A Dynamic and Improved Implementation of Banker's Algorithm

Ms. Kshipra Dixit (M.Tech. Student)
Department of Computer Science & Engineering,
Poornima College of Engineering,
Jaipur, Rajasthan, India

Dr. Ajay Khuteta
Department of Computer Science & Engineering,
Poornima College of Engineering,
Jaipur, Rajasthan, India

Abstract— Banker's algorithm can be described as deadlock avoidance and resource allocation algorithm which ensure the execution safety by simulating the allocation of already determined maximum possible of resources and makes the system into s-state by checking the possible deadlock conditions for all other pending processes.

It needs to know how much of each resource a process could possibly request. Number of processes is static in algorithm, but in most of system processes varies dynamically and no additional process will be started while it is in execution. The number of resources is not allowed to go down while it is in execution.

In this research an approach for Dynamic Banker's algorithm is proposed which allows the number of resources to be changed at runtime that prevents the system to fall in unsafe state. It also gives details about all the resources and processes that which one requires resources and in what quantity.

This modified banker's algorithm performs the process arrangement on the basis of their needs that leads to solve the problem in less time.

Keywords: *Banker's Algorithm, Improved Banker's Algorithm, Deadlock*

*****

I. Introduction

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause. Generally this event is release of a currently held resource.

No process can execute and free the resources.

In an operating system, a deadlock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process. 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 [1].

A. *Conditions for Deadlock*

Deadlock is possible if following four conditions occurs simultaneously

(i) Mutual exclusion condition: Only a single process at a time can execute and utilize a non sharable resource. Either each resource is allocated to a process or is available.

(ii) Hold and wait condition: A process holding at least one resource can request for additional resources.
(iii) No preemption condition: A resource can be released only voluntarily by the process holding it. That is previously granted resources cannot be forcibly taken away.

(iv) Circular wait condition: There exists a set {P0,P1,…,P0} 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 Pn, and P0 is waiting for a resource that is held by P0.

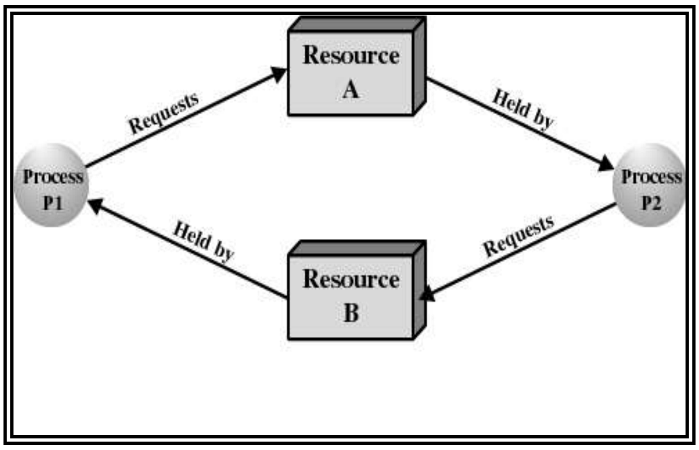

Figure 1: Circular wait

B. *Banker's Algorithm*

For resource categories that contain more than one instance the resource-allocation graph method does not work, and more complex (and less efficient) methods must be chosen.
The Banker's Algorithm is called Banker's algorithm because it is a process which bankers could use to ensure that when they lend out resources they will still be able to satisfy all their customers. A banker won't loan out a little money to start building a house unless they are assured that they will later be able to loan out the rest of the money to finish the house [3].

When a process starts its execution, it must state in advance the maximum allocation of resources it may request, up to the amount available in the system.
When a request is made, the scheduler computes whether approving the request would leave the system in a safe state. If

not, then the process should wait until the request can be approved safely [5].
The banker's algorithm relies on several key data structures: (where n is the number of processes and m is the number of resource categories.)

a. Available[m] indicates how many resources are currently available of each type.
b. Max[n] [m] indicates the maximum demand of each process of each resource.
c. Allocation[n] [m] indicates the number of each resource category allocated to each process.
d. Need[n] [m] indicate the remaining resources needed of each type for each process.

Note that Need[i][j] = Max[i][j] - Allocation[i][j] for all i, j.)

For simplification of discussions, we make the following notations / observations:

a. One row of the Need vector, Need[i], can be treated as a vector corresponding to the Needs of process i, and similarly for Allocation and Max.
b. A vector X is considered to be &lt;= a vector Y if X[i] &lt;= Y[i] for all i.

C. *Limitations of Banker's Algorithm*

Like all the other algorithms, the Banker's algorithm has some limitations when implemented in practical scenario. Few limitations are as follows:

i. It needs to know how much of each resource a process could possibly request.
ii. In most systems, the information about resource and processes is unavailable, which makes it impossible to implement the Banker's algorithm.
iii. It is unrealistic to assume that the number of processes is static since in most systems the number of processes varies dynamically.
iv. Moreover, the requirement that a process will eventually release all its resources (when the process terminates) is sufficient for the correctness of the algorithm, however it is not sufficient for a practical system.
v. Waiting for hours (or even days) for resources to be released is usually not acceptable.
vi. It does not give any information when system is not in safe state. It does not give details about the process which was failed during the execution and also does not give information also about the reason due to which it was not able to give the appropriate safe sequence.
vii. Banker's Algorithm does not show that particular process which needs more resources of what type. It just shows that it is not in the safe sequence, so it is very difficult to identify that which resource is needed to which process so the problem of unsafe sequence can be solved.

D. *Problem Statement*

One of the Major problem in Banker's Algorithm is that it does not provide the details about the process which was failed during the execution and also information about the reason due to which it was not able to give the appropriate safe sequence. Banker's Algorithm does not show that particular process which needs more resources of what type. It just shows that it is not in the safe sequence, so it is very difficult to say that which resource is needed to which process so the problem can be solved.
The Proposed approach will give the details about all the resources and processes that require resources in what quantity. This approach also performs the process arrangement on the basis of their needs that leads to solve the problem in less time.

II. LITERATURE SURVEY

In 2014, Pankaj Kawadkar, Shiv Prasad, Amiya Dhar Dwivedi in their research "Deadlock Avoidance based on Banker's Algorithm for Waiting State Processes" proposed an algorithm for deadlock avoidance used for Waiting State processes.

They 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. But they didn't give any solution when the system is in unsafe state [2].

In 2013, Smriti Agrawal, Madhavi Devi Botlagunta and Chennupalli Srinivasulu in their research titled "A Total Need based Resource Reservation Technique for Effective Resource Management" and proposed an approach for Total Need Based Resource Reservation (TNRR) that suggests reserving some resources so as to ensure that at least one process will complete after it [4].

The simulation results indicate that the frequency of deadlocks has reduced by approximately 75% for higher load (above 80%) as compared to the Deadlock Recovery technique, while for lower load it tends to be zero.

The turnaround time of the TNRR is approximately 9% better than the existing Banker's algorithm. But in case of insufficient resources when there is no safe sequence is possible they didn't provide details for resources and processes that causes the deadlock if executed or no safe sequence [4].

In 1999, Sheau-Dong Lang in his research titled "An Extended Banker's Algorithm for Deadlock Avoidance" proposed an approach for safety in banker's algorithm assuming that the control flow of the resource-related calls of each process forms a rooted tree, they proposed a quadratic-time algorithm which decomposes these trees into regions and computes the associated maximum resource claims, prior to process execution. This information is then used at runtime to test the system safety using the original banker's algorithm. But this approach was unable to recognize patterns of resource-related calls in real-time system and the practicality was also not proven [6].

TABLE 1
RELATED IMPROVEMENTS DONE IN BANKER'S ALGORITHM

Year	**Research Title (Author/s)**	**Improvements in Banker's Algorithm**
2014	Deadlock Avoidance based on Banker's Algorithm for Waiting State Processes. (PankajKawadkar, Shiv Prasad, Amiya DharDwivedi)	if process is going to waiting state then consideration of number of allocated resources and instances as need of resources in order to select a waiting process for the execution
2013	A Total Need based Resource Reservation Technique for Effective Resource Management. (Smriti Agrawal, Madhavi Devi Botlagunta and ChennupalliSrinivasulu)	Reserving resources so as to ensure that at least one process will complete after it, results indicate that the frequency of deadlocks has reduced.
1999	An Extended Banker's Algorithm for Deadlock Avoidance. (Sheau-Dong Lang)	Based on control flow of the resource-related calls computes the associated maximum resource claims, prior to process execution

TABLE 2
LIMITATIONS OF RELATED WORKS

Year	**Research Title (Author/s)**	**Limitations**
2014	Deadlock Avoidance based on Banker's Algorithm for Waiting State Processes. (PankajKawadkar, Shiv Prasad, Amiya DharDwivedi)	This method does not give any solution when the system is in unsafe state.
2013	A Total Need based Resource Reservation Technique for Effective Resource Management. (Smriti Agrawal, Madhavi Devi Botlagunta and Chennupalli Srinivasulu)	Didn't provide details for resources and processes that causes the deadlock if executed or no safe sequence
1999	An Extended Banker's Algorithm for Deadlock Avoidance. (Sheau-Dong Lang)	Unable to recognize patterns of resource-related calls in real-time system and the practicality was also not proven.

A. *Motivation*

Many researchers have been done for the improvement of Banker's Algorithm. Most of the researchers has worked on the limitations of waiting time and resource allocation to improve the performance or minimizing the deadlocks. But if at the end when system is not in safe state and traditional Banker's algorithm cannot be applied then what? We do not have any information about the process or resources due to which the system was in unsafe state. This specific problem leads us towards this approach.
Proposed approach gives the details about all the resources and processes that require resources in what quantity. This also allocates the resource automatically to the stopped process for the execution and will always give the appropriate safe sequence for the given processes.

III. PPOPOSED APPROACH

Banker's algorithm was originally designed to check whether the allocation of resources leave the system in safe state or not and if it is in safe state then it gives the safe sequence of processes and allocate the resources.

In banker's algorithm when, a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system.

When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If it will, the resources are allocated; otherwise, the process must wait until some other process releases enough resources.

Several data structures must be maintained to implement the banker's algorithm. These data structures encode the state of the resource-allocation system.

A. *Modified Banker's Algorithm:*

Input:

A stack of Needed Resources (Min Need will be on top always).

N = total number of process; i = current process

Available = total number of resources which are free to use

Allocation (i) = number or resources already held by Process (i).

Output: Safe sequence for Process Execution.

Algorithm:

```
1.  FOR Process (i)
        WHERE i=0 to n-1
        CHECK Need (i)
2.  IF Need (i)<= Available
        THEN
      a. Available = Available – Need (i);
      b. execute (i);
      c. Available = Available + Allocation;
      d. write process execute
      e. go to next process
3.  ELSE
        INSERT Need INTO STACK
4.  END
```

B. *Example*

Example to find safe sequence for process execution with five processes and three resources.

TABLE 3
TOTAL NUMBER OF RESOURCES

R1	R2	R3
11	5	10

TABLE 4
AVAILABLE RESOURCES:

R1	R2	R3
0	2	3

TABLE 5
PROCESS EXECUTION

Process	Max Demand			Allocation			Need		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P0	11	3	4	3	0	2	8	3	2
P1	8	3	4	3	0	0	5	3	4
P2	1	3	5	0	1	3	1	2	2
P3	2	3	4	2	1	1	0	2	3
P4	3	3	7	3	1	1	0	2	6
Allocated resources	11			3			7		

Algorithm Working:

Step1: For Process P0 check if need is more than available then PUSH Process P0 into STACK.

Step 2: For Process P1 check if need is more than available then PUSH process P1 into STACK.

(Compare stacks values and arranges process in ascending order in accordance with their need)

Step 3: For Process P2 check if need is more than available Then PUSH Process P2 into STACK and arrange STACK.

Step 4: For Process P3 check if need is less than available then Process is executed.
GOTO the next Process.

5 For Process p4 check if need is more than available Then PUSH Process P4 into STACK and arrange STACK.

Process STACK Values:

P2
P4
P1
P0

Now the Process will be executed from the top of the stack.

IV. RESULTS AND ANALYSIS

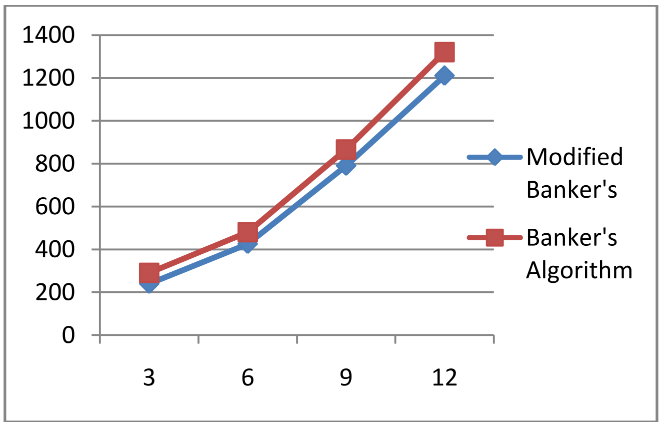

Figure 2: Execution Time Vs Number of Process

The effect of the increasing number of process with fixed number of resources over the average execution time can be seen in the figure 2.
The average execution time increases for both the techniques as the process increases. This is because, more process leads to higher contention for the resources and more frequent deadlocks.
However, the performance of the proposed is better for all ranges because the overhead involved for resource allocation is much lower than that of the Banker's algorithm (BA). The average improvement is approximately 9%.

V. CONCLUSION AND FUTURE SCOPE

A. *Conclusion*

This research shows the Banker's Algorithm working, problem in original algorithm to identify the reason for failing the process execution. Here Dynamic Banker's algorithm solves the existing problem of the original Banker's algorithm.

Results prove that modified Banker's Algorithm shows that particular process which needs more resources of what type. It also shows that it is in the safe sequence or not, so it is very easy to add which resource is needed to the process so the problem can be solved by this approach.

B. *Future Scope*

The present and future of this area is bright, and full of opportunities and great challenges as it processes high demands.

In future it can be used for the auto added process and killing the undesired process.

References

[1] Goswami, Vaisla and Ajit Singh, "VGS Algorithm: An Efficient Deadlock Prevention Mechanism for Distributed Transactions using Pipeline Method" International Journal of Computer Applications (0975 – 8887) Volume 46–No.22, May 2012.

[2] Pankaj Kawadkar, Shiv Prasad, Amiya Dhar Dwivedi, Deadlock Avoidance based on Banker's Algorithm for Waiting State Processes, International Journal of Innovative Science and Modern Engineering, Volume-2 Issue-12, November 2014

[3] N. Ramasubramanian, Srinivas V.V., Chaitanya V, "Studies on Performance Aspects of Scheduling Algorithms on Multicore Platforms," International Journal of Advanced Research in Computer Science and Software Engineering, Vol 2, Issue 2, February 2012.

[4] Smriti Agrawal, Madhavi Devi Botlagunta and Chennupalli Srinivasulu; A Total Need based Resource Reservation Technique for Effective Resource Management&quot;, International Journal of Computer Applications (0975 – 8887), Volume 68– No.18, April 2013

[5] B Madhavi Devi, Smriti Agrawal, Ch. Srinivasulu, "An Efficient Resource Allocation Technique for Uni-Processor System" International Journal of Advances in Engineering & Technology (IJAET) Volume 6 Issue 1, March 1, 2013.

[6] Sheau-Dong Lang, An Extended Banker's Algorithm for Deadlock Avoidance, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 25, NO. 3, MAY/JUNE 1999

[7] H. S. Behera, Ratikanta Pattanayak, Priyabrata Mallick, "An Improved Fuzzy-Based CPU Scheduling (IFCS) Algorithm for Real Time Systems," International Journal of Soft Computing and Engineering (IJSCE) (2231-2307), Volume-2, Issue-1, March 2012.

[8] Saroj Hiranwal, Dr. K.C.Roy, "Adaptive Round Robin Scheduling using Shortest Burst Approach Based on Smart Time Slice," International Journal of Data Engineering (IJDE), Volume 2, Issue 3 2012.

[9] A. Silberschatz, P. B. Galvin, and G. Gagne, Operating Systems Concepts, 6th edition, Addison-Wesley, Reading, Mass, pp. 204, 243, 244, 266, 2002.

[10] G. Nutt, Operating Systems, a Modern Perspective, 2nd edition, Addison-Wesley, Reading, Mass, Pages.150-279, 2000.