Q.NO.1

a. Explain the operating system.

Ans:- An operating system (OS) is a software that acts as an intermediary between computer hardware and software applications. It provides a set of services and functionalities that enable the efficient management of hardware resources and the execution of various software programs. The operating system controls and coordinates the activities of hardware components, manages memory and storage, provides user interfaces, and facilitates communication between software applications and hardware devices.

b. Describe how abstractions, standard interface and resource usage support the system components of modern operating system.

Ans:- Abstractions, standard interfaces, and efficient resource usage are essential concepts that support the system components of modern operating systems. Let's break down each of these aspects and see how they contribute to the functionality and efficiency of operating system components:

1. Abstractions: Abstractions provide a simplified and consistent view of underlying complex hardware and software resources. They hide the intricate details and complexities, allowing application developers and system programmers to interact with the system in a more user-friendly and efficient manner. Abstractions in modern operating systems include:

- **Process Abstraction**
- **File Abstraction**
- **Device Abstraction**

2. Standard Interfaces: Standard interfaces define a set of rules and protocols that enable components of the operating system to communicate with each other and with external applications. These interfaces promote interoperability and modularity, allowing different components to work together seamlessly. Examples of standard interfaces include:

- **APIs (Application Programming Interfaces)**
- **System Calls**
- **Network Protocols**

3. Resource Usage: Efficient resource usage is crucial for optimizing system performance and ensuring fair allocation of resources among different processes and applications. Modern operating systems manage resources like CPU time, memory, disk I/O, and network bandwidth to ensure efficient utilization. Techniques for resource usage support include:

- **Process Scheduling**
- **Memory Management**
- **I/O Management**
- **Power Management**

c. Discuss any FOUR (4) operating system managers, the responsibilities, resources and relationships with each other in managing the operating system.

Ans:- Operating systems are complex software that manage computer hardware and provide an environment for other software to run. Let's discuss five key components involved in managing an operating system:

1. Process Manager

- Responsibilities: The process manager is responsible for creating, scheduling, and terminating processes. It ensures fair and efficient allocation of CPU time to various processes.

- Resources: It manages CPU resources, process states, and execution order. It maintains process control blocks (PCBs) containing process information.

- Relationships: The process manager interacts with the CPU scheduler, memory manager, and I/O manager to coordinate process execution, memory allocation, and I/O operations.

2. Memory Manager

- Responsibilities: The memory manager is responsible for managing the system's memory resources. It handles memory allocation, protection, and swapping.

- Resources: It manages physical and virtual memory, keeps track of allocated and free memory blocks, and handles memory paging or segmentation.

- Relationships: The memory manager collaborates with the process manager to allocate memory for processes and the file system manager to manage memory used for file caching.

3. File System Manager

- Responsibilities: The file system manager manages files and directories on storage devices. It handles file creation, deletion, manipulation, and access permissions.

- Resources: It manages disk space, maintains file metadata (e.g., file size, timestamps), and handles read/write operations.

- Relationships: The file system manager interacts with the I/O manager for data transfers, the process manager for file-related process operations, and the device driver manager to communicate with storage devices.

4. Device Driver Manager

- Responsibilities: The device driver manager manages communication between the operating system and hardware devices. It provides a standardized interface for various device types.

- Resources: It manages device drivers, which are software modules responsible for interacting with specific hardware components.

- Relationships: The device driver manager collaborates with the process manager for I/O operations, the memory manager to transfer data between memory and devices, and the I/O manager for efficient data movement.

5. I/O Manager

- Responsibilities: The I/O manager handles input and output operations between the computer and its external environment. It buffers and schedules data transfers to/from devices.

- Resources: It manages I/O queues, data buffers, and coordinates data transfers to minimize delays.

- Relationships: The I/O manager interacts with the device driver manager for device communication, the process manager for I/O-bound process handling, and the memory manager to move data between devices and memory.

Q.NO.2

a. Describe THREE (3) types of scheduling in operating system.

Ans:- Scheduling in operating systems refers to the process of managing the execution of tasks or processes in a system's CPU to achieve efficient resource utilization and responsiveness. There are various scheduling algorithms that determine the order in which processes are selected for execution.

Here are three types of scheduling algorithms:

1. First-Come, First-Served (FCFS) Scheduling: In this simple scheduling algorithm, processes are executed in the order they arrive in the ready queue. The first process that enters the queue is the first to be executed. FCFS is easy to understand but can lead to poor average waiting times, especially if long processes are followed by short ones (a phenomenon known as the "convoy effect").

2. Shortest Job Next (SJN) Scheduling: Also known as Shortest Job First (SJF) scheduling, this algorithm selects the process with the smallest execution time next. It aims to minimize the average waiting time by executing shorter processes first. However, predicting the execution time accurately can be challenging, and this algorithm might cause longer processes to wait indefinitely.

3. Round Robin (RR) Scheduling: Round Robin is a pre-emptive scheduling algorithm where each process is assigned a fixed time quantum or time slice. Processes are executed in a cyclic order for the duration of their time quantum. If a process doesn't complete within its time quantum, it's moved to the end of the queue. RR provides fairness and responsiveness, but its performance can degrade with short time quanta and long processes.

b. Table 1 contains the processes, arrival time, CPU time and priority:

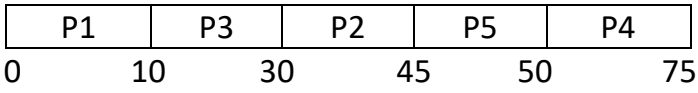
Table 1

Processes	Arrival Time	CPU Time	Priority
P1	0	10	1
P2	1	15	2
P3	2	20	1
P4	3	25	4
P5	4	5	3

Based on the information in Table 1, draw the Gantt Chart for the following process scheduling:

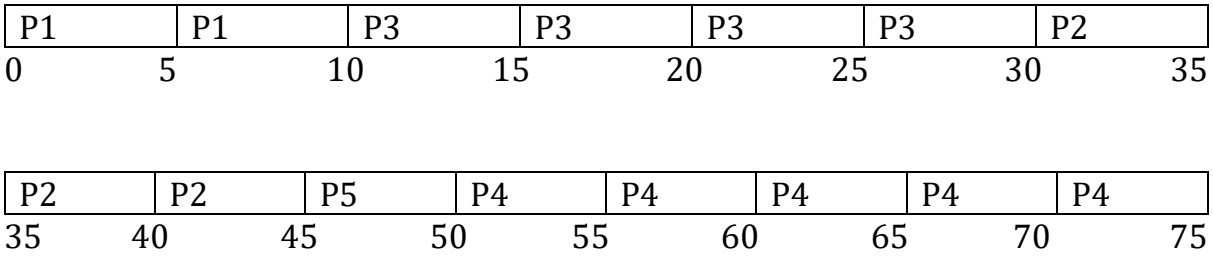
i. Priority Scheduling.

Ans:-



ii. Round Robin Scheduling with Time Quantum = 5 ms.

Ans:-



c. Explain the concepts of thread in operating system.

Ans:- Threads are a fundamental concept in operating systems and provide a way to execute multiple tasks concurrently within a single process. Here are 3 to 5 key concepts related to threads in operating systems:

1. Thread vs. Process: Threads are smaller units of a process. While a process is a self-contained program with its own memory space, threads share the same memory space within a process. This allows threads to communicate and share data more efficiently than separate processes.

2. Concurrency: Threads allow for concurrent execution of tasks within a single process. Each thread can execute independently and perform its own set of instructions, allowing for parallelism and potentially faster execution of tasks, especially on multi-core processors.

3. Thread States: Threads typically go through different states during their lifecycle, such as "running," "ready," and "blocked." The operating system scheduler manages these states to ensure efficient CPU utilization. Threads can be preemptively switched between these states to give the illusion of simultaneous execution.

4. Synchronization: Because threads within a process share the same memory space, they can access and modify the same data concurrently. This can lead to data inconsistencies and race conditions. Synchronization mechanisms like mutexes, semaphores, and condition variables are used to coordinate and control access to shared resources, ensuring data integrity.

5. User-Level vs. Kernel-Level Threads: Threads can be managed at the user level by the application itself or at the kernel level by the operating system. User-level threads provide more control to the application but may not take full advantage of multi-core processors. Kernel-level threads are managed by the OS and can leverage multiple processor cores efficiently.

d. Assume that five philosophers are waiting to have their meal at dining table with starvation rule. Discuss with a figure, the concepts of starvation in this situation.

Ans:- Concepts of Starvation:

In the context of the dining philosophers problem, starvation refers to a situation where a philosopher is unable to access the resources they need (forks in this case) to perform their task (eating), even though other philosophers are continuously accessing those resources.

1. Resource Allocation: The forks are resources that the philosophers need to eat. If multiple philosophers try to pick up the same fork simultaneously, they will need to wait until the fork becomes available. This resource allocation can lead to one or more philosophers waiting for an indefinite amount of time, causing starvation.

2. Deadlock: In a system with improper synchronization mechanisms, philosophers might end up in a deadlock. Deadlock occurs when every philosopher picks up one fork and then waits indefinitely for the other fork. This situation prevents any philosopher from eating, resulting in starvation for all philosophers.

3. Unequal Access: Even with proper synchronization, a situation might arise where some philosophers consistently have more access to the resources than others. For example, if one philosopher is always surrounded by empty seats, they might have an easier time acquiring both forks, leading to other philosophers starving.

4. Priority Inversion: Priority inversion occurs when a lower-priority philosopher holds a resource that a higher-priority philosopher needs. If the lower-priority philosopher doesn't release the resource in a timely manner, the higher-priority philosopher might starve.

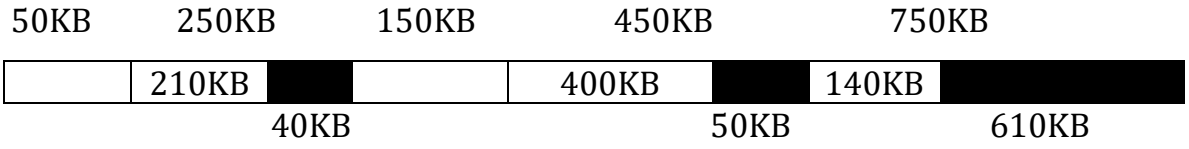
5. Fairness and Scheduling: Starvation can be mitigated through fair scheduling mechanisms. Philosophers could be given a turn to eat in a round-robin fashion, ensuring that each philosopher gets a fair chance to access the resources and eat. This prevents any single philosopher from being perpetually starved.

Q.NO.3

a. Assume that the main memory has the following five fixed partitions with the following sizes: 50KB, 250KB, 150KB, 450KB and 750KB (in order).

Draw an appropriate table on how the First-fit algorithm would allocate the processes 210KB, 400KB, 140KB and 650KB (in order).

Ans:- First-fit algorithm



650KB didn't get to allocate memory.

b. Identify FOUR (4) characteristics of paging and segmentation in virtual memory.

Ans:- Paging and segmentation are two memory management techniques used in virtual memory systems to efficiently utilize physical memory. Here are four characteristics of paging and segmentation:

1. Granularity

- Paging: Paging divides both the physical and virtual memory into fixed-size blocks called "pages." This fixed-size granularity simplifies memory management but can lead to internal fragmentation.

- Segmentation: Segmentation divides virtual memory into variable-sized segments, which represent logical units of a program or data. This flexibility allows for better memory utilization but can result in fragmentation.

2. Address Space Organization

- Paging: In paging, the address space is divided into equal-sized pages, and these pages may not correspond to the logical structure of the program. This can lead to inefficient memory usage if some pages are not fully utilized.

- Segmentation: Segmentation allows the address space to be divided into logical segments, such as code, data, and stack segments. This reflects the program's natural structure, making it easier to manage and reducing wasted memory.

3. Page Table vs. Segment Table

- Paging: Paging typically uses a single page table that maps virtual page numbers to physical frame numbers. The page table is relatively simple but can become large for systems with a large address space.

- Segmentation: Segmentation uses multiple segment tables, each corresponding to a different segment type. These tables can be more complex as they need to manage variable-sized segments efficiently.

4. Fragmentation

- Paging: Paging can suffer from internal fragmentation, where a page may not be fully utilized, wasting some memory space within each page.

- Segmentation: Segmentation can suffer from external fragmentation, where memory is fragmented between segments, making it challenging to allocate large contiguous blocks of memory even if there is enough free memory in total.

c. Write about THREE (3) levels page table structure of Linux virtual memory.

Ans:- Linux uses a three-level page table structure in its virtual memory management system to efficiently map virtual addresses to physical addresses. This hierarchical approach helps manage the translation process and conserve memory resources. Here's a brief overview of the three levels:

1. Page Global Directory (PGD) or Page Global Directory Pointer (PGDP)

- The first level of the page table structure.

- PGD contains pointers to the Page Middle Directory (PMD) tables.

- It divides the entire virtual address space into a set of smaller regions.

2. Page Middle Directory (PMD)

- The second level of the page table structure.

- PMD tables contain pointers to Page Tables (PTs) or Huge Page Tables (HPs).

- They further refine the address translation process, narrowing down the search for the appropriate page table.

3. Page Table (PT) or Huge Page (HP)

- The third and final level of the page table structure.

- PTs map individual pages of memory, whereas HPs are used for large contiguous memory regions.

- They provide the final translation of virtual addresses to physical addresses.

Q.NO.4.

a. Given the following information in Table 1:

Table 1

User	File Name				
	Stud_1	**Stud_2**	**Stud_3**	**Stud_4**	**Stud_5**
Damia	Read	Read Write	Read Write	Owner Read Write Execute	Read Write
Kelvin	Owner Read Write Execute	Read Write	Read Write	Read	
John	Read Write	Read	Owner Read Write Execute	Read	Read Write

Draw a matrix that shows the access control list of files Stud_1, Stud_2, Stud_3, Stud_4 and Stud_5 for each user.

Ans:-

User	File Name				
	Stud_1	**Stud_2**	**Stud_3**	**Stud_4**	**Stud_5**
Damia	r	rw	rw	rwx	rw
Kelvin	rwx	rw	rw	r	-
John	rw	r	rwx	r	rw

b. Write about the concept of acyclic graph structure and file sharing in file management.

Ans:- An acyclic graph structure, often referred to as a directed acyclic graph (DAG), is a data structure composed of nodes and edges, where the edges have a defined direction and the structure contains no cycles. In the context of file management and file sharing, acyclic graph structures play a crucial role in organizing and representing relationships between files and directories, facilitating efficient file access and sharing. Let's delve into this concept further.

1. Organizing Files and Directories: In file management systems, files and directories can be organized hierarchically, forming a tree structure. An acyclic graph structure extends this by allowing for more complex relationships between

files and directories. Each node in the graph represents either a file or a directory, and the edges represent relationships like containment or links.

2. Dependency Management: Acyclic graph structures are particularly useful when dealing with dependencies between files or tasks. For example, in a software development project, code files might depend on libraries or other code files. Representing these dependencies as a DAG ensures that there are no circular dependencies, preventing infinite loops and ensuring orderly execution.

3. Version Control Systems: Version control systems like Git employ acyclic graph structures to track changes in code repositories. Each commit represents a node in the graph, and the edges connect commits in chronological order, forming a history of changes. This structure helps in branching, merging, and tracking the development history of code.

4. File Sharing: Acyclic graph structures are relevant to file sharing in a distributed environment. When multiple users collaborate on files or projects, the DAG can be used to track changes, permissions, and relationships. It enables users to share files with others, maintain version histories, and control access rights efficiently.

5. Access Control: Access control lists (ACLs) can be implemented using DAGs to manage permissions and sharing settings for files and directories. Users or groups can be associated with nodes in the graph, determining who can access or modify specific files or folders.

c. Describe the following types of file access:

i. Indexed Sequential File

ii. Direct File

Ans:-

i. Indexed Sequential File:- An Indexed Sequential File is a type of data storage and retrieval method that combines elements of both sequential and indexed access. It consists of a sequential data file in which records are stored in a specific order, and an index is maintained to facilitate faster random access to these records. This allows for efficient searching and retrieval of data while still maintaining the sequential order of the file.

ii. Direct File:- A Direct File, also known as a random access file, is a type of data file where records or data elements can be directly accessed using a unique identifier, such as a record number or a key, without the need to read through the entire file sequentially. This provides quick and efficient access to specific data points within the file, making it suitable for applications that require frequent random access operations.

Q.NO.5

a. Identify TEN (10) steps of after hardware interrupt completes.

Ans:- After a hardware interrupt completes, there are several steps that typically occur to ensure the proper functioning of the system. These steps may vary slightly depending on the specific operating system and hardware architecture, but here are ten common steps:

1. Context Saving: The CPU saves the current state of the interrupted process, including the program counter, registers, and other relevant information, onto the stack or in a designated area of memory.

2. Interrupt Acknowledgment: The interrupt controller (e.g., the Programmable Interrupt Controller or PIC) acknowledges the interrupt and sends an acknowledgment signal to the interrupting device to confirm that the interrupt request has been received.

3. Interrupt Dispatch: The interrupt handler or interrupt service routine (ISR) associated with the interrupt is determined. The interrupt vector or table is used to find the appropriate ISR.

4. Disable Interrupts: To prevent nested interrupts and maintain interrupt handling integrity, the CPU often disables interrupts during the execution of the ISR.

5. Execution of ISR: The CPU begins executing the ISR, which is a specific piece of code written to handle the interrupt. This code performs tasks related to the interrupt, such as reading data from the hardware device or processing data.

6. Processing Interrupt: The ISR processes the interrupt and may perform actions like updating data structures, setting flags, or initiating other operations based on the interrupt's purpose.

7. Restoring Context: After completing the ISR, the CPU restores the context of the interrupted process, which includes reloading the saved program counter, registers, and other state information from step 1.

8. Enabling Interrupts: The CPU re-enables interrupts if they were disabled during the ISR execution to allow for the handling of other interrupts that might occur concurrently.

9. Return from Interrupt: The CPU executes a return-from-interrupt (RTI) or similar instruction to resume the interrupted process. This instruction typically pops the saved context from the stack and sets the program counter to the address where the interrupted code should continue.

10. Resuming Execution: The interrupted program continues its execution from the point where it was interrupted, with all the processor state and registers restored, as if the interrupt never occurred.

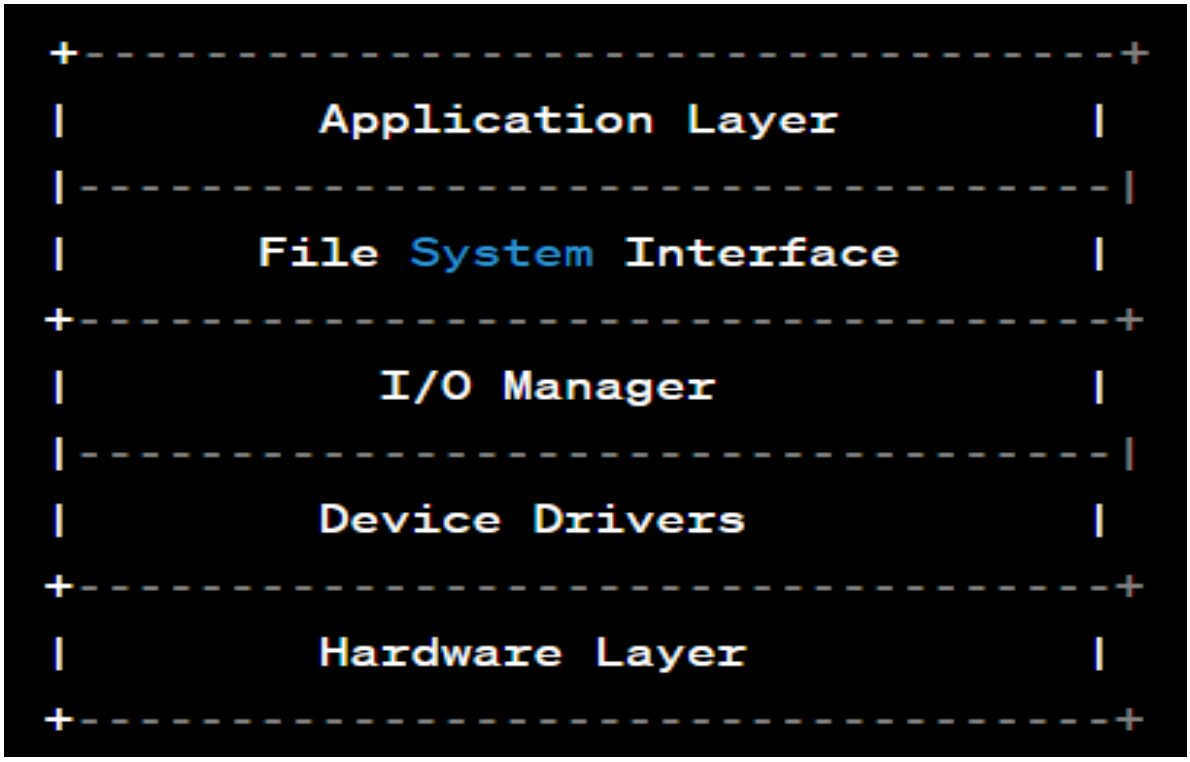
b. Differentiate between character-oriented device and blockoriented device.

Ans:-

Aspect	Character-Oriented Device	Block-Oriented Device
Data Transfer Unit	Transfers data one character at a time.	Transfers data in blocks or groups of bytes.
Data Buffering	Typically small or nonexistent buffers.	Usually has larger buffers to store blocks of data.
Typical Usage	Commonly used for devices like keyboards, mice, and printers.	Used for devices like hard drives, SSDs, and optical drives.
Latency	Lower latency as data is transferred character by character.	Higher latency due to the need to read or write entire blocks.
Performance	Suitable for low-speed, interactive devices.	Designed for high-speed, data-intensive operations.

c. Draw with a label the FIVE (5) layers of I/O systems.

Ans:-



1. Application Layer: This is the top layer where user applications interact with the I/O system. It includes software that initiates and manages I/O requests. Users or applications request I/O operations through this layer.

2. File System Interface: This layer abstracts the underlying file system and provides a standardized way for applications to access files and directories. It manages file metadata, permissions, and file-level operations.

3. I/O Manager: The I/O manager is responsible for coordinating I/O operations between the application layer and the lower layers. It queues and schedules I/O requests, manages buffers, and handles error conditions.

4. Device Drivers: Device drivers are specific to each hardware device and serve as intermediaries between the I/O manager and hardware. They translate high-level I/O requests into low-level commands that the hardware can understand and execute.

5. Hardware Layer: This is the lowest layer, consisting of the physical hardware devices such as hard drives, network adapters, and other peripherals. It is responsible for carrying out the actual data transfer and device control operations.