SECTION – A

1. What do you understand by O.S. by giving real life example.

An Operating System (O.S.) is software that manages hardware and software resources in a computer. It acts as a bridge between users and hardware.

Example: Like a hotel manager organizes staff, rooms, and services for guests, an O.S. manages processes, memory, and devices for users.

2. Which feature of O.S. is most important and why?

Multitasking is one of the most important features of an O.S. It allows multiple programs to run simultaneously, improving system efficiency and user productivity. For instance, users can browse the web, play music, and download files concurrently without system interruption.

3. Write down the types of O.S. available.

Types of Operating Systems:

1. **Batch OS**: Executes jobs in batches without user interaction.
2. **Time-Sharing OS**: Provides multitasking with time slots.
3. **Distributed OS**: Shares resources across multiple systems.
4. **Real-Time OS**: Processes data in real-time for critical tasks.

5. **Mobile OS**: Designed for smartphones and tablets.

4. What do you understand by Pre-emptive and Non Pre-emptive scheduling algorithm.

Pre-emptive Scheduling: Allows the operating system to interrupt a running process and assign the CPU to another process. Example: Round Robin.

Non-Preemptive Scheduling: A process runs till completion or voluntarily releases the CPU. Example: First Come First Serve (FCFS).

5. Define all the states of a process and also draw the process state diagram.

Process States:

1. **New**: Process is being created.
2. **Ready**: Process is ready for execution but waiting for CPU.
3. **Running**: Process is being executed by the CPU.
4. **Waiting**: Process is waiting for an event (e.g., I/O completion).
5. **Terminated**: Process has completed execution.

Process State Diagram:

sql

Copy code

New → Ready → Running → Terminated

↓ ↑

Waiting

Illustrates transitions based on events or CPU allocation.

6. Define the types of schedulers.

Types of Schedulers:

1. **Long-Term Scheduler**: Selects processes from the job pool to load into memory for execution.
2. **Short-Term Scheduler**: Allocates CPU to one of the ready processes.
3. **Medium-Term Scheduler**: Temporarily removes processes from memory (swapping) to optimize performance.

7. Define: a. Convoy Effect b. Starvation

a. Convoy Effect: Occurs when a long process holds the CPU, causing shorter processes to wait. It slows system performance, common in First Come First Serve (FCFS) scheduling.

b. Starvation: Happens when low-priority processes are indefinitely delayed because higher-priority processes continuously execute, common in Priority Scheduling.

8. Define Deadlock and write the necessary condition for deadlock.

Deadlock: A state where processes are stuck indefinitely, each waiting for resources held by others.

Necessary Conditions for Deadlock:

1. **Mutual Exclusion**: Only one process can use a resource at a time.
2. **Hold and Wait**: A process holding resources waits for more resources.
3. **No Preemption**: Resources can't be forcibly removed.
4. **Circular Wait**: Processes form a circular chain, each waiting for the next.

9. What is system call? How a system call works?

System Call: A mechanism for programs to request services from the operating system, such as file handling or process management.

How it Works:

1. Program triggers a system call using an interrupt.
2. Control switches to the OS (kernel mode).
3. OS executes the requested service and returns control to the program.

10. Write the various memory management techniques.

Memory Management Techniques:

1. **Contiguous Memory Allocation**: Allocates a single continuous block of memory to processes.

2. **Paged Memory Allocation**: Divides memory into fixed-size pages and stores them non-contiguously.
3. **Segmented Memory Allocation**: Divides memory into variable-sized segments based on logical divisions.
4. **Virtual Memory**: Uses disk space to extend the apparent physical memory.
5. **Swapping**: Moves processes between main memory and secondary storage to optimize memory usage.

SECTION - B

11. Explain Process as a Data Structure.

A **process** in an operating system is treated as a **data structure** that contains information about the execution of a program. It includes:

- **Process Control Block (PCB)**: Stores process-specific details like process ID, program counter, registers, memory management information, and state.
- **Execution State**: Indicates whether the process is ready, running, waiting, or terminated.
- **Resources**: Represents memory, I/O devices, and other system resources assigned to the process.

The operating system manages these data structures to ensure proper scheduling, resource allocation, and synchronization of processes.

12. What is CPU scheduling algorithms? Write down its types.

CPU Scheduling Algorithms are used by the operating system to allocate the CPU to processes efficiently. These algorithms determine the order in which processes are executed, aiming to optimize performance.

Types of CPU Scheduling Algorithms:

1. **First-Come, First-Served (FCFS)**: Executes processes in the order they arrive.
2. **Shortest Job Next (SJN)**: Prioritizes processes with the shortest burst time.
3. **Round Robin (RR)**: Allocates CPU time in fixed time slices to each process.
4. **Priority Scheduling**: Processes are executed based on priority levels.

5. **Multilevel Queue Scheduling**: Divides processes into different queues based on priority or type.

13. An OS uses SJF scheduling algorithm for non- pre-emptive scheduling of process. There are 4 processes with their arrival times 0, 2, 3 and 8 and their burst times are 12, 4, 6 and 5 respectively. Find the average waiting time of the processes if the unit of time used is in milliseconds?

To calculate the **Average Waiting Time** for the processes using the **Shortest Job First (SJF)** algorithm, follow these steps:

Given:

- Process 1: Arrival Time = 0, Burst Time = 12
- Process 2: Arrival Time = 2, Burst Time = 4
- Process 3: Arrival Time = 3, Burst Time = 6
- Process 4: Arrival Time = 8, Burst Time = 5

Steps:

1. **Sort by Arrival Time**:
   P1 (0, 12), P2 (2, 4), P3 (3, 6), P4 (8, 5)
2. **Execution Order**:
   The processes will execute based on burst time, with the shortest burst time selected first, considering the arrival time.
   - P1 (time 0-12),
   - P2 (time 12-16),
   - P4 (time 16-21),
   - P3 (time 21-27)
3. **Waiting Time Calculation**:

- **P1**: Waiting time = 0 ms
- **P2**: Waiting time = 12 - 2 = 10 ms
- **P3**: Waiting time = 21 - 3 = 18 ms
- **P4**: Waiting time = 16 - 8 = 8 ms

4. **Average Waiting Time**:

Average Waiting Time=0+10+18+84=364=9 ms\text{Average Waiting Time} = \frac{0 + 10 + 18 + 8}{4} = \frac{36}{4} = 9 \text{ ms}Average Waiting Time=40+10+18+8=436=9 ms

Final Answer:

The average waiting time is **9 milliseconds**.

14. Explain Critical section and describe solution of critical section problem.

Critical Section:

A **Critical Section** is a part of a program where shared resources (like variables, memory, or devices) are accessed by multiple processes. When multiple processes access shared resources simultaneously, it can lead to data inconsistency or errors. Thus, only one process should be allowed to execute its critical section at a time.

Solution to the Critical Section Problem:

The solution ensures that:

1. **Mutual Exclusion**: Only one process can be in its critical section at any given time.
2. **Progress**: If no process is in the critical section and some processes want to enter, then one of them must be allowed to do so.
3. **Bounded Waiting**: A process must not have to wait indefinitely to enter the critical section.

Common solutions include:

- **Locks/Mutexes**: Ensure mutual exclusion by allowing only one process to access the critical section.
- **Semaphores**: Use binary or counting semaphores to control access.
- **Monitors**: High-level synchronization constructs to manage critical sections.

15. Explain Reader-Writer problem.

Reader-Writer Problem:

The **Reader-Writer problem** is a classic synchronization issue in concurrent programming. It involves a scenario where:

- **Readers**: Processes that only read shared data.
- **Writers**: Processes that modify shared data.

The challenge is to ensure that:

1. **Multiple readers** can access the shared resource simultaneously without conflict.
2. **Writers** must have exclusive access, meaning no readers or other writers can access the resource while a writer is updating it.

Problem Conditions:

- **Readers Priority**: Readers can access the resource concurrently, but if a writer is writing, no readers can access it.
- **Writers Priority**: Writers must wait for readers to finish, but once a writer starts, no readers can access the resource.

Solution:

- **Reader-Writer Locks**: Use locks to allow multiple readers or one writer at a time.
- **Semaphores/Mutexes**: Control access, ensuring writers have exclusive access while readers can share the resource safely.

16. Explain the prevention methods of Deadlock.

Prevention Methods of Deadlock:

Deadlock prevention aims to ensure that the necessary conditions for deadlock are not met. The four necessary conditions are **mutual exclusion**, **hold and wait**, **no preemption**, and **circular wait**. Preventing these conditions can avoid deadlock.

Methods:

1. **Mutual Exclusion**:
    - **Prevention**: Avoid mutual exclusion by allowing resources to be shared if possible, but this is not always feasible for all resources.
2. **Hold and Wait**:
    - **Prevention**: Require processes to request all required resources at once, before execution begins, preventing waiting for additional resources after starting.
3. **No Preemption**:
    - **Prevention**: If a process holding some resources is denied further resources, the resources it holds are preempted and given to another process, allowing the system to avoid circular waits.
4. **Circular Wait**:
    - **Prevention**: Impose a linear ordering of resources, where each process must request resources in a predefined order, eliminating the possibility of circular waiting.

SECTION-C

18. What is Deadlock Detection? Write and explain the Deadlock detection algorithm.

Deadlock Detection:

Deadlock detection is a method used to identify whether a deadlock has occurred in a system. Unlike prevention, detection allows deadlocks to happen but detects them after the fact to take corrective actions.

Deadlock Detection Algorithm:

One common algorithm is the **Resource Allocation Graph (RAG)** method, particularly useful in systems with resources that can be shared between processes.

1. **Resource Allocation Graph**:
    - Nodes represent processes and resources.
    - Directed edges indicate request (from process to resource) or assignment (from resource to process).
2. **Detection Process**:
    - Check for cycles in the graph. If a cycle exists, deadlock is detected because processes in the cycle are mutually waiting for resources that are held by other processes in the same cycle.
3. **Corrective Actions**:
    - If a deadlock is detected, the system may either terminate processes or preempt resources to resolve the deadlock.

This method allows the system to detect and resolve deadlocks but may incur overhead due to cycle detection.

19. Describe the first fit, best fit and worse fit strategies for disk space allocation.

Disk Space Allocation Strategies:

1. **First Fit**:
    - Allocates the first available space large enough to accommodate the requested file.
    - **Pros**: Simple and fast, as it stops searching once a suitable block is found.
    - **Cons**: Can lead to fragmentation as smaller gaps may appear between allocated spaces over time.
2. **Best Fit**:
    - Allocates the smallest available block that is large enough to store the file, mizing wasted space.
    - **Pros**: Reduces wasted space by using the smallest possible block.
    - **Cons**: Requires searching the entire list, which can be slower. This strategy often leads to small leftover gaps, causing external fragmentation.
3. **Worst Fit**:
    - Allocates the largest available block to the file, leaving potentially large remaining free blocks.
    - **Pros**: mizes the risk of very small fragments.
    - **Cons**: Can leave large unused portions, which might not be efficiently utilized, leading to fragmentation over time.

20. What is the purpose of a TLB? Explain the TLB lookup with the help of a block diagram, explaining the hardware required.

Purpose of TLB (Translation Lookaside Buffer):

The **Translation Lookaside Buffer (TLB)** is a small, high-speed memory cache used by the CPU to store recent virtual-to-physical address translations. It speeds up the process of address translation, improving system performance by reducing the need to access slower memory structures like page tables.

TLB Lookup Process:

1. **Address Translation**:
    - When a process accesses memory, the virtual address needs to be translated into a physical address.
    - The **TLB** holds recently used page table entries to avoid accessing the slower main memory.
2. **Lookup Process**:
    - The virtual address is divided into the **page number** and **page offset**.
    - The **page number** is checked against the TLB to see if there's a match.
    - If a match is found (TLB hit), the corresponding physical address is retrieved quickly.
    - If no match is found (TLB miss), the page table is accessed to retrieve the translation and load it into the TLB.

Hardware Required:

- **TLB Cache**: Small, fast memory where recent translations are stored.
- **Page Table**: Large memory structure that stores full address translations.
- **Address Register**: Holds the virtual address for lookup.
- **Comparison Logic**: Checks if the virtual address matches entries in the TLB.

21. Explain various page replacement algorithms. I. Page reference string: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1 How many page fault occur using 3 page frames? II. Page reference string: 3,8,2,3,9,1,6,3,8,9,3,6,2,1,3 How many page fault occur using 5 page frames?

Page Replacement Algorithms:

1. **FIFO (First In First Out)**:
    - Replaces the oldest page in memory when a new page needs to be loaded, based on the order of arrival.
2. **LRU (Least Recently Used)**:
    - Replaces the page that has not been used for the longest time.
3. **Optimal (OPT)**:
    - Replaces the page that will not be used for the longest period in the future.
4. **LFU (Least Frequently Used)**:
    - Replaces the page with the fewest accesses.

I. Page Faults Using 3 Page Frames (FIFO Algorithm)

Page Reference String: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

- **Page Faults Calculation**:
    - Frames: (7), (7, 0), (7, 0, 1), (2, 0, 1), (2, 0, 1), (2, 0, 3), (2, 3, 0), (4, 3, 0), (4, 3, 2), (4, 3, 2), (0, 3, 2), (3, 2, 0), (3, 2, 1), (3, 1, 2)
    - **Total Page Faults**: **15**

II. Page Faults Using 5 Page Frames (FIFO Algorithm)

Page Reference String: 3, 8, 2, 3, 9, 1, 6, 3, 8, 9, 3, 6, 2, 1, 3

- **Page Faults Calculation**:
    - Frames: (3), (3, 8), (3, 8, 2), (3, 8, 2), (3, 8, 9), (3, 8, 9, 1), (3, 8, 9, 1, 6), (3, 8, 9, 1, 6), (8, 9, 1, 6, 3)
    - **Total Page Faults**: **10**

22. Consider a disk with 200 tracks and the queue has random requests from different processes in the order: 55, 58, 39, 18, 90, 160, 150, 38, 184 Initially arm is at 100. Find the Average Seek length using SCAN and C-SCAN algorithm.

Given:

- **Disk Tracks**: 200
- **Queue of Requests**: 55, 58, 39, 18, 90, 160, 150, 38, 184
- **Initial Position**: 100

SCAN Algorithm:

The SCAN algorithm moves the disk arm in one direction, servicing requests along the way. Once it reaches the end, it reverses direction and processes the requests.

1. **Sort the Requests**:
    - Sorted requests: 18, 38, 39, 55, 58, 90, 150, 160, 184
2. **SCAN Movement**:
    - Move from initial position (100) towards the end (200).
    - Serve requests from 100 to 184 (increasing direction), then reverse.

First pass: 100 → 160 → 150 → 90 → 58 → 55 → 39 → 38 → 18

- Reverse at 18, but no further requests are available.

Seek Calculation:

- Initial position: 100
- Seek distance = (160 - 100) + (150 - 160) + (90 - 150) + (58 - 90) + (55 - 58) + (39 - 55) + (38 - 39) + (18 - 38)
- Total seek distance = 60 + 10 + 60 + 32 + 3 + 16 + 1 + 20 = **202**
- **Average Seek Length** = 202 / 9 requests = **22.44 tracks**.

C-SCAN Algorithm:

C-SCAN operates by moving the arm in one direction and then returning to the beginning after reaching the last track, without servicing requests during the return.

1. **Sort the Requests**:
    - Sorted requests: 18, 38, 39, 55, 58, 90, 150, 160, 184
2. **C-SCAN Movement**:
    - Start at 100, move to 184 (right end), then jump back to the beginning (0), and process requests in increasing order.

First pass: 100 → 160 → 150 → 184 → 0 → 18 → 38 → 39 → 55 → 58 → 90

- No reversing in C-SCAN; it wraps back to the start.

Seek Calculation:

- Initial position: 100
- Seek distance = (160 - 100) + (150 - 160) + (184 - 150) + (184 - 0) + (18 - 0) + (39 - 18) + (38 - 39) + (55 - 39) + (58 - 55) + (90 - 58)
- Total seek distance = 60 + 10 + 34 + 184 + 18 + 21 + 1 + 16 + 3 + 32 = **379**
- **Average Seek Length** = 379 / 9 requests = **42.11 tracks**.

Summary:

- **SCAN Average Seek Length**: **22.44 tracks**
- **C-SCAN Average Seek Length**: **42.11 tracks**