Methods for Handling Deadlocks

There are basically three methods to deal with the deadlock problem:

1. We can use a protocol to ensuring that the system will never enter a deadlock state.
2. We can allow the system to enter in deadlock state and then recover it.
3. We can ignore the problem altogether, and pretend that deadlocks will never occur in the system. This method is used in Windows and UNIX operating systems.

To implement the first method, we have two approaches:

- Deadlock Prevention
- Deadlock Avoidance

To implement the second method, we have two approaches:

- Deadlock Detection
- Deadlock Recovery

Deadlock Prevention

It is a set of methods for ensuring that at least one of the necessary conditions for deadlock cannot hold.

1. Mutual Exclusion

The mutual-exclusion condition is not required for sharable resources. For example, if several processes attempt to open a read-only file at the same time, they can be granted simultaneous access to the file. A process never needs to wait for a sharable resource.

The mutual-exclusion condition must hold for non sharable resources. For example, a printer cannot be simultaneously shared by several processes.

2. Hold and Wait

To ensure that the hold-and-wait condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources. For this two protocols can be used:

- Each process to request and be allocated all its resources before it begins execution.
- An alternative protocol allows a process to request resources only when the process has none.

These protocols have two main disadvantages.

- Resource Utilization: Since many of the resources may be allocated but unused for a long period.
- Starvation: A process that needs several popular resources may have to wait indefinitely, because at least one of the resources that it needs is always allocated to some other process.

3. No Preemption

To ensure that this condition does not hold, we can use the following protocol.

- If a process is holding some resources and requests another resource that cannot be immediately allocated to it then all resources currently being held are preempted(released) and the process is pre-empted to wait in waiting queue. The process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.
- If a process requests some resources, we first check whether they are available. If they are, we allocate them. If they are not available, we check whether they are allocated to

some other process that is waiting for additional resources. If so, we preempt the desired resources from the waiting process and allocate them to the requesting process. If the resources are not either available or held by a waiting process, the requesting process must wait. A process can be restarted only when it is allocated the new resources it is requesting and recovers any resources that were preempted while it was waiting.

4. Circular Wait

To ensure that this 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.
For example, if the set of resource types R includes tape drives, disk drives, and printers, then the ordering might be defined as follows:
F(tape drive) = 1,
F(disk drive) = 5,
F(printer) = 12.
Each process can request resources only in an increasing order of enumeration.

Deadlock Avoidance

Deadlock-prevention algorithms prevent deadlocks by restraining how requests can be made. A side effect of preventing deadlock by this method is possibly provide low device utilization and reduced system throughput. An alternative way for avoiding deadlocks is to require additional information about how resources are to be requested. There are various algorithms which differ in the amount and type of information required. The simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need before execution.
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.

Safe State

A state is said to be *safe state,* if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock. 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. A safe state is not a deadlock state. An unsafe state may lead to deadlock.

Question- Consider a system with 12 magnetic tape drives and **3** processes (P0, P1, and P2). Process P0 requires 10 tape drives, process P1 require 4, and process P2 may need 9 tape drives. Suppose that, at time t0, process P0 is holding 5 tape drives, process P1 is holding 2, and process P2 is also holding 2 tape drives.

- What is the process sequence?

  OR

- Check whether the system in safe state or not.

Resource-Allocation Graph Algorithm (Single instance of Resource)

In addition to the request and assignment edges a new type of edge, called a **claim edge** is used. A claim edge $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j at some time in the future. It is represented by a dashed line. The claim edge $P_i \rightarrow R_j$ is converted to a request

edge, when a process requests a resource. When a resource is released the assignment edge $R_j \rightarrow P_i$ is reconverted to a claim edge.

Suppose that process P_i requests resource R_j. The request can be granted only if converting the request edge $P_i \rightarrow R_j$ to an assignment edge $R_j \rightarrow P_i$ does not result in the formation of a cycle in the resource-allocation graph.

- If no cycle exists, then the allocation of the resource will leave the system in a safe state.
- If a cycle is found, then the allocation will put the system in an unsafe state.

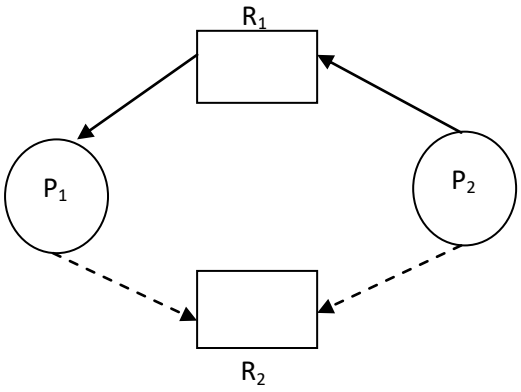

Fig. Resource-allocation graph for deadlock avoidance

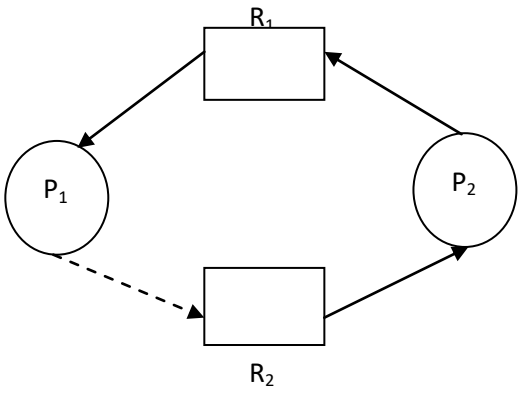

Fig An unsafe state in a resource-allocation graph

Banker's Algorithm (Multiple instance of resource)

The name was chosen because this algorithm could be used in a banking system to ensure that the bank never allocates its available cash such that it can no longer satisfy the needs of all its customers.

When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system. The system must determine whether the allocation of these resources will leave the system in a safe state. If it will, the resources are allocated; otherwise, the process must wait until some other process releases enough resources.

Data Structures for Banker's Algorithm

Let ***n*** be the number of processes in the system and ***m*** be the number of resource types. We need the following data structures:

- **Available**: A vector of length ***m*** indicates the number of available resources of each type. If *Available[j]* = k, there are k instances of resource type *Rj* available.
- **Max:** An ***n*** **x** ***m*** matrix defines the maximum demand of each process. If *Max[i,j]* = k, then process *Pi* may request at most ***k*** instances of resource type ***Rj***.
- **Allocation:** An ***n*** **x** ***m*** matrix defines the number of resources of each type currently allocated to each process. If *Allocation[i,j]* = k, then process ***Pi*** is currently allocated ***k*** instances of resource type *Rj*.
- **Need:** *An* ***n*** **x** ***m*** matrix indicates the remaining resource need of each process. If *Need[i,j]* = k, then process ***Pi*** may need ***k*** more instances of resource type *Ri* to complete its task.

$$Need[i,j] = Max[i,j] - Allocation[i,j]$$

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

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

- *Work = Available*
- *Finish[i] = false (*for *i* = 1,2, ..., *n)*

Step-2: Find an *i* such that both

- *Finish[i] =false*
- $Need_i \leq Work$.

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] = true* for all *i*, then the system is in a safe state

Question: Consider a system with five processes P_o,P_1,P_2,P_3,P_4 *and* three resource types *A, B, C.* Resource type *A* has 10 instances, resource type *B* has 5 instances, and resource type *C* has 7 instances. Suppose that, at time T_o, the following snapshot of the system has been taken:

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

Determine whether the system is in safe state or not.
Solution: *Refer you class notebook (Discussed during the lecture)*

Resource-Request Algorithm

This algorithm determines if request can be safely granted. Let $Request_i$ be the request vector for process P_i. If $Request_i[j] == k$, then process P_i wants ***k*** instances of resource type R_j. When a request for resources is made by process P_i, the following actions are taken:

Step-1: If $Request_i \leq Needi$, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.

Step-2: If $Requesti \leq Available$, go to step **3.** Otherwise, *Pi* must wait, since the resources are not available.

Step-3: Have the system pretend to have allocated the requested resources to process *Pi* by modifying the state as follows:

$Available = Available - Request_i$
$Allocation_i = Allocation_i + Request_i$
$Need_i = Need_i - Request_i$

If safe: The resources are allocated to P_i.
If unsafe: P_i must wait and the old resource-allocation state is restored.

Deadlock Detection

For detection of deadlock, the system must provide:

- An algorithm that examines the state of the system to determine whether a deadlock has occurred or not.
- An algorithm to recover from the deadlock

We can perform deadlock detection by two methods:

1. Single Instance of Each Resource Type

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

A deadlock exists in the system if and only if 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. **An** algorithm to detect a cycle in a graph requires an order of n^2 operations, where *n* is the number of vertices in the graph.

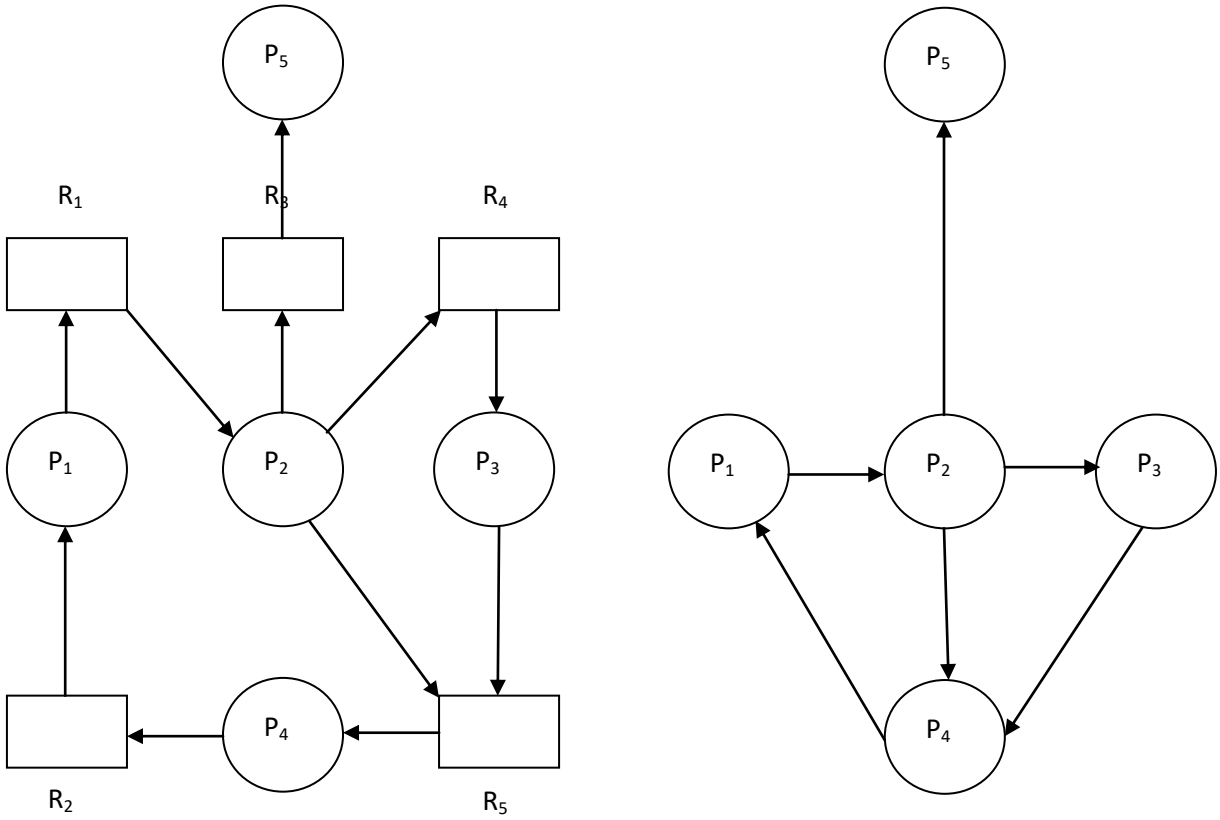

Figure: (a) Resource-allocation graph. (b) Corresponding wait-for graph.

2. 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. For this deadlock-detection algorithm is used. It uses three data structures:

- **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 *Ri.*

Algorithm

Step-1: Let *Work* and *Finish* be vectors of length *m* and *n,* respectively. Initialize *Work = Available.* For *i* = 1, 2, …, *n,* if $Allocation_i \neq 0$, then *Finish[i]=false;* otherwise, *Finish[i]= true.*

Step-2: Find an index *i* such that both
a. *Finish[i] ==false.*
b. $Request_i \leq Work.$
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,* $0 \leq i \leq n$, then the system is in a deadlock state. Moreover, if *Finish[i] ==false,* then process *Pi* is deadlocked.

This algorithm requires an order of *m* x n^2 operations to detect whether the system is in a deadlocked state.

Example: Consider a system with five processes P_o through P_4 and three resource types *A, B, C. A* has 7 instances, *B* has 2 instances, and resource type *C* has *6* instances. Suppose at time T_0 :

	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			

The above system is not in a deadlocked state. The sequence <P_0, P_2, P_3, P_1, P_4> will result in *Finish[i] ==* true for all *i.*

Now consider another instance

	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	1			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

The above system is now deadlocked because the number of available resources is not sufficient to fulfill the requests of the other processes. Thus, a deadlock exists, consisting of processes P_1, P_2, P_3, and P_4.

Recovery from Deadlock

When a detection algorithm determines that a deadlock exists, then we have to break deadlock. There are two options for breaking a deadlock:

- Abort one or more processes to break circular wait.
- Preempt some resources from one or more deadlock processes

1. Process Termination

To eliminate deadlocks by aborting a process, we use one of two following methods:

- **Abort all deadlocked processes:** This method will break the deadlock cycle, but at a great expense; since these processes may have computed for a long time, and the results of these partial computations must be discarded and recomputed later.
- **Abort one process at a time until the deadlock cycle is eliminated:** Here we 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 or not.

  It also include some decision making process i.e. which process (or processes) should be terminated. Solution for this is to abort those processes, termination of which will incur the minimum cost. There are many cost factors such as:

  1. Priority of the process
  2. How long the process has computed, and how much longer to compute
  3. Resources the process has used
  4. Resources the process needs to complete

2. 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. While doing resource preemption we need to consider following three issues:

- **Selecting a victim:** As in process termination, we must determine the order of preemption to minimize cost.
- **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.
- **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, a starvation situation occurred. This starvation situation need to be solved for which 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.

Questions asked in semester exam:

Question: Is it possible to have a deadlock involving only a single process? Explain.

[2017-2018][7 Marks]

Question: Consider the following snapshot of a system:

[2017-2018][2018-2019][7 Marks]

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P1	2	2	3	3	6	8	7	7	10
P2	2	0	3	4	3	3			
P3	1	2	4	3	4	4			

Answer the following questions using the Banker's algorithm:

(i) What is the content of the matrix need?

(ii) Is the system in safe state?

Question: Discuss the usage of wait-for graph method.

[2017-2018][2 Marks]

Question: What is a deadlock? Discuss the necessary conditions for deadlock with Examples
[2016-2017] [7.5 Marks]

Question: What is a deadlock? Discuss the necessary conditions for deadlock with examples.
[2015-2016] [10 Marks]

Question: Describe Banker's algorithm for safe allocation. [2016-2017] [7.5 Marks]

Question: Consider the following snapshot of the system:- [2015-2016] [10 Marks]
[2014-2015] [10 Marks]

	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				

Answer the following questions using the banker's algorithm.

(i) Compute the total number of resources of each type.
(ii) What is the content of the matrix Need?
(iii) Is the system in a safe state? If yes then find the safe sequence.
(iv) If a request from process P_1 arrives for (0, 4, 2, 0) can the request be granted immediately?

Question: Describe the Banker's algorithm for safe allocation. Consider a system with five processes and three resource types and at time 'T_0' the following snapshot of the system has been taken: [2015-2016] [10 Marks]
[2013-2014] [10 Marks]

Process Id	Allocated			Maximum			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	1	1	2	4	3	3	3	1	0
P2	2	1	2	3	2	2			
P3	4	0	1	9	0	2			
P4	0	2	0	7	5	3			
P5	1	1	2	11	2	3			

(i) Determine the total amount of resources of each type.
(ii) Compute the Need matrix
(iii) Determine if the state is safe or not using Banker's algorithm
(iv) Would the following request be granted in the current state?
  (a) P1 <3, 3, 1>
  (b) P2<2, 1, 0>

Question: Describe Banker's algorithm for deadlock avoidance. Consider a system with three processes and three resources. The snapshot of a system at time t_0 is given below:
[2014-2015] [10 Marks]

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P1	2	2	3	3	6	8	7	7	10
P2	2	0	3	4	3	3			

P3	1	2	4	3	4	4			

(iii) Is the current allocation in safe state?

(iv) Would the following requests be granted in the current state?

(a) Process P2 requests<1, 0, 0>

(b) Process P1requests <1, 0, 0>

Question: Consider the following snapshot of a system [2014-2015] [5 Marks]

Answer the following questions using the banker's algorithm

(i) What is the content of the matrix Need?

(ii) Is the system in a safe state?

(iii) If a request from process P2 arrives for (2, 1, 0). Can the request be granted immediately?

Process	Allocation			Max			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	2	2	3	3	6	8	4	3	10
P2	2	0	3	4	3	3			
P3	1	2	4	3	4	4			

Question: Explain Banker's algorithm. Why this is used? [2014-2015] [10 Marks]

Question: What is deadlock? Explain the various characteristics of a deadlock in detail. Discuss deadlock prevention schemes. [2014-2015] [10 Marks]

Question: What are deadlocks? Explain the different methods for dealing with the deadlocks. [2014-2015] [5 Marks]

Question: What are the necessary conditions to hold a deadlock in a system? [2013-2014] [5 Marks]

Question: What are the approaches that can be used for Prevention of deadlock? [2013-2014] [5 Marks]

Question: Define deadlock. List four necessary conditions for occurrence of deadlock. A system contains 6 units of resource and n processes that use the resource. What is the maximum value of n for which the system will be deadlock free if the maximum requirement of each process is 3? [2012-2013] [10 Marks]

Question: In a system, n processes share m resource units that can be reserved and released only one at a time. The maximum need of each process does not exceed m and sum of all maximum need is less than m + n. Show that a deadlock cannot occur. [2011-2012] [5 Marks]

Question: Write and explain Banker's algorithm for avoidance of deadlock. [2011-2012] [10 Marks]

Question: Describe the necessary condition for deadlock to occur. [2010-2011] [5 Marks]

Question: In the respect of Banker's Algorithm discuss whether system is safe or unsafe. If a system is safe, show how it is possible for all users to complete: [2010-2011] [5 Marks]

	Current loan	Maximum Need
User (1)	2	6
User (2)	4	7
User (3)	5	6
User (4)	0	2
Available	1	

Question: Explain the different conditions of deadlock. [2009-2010] [5 Marks]

Question: Write down the methods for deadlock prevention. [2009-2010] [5Marks]

Question: How the recovery from deadlock is done using combined approach? [2009-2010] [5 Marks]

Question: What is deadlock and its conditions? [2008-2009] [5 Marks]

Question: How dead lock can be avoided? [2008-2009] [5 Marks]

Question: Write short notes on: [2007-2008] [10 Marks]

(i) Resource allocation graph and resource allocation graph algorithm.

(ii) Recovery from deadlock.

Question: What are necessary conditions to hold a deadlock in a system? Explain the resource allocation graph algorithm to deal with deadlock problem. What are the limitations of this approach? [2006-2007] [10 Marks]

Question: Differentiate between Deadlock Prevention and Avoidance [2006-2007] [5 Marks]