6...

Deadlock

Objectives...

- To learn about concept and principles of Deadlock.
- To study the Deadlock Characterization using Necessary Conditions.
- To learn about different Deadlock handling techniques such as Deadlock Prevention and Avoidance, Detection and Recovery.

6.1 INTRODUCTION [S-18, 23]

- Every process needs some resources to complete its execution. However, the resource is granted in a sequential order.
  1. The process requests for some resource.
  2. Operating system grant the resource if it is available otherwise let the process waits.
  3. The process uses it and release on the completion.
- In an operating system, deadlock state happens when two or more processes are waiting for the same event to happen which never happen, then we can say that those processes are involved in the deadlock.
- For example, Process P1 wants to access the resource R1 while it has access to resource R2. It will release the resource only after it has acquired R2. There is another process P2, which has the access to resource R2 and wants the access to resource R1. This process is also developed in such a way that it will release R2 only when it has acquired R1.

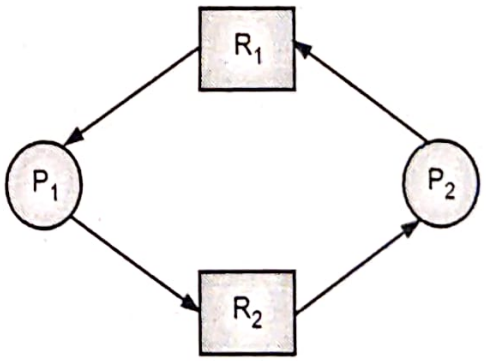


Fig. 6.1: Deadlock state in Operating System

- So, currently in the system there are two processes which have access to one of the system resources and want access to another resource. And they will leave the resource only when they get access to the new resource. This condition is called a deadlock as no process will get to access the required resource and continue execution.

Example of Deadlock State:

(A) Deadlock involving the same resource type:

- To illustrate a deadlock state, we consider a system with three CD-RW drives.
  - Suppose each of the three processes holds one of these CD-RW drives.
  - If each process now requests another drive, the three processes will be in a deadlock state.
  - Each is waiting for the event "CD-RW is released" which can be caused only by one of the other waiting processes.
- This example illustrates a deadlock involving the same resource type.

(B) Deadlock involving the different resource types:

- Deadlocks may also involve different resource types. For example, consider a system with one printer and one tape drive. Suppose process P_i is holding the tape drive and process P_j is holding the printer.
- If P_i requests the printer and P_j requests the tape drive, deadlock occurs.
- A programmer who is developing multithreaded application must pay particular attention to this problem. Multithreaded programs are good candidates for deadlock because multiple threads can compete for shared resources.

6.2 DEADLOCK CHARACTERIZATION [S-23]

- In a deadlock, processes never finish executing and system resources are tied up, preventing other jobs from starting.
- Before we discuss the various methods for dealing with the deadlock problems, we shall describe features that characterize deadlocks.

6.3 NECESSARY CONDITIONS [W-18, S-19, 23]

- A deadlock situation can arrive if the following four conditions hold simultaneously in a system.

1. **Mutual Exclusion**: At least one resource is held in a non-sharable mode; that is only one process at a time can use the resource. Example monitor, printer etc. If another process requests that resource, the requesting process must be delayed until the resource has been released. Each resource is either currently assigned to exactly one process or is available.
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:** The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.

4. **Circular Wait:** All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.

6.4 DEADLOCK HANDLING TECHNIQUES

- There are mainly four methods for handling technique:
  1. Deadlock Prevention
  2. Deadlock Avoidance
  3. Deadlock Detection
  4. Recovery from Deadlock

6.4.1 Deadlock Prevention

- Deadlock prevention is a set of methods for ensuring that at least one of the necessary conditions cannot hold.
- That means that we design such a system where there is no chance of having a deadlock.
- By ensuring that at least one of the following conditions cannot hold, we can prevent the occurrence of a deadlock.
  - **(a) Mutual exclusion:** It can't be resolved as it is the hardware property. For example, the printer cannot be simultaneously shared by several processes. This is very difficult because some resources are not sharable.
  - **(b) Hold and wait:** Hold and wait can be resolved using the conservative approach where a process can start it and only if it has acquired all the resources.
  - **(c) Active approach:** Here the process acquires only requires resources but whenever a new resource requires it must first release all the resources.
  - **(d) Wait time out:** Here there is a maximum time bound until which a process can wait for other resources after which it must release the resources.
  - **(e) Circular wait:** In order to remove circular wait, we assign a number to every resource and the process can request only in the increasing order otherwise the process must release all the high number acquires resources and then make a fresh request.
  - **(f) No preemption:** In no preemption, we allow forceful pre-emption where a resource can be forcefully pre-empted. The pre-empted resource is added to the list of resources where the process is waiting. The new process can be restarted only when it regains its old resources. Priority must be given to a process which is in waiting for state.

6.4.2 Deadlock Avoidance [S-19, 23]

- It requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional information, we can decide for each request can be satisfied or must be delayed.
- There are various algorithms are available to handle this situation.

- **A Deadlock Avoidance Algorithm** dynamically examines the resources allocation state to ensure that a circular wait condition case never exists. Where the resources allocation state is defined by the available and allocated resources and the maximum demand of the process.
- **Examples:** Resource Allocation Graph algorithm, Banker's algorithm.

6.4.2.1 Safe State [S-23]

- There are 3 states of the system: safe, unsafe and deadlock.
- When a system can allocate the resources to the process in such a way so that they still avoid deadlock then the state is called **safe state.**
- A system is in safe state only if there exists a **safe sequence.**
- A sequence of process $P_1, P_2, ..., P_n$ is a safe sequence for the current allocation state if for each P_i the resources request that P_i can still make can be satisfied by currently available resources pulls the resources held by all P_j with $j < i$.

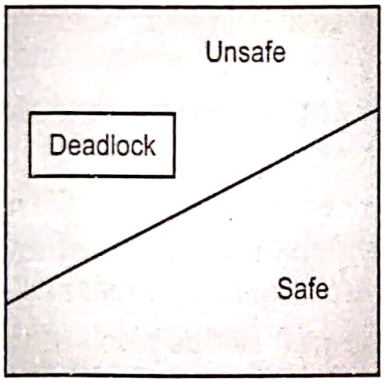


Fig. 6.2: Space of Safe, Unsafe and Deadlock States

- If a safe sequence does not exist, then the system is in an **unsafe** state, which may lead to deadlock. (All safe states are deadlock free, but not all unsafe states lead to deadlocks).

6.4.2.2 Resource Allocation Graph Algorithm [S-18, 22; W-18]

- Deadlock can be described more precisely in terms of a directed graph called a system Resource Allocation Graph.
- In Resource Allocation Graph, processes are represented by circle and resources are represented by boxes. Resource boxes have some number of dots inside indicating available number of that resource that is number of instances.

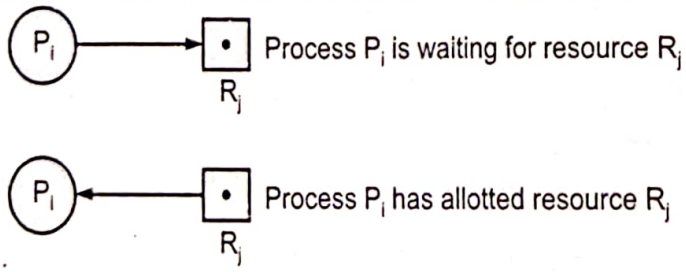


Fig. 6.3: Representation of Process and Resources in the Graph

- This graph consist of a set of vertices V and a set of edges E.
- The set of vertices V is partitioned into two different types of nodes:

$$P = \{P_1, P_2, P_3, ... P_n\}$$

- The set consisting of all the active processes in the system, and

$$R = \{R_1, R_2, R_3, \ldots R_m\}$$

- The set consisting of all resource types in the system.
- A directed edge from process P_i to resource type R_j is denoted by:

$$P_i = R_j - \text{request edge}$$

- It signifies that process P_i requested an instance of resource type R_j and is currently waiting for that resource.
- A directed edge from resource type R_j to process P_i is denoted by:

$$R_j = P_i - \text{assignment edge}$$

- It signifies that an instance of resource type R_j has been allocated to process P_i.
- When process P_i request an instance of resource type R_j, a request edge is inserted in the resource allocation graph.
- When this request can be fulfilled, the request edge is instantaneously transformed to an assignment edge. When the process no longer needs access to the resource, it release the resource and result is assignment edge is deleted.
- The resource allocation graph is shown in Fig. 6.4.

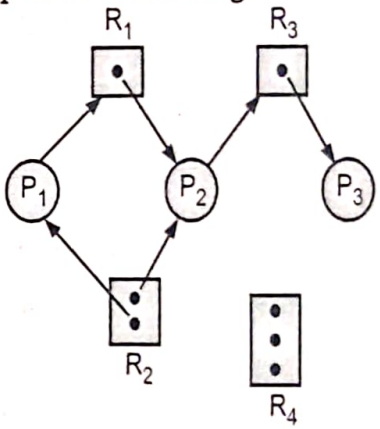


Fig. 6.4: Resource Allocation Graph

- Pictorially process P_i is represented as a circle and each resource, R_j as square since resource type R_j may have more than one instance, we represent each such instance as a dot within the square.
- Also note that a request edge points to only square R_j, whereas an assignment edge must also designate one of the square.
  - The sets P, R and E

$$P = \{P_1, P_2, P_3\}$$

$$R = \{R_1, R_2, R_3, R_4\}$$

$$E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$$

  - Resource instances
    - One instance of resource type R_1.
    - Two instances of resource type R_2.
    - One instance of resource type R_3.
    - Three instances of resource type R_4.

- Process states
  - Process P_1 is holding an instance of resource type R_2, and is waiting for an instance of resource type R_1.
  - Process P_2 is holding an instance of R_1 and R_2 and is waiting for an instance of resource type R_3.
  - Process P_3 is holding an instance of R_3.

- Given the definition of resource allocation graph, it can be shown that if the graph contains no cycles, then no process in a system is deadlocked. If the graph contains a cycle, then a deadlock may exist.
- If each resource type has one instance, then a cycle implies that a deadlock has occurred. If the cycle involves only a set of resource types, each of which has only a single instance, then a deadlock has occurred. Each process involved in a cycle is deadlocked.
- In this case, cycle in the graph is both necessary and a sufficient condition for the existence of deadlock if single instance of each and every resource is there in graph.
- If each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occurred. In this case, cycle is necessary but not sufficient condition for the existence of deadlock.
- To illustrate this concept, let us return to the resource-allocation graph shown in Fig. 6.4. Suppose the process P_3 requests an instance of resource type R_2.
- Since, no resource instance is currently available, a request edge $P_3 \rightarrow R_2$ is added to the graph. At this time, two minimal cycles exist in the system.

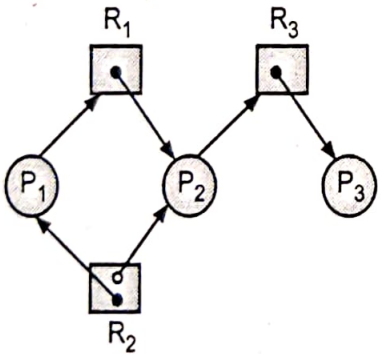
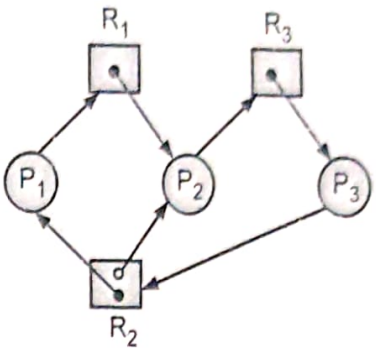


Fig. 6.5: Resource Allocation Graph with a Deadlock

- Process P_1, P_2, P_3 is deadlocked. Process P_2 is waiting for resource R_3, which is held by process P_3, on the other hand, is waiting for either process P_1 or process P_2 to release resource R_2.
- In addition, process P_1 is waiting for process P_2 to release resource R_1.
- Now consider the resource allocation graph in Fig. 6.6. In this example, we have a cycle.



$$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$$

$$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$$

Fig. 6.6: Resource-Allocation Graph with a Cycle but no Deadlock

- However, there is no deadlock because process P_4 may release its instance of resource type R_2. That resource can be allocated to P_3, breaking the cycle.
- In short, if a resource allocation graph does not have a cycle, then the system is not in a deadlock state. On the other hand, if there is a cycle, then the system may or may not be in a deadlock state.

6.4.2.3 Banker's Algorithm [S-19]

- Banker's Algorithm is a **deadlock avoidance algorithm**. It is also used for deadlock detection.
- This algorithm tells that if any system can go into a deadlock or not by analyzing the currently allocated resources and the resources required by it in the future.

Claim Edge:

- Let us introduce a new kind of edge, a *claim edge*, represented by a dashed line. Like a request edge, a claim edge points from a process to a resource type.

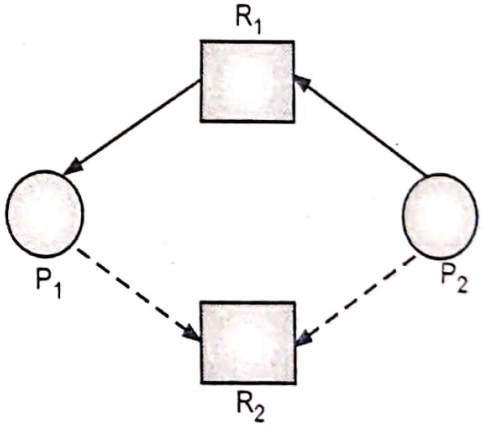


Fig. 6.7: Resource Allocation Graph for Deadlock Avoidance

- A claim edge indicates that a process P_i *may request* the resource R_j some time *in the future.*
- All claim edges for P_i must be present before P_i starts executing.
- Thus, a request edge (P_i, R_j) may be added only if a claim edge (P_i, R_j) is already present.
- A resource-allocation graph with claim edges is called a *maximum-claim graph.* It reflects the projected worst-case future state in resource allocation.
- A state is safe if its corresponding maximum-claim graph is deadlock free.

Data Structures used to implement Banker's Algorithm:

- Several data structures must be maintained to implement the Banker's algorithm. These data structures encode the state of the resource allocation system. Let P_1 be the number of processes in the system and R_1 be the number of resource types.

1. **Available:** It is a 1-D array that tells the number of each resource type (instance of resource type) currently available.

   Example: Available $[R_1]$ = A, means that there are **A** instances of R_1 resources are currently available.

2. **Max:** It is a 2-D array that tells the maximum number of each resource type required by a process for successful execution.

   Example: $Max[P_1][R_1]$ = A, specifies that the process P_1 needs a maximum of **A** instances of resource R_1 for complete execution.

3. **Allocation:** It is a 2-D array that tells the number of types of each resource type that has been allocated to the process.

   Example: $Allocation[P_1][R_1]$ = A, means that **A** instances of resource type R_1 have been allocated to the process P_1.

4. **Need:** It is a 2-D array that tells the number of remaining instances of each resource type required for execution.

   Example: Need $[P_1][R_1]$ = A tells that **A** instances of R_1 resource type are required for the execution of process P_1.

   Need[i][j] = Max[i][j] – Allocation[i][j], where i corresponds any process P(i) and j corresponds to any resource type R(j).

The Bankers Algorithm consists of the following two algorithms:

1. Request-Resource Algorithm
2. Safety Algorithm

1. Resource - Request Algorithm:

- Whenever a process makes a request of the resources then this algorithm checks that if the resource can be allocated or not.
- This algorithm includes following three steps:

Step 1 : The algorithm checks that if the request made is valid or not. A request is valid if the number of requested resources of each resource type is less than the **Need** (which was declared previously by the process). If it is a valid request then Step 2 is executed else aborted.

Step 2 : Here, the algorithm checks that if the number of requested instances of each resource is less than the available resources. If not then the process has to wait until sufficient resources are available else go to Step 3.

Step 3 : Now, the algorithm assumes that the resources have been allocated and modifies the data structure accordingly.

Available = Available – Request(i)

Allocation(i) = Allocation(i) + Request(i)

Need(i) = Need(i) – Request(i)

- After the allocation of resources, the new state formed may or may not be a safe state. So, the **Safety algorithm** is applied to check whether the resulting state is a safe state or not.
- **Safe state:** A safe state is a state in which all the processes can be executed in some arbitrary order with the available resources such that no deadlock occurs.
  1. If it is a safe state, then the requested resources are allocated to the process in actual.
  2. If the resulting state is an unsafe state then it rollbacks to the previous state and the process is asked to wait longer.

2. Safety Algorithm:

- The safety algorithm is applied to check whether a state is in a safe state or not.
- This algorithm involves the following four steps:

Step 1 : Suppose currently all the processes are to be executed. Define two data structures as work and finish as vectors of length m (where m is the length of **Available** vector) and n (is the **number of processes** to be executed).

Work = Available

Finish[i] = false for i = 0, 1, ... , n - 1.

Step 2 : This algorithm will look for a process that has **Need** value less than or equal to the **Work**. So, in this step, we will find an index i such that,

Finish[i] = = false &&

Need[i] <= Work

If no such 'i' is present then go to Step 4 else to Step 3.

Step 3 : The process '**i**' selected in the above step runs and finishes its execution. Also, the resources allocated to it get free. The resources which get free are added to the Work and Finish(i) of the process is set as true. The following operations are performed:

Work = Work + Allocation

Finish[i] = true

After performing the 3rd step, go to step 2.

Step 4 : If all the processes are executed in some sequence then it is said to be a safe state. Or, we can say that

If Finish[i] == true for all i,

Then the system is said to be in a **Safe State.**

Examples of Banker's Algorithm

Example 1: Suppose we have 3 processes (A, B, C) and 3 resource types (R_1, R_2, R_3) each having 5 instances. Suppose at any time t if the snapshot of the system taken is as follows then find the system is in a safe state or not.

Solution:

Total instances of each Resource

R_1	R_2	R_3
5	5	5

Process	Allocation			Max.		
	R_1	R_2	R_3	R_1	R_2	R_3
A	1	2	1	2	2	4
B	2	0	1	2	1	3
C	2	2	1	3	4	1
Total_alloc	5	4	3			

Available = Total – Total_alloc
= [5, 5, 5] – [5, 4, 3]
= [0, 1, 2]

So, the total allocated resources(total_alloc) are [5, 4, 3]. Therefore, the **Available** (the resources that are currently available) resources are

Available = [0, 1, 2]

Now, we will make the **Need** Matrix for the system according to the given conditions. As we know, **Need(i) = Max(i) – Allocation(i)**, so the resultant Need matrix will be as follows:

Process	Need		
	R_1	R_2	R_3
A	1	0	3
B	0	1	2
C	1	2	0

Now, we will apply the safety algorithm to check that if the given state is a safe state or not.

1. Work = Available = [0, 1, 2].
2. Also Finish[i] = false, for i = 0, 1, 2, are set as false as none of these processes have been executed.
3. Now, we check that **Need[i] ≤ Work**. By seeing the above **Need** matrix we can tell that only B[0, 1, 2] process can be executed. So, process B(**i = 1**) is allocated the resources and it completes its execution. After completing the execution, it frees up the resources.
4. Again, Work = Work + Available i.e. Work = [0, 1, 2] + [2, 0, 1] = [2, 1, 3] and Finish[1] = true.
5. Now, as we have more instances of resources available we will check that if any other process resource needs can be satisfied. With the currently available

resources [2, 1, 3], we can see that only process A [1, 2, 1] can be executed. So, process A(**i = 0**) is allocated the resources and it completes its execution. After completing the execution, it frees up the resources.

6. Again, Work = Work + Available i.e. Work = [2, 1, 3] + [1, 2, 1] = [3, 3, 4] and Finish[0] = true.
7. Now, as we have more instances of resources available we will check that if the remaining last process resource requirement can be satisfied. With the currently available resources [3, 3, 4], we can see that process C[2, 2, 1] can be executed. So, process C(**i = 2**) is allocated the resources and it completes its execution. After completing the execution, it frees up the resources.
8. Finally, Work = Work + Available i.e. Work = [3, 3, 4] + [2, 2, 1] = [5, 5, 5] and Finish[2] = true.
9. Finally, all the resources are free and there exists a safe sequence B, A, C in which all the processes can be executed. So the system is in a safe state and deadlock will not occur.

This is how Banker's Algorithm is used to check if the system is in a safe state or not.

Example 2: Consider the total amount of resources R_1, R_2 and R_3 are 9, 3 and 6 units. In the current state, allocation have been made to the four processes; leaving 1 unit of resource 2 and 1 unit of resource 3 available. Now, we will find whether the system is in safe state.

Solution: We need to find whether the difference between the current allocation and maximum requirement. In any process be met with the available resources clearly, this is not possible for P_1, which has only 1 unit of R_1 and require 2 more units of R_1, 2 units of R_2 and 2 units of R_3.

However, by assigning one unit of R_3 to process P_2, P_2 has its maximum required resources allocated and can run to completion. Let us assume that this is accomplished.

	R_1	R_2	R_3
P_1	3	2	2
P_2	6	1	3
P_3	3	1	4
P_4	4	2	2

Need Matrix

	R_1	R_2	R_3
P_1	1	0	0
P_2	6	1	2
P_3	2	1	1
P_4	0	0	2

Allocation Matrix

R_1	R_2	R_3
9	3	6

Resource Vector

R_1	R_2	R_3
0	1	1

Available Vector

(a) Initial State

Solution:

	R_1	R_2	R_3
P_1	3	2	2
P_2	0	0	0
P_3	3	1	4
P_4	4	2	2

Need Matrix

	R_1	R_2	R_3
P_1	1	0	0
P_2	0	0	0
P_3	2	1	1
P_4	0	0	2

Allocation Matrix

R_1	R_2	R_3
6	2	3

Available Vector

(b) P_2 runs to completion

	R_1	R_2	R_3
P_1	0	0	0
P_2	0	0	0
P_3	3	1	4
P_4	4	2	2

Need Matrix

	R_1	R_2	R_3
P_1	0	0	0
P_2	0	0	0
P_3	2	1	1
P_4	0	0	2

Allocation Matrix

R_1	R_2	R_3
7	2	3

Available Vector

(c) P_1 runs to completion

	R_1	R_2	R_3
P_1	0	0	0
P_2	0	0	0
P_3	0	0	0
P_4	4	2	2

Need Matrix

	R_1	R_2	R_3
P_1	0	0	0
P_2	0	0	0
P_3	0	0	0
P_4	0	0	2

Allocation Matrix

R_1	R_2	R_3
9	3	4

Available Vector

(d) P_3 runs to completion

Fig. 6.8: Determination of a Safe State

- When P_2 completes, its resources can be returned to the pool of available resources. The resulting state is shown in Fig. 6.8 (b). Now, we can ask again if any of remaining processes can be completed.
- In this case, each of the remaining processes could be completed. Suppose we choose P_1, allocate the required resources, complete P_1, and return all P_1's resources to the available pool.
- We are left in the state shown in Fig. 6.8 (c). Next, we can complete P_3, resulting in state (Fig. 6.8 (d)).
- Finally, we can complete P_4. At this point, all of the processes have been run to completion. Thus, the state defined by Fig. 6.8 (a) is a safe state, and $<P_2, P_1, P_3, P_4>$ is the safe sequence.

Example 3: The operating system contains 3 resources. The number of instances of each resource type are 7, 7, 10. The current resource allocation state is as shown as follows:

Process	Current Allocation			Maximum Need		
	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	2	3	3	6	8
P_2	2	0	3	4	4	3
P_3	1	2	4	3	3	4

Resources	Instances
R_1	7
R_2	7
R_3	10

(i) Is the current allocation in a safe state?

Solution:

(i) First we calculate available matrix.

Process	Current Allocation			Max			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	2	3	3	6	8	2	3	0
P_2	2	0	3	4	4	3			
P_3	1	2	4	3	3	4			

Then, the Need matrix will be,

Process	Need		
	R_1	R_2	R_3
P_1	1	4	5
P_2	2	4	0
P_3	2	1	0

- Then, find safe sequence. Currently (2, 3, 0) resources available and need of resources of each process is given in Need matrix. So we can fulfill the requirement of P_3 process first.
- After finishing execution, P_3 will return all the resources to the system so new available resources will resources will be (3, 5, 4).
- Now the request of P_2 process can be satisfied after P_2 finishes it will also return the resources allocated to it.
- Now available resources are (5, 5, 7), then the requirement of P_1 can be easily satisfied therefore, the safe sequence is $<P_3, P_2, P_1>$ and the system is in safe state.

Example 4: Consider the following snapshot of the system. [W-22]

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

- Is the system safe? Justify?
- If Yes, give safe sequence.

Solution:

(i) Total number of instances,

A	B	C	D
3	14	12	12

(ii) Then Need matrix is,

	A	B	C	D
P_0	0	0	0	0
P_1	0	7	5	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	6	4	2

So the system is in safe state and safe sequence is, $<P_0, P_2, P_3, P_4, P_1>$

Example 5: Consider a system with five processes P_0 through P_4 and three resource types A, B, C. Resource type A has 7 instances, resource type B has 2 instances, and resource type C has 6 instances. Suppose that at time T_0, we have the following resource allocation state.

Process	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

Solution:

- We claim that the system is not in deadlocked state. Indeed if we execute our algorithm, we will find that sequence $<P_0, P_2, P_3, P_1, P_4>$ will result in

  Finish[i] = true for all i.
- Suppose now that process P_2 makes one additional request for an instance of type C.
- The request matrix is modified as follows:

Process	Request		
	A	B	C
P_0	0	0	0
P_1	2	0	2
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

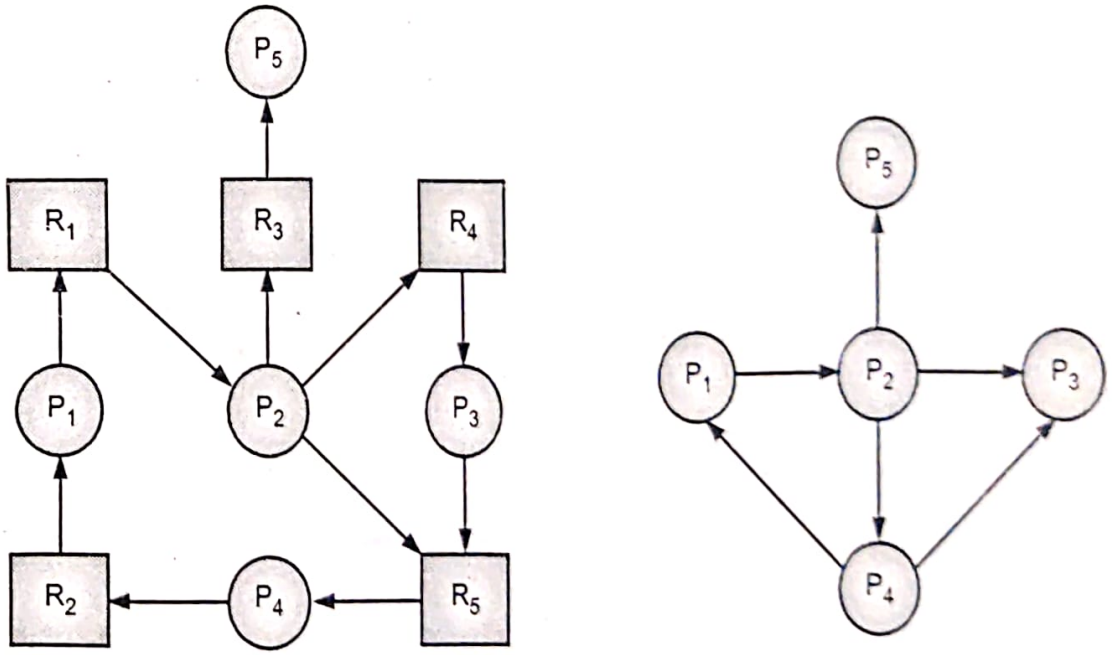
- We claim that the system is now deadlocked. Although we can reclaim the resources held by process P_0 the number of available resources is not sufficient to fulfill the request of other processes. Thus, a deadlock exists, consisting of processes P_1, P_2, P_3 and P_4.

6.4.3 Deadlock Detection

- If a system does not employ either deadlock prevention or a deadlock avoidance algorithm, then a deadlock situation may occur.
- In this environment, the system must provide:
  - An algorithm that examines the state of the system to determine whether a deadlock has occurred.
  - An algorithm to recover from the deadlock.
- In the following sections, we explain on these two requirements as they relate to systems with only a single instance of each resource type, as well as systems with several instances of each resource type.

6.4.3.1 Single Instance of each Resource Type [S-19]

- If each resource category has a single instance, then we can use a variation of the resource-allocation graph known as a wait-for graph.
- A wait-for graph can be obtained from a resource-allocation graph by eliminating the resources and collapsing the associated edges, as shown in the figure below.
- An arc from P_i to P_j in a wait-for graph indicates that process P_i is waiting for a resource that process P_j is currently holding.



(a) Resource Allocation Graph (b) Corresponding Wait for Graph

Fig. 6.9

- As before, cycles in the wait-for graph indicate deadlocks.
- This algorithm must maintain the wait-for graph, and periodically search it for cycles.

6.4.3.2 Several Instances of a Resource Type

- The algorithm employs several time-varying data structures that are similar to those used in the banker's algorithm.
  - **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 Pi is requesting k more instances of resource type R_j.

Detection Algorithm:

- The detection algorithm described here simply investigates every possible allocation sequence for the processes that remain to be completed.

Step 1 : Let Work and Finish be vectors of length m and n, respectively Initialize:

(a) Work = Available

(b) For i = 1, 2, ..., n, if Allocation i ≠ 0
then Finish[i] = false; otherwise, Finish[i] = true

Step 2 : Find an index i such that both:

(a) Finish[i] == false

(b) Request i ≤ Work

If no such i exists, go to step 4

Step 3 : Work = Work + Allocation i

Finish[i] = true, go to step 2

Step 4 : If Finish[i] == false, for some i, $1 \leq i \leq n$,

Then the system is in deadlock state.

Moreover, if Finish[i] == false, then P_i is deadlocked

Example 6: To illustrate this algorithm, we consider a system with five processes P_0 through P_4 and three resource types A, B, C. Resource type A has 7 instances, resource type B has 2 instances, and resource type C has 6 instances. Suppose that at time T_0, we have the following resource allocation state.

Process	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

Solution:

- We claim that the system is not in deadlocked state. Indeed if we execute our algorithm, we will find that sequence $<P_0, P_2, P_3, P_1, P_4>$ will result in, Finish[i] = true for all i.
- Suppose now that process P_2 makes one additional request for an instance of type C.
- The request matrix is modified as follows:

Process	Request		
	A	B	C
P_0	0	0	0
P_1	2	0	2
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

- We claim that the system is now deadlocked. Although we can reclaim the resources held by process P_0, but the number of available resources is not sufficient to fulfill the request of other processes.
- Thus, a deadlock exists, consisting of processes P_1, P_2, P_3 and P_4.

Example 7: Apply the deadlock detection algorithm to the following data and show the results.

$$\text{Available} = (2, 1, 0, 0)$$

$$\text{Request} = \begin{matrix} P_1 \\ P_2 \\ P_3 \end{matrix}\begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

$$\text{Allocation} = \begin{matrix} P_1 \\ P_2 \\ P_3 \end{matrix}\begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Solution:

Step 1: $\text{Work} = \text{Available}$

$= (2, 1, 0, 0)$

Step 2: $\text{Request} \leq \text{Work}$

In this Step 2 process P_3 fulfill the condition.

$(2, 1, 0, 0) \leq (2, 1, 0, 0)$

Step 3:

$\text{Work} = \text{Work} + \text{Allocation}$

$\text{Work} = (2, 1, 0, 0) + (0, 1, 2, 0)$

$= (2, 2, 2, 0)$

- In this way, we can finish the execution of P_3 Now we have (2, 2, 2, 0) available resources.
- We have to repeat the procedure and finishes the execution of processes and here we will find that $<P_3, P_2, P_1>$ will result in finish[i] = true for all i.

6.4.4 Recovery from Deadlock [S-18, 22; W-22]

- There are three basic approaches to recovery from deadlock:
  1. Inform the system operator, and allow him/her to take manual intervention.
  2. Terminate one or more processes involved in the deadlock
  3. Preempt resources.

6.4.4.1 Process Termination

- To eliminate deadlocks by aborting a process we use one of the two methods. In both methods, the system retains all resources allocated to the terminated processes.
  - **Abort all deadlocked processes:** This method clearly will break the deadlock cycle, but at a great expense, these processes may have computed for a long time, and the result of these partial computations must be discarded and probably recomputed later.
  - **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 of file, terminating it will leave that file in an incorrect state. Similarly, if the process was in the middle of printing data on the printer, the system must reset the printer to a correct state before printing the next job.
- If the partial termination method is used, then given a set of deadlocked processes, we must determine which process should be terminated in an attempt to break the deadlock.
- This determination is a policy decision, similar to CPU scheduling problems. The question is basically an economic one we should abort those processes termination of which will incur the minimum cost. Unfortunately, the term minimum cost is not a precise one.
- 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 (Example whether the resources are simple to preempt)?
  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?

6.4.4.2 Resource Preemption [S-18, W-22]

- When preempting resources to relieve deadlock, there are three important issues to be addressed:

1. **Selecting a victim:** Deciding which resources to preempt from which processes involves many of the same decision criteria outlined above.
2. **Rollback:** Ideally one would like to roll back a preempted process to a safe state prior to the point at which that resource was originally allocated to the process. Unfortunately it can be difficult or impossible to determine what such a safe state is, and so the only safe rollback is to roll back all the way back to the beginning. (i.e. abort the process and make it start over.)
3. **Starvation:** How do you guarantee that a process will not starve because its resources are constantly being preempted? One option would be to use a priority system, and increase the priority of a process every time its resources get preempted. Eventually it should get a high enough priority that it won't get preempted any more.

Solved Examples of Previous Exams

Example 8: Consider the fine processes P_0, P_1, P_2, P_3, P_4 and three resources R_1, R_2, R_3 resource, type R_1 has 10 instances, R_2 has 5 instances and R_3 has 7 instances. Allocation and max matrix is given below:

Process	Allocation			MAX		
	R_1	R_2	R_3	R_1	R_2	R_3
P_0	0	1	0	7	5	3
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

Answer the following questions using Banker's algorithm.

(i) What is the content of Need Matrix?

(ii) Is the system in a safe sequence? If yes, give the safe sequence.

Solution: (i) Need [i, j] = Max [i, j] – Allocation [i, j]

So, the content of Need Matrix is:

Process	Need Matrix		
	R_1	R_2	R_3
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

(ii) Then, find safe sequence. Currently (3, 3, 2) resources available and need of resources of each process is given in Need matrix.

Need P_0 <= Work (Available Resources)

So Need P_0 (7, 4, 3) <= (3, 3, 2)

Which is not equal so available resources will not be allocated to P_0.

So Safe sequence = {}

Need P_1 <= Work (Available Resources)

So Need P_1(1, 2, 2) <= (3, 3, 2)

So resources will be allocated to p_1. After successful execution of P_1. Allocated resources of P_1 will be added to available resources.

Available Resources = Allocation of P_1(2, 0, 0)
+ Available Resources (3, 3, 2) = (5, 3, 2)

Safe Sequence = {P_1}

Again Need P_2 <= Work (Available Resources)

So Need P_2 (6, 0, 0) <= (5, 3, 2)

Resources which are required by P_2 are greater than available resources.

So resources will not be given to P_2 process.

Again Need P_3(0, 1, 1) <= (5, 3, 2)

So resources will be allocated to P_3. After completion of P_3 the allocated resources of P_3 will be added to available resources.

So available resources = (7, 4, 3)

Safe Sequence = {P_1, P_3}

Again Need P_4(4, 3, 1) <= (7, 4, 3)

So resources will be allocated to P_4. After completion of P_4 the allocated resources of P_4 will be added to available resources.

So available resources = (7, 4, 5)

Safe Sequence = {P_1, P_3, P_4}

Again Need P_0(7, 4, 3) <= (7, 4, 5)

So resources will be allocated to P_0. After completion of P_0 the allocated resources of P_0 will be added to available resources.

So Available resources = (7, 5, 5)

Safe Sequence = {P_1, P_3, P_4, P_0}

Again Need P_2(6, 0, 0) <= (7, 5, 5)

So resources will be allocated to P_2. After completion of P_2 the allocated resources of P_2 will be added to available resources.

So Available resources = (10, 5, 7)

Safe Sequence = {P_1, P_3, P_4, P_0, P_2}

So if we follow this safe sequence the system will be in safe state and there will be no deadlock.

Example 9: Consider the five processes P_0, P_1, P_2, P_3, P_4 and three resources R_1, R_2, R_3 resource. Type R_1 has 8 instances, R_2 has 4 instances and R_3 has 9 instances. Allocation and max matrix is given below:

Process	Allocation			Max.		
	R_1	R_2	R_3	R_1	R_2	R_3
P_0	1	0	2	5	4	2
P_1	2	1	1	3	2	2
P_2	2	0	3	8	0	4
P_3	1	1	2	2	2	2
P_4	0	1	0	5	2	3

Answer the following questions using Banker's algorithm:

(i) What is the content of need matrix?

(iii) Is the system in a safe sequence? If yes, give the safe sequence.

Solution: First we calculate available matrix.

Process	Current Allocation			Max.					
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	1	0	2	5	4	2	2	1	1
P_1	2	1	1	3	2	2			
P_2	2	0	3	8	0	4			
P_3	1	1	2	2	2	2			
P_4	0	1	0	5	2	3			

(i) Then, the Need matrix will be,

Process	Need Matrix		
	R_1	R_2	R_3
P_0	4	4	0
P_1	1	1	1
P_2	6	0	1
P_3	1	1	0
P_4	5	1	3

(ii) Then, find safe sequence. Currently (2, 1, 1) resources available and need of resources of each process is given in Need matrix.

Need P_0 <= Work

So Need P_0 (4, 4, 0) <= (2, 2, 1)

Which are not equal so available resources will not be allocated to P_0.

Need P_1 <= Work

So Need P_1(1, 1, 1) <= (2, 2, 1)

So resources will be allocated to P_1. After successful execution of P_1, allocated resources of P_1 will be added to available resources.

Available Resources = Allocation of P_1(2, 1, 1) + Available Resources (2,1,1)

= (4, 2, 2)

Safe Sequence = $\{P_1\}$

Again Need P_2 <= Work

So Need P_2 (6, 0, 1) <= (4, 2, 2)

Which are not less so resources will not be allocated to P_2.

Need P_3(1, 1, 0) <= (4, 4, 2)

So resources will be allocated to P_3. After completion of P_3 the allocated resources of P_3 will be added to available resources.

So available resources = (5, 3, 4)

Safe Sequence = $\{P_1, P_3\}$

Again Need P_4(5, 1, 3) <= (5, 3, 4)

So resources will be allocated to P_4. After completion of P_4 the allocated resources of P_4 will be added to available resources.

So available resources = (5, 4, 4)

Safe Sequence = $\{P_1, P_3, P_4\}$

Again Need P_0(4, 4, 0) <= (5, 4, 4)

So resources will be allocated to P_0. After completion of P_0 the allocated resources of P_0 will be added to available resources.

So available resources = (6, 4, 6)

Safe Sequence = $\{P_1, P_3, P_4, P_0\}$

Again Need P_2(6, 0, 1) <= (6, 4, 6)

So resources will be allocated to P_2. After completion of P_2 the allocated resources of P_2 will be added to available resources.

So available resources = (8, 4, 9)

Safe Sequence = $\{P_1, P_3, P_4, P_0, P_2\}$

So if we follow this safe sequence the system will be in safe state and there will be no deadlock.

Summary

- In an operating system, deadlock state happens when two or more processes are waiting for the same event to happen which never happen, then we can say that those processes are involved in the deadlock.
- A deadlock can occur only if four necessary conditions hold simultaneously in the system: Mutual Exclusion, Hold and Wait, No Preemption, and Circular Wait.
- Methods for handling deadlock: Deadlock prevention, Deadlock avoidance, Deadlock detection and recovery.

- A method for avoiding deadlocks, rather than preventing them, requires that the operating system have a priori information about how each process will utilize system resources.
- The banker's algorithm, for example, requires a priori information about the maximum number of each resource class that each process may request. Using this information, we can define a deadlock-avoidance algorithm.
- If a system does not employ a protocol to ensure that deadlocks will never occur, then a detection-and-recovery scheme may be employed.
- Where preemption is used to deal with deadlocks, three issues must be addressed: Selecting a victim, Rollback, and Starvation.

Check Your Understanding

1. What is a reusable resource?
   (a) that can be used by one process at a time and is not depleted by that use.
   (b) that can be used by more than one process at a time.
   (c) that can be shared between various threads.
   (d) none of the mentioned.
2. Which of the following condition is required for a deadlock to be possible?
   (a) mutual exclusion
   (b) a process may hold allocated resources while awaiting assignment of other resources.
   (c) no resource can be forcibly removed from a process holding it.
   (d) all of the mentioned.
3. A system is in the safe state if ____.
   (a) the system can allocate resources to each process in some order and still avoid a deadlock.
   (b) there exist a safe sequence.
   (c) all of the mentioned
   (d) none of the mentioned
4. The circular wait condition can be prevented by ____.
   (a) defining a linear ordering of resource types
   (b) using thread
   (c) using pipes
   (d) all of the mentioned
5. Which one of the following is the deadlock avoidance algorithm?
   (a) banker's algorithm (b) round-robin algorithm
   (c) elevator algorithm (d) Karn's algorithm
6. What is the drawback of banker's algorithm?
   (a) in advance processes rarely know how much resource they will need.
   (b) the number of processes changes as time progresses.
   (c) resource once available can disappear.
   (d) all of the mentioned

7. For an effective operating system, when to check for deadlock?
   (a) every time a resource request is made.
   (b) at fixed time intervals.
   (c) every time a resource request is made at fixed time intervals.
   (d) none of the mentioned
8. A problem encountered in multitasking when a process is perpetually denied necessary resources is called _____.
   (a) deadlock (b) starvation
   (c) inversion (d) aging
9. Which one of the following is a visual (mathematical) way to determine the deadlock occurrence?
   (a) resource allocation graph (b) starvation graph
   (c) inversion graph (d) none of the mentioned
10. To avoid deadlock _____.
   (a) there must be a fixed number of resources to allocate.
   (b) resource allocation must be done only once.
   (c) all deadlocked processes must be aborted.
   (d) inversion technique can be used.

Answers

1. (a)	2. (d)	3. (a)	4. (a)	5. (a)	6. (d)	7. (c)	8. (b)	9. (a)	10. (a)

Practice Questions

Q.I Answer the following questions in short:

1. What is meant by deadlock?
2. What are the principles of deadlock?
3. What are the necessary conditions for deadlock occurs.
4. List the methods for deadlock handling.
5. What is safe sequence?
6. What is claim age?
7. What is mean by request edge?

Q.II Answer the following questions:

1. Describe safe state in detail.
2. Describe banker's algorithm with suitable example.
3. Explain the term resource allocation graph with example.
4. Explain working of Banker's algorithm for deadlock avoidance with suitable example.
5. Write and explain deadlock detection algorithm with suitable example.
6. Describe deadlock characterization in detail.
7. Explain resource allocation graph with suitable example.
8. Enlist various deadlock handling methods in short.
9. Write short note on Recovery from deadlock.

10. For the following resource allocation graph, show that four necessary conditions for deadlock indeed hold.

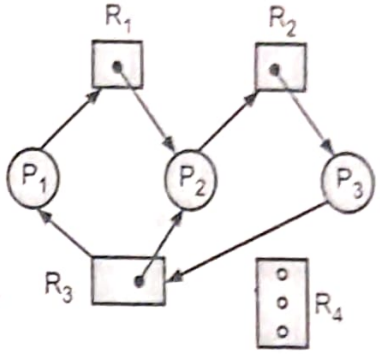


Fig. 6.10

11. Consider the system with 5 processes P = {P_0, P_1, P_2, P_3, P_4} and four resource types {A, B, C, D}. There are 3 instances of type A, 10 instances of type B, 15 instances of type C and 7 instances of type D. The allocation and maximum demand matrix are as follows:

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

Answer the following question using Banker's Algorithm:

(i) Is the system in a Safe State?

(ii) If a request from process P_4 arrives for (0, 2, 0, 2) can it be granted?

12. Consider the system with 5 processes P = {P_0, P_1, P_2, P_3, P_4} and four resources type {A, B, C, D}. There are 3 instances of type A, 14 instances of type B, 12 instances of type C and 12 instance of type D. The allocation and maximum demand matrix are as follows:

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

Answer the following question using Bankers Algorithm:

(i) Is the system in a Safe State?

(ii) If a request from process P_4 arrives for (0, 0, 4, 1) can be the request immediately granted.

Q.III Define terms:

1. Process termination.
2. Resource preemption.
3. Assignment edge in RAG.
4. Request edge in RAG.
5. Deadlock avoidance.

Previous Exam Questions

Summer 2018

1. Define Rollback. **[2 M]**

Ans. Refer to Section 6.4.4.2.

2. Explain different methods for recovery from a deadlock. **[4 M]**

Ans. Refer to Section 6.4.4.

3. Explain Resource Allocation graph in detail. **[4 M]**

Ans. Refer to Section 6.4.2.2.

Winter 2018

1. What is meant by Deadlock? **[2 M]**

Ans. Refer to Section 6.1.

2. Explain Resource Allocation Graph in detail. **[4 M]**

Ans. Refer to Section 6.4.2.2.

3. List and explain necessary conditions for Deadlock occurrence. **[4 M]**

Ans. Refer to Section 6.3.

Summer 2019

1. What is the role of MAX and NEED array used in Banker's Algorithm? **[2 M]**

Ans. Refer to Section 6.4.2.3.

2. Wait for graph is used for deadlock avoidance in the system. True/False. Justify. **[2 M]**

Ans. Refer to Section 6.4.3.1.

3. List and explain necessary conditions for deadlock. **[4 M]**

Ans. Refer to Section 6.3.