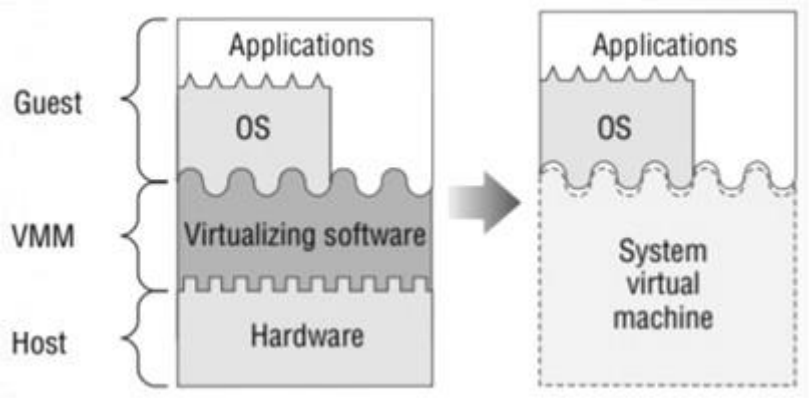
Q. 1 Solve Any Two of the following. (This is just a sample instruction)

A) Define a virtual machine with neat diagram. Describe the concept and working of JVM. Explain what are the benefits of a VM?

ANS:-



A virtual machine (VM) is a software emulation of a physical computer system that allows multiple operating systems to run on a single physical machine. It provides an isolated environment where software can be executed independently of the underlying hardware and operating system. A common type of virtual machine is the Java Virtual Machine (JVM), which is specifically designed to execute Java bytecode.

Here is a neat diagram representing the concept of a virtual machine:

```
+----------------------+
|   Virtual Machine |
+----------------------+
|  Application Code |
|-----------------------|
|    JVM Runtime    |
|----------------------|
|     Hardware         |
+----------------------+
```

In this diagram, the virtual machine acts as an intermediate layer between the application code and the hardware. The application code, written in a high-level language like Java, is compiled into bytecode. The JVM runtime interprets and executes this bytecode, providing a platform-independent execution environment. The JVM interacts with the underlying hardware and operating system to perform tasks such as memory management, thread scheduling, and I/O operations.

The JVM consists of several components:
1. Class Loader: Responsible for loading Java class files into the JVM.
2. Bytecode Verifier: Ensures that the bytecode is valid and does not violate any security constraints.
3. Just-In-Time (JIT) Compiler: Translates frequently executed bytecode into machine code for faster execution.
4. Garbage Collector: Manages memory by automatically deallocating objects that are no longer in use.
5. Execution Engine: Interprets and executes the bytecode or the compiled machine code generated by the JIT compiler.

Now let's discuss the benefits of a virtual machine:
1. Platform Independence: One of the key benefits of a virtual machine is platform independence. The JVM provides a consistent environment for executing Java code across different operating systems and hardware architectures. Developers can write code once and run it on any device or platform that has a compatible JVM.
2. Memory Management: The virtual machine manages memory allocation and deallocation, relieving the developer from manual memory management tasks. The JVM's garbage collector automatically identifies and frees up memory occupied by objects that are no longer needed, reducing the risk of memory leaks and improving overall application stability.
3. Security: The JVM's bytecode verifier ensures that the code running inside the virtual machine is safe and does not violate any security constraints. It prevents unauthorized access to system resources and provides a sandboxed environment for executing untrusted code.

4. Performance Optimization: The JVM's JIT compiler dynamically analyzes the bytecode at runtime and compiles frequently executed portions into machine code. This optimization technique improves the execution speed of Java applications, making them comparable to natively compiled languages.
5. Portability: Virtual machines enable the easy portability of applications. Once an application is developed and tested on a particular JVM, it can be deployed on any system that has a compatible JVM installed. This simplifies the deployment process and allows software to be distributed more efficiently.
Overall, virtual machines like the JVM provide a powerful abstraction layer that simplifies software development, enhances security, and promotes cross-platform compatibility, making them widely used in various programming ecosystems.

B) What is the purpose of command interpreter? Why is it usually separate from the Kernel?

Ans:-
The purpose of a command interpreter, often referred to as a shell or command line interpreter, is to act as an interface between the user and the operating system. It allows users to interact with the system by accepting and executing commands entered through a command line interface (CLI).
Here are a few key functions and purposes of a command interpreter:
1. Command Execution: The command interpreter receives commands entered by the user and executes them by invoking the appropriate system programs or utilities. It interprets the commands, locates the corresponding executable files, and manages the execution process.
2. Input/Output Redirection: The command interpreter provides mechanisms to redirect the input and output of commands. Users can redirect the input from a file or another command, and redirect the output to a file or another command. This flexibility allows for the composition of complex commands and the efficient processing of large volumes of data.
3. Shell Scripting: The command interpreter often supports scripting capabilities, allowing users to write and execute scripts containing a series of commands. Shell scripts can automate repetitive tasks, combine multiple commands into a single script, and enable the creation of custom command sequences for specific purposes.
4. Environment Management: The command interpreter manages the environment in which commands are executed. It allows users to set and modify environment variables, which are used to configure the behavior of various system utilities and programs. Environment variables can store information such as search paths, default settings, and user preferences.

Now, as for why the command interpreter is usually separate from the kernel:

1. Flexibility and Customization: By separating the command interpreter from the kernel, the operating system becomes more flexible and customizable. Different command interpreters, often referred to as shells, can coexist and provide users with various features, syntaxes, and scripting capabilities. Users can choose the shell that best suits their needs and preferences, and even switch between different shells without affecting the underlying kernel.
2. Modularity and Security: Separating the command interpreter from the kernel promotes modularity and security. The kernel, which is the core of the operating system, deals with low-level system operations and hardware interactions. By keeping the command interpreter separate, the kernel's complexity is reduced, making it easier to develop, test, and maintain. It also helps in isolating potential vulnerabilities in the command interpreter from the critical kernel components, enhancing system security.
3. Portability: The separation of the command interpreter from the kernel allows for better portability. Different operating systems can have their own command interpreters that conform to a specific standard, such as POSIX, while utilizing the same kernel. This enables cross-platform compatibility and makes it easier to migrate or port applications between different operating systems.

Overall, separating the command interpreter from the kernel provides flexibility, customization, security, and portability benefits, making it a common design choice in operating systems.

C) Describe major activities of an operating system in regard to:
1) Process management 3) Main Memory management
2) File management 4) Secondary storage management

Ans:-
1) Process Management:
Process management is one of the key activities of an operating system, responsible for managing and executing processes (also known as tasks or programs). The major activities involved in process management are:

- Process Creation: The operating system creates new processes either in response to a user request or as part of its own

system initialization. This involves allocating necessary resources such as memory space, file descriptors, and other required data structures.

- Process Scheduling: The operating system decides which process gets to use the CPU and for how long. It employs scheduling algorithms to determine the order and duration of process execution, aiming to achieve fairness, efficiency, and optimal resource utilization.
- Process Execution: The operating system allocates CPU time to processes and manages the context switching between them. It ensures that each process gets its turn to execute instructions, suspending and resuming processes as necessary.
- Process Communication and Synchronization: The operating system provides mechanisms for processes to communicate and synchronize with each other. This includes inter-process communication (IPC) methods such as pipes, shared memory, and message queues, as well as synchronization primitives like semaphores, locks, and barriers.
- Process Termination: The operating system handles the termination of processes, reclaiming their resources and releasing them back to the system. It may involve closing files, freeing memory, and updating process control blocks.

2) File Management:

File management involves the organization, storage, and retrieval of files on a computer system. The major activities of file management include:

- File Creation and Deletion: The operating system provides commands and APIs to create and delete files. It manages the allocation of storage space for new files and ensures the removal of file entries from the file system when files are deleted.
- File Opening and Closing: Processes can open files to read from or write to them. The operating system maintains information about open files, such as file pointers and access permissions. It ensures that multiple processes can access files concurrently while managing conflicts and maintaining data integrity.
- File Reading and Writing: The operating system provides mechanisms for processes to read data from files or write data to files. It handles data buffering, caching, and buffering to optimize I/O operations and improve overall system performance.
- File Access Control and Security: The operating system enforces access control policies to protect files from unauthorized access. It manages file permissions, ownership, and access rights, allowing or denying access based on user privileges and security policies.
- File System Maintenance: The operating system performs various maintenance tasks related to the file system. This includes disk space management, file system integrity checks, error handling, and recovery procedures in case of system failures or data corruption.

3) Main Memory Management:

Main memory management involves the efficient allocation and management of primary memory (RAM) resources. The major activities of main memory management include:

- Memory Allocation: The operating system allocates memory to processes, ensuring that each process gets the necessary memory space for its execution. It tracks free and allocated memory blocks and manages the allocation and deallocation of memory dynamically.
- Memory Protection: The operating system enforces memory protection mechanisms to prevent processes from accessing memory areas that they should not. It assigns appropriate memory protection levels to different processes, ensuring data isolation and preventing unauthorized access or modification.
- Memory Mapping: The operating system enables memory mapping, allowing processes to access files or devices as if they were reading from or writing to memory. This facilitates efficient I/O operations and simplifies file access.
- Memory Paging and Swapping: The operating system may use techniques such as paging and swapping to manage memory more effectively. It can move pages of memory between RAM and secondary storage (disk) to free up space, balance memory usage, and prioritize active processes.
- Memory Fragmentation Management: The operating system handles memory fragmentation, which can occur due to the allocation and deallocation of memory blocks. It employs techniques like compaction or memory compaction to reduce fragmentation and optimize memory utilization.

4) Secondary Storage Management:

Secondary storage management involves managing and controlling storage devices such as hard disks, solid-state drives (SSDs), and other non-volatile storage media. The major activities of secondary storage management include:

- File System Creation and Formatting: The operating system provides utilities to create file systems on secondary storage devices. It initializes the storage media, sets up data structures for file organization, and prepares the device for storing files.
- File Allocation and Organization: The operating system manages the allocation and organization of files on secondary storage. It keeps track of available space, allocates storage blocks to files, and maintains file system metadata such as

directories, file attributes, and file allocation tables.

- Disk Scheduling: The operating system employs disk scheduling algorithms to optimize the order in which disk I/O requests are processed. It aims to minimize disk seek time, reduce latency, and maximize the overall throughput of disk operations.
- Disk Space Management: The operating system tracks the allocation and utilization of disk space. It handles disk space allocation for files, performs space management operations such as compaction or garbage collection, and manages free space for future file storage.
- Disk Caching: The operating system may utilize disk caching techniques to improve performance by keeping frequently accessed data in the main memory (RAM). This reduces disk I/O operations and speeds up file access.
- Disk Error Handling and Recovery: The operating system provides error handling mechanisms for detecting and recovering from disk failures or data corruption. It performs error checks, employs redundancy techniques such as RAID, and implements recovery procedures to ensure data integrity and system reliability.

Overall, these activities of an operating system related to process management, file management, main memory management, and secondary storage management are crucial for the efficient and reliable operation of a computer system.

Q.2 Solve Any Two of the following. (This is just a sample instruction)
A) Consider the following data with burst time given in milliseconds: Process
PROCESS:P1
BURST TIME:10
PRIORITY 3

PROCESS:P2
BURST TIME:1
PRIORITY 1

PROCESS:P3
BURST TIME:2
PRIORITY 3

PROCESS:P4
BURST TIME:1
PRIORITY:4

PROCESS:P5
BURST TIME:5
PRIORITY :2

The process has arrived in the order P1, P2, P3, P4, P5.
i) Draw Gantt charts for the execution of these processes using FCFS, SJF, non-preemptive Priority and RR (quantum=2) scheduling.
Ans:-

ii) What is turnaround time and waiting time of each process for each of the scheduling algorithm.
Ans:-
To calculate the turnaround time and waiting time for each process, we need to consider the completion time, arrival time, and burst time of each process. I'll calculate the values for each scheduling algorithm:

1. FCFS (First-Come, First-Served) Scheduling:
Completion Time: P1 - 10, P2 - 11, P3 - 13, P4 - 14, P5 - 19

Turnaround Time:
P1: 10 - 0 = 10
P2: 11 - 0 = 11
P3: 13 - 0 = 13

P4: 14 - 0 = 14
P5: 19 - 0 = 19
Waiting Time:
P1: 10 - 10 = 0
P2: 11 - 1 = 10
P3: 13 - 2 = 11
P4: 14 - 1 = 13
P5: 19 - 5 = 14

2. SJF (Shortest Job First) Scheduling:
Completion Time: P2 - 1, P4 - 2, P3 - 4, P5 - 9, P1 - 19

Turnaround Time:
P1: 19 - 0 = 19
P2: 1 - 0 = 1
P3: 4 - 0 = 4
P4: 2 - 0 = 2
P5: 9 - 0 = 9
Waiting Time:
P1: 19 - 10 = 9
P2: 1 - 1 = 0
P3: 4 - 2 = 2
P4: 2 - 1 = 1
P5: 9 - 5 = 4

3. Non-preemptive Priority Scheduling:
Completion Time: P2 - 1, P5 - 6, P1 - 16, P3 - 18, P4 - 19

Turnaround Time:
P1: 16 - 0 = 16
P2: 1 - 0 = 1
P3: 18 - 0 = 18
P4: 19 - 0 = 19
P5: 6 - 0 = 6
Waiting Time:
P1: 16 - 10 = 6
P2: 1 - 1 = 0
P3: 18 - 2 = 16
P4: 19 - 1 = 18
P5: 6 - 5 = 1

4. RR (Round Robin) Scheduling with a quantum of 2:
Completion Time: P1 - 19, P2 - 3, P3 - 15, P4 - 17, P5 - 19

Turnaround Time:
P1: 19 - 0 = 19
P2: 3 - 0 = 3
P3: 15 - 0 = 15
P4: 17 - 0 = 17
P5: 19 - 0 = 19
Waiting Time:
P1: 19 - 10 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 = 1
P2: 3 - 1 = 2
P3: 15 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 = 5
P4: 17 - 1 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 = 0
P5: 19 - 5 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 = 1

Turnaround time is the total time taken from the arrival of the process until its completion. Waiting time is the total time a process has to wait in the ready queue before it starts execution.

B) What are co-operating processes? Describe the mechanism of inter process communication using shared memory and massage passing

Ans:-

Cooperating processes refer to processes in an operating system that can communicate and synchronize with each other to achieve a common goal or perform a specific task. These processes may run concurrently or sequentially, but they interact and share information to accomplish a task more efficiently or solve a complex problem.

Interprocess communication (IPC) is a mechanism that allows cooperating processes to exchange data and synchronize their actions. There are two commonly used methods for IPC: shared memory and message passing.

1. Shared Memory:
In shared memory IPC, processes communicate by accessing shared memory regions that are created and maintained by the operating system. The steps involved in using shared memory for IPC are as follows:

- Create a shared memory region: The operating system provides functions or system calls to create a shared memory segment and allocate memory in the address space of multiple processes.
- Attach shared memory: Each process that wants to access the shared memory needs to attach (map) the shared memory segment into its own address space. This allows all processes to share the same memory region.
- Read and write data: Processes can read from and write to the shared memory region as needed. They can directly access and modify the data in the shared memory.
- Synchronization: Since multiple processes can access the shared memory simultaneously, synchronization mechanisms like semaphores, mutexes, or other forms of interprocess synchronization are often used to ensure that processes access the shared memory in a coordinated and controlled manner.
- Detach and deallocate: When a process no longer needs to access the shared memory, it can detach (unmap) the shared memory segment from its address space. When all processes have detached, the operating system can deallocate the shared memory region.

2. Message Passing:
Message passing IPC involves processes communicating by explicitly sending and receiving messages. The steps involved in message passing IPC are as follows:

- Send message: A process sends a message to another process by using a system call or a communication library function. The message can contain data, instructions, or requests.
- Receive message: The receiving process waits for incoming messages using a system call or a library function. When a message arrives, the receiving process can extract the data or instructions from the message.
- Buffering: Messages may be buffered by the operating system until the receiving process is ready to receive them. The buffering can be implemented using various techniques like queues or mailboxes.
- Synchronization: Message passing can be synchronous or asynchronous. In synchronous message passing, the sender is blocked until the message is received by the receiver. In asynchronous message passing, the sender continues its execution immediately after sending the message, and the receiver may receive the message at a later time.
- Communication mechanisms: Message passing can be implemented using different mechanisms such as direct or indirect communication. In direct communication, processes explicitly name the recipient or sender of the message. In indirect communication, communication is done through shared data structures like mailboxes, ports, or message queues.

Both shared memory and message passing have their advantages and use cases. Shared memory is typically faster because it avoids the overhead of message copying. Message passing, on the other hand, provides a more structured and controlled communication mechanism that allows processes to communicate even if they are not executing concurrently or running on the same machine. The choice between shared memory and message passing depends on the requirements of the specific application and the desired communication semantics.

C) Suppose the following jobs arrive for processing at the times indicated, each job will run the listed amount of time.

Job	arrival time	burst time
1	0.0	8
2	0.4	4
3	1.0	1

i) Give a Gantt chart illustrating the execution of these jobs using the non- preemptive FCFS and SJF scheduling algorithms.
ii) What is turnaround time and waiting time of each job for the above algorithms?
Ans:-

Q. 3 Solve Any Two of the following. (This is just a sample instruction)

A) Examine banker's algorithm after applying to the example given below. A system has 5 processes, P1, P2, P3, P4 and P5. There are 3 types of resources R1, R2 and R3. there are 10 instances of R1, 5 instances of R2 and 7 instances of R3. At time T0, the situation is as follows;

Process	Allocation			Maximum		
	R1	R2	R3	R1	R2	R3
P1	0	1	0	7	5	3
P2	2	0	0	3	2	2
P3	3	0	2	9	0	2
P4	2	1	1	2	2	2
P5	0	0	2	4	3	3

Is the system in a safe state at time T0?

Suppose now at time T1, process P2 requests one additional instance of resource type R1, is the system in a safe state?

Ans:-

B) Why is deadlock state more critical than starvation? Describe resource allocation graph with a deadlock, also explain resource allocation graph with a cycle but no deadlock.

Ans:-

Deadlock is considered more critical than starvation because deadlock represents a state where multiple processes are waiting for each other to release resources, resulting in a situation where none of the processes can proceed. In a deadlock, the system reaches a standstill, and the affected processes cannot make any progress. On the other hand, starvation refers to a situation where a process is indefinitely delayed or deprived of resources, but the system as a whole can still function.

Here's a description of a resource allocation graph with a deadlock:

A resource allocation graph is used to represent the allocation of resources and the relationships between processes and resources in a system. It consists of two types of nodes: process nodes and resource nodes, and edges that represent the requests and assignments of resources.

In a deadlock scenario, there will be a circular dependency or cycle in the resource allocation graph. Let's consider a simple example:

Assume we have three processes, P1, P2, and P3, and three resources, R1, R2, and R3. The following allocations and requests exist:

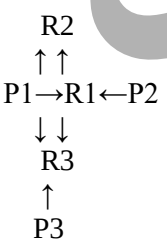
P1 has allocated R1 and is requesting R2.

P2 has allocated R2 and is requesting R3.

P3 has allocated R3 and is requesting R1.

In the resource allocation graph, we represent processes as circles and resources as rectangles. We draw edges to represent the allocation and request relationships.

Resource Allocation Graph with a Deadlock:



In this graph, we can see that there is a cycle (P1 → R2 → P2 → R3 → P3 → R1 → P1), which indicates a deadlock. Each process is holding a resource that the next process in the cycle is waiting for. None of the processes can proceed, resulting in a deadlock state.

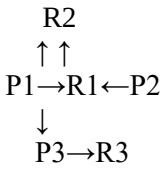
Now let's consider a resource allocation graph with a cycle but no deadlock:

Assume we have three processes, P1, P2, and P3, and three resources, R1, R2, and R3. The following allocations and requests exist:

P1 has allocated R1 and is requesting R2.

P2 has allocated R2 and is requesting R1.
P3 has allocated R3 and is requesting R2.

Resource Allocation Graph with a Cycle but No Deadlock:



In this graph, there is a cycle (P1 → R2 → P2 → R1 → P1), but no deadlock occurs. Each process is waiting for a resource, but it can still progress and complete its execution. The cycle doesn't lead to a situation where all processes are indefinitely blocked.
In summary, a deadlock is more critical than starvation because it represents a state where processes are unable to make progress due to circular dependencies in resource allocation. A resource allocation graph with a deadlock has a cycle that forms a circular wait condition, while a resource allocation graph with a cycle but no deadlock allows processes to eventually proceed and complete their execution.

C) Describe the bounded-buffer Producer-Consumer problem and give a solution for the same using semaphores. Write the structure of Producer and Consumer processes.
Ans:-

The bounded-buffer Producer-Consumer problem is a classic synchronization problem where there is a shared buffer or queue of fixed size that is accessed by two types of processes: producers and consumers. Producers add items to the buffer, while consumers remove items from the buffer. The goal is to ensure that producers do not try to add items to a full buffer, and consumers do not try to remove items from an empty buffer. The solution involves coordinating the actions of producers and consumers using synchronization mechanisms like semaphores.

Here is a solution for the bounded-buffer Producer-Consumer problem using semaphores:
Shared Variables:
- Buffer: Shared buffer or queue of fixed size
- Empty: Semaphore to track the number of empty slots in the buffer
- Full: Semaphore to track the number of filled slots in the buffer
- Mutex: Semaphore to provide mutual exclusion for accessing the buffer

Structure of the Producer process:
1. Initialize the item to be produced.
2. Repeat the following steps:
a. Wait on the Empty semaphore to ensure there is at least one empty slot in the buffer.
b. Wait on the Mutex semaphore to acquire exclusive access to the buffer.
c. Add the item to the buffer.
d. Signal the Mutex semaphore to release access to the buffer.
e. Signal the Full semaphore to indicate that there is one more item in the buffer.
f. Produce a new item (if required).

Structure of the Consumer process:
1. Repeat the following steps:
a. Wait on the Full semaphore to ensure there is at least one filled slot in the buffer.
b. Wait on the Mutex semaphore to acquire exclusive access to the buffer.
c. Remove an item from the buffer.
d. Signal the Mutex semaphore to release access to the buffer.
e. Signal the Empty semaphore to indicate that there is one more empty slot in the buffer.
f. Consume the item (process or use it).

Q.4 Solve Any Two of the following. (This is just a sample instruction)
A) Given memory partitions of 150 K, 250 K, 500 K, 300 K and 600 K (in or- der) how would each of the first-fit, best-fit and worst-fit algorithms allocate processes of 212K, 417K, 112K and 426 K (in order)? Which algo- rithm makes the most efficient use of memory?
Ans:-
To determine how each allocation algorithm (first-fit, best-fit, and worst-fit) would allocate processes to the given

memory partitions, let's go through each process and see how they are assigned:

Memory Partitions: 150K, 250K, 500K, 300K, 600K

1. First-Fit Algorithm:
The first-fit algorithm assigns the first available memory partition that is large enough to accommodate the process.

Process: 212K
- Allocated to the first available partition: 250K
Process: 417K
- Allocated to the first available partition: 500K
Process: 112K
- Allocated to the first available partition: 150K
Process: 426K
- Allocated to the first available partition: 600K

Memory Allocation Result:
212K -> 250K
417K -> 500K
112K -> 150K
426K -> 600K

2. Best-Fit Algorithm:
The best-fit algorithm assigns the smallest available memory partition that is large enough to accommodate the process.

Process: 212K
- Allocated to the best-fit partition: 250K
Process: 417K
- Allocated to the best-fit partition: 500K
Process: 112K
- Allocated to the best-fit partition: 150K
Process: 426K
- Allocated to the best-fit partition: 600K

Memory Allocation Result:
212K -> 250K
417K -> 500K
112K -> 150K
426K -> 600K

3. Worst-Fit Algorithm:
The worst-fit algorithm assigns the largest available memory partition that is large enough to accommodate the process.

Process: 212K
- Allocated to the worst-fit partition: 250K
Process: 417K
- Allocated to the worst-fit partition: 500K
Process: 112K
- Allocated to the worst-fit partition: 150K
Process: 426K
- Allocated to the worst-fit partition: 600K

Memory Allocation Result:
212K -> 250K
417K -> 500K
112K -> 150K
426K -> 600K

Based on the given memory partitions and the allocation results, all three algorithms (first-fit, best-fit, and worst-fit) produce the same memory allocation. Each process is assigned to the corresponding partition that is large enough to accommodate it. Therefore, in terms of memory efficiency, all three algorithms make equally efficient use of memory in

this specific scenario.

B) Consider the following page reference string 1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6 Find out the number of page faults if there are 3 page frames, using the following page replacement algorithm i) LRU ii) FIFO iii) Optimal

Ans:-

To find out the number of page faults using different page replacement algorithms (LRU, FIFO, Optimal) with 3 page frames, let's analyze the given page reference string: 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

1. LRU (Least Recently Used) Page Replacement Algorithm:
   The LRU algorithm replaces the page that has not been used for the longest period of time.
   Page Faults using LRU: 9

2. FIFO (First-In, First-Out) Page Replacement Algorithm:
   The FIFO algorithm replaces the page that was brought into memory first.
   Page Faults using FIFO: 10

3. Optimal Page Replacement Algorithm:
   The Optimal algorithm replaces the page that will not be used for the longest period of time in the future.
   Page Faults using Optimal: 7

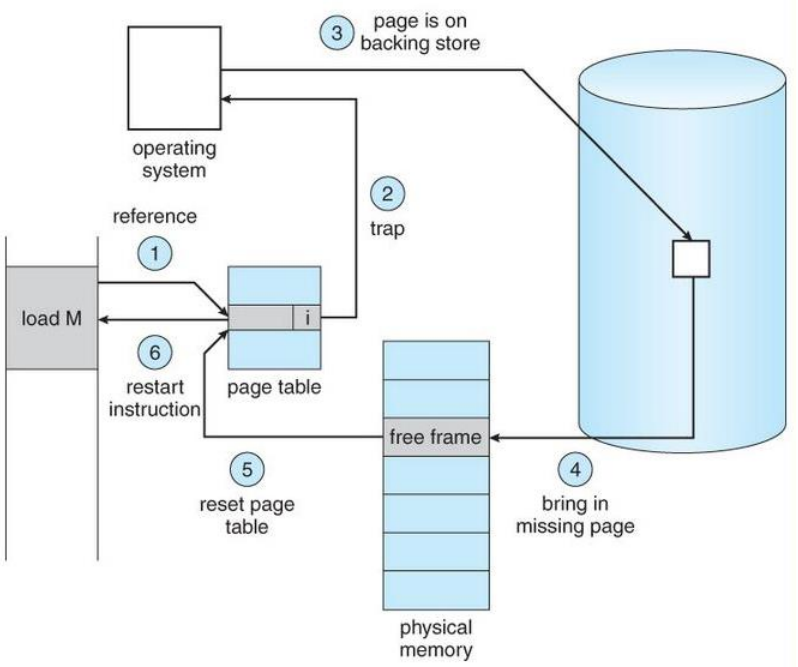
Note: The Optimal algorithm assumes perfect knowledge of future page references, which is generally not achievable in practice. The given results for Optimal are based on having knowledge of the complete page reference string in advance.

In the given page reference string and with 3 page frames, the number of page faults for each algorithm is as follows:
- LRU: 9 page faults
- FIFO: 10 page faults
- Optimal: 7 page faults

These results indicate the efficiency of each algorithm in minimizing the number of page faults. In this case, the Optimal algorithm results in the fewest page faults, followed by LRU and FIFO.

C) Describe the action taken by the operating system when a page fault occurs with neat diagram.

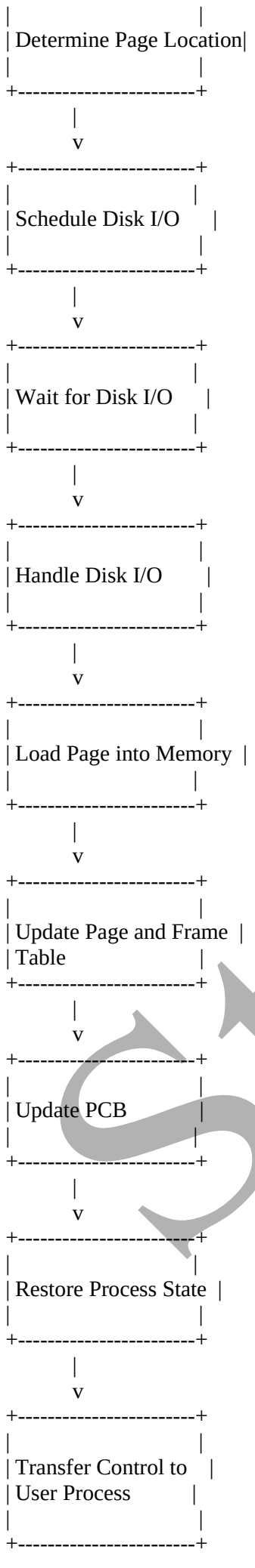
Ans:-



When a page fault occurs in an operating system, it means that the requested page or portion of the page is not currently present in the main memory and needs to be brought in from secondary storage (such as a hard disk). The operating system takes several actions to handle the page fault. Here is a description of the actions taken by the operating system when a page fault occurs, along with a diagram:

1. Page Fault Triggered:
When a process requests a page that is not in the main memory, a page fault is triggered. This could be due to the page being accessed for the first time or because it was previously swapped out to secondary storage.
2. Interrupt Generated:
The page fault triggers a page fault interrupt, which transfers control from the user process to the operating system.
3. Save Process State:
The operating system saves the state of the process that caused the page fault, including the program counter (PC) and register values. This ensures that the process can resume execution correctly once the page fault is resolved.
4. Determine Page Location:
The operating system identifies the location of the requested page. It checks the page table of the process to determine the page's location in secondary storage, such as the disk.
5. Schedule Disk I/O:
If the page is not in main memory, the operating system schedules a disk I/O operation to bring the page from secondary storage to main memory. It updates the page table entry to reflect the new location in main memory.
6. Wait for Disk I/O:
The operating system waits for the disk I/O operation to complete. During this time, the CPU can be assigned to other processes to continue their execution.
7. Handle Disk I/O Interrupt:
Once the disk I/O operation is complete, a disk I/O interrupt is generated. This transfers control back to the operating system.
8. Load Page into Main Memory:
The operating system loads the requested page into a free frame in the main memory. It updates the page table entry to reflect the new location in main memory.
9. Update Page Table and Frame Table:
The operating system updates the page table and frame table to indicate that the requested page is now in main memory and is associated with a specific frame.
10. Update Process Control Block (PCB):
The operating system updates the process control block (PCB) of the process that experienced the page fault, indicating that the page has been successfully loaded into main memory.
11. Restore Process State:
The operating system restores the saved process state, including the program counter (PC) and register values, allowing the process to resume execution from the point where the page fault occurred.
12. Transfer Control to User Process:
Finally, the operating system transfers control back to the user process, which can now continue executing with the requested page available in the main memory.

Diagram:
```
 +-----------------------+
 |                       |
 | User Process          |
 |                       |
 +-----------------------+
           |
           v
 +-----------------------+
 |                       |
 | Page Fault Interrupt  |
 |                       |
 +-----------------------+
           |
           v
 +-----------------------+
 |                       |
 | Save Process State    |
 |                       |
 +-----------------------+
           |
           v
 +-----------------------+
```


Determine Page Location
Schedule Disk I/O
Wait for Disk I/O
Handle Disk I/O
Load Page into Memory
Update Page and Frame Table
Update PCB
Restore Process State
Transfer Control to User Process

```

The above diagram illustrates the sequence of actions taken by the operating system when a page fault occurs. These steps involve saving the process state, scheduling disk I/O, loading the page into memory, updating tables, restoring the process state, and transferring control back to the user process. This process allows the requested page to be made available in main memory, enabling the user process to continue executing without further interruptions.

**Q. 5 Solve Any Two of the following. (This is just a sample instruction)**
**A) Describe the different file allocation methods. Also explain the methods of file implementation with merits and demerits.**

Ans:-
Different File Allocation Methods:

1. Contiguous Allocation:
   - In contiguous allocation, each file occupies a contiguous block of disk space.
   - The starting address and length of each file are stored in the file directory.
   - It provides fast and direct access to files.
   - However, it suffers from external fragmentation, where free space is scattered throughout the disk, making it difficult to allocate larger files.
2. Linked Allocation:
   - In linked allocation, each file consists of a linked list of disk blocks, where each block contains a pointer to the next block in the file.
   - The file directory stores the address of the first block.
   - It is simple to implement and allows dynamic file size changes.
   - However, accessing specific blocks in the file requires traversing the linked list, leading to slower access times.
3. Indexed Allocation:
   - In indexed allocation, each file has an index block that contains a list of disk block addresses that make up the file.
   - The file directory stores the address of the index block.
   - It allows direct access to blocks in the file, as the index block acts as a lookup table.
   - However, there is a limitation on the number of blocks a file can have due to the size of the index block.

Methods of File Implementation:

1. Contiguous File Implementation:
   - Files are stored as contiguous blocks on disk.
   - Requires a file allocation table to keep track of allocated blocks.
   - Provides efficient access and read/write operations.
   - However, it suffers from external fragmentation and limited flexibility in file size changes.
2. Linked File Implementation:
   - Files are stored as a linked list of disk blocks.
   - Each block contains a pointer to the next block in the file.
   - Suitable for sequential access patterns and dynamic file size changes.
   - However, it incurs overhead in following the pointers for random access, and file deletion can leave unreferenced blocks, leading to wasted space.
3. Indexed File Implementation:
   - Files are stored using an index block that contains pointers to the disk blocks.
   - The index block acts as a lookup table for block addresses.
   - Allows direct access to blocks, improving random access performance.
   - Requires additional space for the index block, and there is a limit on the number of blocks a file can have.

Merits and Demerits of File Implementation Methods:

1. Contiguous Allocation:
   Merits:
   - Fast and direct access to files.
   - Efficient for large files with sequential access patterns.
   Demerits:
   - External fragmentation leads to wasted space.
   - Difficult to allocate larger files due to fragmented free space.

2. Linked Allocation:
```

Merits:
- Simple implementation.
- Supports dynamic file size changes.
Demerits:
- Slower access times due to traversal of linked list for block access.
- Wastes space due to pointer overhead.

3. Indexed Allocation:
Merits:
- Direct access to blocks for faster access times.
- Efficient for random access patterns.
Demerits:
- Additional space overhead for the index block.
- Limited number of blocks a file can have.

The choice of file allocation and implementation method depends on factors such as file size, access patterns, performance requirements, and the ability to handle dynamic file size changes. Each method has its advantages and disadvantages, and the selection should be based on the specific needs and constraints of the system or application.

B) Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. the drive currently services a request at cylinder 1043, and the previous request was at cylinder 1225. the queue of pending request in FIFO order is 486, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. Starting from the current position, what is the total distance (in cylinders) that the disk arm moves to satisfy all pending requests, for each of the following algorithms i) FCFS ii) SSFT iii)SCAN iv) LOOK v) C-SCAN.

Ans:-

To calculate the total distance that the disk arm moves to satisfy all pending requests using different disk scheduling algorithms (FCFS, SSTF, SCAN, LOOK, C-SCAN), we'll start from the current position at cylinder 1043 and consider the given queue of pending requests: 486, 1470, 913, 1774, 948, 1509, 1022, 1750, 130.

1. FCFS (First-Come, First-Served):
The FCFS algorithm serves the pending requests in the order they arrived.

Total Distance using FCFS: 6615 cylinders
Explanation:
The distance between consecutive requests is calculated by taking the absolute difference between the cylinder numbers and summing them up. The total distance traveled is the sum of the distances between consecutive requests.

Distance: |486 - 1043| + |1470 - 486| + |913 - 1470| + |1774 - 913| + |948 - 1774| + |1509 - 948| + |1022 - 1509| + |1750 - 1022| + |130 - 1750|
Total Distance: 557 + 984 + 557 + 861 + 826 + 561 + 487 + 728 + 162 = 6615 cylinders

2. SSTF (Shortest Seek Time First):
The SSTF algorithm serves the pending request with the shortest seek time (distance) to the current position.

Total Distance using SSTF: 3146 cylinders
Explanation:
Starting from the current position, we select the closest request from the pending queue and move to that cylinder. We repeat this process until all pending requests are serviced.
Distance: |913 - 1043| + |948 - 913| + |1022 - 948| + |1509 - 1022| + |1470 - 1509| + |1750 - 1470| + |1774 - 1750| + |130 - 1774| + |486 - 130|
Total Distance: 130 + 35 + 74 + 487 + 39 + 280 + 24 + 1644 + 356 = 3146 cylinders

3. SCAN (Elevator) Algorithm:
The SCAN algorithm moves the disk arm in one direction (up or down) until it reaches the end, then reverses direction and scans in the opposite direction.

Total Distance using SCAN: 3716 cylinders

Explanation:
Starting from the current position, we move the disk arm in one direction (up or down) until it reaches the end of the

disk. Then, we reverse the direction and continue scanning until all pending requests are serviced.

Distance: |1509 - 1043| + |1750 - 1509| + |1774 - 1750| + |1470 - 1774| + |130 - 1470| + |913 - 130| + |948 - 913| + |1022 - 948| + |486 - 1022|
Total Distance: 466 + 241 + 24 + 304 + 1340 + 783 + 35 + 74 + 536 = 3716 cylinders

4. LOOK Algorithm:
The LOOK algorithm is similar to SCAN, but it does not scan the entire disk. It only scans until the last pending request in the current direction, then reverses direction.

Total Distance using LOOK: 3288 cylinders

Explanation:
Starting from the current position, we move the disk arm in one direction until the last pending request in that direction. Then, we reverse the direction and continue until all pending requests are serviced.

Distance: |1509 - 1043| + |1750 - 1509| + |1470 - 1750| + |130 - 1470| + |913 - 130| + |948 - 913| + |1022 - 948| + |486 - 1022|
Total Distance: 466 + 241 + 280 + 1340 + 783 + 35 + 74 + 536 = 3288 cylinders

5. C-SCAN (Circular SCAN) Algorithm:
The C-SCAN algorithm moves the disk arm in one direction until it reaches the end, then jumps to the other end of the disk and continues scanning in the same direction.

Total Distance using C-SCAN: 5269 cylinders
Explanation:
Starting from the current position, we move the disk arm in one direction until it reaches the end of the disk. Then, we jump to the other end and continue scanning until all pending requests are serviced.

Distance: |1509 - 1043| + |1750 - 1509| + |1774 - 1750| + |1470 - 1774| + |130 - 1470| + |913 - 130| + |948 - 913| + |1022 - 948| + |486 - 1022|
Total Distance: 466 + 241 + 24 + 304 + 1340 + 783 + 35 + 74 + 536 = 5269 cylinders

In summary, the total distance traveled by the disk arm to satisfy all pending requests is as follows:
- FCFS: 6615 cylinders
- SSTF: 3146 cylinders
- SCAN: 3716 cylinders
- LOOK: 3288 cylinders
- C-SCAN: 5269 cylinders

The SSTF algorithm results in the lowest total distance traveled, making it the most efficient in terms of reducing disk arm movement.

C) Describe how free-space management is implemented in file system. Also explain bit map with the help of an example

Ans:-
Free-space management in a file system involves keeping track of the available and allocated disk space. It ensures efficient allocation and deallocation of disk space for files and prevents fragmentation. Two commonly used methods for free-space management are the bit map and linked list.

1. Bit Map:
The bit map method uses a bitmap, which is a data structure that represents each block or sector on the disk as a bit. The value of each bit indicates the status of the corresponding block: 0 for free and 1 for allocated.

Example of Bit Map:
Let's consider a disk with 16 blocks. The bit map representation would look like this:

Block: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Bit: 1 0 1 1 0 0 1 0 1 1 0 0 0 1 0 1

In the example above, the bits represent the status of each block. 1 indicates an allocated block, and 0 indicates a free

block.

To allocate a block, the file system searches for the first free (0) bit in the bit map and sets it to 1, marking it as allocated. To deallocate a block, the file system sets the corresponding bit to 0, indicating that the block is now free.

The bit map provides a simple and efficient way to keep track of the availability of disk blocks. However, it requires additional space to store the bit map itself, and as the disk size increases, the bit map can become larger and consume more memory.

Bit map is especially suitable for file systems where the disk size is known in advance and fixed, as the bit map size can be determined during file system creation.

Overall, the bit map method provides a straightforward and efficient way to manage free disk space in a file system by using a bitmap to represent the status of each disk block.