Q.1 Explain Semaphores?

Ans

<u>**Semaphores:**</u>

- The solutions of the critical section problem represented in the section is not easy to generalize to more complex problems.
- To overcome this difficulty, we can use a synchronization tool call a semaphore.
- A semaphore S is an integer variable that, a part from initialization, is a accessed two standard atomic operations: wait and signal. This operations were originally termed P (for wait;from the Dutch proberen, to test) and V (for signal ; from verhogen, to increment).

The Classical definition of wait and signal are:

```
Wait (S)
{
while (S <=0)
S =S – 1;
}
signal(S)
{
S = S + 1;
}
```

- The integer value of the semaphore in the wait and signal operations must be executed indivisibly.
- That is, when one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value.
- In addition, in the case of the wait(S), the testing of the integer value of S (S 0), and its possible modification (S := S – 1), must also be executed without interruption.

Semaphores are not provided by hardware. But they have several attractive properties:

1. Semaphores are machine independent.
2. Semaphores are simple to implement.
3. Correctness is easy to determine.
4. Can have many different critical sections with different semaphores.
5. Semaphore acquire many resources simultaneously

Usage

1- Semaphores can be used to deal with n process critical section problem.
3- Semaphores can also be used to solve various synchronization problems

Drawback of Semaphore

1. They are essentially shared global variables.
2. Access to semaphores can come from anywhere in a program.

	3. There is no control or guarantee of proper usage. 4. There is no linguistic connection between the semaphore and the data to which the semaphore controls access. 5. They serve two purposes, mutual exclusion and scheduling constraints.
Q2.	Explain Monitors?

Ans	**Monitors:**
	
	• The monitor is a programming-language construct that provides equivalent functionality to that of semaphores and that is easier to control.
	• The monitor construct has been implemented in a number of programming languages, including Concurrent Pascal, Pascal-Plus, Modula-2, Modula-3, and Java.
	• It has also been implemented as a program library. This allows programmers to put monitor locks on any object.
	
	A monitor supports synchronization by the use of condition variables that are contained Within the monitor and accessible only within the monitor. Condition variables are a special data type in monitors, which are operated on by two functions:
	
	• cwait (c): Suspend execution of the calling process on condition c. The monitor is now available for use by another process.
	• csignal (c): Resume execution of some process blocked after a cwait on the same condition)lf there are several such processes, choose one of them; if there is no such process, do nothing.
	
	Syntax of a monitor:
	
	monitor monitor name
	{
	/* shared variable declarations */
	function P1 (. . .) {
	. . .
	}
	function P2 (. . .) {
	. . .
	}
	.
	.
	.
	function Pn (. . .) {
	. . .
	}
	initialization code (. . .) {
	. . .
	}
	}

Monitor Usage:

a. **Schematic view of monitor:**

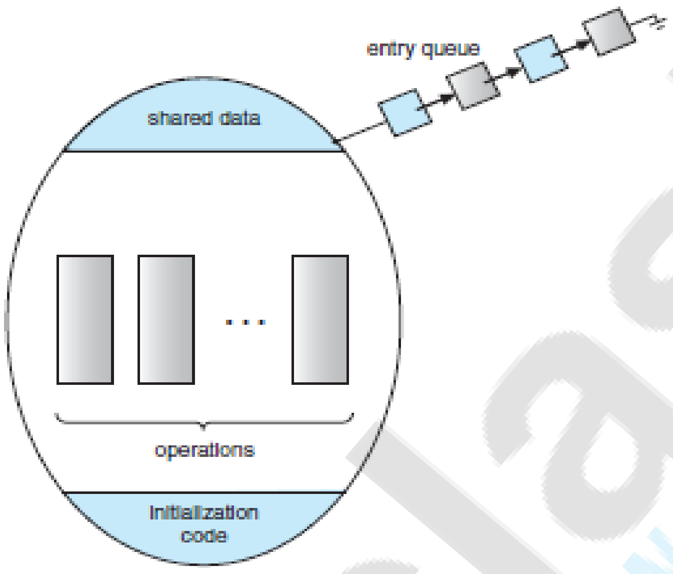


Figure 5.16 Schematic view of a monitor.

A programmer who needs to write a tailor-made synchronization scheme can define one or more variables of type condition:

condition x, y;

The only operations that can be invoked on a condition variable are wait() and signal(). The operation

x.wait();

means that the process invoking this operation is suspended until another process invokes

x.signal();

b. Monitor with condition variable:

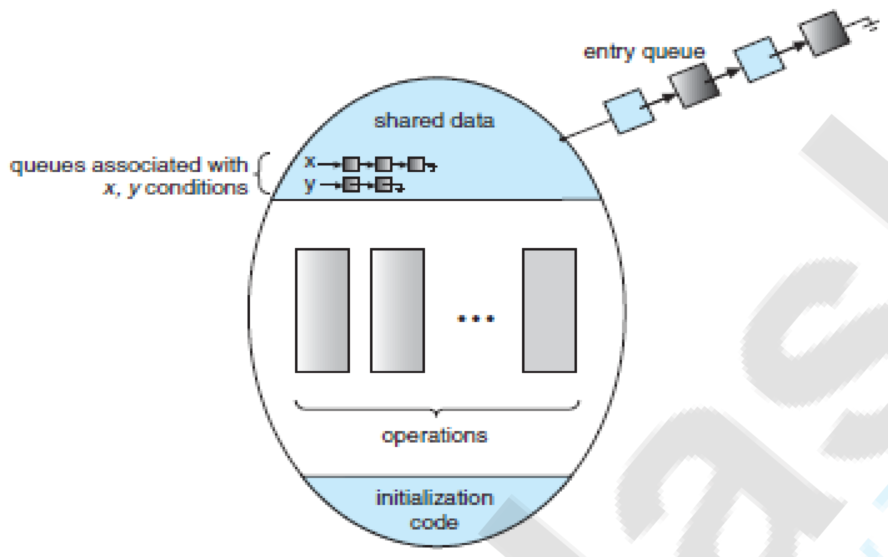


Figure 5.17 Monitor with condition variables.

Note, however, that conceptually both processes can continue
with their execution. Two possibilities exist:
1. Signal and wait. P either waits until Q leaves the monitor or waits for
another condition.
2. Signal and continue. Q either waits until P leaves the monitor or waits
for another condition.

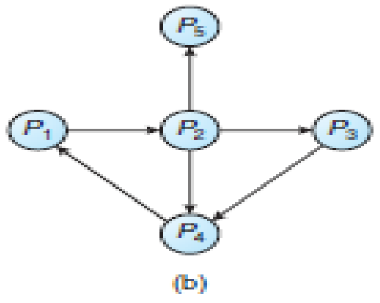
Q.3 What is Deadlock Detection.

Ans. **Deadlock Detection:**

- If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may occur.
- In this environment Operating System 27 An algorithm that examines the state of the system to determine whether a deadlock has occurred An algorithm to recover from the deadlock

Single Instance of Each Resource Type

- If all resources have only a single instance, then we can define a deadlock detection algorithm that uses a variant of the resource-allocation graph, called a wait-for graph.
- We obtain this graph from the resource-allocation graph by removing the nodes of type resource and collapsing the appropriate edges.



(b) Corresponding wait-for graph.

Several Instances of a Resource Type

The wait-for graph scheme is not applicable to a resource-allocation system with multiple instances of each resource type.
The data structures used are: Available Allocation Request

The algorithm used are :
• **Available:** A vector of length m indicates the number of available resources of each type.
• **Allocation:** An n x m matrix defines the number of resources of each type currently allocated to each process.
• **Request:** An n x m matrix indicates the current request of each process. If Request [i, j] = k, then process P, is requesting k more instances of resource type Rj.

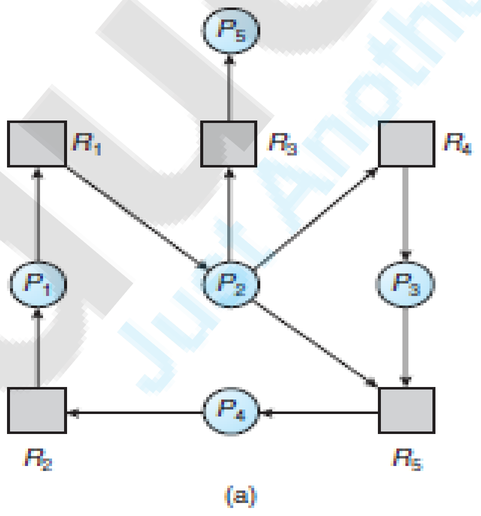
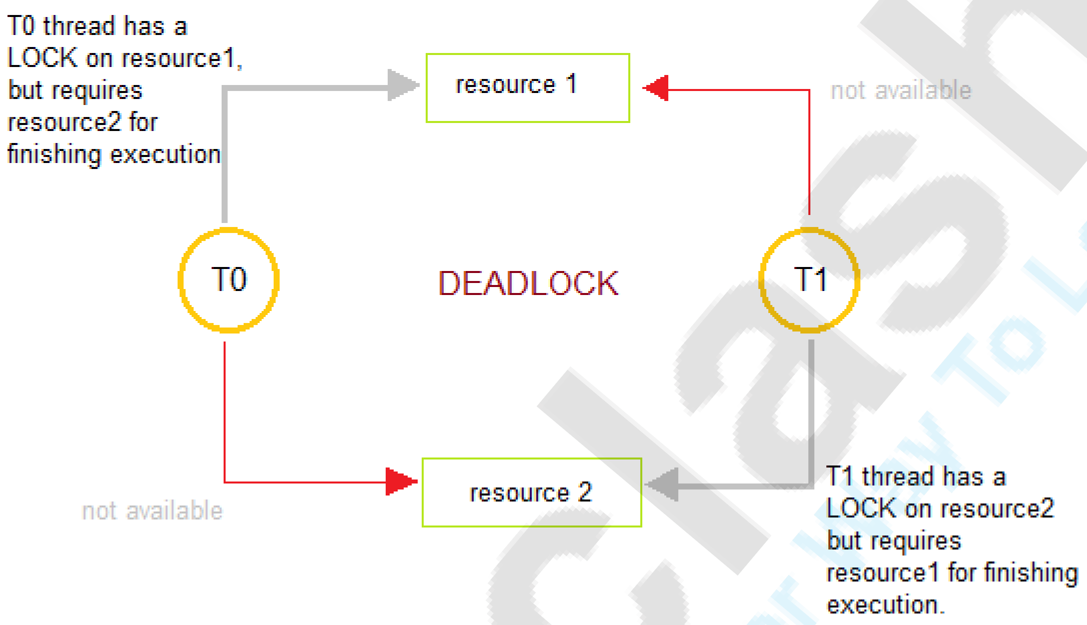


Figure 7.9 (a) Resource-allocation graph.

Q.4	Explain Deadlock.

<u>Deadlock:</u>

A deadlock is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function.


T0 thread has a LOCK on resource1, but requires resource2 for finishing execution
resource 1
not available
T0
DEADLOCK
T1
not available
resource 2
T1 thread has a LOCK on resource2 but requires resource1 for finishing execution.

<u>How to avoid Deadlocks</u>

Deadlocks can be avoided by avoiding at least one of the four conditions, because all this four conditions are required simultaneously to cause deadlock.

1. Mutual Exclusion

Resources shared such as read-only files do not lead to deadlocks but resources, such as printers and tape drives, requires exclusive access by a single process.

2. Hold and Wait

In this condition processes must be prevented from holding one or more resources while simultaneously waiting for one or more others.

3. No Preemption

Preemption of process resource allocations can avoid the condition of deadlocks, where ever possible.

4. Circular Wait

Circular wait can be avoided if we number all resources, and require that processes request resources only in strictly increasing(or decreasing) order.

	Handling Deadlock The above points focus on preventing deadlocks. But what to do once a deadlock has occured. Following three strategies can be used to remove deadlock after its occurrence. 1. Preemption We can take a resource from one process and give it to other. This will resolve the deadlock situation, but sometimes it does causes problems. 2. Rollback In situations where deadlock is a real possibility, the system can periodically make a record of the state of each process and when deadlock occurs, roll everything back to the last checkpoint, and restart, but allocating resources differently so that deadlock does not occur. 3. Kill one or more processes This is the simplest way, but it works.
Q.5	Explain Deadlock Characterization .
Ans.	**Deadlock characterization:** In deadlock, processes never finish executing and system resources are tied up, preventing other jobs from ever starting. **Necessary Conditions** A deadlock situation can arise if the following four conditions hold simultaneously in a system: 1. **Mutual exclusion:** At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released. 2. **Hold and wait** : There must exist a process that is holding at least one resource and is waiting to acquire additional resources that are currently being held by other processes. 3. **No preemption :** Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process, has completed its task. 4. **Circular wait:** There must exist a set {P0, P1, ..., Pn } of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, …., Pn-1 is waiting for a resource that is held by Pn, and Pn is waiting for a resource that is held by P0. **Resource-Allocation Graph** The resource-allocation graph shown in Figure 7.1 depicts the following situation. • The sets P, R, and E: ◦ P = {P1, P2, P3} ◦ R = {R1, R2, R3, R4} ◦ E = {P1 → R1, P2 → R3, R1 → P2, R2 → P2, R2 → P1, R3 → P3} • Resource instances:

◦ One instance of resource type R1
◦ Two instances of resource type R2
◦ One instance of resource type R3
◦ Three instances of resource type R4
• Process states:
◦ Process P1 is holding an instance of resource type R2 and is waiting for an instance of resource type R1.
◦ Process P2 is holding an instance of R1 and an instance of R2 and is waiting for an instance of R3.
◦ Process P3 is holding an instance of R3.

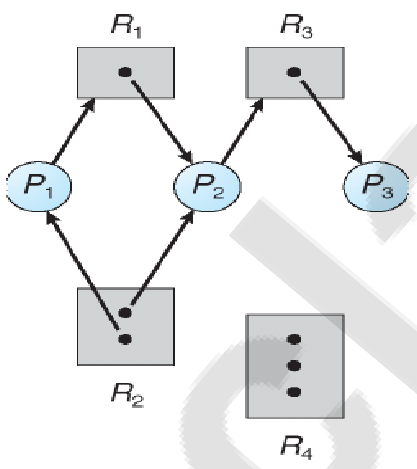


Fig. Resource Allocation Graph

Resource Allocation Graph with Deadlock

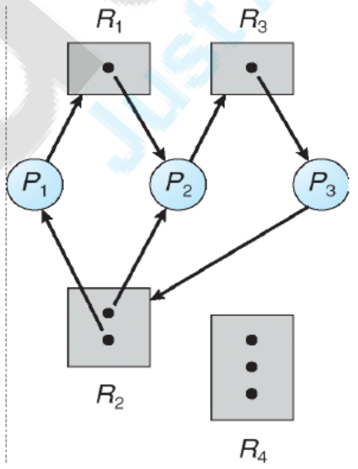


Fig. Resource Allocation Graph with Deadlock

Q. 6	Explain Deadlock Recovery.
Ans	**Recovery:** When a detection algorithm determines that a deadlock exists, several alternatives are available. One possibility is to inform the operator that a deadlock has occurred and to let the operator deal with the deadlock manually. Another possibility is to let the system recover from the deadlock automatically. There are two options for breaking a deadlock: 1. Process Termination 2. Resource preemption **Process Termination:** To eliminate deadlocks by aborting a process, we use one of two methods. In both methods, the system reclaims all resources allocated to the terminated processes. a. Abort all deadlocked processes. This method clearly will break the deadlock cycle, but at great expense. The deadlocked processes may have computed for a long time, and the results of these partial computations must be discarded and probably will have to be recomputed later. b. Abort one process at a time until the deadlock cycle is eliminated. This method incurs considerable overhead, since after each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked. **Resource preemption:** To eliminate deadlocks using resource preemption, we successively preempt some resources fromprocesses and give these resources to other processes until the deadlock cycle is broken. 1. **Selecting a victim**. Which resources and which processes are to be preempted? As in process termination, we must determine the order of preemption to minimize cost. Cost factors may include such parameters as the number of resources a deadlocked process is holding and the amount of time the process has thus far consumed. 2. **Rollback:** If we preempt a resource from a process, what should be done with that process? Clearly, it cannot continue with its normal execution; it is missing some needed resource. We must roll back the process to some safe state and restart it from that state. 3. **Starvation:** How do we ensure that starvation will not occur? That is, how can we guarantee that resources will not always be preempted from the same process?

Q.7 What is Deadlock Avoidance.

Deadlock Avoidance :-

- Avoiding deadlocks is to require additional information about how resources are to be requested.
- A deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular wait condition.
- The resource allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

There are various methods used for the purpose of deadlock avoidance:-

A. Safe State

A state is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock. More formally, a system is in a safe state only if there exists a safe sequence. If no such sequence exists, then the system state is said to be unsafe.

B. Resource-Allocation Graph Algorithm

Suppose that process Pi requests resource Rj. The request can be granted only if Converting the request edge Pi □□Rj to an assignment edge Rj □□Pi does not result in the formation of a cycle in the resource-allocation graph.

C. Banker's Algorithm

This resource-allocation graph algorithm is applicable to a resource-allocation system with multiple instances of each resource type. This algorithm is commonly known as the banker's algorithm.

Q. 8 What is Deadlock Prevention.

Methods to prevent a deadlock situation:- For a deadlock to occur, each of the four necessary-conditions must hold. By ensuring that at least on one these conditions cannot hold, we can prevent the occurrence of a deadlock.

a.Mutual Exclusion
The mutual-exclusion condition must hold for non-sharable resources. Sharable resources, on the other hand, do not require mutually exclusive access, and thus cannot be involved in a deadlock.
b. Hold and Wait
1. Whenever a process requests a resource, it does not hold any other resources.
2. An alternative protocol allows a process to request resources only when the process has none.
c. No Preemption
If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are

	released implicitly. Then the preempted resources are added to the list of resources for which the process is waiting. **d.Circular Wait** Circular-wait condition never holds is to impose a total ordering of all resource types, and to require that each process requests resources in an increasing order of enumeration.
Q. 9	Topic left…
	Concurrency control: concurrency and race condition Mutual Exclusion requirements Software and hardware solution
Q. 10	
	•
Q.11	
	•
Q.12	
Q.13	
Q.14	
Q.15	