B.Tech. V-Sem Main/Back/ReBack Exam 2023-24

Computer Science & Engineering

Subject: Operating Systems

PART-A

Q1) Name 5 process control functions/services of system calls.

1. **Process creation and termination:** Allows creating new processes or terminating existing ones.
2. **Process scheduling:** Manages the execution of multiple processes.
3. **Process synchronization:** Ensures orderly execution of processes by providing locks or semaphores.
4. **Inter-process communication (IPC):** Enables communication between processes via shared memory, pipes, etc.
5. **Process management and control:** Includes resource allocation, process suspension, or resumption.

Q2) What is the difference between offline and online operations?

Aspect	Offline Operations	Online Operations
Definition	Processing performed without direct interaction with users.	Processing done in real-time with direct user interaction.
Response Time	Delayed response or batch processing.	Immediate response and real-time processing.
Dependency	Does not require constant system or network connectivity.	Requires constant connectivity to systems or networks.
Examples	Batch job processing, printing.	Online banking, real-time chat applications.

Q3) What is a resident monitor?
A **resident monitor** is a simple operating system component that always resides in memory. It is responsible for managing the execution of jobs in batch systems, handling job transitions, and ensuring efficient utilization of system resources. It loads jobs, executes them, and ensures smooth transition from one job to another.

Q4) Define spooling.
Spooling (Simultaneous Peripheral Operation On-Line) refers to a process where data is temporarily stored on a disk or buffer to manage input/output operations efficiently. It allows multiple jobs to share the same device without waiting for each to complete, like sending multiple print jobs to a printer queue.

Q5) What do you mean by multiprogramming?
Multiprogramming is a method where multiple programs are loaded into the memory and executed by the CPU concurrently. It maximizes CPU utilization by ensuring that the CPU always has a job to execute, reducing idle time.

Q6) What is the concept of CPU protection?
CPU protection ensures that the CPU is allocated and used fairly among processes. Key components include:

1. **Timer interrupts:** Prevents one process from monopolizing the CPU.
2. **Privileged instructions:** Certain operations can only be executed in kernel mode to avoid misuse.
3. **Context switching:** Ensures a smooth transition between processes.

Q7) Is there any difference between a process and a program?

Aspect	Process	Program
Definition	An active instance of an executing program.	A passive set of instructions stored on disk.
State	Dynamic (changes during execution).	Static (remains the same).
Lifespan	Temporary (exists during execution).	Permanent (until deleted).
Components	Contains code, data, stack, and registers.	Only code and instructions.

Q8) What is the ostrich algorithm regarding deadlocks?
The **ostrich algorithm** is a method of ignoring deadlocks under the assumption that they are rare or their cost of handling outweighs their impact. This approach is used when the system can tolerate occasional deadlocks, relying on manual intervention or system restarts to resolve them.

Q9) What is the concept of worst-fit memory allocation?
In **worst-fit memory allocation**, the largest available memory block is allocated to a process. It leaves the biggest leftover block to minimize fragmentation. This approach is less efficient as it can lead to unusable small memory fragments.

Q10) What do you understand by file access control?
File access control is a security mechanism that regulates who can access, read, write, or execute files in a system. Common methods include:

1. **Access Control Lists (ACLs):** Define permissions for users/groups.
2. **File permissions:** Set read, write, and execute privileges.
3. **Authentication and authorization:** Verify the identity and permissions of users.

PART-B

Q1: Difference between Real-Time Systems and Time-Sharing Systems.

Feature	Real-Time Systems	Time-Sharing Systems
Purpose	Used for critical tasks where responses are required immediately.	Used for sharing system resources among multiple users efficiently.
Response Time	Extremely low and predictable (must meet strict deadlines).	Moderate response time, but delays are acceptable.
Examples	Industrial control systems, medical devices, flight	Multi-user operating systems like Linux, Unix, or Windows.

	systems.	
Task Priority	Tasks are assigned strict priorities, and high-priority tasks are executed first.	Tasks are scheduled in a round-robin or priority-based manner to ensure fairness.
User Interaction	Minimal to no user interaction; works autonomously.	Designed for heavy user interaction (e.g., via terminals).
Failure Consequences	A missed deadline can cause critical failures or disasters.	Delays or system failures are inconvenient but not catastrophic.
Scheduling Algorithms	Often uses Rate Monotonic or Earliest Deadline First (EDF).	Uses Round Robin, Shortest Job Next, or similar algorithms.

Example:

- **Real-Time System**: Airplane autopilot system, where a delayed response can result in accidents.
- **Time-Sharing System**: A shared university server where students run programs simultaneously

---

Q2: What is the concept of Memory Protection?

Memory Protection is a mechanism provided by operating systems to prevent one process from accessing the memory allocated to another process or the operating system itself. This ensures **data security**, **system stability**, and **multi-programming efficiency**.

Key Concepts of Memory Protection

1. **Base and Limit Registers**:

   - **Base Register**: Holds the smallest physical memory address that a process can access.
   - **Limit Register**: Holds the size of the memory allocated to the process. It ensures that the process cannot access memory beyond this limit.
   - Together, these registers ensure that each process stays within its own memory boundaries.

2. **Protection During Execution**:

   - During a program's execution, the CPU checks every memory reference to ensure it falls within the allowed range (base + limit).
   - Any violation generates a **trap** or **exception** (e.g., segmentation fault).

3. **Segmentation and Paging**:

   - Memory protection can also be implemented using segmentation or paging, where processes operate on virtual memory addresses mapped to physical addresses.

Advantages of Memory Protection

1. Prevents accidental or malicious memory access by processes.
2. Increases system stability by isolating process memory.
3. Supports multi-programming by ensuring memory safety.

Violation Example

If a process tries to access memory outside the range specified by the **base** and **limit** registers, the OS triggers an exception (e.g., **segmentation fault**).

Q3: Differentiate between External Fragmentation and Internal Fragmentation

Aspect	External Fragmentation	Internal Fragmentation
Definition	Occurs when free memory is scattered in small, non-contiguous blocks, making it impossible to allocate memory to processes, even if the total free space is sufficient.	Occurs when allocated memory to a process exceeds its actual requirement, leaving unused space within the allocated block.
Cause	Arises due to dynamic allocation of memory in variable-sized blocks.	Arises due to fixed-sized memory allocation (e.g., paging or block allocation).
Memory Wastage	Wastes memory **outside** allocated blocks (free fragments between allocations).	Wastes memory **inside** allocated blocks (unused reserved space).
Example	Suppose there are free spaces of 10KB, 20KB, and 15KB, and a process requiring 25KB cannot be allocated, even though the total free space is 45KB.	If a process requires 18KB, but a fixed block of 20KB is allocated, 2KB is wasted as internal fragmentation.
Solution	1. Compaction (rearrange memory to create contiguous free space). Paging and segmentation (use logical addressing).	1. Use variable-sized partitions. Reduce block size to minimize wasted space.

Q4: Advantages and Disadvantages of Paged Memory Management

Paged memory management is a memory allocation technique where a process's logical memory is divided into fixed-size blocks called **pages**, and physical memory is divided into blocks of the same size called **frames**. Pages are loaded into available frames.

Advantages of Paged Memory Management

1. **Eliminates External Fragmentation**:
   - Since pages and frames are of the same size, there are no gaps between allocated memory blocks, avoiding external fragmentation.
2. **Efficient Memory Utilization**:
   - Small, fixed-sized pages allow better utilization of memory as processes can be split across non-contiguous frames.
3. **Simplifies Memory Allocation**:
   - Physical memory management becomes easier because processes can be loaded into any available frame.
4. **Supports Virtual Memory**:
   - Paging works seamlessly with virtual memory, allowing processes larger than the available physical memory to run by swapping pages in

and out of secondary storage.

5. **Faster Process Switching**:

   - Pages can be loaded or swapped independently, speeding up process switching.

Disadvantages of Paged Memory Management

1. **Internal Fragmentation**:

   - If a process's last page doesn't fill a frame completely, the unused space results in internal fragmentation.

2. **Overhead of Page Tables**:

   - Each process requires a **Page Table** to map logical pages to physical frames, which consumes additional memory and can slow down lookups.

3. **Translation Time Overhead**:

- Translating logical addresses to physical addresses requires the use of a **Memory Management Unit (MMU)**, adding processing time.

4. **Page Faults**:

   - If a required page is not in memory, a **page fault** occurs, requiring the page to be fetched from secondary storage, causing delays.

Q5) What are system programs? Explain different categories of system programs to provide operating system services.

What are system programs?

System programs are a collection of programs that provide a user interface to the hardware and operating system services. They help in managing, maintaining, and controlling computer resources effectively. These programs act as intermediaries between users and the operating system, simplifying the complexity of managing hardware resources.

Key Features of System Programs:

1. They ensure smooth functioning of the system.
2. Provide an abstraction for hardware functionality.
3. Improve user productivity by offering tools and utilities.

Categories of System Programs

1. **File Management Programs**

   - Handle file creation, deletion, reading, and writing.
   - Examples: Commands like cp, rm, ls, or file utilities in GUI.

2. **Device Management Programs**

   - Enable users to manage and interact with I/O devices such as printers, disks, or USBs.
   - Examples: Drivers and utility programs like Device Manager in Windows.

3. **Information Management Programs**

   - Provide details about system performance, configurations, and status.
   - Examples: Task Manager, top, or df.

4. **Communication Programs**

   - Allow data transfer between systems over a network.
   - Examples: FTP clients, Telnet, or Ping tools.

5. **Process Management Programs**

   - Help in process execution, scheduling, and termination.
   - Examples: ps, kill, or job scheduling commands like cron.

6. **Utilities for System Maintenance**

   - Programs that maintain system health, remove unnecessary files, or check for errors.
   - Examples: Disk defragmenters, antivirus programs, and backup utilities.

7. **Programming Language Support**

   - Tools that help developers write, debug, and execute programs.
   - Examples: Compilers, interpreters, and debuggers like gcc or Python IDEs.

Q6) What are the necessary conditions of deadlock? Explain.

What is a Deadlock?

A deadlock is a situation in which a group of processes is permanently blocked because each process is waiting for a resource that another

process holds. This creates a circular wait condition, where no process can proceed further.

Necessary Conditions for Deadlock

Deadlock occurs if the following four conditions are satisfied simultaneously:

Condition	Explanation
1. Mutual Exclusion	At least one resource must be held in a non-shareable mode. Only one process can use the resource at a time.
2. Hold and Wait	A process holding at least one resource is waiting to acquire additional resources held by other processes.
3. No Preemption	A resource cannot be forcibly taken from a process; it must be released voluntarily by the holding process.
4. Circular Wait	A set of processes exists where each process is waiting for a resource that the next process in the chain holds, forming a circular chain of waits.

Example of Deadlock

1.

Processes and Resources:

- Process A holds Resource 1 and requests Resource 2.
- Process B holds Resource 2 and requests Resource 1.

2.

Deadlock Scenario:

- Process A cannot proceed because it needs Resource 2, which is held by Process B.
- Process B cannot proceed because it needs Resource 1, which is held by Process A.

3.

Result: Circular waiting occurs, causing a deadlock.

Q7) Show the relationship among pages, blocks, PMTAR, and PMTs in memory paging, using an example.

Memory Paging Concept

Paging is a memory management technique that divides both the physical memory and the logical address space into fixed-size units called **blocks** (physical memory) and **pages** (logical memory). The **Page Table** maps each logical page to its corresponding physical block.

Key Terms

1. **Pages**: Fixed-size units of logical memory.
2. **Blocks (Frames)**: Fixed-size units of physical memory.
3. **Page Table**: Maintains the mapping between pages and blocks.
4. **PMTAR (Page Table Address Register)**: Holds the base address of the Page Table in memory for the CPU to access.

Relationship with Example

Let’s assume:

- Logical memory has **4 pages (P0, P1, P2, P3)**.
- Physical memory has **5 blocks (B0, B1, B2, B3, B4)**.
- Page size = Block size = 1 KB.

Mapping in Paging

Page Number	Mapped Block Number
P0	B3
P1	B0
P2	B4
P3	B2

1. **PMTAR**: Points to the starting address of the Page Table in memory.
2. **PMT (Page Mapping Table)**: The table contains the mapping of each page to its block.

How It Works

1. The CPU generates a **logical address**, consisting of:

   - Page number (Pn).
   - Offset (within the page).

2. **Translation Steps**:

   - The CPU uses **PMTAR** to locate the Page Table.
   - The Page Table maps the page number to the corresponding block number.
   - Add the offset to the base address of the block to get the final physical address.

Example Walkthrough

Logical Address: P2, Offset = 400 bytes

- From the Page Table: **P2 maps to B4**.
- Physical Address = Base of **B4** + Offset = (Base of B4) + 400 bytes.

Q8) Memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB are available. How will Partitioned Allocation place processes of sizes 212 KB, 417 KB, 112 KB, and 426 KB in sequence?

Concept of Partitioned Allocation

Partitioned allocation is a memory management technique where physical memory is divided into fixed or variable-sized partitions, and processes are allocated to these partitions based on their size.

There are different strategies for allocating memory:

1. **First Fit**: Allocates the first available partition that is large enough for the process.
2. **Best Fit**: Allocates the smallest partition that is just big enough for the process.
3. **Worst Fit**: Allocates the largest available partition.

For this explanation, let's assume **First Fit** as the allocation strategy.

Given Data

- **Memory Partitions**: 100 KB, 500 KB, 200 KB, 300 KB, 600 KB
- **Process Sizes**: 212 KB, 417 KB, 112 KB, 426 KB

Step-by-Step Allocation (First Fit)

Process	Size (KB)	Partition Allocated	Reason
P1	212	500 KB	First partition large enough.
P2	417	600 KB	First partition large enough.
P3	112	200 KB	First partition large enough.
P4	426	**Not Allocated**	No partition is large enough.

Final Partition Status

Partition	Size (KB)	Status	Remaining Space
100 KB	100 KB	Free	100 KB
500 KB	500 KB	Allocated to P1	288 KB
200 KB	200 KB	Allocated to P3	88 KB
300 KB	300 KB	Free	300 KB
600 KB	600 KB	Allocated to P2	183 KB

Unallocated Process

- Process **P4 (426 KB)** could not be allocated because no available partition was large enough.

PART-C

Q1) For the following set of processes, find the Average Waiting Time, showing the Gantt Chart for the following scheduling algorithms:

i) Shortest Job First (SJF)
ii) Priority Scheduling

Process	Burst Time (BT)	Priority	Arrival Time
P1	5	5	1
P2	3	4	0

P3	8	3	3
P4	2	1	2
P5	1	2	4

ANSWER: Given Data

Process	Burst Time (BT)	Priority	Arrival Time
P1	5	5	1
P2	3	4	0
P3	8	3	3
P4	2	1	2
P5	1	2	4

i) Shortest Job First (SJF)

In **SJF**, the process with the shortest burst time is executed first, considering arrival times. Let's sort the processes by their arrival times and then choose the one with the smallest burst time to execute next.

Step 1: Sort the processes by their arrival time and burst time.

Process	Burst Time (BT)	Arrival Time	Execution Order
P2	3	0	1st
P4	2	2	2nd
P5	1	4	3rd

P1	5	1	4th
P3	8	3	5th

Step 2: Create the Gantt Chart.

- **P2** arrives first (at t = 0) and has the shortest burst time (3 ms), so it executes first.
- After **P2** completes at t = 3, **P4** is the shortest remaining process, so it executes next.
- Then **P5** executes, followed by **P1** and finally **P3**.

Gantt Chart	P2	P4	P5	P1	P3
Time (t)	0-3	3-5	5-6	6-11	11-19

Step 3: Calculate Waiting Time (WT) for each process.

- **Waiting Time** = Start Time - Arrival Time

Process	Arrival Time	Start Time	Burst Time	Completion Time	Waiting Time (WT)
P2	0	0	3	3	0
P4	2	3	2	5	1
P5	4	5	1	6	1
P1	1	6	5	11	5
P3	3	11	8	19	8

Step 4: Calculate Average Waiting Time.

Average Waiting Time=(0+1+1+5+8)/5=15/5=3 ms

---

ii) Priority Scheduling

In **Priority Scheduling**, the process with the highest priority (smallest number) is executed first. If multiple processes have the same priority, we schedule the one with the smallest arrival time.

Step 1: Sort the processes by their arrival time, then by priority.

Process	Burst Time (BT)	Priority	Arrival Time	Execution Order
P2	3	4	0	1st
P5	1	2	4	2nd
P4	2	1	2	3rd
P1	5	5	1	4th
P3	8	3	3	5th

Step 2: Create the Gantt Chart.

- **P2** arrives first (at t = 0) and has the highest priority (priority 4).
- After **P2** completes at t = 3, **P5** has the highest priority (priority 2) and executes next.
- Then **P4** executes, followed by **P3** and finally **P1**.

Gantt Chart	P2	P5	P4	P3	P1

Time (t)	0-3	3-4	4-6	6-14	14-19

Step 3: Calculate Waiting Time (WT) for each process.

Process	Arrival Time	Start Time	Burst Time	Completion Time	Waiting Time (WT)
P2	0	0	3	3	0
P5	4	3	1	4	0
P4	2	4	2	6	2
P3	3	6	8	14	11
P1	1	14	5	19	13

Step 4: Calculate Average Waiting Time.

Average Waiting Time=(0+0+2+11+13)5=265=5.2 ms

Final Results

Scheduling Algorithm	Average Waiting Time
SJF	**3 ms**
Priority Scheduling	**5.2 ms**

Q:2) Apply FCFS scheduling algorithm. Find the average Turnaround time and average Waiting time for the processes given below.

Answer

Given Data

Process	Burst Time (ms)	Arrival Time (ms)
P1	14	2
P2	3	1
P3	5	3

Step 1: Arrange Processes by Arrival Time

FCFS executes processes in the order of their arrival. Sorting by Arrival Time:

Process	Burst Time (ms)	Arrival Time (ms)
P2	3	1
P1	14	2
P3	5	3

Step 2: Calculate Completion Time (CT)

Completion Time is the time at which a process finishes execution.

- **P2** starts at **1 ms** (its arrival time).
- **P1** starts after P2 completes.
- **P3** starts after P1 completes.

Process	Arrival Time	Burst Time	Start Time	Completion Time (CT)
P2	1	3	1	4
P1	2	14	4	18
P3	3	5	18	23

Step 3: Calculate Turnaround Time (TAT)

Turnaround Time = Completion Time - Arrival Time

Process	Completion Time (CT)	Arrival Time (AT)	Turnaround Time (TAT)
P2	4	1	3
P1	18	2	16
P3	23	3	20

Step 4: Calculate Waiting Time (WT)

Waiting Time = Turnaround Time - Burst Time

Process	Turnaround Time (TAT)	Burst Time (BT)	Waiting Time (WT)
P2	3	3	0
P1	16	14	2
P3	20	5	15

Step 5: Calculate Averages

- **Average Turnaround Time (TAT)**:
  Average TAT=Total TAT/Number of Processes=(3+16+20)/3=39/3=13 ms

- **Average Waiting Time (WT)**:
  Average WT=Total WT/Number of Processes=(0+2+15)/3=17/3=5.67 ms

---

Final Results

Metric	Value
Average Turnaround Time	**13 ms**
Average Waiting Time	**5.67 ms**

Q:3) Consider the following page reference string 1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6. Find out the number of page faults, if there are 4-page frames, using the following Page Replacement Algorithm

(i) LRU (ii) FIFO.

Given Data

- **Page Reference String**: 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6
- **Number of Page Frames**: 4

---

1. LRU (Least Recently Used) Algorithm

LRU replaces the page that hasn't been used for the longest time. Let's simulate the page accesses and replacements:

Page Reference	Page Frames (State)	Page Fault?	Explanation
1	[1]	Yes	Page 1 is loaded
2	[1, 2]	Yes	Page 2 is loaded
3	[1, 2, 3]	Yes	Page 3 is loaded
4	[1, 2, 3, 4]	Yes	Page 4 is loaded
2	[1, 2, 3, 4]	No	Page 2 is already in frame

1	[1, 2, 3, 4]	No	Page 1 is already in frame
5	[2, 3, 4, 5]	Yes	Page 5 replaces Page 1
6	[3, 4, 5, 6]	Yes	Page 6 replaces Page 2
2	[4, 5, 6, 2]	Yes	Page 2 replaces Page 3
1	[5, 6, 2, 1]	Yes	Page 1 replaces Page 4
2	[5, 6, 1, 2]	No	Page 2 is already in frame
3	[6, 1, 2, 3]	Yes	Page 3 replaces Page 5
7	[1, 2, 3, 7]	Yes	Page 7 replaces Page 6
6	[2, 3, 7, 6]	Yes	Page 6 replaces Page 1
3	[2, 7, 6, 3]	No	Page 3 is already in frame
2	[7, 6, 3, 2]	No	Page 2 is already in frame
1	[7, 6, 3, 1]	Yes	Page 1 replaces Page 7
2	[6, 3, 1, 2]	No	Page 2 is already in frame
3	[6, 1, 2, 3]	No	Page 3 is already in frame
6	[1, 2, 3, 6]	No	Page 6 is already in frame

LRU Page Fault Count

- **Total Page Faults**: 14

---

2. FIFO (First In, First Out) Algorithm

FIFO replaces the oldest page in memory. Let's simulate the page accesses and replacements:

Page Reference	Page Frames (State)	Page Fault?	Explanation
1	[1]	Yes	Page 1 is loaded
2	[1, 2]	Yes	Page 2 is loaded
3	[1, 2, 3]	Yes	Page 3 is loaded
4	[1, 2, 3, 4]	Yes	Page 4 is loaded
2	[1, 2, 3, 4]	No	Page 2 is already in frame
1	[1, 2, 3, 4]	No	Page 1 is already in frame
5	[2, 3, 4, 5]	Yes	Page 5 replaces Page 1
6	[3, 4, 5, 6]	Yes	Page 6 replaces Page 2
2	[4, 5, 6, 2]	Yes	Page 2 replaces Page 3
1	[5, 6, 2, 1]	Yes	Page 1 replaces Page 4
2	[5, 6, 2, 1]	No	Page 2 is already in frame
3	[6, 2, 1, 3]	Yes	Page 3 replaces Page 5
7	[2, 1, 3, 7]	Yes	Page 7 replaces Page 6
6	[1, 3, 7, 6]	Yes	Page 6 replaces Page 2
3	[1, 7, 6, 3]	No	Page 3 is already in frame
2	[7, 6, 3, 2]	Yes	Page 2 replaces Page 1
1	[6, 3, 2, 1]	Yes	Page 1 replaces Page 7
2	[6, 3, 1, 2]	No	Page 2 is already in frame
3	[6, 1, 2, 3]	No	Page 3 is already in frame
6	[1, 2, 3, 6]	No	Page 6 is already in frame

FIFO Page Fault Count

- **Total Page Faults**: 15

Final Results

Page Replacement Algorithm	Total Page Faults
LRU	14
FIFO	15

Q:4) Five batch jobs A, B, C, D and E arrive at a computer center at almost at the same time. They have estimated running times of 10,6,2,4 and 8 minutes. Their priorities are 3,5,2,1 and 4 respectively, with 5 being the highest priority. For the following scheduling algorithms determine the Average Turnaround Time and Average Waiting Time. Mention which algorithm results in minimal average waiting time.

i) Round Robin

ii) Priority scheduling

For case (i) assume that each job gets its fair share of the CPU. (Time Quantum 3 minutes).

For case (ii) assume that only one job runs at a time, until it finishes.

Given Data

- **Jobs**: A, B, C, D, E
- **Burst Times (BT)**: 10, 6, 2, 4, 8
- **Priorities**: 3, 5, 2, 1, 4
  (1 = lowest priority, 5 = highest priority)

Case (i): Round Robin (Time Quantum = 3 minutes)

Steps for Round Robin Scheduling

1. **Time Quantum**: Each job gets **3 minutes** of CPU time per round.
2. Repeat until all jobs are finished.

Job	Burst Time (BT)
A	10

B	6
C	2
D	4
E	8

Execution Order

- **Round 1**: A(3), B(3), C(2), D(3), E(3)
  Remaining Burst Times: A(7), B(3), D(1), E(5)
- **Round 2**: A(3), B(3), D(1), E(3)
  Remaining Burst Times: A(4), E(2)
- **Round 3**: A(3), E(2)
  Remaining Burst Times: A(1)
- **Round 4**: A(1)

Completion Times (CT)

- **A**: 23
- **B**: 12
- **C**: 5
- **D**: 10
- **E**: 21

Turnaround Time (TAT) and Waiting Time (WT)

- **TAT** = CT - Arrival Time (AT) (Assume AT = 0 for all jobs).
- **WT** = TAT - Burst Time (BT).

Job	BT	CT	TAT = CT - AT	WT = TAT - BT
A	10	23	23	13
B	6	12	12	6
C	2	5	5	3
D	4	10	10	6
E	8	21	21	13

Averages for Round Robin

- **Average TAT**: Average TAT=(23+12+5+10+21)/5=71/5=14.2 minutes
- **Average WT**: Average WT=(13+6+3+6+13)/5=41/5=8.2 minutes

---

Case (ii): Priority Scheduling

Steps for Priority Scheduling

- Jobs are scheduled based on **priority** (higher number = higher priority).
- If two jobs have the same priority, they are scheduled in arrival order.

Job	Priority	Burst Time (BT)
B	5	6
E	4	8
A	3	10

C	2	2
D	1	4

Execution Order

- Order: **B → E → A → C → D**

Completion Times (CT)

- **B**: 6
- **E**: 14
- **A**: 24
- **C**: 26
- **D**: 30

Turnaround Time (TAT) and Waiting Time (WT)

- **TAT** = CT - AT (Assume AT = 0 for all jobs).
- **WT** = TAT - BT.

Job	BT	Priority	CT	TAT = CT - AT	WT = TAT - BT
B	6	5	6	6	0
E	8	4	14	14	6
A	10	3	24	24	14
C	2	2	26	26	24
D	4	1	30	30	26

Averages for Priority Scheduling

- **Average TAT**: Average TAT=(6+14+24+26+30)/5=100/5=20.0 minutes
- **Average WT**: Average WT=(0+6+14+24+26)/5=70/5=14.0 minutes

---

Comparison of Results

Metric	Round Robin (RR)	Priority Scheduling
Average TAT	14.2 minutes	20.0 minutes
Average WT	8.2 minutes	14.0 minutes

Conclusion

- **Round Robin** results in the **minimal average waiting time** of **8.2 minutes**, as it ensures fair CPU sharing among jobs.
- However, Priority Scheduling may be preferred when certain tasks must be prioritized based on their importance.

Q:5) Five batch jobs L, M, N, O and R arrive at a Data Warehouse. Their estimated CPU burst times are 10,6,2,4 and 8 seconds. For the given scheduling algorithms

determine the turnaround time and waiting time of each process and also the average. Mention which algorithm results in minimal average waiting time.

i) First Come First Served

ii) Shortest Job First.

Assume that only one job runs at a time, until it finishes. All jobs are completely CPU bound.

Let's solve the problem step by step for the given jobs (**L, M, N, O, R**) with their burst times (**10, 6, 2, 4, 8**) using the two scheduling algorithms: **First Come First Served (FCFS)** and **Shortest Job First (SJF)**.

---

1. First Come First Served (FCFS)

Assumption: The jobs arrive in the order L, M, N, O, R.

Burst Times

- **L**: 10, **M**: 6, **N**: 2, **O**: 4, **R**: 8

Calculation of Turnaround Time (TAT) and Waiting Time (WT)

1. **Turnaround Time (TAT)** = Completion Time (CT) - Arrival Time (AT)
   (Assume all jobs arrive at time 0.)
2. **Waiting Time (WT)** = Turnaround Time (TAT) - Burst Time (BT)

Job	Burst Time (BT)	Completion Time (CT)	Turnaround Time (TAT)	Waiting Time (WT)
L	10	10	10	0
M	6	16	16	10
N	2	18	18	16
O	4	22	22	18
R	8	30	30	22

Summary for FCFS

- **Average Turnaround Time (TAT)**: Average TAT=(10+16+18+22+30)/5=96/5=19.2 seconds
- **Average Waiting Time (WT)**: Average WT=(0+10+16+18+22)/5=66/5=13.2 seconds

---

2. Shortest Job First (SJF)

Assumption: All jobs are available at time 0. SJF schedules the job with the **shortest burst time** first.

Reorder Jobs by Burst Time

- Order: **N (2) → O (4) → M (6) → R (8) → L (10)**

Calculation of Turnaround Time (TAT) and Waiting Time (WT)

Job	Burst Time (BT)	Completion Time (CT)	Turnaround Time (TAT)	Waiting Time (WT)
N	2	2	2	0
O	4	6	6	2
M	6	12	12	6
R	8	20	20	12
L	10	30	30	20

Summary for SJF

- **Average Turnaround Time (TAT)**: Average TAT=(2+6+12+20+30)/5=70/5=14.0 seconds
- **Average Waiting Time (WT)**: Average WT=(0+2+6+12+20)/5=40/5=8.0 seconds

---

Comparison of Results

Metric	FCFS	SJF
Average TAT (sec)	19.2	14.0
Average WT (sec)	13.2	8.0

Conclusion

- **Shortest Job First (SJF)** results in the **minimal average waiting time (8.0 seconds)** and also achieves a lower average turnaround time compared to FCFS.
- SJF is more efficient in scenarios where all jobs arrive simultaneously and CPU burst times vary significantly.

Q:6) Explain Belady's anomaly in paging system, with a suitable example.

Belady's Anomaly in Paging System

Belady's Anomaly refers to the counterintuitive situation in a **paging system** where increasing the number of page frames (memory blocks) results in **more page faults**, rather than fewer. This phenomenon is observed in certain page replacement algorithms, such as **FIFO (First-In-First-Out)**.

Why Does Belady's Anomaly Occur?

This happens because the **FIFO page replacement algorithm** removes the oldest page from memory, regardless of how frequently or recently it is used. Adding more frames can sometimes disrupt the existing sequence of pages in memory, leading to an increase in page faults.

Example to Demonstrate Belady's Anomaly

Given:

- **Page Reference String**: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- **Page Replacement Algorithm**: FIFO
- Compare for **3 frames** and **4 frames**.

Case 1: 3 Frames

Step	Reference	Memory (Frames)	Page Fault
1	1	1	Fault
2	2	1, 2	Fault
3	3	1, 2, 3	Fault
4	4	4, 2, 3	Fault
5	1	1, 4, 3	Fault
6	2	1, 2, 3	Fault
7	5	5, 2, 3	Fault
8	1	1, 2, 3	Fault
9	2	1, 2, 3	No Fault
10	3	1, 2, 3	No Fault
11	4	4, 2, 3	Fault

12	5	5, 2, 3	Fault

- **Total Page Faults: 9**

Case 2: 4 Frames

Step	Reference	Memory (Frames)	Page Fault
1	1	1	Fault
2	2	1, 2	Fault
3	3	1, 2, 3	Fault
4	4	1, 2, 3, 4	Fault
5	1	1, 2, 3, 4	No Fault
6	2	1, 2, 3, 4	No Fault
7	5	5, 2, 3, 4	Fault
8	1	5, 1, 3, 4	Fault
9	2	5, 1, 2, 4	Fault
10	3	5, 1, 2, 3	Fault
11	4	4, 1, 2, 3	Fault
12	5	5, 1, 2, 3	Fault

- **Total Page Faults: 10**

Observation

- With **3 frames**, the total page faults are **9**.
- With **4 frames**, the total page faults increase to **10**.
  This demonstrates **Belady's Anomaly**: increasing the number of frames has caused more page faults.

Conclusion

Belady's Anomaly highlights a limitation of the FIFO page replacement algorithm. Algorithms like **LRU (Least Recently Used)** and **Optimal Page Replacement** do not suffer from this anomaly, as they make more informed decisions about which pages to replace.

Q:7) Explain Banker's algorithm for deadlock avoidance using a suitable example.

Banker's Algorithm for Deadlock Avoidance

The Banker's Algorithm is a deadlock avoidance algorithm used in operating systems to allocate resources safely to processes while avoiding deadlocks. It ensures that the system remains in a *safe state* by checking if resources can be allocated to a process without causing a deadlock.

Key Concepts

1. **Safe State**: A state where at least one sequence of processes exists that can finish execution without causing a deadlock.
2. **Data Structures Used**
   :
   - **Max Matrix**: Maximum resources each process may need.
   - **Allocation Matrix**: Current resources allocated to each process.
   - **Need Matrix**: Remaining resources required by each process (Need = Max - Allocation).
   - **Available Resources**: Total free resources in the system.

Steps of Banker's Algorithm

1. **Check Need ≤ Available**: For a process to execute, its needed resources must be less than or equal to the currently available resources.
2. **Simulate Resource Allocation**
   :
   - Temporarily allocate resources to the process.
   - Update the Available resources and the Allocation matrix.

3. **Check System Safety**
   :
   - Determine if all processes can execute to completion in some order.
   - If yes, the system is in a safe state; otherwise, deny the request.

Example of Banker's Algorithm

Scenario

- Total Resources: [10, 5, 7] (3 types of resources)
- Processes: 3 (P1, P2, P3)
- Allocation, Max, and Available matrices are as follows:

Process	Allocation	Max	Need
P1	[1, 0, 1]	[7, 5, 3]	[6, 5, 2]
P2	[2, 1, 2]	[3, 2, 2]	[1, 1, 0]
P3	[3, 1, 4]	[9, 0, 2]	[6, -1, -2]

Let’s refine the example for clarity and correctness.

Example of Banker's Algorithm

Scenario

- **Total Resources**: [10, 5, 7]
  (3 types of resources: A, B, and C)
- **Allocation Matrix (resources currently allocated to processes)**:

Process	A	B	C
P1	1	0	1
P2	2	1	1
P3	3	1	2

- **Maximum Matrix (maximum resources each process might need)**:

Process	A	B	C
P1	7	5	3
P2	3	2	2
P3	9	0	2

- **Available Resources**: [3, 3, 3]
  (calculated as Total - Σ(Allocation) for each resource type).

Step 1: Calculate the Need Matrix

The Need Matrix is calculated as:

Need=Max−Allocation

Process	A	B	C
P1	6	5	2
P2	1	1	1
P3	6	0	0

Step 2: Safe State Check

We simulate the allocation process for each process and check if the system is in a safe state.

1. **Initial Available Resources**: [3, 3, 3]

2. **Process P2** can be executed first (its Need [1, 1, 1] ≤ Available [3, 3, 3]):

   - Allocate resources to P2 temporarily.
   - Update Available: Available=Available+Allocation(P2)=[3,3,3]+[2,1,1]=[5,4,4]
   - P2 completes.

3.

Process P1 can now execute (its Need [6, 5, 2] ≤ Available [5, 4, 4] after adjustment):

- Allocate resources to P1 temporarily.
- Update Available: Available=Available+Allocation(P1)=[5,4,4]+[1,0,1]=[6,4,5]
- P1 completes.

4.

Process P3 can now execute (its Need [6, 0, 0] ≤ Available [6, 4, 5]):

- Allocate resources to P3 temporarily.
- Update Available: Available=Available+Allocation(P3)=[6,4,5]+[3,1,2]=[9,5,7]
- P3 completes.

Step 3: Safe Sequence

The processes can execute in the order **P2 → P1 → P3** without causing a deadlock. Hence, the system is in a **safe state**.
