B.Tech. V-Sem. (Main/Back) Exam. - 2024
Operating System

PART-A

Q1. What is Kernel?
The kernel is the core component of an operating system. It acts as a bridge between applications and the hardware of a computer. The kernel manages system resources, including memory, CPU, and input/output devices. It is responsible for process management, memory management, device management, and system calls. The kernel operates in a privileged mode called kernel mode.

Q2. What do you mean by system call?
A system call is a mechanism that allows user-level processes to request services from the operating system. It provides an interface between a process and the OS kernel. Examples include file operations (read, write), process control (fork, exec), and communication (socket). System calls are essential for accessing hardware resources safely and efficiently.

Q3. What is process control block (PCB)?
A Process Control Block (PCB) is a data structure used by the operating system to store information about a process. It contains details such as process ID, process state, program counter, CPU registers, memory allocation, and I/O status. The PCB enables the OS to manage and switch between processes efficiently.

Q4. Define effective access time.
Effective Access Time (EAT) is the average time taken to access a memory location, considering both hit and miss rates in the memory hierarchy. It is calculated using:

Q5. What are the various file operations?
Common file operations include:
Create: To create a new file.
Open: To open an existing file for reading, writing, or both.
Read: To read data from a file.
Write: To write data to a file.
Close: To close an open file.
Delete: To remove a file.
Seek: To move the file pointer to a specific location within a file.

Q6. Name any five page replacement algorithms used for page replacement.
First-In-First-Out (FIFO)
Least Recently Used (LRU)
Optimal Page Replacement
Clock Algorithm
Second Chance Algorithm

Q7. What are safe and unsafe states in a deadlock?
Safe State: A state where the system can allocate resources to processes in such a way that deadlock is avoided.
Unsafe State: A state where resource allocation can lead to a deadlock if processes proceed in a certain sequence.

Q8. Define deadlock prevention.
Deadlock prevention is a set of strategies to ensure that at least one of the four necessary conditions for deadlock (mutual exclusion, hold and wait, no preemption, circular wait) cannot occur. For example, requiring processes to request all resources at once prevents hold and wait.

Q9. What is the main function of Memory Management Unit (MMU)?
The MMU translates logical addresses generated by the CPU into physical addresses in memory. It enables virtual memory, manages address spaces, and provides memory protection by restricting access to certain memory regions.

Q10. What is the importance of disk scheduling in an operating system?
Disk scheduling optimizes the order in which disk I/O requests are serviced, reducing seek time and improving system performance. Algorithms like FCFS, SSTF, SCAN, and C-SCAN balance efficiency and fairness in handling I/O requests.

PART - B

Q1. Differentiate among multiprogramming, multiprocessing, and multitasking.

Differences Among Multiprogramming, Multiprocessing, and Multitasking:

Aspect	Multiprogramming	Multiprocessing	Multitasking
Definition	Running multiple programs on a single processor by switching between them to improve CPU utilization.	Using two or more processors (CPUs) simultaneously to execute multiple processes.	Running multiple tasks (processes or threads) on a single processor by switching between them.
Execution Units	Single CPU executes multiple programs one at a time by switching.	Multiple CPUs execute multiple processes simultaneously.	Single CPU executes multiple tasks by rapid context switching.
Parallelism	No true parallel execution; processes are executed one by one.	True parallelism as multiple processors work concurrently.	No true parallelism on a single CPU; gives the illusion of parallelism via time-sharing.
Purpose	To maximize CPU utilization by reducing idle time.	To increase computing power and handle multiple processes faster.	To improve user experience by allowing multiple applications or tasks to run seemingly at the same time.
Resource	Programs share CPU time and other system	Processes can share resources but	Tasks share CPU time, with rapid

Sharing	resources sequentially.	also use independent CPUs for execution.	switching giving an illusion of concurrency.
Examples	Running a text editor while downloading a file on older systems.	A server with multiple CPUs handling numerous client requests simultaneously.	Typing in a word processor while playing music in the background on modern operating systems.
System Complexity	Simpler to implement compared to multiprocessing but less efficient.	Requires complex hardware and operating system support for managing multiple CPUs.	Requires an OS capable of scheduling and context switching efficiently.

Q2. State the differences between logical and physical address space.

Differences Between Logical and Physical Address Space:

Aspect	Logical Address Space	Physical Address Space
Definition	The address generated by the CPU during program execution.	The actual address in the memory unit where data is stored.
Generation	Generated by the CPU.	Mapped by the memory management unit (MMU) or similar hardware.
Visibility	It is virtual and not visible to the user.	It represents the actual memory and is visible to the system.
Mapping	Mapped to physical addresses via address translation.	Does not require further mapping; it is direct and real.
Dependency	Exists in the user's perspective; depends on the process.	Exists at the hardware level; depends on system configuration.
Size	Can be larger than physical memory if virtual memory is used.	Limited by the size of the physical memory (RAM).
Access	Requires translation before accessing physical memory.	Accessed directly without further translation.

Example:

- **Logical Address**: A program might request data at address 1200.
- **Physical Address**: The memory management unit (MMU) maps 1200 to an actual physical memory location, say, 30200.

Key Point: Logical addresses are part of the abstraction provided by the operating system, while physical addresses are the actual memory locations in hardware.

Q.3. What do you mean by a Deadlock? Explaif Banker's algorithm with an example.

A **deadlock** occurs in a system when two or more processes are unable to proceed because each process is waiting for a resource held by another process. It creates a state where no process can complete its execution, and the system gets stuck.

Conditions for Deadlock (Coffman’s conditions):

1. **Mutual Exclusion**: At least one resource is held in a non-sharable mode.
2. **Hold and Wait**: A process is holding at least one resource and waiting for additional resources.
3. **No Preemption**: Resources cannot be forcibly removed from a process.
4. **Circular Wait**: A set of processes is waiting in a circular chain for resources.

Banker’s Algorithm

The **Banker’s Algorithm** is a resource allocation and deadlock avoidance algorithm. It is used to ensure that the system remains in a **safe state** by simulating the allocation of resources and checking if it leads to a deadlock.

Key Terms:

1. **Available**: The number of available instances of each resource type.
2. **Max**: The maximum demand of each process for each resource.
3. **Allocation**: The number of resources of each type currently allocated to each process.
4. **Need**: Remaining resource needs of each process (calculated as Need = Max - Allocation).

Steps of the Algorithm:

1. **Initialization**: Initialize the available resources, maximum needs, allocation, and need matrices.
2. **Safety Check**
   :
   - Find a process whose Need can be satisfied by the Available resources.
   - Assume the process completes, release its allocated resources back to Available.
   - Repeat until all processes are completed or no such process can be found.

3. **Safe State**
   :
   - If all processes can complete, the system is in a safe state.
   - Otherwise, it is in an unsafe state (deadlock is possible).

Example of Banker's Algorithm

Given:

- **Processes**: P1, P2, P3
- **Resource Types**: R1, R2, R3 (instances = 10, 5, 7)

Initial State:

Process	Max (R1, R2, R3)	Allocation (R1, R2, R3)	Need (R1, R2, R3)
P1	(7, 5, 3)	(0, 1, 0)	(7, 4, 3)
P2	(3, 2, 2)	(2, 0, 0)	(1, 2, 2)
P3	(9, 0, 2)	(3, 0, 2)	(6, 0, 0)

- **Available Resources**: (3, 3, 2)

Execution:

1.

Find a Process that Can Execute:

- Check if

Need ≤ Available
for each process.

- P1: Need = (7, 4, 3) > Available = (3, 3, 2) ❌ **Cannot Execute**
- P2: Need = (1, 2, 2) ≤ Available = (3, 3, 2) ✅ **Can Execute**

2. **Simulate Execution for P2**:

   - Allocate resources to P2, let it finish, and release its allocation back to Available
     .
     - New Available = (3 + 2, 3 + 0, 2 + 0) = (5, 3, 2).

3. **Repeat for Remaining Processes**:

   - P1: Need = (7, 4, 3) > Available = (5, 3, 2) ❌ **Cannot Execute**
   - P3: Need = (6, 0, 0) ≤ Available = (5, 3, 2) ✅
     Can Execute

     - New Available = (5 + 3, 3 + 0, 2 + 2) = (8, 3, 4).

   - P1: Need = (7, 4, 3) ≤ Available = (8, 3, 4) ✅
     Can Execute

     - New Available = (8 + 0, 3 + 1, 4 + 0) = (8, 4, 4).

4.

Final State:

- All processes have executed successfully.
- The system is in a **safe state**.

Q.4. What is Directory? What are UFD and MFD? Also state the operations that can be 4 performed on a directory.

A **directory** is a data structure used in file systems to organize and manage files. It stores information such as file names, attributes, and the location of files on storage devices. Directories allow users to create a hierarchical file organization for efficient file management.

Types of Directory Structures:

1. **Single-level Directory**: All files are stored in a single directory.
2. **Two-level Directory**: Each user has their own separate directory.
3. **Tree-structured Directory**: Hierarchical organization of directories and files.

UFD (User File Directory):

- **Definition**: A **User File Directory** is specific to each user in a two-level or tree-structured directory system.
- **Purpose**: It stores and manages the files created by a single user.
- **Example**: A user directory like /home/user1 contains all files belonging to user1.

MFD (Master File Directory):

- **Definition**: The **Master File Directory** is a global directory at the system level that contains references to all **UFDs**.
- **Purpose**: It acts as the root or top-level directory and manages user directories.
- **Example**: In a UNIX-like file system, / is the MFD, and directories like /home, /etc are part of it.

Operations on a Directory:

1. **Search for a File**: Locate a file in the directory by its name.
2. **Create a File**: Add a new file entry to the directory.
3. **Delete a File**: Remove a file entry and its associated metadata.
4. **Rename a File**: Change the name of a file entry in the directory.
5. **List Files**: Display the files and subdirectories present in a directory.
6. **Traverse the Directory**: Navigate through directories to access files or subdirectories.
7. **Modify Attributes**: Change file attributes like permissions, timestamps, etc.

Q5. What is a race condition? Illustrate with an example why presence of race condition is considered as bad design.

A **race condition** occurs in a system when two or more processes or threads access shared data simultaneously, and the final outcome depends on the timing of their execution. In such situations, the system's behavior becomes **unpredictable** and can lead to incorrect or inconsistent results.

Race conditions typically happen in **multi-threaded or concurrent systems** where processes do not have proper synchronization mechanisms to coordinate their actions.

Why is the Presence of Race Condition Considered Bad Design?

1. **Unpredictable Behavior**: Race conditions make the system's behavior inconsistent and difficult to debug.
2. **Data Corruption**: Shared resources may get corrupted due to simultaneous access without proper synchronization.
3. **Security Vulnerabilities**: An attacker might exploit race conditions to manipulate data or execute unauthorized actions.
4. **System Failures**: Critical applications like banking, healthcare, or embedded systems may fail, leading to severe consequences.

Illustration of a Race Condition

Scenario: Bank Account Withdrawal

Imagine a banking application where two users (or threads) are trying to withdraw money from a shared account at the same time.

- **Initial Account Balance**: ₹10,000
- **Thread 1**: Withdraws ₹5,000
- **Thread 2**: Withdraws ₹7,000

Steps without Synchronization:

1. **Thread 1 Reads Balance**: ₹10,000
2. **Thread 2 Reads Balance**: ₹10,000
3. **Thread 1 Deducts ₹5,000**: New balance = ₹5,000 (locally updated)
4. **Thread 2 Deducts ₹7,000**: New balance = ₹3,000 (locally updated)
5. **Thread 1 Writes Back**: Balance = ₹5,000
6. **Thread 2 Writes Back**: Balance = ₹3,000

Final Balance in Account: ₹3,000 (Incorrect)

- The system ignored one of the withdrawals completely.

Correct Design (Using Synchronization):

To avoid race conditions, a synchronization mechanism like a **lock** or **mutex** can be used.

1. **Thread 1 Acquires Lock**: Reads ₹10,000
2. **Thread 1 Deducts ₹5,000**: Writes ₹5,000 back
3. **Thread 1 Releases Lock**
4. **Thread 2 Acquires Lock**: Reads ₹5,000
5. **Thread 2 Deducts ₹7,000**: Operation denied or balance updated correctly
6. **Thread 2 Releases Lock**

Final Balance: Properly handled, preventing incorrect results.

Q.6 Consider there are three page frames which are initially empty. If the page reference string is 1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6. Calculate the number of page faults using the optimal page replacement policy.

To calculate the number of page faults using the **Optimal Page Replacement Policy** with 3 empty frames and the page reference string **1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6**, follow these steps:

Step-by-Step Simulation:

Page Reference String: 1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6

Number of Frames: 3 (Initially empty)

Step	Page Reference	Current Frames	Replaced Page	Page Fault?	Explanation
1	**1**	1 _ _	-	**Yes**	Frame is empty; load 1.
2	**2**	1 2 _	-	**Yes**	Frame is empty; load 2.
3	**3**	1 2 3	-	**Yes**	Frame is empty; load 3.
4	**4**	4 2 3	**1**	**Yes**	Replace 1 (used much later than 2 and 3).
5	**2**	4 2 3	-	**No**	2 is already in the frame.
6	**1**	1 2 3	**4**	**Yes**	Replace 4 (used much later).
7	**5**	5 2 3	**1**	**Yes**	Replace 1 (used much later).
8	**3**	5 2 3	-	**No**	3 is already in the frame.
9	**2**	5 2 3	-	**No**	2 is already in the frame.
10	**4**	4 2 3	**5**	**Yes**	Replace 5 (not used again).
11	**6**	6 2 3	**4**	**Yes**	Replace 4 (not used again).

Total Page Faults:

1. Count all the steps where a **page fault** occurs (marked as **Yes** in the table).
2. **Steps with Page Faults**: 1, 2, 3, 4, 6, 7, 10, 11.

- **Total Page Faults = 8**

Q.7. Explain the performance of demand paging with necessary examples.

Answer : Performance of Demand Paging

Demand Paging is a memory management technique in which a page (a fixed-size block of memory) is loaded into memory only when it is required during program execution. This helps in efficient utilization of memory as only the necessary pages are loaded.

The performance of demand paging depends on several factors:

Key Metrics to Measure Performance

1.

Page Fault Rate (P):
This is the fraction of memory accesses that result in a page fault (i.e., when a required page is not in memory and must be loaded from secondary storage).

- **0 ≤ P ≤ 1**
- P=0P = 0P=0: No page faults (ideal case).
- P=1P = 1P=1: Every memory access results in a page fault (worst case).

2.

Effective Access Time (EAT):
The average time taken to access a memory location considering both page faults and memory accesses.

$$EAT = (1 - P) \times \text{Memory Access Time} + P \times \text{Page Fault Service Time}$$

Performance Equation

Let:

- Memory Access Time = m (usually in nanoseconds).
- Page Fault Service Time = s (time to handle a page fault, including disk I/O and page replacement, typically in milliseconds).

Then:

$$EAT = (1 - P) \times m + P \times s$$

Example for Calculation

Scenario:

- **Memory Access Time (m):** 200 nanoseconds.
- **Page Fault Service Time (s):** 8 milliseconds = 8,000,000 nanoseconds.
- **Page Fault Rate (P):** 0.01 (1%).

Calculation:

$EAT = (1 - 0.01) \times 200 + 0.01 \times 8{,}000{,}000$

$EAT = 0.99 \times 200 + 0.01 \times 8{,}000{,}000$

$EAT = 198 + 80{,}000 = 80{,}198$ nanoseconds.

Interpretation:

- Without page faults: $EAT = m = 200$ nanoseconds.
- With 1% page faults: $EAT = 80{,}198$ nanoseconds, which is significantly slower due to the high cost of servicing page faults.

Impact of Page Fault Rate

1. **Low Page Fault Rate ($P \approx 0$):**
   - The system performs close to the normal memory access time.
   - Example: $P = 0.001$, $EAT \approx 200 + 0.001 \times 8{,}000{,}000 = 8{,}200$ ns.

2.

High Page Fault Rate (P ≈ 1):

- The system becomes extremely slow as most memory accesses require disk I/O.
- Example: P=0.5P = 0.5P=0.5, EAT=0.5× 200+0.5× 8,000,000=4,000,100 ns.

Factors Affecting Demand Paging Performance

1. **Page Replacement Algorithms:**
   - Algorithms like FIFO, LRU, and Optimal affect the frequency of page faults.
2. **Working Set Size:**
   - If the working set (frequently accessed pages) fits in memory, page faults are minimized.
3. **Disk Speed:**
   - Faster secondary storage (like SSDs) reduces page fault service time sss.
4.

Degree of Multiprogramming:

- High degrees of multiprogramming increase competition for memory, leading to more page faults.

Example in Real-Life Terms

Imagine a bakery:

- **Freshly baked goods (RAM):** If the goods are on display (loaded in memory), they are instantly available to customers.
- **Bakery storage (Disk):** If the goods are not on display (page fault), staff need to retrieve them from storage (disk I/O), causing delays.

If 1 out of every 10 customers requests something from storage, the service time increases significantly, just like a higher page fault rate slows down the system.

PART-C

Q1. (a) Describe the actions taken by Kernel to context switch between processes.

Context Switching

Context switching is the process in which the CPU switches from one process to another. This is handled by the operating system's kernel to ensure multitasking and efficient use of the CPU.

Actions Taken by the Kernel During Context Switching

1. **Save the State of the Current Process**:

   - The kernel saves the state of the currently running process in its **PCB (Process Control Block)**.
   - This state includes:
     - Program Counter (PC)
     - CPU registers
     - Process priority
     - Memory management information

2. **Change the Process State**:

   - The current process is moved from the **Running** state to the **Ready** or **Waiting** state, depending on whether it needs to wait for a resource.

3. **Select the Next Process to Run**:

- The **scheduler** selects the next process to be executed based on the scheduling algorithm (e.g., FCFS, SJF, Round Robin).

4. **Load the State of the Next Process**:

   - The kernel restores the state of the selected process from its PCB. This involves:
     - Loading the program counter (to continue execution from where it left off)
     - Restoring CPU registers and memory mapping.

5. **Update Process Tables and CPU Registers**:

   - The kernel updates internal tables and CPU registers to reflect the new process that is running.

6. **Switch to User Mode**:

   - The CPU switches to **user mode** to begin executing the next process.

---

b) **For the given set of processes, calculate the Average Waiting Time (AWT) and Average Turnaround Time (ATAT) using the following scheduling algorithms:**

Given processes:

Process	Burst Time	Priority
P1	8	4
P2	6	1
P3	1	2
P4	9	2
P5	3	3

We will calculate **average waiting time** and **average turnaround time** for the following scheduling algorithms:

First Come First Serve (FCFS)

FCFS Scheduling executes processes in the order they arrive.

Process	Burst Time	Completion Time	Turnaround Time (TAT)	Waiting Time (WT)
P1	8	8	8	0
P2	6	14	14	8
P3	1	15	15	14
P4	9	24	24	15
P5	3	27	27	24

Average Waiting Time (AWT) = (0 + 8 + 14 + 15 + 24) / 5 = **12.2**

Average Turnaround Time (ATAT) = (8 + 14 + 15 + 24 + 27) / 5 = **17.6**

Shortest Job First (SJF)

SJF Scheduling selects the process with the smallest burst time next.

Process	Burst Time	Completion Time	Turnaround Time (TAT)	Waiting Time (WT)
P3	1	1	1	0
P5	3	4	4	1
P2	6	10	10	4
P1	8	18	18	10
P4	9	27	27	18

Average Waiting Time (AWT) = (0 + 1 + 4 + 10 + 18) / 5 = **6.6**

Average Turnaround Time (ATAT) = (1 + 4 + 10 + 18 + 27) / 5 = **12.0**

Round Robin (RR)

Round Robin Scheduling executes each process in a time slice (quantum). Assume **Time Quantum = 3**.

Time	Executed Process	Remaining Burst
0-3	P1	5
3-6	P2	3
6-7	P3	0 (Done)
7-10	P4	6
10-13	P5	0 (Done)
13-16	P1	2
16-18	P2	0 (Done)
18-21	P4	3
21-24	P4	0 (Done)

Completion Times:

- P1 = 18, P2 = 16, P3 = 7, P4 = 24, P5 = 13

Turnaround Times:

- P1 = 18, P2 = 16, P3 = 7, P4 = 24, P5 = 13

Waiting Times:

- P1 = 10, P2 = 10, P3 = 6, P4 = 15, P5 = 10

Average Waiting Time (AWT) = (10 + 10 + 6 + 15 + 10) / 5 = **10.2**

Average Turnaround Time (ATAT) = (18 + 16 + 7 + 24 + 13) / 5 = **15.6**

Final Results

Scheduling Algorithm	Average Waiting Time	Average Turnaround Time
FCFS	12.2	17.6
SJF	6.6	12.0
RR (Time Quantum = 3)	10.2	15.6

Q.2 What is Fork system call? What will be the output of the following code and justify the output?

```
#include<stdio.h>
#include<unistd.h>
int main()
{ if (fork()::
fork ())
fork ();
```

Printf("1"):
return 0;
}

Answer:

The **fork()** system call is used in Unix/Linux systems to create a new process. It is one of the most commonly used system calls for multitasking. Here's how it works:

1. **Parent and Child Process**:

   - The fork() call creates a **child process**, which is an exact copy of the parent process (with a unique process ID).
   - After fork(), the execution continues in **both parent and child processes** from the point where the fork was called.

2. **Return Values of fork()**:

   - **Parent Process**: fork() returns the PID of the child process (> 0).
   - **Child Process**: fork() returns 0.
   - If the fork() fails, it returns -1.

Provided Code

c

```
#include<stdio.h>
#include<unistd.h>
int main()
{ if (fork()::
fork ())
fork ();
Printf("1"):
return 0;
}
```

Output Justification

1. **Number of Forks**
   :

- **Initial State**: At the start, there is **1 process (parent)**.
- **First fork()**: Splits the process into **2 processes** (parent and child).
- **Second fork() (inside if)**
  :
  - Only the process where fork() in if(fork()) evaluates true executes this line, creating **2 more processes**.

- **Third fork()**: All existing processes execute this line, creating **4 more processes**.

Breakdown of Processes

Step	Number of Processes	Explanation
Start	1	Initial parent process
First fork()	2	Parent creates 1 child process
Second fork()	4	The if condition runs in 1 parent and 1 child, each creates 1 process
Third fork()	8	Each of the 4 existing processes creates 1 new process

---

Execution of printf("1")

- Each of the 8 processes will execute the printf("1");.
- **Output**: "1" will be printed **8 times**, but the order may vary depending on the scheduling of processes.

Final Output

11111111

Note: The output may appear in different orders (e.g., 11111111, 11111111) because the processes execute concurrently, and the exact order depends on the operating system's process scheduling.

Q.3 Suppose that a disk has 500 cylinders. (0-4999). The drive is currently serving a request at 143, and the previous request was at cylinder 125. The queue of pending requests are 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130 starting from the current head position, what is the total distance that the disk arm move to satisfy all the pending requests for each of the following:

(i) FCFS

(ii) SSTF

(iii) SCAN

Given Data:

- **Number of cylinders**: 0 to 4999.
- **Current head position**: 143.
- **Previous request**: 125.
- **Pending requests**: 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130.

(i) FCFS (First Come, First Serve):

The head processes the requests in the exact order in which they appear in the queue.

Steps:

1. **Initial position**: 143.
2. **Queue order**: 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130.
3. Calculate the absolute difference between consecutive positions.

Calculation:

- Distance from 143 → 86 = |143 - 86| = 57
- Distance from 86 → 1470 = |86 - 1470| = 1384
- Distance from 1470 → 913 = |1470 - 913| = 557
- Distance from 913 → 1774 = |913 - 1774| = 861
- Distance from 1774 → 948 = |1774 - 948| = 826
- Distance from 948 → 1509 = |948 - 1509| = 561
- Distance from 1509 → 1022 = |1509 - 1022| = 487
- Distance from 1022 → 1750 = |1022 - 1750| = 728
- Distance from 1750 → 130 = |1750 - 130| = 1620

Total Distance = 57 + 1384 + 557 + 861 + 826 + 561 + 487 + 728 + 1620 = **7081**

(ii) SSTF (Shortest Seek Time First):

In this algorithm, the head services the request nearest to its current position first, minimizing seek time.

Steps:

1. **Initial position**: 143.
2. Continuously select the closest request to the current position, and calculate the distance.

Calculation:

- Start at 143. The closest request is **130**: |143 - 130| = 13

- Move to 130. The closest request is **86**: |130 - 86| = 44
- Move to 86. The closest request is **913**: |86 - 913| = 827
- Move to 913. The closest request is **948**: |913 - 948| = 35
- Move to 948. The closest request is **1022**: |948 - 1022| = 74
- Move to 1022. The closest request is **1470**: |1022 - 1470| = 448
- Move to 1470. The closest request is **1509**: |1470 - 1509| = 39
- Move to 1509. The closest request is **1750**: |1509 - 1750| = 241
- Move to 1750. The closest request is **1774**: |1750 - 1774| = 24

Total Distance = 13 + 44 + 827 + 35 + 74 + 448 + 39 + 241 + 24 = **1745**

(iii) SCAN (Elevator Algorithm):

In this algorithm, the head moves in one direction (in this case, assume upward), servicing all requests in its path. After reaching the end, it reverses direction if needed.

Steps:

1. **Initial position**: 143.
2. Head moves upward (higher cylinder numbers) servicing requests until it reaches the maximum request (or end of the cylinder). It then reverses direction and services the remaining requests.

Sorted Queue:

- Requests: 86, 130, 143 (current position), 913, 948, 1022, 1470, 1509, 1750, 1774.

Calculation:

- Move upward from 143:
  - 143 → 913: |143 - 913| = 770
  - 913 → 948: |913 - 948| = 35
  - 948 → 1022: |948 - 1022| = 74
  - 1022 → 1470: |1022 - 1470| = 448
  - 1470 → 1509: |1470 - 1509| = 39
  - 1509 → 1750: |1509 - 1750| = 241
  - 1750 → 1774: |1750 - 1774| = 24

- Reverse direction and service lower requests:
  - 1774 → 130: |1774 - 130| = 1644
  - 130 → 86: |130 - 86| = 44

Total Distance = 770 + 35 + 74 + 448 + 39 + 241 + 24 + 1644 + 44 = **3319**

Final Results

- **FCFS**: 7081
- **SSTF**: 1745
- **SCAN**: 3319

These results show how different algorithms optimize disk head movements in

varying ways. SSTF minimizes seek time the most, but it can lead to starvation for some requests, while SCAN ensures fairness by servicing all requests in one direction before reversing.

Q.4. (a) What is Virtual Memory? How it is different from Cache memory and secondary memory? Also discuss the benefits of virtual memory techniques.

(b) Discuss the indexed file allocation method with proper example.

Virtual memory is a memory management technique that provides an “illusion” of a large, continuous main memory (RAM) to applications, even if the actual physical memory is limited. It uses a portion of the disk (swap space) as an extension of RAM to temporarily store parts of programs and data that are not currently in active use.

Differences Between Virtual Memory, Cache Memory, and Secondary Memory

Feature	Virtual Memory	Cache Memory	Secondary Memory
Purpose	Extends main memory to run larger programs or handle more data than physical RAM can hold.	Temporarily stores frequently accessed data for faster processing.	Permanently stores large volumes of data (e.g., HDD, SSD).
Location	Resides on the hard disk or SSD (swap space).	Resides between the CPU and RAM (inside the CPU or as part of system architecture).	External storage devices (HDD, SSD, USB drives).

Speed	Slower than RAM because it relies on disk access.	Faster than RAM as it uses static memory or high-speed circuits.	Much slower than both RAM and cache memory.
Data Type	Handles temporary program data and instructions that don't fit in physical RAM.	Stores frequently used data and instructions to reduce CPU access time.	Stores all types of data, including applications, operating systems, and user files.
Volatility	Temporary; cleared when the system reboots.	Temporary; cleared when the system shuts down.	Permanent until deleted by the user.

Benefits of Virtual Memory Techniques

1. **Program Size Flexibility**: Allows execution of large programs that may not fit entirely in physical memory.
2. **Efficient Memory Utilization**: Optimizes RAM usage by keeping only active pages in physical memory and moving inactive pages to disk.
3. **Multitasking Support**: Enables multiple programs to run simultaneously by dynamically allocating memory.
4. **Protection and Isolation**: Prevents one process from interfering with another by maintaining separate virtual address spaces.
5. **Cost Effectiveness**: Reduces the need for large amounts of expensive physical RAM.
6. **Demand Paging**: Loads only necessary parts of programs into RAM, reducing initial load time.

(b) Indexed File Allocation Method

In the **Indexed File Allocation Method**, a special index block is created for each file, which contains pointers to all the disk blocks allocated to the file. This solves the problems of fragmentation and sequential access delays associated with other methods like contiguous or linked allocation.

Features of Indexed File Allocation

1. **Fast Direct Access**: Any block of a file can be accessed directly using the index table.
2. **Flexible Storage**: File blocks need not be stored sequentially, reducing fragmentation issues.
3. **Complexity**: Requires additional storage for the index block.

Example

Suppose a file named file1 has five data blocks stored on disk: 5, 9, 14, 21, and 25.

- **Index Block**: A dedicated block is used to store the list of pointers for this file.
  Index Block for file1:

+-------+-------+-------+-------+-------+

```
|   5   |   9   |  14   |  21   | 25  |
+-------+-------+-------+-------+-------+
```

File Blocks on Disk:

Block 5 → Data 1
Block 9 → Data 2
Block 14 → Data 3
Block 21 → Data 4
Block 25 → Data 5

Advantages of Indexed Allocation

1. **Direct Access**: Supports random access to file blocks without traversing a linked list.
2. **Reduced Fragmentation**: File blocks can be stored non-contiguously, avoiding external fragmentation.
3. **Efficient Access**: Index block provides quick access to any part of the file.

Disadvantages

1. **Index Overhead**: Additional disk space is needed to store the index blocks.
2. **File Size Limitation**: The size of the index block limits the maximum number of file blocks.

Q. 5. Write short notes on the following:
(a)Mobile OS
(b)Time OS
(c)Belady Anomaly
(d) Principle of locality of reference

(a) Mobile OS

A **Mobile Operating System** is a software platform designed specifically for mobile devices like smartphones, tablets, and wearables. It manages hardware and software resources, enabling users to run applications efficiently.

Examples: Android, iOS, Windows Phone, HarmonyOS.

Features:

1. **Touch Interface**: Optimized for touch screens with gestures.

2. **Power Management**: Efficiently manages battery life.
3. **Application Ecosystem**: Provides app stores like Google Play or Apple App Store for easy access to applications.
4. **Connectivity**: Supports cellular networks, Wi-Fi, Bluetooth, and GPS.
5. **Security**: Includes features like encryption, app sandboxing, and regular updates to prevent malware.

(b) Time OS

A **Time OS**, often referred to as a **Real-Time Operating System (RTOS)**, is designed to process data and execute tasks within strict time constraints. It is used in systems where response time is critical.

Examples: FreeRTOS, VxWorks, QNX.

Characteristics:

1. **Deterministic Behavior**: Guarantees that tasks will complete within defined time limits.
2. **Priority Scheduling**: Higher-priority tasks are executed first.
3. **Applications**: Found in embedded systems, robotics, automotive control systems, and medical devices.

Types of RTOS:

- **Hard RTOS**: Missing a deadline can lead to catastrophic failure (e.g., pacemakers).
- **Soft RTOS**: Deadlines are important but not critical (e.g., video streaming).

(c) Belady's Anomaly

Belady's Anomaly refers to the counterintuitive situation in page replacement algorithms where increasing the number of page frames **results in more page faults**.

- This phenomenon occurs in the **FIFO (First In, First Out)** page replacement algorithm.
- Normally, adding more page frames should reduce page faults, but in some cases, it worsens the performance due to the way FIFO discards pages.

Example: Consider a reference string 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Frames	3 Frames (Page Faults = 9)	4 Frames (Page Faults = 10)
1	1	1
2	2	2
3	3	3
4	Replace 1 (fault)	Add 4
1	Replace 2 (fault)	No fault

This anomaly highlights the limitations of FIFO in certain cases.

(d) Principle of Locality of Reference

The **Principle of Locality of Reference** states that programs tend to access memory locations that are close to each other (spatial locality) or accessed repeatedly within a short period of time (temporal locality).

Types of Locality:

1. **Temporal Locality**: If a memory location is accessed, it is likely to be accessed again soon. Example: Loop variables.
2. **Spatial Locality**: If a memory location is accessed, its neighbors are likely to be accessed soon. Example: Sequential array traversal.

Applications:

- **Cache Design**: Makes use of spatial and temporal locality for faster memory access.
- **Paging and Virtual Memory**: Optimizes page replacement policies.

By understanding this principle, systems can predict future memory access patterns and improve performance.