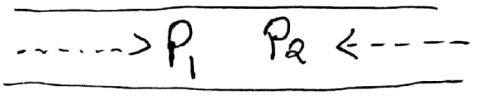
Jenny's

Find

Deadlock

- Waiting for something for infinite time in which there is no progress for waiting processes.
- Processes waits for one another's action indefinitely



- multiprogramming Os → infinit of resources } competing for

Procen → req → Resources

↓ waiting

↓ holding

waiting

P2 → Resource

- Condition (coffman) : (All these condition will be held for deadlock)

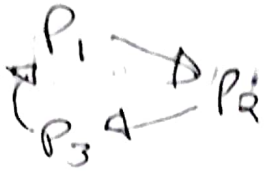
(I) Mutual exclusion : At least one resource must be held in non-sharable mode.

(II) Hold and Wait : Procen holding one resource & Waiting to acquire resources that are currently held by another

(iii) Non-Preemption : Resources cannot be preempted.

(iv) ~~Hold and Wait~~ :

(iv) Circular Wait :



Resource Allocation Graph

Set of vertices : V — Set of Processes { P1, P2, P3 ... }
Set of Resources { R1, R2, R3 ... }

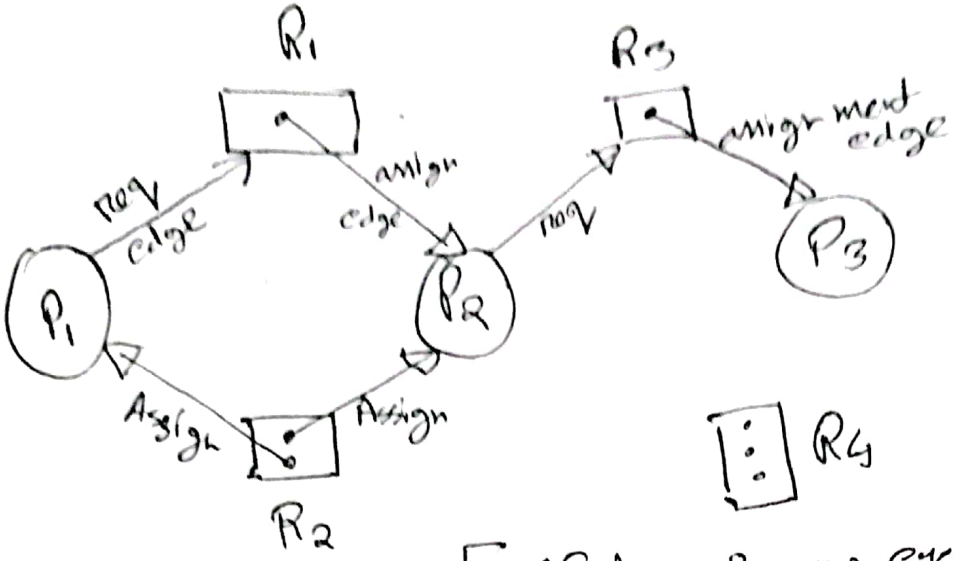
Set of Edges : E — Pi → Rj (Req edges)
Rj → Pi (Assignment edges)

□ → Resource type

○ → Process

⊡ → instance of resource type

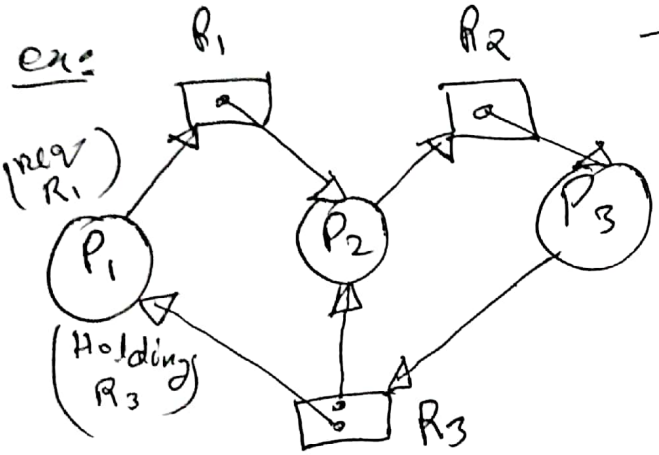
eg:



- If there is no-cycle than no-Dead lock
- " " cycle than there may Dead lock exists.

(NOTE)

ex:



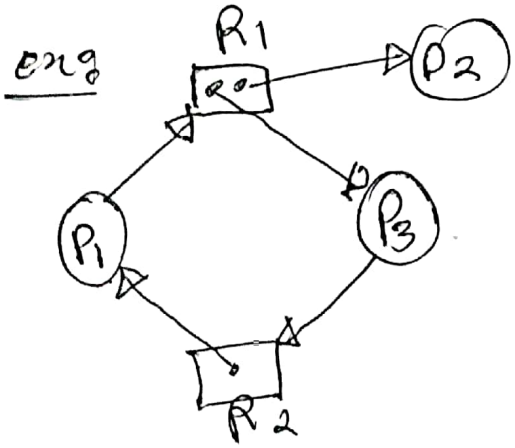
→ Vor ⇐

	← Allocation →			Request →		
	R_1	R_2	R_3	R_1	R_2	R_3
P_1	0	0	1	1	0	0
P_2	1	0	1	0	1	0
P_3	0	1	0	0	0	1

	Available		
	R_1	R_2	R_3
P_1	0	0	0
P_2	1	0	0
P_3	1	0	0

Dead lock : No progress (No Procen (P_1, P_2, P_3) in not oriented)

eg:



	Allocation		Request		Availability	
	R_1	R_2	R_1	R_2	R_1	R_2
P_1	0	1	1	0	0	0
P_2	1	0	0	0	1	0 ✓
P_3	1	0	0	1	1	1

So, NO Dead Lock ✓

✱ Deadlock Handeling : (deadlock never Occure.)

(I) Deadlock prevention.

(II) Deadlock Avoidence.

(III) Deadlock Detection & Recovery.

(IV) Deadlock Ignorance (Ostrich Method)

(I) Deadlock prevention : (condition of Deadlock) (remove all 4 there)

(1) Mutual ~~exclusion~~ exclusion : Must hold non-sharable resources. [U have to remove it]

(2) Hold and Wait : May hold while awaiting for anig

✓ A procen must acquire all necenary resources before execution stats.

✓ Starbation.

✓ wait time out.

(3) No-Preemption : Resources can be preempted from procenes.

(4) Circular wait : 

Avoidance - Resource Allocation:

- system maitains some database using which can face decision whether to entertain a request or not, just to be in safe ~~state~~.
- System (Kernel) analyse the database (allocation state) to determine whether granting a request can lead to deadlock in future
  - if not lead to deadlock then granted
  - Otherwise keep pending until they can be granted.

(Procen may face delay for obtaining a resource.)

unsafe (Deadlock) / Safe

① Req edge $P_i \rightarrow R_j$, ② Assign edge $R_j \rightarrow P_i$,
③ claim edge $P_i \dashrightarrow R_j$.

$P_1 \dashrightarrow R_1$
$P_1 \longrightarrow R_1$
$P_1 \longleftarrow R_1$
$P_1 \dashleftarrow R_1$

(22) Condition : resource must be claimed in advance

→ if P_i req R_j then the req edge can only be converted to assign edge if it doesn't form a cycle in resource allocation graph.

(*) Bankers algo : Handle multiple instance of same resources.

(i) How many instances of each resource each process can request [Max]

(ii) How many instances of each resources each process currently holds [Allocation]

(iii) How many instance of each resource is available in the system [Available]

	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				
	2	9	10	12								

(i) Need Matrix?

(ii) Is system in safe state? if yes! the find safe sequence?

✓ [Need = Allocation - Max.]

✓ [Total ⇒ Sum of Allocation + Available]

∴ Total = A B C D
3 14 12 12

Need				
	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

[NOTE: Need ≤ Available satisfy]

✓
x
x
✓
x

∴ Safe sequence: P_0 P_3

(+) (compare)

	Allocation				Max				Available				Need			
	A	B	C	D	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	0	0	0	0 (I)
P_1	1	0	0	0	1	7	5	0	1	5	3	2 (I)	0	7	5	0
P_2	1	3	5	4	2	3	5	6	2	8	8	6	1	0	0	2 (II)
P_3	0	6	3	2	0	6	5	2	2	14	11	8	0	0	2	0 (III)
P_4	0	0	1	4	0	6	5	6	2	14	12	12	0	6	4	2 (IV)
									3	14	12	12				

Safe seq: P_0 P_2 P_3 P_4 P_1

yes: system in in safe state.

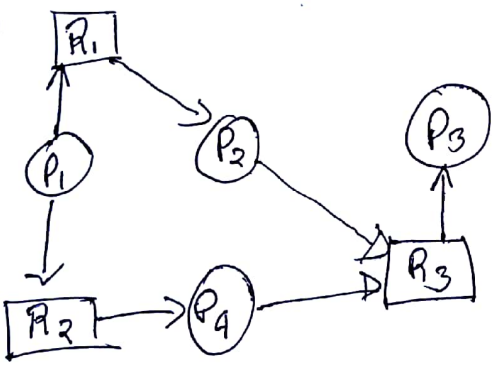
Deadlock Detection & Recovery

- Allow they system to enter into deadlocked state
- Deadlock detection algo (2 types) → [single instance (wait-for). Multiple " (Bankers).
- Recovery technique

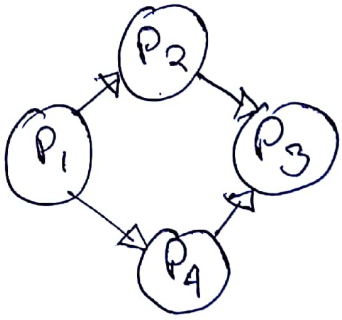
✓ Wait for graph : (Detect cycle)

Resource Allocation Graph

(1)

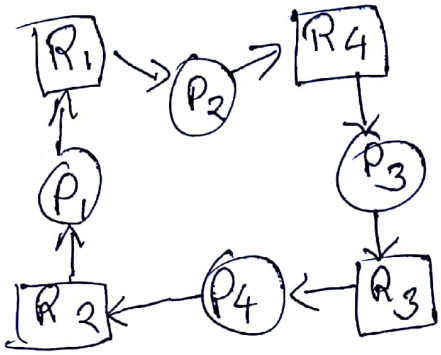


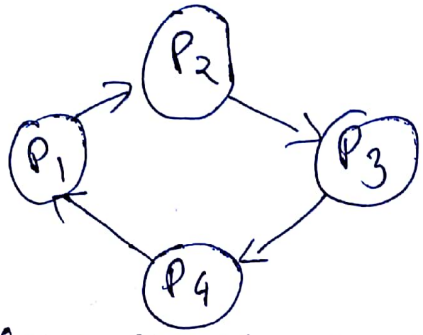
wait for graph



(no-Dead Lock, cuz no-cycle)

(ii)





(yes-Dead Lock, cycle)

Banker's algo (safety algo):

	Allocation			Req			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	~~0~~	1	~~0~~
P_2	3	0	3	0	0	0	0	1	3
P_3	2	1	1	1	0	0	5	2	4
P_4	0	0	2	0	0	2	7	2	4
P_5							7	2	6

P_0 (Allocated...

P_2 (...=0)

Safe seq: $\langle P_0 \; P_2 \; P_3 \; P_1 \; P_4 \rangle$