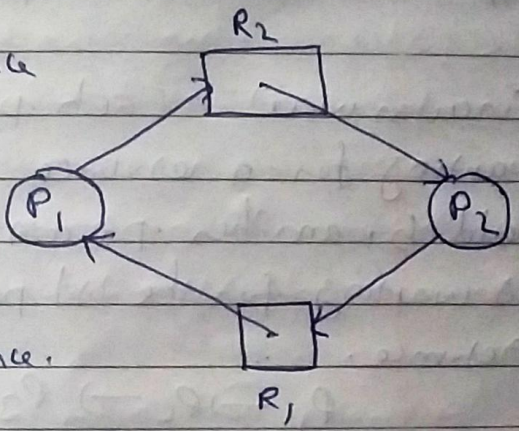
Page No.:
Date: / /

## Deadlock

In a multiprogramming system, a number of process compete for limited no. of resources and if a resource is not available at that instance then process enters into waiting state.

If a process unable to change its waiting state indefinitely bcoz the resources required by it are held by another waiting process then the system is said to be in deadlock.

$P_1$ has acquired $R_1$ resource and requesting for $R_2$ resource.
$P_2$ has acquired $R_2$ and requesting for $R_1$ resource.




Resource allocation graph

### System model

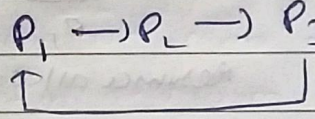
→ every process will request for resource
→ If resource is granted then process will use the resource.
→ Process must release the resource after use.

Necessary Cond's of deadlock
→ mutual exclusion :- at least one resource type in the system which can be used in non-sharable mode; i.e. mutual exclusion. e.g. Printer.

Hold and wait :- A process is currently holding at least one resource and requesting additional resources which are being held by other processes.

No preemption :- A resource cannot be preempted from a process by any other process. Resource can be released only voluntarily by the process holding it.

Circular wait :- Each process must be waiting for a resource which is being held by another process which in turn is waiting for the 1st process to release the resource.




here, $P_1$ is waiting for $P_2$ and $P_2$ is waiting for $P_3$. upto here, all 3 cond$^n$ of deadlock is satisfied. but $P_3$ is not waiting for resource then in this case, after sometime $P_3$ will be finished and it will release its resources. So, deadlock will not occur. So, to occur the deadlock, circular wait must be there.

Page No.:
Date: / /

# Deadlock handling methods

(1) Prevention:- means design such a system which violate at least one of four necessary cond's of deadlock and ensure independence from deadlock.

(2) Avoidance:- System maintains a set of data using which it takes a decision whether to entertain a new request or not, to be in safe state.

(3) Detection and Recovery:- Here we wait until deadlock occurs and once we detect it, we recover from it.

Deadlock Prevention:
mutual exclusion cannot be violated bcoz, whether a resource is shareable or not, depends on its hardware property. So e.g. Printer. So, while using the printer by a process, the OS must have to ensure mutual exclusion, i.e. no other process can access the printer at that time.

Violation of hold and wait
Conservative approach:- Process is allowed to start execution iff it has acquired all the resources (less efficient, not implementable, easy, deadlock independence)

Do not hold:- Process will acquire only desired resources but before making any fresh request it must release all the resources that it currently

hold. (efficient, implementable)

hold time-out :- A Process can hold the resources upto a certain time limit and then it must release the resources.

These sol's ~~are useful~~ will help in preventing the system from deadlock but after releasing the resources, some other process can use the released resources and can complete its execution.

Violation of no preemption

forceful preemption :- we allow a process to forcefully preempt the resource holded by other processes

→ this method may be used by high priority process or system process

→ the process which are in waiting state must be selected as a victim instead of process in the running state.

Violation of circular wait

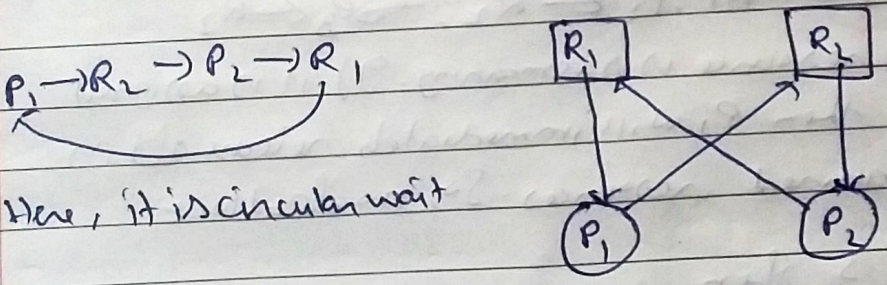
→ Circular wait can be eliminated by first ~~giving a natural~~ numbering the resources e.g. $R_1, R_2, \dots R_n$

→ allow every process to either only in the increasing or decreasing order of the resource number. e.g. if a process wants $R_1, R_3, R_7$ resources then it will request first for $R_1$ then $R_3$ then $R_7$. or if $R_7, R_3, R_1$.

Let Process $P_1$ and $P_2$ wants $R_1$ and $R_2$ resources.
So, both $P_1$ and $P_2$ will try to get $R_1$ first.
Let $P_1$ got $R_1$, then as per the rule,
$P_2$ will not acquire $R_1$ and it will wait to release $R_1$ by $P_1$. So, circular wait will be avoided.

→ If during the execution, if a process wants a resource with less ~~order~~ number then it must release all resources with high number than the required resource.
e.g. If $P_1$ has acquired $R_1$, $R_{30}$ $R_{40}$ then suddenly it wants $R_{25}$ then it will have to release $R_{30}$ and $R_{40}$ resource.

$P_1 \rightarrow R_2 \rightarrow P_2 \rightarrow R_1$




Here, it is circular wait

⇒ Consider a system with 3 processes that share 4 instances of the same resource type. Each process can request a max. of K instances. Resource instances can be requested and released only one at a time. The largest value of K to avoid the deadlock.

for K=1, no deadlock

for K=2, in the worst case, we should not give 2 resources to a process. we should give one less resource to a process to analyze the worst case.
So, $P_1 \leftarrow R_1$, $P_2 \leftarrow R_2$, $P_3 \leftarrow R_3$
now, 1 resource is remaining. If it is assigned to $P_1$ then $P_1$ will immediately release its 2 acquired resources. So, no deadlock

if K=3, then deadlock must occur.
So, K=2 is largest.

⇒ Suppose n processes, $P_1$, ... $P_n$ share m identical resource units, which can be reserved and released one at a time. The max. resource requirement of process $P_i$ is $S_i$, where $S_i > 0$
which one of the following is a sufficient cond$^n$ for ensuring that deadlock does not occur?

$P_1$ need $S_1$ resources

$P_2$ ~ $S_2$ ~

$P_n$ ~ $S_n$ ~

Page No.:____
Date:__/__/__

to analyse the worst case, let $P_1$ holds $S_1 - 1$, $P_2$ holds $S_2 - 1$ resources, $P_3$ holds $S_3 - 1$ resources $P_n$ holds $S_n - 1$ resources. So, prevents the deadlock

$$(S_1 - 1) + (S_2 - 1) + (S_3 - 1) + \cdots + (S_n - 1) < m$$

$$\sum_{i=1}^{n} S_i - n < m$$

$$\therefore \sum_{i=1}^{n} S_i < m + n$$

Mukesh

Page No.:
Date: / /

## Deadlock Avoidance

Problem with prevention:- Different deadlock prevention approach put different type of restric$^{n}$ on cond$^{n}$s on the processes and resources bcoz of which system becomes slow and resource utilisa$^{n}$ and reduced system throughput.

### avoidance

→ to avoid deadlocks we require additional information about how resources are to be requested, which resources a proces will request and use during its lifetime, i.e. max. no. of resources of each type that it may need.

→ with this additional knowledge, the OS can decide for each request whether process should wait or not.

| | max need | | | Allocation | | |
|---|---|---|---|---|---|---|
| | E | F | G | E | F | G |
| $P_0$ | 4 | 3 | 1 | 1 | 0 | 1 |
| $P_1$ | 2 | 1 | 4 | 1 | 1 | 2 |
| $P_2$ | 1 | 3 | 3 | 1 | 0 | 3 |
| $P_3$ | 5 | 4 | 1 | 2 | 0 | 0 |

| | current need | | | System max | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|
| | E | F | G | E | F | G | E | F | G |
| $P_0$ | 3 | 3 | 0 | 8 | 4 | 6 | 3 | 3 | 0 |
| $P_1$ | 1 | 0 | 2 | | | | | | |
| $P_2$ | 0 | 3 | 0 | | | | | | |
| $P_3$ | 3 | 4 | 1 | | | | | | |

Page No.:
Date: / /

→ A deadlock avoidance algo. dynamically examines the resource allocⁿ state

→ the resource allocⁿ state is defined by the no. of available and allocated resources and the maxi. demands of the process before allowing that request first.

Currently, current need of $P_0$ and $P_2$ can be satisfied.

let current need of $P_0$ is satisfied then $P_0$ will release all its resources.

so, now available will be 3 3 0
1 0 1
4 3 1

now, let current need of $P_2$ is satisfied then $P_2$ will released all its resources

so, now available will be 4 3 1
1 0 3
5 3 4

now, let current need of $P_1$ is satisfied

so, available will be 5 3 4
1 1 2
6 4 6

now, $P_3$ is satisfied

so, available 6 4 6
2 0 0
8 4 6 - which is maxi. system resources.

Page No.:
Date: / /

So, $P_0 \rightarrow P_2 \rightarrow P_1 \rightarrow P_3$ can be said as safe seaence, i.e. if processes are given the resources it in that seaence then deadlock can be avoided.

A Single processor system has 3 resources types X, Y, Z, which shared by 3 processes. There are 5 units of each resource type. Consider the following Scenario, where the column 'Allocation' denotes the no. of units of each resource type allocated to each process and the column 'Seaeest' denotes the no. of units of each resource type seaested by a process in order to complete execn. Find the safe seaena.

max. System resources - X Y Z
5 5 5

| | Allocn X | Y | Z | | Reaest X | Y | Z |
|---|---|---|---|---|---|---|---|
| $P_0$ | 1 | 2 | 1 | $P_0$ | 1 | 0 | 3 |
| $P_1$ | 2 | 0 | 1 | $P_1$ | 0 | 1 | 2 |
| $P_2$ | 2 | 2 | 1 | $P_2$ | 1 | 2 | 0 |

Here, seaest means current need

Avail

0 1 2

Page No.:
Date: / /

with this availability of ~~request~~ resources, current need of $P_1$ can be satisfied.

Let $P_1$ is granted. So, $P_1$ will release its resources. So, Available

| X | Y | Z |
|---|---|---|
| 0 | 1 | 2 |
| 2 | 0 | 1 |
| 2 | 1 | 3 |

now, $P_0$ can be granted.

So, available

| | | |
|---|---|---|
| 2 | 1 | 3 |
| 1 | 2 | 1 |
| 3 | 3 | 4 |

now, $P_2$ can be granted. So, ~~available should be~~ after resources released by $P_2$, available resources should be equal to max. system resources.

Available

| | | |
|---|---|---|
| 3 | 3 | 4 |
| 2 | 2 | 1 |
| 5 | 5 | 5 |

So, safe sequence is $P_1 \rightarrow P_0 \rightarrow P_2$

=)

1 resource with 12 instances

| | max need | Allocation |
|---|---|---|
| $P_1$ | 10 | 5 |
| $P_2$ | 4 | 2 |
| $P_3$ | 9 | 3 |

| | Current need |
|---|---|
| $P_1$ | 5 |
| $P_2$ | 2 |
| $P_3$ | 6 |

Available:
12 - 10 = 2

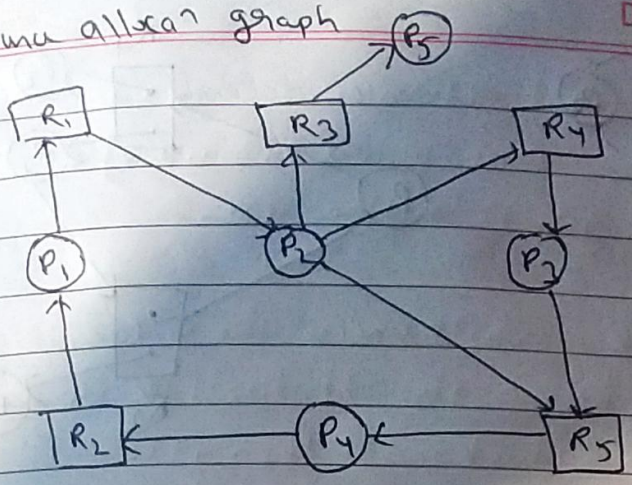
Page No.:
Date: / /

Let $P_2$ is granted. So, available 2+2 = 4

but now no process can be granted. So system is in unsafe state. However, it does not mean that system is in deadlock.

Page No.
Date: / /

Resource allocation graph




here, $P_1$ is waiting for $R_1$ which is allocated to $P_2$. So, $P_1 \rightarrow P_2$

$P_2$ is waiting for $R_4$ which is allocated to $P_3$.

So, $P_2 \rightarrow P_3$

$P_3$ is waiting for $R_5$ which is allocated to $P_4$

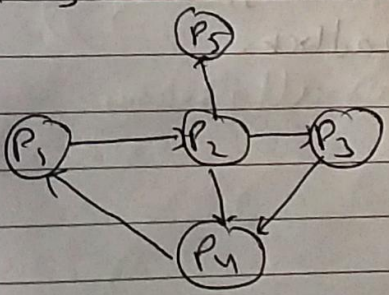
So, $P_3 \rightarrow P_4$

$P_2$ is waiting for $R_5$ which is allocated to $P_4$

So, $P_2 \rightarrow P_4$

$P_2$ is waiting for $R_3$ which is allocated to $P_5$.

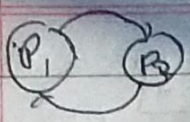
So, $P_2 \rightarrow P_5$

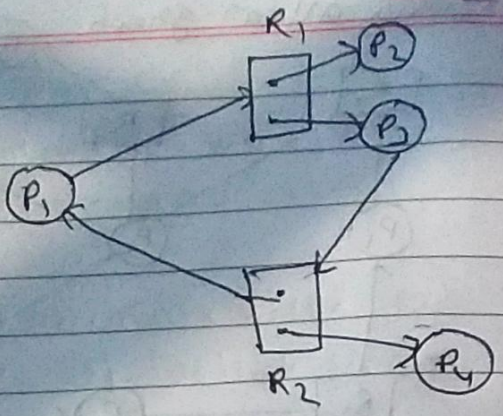



Here, a cycle is formed. ~~So, deadlock~~

If only one instance of each resource is available then detection of cycle is necessary and sufficient condn for detecn of deadlock.

Page No.:
Date: / /

Initially, 


Initially, $P_1$ and $P_3$ are forming a cycle but once, $P_2$ and $P_4$ are served, they will release the acquired resource, i.e. 1 instance of $R_1$ and 1 instance of $R_2$ which can be served to $P_2$ and $P_4$. So, even though cycle exist, it may not lead to deadlock.

So, in case of multiple instances of resources, detecting a cycle is necessary but not sufficient cond$^n$ of detection of deadlock.

Page No.:
Date: / /

## Deadlock Detection and Recovery

Here we do not check safety. In this case, when a process request for some resource then its request is granted immediately.

To check the possibility of deadlock, two approaches are followed

(a) Active approach :- Here, detection algo. are invoked at definite interval of time.

(b) lazy approach :- Detection algo. are invoked when unusual performance is observed. e.g. CPU utilization goes below 40%.

If resources have single instances then deadlock can be detected by detecting a cycle in resource allocation graph.

In case of multiple instances of resources, if cycle detected, safety algo. need to be run.

After detecting the deadlock, recovery from the deadlock is needed

Page No.:
Date: / /

Recovery in terms of resources:

(a) Preempt :- Preempt a resource of one user and give it to other user.

(b) Rollback :- while execution, the OS can maintain checkpoints. If deadlock is detected in a checkpoint then it can be rollbacked to the previous checkpoint where there was no deadlock. In this process, some processes will have to release their acquired resources.

Recovery in terms of processes

(a) Kill one process at a time. However, here choosing which process to kill is a difficult task.

(b) Kill all process.