GLA UNIVERSITY
MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A+ Grade by NAAC
3.46 Score
12-B Status from UGC

OPERATING SYSTEMS

DEADLOCKS

Class Presentations on Operating System

by Subhash Chand Agrawal

Resources

- A resource used by process can be a piece of hardware such as
  - Memory
  - CPU cycle
  - Printer
  - CD ROM

or a piece of information such as

  - a file
  - a record within a file
  - a shared variable
  - a critical section
  - etc.

Types of Resources

Resources come in two flavors: preemptable and nonpreemptable.

A **preemptable resource** is one which can be allocated to a given process for a period of time, then be allocated to another process and then be reallocated to the first process without any ill effects.

Examples of preemptable resources include:

- memory
- buffers
- CPU
- etc.

- A **nonpreemptable resource** cannot be taken from one process and given to another without side effects.
- One obvious example is a printer: certainly we would not want to take the printer away from one process and give it to another in the middle of a print job

DEADLOCKS

EXAMPLES:

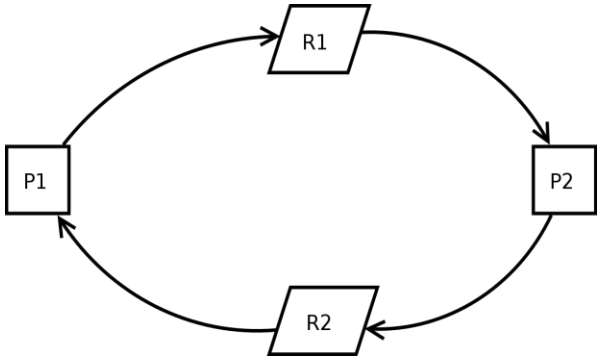
- You can't get a job without experience; you can't get experience without a job.

BACKGROUND:

Each resource type R_i has W_i instances.

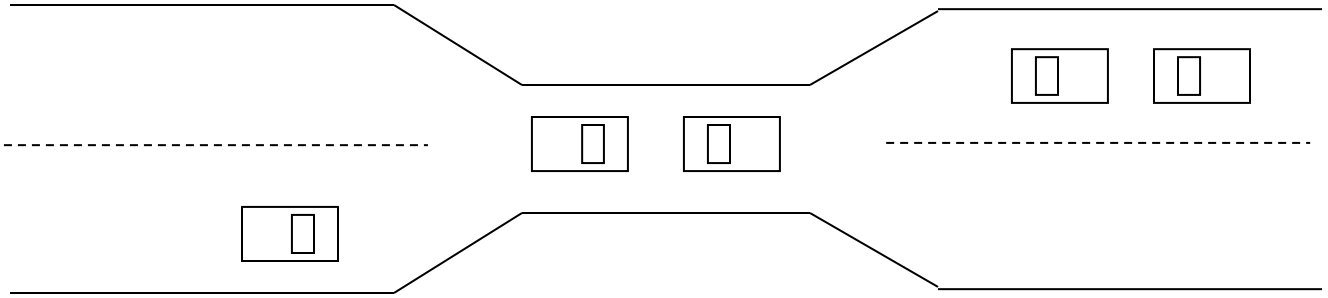
Under normal operation, a resource allocations proceed like this:

1. Request a resource (suspend until available if necessary).
2. Use the resource.
3. Release the resource.



Bridge Crossing Example

DEADLOCKS



- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.

Q1.

- A single-lane bridge connects the two villages X and Y. Farmers in the two villages use this bridge to deliver their produce to the neighboring town. The bridge can become deadlocked if both a X and a Y farmer get on the bridge at the same time. Using exactly one semaphore, design an algorithm that prevents deadlock. Do not be concerned about starvation and inefficiency.

Solution

```
semaphore ok_to_cross = 1;
void enter_bridge()
 {
P(ok_to_cross);
 }
void exit_bridge()
{
V(ok_ to_cross);
 }
```

GLA UNIVERSITY MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A+ Grade by NAAC 3.46 Score
12-B Status from UGC

DEADLOCK CHARACTERISATION

NECESSARY CONDITIONS

ALL of these four **must** happen simultaneously for a deadlock to occur:

Mutual exclusion

At least one resource must be held in a non sharable mode.

only one process at a time can use a resource.

Hold and Wait

A process holds a resource while waiting for another resource, currently held by another process.

GLA UNIVERSITY MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A+ Grade by NAAC 3.46 Score
12-B Status from UGC

DEADLOCK CHARACTERISATION

No Preemption

a resource can be released only voluntarily by the process holding it, after that process has completed its task.

Circular Wait

There exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

RESOURCE ALLOCATION GRAPH

A visual (mathematical) way to determine if a deadlock has, or may occur.

G = (V, E) The graph contains nodes and edges.

- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system

RESOURCE ALLOCATION GRAPH

- **E** **request edge** – directed edge $P_i \rightarrow R_j$

  assignment edge – directed edge $R_j \rightarrow P_i$

➢An arrow from the **process** to **resource** indicates the process is **requesting** the resource.

➢An arrow from **resource** to **process** shows an instance of the resource has been **allocated** to the process.

➢ Process is a circle,

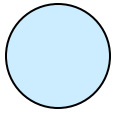
➢resource type is square;

➢dots represent number of instances of resource in type.

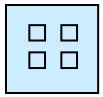
➢ Request points to square, assignment comes from dot.

Resource-Allocation Graph (Cont.)

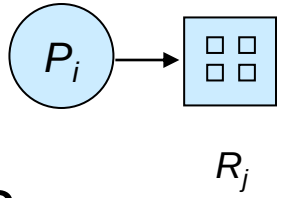
- Process



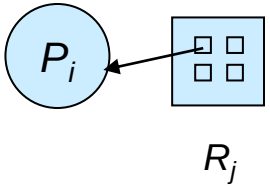
- Resource Type with 4 instances



- P_i requests instance of R_j

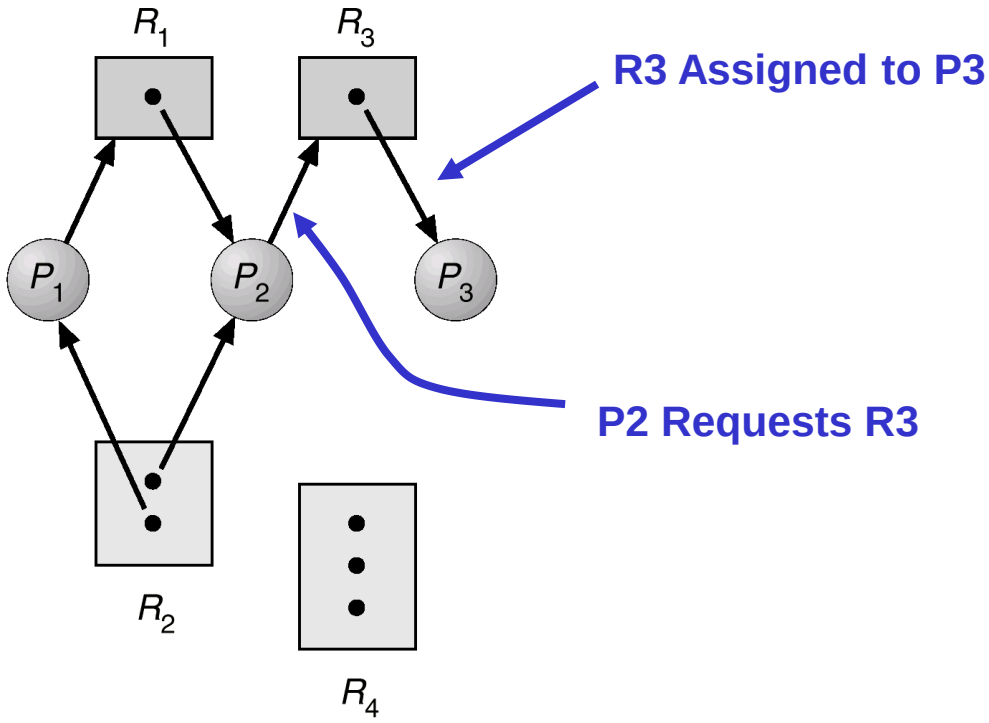


- P_i is holding an instance of R_j



RESOURCE ALLOCATION GRAPH

Resource allocation graph

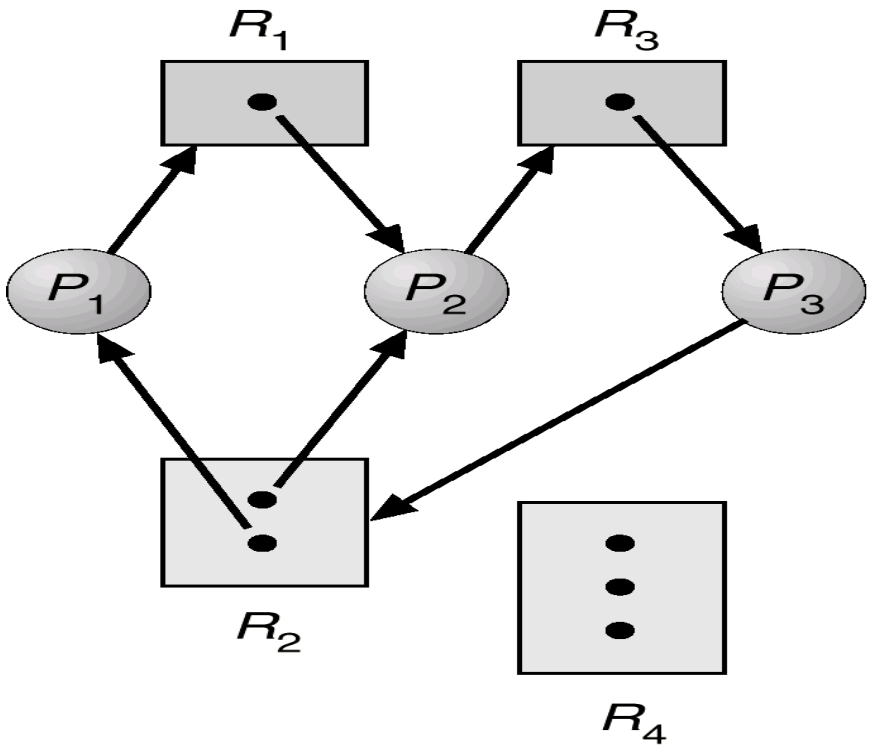


RESOURCE ALLOCATION GRAPH

- If the graph contains no cycles, then no process is deadlocked.
- If there is a cycle, then:
  a) If resource types have multiple instances, then deadlock MAY exist.
  b) If each resource type has 1 instance, then deadlock has occurred.

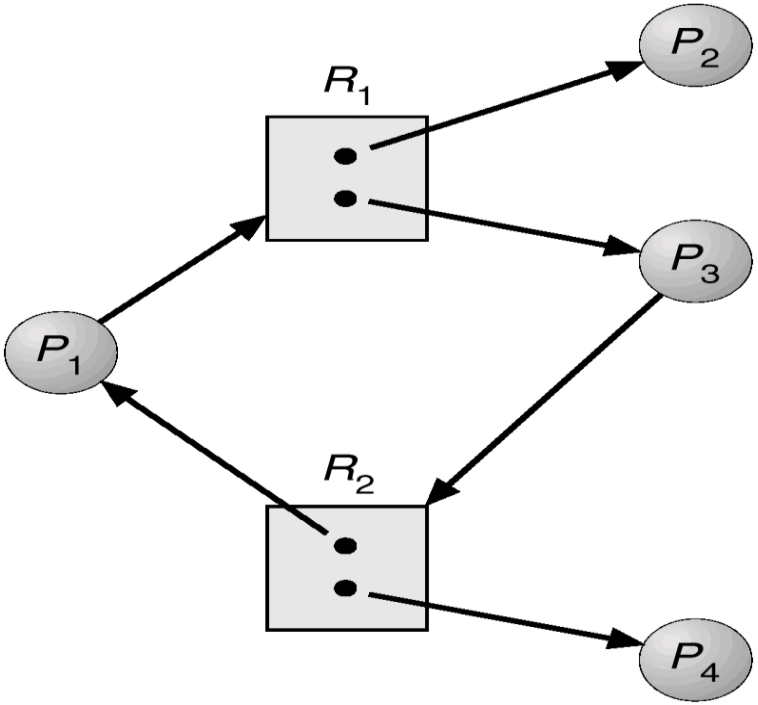
RESOURCE ALLOCATION GRAPH

Resource allocation graph with a deadlock.



RESOURCE ALLOCATION GRAPH

Resource allocation graph with a cycle but no deadlock.



HOW TO HANDLE DEADLOCKS – GENERAL STRATEGIES

There are three methods:

Ignore Deadlocks pretend that deadlocks never occur in the system :

Most Operating systems do this!!

Ensure deadlock **never** occurs using either

Prevention Prevent any one of the 4 conditions from happening.

Avoidance In this method, the request for any resource will be granted only if the resulting state of the system doesn't cause any deadlock in the system. Any process continues its execution until the system is in a safe state. Once the system enters into an unsafe state, the operating system has to take a step back.

Example: Banker's Algorithm

HOW TO HANDLE DEADLOCKS – GENERAL STRATEGIES

Allow deadlock to happen. This requires using both:

If deadlocks do occur, the operating system must detect and resolve them.

Deadlock Detection

Deadlock detection algorithms, such as the Wait-For Graph, are used to identify deadlocks, and

Deadlock Recovery

Recovery algorithms, such as the Rollback and Abort algorithm, are used to resolve them. The recovery algorithm releases the resources held by one or more processes, allowing the system to continue to make progress.

Deadlock Prevention

Do not allow one of the four conditions to occur.

Mutual Exclusion

a) Shared entities (read only files) don't need mutual exclusion (and aren’t susceptible to deadlock.)

(b) Prevention not possible, since some devices are naturally non-sharable.

(c) We cannot force a resource to be used by more than one process at the same time since it will not be fair enough and some serious problems may arise in the performance.

Deadlock Prevention

Do not allow one of the four conditions to occur.

Hold and wait:

!(Hold and wait) = !hold or !wait

a) Collect all resources before execution. (**By eliminating wait)**

b) A particular resource can only be requested when no others are being held. (**By eliminating hold)**

c) Utilization is low, starvation possible.

(Possibility of getting starved will be increases due to the fact that some process may hold a resource for a very long time.)

- **Challenges:**
- As a process executes instructions one by one, it cannot know about all required resources before execution.
- Releasing all the resources a process is currently holding is also problematic as they may not be usable by other processes and are released unnecessarily.
- **For example:** When Process1 releases both Resource2 and Resource3, Resource3 is released unnecessarily as it is not required by Process2.

Deadlock Prevention

No preemption:

- Deadlock arises due to the fact that a process can't be stopped once it starts.
- However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.

(Consider a printer is being used by any process. If we take the printer away from that process and assign it to some other process then all the data which has been printed can become inconsistent and ineffective and also the fact that the process can't start printing again from where it has left which causes performance inefficiency.)

Deadlock Prevention

Circular wait:

a) impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

$F: R \longrightarrow N$

Tape drive, disk drive, printers

F(Tape drive)=1

F(disk drive)=5

F(printer)=12

$F(R_j) > F(R_i)$

Deadlock Avoidance

If we have prior knowledge of how resources will be requested, it's possible to determine if we are entering an "unsafe" state.

Possible states are:

Deadlock No forward progress can be made.

Unsafe state A state that **may** allow deadlock.

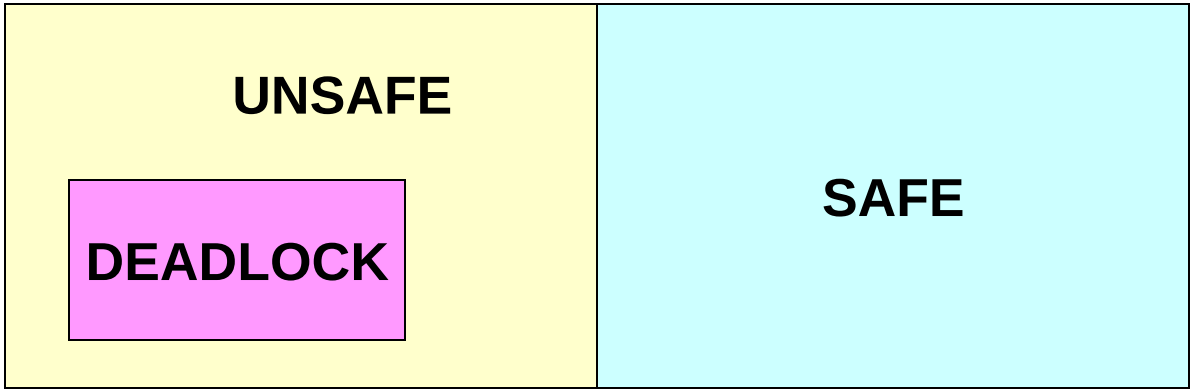
Deadlock Avoidance

Safe state A state is safe if a sequence of processes exist such that there are enough resources for the first to finish, and as each finishes and releases its resources there are enough for the next to finish.

The rule is simple: If a request allocation would cause an unsafe state, do not honor that request.

Deadlock Avoidance

NOTE: All deadlocks are unsafe, but all unsafes are NOT deadlocks.



Only with luck will processes avoid deadlock.

O.S. can avoid deadlock.

Deadlock Avoidance

Let's assume a very simple model: each process declares its maximum needs. In this case, algorithms exist that will ensure that no unsafe state is reached.

There are multiple instances of the resource in these examples.

EXAMPLE:

There exists a total of 12 tape drives. The current state looks like this:

<Available=3>

In this example, < p1, p0, p2 > is a workable sequence.

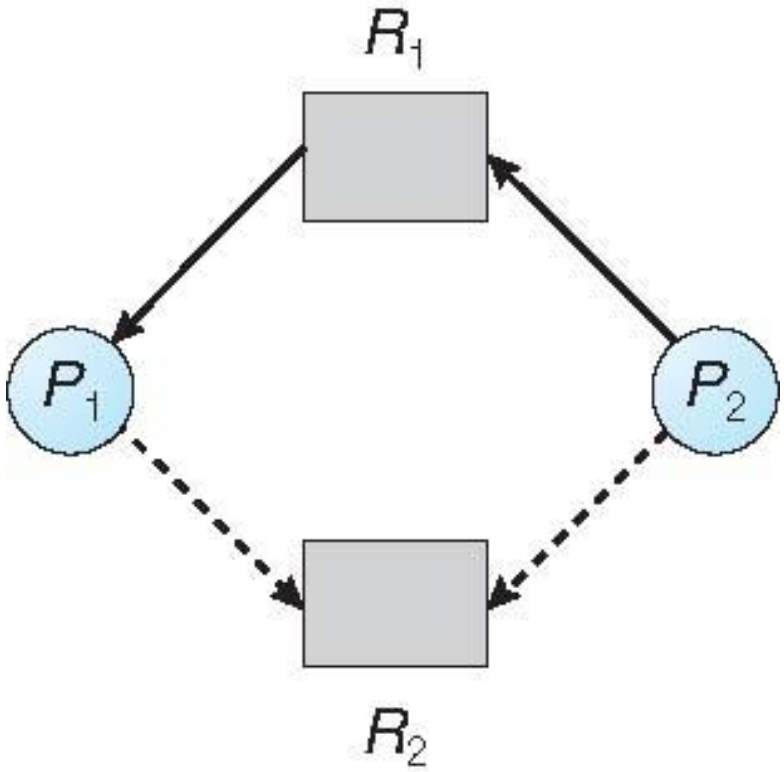
if process P2 requests and is granted one more tape drive?What happens then?

Process	Max Needs	Allocated	Current Needs
P0	10	5	5
P1	4	2	2
P2	9	2	7

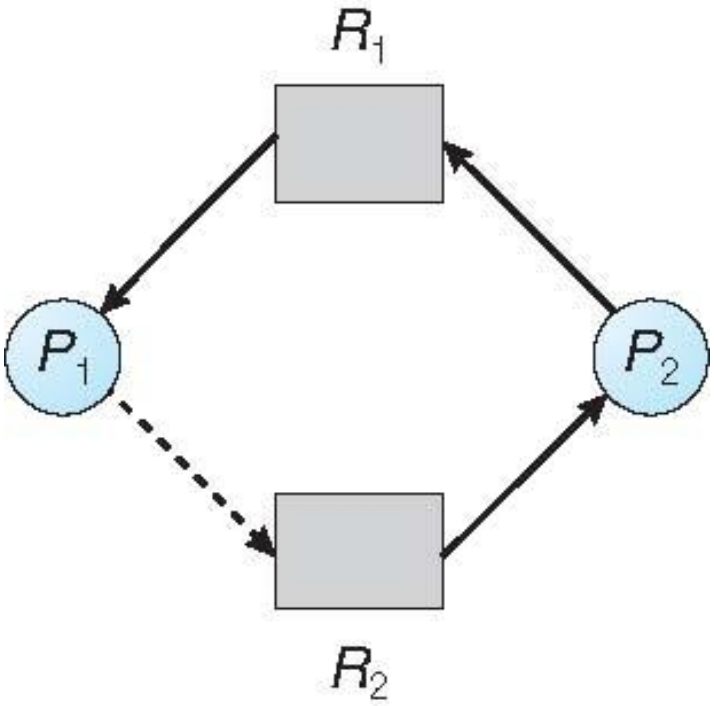
Resource-Allocation Graph Scheme

- **Claim edge** $P_i \rightarrow R_j$ indicated that process P_i may request resource R_j; represented by a dashed line
- Claim edge converts to request edge when a process requests a resource
- Request edge converted to an assignment edge when the resource is allocated to the process
- When a resource is released by a process, assignment edge reconverts to a claim edge
- Resources must be claimed *a priori* in the system

Resource-Allocation Graph



Unsafe State In Resource-Allocation Graph



Resource-Allocation Graph Algorithm

- Suppose that process P_i requests a resource R_j
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph

Deadlock Avoidance

Safety Algorithm

A method used to determine if a particular state is safe.

It's safe if there exists a sequence of processes such that for all the processes, there's a way to avoid deadlock:

The algorithm uses these variables:

Need[I] – the remaining resource needs of each process.

Work - Temporary variable – how many of the resource are currently available.

Deadlock Avoidance

Safety Algorithm

Finish[I] – flag for each process showing we've analyzed that process or not.

Let **work** and **finish** be vectors of length **m** and **n** respectively.

Now tell, What is the size of array or matrix?

Available?

Max need?

Allocation?

Need?

Deadlock Avoidance

Safety Algorithm

1. Initialize work = available

   Initialize finish[i] = false, for i = 0,1,2,3,..n-1

2. Find an i such that:

   finish[i] == false and need[i] <= work

   If no such i exists, go to 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

Safety Algorithm

Consider a system with: five processes, P0 → P4, three resource types, A, B, C. Type A has 10 instances, B has 5 instances, C has 7 instances. At time T0 the following snapshot of the system is taken.

Max Needs = allocated + can-be-requested

	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3	3	3	2
P1	3	2	2	2	0	0	1	2	2			
P2	9	0	2	3	0	2	6	0	0			
P3	2	2	2	2	1	1	0	1	1			
P4	4	3	3	0	0	2	4	3	1			

Is the system in a safe state?

<P1, P3, P4, P2, P0>

Deadlock Avoidance

Safety Algorithm

P1 requests one additional resource of type A, and two more of type C.

Request1 = (1,0,2).

Is Request1 < available?

	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3	2	3	0
P1	3	2	2	3	0	2	0	2	0			
P2	9	0	2	3	0	2	6	0	0			
P3	2	2	2	2	1	1	0	1	1			
P4	4	3	3	0	0	2	4	3	1			

<P1, P3, P4,P0,P2>

Produce the state chart as if the request is Granted and see if it's safe.

Deadlock Avoidance

Do these examples:

P4 request additional (3,3,0)

Can the request be granted?

Request can not be granted since the resources are not available

	←	Alloc	→		←	Need	→		←	Avail	→
	A	B	C		A	B	C		A	B	C
P0	0	1	0		7	4	3		2	3	0
P1	3	0	2		0	2	0				
P2	3	0	2		6	0	0				
P3	2	1	1		0	1	1				
P4	0	0	2		4	3	1				

Deadlock Avoidance

P0 request additional (0,2,0)
Can the request be granted?

	←	Alloc	→		←	Need	→		←	Avail	→
	A	B	C		A	B	C		A	B	C
P0	0	3	0		7	2	3		2	1	0
P1	3	0	2		0	2	0				
P2	3	0	2		6	0	0				
P3	2	1	1		0	1	1				
P4	0	0	2		4	3	1				

Even resources are available, since the resulting state is unsafe

Deadlock Detection

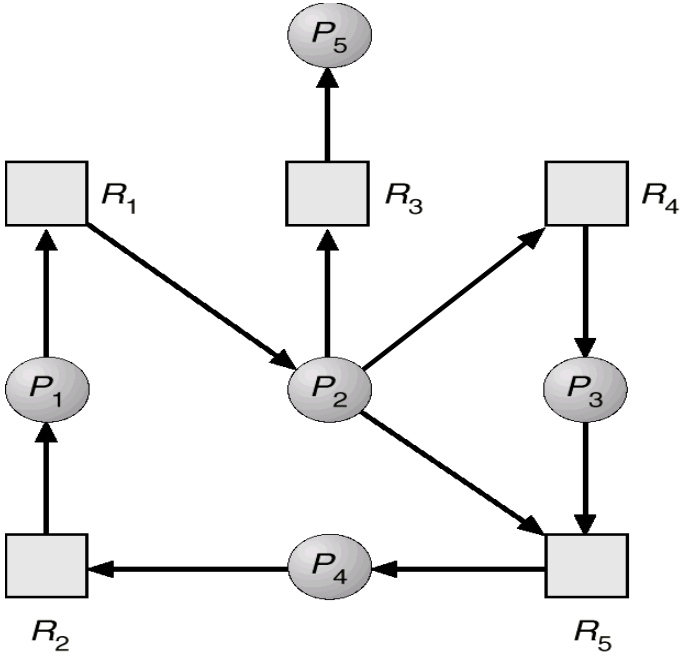
Need an algorithm that determines if deadlock occurred.

Also need a means of recovering from that deadlock.

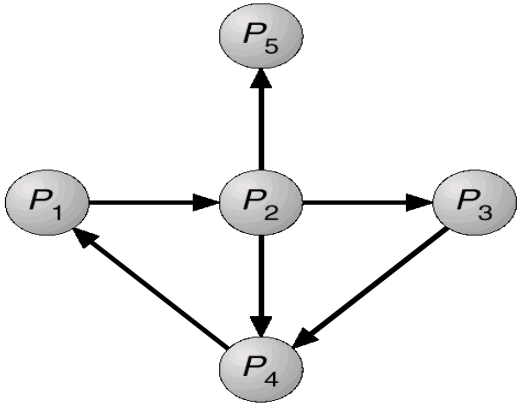
SINGLE INSTANCE OF A RESOURCE TYPE

Wait-for graph == remove the resources from the usual graph and collapse edges.

- An edge from p(i) to p(j) implies that p(i) is waiting for p(j) to release.



(a)



(b)

Deadlock Detection

SEVERAL INSTANCES OF A RESOURCE TYPE

Complexity is of order m * n * n.

We need to keep track of:

available - records how many resources of each type are available.

allocation - number of resources of type m allocated to process n.

request - number of resources of type m requested by process n.

Let **work** and **finish** be vectors of length **m** and **n** respectively.

Deadlock Detection

1. Initialize work[] = available[]
   For i = 1,2,...n, if allocation[i] != 0 then
   finish[i] = false; otherwise, finish[i] = true;

2. Find an i such that:
   finish[i] == false and request[i] <= work

   If no such i exists, go to step 4.

3. work = work + allocation[i]
   finish[i] = true
   goto step 2

4. if finish[i] == false for some i, then the system is in deadlock state.
   IF finish[i] == false, then process p[i] is deadlocked.

Deadlock Detection

We have three resources, A, B, and C. A has 7 instances, B has 2 instances, and C has 6 instances. At this time, the allocation, etc. looks like this:

Is there a sequence that will allow deadlock to be avoided?

Is there more than one sequence that will work?

		←	Alloc	→		←	Req	→		←	Avail	→
		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	0				
P3		2	1	1		1	0	0				
P4		0	0	2		0	0	2				

Deadlock Detection

EXAMPLE

Suppose the Request matrix is changed like this. In other words, the maximum amounts to be allocated are initially declared so that this request matrix results.

Is there now a sequence that will allow deadlock to be avoided?

		←	Alloc	→		←	Req	→		←	Avail	→
		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				

USAGE OF THIS DETECTION ALGORITHM

Frequency of check depends on how often a deadlock occurs and how many processes will be affected.

Deadlock Recovery

So, the deadlock has occurred. Now, how do we get the resources back and gain forward progress?

PROCESS TERMINATION:

- Could delete all the processes in the deadlock -- this is expensive.
- Delete one at a time until deadlock is broken (time consuming).
- Select who to terminate based on priority, time executed, time to completion, needs for completion, or depth of rollback
- In general, it's easier to preempt the resource, than to terminate the process.

Deadlock Recovery

RESOURCE PREEMPTION:

- Select a victim - which process and which resource to preempt.
- Rollback to previously defined "safe" state.
- Prevent one process from always being the one preempted (starvation).

Q2.

e following snapshot of a system:

	cation				*Max*				*Available*			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

Answer the following questions using the 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?*

Solution

(a) The values of *Need for processes P0 through P4 respectively are (0, 0, 0, 0), (0, 7, 5, 0),* (1,0, 0, 2), (0, 0, 2, 0), and (0, 6, 4, 2).

b. Yes. With *Available being equal to (1,5, 2, 0), either process P0 or P3 could run. Once*

process P3 runs, it releases its resources, which allow all other existing processes to run.

c. Yes, it can. This results in the value of *Available being (1, 1, 0, 0). One ordering of* processes that can finish is *P0, P2, P3, P1, and P4.*

Q3.

Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Is this system deadlock-free? Why or why not?

Solution

Yes, this system is deadlock-free.
Proof by contradiction. Suppose the system is deadlocked. This implies that each process is holding one resource and is waiting for one more. Since there are three processes and four resources, one process must be able to obtain two resources. This process requires no more resources and, therefore it will return its resources when done.

Q4.

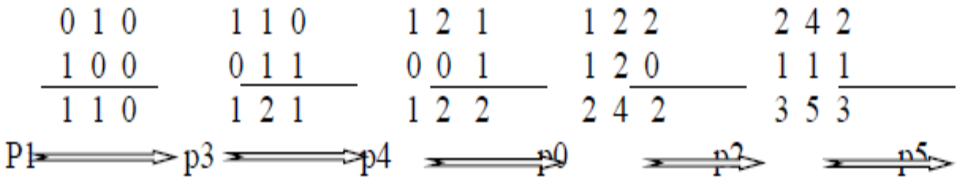
Is it possible to have a deadlock involving only a single process? Explain your answer.

- Consider we have five processes P0, P1, . . . P5 and three resources A, B, and C. Is the executing the following processes in the safe state?

Process	Allocation			Maximum need		
	A	B	C	A	B	C
P0	1	2	0	2	2	2
P1	1	0	0	1	1	0
P2	1	1	1	1	4	3
P3	0	1	1	1	1	1
P4	0	0	1	1	2	2
P5	1	0	0	1	5	1

Available		
A	B	C
0	1	0

Process	Need		
P0	1	0	2
P1	0	1	0
P2	0	3	2
P3	1	0	0
P4	1	2	1
P5	0	5	1



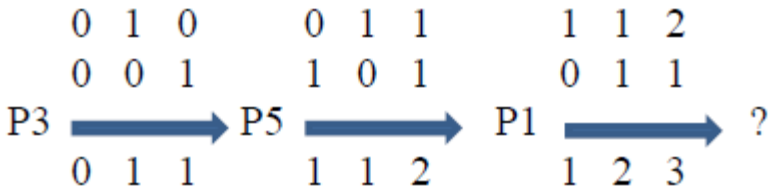
The process in safe state if they are executed in the sequence<P1, P3, P4, P0, P2, P5>

Suppose we have five processes and three resources, A, B, and C. A has 2 instances, B has 5 instances and C has 4 instances. Can the system execute the following processes without deadlock occurring, where we have the following?

Process	Maximum need			Allocation		
	A	B	C	A	B	C
P1	1	2	3	0	1	1
P2	2	2	0	0	1	0
P3	0	1	1	0	0	1
P4	3	5	3	1	2	1
P5	1	1	2	1	0	1

The available is A=0, B=1, C=0.
The current need is

Process	Current need		
	A	B	C
P1	1	1	2
P2	2	1	0
P3	0	1	0
P4	2	3	2
P5	0	1	1



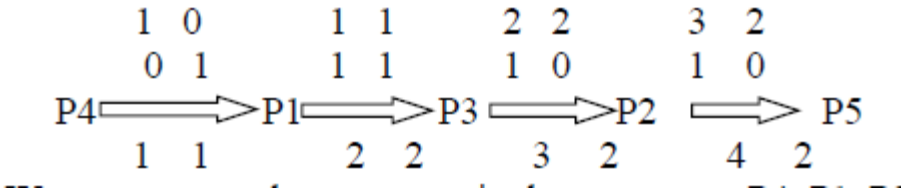
The deadlock is occurred since the available resources is less than the needs of P2 and P4

P-1

- Suppose we have two resources, A, and B. A has 6 instances and B has 3 instances. Can the system execute the following processes without deadlock occurring?

Process	Allocate		Maximum need	
	A	B	A	B
P1	1	1	2	2
P2	1	0	4	2
P3	1	0	3	2
P4	0	1	1	1
P5	2	1	6	3

Process	Need	
	A	B
P1	1	1
P2	3	2
P3	2	2
P4	1	0
P5	4	2



P-2

... means only one process at a time can use a resource

(a) Hold and wait

(b) Mutual Exclusion

(c) No preemption

(d) Circular wait

P-3

... means process acquiring at least one resource is waiting to acquire additional resources held by other processes

(a) Hold and wait

(b) Mutual Exclusion

(c) No preemption

(d) Circular wait

P-4

- ... means a resource can be released only voluntarily by the process holding it, after that process has completed its task
- (a) Hold and wait
- (b) Mutual Exclusion
- (c) No preemption
- (d) Circular wait

P-5

- What is a reusable resource?
  a) that can be used by one process at a time and is not depleted by that use
  b) that can be used by more than one process at a time
  c) that can be shared between various threads
  d) none of the mentioned
- Ans: a

```
P1                              P2
...                             ...
receivefrom(P2, &M2);           receivefrom(P1, &M1);
...                             ...
sendto(P2, M1);                 sendto(P1, M2);
```

P-6

- The number of resources requested by a process _____

  a) must always be less than the total number of resources available in the system
  b) must always be equal to the total number of resources available in the system
  c) must not exceed the total number of resources available in the system
  d) must exceed the total number of resources available in the system
- Ans: C

P-7

- Deadlock prevention is a set of methods

  _____

  a) to ensure that at least one of the necessary conditions cannot hold
  b) to ensure that all of the necessary conditions do not hold
  c) to decide if the requested resources for a process have to be given or not
  d) to recover from a deadlock
- Ans: a

P-8

- For non sharable resources like a printer, mutual exclusion _____
  a) must exist
  b) must not exist
  c) may exist
  d) none of the mentioned

- Answer: a
  Explanation: A printer cannot be simultaneously shared by several processes.

P-9

- Given a priori information about the _____ number of resources of each type that maybe requested for each process, it is possible to construct an algorithm that ensures that the system will never enter a deadlock state.
  a) minimum
  b) average
  c) maximum
  d) approximate

- Ans: c

P-10

- All unsafe states are _____
  a) deadlocks
  b) not deadlocks
  c) fatal
  d) none of the mentioned

- Ans: b

P-11

- The resource allocation graph is not applicable to a resource allocation system _____
  a) with multiple instances of each resource type
  b) with a single instance of each resource type
  c) single & multiple instances of each resource type
  d) none of the mentioned
- Ans: a

P-12

- The content of the matrix Need is _____
  a) Allocation – Available
  b) Max – Available
  c) Max – Allocation
  d) Allocation – Max

- Ans: c