CSC-203 Operating Systems[3]

Lecture: 14-15-16

Instructor: Sanjog Sigdel

sigdelsanjog.com.np

Date: June 8, 2019

Deadlocks

Introduction, Definition, Deadlock Characterization, Preemptable and Non-preemptable resources, Resource Allocation Graph, Necessary Condition for Deadlock

Deadlocks

Deadlock can be defined formally as follows:

> A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.
>
> In other words, deadlock is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in non of the process can run, none of them can release any resources, and none of them can be awakened

Deadlocks

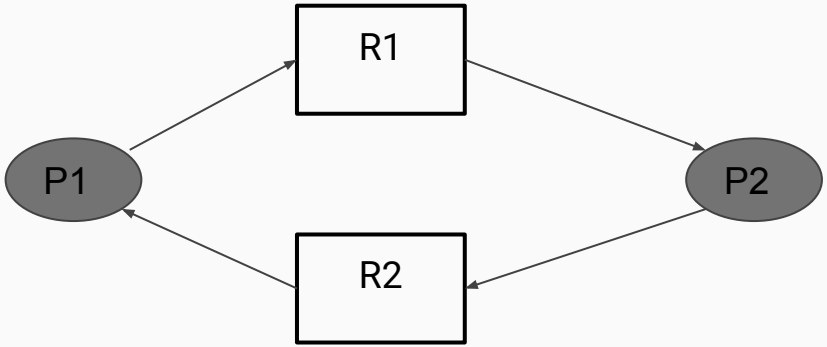
The resources may be either physical or logical.

Physical Resources: Printers, Tape, Drivers, Memory Space and CPU cycles

Logical Resources: Files, Semaphores, and Monitors

Deadlocks

Deadlocks



Program 1 requests resource R1 and receives it.

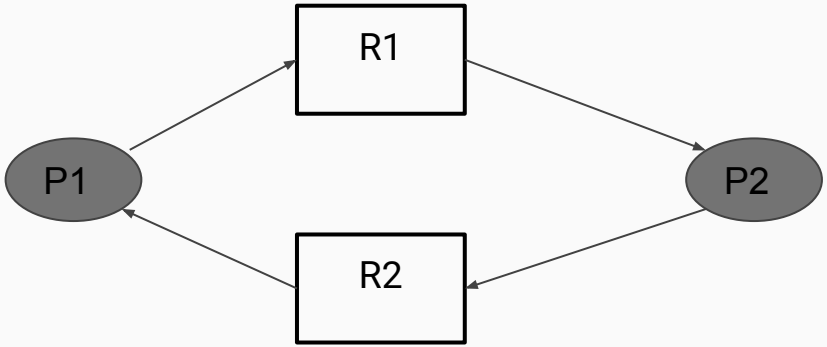
Program 2 requests resource R2 and receives it

Program 1 requests resource R2 and is queued up, pending the release of R2

Program 2 requests resource R1 and is queued up, pending the release of R1

Deadlocks

Deadlocks



Program 1 requests resource R1 and receives it.

Program 2 requests resource R2 and receives it

Program 1 requests resource R2 and is queued up, pending the release of R2

Program 2 requests resource R1 and is queued up, pending the release of R1

Lecture 14

Deadlocks

Deadlocks Characterization

- Mutual Exclusion Condition
- Hold and Wait Condition
- No-preemptive Condition
- Circular Wait Condition

Deadlocks

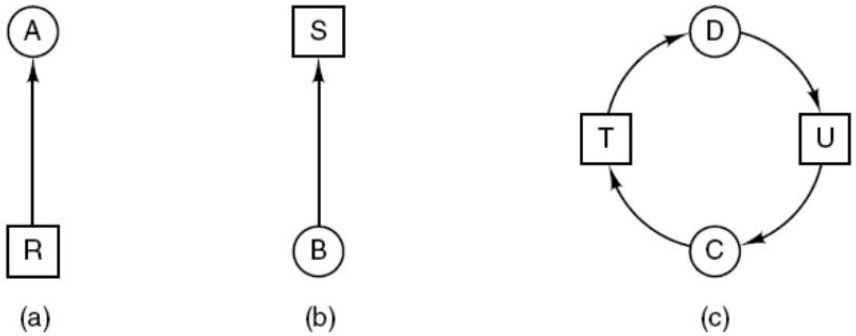
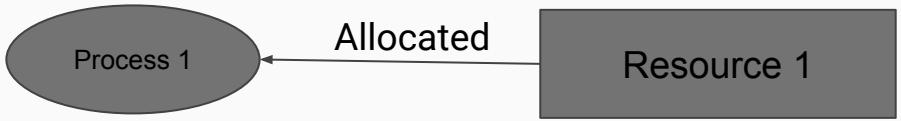


Figure 6-3. Resource allocation graphs. (a) Holding a resource. (b) Requesting a resource. (c) Deadlock.

Mutual Exclusion Condition

The resource involved are non-sharable, At least one resource must be held in a non-shareble mode i.e. only one process at a time claims exclusive control of the resource.

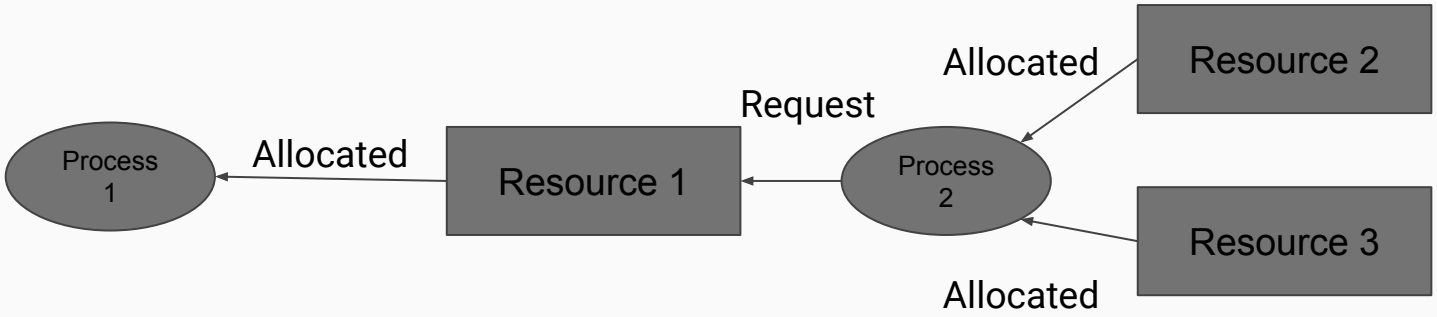
If another process requests that resource, the requesting process must be delayed until the resource has been released



Deadlocks

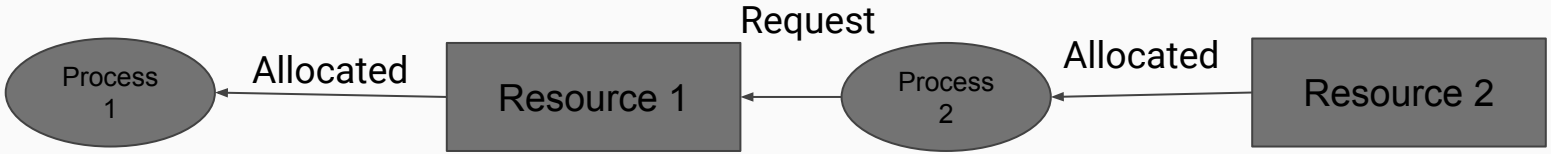
Hold and Wait Condition

Requesting process hold already, resources while waiting for requested reaoruces, there must exist a process that is holding a resource already allocated to it while waiting for additional resource that are currently being held by other process



No-Preemptive Condition

Resources already allocated to a process cannot be preempted, Resource cannot be removed from the processes are used to completion or released voluntarily by the process holding it.



Circular Wait Condition

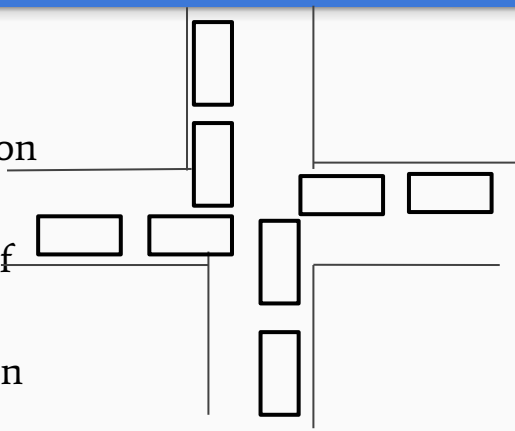
A set {P0, P1, P2, ... Pn} of waiting process must exist such that P1 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, ..., Pn-1 is waiting for a resource that is held by Pn and Pn is waiting for a resource that is held by P0.

The processes in the system form a circular list or chain where each process in the list is waiting for a resource held by the next process in the list.

Consider the traffic deadlock example:

Deadlocks

Consider each section of the street as a resource



- **Mutual exclusion condition applies,** since only one vehicle can be on a section of the street at a time
- **Hold-and-wait condition applies,** since each vehicle is occupying a section of the street and waiting to move on the next section
- **No-preemptive condition applies,** since a section of the street that is a section of that is occupied by a vehicle cannot be taken away from it
- **Circular wait condition applies,** since each vehicle is waiting on the next vehicle to move. I.e. each vehicle in the traffic is waiting for a section held by the next vehicle in the traffic

Deadlocks

In general, four strategies are used for dealing with deadlocks.

1. Just ignore the problem. Maybe if you ignore it, it will ignore you.

2. Detection and recovery. Let them occur, detect them, and take action.

3. Dynamic avoidance by careful resource allocation.

4. Prevention, by structurally negating one of the four conditions

Deadlock Prevention

- Attacking the mutual exclusion condition
- Attacking the hold and wait condition
- Attacking the no preemption condition
- Attacking the circular wait condition

Deadlocks

Preemptable and Non-preemptable Resources

Preemptable

- Resource that can be taken away from the process with no ill effects
- Example

Non-preemptable

- Resource that cannot be taken away from process(without causing ill effect)
- Example: CD resources are not premptable at an arbitrary moment.

Lecture 15

Deadlocks

Resource Allocation Graph,Ostrich Algorithms, Deadlock prevention, Safe and Unsafe States, Deadlock Avoidance (Bankers algorithm for Single and Multiple Resource Instances)

Resource Allocation Graph

- Explained as what is the state of the system in terms of processes and resources
- How many resources are available, how many are allocated and what is the request of each process
- Everything can be presented in terms of the diagram

Advantage: sometimes it is possible to see a deadlock directly using the RAG, but then it might not be able to know that by looking at the table

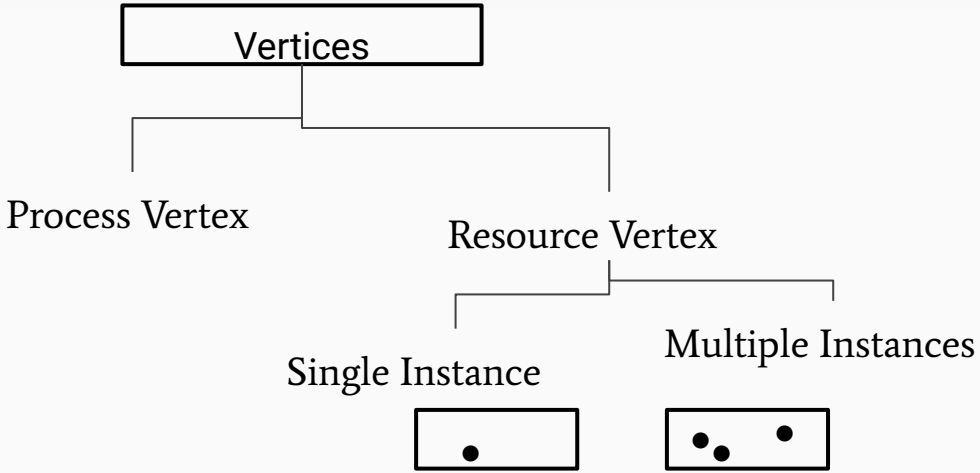
Tables are better if the system contains lots of process and resource and Graph is better if the system contains less number process and resource

Resource Allocation Graph

In RAG vertices are two types:

1. **Process Vertex** - Every process will be presented as a process vertex. Generally process vertex will be represented with a circle
2. **Resource vertex** - Every resource will be represented as a resource vertex.
    a. **Single instance type resource** - It represents a box, inside the box, there will be one dot. So the number of dots indicates how many instances are present of each resource type.
    b. **Multi-resource instance type resource**- It also represents as a box, inside the box there will be many dots present

Deadlocks


Vertices
Process Vertex
Resource Vertex
Single Instance
Multiple Instances

Resource Allocation Graph

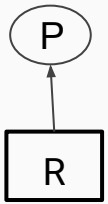
In RAG vertices are two types:

1. **Process Vertex** - Every process will be presented as a process vertex. Generally process vertex will be represented with a circle
2. **Resource vertex** - Every resource will be represented as a resource vertex.
    a. **Single instance type resource** - It represents a box, inside the box, there will be one dot. So the number of dots indicates how many instances are present of each resource type.
    b. **Multi-resource instance type resource**- It also represents as a box, inside the box there will be many dots present

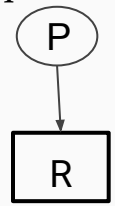
Deadlocks

Edges

Assign Edge



Request Edge



Deadlocks

Example: (Single instances RAG)

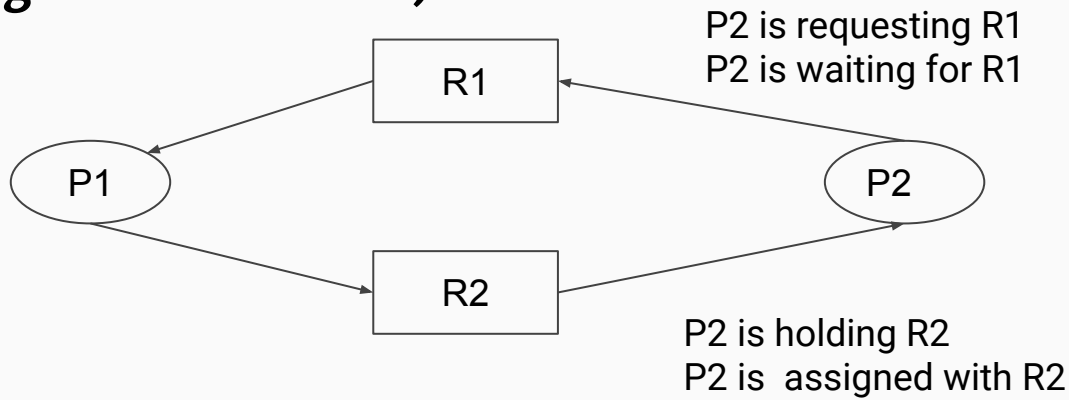


Fig: Single Instance Resource Type with Deadlock

Deadlocks

Example: (Single instances RAG)

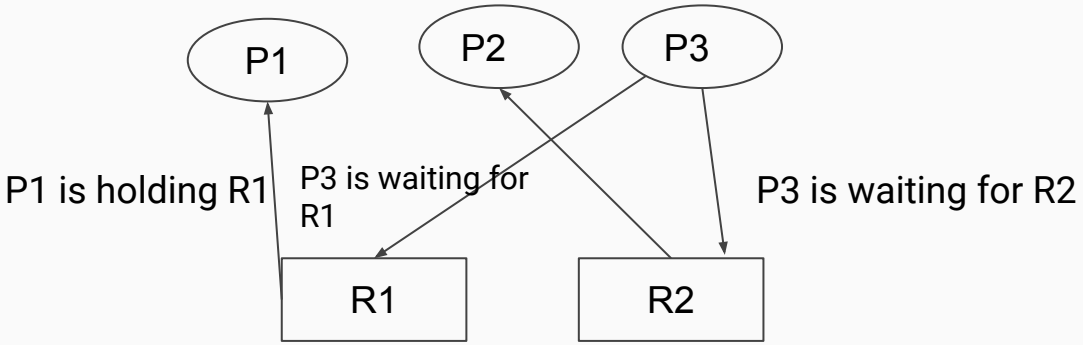


Fig: Single Instance Resource Type without Deadlock

Deadlocks

Example: (Multi-instances RAG)

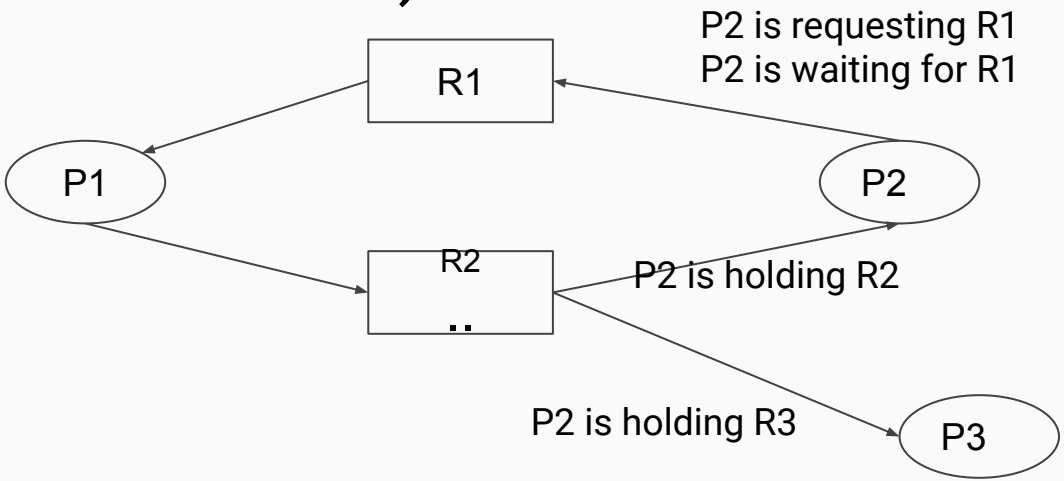


Fig: Multi Instances Resource without Deadlock

Deadlocks

For above example, it is not possible to say the RAG is in a safe state or in an unsafe state. Let's construct the allocation matrix and request matrix:

Process	Allocation Resources		Request Resources	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	0	0

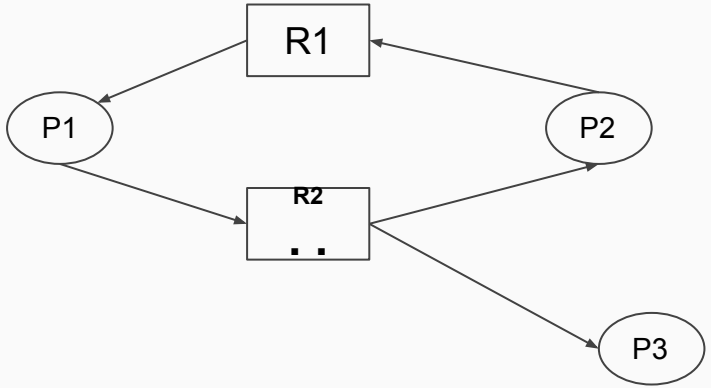


Fig: Multi Instances Resource without Deadlock

Deadlocks

Checking Deadlock(Safe or Not)

Available=[0 0] (As P3 doesn't require any extra resources to complete the execution and after completion P3[01] P3 releases its own resource)

New Available = [0 0] (As using new available resource we can satisfy the requirement of process P1 and P1 also P1[1 0] release its previous resource)

New AVAILABLE=[1 1] (Now easily we can satisfy the requirement of Process P2 [0 1]

New Available=[1 2]

Even though there is a cycle still there is no deadlock

Process	Allocation Resources		Request Resources	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	0	0

Deadlocks

Example: (Multi-instances RAG)

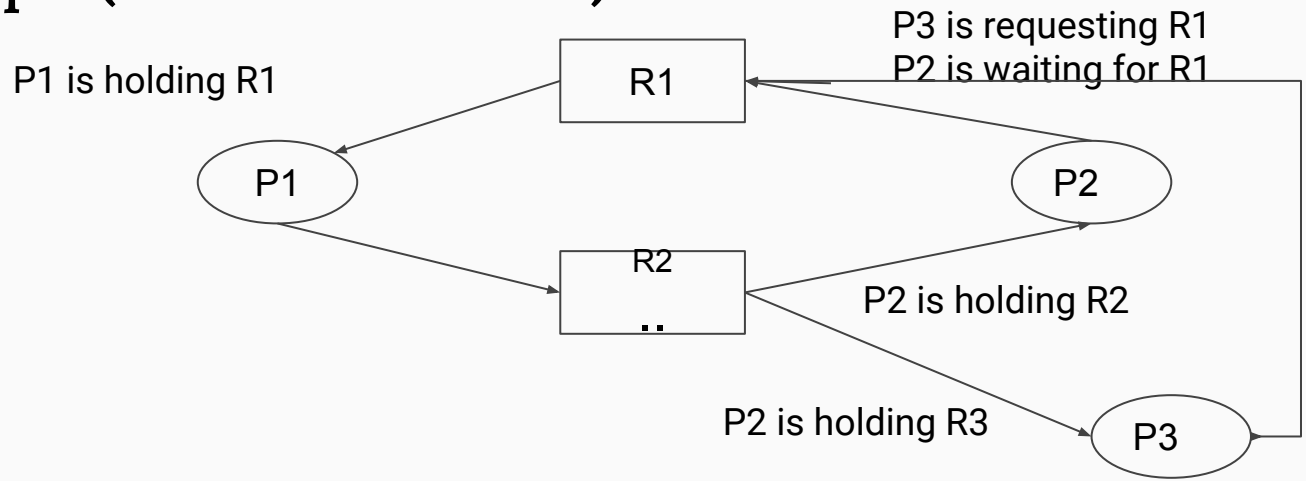


Fig: Multi Instances Resource With Deadlock

Deadlocks

Process	Allocation Resources		Request Resources	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	1	0

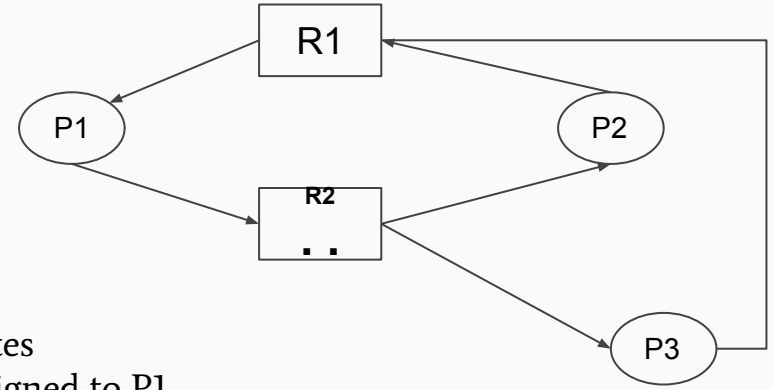


Fig: Multi Instances Resource without Deadlock

After P3 uses R2, P2 is assigned with R2 so it takes R2 and executes
But R1 is requested by P3 and P2 at the same time while R1 is assigned to P1
Therefore deadlock exists. So in case of RAG with multi-instance resource type, the cycle is a necessary condition for deadlock but not sufficient.

Deadlock Prevention

Deadlock prevention can be done by eliminating the four of the following factors which might cause deadlock

- Elimination of “Mutual Exclusion” Condition
- Elimination of “Hold and Wait” Condition
- Elimination of “No-Preemption” Condition
- Elimination of “Circular Wait” Condition

Ostrich Approach

- Strategy of ignoring potential problems on the basis that they may be exceedingly rare
- It is named for the ostrich effect which is defined as “to stick one’s head in the sand and pretend there is no problem”
- It is used when it is more cost-effective to allow the problem to occur than to attempt its prevention
- Example: if each PC deadlocks once per 10 years, then one reboot may be less painful than the restrictions needed to prevent it.
- UNIX and Windows operating systems take this approach

Deadlocks

Safe and Unsafe States

- A state is said to be safe if there is at least one way for all users to finish
- If there aren't any such way for users to finish the process execution than the process leads to unsafe state causing

Deadlocks

Safe State

Process	Has	Max
A	3	9
B	2	4
C	2	7
Free: 3		

- Total resources are 10
- 7 resources are already allocated
- 3 resources are free
- A needs 6 resource more to complete the execution
- B needs 2 resource more to complete the execution
- C needs 5 resource more to complete the execution

Process	Has	Max
A	3	9
B	2	4
C	2	7
Free: 3		

a)

Process	Has	Max
A	3	9
B	4	4
C	2	7
Free: 1		

b)

Process	Has	Max
A	3	9
B	0	-
C	2	7
Free: 5		

c)

Process	Has	Max
A	3	9
B	0	-
C	7	7
Free: 0		

d)

Process	Has	Max
A	3	9
B	0	-
C	0	-
Free: 7		

e)

Process	Has	Max
A	9	9
B	0	-
C	0	-
Free: 1		

f)

Safe State

Process	Has	Max
A	3	9
B	2	4
C	2	7
Free: 3		

a)

Process	Has	Max
A	4	9
B	2	4
C	2	7
Free: 2		

b)

Process	Has	Max
A	4	9
B	4	4
C	2	7
Free: 0		

c)

Unsafe State

Process	Has	Max
A	4	9
B	0	-
C	2	7
Free: 4		

d)

Banker's Algorithm

Banker's Algorithm, sometimes referred to as the detection algoriths, is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources, and then makes an "s-table" check to test for deadlock conditions for all the other pending activities, before deciding whether allocation should be allowed to continue.

Banker's Algorithm

For the Banker's Algorithm to work, it looks three things:

- How much of each resource each process could possibly request**[MAX]**
- How much of each resource each process is currently holding**[ALLOCATED]**
- How much of each resource the system currently has available**[AVAILABLE]**

Deadlocks

Banker's ALgorithm:- Input:- Processes *n: no. of processes,*

*** any 2 out of 3(Max, Need, Allocation)** *m: no. of resources* **]**

*** available or total number of resources**

Step1: flag[i]=0 for i=0 to(n-1) & find Need[n][m]=Max[n][m]-allocation[n][m]

Step2: Find a process Pi such that:- flag[i]=0 & Needi <=Available

Step3: If such i exists then

flag[i]=1, available = available + Allocation

Goto step 2,

Else goto step 4

Step4: If flag[i] = 1 for all i

Then system is in safe state otherwise unsafe state

Deadlocks

From the given values(Allocation, Maximum and Available Resources) construct the Need Matrix. Identify if system is in safe state and if yes then find the safe sequence

Process	Allocation				Max				Available			
	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				

Deadlocks

Process	Allocation				Max				Available			
	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				

In exam question can give either Available resources of the total resources.

Total= Available + Allocated

Deadlocks

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

Need = Maximum - Allocation

Next check for the **safe states:**
I.e. Given the available resources, Can you satisfy the need of the metrics?

Deadlocks

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

Based upon the available resources need of the following process can be satisfied:

P0 and P3

P1, P2 and P4 have either of the resources deficient to fulfill the need

Safe sequences can be achieved from P0 and P3

We can execute any one process

Deadlocks

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

Execute P0

After execution available resources will increase

i.e. resources allocated to P0 are now free

So available resources are **[1 5 3 2] (Add Allocated P0 + Available)**

Safe Sequence: P0

Deadlocks

Process	Allocation				Max				Available				Need			
	A	B	C	D	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	0	0	0	0
P1	1	0	0	0	1	7	5	0	1	5	3	2	0	7	5	0
P2	1	3	5	4	2	3	5	6	2	8	8	6	1	0	0	2
P3	0	6	3	2	0	6	5	2					0	0	2	0
P4	0	0	1	4	0	6	5	6					0	6	4	2

With the help of newly available resources, check which need can be satisfied

Execute P2

After execution, available resources will increase

i.e. resources allocated to P2 are now free

So available resources are **[2 8 8 6] (Add Allocated P0 + Available)**

Safe Sequence: P0 P2

Deadlocks

Process	Allocation				Max				Available				Need			
	A	B	C	D	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	0	0	0	0
P1	1	0	0	0	1	7	5	0	1	5	3	2	0	7	5	0
P2	1	3	5	4	2	3	5	6	2	8	8	6	1	0	0	2
P3	0	6	3	2	0	6	5	2	2	14	11	8	0	0	2	0
P4	0	0	1	4	0	6	5	6					0	6	4	2

With the help of newly available resources, check which need can be satisfied

Execute P3

After execution, available resources will increase

i.e. resources allocated to P3 are now free

So available resources are **[2 14 11 8] (Add Allocated P0 + Available)**

Safe Sequence: P0 P2 P3

Deadlocks

Process	Allocation				Max				Available				Need			
	A	B	C	D	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	0	0	0	0
P1	1	0	0	0	1	7	5	0	1	5	3	2	0	7	5	0
P2	1	3	5	4	2	3	5	6	2	8	8	6	1	0	0	2
P3	0	6	3	2	0	6	5	2	2	14	11	8	0	0	2	0
P4	0	0	1	4	0	6	5	6	3	14	11	8	0	6	4	2

With the help of newly available resources, check which need can be satisfied

Execute P1

After execution, available resources will increase

i.e. resources allocated to P1 are now free

So available resources are **[3 14 11 8] (Add Allocated P0 + Available)**

Safe Sequence: P0 P2 P3 P1

Deadlocks

Process	Allocation				Max				Available				Need			
	A	B	C	D	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	0	0	0	0
P1	1	0	0	0	1	7	5	0	1	5	3	2	0	7	5	0
P2	1	3	5	4	2	3	5	6	2	8	8	6	1	0	0	2
P3	0	6	3	2	0	6	5	2	2	14	11	8	0	0	2	0
P4	0	0	1	4	0	6	5	6	3	14	11	8	0	6	4	2
									3	14	12	12				

With the help of newly available resources, check which need can be satisfied

Execute P4

After execution, available resources will increase

i.e. resources allocated to P4 are now free

So available resources are **[3 14 12 12] (Add Allocated P0 + Available)**

Safe Sequence: P0 P2 P3 P1 P4

Problem With Banker's Algorithm

- Algorithm requires fixed number of resources; some processes dynamically change the number of resources
- Algorithm requires the number of resources in advance; it is very difficult to predict the resources in advanced
- Algorithm predict all process execute within finite time but the system doesn't guarantee it.

Deadlock Detection for Single and Multiple Instances

Deadlock Detection algorithms

- For **Single** Instance
  - **Wait-for graph**
- For **Multiple** Instances
  - **Banker's Algorithm**

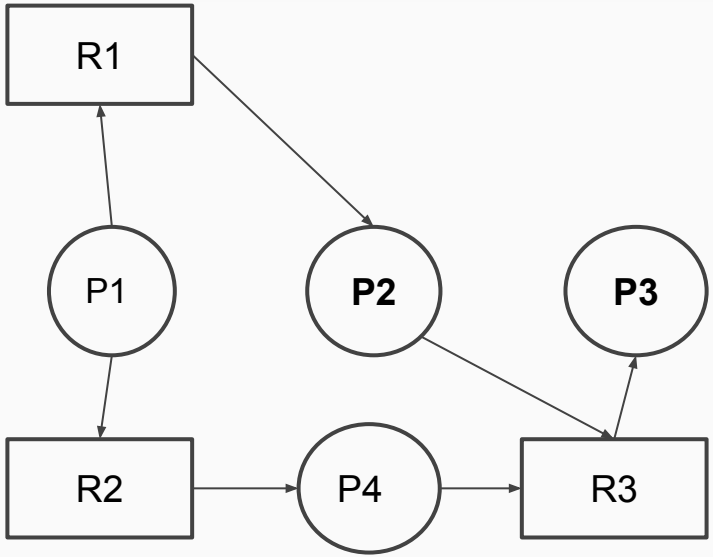
Single Instance

In single instance resource if a cycle is detected than the system is in deadlock state

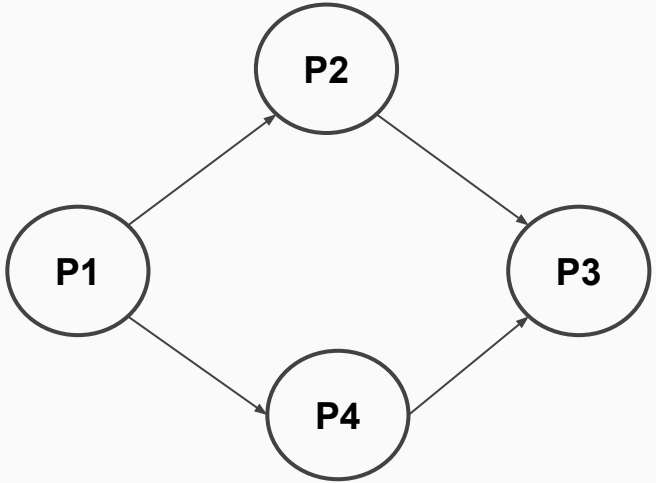
Multiple Instances

In a multiple instances resource if a cycle if detected than it is only necessary condition for occuring deadlock but not sufficient.

Deadlocks



Resource Allocation Graph



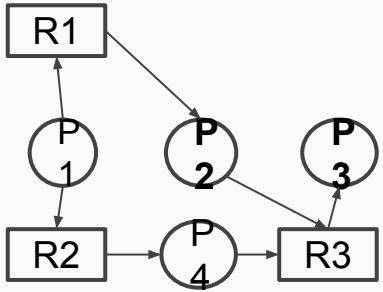
Wait-for Graph

Deadlocks

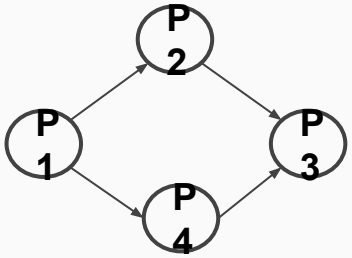
- A **Wait for Graph** is drawn corresponding to the resource allocation graph.
- Cycle is identified on the Wait-for-graph

- If cycle exists in the wait-for graph than the Deadlock exists

There is no cycle in the given example, so the deadlock doesn't exist.

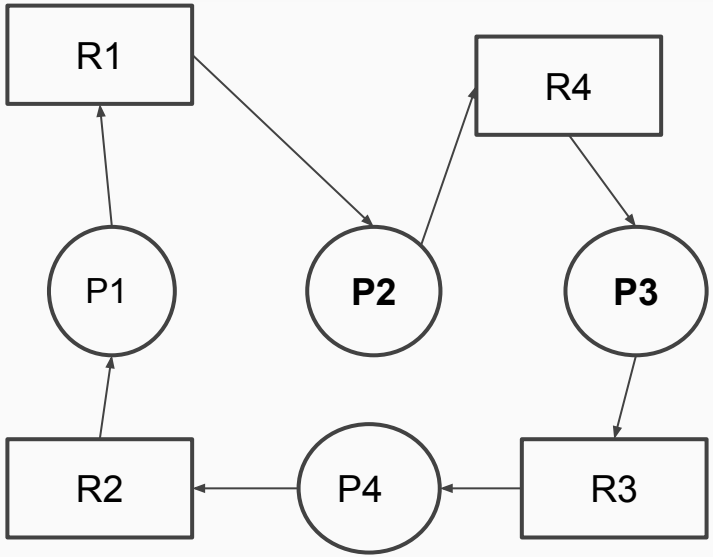


Resource Allocation Graph

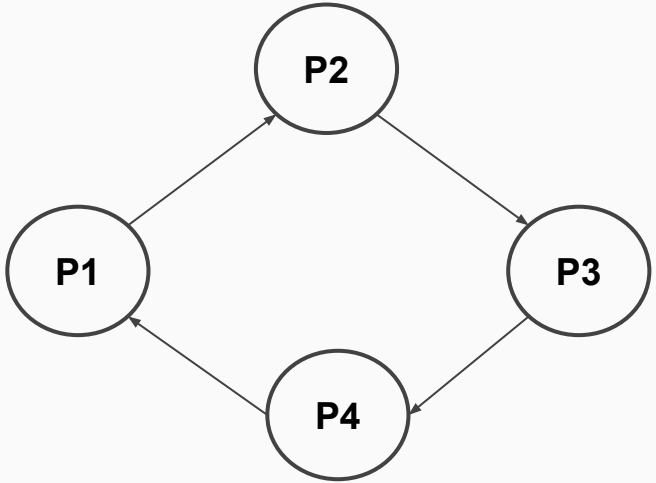


Wait-for Graph

Deadlocks



Resource Allocation Graph



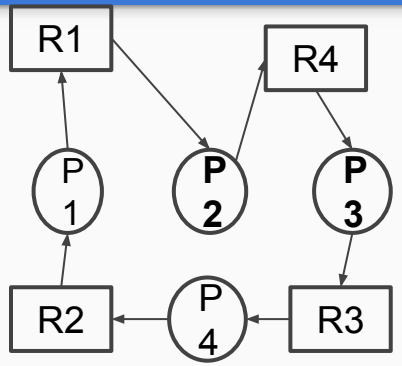
Wait-for Graph

Deadlocks

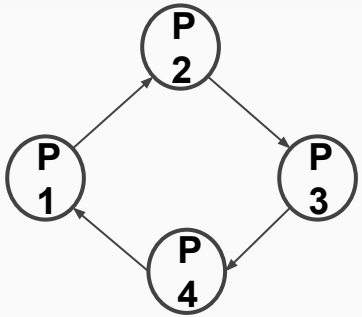
- A **Wait for Graph** is drawn corresponding to the resource allocation graph.
- Cycle is identified on the Wait-for-graph

- If cycle exists in the wait-for graph than the Deadlock exists

There is a cycle{ P1 P2 P3 P4} in the given example, so the deadlock exists.



Resource Allocation Graph



Wait-for Graph

Deadlocks

Multiple Instances

Banker's Algorithm

Process	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	1			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

Total:
A B C
7 2 6

Safe State: P0
Deadlock State: P1, p2, p3, p4

Deadlock Recovery

- Traditional operating system such as Windows doesn't deal with deadlock recovery
- Real time operating system use Deadlock Recovery

Recovery mechanisms are applied for the following:

- For Resource
- For Process

Deadlock Recovery For Resource

Preempt the resource

- We can snatch one of the resources from the owner of the resource(process) and give it to the other process with the expectation that it will complete the execution and will release the resource sooner
- Well. choosing resource which will be snatched is goind to be a bit difficult

Deadlock Recovery For Resource

Rollback to a safe state

- System passes through various states to get into the deadlock state
- The operating system can roll back the system to the previous safe state.
- For this, OS needs to implement check pointing at every state
- The moment we get deadlock, we will rollback all the allocations to get into previous safe state

Deadlock Recovery For Process

Kill a process

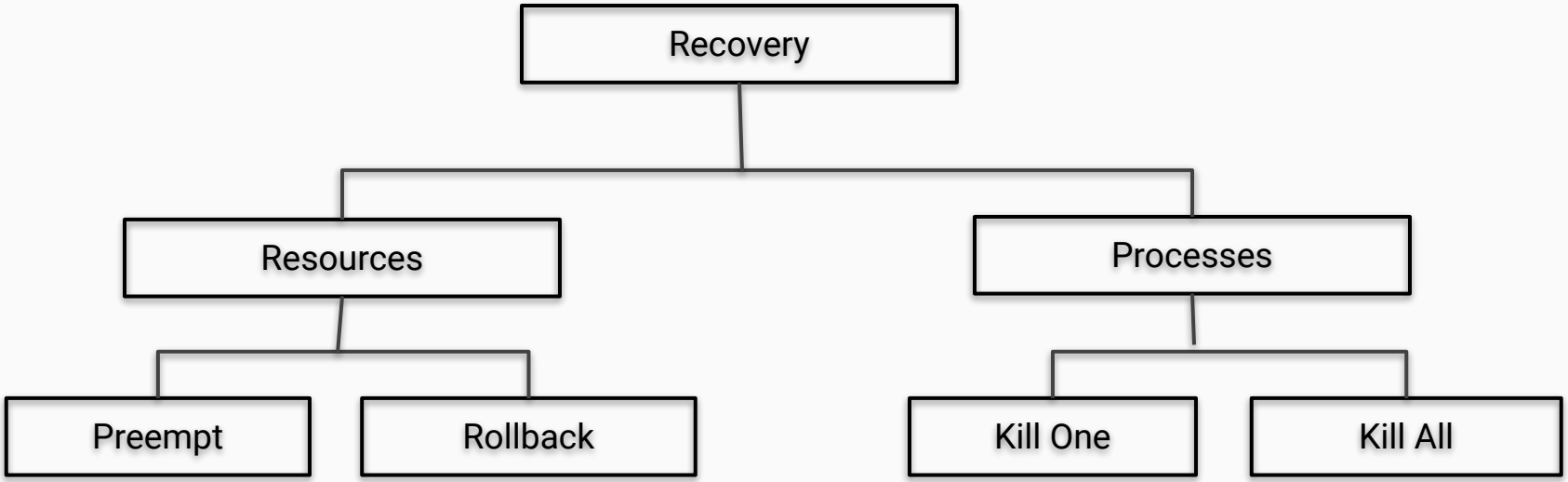
- Killing a process can solve our problem bt the bigger concern is to decide which process to kill.
- Generally, operating system kills a process which has done least amount of work until now

Deadlock Recovery For Process

Kill all processes

- This is not suggestable approach but can be implemented if the problem becomes very serious
- Killing all the process will lead to inefficiency in the system because all the process will execute from the starting

Deadlocks


Recovery
Resources
Processes
Preempt
Rollback
Kill One
Kill All