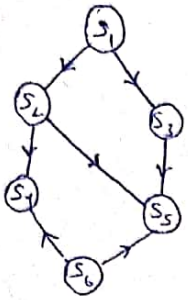
35 - GATE 1993 Question on Concurrent processes and Semaphores



begin
co begin
begin S1 V(a), V(b) end
begin P(a) S2 V(c) V(d) end
begin P(b) S3 V(e) end
begin P(c) P(d) S4 end
begin P(e), P(e) S5 end
begin S6 V(f) V(g) end
co end
end

No. of incoming edges = downop
No. of outgoing edges = upop

* 03 - Deadlocks

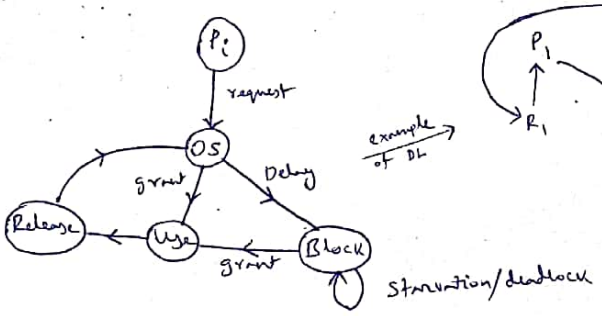
01 - Introduction to deadlocks

Necessary Conditions for DL

- → Mutual Exclusion
- → Hold & Wait
- → No preemption
- → Circular Wait

Deadlock

- → A set of processes are said to be in deadlock, if they wait for happening of an event caused by others in the same set.
- → Starvation is long waiting.
- → Deadlock is infinite waiting.



02 - GATE 1997 Question & More Examples

a) A system is having 3 user processes, each requiring ~~two~~ 2 units of resource 'R'. The minimum number of ~~resources~~ units of 'R' such that no deadlock will occur - a) 3, b) 5, c) 4, d) 6

Ans → C

Max # resources cause deadlock → 3
Min " " no " → 4

P_1 P_2 - - - - - - P_n

x_1 x_2 - - - - - - x_n → ~~Allocated~~ need

(x_1-1) (x_2-1) (x_n-1) → Allocated

$\left(\sum_{i=1}^{n} x_i\right) - n$ → deadlock

↳ +1 → No deadlock

P_1 P_2 P_3

2 3 4 → need

① + ② + ③ → given

∴ 6 → max resources that cause deadlock

∴ 6+1 → 7 → min resources that cause no deadlock

a) $R = 6$; $P_i = 2$ [i.e each process needs 2 resources]

Then → how many processes can be present at max so that deadlock doesn't occur?

P_1	P_2	P_3	P_4	P_5	R_c
1	1	1	1	1	1

∴ 5 processes max

b) $R = 6$; $P_i = 3$

P_1	P_2	P_3
2	2	2

∴ 2 processes max

c) $R = 100$; $P_i = 2$

∴ 99 processes max

d) $R = 100$; $P_i = 3$

∴ 49 processes max

$\left[\frac{\text{Resources}}{(P_i - 1)}\right] - 1$ → max processes present (min)

e) $R = 100$; $P_i = 4$

∴ 33 processes max

$n \times (P_i - 1) = 100$

$\Rightarrow n \times 3 = 100$

$\Rightarrow n = \left\lceil \frac{100}{3} \right\rceil$ / $\left\lfloor \frac{100}{3} \right\rfloor$

↓ max processes for deadlock

↓ min processes that cause no deadlock

* i) $\left[\frac{\text{Resources}}{(P_i - 1)}\right] - 1$ → Max no. processes that can present.

ii) $\left[\frac{\text{Resources}}{(P_i - 1)}\right]$ → Min no. processes that cause deadlock

f) $R = 100$; $P_i = 5$ → $n = 24$ (i)

g) $P = 10$, $P_i = 3$, $R = ?$ → $10 \times 2 + 1 = 21$ units for no deadlock

(No. of processes $\times (P_i - 1)) + 1$ → min no. resources for no deadlock.

(No. of processes $\times (P_i - 1)$) → Max no. resources for deadlock

P_1 : 21 P_2 : 31 P_3 : 41

min no. resources such that no deadlock occurs?

P_1 P_2 P_3

$20 + 30 + 40$

↳ $= 90 + 1 = 91$ min no of resources

$\left(\sum_{i=1}^{n} (P_i - 1)\right) + 1$ or $\sum_{i=1}^{n} P_i - n + 1$ → min no. of resources for no deadlock

$\sum_{i=1}^{n} (P_i - 1)$ or $\sum_{i=1}^{n} P_i - n$ → max no. of resources for deadlock

03 - GATE 1992 Question

A computer system has 6 tape drives, with n processes ~~competing~~ competing for them. Each process needs 3 tape drives. The maximum value ~~of~~ of n for which the system is guaranteed to be deadlock free —

a) 2, b) 3, c) 4, d) 1

$$R = 6,\ P_i = 3 \rightarrow \left\lceil \frac{6}{3-1} \right\rceil - 1 = 2 \text{ processes max}$$

04 - GATE 1993 question on Minimum Resources Required

Consider a system having 'm' resources of the same type. These resources are shared by 3 processes A, B, and C, which have peak demands of 5, 4, and 6 respectively. For what value of 'm' deadlock will not occur? a) 7, b) 9, c) 10, d) 13

Processes →	A	B	C
Demands →	3	4	6

② + ③ + ⑤ = 10 + 1 = 11 min required for avoiding deadlock

∴ Ans ⇒ d → 13

05 - GATE 2005 Question

Suppose 'n' processes $P_1 \dots P_n$ share 'm' identical resource units, which can be reserved and released one at a time. The maximum resource requirement of process P_i is S_i where $S_i > 0$. Which one of the following is a sufficient for ensuring that deadlock doesn't occur.

a) $\forall i,\ S_i < m$, b) $\forall i,\ S_i < n$, c) $\sum_{i=1}^{n} S_i < (m+n)$, d) $\sum_{i=1}^{n} S_i < (m \times n)$

m resources

P_1 P_2 ---- P_n

S_1 S_2 ---- S_n

max 'm' → Deadlock

P_1 P_2 ---- P_n

(S_1-1) (S_2-1) ---- (S_n-1)

$$m \geq \sum_{i=1}^{n} S_i - n \rightarrow \text{Max 'm'}$$

$$\sum_{i=1}^{n} S_i < m+n \quad \text{Ans} \rightarrow c$$

06 - GATE 06 question about necessary condition for Deadlock

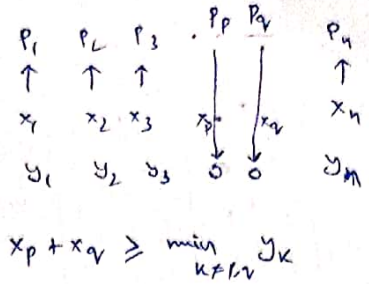
Consider the following snapshot of a system running 'n' processes. Process 'i' is holding 'x_i' instances of a resource 'R' for $1 \leq i \leq n$. Currently all instances of 'R' are occupied. Further, for all 'i', process 'i' has placed a request for an additional y_i instances it already has. There are exactly two processes 'p' and 'q' such that $y_p = y_q = 0$. Which of the following can be serve as necessary condition to guarantee that the system is not approaching a deadlock.

a) $\min(x_p, x_q) < \max_{k \neq p,q} y_k$, b) $x_p + x_q \geq \min_{k \neq p,q} y_k$, c) $\max(x_p, x_q) \geq 1$, d) $\min(x_p, x_q) > 1$

R = 9

	P_1	P_2	P_3
	5	4	6
alloc	3	4	2
req	2	0	4

(4)



$x_p + x_q \geq \min_{k \neq p,q} y_k$

Ans → B It is necessary condition &, but not sufficient condition

07 – Deadlock Handling Mechanisms

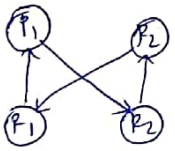
* 1) Deadlock Ignorance → Solved by restarting the system
2) Deadlock Prevention → Solved by disabling one of the necessary conditions
3) Deadlock Avoidance → Solved by Banker's Algorithm
* 4) Deadlock Detection & Recovery → Solved by detection mechanism periodically & recover

* Deadlock Ignorance is also called Austridge Approach.

08 – Deadlock Prevention

	Condition	Approach
*	Mutual Exclusion	Spool Everything
*	Hold & Wait	Request all resources initially
*	No preemption	Take resources away
✓	Circular Wait	Order resources numerically

[* → not practically possible]



So, using mutual exclusion deadlocks can be prevented to some extent, but not always.

* Spooling : There is a buffer (queue) at which all jobs are queued & from there jobs are selected by one by one & give to the printer. It is handled by a system deamon process (printer).

✓ !(Hold & wait)

≥ !Hold or !wait (If process does not get all resource then don't hold or don't wait)

But, practically it is impossible for a process to say about all the resources it needs during execution.

* No preemption → take all resources → process cannot be restarted → so not possible / not good approach

* Circular Wait → Process can only ask for a resource number greater than the previous resource no. numerically. or equal

To all the resources → give them ordering → It can be implemented practically → So no cycle will occur generally used in RTS

09 – Safe, Unsafe, Deadlock avoidance & Banker's Algo

Total no. of resource instances present

Resources allocated

Need after allocated

$E = (6\ 3\ 4\ 2)$ — T P S C

$P = (5\ 3\ 2\ 2)$

$A = (1\ 0\ 2\ 0)$

Process	Tape drives	Printers	Scanners	CD ROM's
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Printers	Scanners	CD ROM's
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

Available → 1 0 2 0

D → 0 0 1 0

Available after alloc → 1 0 1 0

After completion of D → 2 1 2 1 (Allocated + Need + Available)

Available → 2 1 2 1

A → 1 1 0 0

1 0 2 1

After completion of A → 5 1 3 2

B → 0 1 1 2

5 0 2 0

After B's comp → 5 2 3 2

C → 3 1 0 0

2 1 3 2

after C's comp → 6 3 4 2

E → 2 1 1 0

4 2 3 2

after E's comp → 6 3 4 2

DABCE → Safe State

B = (0 0 1 0)

Allocate this first & then check the system state is safe or not.

E = (6 3 4 2)

P = (5 3 3 2)

A = (1 0 1 0)

Now check it like previous ~~method~~ one

∴ This state is also safe.

Now after this E = (0 0 1 0)

E = (6 3 4 2)

P = (5 3 4 2)

A = (1 0 0 0)

→ Not Safe State → Deadlock if (0 0 1 0) is assigned to E

* This is Banker's Algorithm

10 – GATE 2007 Question on Safe State

	alloc			request		
	X	Y	Z	X	Y	Z
P_0	1	2	1	1	0	3
P_1	2	0	1	0	1	2
P_2	2	2	1	1	2	0

There are 5 units of each resource type.

~~T = (5 5 5)~~

~~A = (0 0 0)~~

Total = (5 5 5)

Alloc = (5 4 3)

Avail = (0 1 2)

(0 1 2) → P_1 → (2 1 3) → P_0 → (3 3 4) → P_2 → (5 5 5)

∴ The state is safe.

11 - Question on Safe State

	Allocated	Maximum	Future need
PA	1 0 2 1 1	1 1 2 1 3	0 1 0 0 2
PB	2 0 1 1 0	2 2 2 1 0	0 2 1 0 0
PC	1 1 0 1 1	2 1 3 1 1	1 0 3 0 0
PD	1 1 1 1 0	1 1 2 2 0	0 0 1 1 0

Available

0 0 X 1 1

Q) What is the smallest value of 'x' for which this is a safe state.

if x = 1, then PD can be satisfied, but after that PA, PB and PC could not be satisfied.

if x = 2, then PA, PB, PC, PD all can be satisfied.

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

∴ Ans → x = 2

12 - GATE 2014 question on Bankers Algorithm

	Alloc			MAX			Need		
	x	y	z	x	y	z	x	y	z
P_0	0	0	1	8	4	3	8	4	2
P_1	3	2	0	6	2	0	3	0	0
P_2	2	1	1	3	3	3	1	2	2

} derived → (MAX - ALLOC)

Available : (3, 2, 2) → P_1 → (6, 4, 2) → P_2 → (8, 5, 3) → P_0 → (8, 5, 4)

Req 1 : P_0 (0, 0, 2)

Req 2 : P_1 (2, 0, 0)

($P_1 \rightarrow P_2 \rightarrow P_0$) Safe Sequence

After Req 1 ~~flag~~ :-

P_0 :	0 0 3	8 4 3	8 4 0	Avail → (3, 2, 0)
P_1 :	3 2 0	6 2 0	3 0 0	→ Not Safe
P_2 :	2 1 1	3 3 3	1 2 2	so req 1 will not be granted

After Req 2 :-

P_0 :	0 0 1	8 4 3	8 4 2	Avail → (1, 2, 2)
P_1 :	5 2 0	6 2 0	1 0 0	⇒ Safe State
P_2 :	2 1 1	3 3 3	1 2 2	so req 2 will be granted

13 - GATE 1996 on Bankers Algorithm

	Alloc			Need			Future Need		
	R_0	R_1	R_2	R_0	R_1	R_2	R_0	R_1	R_2
P_0	1	0	2	4	1	2	3	1	0
P_1	0	3	1	1	5	1	1	2	0
P_2	1	0	2	1	2	3	0	2	1

Available : (2 2 0) → P_1 → P_2 → P_0 → Safe State

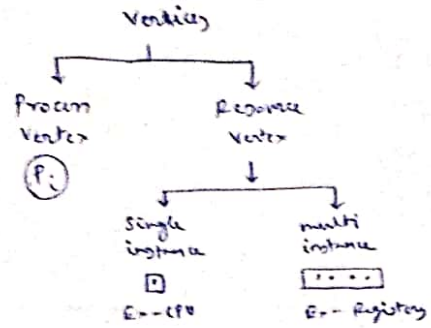
Req : P_0 (0 1 0) ↘

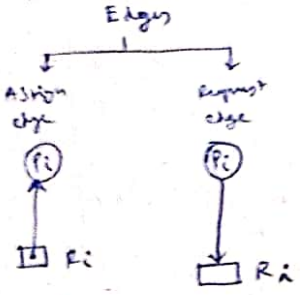
Available = (2 1 0)

	Alloc			MAX			NEED		
	R_0	R_1	R_2	R_0	R_1	R_2	R_0	R_1	R_2
P_0 :	1	1	2	4	0	2	3	0	0
P_1 :	0	3	1	1	5	1	1	2	0
P_2 :	1	0	2	1	2	3	0	2	1

No process cannot be allocated. So, unsafe state & request cannot be granted.

14 – Resource Allocation Graph

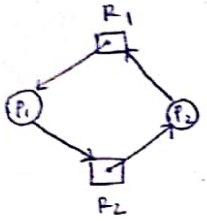




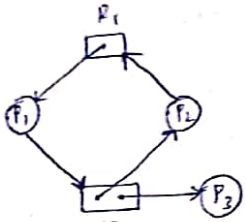
* RAG represents the state of a system in terms of processes & resources.
* RAG is visually easy, so deadlock can be detected easily if no of processes & resources are less.

* dot represents no. of instances of a particular resource

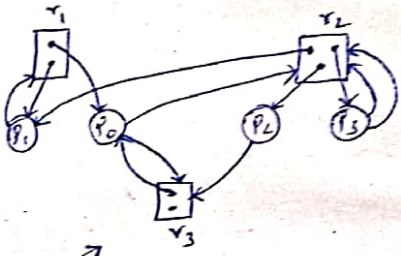
15 – Resource Allocation Graph Examples



Single instance resource type graph



Multi instance resource type graph



- If there is a cycle, then there is a deadlock

	Alloc		Req	
	P_1	P_2	P_1	P_2
P_1	1	0	0	1
P_2	0	1	1	0

Available : (0, 0)

- In case of RAG, ~~cycle~~ with multi instance resource types, cycle is necessary condition ~~for~~ Deadlock but not sufficient

Cycle is not a sufficient condition for a deadlock

	Alloc		Req	
	P_1	P_2	P_1	P_2
P_1	1	0	0	1
P_2	0	1	1	0
P_3	0	1	0	0

Available : (0, 0)

00
P_3 01
01
P_1 10
11
P_2 01
$\phi 2$

- No deadlock & Safe state if P_3's req was 1 0, then there was a deadlock.

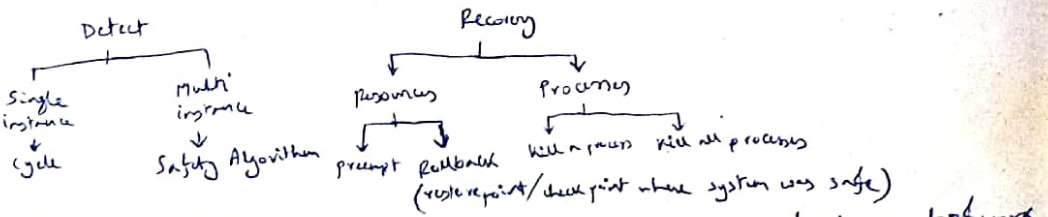
	Alloc			Req		
	r_1	r_2	r_3	r_1	r_2	r_3
P_0	1	0	1	0	1	1
P_1	1	1	0	1	0	0
P_2	0	1	0	0	0	1
P_3	0	1	0	0	2	0

Available : (0, 0, 1)
P_2 : (0, 1, 0)
(0, 1, 1)
P_0 : (1, 0, 1)
(1, 1, 2)
P_1 : (1, 1, 0)
(2, 1, 2)
P_3 : (0, 1, 0)
(2, 2, 2)

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

- Cycle doesn't mean there is always a deadlock, but if there is a deadlock then ~~there~~ ~~if~~ cycle is surely present there.

16 – Deadlock Detection & Recovery



* To kill a process, a process must be chosen that would do least amount of work