**International Journal of Multidisciplinary Research and Development**
**Online ISSN: 2349-4182 Print ISSN: 2349-5979**
**www.allsubjectjournal.com**
**Volume 3; Issue 5; May 2016; Page No. 48-51**

# Analysis for the detection of deadlock, its prevention and resolution techniques

**Monika Jain**

Asst. Prof. Dayanand P.G. College, Hisar, Haryana, India.

**Abstract**
In a distributed database environment, where the data is spread across several sites there are many concerns to deal with such as concurrency control, deadlock. Deadlocks impact the overall performance of the system. Deadlock is a process in which a number of processes share the same resource are effective preventing one another from accessing the resource. The main objective of this paper is to study the conditions of deadlock & to present a number of different possible methods to avoid deadlocks in computer system. Deadlock detection and resolution is one among the major challenges faced by a Distributed System. In this paper, we discuss deadlock detection techniques and present approaches for detecting deadlocks in Distributed Systems. A deadlock can be resolved by aborting one or more processes in the deadlocked-set and restart that process such that its previous state is resumed. A process is aborted when all of the resources it is holding is released, and withdraw all the resource requests it has made.

**Keywords:** Deadlock, processes, resources, conditions of deadlock, deadlock detection, deadlock prevention, deadlock avoidance and recovery.

## Introduction

In the multi programming environment, many processes may complete for a finite no of resources. But it may also happen that the processes in waiting state will never change their state again because other waiting processes hold the resources they have requested. This condition is called deadlock.

Thus, a deadlock is a situation in which two computer programs sharing the same resource are effectively preventing each other from the accessing of resources, resulting in both programs ceasing to the function.

A deadlock lowers the system utilization and hinders the progress of processes. Also the presence of deadlocks affects the throughput of the system. The dependency relationship among processes with respect to resources in a distributed system is often represented by a directed graph, known as the Wait for Graph (WFG). In the WFG each node represents a process and an arc is originated from a process waiting for a resource to a process holding the resource.

### Resources can be of various types

- I/O devices (like printer, tape drive etc.)
- Memory space
- CPU cycles
- Files & semaphores

### Consider an example

- Process1 request resource A and receives it
- Process 1 requests resource B & receives it.
- Process 1 request resource B and is queued up, pending the release of resource B.
- Process 2 request for resource A & is queued up, pending the release of resource A.

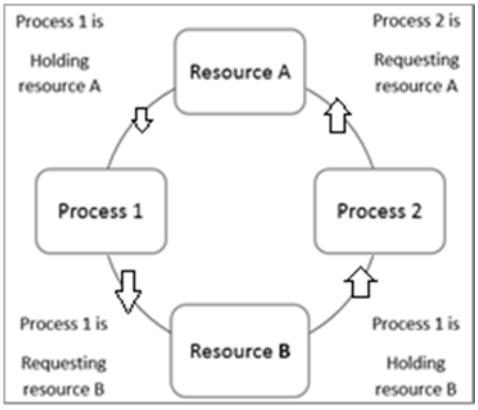


**Fig 1:** Deadlock State

Now, in this situation, operating system is unable to take any action. So at this stage, there is only one alternative is to abort (stop) one of the processes.

### Necessary and sufficient conditions for deadlock

Coffman (1971) identified that there are 4 conditions that must hold simultaneously for a deadlock to occur.

1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait

**Mutual Exclusion:** Only one process can use a resource at a time, if some other process request for that resource, the requesting process is delayed until the resource has been released. The resources involved are non-sharable.

**Hold and Wait:** The process holds one or more resources and requests for other resources which are held by other processer (i.e. Held a resource and wait for other)

**No Preemption:** No resource can be forcibly removed from the process holding it. It will be released only by explicit action of process.

**Circular Wait:** Two or more processes are in chain where each process waits for the resource that the next process in the chain holds
Example- Process A waits for process B which waits for process C which further waits for some other process D.

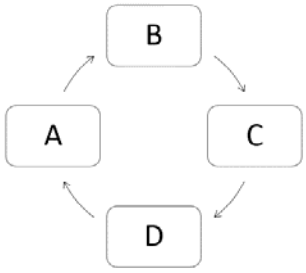


**Fig 2:** circular wait

**How can we deal with deadlock?**
In general, there are 4 methods to deal with the problem of deadlock. Methods for handing deadlock are
1. Ostrich approach
2. Deadlock prevention
3. Deadlock avoidance
4. Deadlock detection and recovery

**A) Ostrich Method:** In this method, the problem is ignored. It is used only when it is more cost effective to allow the problem to occur than to attempt its prevention.

**B) Deadlock Prevention:** Deadlock can be prevented by making sure that attest one of the 4 conditions of deadlock fails to hold:

**1. Elimination of Mutual Exclusion Condition**
- It can be done by making all the resources sharable. Example: if number of processes access read only files, then it can happen. But the resources like printers are non-sharable.

Thus, resources can't be changed from non-sharable to sharable. Thus mutual exclusive can't be fully eliminated.

**2. Elimination of Hold and Wait**
- It must be sure that whenever some process requests for a resource, it does not hold any other resource.
- The process should request and acquire all the requested resources before its execution starts.

**Related Problems**
- Many processes do not know about the number of resources before they started running.
- Starvation may occur when the process needs too many resources to start its execution. Process may be waiting for one or other resources for indefinite time.
- Low resource utilization: - many of the resources may be allocated to a process but stay unused for long time.

**3. Elimination of No – Preemption Condition**
- All the resources that are held by a process should be released when it requests for a new resource
- 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. Elimination of Circular Wait Condition:**
To avoid circular wait condition wait condition, order the resources by numbering and each process can request the resources in increasing order only.
This resource allocation graph will never have a cycle.

**C) Dead lock Avoidance: -** It requires the information about the processes in advance of resource allocation
- For this, each process declares the maximum number of resources required of each type that it may need
- System check while allocating the resource, whether the system in safe state or not.

**Safe State**
- It is a deadlock free state.
- There is some sequence by which are requests can be satisfied.
- To avoid deadlock, we should take care that the process will never enter to an unsafe state.

**Unsafe State**
A state is said to be unsafe, if there is no safe sequence.
- A deadlock state is an unsafe state.

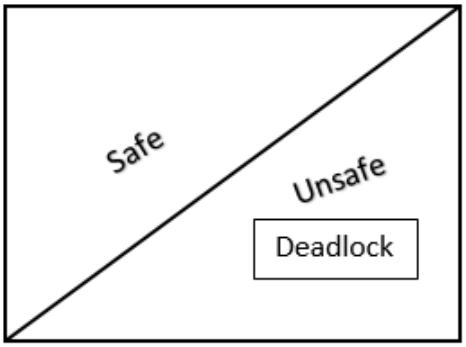


**Fig 3:** safe and unsafe state

**Example of a Safe State**
Let us consider that there are 11 tape drives in a system in a system & 3 processes. Maximum requirement and currently allocated tape drive for each process at time t0 is given.

| Process | Allocated | Maximum Required |
|---|---|---|
| P0 | 5 | 11 |
| P1 | 2 | 4 |
| P2 | 2 | 6 |

Available tape drives = 11- (5+2+2) =2

**Safe state**
The system is in safe state because sequence <p1, p2, p0> of allocation allows all the processes to complete.

Safe state exists at this time because system has 2 tape drives and it allocates these tape drives to process p1. As the process p1 completes, it releases all the 4 tape drives. System will allocate these tape drives to the process p2 and after the completion of process p2; system allocates these tape drives to $P_0$.

| | | |
|---|---|---|
| $P_0$=5, | $P_1$=2, | $P_2$=2, |
| Left=2 | $P_1$ = 2+2 =4 | completes and release 4 tape drives |
| Left=4 | $P_2$ = 2+4 =6 | completes and release 5 tape drives |
| Left=6 | $P_0$ = 5+6 =11 | complete without deadlock. |

**Unsafe State**

Maximum requirement & currently allocated type drives for each process at time $t_1$ are:

| process | allocated | Max required |
|---|---|---|
| P0 | 6 | 11 |
| P1 | 2 | 4 |
| P2 | 2 | 6 |

Available tape drive = 11- (6+2+2) =1

- It is unsafe state because no process will be completed.

Hence, deadlock avoidance algorithm should be able to tell whether the state of system is safe or not, after the request of particular process is granted.

If algorithm sees that granting the request lead to an unsafe state, then it asks the process to wait.

So, Banker's algorithm is used to avoid deadlock. It requires resource usage time to be known in advance.

**Banker's algorithm:** it is called banker's algorithm because the process is similar to that used by a banker in deciding if a loan can be made safely.

It checks that if granting the request leads to unsafe state, then request is denied.

**Example:** let there are 4 customers C1, C2 C3 & C4 each granted some credit units (1 unit = 1 lakh Rs). The banker assumes that all customers will not need their maximum credits immediately, so that he reserved 10 units to service them instead of 24 units.

| Process | Allocated | Maximum Required |
|---|---|---|
| C1 | 0 | 6 |
| C2 | 0 | 5 |
| C3 | 0 | 4 |
| C4 | 0 | 9 |

Here, customers = processes
Unit = resources
Banker= operating system

Customer makes loan requests time to time. At some stage, let the situation will be

| Customer | Allocated | Max required |
|---|---|---|
| C1 | 1 | 6 |
| C2 | 1 | 5 |
| C3 | 2 | 4 |
| C4 | 4 | 9 |

Available units = 10- (1+1+2+4) =2

**Safe state**

Here, the state is safe because 2 units are left. Banker delays request of others except C3. When C3 finish, it releases all the 4 resources. Now these 4 units will be given to C2 & C2 will finish its work & release all 5 units & so on.

**Unsafe state**

Let if C2 is requesting for one more resource until granted.

Then the situation will be

| Customer | Used | maximum |
|---|---|---|
| C1 | 1 | 6 |
| C2 | 2 | 5 |
| C3 | 3 | 4 |
| C4 | 4 | 9 |

Available units = 10 – (1+2+2+4) = 1

This is unsafe state.

Thus, banker's algorithm consider each request as it occurs & check if it leads to the safe state

If it is so, the request is granted, else it is postponed this algorithm,. Is used to handle multiple resources.

**D) Deadlock detection & Recovery**

It is the method to determine that a deadlock has occurred in the system & find out the processes & resources involved in the deadlock.

Once deadlock is detected, then recovery scheme will be applied.

**1 Deadlock detection algorithm**

It is a resource allocation graph. The graph contains

1. {$P_1$, $P_2$---------- $P_n$.}processes denoted by circles
2. { $R_1$, $R_2$---------- Rm } resources denoted by circles
3. $P_i$ to Rj, a dissected edge from process to resource indicates that process is requesting for resource
4. $R_i$ to Pj, a directed edge from resource to process tells that resource is allocated to the process

- First create a resource allocation graph, if there is a circular wait among the processes indicates that the deadlock is occurred.

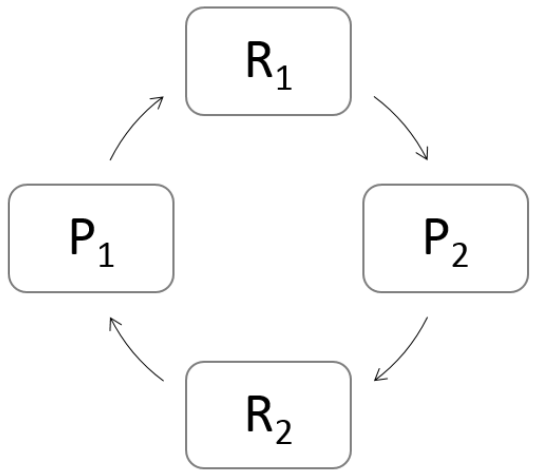


**Fig 4:** Resource Allocation Graph

Here, P1process is waiting for resource R1
R1 resource is held by process P2
P2 process is waiting for resource R2
R2 resource is held by process P1

Thus, there is a cycle among processes and resources, so processes are deadlocked.

**Recovery from deadlock**

When we find that the deadlock exists in the system from resource allocation graph. Now the system will be recovered from this state.

There are 2 ways to break a deadlock. These are:

**1. Process Termination**

1) Abort all deadlocked processes. It is a fast technique but lot of work done by the processes will be lost.
2) Abort one process at a time till the deadlock is eliminated. This method is better because all the processes will not be aborted. It is not appreciable because it involves more work and time since detection algorithm will have to be invoked again after aborting each process one by one.

**2. Resource Preemption:** We preempt some resources from processes & give these resources to other processes until deadlock cycle is broken.

**Issues in resource preemption:**

1) **Selecting a victim:** Which resource & process is chosen for preemption? If process is completed almost 90% of its execution, it is not appreciable to preempt it.
   It number of resources held by a process is too much, then it is appreciable to preempt that process.
2) **Rollback:** In rollback, the system returns to some safe state & restart the process for that state.
3) **Starvation process:** There should be a mechanism which looks that the particular process should not be preempted of resources again and again.
   A counter can be associated with each process to count the number of times its resources is taken away.

- The counter value must be least

**Conclusion:** Deadlock is a major problem in operating systems. Deadlocks in a distributed system drastically reduce the performance of the system and therefore have to be detected and resolved as soon as possible for the efficiency of the systems. Several techniques to resolve deadlock are mentioned above. One can use any of the above technique to resolve deadlock. In computer science, deadlock refers to a specific condition when two or more processes are each waiting for the other to release a resource, or more than two processes are waiting for resources in a circular chain. Deadlock is a common problem in multiprocessing where many processes share a specific type of mutually exclusive resource known as a software lock or soft lock. If a deadlock is detected, it must be broken by aborting and rolling back at least one process. An effective and efficient rollback and recovery mechanism is of immense need in computer system. Such a mechanism can make our approach to deadlock detection much more attractive. Research efforts are planned in this direction.

**References**


1. Seema Payal1, Ruchi Taneja2, Deadlock: A Problem of Computer System, International Journal of Advanced Research in Computer Science and Software Engineering, ISSN: 2277 128X. August 2015, 5(8),
2. Swati Gupta, Meenu Vijarania. Analysis for Deadlock Detection and Resolution Techniques in Distributed Database", International Journal of Advanced Research in Computer Science and Software Engineering, ISSN: 2277 128X. July 2013, 3(7).
3. Gupta Dhiraj, Gupta VK. Approaches for Deadlock Detection and Deadlock Prevention for Distributed systems. Research Journal of Recent Sciences. ISSN 2277-2502 (ISC-2011). 2012; 1:422-425
4. Peterson JL, Silberschatz. A Operating System concepts, Addison Wesley
5. Tanenbaum, An introduction to operating system.
6. Haberman, An introduction to operating system design, Galgotia Publication, New Delhi.