Q1 Define Deadlock. What are 4 necessary Conditions to hold good simultaneously for a deadlock to occur (or) deadlock characterization.

A Deadlock is a situation where n proceses are executing each process is holding some resources & waiting for other proceses to release some more resources it's needs. ∴ none of the proceses get executed. This state is called deadlock state.

Dead lock characterization (or) 4 necessary conditions to hold good for deadlock to occur.

1. Mutual exclusion. Atleast one resource must be held in a nonsharable mode; i.e only one process can use a resource at a time, if another process wants same resource then it should wait until first process releases it.
2. Hold & wait: A process should hold atleast one resource & it should wait for additional resources that r presently allocated to other process.
3. No preemption: No process will release resources it's having until it completes.
4. Circular wait: Given set of processes $\{P_0, P_1, P_2 \cdots P_n\}$ then P_0 is waits for P_1 to release it's resources, P_1 is waiting for P_2 soon P_n is again waiting for P_0 to release its resources.

Above 4 conditions must occur for a deadlock to occur.

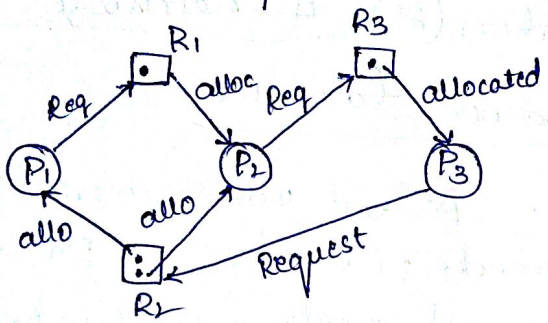
Q) Explain about Resource allocation graph (RAG).

RAG can be used to find if deadlock occurs or not. When we have single instance of each resource type then a cycle in Resource Allocation graph leads to deadlock.

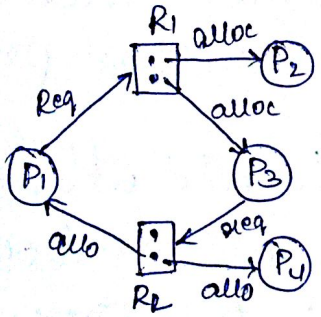
When we are having single incidence (copy) of each & every resource $R_1, R_2 \dots R_n$ then if we have a cycle in RAG then deadlock occurs

When we are having multiple instances of each & every resource type $R_1, R_2 \dots R_n$ then if we have a cycle in RAG deadlock may or may to not occur i.e having cycle in RAG is necessary but not sufficient condition

a)

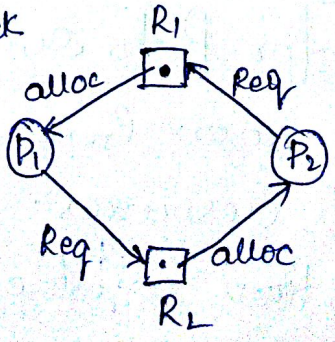


b)

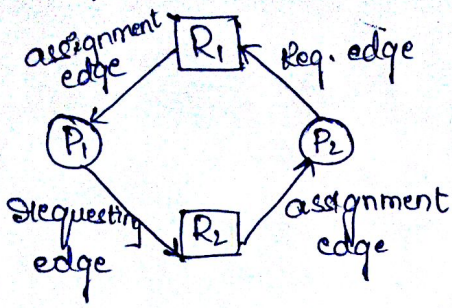


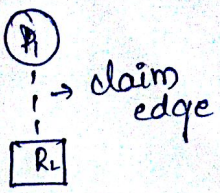
→ In fig (a) we have multiple instance of resources & a cycle & a deadlock therefore from a & b we say that when multiple instance of resources are available. If we have a cycle deadlock may or may not occur

→ In fig(b) we have multiple instance of resources & a cycle but we donot have deadlock



we have single instance of a resource shown in figure (2) if we have cycle deadlock occurs





Claim edge means next a process can request for that resource claim edge will get converted into request edge when the process request for that resource

⇒ RAG gives information about which process is requesting for which resources & what processes are having what resources

Q3 Explain various methods for handling deadlocks (3)

There are 3 ways to deal with deadlock problem.

1. use a protocols for deadlock prevention and deadlock avoidence so that system will never enter deadlock state.
2. Allow the system to enter deadlock state, then detect that deadlock has occured & recover it.
3. We can ignore the problem altogether & pretend or assume that deadlock will not occur & continue.

a) Deadlock prevention provides a set of methods to ensure that atleast one of the necessary conditions for deadlocks doesnot hold good.

b) Deadlock avoidence requires additional information about which resources a proces will request during its lifetime & using this if we grant request of a proces if deadlock doesnot occur then only requested resources will be granted else proces should wait for sometime. Deadlock not causing proceses requests will be granted and after they complete & release resources deadlock causing processes will be executed.

c) Deadlock detection & Recovery:

a) Deadlock prevention

By failing one of the four necessary conditions for deadlocks we can prevent a deadlock from occuring.

1. Mutual exclusion: If we make mutual exclusion condition to fail deadlock will not occur but for that all resources should be made sharable. But some resources like printer are nonsharable i.e they should be used by single person at a time. Some resources like files in read mode r sharable. ∴ mutual exclusion condition cannot be failed & using this we cannot avoid deadlocks.

2. Hold & wait: To fail this condition we say that

(i) Process cannot request resources if it holds other resources. (or)

(ii) A process should request all the resources it needs before it starts execution. (or)

(iii) A process can request resources if it has none (or)

(iv) A process can request new resources only after it releases all the resources it's having previously.

For suppose we want to copy data from DVD to a file on disk & then print those files. Then we first ask for DVD & disk then release DVD & disk and ask for new resources DVD & printer.

3. No preemption: To fail this condition.

a) If a process P_1 holds some resources & it's waiting for some other resources which other process is using & not releasing now. Then P_1 should release all it's resources ∵ other

resources needed by it are not available right now. (or) (4)

(b) If P_1 asks P_2 for resources & if P_2 is waiting for P_3 for extra resources then we stop P_2 & allocate P_2's resources to P_1 so that atleast P_1 will execute.

(4) Circular wait: To fail this condition we say that A process can request resources in increasing order only. If a process has resource R_6 then it can request R_7, R_8 - - - son, But it cannot request resources R_0 to R_7.

For this we define a function $F: R \to N$ where R is set of resources, N is set of natural numbers.

Suppose if F(tapedrive) = 1
F(disk) = 5
F(printer) = 12 then if Process which has tapedrive can request printer but a process which has printer cannot request tapedrive

rule: i.e a process having R_j can request R_i if and only if $F(r_j) \geq F$

Proof by Contradiction: If there is no above rule then $F(R_0) < F(R_1) < F(R_2) - - < F(R_0)$. By transitivity $F(R_0) < F$ which is not correct.

(b) deadlock avoidance: Dead lock can be avoided without occuring in 2 ways.

(i) Resource allocation Graph (when single instance of each variable is there)

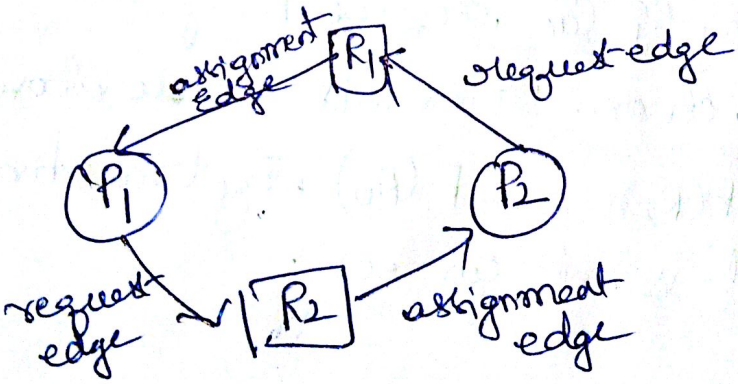
(ii) Bankers Algorithm (when multiple instances of each resource are available)

(i) Resource allocation Graph :- This is used for deadlock avoidence when we have single instance of each resource type.

If resource allocation graph has cycles in it then deadlock occurs else deadlock will not occur.

- When process P_i may request Resource R_j then we have claim edge - - -> from P_i - - -> R_j
- When resource R_j is allocated to P_i we have assignment edge → from R_j → P_i
- When process P_i requests R_j then ~~then will be~~ claim edge will be converted into request edge

→ P_i → R_j



P_1 has R_1 & is requesting R_2.
P_2 has R_2 & is requesting R_1. So cycle occured.

Resource allocation graph with cycle so deadlock has occured

(ii) Banker's algorithm for deadlock avoidance. (5)

Bankers algorithm to used for deadlock avoidence.
Resource allocation graph can be used to find out if a request allocation for a process causes deadlock or not but it is used for works only when we have one copy of each resource.
When multiple instances of a resource exists then whether a process requests of resources causes deadlock or not can be found by bankers algorithm.

Data structures used in Bankers algorithm.

1. available : It is vector of length m indicates number of available resources of each type. If available[j]=k then k instances of resource type Rj are available
2. Max : It defines maximum need of resources for each process. It is n x m matrix. Max[i,j]=k then process i can request k instances of resource Rj.
3. Allocation : It defines number of resources that are currently allocated to each process. If allocation[i,j]=k then process Pi is allocated k instances of resource r
4) Need : It is n x m matrix if defines no. of resources th are still needed by a process after allocating som resources to that process. Need[i,j] = k Process i ne still k more instances of Rj

need[i,j] = Max[i,j] - allocated[i,j]

1. Safety algorithm:

This algorithm is used to find out whether the system may be in safe state or unsafe state.

1. Initialize work = available resources
   finish[i] = false for i = 1, 2, 3 ... n processes.
2. find an i such that
   finish[i] = false; and Need[i] ≤ work
   if no such i exists goto step 4.
3. work = work + allocation i;
   finish[i] = true;
   goto step 2
4. if finish[i] = true for all i then system is in safe state.

---

2. Resource Request Algorithm

When system is in safe state & process P_i out of n process P_1 to P_n request for additional resources then ~~they~~ P_i will be granted those resources only if deadlock doesn't occur.

Algorithm

Let Request i be the request vector for process P_j. If Request i[j] = K then process P_i wants K instances of resource j (R_j). When a request for resources ~~are~~ is made by process P_i the following actions are taken.

1) If Request i ≤ need goto step 2
   else print("process has exceeded its maximum claim");
2) If request i ≤ available goto step 3 otherwise ~~P~~
   Print("~~requested~~ wait until requested resources become available");
3) available = available - Request i;
   allocation i = allocation i + Request i;
   Need i = need i - Request i;

4) e) resulting resource allocation state is safe the (6)
process P_i is allocated requested resources else
If P_i's request causes deadlock then P_i is not
allocated resources and old resource allocation state
is restored.

Problem: A system consists of five processes (P_1, P_2, P_3, P_4, P_5) and 3 resources (R_1, R_2, R_3). R_1 has 10 instances, R_2 has 5 inst, R_3 has 7 instances. The following snapshot of system has been taken, find out whether system is in safe state or not? & what is the safe sequence.

Process	Allocation			Max			available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	0	1	0	7	5	3	3	3	2
P_2	2	0	0	3	2	2			
P_3	3	0	2	9	0	2			
P_4	2	1	1	2	2	2			
			2	4	3	3			

Now safety algorithm will be done & if all $P_1 - P_5$ can be completed with no deadlock in any order then system is in safe state.

If need $\leq$ allocation that process is allocated resources & completed else we goto next process.

1) P_1 need $= (7,4,3)$
available $= (3,3,2)$
So P_1 is ~~stop~~ not done P_2 gets chance

2) P_2 need $= (1,2,2)$ ~~or~~
available $= (3,3,2)$
need of $P_2 \leq$ available
So P_2 is completed & all resources of P_2 are released and added to (available or) work
work = work + allocation for P_2 (or)
available = available + allocation for P_2
$= (3,3,2) + (2,0,0)$
available $= (5,3,2)$

(3) P_3 need $= (6,0,0)$
need of $P_3 >$ available $(5,3,2)$
So P_4 gets chance

(4) P_4 need $(0,1,1)$
P_4 need $<$ available $(5,3,2)$
So P_4 completed
available = available + allocation for P_4
$= (5,3,2) + (2,1,1)$
$= (7,4,3)$.

(5) Now P_5 need $= (4,3,1)$
P_5 need $<$ available $(7,4,3)$
So P_5 executed
available = available + allocation of P_5 —
$(7,4,5) = (7,4,3) + (0,0,2)$

8) Now one cycle completed.
P_1 & P_3 are left
Now need P_1 (7,4,3) < available (7,4,5)
So P_1 completed
available = available + Allocation for P_1
= (7,4,5) + (0,1,0)
= (7,5,5)

7) Now P_3 need (6,0,0) < available (7,5,5)
Now P_3 completed
available = available + Allocation for P_3
= (7,5,5) + (3,0,2)
= (10,5,7)

Now all processes are executed without deadlock
So ~~for are~~ system is in safe state &
safe sequence is (P_2, P_4, P_5, P_1, P_3)

---

2) The safe sequence of P_2, P_4, P_5, P_1, P_3 system contains 3 processes and 3 resources, no. of instances of each resources are (7,7,10) Current resources allocation is shown below

⇒ process	allocation			maxneed			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	3	3			
P_3	1	2	4	3	4	4			
	5	4	10						

P	Need R_1	R_2	R_3
P_1	1	4	5
P_2	2	3	0
P_3	2	2	0

i) ⇒ Is current allocation in safe state? can the request $P_1(1,1,0)$

⇒ P_1 needs $(1,4,5) >$ avail $(2,3,0)$

∴ P_1 is not executed

⇒ P_2 needs $(2,3,0) = (2,3,0)$

avail = avail + allocation for P_2

$= (2,3,0) + (2,0,3)$

$= (4,3,3)$

⇒ P_1 needs $(1,4,5) >$ avail $(4,3,3)$

~~⇒ P_3 needs $(2,4,0$~~ ∴ P_1 is not executed

⇒ P_3 needs $(2,2,0) < (4,3,3)$

new avail = avail + allocation for P_3

$= (4,3,3) + (1,2,4)$

$= (5,5,7)$

P_1 needs $(1,4,5) < (5,5,7)$

= avail + allocation for P_1

$= (5,5,7) + (2,2,3) \Rightarrow (7,7,10)$

System is in Safe state

Safe state $< P_2, P_3, P_1 >$

ii) can the request $P(1,1,0)$ be granted

Process	allocation			max need			available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	3	3	3	6	8	1	2	0
P2	2	0	3	4	3	3			
P3	1	2	4	3	9	4			
	6	5	10						

(4)

P	needs		
	R1	R2	R3
P1	0	3	5
P2	2	3	0
P3	2	2	0

⇒ P1 need (0, 3, 5) > (1, 2, 0) avail so P1 not executed

⇒ P2 need (2, 3, 0) > (1, 2, 0) avail so P2 not executed

⇒ P3 need (2, 2, 0) > (1, 2, 0) avail so P3 not executed

∴ if we grant the request P1(1, 1, 0) is addition to the old request

3) P1 (2, 2, 3) total allocated of P1 will become (2, 2, 3) + (1, 1, 0) = (3, 3, 3) which will resulting unsafe state and Deadlock also

Process	allocation				max need				available			
	R1	R2	R3	R4	R1	R2	R3	R4	R1	R2	R3	R4
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				
	2	9	10	12								

consider following snapshot of the system P_0, P_1, P_2, P_3, P_4

$R_1 = 2,\ R_2 = 14,\ R_3 = 12,\ R_4 = 12$

(i) Is the system in safe state

(ii) can the request P_1 (0, 4, 2, 0) be accepted

A

Process	needs			
	R_1	R_2	R_3	R_4
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

$\Rightarrow P_0$ needs $(0,0,0,0) < (1,5,2,0)$

avail = $(1,5,2,0)$ + P_0 allocated $(0,0,1,2)$

$= (1,5,3,2)$ so P_0 is executed

$\Rightarrow P_1$ needs $(0,7,5,0) > (1,5,3,2)$ P_1 is not executed

$\Rightarrow P_2$ needs $(1,0,0,2) < (1,5,3,2)$ so P_2 is executed

$= (1,5,3,2) + (1,3,5,4)$

$= (2,8,8,6)$

$\Rightarrow P_3$ needs $(0,0,2,0) < (2,8,8,6)$

$= (2,8,8,6) + (0,6,3,2)$

$= (2,14,11,8)$

(9)

=> P4 needs (0, 6, 4, 2) < (2, 14, 11, 8)

= (2, 14, 11, 8) + (0, 0, 1, 4)

= (2, 14, 12, 12)

=> P1 needs (0, 7, 5, 0) < (2, 14, 12, 12)

= (2, 14, 12, 12) + (1, 0, 0, 0)

= (3, 14, 12, 12)

=> system is in safe state P0, P2, P3, P4, P1

(ii) can the request P1 (0, 4, 2, 0) be accepted

P1 (0, 4, 2, 0)
+ P1 (1, 0, 0, 0)

P1 (1, 4, 2, 0)

avail

1 5 2 0
- 0 4 2 0
---
1 1 0 0

Process	need				available			
	R1	R2	R3	R4	R1	R2	R3	R4
P0	0	0	0	0	1	1	0	0
P1	0	3	3	0				
P2	1	0	0	2				
P3	0	0	2	0				
P4	0	6	4	2				

=> P0 needs (0, 0, 0, 0) < avail (1, 1, 0, 0)

avail = (1, 1, 0, 0) + (0, 0, 1, 2)

= (1, 1, 1, 2)

⇒ P_1 needs $(0, 3, 3, 0) > (1, 1, 1, 2)$

P_1 not executed

⇒ P_2 needs $(1, 0, 0, 2) < (1, 1, 1, 2)$

$(1, 1, 1, 2) + (1, 3, 5, 4)$

$= (2, 4, 6, 6)$

P_2 is executed

⇒ P_3 needs $(0, 0, 2, 0) < (2, 4, 6, 6)$ P_3 executed

available = available + allocation P_3

$= (2, 4, 6, 6) + (0, 6, 3, 2)$

$= (2, 10, 9, 8)$

⇒ P_4 needs $(0, 6, 4, 2) < (2, 10, 9, 8)$

$= (2, 10, 9, 8) + (0, 0, 1, 4)$

$= (2, 10, 10, 12)$

P_4 executed

⇒ P_1 needs $(0, 3, 3, 0) < (2, 10, 10, 12)$

P_1 executed

available = available + allocation P_1

$= (2, 10, 10, 12) + (1, 4, 2, 0)$

$= (3, 14, 12, 12)$

Safe state $< P_0, P_2, P_3, P_4, P_1 >$

(c)

Deadlock detection

a) Single instance
b) multiple instance

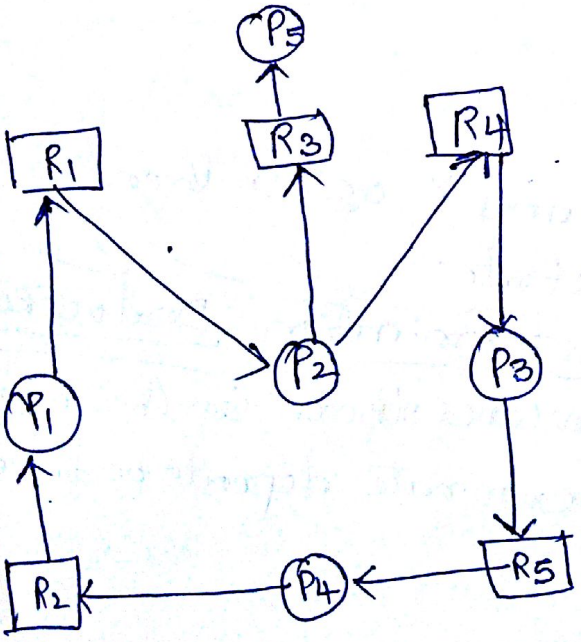
(10)

If deadlock prevention or avoidance algorithms are not used then deadlock occurs. We should have an algorithm that detects that deadlock has occured and recovers the system from deadlock.

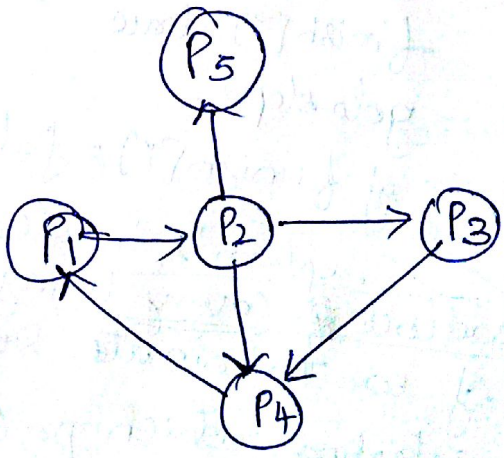
a) Deadlock detection ~~when~~ for Single Instance of each resource type.

Here we use wait for graph. If P_1 is waiting for P_2 to release its resources then we have a edge from P_1 to P_2. Like this we draw the graph. If there are cycles in dracon graph then deadlock occurs.

A resource allocation graph can be converted into wait for graph if we remove all the resources from graph and ~~edges~~ edges from P_1 to R_1 and P_1 to P_2 are replaced with single edge from P_1 to P_2.



Resource allocation Graph



Wait for Graph.

(b) Deadlock detection when we have multiple instances of each resource type.

available: It's of length m and it indicates total no. of available resources.

allocation: It's n x m matrix which shows no. of resources allocated to each process.

Request: It's n x m matrix. request[i][j]=k then process i is requesting k more instances of resource j.

Algorithm

1. Initialize work = available.
   If allocation$_i$ ≠ 0 then
   finish[i]=false;
   else finish[i]=true;
2. find an index i such that
   finish[i] = false and Request$_i$ ≤ work.
   If no such i exists goto step4.
3. work = work + allocation$_i$
   finish[i]=true
   goto step2
4. If finish[i]=false for any i 0≤i<n then system is in deadlock state.

Q9) Deadlock Recovery using process termination Deadlock Recovery

If we terminate some processes others can continue. But selecting which process to terminate depends on following factors.

1. what is priority of process

2. Already how much time process executed & how much time it still needs.
3. How many & what types of resources process has used.
4. How many more resources process needs.
5. To continue this process how many processes needs to be terminated.
6. Whether process is interactive or batch.

(b) Deadlock recovery using Resource preemption

1. Selecting a victim: which resources are to be preempted from which process. A process which has executed for longtime it's resources should not be preempted, a process just started has been deadlocked it's resources should be preempted to minimize cost.
2. Rollback: If we stop a deadlocked process it should be again started from the beginning so we should rollback it to some safestate ~~to that~~ taking some resources from it so that it can restart from rollbacked point.
3. Starvation: We should see that we don't always take a same resources from single process making it to wait indefinitely. A process rollbacked many times should not be rollbacked again.

Deadlock Recovery
- Recovery by process termination
- Recovery by Resource preemption.