Unite Test – II

Name: - Hansraj Mahadev Pawar

Class: - TE11 **Roll No: -** 33360

Qu. 1-a)	What is the semaphore and what are the different types of it, explain with syntax?

Ans:-	A semaphore is a synchronization mechanism used in concurrent programming to control access to shared resources by multiple threads or processes. Semaphores are often used to prevent race conditions and coordinate the execution of threads or processes.
	
	There are two main types of semaphores:
	
	1. Binary Semaphore:
	
	A binary semaphore can only have two values: 0 and 1. It is typically used to control access to a single resource, where 0 indicates that the resource is unavailable (locked), and 1 indicates that the resource is available (unlocked). It is also known as a mutex (mutual exclusion).
	
	Syntax:-
	
	// Initialization: Initialize to 1 to indicate the resource is available.
	semaphore_t semaphore ;
	initialize_semaphore(&semaphore, 1);
	
	// Locking (acquiring the resource)
	wait(&semaphore); // If the semaphore value is 1, decrement it to 0 and proceed; if it's 0, block the thread.
	
	// Unlocking (releasing the resource)
	signal(&semaphore); // Increment the semaphore value from 0 to 1, allowing other threads to access the resource.
	
	2) Counting Semaphore:
	
	A counting semaphore can have a value greater than 1, and it is used to control access to a pool of identical resources. It allows you to limit the number of threads that can access the resource simultaneously.
	
	Syntax:-
	
	// Initialization: Initialize with the maximum number of resources available.
	semaphore_t semaphore;
	initialize_semaphore(&semaphore, max_resources);

```
// Locking (acquiring a resource)
wait(&semaphore);  // If the semaphore value is greater than 0, decrement it and proceed; if it's 0, block the thread.

// Unlocking (releasing a resource)
signal(&semaphore); // Increment the semaphore value, allowing another thread to access a resource.

// You can also wait with a timeout and check if the semaphore was acquired or not.
if (wait_with_timeout(&semaphore, timeout)) {
   // Resource acquired
} else {
   // Resource not acquired within the specified timeout
}
```

Qu. 1-b)	What is the dead lock ? explain the four conditions for a deadlock to occur?

A deadlock is a situation in concurrent computing where two or more threads or processes are unable to proceed because they are each waiting for a resource that is held by another, resulting in a standstill. Deadlocks can significantly impact the efficiency and reliability of a computer system. To occur, a deadlock generally requires four necessary conditions, which are often referred to as the "Four Conditions for a Deadlock."

1. Mutual Exclusion: This condition states that at least one resource must be held in a mutually exclusive manner, meaning that only one thread or process can use the resource at any given time. If multiple threads/processes can access the resource simultaneously, a deadlock cannot occur due to this condition. For example, a printer can be a mutually exclusive resource because only one job can be printed at a time.

2. Hold and Wait (Resource Holding):This condition asserts that a thread or process must be holding at least one resource while waiting to acquire another resource. In other words, a process cannot request a resource if it already holds resources and is waiting for additional ones. This condition ensures that resources are not immediately released after acquisition and are held for some time, which can lead to potential deadlocks.

3. No Pre-emption : Pre-emption means forcibly taking a resource away from a thread or process that holds it. The third condition stipulates that resources cannot be preempted. In other words, if a thread or process is currently holding a resource and requests another resource that is not available, it cannot be forced to release the resource it already holds. This condition helps create the potential for deadlock, as threads cannot be forced to release resources that might break the deadlock.

4. Circular Wait: Circular wait occurs when a set of two or more threads or processes are each waiting for a resource held by another thread or process in a circular chain. For example, Thread A is waiting for a resource held by Thread B, which is waiting for a resource held by Thread C, and so on, until a thread in the chain is waiting for a resource held by Thread A. This circular dependency among threads or processes creates the conditions for a deadlock.

A deadlock can only occur when all four of these conditions are present simultaneously. If any of these conditions are missing, a deadlock cannot happen. The prevention and resolution of deadlocks involve various strategies, such as resource allocation graphs, deadlock detection and recovery, and ensuring

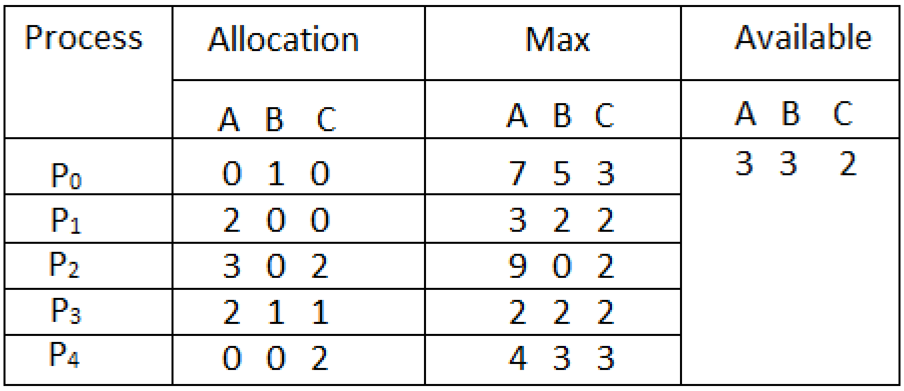
that one or more of the four conditions are not satisfied. Properly managing resources and their allocation is essential to minimize the risk of deadlocks in concurrent systems.
.

Qu. 1-c) Explain the resource Allocation denial approach with suitable examples by evaluating safe and unsafe states for at least one sequence of the processes involved for the resource allocation.

Ans: - The Resource Allocation Denial Approach, often referred to as the Banker's Algorithm, is a resource management and deadlock avoidance technique used in operating systems to ensure the safe allocation of resources to processes. The approach allows the system to determine if a resource request from a process can be granted without leading to a deadlock. The system maintains a data structure called the "claim matrix" or "maximum matrix" to represent the maximum needs of each process and the "allocation matrix" to keep track of the currently allocated resources. The system also maintains the "available vector," which tracks the available resources at any given time.

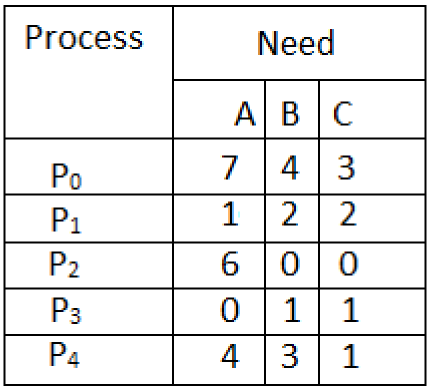
example:

Considering a system with five processes P0 through P4 and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t0 following snapshot of the system has been taken:



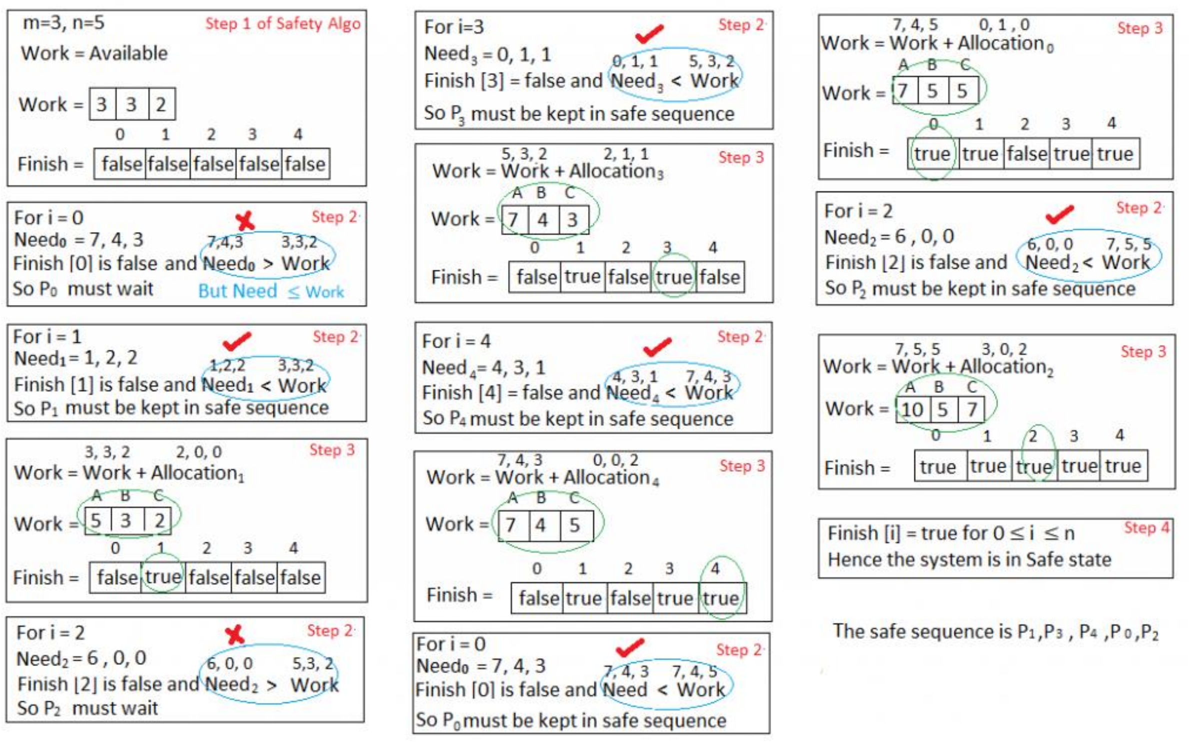
Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

What will be the content of the Need matrix?
Need [i, j] = Max [i, j] – Allocation [i, j]
So, the content of Need Matrix is:



Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Is the system in a safe state? If yes, then what is the safe sequence?
Applying the Safety algorithm on the given system,



m=3, n=5 — Step 1 of Safety Algo
Work = Available
Work = 3 | 3 | 2

	0	1	2	3	4
Finish =	false	false	false	false	false

For i = 0 — Step 2
$Need_0$ = 7, 4, 3
Finish [0] is false and $Need_0$ > Work (7,4,3 > 3,3,2) ✗
So P_0 must wait — But Need ≤ Work

For i = 1 — Step 2
$Need_1$ = 1, 2, 2
Finish [1] is false and $Need_1$ < Work (1,2,2 < 3,3,2) ✓
So P_1 must be kept in safe sequence

Work = Work + $Allocation_1$ (3, 3, 2 + 2, 0, 0) — Step 3
Work = 5 | 3 | 2 (A B C)

	0	1	2	3	4
Finish =	false	true	false	false	false

For i = 2 — Step 2
$Need_2$ = 6 , 0, 0
Finish [2] is false and $Need_2$ > Work (6, 0, 0 > 5,3, 2) ✗
So P_2 must wait

For i=3 — Step 2
$Need_3$ = 0, 1, 1
Finish [3] = false and $Need_3$ < Work (0, 1, 1 < 5, 3, 2) ✓
So P_3 must be kept in safe sequence

Work = Work + $Allocation_3$ (5, 3, 2 + 2, 1, 1) — Step 3
Work = 7 | 4 | 3 (A B C)

	0	1	2	3	4
Finish =	false	true	false	true	false

For i = 4 — Step 2
$Need_4$ = 4, 3, 1
Finish [4] = false and $Need_4$ < Work (4, 3, 1 < 7, 4, 3) ✓
So P_4 must be kept in safe sequence

Work = Work + $Allocation_4$ (7, 4, 3 + 0, 0, 2) — Step 3
Work = 7 | 4 | 5 (A B C)

	0	1	2	3	4
Finish =	false	true	false	true	true

For i = 0 — Step 2
$Need_0$ = 7, 4, 3
Finish [0] is false and Need < Work (7, 4, 3 < 7, 4, 5) ✓
So P_0 must be kept in safe sequence

Work = Work + $Allocation_0$ (7, 4, 5 + 0, 1 , 0) — Step 3
Work = 7 | 5 | 5 (A B C)

	0	1	2	3	4
Finish =	true	true	false	true	true

For i = 2 — Step 2
$Need_2$ = 6 , 0, 0
Finish [2] is false and $Need_2$ < Work (6, 0, 0 < 7, 5, 5) ✓
So P_2 must be kept in safe sequence

Work = Work + $Allocation_2$ (7, 5, 5 + 3, 0, 2) — Step 3
Work = 10 | 5 | 7 (A B C)

	0	1	2	3	4
Finish =	true	true	true	true	true

Finish [i] = true for 0 ≤ i ≤ n — Step 4
Hence the system is in Safe state

The safe sequence is P_1, P_3 , P_4 , P_0 , P_2

What will happen if process P1 requests one additional instance of resource type A and two instances of resource type C?

$$\begin{array}{l} \quad\quad\quad\quad\ A\ \ B\ \ C \\ Request_1 = 1, 0, 2 \end{array}$$

To decide whether the request is granted we use Resource Request algorithm

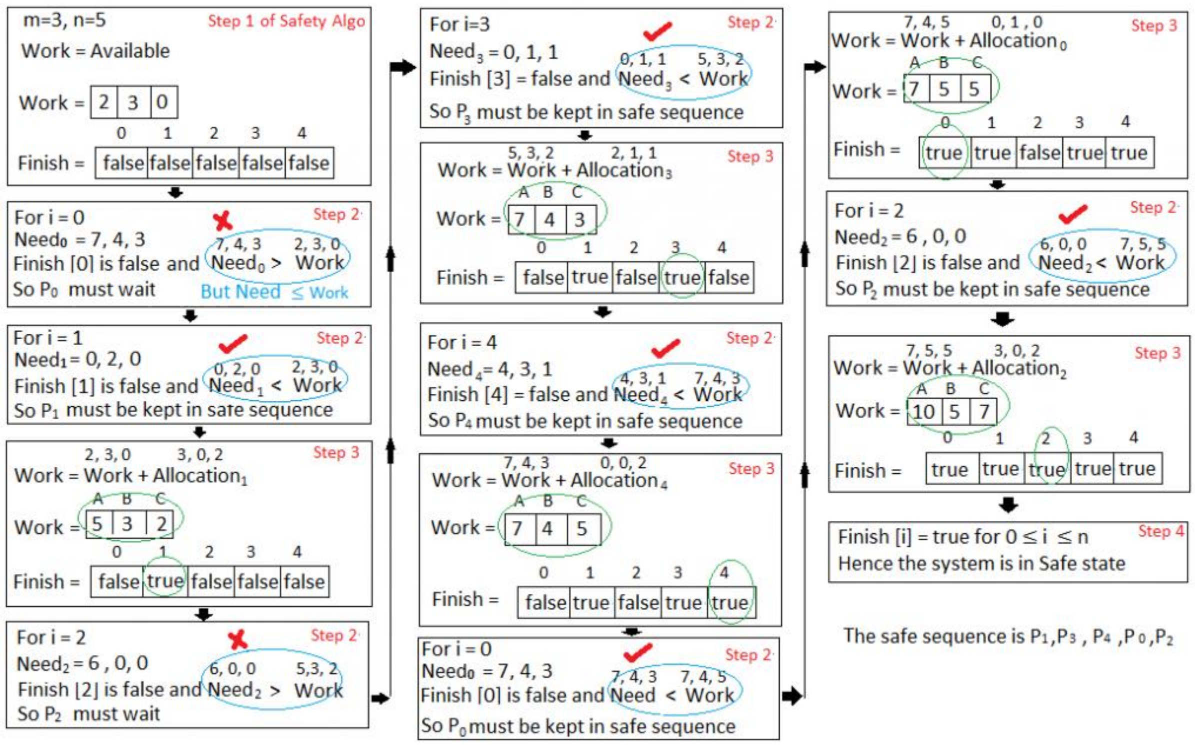
Step 1 ✓
$Request_1 < Need_1$ (1, 0, 2 < 1, 2, 2)

Step 2 ✓
$Request_1 < Available$ (1, 0, 2 < 3, 3, 2)

Step 3
Available = Available – $Request_1$
$Allocation_1$ = $Allocation_1$ + $Request_1$
$Need_1$ = $Need_1$ - $Request_1$

Process	Allocation	Need	Available
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

We must determine whether this new system state is safe. To do so, we again execute Safety algorithm on the above data structures.


m=3, n=5
Step 1 of Safety Algo
Work = Available
Work = 2 3 0
Finish = false false false false false
For i = 0
Need₀ = 7, 4, 3
Finish [0] is false and Need₀ > Work
So P₀ must wait
But Need ≤ Work
For i = 1
Need₁ = 0, 2, 0
Finish [1] is false and Need₁ < Work
So P₁ must be kept in safe sequence
Work = Work + Allocation₁
Work = 5 3 2
Finish = false true false false false
For i = 2
Need₂ = 6 , 0, 0
Finish [2] is false and Need₂ > Work
So P₂ must wait
For i=3
Need₃ = 0, 1, 1
Finish [3] = false and Need₃ < Work
So P₃ must be kept in safe sequence
Work = Work + Allocation₃
Work = 7 4 3
Finish = false true false true false
For i = 4
Need₄ = 4, 3, 1
Finish [4] = false and Need₄ < Work
So P₄ must be kept in safe sequence
Work = Work + Allocation₄
Work = 7 4 5
Finish = false true false true true
For i = 0
Need₀ = 7, 4, 3
Finish [0] is false and Need < Work
So P₀ must be kept in safe sequence
Work = Work + Allocation₀
Work = 7 5 5
Finish = true true false true true
For i = 2
Need₂ = 6 , 0, 0
Finish [2] is false and Need₂ < Work
So P₂ must be kept in safe sequence
Work = Work + Allocation₂
Work = 10 5 7
Finish = true true true true true
Finish [i] = true for 0 ≤ i ≤ n
Hence the system is in Safe state
The safe sequence is P₁,P₃ , P₄ ,P₀ ,P₂

Hence the new system state is safe, so we can immediately grant the request for process P1 .
Code for Banker's Algorithm

Qu.2-a)

Consider the following page reference using three frames that are initially empty. find the page faults using LRU Algorithm, where the page reference sequence is : 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7.0,1

To find the page faults using the Least Recently Used (LRU) page replacement algorithm, we'll simulate the process step by step for the given page reference sequence:

Page Reference Sequence: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
Number of Frames (F): 3 (as there are three frames initially empty)

We'll use a table to keep track of the page frames and their usage order. Initially, all frames are empty:

Page Reference	7	0	1	Page Fault?	Page Frames
7	7			Yes	7
0	0			Yes	7, 0
1	1			Yes	7, 0, 1
2	2			Yes	0, 1, 2
0	2	0		Yes	1, 2, 0
3	2	3		Yes	0, 3, 2
0	0	3		Yes	0, 3, 2
4	4	0		Yes	4, 0, 3
2	2	4		Yes	2, 4, 0

3	3	2		Yes	3, 2, 0
0	0	3		Yes	0, 3, 2
3	0	3		No	0, 3, 2
2	0	2		Yes	2, 0, 3
1	1	0		Yes	2, 0, 1
2	2	0		Yes	2, 0, 1
0	0	2		Yes	0, 2, 1
1	1	0		Yes	0, 2, 1
7	7	0		Yes	7, 0, 2
0	0	7		Yes	0, 7, 2
1	1	0		Yes	1, 0, 7

Count of page faults: 17

So, using the LRU algorithm with three frames, the number of page faults for the given page reference sequence is 17.

Qu.2-b) What is difference between segmentation and fragmentation?

Fragmentation	Segmentation
In this, storage space is used inefficiently that in turn reduce capacity and performance.	In this, memory is divided into variable size parts usually known as segments.
Types of fragmentation includes internal and external fragmentation.	Types of segmentation includes virtual memory and simple segmentation.
Its main purpose is to help operating system use the available space on storage device.	Its main purpose is to give user's view of process.
It reduces efficiency in memory management.	It simply allows for better efficiency in memory management.
In this, memory blocks are not used i.e., it remains unused.	It usually works a memory management technique to execute processes.
It is generally associated with IP.	It is generally associated with TCP.
It is an unwanted problem that causes wastage of memory and inflexibility.	Its advantages include less overhead, larger segment size than actual page size, no internal fragmentation, etc.

Qu.2-c) Describe first fit, best fit ,worst fit and next fit placement strategies , Also discuss their suitability along with the suitable example .

Memory allocation strategies are used to allocate memory blocks to processes in a computer system. Four common memory allocation strategies are:

First Fit:

Description: In the first-fit memory allocation strategy, the first available memory block that is large enough to accommodate the process is allocated. It starts searching from the beginning of the available memory space and selects the first block that satisfies the process's requirements.
Suitability: First fit is simple and efficient in terms of time complexity. It can be suitable for systems with a varying number of processes and memory blocks. However, it may lead to fragmentation issues over time.

Example: Suppose there are memory blocks of sizes [100KB, 200KB, 50KB, 300KB, 150KB] and processes with sizes [120KB, 70KB, 200KB, 80KB]. First fit would allocate the 120KB process to the 200KB memory block, the 70KB process to the 100KB memory block, and so on.
Best Fit:

Description: The best-fit strategy allocates the smallest available memory block that is just large enough to accommodate the process. It searches through all available blocks to find the one that best fits the process.
Suitability: Best fit minimizes memory wastage but can be less efficient in terms of time complexity because it requires searching for the best fit. It's suitable for systems where memory fragmentation is a concern.
Example: Using the same memory blocks and process sizes as in the first fit example, best fit would allocate the 120KB process to the 150KB memory block, the 70KB process to the 100KB memory block, and so on.
Worst Fit:

Description: The worst-fit strategy allocates the largest available memory block to the process. It aims to leave the largest possible holes in memory for future allocations.
Suitability: Worst fit can lead to more memory fragmentation but is suitable when you want to maximize the utilization of large memory blocks. It's less commonly used than first fit or best fit.
Example: Using the same memory blocks and process sizes as in the previous examples, worst fit would allocate the 120KB process to the 300KB memory block, the 70KB process to the 200KB memory block, and so on.
Next Fit:

Description: Next fit is similar to first fit but remembers where it left off in the memory allocation process. It starts searching for the next available block from where it previously allocated memory, which reduces external fragmentation compared to first fit.
Suitability: Next fit is suitable for systems where there is a high chance of processes arriving in a sequential order, as it reduces fragmentation. It's more efficient than worst fit but may not be as efficient as best fit.
Example: Using the same memory blocks and process sizes as in the first fit example, next fit would allocate the 120KB process to the 200KB memory block, the 70KB process to the 150KB memory block, and so on.
The suitability of each allocation strategy depends on the specific requirements of the system. First fit is simple and often efficient, while best fit minimizes memory wastage. Worst fit maximizes large memory block utilization, and next fit is useful when processes arrive sequentially. The choice of strategy can impact memory utilization, fragmentation, and system performance.