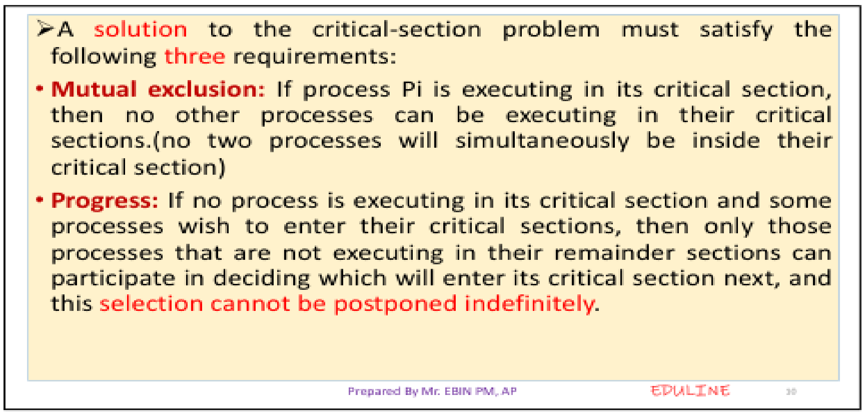
Solved Questions Module-3

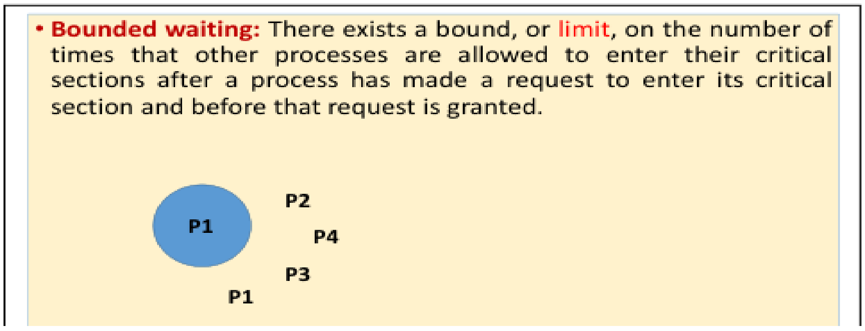
1. Explain the conditions to be satisfied by a solution to the critical section problem.

2. What are the three conditions to be satisfied by a solution to critical section problem?



➢A solution to the critical-section problem must satisfy the following three requirements:

- **Mutual exclusion:** If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.(no two processes will simultaneously be inside their critical section)
- **Progress:** If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.



- **Bounded waiting:** There exists a bound, or limit, on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

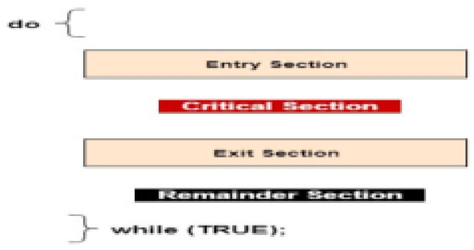
P1
P2
P4
P3
P1

3. What is meant by critical section? What is critical section problem?

Definition of critical section – 1 mark

The **critical section** refers to the segment of code where processes access shared resources, such as common variables and files, and perform write operations on them.

General structure of critical section – 1 Mark



Definition of critical section problem - 1 mark

The **critical section problem** is used to design a protocol followed by a group of processes, so that when one process has entered its critical section, no other process is allowed to execute in its critical section.

4. Define semaphore with its operations. What are the two types of Semaphores?

5. Explain the two operations of semaphores.

6. Explain the wait and signal operations used in semaphores.

SEMAPHORE

The hardware-based solutions to the critical-section problem presented are complicated for application programmers to use. To overcome this difficulty a synchronization tool called a semaphore is used. **A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait () and signal ().** The wait () operation was originally termed P (from the Dutch *proberen,* "to test"); signal() was originally called V (from *verhogen,* "to increment"). The definition of wait () is as follows:

```
wait(S)
{

while S <= 0
II busy waiting
s--;

}
```

The definition of signal() is as follows:

```
signal(S)
{
S++;
}
```

All modifications to 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 wait (S), the testing of the integer value of S (S <- 0), as well as its possible modification (S--), must be executed without interruption.

Semaphore Usage

Operating systems often distinguish between counting and binary semaphores.

- The value of **a counting semaphore** can range over an unrestricted domain.
- The value of **a binary semaphore** can range only between 0 and 1.

Counting semaphores can be used to control access to a given resource consisting of a finite number of instances. The semaphore is initialized to the number of resources available. Each process that wishes to use a resource performs a wait() operation on the semaphore (thereby decrementing the count). When a process releases a resource, it performs a signal() operation (incrementing the count). When the count for the semaphore goes to 0, all resources are being used. After that, processes that wish to use a resource will block until the count becomes greater than 0.

- Consider P_1 and P_2 that require S_1 to happen before S_2
  Create a semaphore “synch” initialized to 0

```
P1:
  S1;
  signal(synch);
P2:
  wait(synch);
  S2;
```

Semaphore Implementation

When a process executes the wait() operation and finds that the semaphore value is not positive, it must wait. However, rather than engaging in busy waiting, the process can block itself. The block operation places a process into a waiting queue associated with the semaphore, and the state of the process is switched to the waiting state. Then control is transferred to the CPU scheduler, which selects another process to execute.

A process that is blocked, waiting on a semaphore S, should be restarted when some other process executes a signal() operation. The process is restarted by a wakeup() operation, which changes the process from the waiting state to the ready state. The process is then placed in the ready queue.

- a semaphore can be defined as follows:

```
typedef struct{
 int value;
 struct process *list;
 } semaphore;
```

- The wait() semaphore operation can be defined as

```
wait(semaphore *S) {
  S->value--;
  if (S->value < 0) {
    add this process to S->list;
    block();
  }
}
```

- the signal() semaphore operation can be defined as

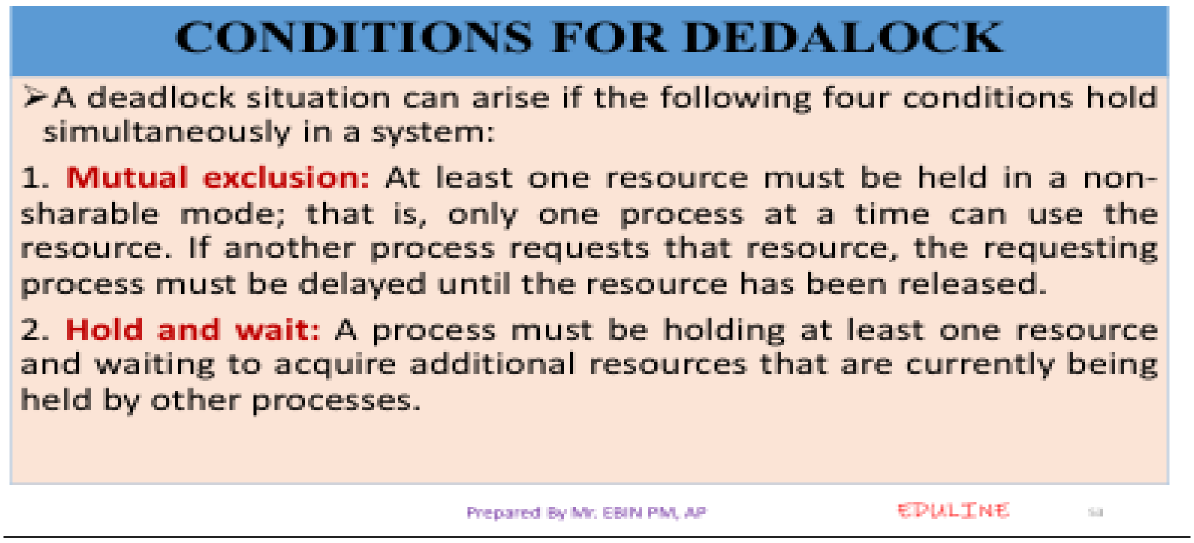
```
signal(semaphore *S) {
  S->value++;
  if (S->value <= 0) {
    remove a process P from S->list;
    wakeup(P);
  }
```

The block() operation suspends the process that invokes it. The wakeup(P) operation resumes the execution of a blocked process P.

If a semaphore value is negative, its magnitude is the number of processes waiting on that semaphore. This fact results from switching the order of the decrement and the test in the implementation of the wait() operation.

7. Explain four necessary conditions for deadlock to occur.

8. What are necessary conditions which can lead to a deadlock situation in a system?

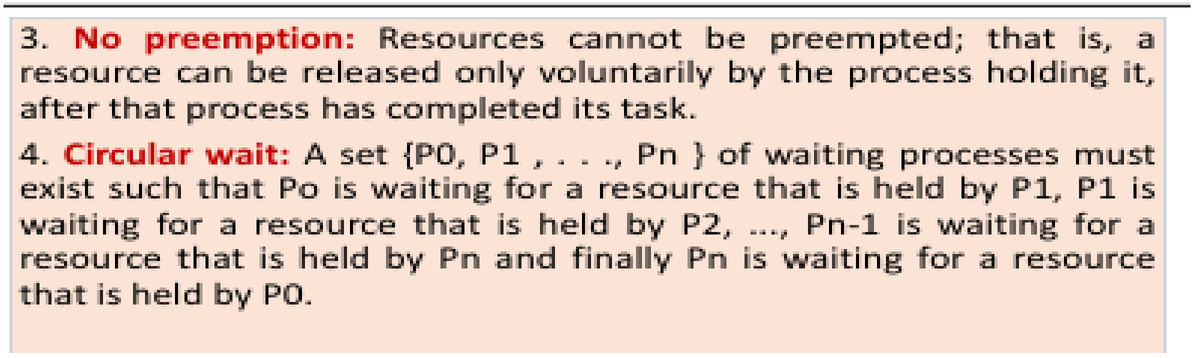


CONDITIONS FOR DEDALOCK

➢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:** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

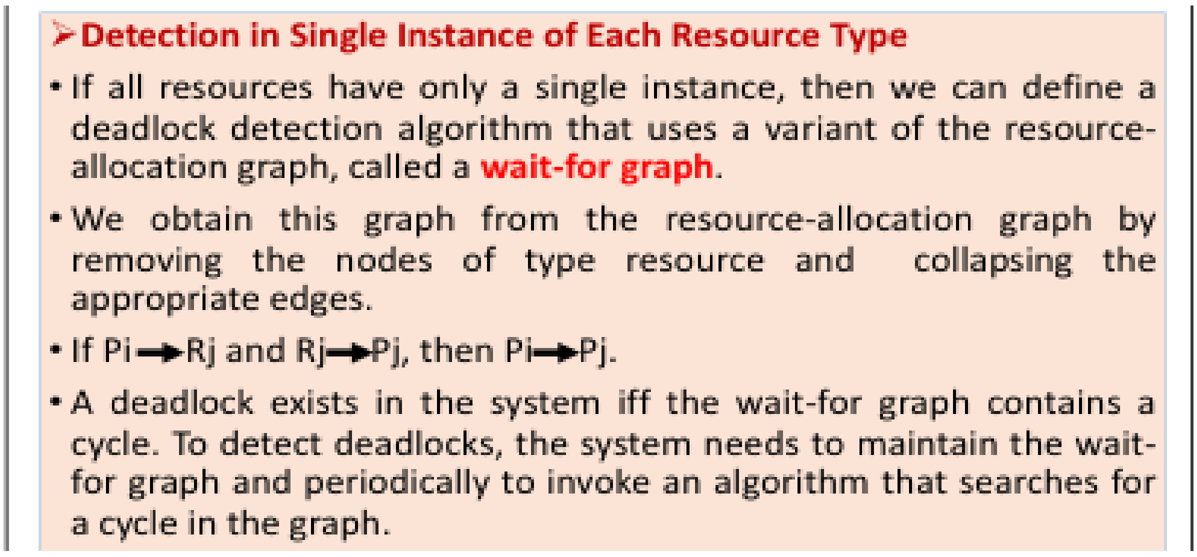


3. **No preemption:** Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.

4. **Circular wait:** A set {P0, P1 , . . ., Pn } of waiting processes must exist such that Po 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 finally Pn is waiting for a resource that is held by P0.

9. Illustrate the use of wait for graph with suitable example.

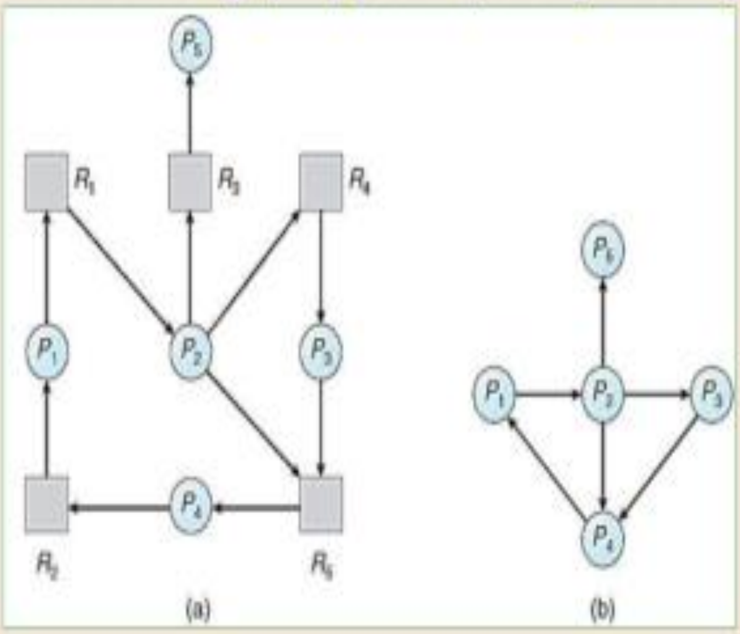
10. Explain with an example, how wait for graph is used to detect deadlocks?



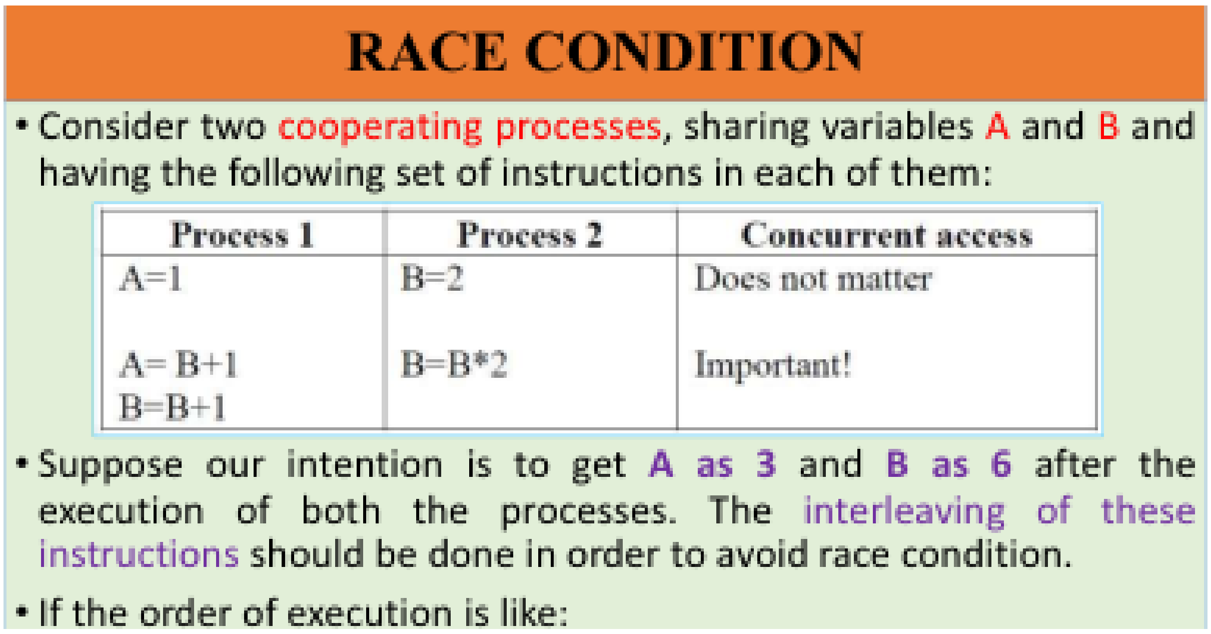
➢**Detection in Single Instance of Each Resource Type**

- 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.
- If Pi→Rj and Rj→Pj, then Pi→Pj.
- A deadlock exists in the system iff the wait-for graph contains a cycle. To detect deadlocks, the system needs to maintain the wait-for graph and periodically to invoke an algorithm that searches for a cycle in the graph.

Resource-allocation graph. (b) Corresponding wait-for graph



11. What is meant by race condition? Explain with the help of an example.



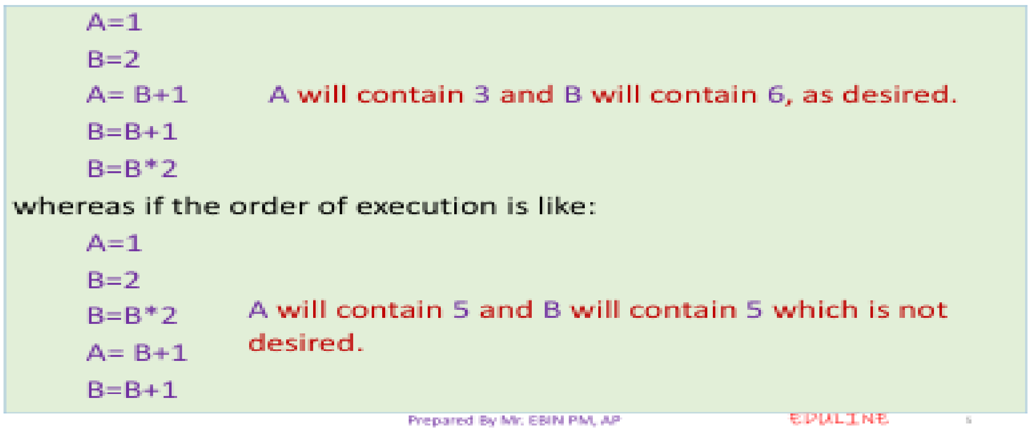
RACE CONDITION

- Consider two cooperating processes, sharing variables A and B and having the following set of instructions in each of them:

Process 1	Process 2	Concurrent access
A=1	B=2	Does not matter

A= B+1
B=B+1
	B=B*2	Important!

- Suppose our intention is to get A as 3 and B as 6 after the execution of both the processes. The interleaving of these instructions should be done in order to avoid race condition.
- If the order of execution is like:

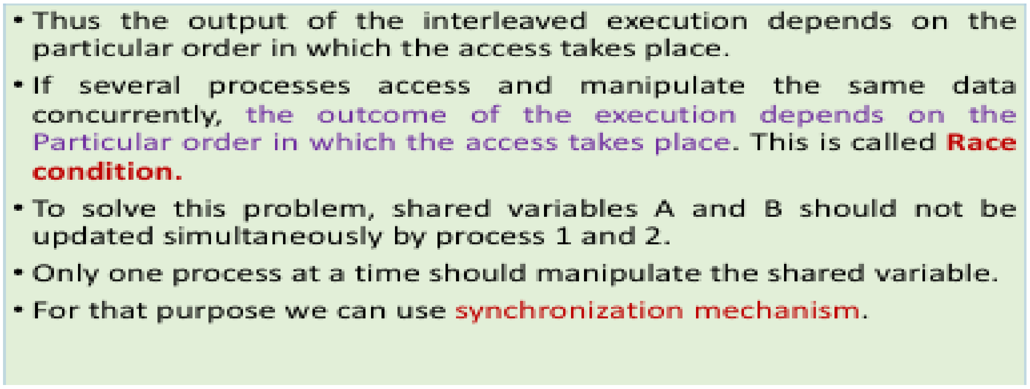


```
A=1
B=2
A= B+1        A will contain 3 and B will contain 6, as desired.
B=B+1
B=B*2
```

whereas if the order of execution is like:

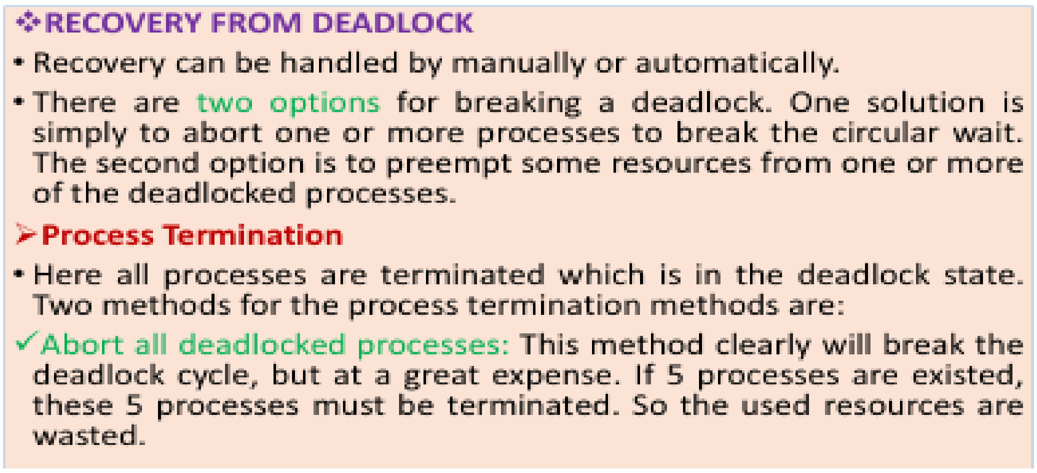
```
A=1
B=2
B=B*2        A will contain 5 and B will contain 5 which is not
A= B+1       desired.
B=B+1
```

Prepared By Mr. EBIN PM, AP EDULINE 5



- Thus the output of the interleaved execution depends on the particular order in which the access takes place.
- If several processes access and manipulate the same data concurrently, the outcome of the execution depends on the Particular order in which the access takes place. This is called **Race condition.**
- To solve this problem, shared variables A and B should not be updated simultaneously by process 1 and 2.
- Only one process at a time should manipulate the shared variable.
- For that purpose we can use synchronization mechanism.

12. Explain the two strategies used to recover from a deadlock.



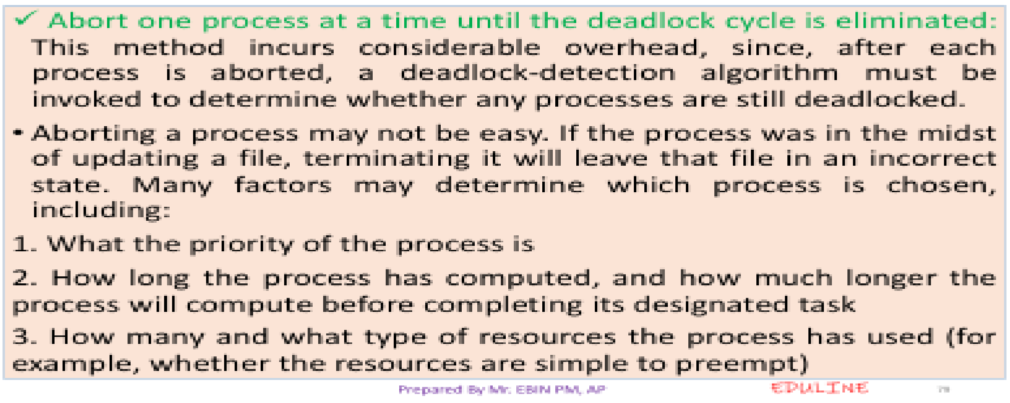
❖RECOVERY FROM DEADLOCK

- Recovery can be handled by manually or automatically.
- There are two options for breaking a deadlock. One solution is simply to abort one or more processes to break the circular wait. The second option is to preempt some resources from one or more of the deadlocked processes.

➢Process Termination

- Here all processes are terminated which is in the deadlock state. Two methods for the process termination methods are:

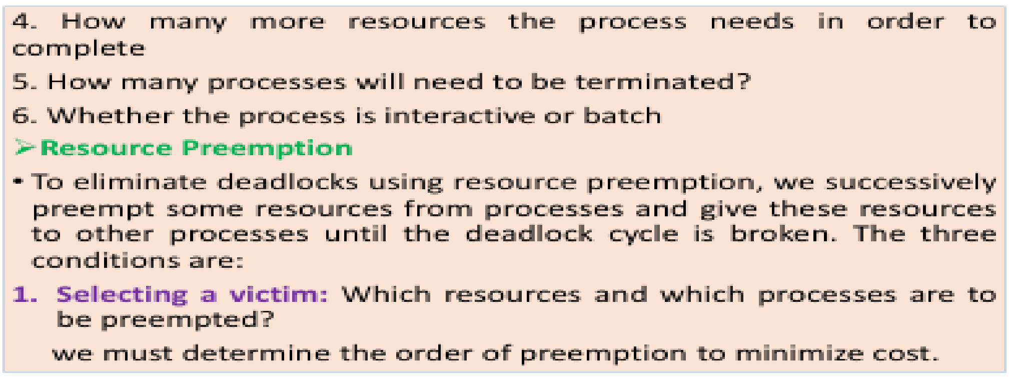
✓Abort all deadlocked processes: This method clearly will break the deadlock cycle, but at a great expense. If 5 processes are existed, these 5 processes must be terminated. So the used resources are wasted.


✓ 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.

- Aborting a process may not be easy. If the process was in the midst of updating a file, terminating it will leave that file in an incorrect state. Many factors may determine which process is chosen, including:

1. What the priority of the process is
2. How long the process has computed, and how much longer the process will compute before completing its designated task
3. How many and what type of resources the process has used (for example, whether the resources are simple to preempt)

Prepared By Mr. EBIN PM, AP EDULINE

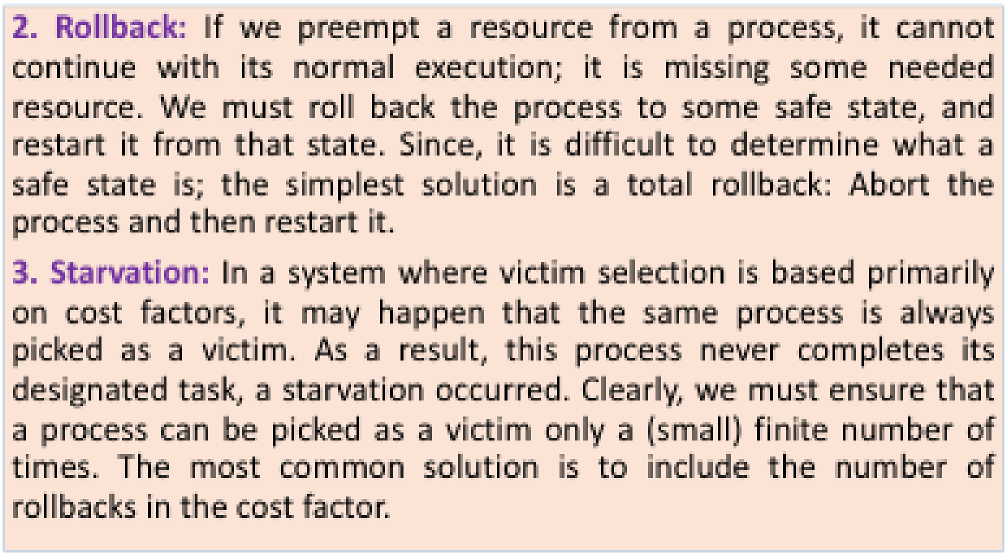

4. How many more resources the process needs in order to complete
5. How many processes will need to be terminated?
6. Whether the process is interactive or batch

➢ Resource Preemption

- To eliminate deadlocks using resource preemption, we successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken. The three conditions are:

1. Selecting a victim: Which resources and which processes are to be preempted?

   we must determine the order of preemption to minimize cost.


2. Rollback: If we preempt a resource from a process, 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. Since, it is difficult to determine what a safe state is; the simplest solution is a total rollback: Abort the process and then restart it.

3. Starvation: In a system where victim selection is based primarily on cost factors, it may happen that the same process is always picked as a victim. As a result, this process never completes its designated task, a starvation occurred. Clearly, we must ensure that a process can be picked as a victim only a (small) finite number of times. The most common solution is to include the number of rollbacks in the cost factor.

13. Explain with an example the improper usage of semaphore causing deadlocks?

Deadlocks and Starvation

The implementation of a semaphore with a waiting queue may result in a situation where two or more processes are waiting indefinitely for an event that can be caused only by one of the waiting processes. When such a state is reached, these processes are said to be deadlocked.

- **Let *S* and *Q* be two semaphores initialized to 1**

```
        P0                P1
    wait(S);          wait(Q);
    wait(Q);          wait(S);
      ...               ...
    signal(S);        signal(Q);
    signal(Q);        signal(S);
```

Suppose that P0 executes wait(S) and then P1 executes wait(Q). When P0 executes wait(Q), it must wait until P1 executes signal(Q). Similarly, when P1 executes wait(S), it must wait until P0 executes signal(S). Since these signal() operations cannot be executed, P0 and P1 are deadlocked.

Another problem related to deadlocks is indefinite blocking or starvation, a situation in which processes wait indefinitely within the semaphore.

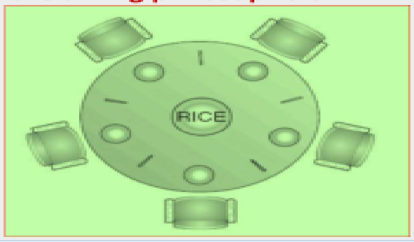
Priority Inversion

- Scheduling problem when lower-priority process holds a lock needed by higher-priority process is called priority inversion.
- Solution
    - to have only two priorities
    - implement a priority-inheritance protocol- According to this protocol, all processes that are accessing resources needed by a higher-priority process inherit the higher priority until they are finished with the resources in question. When they are finished, their priorities revert to their original values.

14. Explain dining philosopher problem.

DINING PHILOSOPHERS PROBLEM

❖ The situation of the dining philosophers



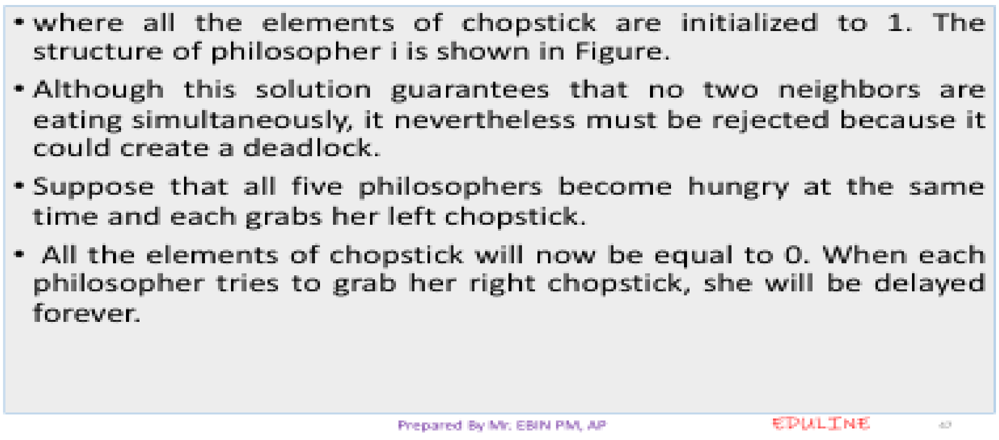
- Table is a bowl of rice, and the table is laid with five single chopsticks. When a philosopher thinks, she does not interact with her colleagues.
- From time to time, a philosopher gets hungry and tries to pick up the two chopsticks that are closest to her (the chopsticks that are between her and her left and right neighbors).
- A philosopher may pick up only one chopstick at a time. Obviously, she cannot pick up a chopstick that is already in the hand of a neighbor.
- When a hungry philosopher has both her chopsticks at the same time, she eats without releasing the chopsticks. When she is finished eating, she puts down both chopsticks and starts thinking again.

- One simple solution is to represent each chopstick with a semaphore.
- A philosopher tries to grab a chopstick by executing a wait() operation on that semaphore.
- She releases her chopsticks by executing the signal() operation on the appropriate semaphores. Thus, the shared data are

semaphore chopstick[5];

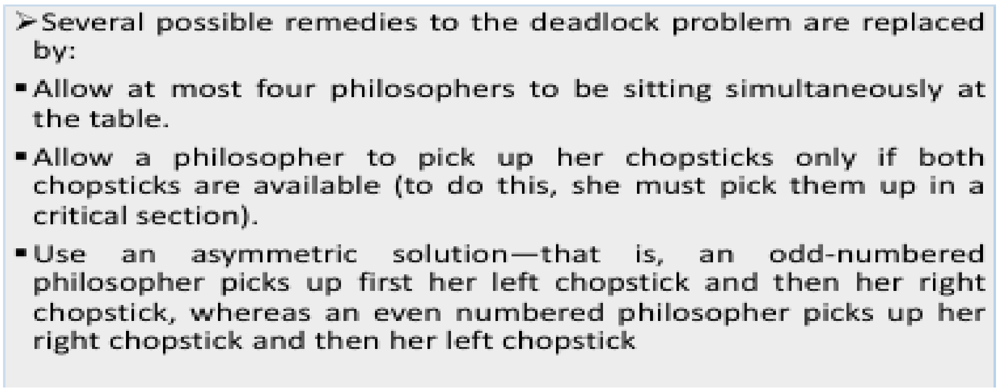
The structure of philosopher I

```
do {
    wait(chopstick[i]);
    wait(chopstick[(i+1) % 5]);
        . . .
    /* eat for awhile */
        . . .
    signal(chopstick[i]);
    signal(chopstick[(i+1) % 5]);
        . . .
    /* think for awhile */
        . . .
} while (true);
```



- where all the elements of chopstick are initialized to 1. The structure of philosopher i is shown in Figure.
- Although this solution guarantees that no two neighbors are eating simultaneously, it nevertheless must be rejected because it could create a deadlock.
- Suppose that all five philosophers become hungry at the same time and each grabs her left chopstick.
- All the elements of chopstick will now be equal to 0. When each philosopher tries to grab her right chopstick, she will be delayed forever.

Prepared By Mr. EBIN PM, AP EDULINE



➢ Several possible remedies to the deadlock problem are replaced by:
- Allow at most four philosophers to be sitting simultaneously at the table.
- Allow a philosopher to pick up her chopsticks only if both chopsticks are available (to do this, she must pick them up in a critical section).
- Use an asymmetric solution—that is, an odd-numbered philosopher picks up first her left chopstick and then her right chopstick, whereas an even numbered philosopher picks up her right chopstick and then her left chopstick

15. Explain the priority inheritance protocol for solving the priority inversion problem.

Priority Inversion

- ➢ Scheduling problem when lower-priority process holds a lock needed by higher-priority process is called priority inversion.
- ➢ Solution
  - to have only two priorities
  - implement a priority-inheritance protocol- According to this protocol, all processes that are accessing resources needed by a higher-priority process inherit the higher priority until they are finished with the resources in question. When they are finished, their priorities revert to their original values.