Chapter 5
Deadlocks

5.1 Definition

In a multiprogramming system, processes request resources. If those resources are being used by other processes then the process enters a waiting state. However, if other processes are also in a waiting state, we have deadlock.

The formal definition of deadlock is as follows:

Definition: A set of processes is in a deadlock state if every process in the set is waiting for an event (release) that can only be caused by some other process in the same set.

Example 5.1

Process-1 requests the printer, gets it
Process-2 requests the tape unit, gets it
Process-1 requests the tape unit, waits
Process-2 requests the printer, waits

} Process-1 and Process-2 are deadlocked!

In this chapter, we shall analyze deadlocks with the following assumptions:

- A process must request a resource before using it. It must release the resource after using it. (request → use → release)

- A process cannot request a number more than the total number of resources available in the system.

For the resources of the system, a resource table shall be kept, which shows whether each process is free or if occupied, by which process it is occupied. For every resource, queues shall be kept, indicating the names of processes waiting for that resource.

A deadlock occurs if and only if the following four conditions hold in a system simultaneously:

1. Mutual Exclusion: At least one of the resources is non-sharable (that is; only a limited number of processes can use it at a time and if it is requested by a process while it is being used by another one, the requesting process has to wait until the resource is released.).

2. Hold and Wait: There must be at least one process that is holding at least one resource and waiting for other resources that are being hold by other processes.

3. No Preemption: No resource can be preempted before the holding process completes its task with that resource.

4. Circular Wait: There exists a set of processes: {P_1, P_2, ..., Pn} such that

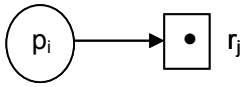
   P_1 is waiting for a resource held by P_2
   P_2 is waiting for a resource held by P_3
   ...
   P_{n-1} is waiting for a resource held by P_n
   P_n is waiting for a resource held by P_1

Methods for handling deadlocks are:

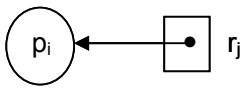
- Deadlock prevention
- Deadlock avoidance
- Deadlock detection and recovery.

5.2 Resource Allocation Graphs

Resource allocation graphs are drawn in order to see the allocation relations of processes and resources easily. In these graphs, processes are represented by circles and resources are represented by boxes. Resource boxes have some number of dots inside indicating available number of that resource, that is number of instances.



Process p_i is waiting for resource r_j



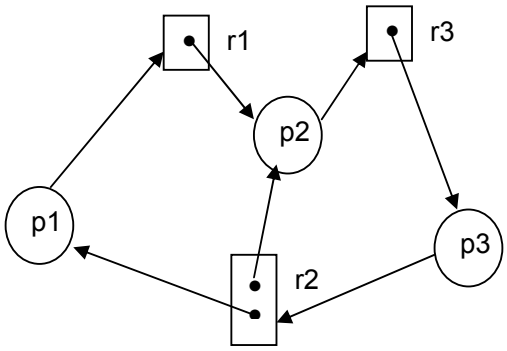
Process p_i has allocated resource r_j

- If the resource allocation graph contains no cycles then there is no deadlock in the system at that instance.

- If the resource allocation graph contains a cycle then a deadlock may exist.

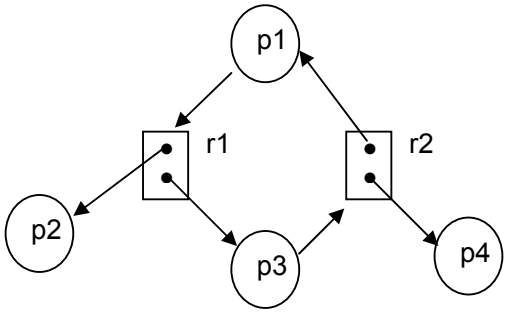
- If there is a cycle, and the cycle involves only resources which have a single instance, then a deadlock has occurred.

Example 5.2



There are three cycles, so a deadlock may exists. Actually p1, p2 and p3 are deadlocked

Example 5.3



There is a cycle, however there is no deadlock. If p4 releases r2, r2 may be allocated to p3, which breaks the cycle.

5.3 Deadlock Prevention

To prevent the system from deadlocks, one of the four discussed conditions that may create a deadlock should be discarded. The methods for those conditions are as follows:

Mutual Exclusion:

In general, we do not have systems with all resources being sharable. Some resources like printers, processing units are non-sharable. So it is not possible to prevent deadlocks by denying mutual exclusion.

Hold and Wait:

One protocol to ensure that hold-and-wait condition never occurs says each process must request and get all of its resources before it begins execution.

Another protocol is “Each process can request resources only when it does not occupies any resources.”

The second protocol is better. However, both protocols cause low resource utilization and starvation. Many resources are allocated but most of them are unused for a long period of time. A process that requests several commonly used resources causes many others to wait indefinitely.

No Preemption:

One protocol is "If a process that is holding some resources requests another resource and that resource cannot be allocated to it, then it must release all resources that are currently allocated to it."

Another protocol is "When a process requests some resources, if they are available, allocate them. If a resource it requested is not available, then we check whether it is being used or it is allocated to some other process waiting for other resources. If that resource is not being used, then the OS preempts it from the waiting process and allocate it to the requesting process. If that resource is used, the requesting process must wait." This protocol can be applied to resources whose states can easily be saved and restored (registers, memory space). It cannot be applied to resources like printers.

Circular Wait:

One protocol to ensure that the circular wait condition never holds is "Impose a linear ordering of all resource types." Then, each process can only request resources in an increasing order of priority.

For example, set priorities for r1 = 1, r2 = 2, r3 = 3, and r4 = 4. With these priorities, if process P wants to use r1 and r3, it should first request r1, then r3.

Another protocol is "Whenever a process requests a resource r_j, it must have released all resources r_k with priority(r_k) ≥ priority (r_j).

5.4 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.

When a process requests a resource that is already available, the system must decide whether that resource can immediately be allocated or not. The resource is immediately allocated only if it leaves the system in a *safe state*.

A state is safe if the system can allocate resources to each process in some order avoiding a deadlock. A deadlock state is an unsafe state.

Example 5.4

Consider a system with 12 tape drives. Assume there are three processes : p1, p2, p3.
Assume we know the maximum number of tape drives that each process may request:

p1 : 10, p2 : 4, p3 : 9

Suppose at time t_{now}, 9 tape drives are allocated as follows :

p1 : 5, p2 : 2, p3 : 2

So, we have three more tape drives which are free.

This system is in a safe state because it we sequence processes as: <p2, p1, p3>, then p2 can get two more tape drives and it finishes its job, and returns four tape drives to the system. Then the system will have 5 free tape drives. Allocate all of them to p1, it gets 10 tape drives and finishes its job. p1 then returns all 10 drives to the system. Then p3 can get 7 more tape drives and it does its job.

It is possible to go from a safe state to an unsafe state:

Example 5.5

Consider the above example. At time t_{now+1}, p3 requests one more tape drive and gets it. Now, the system is in an unsafe state.

There are two free tape drives, so only p2 can be allocated all its tape drives. When it finishes and returns all 4 tape drives, the system will have four free tape drives.

p1 is allocated 5, may request 5 more → has to wait
p3 is allocated 3, may request 6 more → has to wait

We allocated p3 one more tape drive and this caused a deadlock.

Banker's Algorithm (Dijkstra and Habermann)

It is a deadlock avoidance algorithm. The following data structures are used in the algorithm:

m = number of resources
n = number of processes

Available [m] : One dimensional array of size m. It indicates the number of available resources of each type. For example, if Available [i] is k, there are k instances of resource r_i.

Max [n,m] : Two dimensional array of size n*m. It defines the maximum demand of each process from each resource type. For example, if Max [i,j] is k, process p_i may request at most k instances of resource type r_j.

Allocation [n,m] : Two dimensional array of size n*m. It defines the number of resources of each type currently allocated to each process.

Need [n,m] : Two dimensional array of size n*m. It indicates the remaining need of each process, of each resource type. If Need [i,j] is k, process pi may need k more instances of resource type r_j. Note that Need [i,j] = Max [i,j] - Allocation [i,j].

Request [n,m] : Two dimensional array of size n*m. It indicates the pending requests of each process, of each resource type.

Now, take each row vector in Allocation and Need as Allocation(i) and Need(i). (Allocation(i) specifies the resources currently allocated to process p_i.)

Define the ≤ relation between two vectors X and Y , of equal size = n as :

$X \leq Y \quad \Leftrightarrow \quad X[i] \leq Y[i]$, i = 1,2, ..., n
$X \not\leq Y \quad \Leftrightarrow \quad X[i] > Y[i]$ for some i

The algorithm is as follows:

1. Process pi makes requests for resources. Let Request(i) be the corresponding request vector. So, if p_i wants k instances of resource type r_j, then Request(i)[j] = k.

2. If Request(i) $\not\leq$ Need(i), there is an error.

3. Otherwise, if Request(i) $\not\leq$ Available, then p_i must wait.

4. Otherwise, Modify the data structures as follows :

Available = Available - Request(i)
Allocation(i) = Allocation(i) + Request(i)
Need(i) = Need(i) - Request(i)

5. Check whether the resulting state is safe. (Use the safety algorithm presented below.)

6. If the state is safe, do the allocation. Otherwise, p_i must wait for Request(i).

Safety Algorithm to perform Step 5:

Let Work and Finish be vectors of length m and n, respectively.

1. Initialize Work = Available, Finish [j] = false, for all j.

2. Find an i such that Finish [i] = false and Need(i) ≤ Work

If no such i is found, go to step **4.**

3. If an i is found, then for that i, do :

Work = Work + Allocation(i)
Finish [i] = true

Go to step **2.**

4. If Finish [j] = true for all j, then the system is in a safe state.

Banker's algorithm is $O(m \times (n^2))$.

Example 5.6*:* (Banker's algorithm)

Given

Available = $\begin{bmatrix}1 & 4 & 1\end{bmatrix}$

Max = $\begin{bmatrix}1 & 3 & 1\\ 1 & 4 & 1\end{bmatrix}$

Allocation= $\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}$

Need = $\begin{bmatrix}1 & 3 & 1\\ 1 & 4 & 1\end{bmatrix}$

Request = $\begin{bmatrix}1 & 2 & 0\\ 0 & 2 & 1\end{bmatrix}$

Request(1) is to be processed. If it is satisfied data would become:

Available = $\begin{bmatrix}0 & 2 & 1\end{bmatrix}$

Allocation= $\begin{bmatrix}1 & 2 & 0\\ 0 & 0 & 0\end{bmatrix}$

Need = $\begin{bmatrix}0 & 1 & 1\\ 1 & 4 & 1\end{bmatrix}$

Now, apply the safety algorithm:

Work = [0 2 1]

Finish = $\begin{bmatrix}false\\ false\end{bmatrix}$

i = 1 :

Need(1) = [0 1 1] $\leq$ Work ? Yes.
Work = Work + Allocation(1) = [1 4 1]
Finish (1) = true

i = 2 :

Need(2) = [1 4 1] $\leq$ Work ? Yes.
Work = Work + Allocation(2) = [1 4 1]
Finish (2)= true

System is in a safe state, so do the allocation. If the algorithm is repeated for Request(2), the system will end up in an unsafe state.

5.5 Deadlock Detection

If a system has no deadlock prevention and no deadlock avoidance scheme, then it needs a deadlock detection scheme with recovery from deadlock capability. For this, information should be kept on the allocation of resources to processes, and on outstanding allocation requests. Then, an algorithm is needed which will determine whether the system has entered a deadlock state. This algorithm must be invoked periodically.

Deadlock Detection Algorithm (Shoshani and Coffman)

Data Structure is as:

Available [m]
Allocation [n,m] as in Banker's Algorithm
Request [n,m] indicates the current requests of each process.

Let Work and Finish be vectors of length m and n, as in the safety algorithm.

The algorithm is as follows:

1. Initialize Work = Available
For i = 1 to n do
If Allocation(i) = 0 then Finish[i] = true else Finish[i] = false

2. Search an i such that

Finish[i] = false and Request(i) ≤ Work

If no such i can be found, go to step 4.

3. For that i found in step 2 do:

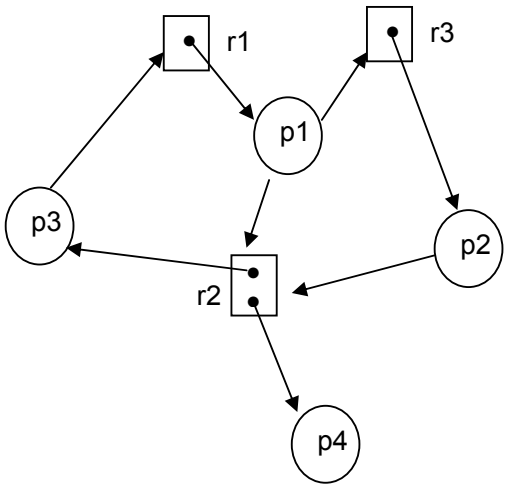
Work = Work + Allocation(i)
Finish[i] = true

Go to step 2.

4. If Finish[i] ≠ true for a some i then the system is in deadlock state else the system is safe

Example 5.7

Examine whether the system whose resource allocation graph is given below is deadlocked or not.



First, let's form the required structures:

Available = [0 0 0]

$$\text{Allocation} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{Request} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\text{Finish} = \begin{bmatrix} False \\ False \\ False \\ False \end{bmatrix}$$

Work = [0 0 0]

Request(4) ≤ Work → i = 4:

Work = Work + Allocation(4) = [0 0 0] + [0 0 1] = [0 0 1] ;
Finish[4] = True

Request(2) ≤ Work → i = 2:

Work = Work + Allocation(2) = [0 0 1] + [0 1 0] = [0 1 1] ;
Finish[2] = True

Request(1) ≤ Work→ i = 1:

Work = Work + Allocation(1) = [0 1 1] + [1 0 0] = [1 1 1] ;
Finish[1] = True

Request(3) ≤ Work → i = 3:

Work = Work + Allocation(3) = [1 1 1] + [0 0 1] = [1 1 2] ;
Finish[3] = True

Since Finish[i] = true for all i, there is no deadlock in the system .

Recovery From Deadlock

If the system is in a deadlock state, some methods for recovering it from the deadlock state must be applied. There are various ways for recovery:

- Allocate one resource to several processes, by violating mutual exclusion.
- Preempt some resources from some of the deadlocked processes.
- Abort one or more processes in order to break the deadlock.

If preemption is used:

1. Select a victim. (Which resource(s) is/are to be preempted from which process?)

2. Rollback: If we preempt a resource from a process, roll the process back to some safe state and mak it continue.

Here the OS may be probably encounter the problem of starvation. How can we guarantee that resources will not always be preempted from the same process?

In selecting a victim, important parameters are:

- Process priorities
- How long the process has occupied?
- How long will it occupy to finish its job
- How many resources of what type did the process use?
- How many more resources does the process need to finish its job?
- How many processes will be rolled back? (More than one victim may be selected.)

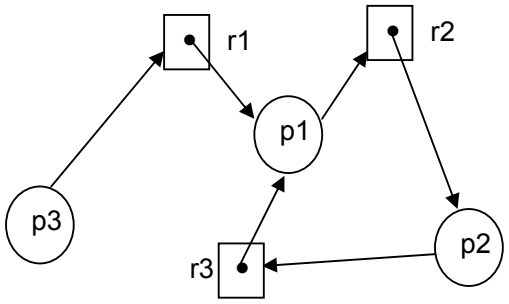
For rollback, the simplest solution is a total rollback. A better solution is to roll the victim process back only as far as it's necessary to break the deadlock. However, the OS needs to keep more information about process states to use the second solution.

To avoid starvation, ensure that a process can be picked as a victim for only a small number of times. So, it is a wise idea to include the number of rollbacks as a parameter.

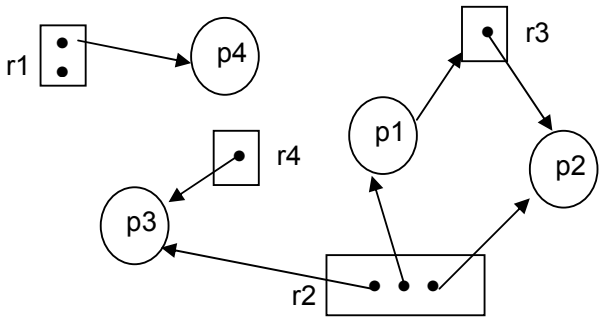
QUESTIONS

1. For each of the following resource allocation graphs, find out and explain whether there is a deadlock or not

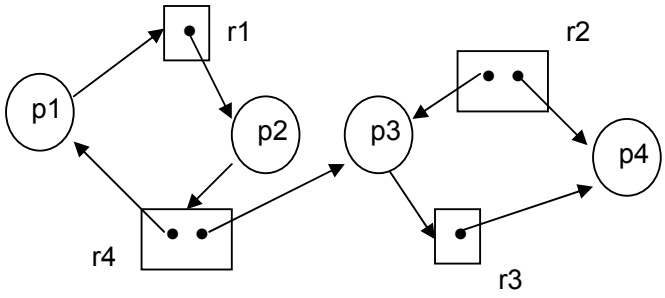
a.



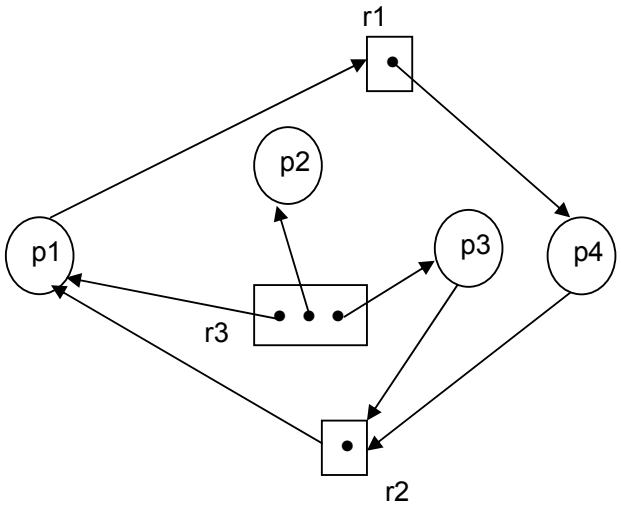
b.



c.



d.



2. A computer system has m resources of the same type and n processes share these resources. Prove or disprove the following statement for the system:

This system is deadlock free if sum of all maximum needs of processes is less than m+n.

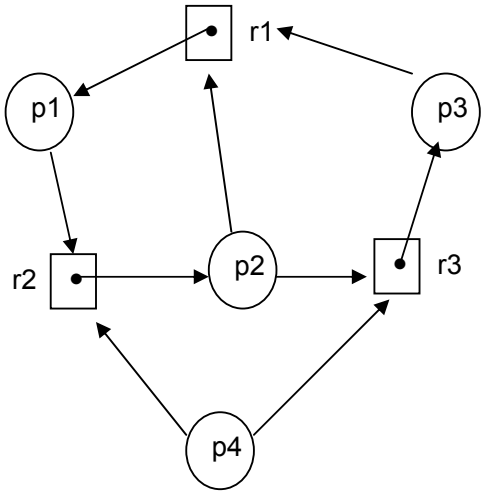
3. There are four processes which are going to share nine tape drives. Their current and maximum number of allocation numbers are as follows :

process	current	maximum
p1	3	6
p2	1	2
p3	4	9
p4	0	2

a. Is the system in a safe state? Why or why not?

b. Is the system deadlocked? Why or why not?

4. Given the following resource allocation diagram,



a. If another instance of resource r1 is made available, is the deadlock resolved ? If yes specify the allocation sequence, if no explain why?

b.& c. repeat part **a.** for resource r2 and r3.

5. Given that all the resources are identical, they can be acquired and released strictly one at a time, and no process ever needs more than the total resources on the system, state whether deadlock can occur in each of the following systems. Explain why or how.

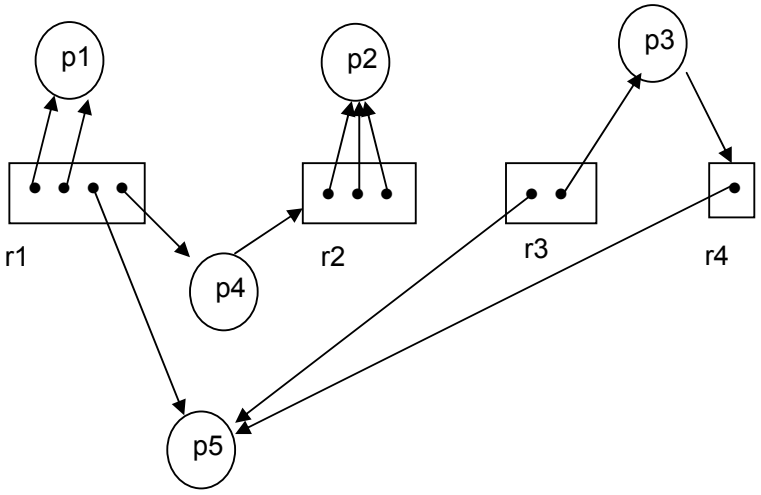
	Number of processes	Number of resources
a.	1	1
b.	1	2
c.	2	1
d.	2	2
e.	2	3

6. a. What are the four conditions necessay for deadlock to appear?

b. Cinderella and the Prince are getting divorced. To divide their property, they have agreed on the following algorithm. Every morning, each of one may send a letter to the other's lawyer requesting one item of property. Since it takes a day for letters to be delivered, they have agreed that if both discover that they have requested the same item on the same day, the next day they will send a letter cancelling the request. Among their property is the glass shoe, their dog Woofer, Woofer's doghouse, their canary Tweeter, Tweeter's cage and a sword. The animals love their houses, so it has been agreed that any division of property separating an animal from its house is invalid, requiring the lawyers to negotiate on which items they already have should be returned back. Unfortunately the lawyers are stubborn and never agree. Is deadlock or starvation possible in such a scheme? Explain.

c. What happens if it has been agreed that in the case of any division of property separating an animal from its house the whole division to start over from scratch, instead of letting the lawyers to discuss. Explain if starvation or deadlock possible now.

7. Given the following resource allocation diagram:



a. Apply the deadlock detection algorithm and either indicate why the system is deadlocked, or specify a safe allocation sequence.

b. If the process P2 also request 2 instances of resource r1, does the system enter a deadlock? Why?

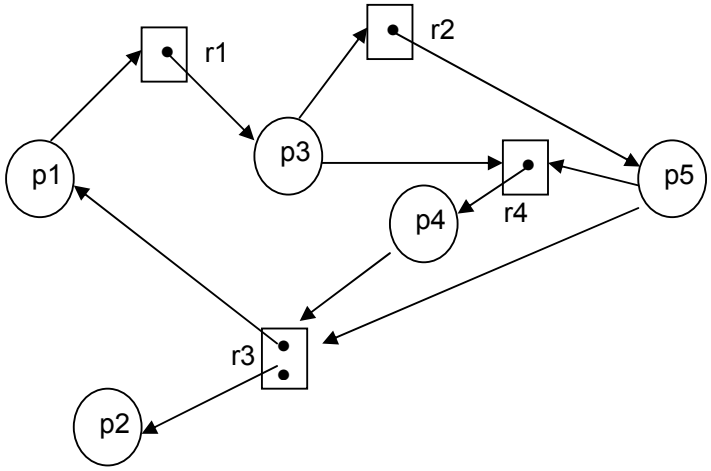
c. If a deadlock occurs in part **a.** and/or **b.**, killing which process would resolve the deadlock?

d. If the maximum declared needs are:

process	r1	r2	r3	r4
P1	4	0	0	0
P2	1	3	1	1
P3	0	0	2	1
P4	1	1	1	0
P5	1	0	1	1

does the current allocation given in part a constitute a safe state? Why?

8. Explain if the system in the figure is dedlocked. If not, give an execution order of the processess which successfully terminates.



9. a. Explain if the following system is deadlocked or not?

b. For the following resource allocation graph, for deadlock detection show the current contents of the AVAILABLE, ALLOCATION, REQUEST .