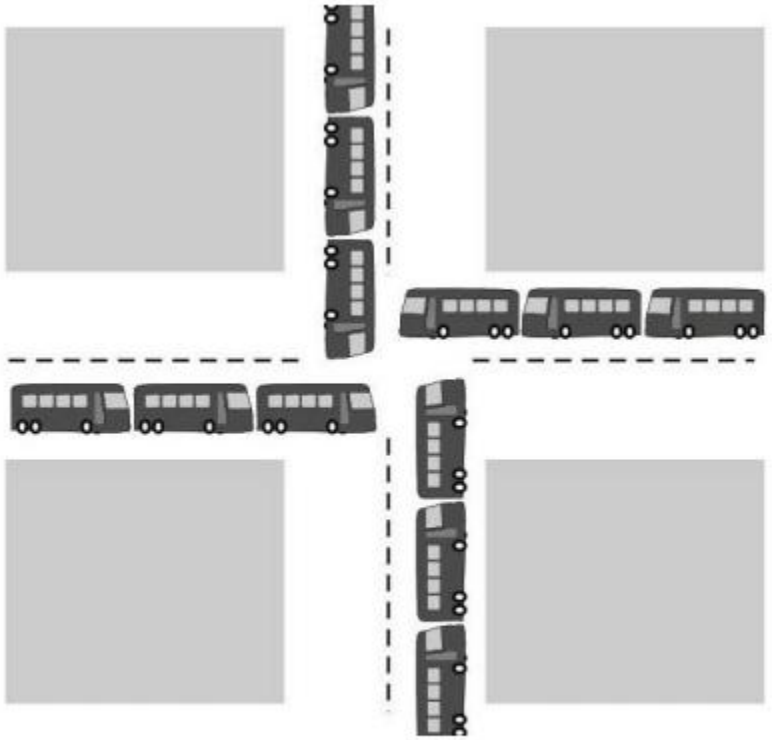
BCA
Fourth Semester
" Operating System"

Deadlock

A process in a multiprogramming system is said to be in dead lock if it is waiting for a particular event that will never occur.

Deadlock Example

- All automobiles trying to cross.
- Traffic Completely stopped.
- Not possible without backing some.



A Traffic Deadlock

Resource Deadlock

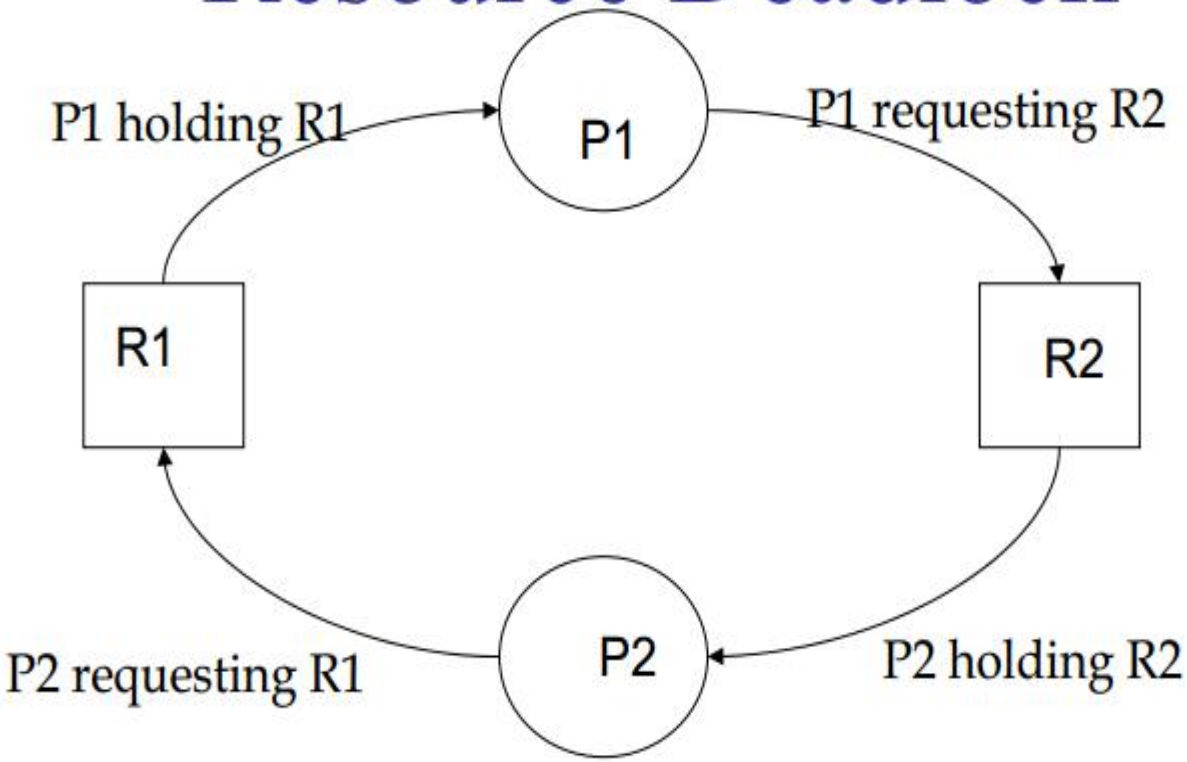
A process request a resource before using it, and release after using it.

1. Request the resource.
2. Use the resource.
3. Release the resource.

If the resource is not available when it is requested, the requesting process is force to wait.

Most deadlocks in OS developed because of the normal contention for dedicated resources.

Resource Deadlock



- Process P1 holds resource R1 and needs resource R2 to continue; Process P2 holds resource R2 and needs resource R1 to continue – deadlock.

Key: Circular wait - deadlock

Conditions for Deadlock

1. *Mutual Exclusion:* Process claims exclusive control of resources they require.
2. *Hold and Wait:* Processes hold resources already allocated to them while waiting for additional resources.
3. *No preemption:* Resources previously granted can not be forcibly taken away from the process.
4. *Circular wait:* Each process holds one or more resources that are requested by next process in the chain.

A deadlock situation can arise if all four conditions hold simultaneously in the system.

Handling Deadlocks

- Deadlock handling strategies:
- We can use a protocol to *prevent* or *avoid* deadlocks, ensuring that the system never enter a deadlock state.
- We can allow the system to enter a deadlock state, *detect* it, and *recover*.

- We can ignore the problem all to gather, and pretended that deadlock never occur in the system.

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

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.

Deadlock prevention

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

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

Avoiding deadlock by careful resource allocation.

Decide whether granting a resource is safe or not, and only make the allocation when it is safe.

Need extra information in advanced, maximum number of resources of each type that a process may need.

The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.

Bankers Algorithm

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Following **Data structures** are used to implement the Banker's Algorithm:

Let '**n**' be the number of processes in the system and '**m**' be the number of resources types.

Available :

- It is a 1-d array of size '**m**' indicating the number of available resources of each type.
- Available[j] = k means there are '**k**' instances of resource type $\mathbf{R_j}$

Max :

- It is a 2-d array of size '**n*m**' that defines the maximum demand of each process in a system.
- Max[i, j] = k means process $\mathbf{P_i}$ may request at most '**k**' instances of resource type $\mathbf{R_j}$.

Allocation :

- It is a 2-d array of size '**n*m**' that defines the number of resources of each type currently allocated to each process.
- Allocation[i, j] = k means process $\mathbf{P_i}$ is currently allocated '**k**' instances of resource type $\mathbf{R_j}$

Need :

- It is a 2-d array of size '**n*m**' that indicates the remaining resource need of each process.
- Need [i, j] = k means process $\mathbf{P_i}$ currently allocated '**k**' instances of resource type $\mathbf{R_j}$
- Need [i, j] = Max [i, j] – Allocation [i, j]

$Allocation_i$ specifies the resources currently allocated to process P_i and $Need_i$ specifies the additional resources that process P_i may still request to complete its task.

Banker's algorithm consist of Safety algorithm and Resource request algorithm

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, ..., n

2) Find an i such that both

   a) Finish [i] = false

   b) $Need_i \leq work$

      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 safe state.

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 $Request_i \leq Need_i$

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

2) If $Request_i \leq Available$

   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 P_i by modifying the state as follows:

   $Available = Available - Request_i$

   $Allocation_i = Allocation_i + Request_i$

   $Need_i = Need_i - Request_i$

Example

- See the files provided in Google classroom

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.

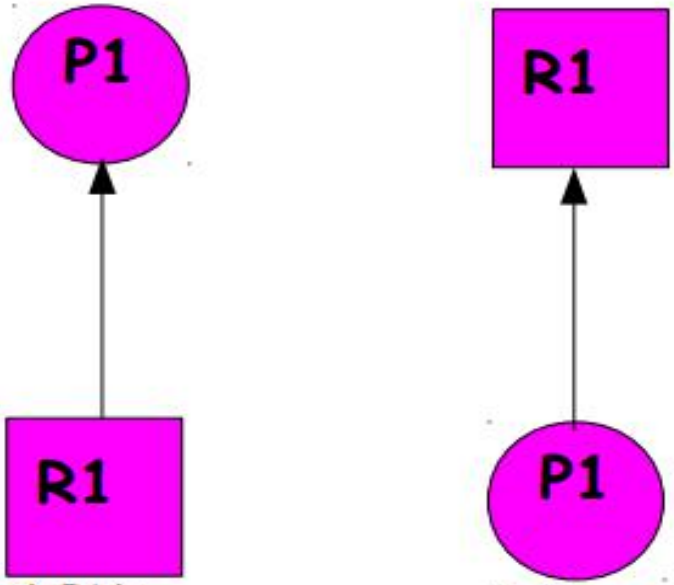
Deadlock Modeling:

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.

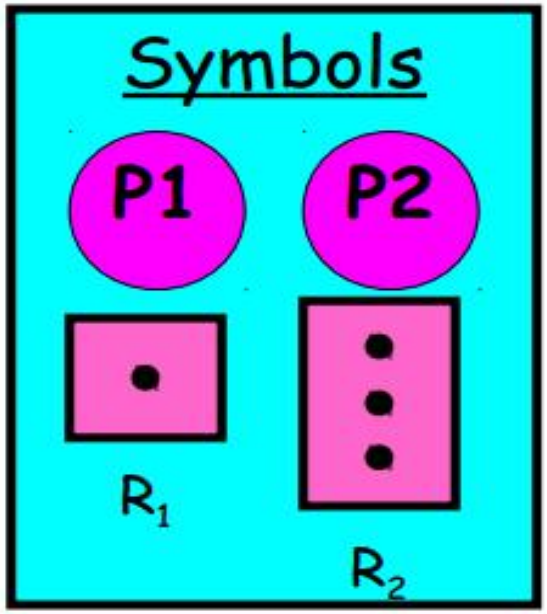
request edge – directed edge Pi ⟹ Rj

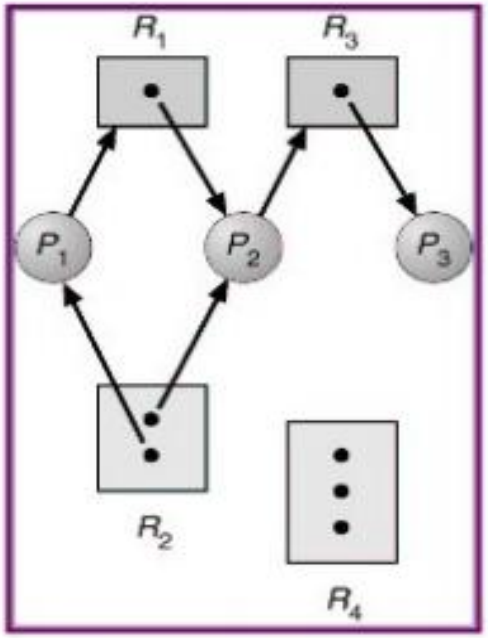
assignment edge – directed edge Rj ⟹ Pi



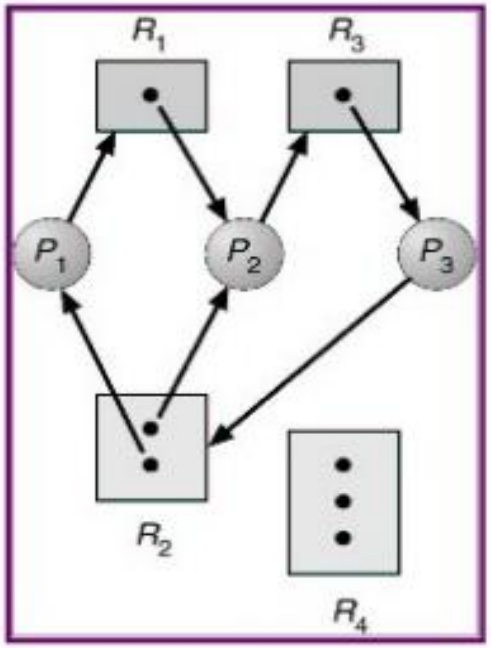
a). P1 is holding R1

b). P1 requests R1

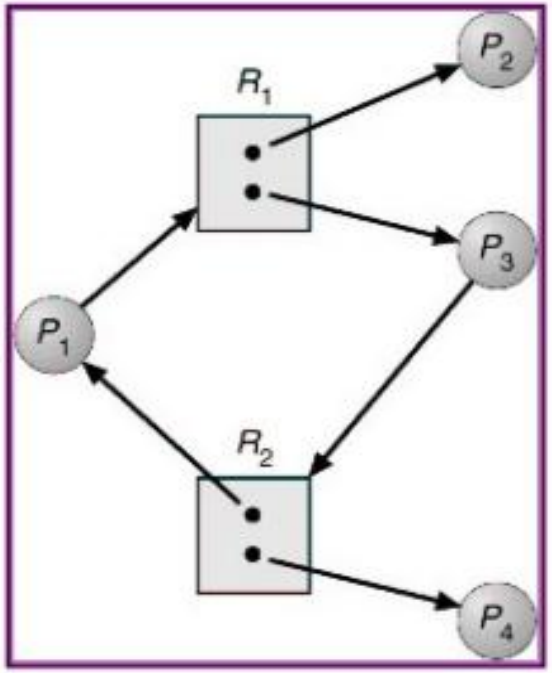




a).eg. of a Resource allocation graph



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

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:

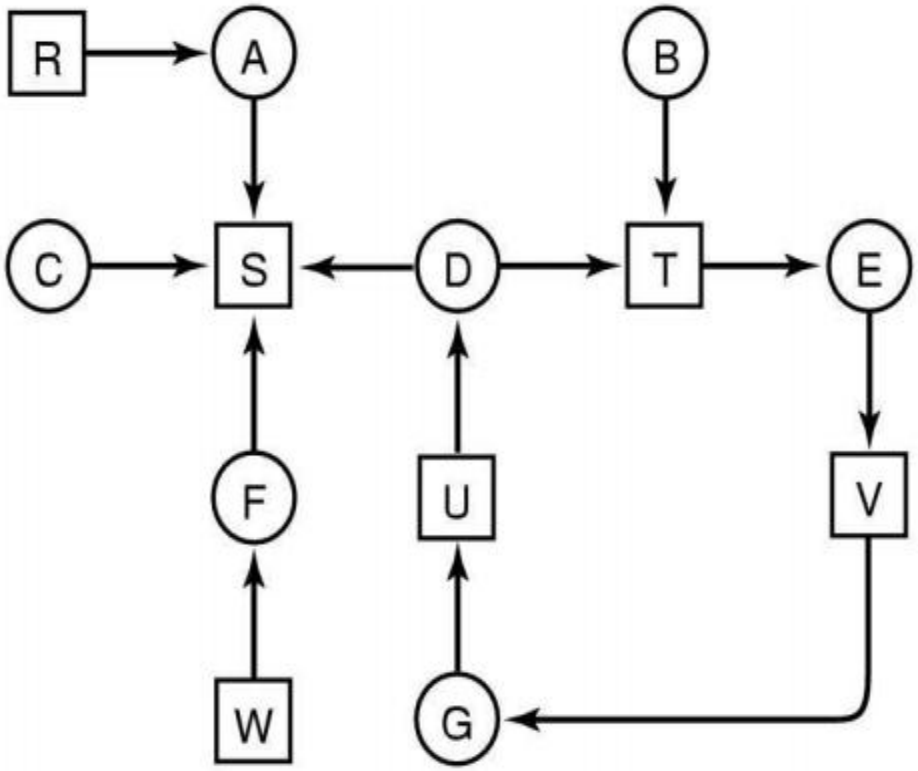
a system with 7 processes (A – G) , and 6 resources (R – W) are in following state.

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.

Benefits of the exokernel operating system

- Improved performance of applications
- More efficient use of hardware resources through precise resource allocation and revocation
- Easier development and testing of new operating systems
- Each user-space application is allowed to apply its own optimized memory management

Drawbacks of the exokernel operating system

- Reduced consistency
- Complex design of exokernel interfaces

Shell

- A shell is software that provides an interface for an operating system's users to provide access to the kernel's services.
- An OS starts a shell for each user when the user logs in or opens a terminal or console window.
- It is named a shell because it is the outermost layer around the operating system kernel.

A kernel is a program that:

- Controls all computer operations.
- Coordinates all executing utilities
- Ensures that executing utilities do not interfere with each other or consume all system
- resources.
- Schedules and manages all system processes.

- By interfacing with a kernel, a shell provides a way for a user to execute utilities and programs.

Shell Contd…

- The shell also provides a user environment that you can customize using initialization files.
- These files contain settings for user environment characteristics, such as:

❑Search paths for finding commands.

❑Default permissions on new files.

❑Values for variables that other programs use.

❑Values that you can customize.

END