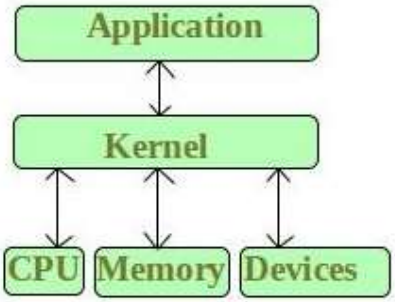
1)Describe microkernel operating system structure

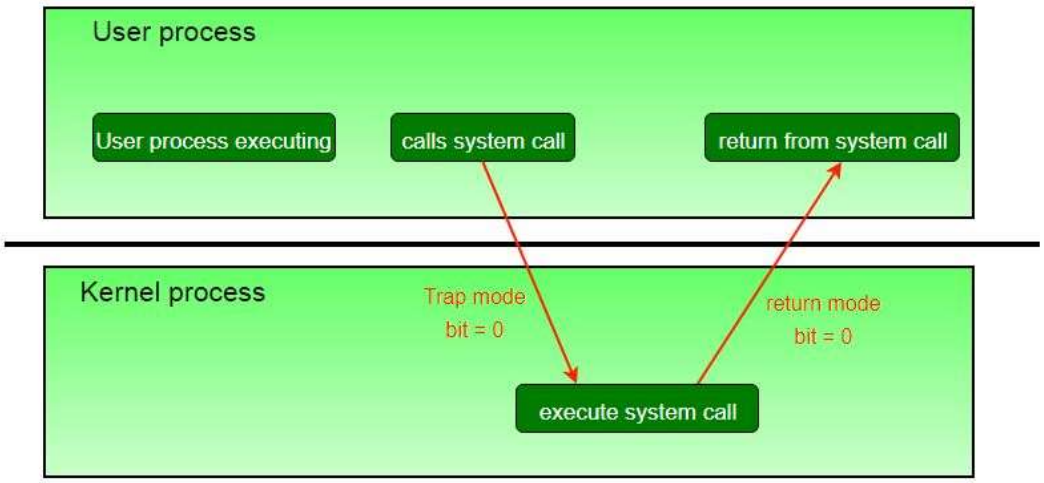
Kernel is the core part of an operating system that manages system resources. It also acts as a bridge between the application and hardware of the computer. It is one of the first programs loaded on start-up (after the Bootloader).



Kernel mode and User mode of CPU operation

The CPU can execute certain instructions only when it is in kernel mode. These instructions are called privilege instruction. They allow the implementation of special operations whose execution by the user program could interface with the functioning of the operating system or activity of another user program. For example, instruction for managing memory protection.

- The operating system puts the CPU in kernel mode when it is executing in the kernel so, that kernel can execute some special operation.
- The operating system puts the CPU in user mode when a user program is in execution so, that the user program cannot interface with the operating system program.
- User-level instruction does not require special privilege. Example are ADD,PUSH,etc.

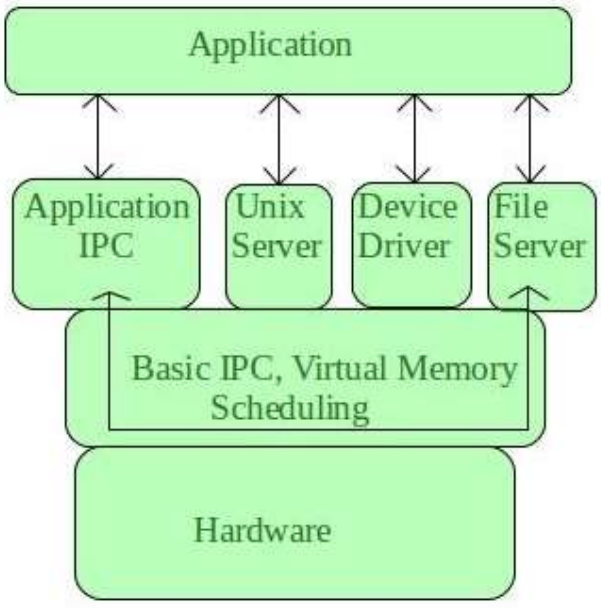


The concept of modes can be extended beyond two, requiring more than a single mode bit CPUs that support virtualization use one of these extra bits to indicate when the virtual machine manager, VMM, is in control of the system. The VMM has more privileges than ordinary user programs, but not so many as the full kernel.

System calls are typically implemented in the form of software interrupts, which causes the hardware's interrupt handler to transfer control over to an appropriate interrupt handler, which is part of the operating system, switching the mode bit to kernel mode in the process. The interrupt handler checks exactly which interrupt was generated, checks additional parameters (generally passed through registers) if appropriate, and then calls the appropriate kernel service routine to handle the service requested by the system call.

User programs' attempts to execute illegal instructions (privileged or non-existent instructions), or to access forbidden memory areas, also generate software interrupts, which are trapped by the interrupt handler, and control is transferred to the OS, which issues an appropriate error message, possibly dumps data to a log (core) file for later analysis, and then terminates the offending program.

What is Microkernel?
A microkernel is one of the classifications of the kernel. Being a kernel it manages all system resources. But in a microkernel, the user services and kernel services are implemented in different address spaces. The user services are kept in user address space, and kernel services are kept under kernel address space, thus also reduces the size of kernel and size of an operating system as well.



It provides minimal services of process and memory management. The communication between client program/application and services running in user address space is established through message passing, reducing the speed of execution microkernel. The Operating System remains unaffected as user services and kernel services are isolated so if any user service fails it does not affect kernel service. Thus it adds to one of the advantages

of a microkernel. It is easily extendible i.e. if any new services are to be added they are added to user address space and hence require no modification in kernel space. It is also portable, secure, and reliable.

Microkernel Architecture –

Since the kernel is the core part of the operating system, so it is meant for handling the most important services only. Thus in this architecture, only the most important services are inside the kernel and the rest of the OS services are present inside the system application program. Thus users are able to interact with those not-so-important services within the system application. And the microkernel is solely responsible for the most important services of the operating system they are named as follows:

- Inter process-Communication
- Memory Management
- CPU-Scheduling

Advantages of Microkernel –

- The architecture of this kernel is small and isolated hence it can function better.
- Expansion of the system is easier, it is simply added to the system application without disturbing the kernel.

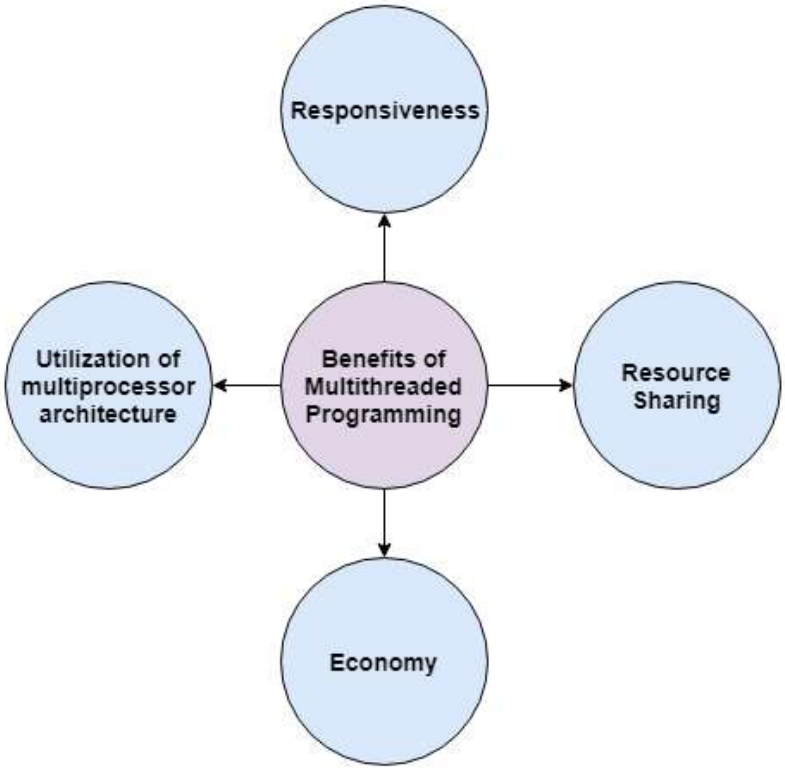
Eclipse IDE is a good example of Microkernel Architecture.

2) What is thread? Describe any four advantages of multithreading model.

A thread is a path of execution within a process. A process can contain multiple threads. A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc. More advantages of multithreading are discussed below

Multithreading allows the execution of multiple parts of a program at the same time. These parts are known as threads and are lightweight processes available within the process. So multithreading leads to maximum utilization of the CPU by multitasking.

Some of the benefits of multithreaded programming are given as follows −



- **Resource Sharing**
  All the threads of a process share its resources such as memory, data, files etc. A single application can have different threads within the same address space using resource sharing.
- **Responsiveness**
  Program responsiveness allows a program to run even if part of it is blocked using multithreading. This can also be done if the process is performing a lengthy operation. For example - A web browser with

multithreading can use one thread for user contact and another for image loading at the same time.

- **Utilization of Multiprocessor Architecture**
  In a multiprocessor architecture, each thread can run on a different processor in parallel using multithreading. This increases concurrency of the system. This is in direct contrast to a single processor system, where only one process or thread can run on a processor at a time.
- **Economy**
  It is more economical to use threads as they share the process resources. Comparatively, it is more expensive and time-consuming to create processes as they require more memory and resources. The overhead for process creation and management is much higher than thread creation and management.

3) Why is semaphore known as a synchronisation tool? Give an example

Semaphore is simply a variable that is non-negative and shared between threads. A semaphore is a signaling mechanism, and a thread that is waiting on a semaphore can be signaled by another thread. It uses two atomic operations, 1) Wait, and 2) Signal for the process synchronization.

A semaphore either allows or disallows access to the resource, which depends on how it is set up.

Characteristic of Semaphore

Here, are characteristic of a semaphore:

- It is a mechanism that can be used to provide synchronization of tasks.
- It is a low-level synchronization mechanism.
- Semaphore will always hold a non-negative integer value.
- Semaphore can be implemented using test operations and interrupts, which should be executed using file descriptors.

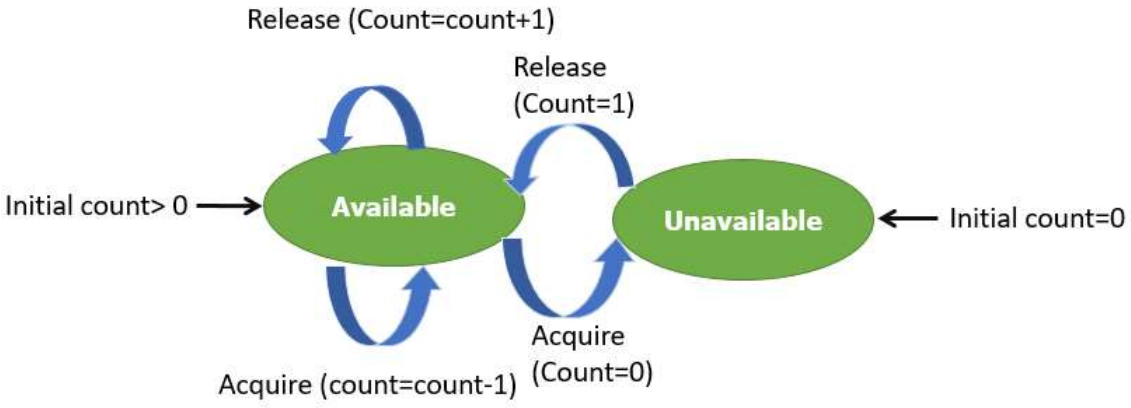
Types of Semaphores

The two common kinds of semaphores are

- Counting semaphores
- Binary semaphores.

Counting Semaphores

This type of Semaphore uses a count that helps task to be acquired or released numerous times. If the initial count = 0, the counting semaphore should be created in the unavailable state.

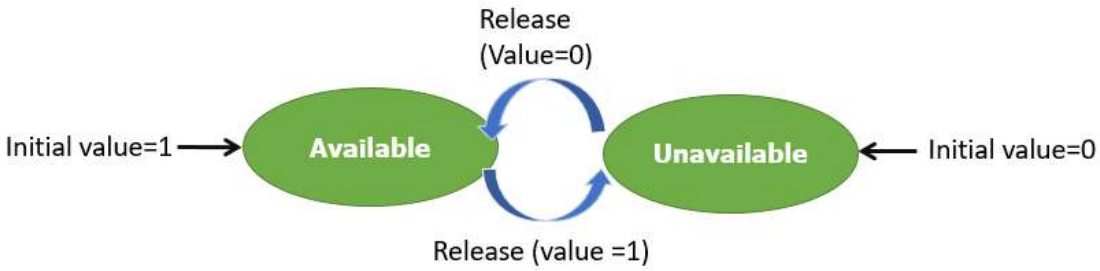


Counting Semaphores

However, If the count is > 0, the semaphore is created in the available state, and the number of tokens it has equals to its count.

Binary Semaphores

The binary semaphores are quite similar to counting semaphores, but their value is restricted to 0 and 1. In this type of semaphore, the wait operation works only if semaphore = 1, and the signal operation succeeds when semaphore= 0. It is easy to implement than counting semaphores.



Binary Semaphores

Example of Semaphore

The below-given program is a step by step implementation, which involves usage and declaration of semaphore.

```
Shared var mutex: semaphore = 1;
Process i
   begin
   .
   .
   P(mutex);
   execute CS;
   V(mutex);
   .
   .
   End;
```

Wait and Signal Operations in Semaphores

Both of these operations are used to implement process synchronization. The goal of this semaphore operation is to get mutual exclusion.

Wait for Operation

This type of semaphore operation helps you to control the entry of a task into the critical section. However, If the value of wait is positive, then the value of the wait argument X is

decremented. In the case of negative or zero value, no operation is executed. It is also called P(S) operation.

After the semaphore value is decreased, which becomes negative, the command is held up until the required conditions are satisfied.

Copy CodeP(S)

```
{
   while (S<=0);
   S--;
}
```

Signal operation

This type of Semaphore operation is used to control the exit of a task from a critical section. It helps to increase the value of the argument by 1, which is denoted as V(S).

Copy CodeP(S)

```
{
   while (S>=0);
   S++;
}
```

Advantages of Semaphores

Here, are pros/benefits of using Semaphore:

- It allows more than one thread to access the critical section
- Semaphores are machine-independent.
- Semaphores are implemented in the machine-independent code of the microkernel.
- They do not allow multiple processes to enter the critical section.
- As there is busy waiting in semaphore, there is never a wastage of process time and resources.
- They are machine-independent, which should be run in the machine-independent code of the microkernel.
- They allow flexible management of resources.

Disadvantage of semaphores

Here, are cons/drawback of semaphore

- One of the biggest limitations of a semaphore is priority inversion.
- The operating system has to keep track of all calls to wait and signal semaphore.
- Their use is never enforced, but it is by convention only.
- In order to avoid deadlocks in semaphore, the Wait and Signal operations require to be executed in the correct order.
- Semaphore programming is a complicated, so there are chances of not achieving mutual exclusion.
- It is also not a practical method for large scale use as their use leads to loss of modularity.

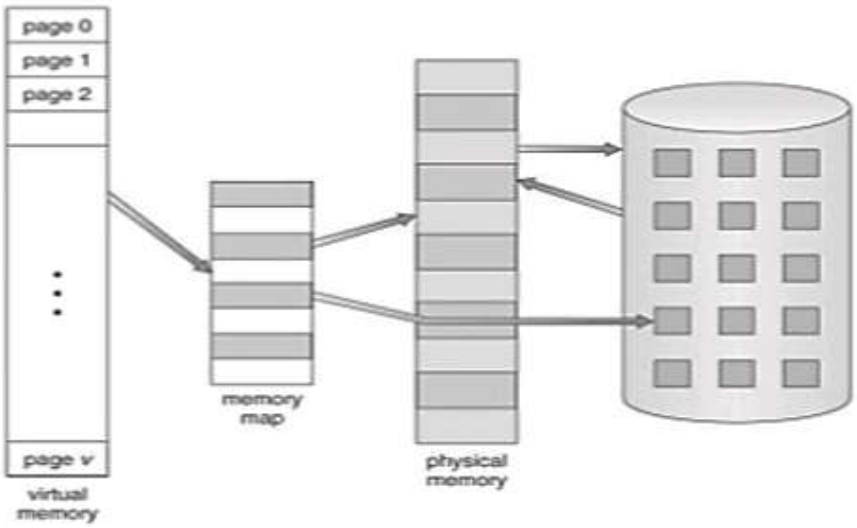
- Semaphore is more prone to programmer error.
- It may cause deadlock or violation of mutual exclusion due to programmer error.

4) Describe how logical address is converted into physical address when the program and its associated data is divided into segments

- Paging is one of the memory management schemes by which a computer stores and retrieves data from the secondary storage for use in main memory.
- In the paging memory-management scheme, the operating system retrieves data from secondary storage in same-size blocks called pages.
- Paging allows the physical address space of the process to be non contiguous
- Paging is to deal with external fragmentation problem. This is to allow the logical address space of a process to be non-contiguous, which makes the process to be allocated physical memory.
- Logical address space of a process can be non-contiguous; process is allocated physical memory.

Whenever the latter is available.

- Divide physical memory into fixed-sized blocks called frames (size is power of 2, between 512 bytes and 8192 bytes).
- Divide logical memory into blocks of same size called pages.
- 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-allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- The logical memory of the process is contiguous, but pages need not be allocated contiguously in memory.
- By dividing memory into fixed size pages, we can eliminate external fragmentation.
- Paging does not eliminate internal fragmentation



Address generated by CPU is divided into:

1. Page number (p) – used as an index into a page table which contains base address of each page in physical memory.

2. Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit.

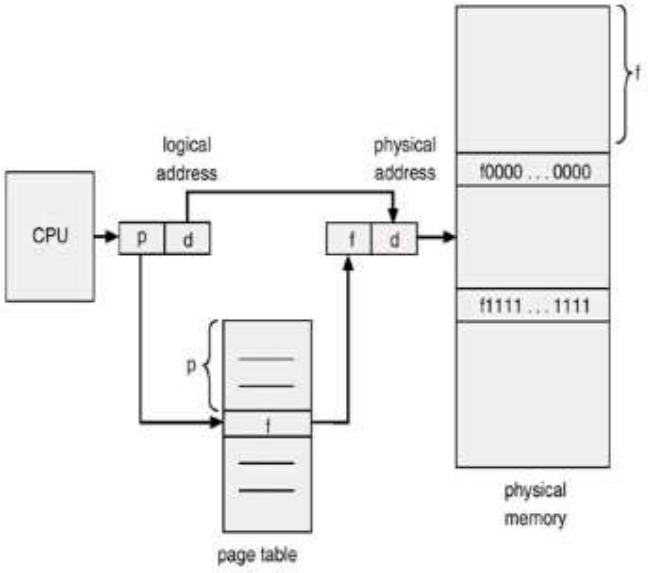


Fig. Address translation scheme

Implementation of page table:

- Page table is kept in main memory.
- Page-table base register (PTBR) points to the page table.
- Page-table length register (PTLR) indicates size of the page table.
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called associative registers or translation look-aside buffers (TLBs).
- Associative registers - parallel search

Page No	Frame No

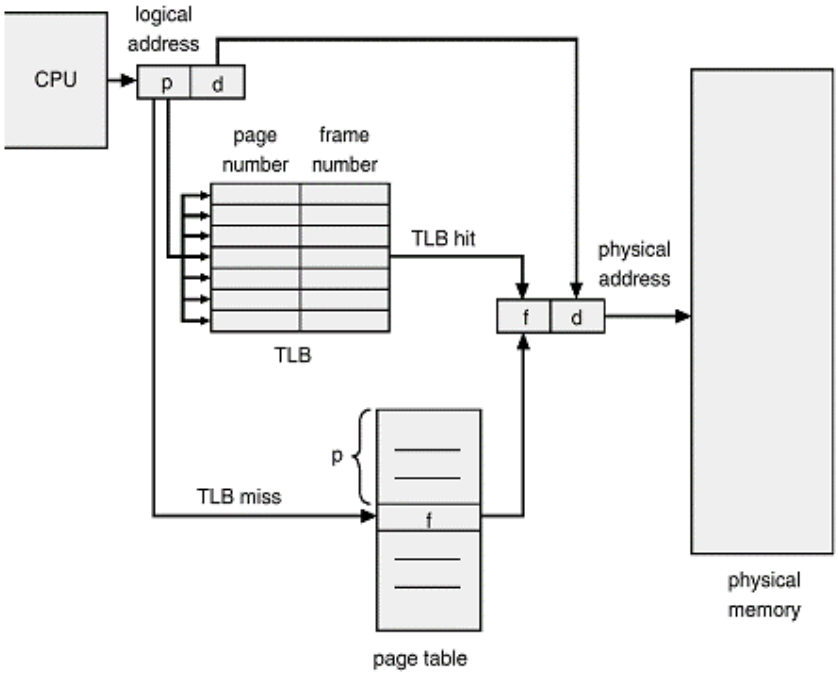
Address translation (A', A")

- If A' in associative register, get frame number out.
- Otherwise get frame number from page table in memory.

    - Hit ratio - percentage of times that a page number is found in the associative registers; ratio related to number of associative registers.
    - Effective Access Time (EAT)
    - associative lookup = e time units
    - memory cycle time = m time units
    - hit ratio = a

EAT = (m + e) a+ (2m + e) (1 - a) = 2m + e - ma

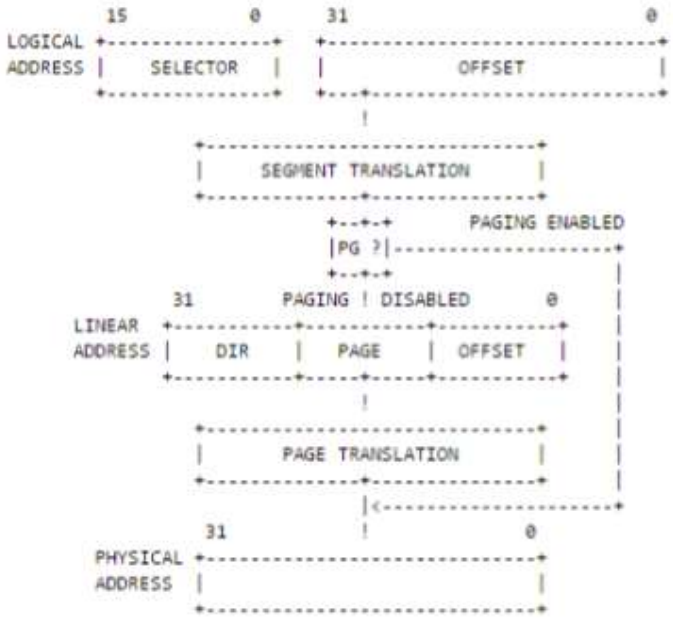
- Memory protection implemented by associating protection bits with each frame.
- Valid-invalid bit attached to each entry in the page table:
    - ``valid'' indicates that the associated page is in the process' logical address space, and is thus a legal page.
    - ``invalid'' indicates that the page is not in the process' logical address space.
- Write bit attached to each entry in the page table.
    - Pages which have not been written may be shared between processes
    - Do not need to be swapped - can be reloaded.



Conversion of logical address to physical address

- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management.
    - Logical address - generated by the CPU; also referred to as virtual address.
    - Physical address - address seen by the memory unit.

- 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.

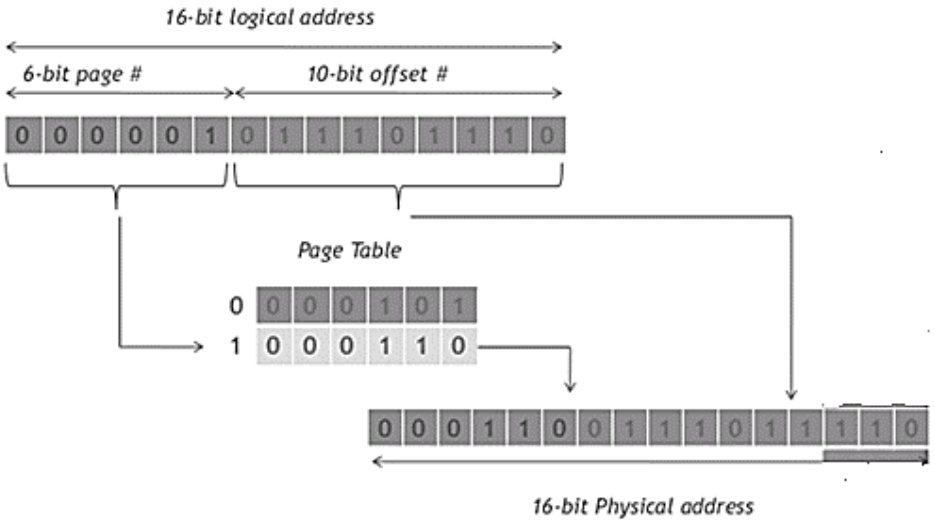


There are three steps to translate logical address to physical address

Step1: Find the index field from the segment selector and use the index field to locate the segment descriptor for the segment in global descriptor table (GDT)

Step 2: Test the access and limit the field of descriptor to make sure that the segment is accessible and the offset is within the limit of the segment

Step3: The base address of the segment will be obtained from the segment descriptor. Then the base address of the segment will be added to the offset to determine a linear address



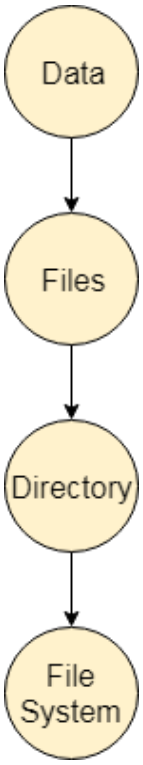
5) Summarize various File Attributes

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

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

Functions of File

- Create file, find space on disk, and make an entry in the directory.
- Write to file, requires positioning within the file
- Read from file involves positioning within the file
- Delete directory entry, regain disk space.
- Reposition: move read/write position.
- he collection of files is known as Directory. The collection of directories at the different levels, is known as File System.
-



- Attributes of the File
- **1.Name**
- Every file carries a name by which the file is recognized in the file system. One directory cannot have two files with the same name.
- **2.Identifier**

- Along with the name, Each File has its own extension which identifies the type of the file. For example, a text file has the extension **.txt,** A video file can have the extension **.mp4.**
- **3.Type**
- In a File System, the Files are classified in different types such as video files, audio files, text files, executable files, etc.
- **4.Location**
- In the File System, there are several locations on which, the files can be stored. Each file carries its location as its attribute.
- **5.Size**
- The Size of the File is one of its most important attribute. By size of the file, we mean the number of bytes acquired by the file in the memory.
- **6.Protection**
- The Admin of the computer may want the different protections for the different files. Therefore each file carries its own set of permissions to the different group of Users.
- **7.Time and Date**
- Every file carries a time stamp which contains the time and date on which the file is last modified.

6) With the help of a diagram explain I/O management.

7) Compare short term, medium term and long-term scheduler along with diagram

- **Summary**: **Schedulers** are special system software that handles process scheduling in various ways. Their main task is to select the job to decide which process to run first.
- **Schedulers** are **Three** types:
    0. **Long Term Scheduler**
    1. **Short Term Scheduler**
    2. **Medium Term Scheduler**
- **Long Term Scheduler is also called job scheduler. Short Term Scheduler** is also called **CPU scheduler.** Medium term scheduling is part of the swapping.

Comparison Chart

Long Term	Short Term	Medium Term
It is a job scheduler.	It is a CPU scheduler.	It is swapping.
Speed is less than short term scheduler.	Speed is very fast.	Speed is in between both
It controls the degree of multiprogramming.	Less control over the degree of multiprogramming.	Reduce the degree of multiprogramming.
Absent or minimal in a time-sharing system.	Minimal in a time-sharing system.	Time-sharing system uses a medium-term scheduler.
It selects processes from the pool and load them into memory for execution.	It selects from among the processes that are ready to execute.	Process can be reintroduced into the meat and its execution can be continued.
Process state is (New to Ready).	Process state is (Ready to Running)	_
Select a good prccess, mix of I/O bound, and CPU bound.	Select a new process for a CPU quite frequently.	_

Long Term Scheduler

- It is also called job scheduler.
- Long term scheduler determines which programs are admitted to the system for processing.
- Job scheduler selects processes from the queue and loads them into memory for execution.
- Process loads into the memory for CPU scheduling. The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound.
- It also controls the degree of multiprogramming.
- If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.
- On some systems, the long term scheduler may not be available or minimal.
- Time-sharing operating systems have no long term scheduler.
- When the process changes the state from new to ready, then there is the use of long term scheduler.

Short Term Scheduler

- It is also called **CPU scheduler.**
- The main objective is increasing system performance in accordance with the chosen set of criteria.
- It is the change of ready state to the running state of the process.
- CPU scheduler selects process among the processes that are ready to execute and allocates CPU to one of them.
- Short term scheduler also known as a **dispatcher**, execute most frequently and makes the fine-grained decision of which process to execute next.
- Short term scheduler is faster than a long term scheduler.

Medium Term Scheduler

- Medium term scheduling is part of the swapping
- It removes the processes from the memory.
- It reduces the degree of multiprogramming.
- Running process may become suspended if it makes an I/O request.
- Suspended processes cannot make any progress towards completion.
- In this condition, to remove the process from memory and make space for other processes, the suspended process is a move to secondary storage.
- This process is called swapping, and the process is said to be swapped out or rolled out.
- Swapping may be necessary to improve the process mix of handling the swapped out-processes.

8) Consider a disk with 51(0 to 50) cylinders. While the seek to cylinder 11 is in progress, the request comes for the following cylinders, in the order 1, 36, 16, 34, 9, 12 and 40. The arm moves in an increasing number of cylinders. What is the total distance the arm moves to complete pending requests using FCFS and LOOK algorithms?

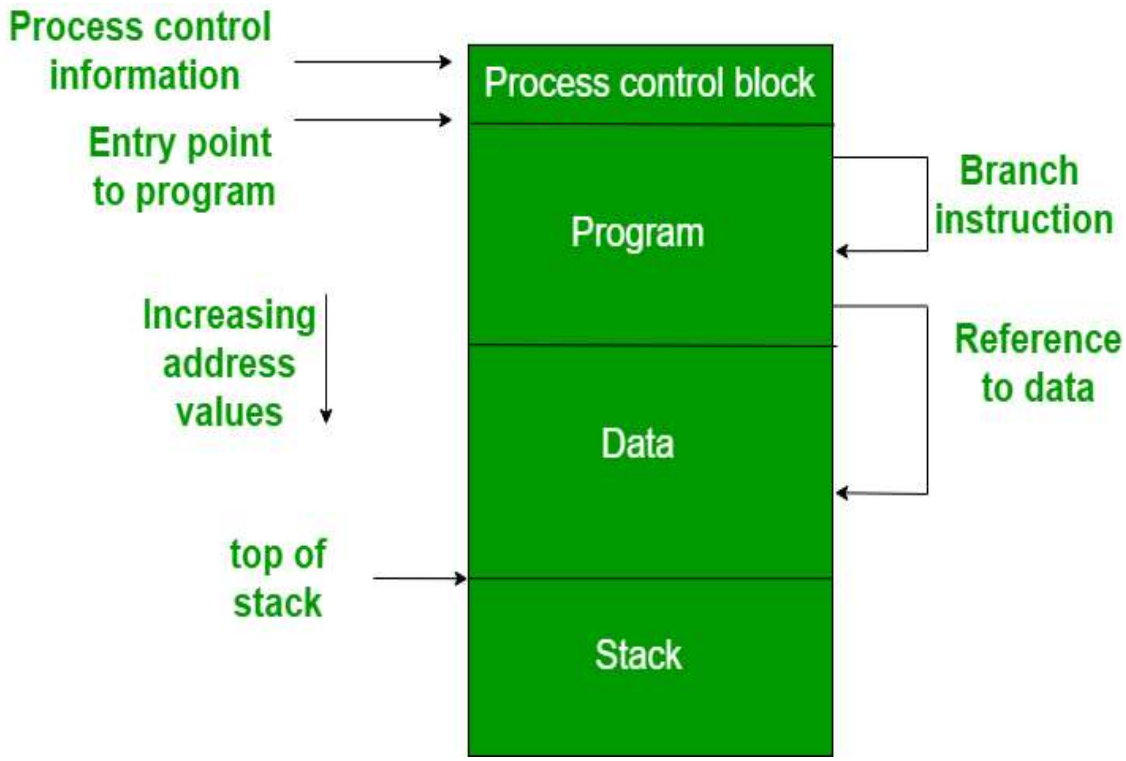
9) describe in detail requirements that intends to achieve memory Management

Memory management keeps track of the status of each memory location, whether it is allocated or free. It allocates the memory dynamically to the programs at their request and frees it for reuse when it is no longer needed. Memory management meant to satisfy some requirements that we should keep in mind.

These Requirements of memory management are:

1. **Relocation –** The available memory is generally shared among a number of processes in a multiprogramming system, so it is not possible to know in advance which other programs will be resident in main memory at the time of execution of his program. Swapping the active processes in and out of the main memory enables the operating system to have a larger pool of ready-to-execute process.

   When a program gets swapped out to a disk memory, then it is not always possible that when it is swapped back into main memory then it occupies the previous memory location, since the location may still be occupied by another process. We may need to **relocate** the process to a different area of memory. Thus there is a possibility that program may be moved in main memory due to swapping.



The figure depicts a process image. The process image is occupying a continuous region of main memory. The operating system will need to know many things including the location of process control information, the execution stack, and the code entry. Within a program, there are memory references in various instructions and these are called logical addresses.

After loading of the program into main memory, the processor and the operating system must be able to translate logical addresses into physical addresses. Branch instructions contain the address of the next instruction to be executed. Data reference instructions contain the address of byte or word of data referenced.

2. **Protection –** There is always a danger when we have multiple programs at the same time as one program may write to the address space of another program. So every process must be protected against unwanted interference when other process tries to write in a process whether accidental or incidental. Between relocation and protection requirement a trade-off occurs as the satisfaction of relocation requirement increases the difficulty of satisfying the protection requirement.

   Prediction of the location of a program in main memory is not possible, that's why it is impossible to check the absolute address at compile time to assure protection. Most of the programming language allows the dynamic calculation of address at run time. The memory protection requirement must be satisfied by the processor rather than the operating system because the operating system can hardly control a process when it occupies the processor. Thus it is possible to check the validity of memory references.

3. **Sharing –** A protection mechanism must have to allow several processes to access the same portion of main memory. Allowing each processes access to the same copy of the program rather than have their own separate copy has an advantage.

   For example, multiple processes may use the same system file and it is natural to load one copy of the file in main memory and let it shared by those processes. It is the task of Memory management to allow controlled access to the shared areas of memory without compromising the protection. Mechanisms are used to support relocation supported sharing capabilities.
4. **Logical organization –** Main memory is organized as linear or it can be a one-dimensional address space which consists of a sequence of bytes or words. Most of the programs can be organized into modules, some of those are unmodifiable (read-only, execute only) and some of those contain data that can be modified. To effectively deal with a user program, the operating system and computer hardware must support a basic module to provide the required protection and sharing. It has the following advantages:

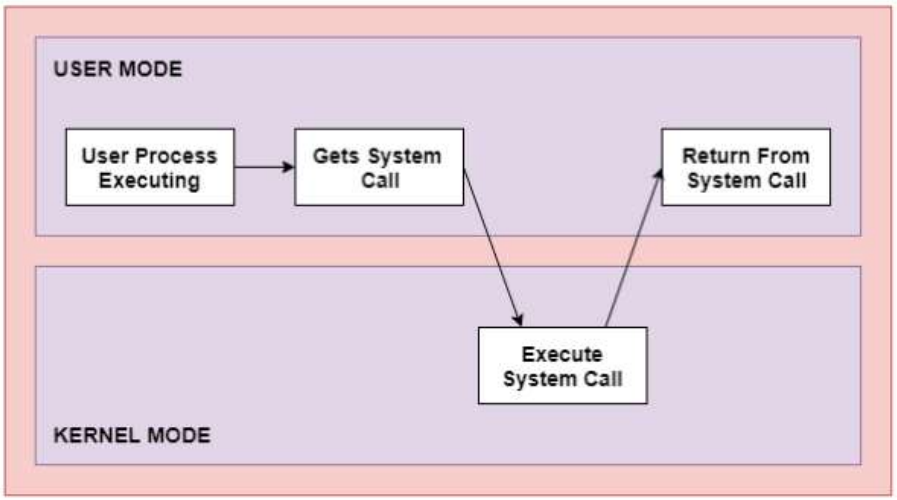
   - Modules are written and compiled independently and all the references from one module to another module are resolved by `the system at run time.
   - Different modules are provided with different degrees of protection.
   - There are mechanisms by which modules can be shared among processes. Sharing can be provided on a module level that lets the user specify the sharing that is desired.
5. **Physical organization –** The structure of computer memory has two levels referred to as main memory and secondary memory. Main memory is relatively very fast and costly as compared to the secondary memory. Main memory is volatile. Thus secondary memory is provided for storage of data on a long-term basis while the main memory holds currently used programs. The major system concern between main memory and secondary memory is the flow of information and it is impractical for programmers to understand this for two reasons:

   - The programmer may engage in a practice known as overlaying when the main memory available for a program and its data may be insufficient. It allows different modules to be assigned to the same region of memory. One disadvantage is that it is time-consuming for the programmer.
   - In a multiprogramming environment, the programmer does not know how much space will be available at the time of coding and where that space will be located inside the memory.

10) With help of a diagram explain how the system call will be generated?

The interface between a process and an operating system is provided by system calls. In general, system calls are available as assembly language instructions. They are also included in the manuals used by the assembly level programmers. System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.

A figure representing the execution of the system call is given as follows −



As can be seen from this diagram, the processes execute normally in the user mode until a system call interrupts this. Then the system call is executed on a priority basis in the kernel mode. After the execution of the system call, the control returns to the user mode and execution of user processes can be resumed.

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

- If a file system requires the creation or deletion of files. Reading and writing from files also require a system call.
- Creation and management of new processes.
- Network connections also require system calls. This includes sending and receiving packets.
- Access to a hardware devices such as a printer, scanner etc. requires a system call.

Types of System Calls

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

Process Control

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

File Management

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

Device Management

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

Information Maintenance

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

Communication

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

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

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

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

open()

The open() system call is used to provide access to a file in a file system. This system call allocates resources to the file and provides a handle that the process uses to refer to the file. A file can be opened by multiple processes at the same time or be restricted to one process. It all depends on the file organisation and file system.

read()

The read() system call is used to access data from a file that is stored in the file system. The file to read can be identified by its file descriptor and it should be opened using open() before

it can be read. In general, the read() system calls takes three arguments i.e. the file descriptor, buffer which stores read data and number of bytes to be read from the file.

write()

The write() system calls writes the data from a user buffer into a device such as a file. This system call is one of the ways to output data from a program. In general, the write system calls takes three arguments i.e. file descriptor, pointer to the buffer where data is stored and number of bytes to write from the buffer.

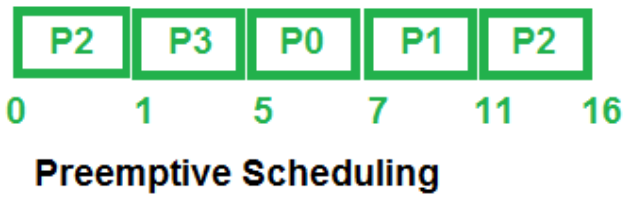
close()

The close() system call is used to terminate access to a file system. Using this system call means that the file is no longer required by the program and so the buffers are flushed, the file metadata is updated and the file resources are de-allocated.

11) Compare preemptive and non preemptive scheduling algorithm?

1. Preemptive Scheduling:
Preemptive scheduling is used when a process switches from running state to ready state or from the waiting state to ready state. The resources (mainly CPU cycles) are allocated to the process for a limited amount of time and then taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining. That process stays in the ready queue till it gets its next chance to execute.
Algorithms based on preemptive scheduling are: Round Robin (RR),Shortest Remaining Time First (SRTF), Priority (preemptive version), etc.

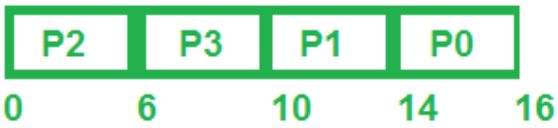
Process	Arrival Time	CPU Burst Time (in millisec.)
P0	3	2
P1	2	4
P2	0	6
P3	1	4



Preemptive Scheduling

2. Non-Preemptive Scheduling:
Non-preemptive Scheduling is used when a process terminates, or a process switches from running to the waiting state. In this scheduling, once the resources (CPU cycles) are allocated to a process, the process holds the CPU till it gets terminated or reaches a waiting state. In the case of non-preemptive scheduling does not interrupt a process running CPU in the middle of the execution. Instead, it waits till the process completes its CPU burst time, and then it can allocate the CPU to another process.
Algorithms based on non-preemptive scheduling are: Shortest Job First (SJF basically non preemptive) and Priority (non preemptive version), etc.

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



Non-Preemtive Scheduling

Key Differences Between Preemptive and Non-Preemptive Scheduling:

1. In preemptive scheduling, the CPU is allocated to the processes for a limited time whereas, in Non-preemptive scheduling, the CPU is allocated to the process till it terminates or switches to the waiting state.
2. The executing process in preemptive scheduling is interrupted in the middle of execution when higher priority one comes whereas, the executing process in non-preemptive scheduling is not interrupted in the middle of execution and waits till its execution.
3. In Preemptive Scheduling, there is the overhead of switching the process from the ready state to running state, vise-verse and maintaining the ready queue. Whereas in the case of non-preemptive scheduling has no overhead of switching the process from running state to ready state.
4. In preemptive scheduling, if a high-priority process frequently arrives in the ready queue then the process with low priority has to wait for a long, and it may have to starve. , in the non-preemptive scheduling, if CPU is allocated to the process having a larger burst time then the processes with small burst time may have to starve.
5. Preemptive scheduling attains flexibility by allowing the critical processes to access the CPU as they arrive into the ready queue, no matter what process is executing currently. Non-preemptive scheduling is called rigid as even if a critical process enters the ready queue the process running CPU is not disturbed.
6. Preemptive Scheduling has to maintain the integrity of shared data that's why it is cost associative which is not the case with Non-preemptive Scheduling.

Comparison Chart:

Parameter	PREEMPTIVE SCHEDULING	NON-PREEMPTIVE SCHEDULING
Basic	In this resources(CPU Cycle) are allocated to a process for a limited time.	Once resources(CPU Cycle) are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.

Parameter	PREEMPTIVE SCHEDULING	NON-PREEMPTIVE SCHEDULING
Interrupt	Process can be interrupted in between.	Process can not be interrupted until it terminates itself or its time is up.
Starvation	If a process having high priority frequently arrives in the ready queue, a low priority process may starve.	If a process with a long burst time is running CPU, then later coming process with less CPU burst time may starve.
Overhead	It has overheads of scheduling the processes.	It does not have overheads.
Flexibility	flexible	rigid
Cost	cost associated	no cost associated
CPU Utilization	In preemptive scheduling, CPU utilization is high.	It is low in non preemptive scheduling.
Examples	Examples of preemptive scheduling are Round Robin and Shortest Remaining Time First.	Examples of non-preemptive scheduling are First Come First Serve and Shortest Job First.

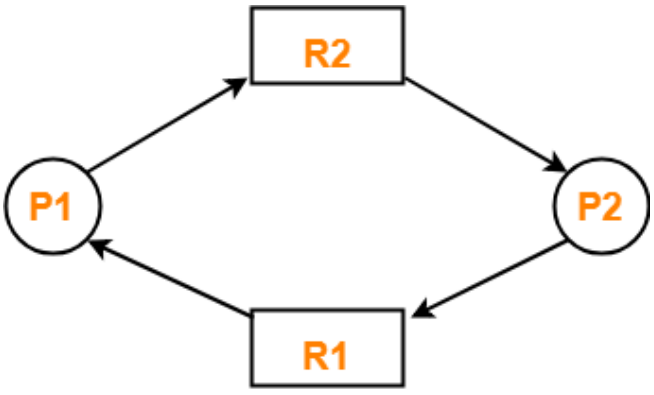
12) Define deadlock. List the conditions that lead to deadlock.

Deadlock is a situation which involves the interaction of more than one resources and processes with each other. We can visualise the occurrence of deadlock as a situation where there are two people on a staircase. One is ascending the staircase while the other is descending. The staircase is so narrow that it can only fit one person at a time. As a result, one has to retreat while the other moves on and uses the staircase. Once that person has finished, the other one can use that staircase. But here, neither person is willing to retreat and waits for the other to retreat. None of them are able to use the staircase. The people here are the processes, and the staircase is the resource. When a process requests for the resource that is been held another process which needs another resource to continue, but is been held by the first process, then it is called a **deadlock**. There are four conditions necessary for the occurrence of a deadlock. They can be understood with the help of the above illustrated example of staircase:

1. **Mutual Exclusion:** When two people meet in the landings, they can't just walk through because there is space only for one person. This condition allows only one person (or process) to use the step between them (or the resource) is the first condition necessary for the occurrence of the deadlock.
2. **Hold and Wait:** When the two people refuse to retreat and hold their ground, it is called holding. This is the next necessary condition for deadlock.
3. **No Preemption:** For resolving the deadlock one can simply cancel one of the processes for other to continue. But the Operating System doesn't do so. It allocates the resources to the processors for as much time as is needed until the task is completed. Hence, there is no temporary reallocation of the resources. It is the third condition for deadlock.
4. **Circular Wait:** When the two people refuse to retreat and wait for each other to retreat so that they can complete their task, it is called circular wait. It is the last condition for deadlock to occur.

Note: All four conditions are necessary for deadlock to occur. If any single one is prevented or resolved, the deadlock is resolved.

Example-



Example of a deadlock

Here

- Process P1 holds resource R1 and waits for resource R2 which is held by process P2.
- Process P2 holds resource R2 and waits for resource R1 which is held by process P1.
- None of the two processes can complete and release their resource.
- Thus, both the processes keep waiting infinitely.

13) Describe how logical address is converted into physical address when the process is strictly divided into equal size chunks

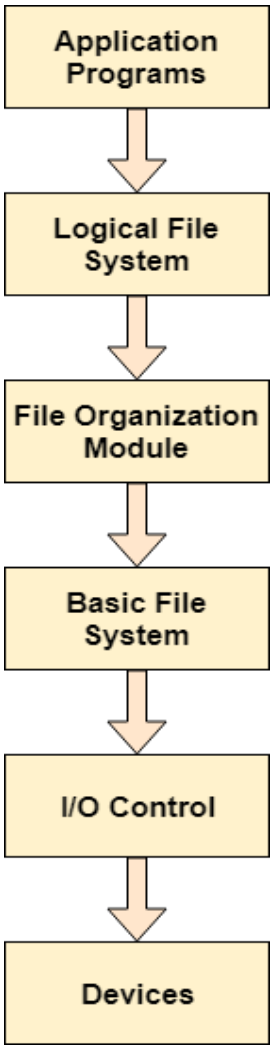
14) Summarize file system organization architecture

File System Structure

File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way. A file System must be able to store the file, locate the file and retrieve the file.

Most of the Operating Systems use layering approach for every task including file systems. Every layer of the file system is responsible for some activities.

The image shown below, elaborates how the file system is divided in different layers, and also the functionality of each layer.



- When an application program asks for a file, the first request is directed to the logical file system. The logical file system contains the Meta data of the file and directory structure. If the application program doesn't have the required permissions of the file then this layer will throw an error. Logical file systems also verify the path to the file.
- Generally, files are divided into various logical blocks. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors. Therefore, in order to store and retrieve the files, the logical blocks need to be mapped to physical blocks. This mapping is done by File organization module. It is also responsible for free space management.
- Once File organization module decided which physical block the application program needs, it passes this information to basic file system. The basic file system is responsible for issuing the commands to I/O control in order to fetch those blocks.
- I/O controls contain the codes by using which it can access hard disk. These codes are known as device drivers. I/O controls are also responsible for handling interrupts.

15) Explain disk organization using diagram.

16) Give the importance of proper time quantum selection in Round Robin CPU Scheduling algorithm. Draw Gantt Chart and Find average waiting time and average turnaround time for following using Round Robin Scheduling (Time quantum of 3 msec) and FCFS scheduling: :

Process	Burst Time(msec)
P1	10
P2	3
P3	5
P4	7

17) What is the producer consumer problem? Provide solution to producer consumer problem using semaphores.

The producer–consumer problem (also known as the bounded-buffer problem) is an example of a multi-process synchronization problem.

The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue.

The producer's job is to generate a piece of data, put it into the buffer and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer) one piece at a time.

The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

The solution for the producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again.

In the same way, the consumer can go to sleep if it finds the buffer to be empty.

The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

The solution can be reached by means of inter-process communication, typically using semaphores.

An inadequate solution could result in a deadlock where both processes are waiting to be awakened. The problem can also be generalized to have multiple producers and consumers.

Problem Statement – We have a buffer of fixed size. A producer can produce an item and can place in the buffer. A consumer can pick items and can consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time consumer should not consume any item. In this problem, buffer is the critical section.
To solve this problem, we need two counting semaphores – Full and Empty. “Full” keeps track of number of items in the buffer at any given time and “Empty” keeps track of number of unoccupied slots.

Initialization of semaphores –
mutex = 1
Full = 0 // Initially, all slots are empty. Thus full slots are 0
Empty = n // All slots are empty initially
Solution for Producer –
do{

//produce an item

wait(empty);

wait(mutex);

//place in buffer

signal(mutex);

signal(full);

}while(true)

When producer produces an item then the value of “empty” is reduced by 1 because one slot will be filled now. The value of mutex is also reduced to prevent consumer to access the buffer. Now, the producer has placed the item and thus the value of “full” is increased by 1. The value of mutex is also increased by 1 because the task of producer has been completed and consumer can access the buffer.

Solution for Consumer –
do{

wait(full);

wait(mutex);

// remove item from buffer

signal(mutex);

signal(empty);

// consumes item

}while(true)

As the consumer is removing an item from buffer, therefore the value of “full” is reduced by 1 and the value is mutex is also reduced so that the producer cannot access the buffer at this moment. Now, the consumer has consumed the item, thus increasing the value of “empty” by 1. The value of mutex is also increased so that producer can access the buffer now.

18) Discuss the operation of translation lookaside buffer(TLB) in terms of memory management

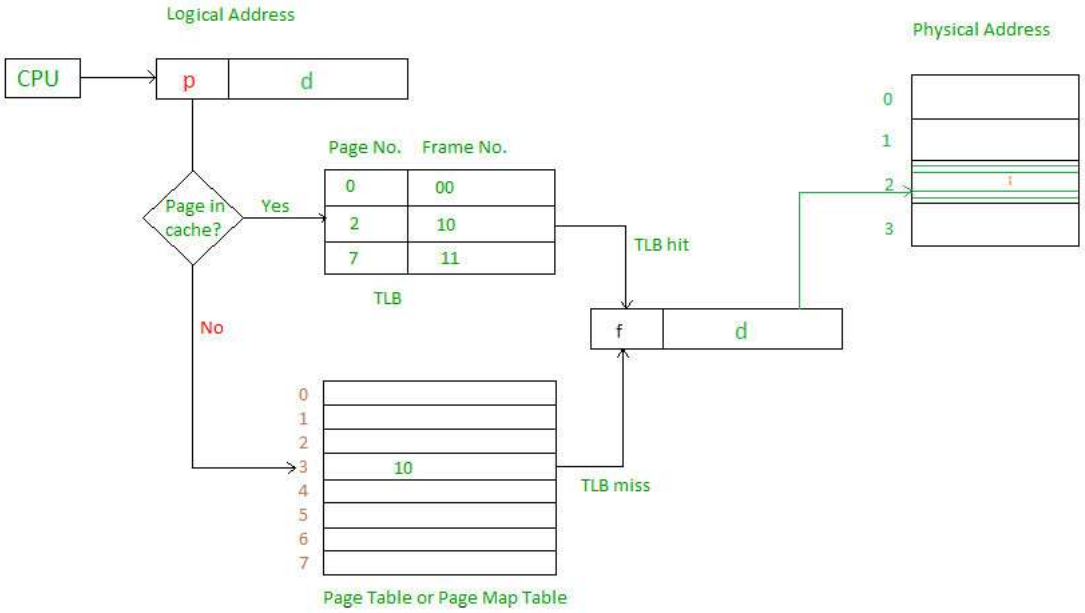
In Operating System (Memory Management Technique : Paging), for each process page table will be created, which will contain Page Table Entry (PTE). This PTE will contain information like frame number (The address of main memory where we want to refer), and some other useful bits (e.g., valid/invalid bit, dirty bit, protection bit etc). This page table entry (PTE) will tell where in the main memory the actual page is residing.
Now the question is where to place the page table, such that overall access time (or reference time) will be less.

The problem initially was to fast access the main memory content based on address generated by CPU (i.e logical/virtual address). Initially, some people thought of using registers to store page table, as they are high-speed memory so access time will be less. The idea used here is, place the page table entries in registers, for each request generated from CPU (virtual address), it will be matched to the appropriate page number of the page table, which will now tell where in the main memory that corresponding page resides. Everything seems right here, but the problem is register size is small (in practical, it can accommodate maximum of 0.5k to 1k page table entries) and process size may be big hence the required page table will also be big (lets say this page table contains 1M entries), so registers may not hold all the PTE's of Page table. So this is not a practical approach.

To overcome this size issue, the entire page table was kept in main memory. but the problem here is two main memory references are required:

1. To find the frame number
2. To go to the address specified by frame number

To overcome this problem a high-speed cache is set up for page table entries called a Translation Lookaside Buffer (TLB). Translation Lookaside Buffer (TLB) is nothing but a special cache used to keep track of recently used transactions. TLB contains page table entries that have been most recently used. Given a virtual address, the processor examines the TLB if a page table entry is present (TLB hit), the frame number is retrieved and the real address is formed. If a page table entry is not found in the TLB (TLB miss), the page number is used as index while processing page table. TLB first checks if the page is already in main memory, if not in main memory a page fault is issued then the TLB is updated to include the new page entry.



Steps in TLB hit:

1. CPU generates virtual (logical) address.
2. It is checked in TLB (present).
3. Corresponding frame number is retrieved, which now tells where in the main memory page lies.

Steps in TLB miss:

1. CPU generates virtual (logical) address.
2. It is checked in TLB (not present).
3. Now the page number is matched to page table residing in main memory (assuming page table contains all PTE).
4. Corresponding frame number is retrieved, which now tells where in the main memory page lies.
5. The TLB is updated with new PTE (if space is not there, one of the replacement technique comes into picture i.e either FIFO, LRU or MFU etc).

Effective memory access time(EMAT) : TLB is used to reduce effective memory access time as it is a high speed associative cache.
EMAT = h*(c+m) + (1-h)*(c+2m)
where, h = hit ratio of TLB
m = Memory access time
c = TLB access time

19) What is an operating system? Describe role of Kernel in operating system

Kernel is central component of an operating system that manages operations of computer and hardware. It basically manages operations of memory and CPU time. It is core component of an operating system. Kernel acts as a bridge between applications and data processing performed at hardware level using inter-process communication and system calls.
Kernel loads first into memory when an operating system is loaded and remains into memory until operating system is shut down again. It is responsible for various tasks such as disk management, task management, and memory management.

It decides which process should be allocated to processor to execute and which process should be kept in main memory to execute. It basically acts as an interface between user applications and hardware. The major aim of kernel is to manage communication between software i.e. user-level applications and hardware i.e., CPU and disk memory.

Objectives of Kernel :

- To establish communication between user level application and hardware.
- To decide state of incoming processes.
- To control disk management.
- To control memory management.
- To control task management.

Types of Kernel :
1. Monolithic Kernel –
It is one of types of kernel where all operating system services operate in kernel space. It has dependencies between systems components. It has huge lines of code which is complex.
Example :

Unix, Linux, Open VMS, XTS-400 etc.

- Advantage :
  It has good performance.
- Disadvantage :
  It has dependencies between system component and lines of code in millions.

2. Micro Kernel –
It is kernel types which has minimalist approach. It has virtual memory and thread scheduling. It is more stable with less services in kernel space. It puts rest in user space.
Example :

Mach, L4, AmigaOS, Minix, K42 etc.

- Advantage :
  It is more stable.

- Disadvantage :
  There are lots of system calls and context switches.

3. Hybrid Kernel –
It is the combination of both monolithic kernel and microkernel. It has speed and design of monolithic kernel and modularity and stability of microkernel.
Example :

Windows NT, Netware, BeOS etc.

- Advantage :
  It combines both monolithic kernel and microkernel.

- Disadvantage :
  It is still similar to monolithic kernel.

4. Exo Kernel –
It is the type of kernel which follows end-to-end principle. It has fewest hardware abstractions as possible. It allocates physical resources to applications.
Example :

Nemesis, ExOS etc.

- Advantage :
  It has fewest hardware abstractions.

- Disadvantage :
  There is more work for application developers.

5. Nano Kernel –
It is the type of kernel that offers hardware abstraction but without system services. Micro Kernel also does not have system services therefore the Micro Kernel and Nano Kernel have become analogous.

Example :

EROS etc.

- Advantage :
  It offers hardware abstractions without system services.

- Disadvantage :
  It is quite same as Micro kernel hence it is less used.

20) Describe criteria in CPU scheduling

Different CPU scheduling algorithms have different properties and the choice of a particular algorithm depends on the various factors. Many criteria have been suggested for comparing CPU scheduling algorithms.
The criteria include the following:

1. CPU utilisation –
   The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible. Theoretically, CPU utilisation can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the load upon the system.

2. Throughput –
   A measure of the work done by CPU is the number of processes being executed and completed per unit time. This is called throughput. The throughput may vary depending upon the length or duration of processes.

3. Turnaround time –
   For a particular process, an important criteria is how long it takes to execute that process. The time elapsed from the time of submission of a process to the time of completion is known as the turnaround time. Turn-around time is the sum of times spent waiting to get into memory, waiting in ready queue, executing in CPU, and waiting for I/O.

4. Waiting time –
   A scheduling algorithm does not affect the time required to complete the process once it starts execution. It only affects the waiting time of a process i.e. time spent by a process waiting in the ready queue.

5. Response time –
   In an interactive system, turn-around time is not the best criteria. A process may produce some output fairly early and continue computing new results while previous results are being output to the user. Thus another criteria is the time taken from submission of the process of request until the first response is produced. This measure is called response time.

There are various CPU Scheduling algorithms such as-

- First Come First Served (FCFS)
- Shortest Job First (SJF)
- Longest Job First (LJF)
- Priority Scheduling
- Round Robin (RR)
- Shortest Remaining Time First (SRTF)
- Longest Remaining Time First (LRTF)

21) What is the Dining Philosophers problem?Give one solution.

The dining philosophers problem states that there are 5 philosophers sharing a circular table and they eat and think alternatively. There is a bowl of rice for each of the philosophers and 5 chopsticks. A philosopher needs both their right and left chopstick to eat. A hungry philosopher may only eat if there are both chopsticks available.Otherwise a philosopher puts down their chopstick and begin thinking again.

The dining philosopher is a classic synchronization problem as it demonstrates a large class of concurrency control problems.

Solution of Dining Philosophers Problem

A solution of the Dining Philosophers Problem is to use a semaphore to represent a chopstick. A chopstick can be picked up by executing a wait operation on the semaphore and released by executing a signal semaphore.

The structure of the chopstick is shown below −

```
semaphore chopstick [5];
```

Initially the elements of the chopstick are initialized to 1 as the chopsticks are on the table and not picked up by a philosopher.

The structure of a random philosopher i is given as follows −

```
do {
   wait( chopstick[i] );
   wait( chopstick[ (i+1) % 5] );
   . .
   . EATING THE RICE
   .
   signal( chopstick[i] );
   signal( chopstick[ (i+1) % 5] );
   .
   . THINKING
   .
} while(1);
```

In the above structure, first wait operation is performed on chopstick[i] and chopstick[(i+1) % 5]. This means that the philosopher i has picked up the chopsticks on his sides. Then the eating function is performed.

After that, signal operation is performed on chopstick[i] and chopstick[(i+1) % 5]. This means that the philosopher i has eaten and put down the chopsticks on his sides. Then the philosopher goes back to thinking.

Difficulty with the solution

The above solution makes sure that no two neighboring philosophers can eat at the same time. But this solution can lead to a deadlock. This may happen if all the philosophers pick their left chopstick simultaneously. Then none of them can eat and deadlock occurs.

Some of the ways to avoid deadlock are as follows −

- There should be at most four philosophers on the table.
- An even philosopher should pick the right chopstick and then the left chopstick while an odd philosopher should pick the left chopstick and then the right chopstick.
- A philosopher should only be allowed to pick their chopstick if both are available at the same time.

22) explain the problem of thrashing in detail

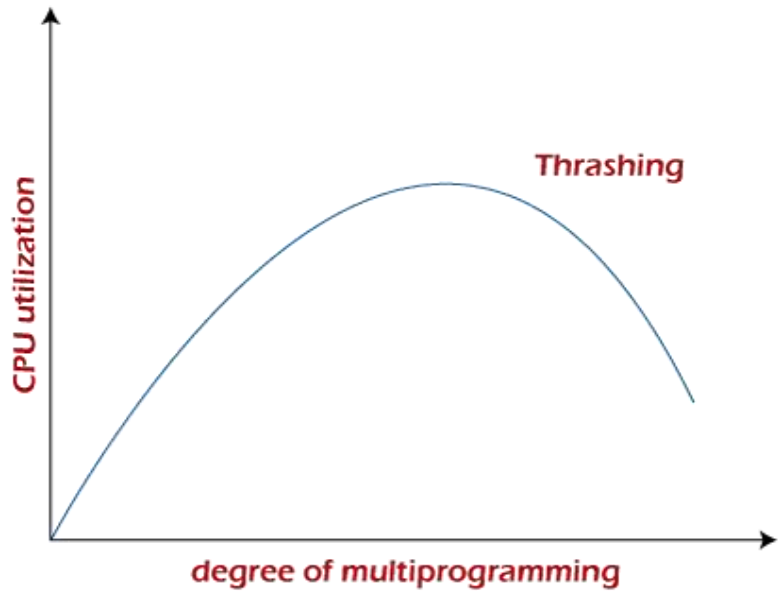
In computer science, *thrash* is the poor performance of a virtual memory (or paging) system when the same pages are being loaded repeatedly due to a lack of main memory to keep them in memory. Depending on the configuration and algorithm, the actual throughput of a system can degrade by multiple orders of magnitude.

In computer science, *thrashing* occurs when a computer's virtual memory resources are overused, leading to a constant state of paging and page faults, inhibiting most application-level processing. It causes the performance of the computer to degrade or collapse. The situation can continue indefinitely until the user closes some running applications or the active processes free up additional virtual memory resources.

To know more clearly about thrashing, first, we need to know about page fault and swapping.

- Page fault: We know every program is divided into some pages. A page fault occurs when a program attempts to access data or code in its address space but is not currently located in the system RAM.
- Swapping: Whenever a page fault happens, the operating system will try to fetch that page from secondary memory and try to swap it with one of the pages in RAM. This process is called swapping.

Thrashing is when the page fault and swapping happens very frequently at a higher rate, and then the operating system has to spend more time swapping these pages. This state in the operating system is known as thrashing. Because of thrashing, the CPU utilization is going to be reduced or negligible.



The basic concept involved is that if a process is allocated too few frames, then there will be too many and too frequent page faults. As a result, no valuable work would be done by the CPU, and the CPU utilization would fall drastically.

The long-term scheduler would then try to improve the CPU utilization by loading some more processes into the memory, thereby increasing the degree of multiprogramming. Unfortunately, this would result in a further decrease in the CPU utilization, triggering a chained reaction of higher page faults followed by an increase in the degree of multiprogramming, called thrashing.

Algorithms during Thrashing

Whenever thrashing starts, the operating system tries to apply either the Global page replacement Algorithm or the Local page replacement algorithm.

1. Global Page Replacement

Since global page replacement can bring any page, it tries to bring more pages whenever thrashing is found. But what actually will happen is that no process gets enough frames, and as a result, the thrashing will increase more and more. Therefore, the global page replacement algorithm is not suitable when thrashing happens.

2. Local Page Replacement

Unlike the global page replacement algorithm, local page replacement will select pages which only belong to that process. So there is a chance to reduce the thrashing. But it is proven that there are many disadvantages if we use local page replacement. Therefore, local page replacement is just an alternative to global page replacement in a thrashing scenario.

Causes of Thrashing

Programs or workloads may cause thrashing, and it results in severe performance problems, such as:

- If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new system. A global page replacement algorithm is used. The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming.
- CPU utilization is plotted against the degree of multiprogramming.
- As the degree of multiprogramming increases, CPU utilization also increases.
- If the degree of multiprogramming is increased further, thrashing sets in, and CPU utilization drops sharply.
- So, at this point, to increase CPU utilization and to stop thrashing, we must decrease the degree of multiprogramming.

How to Eliminate Thrashing

Thrashing has some negative impacts on hard drive health and system performance. Therefore, it is necessary to take some actions to avoid it. To resolve the problem of thrashing, here are the following methods, such as:

- Adjust the swap file size:If the system swap file is not configured correctly, disk thrashing can also happen to you.
- Increase the amount of RAM: As insufficient memory can cause disk thrashing, one solution is to add more RAM to the laptop. With more memory, your computer can handle tasks easily and don't have to work excessively. Generally, it is the best long-term solution.
- Decrease the number of applications running on the computer: If there are too many applications running in the background, your system resource will consume a lot. And the remaining system resource is slow that can result in thrashing. So while closing, some applications will release some resources so that you can avoid thrashing to some extent.
- Replace programs: Replace those programs that are heavy memory occupied with equivalents that use less memory.

Techniques to Prevent Thrashing

The Local Page replacement is better than the Global Page replacement, but local page replacement has many disadvantages, so it is sometimes not helpful. Therefore below are some other techniques that are used to handle thrashing:

1. Locality Model

A locality is a set of pages that are actively used together. The locality model states that as a process executes, it moves from one locality to another. Thus, a program is generally composed of several different localities which may overlap.

For example, when a function is called, it defines a new locality where memory references are made to the function call instructions, local and global variables, etc. Similarly, when the function is exited, the process leaves this locality.

2. Working-Set Model

This model is based on the above-stated concept of the Locality Model.

The basic principle states that if we allocate enough frames to a process to accommodate its current locality, it will only fault whenever it moves to some new locality. But if the allocated frames are lesser than the size of the current locality, the process is bound to thrash.

According to this model, based on parameter A, the working set is defined as the set of pages in the most recent 'A' page references. Hence, all the actively used pages would always end up being a part of the working set.

The accuracy of the working set is dependent on the value of parameter A. If A is too large, then working sets may overlap. On the other hand, for smaller values of A, the locality might not be covered entirely.

If D is the total demand for frames and WSS_i is the working set size for process i,

$D = \sum WSS_i$

Now, if 'm' is the number of frames available in the memory, there are two possibilities:

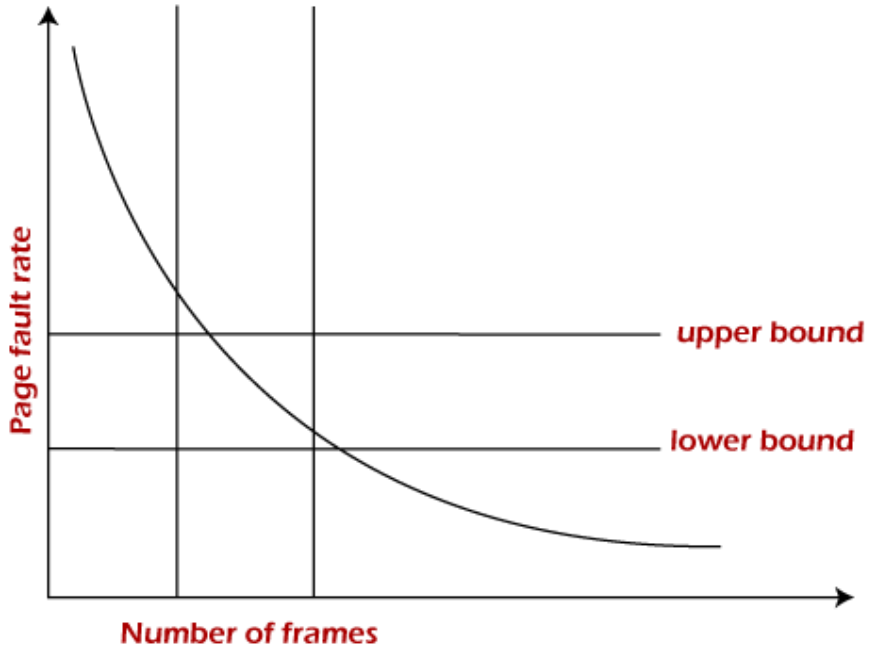
- D>m, i.e., total demand exceeds the number of frames, then thrashing will occur as some processes would not get enough frames.
- D<=m, then there would be no thrashing.

If there are enough extra frames, then some more processes can be loaded into the memory. On the other hand, if the summation of working set sizes exceeds the frames' availability, some of the processes have to be suspended (swapped out of memory).

This technique prevents thrashing along with ensuring the highest degree of multiprogramming possible. Thus, it optimizes CPU utilization.

3. Page Fault Frequency

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



The problem associated with thrashing is the high page fault rate, and thus, the concept here is to control the page fault rate.

If the page fault rate is too high, it indicates that the process has too few frames allocated to it. On the contrary, a low page fault rate indicates that the process has too many frames.

Upper and lower limits can be established on the desired page fault rate, as shown in the diagram.

If the page fault rate falls below the lower limit, frames can be removed from the process. Similarly, if the page faults rate exceeds the upper limit, more frames can be allocated to the process.

In other words, the graphical state of the system should be kept limited to the rectangular region formed in the given diagram.

If the page fault rate is high with no free frames, some of the processes can be suspended and allocated to them can be reallocated to other processes. The suspended processes can restart later.

23) Describe various requirements for file management system

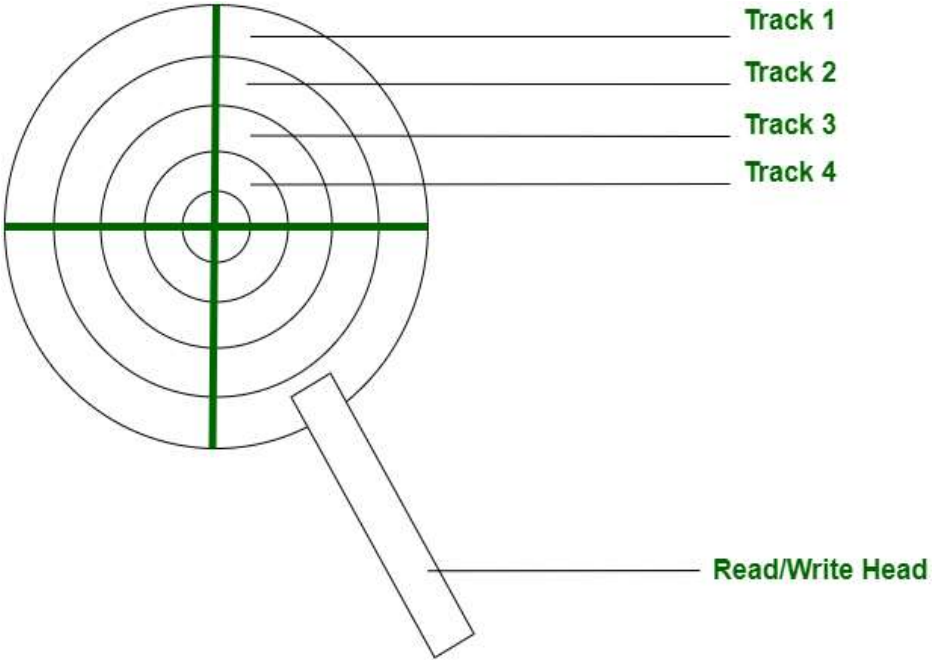
24) Define following terms in relation with disk management: Rotational delay, Transfer rate, Access time, Seek time, Cylinder.

Rotational Delay
The rotational delay is the expected minimum time (in milliseconds) that it takes the CPU to complete a data transfer and initiate a new data transfer on the same disk cylinder. The default delay is zero, as delay-based calculations are not effective when combined with modern on-disk caches.
When writing a file, the UFS allocation routines try to position new blocks on the same disk cylinder as the previous block in the same file. The allocation routines also try to optimally position new blocks within tracks to minimize the disk rotation needed to access the new blocks.
To position file blocks so they are “rotationally well-behaved,” the allocation routines must know how fast the CPU can service transfers and how long it takes the disk to skip over a block. By using options to the mkfs command, you can indicate how fast the disk rotates and how many disk blocks (sectors) it has per track. The allocation routines use this information to figure out how many milliseconds are needed to skip a disk block. Then using the expected transfer time (rotational delay), the allocation routines can position or place blocks so that the next block is just coming under the disk head when the system is ready to read the block.

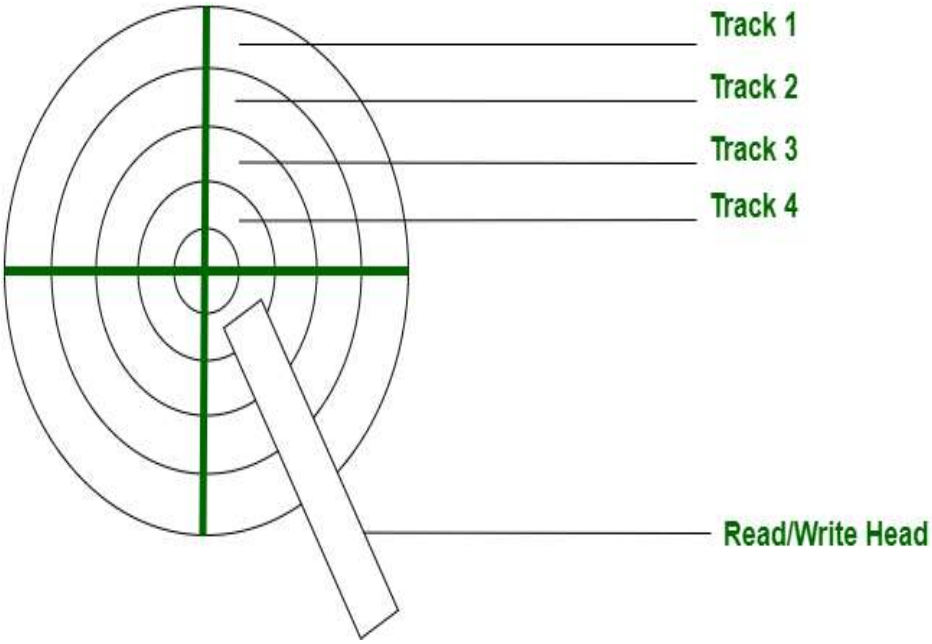
Transfer rate
Transfer rates are measured in bits or bytes per second. Transfer rate and "data rate" are often used synonymously; however, technically, the transfer rate is often more than the data rate, because control signals may be sent along with data, which decreases the rate of actual data being sent.

Access time
Access time is the time from the start of one storage device access to the time when the next access can be started. Access time consists of latency (the overhead of getting to the right place on the device and preparing to access it) and transfer time.

Seek Time:
A disk is divided into many circular tracks. Seek Time is defined as the time required by the read/write head to move from one track to another.
Example,
Consider the following diagram, the read/write head is currently on track 1.



Now, on the next read/write request, we may want to read data from Track 4, in this case, our read/write head will move to track 4. The time it will take to reach track 4 is the seek time.



Cylinder
What is the cylinder in hard disk?

A cylinder is any set of all of tracks of equal diameter in a hard disk drive (HDD). It can be visualized as a single, imaginary, circle that cuts through all of the platters (and both sides of each platter) in the drive

25) With the help of diagrams explain different multithreading models
26) Explain Banker’s algorithm for deadlock avoidance. How is it different from deadlock detection?

It is a banker algorithm used to avoid deadlock and allocate resources safely to each process in the computer system. The 'S-State' examines all possible tests or activities before deciding whether the allocation should be allowed to each process. It also helps the operating system to successfully share the resources between all the processes. The banker's algorithm is named because it checks whether a person should be sanctioned a loan amount or not to help the bank system safely simulate allocation resources. In this section, we will learn the Banker's Algorithm in detail. Also, we will solve problems based on the Banker's Algorithm. To understand the Banker's Algorithm first we will see a real word example of it.

Suppose the number of account holders in a particular bank is 'n', and the total money in a bank is 'T'. If an account holder applies for a loan; first, the bank subtracts the loan amount from full cash and then estimates the cash difference is greater than T to approve the loan amount. These steps are taken because if another person applies for a loan or withdraws some amount from the bank, it helps the bank manage and operate all things without any restriction in the functionality of the banking system.

Similarly, it works in an operating system

. When a new process is created in a computer system, the process must provide all types of information to the operating system
like upcoming processes, requests for their resources, counting them, and delays. Based on these criteria, the operating system decides which process sequence should be executed or waited so that no deadlock occurs in a system. Therefore, it is also known as deadlock avoidance algorithm or deadlock detection in the operating system.

Advantages

Following are the essential characteristics of the Banker's algorithm:

1. It contains various resources that meet the requirements of each process.
2. Each process should provide information to the operating system for upcoming resource requests, the number of resources, and how long the resources will be held.
3. It helps the operating system manage and control process requests for each type of resource in the computer system.
4. The algorithm has a Max resource attribute that represents indicates each process can hold the maximum number of resources in a system.

Disadvantages

1. It requires a fixed number of processes, and no additional processes can be started in the system while executing the process.
2. The algorithm does no longer allows the processes to exchange its maximum needs while processing its tasks.
3. Each process has to know and state their maximum resource requirement in advance for the system.
4. The number of resource requests can be granted in a finite time, but the time limit for allocating the resources is one year.

27) Apply FIFO,LRU,OPTIMAL(OPT) page replacement algorithms on the following page sequence
1,2,3,4,5,1,4,2,3,4
and calculate number page of HIT and MISS occurred