6

UNIT- V

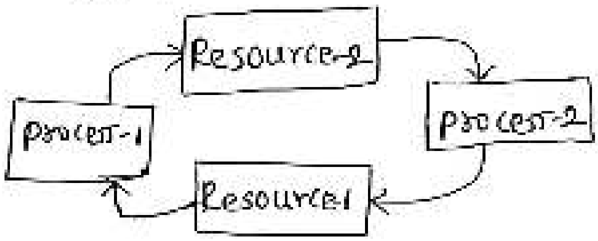
Deadlocks - System model, Deadlock characterization, methods For Handling deadlocks, deadlock prevention, Deadlock avoidance Deadlock dection, and Recovery from Deadlock.

Protection - system protection, goals of protection, principles of protection, Domain of protection, Access matrix, implementation of Access matrix, Access control, Revocation of access Rights capability - based systems, Language - based protection.

Deadlock:

Deadlock is a situation where the execution of two or more processes is blocked because each process holds some resource and waits for another resource held by some other process.

Eg:-



→ here process P_1 holds resource R_1 and waits for resource R_2 which is held by process P_2.

→ process P_2 holds resource R_2 and waits for resource R_1 which is held by process P_1

→ None of the two processes can complete and release their resource.

→ Thus, both the processes keep waiting infinitely, this situation we can call it as deadlock.

In multiprogramming system, process get completed for a finite number of resource. any process requests resources, and that resources are not available at that time, the process goes into a waiting state

System model:

→ A system model or structure consists of a fixed number of resources to be circulated among some processes. The resources are then partitioned into numerous types, each partition consisting of some specific quantity of identical instances.

→ Memory space, cpu cycles, directories and files, I/O devices like keyboards, printers and CD-DVD drives are examples of resource types.

Under the standard mode of operation, any process may use a resource in only the below mentioned sequence.

Request: When the request can't be approved immediately, then the requesting job must remain waited until it can obtain the resource

use: The process can run on the resource like printer etc.

Release: The process release the resource

Deadlock characterization

There are following 4 necessary conditions for the occurrence of deadlock

- Mutual Exclusion
- Hold and wait
- No pre-emption
- Circular wait

Mutual Exclusion: There must exist at least one resource in the system which can be used by only one process at a time. if there exists no such resource, then deadlock will never occur.

printer is an example of a resource that can be used by only one process at a time.

Hold and wait: There must exist a process which holds some resource and waits for another resource held by some other process.

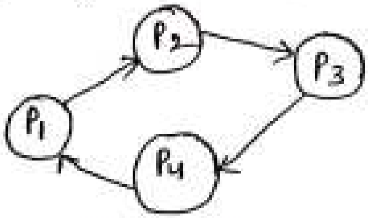
No pre-emption: By this condition, once the resource has been allocated to the process, it can not be preempted.

→ it means resource can not be snatched forcefully from one process and given to the other process.

→ The process must release the resource voluntarily by itself.

Circular wait:

In this condition all the processes must wait for resource in a cyclic manner where the last process waits for the resource held by the first process.



→ process P_1 waits for a resource held by process P_2

→ process P_2 waits for a resource held by process P_3

→ process P_3 waits for a resource held by process P_4

→ process P_4 waits for a resource held by process P_1.

All these four conditions must hold simultaneously for the occurrence of deadlock. If any of these conditions fail, then the system can be ensured deadlock free.

Resource Allocation graph (RAG):

→ Deadlocks can be described in terms of a directed graph called a system resource allocation graph.

→ This graph consists 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: {$P_1, P_2 \cdots P_n$}, R: {$R_1, R_2, \cdots R_m$}

→ A directed edge from process P_i to resource type R_j is denoted by $P_i \to R_j$ & Resource to process $R_j \to P_i$.

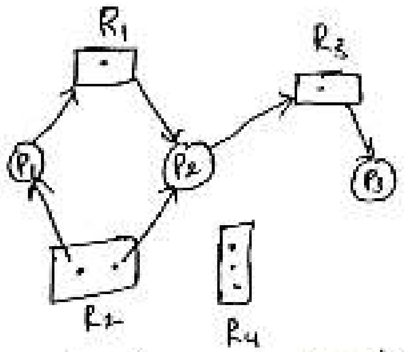
→ A directed edge $P_i \to R_j$ is called a request edge, a directed edge $R_j \to P_i$ is called an assignment edge.

The sets P, R and E

P= {P_1, P_2, P_3}

R= {R_1, R_2, R_3, R_4}

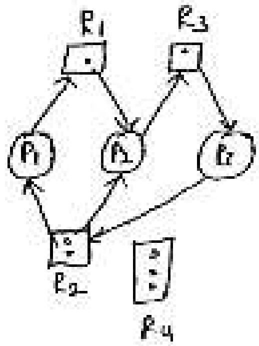
E= {$P_1 \to R_1, P_2 \to R_3, R_1 \to P_2, R_2 \to P_2, R_2 \to P_1, R_3 \to P_3$}



Resource allocation graph

→ In above graph, there is no cycles then no process in the system is deadlocked. if the graph does contain a cycle, then a deadlock may exist.

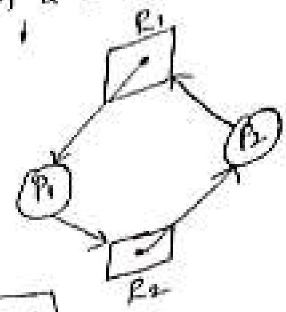
→ process P_1, P_2 and P_3 are deadlocked.
Process P_2 is waiting for resource R_3, which is held by process P_3. & P_3 is waiting for P_1 or P_2 to release R_2.
P_1 is waiting for P_2 to release R_1
and we also have a cycle.



RAG with deadlock

$P_1 \to R_1 \to P_2 \to R_2 \to P_1$

Eg: consider the RAG and find if the system is in a deadlock state Otherwise find a safe sequence.



→ The given graph is single instance with a cycle thus, the system is definitely in a deadlock state

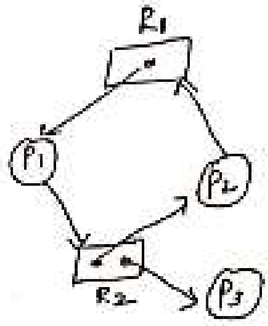
→ There are no instances available currently and both the processes require a resource to execute.

→ none of the process can be executed and both keeps waiting infinitely so system in a deadlock state.

Process	Allocation		Need	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0

available - $[R_1 R_2] = [0\ 0]$

Eg2:



→ The given graph is multi instance with a cycle contained in it. so, the system may or may not be in deadlock state

→ process P_3 does not need any resource so it executes, after execution, process P_3 release its resources.

Then available $= [0\ 0] + [0\ 1]$
$= [0\ 1]$

Process	Allocation		Need	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0
P_3	0	1	0	0

available $[R_1\ R_2] = [0\ 0]$

→ P_1 is allocated the resource, then P_1 completes its execution & release the resource

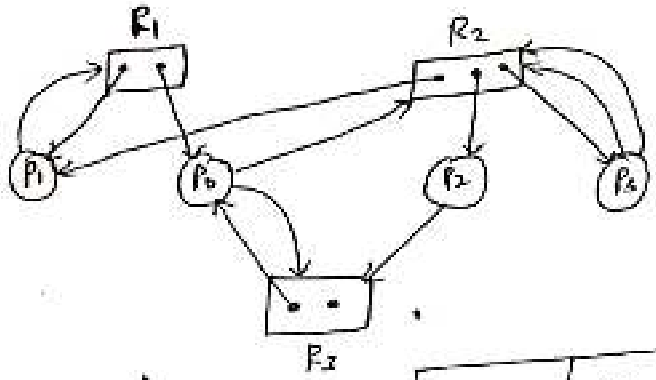
available $= [0\ 1] + [1\ 0]$
$= [1\ 1]$

→ next P_2 is allocated the requested resource & completes its execution and release the resource.

available $= [1\ 1] + [0\ 1]$
$= [1\ 2]$

safe state is $P_3 \rightarrow P_1 \rightarrow P_2$

Eg:



→ The given RAG is multi instance with a cycle. so the system may or may not be in a deadlock state.

→ only process P_2 completes its execution and release the resource.

Process	Allocation			Need		
	R_1	R_2	R_3	R_1	R_2	R_3
P_0	1	0	1	0	1	1
P_1	1	1	0	1	0	0
P_2	0	1	0	0	0	1
P_3	0	1	0	0	2	0

available $[R_1\ R_2\ R_3]$ = [0 0 1]

available = [0 0 1] + [0 1 0]
= [0 1 1]

→ with current available resources P0 can satisfied, so P0 complete the execution and Release the resource

available = [0 1 1] + [1 0 1]
= [1 1 2]

→ with current available resource P_1 can satisfied, so P_1 complete the execution and Release the resource.

available = [1 1 2] + [1 1 0]
= [2 2 2]

→ next P_3 satisfied the available resource

available = [2 2 2] + [0 1 0]
= [2 3 2]

so the system is in safe state, & safe state is

$P_2 \rightarrow P_0 \rightarrow P_1 \rightarrow P_3$

Methods for Handling Deadlocks:

Normally you can deal with the deadlock issues and situations in one of the three ways mentioned below.

→ you can use a protocal to prevent or avoid deadlocks, ensuring that the system will never enter a deadlock state.

→ We can allow the system to enter a deadlocked state, detect it and recover.

→ we can ignore the problem altogether and pretend that deadlocks never occur in the system.

Deadlock handling strategies
- Deadlock prevention
- Deadlock avoidance
- Deadlock Dection & Recovery
- Deadlock ignorance (ostrich method)

Deadlock prevention:

→ It involves designing a system that violates one of the four necessary conditions required for the occurrence of deadlock.

→ This ensures that the system remains free from the deadlock.

The various conditions of deadlock occurrence may be violated as

Mutual Exclusion, Hold and wait, No pre-emption & circular wai

Mutual Exclusion:

→ The mutual-exclusion condition must hold for nonsharable resources. eg:- printer cannot be simultaneously shared by several processes.

→ sharable resources cannot require mutual exclusion and thus cannot be involved in a deadlock.

Eg: If several processes attempt to open a read only file at the same time, they can be granted simultaneously access the file. and a process never need to wait for a sharable resources.

Hold and wait:
This condition can be violated in the following ways

case1:
→ A process has to first request for all the resources it requires for execution. once it has acquired all the resources, only then it can start its execution.
→ In this case, it ensures that the process does not hold some resources and wait for other resources.
→ The main drawback is, it is less efficient, and it is not implementable since it is not possible to predict in advance which resource will be required being execution.

case2:
→ A process is allowed to acquire the resources it desire at the current moment. after acquiring the resources, it starts execution.
→ Now before making any new request, it has to compulsorily release all the resources that it holds currently.
→ This case is efficient & implementable

case3:
→ A timer is set after the process acquires any resource.
→ after the timer expires, a process has to compulsorily release the resource.

No preemption:
→ This condition can be violated by forceful preemption
→ A process is holding some resources and request other resources that can not be immediately allocated it.

Circular wait:

→ This condition can be violated by not allowing the processes to wait for resource in a cyclic manner.

→ To violate this condition, the following steps considered.

→ A natural number is assigned to every resource.

→ Each process is allowed to request for the resources either in only increasing or only decreasing order of the resource number.

→ In case increasing order is followed, if a process requires a lesser number resource, then it must release all the resources having larger number and vice versa.

→ this approach is most practical & implementable.

→ this approach may cause starvation but will never lead to deadlock.

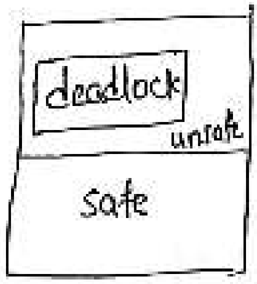
Deadlock avoidance:

In deadlock avoidance each resource is carefully analyzed to see whether it could be safely fulfilled without causing deadlock. the drawback of this method is its requirement of information in advance about how resources are to be requested.

safe state:

A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock. a system is safe then there exists a safe sequence.

Sequence of process <P1,P2.. Pn> is a safe sequence for the current allocation state. a deadlocked state is unsafe state. in unsafe state there is no safe sequence.



Banker's Algorithm:

→ Banker's algorithm is a deadlock avoidance strategy.

→ Whenever a new process is created, it specifies the maximum number of instances of each resource type that it exactly needs.

→ To implement banker's algorithm, following four data structures are used.

- available
- Max
- Allocation
- Need

available: it is a single dimensional array that specifies the number of instances of each resource type currently available.

Eg:- available [R1] = k

Max: it is a two dimensional array that specifies the maximum number of instances of each resource type that a process can request.

Eg:- max [P1] [R1] = k

Allocation: it is a two dimensional array that specifies the number of instances of each resource type that has been allocated to the process.

eg:- Allocation [P1] [R1] = k

Need: it is a two dimensional array that specifies the number of instances of each resource type that a process requires for execution.

Eg:- Need [P1] [R1] = k

→ Banker's algorithm is executed whenever any process puts forward the request for allocating the resources.

→ it achecks whether the request made by the process is valid or not

Eg: consider the following snapshot with Resource types A, B, C. Resource type A has ten instances, resource type B has five instances and resource type C has seven instances. then find Need matrix and safe sequence.

	Allocation A B C	max A B C	Available A B C	Need A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2	5 3 2	1 2 2
P_2	3 0 2	9 0 2	7 4 3	6 0 0
P_3	2 1 1	2 2 2	7 4 5	2 1 1
P_4	0 0 2	4 3 3	7 5 5	5 3 1
	7 2 5		10 5 7	

step1: find the allocated Resources 7 2 5 (A B C).

subtract the allocated Resources from total Resources.

A B C Total Resources
10 5 7
7 2 5 allocated Resources

3 3 2 - available resources.

Need = max - available

if Need ≤ available then execute the particular process

Otherwise move to next process.

~~Eeq~~ and safe sequence is $P_1 \to P_3 \to P_4 \to P_0 \to P_2$

Eg2:

	Allocation A B C	max A B C	available A B C	Need A B C
P_0	1 0 1	4 3 1	3 3 0	3 3 0
P_1	1 1 2	2 1 4	4 3 1	1 0 2
P_2	1 0 3	1 3 3	5 3 4	0 3 0
P_3	2 0 0	5 4 1	6 4 6	3 4 1
			8 4 6	

Safe sequence is $P_0 \to P_2 \to P_1 \to P_3$

→ A request is valid if and only if the number of requested instances of each resource type is less than the need declared by the process in the beginning. If the request is invalid, it aborts the request.

→ if the request is valid, it checks if the number of requested instances of each resource type is less than the number of available instances of each type.

→ if the sufficient number of instances are not available, it asks the process to wait longer.

→ if the sufficient number of instances are available, the requested resources have been allocated to the process.

Available = available - Request [i]

Allocation [i] = Allocation[i] + Request [i]

Need [i] = Need[i] - Request [i]

→ Now, banker's Algorithm follows the safety algorithm to check whether the resulting state it has entered in is a safe state or not

→ if it is a safe state, then it allocates the requested resources to the process in actual.

→ if it is an unsafe state, it asks the process to wait longer.

2017. Safety Algorithm:

→ 1. Let work and finish be vectors of length m & n

initialize work = available & finish[i] = false for i = 0, 1, .. n-1.

→ 2. finish [i] = false ; $Need_i \leq$ work go to step 4

→ 3. work = work + $Allocation_i$

finish[i] = true go to step 2

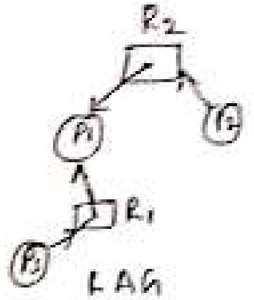
→ 4. if finish[i] = = true for all i, then the system is in safe sate.

→ it requires $m \times n^2$ operations to determine whether a state is safe

Deadlock Dection:

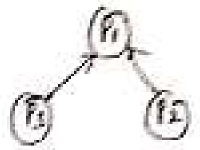
→ Deadlock detection can be done by using the following methods
wait for graph, Banker's Algorithm for detection of deadlock.

→ wait for graph is an enhanced version of Resource allocation graph in which we do not show the resources.



RAG

→



wait for graph

→ if all the resources have single instances and cycle is being formed, then the system is in deadlock. if cycle is there, but resources have more than instance, then the deadlock may or may not be present.

Recovery from Deadlock:

When a deadlock has been detected in the system by deadlock detection algorithm, then it has to be recovered by using some recovery mechanism. they are

Process termination
Resource preemption

process termination:

→ one or more processes are terminated to eliminate deadlock.

→ Terminate all deadlocked processes which will break deadlock immediately, but it is a bit expensive because there may be some processes which have been executing for a long time.

→ Abort one process at a time until the deadlock cycle is eliminated. However, it has some overhead since, after terminating each process, detection algorithm has to be executed for deciding to further terminate the process or not.

Resource preemption:

Here, resources are deallocated or preempted from some processes and the same are allocated to others until deadlock is resolved we have three important issues to implement this scheme they are

selecting a victim:

we need to decide which process or resource are to be preempted, the decision is based on cost factor which include the number of resources, a deadlocked process is holding and CPU time consumed by it.

Rollback: The process which was preempted canot continue normal execution, because its resources are taken back.

Starvation:

we should ensure that a particular process should not starve every time preemption is done.