Deadlock Avoidance based on Banker's Algorithm for Waiting State Processes

Pankaj Kawadkar, Shiv Prasad, Amiya Dhar Dwivedi

Abstract— This paper presents an algorithm for deadlock avoidance used for Waiting State processes. This method is an improvement over Banker's algorithm. In Banker's algorithm, when processes goes to waiting state then there is no proper approach (FCFS is not sufficient) are available for the sequencing of waiting processes. In this paper a methodology has been proposed, which consider the number of allocated resources and/or number of instances as well as need of resources in order to select a waiting process for the execution.

Index Terms—Banker's Algorithm, Circular Wait, Edsger Dijkstra, Hold & Wait, Mutual Exclusion, No Preemption.

I. INTRODUCTION

A deadlock is a situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does. In an operating system, a deadlock is a situation which occurs when a process enters a waiting state because a resource requested by it is being held by another waiting process, which in turn is waiting for another resource. If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock. A deadlock situation can arise if and only if all of the following conditions hold simultaneously in a system: Mutual Exclusion: At least one resource must be non-shareable. Only one process can use the resource at any given instant of time. Hold and Wait or Resource Holding: A process is currently holding at least one resource and requesting additional resources which are being held by other processes. No Preemption: The operating system must not de-allocate resources once they have been allocated; they must be released by the holding process voluntarily. Circular Wait: A process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource [1]. In general, there is a set of waiting processes, P = {P1, P2, ..., PN}, such that P1 is waiting for a resource held by P2, P2 is waiting for a resource held by P3 and so on till PN is waiting for a resource held by P1. In general, three strategies are used for dealing with deadlocks.

1. Detection and recovery. Every time a resource is requested or released, a check is made to see if any deadlocks exist. If exist, some policy will be adopted to recover the system from deadlock.
2. Deadlock prevention, by structurally negating one of the four required deadlock conditions. Deadlocks will be impossible.
3. Dynamic avoidance [2-6] by careful resource allocation: deadlock avoidance. Make judicious choices to assure that the deadlock point is never reached.

In this paper, we focus on the problem of deadlock avoidance.

II. BANKER'S ALGORITHM FOR DEADLOCK AVOIDANCE

The Banker's algorithm is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources.

For the Banker's algorithm to work, it needs to know three things:

- How much of each resource each process could possibly request
- How much of each resource each process is currently holding
- How much of each resource the system currently has available

Basic data structures to be maintained to implement the Banker's Algorithm:

Let n be the number of processes in the system and m be the number of resource types. Then we need the following data structures:

- Available: A vector of length m indicates the number of available resources of each type. If Available[j] = k, there are k instances of resource type Rj available.
- Max: An n×m matrix defines the maximum demand of each process. If Max[i,j] = k, then Pi may request at most k instances of resource type Rj.
- Allocation: An n×m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i,j] = k, then process Pi is currently allocated k instance of resource type Rj.
- Need: An n×m matrix indicates the remaining resource need of each process. If Need[i,j] = k, then Pi may need k more instances of resource type Rj to complete task.

Note: Need = Max - Allocation.

The safety algorithm is used to determine if a system is in a safe state. It works as follows:

STEP 1: initialize Work := Available;
for i = 1,2,...,n
Finish[i] = false
STEP 2: find i such that both
a. Finish[i] is false

Manuscript Received on November 2014.

Mr. Pankaj Kawadkar, Department of Computer Science PIES Bhopal, India.
Mr. Shiv Prasad, M.Tech Scholar, PIES Bhopal, India.
Mr. Amiya Dhar Dwivedi, Department of Computer Science, BCET Durgapur, India.

b. Need_i <= Work
if no such i, goto STEP 4
STEP 3: Work := Work + Allocation_i
Finish[i] = true
goto STEP 2
STEP 4: if Finish[i] = true for all i, system is in safe state. The resource-request algorithm is used to determine whether requests can be safely granted. It works as follows:

- Let requesti be the request vector for process Pi. If request[i][j] = k, then process Pi wants k instances of resources Rj.
- When a request is made by process Pi, the following actions are taken:

1. If requesti ≤need go to step 2; otherwise, raise an error condition.
2. If requesti ≤available, go to step 3, otherwise Pi must wait since resources are not yet available.
3. Have system pretend to allocate the requested resources to process Pi by modifying the state as follows:
Available = available - requesti
Allocationi = allocationi+ requesti
If the resulting resource allocation state is safe, then transaction is complete and Pi is allocated its resources. If unsafe, Pi must wait and old state is restored.

III. IMPROVEMENT OF BANKER'S ALGORITHM

In Banker's algorithm if some processes are going in waiting state then no proper approach is available to provide a safe –sequence.
We are proposed a waiting-state-process algorithm that is provide a safe-sequence:-it depend upon $need_i$=1,2…n, $Allocation_i$=1,2…n and Available resources.
It works as follows:
After the Resource-Request-Algorithm if process is going to waiting state then these step must be followed:
Step.1- $Need_i$ is compare to
Need(i+1 to last) .
Step.2- Take the process with minimum Need and maximum Allocation.
Step.3- If Available<$Need_i$ then Execute process. Set state is executed.
Step.4-
Available=Available + $Allocation_i$
Step.5- Repeat step 1 to 4 until all the
process is going to executed state.

IV. CONCLUSIONS

The Banker's Algorithm for Deadlock Avoidance has no systematic approach for those processes which are in waiting state. In this paper, we proposed that if process is going to waiting state then the consideration of number of allocated resources and/or number of instances as well as need of resources in order to select a waiting process for the execution will make Banker's Algorithm more efficient. Since the number of available resources will increase fastly. But on the same side it will add some extra overhead to the existing Banker's Algorithm.

V. ACKNOWLEDGEMENTS

I would like to acknowledge the assistance of my books and also a sincere thank to all my friends. Finally, I thank my parents for supporting me throughout all my studies.

REFERENCES

1. Operating System Concepts (Silberschatz, Galvin, Gagne)
2. Banaszak, Z.A.; Krogh, B.H. Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows[J]. IEEE Transactions on Robotics and Automation, 1990, 6(6), 724 –734.
3. I.BenAbdallah ; H.ElMaraghy. Deadlock Prevention and Avoidance in FMS:A Petri Net-Based Approach[J]. International Journal of Advanced Manufacturing Technology, 1998, 16(1).
4. Naiqi Wu, MengChu Zhou. Avoiding deadlock and reducing starvation and blocking in automated manufacturing systems[J]. IEEE Transactions on Robotics and Automation, 2001, 17(5), 658 –669.
5. Viswanadham, N.; Narahari, Y.; Johnson, T.L. Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models[J]. IEEE Transactions on Robotics and Automation, 1990, 6(6), 713–723.
6. T. Araki, Y. Sugiyama, and T. Kasami. Complexity of the deadlock avoidance problem. In 2nd IBM Symp. Math. Found. Computer Sci., pages 229–257, 1977.
7. Tricas, F., Colom, J.M., Ezpeleta, J.: Some improvements to the banker's algorithm based on the process structure. Proceedings of IEEE International Conference on Robotics and Automation 3 (2000) 2853{2858 San Francisco, CA, USA.
8. Tricas, F.: Deadlock Analysis, Prevention and Avoidance in Sequential Resource Allocation Systems. PhD thesis, Departamento de Informatics Ingenerate de Sistemas, Universidad de Zaragoza (May 2003)

AUTHORS PROFILE

Mr. Pankaj Kawadkar, (Assistant Professor, PIES Bhopal)

Shiv Prasad, (M.Tech. 4th semester, PIES, Bhopal)

Mr. Amiya Dhar Dwivedi, (Assistant Professor, BCET, Durgapur WB)