiled.

Execution Time - If a program can be moved around in memory during the course of its execution, then binding must be delayed until execution time. This requires special hardware, and is the method implemented by most modern Operating System .

Dynamic loading:- Size of the process depend on the size of physical memory. Hence to obtain the better utilization of the memory space dynamic loading is performed. The major advantage of dynamic loading is that it never load any unused process. This method is useful while handling the large amount of code.

Swapping:-

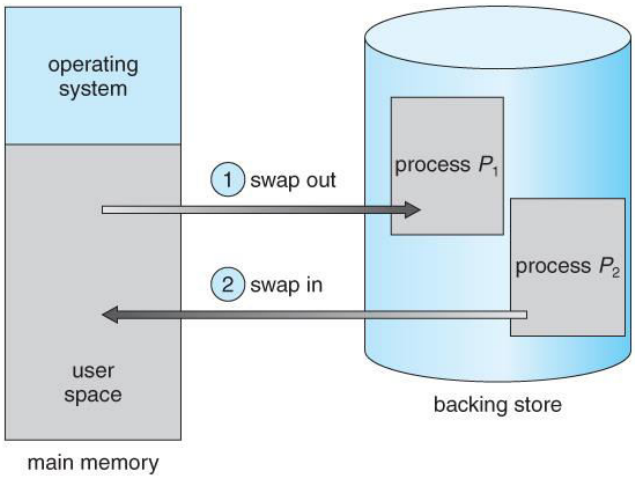
Swapping is a technique for making memory compact. It is a mechanism that is used to temporarily swap processes out of the main memory to secondary memory, and this makes more memory available for some other processes. At some later time, the system can swap back the process from the secondary memory to the main memory.

Swapping does affect the performance of the system, but it helps in running multiple processes parallelly. The total time taken by the swapping of a process includes the time it takes to move the entire process to the secondary memory and then again to the main memory.

It is a method of taking out the current content of memory to backstore(disk) and bring the content of backstore to main memory.

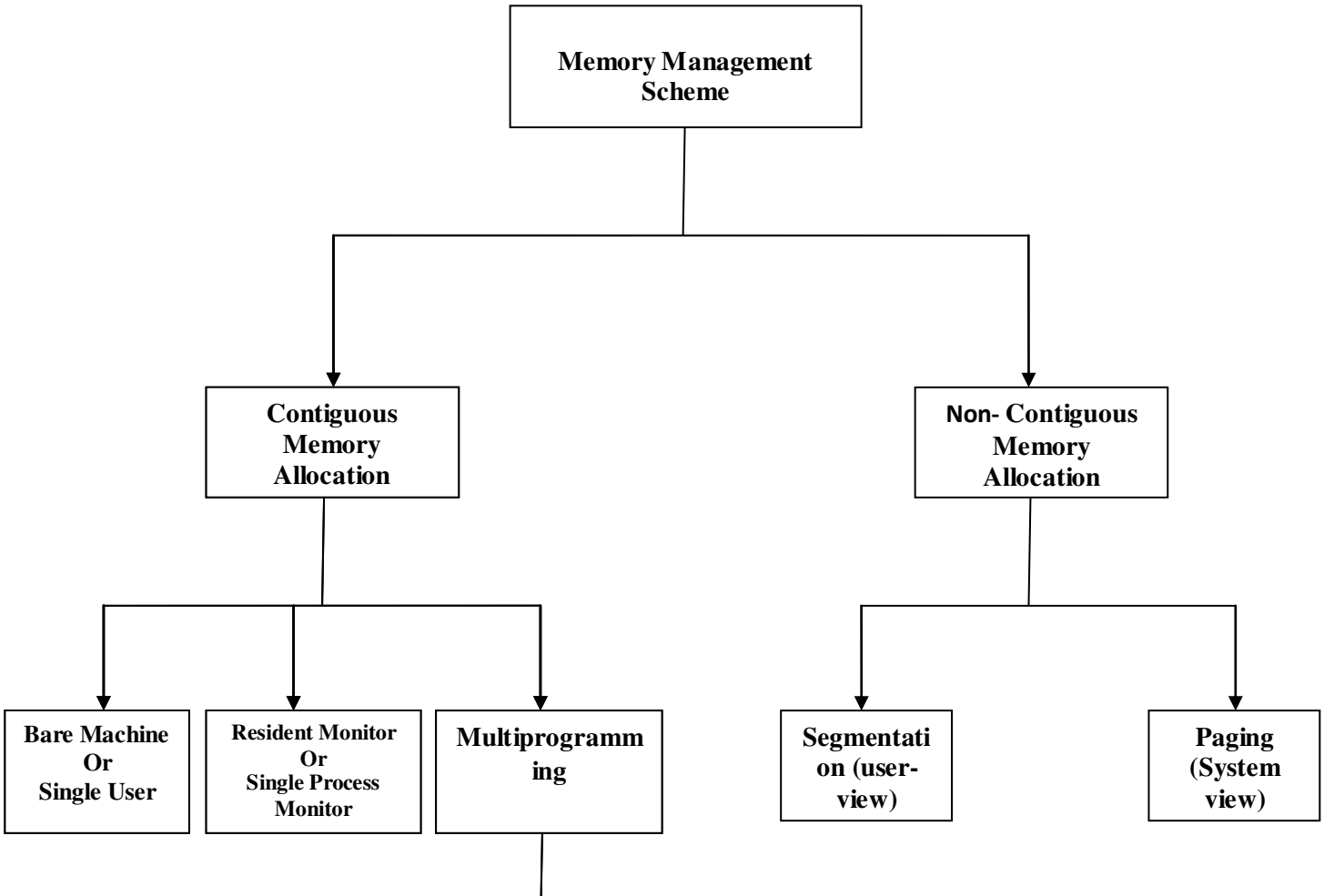
There are two operations in swapping method:-

1. Swap out(Read out):- take out to the current data from the main memory.
2. Swap in(Read in):- bring the data of new user into main memory.



Swapping of two processes using a disk as a backing store

Memory Management Scheme:-



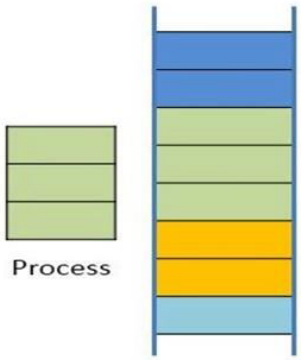
Contiguous Memory Allocation:-

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

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

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

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



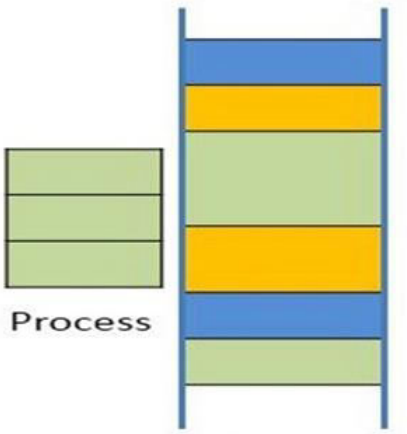
Contiguous Memory Allocation

Non-contiguous memory allocation:-

In the **non-contiguous memory allocation** the available free memory space are scattered here and there and all the free memory space is not at one place. So this is time-consuming.

In the **non-contiguous memory allocation**, a process will acquire the memory space but it is not at one place it is at the different locations according to the process requirement.

This technique of **non-contiguous memory allocation** reduces the wastage of memory which leads to internal and external fragmentation. This utilizes all the free memory space which is created by a different process.



Noncontiguous Memory Allocation

Difference between Contiguous and Non-contiguous Memory Allocation :

S.NO.	CONTIGUOUS MEMORY ALLOCATION	NON-CONTIGUOUS MEMORY ALLOCATION
1.	Contiguous memory allocation allocates consecutive blocks of memory to a file/process.	Non-Contiguous memory allocation allocates separate blocks of memory to a file/process.
2.	Faster in Execution.	Slower in Execution.
3.	It is easier for the OS to control.	It is difficult for the OS to control.
4.	Overhead is minimum as not much address translations are there while executing a process.	More Overheads are there as there are more address translations.
5.	Internal fragmentation occurs in Contiguous memory allocation method.	External fragmentation occurs in Non-Contiguous memory allocation method.
6.	It includes single partition allocation and multi-partition allocation.	It includes paging and segmentation.
7.	Wastage of memory is there.	No memory wastage is there.
8.	In contiguous memory allocation, swapped-in processes are arranged in the originally allocated space.	In non-contiguous memory allocation, swapped-in processes can be arranged in any place in the memory.

Multiprogramming:-

In the multiprogramming, the multiple users can share the memory simultaneously. By multiprogramming we mean there will be more than one process in the main memory and if the running process wants to wait for an event like I/O then instead of sitting ideal CPU will make a context switch and will pick another process.

Multiprogramming are two types:-

1. Fixed (Static)

2. Variable (Dynamic)

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

In this technique, the main memory is divided into partitions of equal or different sizes. The operating system always resides in the first partition while the other partitions can be used to store user processes. The memory is assigned to the processes in contiguous way.

In fixed partitioning,

1. The partitions cannot overlap.
2. A process must be contiguously present in a partition for the execution.

There are various cons of using this technique.

1. Internal Fragmentation

If the size of the process is lesser then the total size of the partition then some size of the partition get wasted and remain unused. This is wastage of the memory and called internal fragmentation.

As shown in the image below, the 4 MB partition is used to load only 3 MB process and the remaining 1 MB got wasted.

2. External Fragmentation

The total unused space of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form.

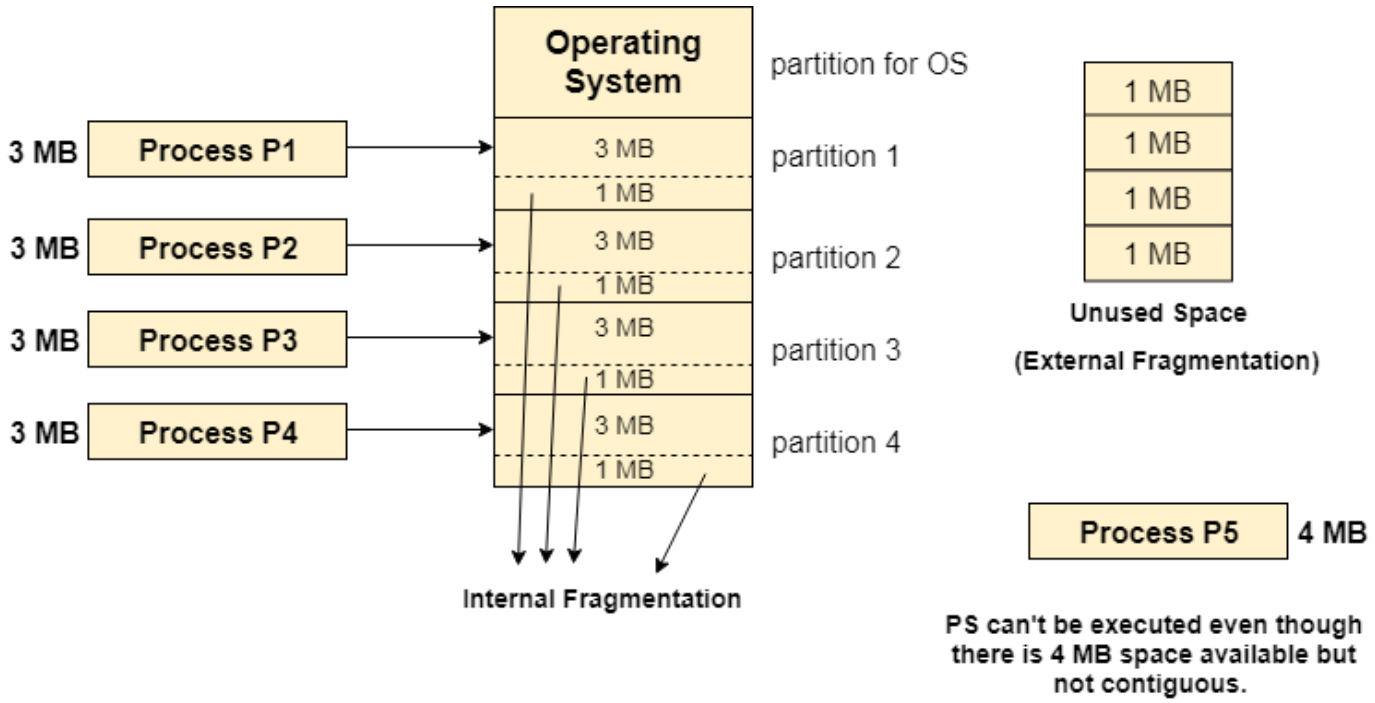
As shown in the image below, the remaining 1 MB space of each partition cannot be used as a unit to store a 4 MB process. Despite of the fact that the sufficient space is available to load the process, process will not be loaded.

3. Limitation on the size of the process

If the process size is larger than the size of maximum sized partition then that process cannot be loaded into the memory. Therefore, a limitation can be imposed on the process size that is it cannot be larger than the size of the largest partition.

4. Degree of multiprogramming is less

By Degree of multi programming, we simply mean the maximum number of processes that can be loaded into the memory at the same time. In fixed partitioning, the degree of multiprogramming is fixed and very less due to the fact that the size of the partition cannot be varied according to the size of processes.



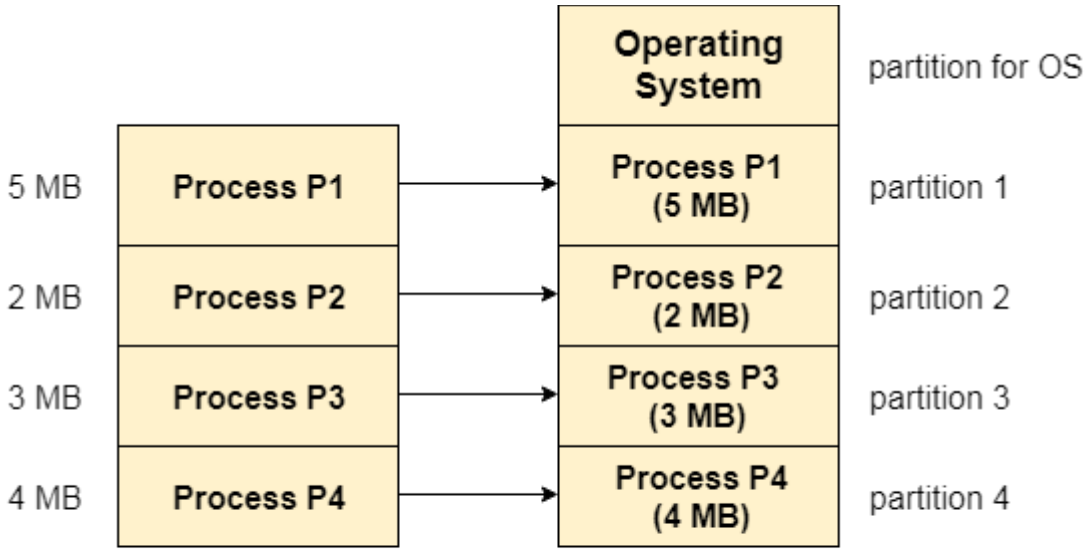
Fixed Partitioning

(Contiguous memory allocation)

Variable (Dynamic) Partitioning:-

Dynamic partitioning tries to overcome the problems caused by fixed partitioning. In this technique, the partition size is not declared initially. It is declared at the time of process loading.

The first partition is reserved for the operating system. The remaining space is divided into parts. The size of each partition will be equal to the size of the process. The partition size varies according to the need of the process so that the internal fragmentation can be avoided.



Dynamic Partitioning

(Process Size = Partition Size)

Advantages of Dynamic Partitioning over fixed partitioning

1. No Internal Fragmentation

Given the fact that the partitions in dynamic partitioning are created according to the need of the process, It is clear that there will not be any internal fragmentation because there will not be any unused remaining space in the partition.

2. No Limitation on the size of the process

In Fixed partitioning, the process with the size greater than the size of the largest partition could not be executed due to the lack of sufficient contiguous memory. Here, In Dynamic partitioning, the process size can't be restricted since the partition size is decided according to the process size.

3. Degree of multiprogramming is dynamic

Due to the absence of internal fragmentation, there will not be any unused space in the partition hence more processes can be loaded in the memory at the same time.

Disadvantages of dynamic partitioning

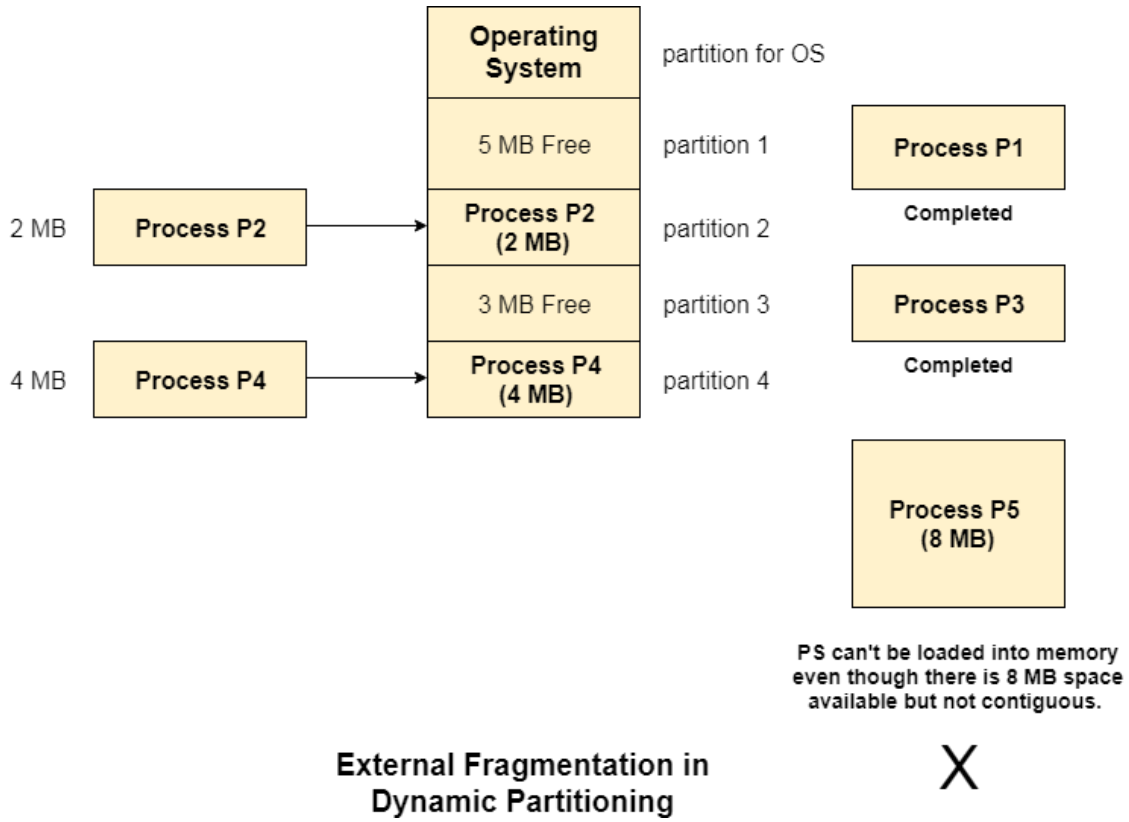
External Fragmentation

Absence of internal fragmentation doesn't mean that there will not be external fragmentation.

Let's consider three processes P1 (1 MB) and P2 (3 MB) and P3 (1 MB) are being loaded in the respective partitions of the main memory.

After some time P1 and P3 got completed and their assigned space is freed. Now there are two unused partitions (1 MB and 1 MB) available in the main memory but they cannot be used to load a 2 MB process in the memory since they are not contiguously located.

The rule says that the process must be contiguously present in the main memory to get executed. We need to change this rule to avoid external fragmentation.



File System

File system is the part of the operating system which is responsible for file management. It provides a mechanism to store the data and access to the file contents including data and programs. Some Operating systems treats everything as a file for example Ubuntu.

A file is a collection of correlated information which is recorded on secondary or non-volatile storage like magnetic disks, optical disks, and tapes. It is a method of data collection that is used as a medium for giving input and receiving output from that program.

In general, a file is a sequence of bits, bytes, or records whose meaning is defined by the file creator and user. Every File has a logical location where they are located for storage and retrieval.

The File system takes care of the following issues

- **File Structure**

We have seen various data structures in which the file can be stored. The task of the file system is to maintain an optimal file structure.

- **Recovering Free space**

  Whenever a file gets deleted from the hard disk, there is a free space created in the disk. There can be many such spaces which need to be recovered in order to reallocate them to other files.

- **Disk space assignment to the files**

  The major concern about the file is deciding where to store the files on the hard disk. There are various disks scheduling algorithm.

- **Tracking data location**

  A File may or may not be stored within only one block. It can be stored in the non contiguous blocks on the disk. We need to keep track of all the blocks on which the part of the files reside.

File Access Methods in Operating System

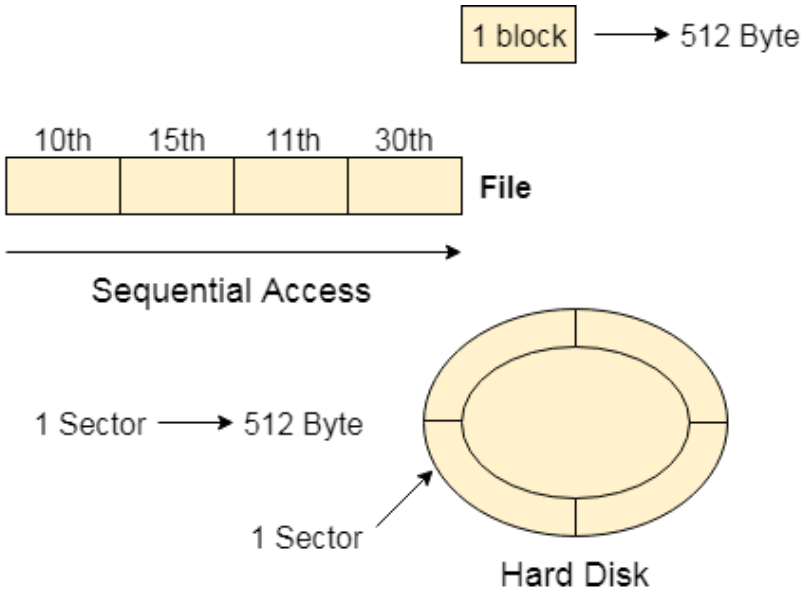
When a file is used, information is read and accessed into computer memory and there are several ways to access this information of the file. Some systems provide only one access method for files. Other systems, such as those of IBM, support many access methods, and choosing the right one for a particular application is a major design problem.

There are three ways to access a file into a computer system: Sequential-Access, Direct Access, Index sequential Method.

Sequential Access –

It is the simplest access method. Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editor and compiler usually access the file in this fashion.

Read and write make up the bulk of the operation on a file. A read operation *-read next-* read the next position of the file and automatically advance a file pointer, which keeps track I/O location. Similarly, for the write *write next* append to the end of the file and advance to the newly written material.



Most of the operating systems access the file sequentially. In other words, we can say that most of the files need to be accessed sequentially by the operating system.

In sequential access, the OS read the file word by word. A pointer is maintained which initially points to the base address of the file. If the user wants to read first word of the file then the pointer provides that word to the user and increases its value by 1 word. This process continues till the end of the file.

Key points:

- Data is accessed one record right after another record in an order.
- When we use read command, it move ahead pointer by one
- When we use write command, it will allocate memory and move the pointer to the end of the file
- Such a method is reasonable for tape.

Direct Access –

Another method is *direct access method* also known as *relative access method*. A filed-length logical record that allows the program to read and write record rapidly. in no particular order. The direct access is based on the disk model of a file since disk allows random access to any file block. For direct access, the file is viewed as a numbered sequence of block or record. Thus, we

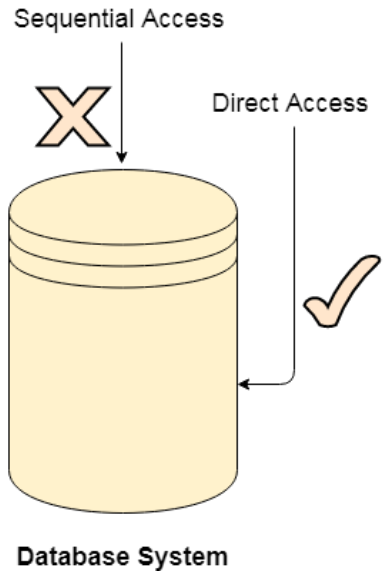
may read block 14 then block 59 and then we can write block 17. There is no restriction on the order of reading and writing for a direct access file.

A block number provided by the user to the operating system is normally a *relative block number*, the first relative block of the file is 0 and then 1 and so on.

The Direct Access is mostly required in the case of database systems. In most of the cases, we need filtered information from the database. The sequential access can be very slow and inefficient in such cases.

Suppose every block of the storage stores 4 records and we know that the record we needed is stored in 10th block. In that case, the sequential access will not be implemented because it will traverse all the blocks in order to access the needed record.

Direct access will give the required result despite of the fact that the operating system has to perform some complex tasks such as determining the desired block number. However, that is generally implemented in database applications.



Index sequential method –

It is the other method of accessing a file which is built on the top of the direct access method. These methods construct an index for the file. The index, like an index in the back of a book, contains the pointer to the various blocks. To find a record in the file, we first search the index and then by the help of pointer we access the file directly.

- Provides solutions to problems of contiguous and linked allocation.
- A index block is created having all pointers to files.

- Each file has its own index block which stores the addresses of disk space occupied by the file.
- Directory contains the addresses of index blocks of files.

Key points:

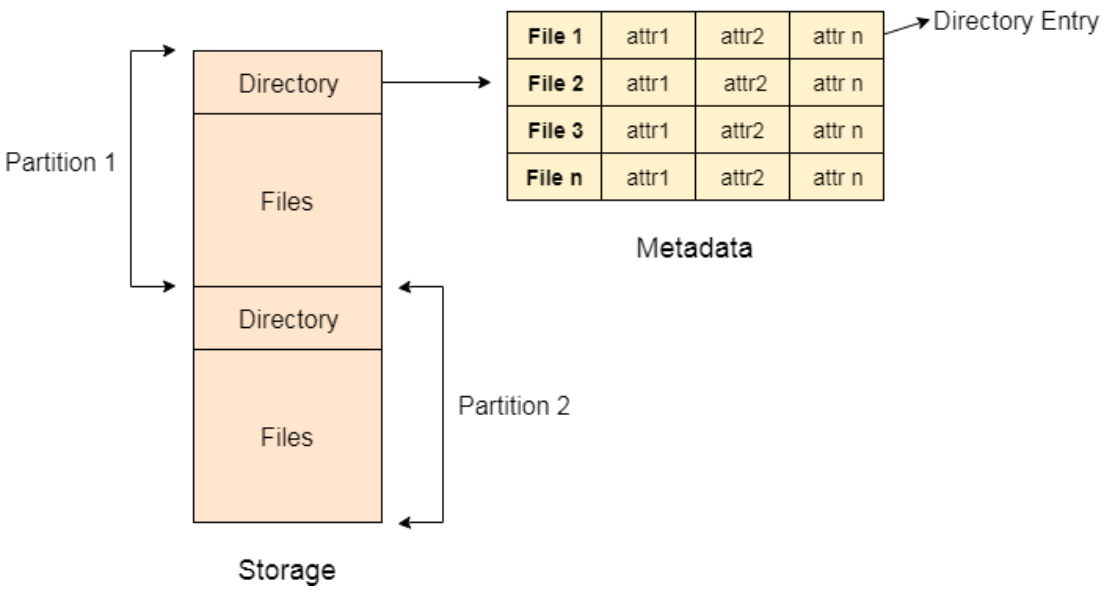
- It is built on top of Sequential access.
- It control the pointer by using index.

Directory Structure:-

Directory can be defined as the listing of the related files on the disk. The directory may store some or the entire file attributes.

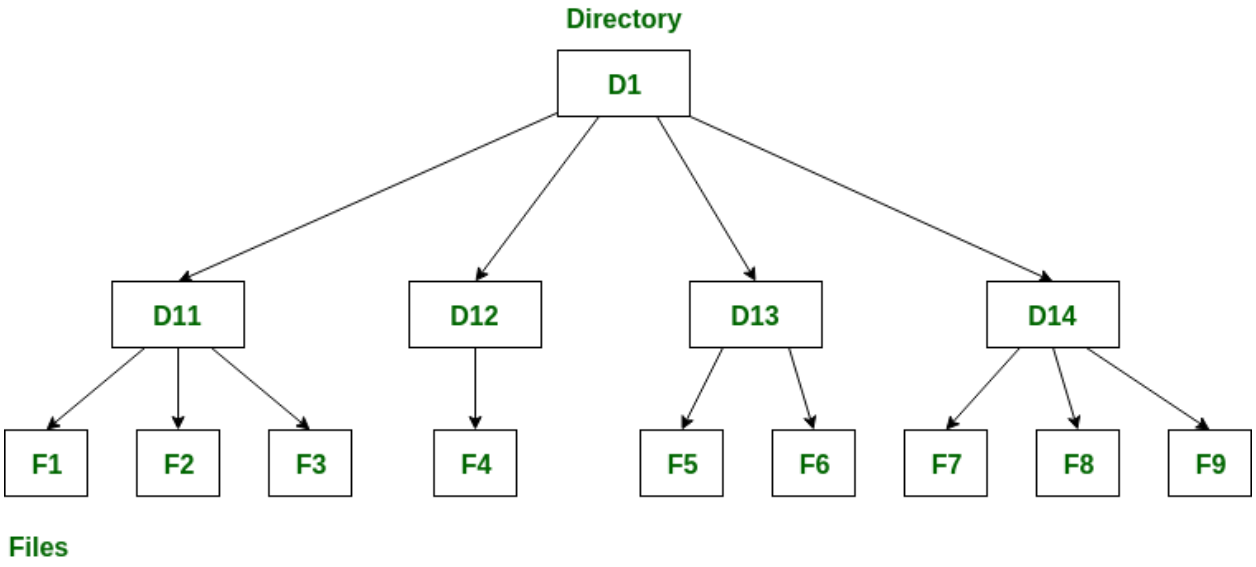
To get the benefit of different file systems on the different operating systems, A hard disk can be divided into the number of partitions of different sizes. The partitions are also called volumes or mini disks.

Each partition must have at least one directory in which, all the files of the partition can be listed. A directory entry is maintained for each file in the directory which stores all the information related to that file.



A directory can be viewed as a file which contains the Meta data of the bunch of files.

A **directory** is a container that is used to contain folders and file. It organizes files and folders into a hierarchical manner.

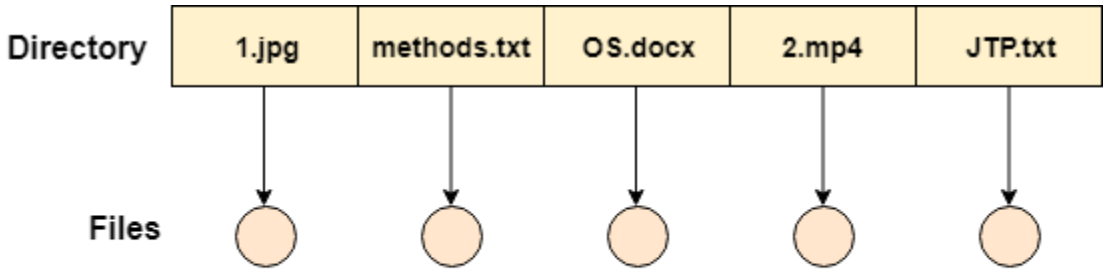


Every Directory supports a number of common operations on the file:

1. File Creation
2. Search for the file
3. File deletion
4. Renaming the file
5. Traversing Files
6. Listing of files

1. Single Level Directory

The simplest method is to have one big list of all the files on the disk. The entire system will contain only one directory which is supposed to mention all the files present in the file system. The directory contains one entry per each file present on the file system.



Single Level Directory

This type of directories can be used for a simple system.

Advantages

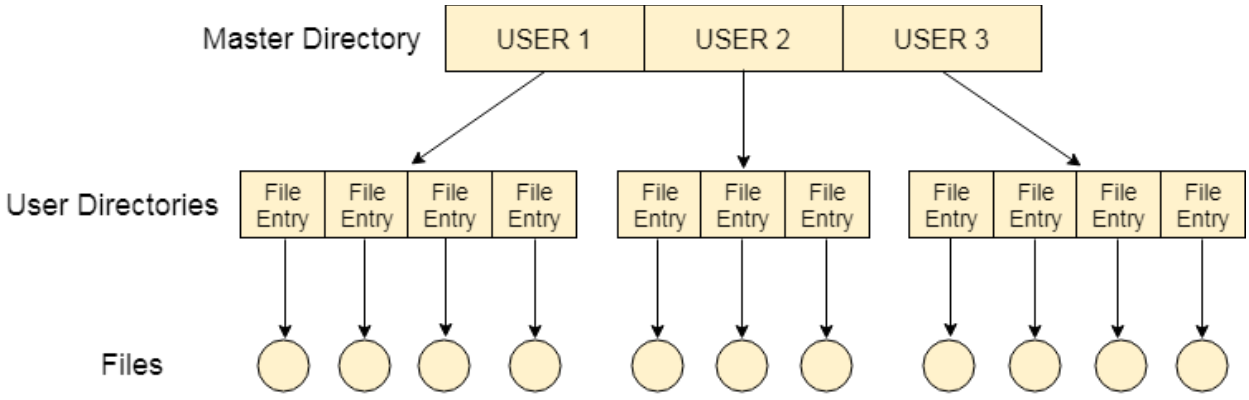
1. Implementation is very simple.
2. If the sizes of the files are very small then the searching becomes faster.
3. File creation, searching, deletion is very simple since we have only one directory.

Disadvantages

1. We cannot have two files with the same name.
2. The directory may be very big therefore searching for a file may take so much time.
3. Protection cannot be implemented for multiple users.
4. There are no ways to group same kind of files.
5. Choosing the unique name for every file is a bit complex and limits the number of files in the system because most of the Operating System limits the number of characters used to construct the file name.

Two Level Directory

In two level directory systems, we can create a separate directory for each user. There is one master directory which contains separate directories dedicated to each user. For each user, there is a different directory present at the second level, containing group of user's file. The system doesn't let a user to enter in the other user's directory without permission.



Two Level Directory

Characteristics of two level directory system

1. Each files has a path name as ***/User-name/directory-name/***
2. Different users can have the same file name.
3. Searching becomes more efficient as only one user's list needs to be traversed.
4. The same kind of files cannot be grouped into a single directory for a particular user.

Every Operating System maintains a variable as **PWD** which contains the present directory name (present user name) so that the searching can be done appropriately.

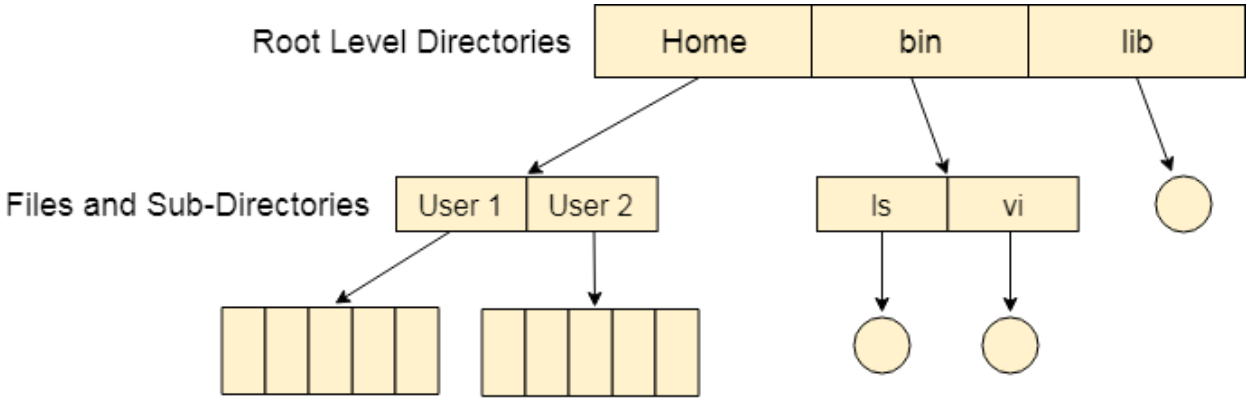
Tree Structured Directory

In Tree structured directory system, any directory entry can either be a file or sub directory. Tree structured directory system overcomes the drawbacks of two level directory system. The similar kind of files can now be grouped in one directory.

Each user has its own directory and it cannot enter in the other user's directory. However, the user has the permission to read the root's data but he cannot write or modify this. Only administrator of the system has the complete access of root directory.

Searching is more efficient in this directory structure. The concept of current working directory is used. A file can be accessed by two types of path, either relative or absolute.

Absolute path is the path of the file with respect to the root directory of the system while relative path is the path with respect to the current working directory of the system. In tree structured directory systems, the user is given the privilege to create the files as well as directories.



The Structured Directory System

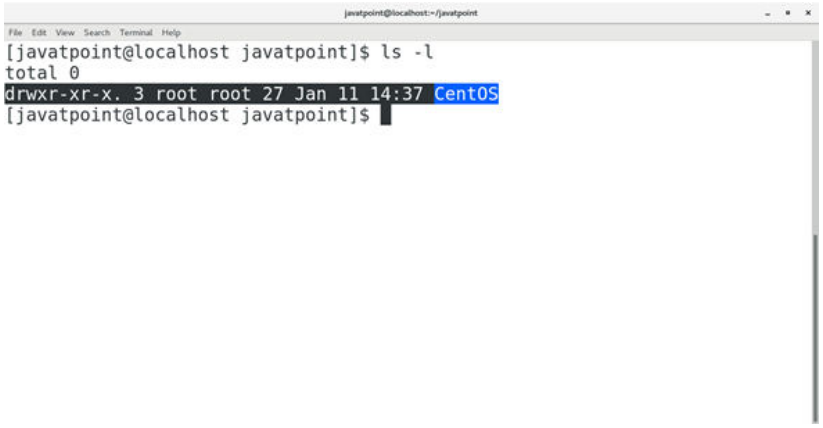
Permissions on the file and directory

A tree structured directory system may consist of various levels therefore there is a set of permissions assigned to each file and directory.

The permissions are **R W X** which are regarding reading, writing and the execution of the files or directory. The permissions are assigned to three types of users: owner, group and others.

There is a identification bit which differentiate between directory and file. For a directory, it is **d** and for a file, it is dot **(.)**

The following snapshot shows the permissions assigned to a file in a Linux based system. Initial bit **d** represents that it is a directory.



Acyclic-Graph Structured Directories

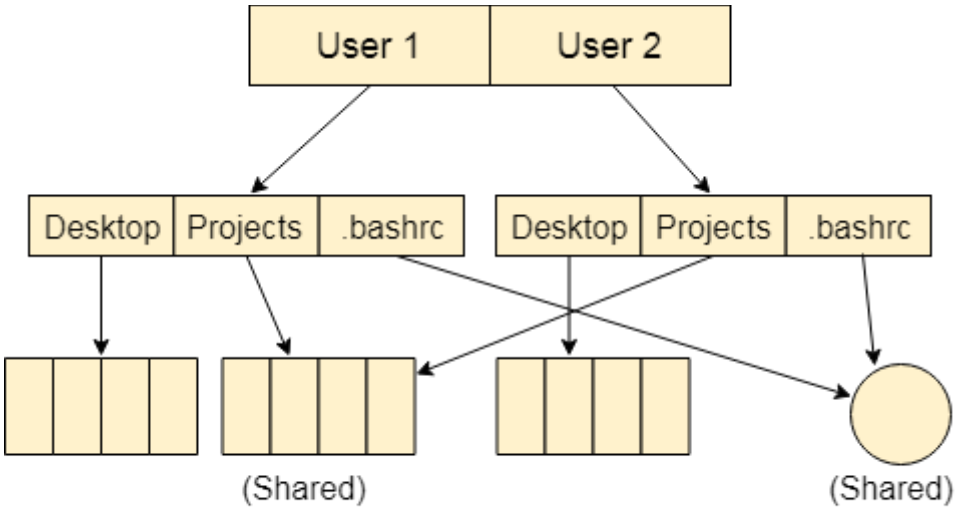
The tree structured directory system doesn't allow the same file to exist in multiple directories therefore sharing is major concern in tree structured directory system. We can provide sharing by making the directory an acyclic graph. In this system, two or more directory entry can point to the same file or sub directory. That file or sub directory is shared between the two directory entries.

These kinds of directory graphs can be made using links or aliases. We can have multiple paths for a same file. Links can either be symbolic (logical) or hard link (physical).

If a file gets deleted in acyclic graph structured directory system, then

1. In the case of soft link, the file just gets deleted and we are left with a dangling pointer.

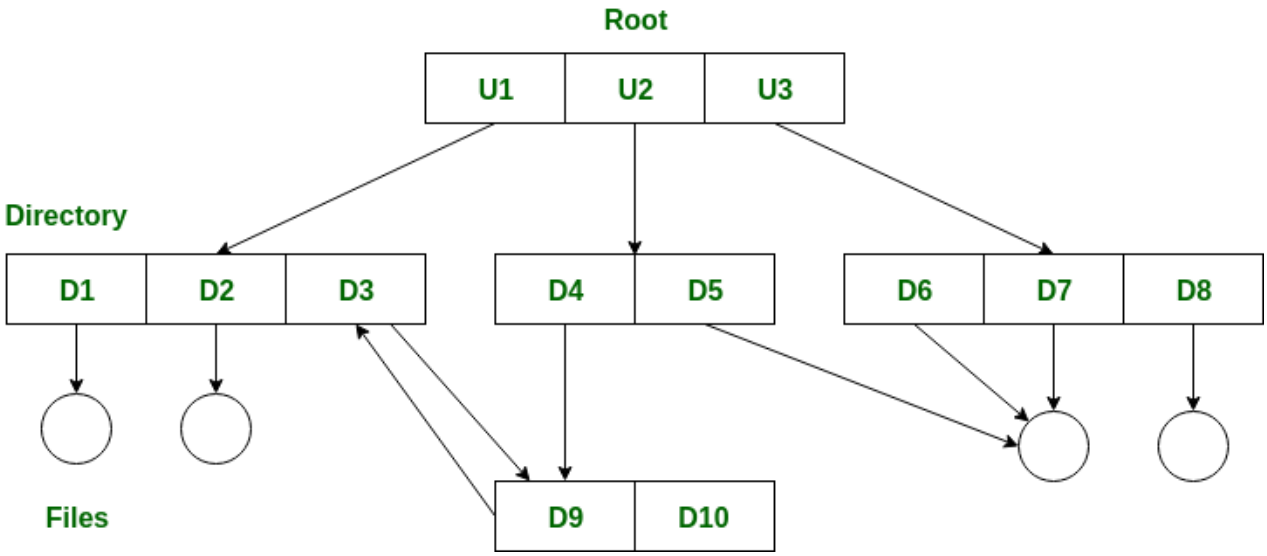
2. In the case of hard link, the actual file will be deleted only if all the references to it gets deleted.



Acyclic-Graph Structured Directory System

General graph directory structure –

In general graph directory structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory. The main problem with this kind of directory structure is to calculate total size or space that has been taken by the files and directories.



Advantages:

- It allows cycles.
- It is more flexible than other directories structure.

Disadvantages:

- It is more costly than others.
- It needs garbage collection.

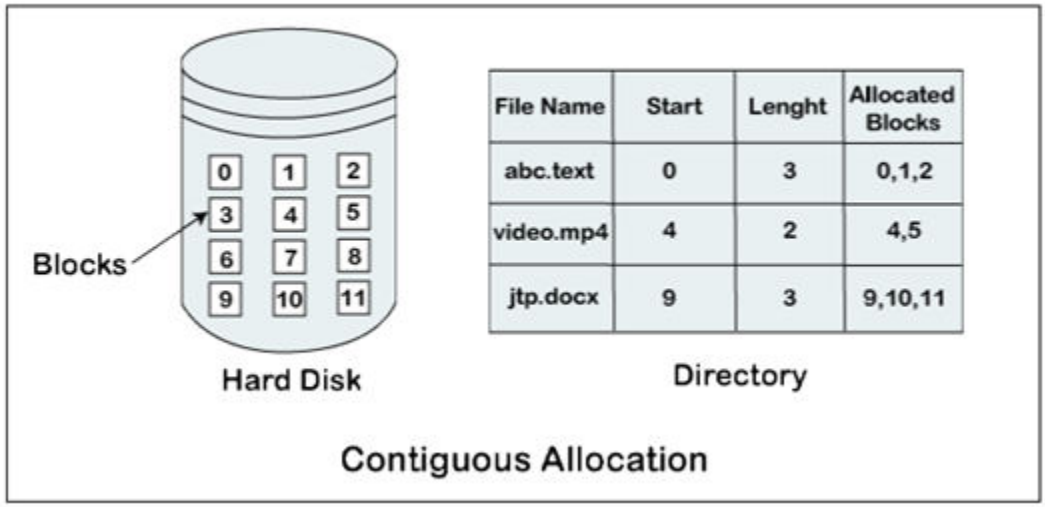
Structures of Directory in Operating System

Allocation Method

The allocation method defines how the files are stored in the disk blocks. The direct access nature of the disks gives us the flexibility to implement the files. In many cases, different files or many files are stored on the same disk. The main problem that occurs in the operating system is that how we allocate the spaces to these files so that the utilization of disk is efficient and the quick access to the file is possible. There are mainly three methods of file allocation in the disk. Each method has its advantages and disadvantages. Mainly a system uses one method for all files within the system.

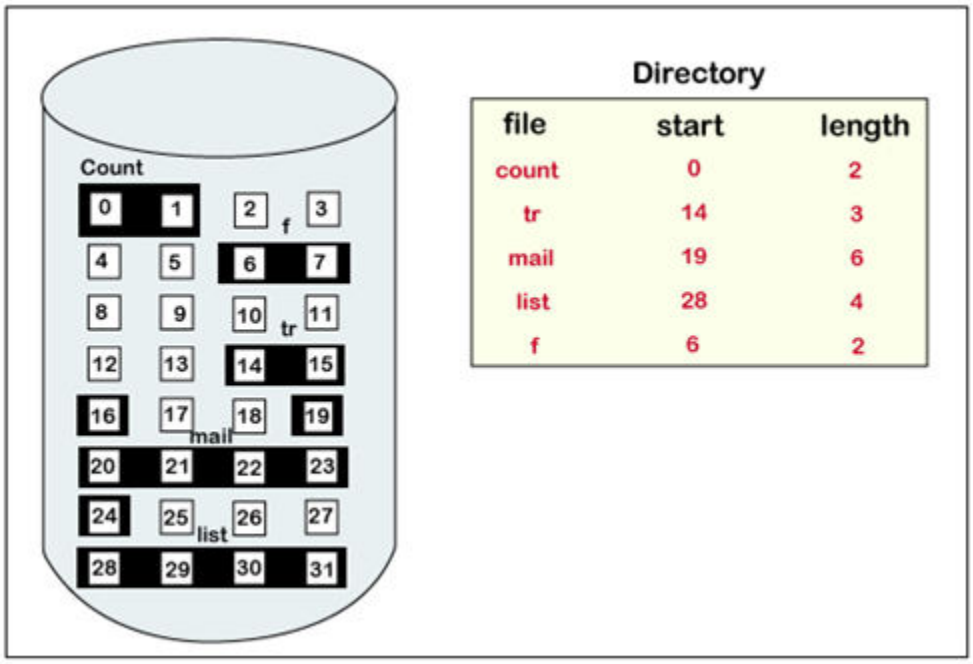
Contiguous Allocation: – Contiguous allocation is one of the most used methods for allocation. Contiguous allocation means we allocate the block in such a manner, so that in the hard disk, all the blocks get the contiguous physical block.

We can see in the below figure that in the directory, we have three files. In the table, we have mentioned the starting block and the length of all the files. We can see in the table that for each file, we allocate a contiguous block.



Example of contiguous allocation

We can see in the given diagram, that there is a file. The name of the file is 'mail.' The file starts from the 19th block and the length of the file is 6. So, the file occupies 6 blocks in a contiguous manner. Thus, it will hold blocks 19, 20, 21, 22, 23, 24.



Advantages of Contiguous Allocation

The advantages of contiguous allocation are:

1. The contiguous allocation method gives excellent read performance.
2. Contiguous allocation is easy to implement.
3. The contiguous allocation method supports both types of file access methods that are sequential access and direct access.
4. The Contiguous allocation method is fast because, in this method number of seeks is less due to the contiguous allocation of file blocks.

Disadvantages of Contiguous allocation

The disadvantages of contiguous allocation method are:

1. In the contiguous allocation method, sometimes disk can be fragmented.

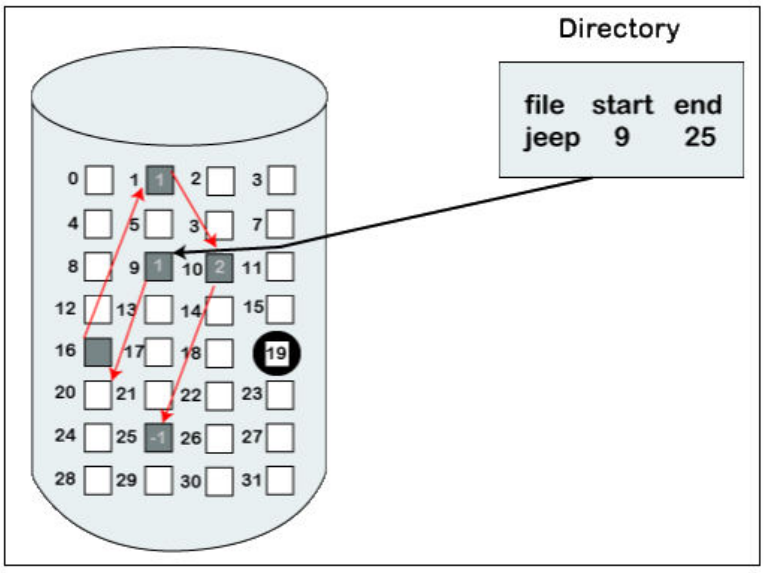
2. In this method, it is difficult to increase the size of the file due to the availability of the contiguous memory block.

Linked List Allocation

The linked list allocation method overcomes the drawbacks of the contiguous allocation method. In this file allocation method, each file is treated as a linked list of disks blocks. In the linked list allocation method, it is not required that disk blocks assigned to a specific file are in the contiguous order on the disk. The directory entry comprises of a pointer for starting file block and also for the ending file block. Each disk block that is allocated or assigned to a file consists of a pointer, and that pointer point the next block of the disk, which is allocated to the same file.

Example of linked list allocation

We can see in the below figure that we have a file named 'jeep.' The value of the start is 9. So, we have to start the allocation from the 9th block, and blocks are allocated in a random manner. The value of the end is 25. It means the allocation is finished on the 25th block. We can see in the below figure that the block (25) comprised of -1, which means a null pointer, and it will not point to another block.



Advantages of Linked list allocation

There are various advantages of linked list allocation:

1. In liked list allocation, there is no external fragmentation. Due to this, we can utilize the memory better.

2. In linked list allocation, a directory entry only comprises of the starting block address.

3. The linked allocation method is flexible because we can quickly increase the size of the file because, in this to allocate a file, we do not require a chunk of memory in a contiguous form.

Disadvantages of Linked list Allocation

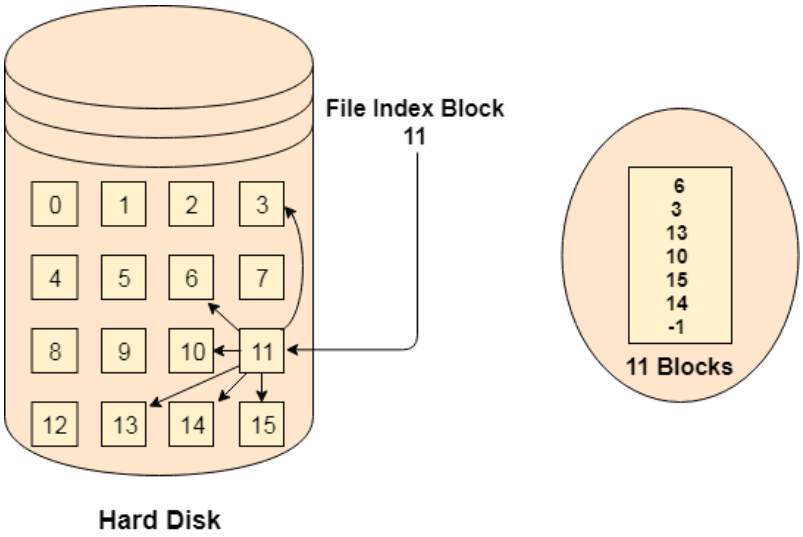
There are various disadvantages of linked list allocation:

1. Linked list allocation does not support direct access or random access.
2. In linked list allocation, we need to traverse each block.
3. If the pointer in the linked list break in linked list allocation, then the file gets corrupted.
4. In the disk block for the pointer, it needs some extra space.

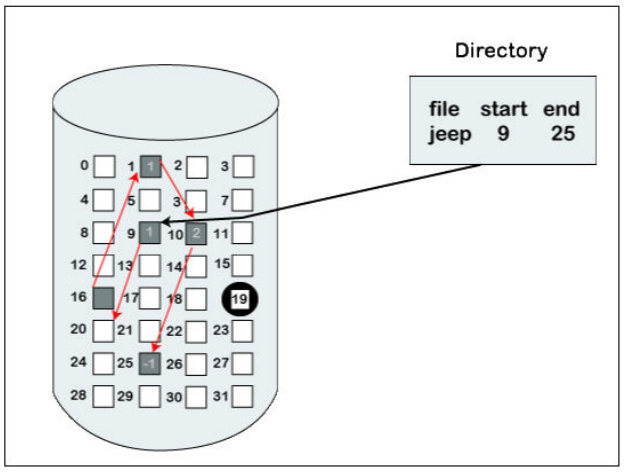
Indexed Allocation

The Indexed allocation method is another method that is used for file allocation. In the index allocation method, we have an additional block, and that block is known as the index block. For each file, there is an individual index block. In the index block, the ith entry holds the disk address of the ith file block. We can see in the below figure that the directory entry comprises of the address of the index block.

Instead of maintaining a file allocation table of all the disk pointers, Indexed allocation scheme stores all the disk pointers in one of the blocks called as indexed block. Indexed block doesn't hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file. Directory entry will only contain the index block address.



or



Advantages of Index Allocation

The advantages of index allocation are:

1. The index allocation method solves the problem of external fragmentation.
2. Index allocation provides direct access.

Disadvantages of Index Allocation

The disadvantages of index allocation are:

1. In index allocation, pointer overhead is more.
2. We can lose the entire file if an index block is not correct.
3. It is totally a wastage to create an index for a small file.

A single index block cannot hold all the pointer for files with large sizes.

To resolve this problem, there are various mechanism which we can use:

1. Linked scheme
2. Multilevel Index
3. Combined Scheme

1. **Linked Scheme: –** In the linked scheme, to hold the pointer, two or more than two index blocks are linked together. Each block contains the address of the next index block or a pointer.
2. **Multilevel Index: –** In the multilevel index, to point the second-level index block, we use a first-level index block that in turn points to the blocks of the disk, occupied by the file. We can extend this up to 3 or more than 3 levels depending on the maximum size of the file.
3. **Combined Scheme: –** In a combined scheme, there is a special block which is called an information node (Inode). The inode comprises of all the information related to the file like authority, name, size, etc. To store the disk block addresses that contain the actual file, the remaining space of inode is used. In inode, the starting pointer is used to point the direct blocks. This means the pointer comprises of the addresses of the disk blocks, which consist of the file data. To indicate the indirect blocks, the next few pointers are used. The indirect blocks are of three types, which are single indirect, double indirect, and triple indirect.

Protection in File System:-

In computer systems, alot 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. There are many protection mechanism. each of them mechanism have different advantages and disadvantages and must be appropriate for the intended application.

Access Control :

There are different methods used by different users to access any file. The general way of protection is to associate *identity-dependent access* with all the files and directories an list called access-control list (ACL) which specify the names of the users and the types of access associate with each of the user. The main problem with the access list is their length. If we want to allow everyone to read a file, we must list all the users with the read access. This technique has two undesirable consequences:

Constructing such a list may be tedious and unrewarding task, especially if we do not know in advance the list of the users in the system.

Previously, the entry of the any directory is of the fixed size but now it changes to the variable size which results in the complicates space management. These problems can be resolved by use of a condensed version of the access list. To condense the length of the access-control list, many systems recognize three classification of users in connection with each file:

- **Owner –** Owner is the user who has created the file.
- **Group –** A group is a set of members who has similar needs and they are sharing the same file.
- **Universe –** In the system, all other users are under the category called universe.

The most common recent approach is to combine access-control lists with the normal general owner, group, and universe access control scheme. For example: Solaris uses the three categories of access by default but allows access-control lists to be added to specific files and directories when more fine-grained access control is desired.

Other Protection Approaches:

The access to any system is also controlled by the password. If the use of password are is random and it is changed often, this may be result in limit the effective access to a file. The use of passwords has a few disadvantages:

- The number of passwords are very large so it is difficult to remember the large passwords.
- If one password is used for all the files, then once it is discovered, all files are accessible; protection is on all-or-none basis.

Disk scheduling:-

- As we know, a process needs two type of time, CPU time and IO time. For I/O, it requests the Operating system to access the disk.
- However, the operating system must be fare enough to satisfy each request and at the same time, operating system must maintain the efficiency and speed of process execution.
- The technique that operating system uses to determine the request which is to be satisfied next is called disk scheduling.

Let's discuss some important terms related to disk scheduling.

- **Seek Time** - Seek time is the time taken in locating the disk arm to a specified track where the read/write request will be satisfied.
- **Rotational Latency** - It is the time taken by the desired sector to rotate itself to the position from where it can access the R/W heads.
- **Transfer Time** - It is the time taken to transfer the data.
- **Disk Access Time** - Disk access time is given as,
  Disk Access Time = Rotational Latency + Seek Time + Transfer Time
- **Disk Response Time** - It is the average of time spent by each request waiting for the IO operation.

- **Purpose of Disk Scheduling** - The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.

Goal of Disk Scheduling Algorithm

- Fairness
- High throughout
- Minimal traveling head time

Disk Scheduling Algorithms:-

- The list of various disks scheduling algorithm is given below. Each algorithm is carrying some advantages and disadvantages. The limitation of each algorithm leads to the evolution of a new algorithm.
- FCFS scheduling algorithm
- SSTF (shortest seek time first) algorithm
- SCAN scheduling
- C-SCAN scheduling
- LOOK Scheduling
- C-LOOK scheduling

FCFS Scheduling Algorithm:-

- It is the simplest Disk Scheduling algorithm. It services the IO requests in the order in which they arrive. There is no starvation in this algorithm, every request is serviced.

Disadvantages

- The scheme does not optimize the seek time.
- The request may come from different processes therefore there is the possibility of inappropriate movement of the head.

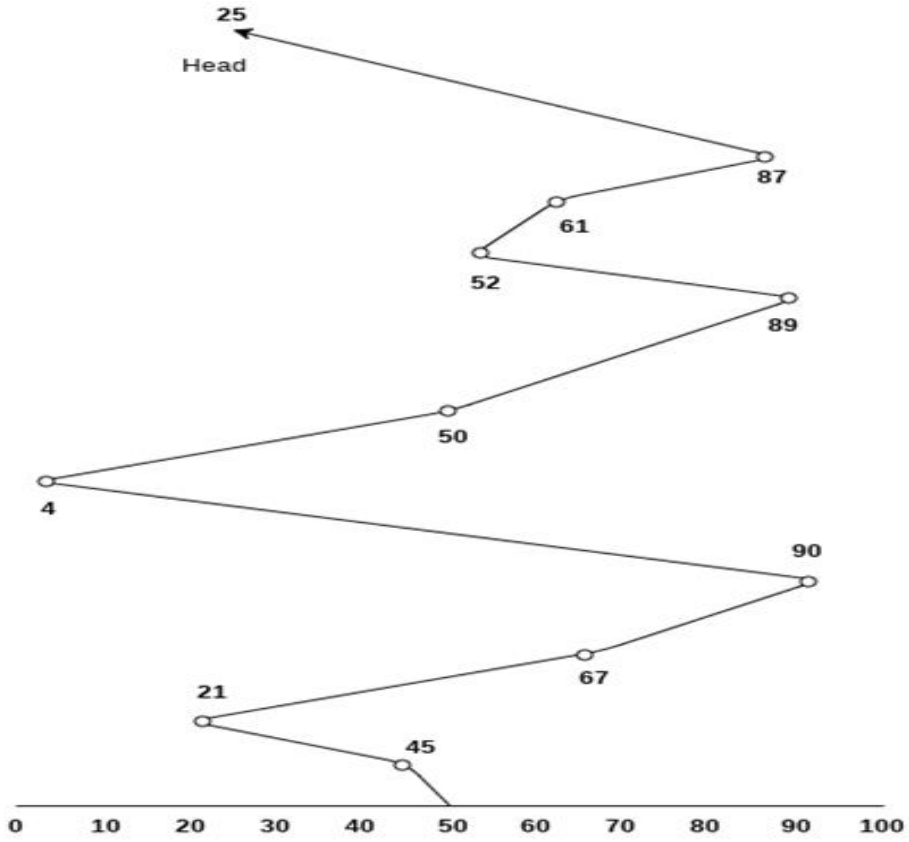
Example:- Consider the following disk request sequence for a disk with 100 tracks 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25 Head pointer starting at 50 and moving in left direction. Find the number of head movements in cylinders using FCFS scheduling.

Solution:- Number of cylinders moved by the head

= (50-45)+(45-21)+(67-21)+(90-67)+(90-4)+(50-4)+(89-50)+(61-52)+(87-61)+(87-25)

= 5 + 24 + 46 + 23 + 86 + 46 + 49 + 9 + 26 + 62

= 376

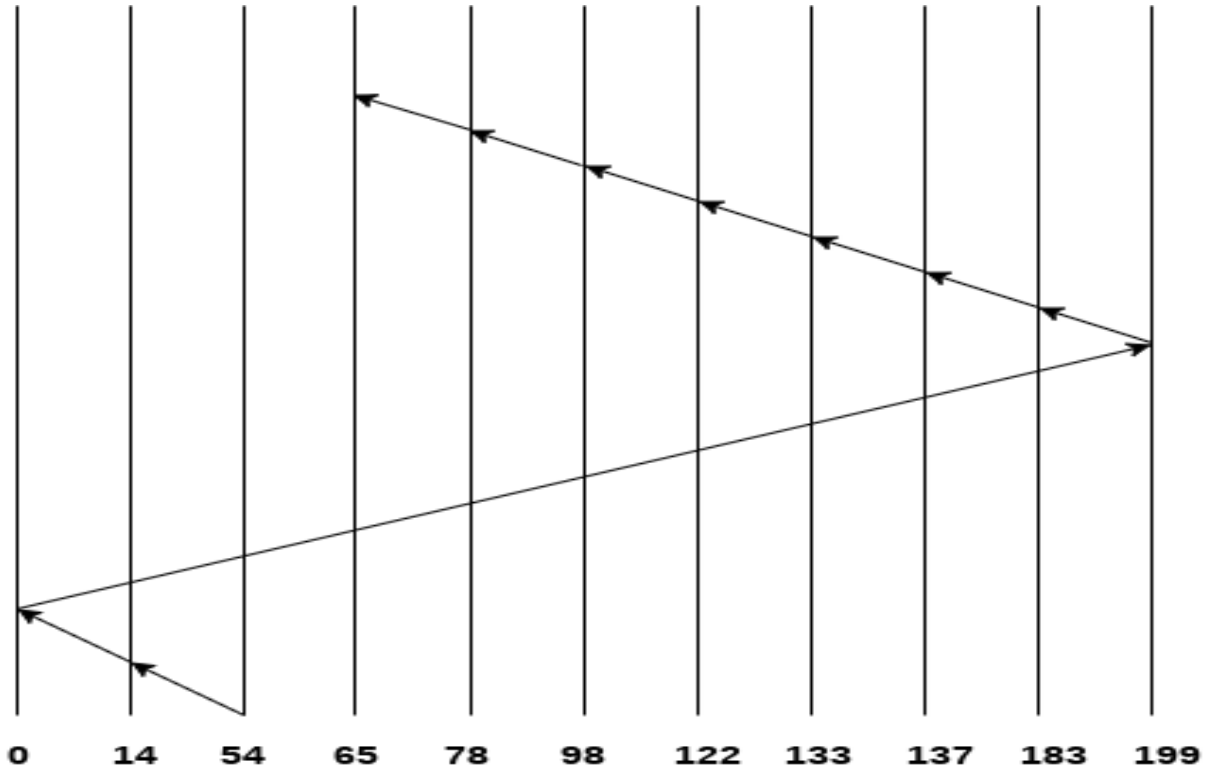


C-SCAN Algorithm:-

In C-SCAN algorithm, the arm of the disk moves in a particular direction servicing requests until it reaches the last cylinder, then it jumps to the last cylinder of the opposite direction without servicing any request then it turns back and start moving in that direction servicing the remaining requests.

Example: - Consider the following disk request sequence for a disk with 100 tracks98, 137, 122, 183, 14, 133, 65, 78 Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using C-SCAN scheduling.

Solution:- No. of cylinders crossed = 40 + 14 + 199 + 16 + 46 + 4 + 11 + 24 + 20 + 13 = 387



Page Replacement Algorithms in Operating Systems

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.

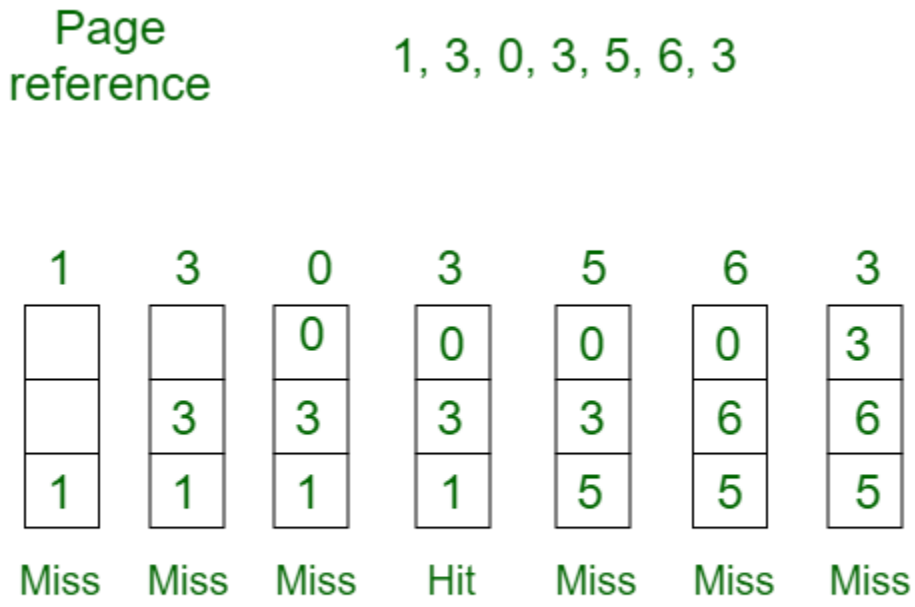
Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target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

Page Replacement Algorithms :

- **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 page in the front of the queue is selected for removal.

Example-1 Consider page reference string 1, 3, 0, 3, 5, 6 with 3 page frames. Find number of page faults.



Total Page Fault = 6

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.**

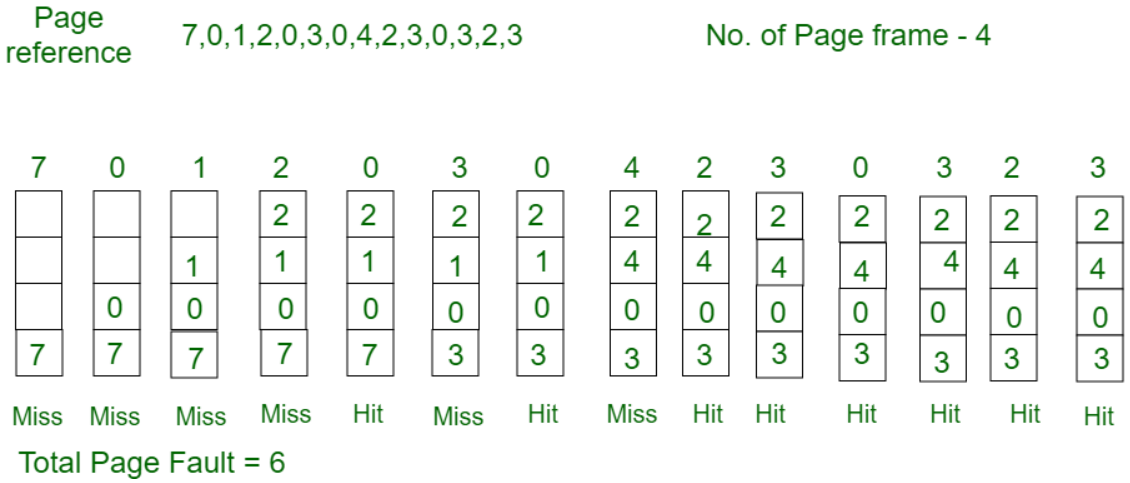
Finally when 3 come it is not avilable so it replaces 0 **1 page fault**

Belady's anomaly – Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

Optimal Page replacement –

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Example-2: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.



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.**

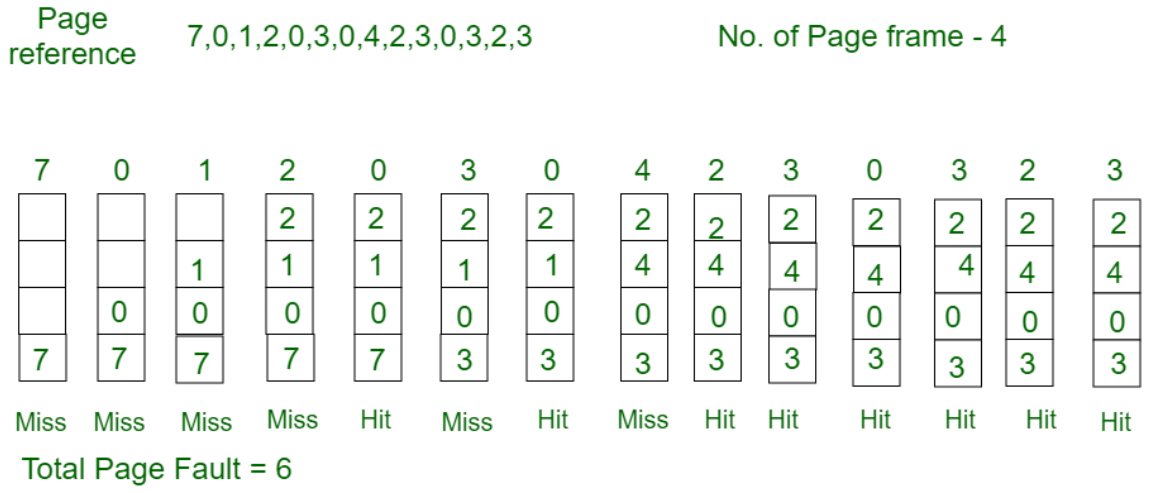
Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

- **Least Recently Used –**

  In this algorithm page will be replaced which is least recently used.

  Example-3 Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames.Find number of page faults.



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.

Allocation of frames in Operating System

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 you have multiple processes; it helps decide how many frames to allocate to each process.

There are various constraints to the strategies for the allocation of frames:

- You cannot allocate more than the total number of available frames.
- At least a minimum number of frames should be allocated to each process. This constraint is supported by two reasons. The first reason is, as less number of frames are allocated, there is an increase in the page fault ratio, decreasing the performance of the execution of the process. Secondly, there should be enough frames to hold all the different pages that any single instruction can reference.

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.

- **Advantage:** The pages in memory for a particular process and the page fault ratio is affected by the paging behavior of only that process.
- **Disadvantage:** A low priority process may hinder a high priority process by not making its frames available to the high priority process.

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.
   - **Advantage:** Does not hinder the performance of processes and hence results in greater system throughput.
   - **Disadvantage:** The page fault ratio of a process can not be solely controlled by the process itself. The pages in memory for a process depends on the paging behavior of other processes as well.

UNIT-5

Cyber Security Goals

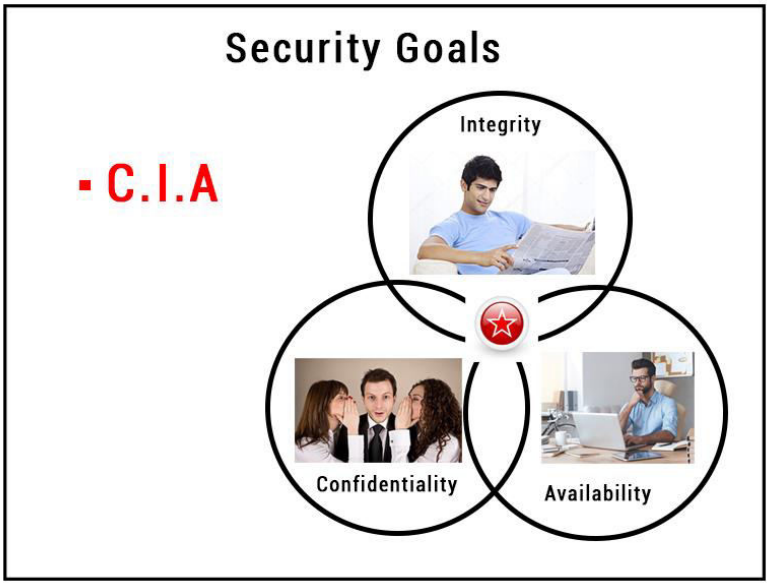
The objective of Cybersecurity is to protect information from being stolen, compromised or attacked. Cybersecurity can be measured by at least one of three goals-

1. Protect the confidentiality of data.
2. Preserve the integrity of data.
3. Promote the availability of data for authorized users.

These goals form the confidentiality, integrity, availability (CIA) triad, the basis of all security programs. The CIA triad is a security model that is designed to guide policies for information security within the premises of an organization or company. This model is also referred to as the **AIC (Availability, Integrity, and Confidentiality)** triad to avoid the confusion with the Central Intelligence Agency. The elements of the triad are considered the three most crucial components of security.

The CIA criteria are one that most of the organizations and companies use when they have installed a new application, creates a database or when guaranteeing access to some data. For data to be completely secure, all of these security goals must come into effect. These are security policies that all work together, and therefore it can be wrong to overlook one policy.

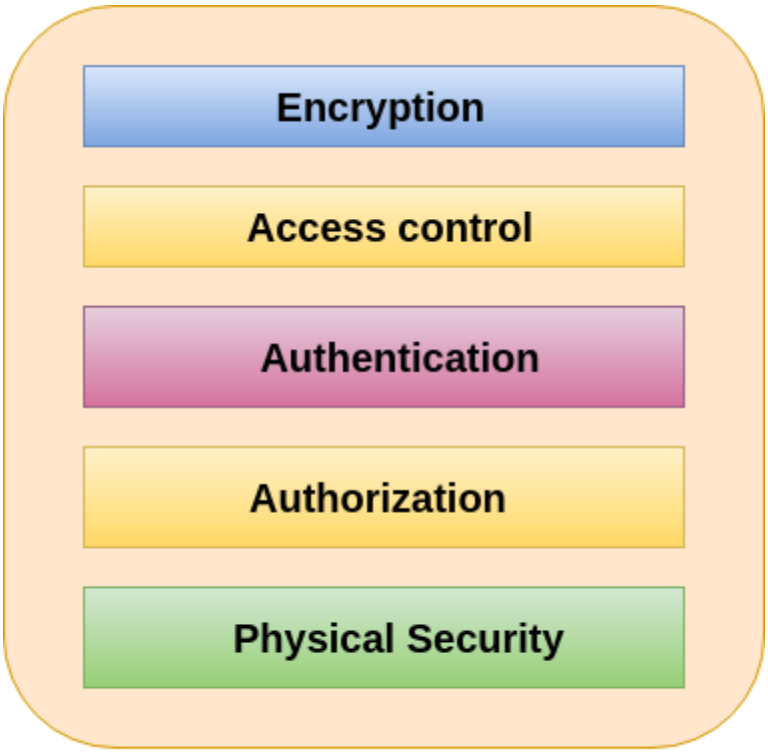
The CIA triad are-



1. Confidentiality

Confidentiality is roughly equivalent to privacy and avoids the unauthorized disclosure of information. It involves the protection of data, providing access for those who are allowed to see it while disallowing others from learning anything about its content. It prevents essential information from reaching the wrong people while making sure that the right people can get it. Data encryption is a good example to ensure confidentiality.

Tools for Confidentiality



Confidentiality Tools

Encryption

Encryption is a method of transforming information to make it unreadable for unauthorized users by using an algorithm. The transformation of data uses a secret key (an encryption key) so that the transformed data can only be read by using another secret key (decryption key). It protects sensitive data such as credit card numbers by encoding and transforming data into unreadable cipher text. This encrypted data can only be read by decrypting it. Asymmetric-key and symmetric-key are the two primary types of encryption.

Access control

Access control defines rules and policies for limiting access to a system or to physical or virtual resources. It is a process by which users are granted access and certain privileges to systems, resources or information. In access control systems, users need to present credentials before they can be granted access such as a person's name or a computer's serial number. In physical systems, these credentials may come in many forms, but credentials that can't be transferred provide the most security.

Authentication

An authentication is a process that ensures and confirms a user's identity or role that someone has. It can be done in a number of different ways, but it is usually based on a combination of-

- something the person has (like a smart card or a radio key for storing secret keys),
- something the person knows (like a password),
- something the person is (like a human with a fingerprint).

Authentication is the necessity of every organizations because it enables organizations to keep their networks secure by permitting only authenticated users to access its protected resources. These resources may include computer systems, networks, databases, websites and other network-based applications or services.

Authorization

Authorization is a security mechanism which gives permission to do or have something. It is used to determine a person or system is allowed access to resources, based on an access control policy, including computer programs, files, services, data and application features. It is normally preceded by authentication for user identity verification. System administrators are typically assigned permission levels covering all system and user resources. During authorization, a system verifies an authenticated user's access rules and either grants or refuses resource access.

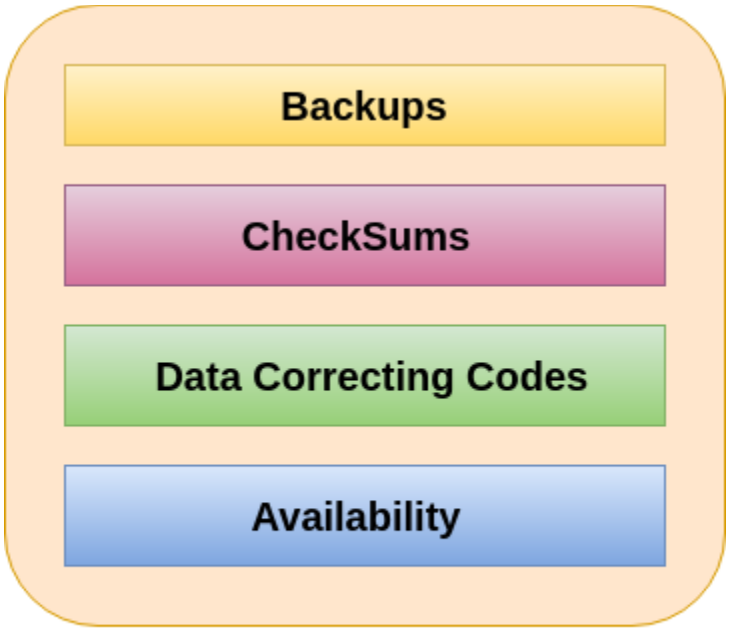
Physical Security

Physical security describes measures designed to deny the unauthorized access of IT assets like facilities, equipment, personnel, resources and other properties from damage. It protects these assets from physical threats including theft, vandalism, fire and natural disasters.

2. Integrity

Integrity refers to the methods for ensuring that data is real, accurate and safeguarded from unauthorized user modification. It is the property that information has not be altered in an unauthorized way, and that source of the information is genuine.

Tools for Integrity



Integrity Tools

Backups

Backup is the periodic archiving of data. It is a process of making copies of data or data files to use in the event when the original data or data files are lost or destroyed. It is also used to make copies for historical purposes, such as for longitudinal studies, statistics or for historical records or to meet the requirements of a data retention policy. Many applications especially in a Windows environment, produce backup files using the .BAK file extension.

Checksums

A checksum is a numerical value used to verify the integrity of a file or a data transfer. In other words, it is the computation of a function that maps the contents of a file to a numerical value. They are typically used to compare two sets of data to make sure that they are the same. A checksum function depends on the entire contents of a file. It is designed in a way that even a small change to the input file (such as flipping a single bit) likely to results in different output value.

Data Correcting Codes

It is a method for storing data in such a way that small changes can be easily detected and automatically corrected.

3. **Availability**

Availability is the property in which information is accessible and modifiable in a timely fashion by those authorized to do so. It is the guarantee of reliable and constant access to our sensitive data by authorized people.

Tools for Availability

- Physical Protections
- Computational Redundancies

Physical Protections

Physical safeguard means to keep information available even in the event of physical challenges. It ensure sensitive information and critical information technology are housed in secure areas.

Computational redundancies

It is applied as fault tolerant against accidental faults. It protects computers and storage devices that serve as fallbacks in the case of failures.

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. We're going to discuss following topics in this chapter.

- Authentication
- One Time passwords
- Program Threats
- System Threats
- Computer Security Classifications

Authentication

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 −

- **Username / Password** − User need to enter a registered username and password with Operating system to login into the system.
- **User card/key** − User need to punch card in card slot, or enter key generated by key generator in option provided by operating system to login into the system.
- **User attribute - fingerprint/ eye retina pattern/ signature** − User need to pass his/her attribute via designated input device used by operating system to login into the system.

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. System asks 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.

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 who can 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 perform illegal action without knowledge of user then it is called to have a trap door.
- **Logic Bomb** − Logic bomb is a situation when a program misbehaves only when certain conditions met otherwise it works as a genuine program. It is harder to detect.
- **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

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. Worms processes can even shut down an entire network.
- **Port Scanning** − Port scanning is a mechanism or means by which a hacker can detects system vulnerabilities to make an attack on the system.
- **Denial of Service** − Denial of service attacks normally prevents user to make legitimate use of the system. For example, a user may not be able to use internet if denial of service attacks browser's content settings.

Computer Security Classifications:-

As per the U.S. Department of Defense Trusted Computer System's Evaluation Criteria there are four security classifications in computer systems: A, B, C, and D. This is widely used specifications to determine and model the security of systems and of security solutions. Following is the brief description of each classification.

S.N.	Classification Type & Description
1	**Type A** Highest Level. Uses formal design specifications and verification techniques. Grants a high degree of assurance of process security.
2	**Type B** Provides mandatory protection system. Have all the properties of a class C2 system. Attaches a sensitivity label to each object. It is of three types. • **B1** − Maintains the security label of each object in the system. Label is used for making decisions to access control. • **B2** − Extends the sensitivity labels to each system resource, such as storage objects, supports covert channels and auditing of events.

	• **B3** − Allows creating lists or user groups for access-control to grant access or revoke access to a given named object.
3	**Type C** Provides protection and user accountability using audit capabilities. It is of two types. • **C1** − Incorporates controls so that users can protect their private information and keep other users from accidentally reading / deleting their data. UNIX versions are mostly Cl class. • **C2** − Adds an individual-level access control to the capabilities of a Cl level system.
4	**Type D** Lowest level. Minimum protection. MS-DOS, Window 3.1 fall in this category.

Threat Monitoring

■ Check for suspicious patterns of activity – i.e., several incorrect password attempts may signal password guessing.

■ Audit log – records the time, user, and type of all accesses to an object; useful for recovery from a violation and developing better security measures.

■ Scan the system periodically for security holes; done when the computer is relatively unused.

Check for:

✦ Short or easy-to-guess passwords

✦ Unauthorized set-uid programs

✦ Unauthorized programs in system directories

✦ Unexpected long-running processes

✦ Improper directory protections

✦ Improper protections on system data files

✦ Dangerous entries in the program search path (Trojan horse)

✦ Changes to system programs: monitor checksum values

Encryption

- The basic idea of encryption is to encode a message so that only the desired recipient can decode and read it. Encryption has been around since before the days of Caesar, and is an entire field of study in itself. Only some of the more significant computer encryption schemes will be covered here.
- The basic process of encryption is shown in Figure 15.7, and will form the basis of most of our discussion on encryption. The steps in the procedure and some of the key terminology are as follows:
    1. The **sender** first creates a **message, m** in plaintext.
    2. The message is then entered into an **encryption algorithm, E,** along with the **encryption key, Ke.**
    3. The encryption algorithm generates the **ciphertext, c, = E(Ke)(m).** For any key k, E(k) is an algorithm for generating ciphertext from a message, and both E and E(k) should be efficiently computable functions.
    4. The ciphertext can then be sent over an unsecure network, where it may be received by **attackers.**
    5. The **recipient** enters the ciphertext into a **decryption algorithm, D,** along with the **decryption key, Kd.**
    6. The decryption algorithm re-generates the plaintext message, m, = D(Kd)(c). For any key k, D(k) is an algorithm for generating a clear text message from a ciphertext, and both D and D(k) should be efficiently computable functions.
    7. The algorithms described here must have this important property: Given a ciphertext c, a computer can only compute a message m such that c = E(k)(m) if it possesses D(k). (In other words, the messages can't be decoded unless you have the decryption algorithm and the decryption key.)

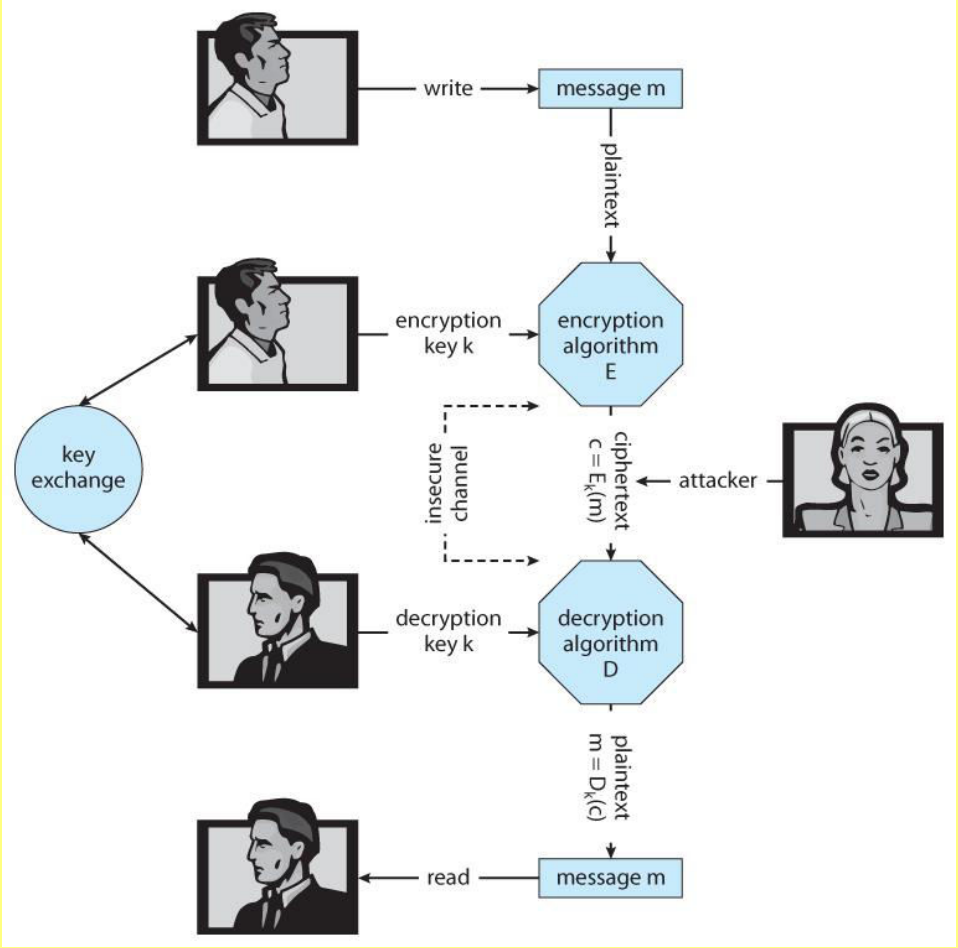


Figure 15.7 - A secure communication over an insecure medium.

15.4.1.1 Symmetric Encryption

- With ***symmetric encryption*** the same key is used for both encryption and decryption, and must be safely guarded. There are a number of well-known symmetric encryption algorithms that have been used for computer security:
    - The ***Data-Encryption Standard, DES,*** developed by the National Institute of Standards, NIST, has been a standard civilian encryption standard for over 20 years. Messages are broken down into 64-bit chunks, each of which are encrypted using a 56-bit key through a series of substitutions and transformations. Some of the transformations are hidden (black boxes), and are classified by the U.S. government.
    - DES is known as a ***block cipher,*** because it works on blocks of data at a time. Unfortunately this is a vulnerability if the same key is used for an extended amount of data. Therefore an enhancement is to not only encrypt each block, but also to XOR it with the previous block, in a technique known as ***cipher-block chaining.***
    - As modern computers become faster and faster, the security of DES has decreased, to where it is now considered insecure because its keys can be exhaustively searched within a reasonable amount of computer time. An

enhancement called ***triple DES*** encrypts the data three times using three separate keys (actually two encryptions and one decryption) for an effective key length of 168 bits. Triple DES is in widespread use today.

- The ***Advanced Encryption Standard, AES,*** developed by NIST in 2001 to replace DES uses key lengths of 128, 192, or 256 bits, and encrypts in blocks of 128 bits using 10 to 14 rounds of transformations on a matrix formed from the block.
- The ***twofish algorithm,*** uses variable key lengths up to 256 bits and works on 128 bit blocks.
- ***RC5*** can vary in key length, block size, and the number of transformations, and runs on a wide variety of CPUs using only basic computations.
- ***RC4*** is a ***stream cipher,*** meaning it acts on a stream of data rather than blocks. The key is used to seed a pseudo-random number generator, which generates a ***keystream*** of keys. RC4 is used in ***WEP,*** but has been found to be breakable in a reasonable amount of computer time.

15.4.1.2 Asymmetric Encryption

- With ***asymmetric encryption,*** the decryption key, Kd, is not the same as the encryption key, Ke, and more importantly cannot be derived from it, which means the encryption key can be made publicly available, and only the decryption key needs to be kept secret. (or vice-versa, depending on the application.)
- One of the most widely used asymmetric encryption algorithms is ***RSA,*** named after its developers - Rivest, Shamir, and Adleman.
- RSA is based on two large prime numbers, ***p*** and ***q,*** (on the order of 512 bits each), and their product ***N.***
    - Ke and Kd must satisfy the relationship:
      (Ke * Kd) % [(p - 1) * (q - 1)] = = 1
    - The encryption algorithm is:
      c = E(Ke)(m) = m^Ke % N
    - The decryption algorithm is:
      m = D(Kd)(c) = c^Kd % N
- An example using small numbers:
    - p = 7
    - q = 13
    - N = 7 * 13 = 91
    - (p - 1) * (q - 1) = 6 * 12 = 72
    - Select Ke < 72 and relatively prime to 72, say 5
    - Now select Kd, such that (Ke * Kd) % 72 = = 1, say 29
    - The public key is now (5, 91) and the private key is (29, 91)
    - Let the message, m = 42
    - Encrypt: c = 42^5 % 91 = 35
    - Decrypt: m = 35^29 % 91 = 42

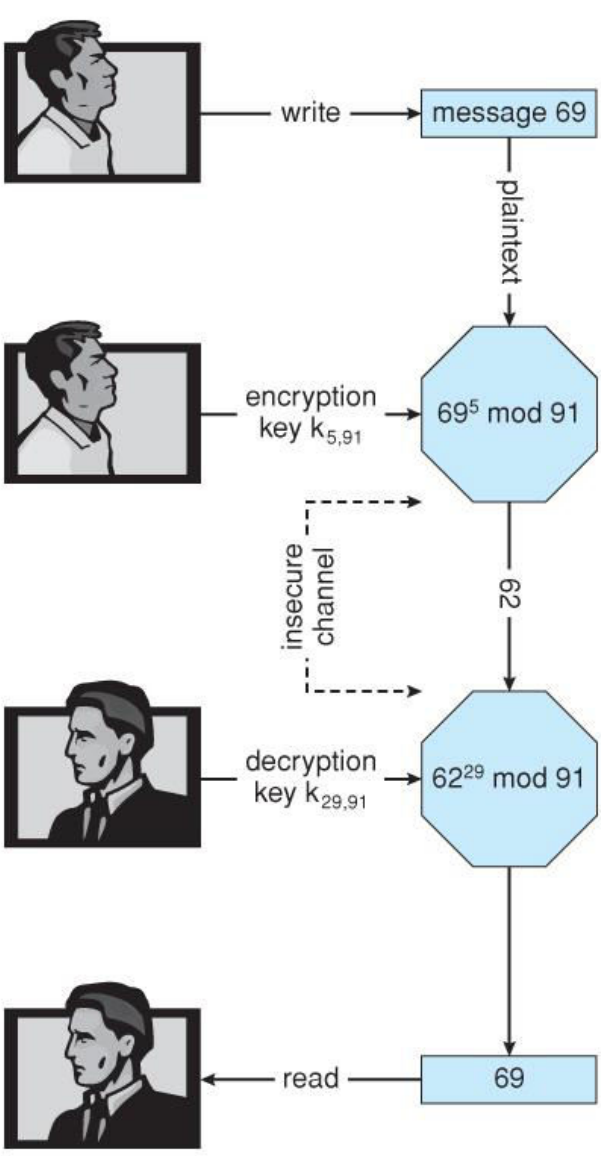


Figure 15.8 - Encryption and decryption using RSA asymmetric cryptography

- Note that asymmetric encryption is much more computationally expensive than symmetric encryption, and as such it is not normally used for large transmissions. Asymmetric encryption is suitable for small messages, authentication, and key distribution, as covered in the following sections.

Chapter 13: Protection

Chapter 13: Protection

Objectives

- Discuss the goals and principles of protection in a modern computer system
- Explain how protection domains combined with an access matrix are used to specify the resources a process may access
- Examine capability and language-based protection systems

13.1 Goals of Protection

- Requirements of reliable systems
  - The need to prevent the mischievous, intentional violation of an access restriction by a user
  - The need to ensure that each program component active in a system uses system resources only in ways consistent with stated ***policies***
- Goal: The role of protection in a computer system is to provide a ***mechanism*** for the enforcement of the policies governing resource use
  - In one protection model, computer consists of a collection of objects, hardware or software
  - Each object has a unique name and can be accessed through a well-defined set of operations
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so

13.2 Principles of Protection

- Guiding principle – **principle of least privilege**
  - Programs, users and systems ***should be given*** just enough **privileges** to perform their tasks
    - Limits damage if entity has a bug, gets abused
  - Managing users with the principle of least privilege entails creating a separate account for each user, with just the privileges that the user needs
    - Usually implements role-based access control (RBAC)

- "**Need to know**": a similar concept regarding access to data
  - A process should be able to access only those resources that it ***currently requires*** to complete its task
  - Information in regards to some activity is not to be communicated to everyone

13.3 Domain of Protection

- **Protection domain**:
  - Specifies the resources that the process may access
  - Defines
    - a set of objects and
      - Hardware objects (CPU, memory segments, printers, disks, etc) and
      - Software objects (files, programs, semaphores, etc)
    - the types of operations that may be invoked on each object
      - The ability to execute an operation on an object is an ***access right***

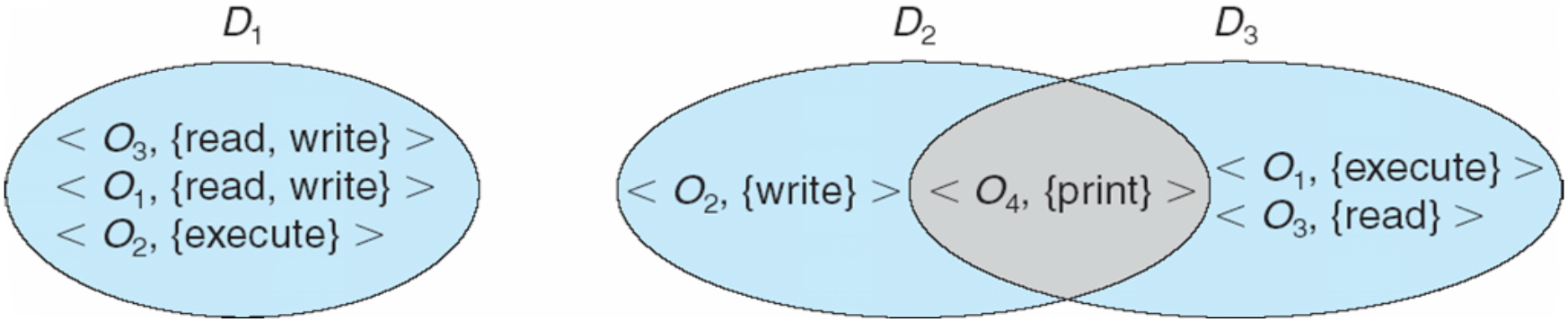
- Domain can be user, process, procedure

- A process should be allowed to access only those resources for which it has authorization

Domain Structure

A domain is a collection of access rights, each of which is an ordered pair <object-name, rights-set>

- Domain = set of access-rights
- Access-right = <*object-name*, *rights-set*>
  where *rights-set* is a subset of all valid operations that can be performed on the object



Domain Considerations

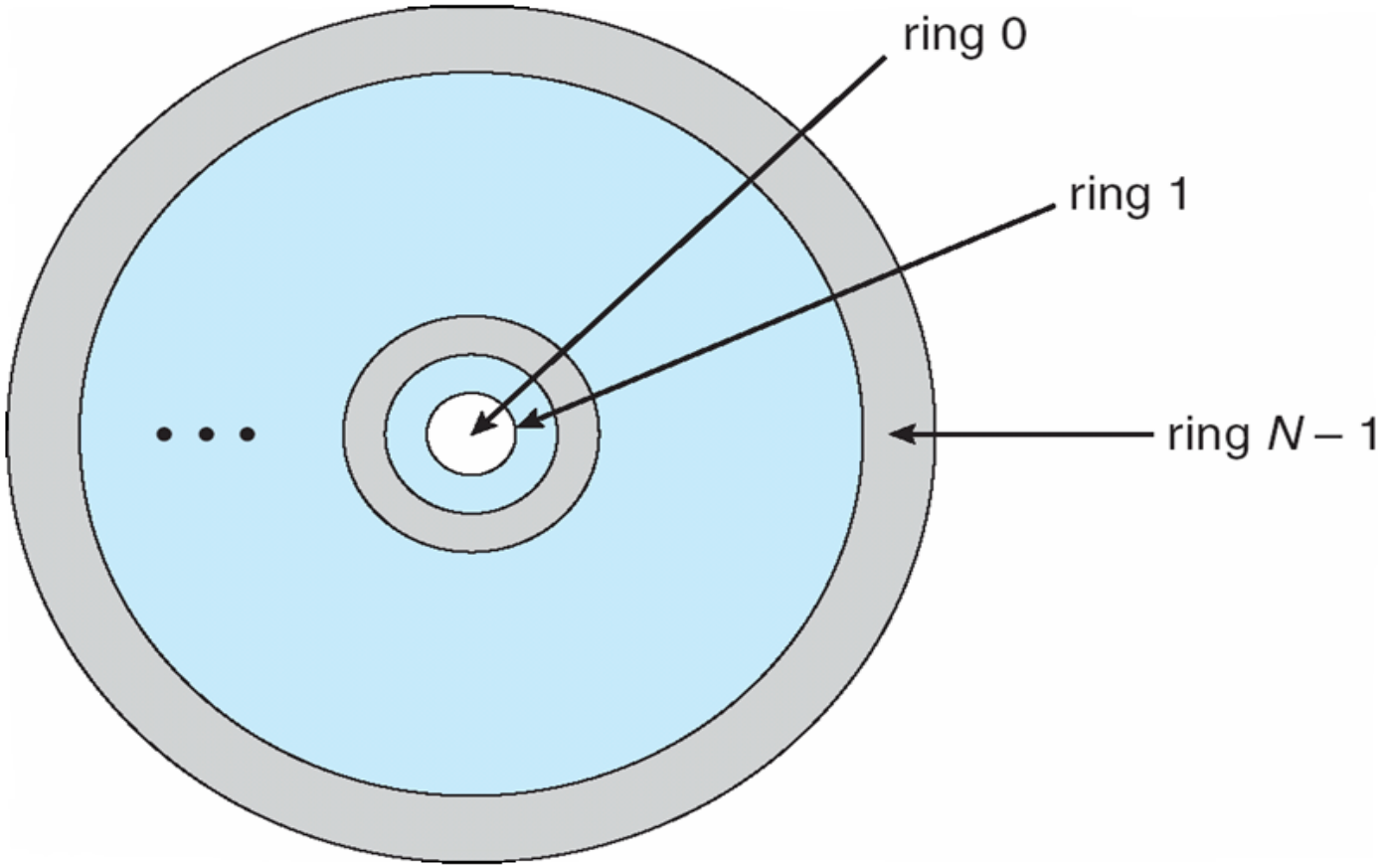
- Associations: static or dynamic
  - Can be static (during life of system, during life of process)
  - Or dynamic (changed as needed)
    - **domain switching**: enabling the process to switch from one domain to another
    - **privilege escalation**: allowing the content of a domain to be changed
- "grain" aspect: rough or fine
  - Rough-grained privilege management easier, simpler, but least privilege now done in large chunks
    - For example, traditional Unix processes either have abilities of the associated user, or of root
  - Fine-grained management more complex, more overhead, but more protective
    - File ACL lists, RBAC

Domain Implementation (UNIX)

- Domain = user-id
- Domain switch accomplished via file system
  - Each file has associated with it a domain bit (setuid bit)
  - When file is executed and setuid = on, then user-id is set to owner of the file being executed
  - When execution completes user-id is reset
- Domain switch accomplished via passwords
  - su command temporarily switches to another user's domain when other domain's password provided
- Domain switching via commands
  - sudo command prefix executes specified command in another domain (if original domain has privilege or password given)

Domain Implementation (MULTICS)

- The protection domains are organized hierarchically into a ring structure (0 to 7). A current-ring-number counter is associated with each process
- Let D_i and D_j be any two domain rings
  If $j < i \Rightarrow D_i \subseteq D_j$: a process executing in domain D_j has more privileges than does a process executing in domain D_i



Multics Benefits and Limits

- Ring / hierarchical structure provided more than the basic kernel / user or root / normal user design

- Fairly complex -> more overhead

- But does not allow strict need-to-know
  - Object accessible in D_j but not in D_i, then j must be $< i$
  - But then every segment accessible in D_i also accessible in D_j

13.4 Access Matrix

- View protection as a matrix (*access matrix*)
  - Rows represent domains
  - Columns represent objects
- *Access(i, j)* is the set of operations that a process executing in $Domain_i$ can invoke on Object

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

four domains and four objects—three files (*F*1, *F*2, *F*3) and one printer

Use of Access Matrix

- Mechanism: If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix
- User who creates object can define access column for that object
- Can be expanded to dynamic protection
  - Operations to add, delete access rights
  - Special access rights:
    - *owner of* O_i
    - *copy op from* O_i *to* O_j *(denoted by "*")*
    - *control –* D_i *can modify* D_j *access rights (applicable to domain only)*
    - *transfer – switch from domain* D_i *to* D_j
  - *Copy* and *Owner* applicable to an object
  - *Control* applicable to domain object

Use of Access Matrix (Cont.)

- **Access matrix** design separates mechanism from policy
  - Mechanism
    - Operating system provides access-matrix + rules
    - If ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced
  - Policy
    - User dictates policy
    - Who can access what object and in what mode

- But doesn't solve the ***general confinement problem***
  - The problem of guaranteeing that no information initially held in an object can migrate outside of its execution environment

Access Matrix of Figure A with Domains as Objects

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

- Switching from domain *Di* to domain *Dj* is allowed if and only if the access right switch ∈ access(*i*, *j*)
  - i.e. domain D_2 can switch to domain D_3 and D_4

Access Matrix with *Copy* Rights

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b) (limited) copy the read operation with file F_2

- Assume domain D_1 is the owner of F_1 and domain D_2 is the owner of F_2 and F_3
- The ability to copy an access right from one domain (or row) of the access matrix to another is denoted by an asterisk (*) appended to the access right

Variants of copy right:

1. (propagation) copy: access right and '*'
2. transfer: A right is copied from access(i, j) to access(k, j); it is then removed from access(i, j)
3. limited copy: only the right R (not R*) is created

Access Matrix With *Owner* Rights

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

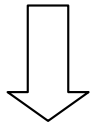
(b)

- ***Owner right*** controls addition of new rights and removal of some rights
  - Owner v.s. copy: *Owner* need not have some rights before addition, but *domain with * copy right* must have some rights before copy
- Assume domain D1 is the owner of F1 and domain D2 is the owner of F2 and F3

- D1 removes F1 execute from D3
- D2 adds F2 write* to itself
- D2 adds F2 write and F3 write to D3

Modified Access Matrix of Figure 13.4

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			



object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

- If access(i, j) includes the ***control* right**, then a process executing in domain *Di* can *remove* any access right from row *j*
- Suppose include the control right in access(D2, D4) to above Figure
- A process executing in domain *D2* could modify domain *D4* (F1 read and F3 read) => bottom Figure

13.5 Implementation of Access Matrix

- Generally, a sparse matrix
- Option 1 – Global table
  - Store ordered triples < *domain, object, rights-set* > in table
  - A requested operation M on object O_j within domain D_i -> search table for < D_i, O_j, R_k >
    - with M $\in$ R_k
  - But table could be large -> won't fit in main memory
  - Difficult to group objects (consider an object that all domains can read)

- Option 2 – Access lists for objects
  - Each column implemented as an access list for one object
  - Resulting per-object list consists of ordered pairs < *domain, rights-set* > defining all domains with non-empty set of access rights for the object
  - Easily extended to contain default set -> If M $\in$ default set, also allow access

Implementation of Access Matrix: Separating into Lists

- Each column = **Access-control List** for one object
  Defines who can perform what operation

  Domain 1 = Read, Write
  Domain 2 = Read
  Domain 3 = Read

- Each Row = **Capability List** (like a key)
  For each domain, what operations allowed on what objects

  Object F1 – Read

  Object F4 – Read, Write, Execute

  Object F5 – Read, Write, Delete, Copy

Implementation of Access Matrix (Cont.)

- Option 3 – Capability list for domains
  - Instead of object-based, list is domain based
  - **Capability list** for domain is list of objects together with operations allows on them
  - Object represented by its name or address, called a **capability**
  - Execute operation M on object O_j, process requests operation and specifies capability as parameter
    - Possession of capability means access is allowed
  - Capability list associated with domain but never directly accessible by domain
    - Rather, protected object, maintained by OS and accessed indirectly
    - Like a “secure pointer”
    - Idea can be extended up to applications

- Option 4 – Lock-key
  - Compromise between access lists and capability lists
  - Each object has list of unique bit patterns, called **locks**
  - Each domain as list of unique bit patterns called **keys**
  - Process in a domain can only access object if domain has key that matches one of the locks

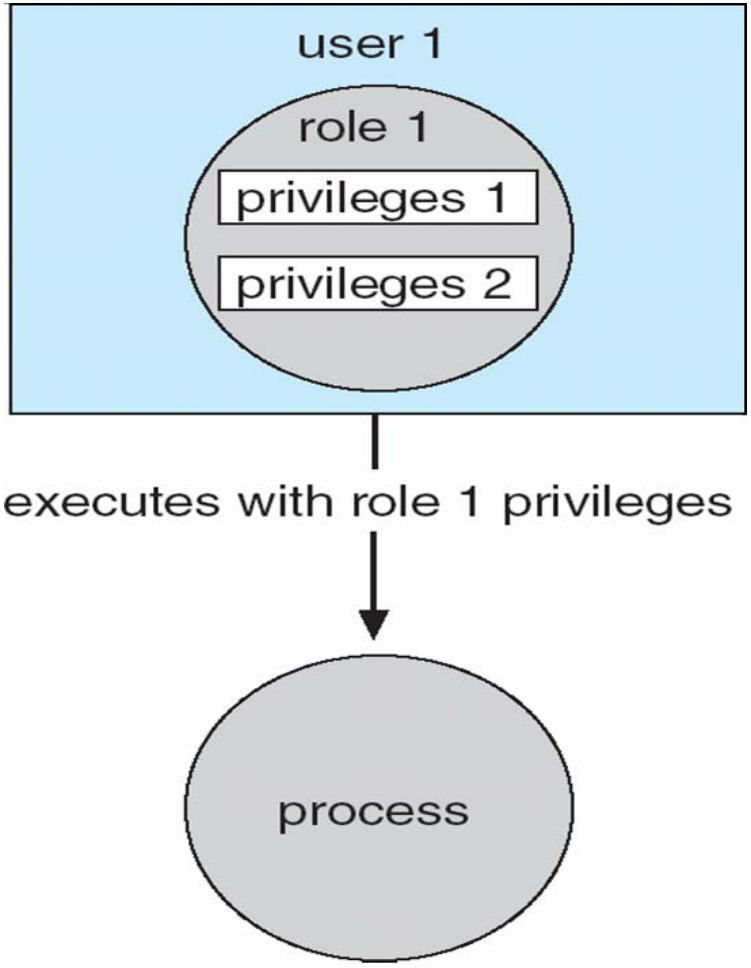
Comparison of Implementations

- Many trade-offs to consider
  - Global table is simple, but can be large
  - Access lists correspond to needs of users
    - Determining set of access rights for domain non-localized so difficult
    - Every access to an object must be checked
      - Many objects and access rights -> slow
  - Capability lists useful for localizing information for a given process
    - But revocation capabilities can be inefficient
  - Lock-key effective and flexible, keys can be passed freely from domain to domain, easy revocation

- Most systems use combination of access lists and capabilities
  - First access to an object -> access list searched
    - If allowed, capability created and attached to process
      - Additional accesses need not be checked
    - After last access, capability destroyed
    - Consider file system with ACLs per file

13.6 Access Control

- Protection can be applied to non-file resources
- Solaris 10 provides **role-based access control** (**RBAC**) to implement least privilege
  - *Privilege* is right to execute system call or use an option within a system call
  - Can be assigned to processes
  - Users assigned *roles* granting access to privileges and programs
    - Enable role via password to gain its privileges
  - Similar to access matrix



13.7 Revocation of Access Rights

- Various options to remove the access right of a domain to an object
  - Immediate vs. delayed
  - Selective vs. general
  - Partial vs. total
  - Temporary vs. permanent
- **Access List** – Delete access rights from access list
  - Simple – search access list and remove entry
  - Immediate, general or selective, total or partial, permanent or temporary
- **Capability List** – Scheme required to locate capability in the system before capability can be revoked
  - Reacquisition – periodic delete, with require and denial if revoked
  - Back-pointers – set of pointers from each object to all capabilities of that object (Multics)
  - Indirection – capability points to global table entry which points to object – delete entry from global table, not selective (CAL)
  - Keys – unique bits associated with capability, generated when capability created
    - Master key associated with object, key matches master key for access
    - Revocation – create new master key
    - Policy decision of who can create and modify keys – object owner or others?

13.8 Capability-Based Systems

- Hydra
  - Fixed set of access rights known to and interpreted by the system
    - i.e. read, write, or execute each memory segment
    - User can declare other **auxiliary rights** and register those with protection system
    - Accessing process must hold capability and know name of operation
    - **Rights amplification** allowed by trustworthy procedures for a specific type
  - Interpretation of user-defined rights performed solely by user's program; system provides access protection for use of these rights
  - Operations on objects defined procedurally – procedures are objects accessed indirectly by capabilities
  - Solves the *problem of mutually suspicious subsystems*
  - Includes library of prewritten security routines
- Cambridge CAP System
  - Simpler but powerful
  - **Data capability** - provides standard read, write, execute of individual storage segments associated with object – implemented in microcode
  - **Software capability** -interpretation left to the subsystem, through its protected procedures
    - Only has access to its own subsystem
    - Programmers must learn principles and techniques of protection

13.9 Language-Based Protection

- Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources
- Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable
- Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system

Protection in Java 2

- Protection is handled by the Java Virtual Machine (JVM)
- Load-time and run-time check
  - A class is assigned a protection domain when it is loaded by the JVM
    - The protection domain indicates what operations the class can (and cannot) perform
  - If a library method is invoked that performs a privileged operation, the stack is inspected to ensure the operation can be performed by the library

protection domain:	untrusted applet	URL loader	networking
socket permission:	none	*.lucent.com:80, connect	any

class:	gui:
	...
	vget(url);
	xopen(addr);
	...
		get(URL u):
		...
		doPrivileged {
		open('proxy.lucent.com:80');
		}
		<request u from proxy>
		...
			open(Addr a):
			...
			checkPermission (a, connect);
			connect (a);
			...

URL loader succeeds since proxy.lucent.com is permitted (below *.lucent.com)

Stack Inspection (reject if no permission)

Summary

- Goals and principles of system protection are explained
- Protection domain, access rights and access matrix are introduced
- Access control list, capability-based systems, and examples are given