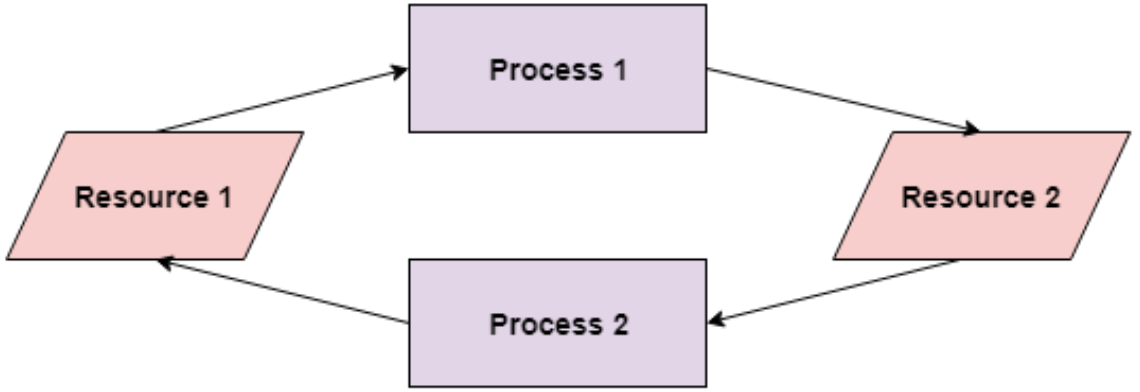
BCA Fourth Semester

Operating Systems

Unit -4 Process Deadlocks [4 Hrs]

Deadlock Introduction

- **Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.**
- **A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.**

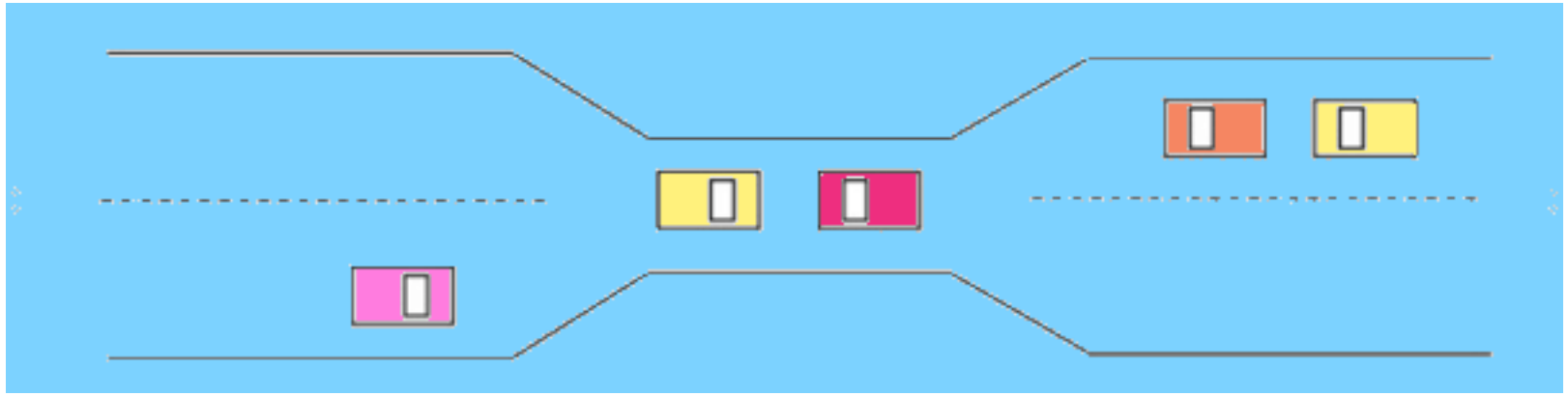


Deadlock in Operating System

In the above diagram, the process 1 has resource 1 and needs to acquire resource 2. Similarly process 2 has resource 2 and needs to acquire resource 1. Process 1 and process 2 are in deadlock as each of them needs the other's resource to complete their execution but neither of them is willing to relinquish their resources.

Example of Deadlock

- **A real-world example would be traffic, which is going only in one direction**.
- Here, a **bridge** is considered a **resource**.
- So, when Deadlock happens, it can be easily resolved if one car backs up (Preempt resources and rollback).
- Several cars may have to be backed up if a deadlock situation occurs.
- So starvation is possible.



Deadlock vs Starvation

Deadlock	Starvation
The deadlock situation occurs when one of the processes got blocked.	Starvation is a situation where all the low priority processes got blocked, and the high priority processes execute.
Deadlock is an infinite process.	Starvation is a long waiting but not an infinite process.
Every Deadlock always has starvation.	Every starvation does n't necessarily have a deadlock.
Deadlock happens then Mutual exclusion, hold and wait. Here, preemption and circular wait do not occur simultaneously.	It happens due to uncontrolled priority and resource management.

Resources

- **Resources are the passive entities needed by processes to do their work**.
- A resource can be a hardware device (eg. a disk space) or a piece of information (a locked record in the database).
- Example of resources include CPU time, disk space, memory etc.
- There are two types of resources:

Preemptable –

- A Preemptable resources is one that can be taken away from the process owing it with no ill effect. Memory is an example of preemptable resources.

Non-preemptable –

- A non-preemptable resources in contrast is one that cannot be taken away from its current owner without causing the computation to fail.
- Examples are CD-recorder and Printers. If a process has begun to burn a CD-ROM, suddenly taking the CD recorder away from it and giving it to another process will result in a garbled CD. CD recorders are not preemptable at any arbitrary moment.
- In general Deadlocks involves non preemptable resources.

Conditions for Deadlock

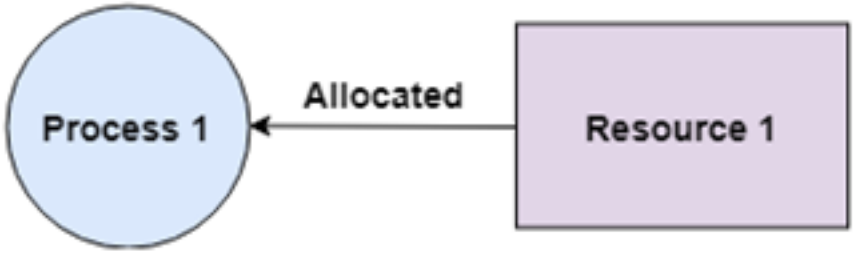
Coffman et al. (1971) showed that four conditions must hold for there to be a deadlock:

A deadlock occurs if the four Coffman conditions hold true. But these conditions are not mutually exclusive.

The Coffman conditions are given as follows −

1) Mutual Exclusion

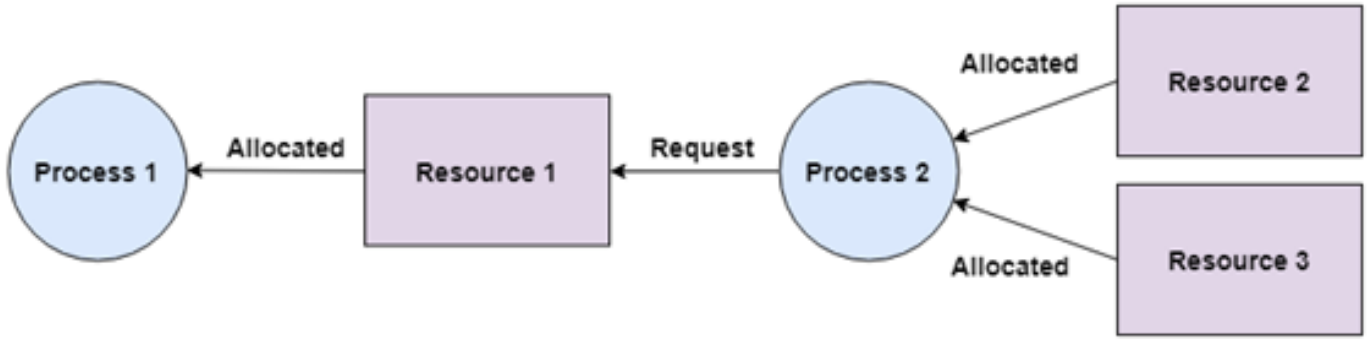
- There should be a resource that can only be held by one process at a time.
- In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



Conditions for Deadlock (cont.)

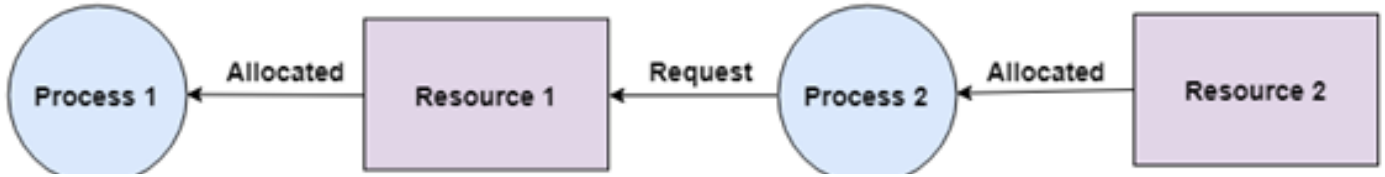
2) Hold and Wait

- **A process can hold multiple resources and still request more resources from other processes which are holding them.**
- In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



3) No Preemption

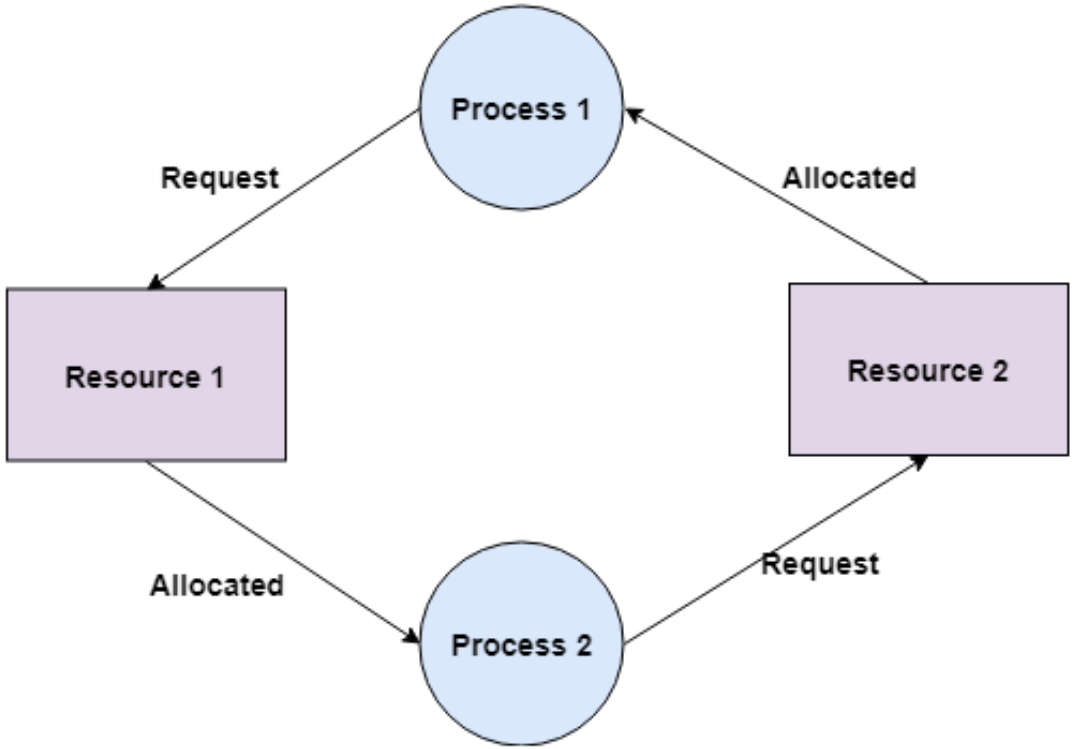
- **A resource cannot be preempted from a process by force.**
- A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



Conditions for Deadlock (cont.)

4) Circular Wait

- **A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain**.
- For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



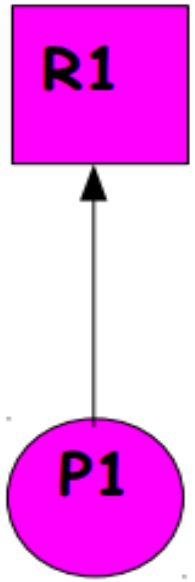
Deadlock Modeling

- Deadlocks can be described more precisely in terms of **Resource allocation graph**.
- **Its a set of vertices V and a set of edges E. V is partitioned into two types:**

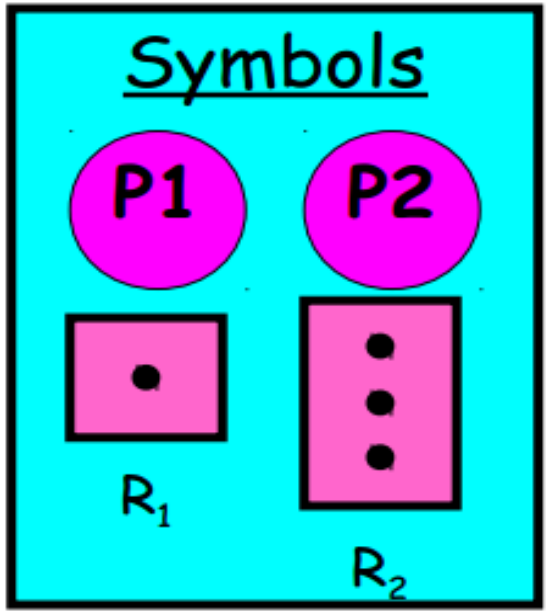
P = {P1, P 2, ..., P n}, the set consisting of all the processes in the system.
R = {R1, R2 , ..., Rm}, the set consisting of all resource types in the system.
request edge – directed edge Pi ⟹ Rj

assignment edge – directed edge Rj ⟹ Pi

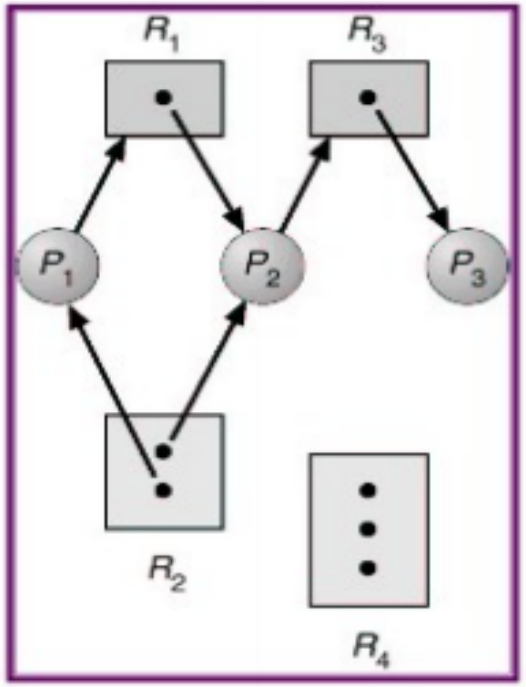
a). P1 is holding R1



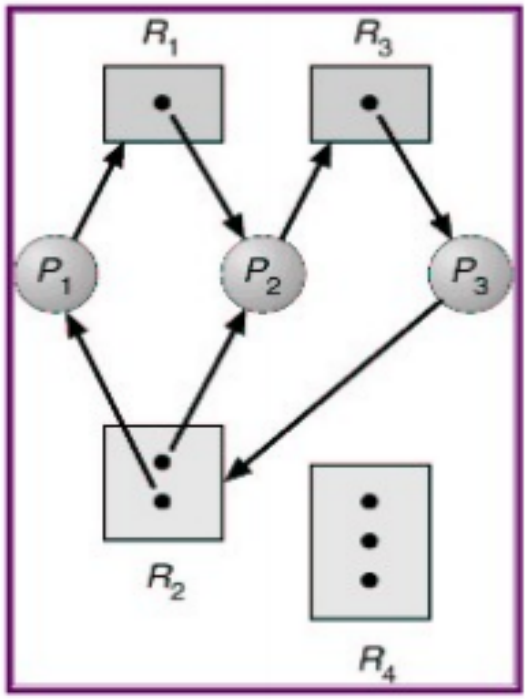
b). P1 requests R1



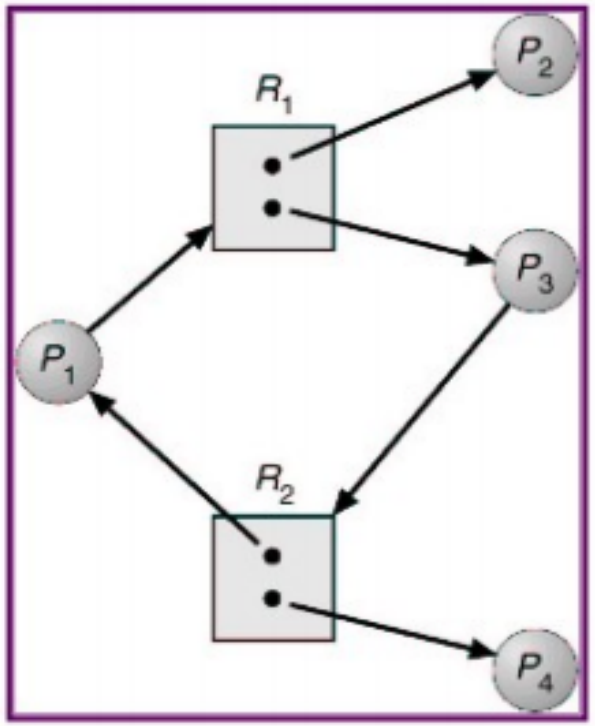
Deadlock Modeling (cont.)



a).eg. of a Resource allocation graph



b).Resource allocation graph with Deadlock



c).Resource Allocation graph with a Cycle but no Deadlock

Basic Facts:

If graph contains no cycles ⇒ no deadlock.

If graph contains a cycle ⇒

- If only one instance per resource type, then deadlock.
- If several instances per resource type, possibility of Deadlock

Methods for Handling Deadlock:

Allow system to enter deadlock and then recover

- Requires deadlock detection algorithm (**Banker's Algorithm**)
- Some technique for forcibly preempting resources and/or terminating tasks

Ensure that system will never enter a deadlock

- Need to monitor all lock acquisitions
- Selectively deny those that might lead to deadlock

Ignore the problem and pretend that deadlocks never occur in the system (Ostrich Algorithm)

- Used by most operating systems, including UNIX

Deadlock Prevention:

Fact: *If any one of the four necessary conditions is denied, a deadlock can not occur.*

Denying Mutual Exclusion:

- Sharable resources do not require mutually exclusive access such as read only shared file.
- Problem: Some resources are strictly nonsharable, mutually exclusive control required.

We can not prevent deadlock by denying the mutual exclusion

Deadlock Prevention (cont.):

Denying Hold and Wait:

Resources grant on *all or none* basis;

If all resources needed for processing are available then granted and allowed to process.

If complete set of resources is not available, the process must wait set available.

While waiting, the process should not hold any resources.

Problem:

Low resource utilization.

Starvation is possible.

Denying No-preemption:

When a process holding resources is denied a request for additional resources, that process must release its held resources and if necessary, request them again together with additional resources.

Problem: When process releases resources the process may loose all its works to that point .

Indefinite postponement or starvation.

Deadlock Prevention (cont.):

Denying Circular Wait:

All resources are uniquely numbered, and processes must request resources in linear ascending order.

The only ascending order prevents the circular.

Problem:

Difficult to maintain the resource order; dynamic update in addition of new resources. Indefinite postponement or starvation.

Deadlock Avoidance:

- Given some additional information on how each process will request resources, it is possible to construct an algorithm that will avoid deadlock states.
- The algorithm will dynamically examine the resource allocation operations to ensure that there won't be a circular wait on resources.

Two deadlock avoidance algorithms:

- Resource-allocation graph algorithm
- Banker's algorithm

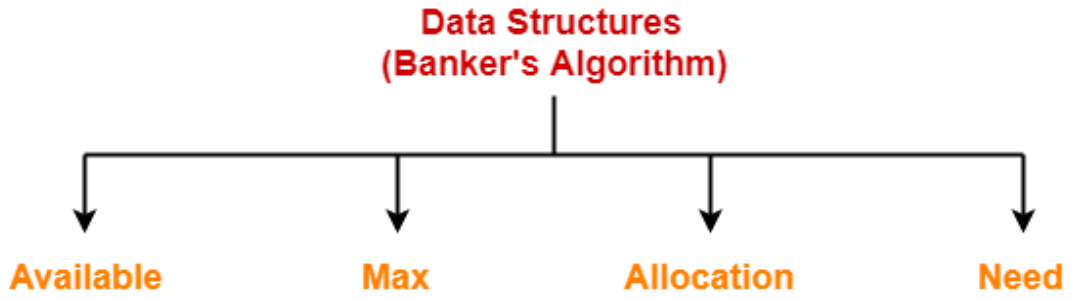
Banker’s Algorithm

Banker's algorithm is a **deadlock avoidance algorithm**. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not.

Consider there are `n` account holders in a bank and the sum of the money in all of their accounts is `S`. Everytime a loan has to be granted by the bank, it subtracts the **loan amount** from the **total money** the bank has. Then it checks if that difference is greater than `S`. It is done because, only then, the bank would have enough money even if all the `n` account holders draw all their money at once.

Banker's algorithm works in a similar way in computers.

> Whenever a new process is created, it must specify the maximum instances of each resource type that it needs, exactly.



Banker’s algorithm consists of Safety algorithm and Resource request algorithm

Banker's Algorithm (cont.)

Let us assume that there are `n` processes and `m` resource types. Some data structures that are used to implement the banker's algorithm are:

1. `Available`

It is an **array** of length `m`. It represents the number of available resources of each type. If `Available[j] = k`, then there are `k` instances available, of resource type `R(j)`.

2. `Max`

It is an `n x m` matrix which represents the maximum number of instances of each resource that a process can request. If `Max[i][j] = k`, then the process `P(i)` can request atmost `k` instances of resource type `R(j)`.

3. `Allocation`

It is an `n x m` matrix which represents the number of resources of each type currently allocated to each process. If `Allocation[i][j] = k`, then process `P(i)` is currently allocated `k` instances of resource type `R(j)`.

4. `Need`

```
Need[i][j] = Max[i][j] - Allocation [i][j]
```

It is an `n x m` matrix which indicates the remaining resource needs of each process. If `Need[i][j] = k`, then process `P(i)` may need `k` more instances of resource type `R(j)` to complete its task.

Resource Request Algorithm

Let $Request_i$ be the request array for process P_i. $Request_i$ [j] = k means process P_i wants k instances of resource type R_j. When a request for resources is made by process P_i, the following actions are taken:

1) If $Request_i <= Need_i$

Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2) If $Request_i <= Available$

Goto step (3); otherwise, P_i must wait, since the resources are not available.

3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:

Available = Available − Requesti

$Allocation_i = Allocation_i + Request_i$

$Need_i = Need_i - Request_i$

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1) Let Work and Finish be vectors of length 'm' and 'n' respectively.

Initialize: Work = Available

Finish[i] = false; for i=1, 2, 3, 4....n

2) Find an i such that both

a) Finish[i] = false

b) $Need_i$ <= Work

if no such i exists goto step (4)

3) Work = Work + Allocation[i]

Finish[i] = true

goto step (2)

4) if Finish [i] = true for all i

then the system is in a safe state

Deadlock Avoidance
Problem with Banker's Algorithm

Algorithms requires fixed number of resources, some processes dynamically changes the number of resources.

Algorithms requires the number of resources in advanced, it is very difficult to predict the resources in advanced.

Algorithms predict all process returns within finite time, but the system does not guarantee it.

Deadlock Detection and Recovery

Instead of trying to prevent or avoid deadlock, system allow to deadlock to happen, and recover from deadlock when it does occur.

Hence, the mechanism for deadlock detection and recovery from deadlock required.

Deadlock Detection

Consider the following scenario:

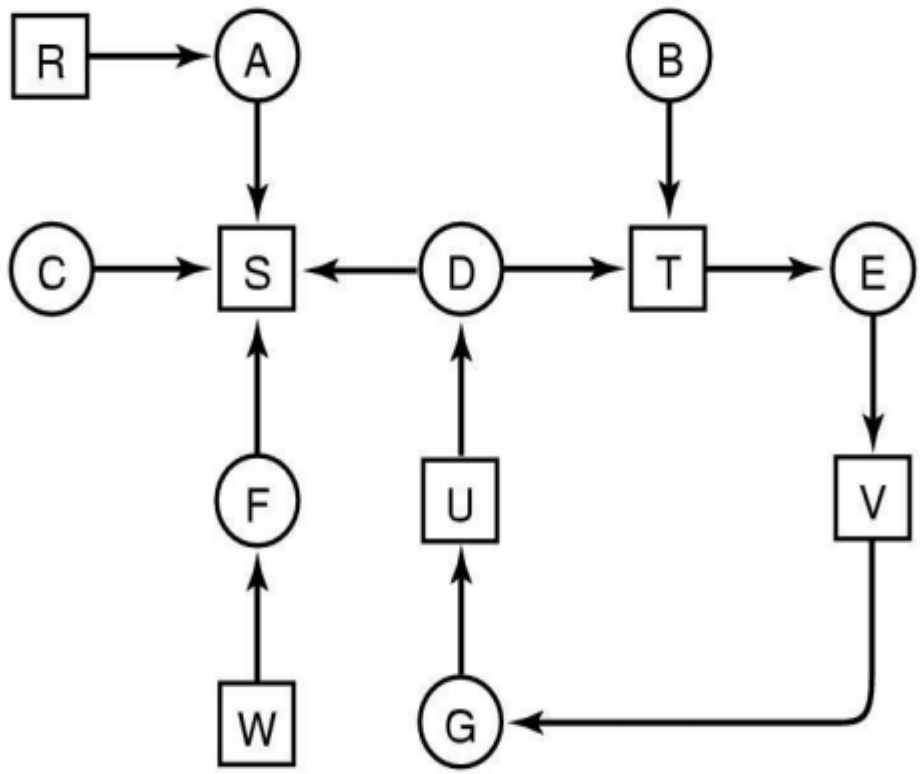
a system with 7 processes (A – G) , and 6 resources (R – W) are in following state.

1. Process A holds R and wants S.
2. Process B holds nothing but wants T.
3. Process C holds nothing but wants S.
4. Process D holds U and wants S and T.
5. Process E holds T and wants V.

6. Process F holds W and wants S. 7 . Process G holds V and wants U.

Is there any deadlock situation?

Deadlock Detection



How to detect the cycle in directed graph?

Deadlock Detection Algorithm

```
List L;
Boolean cycle = False;
for each node
    L = Φ   /* initially empty list*/ add node to L; for each
    node in L for each arc if (arc[i] = true)
                add corresponding node to L;
                if (it has already in list)
                        cycle = true;
                print all nodes between these nodes;
                exit;
```

```
else if (no such arc) remove
        node from L;
        backtrack;
```

Recovery from Deadlock

What do next if the deadlock detection algorithm succeed? – recover from deadlock.

By Resource Preemption

Preempt some resources temporarily from a processes and give these resources to other processes until the deadlock cycle is broken.

Problem:

Depends on the resources.

Need extra manipulation to suspend the process.

Difficult and sometime impossible.

Recovery from Deadlock

By Process Termination

Eliminating deadlock by killing one or more process in cycle or process not in cycle.

Problem:

If the process was in the midst of updating file, terminating it will leave file in incorrect state.

Key idea: we should terminate those processes the termination of which incur the minimum cost.

Ostrich Algorithm

Fact: there is no good way of dealing with deadlock.

Ignore the problem altogether.

For most operating systems, deadlock is a rare occurrence;

So the problem of deadlock is ignored, like an ostrich sticking its head in the sand and hoping the problem will go away.