Disk Scheduling Algorithms

What is Disk Scheduling?

Disk scheduling is the technique used by the **operating system** to decide the **order in which disk I/O requests are serviced**.

👉 Goal: **minimize seek time**, **reduce head movement**, and **improve disk performance**.

Key Terms

- **Seek Time:** Time to move disk head to required track
- **Seek Distance:** Number of tracks crossed
- **Head Position:** Current position of disk head
- **Request Queue:** Pending disk requests

---

1. FCFS (First Come First Serve)

Definition

Requests are served in the **order they arrive**.

Pros

- Simple
- Fair

Cons

- High seek time
- Poor performance

---

2. SSTF (Shortest Seek Time First)

Definition

Selects the request **closest to the current head position**.

Pros

- Better than FCFS
- Lower seek time

Cons

- **Starvation**
- Not fair

---

3. SCAN (Elevator Algorithm)

Definition

Disk head moves in **one direction**, servicing requests, then reverses.

Pros

- No starvation
- Good performance

Cons

- Edge tracks may wait longer

---

4. C-SCAN (Circular SCAN)

Definition

Moves in **one direction only**.
After reaching end, jumps to start **without servicing requests**.

Pros

- Uniform waiting time
- Fair

Cons

- More head movement than SCAN

---

5. LOOK Algorithm

Definition

Like SCAN, but **does not go to disk end** unless needed.

Pros

- Efficient
- Less movement than SCAN

Cons

- Slight complexity

---

6. C-LOOK Algorithm

Definition

Circular version of LOOK.

Pros

- Uniform wait time
- Efficient

Cons

- Jump causes overhead

Comparison Table

Algorithm	Starvation	Seek Time	Performance
FCFS	No	High	Poor
SSTF	Yes	Low	Good
SCAN	No	Medium	Very Good
C-SCAN	No	Medium	Good

Algorithm	Starvation	Seek Time	Performance
LOOK	No	Low	Best
C-LOOK	No	Low	Very Good

1. What is Deadlock?

Deadlock is a situation in which **two or more processes are blocked forever**, each waiting for resources held by the others.

👉 No process can proceed.

Example of Deadlock

- Process P1 holds **Resource R1** and waits for **R2**
- Process P2 holds **Resource R2** and waits for **R1**

➡️ Both processes wait forever.

---

2. Necessary Conditions for Deadlock (Coffman Conditions)

Deadlock occurs **only if all four conditions hold simultaneously**:

1. **Mutual Exclusion** – Resource cannot be shared
2. **Hold and Wait** – Process holds one resource and waits for another
3. **No Preemption** – Resource cannot be forcibly taken
4. **Circular Wait** – Circular chain of processes waiting

---

3. Deadlock Handling Methods

1. Deadlock Prevention
2. Deadlock Avoidance
3. Deadlock Detection & Recovery
4. Ignore deadlock (Ostrich approach)

---

4. Deadlock Avoidance

What is Deadlock Avoidance?

Deadlock avoidance ensures that the system **never enters an unsafe state**.

👉 Requires **advance information** about resource needs.

Safe State

A system is in a **safe state** if there exists a **safe sequence** in which all processes can complete execution.

---

5. Banker's Algorithm (Deadlock Avoidance)

Definition

Banker's Algorithm is a deadlock avoidance algorithm that:

- Allocates resources **only if the system remains in a safe state**
- Works like a bank that lends money cautiously

Data Structures Used

Let:

- **n** = number of processes
- **m** = number of resource types

1. **Available[m]** – available instances of each resource
2. **Max[n][m]** – maximum demand of each process
3. **Allocation[n][m]** – resources currently allocated
4. **Need[n][m] = Max – Allocation**

Banker's Algorithm – Steps

Step 1: Calculate Need matrix

Need = Max − Allocation

Step 2: Find a process whose

Need ≤ Available

Step 3:

- Allocate resources
- Add its Allocation back to Available
- Mark process as finished

Step 4:

- Repeat until all processes finish
- If not possible → **Unsafe state**

---

6. Numerical Example (Banker's Algorithm)

Number of processes = 5 (P0–P4)

Resource types = 3 (A, B, C)

Available Resources

A B C

3 3 2

Allocation Matrix

Process	**A**	**B**	**C**
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

Max Matrix

Process	**A**	**B**	**C**
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

Need Matrix (Max – Allocation)

Process	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

Safety Check

Available = (3, 3, 2)

Safe Sequence:

1. **P1** → Available = (5, 3, 2)
2. **P3** → Available = (7, 4, 3)
3. **P4** → Available = (7, 4, 5)
4. **P0** → Available = (7, 5, 5)
5. **P2** → Available = (10, 5, 7)

✅ **System is in SAFE state**

---

7. Resource Request Algorithm

When process Pi requests resources:

1. Check if Request ≤ Need
2. Check if Request ≤ Available
3. Temporarily allocate
4. Perform safety check
5. If safe → grant request
   Else → deny request

---

8. Advantages of Banker's Algorithm

- Avoids deadlock
- Guarantees safe state
- System stability

---

9. Disadvantages of Banker's Algorithm

- Requires maximum resource info in advance
- High overhead
- Not suitable for real-time systems
- Limited practical use

---

10. Short Exam Definitions

Deadlock:

"Deadlock is a situation where processes wait indefinitely for resources held by each other."

Banker's Algorithm:

"It is a deadlock avoidance algorithm that allocates resources only if the system remains in a safe state."

Device Management (in Operating System)

What is Device Management?

Device management is the function of the operating system that **controls, coordinates, and manages all input/output (I/O) devices** connected to the computer.

👉 The OS acts as an **interface between hardware devices and user programs**.

---

2. Goals of Device Management

- Efficient utilization of I/O devices
- Provide uniform interface to applications
- Handle device conflicts
- Improve system performance
- Ensure device security and protection

---

3. I/O Devices Examples

- **Input Devices:** Keyboard, mouse, scanner
- **Output Devices:** Monitor, printer
- **Storage Devices:** Hard disk, SSD
- **Communication Devices:** Network card

---

4. Functions of Device Management

1. Device Tracking

- Keeps track of **all devices** connected to the system
- Maintains information like device status (free/busy)

---

2. Device Allocation & Deallocation

- Allocates devices to processes when required
- Deallocates devices after use

---

3. I/O Scheduling

- Decides **order of I/O requests**
- Improves device efficiency (e.g., disk scheduling)

---

4. Buffering

- Temporary storage for data during I/O
- Handles speed mismatch between CPU and devices

---

5. Caching

- Stores frequently used data in fast memory
- Reduces access time

---

6. Spooling

- Stores I/O data on disk before processing
- Used for printers

---

7. Error Handling

- Detects and handles I/O errors
- Provides recovery mechanisms

---

5. Device Driver – Definition

A **device driver** is a **software program** that allows the **operating system to communicate with a specific hardware device**.

👉 Without drivers, the OS **cannot control hardware**.

---

6. Role of Device Drivers

1. Hardware Abstraction

- Hides hardware details from OS and applications

- Provides standard interface

---

2. Device Control

- Sends commands to hardware
- Receives status and data

---

3. Interrupt Handling

- Handles hardware interrupts
- Notifies OS when I/O operation is complete

---

4. Data Transfer

- Manages data flow between memory and device
- Uses DMA or programmed I/O

---

5. Error Reporting

- Detects device errors
- Reports to OS for action

---

7. Device Management Architecture

User Program

↓

Operating System

↓

Device Driver

↓

Hardware Device

---

8. Types of Device Drivers

1. **Character Device Drivers**
    - Data transferred character by character
    - Example: Keyboard, mouse
2. **Block Device Drivers**
    - Data transferred in blocks
    - Example: Hard disk, SSD
3. **Network Device Drivers**
    - Handle network communication
    - Example: Network Interface Card (NIC)

---

9. Why Device Drivers Are Important?

- Enable hardware functionality
- Allow new hardware to work without OS redesign
- Improve system portability
- Support plug-and-play devices

DEADLOCK (Overview)

What is Deadlock?

A **deadlock** is a situation where a set of processes are **blocked forever**, each waiting for a resource held by another process.

Simple Example:

- Process P1 holds Resource R1 and waits for R2
- Process P2 holds Resource R2 and waits for R1
  ➡️ Both wait forever → **Deadlock**

Necessary Conditions for Deadlock (Coffman Conditions)

Deadlock can occur **only if all four conditions hold simultaneously**:

1. **Mutual Exclusion** – Resources cannot be shared
2. **Hold and Wait** – Process holds one resource and waits for another
3. **No Preemption** – Resources cannot be forcibly taken
4. **Circular Wait** – Circular chain of processes exists

---

1. DEADLOCK PREVENTION

Definition

Deadlock prevention ensures that **at least one necessary condition never occurs**, so deadlock is impossible.

Techniques of Deadlock Prevention

1. Prevent Mutual Exclusion

- Make resources sharable
  ❌ Not always possible (printer, CPU)

---

2. Prevent Hold and Wait

- Process must request **all resources at once**
- Or release held resources before requesting new ones

Disadvantage:

- Poor resource utilization
- Starvation possible

---

3. Prevent No Preemption

- If a process requests unavailable resource:
    - Force it to **release all held resources**

Used in: CPU scheduling

---

4. Prevent Circular Wait

- Assign **global ordering** to resources
- Processes must request resources in **increasing order**

Example:

R1 → R2 → R3

Advantages of Prevention

✓ Simple concept
✓ Deadlock never occurs

Disadvantages

❌ Low resource utilization
❌ Reduced system performance

---

2. DEADLOCK AVOIDANCE

Definition

Deadlock avoidance allows resource requests **only if the system remains in a safe state**.

👉 System predicts future deadlocks before allocation.

Safe State

A system is in **safe state** if there exists a sequence of processes that can complete execution **without deadlock**.

Banker's Algorithm (Most Important)

Used When:

- Number of resources is known in advance
- Maximum resource need of each process is declared

Data Structures Used:

Name	Description
Available	Free instances of resources
Max	Maximum demand of processes
Allocation	Resources currently allocated
Need	Max – Allocation

Banker's Algorithm Steps:

1. Calculate **Need = Max – Allocation**
2. Find a process whose Need ≤ Available
3. Allocate resources temporarily
4. Release resources after execution
5. Repeat for all processes
6. If all finish → **Safe state**

Advantages of Avoidance

✓ Better resource utilization
✓ Deadlock avoided dynamically

Disadvantages

❌ Needs advance information
❌ High overhead
❌ Not suitable for real-time systems

---

3. DEADLOCK DETECTION

Definition

Deadlock detection allows deadlock to occur and then **detects and recovers** from it.

Detection Techniques

1. Resource Allocation Graph (Single Instance)

- Nodes → Processes & Resources
- Edge:
    - P → R : request
    - R → P : allocated

👉 **Cycle = Deadlock**

P1 → R1 → P2 → R2 → P1

---

2. Detection Algorithm (Multiple Instances)

Similar to Banker's Algorithm but:

- No Max matrix
- Uses:
    - Available
    - Allocation
    - Request

If no process can satisfy request → **Deadlock detected**

Recovery Methods

1. Process Termination

- Abort all deadlocked processes
- Abort one by one until resolved

2. Resource Preemption

- Take resource from one process
- Rollback process

Advantages of Detection

✓ High resource utilization
✓ No restriction on resource request

Disadvantages

❌ Deadlock occurs first
❌ Recovery is costly

COMPARISON TABLE (Very Important for Exams)

Feature	Prevention	Avoidance	Detection
Deadlock Occurrence	Never	Never	Allowed
Resource Utilization	Low	Medium	High
Overhead	Low	High	Medium
Future Knowledge Needed	No	Yes	No
Complexity	Simple	Complex	Moderate
