OPERATING SYSTEMS

OBJECTIVES

- To provide knowledge about the services rendered by operating systems
- To provide a detailed discussion of the various memory management techniques
- To discuss the various file-system design and implementation issues
- To discuss how the protection domains help to achieve security in a system

MODULE:I (8Hours)

Operating Systems -Definition- Types- Functions -Abstract view of OS-System Structures-System Calls- Virtual Machines -Process Concepts -Threads -Multithreading

MODULE:II (4Hours)

Process Scheduling- Process Co-ordination -Synchronization -Semaphores -Monitors Hardware Synchronization-Deadlocks -Methods for Handling Deadlocks

MODULE:III (12 Hours)

Memory Management Strategies -Contiguous and Non-Contiguous allocation -Virtual memory Management -Demand Paging- Page Placement and Replacement Policies

MODULE:IV (6 Hours)

File System -Basic concepts - File System design and Implementation -Case Study: Linux File Systems Mass Storage Structure -Disk Scheduling -Disk Management -V/O Systems-System Protection and Security.

MODULE:V (10 Hours)

Distributed Systems -Distributed operating systems -Distributed file systems -Distribute

Synchronization

MODULE I

Operating Systems -Definition- Types- Functions -Abstract view of OS-System Structures-System Calls- Virtual Machines -Process Concepts -Threads -Multithreading

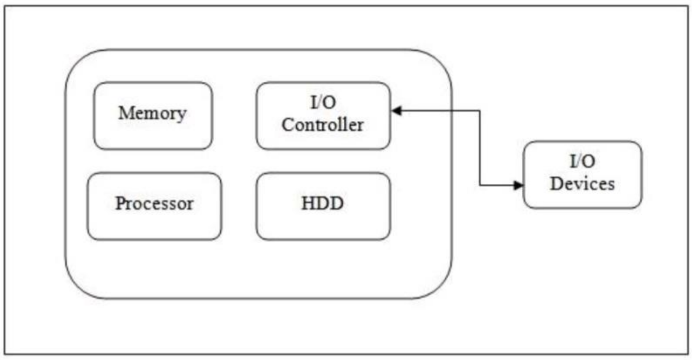
OPERATING SYSTEM:

DEFINATION:

- An operating system is a program which manages all the computer hardwares.
- It provides the base for application program and acts as an intermediary between a user and the computer hardware.
- The operating system has two objectives such as:

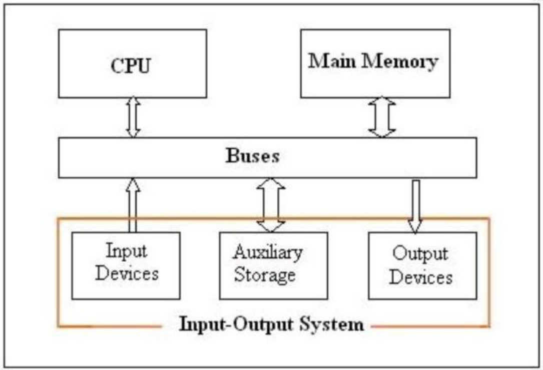
  Firstly, an operating system controls the computer's hardware.

  The second objective is to provide an interactive interface to the user and interpret commands so that it can communicate with the hardware.
- The operating system is very important part of almost every computer system. **Managing Hardware**



- The prime objective of operating system is to manage & control the various hardware resources of a computer system.
- These hardware resources include processer, memory, and disk space and so on.
- The output result was display in monitor. In addition to communicating with the hardware theoperating system provides on error handling procedure and display an error notification.
- If a device not functioning properly, the operating system cannot be communicate with the device.

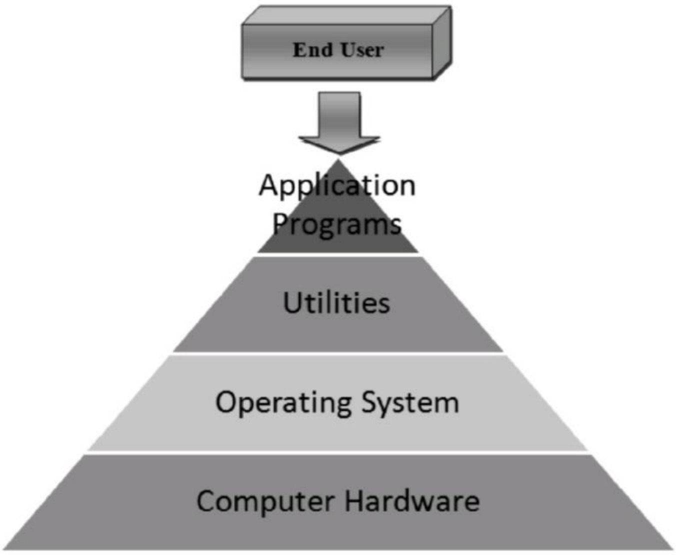
Providing an Interface :



- The operating system organizes application so that users
- can easily access, use and store them.
- It provides a stable and consistent way for applications to deal with the hardware without the user having known details of the hardware.
- If the program is not functioning properly, the operating system again takes control, stops the application and displays the appropriate error message.
- Computer system components are divided into 5 parts Computer hardware operating system utilities

Application programs

End user



- The operating system controls and coordinate a user of hardware and various application programs for various users.
- It is a program that directly interacts with the hardware.
- The operating system is the first encoded with the Computer and it remains on the memory all time thereafter.

System goals

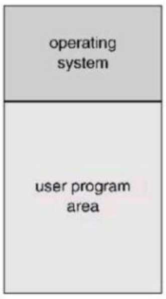
- The purpose of an operating system is to be provided an environment in which an user can execute programs.
- Its primary goals are to make the computer system convenience for the user.
- Its secondary goals are to use the computer hardware in efficient manner.

- .

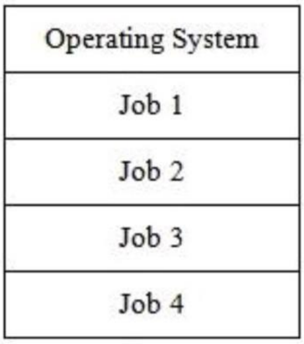
Types of Operating System

1. **Mainframe System:** It is the system where the first computer used to handle many commercial scientific applications. The growth of mainframe systems traced from simple batch system where the computer runs one and only one application to time shared systems which allowed for user interaction with the computer system
   a. **Batch /Early System:** Early computers were physically large machine. The common input devices were card readers, tape drivers. The common output devices were line printers, tape drivers and card punches. In these systems the user did not interact directly with the computer system. Instead the user preparing a job which consists of programming data and some control information and then submitted it to the computer operator after some time the output is appeared. The output in these early computer was fairly simple is main task was to transfer control automatically from one job to next. The operating system always resides in the memory. To speed up processing operators batched the jobs with similar needs and ran then together as a group. The disadvantages of batch system are that in this execution environment the CPU is often idle because the speed up of I/O devices is much slower than the CPU.

Memory Layout for a Simple Batch System



   b. **Multiprogrammed System:** Multiprogramming concept increases CPU utilization by organization jobs so that the CPU always has one job to execute the idea behind multiprogramming concept. The operating system keeps several jobs in memory simultaneously as shown in below figure.



Operating System
Job 1
Job 2
Job 3
Job 4

This set of job is subset of the jobs kept in the job pool. The operating system picks and beginning to execute one of the jobs in the memory. In this environment the operating system simply switches and executes another job. When a job needs to wait the CPU is simply switched to another job and so on. The multiprogramming operating system is sophisticated because the operating system makes decisions for the user. This is known as scheduling. If several jobs are ready to run at the same time the system choose one among them. This is known as CPU scheduling. The disadvantages of the multiprogrammed system are

- It does not provide user interaction with the computer system during the program execution.
- The introduction of disk technology solved these problems rather than reading the cards from card reader into disk. This form of processing is known as spooling.

SPOOL stands for simultaneous peripheral operations online. It uses the disk as a huge buffer for reading from input devices and for storing output data until the output devices accept them. It is also use for processing data at remote sides. The remote processing is done and its own speed with no CPU intervention. Spooling overlaps the input, output one job with computation of other jobs. Spooling has a beneficial effect on the performance of the systems by keeping both CPU and I/O devices working at much higher time.

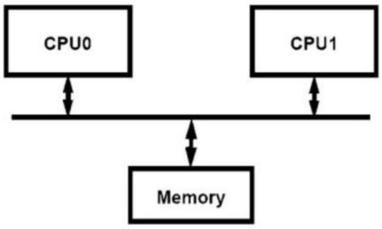
c. **Time Sharing System:**The time sharing system is also known as multi user systems. The CPU executes multiple jobs by switching among them but the switches occurs so frequently that the user can interact with each program while it is running. An interactive computer system provides direct communication between a user and system. The user gives instruction to the operating systems or to a program directly using keyboard or mouse and wait for immediate results. So the response time will be short. The time sharing system allows many users to share the computer simultaneously. Since each action in this system is short, only a little CPU time is needed for each user. The system switches rapidly from one user to the next so each user feels as if the entire computer system is dedicated to his use, even though it is being shared by many users. The disadvantages of time sharing system are:

- It is more complex than multiprogrammed operating system
- The system must have memory management & protection, since several jobs are kept in memory at the same time.
- Time sharing system must also provide a file system, so disk management is required.
- It provides mechanism for concurrent execution which requires complex CPU scheduling schemes.

2. **Personal Computer System/Desktop System:** Personal computer appeared in 1970's. They are microcomputers that are smaller & less expensive than mainframe systems. Instead of maximizing CPU & peripheral utilization, the systems opt for maximizing user convenience & responsiveness. At first file protection was not necessary on a personal machine. But when other computers 2nd other users can access the files on a pc file protection becomes necessary. The lack of protection made if easy for malicious programs to destroy data on such systems. These programs may be self replicating& they spread via worm or virus mechanisms. They can disrupt entire companies or even world wide networks. E.g : windows 98, windows 2000, Linux.
3. **Microprocessor Systems/ Parallel Systems/ Tightly coupled Systems:** These Systems have more than one processor in close communications which share the computer bus, clock, memory & peripheral devices. Ex: UNIX, LINUX. Multiprocessor Systems have 3 main advantages.
    a. **Increased throughput:** No. of processes computed per unit time. By increasing the no. of processors move work can be done in less time. The speed up ratio with N processors is not N, but it is less than N. Because a certain amount of overhead is incurred in keeping all the parts working correctly.
    b. **Increased Reliability:** If functions can be properly distributed among several processors, then the failure of one processor will not halt the system, but slow it down. This ability to continue to operate in spite of failure makes the system fault tolerant.
    c. **Economic scale:** Multiprocessor systems can save money as they can share peripherals, storage & power supplies.

The various types of multiprocessing systems are:

- **Symmetric Multiprocessing (SMP):** Each processor runs an identical copy of the operating system & these copies communicate with one another as required. Ex: Encore's version of UNIX for multi max computer. Virtually, all modern operating system NT, Solaris, Digital UNIX, OS/2 & LINUX now provide support for SMP.



- **Asymmetric Multiprocessing (Master – Slave Processors):** Each processor is designed for a specific task. A master processor controls the system & schedules & allocates the work to the slave processors. Ex- Sun's Operating system SUNOS version 4 provides asymmetric multiprocessing.

4. **Distributed System/Loosely Coupled Systems:** In contrast to tightly coupled systems, the processors do not share memory or a clock. Instead, each processor has its own local memory. The processors communicate with each other by various communication lines such as high speed buses or telephone lines. Distributed systems depend on networking for their functionalities. By being able to communicate distributed systems are able to share computational tasks and provide a rich set of features to the users. Networks vary by the protocols used, the distances between the nodes and transport media. TCP/IP is the most common network protocol. The processor is a distributed system varies in size and function. It may microprocessors, work stations, minicomputer, and large general purpose computers. Network types are based on the distance between the nodes such as LAN (within a room, floor or building) and WAN (between buildings, cities or countries). The advantages of distributed system are resource sharing, computation speed up, reliability, communication.
5. **Real time Systems:** Real time system is used when there are rigid time requirements on the operation of a processor or flow of data. Sensors bring data to the computers. The computer analyzes data and adjusts controls to modify the sensors inputs. System that controls scientific experiments, medical imaging systems and some display systems are real time systems. The disadvantages of real time system are:

a. A real time system is considered to function correctly only if it returns the correct result within the time constraints.

b. Secondary storage is limited or missing instead data is usually stored in short term memory or ROM.

c. Advanced OS features are absent.

Real time system is of two types such as:

- **Hard real time systems:** It guarantees that the critical task has been completed on time. The sudden task is takes place at a sudden instant of time.
- **Soft real time systems:** It is a less restrictive type of real time system where a critical task gets priority over other tasks and retains that priority until it computes. These have more limited utility than hard real time systems. Missing an occasional deadline is acceptable e.g. QNX, VX works. Digital audio or multimedia is included in this category.

It is a special purpose OS in which there are rigid time requirements on the operation of a processor. A real time OS has well defined fixed time constraints. Processing must be done within the time constraint or the system will fail. A real time system is said to function correctly only if it returns the correct result within the time constraint. These systems are characterized by having time as a key parameter.

Basic Functions of Operation System

The various functions of operating system are as follows:

1. Process Management:

- A program does nothing unless their instructions are executed by a CPU.A process is a program in execution. A time shared user program such as a complier is a process. A word processing program being run by an individual user on a pc is a process.
- A system task such as sending output to a printer is also a process. A process needs certain resources including CPU time, memory files & I/O devices to accomplish its task.
- These resources are either given to the process when it is created or allocated to it while it is running. The OS is responsible for the following activities of process management.
- Creating & deleting both user & system processes.
- Suspending & resuming processes.
- Providing mechanism for process synchronization.
- Providing mechanism for process communication.
- Providing mechanism for deadlock handling.

2. **Main Memory Management:**

   The main memory is central to the operation of a modern computer system. Main memory is a large array of words or bytes ranging in size from hundreds of thousand to billions. Main memory stores the quickly accessible data shared by the CPU & I/O device. The central processor reads instruction from main memory during instruction fetch cycle & it both reads &writes data from main memory during the data fetch cycle. The main memory is generally the only large storage device that the CPU is able to address & access directly. For example, for the CPU to process data from disk. Those data must first be transferred to main memory by CPU generated E/O calls. Instruction must be in memory for the CPU to execute them. The OS is responsible for the following activities in connection with memory management.

   - Keeping track of which parts of memory are currently being used & by whom.
   - Deciding which processes are to be loaded into memory when memory space becomes available.
   - Allocating &deallocating memory space as needed.

3. **File Management:**

   File management is one of the most important components of an OS computer can store information on several different types of physical media magnetic tape, magnetic disk & optical disk are the most common media. Each medium is controlled by a device such as disk drive or tape drive those has unique characteristics. These characteristics include access speed, capacity, data transfer rate & access method (sequential or random).For convenient use of computer system the OS provides a uniform logical view of information storage. The OS abstracts from the physical properties of its storage devices to define a logical storage unit the file. A file is collection of related information defined by its creator. The OS is responsible for the following activities of file management.

   - Creating & deleting files.
   - Creating & deleting directories.
   - Supporting primitives for manipulating files & directories.
   - Mapping files into secondary storage.
   - Backing up files on non-volatile media.

4. **I/O System Management:**

One of the purposes of an OS is to hide the peculiarities of specific hardware devices from the user. For example, in UNIX the peculiarities of I/O devices are hidden from the bulk of the OS itself by the I/O subsystem. The I/O subsystem consists of:

- A memory management component that includes buffering, catching & spooling.
- A general device- driver interfaces drivers for specific hardware devices. Only the device driver knows the peculiarities of the specific device to which it is assigned.

5. **Secondary Storage Management:**

The main purpose of computer system is to execute programs. These programs with the data they access must be in main memory during execution. As the main memory is too small to accommodate all data & programs & because the data that it holds are lost when power is lost. The computer system must provide secondary storage to back-up main memory. Most modern computer systems are disks as the storage medium to store data & program. The operating system is responsible for the following activities of disk management.

- Free space management.
- Storage allocation.
- Disk scheduling

Because secondary storage is used frequently it must be used efficiently.

Networking:

A distributed system is a collection of processors that don't share memory peripheral devices or a clock. Each processor has its own local memory & clock and the processor communicate with one another through various communication lines such as high speed buses or networks. The processors in the system are connected through communication networks which are configured in a number of different ways. The communication network design must consider message routing & connection strategies are the problems of connection & security.

Protection or security:

If a computer system has multi users & allow the concurrent execution of multiple processes then the various processes must be protected from one another's activities. For that purpose, mechanisms ensure that files, memory segments, CPU & other resources can be operated on by only those processes that have gained proper authorization from the OS.

Command interpretation:

One of the most important functions of the OS is connected interpretation where it acts as the interface between the user & the OS.

View of operating system

• User view:

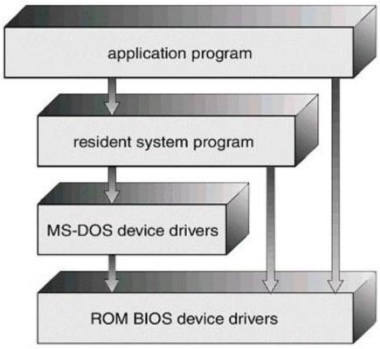
The user view of the computer varies by the interface being used. The examples are -windows XP, vista, windows 7 etc. Most computer user sit in the in front of personal computer (pc) in this case the operating system is designed mostly for easy use with some attention paid to resource utilization. Some user sit at a terminal connected to a mainframe/minicomputer. In this case other users are accessing the same computer through the other terminals. There user are share resources and may exchange the information. The operating system in this case is designed to maximize resources utilization to assume that all available CPU time, memory and I/O are used efficiently and no individual user takes more than his/her fair and share.The other users sit at workstations connected to network of other workstations and servers. These users have dedicated resources but they share resources such as networking and servers like file, compute and print server. Here the operating system is designed to compromise between individual usability and resource utilization.

• System view:

From the computer point of view the operating system is the program which is most intermediate with the hardware. An operating system has resources as hardware and software which may be required to solve a problem like CPU time, memory space, file storage space and I/O devices and so on. That's why the operating system acts as manager of these resources. Another view of the operating system is it is a control program. A control program manages the execution of user programs to present the errors in proper use of the computer. It is especially concerned of the user the operation and controls the I/O devices.

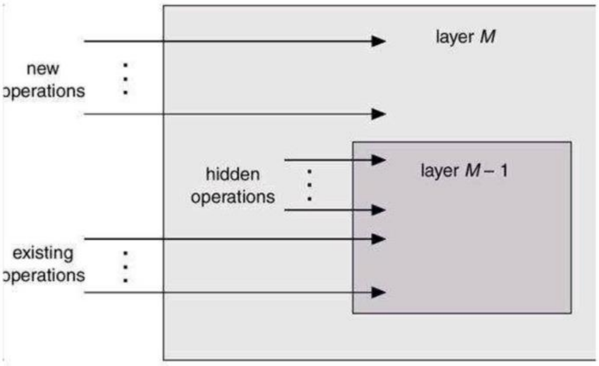
System structure

1. **Simple structure:** There are several commercial system that don't have a well- defined structure such operating systems begins as small, simple & limited systems and then grow beyond their original scope. MS-DOS is an example of such system. It was not divided into modules carefully. Another example of limited structuring is the UNIX operating system.



(MS DOS Structure)

2. **Layered approach:** In the layered approach, the OS is broken into a number of layers (levels) each built on top of lower layers. The bottom layer (layer o) is the hardware & top most layer (layer N) is the user interface. The main advantage of the layered approach is modularity.



- The layers are selected such that each users functions (or operations) & services of only lower layer.
- This approach simplifies debugging & system verification, i.e. the first layer can be debugged without concerning the rest of the system. Once the first layer is debugged, its correct functioning is assumed while the 2nd layer is debugged & so on.
- If an error is found during the debugging of a particular layer, the error must be on that layer because the layers below it are already debugged. Thus the design & implementation of the system are simplified when the system is broken down into layers.
- Each layer is implemented using only operations provided by lower layers. A layer doesn't need to know how these operations are implemented; it only needs to know what these operations do.
- The layer approach was first used in the operating system. It was defined in six layers.

Layers	Functions
5	User Program
4	I/O Management
3	Operator Process Communication
2	Memory Management
1	CPU Scheduling
0	Hardware

The main disadvantage of the layered approach is:

- The main difficulty with this approach involves the careful definition of the layers, because a layer can use only those layers below it. For example, the device driver for the disk space used by virtual memory algorithm must be at a level lower than that of the memory management routines, because memory management requires the ability to use the disk space.
- It is less efficient than a non layered system (Each layer adds overhead to the system call & the net result is a system call that take longer time than on a non layered system).

System Calls:

System calls provide the interface between a process & the OS. These are usually available in the form of assembly language instruction. Some systems allow system calls to be made directly from a high level language program like C, BCPL and PERL etc. systems calls occur in different ways depending on the computer in use. System calls can be roughly grouped into 5 major categories.

1. **Process Control:**
   - **End, abort:** A running program needs to be able to has its execution either normally (end) or abnormally (abort).
   - **Load, execute:**A process or job executing one program may want to load and executes another program.
   - **Create Process, terminate process:** There is a system call specifying for the purpose of creating a new process or job (create process or submit job). We may want to terminate a job

or process that we created (terminates process, if we find that it is incorrect or no longer needed).

- **Get process attributes, set process attributes:** If we create a new job or process we should able to control its execution. This control requires the ability to determine & reset the attributes of a job or processes (get process attributes, set process attributes).
- **Wait time:** After creating new jobs or processes, we may need to wait for them to finish their execution (wait time).
- **Wait event, signal event:** We may wait for a specific event to occur (wait event). The jobs or processes then signal when that event has occurred (signal event).

2. **File Manipulation:**
   - **Create file, delete file:** We first need to be able to create & delete files. Both the system calls require the name of the file & some of its attributes.
   - **Open file, close file:** Once the file is created, we need to open it & use it. We close the file when we are no longer using it.
   - **Read, write, reposition file:** After opening, we may also read, write or reposition the file (rewind or skip to the end of the file).
   - **Get file attributes, set file attributes:** For either files or directories, we need to be able to determine the values of various attributes & reset them if necessary. Two system calls get file attribute & set file attributes are required for their purpose.
3. **Device Management:**
   - **Request device, release device:** If there are multiple users of the system, we first request the device. After we finished with the device, we must release it.
   - **Read, write, reposition:** Once the device has been requested & allocated to us, we can read, write & reposition the device.
4. **Information maintenance:**
   - **Get time or date, set time or date:** Most systems have a system call to return the current date & time or set the current date & time.

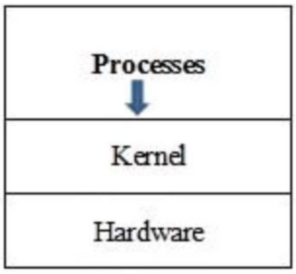
- **Get system data, set system data:** Other system calls may return information about the system like number of current users, version number of OS, amount of free memory etc.
- **Get process attributes, set process attributes:** The OS keeps information about all its processes & there are system calls to access this information.

5. **Communication:** There are two modes of communication such as:

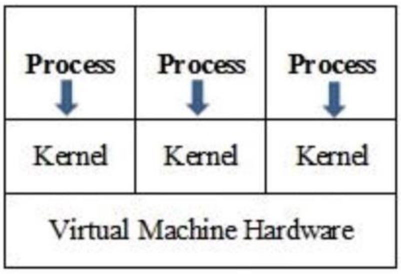
- **Message passing model:** Information is exchanged through an inter process communication facility provided by operating system. Each computer in a network has a name by which it is known. Similarly, each process has a process name which is translated to an equivalent identifier by which the OS can refer to it. The get hostid and get processed systems calls to do this translation. These identifiers are then passed to the general purpose open & close calls provided by the file system or to specific open connection system call. The recipient process must give its permission for communication to take place with an accept connection call. The source of the communication known as client & receiver known as server exchange messages by read message & write message system calls. The close connection call terminates the connection.
- **Shared memory model:** processes use map memory system calls to access regions of memory owned by other processes. They exchange information by reading & writing data in the shared areas. The processes ensure that they are not writing to the same location simultaneously.

6. Virtual Machines

7. By using CPU scheduling & virtual memory techniques an operating system can create the illusion of multiple processes, each executing on its own processors & own virtual memory. Each processor is provided a virtual copy of the underlying computer. The resources of the computer are shared to create the virtual machines. CPU scheduling can be used to create the appearance that users have their own processor.



(Non virtual Machine)



(Virtual Machine)

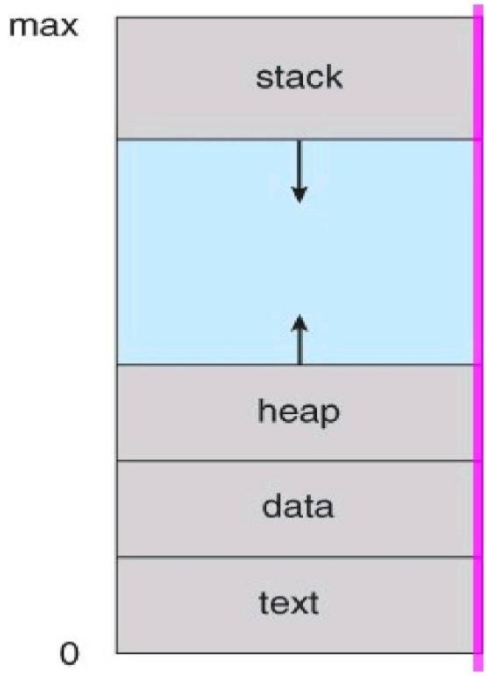
8.

9. **Implementation:** Although the virtual machine concept is useful, it is difficult to implement since much effort is required to provide an exact duplicate of the underlying machine. The CPU is being multiprogrammed among several virtual machines, which slows down the virtual machines in various ways.

10. **Difficulty:** A major difficulty with this approach is regarding the disk system. The solution is to provide virtual disks, which are identical in all respects except size. These are known as mini disks in IBM's VM OS. The sum of sizes of all mini disks should be less than the actual amount of physical disk space available.

Process Concept

- A question that arises in discussing operating systems involves what to call all the CPU activities. A batch system executes jobs, whereas a time-shared system has user programs, or tasks.
- Even on a single-user system such as Microsoft Windows, a user may be able to run several programs at one time: a word processor, a web browser, and an e-mail package.
- Even if the user can execute only one program at a time, the operating system may need to support its own internal programmed activities, such as memory management.
- It would be misleading to avoid the use of commonly accepted terms that include the word job (such as job scheduling) simply because process has superseded job.
- A process generally also includes the process stack, which contains temporary data (such as function parameters, return addresses, and local variables), and a data section, which contains global variables.



STACK :

The process stack contains temporary data such as methods , permanents returns address and local variable.

Heap:

This heap dynamically allocated a memory to a process during its run time.

Data :

This sections contains global variable.

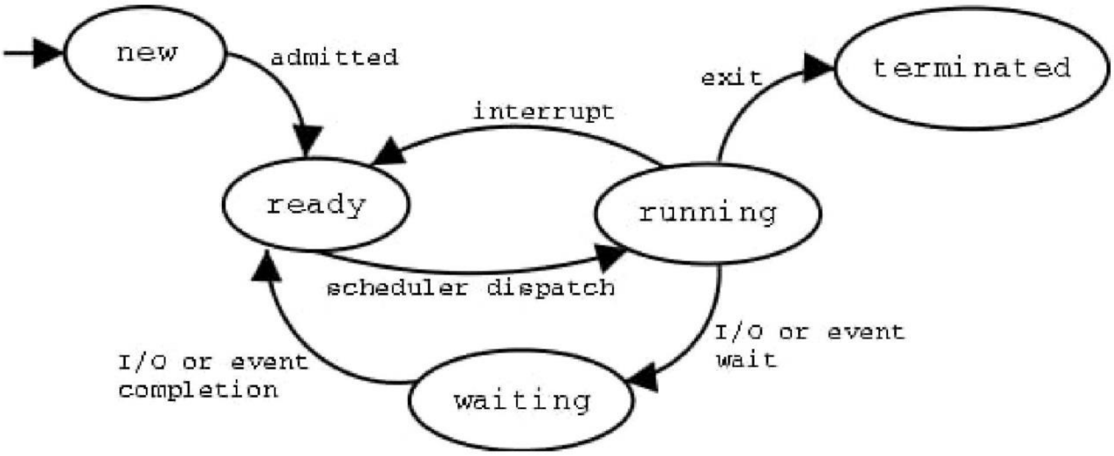
Text:

This includes current activities represent by the value of programs contains of the processor and register.

Process State

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. Each process may be in one of the following states:

- New. The process is being created.
- Running. Instructions are being executed.
- Waiting. The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- Ready. The process is waiting to be assigned to a processor.
- Terminated.

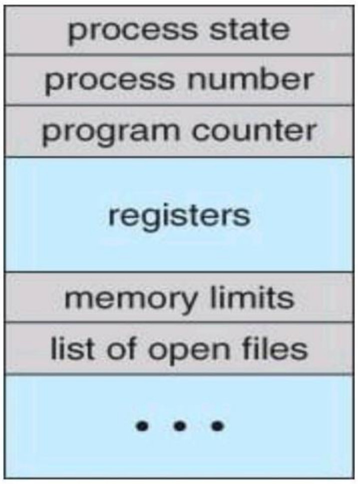


The process has finished execution. These names are arbitrary, and they vary across operating systems.

The states that they represent are fotind on all systems, however. Certain operating systems also more finely delineate process states.

Process Control Block

Each process is represented in the operating system by a process control block (PCB)—also called a task control block. It contains many pieces of information associated with a specific process, including these: •



- **Program counter**

The counter indicates the address of the next instruction to be executed for this process.

- **CPU registers**

The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward

- **CPU-scheduling information**

This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

- **Memory-management information**

This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system .

- **Accounting information**

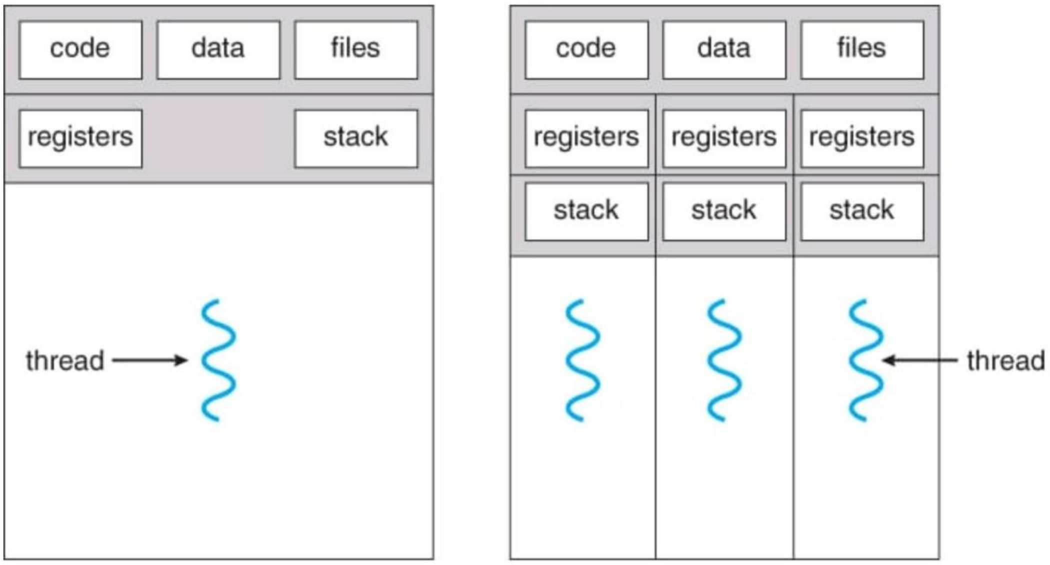
This information includes the amount of CPU and real time used, time limits, account mimbers, job or process numbers, and so on.

- **I/O status information**

This information includes the list of I/O devices allocated to the process, a list of open files, and so on. In brief, the PCB simply serves as the repository for any information that may vary from process to process.

Threads

- The process model discussed so far has implied that a process is a program that performs a single thread of execution. For example, when a process is running a word-processor program, a single thread of instructions is being executed. This single thread of control allows the process to perform only one task at one time.



single-threaded process multithreaded process

- The user cannot simultaneously type in characters and run the spell checker within the same process, for example. Many modern operating systems have extended the process concept to allow a process to have multiple threads of execution and thus to perform more than one task at a time.

Multithreading Models

Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach.

In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

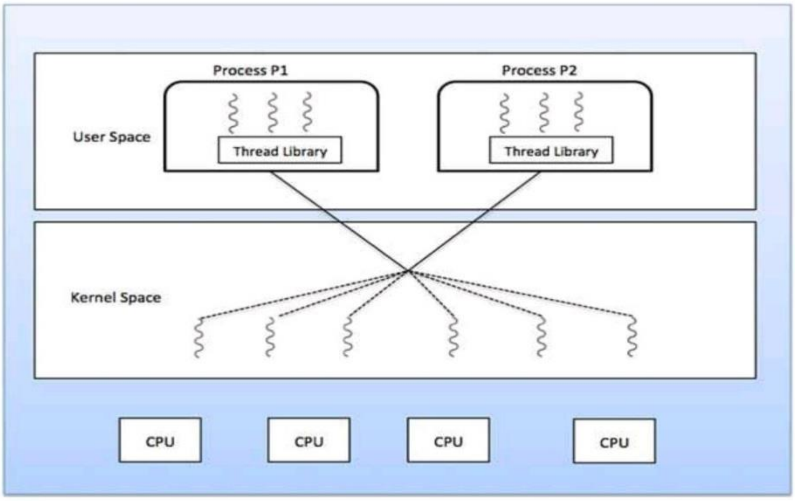
Many to many relationship.

Many to one relationship.

One to one relationship.

Many to Many Model

The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.

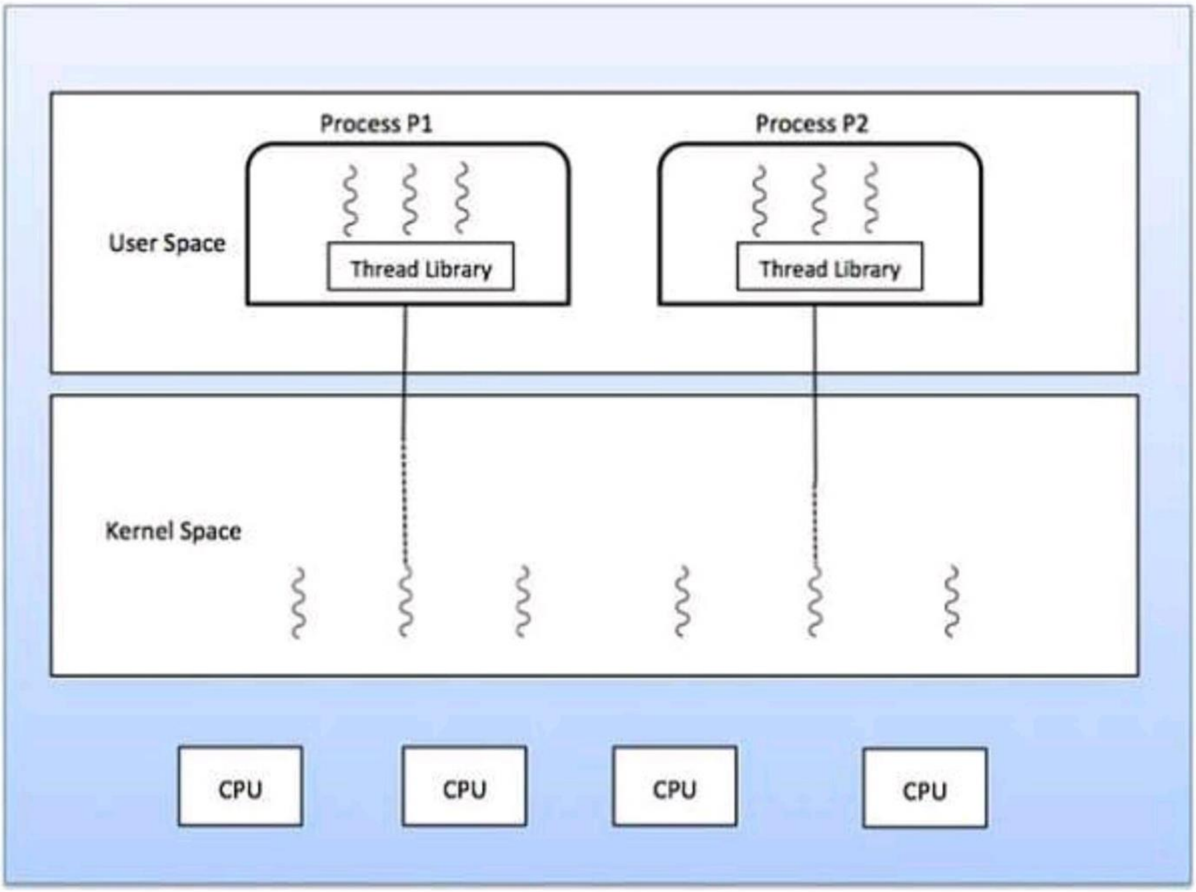


- The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads.
- In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine.
- This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.

Many to One Model

- Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library.
- When thread makes a blocking system call, the entire process will be blocked.
- Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

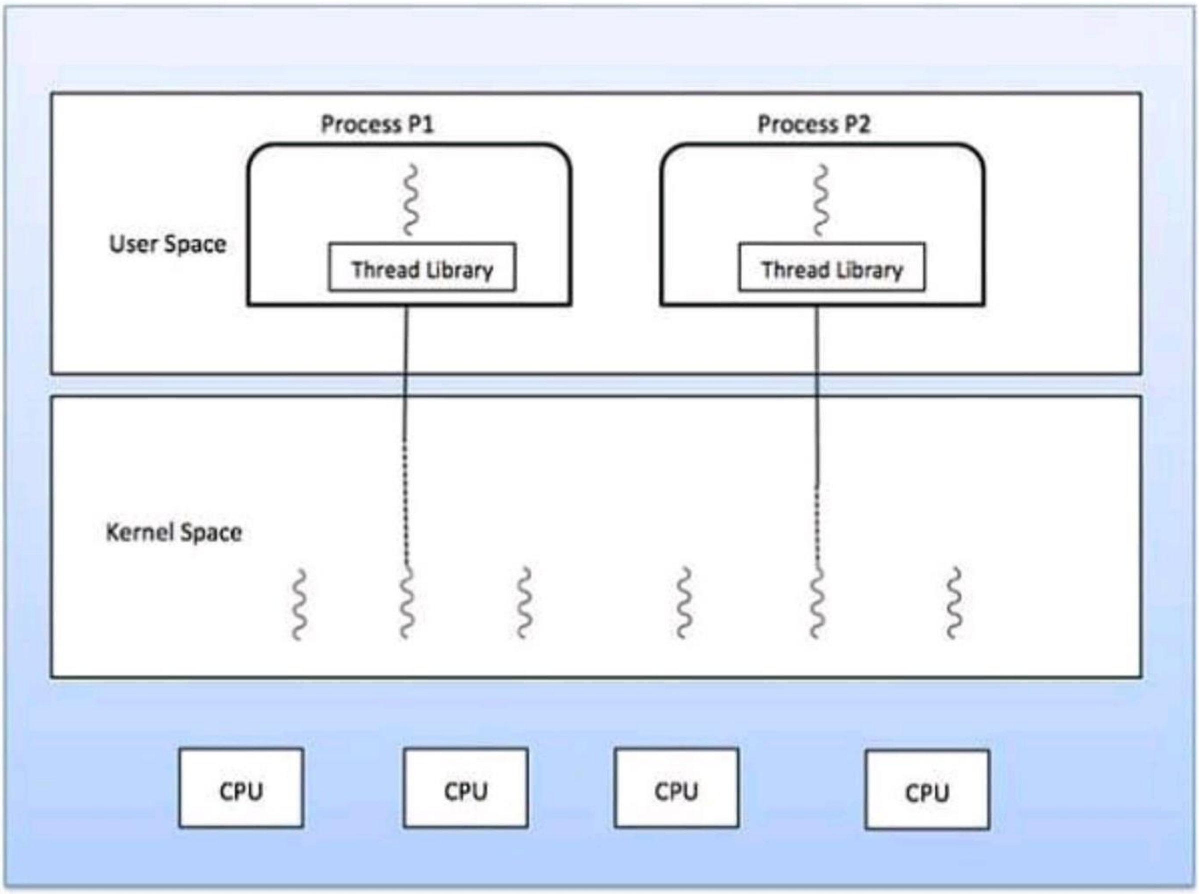
- If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.



One to One Model

- There is one-to-one relationship of user-level thread to the kernel-level thread.
- This model provides more concurrency than the many-to-one model. It also allows another thread to run when a thread makes a blocking system call.
- It supports multiple threads to execute in parallel on microprocessors.

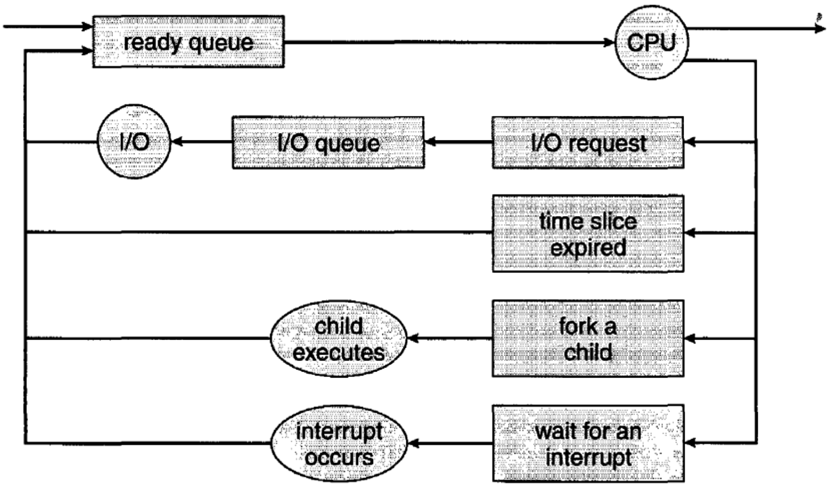
Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.



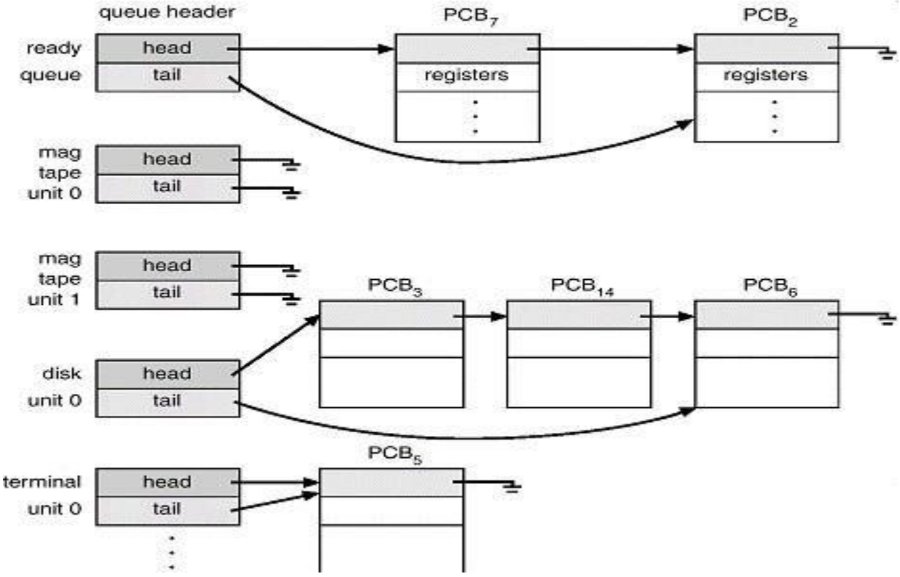
MODULE:II

Process scheduling:

- Scheduling is a fundamental function of OS. When a computer is multiprogrammed, it has multiple processes completing for the CPU at the same time.
- If only one CPU is available, then a choice has to be made regarding which process to execute next. This decision making process is known as scheduling and the part of the OS that makes this choice is called a scheduler. The algorithm it uses in making this choice is called scheduling algorithm.
- **Scheduling queues:** As processes enter the system, they are put into a job queue. This queue consists of all process in the system.
- The process that are residing in main memory and are ready & waiting to execute or kept on a list called ready queue.



This queue is generally stored as a linked list. A ready queue header contains pointers to the first & final PCB in the list. The PCB includes a pointer field that points to the next PCB in the ready queue. The lists of processes waiting for a particular I/O device are kept on a list called device queue. Each device has its own device queue. A new process is initially put in the ready queue. It waits in the ready queue until it is selected for execution & is given the CPU.



SCHEDULERS:

- A process migrates between the various scheduling queues throughout its life-time purposes. The OS must select for scheduling processes from these queues in some fashion.
- This selection process is carried out by the appropriate scheduler. In a batch system, more processes are submittedand then executed immediately. So these processes are spooled to a mass storage device like disk, where they are kept for later execution.

Types of schedulers:

There are 3 types of schedulers mainly used:

1. **Long term scheduler:** Long term scheduler selects process from the disk & loads them into memory for execution.

   It controls the degreeof multi-programming i.e. no. of processes in memory.

   It executes less frequently than other schedulers. If the degree of multiprogramming is stable than the average rate of process creation is equal to the average departure rate of processes leaving the system. So, the long term scheduler is needed to be invoked only when a process leaves the system.

   Due to longer intervals between executions it can afford to take more time to decide which process should be selected for execution. Most processes in the CPU are either I/O bound or CPU bound. An I/O bound process (an interactive ‘C’ program is one that spends most of its time in I/O operation than it spends in doing I/O operation. A CPU bound process is one that spends more of its time in doing computations than I/O operations (complex sorting program).

It is important that the long term scheduler should select a good mix of I/O bound & CPU bound processes.

2. **Short - term scheduler:** The short term scheduler selects among the process that are ready to execute & allocates the CPU to one of them.

The primary distinction between these two schedulers is the frequency of their execution. The short-term scheduler must select a new process for the CPU quite frequently.

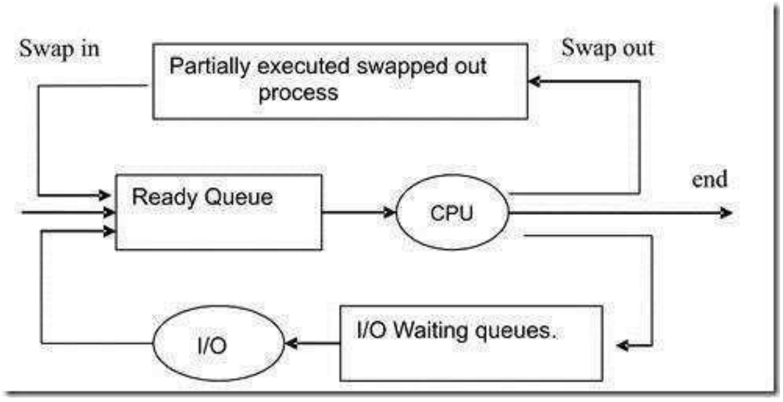
It must execute at least one in 100ms. Due to the short duration of time between executions, it must be very fast.

3. **Medium - term scheduler:** some operating systems introduce an additional intermediate level of scheduling known as medium - term scheduler.

The main idea behind this scheduler is that sometimes it is advantageous to remove processes from memory & thus reduce the degree of multiprogramming.

At some later time, the process can be reintroduced into memory & its execution can be continued from where it had left off.

This is called as swapping. The process is swapped out & swapped in later by medium term scheduler. Swapping is necessary to improve theprocess miss or due to some change in memory requirements, the available memory limit is exceeded which requires some memory to be freed up.



CPU Scheduling Algorithm:

CPU Scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated first to the CPU. There are four types of CPU scheduling that exist.

1. **First Come, First Served Scheduling (FCFS) Algorithm:**This is the simplest CPU scheduling algorithm. In this scheme, the process which requests the CPU first, that is allocated to the CPU first.

   The implementation of the FCFS algorithm is easily managed with a FIFO queue.

When a process enters the ready queue its PCB is linked onto the rear of the queue. The average waiting time under FCFS policy is quiet long. Consider the following example:

Process	**CPU time**
P_1	3
P_2	5
P_3	2
P_4	4

Using FCFS algorithm find the average waiting time and average turnaround time if the order is

P_1, P_2, P_3, P_4.

Solution: If the process arrived in the order P_1, P_2, P_3, P_4 then according to the FCFS the Gantt chart will be:

P_1	P_2	P_3	P_4

0 3 8 10 14

The waiting time for process $P_1 = 0$, $P_2 = 3$, $P_3 = 8$, $P_4 = 10$ then the turnaround time for process $P_1 = 0 + 3 = 3$, $P_2 = 3 + 5 = 8$, $P_3 = 8 + 2 = 10$, $P_4 = 10 + 4 =14$.

Then average waiting time = (0 + 3 + 8 + 10)/4 = 21/4 = 5.25

Average turnaround time = (3 + 8 + 10 + 14)/4 = 35/4 = 8.75

The FCFS algorithm is non preemptive means once the CPU has been allocated to a process then the process keeps the CPU until the release the CPU either by terminating or requesting I/O.

2. **Shortest Job First Scheduling (SJF) Algorithm:** This algorithm associates with each process if the CPU is available.

   This scheduling is also known as shortest next CPU burst, because the scheduling is done by examining the length of the next CPU burst of the process rather than its total length. Consider the following example:

Process	**CPU time**
P_1	3
P_2	5
P_3	2
P_4	4

Solution: According to the SJF the Gantt chart will be

P_3	P_1	P_2	P_4

0 2 5 9 14

The waiting time for process $P_1 = 0$, $P_2 = 2$, $P_3 = 5$, $P_4 = 9$ then the turnaround time for process $P_3 = 0 + 2 = 2$, $P_1 = 2 + 3 = 5$, $P_4 = 5 + 4 = 9$, $P_2 = 9 + 5 = 14$.

Then average waiting time = (0 + 2 + 5 + 9)/4 = 16/4 = 4

Average turnaround time = (2 + 5 + 9 + 14)/4 = 30/4 = 7.5

The SJF algorithm may be either preemptive or non preemptive algorithm. The preemptive SJF is also known as shortest remaining time first.

Consider the following example.

Process	**Arrival Time**	**CPU time**
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

In this case the Gantt chart will be

P_1	P_2	P_4	P_1	P_3

0 1 5 10 17 26

The waiting time for process

$P_1 = 10 - 1 = 9$

$P_2 = 1 – 1 = 0$

P_3 = 17 – 2 = 15

P_4 = 5 – 3 = 2

The average waiting time = (9 + 0 + 15 + 2)/4 = 26/4 = 6.5

3. **Priority Scheduling Algorithm:** In this scheduling a priority is associated with each process and the CPU is allocated to the process with the highest priority.

   Equal priority processes are scheduled in FCFS manner. Consider the following example:

Process	**Arrival**	**CPU**
	Time	**time**
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

According to the priority scheduling the Gantt chart will be

P_2	P_5	P_1	P_3	P_4

0 1 6 16 18 19

The waiting time for process

P_1 = 6

P_2 = 0

P_3 = 16

P_4 = 18

P_4 = 1

The average waiting time = (0 + 1 + 6 + 16 + 18)/5 = 41/5 = 8.2

4. **Round Robin Scheduling Algorithm:** This type of algorithm is designed only for the time sharing system.

It is similar to FCFS scheduling with preemption condition to switch between processes.

A small unit of time called quantum time or time slice is used to switch between the processes.

The average waiting time under the round robin policy is quiet long.

Consider the following example:

Process	**CPU time**
P_1	3
P_2	5
P_3	2
P_4	4

Time Slice = 1 millisecond.

P_1	P_2	P_3	P_4	P_1	P_2	P_3	P_4	P_1	P_2	P_4	P_2	P_4	P_2

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

The waiting time for process

$P_1 = 0 + (4 – 1) + (8 – 5) = 0 + 3 + 3 = 6$

$P_2 = 1 + (5 – 2) + (9 – 6) + (11 – 10) + (12 – 11) + (13 – 12) = 1 + 3 + 3 + 1 + 1 + 1 = 10$

$P_3 = 2 + (6 – 3) = 2 + 3 = 5$

$P_4 = 3 + (7 – 4) + (10 – 8) + (12 – 11) = 3 + 3 + 2 + 1 = 9$

The average waiting time = (6 + 10 + 5 + 9)/4 = 7.5

Process co_ordination:

Process coordination or concurrency control deals with mutual exclusion and synchronization Mutual exclusion-ensure that two concurrent activities do not access shared data (resource) at the same time, critical region-set of instructions that only one process can execute.

Synchronization-asing a condition so coordinate the actions of concurrent activities. A generalization of mutual exclusion.

When considering process coordination, we must keep in mind the following situations:

- Deadlock occurs when two activities are waiting each other and neither can proceed. For example.

Suppose processes A and B each need two tape drives to continue, but only one drive has been assigned to each of them. If the system has only 2 drives, neither process can ever proceed.

- Survation occurs when a blocked activity is consistently passed over and not allowed to run. For

example Consider two CPU bound jobs, one running at a higher priority than the other.

The lower priority process will never be allowed to execute. As we shall see, some synchronization primitives lead to starvation.

Process Synchronization:

A co-operation process is one that can affect or be affected by other processes executing in the system.

Co-operating process may either directly share a logical address space or be allotted to the shared data only through files.

This concurrent access is known as Process synchronization.

Critical Section Problem:

Consider a system consisting of n processes (P_0, P_1, ………P_{n-1}) each process has a segment of code which is known as critical section in which the process may be changing common variable, updating a table, writing a file and so on.

The important feature of the system is that when the process is executing in its critical section no other process is to be allowed to execute in its critical section.

The execution of critical sections by the processes is a mutually exclusive.

The critical section problem is to design a protocol that the process can use to cooperate each process must request permission to enter its critical section.

The section of code implementing this request is the entry section. The critical section is followed on exit section. The remaining code is the remainder section.

Example:

```
       While (1)
         {
      Entry Section;
              Critical Section;
      Exit Section;
             Remainder Section;
         }
```

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

1. **Mutual Exclusion:** If process P_i is executing in its critical section then no any other process can be executing in their critical section.
2. **Progress:** If no process is executing in its critical section and some process wish to enter their critical sections then only those process that are not executing in their remainder section can enter its critical section next.
3. **Bounded waiting:** There exists a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request.

Semaphores:

For the solution to the critical section problem one synchronization tool is used which is known as semaphores.

A semaphore 'S' is an integer variable which is accessed through two standard operations such as wait and signal.

These operations were originally termed 'P' (for wait means to test) and 'V' (for single means to increment). The classical definition of wait is

```
Wait (S)
 {
     While (S <= 0)
        {
                Test;
            }
            S--;
        }
```

The classical definition of the signal is

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

In case of wait the test condition is executed with interruption and the decrement is executed without interruption.

Binary Semaphore:

A binary semaphore is a semaphore with an integer value which can range between 0 and 1. Let 'S' be a counting semaphore. To implement the binary semaphore we need following the structure of data. Binary Semaphores S_1, S_2; int C; Initially S_1 = 1, S2 = 0 and the value of C is set to the initial value of the counting semaphore 'S'.

Then the wait operation of the binary semaphore can be implemented as follows.

```
Wait
($S_1$) C-
-; if (C
< 0)
{
    Signal ($S_1$);
    Wait ($S_2$);
}
Signal ($S_1$);
```

The signal operation of the binary semaphore can be implemented as follows:

```
Wait ($S_1$);
C++;
if (C <=0)
Signal ($S_2$);
Else
Signal ($S_1$);
```

Classical Problem on Synchronization:

There are various types of problem which are proposed for synchronization scheme such as

- **Bounded Buffer Problem:** This problem was commonly used to illustrate the power of synchronization primitives.

In this scheme we assumed that the pool consists of ‘N’ buffer and each capable of holding one item. The ‘mutex’ semaphore provides mutual exclusion for access to the buffer pool and is initialized to the value one.

The empty and full semaphores count the number of empty and full buffer respectively. The semaphore empty is initialized to ‘N’ and the semaphore full is initialized to zero. This problem is known as procedure and consumer problem.

The code of the producer is producing full buffer and the code of consumer is producing empty buffer. The structure of producer process is as follows:

```
do { produce an
item in nextp
. . . . . . . . . . . .
Wait (empty);
Wait  (mutex);
. . . . . . . . . . .
add nextp to buffer
. . . . . . . . . . . .
Signal (mutex);
Signal (full);
} While (1);
```

The structure of consumer process is as follows:

```
do {
Wait (full);
Wait
(mutex);
. . . . . . . . . . .
Remove an item from buffer to nextc
. . . . . . . . . . .
Signal (mutex);
Signal (empty);
. . . . . . . . . . . .
Consume the item in nextc;
. . . . . . . . .. . . .. .
} While (1);
```

- **Reader Writer Problem:** In this type of problem there are two types of process are used such as Reader process and Writer process.

The reader process is responsible for only reading and the writer process is responsible for writing.

This is an important problem of synchronization which has several variations like ○ The simplest one is referred as first reader writer problem which requires that no reader will be kept waiting unless a writer has obtained permission to use the shared object.

In other words no reader should wait for other reader to finish because a writer is waiting. ○ The second reader writer problem requires that once a writer is ready then the writer performs its write operation as soon as possible.

The structure of a reader process is as follows:

```
Wait  (mutex);
Read
count++;      if
(read      count
== 1)
Wait (wrt);
Signal
(mutex);
. . . . . . . . . . .
```

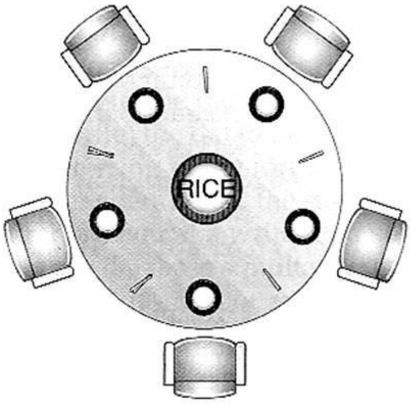
```
Reading is performed
. . . . . . . . . . .
Wait  (mutex);
Read count --;
if (read count
== 0)
Signal (wrt);
Signal (mutex);
```

The structure of the writer process is as follows:

```
Wait (wrt);
Writing is performed;
Signal (wrt);
```

- **Dining Philosopher Problem:** Consider 5 philosophers to spend their lives in thinking & eating.

A philosopher shares common circular table surrounded by 5 chairs each occupies by one philosopher. In the center of the table there is a bowl of rice and the table is laid with 6 chopsticks as shown in below figure.



When a philosopher thinks she does not interact with her colleagues. From time to time a philosopher gets hungry and tries to pickup two chopsticks that are closest to her.

A philosopher may pickup one chopstick or two chopsticks at a time but she cannot pickup a chopstick that is already in hand of the neighbor.

When a hungry philosopher has both her chopsticks at the same time, she eats without releasing her chopsticks. When she finished eating, she puts down both of her chopsticks and starts thinking again.

This problem is considered as classic synchronization problem. According to this problem each chopstick is represented by a semaphore.

A philosopher grabs the chopsticks by executing the wait operation on that semaphore.

She releases the chopsticks by executing the signal operation on the appropriate semaphore. The structure of dining philosopher is as follows:

```
do{
Wait ( chopstick
[i]); Wait (chopstick
[(i+1)%5]);
. . . . . . . . . . . . .
Eat
. . . . . . . . . . . . .
Signal (chopstick [i]);
Signal (chopstick
[(i+1)%5]);
. . . . . . . . . . . . .
Think
. . . . . . . . . . . . .
} While (1);
```

Critical section:

According to the critical section problem using semaphore all processes must share a semaphore variablemutex which is initialized to one. Each process must execute wait (mutex) before entering the critical section and execute the signal (mutex) after completing the execution but there are various difficulties may arise with this approach like:

Case 1: Suppose that a process interchanges the order in which the wait and signal operations on the semaphore mutex are executed, resulting in the following execution:

Signal (mutex);

. . . .

Critical Section

. . . .

Wait (mutex);

In this situation several processes may be executing in their critical sections simultaneously, which is violating mutual exclusion requirement.

Case 2: Suppose that a process replaces the signal (mutex) with wait (mutex). The execution is as follows: Wait (mutex);

. . . .

Critical Section

. . . .

Wait (mutex);

In this situation a deadlock will occur

Case 3: Suppose that a process omits the wait (mutex) and the signal (mutex). In this case the mutual exclusion is violated or a deadlock will occur.

To illustrate the various types or error generated by using semaphore there are some high level language constructs have been introduced such as critical region and monitor.

Critical region is also known as conditional critical regions. It constructs guards against certain simple errors associated with semaphore.

This high level language synchronization construct requires a variable V of type T which is to be shared among many processes. It is declared as V: shared T;

The variable V can be accessed only inside a region statement as like below:

```
Wait (mutex);
While (! B) {
First_count++
; if
(second_count
> 0)
   Signal (second_delay);
Else
      Signal (mutex);
Wait (first_delay);
First_count--;
Second_count++;
if (first_count> 0)
     Signal (first_delay);
Else
    Signal (second_delay);
Wait (second_delay);
Second_count --;
}
S;
if (first_count> 0)
     Signal (first_delay);
Else if (second_count> 0)
    Signal (second_delay);
Else
      Signal (mutex);
```

(Implementation of the conditional region constructs)

Where B is a Boolean variable which governs the access to the critical regions which is initialized to false.Mutex, First_delay and Second_delay are the semaphores which are initialized to 1, 0, and 0 respectively. First_count and Second_count are the integer variables which are initialized to zero.

Monitor:

It is characterized as a set of programmer defined operators.

Its representation consists of declaring of variables, whose value defines the state of an instance. The syntax of monitor is as follows. Monitor monitor_name

```
{
 Shared variable declarations
Procedure body P1 (………) {
       . . . . . . . .
         }
         Procedure body P2
(………) {          . . . . . . . .
         }
                  .
                  .
```

```
              .
        Procedure body Pn
(………) {        . . . . . . . .
        }
        {
             Initialization Code
        }
}
```

Atomic Transaction:

This section is related to the field of database system.

Atomic transaction describes the various techniques of database and how they are can be used by the operating system.

It ensures that the critical sections are executed automatically. To determine how the system should ensure atomicity we need first to identify the properties of the devices used to for storing the data accessed by the transactions. The various types storing devices are as follows:

- **Volatile Storage:** Information residing in volatile storage does not survive in case of system crash. Example of volatile storage is main memory and cache memory.
- **Non volatile Storage:** Information residing in this type of storage usually survives in case of system crash.

- Examples are Magnetic Disk, Magnetic Tape and Hard Disk.
- **Stable Storage:** Information residing in stable storage is never lost. Example is non volatile cache memory.

The various techniques used for ensuring the atomicity are as follows:

1. **Log based Recovery:** This technique is used for achieving the atomicity by using data structure called log. A log has the following fields:
   a. **Transaction Name:** This is the unique name of the transaction that performed the write operation.
   b. **Data Item Name:** This is the unique name given to the data.
   c. **Old Value:** This is the value of the data before to the write operation.
   d. **New value:** This is the value of the data after the write operation.

   This recovery technique uses two processes such as Undo and Redo. Undo restores the value of old data updated by a transaction to the old values.

   Redo sets the value of the data updated by a transaction to the new values.
2. **Checkpoint:** In this principle system maintains the log. The checkpoint requires the following sequences of action.

a. Output all the log records from volatile storage into stable storage.

b. Output all modified data residing in volatile to the stable storage.

c. Output a checkpoint onto the stable storage.

T_0	T_1
Read (A)	
Write (A)	
Read (B)	
Write (B)	

transaction Read (A)
system Write (A) read
and Read (B) their Write
(B)

3. **Serializibility:** In this technique the executed serially in some arbitrary order. Consider a consisting two data items A and B which are both written by two transactions T_0 and T_1. Suppose that transactions are executed automatically in the order

T_0 followed by T_1. This execution sequence is known as schedule which is represented as below.

If transactions are overlapped then their execution resulting schedule is known as non-serial scheduling or concurrent schedule as like below:

T_0	T_1
Read (A)	
Write (A)	Read (A)
	Write (A)
Read (B)	
Write (B)	Read (B)
	Write (B)

4. **Locking:** This technique governs how the locks are acquired and released.
   There are two types of lock such as shared lock and exclusive lock. If a transaction T has obtained a shared lock (S) on data item Q then T

can read this item but cannot write. If a transaction T has obtained an exclusive lock (S) on data item Q then T can both read and write in the data item Q.

5. **Timestamp:** In this technique each transaction in the system is associated with unique fixed timestamp denoted by TS. This timestamp is assigned by the system before the transaction starts. If a transaction T_i has been assigned with a timestamp TS (T_i) and later a new transaction T_j enters the system then TS (T_i) < TS (T_j). There are two types of timestamp such as Wtimestamp and R-timestamp.

   W-timestamp denotes the largest timestamp of any transaction that performed write operation successfully.

   R-timestamp denotes the largest timestamp of any transaction that executed read operation successfully.

Deadlock:

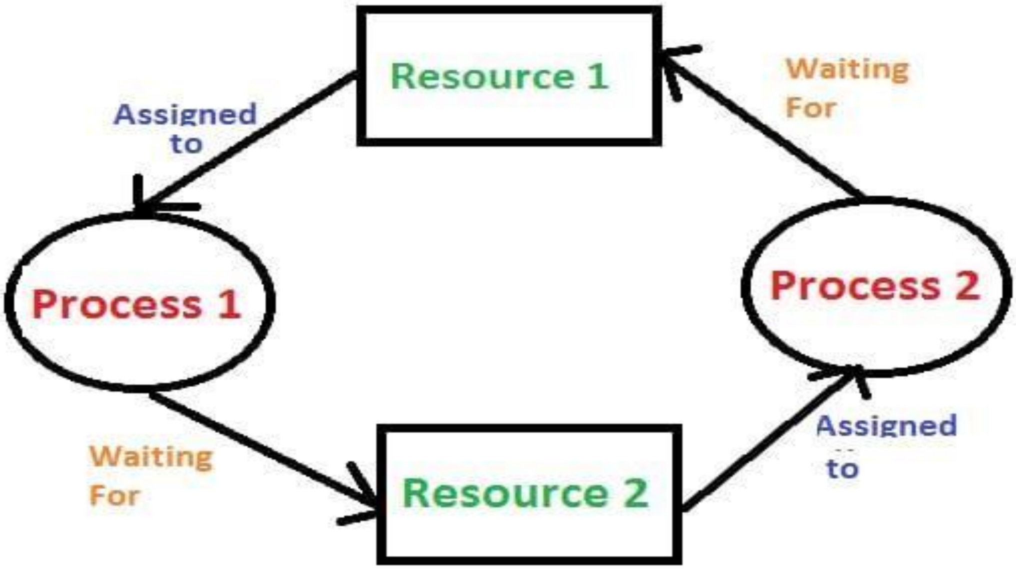
In a multiprogramming environment several processes may compete for a finite number of resources.

A process request resources; if the resource is available at that time a process enters the wait state.

Waiting process may never change its state because the resources requested are held by other waiting process. This situation is known as deadlock.

Example

- System has 2 disk drives.
- P1 and P2 each hold one disk drive and each needs another one.
- 2 train approaches each other at crossing, both will come to full stop and neither shall start until other has gone.



- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible **System Model:**

A system consists of a finite number of resources to be distributed among a number of competing processes.

The resources are partitioned into several types each of which consists of a number of identical instances. A process may utilized a resources in the following sequence

- **Request:** In this state one can request a resource.
- **Use:** In this state the process operates on the resource.
- **Release:** In this state the process releases the resources.

Deadlock Characteristics: In a deadlock process never finish executing and system resources are tied up.

A deadlock situation can arise if the following four conditions hold simultaneously in a system.

- **Mutual Exclusion:** At a time only one process can use the resources. If another process requests that resource, requesting process must wait until the resource has been released.
- **Hold and wait:** A process must be holding at least one resource and waiting to additional resource that is currently held by other processes.
- **No Preemption:** Resources allocated to a process can't be forcibly taken out from it unless it releases that resource after completing the task.
- **Circular Wait:** A set {P_0, P_1, …….P_n} of waiting state/ process must exists such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for the resource that is held by P_2 ….. $P_{(n-1)}$ is waiting

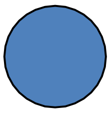
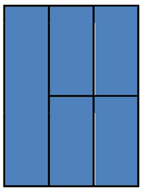
for the resource that is held by P_n and P_n is waiting for the resources that is held by P_4.

Resource Allocation Graph:

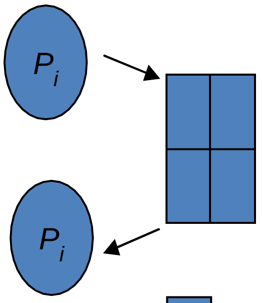
Deadlock can be described more clearly by directed graph which is called system resource allocation graph.

The graph consists of a set of vertices ‘V’ and a set of edges ‘E’. The set of vertices ‘V’ is partitioned into two different types of nodes such as P = {P_1, P_2, …….P_n}, the set of all the active processes in the system and R = {R_1, R_2, …….R_m}, the set of all the resource type in the system. A directed edge from process P_i to resource type R_j is denoted by $P_i \rightarrow R_j$. It signifies that process P_i is an instance of resource type R_j and waits for that resource.

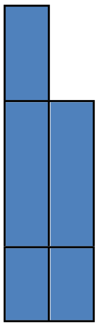
A directed edge from resource type R_j to the process P_i which signifies that an instance of resource type R_j has been allocated to process P_i. A directed edge $P_i \rightarrow R_j$ is called as request edge and $R_j \rightarrow P_i$ is called as assigned edge.

- Process 
- Resource Type with 4  instances

- $P_i$requests instance of R_j



- P_i is holding an instance of R_j



When a process P_i requests an instance of resource type R_j then a request edge is inserted as resource allocation graph.

When this request can be fulfilled, the request edge is transformed to an assignment edge.

When the process no longer needs access to the resource it releases the resource and as a result the assignment edge is deleted.

The resource allocation graph shown in below figure has the following situation.

- The sets P, R, E

$P = \{P_1, P_2, P_3\}$

$R = \{R_1, R_2, R_3, R_4\}$

$E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$

The resource instances

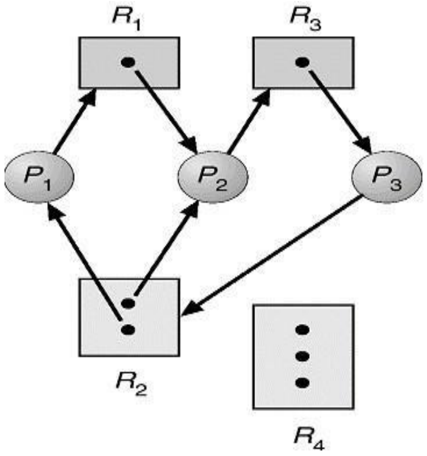
are Resource R_1 has

one instance

Resource R_2 has two instances.

Resource R_3 has one instance

Resource R_4 has three instances.



The process states are:

Process P_1 is holding an instance of R_2 and waiting for an instance of R_1.

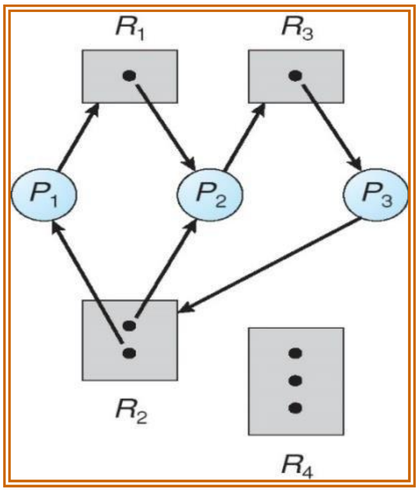
Process P_2 is holding an instance of R_1 and R_2 and waiting for an instance R_3.

Process P_3 is holding an instance of R_3.

The following example shows the resource allocation graph with a deadlock.

P1 -> R1 -> P2 -> R3 -> P3 -> R2 -> P1

P2 -> R3 -> P3 -> R2 -> P1



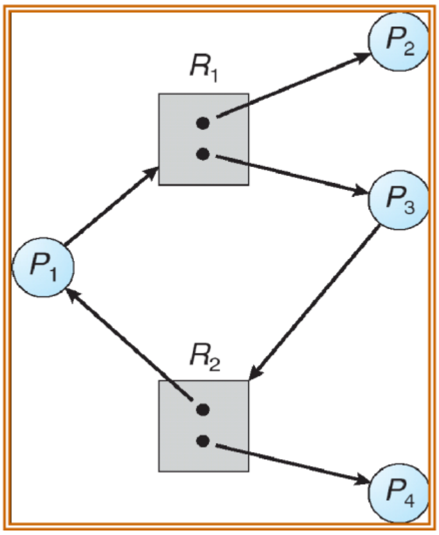
The following example shows the resource allocation graph with a cycle but no deadlock.

P1 -> R1 -> P3 -> R2 -> P1

No deadlock

P4 may release its instance of resource R2

- Then it can be allocated to P3



Methods for Handling Deadlocks

The problem of deadlock can deal with the following 3 ways.

We can use a protocol to prevent or avoid deadlock ensuring that the system will never enter to a deadlock state.

We can allow the system to enter a deadlock state, detect it and recover.

We can ignore the problem all together.

To ensure that deadlock never occur the system can use either a deadlock prevention or deadlock avoidance scheme. **Deadlock Prevention:**

Deadlock prevention is a set of methods for ensuring that at least one of these necessary conditions cannot hold.

Mutual Exclusion: The mutual exclusion condition holds for non sharable. The example is a printer cannot be simultaneously shared by several processes.

Sharable resources do not require mutual exclusive access and thus cannot be involved in a dead lock.

The example is read only files which are in sharing condition. If several processes attempt to open the read only file at the same time they can be guaranteed simultaneous access.

Hold and wait: To ensure that the hold and wait condition never occurs in the system, we must guaranty that whenever a process requests a resource it does not hold any other resources.

There are two protocols to handle these problems such as one protocol that can be used requires each process to request and be allocated all its resources before it begins execution.

The other protocol allows a process to request resources only when the process has no resource.

These protocols have two main disadvantages. First, resource utilization may be low, since many of the resources may be allocated but unused for a long period. Second, starvation is possible.

A process that needs several popular resources may have to wait indefinitely, because at least one of the resources that it needs is always allocated to some other process.

No Preemption: To ensure that this condition does not hold, a protocol is used. If a process is holding some resources and request another resource that cannot be immediately allocated to it.

The preempted one added to a list of resources for which the process is waiting. The process will restart only when it can regain its old resources, as well as the new ones that it is requesting. Alternatively if a process requests some resources, we first check whether they are available. If they are, we allocate them. If they are not available, we check whether they are allocated to some other process that is waiting for additional resources. If so, we preempt the desired resources from the waiting process and allocate them to the requesting process.

If the resources are not either available or held by a waiting process, the requesting process must wait.

Circular Wait: We can ensure that this condition never holds by ordering of all resource type and to require that each process requests resource in an increasing order of enumeration. Let R

= {R_1, R_2, …….R_n}be the set of resource types. We assign to each resource type a unique integer number, which allows us to compare two resources and to determine whether one precedes another in our ordering. Formally, we define a one to one function F: R → N, where N is the set of natural numbers.

For example, if the set of resource types R includes tape drives, disk drives and printers, then the function F might be defined as follows:
F (Tape Drive) = 1,

F (Disk Drive)
= 5, F
(Printer) = 12.

We can now consider the following protocol to prevent deadlocks: Each process can request resources only in an increasing order of enumeration.

That is, a process can initially request any number of instances of a resource type, say R_i. After that, the process can request instances of resource type R_j if and only if $F(R_j) > F(R_i)$.

If several instances of the same resource type are needed, defined previously, a process that wants to use the tape drive and printer at the same time must first request the tape drive and then request the printer.

Deadlock Avoidance

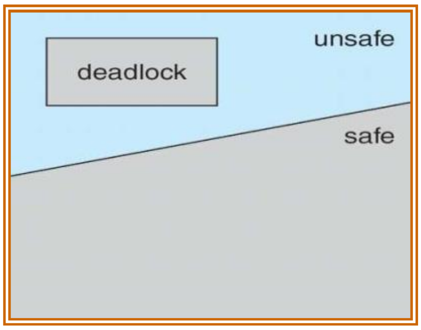
Requires additional information about how resources are to be used.Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

Safe State

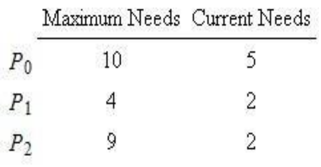
When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.Systems are in safe state if there exists a safe sequence of all process.

A sequence <P_1, P_2, …, P_n> of ALL the processes is the system such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, withj <i.That is:

- If P_i resource needs are not immediately available, then P_i can wait until all $P_j$have finished.
- When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
- When P_i terminates, P_{i+1} can obtain its needed resources, and so on.
- If system is in safe state => No deadlock
- If system in not in safe state => possibility of deadlock
- OS cannot prevent processes from requesting resources in a sequence that leads to deadlock
- Avoidance => ensue that system will never enter an unsafe state, prevent getting into deadlock

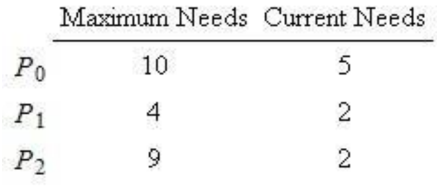


Example:



	Maximum Needs	Current Needs
P_0	10	5
P_1	4	2
P_2	9	2

- Suppose processes P0, P1, and P2 share 12 magnetic tape drives • Currently 9 drives are held among the processes and 3 are available
- Question: Is this system currently in a safe state?
- Answer: Yes!
  - Safe Sequence: <P1, P0, P2>



	Maximum Needs	Current Needs
P_0	10	5
P_1	4	2
P_2	9	2

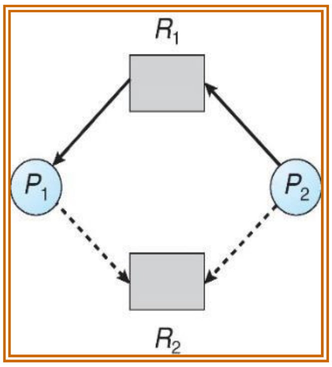
- Suppose process P2 requests and is allocated 1 more tape drive.
- Question: Is the resulting state still safe?
- Answer: No! Because there does not exist a safe sequence anymore.

Only P1 can be allocated its maximum needs.

IFP0 and P2 request 5 more drives and 6 more drives, respectively, then the resulting state will be deadlocked.

Resource Allocation Graph Algorithm

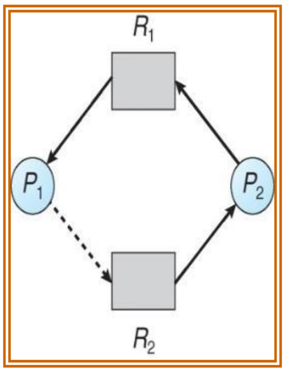
In this graph a new type of edge has been introduced is known as claim edge. Claim edge $P_i \rightarrow R_j$ indicates that process P_j may request resource R_j; represented by a dashed line.Claim edge converts to request edge when a process requests a resource.Request edge converted to an assignment edge when the resource is allocated to the process.When a resource is released by a process, assignment edge reconverts to a claim edge.Resources must be claimed a priori in the system.



P2 requesting R1, but R1 is already allocated to P1.

Both processes have a claim on resource R2

What happens if P2 now requests resource R2?



Cannot allocate resource R2 to process P2

Why? Because resulting state is unsafe

- P1 could request R2, thereby creating deadlock!

Use only when there is a single instance of each resource type

- Suppose that process P_i requests a resource R_j
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph.
- Here we check for safety by using cycle-detection algorithm.

Banker's Algorithm

This algorithm can be used in banking system to ensure that the bank never allocates all its available cash such that it can no longer satisfy the needs of all its customer.

This algorithm is applicable to a system with multiple instances of each resource type.

When a new process enter in to the system it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system.

Several data structure must be maintained to implement the banker's algorithm. Let,

- n = number of processes
- m = number of resources types

Available: Vector of length m. If Available[j] = k, there are k instances of resource type $R_j$available.

Max: n x m matrix. If Max [i,j] = k, then process $P_i$may request at most k instances of resource type R_j.

Allocation: n x m matrix. If Allocation[i,j] = k then P_i is currently allocated k instances of $R_{j.}$

Need: n x m matrix. If Need[i,j] = k, then P_i may need k more instances of $R_j$to complete its task.

Need [i,j] = Max[i,j] – Allocation [i,j].

Safety Algorithm

1. Let Workand Finish be vectors of length m and n, respectively.
   Initialize: Work = Available

Finish [i] = false for i = 0, 1, …,n- 1.

2. Find and i such that both:

(a) Finish [i] = false

(b) $Need_i \leq Work$

If no such i exists, go to step 4.

3. Work = Work + $Allocation_i$ Finish[i] = true
   go to step 2.
4. If Finish [i] == true for all i, then the system
   is in a safe state.

Resource Allocation Algorithm

Request = request vector for process P_i. If $Request_i$[j] = k then process P_i wants k instances of resource type $R_{j.}$

1. If Request$_i$≤Need$_i$go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If Request$_i$≤Available, go to step 3. Otherwise P$_i$ must wait, since resources are not available.
3. Pretend to allocate requested resources to P$_i$ by modifying the state as follows:

Available = Available – Request;

Allocation$_i$= Allocation$_i$ + Request$_i$;

Need$_i$=Need$_i$ – Request$_i$;

- If safe ⇒ the resources are allocated to Pi.
- If unsafe ⇒ Pi must wait, and the old resource-allocation state is restored

Example

- 5 processes P$_0$ through P$_4$;
- 3 resource types:

A (10 instances), B (5instances), and C (7 instances).

- Snapshot at time T$_0$:

	Allocation	Max	Available
	A B C	A B C	A B C
P$_0$	0 1 0	7 5 3	3 3 2
P$_1$	2 0 0	3 2 2	
P$_2$	3 0 2	9 0 2	

P_3	2 1 1	2 2 2
P_4	0 0 2	4 3 3

- The content of the matrix Need is defined to be Max – Allocation.

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

- The system is in a safe state since the sequence <P_1, P_3, P_4, P_2, P_0> satisfies safety criteria.

P_1 requests (1, 0, 2)

- Check that Request ≤ Available (that is, (1,0,2) ≤ (3,3,2) ⇒ true.

Allocation	Need	Available
	A B C	A B C A B C
P_0	0 1 0	7 4 3 2 3 0
P_1	3 0 2	0 2 0
P_2	3 0 1	6 0 0
P_3	2 1 1	0 1 1
P_4	0 0 2	4 3 1

- Executing safety algorithm shows that sequence <P_1, P_3, P_4, P_0, P_2> satisfies safety requirement.
- Can request for (3,3,0) by P_4 be granted? –NO
- Can request for (0,2,0) by P_0 be granted? –NO (Results Unsafe)

Deadlock Detection

If a system doesn't employ either a deadlock prevention or deadlock avoidance, then deadlock situation may occur. In this environment the system must provide

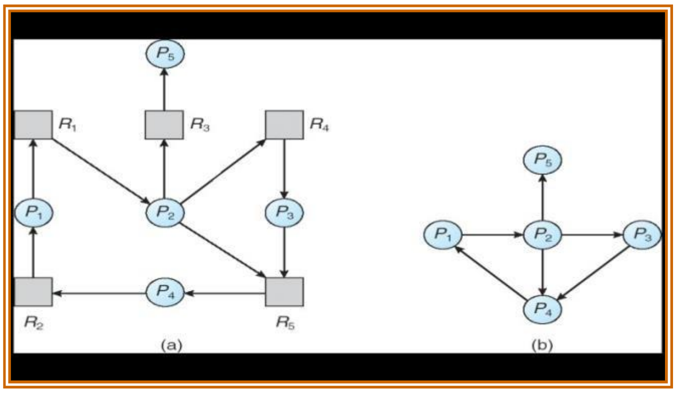
- An algorithm to recover from the deadlock.
- An algorithm to remove the deadlock is applied either to a system which pertains single in instance each resource type or a system which pertains several instances of a resource type.

Single Instance of each Resource type

If all resources only a single instance then we can define a deadlock detection algorithm which uses a new form of resource allocation graph called "Wait for graph".

We obtain this graph from the resource allocation graph by removing the nodes of type resource and collapsing the appropriate edges.

The below figure describes the resource allocation graph and corresponding wait for graph.



Resource-Allocation Correspondin

Graph wait-for graph

- For single instance
- P_i ->P_j(P_i is waiting for P_j to release a resource that P_i needs)
- P_i->P_j exist if and only if RAG contains 2 edges P_i ->R_q and R_q ->P_j for some resource R_q **Several Instances of a Resource type**

The wait for graph scheme is not applicable to a resource allocation system with multiple instances of reach resource type.

For this case the algorithm employs several data structures which are similar to those used in the banker's algorithm like available, allocation and request.

- **Available**: A vector of length m indicates the number of available resources of each type.
- **Allocation**: An n x m matrix defines the number of resources of each type currently allocated to each process.

- **Request**: An n x m matrix indicates the current request of each process. If Request $[i_j]$ = k, then process P_i is requesting k more instances of resource type. R_j.

1. Let Work and Finish be vectors of length m and n, respectively Initialize:

(a) Work = Available

(b) For i = 1,2, …, n, if $Allocation_i \neq 0$, then Finish[i] = false;otherwise, Finish[i] = true.

2. Find an index i such that both:

(a) Finish[i] == false

(b) $Request_i \leq Work$

If no such i exists, go to step 4.

3. Work = Work + Allocation

Finish [i] = true

Go to step 2

4. If Finish [i] = false, for some i, $1\leq i\leq n$, then the system is in a deadlock state. Moreover, if Finish [i] = false, then process P_i is deadlocked.

Recovery from Deadlock

When a detection algorithm determines that a deadlock exists, several alternatives exist. One possibility is to inform the operator that a deadlock has occurred, and to let the operator deal with the deadlock manually.

The other possibility is to let the system recover from the deadlock automatically. There are two options for breaking a deadlock. One solution is simply to abort one or more processes to break the circular wait.

The second option is to preempt some resources from one or more of the deadlocked processes.

Process Termination:

To eliminate deadlocks by aborting a process, we use one of two methods. In both methods, the system reclaims all resources allocated to the terminated processes.

- **Abort all deadlocked processes:** This method clearly will break the deadlock cycle, but at a great expense; these processes may have computed for a long time, and the results of these partial computations must be discarded and probably recomputed later.
- **Abort one process at a time until the deadlock cycle is eliminated:**This method incurs considerable overhead, since after each process is aborted, a deadlock detection algorithm must be invoked to determine whether any processes are still deadlocked.

Resource Preemption:

To eliminate deadlocks using resource preemption, we successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken.

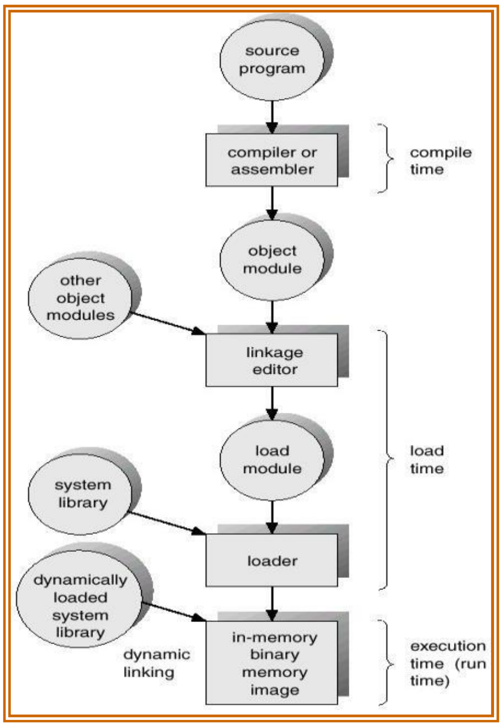
If preemption is required to deal with deadlocks, then three issues need to be addressed.

- **Selecting a victim:** Which resources and which processes are to be preempted? As in process termination, we must determine the order of preemption to minimize cost.
- Cost factors may include such parameters as the numbers of resources a deadlock process is holding, and the amount of time a deadlocked process has thus far consumed during its execution.
- **Rollback:** If we preempt a resource from a process, what should be done with that process? Clearly, it cannot continue with its normal execution; it is missing some needed resource. We must rollback the process to some safe state, and restart it from that state.
- **Starvation:** In a system where victim selection is based primarily on cost factors, it may happen that the same process is always picked as a victim. As a result, this process never completes its designated task, a starvation situation that needs to be dealt with in any practical system. Clearly, we must ensure that a process can be picked as a victim only a small finite number of times. The most common solution is to include the number of rollbacks in the cost factor.

MODULE:III

Memory Management

- Memory consists of a large array of words or bytes, each with its own address. The CPU fetches instructions from memory according to the value of the program counter. These instructions may cause additional loading from and storing to specific memory addresses.
- Memory unit sees only a stream of memory addresses. It does not know how they are generated.
- Program must be brought into memory and placed within a process for it to be run.
- Input queue – collection of processes on the disk that are waiting to be brought into memory for execution.
- User programs go through several steps before being run.



Address binding of instructions and data to memory addresses can happen at three different stages.

- **Compile time**: If memory location known a priori, absolute code can be generated; must recompile code if starting location changes.

  Example: .COM-format programs in MS-DOS.

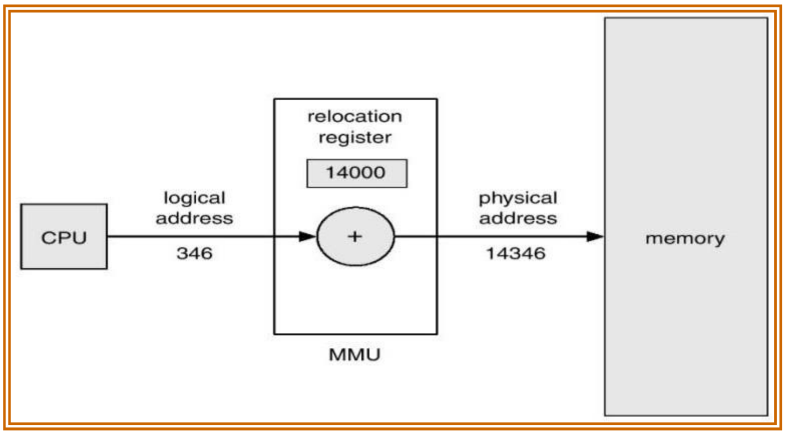
- **Load time**: Must generate relocatable code if memory location is not known at compile time.
- **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps
- (e.g., relocation registers).

Logical Versus Physical Address Space

- The concept of a logical address space that is bound to a separate physicaladdress space is central to proper memory management.

  ○ Logical address – address generated by the CPU; also referred to as virtual address. ○ Physical address – address seen by the memory unit.

- The set of all logical addresses generated by a program is a logical address space; the set of all physical addresses corresponding to these logical addresses are a physical address space.
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme.
- The run-time mapping from virtual to physical addresses is done by a hardware device called the memory management unit (MMU).



- This method requires hardware support slightly different from the hardware configuration. The base register is now called a relocation register. The value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
- The user program never sees the real physical addresses. The program can create a pointer to location 346, store it in memory, manipulate it and compare it to other addresses. The user program deals with logical addresses. The memory mapping hardware converts logical addresses into physical addresses. The final location of a referenced memory address is not determined until the reference is made.

Dynamic Loading

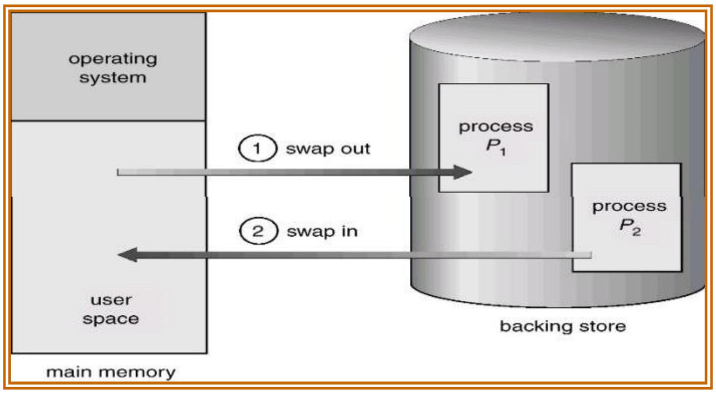
- Routine is not loaded until it is called.
- All routines are kept on disk in a relocatable load format.
- The main program is loaded into memory and is executed. When a routine needs to call another routine, the calling routine first checks to see whether the other the desired routine into memory and to update the program's address tables to reflect this change. Then control is passed to the newly loaded routine.
- Better memory-space utilization; unused routine is never loaded.
- Useful when large amounts of code are needed to handle infrequently occurring cases.

- No special support from the operating system is required.
- Implemented through program design.
- **Dynamic Linking**
- Linking is postponed until execution time.
- Small piece of code, stub, is used to locate the appropriate memory-resident library routine, or to load the library if the routine is not already present.
- When this stub is executed, it checks to see whether the needed routine is already in memory. If not, the program loads the routine into memory.
- Stub replaces itself with the address of the routine, and executes the routine.
- Thus the next time that code segment is reached, the library routine is executed directly, incurring no cost for dynamic linking.
- Operating system is needed to check if routine is in processes' memory address.
- Dynamic linking is particularly useful for libraries.

Swapping

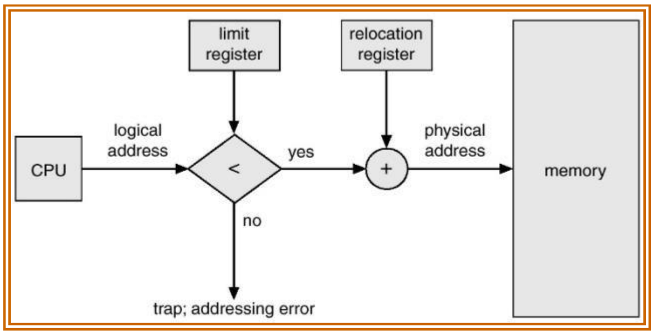
- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution. For example, assume a multiprogramming environment with a round robin CPU scheduling algorithm. When a quantum expires, the memory manager will start to swap out the process that just finished, and to swap in another process to the memory space that has been freed. In the mean time, the CPU scheduler will allocate a time slice to some other process in memory. When each process finished its quantum, it will be swapped with another process. Ideally, the memory manager can swap processes fast enough that some processes will be in memory, ready to execute, when the CPU scheduler wants to reschedule the CPU. The quantum must also be sufficiently large that reasonable amounts of computing are done between swaps.

- Roll out, roll in – swapping variant used for priority-based scheduling algorithms. If a higher priority process arrives and wants service, the memory manager can swap out the lower priority process so that it can load and execute lower priority process can be swapped back in and continued. This variant is some times called roll out, roll in. Normally a process that is swapped out will be swapped back into the same memory space that it occupied previously. This restriction is dictated by the process cannot be moved to different locations. If execution time binding is being used, then a process can be swapped into a different memory space, because the physical addresses are computed during execution time.
- Backing store – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images. It must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images. The system maintains a ready queue consisting of all processes whose memory images are scheduler decides to execute a process it calls the dispatcher. The dispatcher checks to see whether the next process in the queue is in memory. If not, and there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process. It then reloads registers as normal and transfers control to the selected process.
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped.
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows).



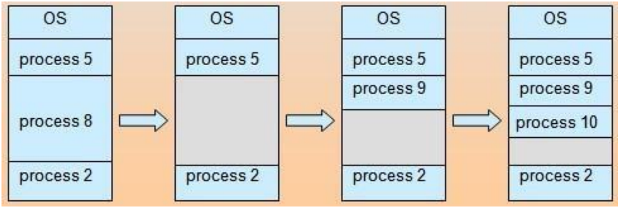
Contiguous Memory Allocation

- Main memory is usually divided into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector.
  - User processes, held in high memory.
- In contiguous memory allocation, each process is contained in a single contiguous section of memory.
- Single-partition allocation ○ Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data.
  - Relocation register contains value of smallest physical address; limit register contains range of logical addresses – each logical address must be less than the limit register.



- Multiple-partition allocation ○ Hole – block of available memory; holes of various size are scattered throughout memory.

- When a process arrives, it is allocated memory from a hole large enough to accommodate it. ○ Operating system maintains information about: a) allocated partitions b) free partitions (hole)
- A set of holes of various sizes, is scattered throughout memory at any given time. When a process arrives and needs memory, the system searches this set for a hole that is large enough for this process. If the hole is too large, it is split into two: one part is allocated to the arriving process; the other is returned to the set of holes. When a process terminates, it releases its block of memory, which is then placed back in the set of holes. If the new hold is adjacent to other holes, these adjacent holes are merged to form one larger hole.
- This procedure is a particular instance of the general dynamic storage allocation problem, which is how to satisfy a request of size n from a list of free holes. There are many solutions to this problem. The set of holes is searched to determine which hole is best to allocate. The first-fit, best-fit and worst-fit strategies are the most common ones used to select a free hole from the set of available holes.



- **First-fit:** Allocate the first hole that is big enough.
- **Best-fit:** Allocate the smallest hole that is big enough; must search entire list, unless ordered by size.
- **Worst-fit:** Allocate the largest hole; must also search entire list.

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous.
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.
- Reduce external fragmentation by compaction ○ Shuffle memory contents to place all free memory together in one large block.

Non contiguous allocation

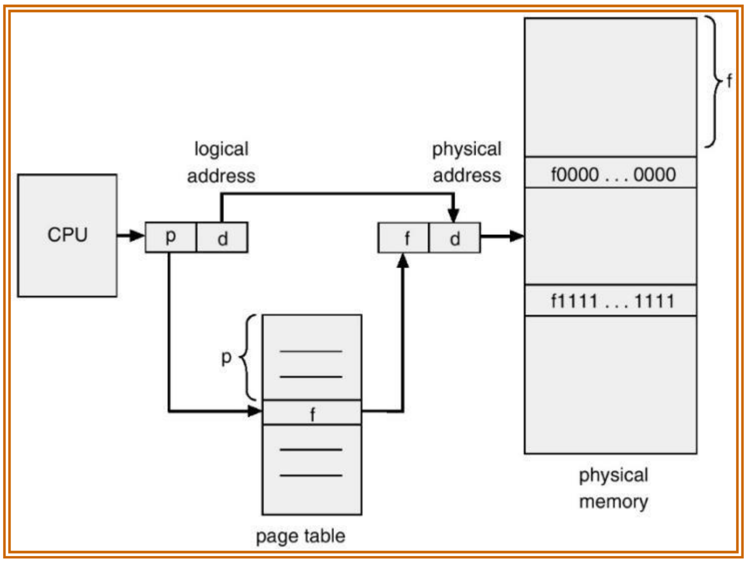
In non-contiguous allocation, the Operating system needs to maintain the table which is called the Page Table for each process which contains the base address of each block that is acquired by the process in memory space. In non-contiguous memory allocation, different parts of a process are allocated to different places in Main Memory. Spanning is allowed which is not possible in other techniques like Dynamic or Static Contiguous memory allocation. That's why paging is needed to ensure effective memory allocation. Paging is done to remove External Fragmentation.
There are five types of Non-Contiguous Allocation of Memory in the Operating System:

1. Paging
2. Multilevel Paging
3. Inverted Paging
4. Segmentation
5. Segmented Paging

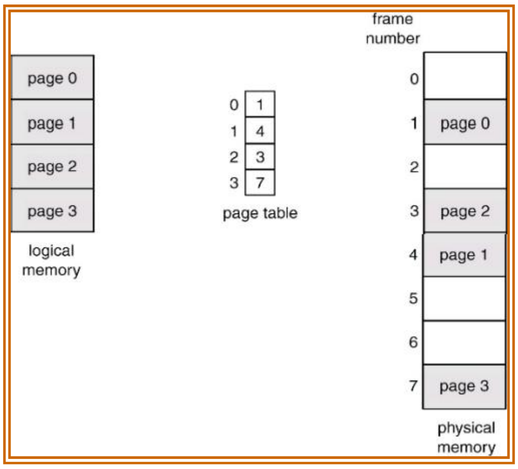
Paging

- Paging is a memory management scheme that permits the physical address space of a process to be non contiguous.
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, for example 512 bytes).

- Divide logical memory into blocks of same size called **pages**. When a process is to be executed, its pages are loaded into any available memory frames from the backing store. The backing store is divided into fixed sized blocks that are of the same size as the memory frames.
- The hardware support for paging is illustrated in below figure.
- Every address generated by the CPU is divided into two parts: a page number (p) and a page offset (d). The page number is used as an index into a page table. The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.



- The paging model of memory is shown in below figure. The page size is defined by the hardware. The size of a page is typically of a power of 2, varying between 512 bytes and 16 MB per page, depending on the computer architecture. The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy. If the size of logical address is 2^m, and a page size is 2^n addressing units, then the high order m-n bits of a logical address designate the page number, and the n low order bits designate the page offset.



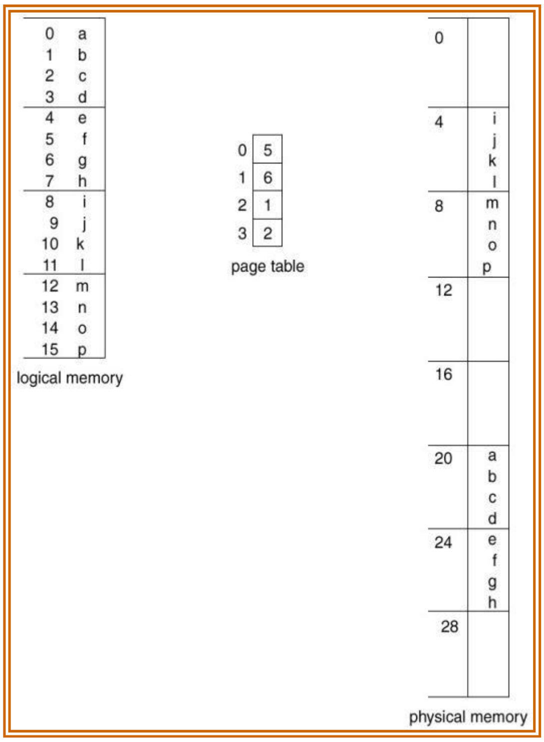
- Keep track of all free frames.
- To run a program of size n pages, need to find n free frames and load program.
- Set up a page table to translate logical to physical addresses.
- Internal fragmentation may occur.

Let us take an example. Suppose a program needs 32 KB memory for allocation. The whole program is divided into smaller units assuming 4 KB and is assigned some address. The address consists of two parts such as:

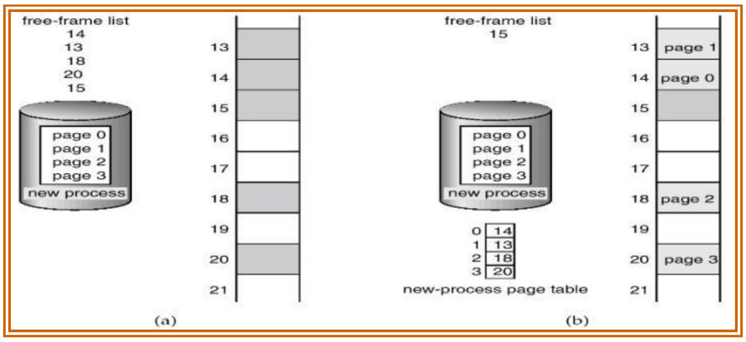
- A large number in higher order positions and
- Displacement or offset in the lower order bits.

The numbers allocated to pages are typically in power of 2 to simplify extraction of page numbers and offsets. To access a piece of data at a given address, the system first extracts the page number and the offset. Then it translates the page number to physical page frame and access data at offset in physical page frame. At this moment, the translation of the address by the OS is done using a page table. Page table is a linear array indexed by virtual page number which provides the physical page frame that contains the particular page. It employs a lookup process that extracts the page number and the offset. The system in addition checks that the page number is within the address space of process and retrieves the page number in the page table. Physical address will calculated by using the formula.

Physical address = page size of logical memory X frame number + offset

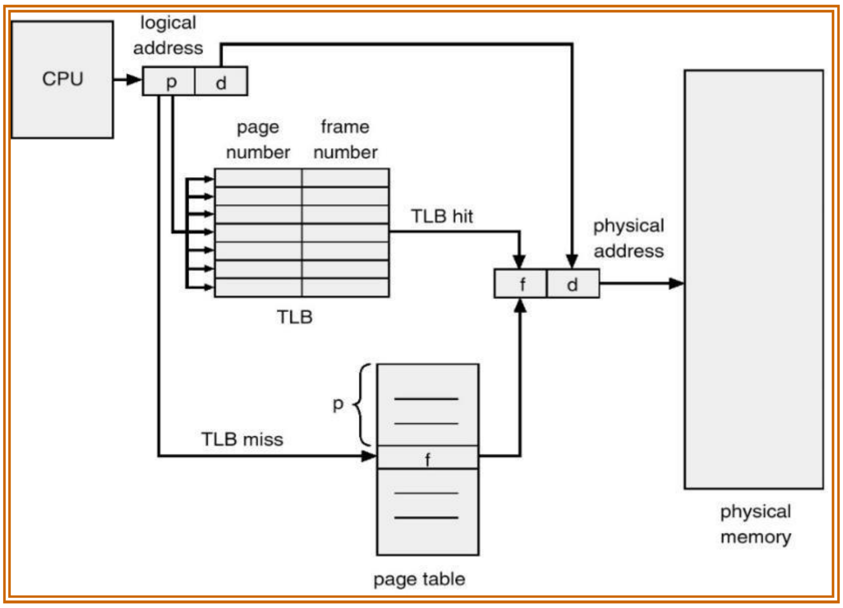


When a process arrives in the system to be executed, its size expressed in pages is examined. Each page of the process needs one frame. Thus if the process requires n pages, at least n frames must be available in memory. If n frames are available, they are allocated to this arriving process. The first page of the process is loaded into one of the allocated frames, and the frame number is put in the page table for this process. The next page is loaded into another frame, and its frame number is put into the page table and so on as in below figure. An important aspect of paging is the clear separation between the user's view of memory and the actual physical memory. The user program views that memory as one single contiguous space, containing only this one program. In fact, the user program is scattered throughout physical memory, which also holds other programs. The difference between the user's view of memory and the actual physical memory is reconciled by the address-translation hardware. The logical addresses are translated into physical addresses. This mapping is hidden from the user and is controlled by the operating system.



Implementation of Page Table

- Page table is kept in main memory.
- Page-tablebase register (PTBR) points to the page table.
- In this scheme every data/instruction-byte access requires two memory accesses. One for the page-table entry and one for the byte.
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called associative registers or associative memory or translation look-aside buffers(TLBs).
- Typically, the number of entries in a TLB is between 32 and 1024.



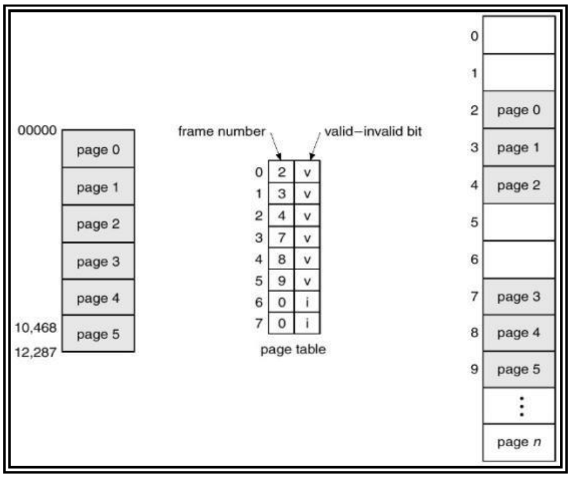
- The TLB contains only a few of the page table entries. When a logical address is generated by the CPU, its page number is presented to the TLB. If the page number is found, its frame number is immediately available and is used to access memory. The whole task may take less than 10 percent longer than it would if an unmapped memory reference were used.

- If the page number is not in the TLB (known as a TLB miss), a memory reference to the page table must be made. When the frame number is obtained, we can use it to access memory. **Hit Ratio**
- Hit Ratio: the percentage of times that a page number is found in the associative registers.
- For example, if it takes 20 nanoseconds to search the associative memory and 100 nanoseconds to access memory; for a 98-percent hit ratio, we have

  Effective memory-access time = 0.98 x 120 +
  0.02 x 220 = 122
  nanoseconds.
- The Intel 80486 CPU has 32 associative registers, and claims a 98-percent hit ratio.

Valid or invalid bit in a page table

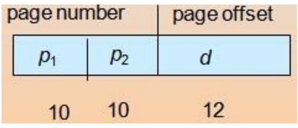
- Memory protection implemented by associating protection bit with each frame.
- Valid-invalid bit attached to each entry in the page table:
  - "Valid" indicates that the associated page is in the process' logical address space, and is thus a legal page.
  - "Invalid" indicates that the page is not in the process' logical address space.
- Pay attention to the following figure. The program extends to only address 10,468, any reference beyond that address is illegal. However, references to page 5 are classified as valid, so accesses to addresses up to 12,287 are valid. This reflects the internal fragmentation of paging.



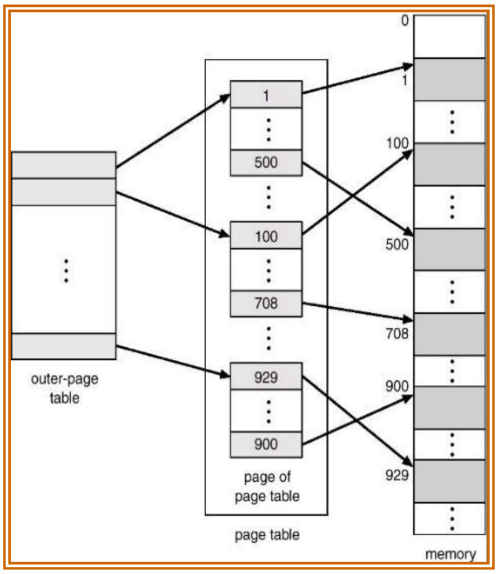
Structure of the Page Table

Hierarchical Paging:

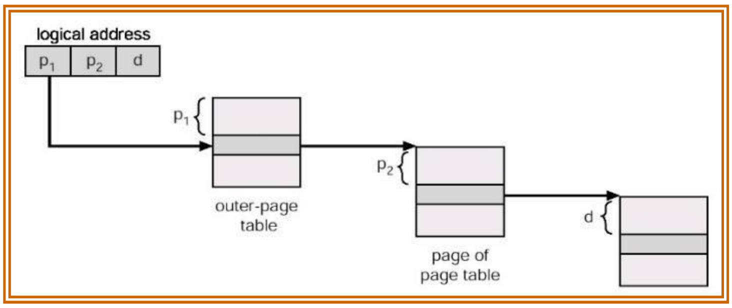
- A logical address (on 32-bit machine with 4K page size) is divided into:
  - A page number consisting of 20 bits. ○ A page offset consisting of 12 bits.
- Since the page table is paged, the page number is further divided into:
  - A 10-bit page number. ○ A 10-bit page offset.
- Thus, a logical address is as follows:



Where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the outer page table.The below figure shows a two level page table scheme.

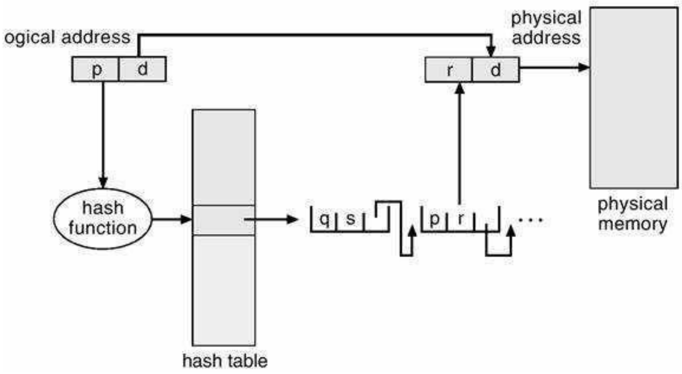


Address-translation scheme for a two-level 32-bit paging architecture is shown in below figure.



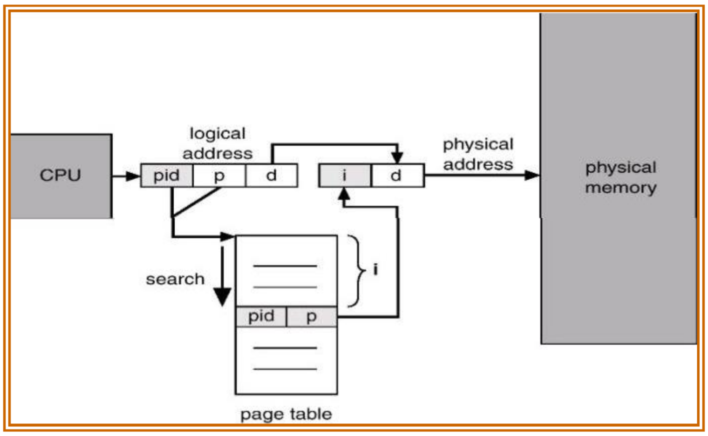
Hashed Page Table:

A common approach for handling address spaces larger than 32 bits is to use a hashed page table, with the hash value being the virtual page number. Each entry in the hash table contains a linked list of elements that has to the same location. Each element consists of three fields: (a) the virtual page number, (b) the value of the mapped page frame, and (c) a pointer to the next element in the linked list. The algorithm works as follows: The virtual page number in the virtual address is hashed into the hash table. The virtual page number is compared to field (a) in the first element in the linked list. If there is a match, the corresponding page frame (field (b)) is used to form the desired physical address. If there is no match, subsequent entries in the linked list are searched for a matching virtual page number. The scheme is shown in below figure.



Inverted Page Table:

- One entry for each real page (frame) of memory.
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.
- There is only one page table in the system. Not per process.
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.
- Use hash table to limit the search to one — or at most a few — page-table entries.



Each virtual address in the system consists of a triple <process-id, page-number, offset>. Each inverted page table entry is a pair <process-id, page-number> where the process-id assumes the role of the address space identifier. When a memory reference occurs, part of the virtual address, consisting of <process-id, page-number>, is presented to the memory subsystem. The inverted page table is then searched for a match. If a match is found say at entry i, then the physical address

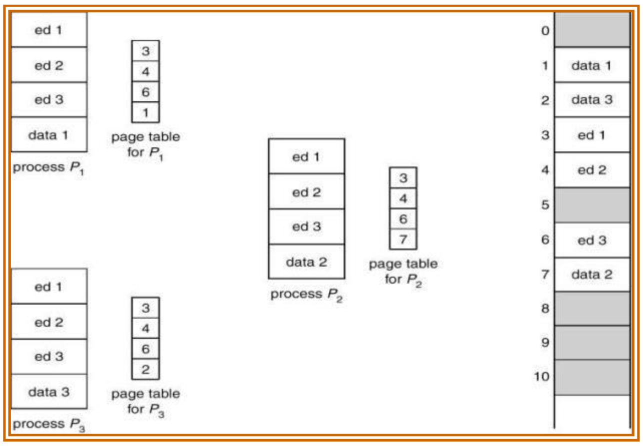
<i, offset> is generated. If no match is found, then an illegal address access has been attempted.

Shared Page:

- Shared code
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
  - Shared code must appear in same location in the logical address space of all processes.

- Private code and data ○ Each process keeps a separate copy of the code and data. ○ The pages for the private cod**e** and data can appear anywhere in the logical address space.

Reentrant code or pure code is non self modifying code. If the code is reentrant, then it never changes during execution. Thus, two or more processes can execute the same code at the same time. Each process has its own copy of registers and data storage to hold the data for the process' execution. The data for two different processes will of course vary for each process.



Segmentation

- Memory-management scheme that supports user view of memory.
- A program is a collection of segments. A segment is a logical unit such as:

  Main program,

Procedure,

Function,

Method,

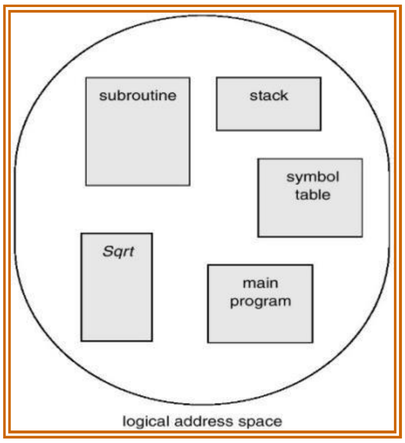
Object,

Local variables, global variables,

Common block,

Stack,

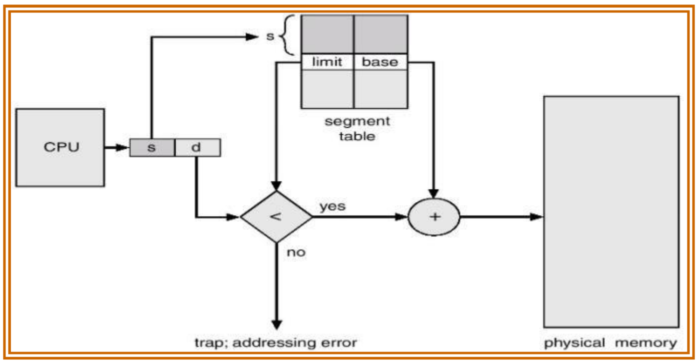
Symbol table, arrays



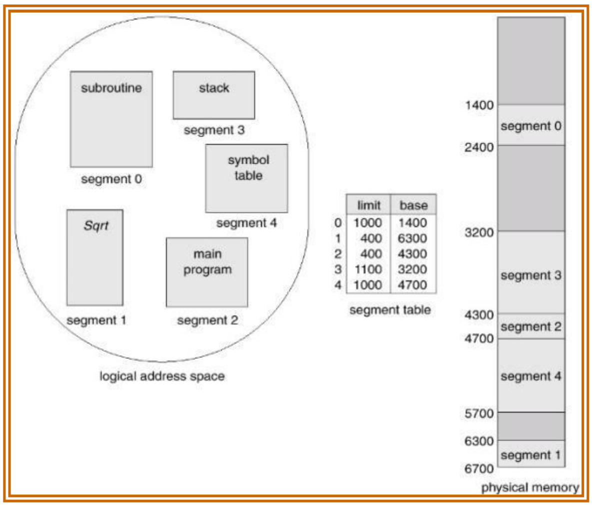
- Segmentation is a memory management scheme that supports this user view of memory.
- A logical address space is a collection of segments. Each segment has a name and a length.
- The addresses specify both the segment name and the offset within the segment.
- The user therefore specifies each address by two quantities such as segment name and an offset. For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name.
- Logical address consists of a two tuples:

<segment-number, offset>

- Segment table – maps two-dimensional physical addresses; each table entry has:
    - Base – contains the starting physical address where the segments reside in memory.
    - Limit – specifies the length of the segment.
- Segment-table base register (STBR) points to the segment table's location in memory.
- Segment-table length register (STLR) indicates number of segments used by a program; Segment number s is legal if s< STLR.



- When the user program is compiled by the compiler it constructs the segments.
- The loader takes all the segments and assigned the segment numbers.
- The mapping between the logical and physical address using the segmentation technique is shown in above figure.
- Each entry in the segment table as limit and base address.
- The base address contains the starting physical address of a segment where the limit address specifies the length of the segment.
- The logical address consists of 2 parts such as segment number and offset.
- The segment number is used as an index into the segment table. Consider the below example is given below.

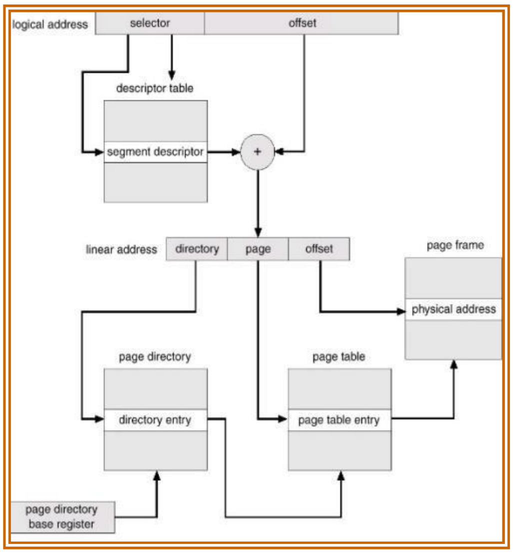


	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Segmentation with Paging

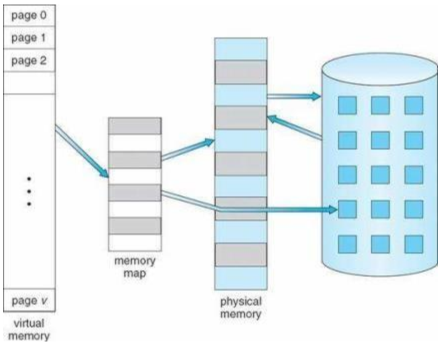
- Both paging and segmentation have advantages and disadvantages, that’s why we can combine these two methods to improve this technique for memory allocation.
- These combinations are best illustrated by architecture of Intel-386.
- The IBM OS/2 is an operating system of the Intel-386 architecture. In this technique both segment table and page table is required.
- The program consists of various segments given by the segment table where the segment table contains different entries one for each segment.
- Then each segment is divided into a number of pages of equal size whose information is maintained in a separate page table.
- If a process has four segments that is 0 to 3 then there will be 4 page tables for that process, one for each segment.
- The size fixed in segmentation table (SMT) gives the total number of pages and therefore maximum page number in that segment with starting from 0.
- If the page table or page map table for a segment has entries for page 0 to 5.
- The address of the entry in the PMT for the desired page p in a given segment s can be obtained by B + P where B can be obtained from the entry in the segmentation table.

- Using the address (B +P) as an index in page map table (page table), the page frame (f) can be obtained and physical address can be obtained by adding offset to page frame.



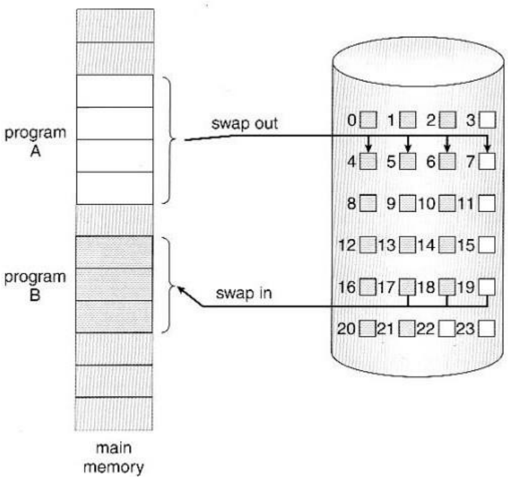
Virtual Memory

- It is a technique which allows execution of process that may not be compiled within the primary memory.
- It separates the user logical memory from the physical memory. This separation allows an extremely large memory to be provided for program when only a small physical memory is available.
- Virtual memory makes the task of programming much easier because the programmer no longer needs to working about the amount of the physical memory is available or not.
- The virtual memory allows files and memory to be shared by different processes by page sharing.
- It is most commonly implemented by demand paging.

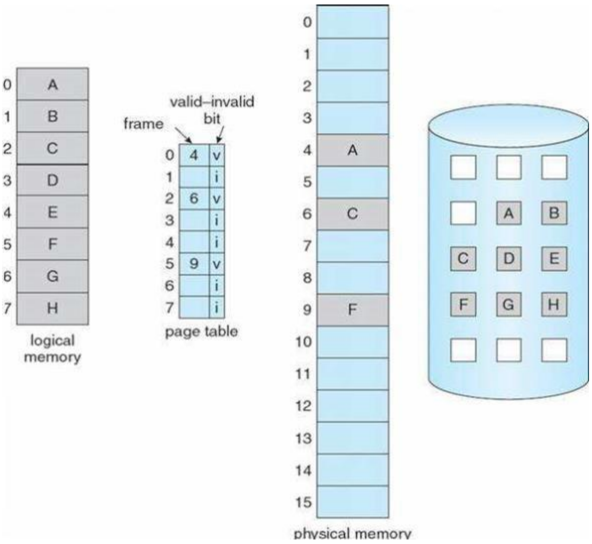


Demand Paging

A demand paging system is similar to the paging system with swapping feature. When we want to execute a process we swap it into the memory. A swapper manipulates entire process where as a pager is concerned with the individual pages of a process. The demand paging concept is using pager rather than swapper. When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again. Instead of swapping in a whole process, the pager brings only those necessary pages into memory. The transfer of a paged memory to contiguous disk space is shown in below figure.



Thus it avoids reading into memory pages that will not used any way decreasing the swap time and the amount of physical memory needed. In this technique we need some hardware support to distinct between the pages that are in memory and those that are on the disk. A valid and invalid bit is used for this purpose. When this bit is set to valid it indicates that the associate page is in memory. If the bit is set to invalid it indicates that the page is either not valid or is valid but currently not in the disk.



Marking a page invalid will have no effect if the process never attempts to access that page. So while a process executes and access pages that are memory resident, execution proceeds normally. Access to a page marked invalid causes a page fault trap. It is the result of the OS's failure to bring the desired page into memory.

Procedure to handle page fault

If a process refers to a page that is not in physical memory then

- We check an internal table (page table) for this process to determine whether the reference was valid or invalid.
- If the reference was invalid, we terminate the process, if it was valid but not yet brought in, we have to bring that from main memory.
- Now we find a free frame in memory.
- Then we read the desired page into the newly allocated frame.
- When the disk read is complete, we modify the internal table to indicate that the page is now in memory.
- We restart the instruction that was interrupted by the illegal address trap. Now the process can access the page as if it had always been in memory.

Page Replacement

- Each process is allocated frames (memory) which hold the process's pages (data)
- Frames are filled with pages as needed – this is called demand paging
- Over-allocation of memory is prevented by modifying the page-fault service routine to replace pages
- The job of the page replacement algorithm is to decide which page gets victimized to make room for a new page
- Page replacement completes separation of logical and physical memory

Page Replacement Algorithm

Optimal algorithm

- Ideally we want to select an algorithm with the lowest page-fault rate
- Such an algorithm exists, and is called (unsurprisingly) the optimal algorithm:
- Procedure: replace the page that will not be used for the longest time (or at all) – i.e. replace the page with the greatest forward distance in the reference string
- Example using 4 frames:

Reference #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Page referenced	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Frames _ = faulting page	$\underline{1}$	1	1	1	1	1	1	1	1	1	$\underline{4}$	4
		$\underline{2}$	2	2	2	2	2	2	2	2	2	2
			$\underline{3}$	3	3	3	3	3	3	3	3	3
				$\underline{4}$	4	4	$\underline{5}$	5	5	5	5	5

- Analysis: 12 page references, 6 page faults, 2 page replacements. Page faults per number of frames = 6/4 = 1.5

- Unfortunately, the optimal algorithm requires special hardware (crystal ball, magic mirror, etc.) not typically found on today's computers
- Optimal algorithm is still used as a metric for judging other page replacement algorithms

FIFO algorithm

- Replaces pages based on their order of arrival: oldest page is replaced
- Example using 4 frames:

Reference #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Page referenced	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Frames	1̲	1	1	1	1	1	5̲	5	5	5	4̲	4
_ = faulting		2̲	2	2	2	2	2	1̲	1	1	1	5̲
page			3̲	3	3	3	3	3	2̲	2	2	2
				4̲	4	4	4	4	4	3̲	3	3

- Analysis: 12 page references, 10 page faults, 6 page replacements. Page faults per number of frames = 10/4 = 2.5

LFU algorithm (page-based)

- procedure: replace the page which has been referenced least often
- For each page in the reference string, we need to keep a reference count. All reference counts start at 0 and are incremented every time a page is referenced.
- example using 4 frames:

Reference #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Page referenced	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**

Frames	${}^{1}\underline{1}$	${}^{1}1$	${}^{1}1$	${}^{1}1$	${}^{2}1$	${}^{2}1$	${}^{2}1$	${}^{3}1$	${}^{3}1$	${}^{3}1$	${}^{3}1$	${}^{3}1$
_ = faulting		${}^{1}\underline{2}$	${}^{1}2$	${}^{1}2$	${}^{1}2$	${}^{2}2$	${}^{2}2$	${}^{2}2$	${}^{3}2$	${}^{3}2$	${}^{3}2$	${}^{3}2$
page ^{n} =			${}^{1}\underline{3}$	${}^{1}3$	${}^{1}3$	${}^{1}3$	${}^{1}\underline{5}$	${}^{1}5$	${}^{1}5$	${}^{2}\underline{3}$	${}^{2}3$	${}^{2}\underline{5}$
reference count				${}^{1}\underline{4}$	${}^{1}4$	${}^{1}4$	${}^{1}4$	${}^{1}4$	${}^{1}4$	${}^{1}4$	${}^{2}4$	${}^{2}4$

- At the 7th page in the reference string, we need to select a page to be victimized. Either 3 or 4 will do since they have the same reference count (1). Let's pick 3.
- Likewise at the 10th page reference; pages 4 and 5 have been referenced once each. Let's pick page 4 to victimize. Page 3 is brought in, and its reference count (which was 1 before we paged it out a while ago) is updated to 2.
- Analysis: 12 page references, 7 page faults, 3 page replacements. Page faults per number of frames = 7/4 = 1.75

LFU algorithm (frame-based)

- Procedure: replace the page in the frame which has been referenced least often
- Need to keep a reference count for each frame which is initialized to 1 when the page is paged in, incremented every time the page in the frame is referenced, and reset every time the page in the frame is replaced
- Example using 4 frames:

Reference #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Page referenced	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
	${}^{1}\underline{1}$	${}^{1}1$	${}^{1}1$	${}^{1}1$	${}^{2}1$	${}^{2}1$	${}^{2}1$	${}^{3}1$	${}^{3}1$	${}^{3}1$	${}^{3}1$	${}^{3}1$

Frames		$^{1}\underline{2}$	$^{1}2$	$^{1}2$	$^{1}2$	$^{2}2$	$^{2}2$	$^{2}2$	$^{3}2$	$^{3}2$	$^{3}2$	$^{3}2$
_ = faulting			$^{1}\underline{3}$	$^{1}3$	$^{1}3$	$^{1}3$	$^{1}\underline{5}$	$^{1}5$	$^{1}5$	$^{1}\underline{3}$	$^{1}3$	$^{1}\underline{5}$
page ^{n} = reference count				$^{1}\underline{4}$	$^{1}4$	$^{1}4$	$^{1}4$	$^{1}4$	$^{1}4$	$^{1}4$	$^{2}4$	$^{2}4$

- At the 7th reference, we victimize the page in the frame which has been referenced least often -- in this case, pages 3 and 4 (contained within frames 3 and 4) are candidates, each with a reference count of 1. Let's pick the page in frame 3. Page 5 is paged in and frame 3's reference count is reset to 1.
- At the 10th reference, we again have a page fault. Pages 5 and 4 (contained within frames 3 and 4) are candidates, each with a count of 1. Let's pick page 4. Page 3 is paged into frame 3, and frame 3's reference count is reset to 1.
- Analysis: 12 page references, 7 page faults, 3 page replacements. Page faults per number of frames = 7/4 = 1.75

LRU algorithm

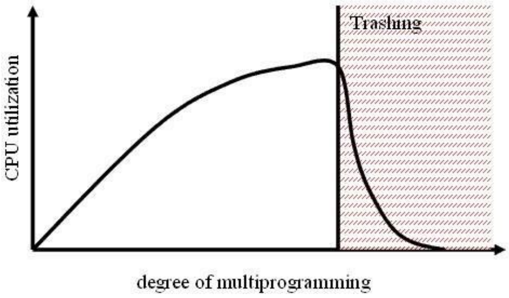
- Replaces pages based on their most recent reference – replace the page with the greatest backward distance in the reference string
- Example using 4 frames:

Reference #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Page referenced	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Frames	$\underline{1}$	1	1	1	1	1	1	1	1	1	1	$\underline{5}$
_ = faulting		$\underline{2}$	2	2	2	2	2	2	2	2	2	2
page			$\underline{3}$	3	3	3	$\underline{5}$	5	5	5	$\underline{4}$	4
				$\underline{4}$	4	4	4	4	4	$\underline{3}$	3	3

- Analysis: 12 page references, 8 page faults, 4 page replacements. Page faults per number of frames = 8/4 = 2
- One possible implementation (not necessarily the best):
  - Every frame has a time field; every time a page is referenced, copy the current time into its frame's time field
  - When a page needs to be replaced, look at the time stamps to find the oldest

Thrashing

- If a process does not have "enough" pages, the page-fault rate is very high
  - low CPU utilization
  - OS thinks it needs increased multiprogramming
  - adds another process to system
- Thrashing is when a process is busy swapping pages in and out
- Thrashing results in severe performance problems. Consider the following scenario, which is based on the actual behaviour of early paging systems. The operating system monitors CPU utilization. If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new process to the system. A global page replacement algorithm is used; it replaces pages with no regard to the process to which they belong. Now suppose that a process enters a new phase in its execution and needs more frames.


Trashing
CPU utilization
degree of multiprogramming

MODULE:IV

FILE SYSTEM

File concept:

A file is a collection of related information that is stored on secondary storage. Information stored in files must be persistent i.e. not affected by power failures & system reboots.

Files may be of free from such as text files or may be formatted rigidly. Files represent both programs as well as data.

Part of the OS dealing with the files is known as file system. The important file concepts include:

1. **File attributes:** A file has certain attributes which vary from one operating system to another.
   - **Name:** Every file has a name by which it is referred.
   - **Identifier:** It is unique number that identifies the file within the file system.
   - **Type:** This information is needed for those systems that support different types of files.
   - **Location:** It is a pointer to a device & to the location of the file on that device
   - **Size:** It is the current size of a file in bytes, words or blocks.
   - **Protection:** It is the access control information that determines who can read, write & execute a file.
   - **Time, date & user identification:** It gives information about time of creation or last modification & last use.
2. **File operations:** The operating system can provide system calls to create, read, write, reposition, delete and truncate files.

- **Creating files:** Two steps are necessary to create a file. First, space must be found for the file in the file system. Secondly, an entry must be made in the directory for the new file.
- **Reading a file:** Data & read from the file at the current position. The system must keep a read pointer to know the location in the file from where the next read is to take place. Once the read has been taken place, the read pointer is updated.
- **Writing a file:** Data are written to the file at the current position. The system must keep a write pointer to know the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.
- **Repositioning within a file (seek):** The directory is searched for the appropriate entry & the current file position is set to a given value. After repositioning data can be read from or written into that position.
- **Deleting a file:** To delete a file, we search the directory for the required file. After deletion, the space is releasedso that it can be reused by other files.
- **Truncating a file:** The user may erase the contents of a file but allows all attributes to remain unchanged expect the file length which is rest to ‘O’ & the space is released.

3. **File types:** The file name is spilt into 2 parts, Name & extension. Usually these two parts areseparated by a period. The user & the OS can know the type of the file from the extension itself.

4.

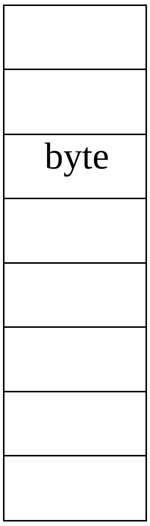
Listed below are some file types along with their extension:

File Type	Extension
Executable File	exe, bin, com
Object File	obj, o (compiled)
Source Code file	C, C++, Java, pas

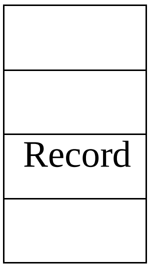
Batch File	bat, sh (commands to command the interpreter)
Text File	txt, doc (textual data documents)
Archieve File	arc, zip, tar (related files grouped together into file compressed for storage)
Multimedia File	mpeg (Binary file containing audio or A/V information)

5. **File structure:** Files can be structured in several ways. Three common possible are:

- **Byte sequence:**The figure shows an unstructured sequence of bytes. The OS doesn't care about the content of file.
- It only sees the bytes. This structure provides maximum flexibility. Users can write anything into their files & name them according to their convenience. Both UNIX & windows use this approach.



- **Record sequence:** In this structure, a file is a sequence of fixed length records. Here the read operation returns one records & the write operation overwrites or append or record.



- **Tree:**In this organization, a file consists of a tree of records of varying lengths. Each record consists of a key field. The tree is stored on the key field to allow first searching for a particular key.

Access methods: Basically, access method is divided into 2 types:

- **Sequential access:** It is the simplest access method. Information in the file is processed in order i.e. one record after another.
- A process can read all the data in a file in order starting from beginning but can’t skip & read arbitrarily from any location.
- Sequential files can be rewound.
- It is convenient when storage medium was magnetic tape rather than disk.
- **Direct access:** A file is made up of fixed length-logical records that allow programs to read & write records rapidly in no particular O order.
-
- This method can be used when disk are used for storing files.
- This method is used in many applications e.g. database systems. If an airline customer wants to reserve a seat on a particular flight, the reservation program must be able to access the record for that flight directly without reading the records before it.
- In a direct access file, there is no restriction in the order of reading or writing. For example, we can read block 14, then read block 50 & then write block 7 etc.
- Direct access files are very useful for immediate access to large amount of information.

Directory structure:

The file system of computers can be extensive. Some systems store thousands of file on disk. To manage all these data, we need to organize them. The organization is done in 2 steps. The file system is broken into partitions. Each partition contains information about file within it.

Operation on a directory:

- **Search for a file:** We need to be able to search a directory for a particular file.
- **Create a file:** New files are created & added to the directory.
- **Delete a file:** When a file is no longer needed, we may remove it from the directory.
- **List a directory:** We should be able to list the files of the directory.
- **Rename a file:** The name of a file is changed when the contents of the file changes.
- **Traverse the file system:** It is useful to be able to access every directory & every file within a directory.

Structure of a directory:

The most common schemes for defining the structure of the directory are:

1. **Single level directory:** It is the simplest directory structure. All files are present in the same directory. So it is easy to manage & understand.
   Limitation: A single level directory is difficult to manage when the no. of files increases or when there is more than one user. Since all files are in same directory, they must have unique names. So, there is confusion of file names between different users.
2. **Two level directories:** The solution to the name collision problem in single level directory is to create a separate directory for each user. In a two level directory structure, each user has its own user file directory. When a user logs

in, then master file directory is searched. It is indexed by user name & each entry points to the UFD of that user.

Limitation: It solves name collision problem. But it isolates one user from another. It is an advantage when users are completely independent.

But it is a disadvantage when the users need to access each other’s files & co-operate among themselves on a particular task.

3. **Tree structured directories:** It is the most common directory structure. A two level directory is a two level tree. So, the generalization is to extend the directory structure to a tree of arbitrary height.

   It allows users to create their own subdirectories & organize their files. Every file in the system has a unique path name.

   It is the path from the root through all the sub-directories to a specified file. A directory is simply another file but it is treated in a special way.

   One bit in each directory entry defines the entry as a file (O) or as sub-directories.

   Each user has a current directory. It contains most of the files that are of current interest to the user.

   Path names can be of two types: An absolute path name begins from the root directory & follows the path down to the specified files. A relative path name defines the path from the current directory.

   E.g. If the current directory is root/spell/mail, then the relative path name is prt/first & the absolute path name is root/ spell/ mail/ prt/ first.

   Here users can access the files of other users also by specifying their path names.

4. **A cyclic graph directory:**It is a generalization of tree structured directory scheme. An a cyclic graph allows directories to have shared sub-directories & files. A shared directory or file is not the same as two copies of a file.

5. Here a programmer can view the copy but the changes made in the file by one programmer are not reflected in the other’s copy. But in a shared file, there is

only one actual file. So many changes made by a person would be immediately visible to others.

This scheme is useful in a situation where several people are working as a team. So, here all the files that are to be shared are put together in one directory. Shared files and sub-directories can be implemented in several ways.

A common way used in UNIX systems is to create a new directory entry called link. It is a pointer to another file or sub-directory.

The other approach is to duplicate all information in both sharing directories. A cyclic graph structure is more flexible then a tree structure but it is also more complex.

Limitation: Now a file may have multiple absolute path names. So, distinct file names may refer to the same file. Another problem occurs during deletion of a shared file. When a file is removed by any one user.

It may leave dangling pointer to the non existing file. One serious problem in a cyclic graph structure is ensuring that there are no cycles. To avoid these problems, some systems do not allow shared directories or files.

E.g. MS-DOS uses a tree structure rather than a cyclic to avoid the problems associated with deletion.

One approach for deletion is to preserve the file until all references to it are deleted. To implement this approach, we must have some mechanism for determining the last reference to the file.

For this we have to keep a list of reference to a file. But due to the large size of the no. of references. When the count is zero, the file can be deleted.

6. **General graph directory:** When links are added to an existing tree structured directory, the tree structure is destroyed, resulting in a simple graph structure. Linking is a technique that allows a file to appear in more than one directory. The advantage is the simplicity of algorithm to transverse the graph & determines when there are no more references to a file. But a similar

problem exists when we are trying to determine when a file can be deleted. Here also a value zero in the reference count means that there are no more references to the file or directory & the file can be deleted.

But when cycle exists, the reference count may be non-zero even when there are no references to the directory or file.

This occurs due to the possibility of self referencing (cycle) in the structure. So, here we have to use garbage collection scheme to determine when the last references to a file has been deleted & the space can be reallocated.

It involves two steps:

- Transverse the entire file system & mark everything that can be accessed.
- Everything that isn't marked is added to the list of free space.

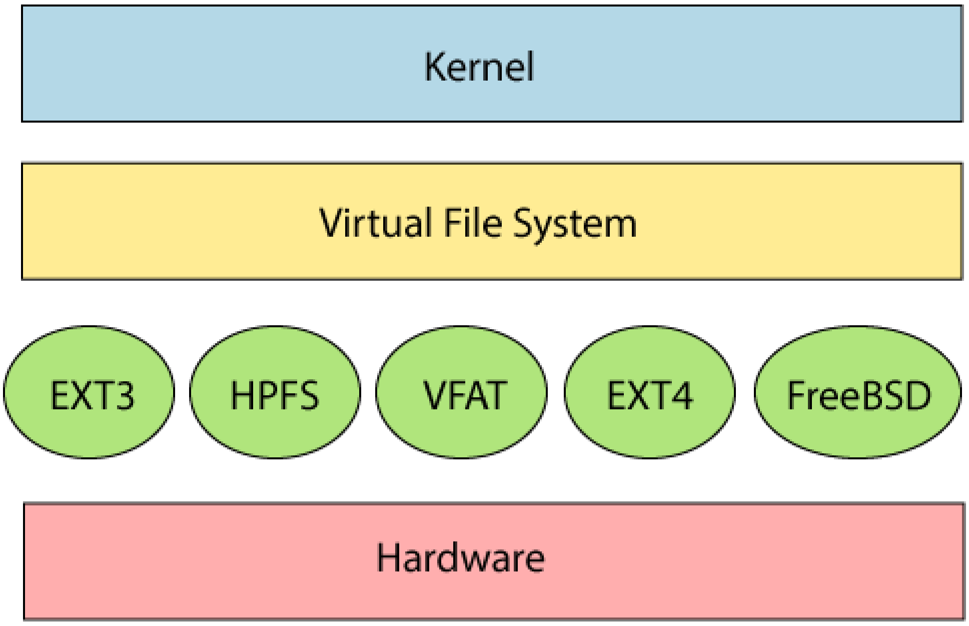
But this process is extremely time consuming. It is only necessary due to presence of cycles in the graph. So, a cyclic graph structure is easier to work than this.

Case study

Linux File System

- Linux file system has a hierarchal file structure as it contains a root directory and its subdirectories.
- All other directories can be accessed from the root directory.
- A partition usually has only one file system, but it may have more than one file system.
- A file system is designed in a way so that it can manage and provide space for non-volatile storage data.
- All file systems required a namespace that is a naming and organizational methodology.

- The namespace defines the naming process, length of the file name, or a subset of characters that can be used for the file name.
- It also defines the logical structure of files on a memory segment, such as the use of directories for organizing the specific files.
- The data structure needs to support a hierarchical directory structure; this structure is used to describe the available and used disk space for a particular block.
- It also has the other details about the files such as file size, date & time of creation, update, and last modified.
- Also, it stores advanced information about the section of the disk, such as partitions and volumes.
- The advanced data and the structures that it represents contain the information about the file system stored on the drive;
- it is distinct and independent of the file system metadata.
- Linux file system contains two-part file system software implementation architecture. Consider the below image:



- The file system requires an API (Application programming interface) to access the function calls to interact with file system components like files and directories.
- The first two parts of the given file system together called a **Linux virtual file system**.
- It provides a single set of commands for the kernel and developers to access the file system.
- This virtual file system requires the specific system driver to give an interface to the file system.

Linux File System Features

- In Linux, the file system creates a tree structure. All the files are arranged as a tree and its branches. The topmost directory called the **root (/) directory**.
- All other directories in Linux can be accessed from the root directory.

Some key features of Linux file system are as following:

  - **Specifying paths:** Linux does not use the backslash (\) to separate the components; it uses forward slash (/) as an alternative.
  - For example, as in Windows, the data may be stored in C:\ My Documents\ Work, whereas, in Linux, it would be stored in /home/ My Document/ Work.
  - **Partition, Directories, and Drives:** Linux does not use drive letters to organize the drive as Windows does.
  - In Linux, we cannot tell whether we are addressing a partition, a network device, or an "ordinary" directory and a Drive.
  - **Case Sensitivity:** Linux file system is case sensitive. It distinguishes between lowercase and uppercase file names. Such as, there is a difference between test.txt and Test.txt in Linux.
  - This rule is also applied for directories and Linux commands.

- **File Extensions:** In Linux, a file may have the extension '.txt,' but it is not necessary that a file should have a file extension. While working with Shell, it creates some problems for the beginners to differentiate between files and directories.
- If we use the graphical file manager, it symbolizes the files and folders.
- **Hidden files:** Linux distinguishes between standard files and hidden files, mostly the configuration files are hidden in Linux OS. Usually.
- we don't need to access or read the hidden files. The hidden files in Linux are represented by a dot (.) before the file name (e.g., .ignore).

Mass Storage Structure

- Systems designed to store enormous volumes of data are referred to as mass storage devices.
- Massive storage devices are sometimes used interchangeably with peripheral storage, which is the management of bigger volumes of data that are larger than the native storage capability of a computer or device.
- The basic idea of Mass Storage is to create a Data Backup or Data Recovery System.

- Along with computer systems, definitions of mass storage technologies and tactics have changed.
- The earliest and most basic mass storage techniques date back to the era of main frame supercomputers, according to experts.
- Today, mass storage may include several kinds of hard disks or solid-state storage devices, as well as tape drives and other physical data storage devices.
- The concepts of data backup and data recovery are frequently linked to mass storage media.

The Mass Storage Structure Devices are:

1. Magnetic Disks
2. Solid State Disks
3. Magnetic Tapes

Magnetic Disks

- Now, we are going to know about all whereabouts of the Magnetic Disk Mass Storage Structure Devices.
- In 1956, IBM created the first magnetic hard drive, a substantial device with 50 21-inch (53-cm) platters.

- Despite being large, it could only hold 5 megabytes of information.
- Since then, magnetic disks' storage capabilities have multiplied dramatically while simultaneously shrinking in size.

Basic Common Examples of Magnetic Disks are:

1. Floppy Disks
2. Hard Disks
3. Zip Disks

The Magnetic Disk basically looks like:

Structure and Working of Magnetic Disks:

The basic structure of Magnetic Disks is:

- A mechanical arm that travels across a revolving magnetic surface, known as the platter, makes up the majority of a magnetic disk.
- They come together to make a "comb." Both reading from and writing to the disk are done using the mechanical arm. A magnetization process is used to read and write data on magnetic disks.
- One or more disk-shaped platters with magnetic material covering them. Unlike "floppy" disks, which are composed of more flexible plastic, hard disk platters are built of stiff metal.
- There are two work areas on each plate. The very top and bottom surfaces of a stack of platters were occasionally avoided by older hard disk drives because they are more prone to damage or even breaking in some cases.
- The time it takes for the requested sector to spin and enter the read-write head is known as the **rotational latency**.
- This can be anything from 0 and 1 complete revolutions, with an average of 12 revolutions.
- This is a physical action that often follows seek time as the second-slowest step. (If a disk rotates at 7200 revolutions per minute, the average rotational delay is 1/2 revolution /

120 revolutions per second, or just over 4 milliseconds, a long time by computer standards.

Solid State Disks

- Old technologies are frequently employed in new ways as economic conditions and technology evolve.
- The growing usage of solid state drives, or SSDs, is one illustration of this.
- SSDs function as a tiny, quick hard disk using memory technology.
- To maintain the information over power cycles, certain implementations may employ either flash memory or DRAM chips protected by a battery.
- SSDs do have certain drawbacks, too, including the fact that they cost more than hard drives, are often smaller, and may have shorter life spans.
- A boot drive is another version that has the OS and certain application executables but no essential user data.
- In order to make laptops thinner, lighter, and quicker, SSDs are also employed in them.
- The throughput of the bus may become a limiting problem due to how much quicker SSDs are than conventional hard

drives, which leads to certain SSDs being linked directly to the system PCI bus.

Magnetic Tapes

- Prior to the advent of hard disk drives, magnetic tapes were frequently utilized for secondary storage; today, they are mostly used for backups.
- It might take a while to get to a specific location on a magnetic tape, but once reading or writing starts, access rates are on par with disk drives.

Disk scheduling

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk.

Disk scheduling is also known as I/O scheduling.

Disk scheduling is important because:

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller.
- Thus other I/O requests need to wait in the waiting queue and need to be scheduled.

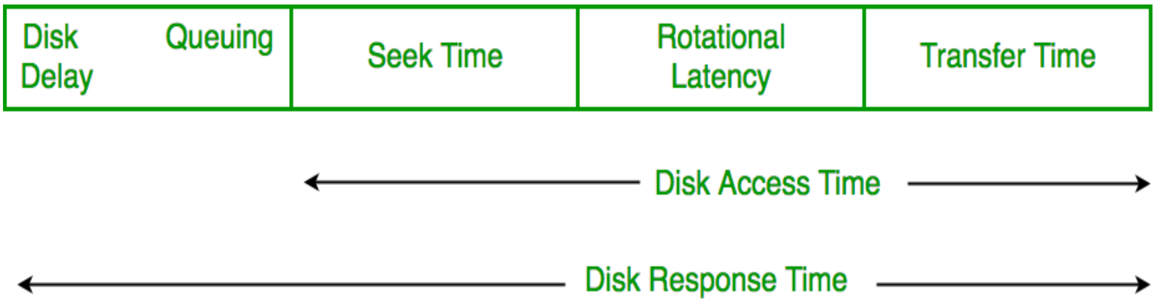
- Two or more request may be far from each other so can result in greater disk arm movement.
- Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

There are many Disk Scheduling Algorithms but before discussing them let’s have a quick look at some of the important terms:

- **Seek Time:**Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write.
- So the disk scheduling algorithm that gives minimum average seek time is better.
- **Rotational Latency:** Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads.
- So the disk scheduling algorithm that gives minimum rotational latency is better.
- **Transfer Time:** Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.
- **Disk Access Time:** Disk Access Time is:

```
Disk Access Time = Seek Time +
                              Rotational Latency +

                              Transfer Time
```



Disk Management

The operating system is responsible for various operations of disk management.

Modern operating systems are constantly growing their range of services and add-ons, and all operating systems implement four essential operating system administration functions. These functions are as follows:

1. **Process Management**
2. **Memory Management**
3. **File and Disk Management**
4. **I/O System Management**

- Most systems include secondary storage devices (magnetic disks). It is a low-cost, non-volatile storage method for data and programs.
- The user data and programs are stored on different storage devices known as files.
- The OS is responsible for allocating space to files on secondary storage devices as required.
- The OS requires tracking the position of the disk drive for each section of every file on the disk.
- It may include tracking many files and file segments on a physical disk drive in some circumstances.
- Furthermore, the OS must be able to identify each file and conduct read and writes operations on it according to the requirements.
- As a result, the OS is mainly responsible for setting the file system, assuring the security and reliability of reading and writing activities to secondary storage, and keeping access times consistent.

Disk Management of the OS includes the various aspects, such as:

1. Disk Formatting

- A new magnetic disk is mainly a blank slate.
- It is platters of the magnetic recording material. Before a disk may hold data, it must be partitioned into sectors that may be

read and written by the disk controller. It is known as **physical formatting** and **low-level formatting.**

- **Low-level formatting** creates a unique data structure for every sector on the drive.
- A data structure for a sector is made up of a header, a data region, and a trailer.
- The disk controller uses the header and trailer to store information like an error-correcting code (ECC) and a sector number.
- The OS may treat every partition as it were a separate disk. For example, one partition could contain a copy of the OS executable code, while another could contain user files.
- The second stage after partitioning is **logical formatting.**

2. Boot Block

- When a system is turned on or restarted, it must execute an initial program.
- The start program of the system is called the bootstrap program. It starts the OS after initializing all components of the system.
- The bootstrap program works by looking for the OS kernel on disk, loading it into memory, and jumping to an initial address to start the OS execution.
- The bootstrap is usually kept in read-only memory on most computer systems.

- It is useful since read-only memory does not require initialization and is at a fixed location where the CPU may begin executing whether powered on or reset.
- As a result, most computer systems include small bootstrap loader software in the boot .
- The bootstrap program is stored in a partition and is referred to as the **boot block.** A **boot disk** or **system disk** is a type of disk that contains a boot partition.

3. Bad Blocks

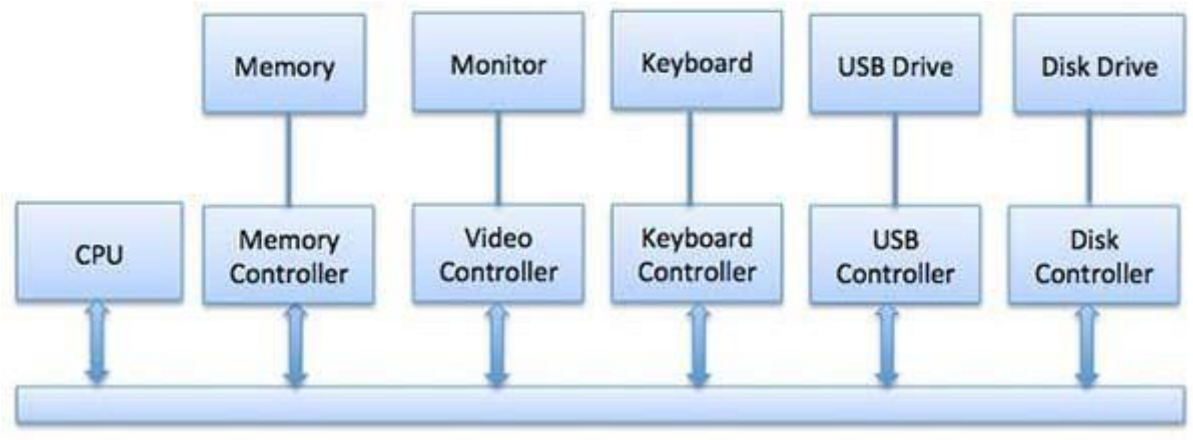
- Disks are prone to failure due to their moving parts and tight tolerances.
- When a disk drive fails, it must be replaced and the contents transferred to the replacement disk using backup media.
- For some time, one or more sectors become faulty. Most disks also come from the company with bad blocks.
- These blocks are handled in various ways, depending on the use of disk and controller.

Input output system

- One of the important jobs of an Operating System is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio I/O, printers etc.
- An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application. I/O devices can be divided into two categories -

.

- Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller.
- Following is a model for connecting the CPU, memory, controllers, and I/O devices where CPU and device controllers all use a common bus for communication.



- The CPU must have a way to pass information to and from an I/O device.
- There are three approaches available to communicate with the CPU and Device.

  - Special Instruction I/O
  - Memory-mapped I/O
  - Direct memory access (DMA)

Special Instruction I/O

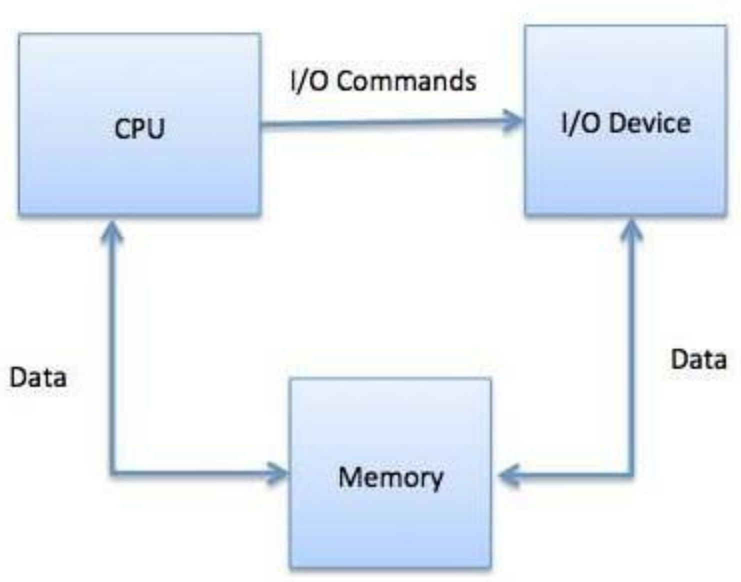
This uses CPU instructions that are specifically made for controlling I/O devices.

These instructions typically allow data to be sent to an I/O device or read from an I/O device.

Memory-mapped I/O

When using memory-mapped I/O, the same address space is shared by memory and I/O devices.

The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.



- While using memory mapped IO, OS allocates buffer in memory and informs I/O device to use that buffer to send data to the CPU. I/O device operates asynchronously with CPU, interrupts CPU when finished.
- The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device.

Protection and security

- 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.

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.
- There are many protection mechanism.
- each of them mechanism have different advantages and disadvantages and must be appropriate for the intended application.

Security

- File security is all about safeguarding your business-critical information from prying eyes by implementing stringent access control measures and flawless permission hygiene.
- Apart from enabling and monitoring security access controls, decluttering data storage also plays an important role in securing files.
- Regularly optimize file storage by purging old, stale, and other junk files to focus on business-critical files.
- Tackle data security threats and storage inefficiencies with periodic reviews and enhancements to your file security strategy.

Why is file security important?

- ***To protect sensitive data***

Personally identifiable information (PII), electronic personal health information (ePHI), confidential contracts, and other business-critical data must be stored safely.

Careless transmission or use of such files could lead to data privacy violations, resulting in heavy fines for the organization.

- ***To secure file sharing***

Files transferred through unsecured channels can be misused by insiders or hackers for malicious activities.
Comprehensive data leak prevention software can help prevent unauthorized movement of business-critical data out of the organization.

- ***To avoid data breaches***

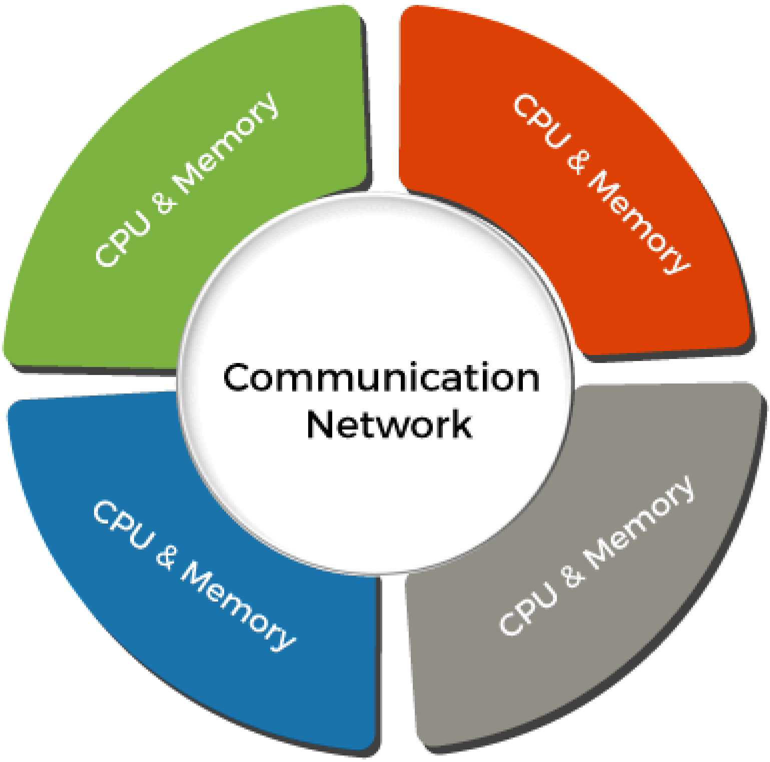
In 2019, personal details of 10.6 million MGM resort guests were breached.
The impact of such a breach can be fatal to any organization. It is not just the fines and legal consequences, but also the loss of trust that can destroy a business.

MODULE:V

DISTRIBUTED SYSTEM

Distributed Operating System

- A distributed operating system **(DOS)** is an essential type of operating system.
- Distributed systems use many central processors to serve multiple real-time applications and users.
- As a result, data processing jobs are distributed between the processors.
- It connects multiple computers via a single communication channel. Furthermore, each of these systems has its own processor and memory.
- Additionally, these **CPUs** communicate via high-speed buses or telephone lines.
- Individual systems that communicate via a single channel are regarded as a single entity.
- They're also known as **loosely coupled systems**.



- This operating system consists of numerous computers, nodes, and sites joined together via **LAN/WAN** lines.
- It enables the distribution of full systems on a couple of center processors, and it supports many real-time products and different users.
- Distributed operating systems can share their computing resources and I/O files while providing users with virtual machine abstraction.

Types of Distributed Operating System

There are various types of Distributed Operating systems. Some of them are as follows:

1. **Client-Server Systems**
2. **Peer-to-Peer Systems**

3. **Middleware**

4. **Three-tier**

5. **N-tier**

Client-Server System

- This type of system requires the client to request a resource, after which the server gives the requested resource.
- When a client connects to a server, the server may serve multiple clients at the same time.
- Client-Server Systems are also referred to as "Tightly Coupled Operating Systems".
- This system is primarily intended for multiprocessors and homogenous multicomputer.
- Client-Server Systems function as a centralized server since they approve all requests issued by client systems.

Server systems can be divided into two parts:

1. Computer Server System

This system allows the interface, and the client then sends its own requests to be executed as an action.

After completing the activity, it sends a back response and transfers the result to the client.

2. File Server System

It provides a file system interface for clients, allowing them to execute actions like file creation, updating, deletion, and more.

Peer-to-Peer System

- The nodes play an important role in this system. The task is evenly distributed among the nodes.
- Additionally, these nodes can share data and resources as needed. Once again, they require a network to connect.
- The Peer-to-Peer System is known as a "Loosely Couple System".
- This concept is used in computer network applications since they contain a large number of processors that do not share memory or clocks.
- Each processor has its own local memory, and they interact with one another via a variety of communication methods like telephone lines or high-speed buses.

Middleware

- Middleware enables the interoperability of all applications running on different operating systems.
- Those programs are capable of transferring all data to one other by using these services.

Three-tier

- The information about the client is saved in the intermediate tier rather than in the client, which simplifies development.
- This type of architecture is most commonly used in online applications.

N-tier

- When a server or application has to transmit requests to other enterprise services on the network, n-tier systems are used.

Features of Distributed Operating System

There are various features of the distributed operating system. Some of them are as follows:

Openness

It means that the system's services are freely displayed through interfaces. Furthermore, these interfaces only give the service syntax.

- For example, the type of function, its return type, parameters, and so on. Interface Definition Languages are used to create these interfaces (IDL).

Scalability

It refers to the fact that the system's efficiency should not vary as new nodes are added to the system.

Furthermore, the performance of a system with 100 nodes should be the same as that of a system with 1000 nodes.

Resource Sharing

Its most essential feature is that it allows users to share resources.

They can also share resources in a secure and controlled manner.

Printers, files, data, storage, web pages, etc., are examples of shared resources.

Flexibility

A DOS's flexibility is enhanced by modular qualities and delivers a more advanced range of high-level services.

The kernel/ microkernel's quality and completeness simplify the implementation of such services.

Transparency

It is the most important feature of the distributed operating system.

The primary purpose of a distributed operating system is to hide the fact that resources are shared.

Transparency also implies that the user should be unaware that the resources he is accessing are shared.

Furthermore, the system should be a separate independent unit for the user.

Heterogeneity

The components of distributed systems may differ and vary in operating systems, networks, programming languages, computer hardware, and implementations by different developers.

Fault Tolerance

Fault tolerance is that process in which user may continue their work if the software or hardware fails.

Examples of Distributed Operating System

There are various examples of the distributed operating system. Some of them are as follows:

Solaris

It is designed for the SUN multiprocessor workstations

OSF/1

It's compatible with Unix and was designed by the Open Foundation Software Company.

Micros

The MICROS operating system ensures a balanced data load while allocating jobs to all nodes in the system.

DYNIX

It is developed for the Symmetry multiprocessor computers.

Locus

It may be accessed local and remote files at the same time without any location hindrance.

Mach

It allows the multithreading and multitasking features.

Applications of Distributed Operating System

There are various applications of the distributed operating system. Some of them are as follows:

Network Applications

DOS is used by many network applications, including the Web, peer-to-peer networks, multiplayer web-based games, and virtual communities.

Telecommunication Networks

DOS is useful in phones and cellular networks. A DOS can be found in networks like the Internet, wireless sensor networks, and routing algorithms.

Parallel Computation

DOS is the basis of systematic computing, which includes cluster computing and grid computing, and a variety of volunteer computing projects.

Real-Time Process Control

The real-time process control system operates with a deadline, and such examples include aircraft control systems.

Advantages and Disadvantages of Distributed Operating System

There are various advantages and disadvantages of the distributed operating system. Some of them are as follows:

Advantages

There are various advantages of the distributed operating system. Some of them are as follow:

1. It may share all resources (CPU, disk, network interface, nodes, computers, and so on) from one site to another, increasing data availability across the entire system.
2. It reduces the probability of data corruption because all data is replicated across all sites; if one site fails, the user can access data from another operational site.
3. The entire system operates independently of one another, and as a result, if one site crashes, the entire system does not halt.
4. It increases the speed of data exchange from one site to another site.
5. It is an open system since it may be accessed from both local and remote locations.
6. It helps in the reduction of data processing time.
7. Most distributed systems are made up of several nodes that interact to make them fault-tolerant.
8. If a single machine fails, the system remains operational.

Disadvantages

There are various disadvantages of the distributed operating system. Some of them are as follows:

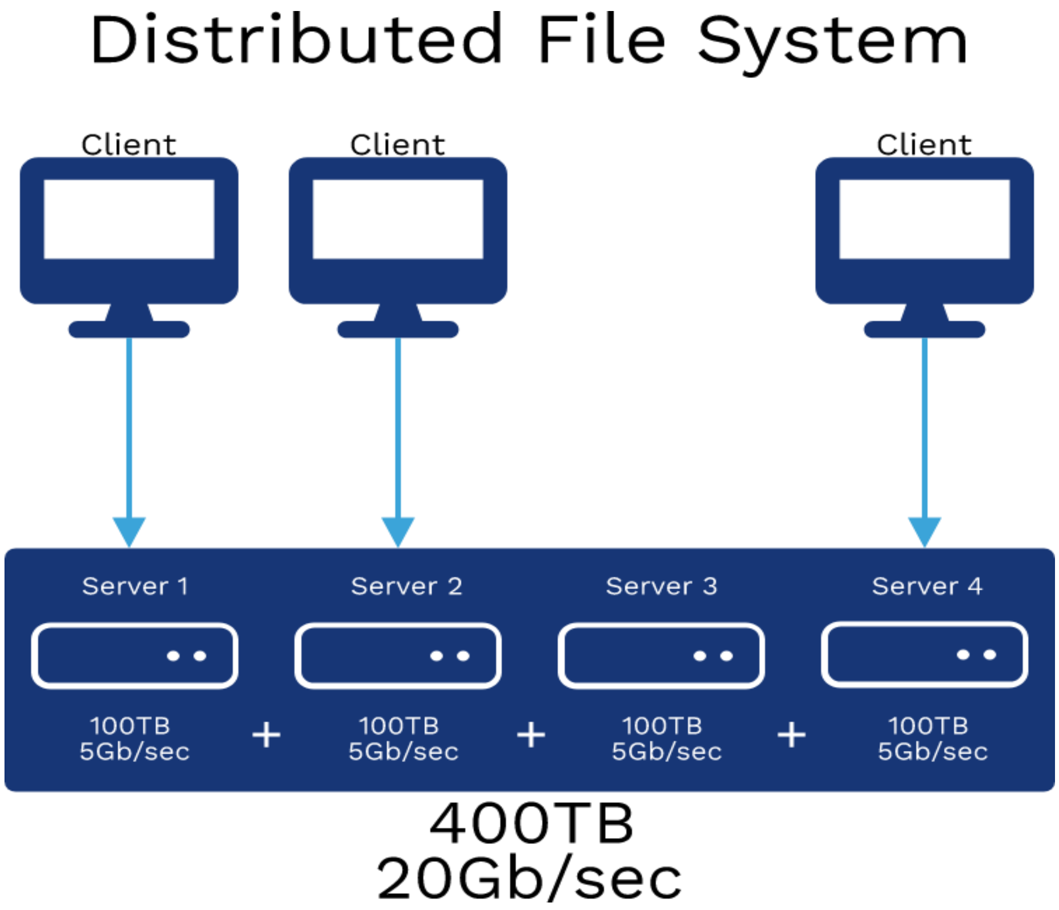
1. The system must decide which jobs must be executed when they must be executed, and where they must be executed.
2. A scheduler has limitations, which can lead to underutilized hardware and unpredictable runtimes.
3. It is hard to implement adequate security in DOS since the nodes and connections must be secured.
4. The database connected to a DOS is relatively complicated and hard to manage in contrast to a single-user system.
5. The underlying software is extremely complex and is not understood very well compared to other systems.
6. The more widely distributed a system is, the more communication latency can be expected.
7. As a result, teams and developers must choose between availability, consistency, and latency.
8. These systems aren't widely available because they're thought to be too expensive.

Distributed File System

- In this article, you will learn about the distributed file system in the operating system and its features, components, advantages, and disadvantages.

What is Distributed File System?

- A **distributed file system (DFS)** is a file system that is distributed on various file servers and locations.
- It permits programs to access and store isolated data in the same method as in the local files.
- It also permits the user to access files from any system.
- It allows network users to share information and files in a regulated and permitted manner.
- Although, the servers have complete control over the data and provide users access control.
- DFS's primary goal is to enable users of physically distributed systems to share resources and information through the **Common File System (CFS)**.
- It is a file system that runs as a part of the operating systems. Its configuration is a set of workstations and mainframes that a LAN connects.
- The process of creating a namespace in DFS is transparent to the clients.



DFS has two components in its services, and these are as follows:

1. **Local Transparency**
2. **Redundancy**

Local Transparency

It is achieved via the namespace component.

Redundancy

- It is achieved via a file replication component.
- In the case of failure or heavy load, these components work together to increase data availability by allowing data from multiple places to

be logically combined under a single folder known as the **"DFS root"**.

- It is not required to use both DFS components simultaneously; the namespace component can be used without the file replication component, and the file replication component can be used between servers without the namespace component.

Distributed File System Replication

Initial versions of DFS used **Microsoft's File Replication Service (FRS)**, enabling basic file replication among servers.

FRS detects new or altered files and distributes the most recent versions of the full file to all servers.

Windows Server 2003 R2 developed the "**DFS Replication" (DFSR)**.

It helps to enhance FRS by only copying the parts of files that have changed and reducing network traffic with data compression.

It also gives users the ability to control network traffic on a configurable schedule using flexible configuration options.

History of Distributed File System

The DFS's server component was firstly introduced as an additional feature.

When it was incorporated into **Windows NT 4.0 Server**, it was called **"DFS 4.1"**. Later, it was declared a standard component of all **Windows 2000 Server** editions.

Windows NT 4.0 and later versions of Windows have client-side support.

Linux kernels 2.6.14 and later include a DFS-compatible SMB client VFS known as **"cifs"**. DFS is available in versions **Mac OS X 10.7** (Lion) and later.

Working of Distributed File System

There are two methods of DFS in which they might be implemented, and these are as follows:

1. **Standalone DFS namespace**
2. **Domain-based DFS namespace**

Standalone DFS namespace

It does not use Active Directory and only permits DFS roots that exist on the local system.

A Standalone DFS may only be acquired on the systems that created it. It offers no-fault liberation and may not be linked to other DFS.

Domain-based DFS namespace

It stores the DFS configuration in Active Directory and creating namespace root at **domainname>dfsroot>** or **FQDN>dfsroot>**.

DFS namespace

SMB routes of the form are used in traditional file shares that are linked to a single server.

Domain-based DFS file share paths are identified by utilizing the domain name for the server's name throughout the form.

When users access such a share, either directly or through mapping a disk, their computer connects to one of the accessible servers connected with that share, based on rules defined by the network administrator.

For example, the default behavior is for users to access the nearest server to them; however, this can be changed to prefer a certain server.

Applications of Distributed File System

There are several applications of the distributed file system. Some of them are as follows:

Hadoop

Hadoop is a collection of open-source software services.

It is a software framework that uses the MapReduce programming style to allow distributed storage and management of large amounts of data.

Hadoop is made up of a storage component known as **Hadoop Distributed File System (HDFS)**.

It is an operational component based on the MapReduce programming model.

NFS (Network File System)

A client-server architecture enables a computer user to store, update, and view files remotely.

It is one of various DFS standards for Network-Attached Storage.

SMB (Server Message Block)

IBM developed an SMB protocol to file sharing. It was developed to permit systems to read and write files to a remote host across a LAN. The remote host's directories may be accessed through SMB and are known as **"shares"**.

NetWare

It is an abandon computer network operating system that is developed by Novell, Inc.

The IPX network protocol mainly used combined multitasking to execute many services on a computer system.

CIFS (Common Internet File System)

CIFS is an accent of SMB. The CIFS protocol is a Microsoft-designed implementation of the SIMB protocol.

Advantages and Disadvantages of Distributed File System

There are various advantages and disadvantages of the distributed file system. These are as follows:

Advantages

There are various advantages of the distributed file system. Some of the advantages are as follows:

1. It allows the users to access and store the data.
2. It helps to improve the access time, network efficiency, and availability of files.
3. It provides the transparency of data even if the server of disk files.
4. It permits the data to be shared remotely.
5. It helps to enhance the ability to change the amount of data and exchange data.

Disadvantages

There are various disadvantages of the distributed file system. Some of the disadvantages are as follows:

1. In a DFS, the database connection is complicated.

2. In a DFS, database handling is also more complex than in a single-user system.
3. If all nodes try to transfer data simultaneously, there is a chance that overloading will happen.
4. There is a possibility that messages and data would be missed in the network while moving from one node to another.

Features

There are various features of the DFS. Some of them are as follows:

Transparency

There are mainly four types of transparency. These are as follows:

1. Structure Transparency

The client does not need to be aware of the number or location of file servers and storage devices.

In structure transparency, multiple file servers must be given to adaptability, dependability, and performance.

2. Naming Transparency

There should be no hint of the file's location in the file's name. When the file is transferred form one node to other, the file name should not be changed.

3. Access Transparency

Local and remote files must be accessible in the same method.

The file system must automatically locate the accessed file and deliver it to the client.

4. Replication Transparency

When a file is copied across various nodes, the copies files and their locations must be hidden from one node to the next.

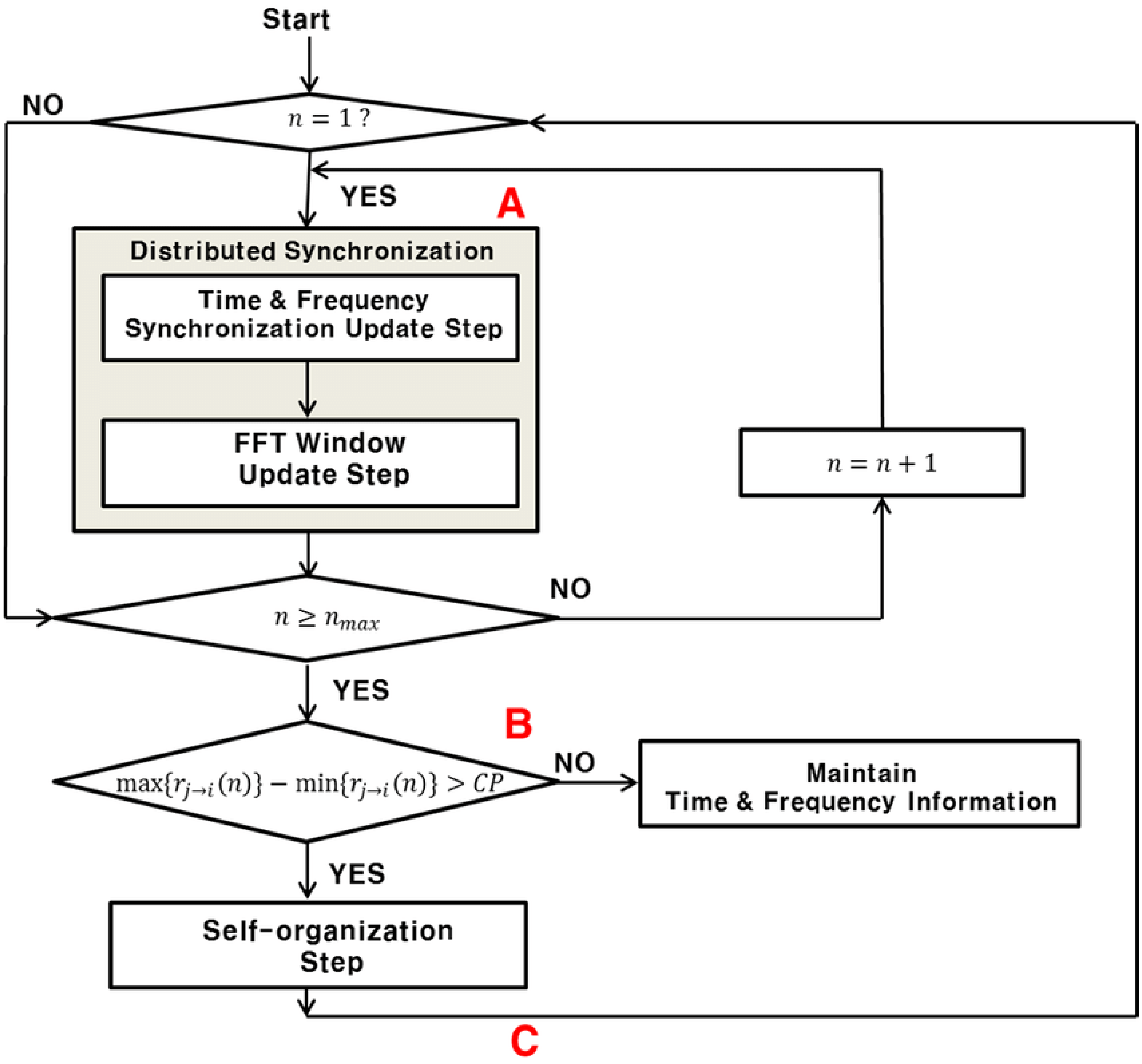
Distributed syncharonization

- Distributed system is a collection of computers connected via the high speed communication network.
- In the distributed system, the hardware and software components communicate and coordinate their actions by message passing.

- Each node in distributed systems can share their resources with other nodes.
- So, there is need of proper allocation of resources to preserve the state of resources and help coordinate between the several processes.
- To resolve such conflicts, synchronization is used. Synchronization in distributed systems is achieved via clocks.
- The physical clocks are used to adjust the time of nodes.Each node in the system can share its local time with other nodes in the system.
- The time is set based on UTC (Universal Time Coordination). UTC is used as a reference time clock for the nodes in the system.
- The clock synchronization can be achieved by 2 ways: External and Internal Clock Synchronization.

1. **External clock synchronization** is the one in which an external reference clock is present.
2. It is used as a reference and the nodes in the system can set and adjust their time accordingly.
3. **Internal clock synchronization** is the one in which each node shares its time with other nodes and all the nodes set and adjust their times accordingly.

There are 2 types of clock synchronization algorithms: Centralized and Distributed.


Start
NO
$n = 1\,?$
YES
A
Distributed Synchronization
Time & Frequency Synchronization Update Step
FFT Window Update Step
$n = n + 1$
NO
$n \geq n_{max}$
YES
B
NO
$\max\{r_{j \to i}(n)\} - \min\{r_{j \to i}(n)\} > CP$
Maintain Time & Frequency Information
YES
Self-organization Step
C

1. **Centralized** is the one in which a time server is used as a reference.

   The single time server propagates its time to the nodes and all the nodes adjust the time accordingly.

   It is dependent on single time server so if that node fails, the whole system will lose synchronization.

   Examples of centralized are- Berkeley Algorithm, Passive Time Server, Active Time Server etc.
2. **Distributed** is the one in which there is no centralized time server present.

Instead the nodes adjust their time by using their local time and then, taking the average of the differences of time with other nodes.
Distributed algorithms overcome the issue of centralized algorithms like the scalability and single point failure.
Examples of Distributed algorithms are – Global Averaging Algorithm, Localized Averaging Algorithm, NTP (Network time protocol) etc.

Properties of Distributed algorithms to maintain Clock synchronization:

- Relevant and correct information will be scattered among multiple machines.
- The processes make the decision only on local information.
- Failure of the single point in the system must be avoided.
- No common clock or the other precise global time exists.
- In the distributed systems, the time is ambiguous.

As the distributed systems has its own clocks.

The time among the clocks may also vary. So, it is possible to synchronize all the clocks in distributed environment.

Types of Clock Synchronization

- Physical clock synchronization
- Logical clock synchronization

- Mutual exclusion synchronization

Physical Synchronization:

- In physical clock synchronization, All the computers will have their own clocks.
- The physical clocks are needed to adjust the time of nodes. All the nodes in the system can share their local time with all other nodes in the system.
- The time will be set based on UTC (Universal Coordinate Timer).
- The time difference between the two computers is known as “Time drift”. Clock drifts over the time is known as “Skew”. Synchronization is necessary here.

Physical clocks: In physical synchronization, physical clocks are used to time stamp an event on that computer.
If two events, E1 and E2, having different time stamps t1 and t2, the order of the event occurring will be considered and not on the exact time or the day at which they are occur.

Several methods are used to attempt the synchronization of the physical clocks in Distributed synchronization:

1. UTC (Universal coordinate timer)
2. Christian’s algorithm
3. Berkely’s algorithm

Universal Coordinate Time (UTC)

- All the computers are generally synchronized to a standard time called Universal Coordinate Time (UTC).
- UTC is the primary time standard by which the time and the clock are regulated in the world.
- It is available via radio signals, telephone line and satellites (GPS).
- UTC is broadcasted via the satellites.
- Computer servers and online services with the UTC resources can be synchronized by the satellite broadcast.
- Kept within 0.9 seconds of UTI.

UTO - Mean solar time on Greenwich meridian, Obtained from astronomical observation.

UT1 - UTO corrected for polar motion.

UT2 - UT1 corrected for seasonal variations in earth's rotation.

UTC - Civil time will be measured on an atomic scale.

Christian's Algorithm:

- The simplest algorithm for setting time, it issues a remote procedure call (RPC) to the time sever and obtains the time.
- The machine which send requests to the time server is "d/z" seconds, where d is the maximum difference between the clock and the UTC.

- The time server sends the reply with current UTC when receives the request from the receiver.