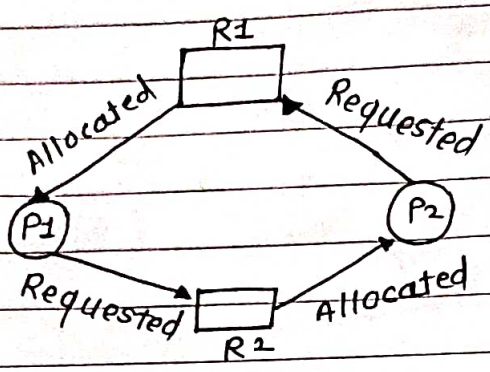
Module 4

Deadlock

→ Two or more processes are waiting for some event to be happen, but that event doesn't happen that is deadlock
→ And those processes are in deadlock state.



Both processes keep waiting infinitely.

Execution of two or more processes is blocked because each process holds some resource and waits for another resource held by some other process.

- Conditions for deadlock

1. Mutual Exclusion
2. Hold and wait
3. No preemption
4. Circular wait

} All conditions must be satisfied for deadlock to occur.

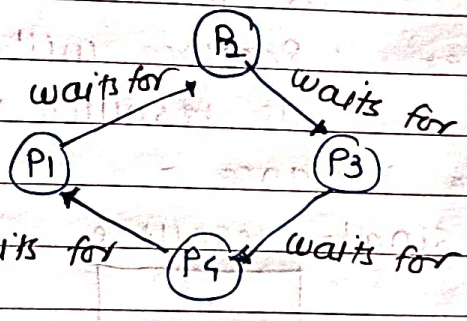
1] Mutual Exclusion :

- There must exist at least 1 resource in system, which can be used by only one process at a time.
- If no such resource, no deadlock
- Eg. printer

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

3] No preemption :
- Once the resource has been allocated to the process, it can't be preempted.
- Process must release the resource voluntarily.

4] Circular wait :
- All the process must wait for resource in a cyclic manner, where the last process waits for resource held by first process.



⊢ Resource allocation graph (RAG)
- as it is graph there exist of vertex and Edges.

- Vertex
  - Process vertex
  - Resource vertex
    - ① Single instance (i.e single resource) CPU, monitor
    - ② multiple instance (multiple resources) Registers.

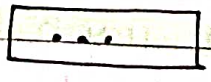
- Edges :
  - Assigned Edge — (P) ← [R] (Allocated)
  - Request Edge — (P) → [R] (Requesting)

→ RAG shows which resource is held by which process and which process is waiting for a resource.

→ shows is the state of the system in terms of processes and resources.

- Vertices in RAG :
  1] Process Vertex : process will be represented as process vertex.
  2] Resource vertex : Recource will be represented as resource vertex.
    A] Single instance - Single resource

    B] Multiple instance - Multi resource

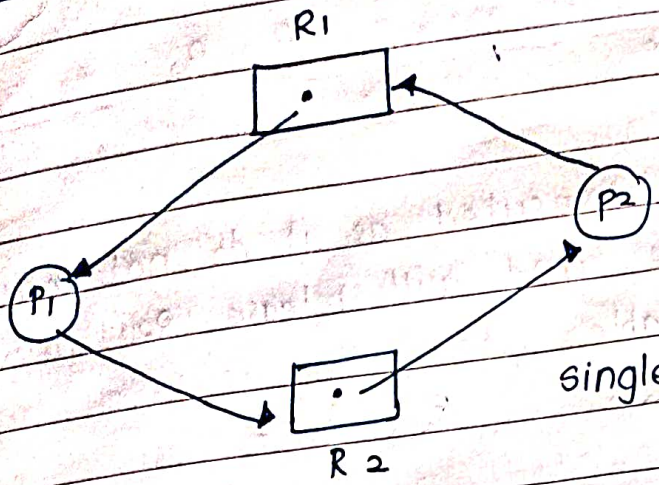


- Edges in RAG :
  1] Assign edge : If resource is already allocated

  2] Request Edge : If process is requesting for certain resource.

cycle in single instance is sufficient for deadlock

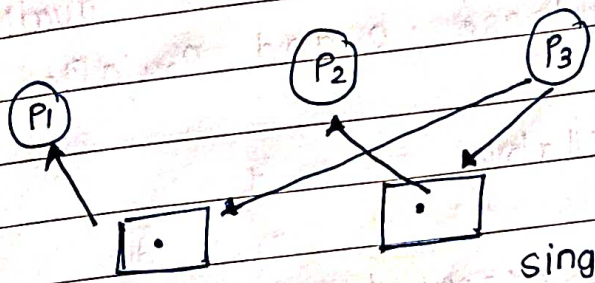
Examples 1



(There exist a cycle)

single instance with deadlock

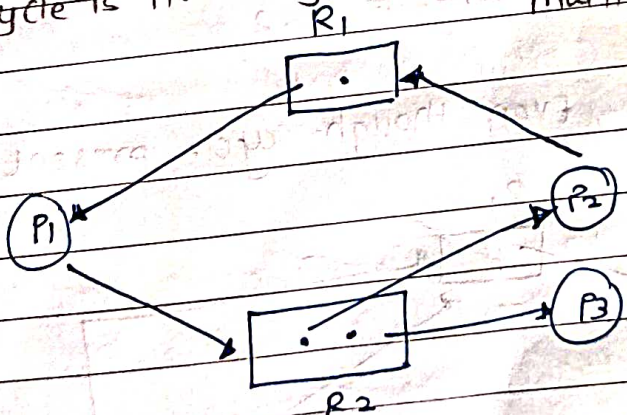
2



(No circular wait)

single instance without deadlock

3 cycle is not only sufficient condition for deadlock in multi instance



multiinstance without deadlock

Find Allocation matrix & Request matrix

Process	Allocation		Request	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	0	0

check for deadlock

current Available resource = 0 0

① P_3 will get executed as it do not require any resource and will release R_2

Now available resources = 0 1

② P_1 will get executed as R_2 is available now and will release R_1 and R_2 after execution.

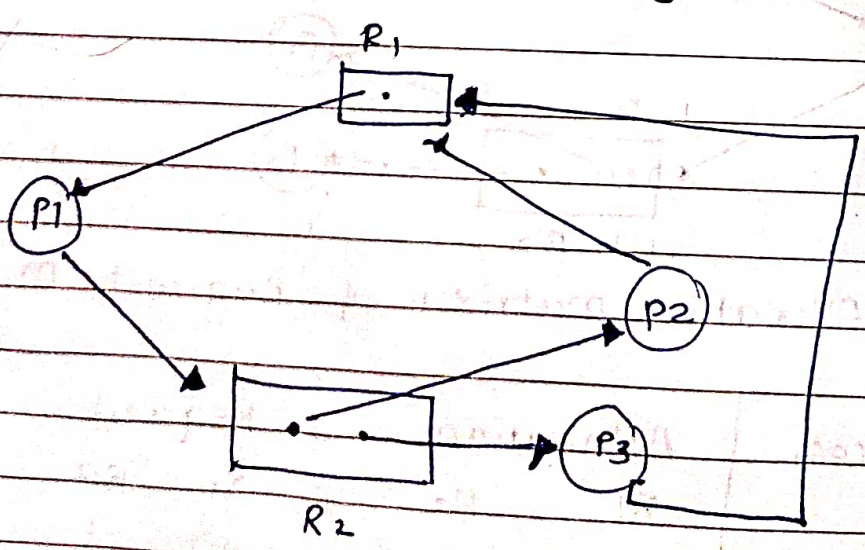
New availabe = 1 1

③ P_2 will get executed as R_1 is available and will release R_1 & R_2 after execution.

New available = 1 2

∴ No deadlock, Even though cycle present.

Ex 4



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

Currently Available resource = (0, 0)

But requirement is P_1 (0,1)
P_2 (1,0)
P_3 (1,0)

So can't fulfill any one requirement.
Thus deadlock.

Deadlock Handling

1. Deadlock prevention
2. Deadlock Avoidance
3. Deadlock detection & Recovery
4. Deadlock Ignorance

(1) Deadlock Ignorance:

- Just ignore the deadlock concept.
- So it will help to avoid extra overhead of handling deadlock.
- Linux, windows eg.
- Also called Ostrich approach.

(2) Deadlock prevention:

- from four necessary conditions required for deadlock occurrence make any condition false.

- This leads to deadlock free system.

① Mutual exclusion:
- In order to make this condition false, make all the resources sharable.
- But, this condition can't be violated as there are some resources which are mutually exclusive by nature.

② Hold and wait: In order to make it false,

[i] - Process has to request for all required resources.
- Once acquired all the resources, only then it can start its execution.
- less efficient, Not implemented (as prediction of resources required for execution is not possible)

[ii] - Acquire the resource needed currently
- Before making new request for resource, release all the resources that it holds currently
- Implementable

[iii] - Timer set after the process acquires any resource, after the end of timer, a process has to compulsorily release the resource.

③ No preemption:
- To violate this condition, prempt the process forcefully.
- Possible if preempting process having low priority

④ Circular wait :-

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

Approach - 1. Assign numbers to resources.
2. Each process is allowed to req. for the resources either in only increasing or in decreasing order of the resource no.
3. In case increasing order is followed, if process requires a lesser no. resource then it must release all resource having larger no. & vice versa.

* This leads to starvation but not to deadlock.

③ Deadlock Avoidance :

- If fulfilment of new request causes the system to move in an unsafe state, then discard that request.
- Requirement : max resource requirement should be known in advance.

• Banker's algo is one of the strategy.

④ Deadlock Ignorance Detection and Recovery :

- Detect the deadlock if present then kill the process until no deadlock.

Bankers Algorithm and Deadlock avoidance :-

- Data structures used
  1. Available - Currently available resources
  2. max - max requirement of resources
  3. Allocation - No. of allocated resources.
  4. Need - max - Allocation

- when process enters in system, it must declare the max. no of instances of each resource type. required for its execution
- It must be less than total no. of resources in the system.
- Now, when new process request for the resource,
  the sys. will calculate whether allocation of requested resource will keep the system in safe mode or not.

  if safe
  - Allocate resource

  else
  - wait until some other process releases requested resource.

Example: A = 10, B = 5 and C = 7

Good

Available ≤ Need

Allocation			Max need			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2	7	4	3 ✓
P2	2	0	0	3	2	2				1	2	2 ✓
P3	3	0	2	9	0	2				6	0	0 ✓
P4	2	1	1	4	2	2				2	1	1 ✓
P5	0	0	2	5	3	3				5	3	1 ✓
	7	2	5									

Execution of

P2

① 10 5 7 ✓

− 7 2 5

3 3 2 ∴ Available = 3 3 2

② 3 3 2

+ 2 0 0 (released by P2) ∴ Available = 5 3 2

5 3 2

P2 → P4

③ 5 3 2

+ 2 1 1 (released by P4) ∴ Available = 7 4 3

7 4 3

P2 → P4 → P5

④ 7 4 3

+ 0 0 2 (released by P5) ∴ Available = 7 4 5

7 4 5

P2 → P4 → P5 → P1

⑤ 7 4 5

+ 0 1 1 (released by P1) ∴ Available = 7 5 5

7 5 5

+ 3 0 2 (release by P3)

10 5 7 ✓

P2 → P4 → P5 → P1 → P3

Problems on Bankers Algo :

Process	Allocation			Max			Available			max-Allo Remainin		
	E	F	G	E	F	G	E	F	G	E	F	G
P_0	1	0	1	4	3	1	3	3	0	3	3	0
P_1	1	1	2	2	1	4				1	0	2
P_2	1	0	3	1	3	3				0	3	0
P_3	2	0	0	5	4	1				3	4	1

→

Execute P_0

① current available = 3 3 0
+ 1 0 1
After execution = 4 3 1

② current available = 4 3 1
+ 0 0 3
After execution = 5 3 4

$P_0 \rightarrow P_2$

③ current available = 5 3 4
+ 1 0 1
After execution = 6 3 5

$P_0 \rightarrow P_2 \rightarrow P_3$

④ current available = 6 4 5
+ 2 0 0
After execution = 8 4 5

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

∴ There is safe state — No deadlock