USN | | | | | | | | | |

Internal Assessment Test 2 – August 2023

Sub:	**Operating Systems**				Sub Code:	21CS44	Branch:	ISE		
Date:	**10/08/2023**	Duration:	90 min's	Max Marks: 50	Sem/Sec:	IV A, B & C			OBE	
	Answer any FIVE FULL Questions							MARKS	CO	RBT
1	Explain Semaphores. What do you mean by binary semaphore and counting semaphore?							10	CO2	L2

- **Binary Semaphore (Mutex)**: Ensures that only one thread or process can access a shared resource at a time, preventing conflicts and ensuring data integrity.
- **Counting Semaphore**: Limits the number of threads or processes that can access a resource simultaneously, controlling concurrency and preventing overuse of the resource.

		MARKS	CO	RBT
2	What are deadlocks and its characteristics with example? Explain the necessary conditions for its occurrence.	10	CO2	L2

Deadlock is a situation in concurrent programming where two or more processes or threads are unable to proceed because each is waiting for the other to release a resource. This results in a standstill, where none of the processes can continue execution, and the system becomes unresponsive.

Characteristics of Deadlock:

1. **Mutual Exclusion**: At least one resource must be held in a non-sharable mode, meaning only one process can use it at a time.
2. **Hold and Wait**: A process must hold at least one resource while waiting to acquire additional resources held by other processes.
3. **No Preemption**: Resources cannot be forcibly taken away from a process; they can only be released voluntarily by the process holding them.
4. **Circular Wait**: A circular chain of two or more processes exists, where each process is waiting for a resource held by the next process in the chain.

Example: Consider a scenario with two processes, P1 and P2, and two resources, R1 and R2. The processes require both resources to complete their tasks.

- P1 acquires R1.
- P2 acquires R2.
- P1 requests R2 but is blocked since P2 is holding it.
- P2 requests R1 but is blocked since P1 is holding it.

Now, P1 and P2 are stuck in a deadlock. Both processes are waiting for resources held by the other process, and neither can proceed.

Necessary Conditions for Deadlock:

1. **Mutual Exclusion**: At least one resource must be held in a non-sharable mode, preventing multiple processes from using it simultaneously.
2. **Hold and Wait**: Processes must hold resources while waiting to acquire additional resources. This means they can enter a state of partial execution while holding some resources and waiting for others.
3. **No Preemption**: Resources cannot be forcibly taken away from processes; they must be released voluntarily. This condition ensures that a process cannot be interrupted while holding resources.

	4. **Circular Wait**: A circular chain of processes must exist, where each process is waiting for a resource held by another process in the chain. These four conditions together create a scenario where processes can become deadlocked, as all the conditions are satisfied simultaneously. To prevent deadlocks, one or more of these conditions must not be true. For example, using techniques like resource allocation graphs, process prioritization, timeouts, and preemption can help avoid or resolve deadlocks.			
3	What is Resource Allocation Graph (RAG)? Explain how RAG is very useful in describing deadly embrace (deadlock) by considering your own example. A **Resource Allocation Graph (RAG)** is a graphical representation used to depict the resource allocation and resource request relationships among processes in a system. It is particularly useful in visualizing the interactions between processes and resources and identifying potential deadlocks. In a RAG, processes are represented as nodes, and resources are represented as resource types connected by edges. There are two types of edges: 1. **Allocation Edge**: Represents a process holding a resource. It is directed from the process to the resource. 2. **Request Edge**: Represents a process requesting a resource. It is directed from the process to the resource. A deadlock can be identified in a RAG when it contains a cycle involving only request edges. This cycle indicates that each process in the cycle is waiting for a resource held by another process in the cycle, leading to a situation where none of the processes can proceed, resulting in a deadlock. **Example**: Let's consider a scenario with three processes (P1, P2, P3) and three resource types (R1, R2, R3). Each process can hold at most one instance of each resource type. The following steps describe the scenario: 1. P1 holds R1. 2. P2 holds R2. 3. P3 holds R3. 4. P1 requests R2 (request edge P1 -> R2). 5. P2 requests R3 (request edge P2 -> R3). 6. P3 requests R1 (request edge P3 -> R1). At this point, the RAG will have a cycle involving the request edges: P1 -> R2 -> P2 -> R3 -> P3 -> R1 -> P1. This cycle indicates a potential deadlock situation. Each process is waiting for a resource held by the next process in the cycle, forming a "deadly embrace." The RAG is very useful in describing the deadly embrace (deadlock) because: 1. **Visualization**: The RAG provides a clear visual representation of the interactions between processes and resources, making it easier to identify patterns that could lead to deadlocks. 2. **Cycle Detection**: By observing cycles involving only request edges, you can quickly identify potential deadlocks in the system. 3. **Identification of Resources**: It helps identify which processes are waiting for which resources, aiding in the analysis of potential deadlock scenarios. 4. **Resolution Strategies**: RAGs can guide the application of deadlock prevention.	10	CO2	L2
4a)	Explain the process of recovery from deadlock.	5	CO2	L2

	Recovery from a deadlock is a process of resolving the deadlock situation and allowing the affected processes to continue their execution. There are several strategies and techniques that can be employed to recover from a deadlock. Here are some common approaches: 1. **Process Termination**: One way to recover from a deadlock is to terminate one or more processes involved in the deadlock. This releases the resources held by the terminated processes, allowing other processes to continue. However, this approach should be used with caution as it might lead to loss of data or disruption of user activities. 2. **Resource Preemption**: In this approach, the operating system forcibly takes resources from one or more processes and gives them to other waiting processes to break the circular wait condition. The preempted processes are rolled back to a safe state, allowing them to restart later. This method requires careful consideration to choose which processes to preempt and a mechanism to roll back their state. 3. **Wait-Die and Wound-Wait Schemes**: These schemes are used in database management systems to handle deadlock situations. In the Wait-Die scheme, older processes wait for younger ones, while in the Wound-Wait scheme, older processes preempt resources from younger ones. These schemes are useful when dealing with transactions in a database environment. 4. **Timeouts**: Processes waiting for resources can be given a certain time limit (timeout). If a process doesn't acquire the required resources within the given time, it releases the resources it holds and goes back to its initial state, allowing other processes to use those resources. 5. **Dynamic Resource Allocation**: This involves allocating resources dynamically to processes as they request them. If a resource allocation leads to a potential deadlock, the system can backtrack and reallocate resources to avoid the deadlock situation. 6. **Process Restart**: In distributed systems, processes can be restarted on different nodes to resolve deadlocks. The idea is to break the circular wait by relocating processes to nodes with available resources. It's important to note that the recovery process should be well-designed and carefully implemented to avoid introducing further issues or complications. Additionally, it's often better to focus on preventing deadlocks from occurring in the first place through careful resource allocation, managing resource dependencies, and using appropriate synchronization mechanisms.			
4b)	Explain three requirements that a solution to critical – section problem must satisfy. The critical section problem is a fundamental synchronization problem in concurrent programming, where multiple processes or threads share a common resource and need to access it in a mutually exclusive manner. A solution to the critical section problem must satisfy the following three requirements to ensure correct and safe execution: 1. **Mutual Exclusion**: The solution must ensure that only one process or thread at a time can be in its critical section, accessing the shared resource. This requirement guarantees that no two processes are executing their critical sections simultaneously, preventing conflicts and ensuring data integrity. 2. **Progress**: The solution must ensure that if no process is in its critical section and some processes are waiting to enter their critical sections, then only those processes that are not in their remainder sections (non-critical sections) can participate in selecting the next process to enter the critical section. In other words, the solution must ensure that processes waiting to enter the critical section do eventually get a chance to do so.	5	CO2	L2

3. **Bounded Waiting**: The solution must ensure that there is a limit on the number of times a process can enter its critical section after another process has requested access to it. This limit prevents any particular process from being indefinitely starved and guarantees that processes waiting to enter the critical section are not continually bypassed by new requests.

These three requirements, mutual exclusion, progress, and bounded waiting, are collectively known as the "three conditions for synchronization" and are essential to ensure that a solution to the critical section problem is correct and prevents potential issues such as race conditions, deadlocks, and starvation.

---

5 | Consider the following snapshot of a system: | 10 | CO3 | L3

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 0 2	0 0 4	1 0 2
P_1	1 0 0	2 0 1	
P_2	1 3 5	1 3 7	
P_3	6 3 2	8 4 2	
P_4	1 4 3	1 5 7	

Answer the following questions using Banker's Algorithm.

i. What is the content of the matrix need?
ii. Write the safe sequence.
iii. Is the system in a safe state?

i. Calculate the Need matrix (N): Need (N) = Max (M) - Allocation (A)

0 0 2
1 0 1
0 0 2
2 1 0
0 1 4

ii. Finding the Safe Sequence: To determine the safe sequence, we'll use the Banker's Algorithm:

1. Find a process whose Need can be satisfied with the Available resources. Once a process is completed, its resources are returned to Available.
2. Repeat the process until all processes are completed or it's not possible to find a process that can be completed.

Starting with the given Available resources **[1 0 2]**, we can follow these steps:

1. At t=0, we can allocate resources to Process 1 (**P1**), leaving Available = **[1 0 2] - [0 0 2] = [1 0 0]**.
2. At t=1, Process 4 (**P4**) can be allocated, leaving Available = **[1 0 0] + [6 3 2] = [7 3 2]**.
3. At t=2, Process 0 (**P0**) can be allocated, leaving Available = **[7 3 2] + [0 0 2] = [7 3 4]**.
4. At t=3, Process 2 (**P2**) can be allocated, leaving Available = **[7 3 4] + [1 3 5] = [8 6 9]**.
5. At t=4, Process 3 (**P3**) can be allocated, leaving Available = **[8 6 9] + [1 4 3] = [9 10 12]**.
6. Finally, at t=5, Process 5 (**P5**) can be allocated.

The safe sequence: **P1 -> P4 -> P0 -> P2 -> P3 -> P5**

iii. System Safety: Since we were able to find a safe sequence, the system is in a safe state.

In conclusion, based on the provided matrices and calculations, we've determined the content of the matrix "Need," found the safe sequence, and determined that the system is in a safe state according to the Banker's Algorithm.

---

6 | | 10 | CO2 | L3

Process	Arrival Time	Burst Time

For the following set of process find the avg. waiting time and avg. turn around using Gantt chart for a) FCFS
b) SJF (preemptive and non- preemptive)
c) RR (quantum= 4)

P1	0	4
P2	1	2
P3	2	5
P4	3	4

SOLUTION……………………….

To calculate the average waiting time and average turnaround time for different scheduling algorithms (FCFS, SJF preemptive and non-preemptive, and RR), we can simulate the execution of the processes using a Gantt chart and then calculate the times accordingly. Below, I'll provide the calculations and Gantt charts for each case.

Process details: Process P1 P2 P3 P4 Arrival Time 0 1 2 3 Burst Time 4 2 5 4

Let's start with FCFS (First-Come-First-Serve) scheduling:

1. **FCFS Scheduling:**
    - Gantt Chart: P1 -> P2 -> P3 -> P4
    - Waiting Times: P1 = 0, P2 = 4, P3 = 6, P4 = 11
    - Turnaround Times: P1 = 4, P2 = 6, P3 = 11, P4 = 15

Average Waiting Time = (0 + 4 + 6 + 11) / 4 = 5.25 Average Turnaround Time = (4 + 6 + 11 + 15) / 4 = 9

Now, let's move to SJF (Shortest Job First) scheduling:

2. **SJF Preemptive Scheduling:**
    - Gantt Chart: P1 -> P2 -> P1 -> P4 -> P2 -> P3 -> P3 -> P3 -> P3
    - Waiting Times: P1 = 6, P2 = 3, P3 = 0, P4 = 6
    - Turnaround Times: P1 = 10, P2 = 5, P3 = 11, P4 = 10

Average Waiting Time = (6 + 3 + 0 + 6) / 4 = 3.75 Average Turnaround Time = (10 + 5 + 11 + 10) / 4 = 9

3. **SJF Non-Preemptive Scheduling:**
    - Gantt Chart: P1 -> P2 -> P4 -> P3
    - Waiting Times: P1 = 6, P2 = 0, P3 = 6, P4 = 3
    - Turnaround Times: P1 = 10, P2 = 3, P3 = 11, P4 = 7

Average Waiting Time = (6 + 0 + 6 + 3) / 4 = 3.75 Average Turnaround Time = (10 + 3 + 11 + 7) / 4 = 7.75

4. **RR (Round Robin) Scheduling (Quantum = 4):**
    - Gantt Chart: P1 -> P2 -> P3 -> P4 -> P1 -> P3 -> P4 -> P3 -> P3
    - Waiting Times: P1 = 5, P2 = 2, P3 = 5, P4 = 8
    - Turnaround Times: P1 = 9, P2 = 7, P3 = 14, P4 = 12

Average Waiting Time = (5 + 2 + 5 + 8) / 4 = 5 Average Turnaround Time = (9 + 7 + 14 + 12) / 4 = 10.5

These are the calculations and Gantt charts for each scheduling algorithm

Faculty Signature CCI Signature HOD Signature