System model

* Deadlock characterization / necessary Conditions

** Methods for Handling Deadlocks
- Deadlock prevention
- *Deadlock Avoidance
- *Deadlock Detection
- Recovery from Deadlocks

} Avoiding conditions of Deadlock.

*Deadlock :-

If two or more process are waiting for happening some event which never happens is called deadlock.

System model :-

→ system consist of resources

→ resource types $R_1, R_2, ---R_m$

CPU cycles, memory space, I/o devices etc

→ each resources type $R_i$ have $W_i$ instance

→ Each process utilizes a resource as follows :-

request

use

release

Resources

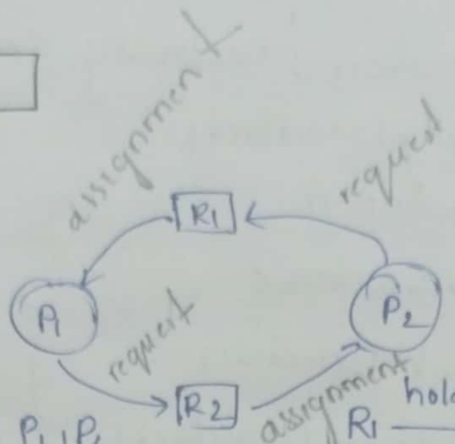
- Physical
  - Eg:- CPU, memory, Hard disk etc
- Logical
  - Eg:- files, libraries, Semaphores etc

representation:-

process :- ○

resources :- □

eg:-




- two process $P_1, P_2$
- two resources $R_1, R_2$

$R_1 \xrightarrow{\text{holds}} P_1$

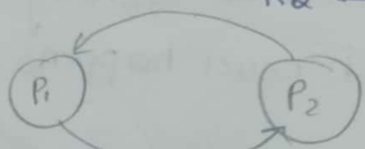
$P_1 \xrightarrow{\text{request}} R_2$

$P_2 \xrightarrow{\text{request}} R_1$

$R_2 \xrightarrow{\text{holds}} P_2$

indirectly $P_1$ waiting for $P_2$

$P_2$ is waiting for $P_1$




∴ Deadlock situation arise

**

## Deadlock characterization :-

Deadlock can arise if four conditions occur simultaneously.

i) Mutual exclusion :- Only one process at a time can use a resource.

ii) Hold and wait :- a process holding at least one resource is waiting to acquire additional resources held by other process.

iii) No preemption :- a resource can be released only voluntarily by the process holding it, after the process had completed its task.

iv) Circular wait :-

there exist a set {$P_0, P_1 -- P_n$} of waiting processes such that $P_0$ is waiting for a resource that is held by $P_1$, $P_1$ is waiting for a resource that is held by $P_2$ -- --, $P_{n-1}$ is waiting for a resource that is held by $P_n$ and $P_n$ is waiting for a resource that is held by $P_0$.

## → Resource Allocation Graph (RAG)

A set of vertices V and edges E

- Graph
  - Vertices V types :-
    - process
    - Resources
      - Single
      - multiple instance
  - edges
    - Assignment
    - request

process → resources [Request edge]

resources → process [Assignment edge] / allocation edge

Eg :-

$P = \{P_1, P_2, -- P_n\}$ [all process]

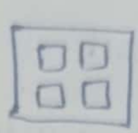
$R = \{R_1, R_2, -- R_n\}$ [all resources]

$P_i \rightarrow R_i$ [request edge]

$R_i \rightarrow P_i$ [assignment edge]

Representation :-

process :- ○

Resource Type with 4 instance :- 

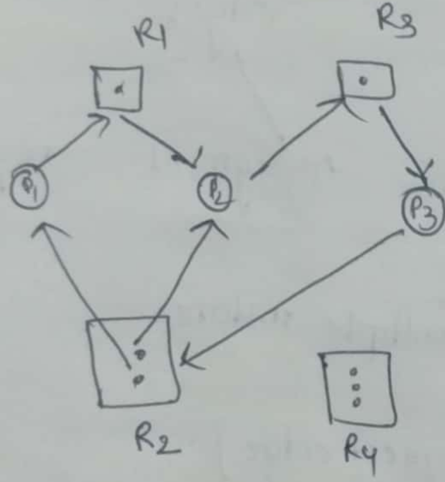
$P_i$ request instance of $R_j$

$$P_i \longrightarrow \boxed{R_j}$$

$R_j$ assignment instance of $P_i$

$$\boxed{R_j} \longrightarrow P_i$$

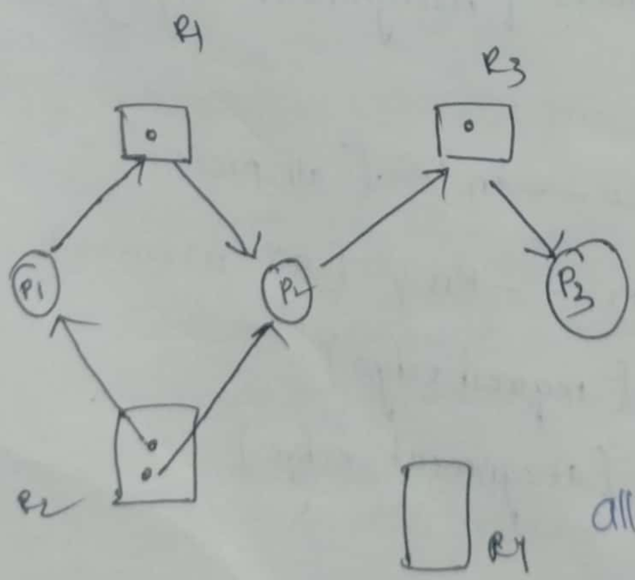
Eg:-




Deadlock situation arise [two cycles]

Hint :- if cycles present deadlock situation will arise

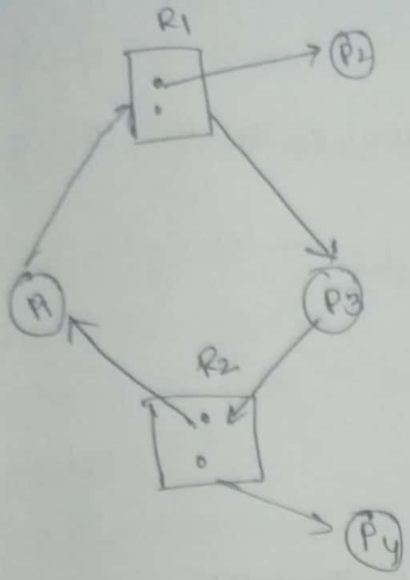
Eg :-




∴ No Deadlock

all process will complete execution.

Hint :- no cycle.




$P_1 \rightarrow R_1$, $R_1 \rightarrow P_3$,

$P_3 \rightarrow R_2$, $R_2 \rightarrow P_1$

[cycle is present]

Execution :- $P_4$, $P_3$, $P_2$, $P_1$

∴ No Deadlock

Hint:-

no cycles → no deadlock

cycles → single instance, deadlock

→ multiple instance, may or may not have deadlock

## Deadlock Prevention :-

If we break atleast 1 condition [mutual exclusion, Hold & wait, No preemption & circular wait] then Deadlock is prevented.

i) Mutual exclusion break Condition :-

make resources sharable but some resources are not sharable (printer).

ii) Hold & wait :-

whenever a process request a resource, it does not hold any other resource.

low resource utilization and starvation problem.

iii) No preemption.

preemption [time slice

iv) Circular wait

process as to request the resources in increasing order.

Every resource is given an order

## Deadlock Avoidance:-

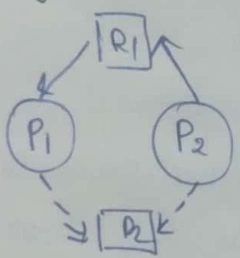
Single Instance [Resource allocation graph]

Multiple Instance [Bankers Algorithm]

- ** Safety
- Resource request

## Single Instance:-

Eg:-

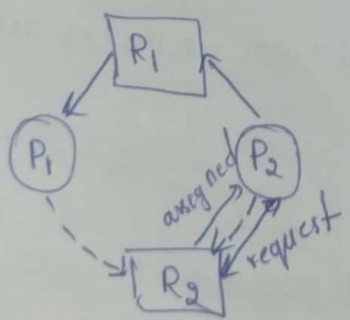



--→ [claim edge]

in future $P_1$ and $P_2$ can request $R_2$

$R_1 \rightarrow P_1$

$P_2 \rightarrow R_1$




[claim edge to request edge]

↑

in future $P_2$ requested if $P_1$ also request then cycle will form.

then it is <u>unsafe</u>

claim to request --→ to →

request to assignment --→ to →

Multiple Instance :-

## Bankers Algorithm :-

Safety Algorithm :-

allocation of resources to process it checks before whether it is safe or not.

formulas :-

1) Work = Available

2) Need ≤ Work True

3) work = work + Allocation.

Datastructures in Bankers Algorithm :-

~~process~~, Allocation, max, Need, Available → matrix

| Process | Allocation A B C | max A B C | Need [max − allocation] A B C | Available A B C |
|---|---|---|---|---|
| $P_0$ | 0 1 0 | 7 5 3 | 7 4 3 | 3 3 2 |
| $P_1$ | 2 0 0 | 3 2 2 | 1 2 2 | |
| $P_2$ | 3 0 2 | 9 0 2 | 6 0 0 | |
| $P_3$ | 2 1 1 | 2 2 2 | 0 1 1 | |
| $P_4$ | 0 0 2 | 4 3 3 | 4 3 1 | |
| | 7 2 5 | | | |

No. of instance :- | A = 10 | B = 5 | C = 7 |

Safe sequence :- all process will get resources in any order. so that deadlock will not be occur.

Allocation :- no. of resources allocated

max :- no. of resources demanding

need :- no. of resources still they required

$$\boxed{Need = Max - Allocation}$$

→ Available

| | | |
|---|---|---|
| 10 | 5 | 7 |
| 7 | 2 | 5 |
| 3 | 3 | 2 |

$$\boxed{Allocation + Total - Allocation}$$

→ i) work = available

work = [3 3 2]

Safe Sequence

$P_1, P_3, P_4, P_0, P_2$

~~For~~ Need ≤ work

$P_0$ : [7 4 3] ≤ [3 3 2] ×

$P_1$ : [1 2 2] ≤ [3 3 2] ✓

work = work + allocation

= [3 3 2] + (2 0 0)

work = [5 3 2]

$P_2$ : [6 0 0] ≤ [5 3 2] ×

$P_3$ : [0 1 1] ≤ [5 3 2] ✓

work = work + allocation

= [5 3 2] + [2 1 1] = [7 4 3]

work = [7 4 3]

$P_4$ : [4 3 1] ≤ [7 4 3] ✓

work = work + allocation

work = [7 4 3] + [0 0 2] = [7 4 5]

P0 :- [7 4 3] ≤ [7 4 5] ✓

work = work + allocation
= [7 4 5] + [0 1 0]
work = [7 5 5]

P2 :- [6 0 0] ≤ [7 5 5] ✓

work = [7 5 5] + [3 0 2] = [10, 5, 7]

work = [10, 5, 7] → same instance (given instance)

| Process | Allocation A B C | Max A B C | Need A B C | Available A B C |
|---|---|---|---|---|
| P0 | 1 1 2 | 4 3 3 | 3 2 1 | 2 1 0 |
| P1 | 2 1 2 | 3 2 2 | 1 1 0 | |
| P2 | 4 0 1 | 9 0 2 | 5 0 1 | |
| P3 | 0 2 0 | 7 5 3 | 7 3 3 | |
| P4 | 1 1 2 | 1 1 2 | 0 0 0 | |
| | 8 5 7 | | | |

A = 10, B = 6, C = 7

10 6 7
8 5 7
———
2 1 0

Safe Sequence :-
P1, P4, P0, P2, P3

work = available
work = [2 1 0]
Need ≤ work
P0 : [3 2 1] ≤ [2 1 0] ✗
P1 : [1 1 0] ≤ [2 1 0] ✓
work + allocation
work = [2 1 0] + [2 1 2]
work = [4 2 2]
P2 : [5 0 1] ≤ [4 2 2] ✗

P3 :- [7 3 3] ≤ [4 2 2] ✗
P4 : [0 0 0] ≤ [4 2 2] ✓
work = [4 2 2] + [1 1 2]
work = [5 3 4]
P0 : [3 2 1] ≤ [5 3 4] ✓
[5 3 4] + [1 1 2] = [6 4 6]
P2 : [5 0 1] ≤ [6 4 6] ✓

# Resource Request Algorithm:-

Steps:-

1) Request ≤ Need
2) Request ≤ Available
3) Allocation = Allocation + Request
4) Available = Available - Request
5) Need = Need - Request

Q) A B C [3 resources total instance]
10 5 6

| | Allocation | Max | Need | Available |
|---|---|---|---|---|
| $P_0$ | 111 | 433 | 322 | 210 / 110 |
| $P_1$ | 212 / 312 | 422 | 210 / 110 | |
| $P_2$ | 401 | 902 | 501 | |
| $P_3$ | 020 | 753 | 733 | |
| $P_4$ | 102 | 102 | 000 | |
| | 846 | | | |

→ $P_1$ (1,0,0) If $P_1$ arise with $P_1$(1,0,0). Whether it is safe state or not.

Request ≤ Need [1,0,0] ≤ [210] ✓

Request ≤ Available [100] ≤ [210] ✓

Allocation = allocation + request
= [212] + [100]
= [312]

Available = available - request
= (210) - [100] = [110]

Request Need = Need - request
[210] - [100] = [110]

Safety Algorithm:-

Safe Sequence
$P_1$, $P_4$, $P_0$, $P_2$, $P_3$

work = avaible
work = 110
Need ≤ work

$P_0$: 322 ≤ 110 X

$P_1$: 110 ≤ 110 ✓

work = work + allocation
work = [110] + [312] = [422]

work = [422]

$P_2$: 501 ≤ 422 X

$P_3$:- 733 ≤ 422 X

$P_4$:- 000 ≤ 422 ✓

work = work + allocation

work = [422] + [102] = [524]

work = [524]

$P_0$ = 322 ≤ 524 ✓

work = [524] + [111] = [635]

~~$P_3$ = 733 ≤ 635~~

$P_2$ = 501 ≤ 635 ✓

work = [635] + [401] = [10, 3, 6]

$P_3$ = 733 ≤ 10, 36 ✓

work = [10 36] + [020] = [10, 5, 6]

| | Allocation | Max | Need | ~~allocation~~ available |
|---|---|---|---|---|
| $P_0$ | 010 | 753 | 743 | 332 |
| $P_1$ | 200 302 | 322 | 1022 | 020 |
| $P_2$ | 302 | 902 | 600 | |
| $P_3$ | 211 | 222 | 011 | |
| $P_4$ | 002 | 433 | 431 | |
| | 725 | | | |

10 5 7

230

$P_1(102)$

request ≤ Need

$[102] \leq [122]$ ✓

request ≤ Available

$[102] \leq [332]$ ✓

Allocation = Allocation + request

$= [200] + [102] = [302]$

Available = available − request

$= [332] - [102] = [230]$

Need = Need − request

$= [122] - [102] = [020]$

Safety algorithm:-

work = available

work = 230

Need ≤ work

Safe Sequence:

$P_1, P_3, P_4, P_0, P_2$

$P_0$ : 743 ≤ 230 ×

$P_1$ : 020 ≤ 230 ✓

work = work + allocation = [230] + [302] = [532]

work = [532]

$P_2$ : 600 ≤ 532 ×

$P_3$ : 011 ≤ 532 ✓

work = [532] + [211] = [743]

work = [743]

$P_4$ : 431 ≤ 743 ✓

work = [743] + [002] = [745]

work = [745]

$P_0$ : 743 ≤ 745 ✓

work = [745] + [010] = [755]

$P_2$ : 600 ≤ 755 ✓

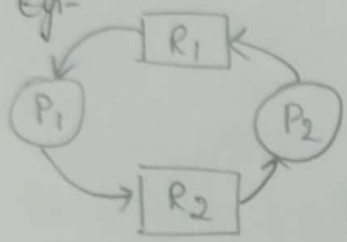
work = [755] + [302] = [10 5 7]

# Deadlock Detection:-

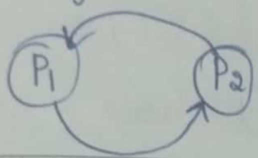
Single instance
[wait for graph]

Multiple instance
[Banker's Algorithm]

Eg-

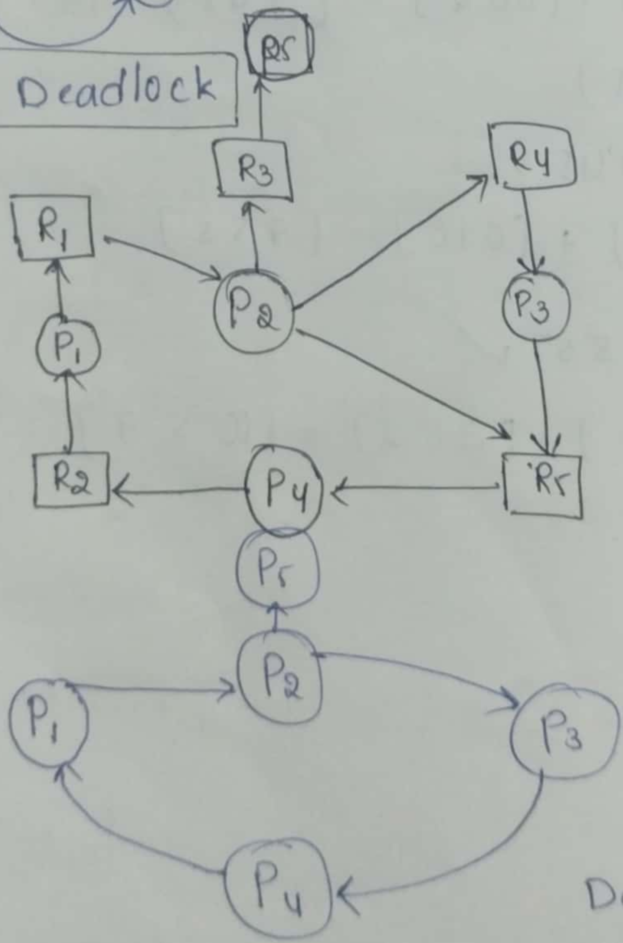



wait for graph:-




$\boxed{\text{cycle - Deadlock}}$

Q)




cycle is present
Deadlock is present

Multiple Instance :-

1) work = Available

2) Request ≤ work

3) work = work + allocation.

Available = 000

| process | allocation A B C | request |
|---|---|---|
| $P_0$ | 010 | 000 |
| $P_1$ | 200 | 202 |
| $P_2$ | 303 | 000 |
| $P_3$ | 211 | 100 |
| $P_4$ | 002 | 002 |

work = [0 0 0]

request ≤ work

$P_0$ : [000] ≤ [000] ✓

work = work + allocation

[000] + [010]

work = [0 1 0]

$P_1$ : [202] ≤ [010] ✗

$P_2$ : [000] ≤ [010] ✓

work = [010] + [303]

work = [313]

$P_3$ :- [100] ≤ [313] ✓

work = [313] + [211]

work = [5 2 4]

$P_4$ :- [002] ≤ [524] ✓

work = [524] + [0 02]

work = [5 2 6]

$P_1$ :- [202] ≤ [526] ✓

work = [526] + [200]

work = [7 26]

Safe Sequence :-

$P_0$, $P_2$, $P_3$, $P_4$, $P_1$

| Process | Allocation | Request | Available |
| --- | --- | --- | --- |
| $P_0$ | 2 1 2 2 | 1 4 4 4 | 3 0 1 1 |
| $P_1$ | 4 0 2 1 | 2 1 2 2 | |
| $P_2$ | 1 3 2 1 | 1 2 2 2 | |
| $P_3$ | 1 1 1 0 | 2 0 0 1 | |
| $P_4$ | 2 0 2 1 | 1 1 1 1 | |

work = available

work = [3011]

request ≤ work

$P_0$ :- [1444] ≤ [3011] x

$P_1$ : [2122] ≤ [3011] x

$P_2$ :- [1222] ≤ [3011] x

$P_3$ :- [2001] ≤ [3011] ✓

work = work + allocation

work = [3011] + [1110]

work = [4121]

$P_4$ :- [1111] ≤ [4121] ✓

work = [4121] + [2021]

work = [6142]

$P_0$ :- [1444] ≤ [6142] x

$P_1$ :- [2122] ≤ [6142] ✓

work = [6142] + [4021]

work = [10163]

Safe Sequence :-

$P_3$, $P_4$, $P_1$

~~$P_0$ :- [1444] ≤ [10163]~~

$P_2$ :- [1222] ≤ [10163] x

~~$P_3$ :- [2001] ≤ [10163]~~

$P_0$ :- [1444] ≤ [10163]

∴ Deadlock arised.

$P_0$, $P_2$ does not get any resources.

## Recovery from Deadlock :-

- Termination
  - Abort all process at a time
  - Abort one process at a time
- Resource preemption
  - Select a victim
  - Starvation
  - Rollback

**Abort all process at a time :-**

all process are removed at a time.

Disadvantage :- The work process has done is lost.

**Abort one process at a time :-**

Disadvantage :- Time consuming

Every time when process is abort then again we should apply deadlock detection algorithm.

**→ Resoure preemption :-**

Taking a resource from process.

Select a victim :-

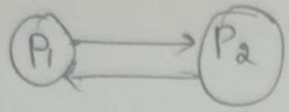
Based on priority [low priority process will give resources]

how long they have completed

what and how many resources.

How many process are terminating

which type of process (interactive or batch)

Eg:- 



If $P_1$ has higher priority then resource is taken from $P_2$.

If $P_1$ completes - 95%
$P_2$ completes - 50%
} resource is given to $P_1$

Rollback :- [undo operation]

One the resource is taken from the process . it starts executing from starting .

Starvation :-

single process

Every time if we take resource from $\underline{P_1}$ then $P_1$ should wait for long time then starvation occurs.