CBCS SCHEME

USN | | | | | | | | | | |

BCS303

Third Semester B.E./B.Tech. Degree Examination, Dec.2023/Jan.2024
Operating Systems

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

			M	L	C
		Module – 1			
Q.1	**a.**	Define Operating System. Explain dual mode of OS with a neat diagram.	**5**	**L1, L2**	**CO1**
	b.	Distinguish between the following terms: i) Multiprogramming and Multitasking ii) Multiprocessor system and clustered system.	**10**	**L2**	**CO1**
	c.	With a neat diagram, explain the concept the concept of VM-WARE architecture.	**5**	**L1, L2**	**CO1**
		OR			
Q.2	**a.**	Explain the operating system services with respect to programs and users.	**5**	**L2**	**CO1**
	b.	List and explain the different computing environments.	**5**	**L1, L2**	**CO1**
	c.	What are system calls? List and explain the different types of system calls.	**10**	**L1, L2**	**CO1**
		Module – 2			
Q.3	**a.**	Define process. Explain different states of a process with state diagram.	**8**	**L1, L2**	**CO1**
	b.	What is IPC? Explain direct and indirect communication with respect to message passing.	**8**	**L1, L2**	**CO2**
	c.	Explain context-switching.	**4**	**L2**	**CO2**
		OR			
Q.4	**a.**	What is multi-threaded process? Explain the four benefits of multithreaded programming.	**6**	**L2**	**CO2**
	b.	Calculate the average waiting time and average turn around time by drawing the Gantt-chart using FCFS, SJF-non preemptive, SRTF, RR(q = 2ms) and porosity algorithms.	**14**	**L3**	**CO2**
		Module – 3			
Q.5	**a.**	What is critical section? What are the requirements for the solution to critical section problem? Explain Peaterson's solution.	**8**	**L1, L2**	**CO3**
	b.	Explain Reader's-Writer's problem using semaphores.	**12**	**L2**	**CO3**

Process	Arrival time	Burst time	Porosity
P1	0	9	3
P2	1	4	2
P3	2	9	1
P4	3	5	4

OR

Q.6	**a.**	What is deadlock? What are the necessary conditions for the deadlock to occur?	**6**	**L1, L2**	**CO3**
	b.	Consider the following snap-shot of a system: Answer the following using Banker's algorithm: i) Is the system in safe state? If so give the safe sequence. ii) If process P2 requests (0, 1, 1, 3) resource can it be granted immediately.	**14**	**L3**	**CO3**

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	2	0	0	1	4	2	1	2	3	3	2	1
P1	3	1	2	1	5	2	5	2				
P2	2	1	0	3	2	3	1	6				
P3	1	3	1	2	1	4	2	4				
P4	1	4	3	2	3	6	6	5				

Module – 4

Q.7	**a.**	What is paging? Explain with neat diagram paging hardware with TLB?	**10**	**L1, L2**	**CO4**
	b.	What are the commonly used strategies to select a free hole from the available holes?	**6**	**L1**	**CO4**
	c.	Explain fragmentation in detail.	**4**	**L2**	**CO4**

OR

Q.8	**a.**	With a neat diagram? Describe the steps in handling the page fault.	**8**	**L2**	**CO4**
	b.	Consider the page reference string: 1, 0, 7, 1, 0, 2, 1, 2, 3, 0, 3, 2, 4, 0, 3, 6, 2, 1 for a memory with 3 frames. Determine the number of page faults using F1, F0, optimal and LRU replacement algorithms which algorithm is more efficient.	**12**	**L3**	**CO4**

Module – 5

Q.9	**a.**	Define file. List and explain the different file attributes and operations.	**10**	**L1**	**CO5**
	b.	Explain the different allocation methods.	**10**	**L2**	**CO5**

OR

Q.10	**a.**	What is Access Matrix? Explain Access Matrix method of system protection with domain as objects and its implementation.	**10**	**L1, L2**	**CO5**
	b.	A drive has 5000 cylinders numbered 0 to 4999. The drive is currently serving a request 143 and previously serviced a request at 125. The queue of pending requests in FIFO order is: 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130 starting from current head position. What is the total distance travelled (in cylinders) by disk arm to satisfy the requests using FCFS, SSTF, SCAN, LOOK and C-LOOK algorithm.	**10**	**L3**	**CO5**

* * * * *

MODULE – 1

1. (a). Define operating Systems. Explain the dual-mode operating system with a neat diagram. 5 Marks (Exp-3, diagram-2)

Answer:

A program that acts as an intermediary between a user of a computer and the computer hardware. An operating System is a collection of system programs that together control the operations of a computer system.

Some examples of operating systems are UNIX, Mach, MS-DOS, MS-Windows, Windows/NT, Chicago, OS/2, MacOS, VMS, MVS, and VM.

The **Dual-Mode** taken by most computer systems is to provide hardware support that allows us to differentiate among various modes of execution.
At the very least, we need two separate modes of operation: **user mode and kernel mode** (also called supervisor mode, system mode, or privileged mode).

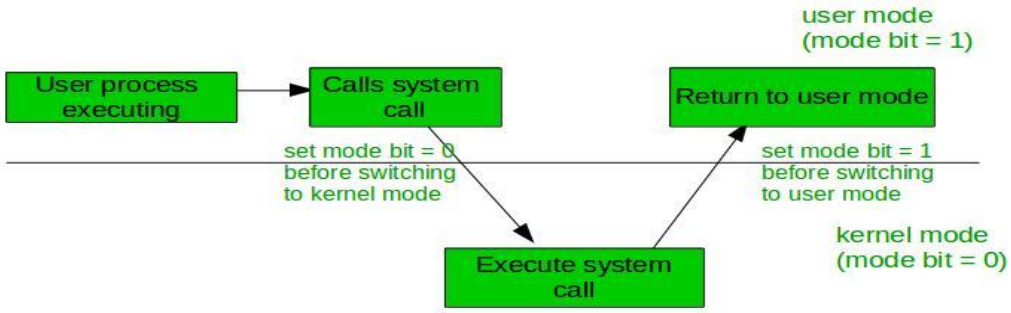
A bit, called the mode bit is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1). With the mode bit, we are able to distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user.

When the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system (via a system call), it must transition from user to kernel mode to fulfill the request.

The architectural enhancement is useful for many other aspects of system operation as well.

At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the state of the mode bit to 0). Thus, whenever the operating system gains control of the computer, it is in kernel mode. The system always switches to user mode (by setting the mode bit to 1) before passing control to a user program.

The dual mode of operation provides us with the means for protecting the operating system from errant users-and errant users from one another.



1. (b). Distinguish between the following terms. (2 M)
(i) Multiprogramming and Multitasking (2M)
(ii) Multiprocessor System and Clustered System. (6M Answer)

Answer:

i. Multitasking System and Multiprogramming:

- Time sharing (or multitasking) is a logical extension of multiprogramming. -In time-sharing systems, the CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.

-A time-shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer.

-Each user has at least one separate program in memory. A program loaded into memory and executing is called a process. Time-sharing and multiprogramming require several jobs to be kept simultaneously in memory.

-Since in general main memory is too small to accommodate all jobs, the jobs are kept initially on the disk in the job pool.

ii. Multiprocessor systems and clustered systems:

-Multiprocessor systems (also known as parallel systems or tightly coupled systems) are growing in importance.

-Such systems have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices.

- Clustered systems differ from multiprocessor systems, however, in that they are composed of two or more individual systems coupled together.

-Clustering is usually used to provide high-availability service; that is, service will continue even if one or more systems in the cluster fail.

-High availability is generally obtained by adding a level of redundancy in the system.

1. (c). With a neat diagram, explain the concept of the VM-WARE architecture. (5M) Exp-3.Diag-2

Answer:

VMware architecture is a framework that allows multiple virtual machines (VMs) to run on a single physical server, providing flexibility, efficiency, and resource optimization. Here's a breakdown of the key components:

1. Physical Hardware

- CPU, Memory, Storage, Network: The physical resources of a server that provide the computational power, memory, data storage, and network connectivity.
- Server: The physical machine that hosts the virtualization environment.

2. VMware Hypervisor (ESXi)

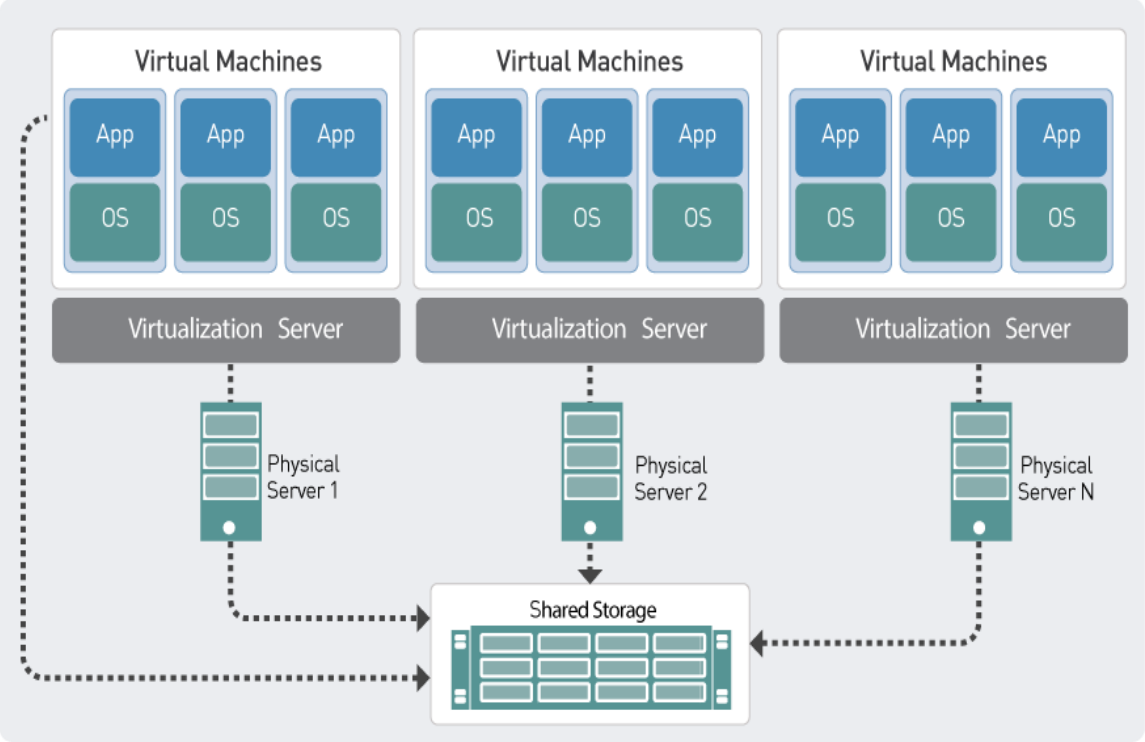
- Hypervisor: A thin software layer that sits directly on the physical hardware, abstracting and managing the hardware resources. VMware's hypervisor is called ESXi.
- Resource Allocation: The hypervisor allocates the server's physical resources to the VMs, managing the distribution and isolation of these resources.

3. Virtual Machines (VMs)

- Virtual Hardware: Each VM is allocated virtualized hardware (e.g., virtual CPU, memory, disk, network adapter) by the hypervisor.
- Guest Operating Systems: On top of this virtual hardware, each VM runs its own operating system (Windows, Linux, etc.).
- Applications: Applications run inside the guest operating system just as they would on a physical machine.

4. Interaction Between Layers

- Resource Management: The hypervisor ensures that each VM operates independently, even though they share the same physical resources.
- Isolation: VMs are isolated from each other, so if one VM crashes or is compromised, the others remain unaffected.



2. (a). Explain the operating system services with respect to programs and users.(5M) Exp-3,Eg-2.

Answer:

Operating system (OS) services are essential functions provided by the OS that enable programs to run efficiently and users to interact with the computer system. These services act as a bridge between the hardware, applications, and users, ensuring smooth and secure operation of the computer. Here's how OS services function with respect to both programs and users:

For Programs:

Program Execution: It handles all necessary operations for a program to run, including managing process states (ready, running, waiting) and ensuring that processes don't interfere with each other.
I/O Operations: Programs use standardized system calls to perform I/O operations without needing to know the specifics of the hardware.
File System Manipulation: Programs can easily access and manipulate files through a well-defined file system API, without worrying about the underlying disk management.
Communication: Programs can communicate with each other, synchronize tasks, or exchange data without needing to handle the complexities of inter-process communication (IPC) mechanisms.
Resource Allocation: Programs receive the necessary resources to execute efficiently and fairly, with the OS managing any conflicts or contention.
Protection and Security: Programs are given access to only the resources they need, with permissions and access controls enforced by the OS.

For Users:

Program Execution: Users can run applications without needing to know how the OS manages the execution. The OS handles all complexities, providing a seamless experience.
I/O Operations: Users can interact with various input and output devices (e.g., keyboard, mouse, printer) effortlessly, as the OS handles all the underlying operations.
File System Manipulation: Users can organize, store, and retrieve their data using a user-friendly file system, often through a graphical user interface (GUI) or command-line interface (CLI).
Communication: Users can run distributed applications, like web browsers or network games, where different processes or systems communicate seamlessly.
Resource Allocation: Users can run multiple applications simultaneously, with the OS ensuring that resources are shared without causing significant

performance degradation.
Protection and Security: Users can set permissions and security policies, ensuring that their data and the system are protected from unauthorized access or attacks.

2. (b). List and explain the different computing environments.(5M)

Answer:

Computing environments refer to the various configurations and contexts in which computing systems operate. These environments differ in terms of scale, purpose, user interaction, and the resources they use. Here's a breakdown of the different computing environments:

Personal Computing Environment

- **Description:** This environment refers to the use of personal computers (PCs), laptops, tablets, or smartphones by individuals for personal tasks.
- **Use Cases:** Activities such as word processing, web browsing, gaming, multimedia consumption, and basic software development.

Distributed Computing Environment

- **Description:** In this environment, multiple computers work together to achieve a common goal. These systems are networked and often work on different parts of a task simultaneously.
- **Use Cases:** Cloud computing, content distribution networks, peer-to-peer networks, and grid computing.

Client-Server Computing Environment

- **Description:** This environment involves a central server that provides resources or services to multiple client machines over a network.
- **Use Cases:** Web applications, email services, file sharing, database management systems.

Cloud Computing Environment

- **Description:** Cloud computing provides on-demand access to computing resources (such as servers, storage, databases, networking, software) over the internet. These resources are managed by third-party providers.
- **Use Cases:** Web hosting, data storage, software as a service (SaaS), platform as a service (PaaS), infrastructure as a service (IaaS).

Embedded Computing Environment

- **Description:** Embedded systems are specialized computing environments where computers are embedded within larger systems to perform specific tasks.
- **Use Cases:** Automotive systems (e.g., ABS), home appliances, medical devices, industrial machines.

Virtualized Computing Environment

- **Description:** Virtualization involves creating virtual versions of physical computing resources, such as servers, storage devices, or networks.
- **Use Cases:** Data center optimization, testing environments, cloud

services.

2. (c). What is system calls? List and explain the different types of system calls.(10M) exp-4,Types-3,Eg-3

Answer:

System calls are fundamental interfaces between a running program (application) and the operating system. They provide a controlled entry point for programs to request services from the kernel, such as performing I/O operations, creating processes, and managing memory. System calls are essential for ensuring that user programs operate within the protected environment of the operating system, maintaining security and stability.

Types of System Calls

System calls can be categorized based on the functionality they provide. The main types are:

1. Process Control

- **Description:** These system calls manage processes, allowing programs to create, execute, and terminate processes.
- **Examples:**
  - **fork()**: Creates a new process by duplicating the existing process.
  - **exec()**: Replaces the current process image with a new one, typically used to run a new program.
  - **exit()**: Terminates a process, returning a status code to the parent process.
  - **wait()**: Causes a process to wait until one of its child processes terminates.

2. File Management

- **Description:** These system calls handle operations related to files and directories, such as creating, deleting, reading, writing, and closing files.
- **Examples:**
  - **open()**: Opens a file, returning a file descriptor that can be used for further operations.
  - **read()**: Reads data from a file into a buffer.
  - **write()**: Writes data from a buffer to a file.
  - **close()**: Closes an open file, releasing the file descriptor.
  - **unlink()**: Deletes a file from the filesystem.

3. Device Management

- **Description:** These system calls manage and interact with hardware devices such as disks, printers, and network interfaces.
- **Examples:**
  - **ioctl()**: Stands for input/output control; used for device-specific operations that are not covered by standard system calls.

    - **read() and write()**: Though typically used for files, these calls can also be used to interact with devices.
    - **mknod()**: Creates a device file, allowing user programs to interact with devices through the filesystem.

4. Information Maintenance

- **Description:** These system calls retrieve and manage information about the system, processes, files, and users.
- **Examples:**
    - **getpid()**: Returns the process ID of the calling process.
    - **getppid()**: Returns the parent process ID.
    - **uname()**: Returns information about the system, such as the operating system name, version, and architecture.
    - **getuid() and getgid()**: Return the user ID and group ID of the calling process.

5. Communication

- **Description:** These system calls enable processes to communicate with each other, either within the same machine or over a network.
- **Examples:**
    - **pipe()**: Creates a unidirectional data channel (pipe) that can be used for communication between processes.
    - **shmget()**: Allocates a shared memory segment.
    - **msgget() and msgsnd()**: Used for message passing between processes.
    - **socket()**: Creates a communication endpoint for network communication.
    - **send() and recv()**: Used to send and receive messages over a socket.

6. Protection

- **Description:** These system calls manage access permissions for files, processes, and system resources, ensuring security and controlled access.
- **Examples:**
    - **chmod()**: Changes the access permissions of a file or directory.
    - **chown()**: Changes the owner and group of a file or directory.
    - **umask()**: Sets the default file creation permissions.
    - **setuid() and setgid()**: Set the user ID and group ID for a process, allowing for privilege escalation or de-escalation.

MODULE - 2

3. (a). Define process. Explain different states of a process with state diagram. (8M) exp-4,eg-2,diag-2

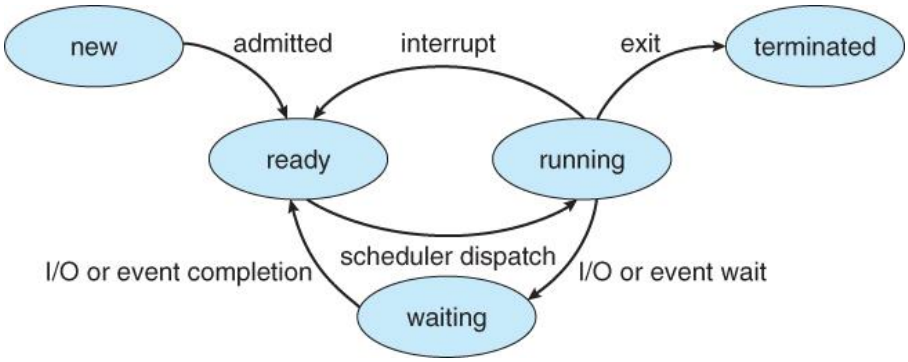
Answer:

A **process** is a program in execution. It is more than just the program code; it includes the program counter, registers, variables, memory, and all the other resources needed to execute the program. A process is an active entity, whereas a program is a passive collection of instructions. When a program is loaded into memory and begins execution, it becomes a process.

States of a Process

A process moves through various states during its lifecycle, from creation to termination. The key states include:

1. **New**: The process is being created.
2. **Ready**: The process is waiting to be assigned to a processor. It is ready to run but is not currently executing.
3. **Running**: The process is currently being executed by the processor.
4. **Waiting** (or Blocked): The process is waiting for some event to occur (like I/O completion or a resource becoming available).
5. **Terminated**: The process has finished execution and is being removed from the system.



A process can transition between these states based on events like scheduling decisions, I/O operations, or interruptions. The state diagram below illustrates these transitions:

- **New → Ready**: When a process is created, it enters the Ready state after being admitted by the OS.
- **Ready → Running**: When the scheduler picks the process for execution, it transitions to the Running state.
- **Running → Waiting**: If the process needs to wait for an I/O operation or a resource, it moves to the Waiting state.
- **Waiting → Ready**: When the event the process was waiting for occurs (e.g., I/O completion), it returns to the Ready state.
- **Running → Ready**: If the running process is preempted by the scheduler (e.g., when the time slice expires), it goes back to the Ready state.
- **Running → Terminated**: When the process finishes its execution, it

moves to the Terminated state.

3. (b). What is IPC? Explain direct and indirect states of a process with respect to message passing.(8M) - Exp-4,types-2,eg-2.

Answer:

Inter-process Communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions. IPC is essential for processes running in a multitasking environment to exchange data, signals, or messages. It helps in coordination, sharing data, and managing dependencies between processes.

Message Passing in IPC

Message passing is one of the key IPC mechanisms where processes communicate by sending and receiving messages. Message passing can be done in two primary ways: **Direct** and **Indirect** communication.

1. Direct Communication

In direct communication, processes exchange messages by addressing each other explicitly. The sender of the message specifies the recipient process by its identifier (PID).

Characteristics:

- **Direct Addressing:** Messages are sent directly from one process to another using the recipient's process identifier (PID).
- **Symmetric Communication:** Both the sender and the receiver need to explicitly name each other to communicate.
- **Simplicity:** The communication is straightforward, with a direct link between communicating processes.
- **Tight Coupling:** Processes are tightly coupled, as they need to know each other's identifiers to exchange messages.

Example:

- Process A sends a message to Process B directly using Process B's identifier.
- Process B receives the message from Process A, knowing that it came from Process A.

2. Indirect Communication

In indirect communication, processes exchange messages through an intermediary, often called a mailbox or message queue. The messages are sent to and received from these mailboxes rather than directly between processes.

Characteristics:

- **Mailbox-Based:** Messages are sent to a mailbox, which acts as an intermediary, and other processes retrieve messages from this mailbox.

- **Asymmetric Communication:** The sender and receiver do not need to know each other's identifiers; they only need to know the mailbox identifier.
- **Decoupling:** Processes are loosely coupled since they communicate indirectly through a mailbox.
- **Flexibility:** Multiple processes can send messages to the same mailbox, and multiple processes can retrieve messages from the same mailbox.

Example:

- Process A sends a message to a mailbox (Mailbox M).
- Process B retrieves the message from Mailbox M without knowing it came from Process A.

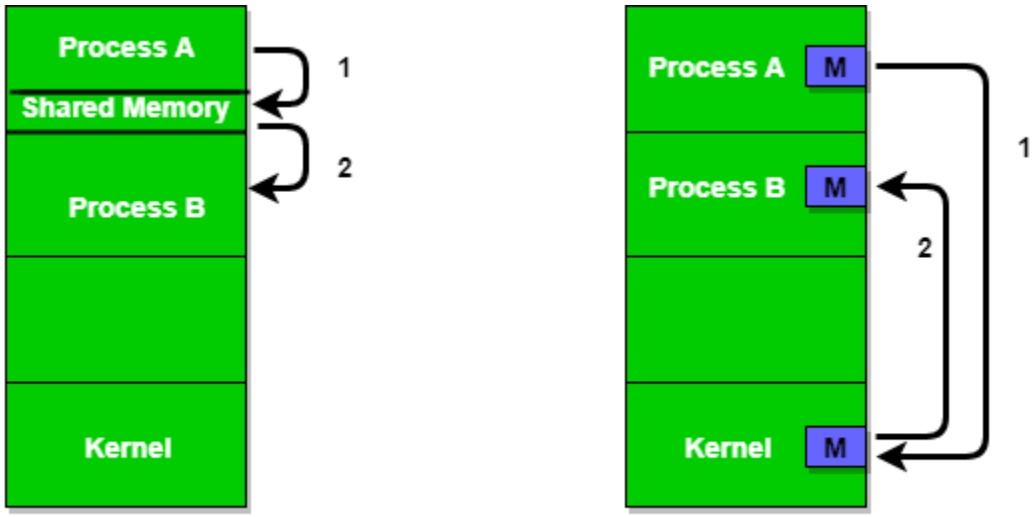


Figure 1 - Shared Memory and Message Passing

3. (c). Explain context-switching?(4M) – Exp-2,Eg-2

Answer:

Context switching is the process by which an operating system (OS) pauses the execution of one process and switches the CPU to another process. This is a fundamental feature of multitasking operating systems, allowing multiple processes to share a single CPU effectively.

Why Context Switching is Necessary

In a multitasking environment, multiple processes are often ready to run, but only one process can use the CPU at a time. The OS must manage the CPU's time among all the active processes. When the OS decides to switch the CPU from one process to another, it performs a context switch.

1. **Process Context:**
   - The context of a process includes all the information necessary to resume the process's execution at a later time. This typically includes:

    - **CPU registers** (including the program counter, stack pointer, and general-purpose registers)
    - **Process Control Block (PCB):** A data structure that contains important information about the process, such as its state, program counter, CPU registers, memory management information, and I/O status.

2. **When Does Context Switching Occur?**
    - **Multitasking:** When the operating system switches between processes to ensure that multiple processes can run seemingly simultaneously.
    - **Interrupts:** When an interrupt occurs (e.g., an I/O device needs attention), the current process's context is saved, and the context of the interrupt handler or another process is loaded.
    - **System Calls:** During certain system calls, context switching might occur if the call involves waiting for a resource or I/O operation.
    - **Time Slicing:** In time-sharing systems, the CPU is shared among processes by dividing time into slices. When a process's time slice expires, a context switch occurs.
3. **Steps in a Context Switch:**
    - **1. Save State of Current Process:** The current state of the running process is saved to its PCB. This includes saving all CPU registers, program counter, and other critical information.
    - **2. Load State of Next Process:** The state of the next process to be executed is loaded from its PCB into the CPU registers.
    - **3. Update Process Control Structures:** The process control structures are updated to reflect the switch, including marking the old process as not running and the new process as running.
    - **4. Resume Execution:** The CPU resumes execution of the newly loaded process from where it was last stopped.

4. (a). What is multi-threaded process? Explain the four benefits of multithreaded programming.(6M), Exp-2,Eg-2,Types-2.

Answer:

A multi-threaded process is a process that contains multiple threads of execution. Each thread represents a separate path of execution, and all threads within a process share the same memory space and resources. This allows them to execute concurrently, either on multiple processors/cores or by being scheduled in time-slices on a single processor.

Benefits of Multithreaded Programming

Multithreaded programming offers several advantages, particularly in terms of performance, responsiveness, and resource utilization. Here are four key benefits:

1. Responsiveness

- Multithreading allows an application to remain responsive even when performing lengthy operations. For example, in a user interface (UI) application, one thread can handle user inputs, while another thread performs background tasks like data processing or file I/O.
- Users experience a more interactive and responsive application, as the UI remains active even when the application is busy with other tasks.
- **Example:** In a word processor, the main thread can handle user inputs (like typing), while another thread performs spell checking in the background, ensuring that the application doesn't freeze.

2. Resource Sharing

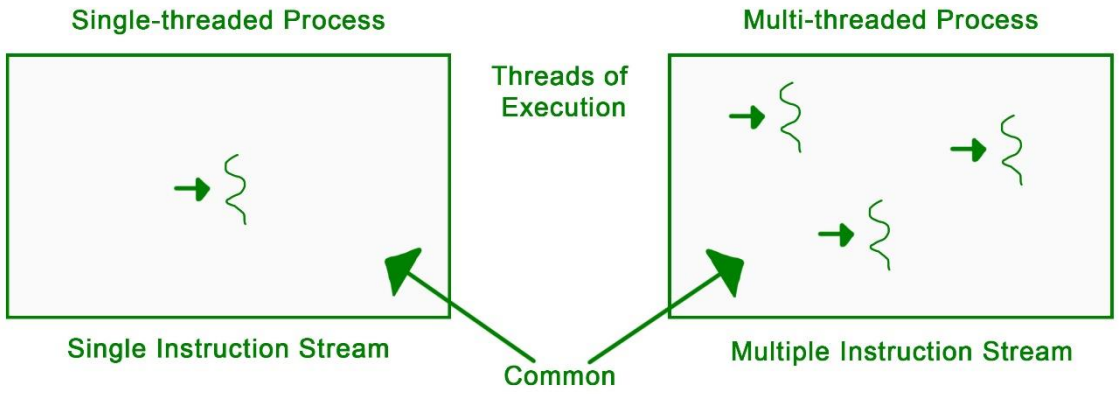
- Since threads within the same process share the same address space and resources, they can easily communicate and share data without the need for complex inter-process communication (IPC) mechanisms.
- This leads to more efficient use of memory and other resources, reducing overhead and improving performance.
- **Example:** In a web server, multiple threads can handle multiple client requests simultaneously, sharing the same cache and database connections.

3. Scalability and Improved Performance

- In systems with multiple processors or cores, multithreading allows a process to run multiple threads in parallel, effectively utilizing available CPU resources. This parallelism can lead to significant performance improvements, especially for compute-intensive tasks.
- Multithreaded applications can achieve higher throughput and reduced execution time, as tasks are performed in parallel rather than sequentially.
- **Example:** In scientific computing, a multi-threaded simulation can split large computations across multiple cores, dramatically speeding up the processing time.

4. Simplified Design and Maintenance

- Multithreading can simplify the design of some applications by allowing separate threads to handle different tasks concurrently. This separation of concerns can make the codebase more modular and easier to maintain.
- By dividing complex tasks into simpler, concurrent threads, developers can manage and update the code more easily, reducing the chances of errors and making it easier to introduce new features.
- **Example:** In a network application, one thread can handle incoming network requests, another can process data, and a third can manage outgoing responses, making the overall application logic cleaner and more maintainable.



Single Thread and Multi Thread Process

4. (b). Calculate average waiting time and average turnaround times by drawing the Gantt chart using FCFS, SJF-non preemptive, SRTF, RR (q=2ms) and priority algorithms. (14 M),Exp=4,Gantchart-5,Cal-5

Process	Arrival time	Burst time	Porosity
P1	0	9	3
P2	1	4	2
P3	2	9	1
P4	3	5	4

First Come First Serve (FCFS):

- Processes are executed in the order of their arrival times.

2. Shortest Job First (SJF) Non-preemptive:

- The process with the smallest burst time is selected, and once it starts, it runs until completion.

3. Shortest Remaining Time First (SRTF):

- This is a preemptive version of SJF. At each time unit, the process with the smallest remaining burst time is selected.

4. Round Robin (RR) (Quantum = 2ms):

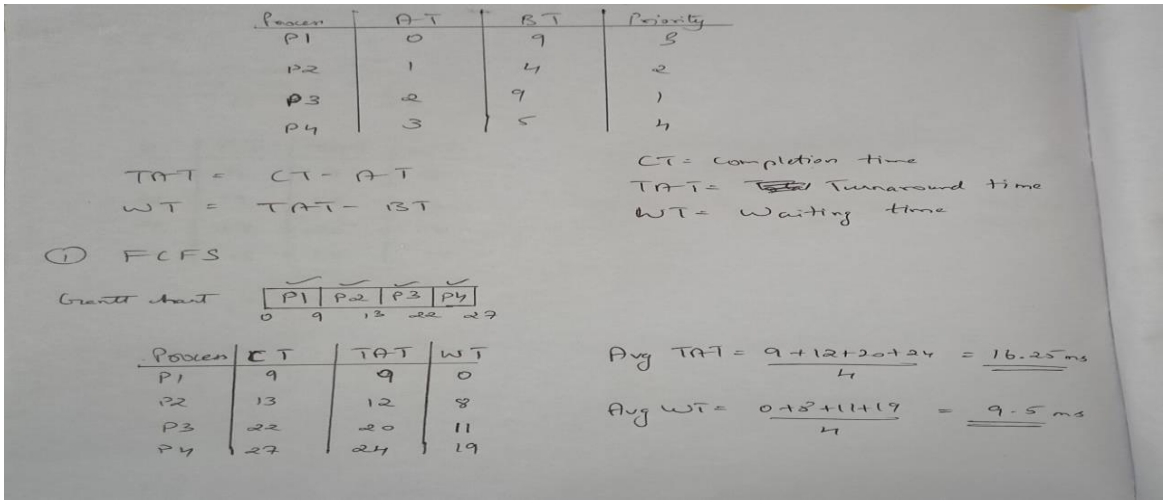
- Each process gets 2ms of CPU time in a circular order until all processes are completed.

5. Priority Scheduling (Non-preemptive):

- The process with the highest priority is selected. In the given table, the "porosity" column represents the priority (lower value indicates higher priority).

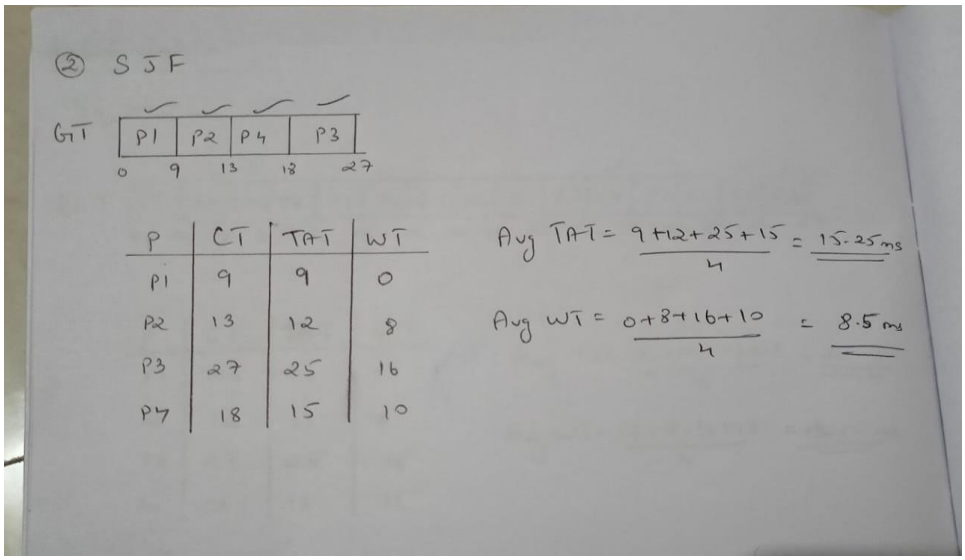
First Come First Serve (FCFS):

- **Average Waiting Time:** 9.5 ms
- **Average Turnaround Time:** 16.25 ms

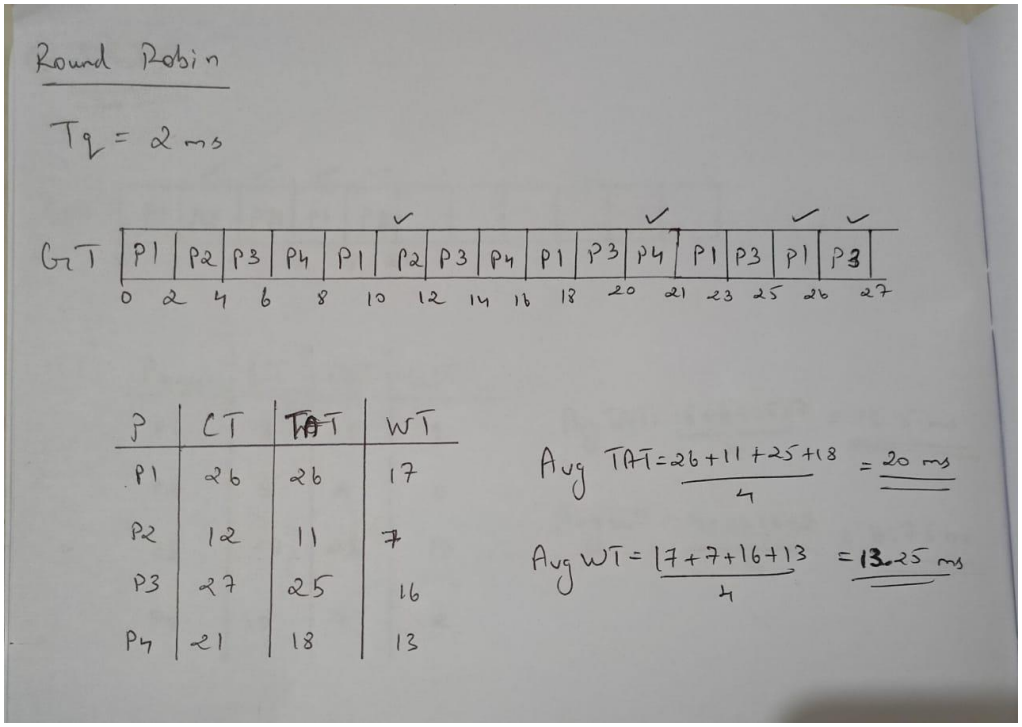


SJF Non-preemptive (Shortest Job First)

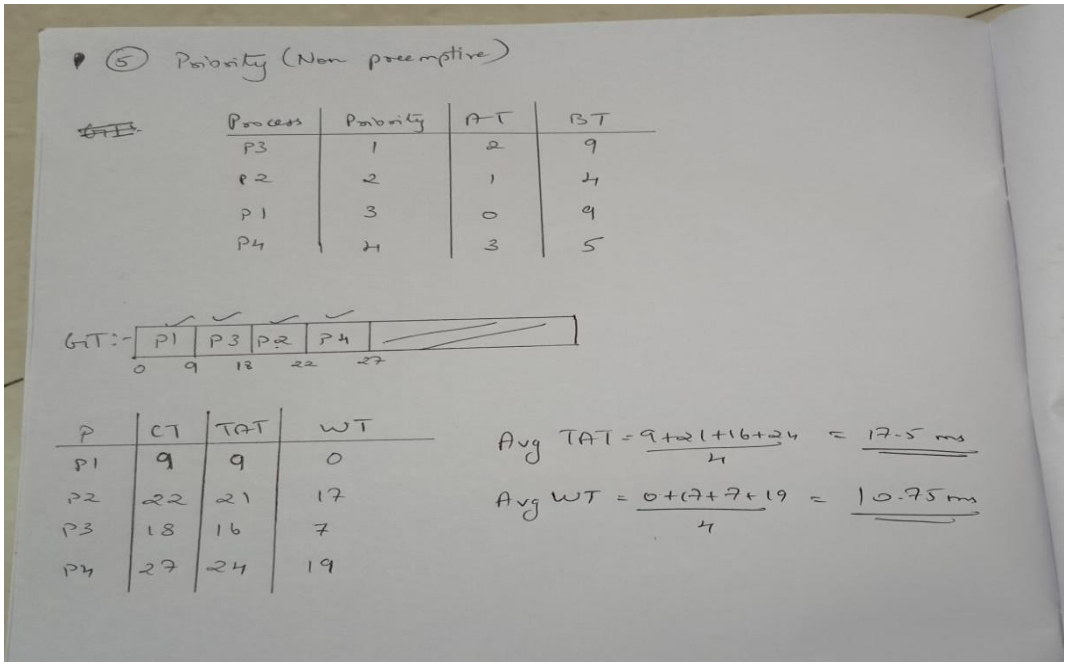
- **Average Waiting Time:** 8.5 ms
- **Average Turnaround Time:** 15.25 ms



Round Robin (RR) (Quantum = 2ms):



Priority Scheduling (Non-preemptive):



Module-3

Q5

a) What is critical section? What are the requirements for the solution to critical section problem? Explain Peterson's solution. (8M)- Exp-4,Eg-2,code-2.

Ans:

The Critical Section Problems

• Consider a system consisting of n processes {Po, P1 , ... ,Pn-1}.

• Each process has a segment of code, called a **critical section** in which the process may be changing common variables, updating a table, writing a file, and soon

• The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. That is, no two processes are executing in their critical sections at the same time.

• The critical-section problem is to design a protocol that the processes can use to cooperate.

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

1.Mutual exclusion: If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.

2.**Progress:** If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.

3. **Bounded waiting:** There exists a bound, or limit, on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Peterson's problem

A classic software-based solution to the critical-section problem known as Peterson's solution. Peterson's solution is restricted to two processes that alternate execution between their critical sections and remainder sections. The processes are numbered P0 and Pl. For convenience, when presenting Pi, we use Pj to denote the other process; that

is, j equals 1 - i. Peterson's solution requires two data items to be shared between the two processes: int turn; boolean flag[2]; The variable turn indicates whose turn it is to enter its critical section. That is, if turn == i, then process Pi is allowed to execute in its critical section.The flag array is used to indicate if a process is ready to enter its critical section.

For example, if flag [i] is true, this value indicates that Pi is ready to enter its critical section. With an explanation of these data structures complete, we are now ready to describe the algorithm shown in Figure below.

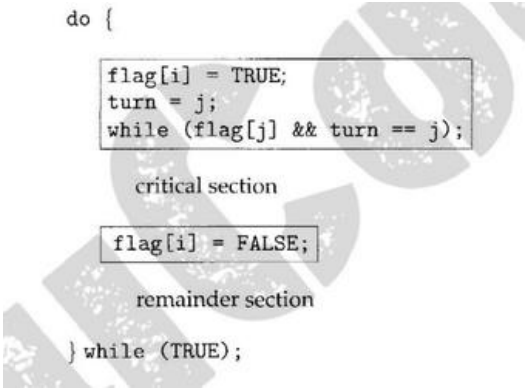


Figure: The structure of process Pi in Peterson's solution.

To enter the critical section, process Pi first sets flag [i] to be true and then sets turn to the value j, thereby asserting that if the other process wishes to enter the critical section it can do so. If both processes try to enter at the same time, turn will be set to both i and j at roughly the same time. Only one of these assignments will last; the other will occur, but will be overwritten immediately.

To prove property Mutual exclusion is preserved, we note that each Pi enters its critical section only if either flag [j] == false or turn == i. Also note that, if both processes can be executing in their critical sections at the same time, then flag [i] ==flag [j] == true. These two observations imply that P0 and P1 could not have successfully executed their while statements at about the same time, since the value of turn can be either 0 or 1, but cannot be both.

Q5

b) Explain Reader's-Writer's problem using semaphores(12M)-Exp-4, code-8.

Ans:

• A data set is shared among a number of concurrent processes

• Readers – only read the data set; they do **not** perform any updates

• Writers – can both read and write.

• Problem – allow multiple readers to read at the same time. Only one single writer can access the shared data at the same time.

- Shared Data
- Dataset
- Semaphore **mutex** initialized to 1.
- Semaphore **wrt** initialized to1.
- Integer **readcount** initialized to 0.

```
while (true)
{
        wait (wrt) ;
        // writing is performed
        signal (wrt) ;
}
```

The structure of a writerprocess

```
while (true)
{
        wait (mutex) ;
        readcount ++ ;
        if (readcount == 1)
                wait (wrt) ;
        signal (mutex)
        // reading is performed
        wait (mutex) ;
        readcount - - ;
        if (readcount == 0)
                signal (wrt) ;
        signal (mutex) ;
}
```

The structure of a readerprocess

Q6

a) **Define deadlock. What are the necessary conditions for deadlock to occur?(6M)-Exp-3,eg-3.**

Ans:

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

process requests resources; and if the resources are not available at that time, the process enters a waiting

state. Sometimes, a waiting process is never again able to change state, because the resources it has

requested are held by other waiting processes. This situation is called a deadlock.

In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs

from starting necessary conditions. A deadlock situation can arise if the following four conditions hold

simultaneously in a system:

❖ Mutual Exclusion Condition: At least one resource must be held in a non-shareable mode, that

is, only one process at a time claims exclusive control of the resource. If another process requests

that resource, the requesting process must be delayed until the resource has been released.

❖ Hold and Wait Condition: Requesting process hold already the resources while waiting for

requested resources. There must exist a process that is holding a resource already allocated to it

while waiting for additional resource that are currently being held by other processes.

❖ No-Premptive Condition: Resources already allocated to a process cannot be prempted.

Resources cannot be removed from the processes are used to completion or released voluntarily

by the process holding it.

❖ Circular Wait Condition: The processes in the system form a circular list or chain where each

process in the list is waiting for a resource held by the next process in the list. There exists a set

{P0, P1, ..., P0} of waiting processes such that P0 is waiting for a resource that is held by P1, P1

is waiting for a resource that is held by P2, ..., Pn–1 is waiting for a resource that is held by Pn,

and P0 is waiting for a resource that is held by P0.

Q6 (14M) ,bankers alg-4.safte state-4.cal-6

b. Consider the following snap-shot of a system:

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	2	0	0	1	4	2	1	2	3	3	2	1
P1	3	1	2	1	5	2	5	2				
P2	2	1	0	3	2	3	1	6				
P3	1	3	1	2	1	4	2	4				
P4	1	4	3	2	3	6	6	5				

Answer the following using Banker's algorithm:

i) Is the system in safe state? If so give the safe sequence.

ii) If process P2 requests (0, 1, 1, 3) resource can it be granted immediately.

Solution:

a) Since, Need = Max − Allocation, the content of Need is

A	B	C	D
0	0	0	0
0	7	5	0
1	0	0	2
0	0	2	0
0	6	4	2

b) Yes, the sequence <P0, P2, P1, P3, P4> satisfies the safety requirement.

c) Yes. Since

i. (0,4,2,0) _ Available = (1,5,2,0)
ii. (0,4,2,0) _ Maxi = (1,7,5,0)
iii. The new system state after the allocation is made is

	Allocation	*Max*	*Need*	*Available*
	A B C D	*A B C D*	*A B C D*	*A B C D*
P0	0 0 1 2	0 0 1 2	0 0 0 0	1 1 0 0
P1	1 4 2 0	1 7 5 0	0 3 3 0	
P2	1 3 5 4	2 3 5 6	1 0 0 2	
P3	0 6 3 2	0 6 5 2	0 0 2 0	
P4	0 0 1 4	0 6 5 6	0 6 4 2	

and the sequence < P0, P2, P1, P3, P4 > satisfies the safety requirement.

Module-4

Q.07

a) What is paging ? Explain with neat diagram paging hardware with TLB? 10M (Exp-4,Eg-3,Diag-3)

Ans :

Paging

• Paging is a memory-management scheme.

• This permits the physical-address space of a process to be non-contiguous.

• This also solves the considerable problem of fitting memory-chunks of varying sizes

onto the backing-store.

• Traditionally: Support for paging has been handled by hardware.

• Recent designs: The hardware & OS are closely integrated.

TLB

The standard solution to this problem is to use a special, small, fast lookup hardware cache, called a translation look-aside buffer (TLB).

❖ The TLB is associative, high-speed memory. Each entry in the TLB consists of two parts: a key(or tag) and a value.

❖ When the associative memory is presented with an item, the item is compared with all keys simultaneously.

❖ The TLB is used with page tables in the following way. The TLB contains only a few of the page-

table entries. When a logical address is generated by the CPU, its page number is presented to the TLB.

❖ If the page number is found, its frame number is immediately available and is used to access memory.

❖ The whole task may take less than 10 percent longer than it would if an unmapped memory reference were used.

❖ If the page number is not in the TLB (known as a TLB miss), a memory reference to the page table must be made. When the frame number is obtained, we can use it to access memory (Figure below).

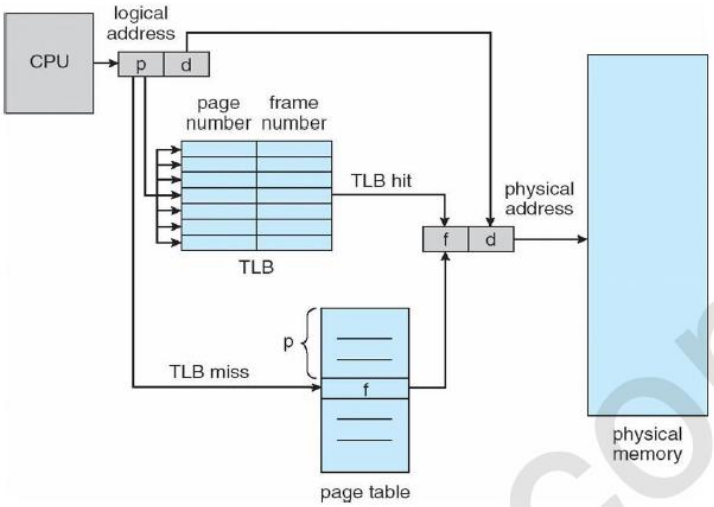


Figure 1: Paging hardware with TLB

Q.7

b) What are the commonly used strategies to select a free hole from the available hole ?(6M) -Exp-3,Eg-3

Ans:

Three strategies used to select a free hole from the set of available holes:

1. **First Fit**: Allocate the first hole that is big enough. Searching can start either at the

beginning of the set of holes or at the location where the previous first-fit search ended.

2. **Best Fit**: Allocate the smallest hole that is big enough. We must search the entire list,

unless the list is ordered by size. This strategy produces the smallest leftover hole.

3. **Worst Fit**: Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole.

First-fit and best fit are better than worst fit in terms of decreasing time and storage utilization.

Q.7

c) Explain fragmentation in detail(4 M)- Exp-2,Eg-2

Ans:

Two types of memory fragmentation:

1. Internal fragmentation

2. External fragmentation

1. Internal Fragmentation

• The general approach is to break the physical-memory into fixed-sized blocks and allocate memory in units based on block size.

• The allocated-memory to a process may be slightly larger than the requested-memory.

• The difference between requested-memory and allocated-memory is called internal fragmentation i.e. Unused memory that is internal to a partition.

2. External Fragmentation

• External fragmentation occurs when there is enough total memory-space to satisfy a request but the available-spaces are not contiguous. (i.e. storage is fragmented into a large number of small holes).

• Both the first-fit and best-fit strategies for memory-allocation suffer from external fragmentation.

• Statistical analysis of first-fit reveals that given N allocated blocks, another 0.5 N blocks will be lost to fragmentation. This property is known as the 50-percent rule.

Two solutions to external fragmentation:

• **Compaction**: The goal is to shuffle the memory-contents to place all free memory together in one large hole. Compaction is possible only if relocation is dynamic and done at execution-time

• Permit the logical-address space of the processes to be non-contiguous. This allows a process to be allocated physical-memory wherever such memory is available. Two techniques achieve this solution: 1) Paging and 2) Segmentation.

Q.8

a) With a neat diagram Describe the steps in handling the page fault.(8M)

Exp-4.Eg-4.

Ans:

If a page is needed that was not originally loaded up, then a ***page fault trap*** is generated.

Steps in Handling a Page Fault

1. The memory address requested is first checked, to make sure it was a valid memory

request.

2. If the reference is to an invalid page, the process is terminated. Otherwise, if the page is

not present in memory, it must be paged in.

3. A free frame is located, possibly from a free-frame list.

4. A disk operation is scheduled to bring in the necessary page from disk.

5. After the page is loaded to memory, the process's page table is updated with the new

frame number, and the invalid bit is changed to indicate that this is now a valid page

reference.

6. The instruction that caused the page fault must now be restarted from the beginning.

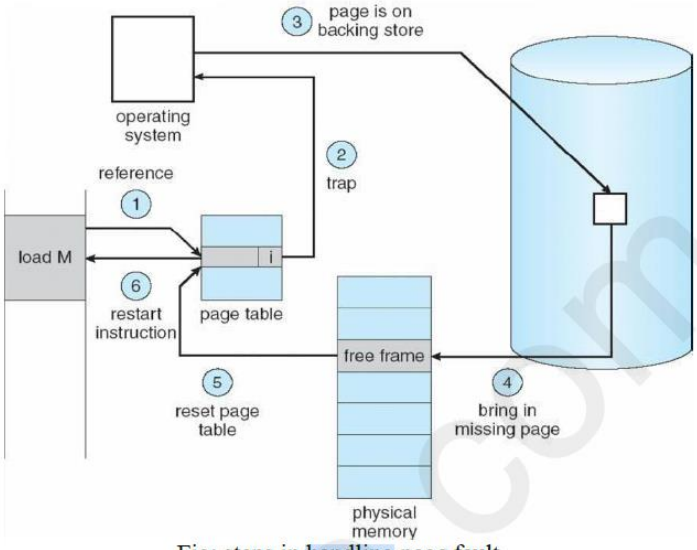


Fig: steps in handling page fault

Q.8

b) Consider the page reference string: 1,0,7,1,0,2,1,2,3,0,3,2,4,0,3,6,2,1 for a memory with 3 frames. Determine the number of page faults using FIFO, optimal and LRU replacement algorithms which algorithm is more efficient.

(12M)- Exp-6,Cal-6

Solution:

FIFO

Pages	1	0	7	1	0	2	1	2	3	0	3	2	4	0	3	6	2	1
f_1	1	1	1	1	1	2	2	2	2	0	0	0	0	0	3	3	3	1
f_2		0	0	0	0	0	1	1	1	1	1	2	2	2	2	6	6	6
f_3			7	7	7	7	7	7	3	3	3	3	4	4	4	4	2	2
	x	x	x	✓	✓	x	x	✓	x	x	✓	x	x	✓	x	x	x	x

No. of fault = 13

Optimal

Pages	1	0	7	1	0	2	1	2	3	0	3	2	4	0	3	6	2	1
f_1	1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	6	6	6
f_2		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2
f_3			7	7	7	2	2	2	2	2	2	2	4	4	4	4	4	1
	x	x	x	✓	✓	x	✓	✓	x	✓	✓	✓	x	✓	✓	x	x	x

No. of fault = 9

LRU

Pages	1	0	7	1	0	2	1	2	3	0	3	2	4	0	3	6	2	1
f_1	1	1	1	1	1	1	1	1	1	0	0	0	4	4	4	6	6	6
f_2		0	0	0	0	0	0	0	3	3	3	3	3	0	0	0	2	2
f_3			7	7	7	2	2	2	2	2	2	2	2	2	3	3	3	1
	x	x	x	✓	✓	x	✓	✓	x	x	✓	✓	x	x	x	x	x	x

No. of fault = 12

Q9)

a)Define file. List and explain the different file attributes and operations.(10M) - Exp-5,Types-5

Answer:

A file is a collection of related information that is stored on secondary storage. Information stored in files must be persistent i.e. not affected by power failures & system reboots. Files may be of free from such as text files or may be formatted rigidly. Files represent both programs as well as data. Part of the OS dealing with the files is known as file system. The important file concepts include:

File attributes: A file has certain attributes which vary from one operating system to another.

• Name: Every file has a name by which it is referred.

• Identifier: It is unique number that identifies the file within the file system.

• Type: This information is needed for those systems that support different types of files.

• Location: It is a pointer to a device & to the location of the file on that device • Size: It is the current size of a file in bytes, words or blocks.

• Protection: It is the access control information that determines who can read, write & execute a file.

• Time, date & user identification: It gives information about time of creation or last modification & last use.

File operations: The operating system can provide system calls to create, read, write, reposition, delete and truncate files.

• Creating files: Two steps are necessary to create a file. First, space must be found for the file in the file system. Secondly, an entry must be made in the directory for the new file.

• Reading a file: Data & read from the file at the current position. The system must keep a read pointer to know the location in the file from where the next read is to take place. Once the read has been taken place, the read pointer is updated.

• Writing a file: Data are written to the file at the current position. The system must keep a write pointer to know the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.

• Repositioning within a file (seek): The directory is searched for the appropriate entry & the current file position is set to a given value. After repositioning data can be read from or written into that position.

• Deleting a file: To delete a file, we search the directory for the required file. After deletion, the space is releasedso that it can be reused by other files.

• Truncating a file: The user may erase the contents of a file but allows all attributes to remain unchanged expect the file length which is rest to 'O' & the space is released.

b) Explain the different file allocation methods 10M (Exp-5,Eg-5)

Answer:

Types of File Allocation Methods in Operating System.

- Contiguous File allocation
- Linked File Allocation
- Indexed File Allocation
- File Allocation Table (FAT)
- Inode

Let's have an in-detail explanation about each of them,

Contiguous File Allocation.

First, let's understand the meaning of contiguous, here contiguous means adjacent or touching. Now let's understand what is contiguous file allocation.

What is Contiguous File allocation?

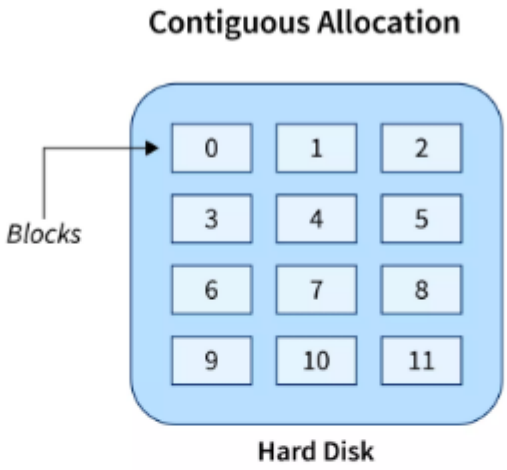
In contiguous file allocation, the block is allocated in such a manner that all the allocated blocks in the hard disk are adjacent.

Assuming a file needs 'n' number of blocks in the disk and the file begins with a block at position'x', the next blocks to be assigned to it will be x+1,x+2,x+3,...,x+n-1 so that they are in a contiguous manner.

Let's understand this diagrammatically.

Example

We have three different types of files that are stored in a contiguous manner on the hard disk.



File Name	Start	Length	Allocated Blocks
file1.txt	0	4	0,1,2,3
sun.jpg	5	3	5,6,7
mov.mp4	9	3	9,10,11

Directory

At first, we have a text file named file1.txt which is allocated using contiguous memory allocation, it starts with the memory block 0 and has a length of 4 so it takes the 4 contiguous blocks 0,1,2,3. Similarly, we have an image file and video file named sun.jpg and mov.mp4 respectively, which you can see in the directory that they are stored in the contiguous blocks. 5,6,7 and 9,10,11 respectively.

Advantages

It is very easy to implement.

There is a minimum amount of seek time.

The disk head movement is minimum.

Memory access is faster.

It supports sequential as well as direct access.

Disadvantages

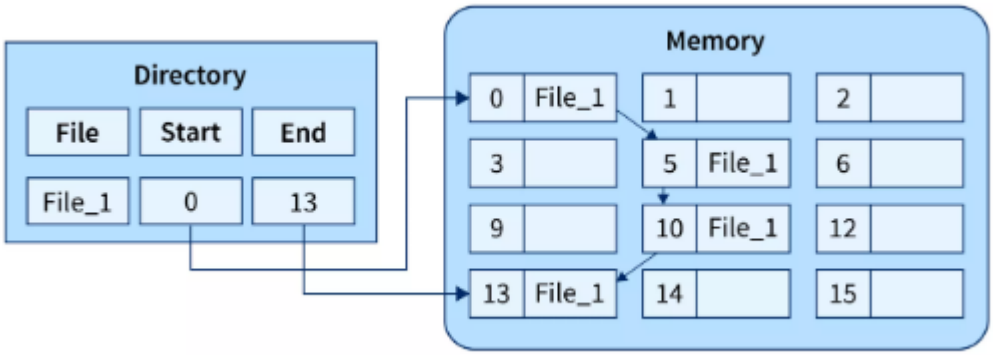
At the time of creation, the file size must be initialized.

As it is pre-initialized, the size cannot increase. As

Due to its constrained allocation, it is possible that the disk would fragment internally or externally.

Linked File Allocation.

The Linked file allocation overcomes the drawback of contiguous file allocation. Here the file which we store on the hard disk is stored in a scattered manner according to the space available on the hard disk. Now, you must be thinking about how the OS remembers that all the scattered blocks belong to the same file. So as the name linked File Allocation suggests, the pointers are used to point to the next block of the same file, therefore along with the entry of each file each block also stores the pointer to the next block.



In this allocation, the starting block given is 0 and the ending block is 15, therefore the OS searches the empty blocks between 0 and 15 and stores the files in available blocks, but along with that it also stores the pointer to the next block in the present block. Hence it requires some extra space to store that link.

Advantages

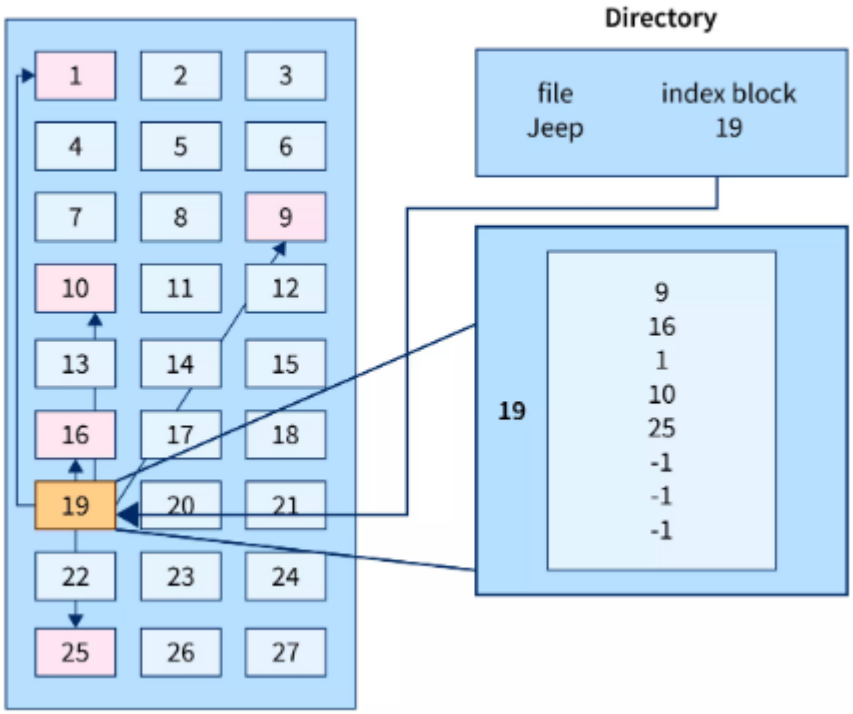
- There is no external fragmentation.
- The directory entry just needs the address of starting block.
- The memory is not needed in contiguous form, it is more flexible than contiguous file allocation.

Disadvantages

- It does not support random access or direct access.
- If pointers are affected so the disk blocks are also affected.
- Extra space is required for pointers in the block.

Indexed File Allocation.

The indexed file allocation is somewhat similar to linked file allocation as indexed file allocation also uses pointers but the difference is here all the pointers are put together into one location which is called index block. That means we will get all the locations of blocks in one index file. The blocks and pointers were spread over the memory in the Linked Allocation method, where retrieval was accomplished by visiting each block sequentially. But here in indexed allocation, it becomes easier with the index block to retrieve.



Advantages

It reduces the possibilities of external fragmentation.

Rather than accessing sequentially it has direct access to the block.

Disadvantages

Here more pointer overhead is there.

If we lose the index block we cannot access the complete file.

It becomes heavy for the small files.

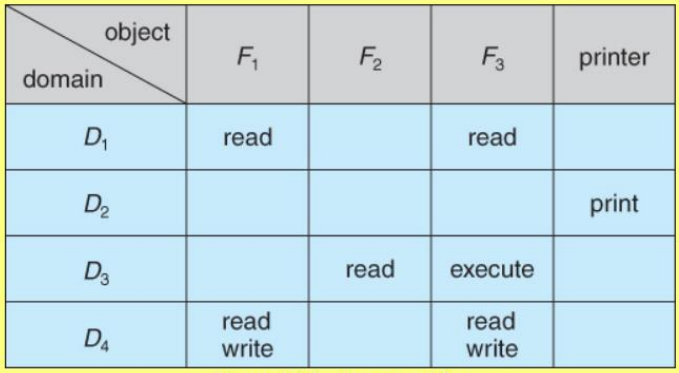
It is possible that a single index block cannot keep all the pointers for some large files.

Q10.

a)What is access Matrix ? Explain Access Matrix method of system protection with domain as objects and its implementation.(10M) Exp-3,Method-3,Imp-4

Answer:

Access Matrix is a security model of protection state in computer system. It is represented as a matrix. Access matrix is used to define the rights of each process executing in the domain with respect to each object. The rows of matrix represent domains and columns represent objects. Each cell of matrix represents set of access rights which are given to the processes of domain means each entry(i, j) defines the set of operations that a process executing in domain Di can invoke on object Oj.



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

Figure 14.3 - Access matrix.

Different types of rights:

There are different types of rights the files can have. The most common ones are:

Read- This is a right given to a process in a domain, which allows it to read the file.

Write- Process in domain can write into the file.

Execute- Process in domain can execute the file.

Print- Process in domain only has access to printer.

	F1	F2	F3	Printer	D1	D2	D3	D4
D1	read		read			switch		
D2				print			switch	switch
D3		read	execute					
D4	read write		read write		switch			

a process executing in domain D2 can switch to domain D3 and D4. A process executing in domain D4 can switch to domain D1 and process executing in domain D1 can switch to domain D2.

Implementation of Access Matrix

1.Global Table

- The simplest approach is one big global table with < domain, object, rights > entries.
- Unfortunately this table is very large (even if sparse) and so cannot be kept in memory (without invoking virtual memory techniques.)
- There is also no good way to specify groupings - If everyone has access to some resource, then it still needs a separate entry for every domain.

2.Access Lists for Objects

- Each column of the table can be kept as a list of the access rights for that particular object, discarding blank entries.
- For efficiency a separate list of default access rights can also be kept, and checked first.

3.Capability Lists for Domains

- In a similar fashion, each row of the table can be kept as a list of the capabilities of that domain.
- Capability lists are associated with each domain, but not directly accessible by the domain or any user process.
- Capability lists are themselves protected resources, distinguished from other data in one of two ways:
    - A ***tag,*** possibly hardware implemented, distinguishing this special type of data. (other types may be floats, pointers, booleans, etc.)
    - The address space for a program may be split into multiple segments, at least one of which is inaccessible by the program itself, and used by the operating system for maintaining the process's access right capability list.

4.A Lock-Key Mechanism

- Each resource has a list of unique bit patterns, termed locks.
- Each domain has its own list of unique bit patterns, termed keys.
- Access is granted if one of the domain's keys fits one of the resource's locks.
- Again, a process is not allowed to modify its own keys.

5.Comparison

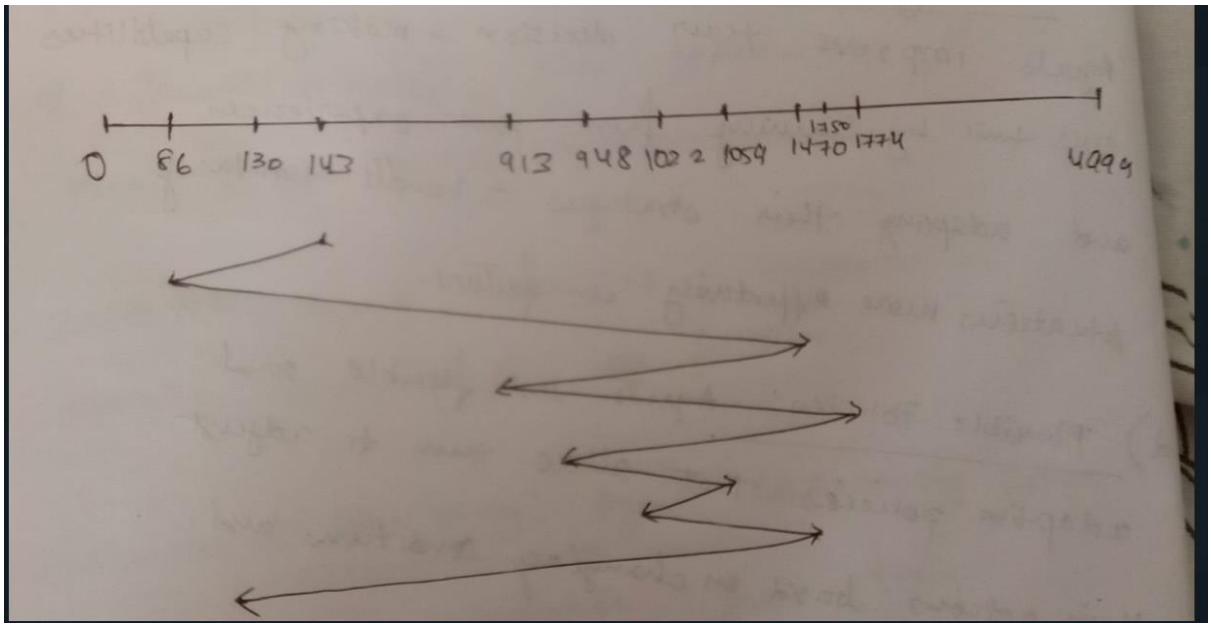
- Each of the methods here has certain advantages or disadvantages, depending on the particular situation and task at hand.
- Many systems employ some combination of the listed methods.

b) Let a disk has 5000 cylinders from 0 to 4999. Currently drive is at 143rd cylinder , and the previous request was at cylinder 125.Queue of pending request in FiFO order is 86,1470,913,1774,948,1059,1022,1750,130 starting from current head position. What is the total distance the disk arm moves to satisfy all the pending request for each of the following disk scheduling algorithms from current position 1)FCFS 2) SCAN 3)LOOK 4)SSTF 5)C-LOOK (10M)- Exp-5,Draw-5.

Answer:

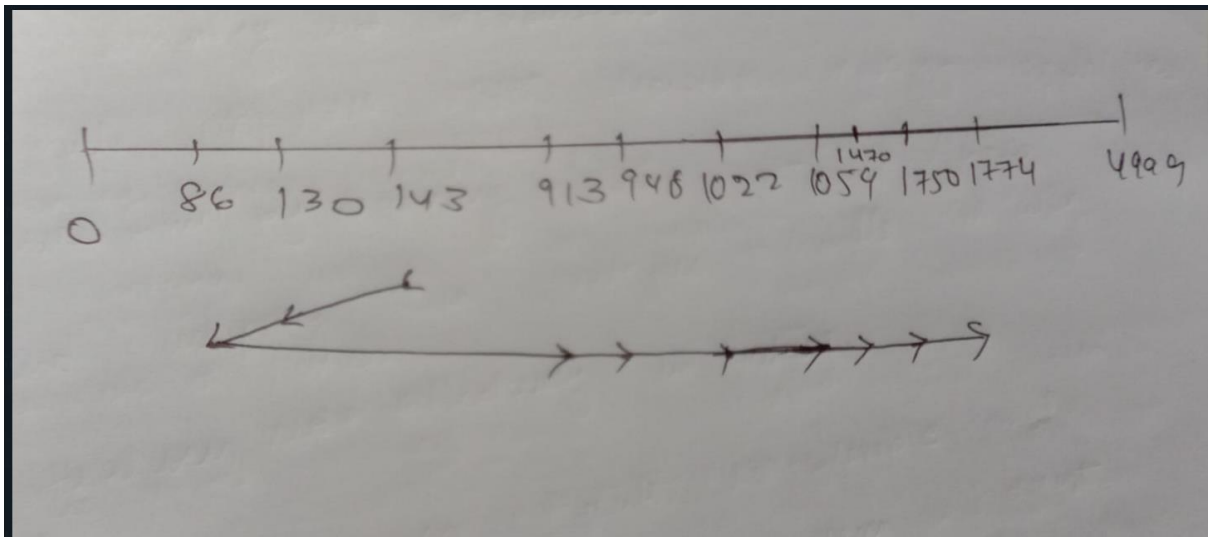
FCFS:

- The FCFS algorithm just follows the order of the requests given.
- The FCFS schedule is: 143, 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130
- **Movements:** 57 + 1384 + 557 + 861 + 826 + 561 + 487 + 728 + 1620
- **The total distance is:** 7,081



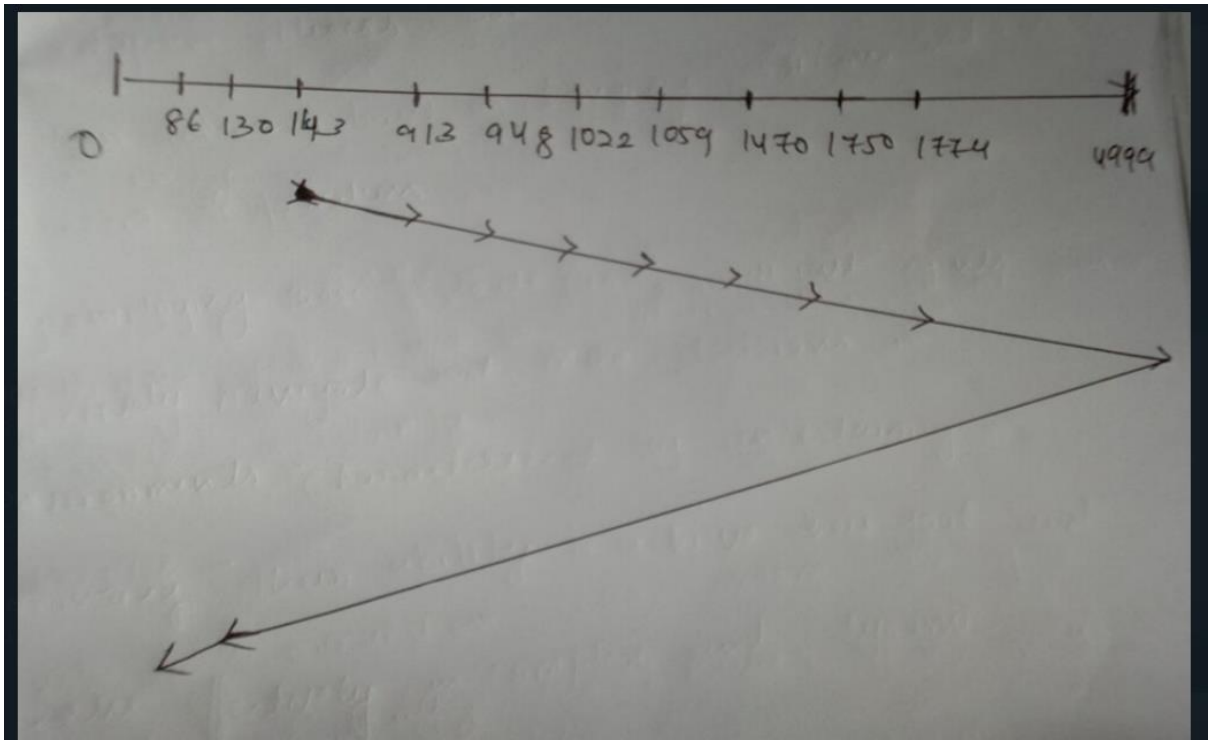
SSTF:

- The SSTF algorithm starts a cylinder 143 and from there successively selects the shortest request from its current location.
- The SSTF schedule is: 143, 130, 86, 913, 948, 1022, 1470, 1509, 1750, 1774
- **Movements:** 13 + 44 + 827 + 35 + 74 + 448 + 39 + 241 + 24
- **The total distance is:** 1,745



SCAN:

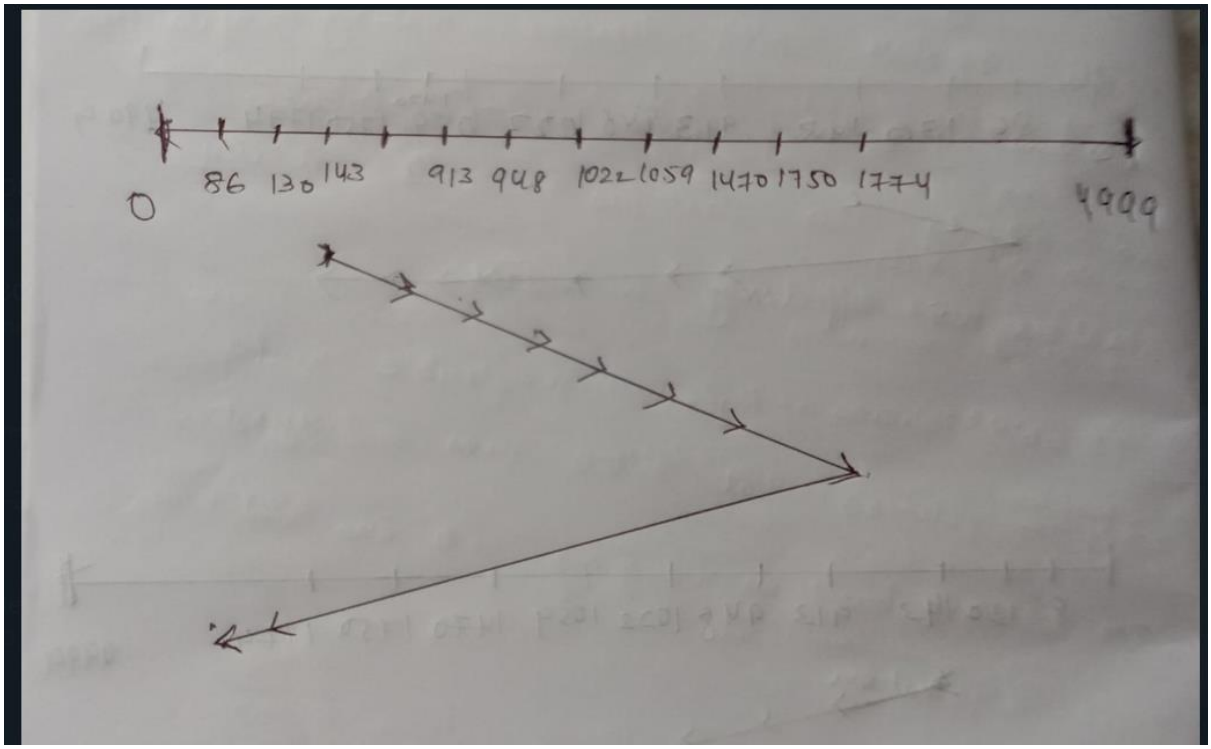
- The SCAN algorithm continues in the direction of the head servicing requests until the end of the disk. It then reverses, catching all requests back to the other end of the disk. The head continuously scans back and forth across the disk.
- The SCAN schedule is: 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 4999, 130, 86
- **Movements:** 770 + 35 + 74 + 448 + 39 + 241 + 24 + 3225 + 4869 + 44
- **The total distance is:** 9,769



LOOK:

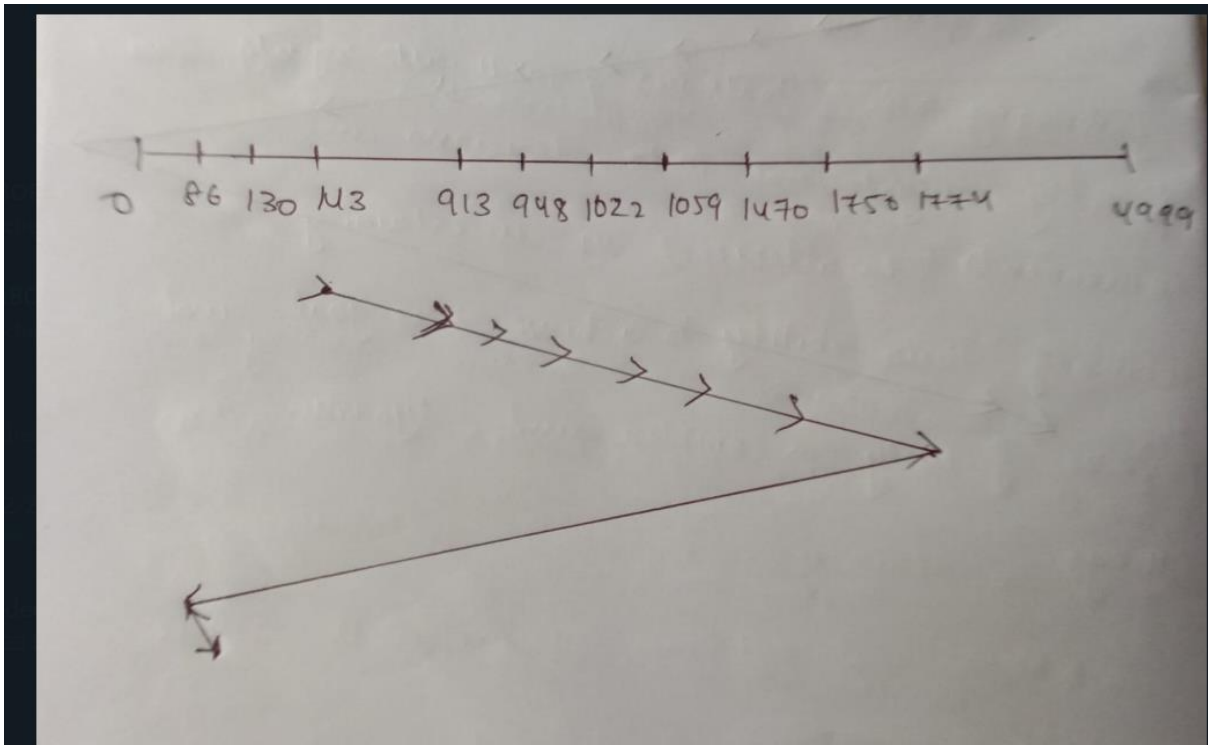
- The LOOK algorithm is just like the SCAN algorithm, except the disk head only goes as far as the last request in each direction.
- The LOOK schedule is: 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 130, 86
- **Movements:** 770 + 35 + 74 + 448 + 39 + 241 + 24 + 1644 + 44

The total distance is: 3,319



C-LOOK:

- *Like C-SCAN, except not all the way to the end*
  The look schedule is: 143 --> 913, 948, 1022, 1470, 1509, 1750, 1774, 86, 130
  Movements: 770 + 35 + 74 + 448 + 39 + 241 + 24 + 1688 + 44
  The Total Distance: 3363


0
86
130
143
913
948
1022
1059
1470
1750
4999