CHAPTER 3: DEADLOCKS

3.1 SYSTEM MODEL

- In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources: if the resources are not available at that time, the process enters a wait state. It may happen that waiting processes will never again change state, because the resources they have requested are held by other waiting processes. This situation is called deadlock.
- If a process requests an instance of a resource type, the allocation of any instance of the type will satisfy the request. If it will not, then the instances are not identical, and the resource type classes have not been defined properly. A process must request a resource before using it and must release the resource after using it. A process may request as many resources as it requires to carry out its designated task.
- Under the normal mode of operation, a process may utilize a resource in only the following sequence:
  1. **Request:** If the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource.
  2. **Use:** The process can operate on the resource.
  3. **Release:** The process releases the resource.

DEADLOCK BASICS

Today most operating systems are multi-programming OS. So, more than one process may be running simultaneously. In such environment, several processes compete for finite number of resources, such as printers, scanners, tam drives, files. Slots in the system's internal tables.

Whenever a process needs any resource. it requests for it. OS grants resource if it is free. In such case, process will progress by using it. But. if resource is not free, process will wait for it to be free.

Now consider the following situations:

1. suppose there are three processes in a system. Also, there are three tap drives in a system. Each process needs two tap drives to complete its execution. Suppose each of them has acquired one tap drive, and now requests for second. But. none of tap drives is free. So, all processes will wait. But, for how long do they need to wait? Almost. forever.

2. Suppose there are two processes. process A and B, in a system. Also, there is one printer and one scanner in a system. Each process must scan an image. and then take print-out of it. So, process-A requests for permission to use the scanner and is granted it. Process-B requests printer first and is also granted it. Now, suppose process-A requests for printer. But this request will be denied until process-B releases it. Unfortunately, instead of releasing printer. B asks for scanner. So, both will wait. For how long? Almost, forever.

3. Consider a database application which contains two tables. T1 and T2. Also, there are two processes, A and B. Suppose process-A locks table T1 and B locks T2. And then each process tries to lock other table. What are consequences? Simply waiting for lock on another table to be released. Almost, forever. Here, all processes have some resource, and they need other resource to complete their execution. But such other resources are already acquired by some different processes.

So, none of them can have further progress. Such situation is called **'Deadlock'**.

Following sub-sections describe detailed description about deadlocks.

RESOURCES

Resource can be defined as:

"Resource is anything that can be used by only a single process at any instance of time."

Resources can be physical resources such as CPU, memory, printer, scanner, tap drive etc. or they can also be logical resources such as files, tables, variables. Deadlocks can occur when processes have been granted exclusive access to such resources.

PREEMPTABLE VERSUS NON-PREEMPTABLE RESOURCES:

Resources can be categorized in one of two categories:

i) Preemptable.
ii) Non-preemptable.

Preemptable Resource:

- Resource that can be taken away from the process currently using it, with no ill effects.
- Here, process doesn't get affected when resource is taken away from it during its execution. (Some care is required.)
- Such resources don't cause deadlocks.
- Example: Memory, CPU.

Non-preemptable Resources:

- Resource that cannot be taken away from the process currently using it, without causing execution to fail.
- Here, process gets affected when resource is taken away from it during its execution. Generally, process execution fails.
- Such resources may cause deadlocks. **Example**: Printer, Scanner.

Events Required to use a Resource:

The sequence of events required to use a resource is given below:

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

- When any process needs some resource, it makes request to Operating System for it.
- If resource is free, Operating System grants that resource to process. Such can continue its execution by using resource. When use of resource completes, releases it.
- But, if resource is not free on request, process simply needs to wait until resource becomes free. Such process goes to 'Waiting' state. It waits there till the becomes free. Once resource becomes available. process is transferred from state to 'Ready' state. After this, it can start execution once it gets scheduled.

3.2 DEADLOCK CHARACTERISTICS

In deadlock, processes never finish executing and system resources are tied up, preventing other jobs from ever starting.

3.1 NECESSARY CONDITIONS

A deadlock situation can arise if the following four conditions hold simultaneously in a system:

1. **Mutual exclusion:** At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.
2. **Hold and wait:** There must exist a process that is holding at least one resource and is waiting to acquire additional resources that are currently being held by other processes.
3. **No preemption:** Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process, has completed its task.
4. **Circular wait:** There must exist a set {P0, P1, ..., P_n} of waiting processes such that P0 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 P_n, and Pn is waiting for a resource that is held by P0.

3.2 RESOURCE-ALLOCATION GRAPH

- Deadlocks can be described more precisely in terms of a directed graph called a system resource-allocation graph.
- The set of vertices V is partitioned into two different types of nodes.
- P = {P1, P2, … P_n} the set consisting of all the active processes in the system; and

- R = {R1, R2, …, R_n}, the set consisting of all resource types in the system.
- A directed edge from process Pi to resource type Rj is denoted by Pi → Rj, it signifies that process Pi requested an instance of resource type Rj and is currently waiting for that resource.
- A directed edge from resource type Rj to process Pi is denoted by Rj_ Pi it signifies that an instance of resource type Rj has been allocated to process Pi.
- A directed edge Pi → Rj is called a ***Request edge***;
- a directed edge Rj →Pi is called an ***Assignment edge***.
- When process Pi requests an instance of resource type Rj, a request edge is inserted in the resource-allocation graph.
- When this request can be fulfilled, the request edge is instantaneously transformed to an assignment edge. When the process no longer needs access to the resource it releases the resource, and as a result the assignment edge is deleted.
- Definition of a resource-allocation graph, it can be shown that, if the graph contains no cycles, then no process in the system is deadlocked. If, on the other hand, the graph contains the cycle, then a deadlock must exist.
- If each resource type has several instances, then a cycle implies that a deadlock has occurred.
- If the cycle involves only a set of resource types, each of which has only a single instance, then a deadlock has occurred.
- Each process involved in the cycle is deadlocked. In this case, a cycle in the graph is both a necessary and a sufficient condition for the existence of deadlock.

A set of vertices V and a set of edges E.

- V is partitioned into two types:
  - P = {P1, P2, …, P_n}, the set consisting of all the processes in the system.
  - R = {R1, R2, …, R_m}, the set consisting of all resource types in the system
- request edge – directed edge P1 → Rj
- assignment edge – directed edge Rj →Pi

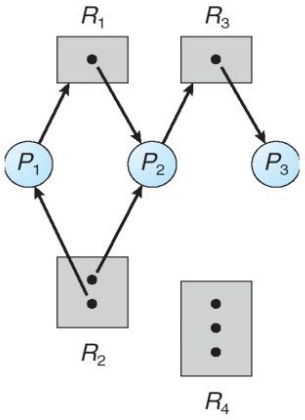


FIGURE: RESOURCE ALLOCATION GRAPH

- If each resource type has several instances, then a cycle does not necessarily imply that a deadlock incurred. In this case, a cycle in the graph is a necessary but not a sufficient condition for the existence of deadlock.
- Suppose that process P3requests an instance of resource type R2 Since no resource instance is currently available, a request edge P3 → R2 is added to the graph. At this point, two minimal cycles exist in the system:

- P1 → R1 → P2 → R3 → P3 → R2 → P1
- P2 → R3 → P3 → R2 → P2

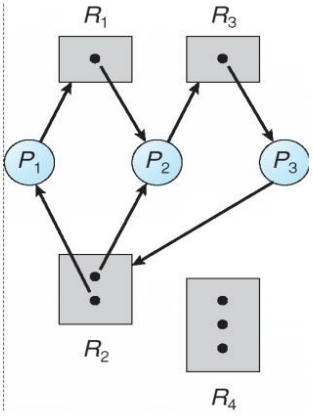
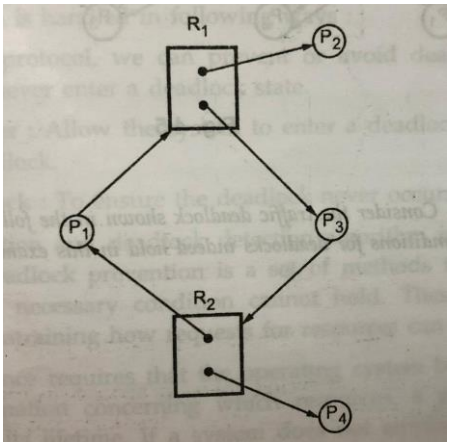


Fig. Resource Allocation Graph with Deadlock

- Processes P1, P2, and P3 are deadlocked. Process P2 is waiting for the resource R3, which is held by process P3. Process P3, on the other hand, is waiting for either process P1 or process P2 to release resource R2. In addition, process PI is waiting for process P2 to release resource R1.
- Below is the resource allocation graph with a cycle but no deadlock.



Let us consider,

P1 -> R1 -> P3 -> R2 -> P1

There is a cycle but no deadlock. Because the process may release its instance of resource type R2. That resource can then be allocated to breaking the cycle.

3.3 METHODS OF HANDLING DEADLOCK

Deadlock problem is handled in following ways:

1. **Protocol:** Using protocol, we can prevent or avoid deadlocks. To take care that, system will never enter a deadlock state.
2. **Detect and recover:** Allow the system to enter a deadlock state, detect it and recover from deadlock.
3. **Ignore the deadlock:** To ensure the deadlock never occurs in the system.

Deadlock prevention and deadlock detection algorithm is used for ignoring the deadlock. Deadlock prevention is a set of methods for ensuring that at least one of the necessary conditions cannot hold. These methods prevent deadlocks by constraining how requests for resources can be made.

Deadlock avoidance requires that the operating system be given in advance, will request additional information concerning which resources, a process and use during its lifetime. If a system does not employ either a deadlock prevention or a deadlock avoidance algorithm, then a deadlock situation may occur. If a system does not ensure that a deadlock will never occur and also does not provide a mechanism for deadlock detection and recovery, then the system is in a deadlock state.

3.4 DEADLOCK PREVENTION

For a deadlock to occur, each of the four necessary conditions must hold. By ensuring that at least on one these conditions cannot hold, we can prevent the occurrence of a deadlock.

1. **Mutual Exclusion**
   The mutual-exclusion condition must hold for non-sharable resources. For example, a printer cannot be simultaneously shared by several processes.

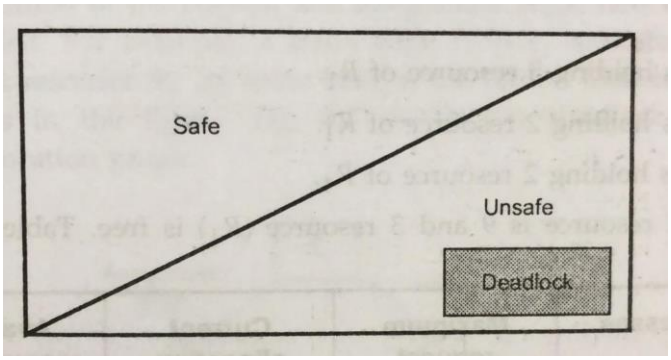
Sharable resources, on the other hand, do not require mutually exclusive access, and thus cannot be involved in a deadlock.

2. **Hold and Wait**
   a. Whenever a process requests a resource, it does not hold any other resources. One protocol that be used requires each process to request and be allocated all its resources before it begins execution.
   b. An alternative protocol allows a process to request resources only when the process has none. A process may request some resources and use them. Before it can request afny additional resources, however it must release all the resources that it is currently allocated here are two main disadvantages to these protocols. First, resource utilization may be low, since many of the resources may be allocated but unused for a long period. In the example given, for instance, we can release the tape drive and disk file, and then again request the disk file and printer, only if we can be sure that our data will remain on the disk file. If we cannot be assured that they will, then we must request all resources at the beginning for both protocols. Second, starvation is possible.
3. **No Preemption**
   If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are preempted. That is these resources are implicitly released. The preempted resources are added to the list of resources for which the process is waiting process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.
4. **Circular Wait**
   Circular-wait condition never holds is to impose a total ordering of all resource types, and to require that each process requests resources in an increasing order of enumeration.

   Let R = {R1, R2, ..., Rn} be the set of resource types. We assign to each resource type a unique integer number, which allows us to compare two resources and to determine whether one precedes another in our ordering. Formally, we define a one-to-one function F: R _ N, where N is the set of natural numbers

3.5 DEADLOCK AVOIDANCE

• Deadlock avoidance allows the three necessary conditions but makes judicious choices to assure that the deadlock point is never reached. Deadlock avoidance therefore allows more concurrency than prevention does. Deadlock avoidance requires additional information about how resources are to be requested. With deadlock avoidance, a decision is made dynamically whether the current resource allocation request could, if granted, potentially lead to a deadlock.



• Two approaches are used to avoid the deadlock.

1) Do not start a process if its demands for the resources might lead to deadlock.
2) Do not grant resource request to a process if this allocation might lead to deadlock.

Above figure shows the relationship between safe, unsafe state and a deadlock state.

- A state is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock. More formally, a system is in a safe state only if there exists a safe sequence.
- A sequence of processes is a safe sequence for the current allocation state if, for each Pi the resources that Pj can still request can be satisfied by the currently available resources plus the resources held by all the Pj, with j < i. In this situation, if the resources that process Pi needs are not immediately available, then Pi can wait until all Pj have finished. When they have finished, Pi can obtain all of its needed resources, complete its designated task return its allocated resources, and terminate. When Pi terminates, Pi + 1 can obtain its needed resources, and so on.
- If no such sequence exists, then the system state is said to be unsafe.

Resource-Allocation Graph Algorithm

In resource-allocation graph, normally request edge and assignment edge is used. In addition to the request and assignment edge, new edge called claim edge is added. For example, a claim edge P_i -> R_j indicates that process P_i may request resource R_j at some time in the future. This claim is shown by dotted lines in the figure below. Figure shows this deadlock avoidance with resource allocation graph.

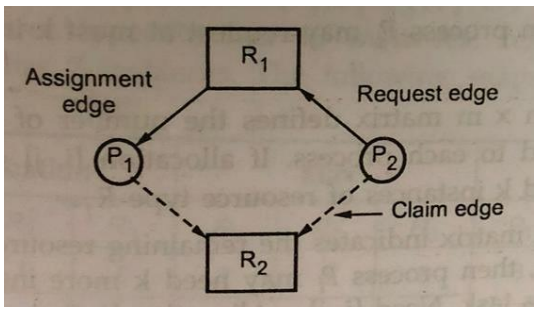


Figure Resource allocation graph for deadlock avoidance

When process P_i request resource R_j, the claim edge is converted to a request edge.

When a resource R_j is released by P_i, the assignment edge R_i -> P_i is reconverted to a claim edge P_i -> R_j.

Cycle detection algorithm is used for detecting cycle in the graph. An algorithm for detecting a cycle in this graph requires an order of n^2 operations, where n is the number of processes in the system. If no cycle exists, then the allocation of the resource will leave the system in a safe state.

If a cycle is found, then the allocation puts the system in an unsafe state.

Banker's Algorithm

The Banker's algorithm is the best known of the DEADLOCK AVOIDANCE strategies / algorithm. The strategy is modelled after the leading policies employed in banking system.

Banker's algorithm is named so because it is used in banking system to check whether loan can be sanctioned to a person or not. Suppose there are n number of account holders in a bank and the total sum of their money is S. If a person applies for a loan then the bank first subtracts the loan amount from the total money that bank has and if the remaining amount is greater than S then only the loan is sanctioned. It is done because if all the account holders comes to withdraw their money then the bank can easily do it.

In other words, the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers. The bank would try to be in safe state always.

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 Rj

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 Pi may request at most 'k' instances of resource type Rj.

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 Pi is currently allocated 'k' instances of resource type Rj

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 Pi currently need 'k' instances of resource type Rj
- **Need [i, j] = Max [i, j] – Allocation [i, j]**

1 Safety Algorithm

Safety algorithm is used to find the state of the system: That is, system may be in safe state or unsafe state. Method for this is as follows:

1) Let work and finish be vector of length m and n respectively.

Initialise work = Available and Finish [i] = False for i = 1, 2, 3, 4, ... n.

2) Find an i such that both

a. Finish [i] = False

b. Need [i] work

If no such i exist, go to step 4.

3) Work : = Work + Allocation i

Finish [i] = true

go to step 2

4) If Finish [il = true for all i, then the system is in a safe state.

2 Resource Request Algorithm

Let $Request_i$ be the request vector for process P_i. If $Request_i$ [j] = k, then 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.

l) If $Request_i$ <= $Need_i$, then go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.

2) If $Request_i$ <= Available, then go to step 3. Otherwise, Pi must wait, since the

resources are not available.

3) Available : = Available — $Request_i$;

$Allocation_i$: = $Allocation_i$ + $Request_i$;

$Need_i$: = $Need_i$ — $Request_i$;

If the resulting resource-allocation state is safe, the transaction is completed and process Pi is allocated its resources. If the new state is unsafe, then Pi must wait for the Requesti and the old resource-allocation state is restored.

Example: Considering a system with five processes P1 through P5 and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instance and type C has 7 instances.

Calculate Available resources and Need Matrix.

Process	Allocation			Max Need		
	A	B	C	A	B	C
P1	0	1	0	7	5	3
P2	2	0	0	3	2	2
P3	3	0	2	9	0	2
P4	2	1	1	2	2	2
P5	0	0	2	5	3	3
Total Allocated	**7**	**2**	**5**			

Q- 1 Calculate Available Resource:

Available Resource = Total Resource – Total Allocated

Resource	Available Resource = Total Resource – Total Allocated
A	10-7 =3
B	5-2 = 3
C	7-5 =2

Process	**Allocation**			**Max Resource**			**Available**		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P1	0	1	0	7	5	3	3	3	2
P2	2	0	0	3	2	2			
P3	3	0	2	9	0	2			
P4	2	1	1	2	2	2			
P5	0	0	2	5	3	3			
Total Allocated	**7**	**2**	**5**						

Q-2. What will be the content of the Need matrix?

Need [i, j] = Max [i, j] – Allocation [i, j]

So, the content of Need Matrix is:

Process	**Allocation**			**Max Resource**			**Need**		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P1	0	1	0	7	5	3	7	4	3
P2	2	0	0	3	2	2	1	2	2
P3	3	0	2	9	0	2	6	0	0
P4	2	1	1	2	2	2	0	1	1
P5	0	0	2	5	3	3	5	3	1

Question2. Is the system in a safe state? If Yes, then what is the safe sequence?

Process Sequence	Available	3	3	2
P2	Need (-)	1	2	2
		2	1	0
	Release (+)	3	2	2
		5	3	2
P4	Need (-)	0	1	1
		5	2	1
	Release (+)	2	2	2
		7	4	3
P1	Need (-)	7	4	3
		0	0	0
	Release (+)	7	5	3
		7	5	3
P3	Need (-)	6	0	0
		1	5	3
	Release (+)	9	0	2
		10	5	5
P5	Need (-)	5	3	1
		5	2	4
	Release (+)	5	3	3
Total Available		10	5	7

Following is the safe sequence: P2 → P4 → P1 → P3 → P5

As the total available for the resource A, B and C is the same as total available after following the sequence, there the system is said to be in safe state.

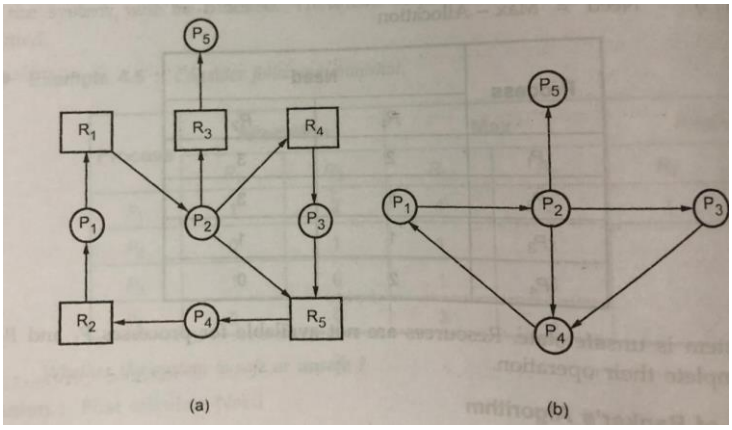
3.6 DEADLOCK DETECTION

If the system is not using any deadlock avoidance and deadlock prevention, then a deadlock situation may occur. Deadlock detection approach do not limit resource access or restrict process actions. With deadlock detection request resources are granted to processes whenever possible. Periodically, the operating system performs an algorithm that allows it to detect the circular wait condition. We discuss the algorithm for deadlock detection.

1 Single Instance of Each Resource Type

• If all resources have only a single instance, then deadlock detection algorithm that uses a variant of the resource-allocation graph, called a wait-for graph. Wait-for graph is obtained from resource-allocation graph. Nodes of resource is removed and collapsing the appropriate edge. i.e. assignment and request edge.

Figure below Resource-allocation graph with corresponding wait for graph



a) An edge from Pi to Pj in a wait for graph implies that process Pi is waiting for process Pi to release a resource that Pi needs.

b) An edge Pi + Pj exists in a wait for graph if and only if the corresponding resource-allocation graph contains two edges Pi -> Rq and Rq -> Pi for some resource Rq.

• In the figure of resource-allocation graph, process P1 is holding resource R2 and requesting resource R1. So this is shown in the following figure. In this, process P1 is request edge with process P2. It is shown in Figure below.

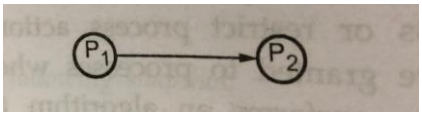


Figure Request edge

• If the wait for graph contains the cycle, then there is a deadlock. To detect deadlock, the system needs to maintain the wait for graph and periodically to invoke an algorithm that searches for a cycle in the graph. For detecting a cycle in the graph, algorithm requires n^2 operation. Where n is the number of vertices in the graph. This method is suitable for single instance of the resource.

2. Several Instances of a Resource Type

In deadlock detection approaches, the resource allocator simply grants each request for an available resource. For checking for deadlock of the system, the algorithm is as follows.

l) Unmark all active processes from allocation, Max and Available in accordance with the system state.

2) Find an unmarked process i such that

Max_i <= Available

If found, mark process i, update Available

Available : = Available + Allocation

and repeat this step.

If no process is found, then go to next step.

3) If all processes are marked, the system is not deadlocked.

Otherwise, system is in deadlock State and the set of unmarked processes is deadlocked.

Deadlock detection is only a part of the deadlock handling task. The system apply break the deadlock to reclaim resources held by blocked processes and to ensure that the affected processes can eventually be completed.

3.7 RECOVERY FROM DEADLOCK

Once deadlock has been detected, some strategy is needed for recovery.

Following are the solutions to recover the system from deadlock

1. Process termination

2. Resource preemption

1 Process Termination

• All deadlocked processes are aborted. Most of the operating system use this type of solution

1. Abort all deadlocked processes. This method clearly will break the deadlock cycle, but at great expense.

2. Abort one process at a time until the deadlock cycle is eliminated. This method incurs considerable overhead. The order in which processes are selected for abortion should be on the basic of some criterion of minimum cost.

• Many factors may affect which process is chosen, including:

a. Least amount of processor time consumed so far.

b. Least amount of output produced so far.

c. Lowest priority

d. Most estimated time remaining.

e. Least total resources allocated so far.

2 Resource Preemption

If preemption is required to deal with deadlocks then three issues need addressed:

1. **Selecting a victim**. Which resources and which processes are to be preempted?

2. **Rollback**. Backup each deadlocked process to some previously defined check point and restart all processes. This requires rollback & restart mechanisms built into the system.

3. **Starvation**. How do we ensure that starvation will not occur?

Advantages and Disadvantages

1 Advantages of Deadlock Prevention

1) No Preemption necessary.

2) Works well for processes that perform a single burst of activity.

3) Needs no run-time computation.

4) Feasible to enforce via compile time checks.

Disadvantages of Deadlock Prevention

1) Inefficient.

2) Delays process initiation.

3) Subject to cyclic restart.

4) Disallows incremental resource requests.

2. Advantages of Deadlock Detection

1) Never delays process initiation.

2) Facilitates online handling.

Disadvantage of Deadlock Detection

1) Inherent preemption losses.

3. Advantage of Deadlock Avoidance

1) No preemption necessary.

Disadvantages of Deadlock Avoidance

1) Future resource requirements must be known.

2) Processes can be blocked for long periods.