Banker's Algorithm or Deadlock Avoidance Method.

In this approach we first have to inform the OS how many process will come, how many resources they are going to request etc. There should be prior information given to the operating system.

So can call it as deadlock detection algorithm. Log of the system is in safe mode then no deadlock if it goes into unsafe mode then there exist deadlock.

Allocation :- how many Resources already allocated

Max need :- total no of Resources required for execution

Available :- How many resources are currently available.

Need :-) How many more resources are needed.

total :- A = 10, B = 5, C = 7

	Allocation			Maxneed			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2	7	4	3
P_1	2	0	0	3	2	2	2	0	0	1	2	2
P_2	3	0	2	9	0	2	5	3	2	6	0	0
P_3	2	1	1	2	2	2	2	1	1	0	1	1
P_4	0	0	2	4	3	3	7	4	3	4	3	1
	7	2	5				0	0	2			
							7	4	5			
							0	1	0			
							7	5	5			
							3	0	2			
							10	5	7			

$\langle P_1, P_3, P_4, P_0, P_2 \rangle$

1, 2, 2

1, 0, 2

0 2 0

	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

(Need = Maxneed - Allocation)

Available = (Available + Allocation.

Extra

For the previous problem if P_1 request one additional instance of resource type A and two instance of resource type C . i,e

Request$_1$ = (1, 0, 2)

Initially P_1 needs (1, 2, 2)

So first check whether the Request$_1$ <= Need$_1$; (1, 0, 2) <= (1, 2, 2) Yes.

Request$_1$ <= available ? (1, 0, 2) <= (3, 3, 2) yes then make need

	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	4	3	2	3	0
P_1	3	0	2	0	2	0	3	0	2
P_2	3	0	2	6	0	0	5	3	2
P_3	2	1	1	0	1	1	2	1	1
P_4	0	0	2	4	3	1	7	4	3
							0	0	2
							7	4	5
							0	1	0
							7	5	5
							3	0	2
	8	2	7				10	5	7

Safe state.

Now if Process P_4 requests for resource (3, 3, 0)

(3, 3, 0) <= (4, 3, 1) yes

(3, 3, 0) <= (2, 3, 0) NO

200 Alloc
+ 102 Req
302 Alloc

322 Maxneed
− 302 Alloc
020 need

<P_1, P_3, P_4, P_0, P_2>

332 Avail
− 102 Req
230 Avail

RRA : will not grant, pretend as if it has granted & apply safety algorithm & check will there be safe seqor not

need = need − Request s/m is pretending.

need was − − − & s/m is pretending it has granted the request so new need will be need − rqst

	Allocation			Maxneed			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2	7	4	3
P_1	2	0	0	3	2	2	5	3	2	1	2	2
P_2	3	0	2	9	0	2	7	4	3	6	0	0
P_3	2	1	1	2	2	2	7	4	5	0	1	1
P_4	0	0	2	4	3	3	7	5	5	4	3	1
	7	2	5				10	5	7			

total:- A = 10, B = 5, C = 7,

1> Need matrix

2> s/m is on safe state?

3> if req from P_1 arrives for (1, 0, 2) can request be immediately granted?

4> if req from P_4 arrives for (3, 3, 0) can request be granted?

5> if req from P0 arrives for (0, 2, 0) can be granted?

Safe sequence $\langle P_1, P_3, P_4, P_0, P_2 \rangle$

3> a Request$_1$ <= Need$_1$ yes (1, 0, 2) <= (1, 2, 2) ✓

b Request$_1$ <= Available (1, 0, 2) <= (3, 3, 2) ✓

	Allocation			Maxneed			Available			Need		
P_0	0	1	0	7	5	3	2	3	0	7	4	3
P_1	3	0	2	3	2	2	5	3	2	0	2	0
P_2	3	0	2	9	0	2	7	4	3	6	0	0
P_3	2	1	1	2	2	2	7	4	5	0	1	1
P_4	0	0	2	4	3	3	7	5	5	4	3	1
							10	5	7			

after a & b s/m will not allocate resource it pretends that it has allocated.

Need = Need − Rqst

(1, 2, 2) − (1, 0, 2)

① Allocation = Allocation + Rqst

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

Available = Available − Rqst

= (3, 3, 2) − (1, 0, 2)

$\langle P_1, P_3, P_4, P_0, P_2 \rangle$.

4> a. Request$_1$ <= needi (3, 3, 0) <= (4, 3, 1) yes

Request <= available (3, 3, 0) <= (2, 3, 0) NO

Not granted.

	Allocation	maxNeed	Available	Need
	0 3 0			7, 2, 3
P_0	0 3 0	7 5 3	2 1 0	7 2 3
P_1	3 0 2	3 2 2		0 2 0
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

(0, 2, 0) <= (7, 4, 3) yes
(0, 2, 0) <= (2, 3, 0) yes.

Assume res are allocated.

Allocation = Allocation + request 0 1 0 + 0 2 0
Available = Available - request 2, 3, 0 - 0, 2 0
Need = need - Request 7, 4, 3 - 0, 2, 0

Now find out safe sequence.
Sfm will enter into unsafe state so resource will not be granted.

0 2 0

Resource Request algorithm

1) If Requesti <= Needi, go to step 2; else Sfm will claim that Process is exceeding it resource needs.
2) If Requesti <= available, go to step 3; Pi must wait resources are not available
3) If both conditions are true then Sfm pretends to have allocated the rqstd resources to Process Pi & does the modification

Allocation = Allocated + Rqst
Available = Available - Rqst
Need = Need - Rqst

4) By doing this if it results in safe sequence generation then Sfm will allocate the resource if not it will not allocate the resource, new state is unsafe so previous state is restored.

Deadlock detection

If a system does not employ deadlock detection prevention or avoidance algorithm then definatly at one point of time deadlock may occur. To check whether the system is in safe state or not deadlock detection algo is used. These algorithm examins the state of the system at that time & determines deadlock has occured or not.

There are 2 types of deadlock detection algorithm.

1) Single Instance of Resource :- Algo used to detect is wait for graph
2) Multiple Instance of Resource :- Bankers algorithm is used.

1) wait for graph (RAG) for single instance of resources.

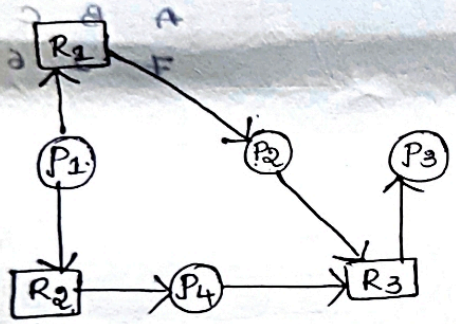
Difference b/w RAG & wait for graph.

In RAG we have Process & Resources

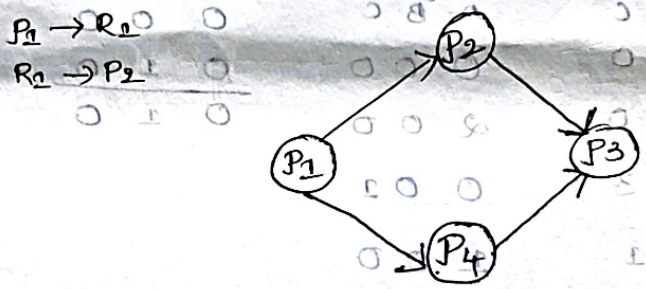
In wait-for graph only process will be der.

In wait for graph detect whether there exist a cycle. If yes then there is a deadlock. cycle is sufficient condition.

1)



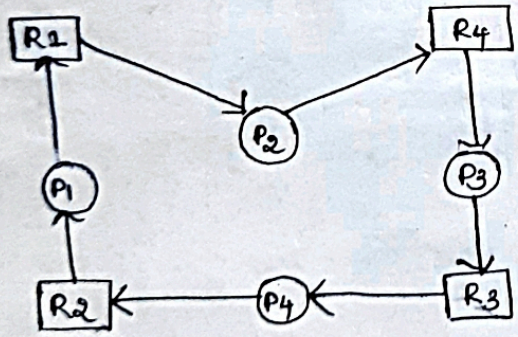
RAG

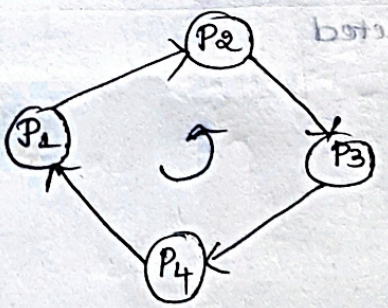


wait-for graph

No cycle - No deadlock.

2)





cycle - deadlock.

Instead of calling deadlock detection for every new request call it after some fixed interval. but many may be deadlocked in the time interval.

Bankers algo is used when multiinstance of resource are available.

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
							0	0	0
P_0	0	1	0	0	0	0	0	1	0
							0	1	0
P_1	2	0	0	2	0	2	3	0	3
							3	1	3
P_2	3	0	3	0	0	0	2	1	1
							5	2	4
P_3	2	1	1	1	0	0	0	0	2
							5	2	6
P_4	0	0	2	0	0	2	2	0	0
							7	2	6

total

A	B	C
7	2	6

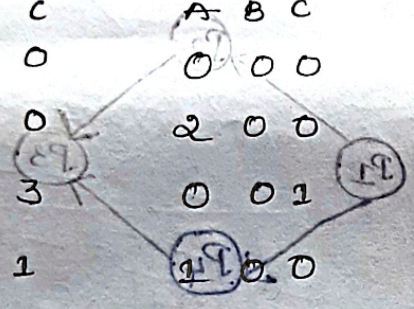
$\langle P_0, P_2, P_3, P_4, P_1 \rangle$ = No deadlock.

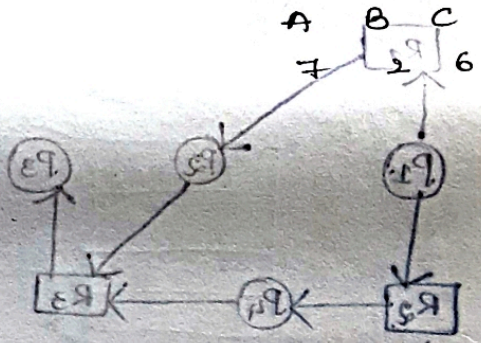
Now assume P_2 makes an additional rqst for resource type C

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
							0	0	0
P_0	0	1	0	0	0	0	0	1	0
							0	1	0
P_1	2	0	0	2	0	0			
P_2	3	0	3	0	0	1			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

total

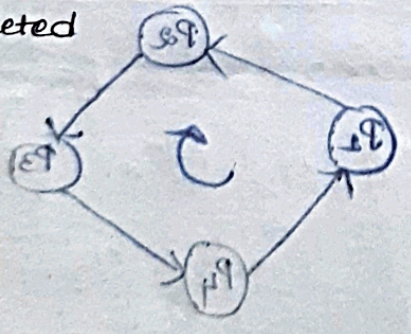
A	B	C
7	2	6

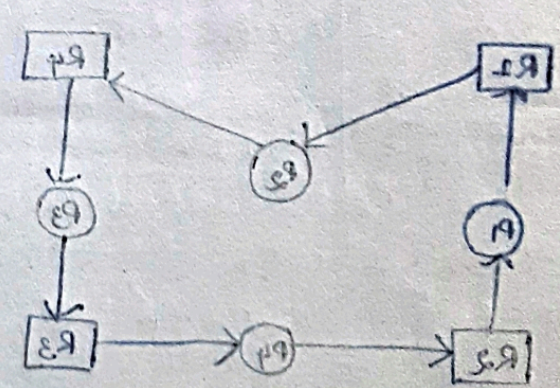




RAG

$\langle P_0 \rangle$ Deadlock state. $\langle P_1, P_2, P_3, P_4 \rangle$ completed





Deadlock Prevention

To handle deadlock there are 4 methods.

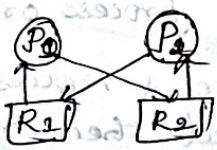
1) Deadlock Prevention
2) Deadlock Avoidance
3) Deadlock Detection
4) Deadlock Recovery (ignorance)

Will see how by using deadlock Prevention how can we violate any one of the necessary conditions for deadlock. So that there wont be deadlock.

Prevention is a concept which makes sure that there wont be deadlock in the system.

Cost for prevention is more, so use it where there are too many chances of deadlock. (Aircraft) (Real time systems)

1) Mutual Exclusion:-



we cannot avoid mutual exclusion.

Consider non sharable resource (printer)

Can any two process access printer at same time? NO

Can a resource be used by 2 process it depends on hardware properties.

Cannot violate ME

2) Hold & Wait : A process to be executed it needs no of resources. If all the resources are not available also process will acquire the resources which are available & then wait (2) of which deadlock can occur.

In prevention method we can violate H&W using three methods having its own advantage & disadvantages.

a) conservative approach :- $P_1 \rightarrow R_1$ $P_2 \rightarrow R_2$.

P_1 acquired R_1 executed something & then went to acquire R_2. But before that P_2 acquired R_2 (If not it will relese R_1)

So let process not start the execution till it gets all the resources.

$P_1 \rightarrow R_1, R_2$ (A process is allowed to start the execution if & only if it has acquired all the resources)

Not efficient coz i want cplt scanner Printer. Even thoug i use scanner & kill then Printer is of no use it could have been used by other process
Not implementable :- Process may not get all process resources. cannot estimate
Easy & deadlock free.

b. Do not hold :- Process wants 10 but for now 4 so hold 1 to 4 & start then when u need remaining release R_1 to R_4 then acquire R_5 to R_{10}
So any process may get all the resources.

c. Hold time out :- hold & wait for the resources for small time quantum
If resource are not allocated then release.

Three approaches to violate hold & wait.

No preemption :- A process cannot snatch the resources of another process.
we have to violate this how?
It can be used by only high priority process or system process.

1) If a process is holding some resources & request for some other resource that cannot be immediately allocated then relese the resource held by requesting process & add it to list of resources for which other process are waiting for.

2) If a process request resources check can resource be allocated. If Cannot then check of the resources are with waiting process. If yes then allocate to the requesting process.

Cirular wait how to violate? & can save system entering into deadlock.
If there exist a cycle then there can be cirular wait.
what ever resources available let us no them.
Process can rqst in either increasing or decreasing order. Acyclic graph

(R_2, R_{21}, R_{41}) P_1
(R_3, R_5) P_2

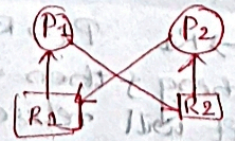
P_1, R_2, R_3, R_{21}, R_{41}

$R_2 \leftarrow P_1(R_1, R_2)$
$R_2 \leftarrow P_2(R_1, R_2)$ x

P_2 will not get
So no cycle.

(R_{41}, R_{21}, R_{41}) It got all 3. now it wants R_{31} So relese resources above 31
R_{31} Relese 41, then 31 can be allowed.

$P_1 \rightarrow R_1$ $P_2 \rightarrow R_2$
$P_2 \rightarrow R_2$ $P_2 \rightarrow R_1$
change the order of request
$P_1 \rightarrow R_1$ $P_2 \rightarrow R_1$
$P_2 \rightarrow R_2$ $P_2 \rightarrow R_2$



Both process will try for R_1
Assume P_2 tries & gets R_1. Now P_2 tries for R_1 it will not & it will not hold R_2 also.
If rqst order is same then we can avoid deadlock.
Problem is we dont no which process wants how many resources how long and on which order it wants to & generalize this one method is der
(31, 41) & 41 not allowed release all resources.

	Allocation				Max need				Available				Need
	A	B	C	D									
P_0	2	0	0	1	4	2	1	2	3	3	2	1	
P_1	3	1	2	1	5	2	5	2					
P_2	2	1	0	3	2	3	1	6					
P_3	1	3	1	2	1	4	2	4					
P_4	1	4	3	2	3	6	6	5					

1) Need matrix

2) Safe sequence

3) Reqst from P_1 arrives (1, 1, 0, 0) Resource granted? Yes

4) Request from P_4 arrives (0, 0, 2, 0) Resource granted? No.