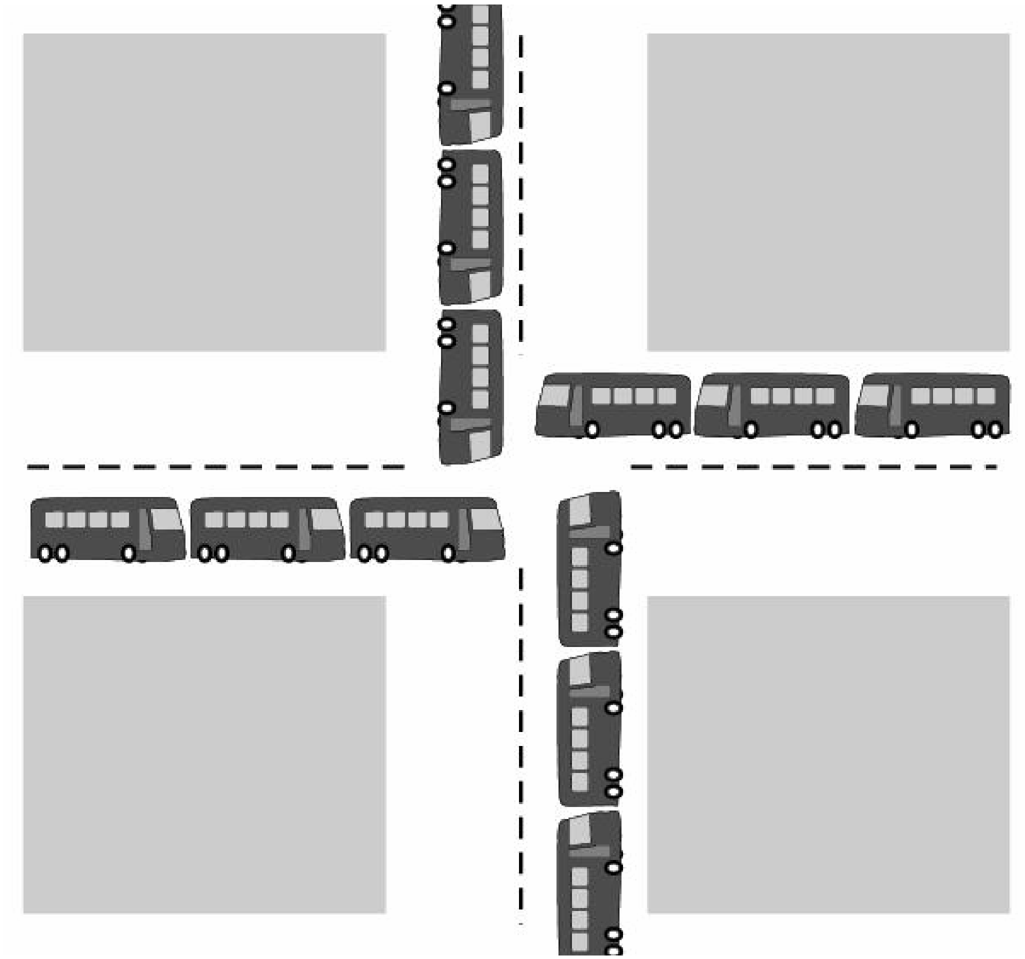
Deadlock

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.



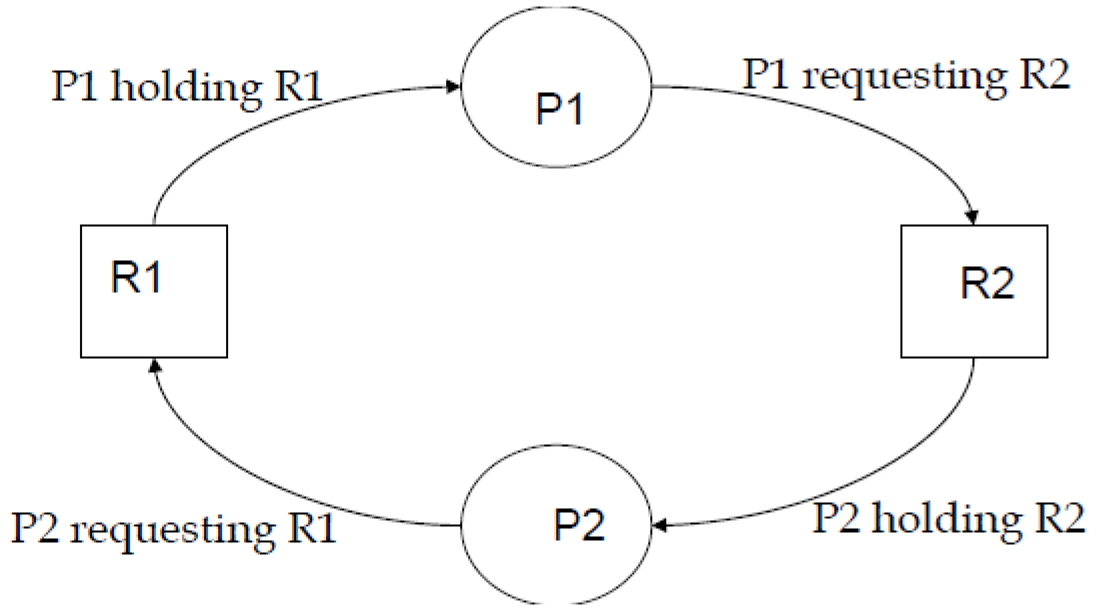
Resource Deadlock

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

- Request the resource.
- Use the resource.
- 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

Condition for Deadlock

- **Mutual Exclusion**: Process claims exclusive control of resources they require.
- **Hold and Wait**: Processes hold resources already allocated to them while waiting for additional resources.
- **No preemption**: Resources previously granted can not be forcibly (or compulsorily) taken away from the process.
- **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.

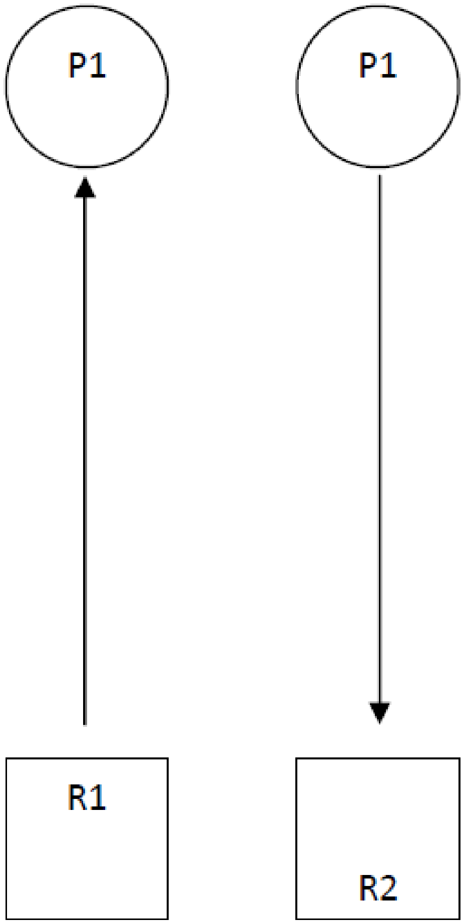
Resource Allocation Graph(RAG)

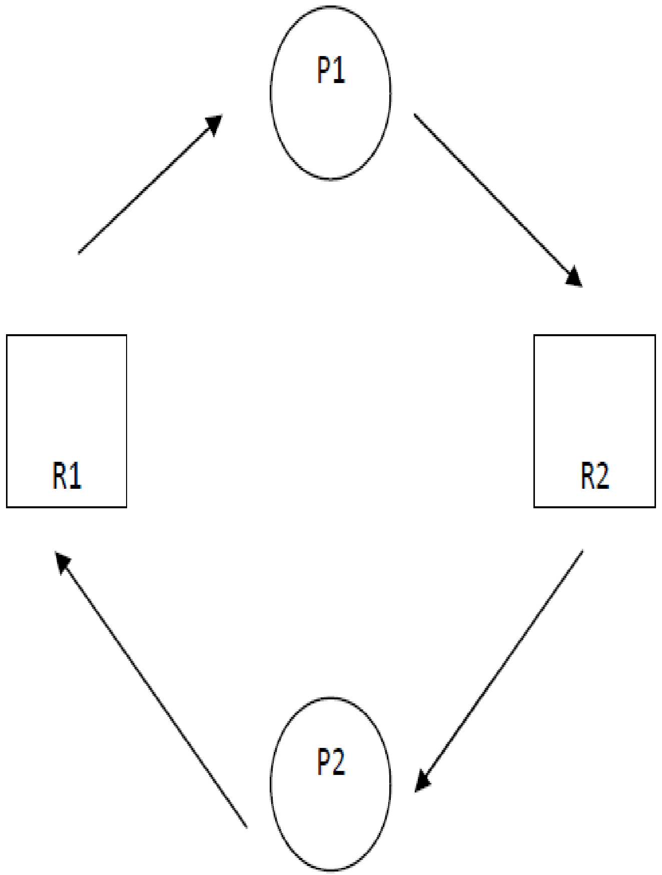
- Deadlock can be described more precisely in terms of a directed graph called a system resource-allocation graph.

- It Consist of Set of vertices and set of edges.

- The set of vertices V is partitioned into two types of nodes:
  - P={p1,p2,…….pn}, the set of process in the system.
  - R={R1, R2, R3,……..Rn}, the set of resource types in the system.

Example





Method for Handling Deadlock

- 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

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

By implementing policy that makes one of the conditions impossible, the system will be assured to be deadlock free.

First : Denying Mutual Exclusion

Sharable resources do not require mutually exclusive access such as read only shared file.

But!!! Some resources are strictly non-sharable, mutually exclusive control required.

So We can not prevent deadlock by denying the mutual exclusion

Second: Denying Hold and Wait:

- Policy:
  - 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. **A process that needs several popular resources may have to wait indefinitely, because at least one of the resources that it needs is always allocated to some other process.**

Third: Denying No-preemption

- policy
  - 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 .

Denying Circular Wait:

- Policy
  - All resources are uniquely numbered, and processes must request resources in linear ascending order.
  - The only ascending order prevents the circular wait .
- Problem:
  - Difficult to maintain the resource order; dynamic update in addition of new resources.

Deadlock Avoidance

Deadlock-prevention algorithms ensure that at least one of the necessary conditions for deadlock cannot occur and hence that deadlocks cannot hold. Possible side effects of preventing deadlocks **are low device utilization and reduced system throughput.**

Alternative

- For example, in a system with one tape drive and one printer, the system might need to know that process P will request first the tape drive and then the printer before releasing both resources, whereas process Q will request first the printer and then the tape drive.
- Assign resource that leads to the system into safe state.

Deadlock Avoidance

Each request requires that in making this decision the system consider:

- the resources currently available,
- the resources currently allocated to each process,
- the future requests and releases of each process.

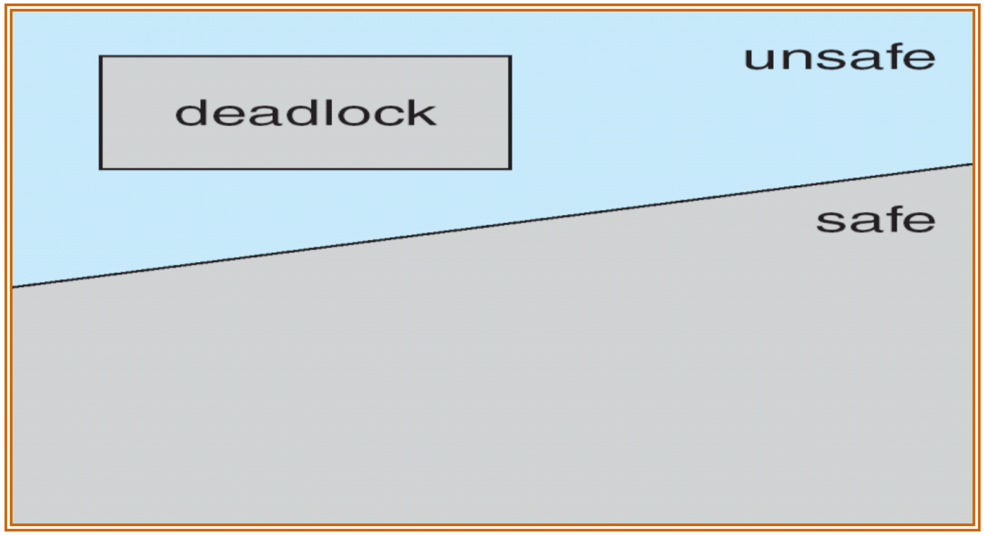
Initially the system is in safe state. Whenever a process request resource that is currently available, the system must decide whether the resource can be allocated immediately or whether the process must wait.

Safe state and Deadlock State

- If a system is in a safe state ⇒No deadlocks.

- If a system is in unsafe state ⇒Possibility of deadlock.

- Avoidance ⇒ensure that a system will never reach an unsafe state.



Example

- A state is said to be safe if it is not deadlocked and there is a some scheduling order in which every process can run to completion.
- To illustrate, we consider a system with 12 magnetic tape drives and three processes:

Process	**Max**	**Has**
P1	10	5
P2	4	2
P3	9	2

- Thus, there are 3 free tape drives.

Continued

- At time t_0, the system is in a safe state. The sequence <p2,p1,p3> satisfies the safety condition.
- A system can go from a safe state to an unsafe state. Suppose that, at time t1 , process p3 requests and is allocated one more tape drive. The system is no longer in a safe state.
- Our mistake was in granting the request from process p3 for one more tape drive.

Processes	**Max.**	**Has**
P1	10	5
P2	4	2
P3	9	2+1

Deadlock Avoidance

Banker's Algorithm

- Models on the way of banking system to ensure that the bank never allocates its available cash such that it can no longer satisfy the needs of all its customers.

- When a process request the set of resources, the system determine whether the allocation of these resources will left the system in safe state, if it will the resources are allocated, otherwise process must wait until some other process release enough resources.

- Data structures: Available, Max, Allocation & Need.

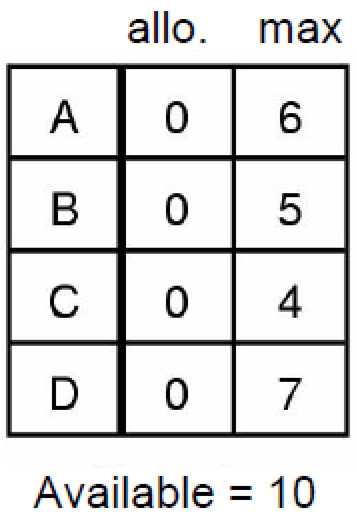
  need = max –allocation.

Banker’s Algorithm

1. If a process is requesting more resource than its need, then raise an error
2. If request is valid, the number of resources requested is compared to the number of available resources.
3. If not enough resources are available, the request is denied and the process must wait.
4. If sufficient resources are available, the modified version of the current state **is created** that reflect the effect of granting request. The algorithm then checks if the modified version of the system is safe.
5. The resource is granted if and only if the modified version of the system state is safe.

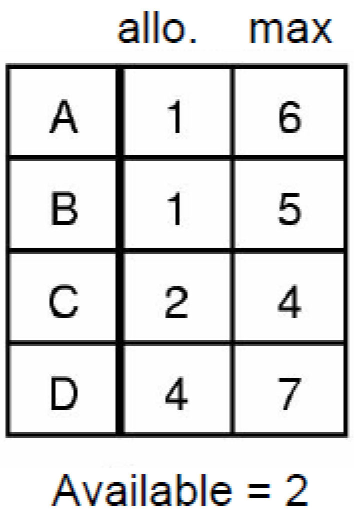
It ensure that the system is never in unsafe state. so it avoid the deadlock

Banker's Algorithm



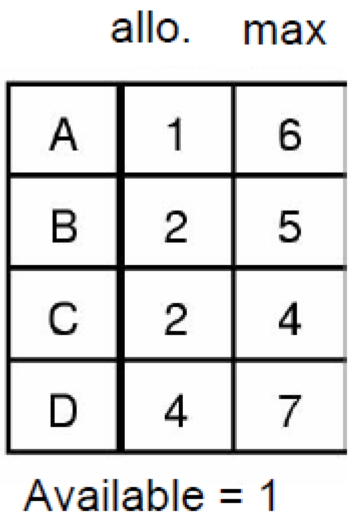
	allo.	max
A	0	6
B	0	5
C	0	4
D	0	7

Available = 10



	allo.	max
A	1	6
B	1	5
C	2	4
D	4	7

Available = 2



	allo.	max
A	1	6
B	2	5
C	2	4
D	4	7

Available = 1

Which one is unsafe?

Deadlock Avoidance
Problem with Banker

- 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,

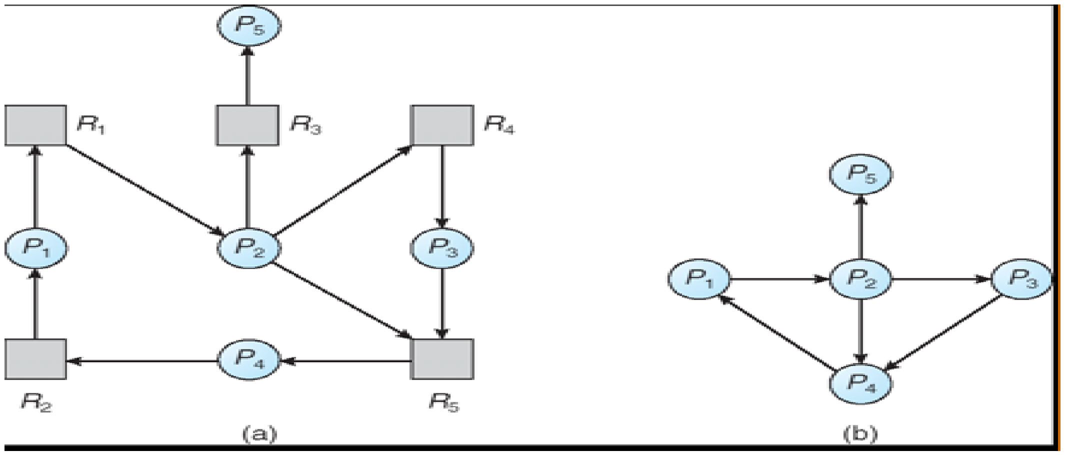
and recover from deadlock

when it does occur.

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

Deadlock Detection

- If all resource have only a single instance, then we can define a deadlock detection algorithm that use a variant of resource allocation graph, called wait for graph.
- Example



Example

- 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 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

- A deadlock exits in the system if and only if the wait for graph contains cycle.
- To detect deadlock the system needs to maintain the wait for graph and periodically invoke an algorithm that searches for a cycle in the graph.
- An algorithm to detect a cycle in graph requires an order of $o(n^2)$

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

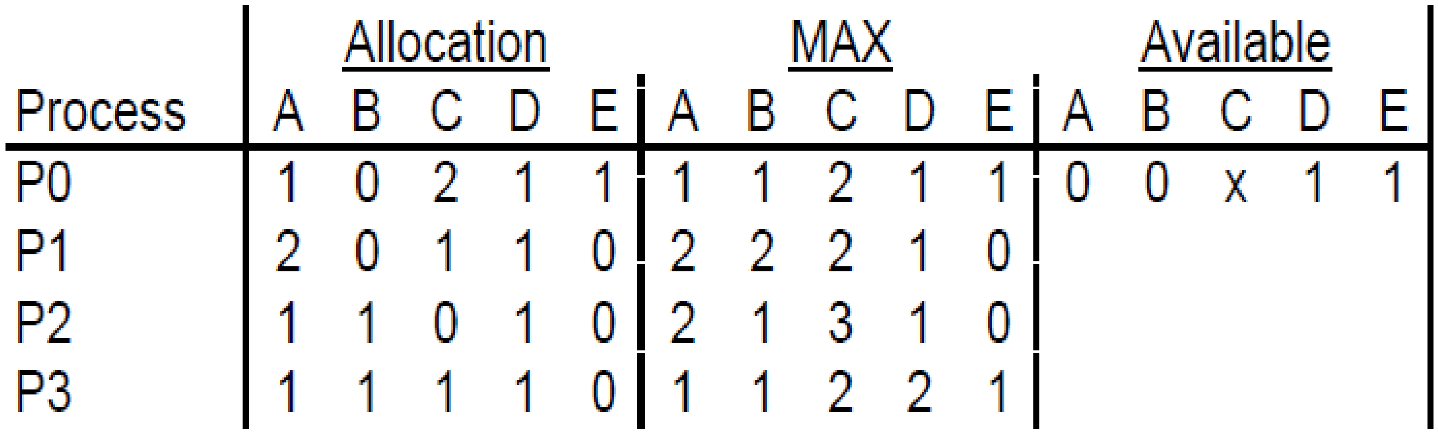
- For most operating systems, deadlock is a rare occurrence; So the problem of deadlock is ignored, like an ostrich sticking its head in the sand and hoping the problem will go away.

Exercise

1. Is it possible to have a deadlock involving only one process? Explain
2. Consider the following snapshot of a system (given bellow) Answer the following questions using Banker's algorithm:
   a) What is the content of the matrix **Need?**
   b) Is the system in a safe state?
   c) If a request from process P1 arrives for (0,4,2,0), can the request be granted immediately ?

	Allocation				MAX				Available			
Process	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

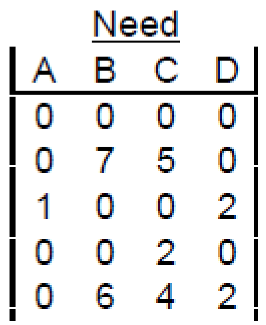
3. A system has four processes and five allocable resources. The current allocation and maximum needs are given bellow:
   – What is the smallest value of ***x for which this is a safe*** state based on the Banker's algorithm?



	Allocation					MAX					Available				
Process	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
P0	1	0	2	1	1	1	1	2	1	1	0	0	x	1	1
P1	2	0	1	1	0	2	2	2	1	0					
P2	1	1	0	1	0	2	1	3	1	0					
P3	1	1	1	1	0	1	1	2	2	1					

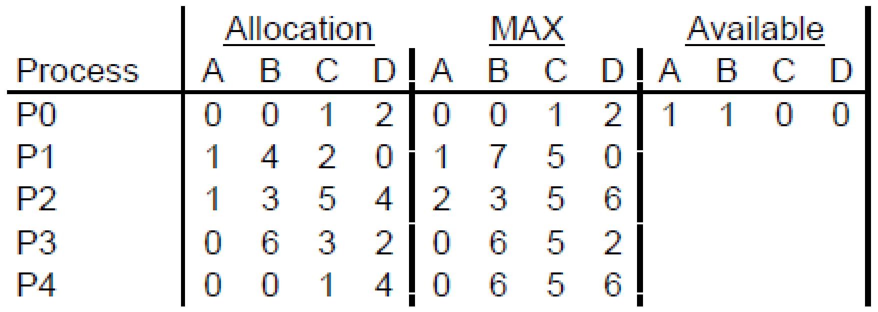
Solutions

1. No.
This follows directly from the hold and wait condition.
Hold and wait: a process holding at least one resource is waiting to acquire additional resources held by other processes



Need

A	B	C	D
0	0	0	0
0	7	5	0
1	0	0	2
0	0	2	0
0	6	4	2



	Allocation				MAX				Available			
Process	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	1	0	0
P1	1	4	2	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

2.
a. Need = Max – Allocation. The matrix is shown bellow on the left-hand side;
b. Yes, a sequence <P3, P4, P2, P1, P0> is safe;
c. Yes, there is a sequence <P0, P2, P1, P4, P3> which is safe.
The snapshot of the system after resources allocation is given bellow on the right-hand side.

Solution for 3

.

A: The matrix Need as shown on the right.

If ***x*** is 0, then we have an unsafe state immediately

If ***x*** is 1, then P3 can run, after it releases resources, P0 can run, after P2 and then P1, i.e. sequence <P3, P0, P2, P1> is safe, thus the system is in the safe state

	Need				
Process	A	B	C	D	E
P0	0	1	0	0	0
P1	0	2	1	0	0
P2	1	0	3	0	0
P3	0	0	1	1	1

Therefore the minimal value of ***x*** is 1

Paper Review on Deadlock