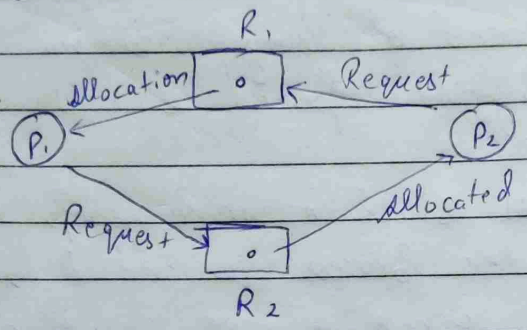
Date: __/__/____

MON TUE WED THR FRI SAT SUN

Subject: _____

Deadlock

If two & more process are waiting on happening of some event which never happens then we say these processes are involved in deadlock and that state is called Deadlock state.



Necessary Conditions for deadlock

1) Mutual exclusion
2) No preemption
3) Hold & Wait
4) Circular Wait

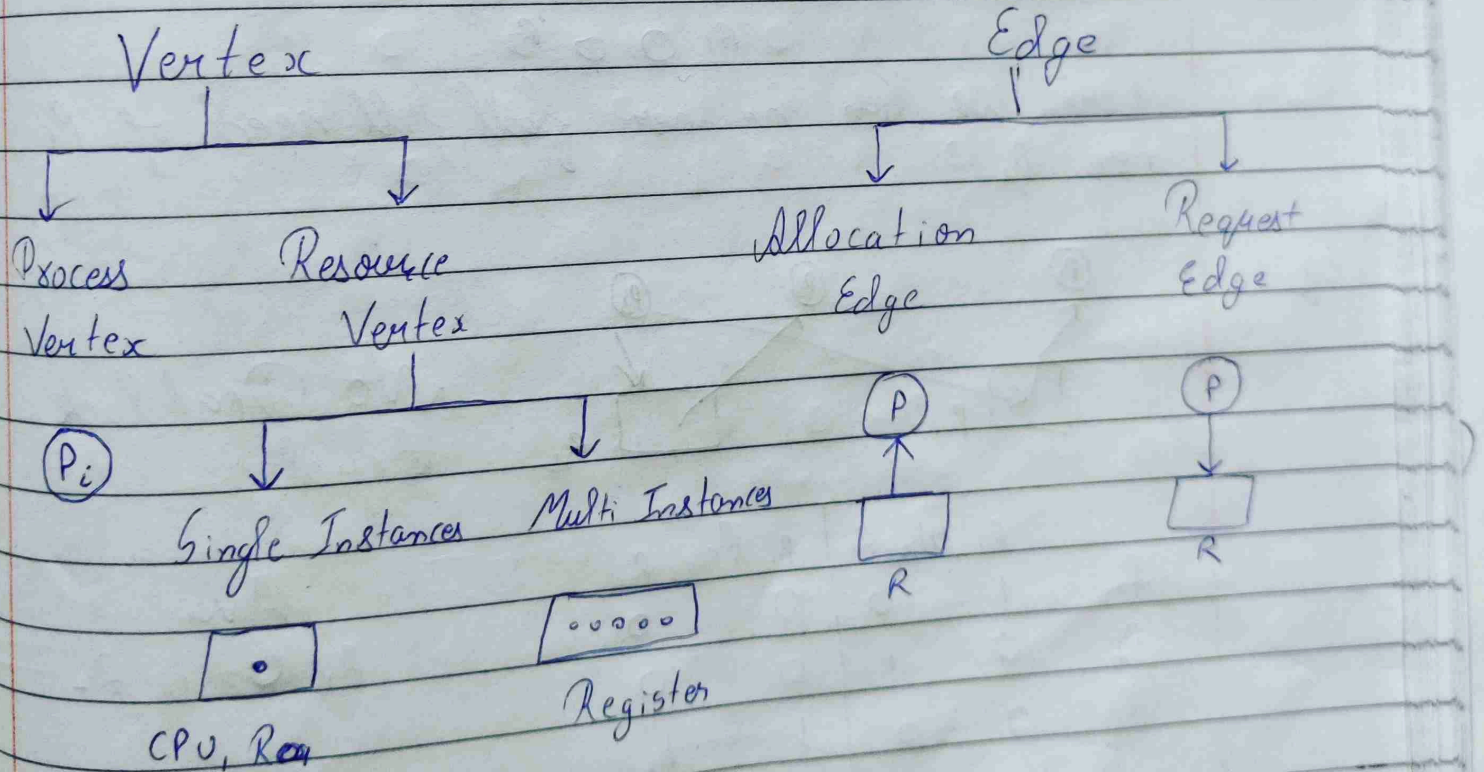
Mutual exclusion:- All the resources that are used by a process must be used in a mutual exclusive manner. It means when one process is using that resources no other process can use that resources.

No preemption:- If a process is holding any resources that it should not be released until its executed whether we get high priority process we do preempt that process.

Hold & Wait:- It means a process is holding a particular resource and also at same time waiting for other resources to get granted. It will not leave hold resource.

Circular Wait:- We there is a cycle in resource allocation graph its called circular wait. It means a process holding a resource R_1 & wait for R_2 & at same time other process hold R_2 & wait for R_1 then its a circular wait.

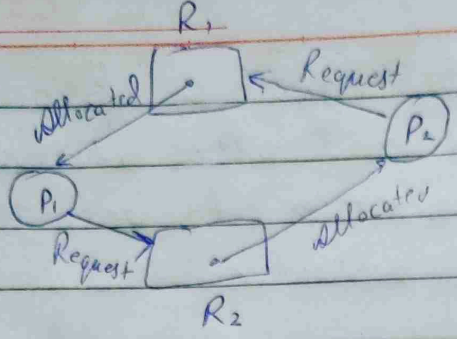
Resource Allocation Graph (RAG)



Date: __/__/__

MON TUE WED THR FRI SAT SUN

Subject : _____



Single Instance

Circular Wait Cycle

Deadlock Situation

Method to check deadlock in above eg

	Allocate	Request
P_1	R_1	R_2
P_2	R_2	R_1

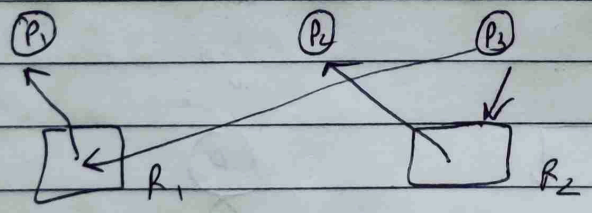
	Allocate		R.A
P_1	1	0	0 1
P_2	0	1	1 0

~~Available~~ Availability

R_1 0 0 R_2

Here we can neither full fill need of P_1 nor P_2

g.



No Deadlock

Acyclic

	R1 Allo.	R2	R1 Req.	R2
P_1	1	0	0	0
P_2	0	1	0	0
P_3	0	0	1	1

Ava

R_1 0 0 R_2

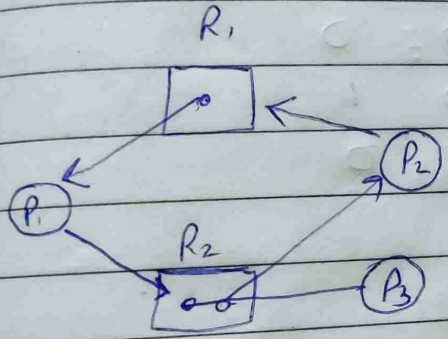
Here P_1 & P_2 not req. any thing then they can be executed after getting back resource allocated to them then we can execute P_3 also.

Waiting ⟶ finite ⇒ Starvation
↳ Infinite ⇒ Deadlock

If RAG has Circular Wait (cycle)
⇑ ⇒ Always Deadlock
Single Instance

If RAG has no cycle then no deadlock.

Multi Instance RAG



Circular Wait

No deadlock

	Allocated R_1	Allocated R_2	Need R_1	Need R_2
P_1	1	0	0	1
P_2	0	1	1	0
P_3	0	1	0	0

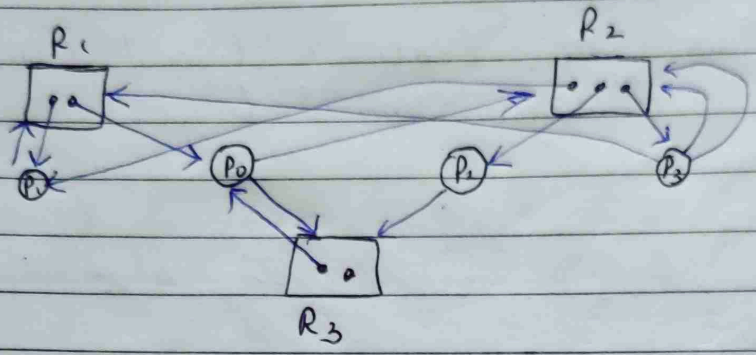
Ava
R_1 0 R_2

first P_3 get executed then resources assign to it got free then P_1 got executed & after getting resources of P_1 also we can fullfill or execute P_2 also.

PAGE NO.

In MI if there is a cycle in then there is no necessary deadlock.

Eg.



No deadlock

	Allo R1	R2	R3	Need R1	R2	R3
P0	1	0	1	0	1	1
P1	1	1	0	1	0	0
P2	0	1	0	0	0	1
P3	0	1	0	1	2	0

Avail

R1	R2	R3
0	0	1

So first P2 execute then P0 then P1 & then P3

P2 → P0 → P1 → P3

* Various methords to handle deadlock

1. Deadlock ignorance (ostrich methord)
2. Deadlock prevention
3. Deadlock avoidance (Banker's Algo)
4. Deadlock det. & Recovery -

1. Deadlock ignorance:- This is done by most of the OS Users when our PC hold we simply restart the System.

We dont want to affect speed/ performance of an OS.

2. Deadlock prevention :- ☆

Conditions of Deadlock prevention

NO

(a) Mutual Exclusion
(b) No preemption
(c) No Hold & Wait
(d) No Circular Wait

} We try to make any one of the four or all four conditions to false.

↓ Just reverse Necessary Condn defination

Process can request for Resources in only ↑ order.

3. Deadlock Avoidance (Banker's Algo.)

☆

To find Safe Sequence of a process we use banker's algorithm.

4. Deadlock detection & Recovery.

First we try to detect deadlock from the use of RAG & any other method then we try to recover that process or System.

for recovery kill the Processes or Process.

PAGE NO.

Also used for deadlock detection

Subject :

Date: __/__/____

MON TUE WED THR FRI SAT SUN

Banker's algorithm

Deadlock avoidance

Total A=10, B=5, C=7

We need to tell system abt about every Process.

Process	Allocation A B C	Max Need A B C	Avail A B C (curr.)	Rem. Need A B C
P_1	0 1 0	7 5 3	3 3 2	7 4 3
P_2	2 0 0	3 2 2		1 2 2
P_3	3 0 2	9 0 2		6 0 0
P_4	2 1 1	4 2 2		2 1 1
P_5	0 0 2	5 3 3		5 3 1
$P_2 \to P_4 \to P_5 \to P_1 \to P_3$	7 2 5			

at i=0 P_1 need 7 4 3 but av. 3 3 2

So we Cant execute

at i=1 P_2 need 1 2 2 but av. 3 3 2

So we Can execute

So new curr. avai = 3 3 2 + 2 0 0 = 5 3 2

(Av. All.)

at i=2 P_3 need 6 0 0 but av. 5 3 2

So we Cant execute

at i=3 P_4 need 2 1 1 but av. 5 3 2

So we Can execute

new av. = 5 3 2 + 2 1 1

= 7 4 3

at i=4 P_3 need 5 3 1 but av. 7 4 3
So we can't execute P_3
new av. = 7 4 3 + 0 0 2 = 7 4 5

at i=5 P_1 need 7 4 3 but av. 7 4 5
So we can execute
new av. = 7 4 5 + 0 1 0 = 7 5 5

at i=6 P_3 need 6 0 0 but av. 7 5 5
So we can execute
new av: 7 5 5 + 3 0 2
= 10 5 7

So the is no deadlock
Safe Sequence

$P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$

Subject :

Date: __/__/____
MON TUE WED THR FRI SAT SUN

Producer

```
while (true) {

        /* Produce an item is next produced */
        while (counter == Buffer_Size);

        buffer[in] = next_produced;
        in = (in+1) % Buffer_Size;
        counter++;
    }
```

Consumer

```
while (true) {
        while (counter == 0);

        next_consumed = buffer[out];
        out = (out+1) % Buffer_Size;
        counter--;
    }
```

Peterson Soln.

1. Mutual Exclusion is Satisfied
   Pi enter only if flag[j] = false or turn = i
2. Progess requirement is Satisfied
3. Bounded Waiting requirement is Satisfied

PAGE NO

Mutual Exclusion is satisfied by while loop & checks that only when one process is executing in C.S. then other loop wont allow other process to enter in CS.

Both P_1 & P_2 are not executing in remainder section they both take part in decision making of which process can execute in C.S & this decision is made by that process who last change the value of turn

In this P_1 & P_2 when P_1 got chance to execute in its critical S. P_2 continously loop inside while condition as soon as P_1 complete execution C.S P_2 get chance to execute in its critical section so Bounded waiting satisfy

Semaphore Implementation with no busy Waiting

With each semaphore there is a Waiting queue

Each entry in Waiting queue has 2 data type

1) Value (of type integer)
2) Pointer to next record in the list.

Two operation

1) block :- place the process invoking the op. in appr Waiting queue
2) Wakeup :- Remove one of process in the Waiting queue & place it in the ready queue.

Subject :

Date :__/__/____
MON TUE WED THR FRI SAT SUN

```
typedef struct{
    int value;
    Struct process *list;
} Semaphore;

wait (Semaphore *s){
    s->value--;
    if (s->value < 0){
        add this process to s->list;
        block();
    }
}

Signal (Semaphore *s){

    s->value++;
    if (s->value <= 0){
        remove a process P from s->list;
        wakeup(P);
    }
}
```

Deadlock Prevention

a) Mutual Exclusion :- Not req. for sharable resources must hold for non sharable resources.

b) Hold & Wait :- Must guarantee that whenever a process req. a resource it does not hold & other resource.

Require process to req. and be allocated all its resources before it begins its execution or allow process to req. resources only when process has none allocated to it.

No preemption :- If a process is holding some resource another resource that cannot be immediately allocated to it then all resources currently being held are released.

Preempted resources are added to list of resources for which the process is waiting.

Circular wait :- Impose a total ordering of all resource types & req. that each process req. resources in an ↑ order of enumeration.

Safe state :- if there exist a sequence of all the process in the system such that for each P_i, the resources that P_i can still req can be satisfied by currently avl. resources + res. held by P_j with $j<i$

If Safe state ⇒ No deadlock
" Unsafe state ⇒ Possibility of deadlock
Avoidance ⇒ ensure that a sys. never enters a deadlock.

Date: __/__/____

MON TUE WED THR FRI SAT SUN

Subject: _____

Detection - Algo Usage

How often a deadlock is likely to occur.

How many processes will need to be rolled back.

If detection algo invoked arbitrary there may be many cycles in the resource graph & so we would not be able to tell which of the many deadlock process "caused" the deadlock.

Recovery Process termination.

1. Abort all deadlock processes
2. Abort one process at a time until deadlock is avoided.

In which order we should abort:

1. Priority of the process
2. How long process has computed, & how much longer to completion.
3. Resources the process has used.
4. Resources process need to complete
5. How many proc. need to be terminated
6. Is process interactive or batch.