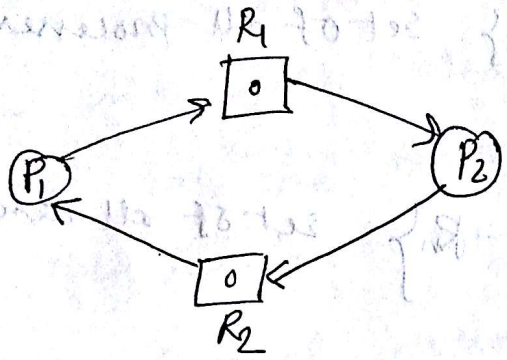
Ch-7

Read locks

What is Read Lock?

Read lock is a set of ~~resources~~ processes in which each process holding a resource & waiting to acquire new resources held by another processes in ~~time~~ the set. This situation is called deadlock.



Read lock can arise if the following four conditions hold simultaneously.

(i) Mutual exclusions: only one process at a time can use a resource

(ii) Hold & wait: A process holding a resources. and waiting to acquire another resource held by another process.

(iii) No preemption: A resource can be ~~prede~~ released only voluntarily by the process holding it.

(iv) Circular wait: There exists a set of waiting processes $\{P_0, P_1, P_2 \dots P_n\}$. Here P_{i-1} waiting for resource held by P_i & P_n waiting for a resource held by P_0.

Resource Allocation graph:

A set of vertices V & Edges E. V is partitioned into two types.

(i) $P = \{P_1, P_2, P_3 \dots P_n\}$ set of all processes in the system

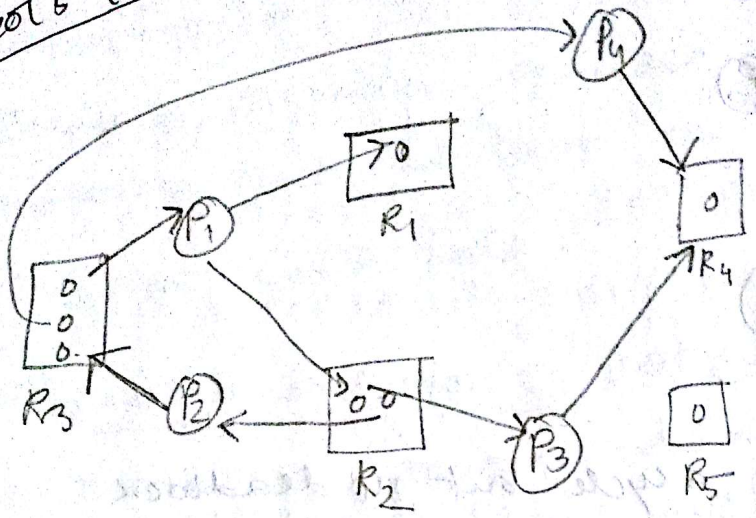
(ii) $R = \{R_1, R_2, R_3 \dots R_n\}$ set of all resources in the system.

There two types of edges:—

(i) requested edge → directed edge $P_i \rightarrow R_j$

(ii) Assigned edge → directed edge $R_j \rightarrow P_i$

2015 (5b)

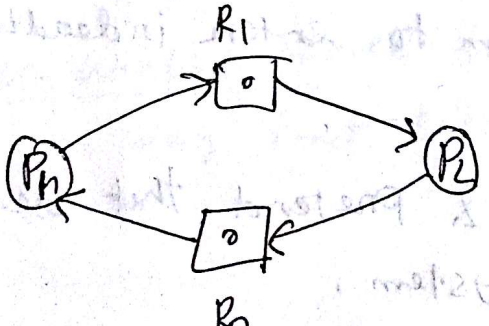


V.V.I A graph with no cycle no deadlocks.

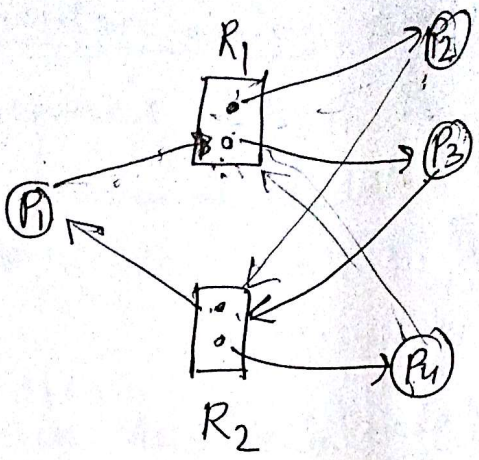
A graph with cycle deadlock may arise:-

(i) If only one instance per resource type then deadlock.

(ii) If several instance per resource type then possibility of deadlock.



Here single instance per resource & contains a cycle so deadlock arise.



This graph contains a cycle but no deadlock occurs.

Methods of handling deadlock:-

(1) Ensure that the system never enters in deadlock. This can be implemented by:
   (i) Deadlock prevention.
   (ii) Deadlock avoidance.

(2) Allow the system to enter in deadlock then recover.

(3) Ignore the problem & pretend that deadlocks never occur in the system.

Deadlock Prevention:

Deadlock prevention provides a set of methods ensuring that at least one of the following condition can't hold.

① Mutual Exclusion: not required for sharable resources. must hold non-sharable resources.

② Hold & wait: Must guarantee that whenever a process requests a resource, it does not hold any other resources.

③ No Preemption: If ~~some~~ some process holding some resource & requests another resource it can't be immediately allocated to it.

④ Circular wait: make total ordering of resources in such a way that circular wait doesn't happen.

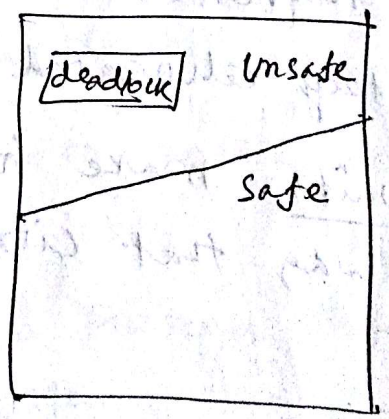
* A system is in safe state if there exist a sequence $\langle P_0, P_1, P_2 \ldots P_n \rangle$ of all processes in the system such that for each P_i, the ~~resources P_i can request~~ the resource requests that P_i can be satisfied by available resources.

otherwise a system is in unsafe state.

If a system is in safe state no deadlock.

If a system is in unsafe state possibility of dead lock.

Avoidance algorithm ensures system never in unsafe state.



Recovery from deadlocks

There are two ways of recovery from deadlock.

① Process termination:

(i) Abort all dead lock processes.

(ii) Abort one process at a time until dead lock cycle is eliminated.

In which order should chose to abort

(a) Priority of the process.

(b) How long process has computed.

(c) Resources the process has used.

(d) Resources process needs to compute.

(e) How many processes will need to be terminated.

(f) Is process interactive & batch.

② Resource Preemption:

① Selecting a victim → minimize lost.

② Rollback → return to some safe state, restart process for that state.

③ Starvation: Some process may always pick as a victim. include no. of rollback in cost factor.

Bankers algorithm

Safety algo:

Work & finish two vector length with m & n.

① Work = available.

Finish [i] = false

② Find an i such that both

(a) Finish [i] = false.

(b) Need$_i$ $\leq$ work.

If no such i exists go to step 4.

③ Work = Work + Allocation$_i$

Finish [i] = true.

④ If Finish [i] == true forall i

then the system is in safe state.

A (10 instance) B (5 instances) C (7 instances)

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

	Need A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Work = available (332)

Need [1] = 122 < Work (332)

∴ Work = 332 + 200

= 532 → P1

Need [2] = 600 > Work (532)

Need [3] = 011 ≤ Work (532)

Work = 532 + 211

= 743 → P3

Need [4] = 431 < Work (743)

Work = 743 + 002 = 745

→ P4

Need [0] = 743 ≤ work [745].

Work = 745 + 010

= 755 ⟶ P_0

Need [2] = 600 ≤ wok (755).

Work = 755 + 302

= 1057. ⟶ P_2

$P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$

<u>P_1 (102) be granted?</u>

Request $P_1 \leq$ Need (P_1)

Request $P_1 \leq$ Available.

Then Need P_1 = Need(P_1) − Req (P_1)

Allocation P_1 = Allocation (P_1) + Req (P_1)

Available = Available − Req (P_1)

$P_1(102) \leq$ Need $P_1(122)$

$P_1(102) \leq$ Available (332).

Allocate $P_1 = 200 + 102 = 302$

Need $P_1 = 122 - 102 = 020$

Available $P_1 = 332 - 102 = 230$

	Allocat	~~Max~~ Need	230
P_0	010	743	
P_1	302	020	Need $P_0 = 743 < 745$
P_2	302	600	Work = 745 + 010 = 755 → P_0
P_3	211	011	Need P_2 600 < 755
P_4	002	431	Wok = 755 + 302 = 1057 → P_2

Need $P_1 = 020 \leq$ Work (230) → P_1

Work = 230 + 302 = 532

Need $P_2 = 600 >$ Work (532).

Need $P_3 = 011 <$ Work (532) → P_3

Work = 532 + 211 = 743

Need $P_4 = 431 <$ Work (743) Work = 743 + 002 = 745 → P_4

2014
6(a)

Allocation	A	B	C	D
P_0	0	0	1	2
P_1	1	0	0	0
P_2	1	3	5	4
P_3	0	5	2	3
P_4	0	1	2	4

Max	A	B	C	D
	0	0	1	2
	1	7	5	0
	2	3	5	6
	0	6	4	5
	0	5	6	4

Available	A	B	C	D
	1	5	2	0
	1	5	3	2
	2	8	8	6
	2	13	10	9
	2	14	12	13
	3	14	12	13

15 32
2886
0523
2 13 10 9

(i) Need (Max - Alloc)

A	B	C	D
0	0	0	0
0	7	5	0
1	0	0	2
0	1	2	2
0	4	4	0

Safe seq $(P_0, P_2, P_3, P_4, P_1)$

(ii) ~~Allocation~~

Allocation			
A	B	C	D
0	0	1	2
1	3	2	0
1	3	5	4
0	5	2	3
0	1	2	4

Max			
A	B	C	D
0	0	1	2
1	7	5	0
2	3	5	6
0	6	4	5
0	5	6	4

Available			
1	2	0	0
1	2	1	2
2	5	6	6
1	2	10	8
2	11	10	13
3	14	12	13

Need			
0	0	0	0
0	4	3	0
1	0	0	2
0	1	2	2
0	4	4	0

Seq (P_0, P_2 P_3, P_4, P_1)

The req can be granted.

Detection algorithm

Main: Request ≤ work (available) वाला (check)