End-Sem Examination- Summer 2025

Exam Seat No.

Model Answer

Academic Year: 2024-2025 **Semester: II**

Name of Programme: MCA **Pattern: 2024**

Name of Course: Elective II: Operating System **Course Code: 2409515B**

Max. Marks: 60 **Duration: 2:30Hr.**

Instructions: Candidates should read carefully the instructions printed on the Question Paper and on the cover page of the Answer Book, which is provided for their use.

1. This question paper contains _____page(s).
2. Answer to each new question is to be started on a new page.
3. Assume suitable data wherever required, but justify it.
4. Draw the neat labelled diagrams, wherever necessary.
5. The last columns indicates the Course Outcome and level of Blooms Taxonomy of the Question/sub-question

<table>
<tr><th>Q. No.</th><th>Details</th><th>Max. Marks</th></tr>
<tr><td>Q.1</td><td>Differentiate between Assembler and Compiler
<table>
<tr><td>Compiler</td><td>Assembler</td></tr>
<tr><td>Compiler converts the source code written by the programmer to a machine level language.</td><td>Assembler converts the assembly code into the machine code.</td></tr>
<tr><td>Compiler input source code.</td><td>Assembler input assembly language code.</td></tr>
<tr><td>It converts the whole code into machine language at a time.</td><td>But the Assembler can’t do this at once.</td></tr>
<tr><td>It takes less execution time compared to an assembler.</td><td>It takes more time than a compiler.</td></tr>
<tr><td>It shows the whole program error after</td><td>It detects errors in the first phase, fixes</td></tr>
</table></td><td>[6]</td></tr>
</table>

the whole program is scanned.	them, and then the second phase is start to execute.
A Compiler is more intelligent than an Assembler.	But, an Assembler is less intelligent than a Compiler.
The compilation phases are lexical analyzer, syntax analyzer, semantic analyzer, intermediate code generated, a code optimizer, code generator, and error handler	Assembler makes two phases over the given input, first phase and the second phase.
The output of compiler is a mnemonic version of machine code.	The output of assembler is binary code.
C, C++, Java, and C# are examples of compiled languages.	GAS, GNU is an example of an assembler.

Q.2

Explain multilevel queue scheduling. How is it different from multilevel feedback queue scheduling? [6]

Multilevel Queue Scheduling

Multilevel Queue Scheduling is a **CPU scheduling algorithm** that divides the ready queue into **multiple separate queues**, where each queue represents a **different priority or type of process**.

Key Features:

Each queue has its **own scheduling algorithm** (e.g., foreground uses Round Robin, background uses FCFS).

No process can move from one queue to another (static queue membership).

Queues are assigned a **fixed priority order**.

The **CPU is first allocated to the highest-priority queue**; only if it's empty, lower-priority queues are considered.

Example:

Queue	Type of Process	Scheduling Algorithm
Q0	System processes	Round Robin
Q1	Interactive processes	Priority Scheduling
Q2	Batch processes	FCFS

Multilevel Feedback Queue Scheduling

Multilevel Feedback Queue Scheduling is a more flexible algorithm where processes **can**

move between queues based on their **behavior and execution history**.

Key *Features:*

Allows a process to **move to a lower-priority queue** if it uses too much CPU time (aging).

New or interactive processes may be **promoted to a higher-priority queue**.

Designed to **favor short or interactive jobs** by giving them more frequent CPU access.

More complex to implement, but **improves overall system responsiveness**.

Example Behavior:

A new job enters at the highest-priority queue.

If it finishes quickly, it stays.

If it uses too much CPU time, it's moved to a lower-priority queue.

Difference Table

Feature	Multilevel Queue Scheduling	Multilevel Feedback Queue Scheduling
Queue movement	No movement between queues (static)	Processes can move between queues (dynamic)
Flexibility	Less flexible	More flexible and adaptive
Suitability	Suitable for systems with well-defined process types	Suitable for general-purpose systems
Complexity	Simple to implement	More complex
Starvation Risk	High for lower queues	Lower (aging and promotion possible)

Multilevel queue scheduling is simple and rigid, suitable for static environments, while multilevel feedback queue scheduling is dynamic and responsive, ideal for general-purpose systems where process behavior varies over time.

Q.3

a) Consider the following snapshot of a system A,B,C and D are resources and Processes are P0,P1,P2,P3,P4

	Allocation				MAX				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				

[16]

P4	0	0	1	4		0	6	5	6				

1.What is the content of need array?

2. Is the system is in safe state? If yes, give safe sequence.

3. If a request from process P1 arrives for (0,4,2,0) then can it granted immediately. (8marks)

Answer:

Sol^{n} 1) Need [i, j] = Max [i, j] - Allocation [i, j]

	A	B	C	D
P_0	0	0	0	0
P_1	0	7	5	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	6	4	2

∴ Need array

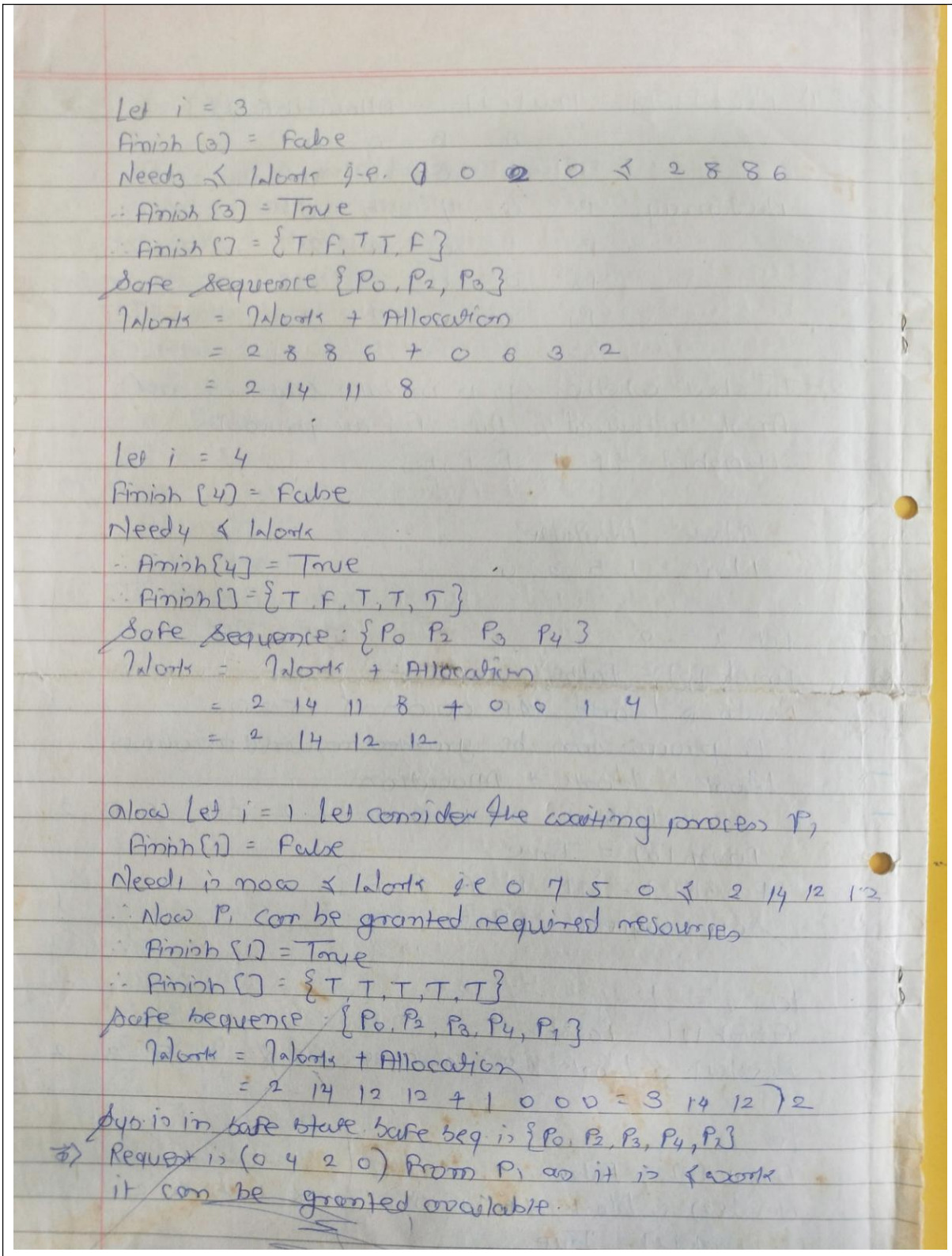
2) To check whether sys is in safe state or not
Finish initialized to false for all processes.
∴ Finish[] = {F, F, F, F, F} (P_0 P_1 P_2 P_3 P_4)

Work = Available
Work = 1 5 2 0

Let i = 0
Finish [0] = False
Need 0 ≤ Work i.e. 0 0 0 0 ≤ 1 5 2 0
∴ P_0 process can be granted required resources
∴ Work = Work + Allocation
1 5 2 0 + 0 0 1 2 = 1 5 3 2
∴ Finish [0] = True
∴ Finish [] = { T F F F F }
Safe sequence = { P_0 }

Let i = 1
Finish [1] = False
Need[1] ≤ Work i.e. 0 7 5 0 ≤ 1 5 3 2

Let i = 2
Finish [2] = False
Need[2] ≤ Work i.e. 1 0 0 2 ≤ 1 5 3 2
∴ Finish [2] = True
∴ Finish [] = { T, F, T, F, F }
Safe sequence { P_0, P_2 }
Work = Work + Allocation
= 1 5 3 2 + 1 3 5 4
= 2 8 8 6



OR

b) What is the important feature of critical section? State the Readers Writers problem and give solution using semaphore.

(8marks)

Answer: Key Feature of a Critical Section

In concurrent programming, a **critical section** is a segment of code that accesses shared resources, such as variables, files, or devices, which must not be concurrently accessed by

more than one thread or process. The paramount feature of a critical section is **mutual exclusion**.

Mutual exclusion ensures that only one process or thread can execute within the critical section at any given time. This is crucial to prevent race conditions, where the outcome depends on the sequence or timing of uncontrollable events.GeeksforGeeks

To manage access to critical sections, synchronization mechanisms like semaphores, mutexes, or monitors are employed. These tools help in coordinating the sequence of process execution to maintain data consistency and system stability.

The Readers-Writers Problem

The **Readers-Writers Problem** is a classic synchronization problem that addresses scenarios where a shared resource (like a database) is accessed by multiple reader and writer processes.

Problem Statement

- **Readers**: Processes that only read the shared resource. Multiple readers can access the resource simultaneously without issues.
- **Writers**: Processes that modify the shared resource. To prevent data inconsistency, writers require exclusive access.GeeksforGeeks

The challenge is to design a synchronization mechanism that allows:

- Multiple readers to read simultaneously.
- Writers to have exclusive access when writing.
- No reader or writer should experience starvation (i.e., be perpetually denied access).Wikipedia+1Wikipedia+1

Solution Using Semaphores (Readers Preference)

One common solution gives preference to readers, allowing them to access the resource as long as no writer is writing.

Shared Variables and Semaphores

- `readcount`: Integer variable to count the number of active readers.
- `mutex`: Semaphore initialized to 1, used to protect access to `readcount`.
- `wrt`: Semaphore initialized to 1, used to grant write access to writers.

Reader Process :

```
wait(mutex);

readcount++;

if (readcount == 1)

   wait(wrt); // First reader locks the resource

signal(mutex);
```

```
// Reading is performed

wait(mutex);

readcount--;

if (readcount == 0)

   signal(wrt); // Last reader releases the resource

signal(mutex);

writer process

wait(wrt); // Writer locks the resource

// Writing is performed

signal(wrt); // Writer releases the resource
```

(8marks)

c) Describe deadlock prevention strategies . (8marks)

Answer : Deadlock prevention strategies aim to design systems where deadlocks are impossible, which is achieved by ensuring that at least one of the four necessary conditions for deadlock does not occur. These conditions are mutual exclusion, hold and wait, no preemption, and circular wait.

1. Preventing Mutual Exclusion:

Making Resources Shareable:

If resources can be used by multiple processes simultaneously (e.g., read-only files), the need for mutual exclusion is reduced, and the possibility of deadlock is decreased.

Resource Ordering:

If resources are non-shareable, a total ordering can be imposed, requiring processes to request resources in a predefined sequence, preventing circular waits.

2. Preventing Hold and Wait:

Resource Request Upfront:

Processes are required to request all resources they need before starting, or they can only request resources when they are not holding any.

Release Resources Before Requesting:

Processes must release all resources they currently hold before requesting any new resources.

3. Preventing No Preemption:

Preempting Resources:

The operating system should be able to take resources away from waiting processes and reallocate them to other processes that need them.

Timeouts:

If a process is waiting for a resource for too long, it can be preempted, allowing other processes to proceed.

4. Preventing Circular Wait:

Resource Ordering:

As mentioned earlier, processes can be required to request resources in a specific order, preventing circular dependencies.

Resource Allocation Graph:

A Resource Allocation Graph (RAG) can be used to visualize resource allocation and requests, and detect cycles that could lead to deadlock.

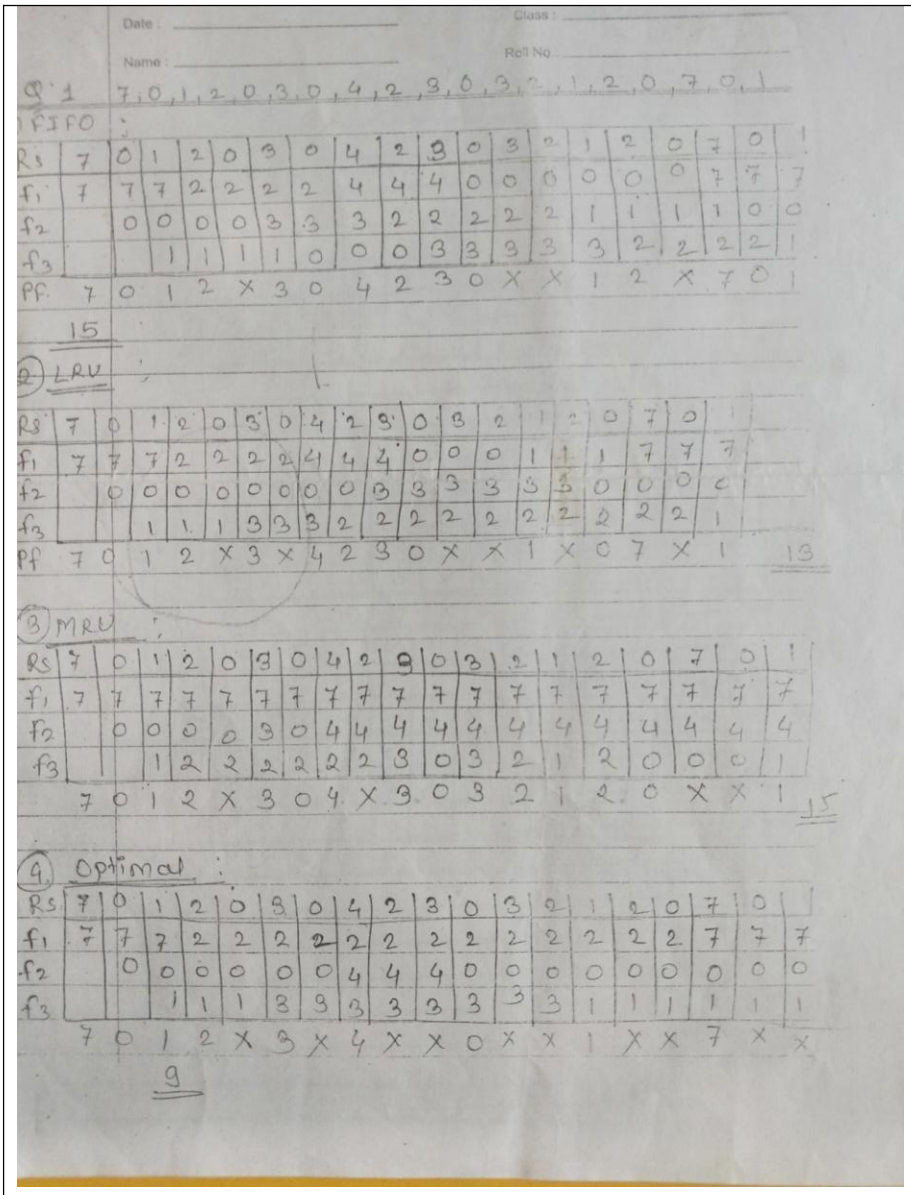
OR

d) Define deadlock. What are the four necessary conditions for deadlock to occur. (8marks)

Answer:

A deadlock is a situation where two or more processes are blocked indefinitely, each waiting for the other to release a resource that they need to continue. When a process runs it needs resources and it runs in its own address space. It is the responsibility of the operating system to make sure that the resources are allocated in the proper manner. The resources might be shareable or non-shareable. For example, a printer is a non-shareable resource. Only one process can print at a time. The operating system ensures that only one process is assigned to the printer. Let's understand the deadlock with an example, let's say there are two processes P1 and P2, and two resources R1 and R2. Say R1 is a printer, which is non-shareable. R2 is a file that is also non-shareable in write mode. Only one process can write at a time. So process P1 begins. It asks the operating system to get the hold of the printer and since nobody else was using the printer, the operating system assigns the printer to it. Next P2 is assigned to the processor and it starts running. P2 asks the operating system to get write access to the file R2 and since nobody else was using it, the operating system assigns R2 to P2. P1 resumes its execution and requests R2. But R2 is already assigned to P2 and cannot be assigned to more than one process at a time. So the operating system asks P1 to wait for R2 till P2 finishes writing into the R2's file. Now P2 resumes its execution and it requests the printer. The same happens with P2 since the printer is already assigned to the P1, and the operating system asks P2 to wait for R2. So none of the processes can progress further because both of them are holding one non-shareable resource and waiting for other non-shareable resources. This situation is called deadlock.

	The four necessary conditions for a deadlock to occur are: **1. Mutual Exclusion:** Resources can only be held by one process at a time. **2. Hold and Wait:** A process must hold at least one resource while waiting to acquire another. **3. No Preemption:** Resources cannot be forcibly taken away from a process; they must be released voluntarily. **4. Circular Wait:** There must be a circular chain of processes, where each process is waiting for a resource held by the next process in the chain. In simpler terms, a deadlock occurs when multiple processes are stuck, each waiting for a resource held by another process in a circular dependency. If any one of these conditions is not met, deadlock cannot occur. (8marks)	
Q.4	a) Consider the following page reference string : 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,7,0,1 Analyse number of page faults would occur for the FIFO , LRU and Optimal page replacement algorithm, assuming 3 frames? All frames are initially empty. (8marks) Answer :	[16]



OR

b) Categorize fixed and variable partitioning in contiguous memory management scheme (8marks)

Contiguous Memory Allocation:

In multiprg.ing, more than one prg. resides in the main memory of the system. There r many jobs with diff. mem requirements. Mem. mgr. has to select few of them & allocate mem to them. Mem. mgt has a problem in deciding which job to select.

Mem. is divided into no. of regions, each of which may contain one job. No. of regions in the mem. decide degree of multiprg.ing.

There r 2 approaches used for mem. mgt:

① Multiple contiguous fixed partition allocation:

Regions or partitions in this scheme r static, or fixed, can never be changed.

The common algorithm for this scheme is MFT (Multiprg.ing with fixed no. of tasks)

② Multiple contiguous variable partition allocation:

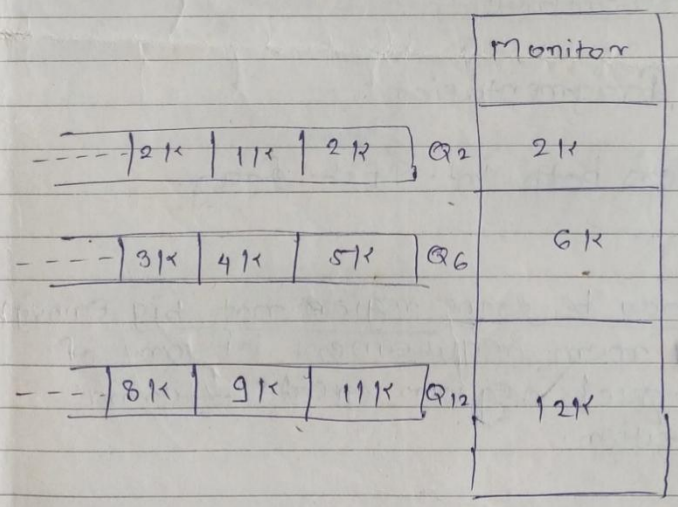
Partitions in this scheme r dynamic, can change eg. MVT (Multiprogramming with variable no. of tasks)

H/w for both MFT & MVT is the same, only the S/w is different.

MFT job scheduling:

In this scheme, the memory will be divided into some fixed no. of regions.
We can assign jobs to these regions by either of the foll. methods:

① There will be a separate ready queue for each of the regions. A region will be big enough to satisfy mem. requirement of all jobs in its own queue.
In this scheme a system will have to prepare over the common ready queue, in which all the jobs were initially placed. And then assign each of these jobs to the queue of appropriate region.
Each queue is scheduled separately.



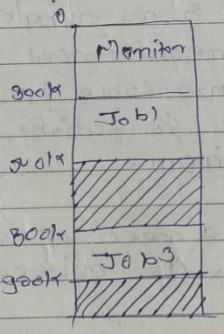
② In the other scheme, all the jobs r in only one ready queue. Whenever a partition is free by using one of the foll. algorithms the next job to be assigned to the partition is selected by FCFS, Round robin, Best Fit/ First Fit or priority algorithm.
If while one job is executing & a high priority job arrives, then the job from appropriate partition is swapped out & high priority job is swapped in.

Improper select of region size will lead to int. & ext. frag. & in turn to poor mem. utilization.

MVT Job scheduling

The problem of MFT of selecting partition size is not there in this scheme.

The region sizes r not fixed & vary dynamically O.S maintains a table indicating which parts of the mem. r available & which r occupied. Whenever a job request for the mem., the table indicating the current status of the mem is referred. If mem. is available, then the job is assigned the mem. & the table is updated to reflect the new status.



When the job terminates, it releases the mem occupied by it. Therefore at any instance of time the snapshot of mem. will show blocks of allocated memory & free holes of not allocated mem. distributed all over the mem.

(8marks)

c) Consider the following segment table :

Segment	Base	Length
0	363	500
1	1272	20
2	1675	1500
3	986	240
4	211	130

Simplify physical addresses for the following logical addresses

1. 0,425
2. 2,500
3. 1,150
4. 3,285

Logical Addresses and Their Physical Equivalents:

1. **Logical Address: ⟨0, 425⟩**
   - **Segment 0**: Base = 363, Length = 500
   - **Offset**: 425
   - **Validation**: 425 < 500 → ✓ Valid
   - **Physical Address**: 363 + 425 = **788**
2. **Logical Address: ⟨2, 500⟩**
   - **Segment 2**: Base = 1675, Length = 1500
   - **Offset**: 500
   - **Validation**: 500 < 1500 → ✓ Valid
   - **Physical Address**: 1675 + 500 = **2175**
3. **Logical Address: ⟨1, 150⟩**
   - **Segment 1**: Base = 1272, Length = 20
   - **Offset**: 150
   - **Validation**: 150 > 20 → ✗ Invalid
   - **Result**: **Segmentation Fault**
4. **Logical Address: ⟨3, 285⟩**
   - **Segment 3**: Base = 986, Length = 240
   - **Offset**: 285
   - **Validation**: 285 > 240 → ✗ Invalid
   - **Result**: **Segmentation Fault**

(8marks)
OR
d) Write a short note on swapping and overlap swapping.

Swapping is a memory management technique where processes are temporarily moved between main memory (RAM) and secondary storage (like a hard disk) to optimize memory usage.

How Swapping Works:

- **Swap Out**: When the main memory is full, inactive or lower-priority processes are moved to secondary storage to free up RAM.
- **Swap In**: When these processes are needed again, they are brought back into main memory from secondary storage.

This mechanism allows the system to handle more processes than the available physical memory would typically permit.

Benefits:

- **Enhanced Multiprogramming**: Facilitates the execution of multiple processes by efficiently utilizing memory.
- **Priority Handling**: Allows high-priority processes to be loaded into memory by swapping out lower-priority ones.
- ⚠Limitations:

- **Performance Overhead**: Frequent swapping can lead to increased I/O operations, potentially slowing down the system.
- **Disk Dependency**: Relies on the speed of secondary storage, which is typically

slower than RAM.

(8marks)

Q.5

a) Compare allocation methods used in a file system. (8marks)

File allocation methods determine how disk blocks are assigned to files. Each method has its own way of organizing and managing file storage on disk, affecting performance, access speed, and disk utilization. The three primary file allocation methods are:

1. Contiguous Allocation

Description:
Each file occupies a set of **contiguous** blocks on the disk.

Diagram:

```
less

CopyEdit

File A:  [10][11][12][13]
```

Pros:

- Fast access (both sequential and direct).
- Simple to implement.

Cons:

- Difficult to grow files (must be contiguous).
- Can lead to **external fragmentation**.
- Hard to find space for new files of varying sizes.

Use Case: Suitable for systems with static file sizes (e.g., read-only media like CDs).

2. Linked Allocation

Description:
Each file is a linked list of disk blocks, not necessarily contiguous. Each block contains a pointer to the next.

Pros:

- No external fragmentation.
- Easy to grow files.

Cons:

Slow direct access (only sequential access is efficient).

Pointer overhead in each block reduces effective storage.

Risk of lost blocks if a pointer is corrupted.

Use Case: Good for sequential file access (e.g., logs, multimedia files).

3. Indexed Allocation

Description:
Each file has an index block containing pointers to all the file blocks.

Pros:

Supports both direct and sequential access.

No external fragmentation.

Easier to grow files than contiguous.

Cons:

Overhead of index block(s), especially for large files.

Needs extra space for index tables.

Use Case: Suitable for systems needing fast access to large files (e.g., databases).

Comparison Table

Feature	Contiguous Allocation	Linked Allocation	Indexed Allocation
Access type	Fast (direct & seq.)	Slow (only sequential)	Fast (direct & seq.)
Fragmentation	External	None	None
File growth	Difficult	Easy	Easy
Space overhead	Low	High (pointers)	Moderate (index blocks)
Complexity	Low	Medium	High

Contiguous: Simple and fast but rigid.

Linked: Flexible but slow.

Indexed: Balanced with good access speed and flexibility, but more complex.

OR

b) Categorise various directory structures in detail.

Directory structures define how files and directories are organized in a file system. An efficient directory structure facilitates easy file access, sharing, and management. The **various types of directory structures** can be categorized as follows:

1. Single-Level Directory

Structure:

All files are in the same directory.

There is only one directory for all users.

Diagram:

```
Root
  ├── file1
  ├── file2
  └── file3
```

Pros:

Simple to implement.

Easy to locate files.

Cons:

Name collisions (two files cannot have the same name).

Poor scalability.

No support for grouping files by user or project.

Use Case: Small, single-user systems.

2. Two-Level Directory

Structure:

A separate directory for each user under a common root.

Each user has their own user directory.

Diagram:

```
Root
  ├── User1
```

```
|      ├── fileA
|      └── fileB
└── User2      ├── fileX
       └── fileY
```

Pros:

Avoids filename collisions between users.

Simple user-wise file management.

Cons:

No way to group files within a user's directory.

Inflexible hierarchy (only two levels).

Use Case: Multi-user systems with basic needs.

3. Tree-Structured Directory

Structure:

Hierarchical structure with directories and subdirectories.

Each directory can contain files or other directories.

Diagram:

```
Root
├── bin
├── home
|      ├── alice
|      |      ├── file1
|      |      └── docs
|      |             └── file2
|      └── bob
└── etc
```

Pros:

Supports grouping files logically.

Allows nested organization.

Supports both absolute and relative path names.

Cons:

Slightly more complex implementation.

Path management overhead.

Use Case: General-purpose systems (e.g., Linux, Windows).

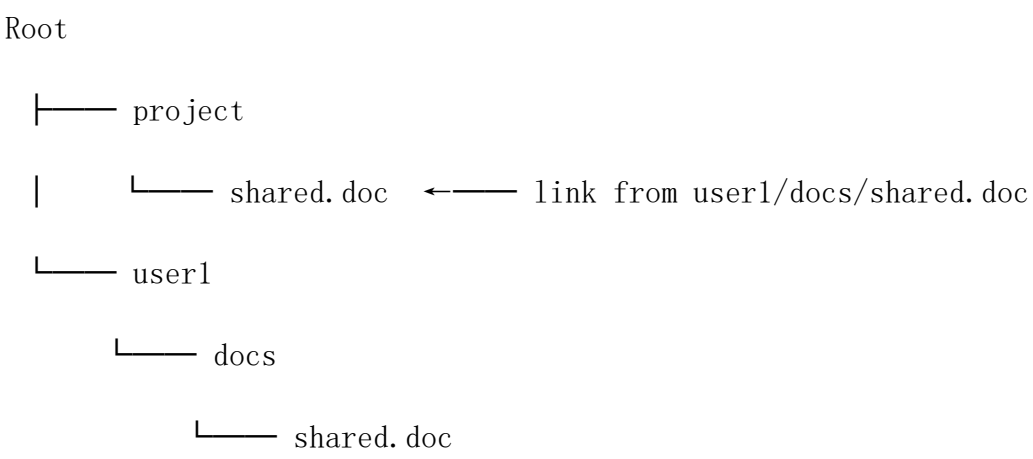
4. Acyclic Graph Directory

Structure:

Like a tree but allows **sharing of files/directories** using links (hard or symbolic).

No cycles are allowed.

Diagram:



Pros:

Supports file sharing without duplication.

Efficient space usage.

Cons:

More complex to implement (especially in tracking shared references).

Difficult to handle deletion (must manage links properly).

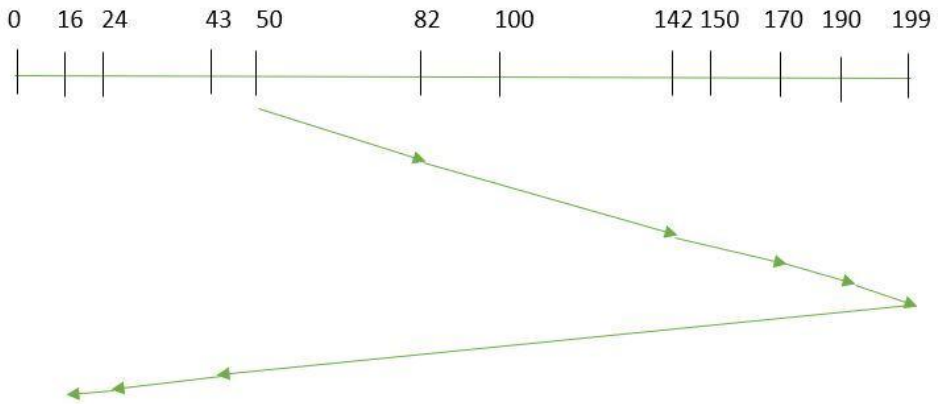
Use Case: Systems requiring file sharing (e.g., collaborative environments).

(8marks)

c) Analyse the following request received for SCAN and C-SCAN disk scheduling algorithm and compute total head movement. Consider current read/write head position as 50.
Request queue : 82,170,43,140,24,16,190

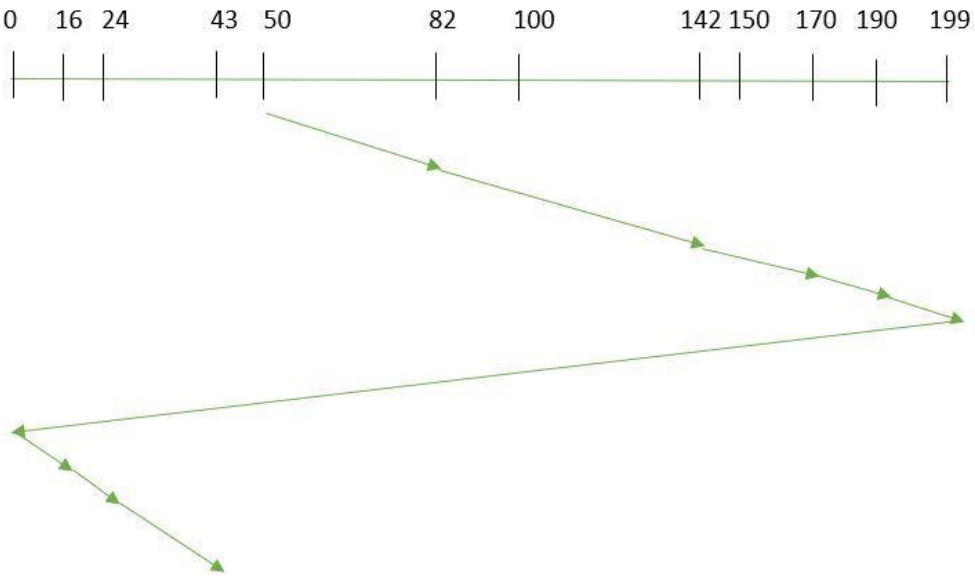
(8marks)

SCAN :



= (199-50) + (199-16) = 332

C-SCAN :

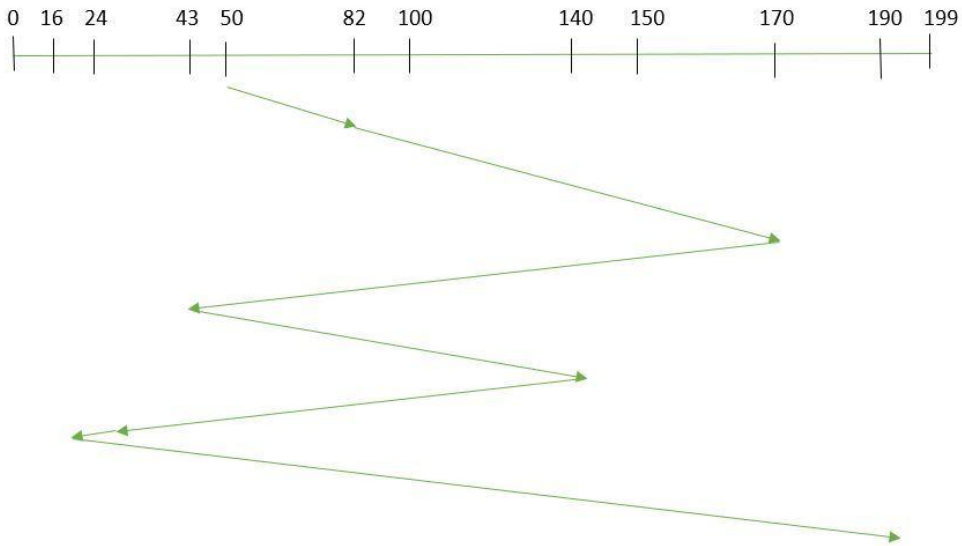


=(199-50) + (199-0) + (43-0) = 391

OR

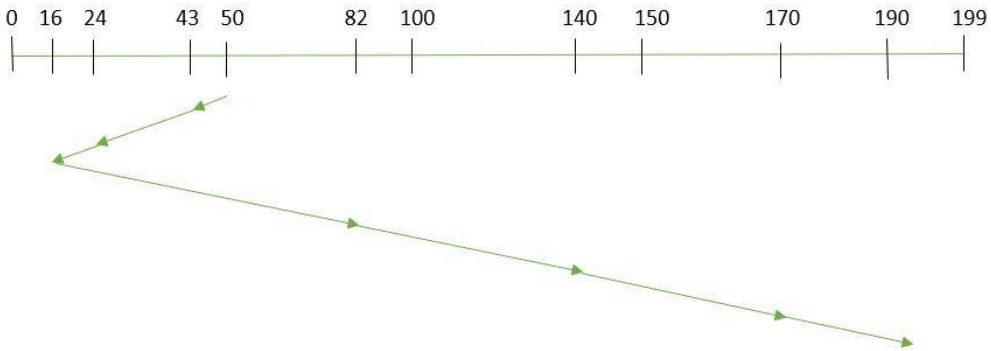
d) Analyse the following request received for FCFS(First come first serve) and SSTF(shortest seek time first) disk scheduling algorithm and compute total head movement. Consider current read/write head position as 50.

Request queue : 82,170,43,140,24,16,190
Answer :
FCFS



So, total overhead movement (total distance covered by the disk arm) =
(82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16) =642

SSTF



total overhead movement (total distance covered by the disk arm) =
(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-140)+(190-170) =208