AN EFFICIENT RESOURCE ALLOCATION TECHNIQUE FOR UNIPROCESSOR SYSTEM

Botlagunta Madhavi Devi, Smriti Agrawal, Chennupalli Srinivasulu
Department of Information Technology,
JB Institute of Engineering and Technology, Hyderabad, India

ABSTRACT

In a system several processes may compete for a finite number of resources. A process request resources; and if the resources are not available at that time, it enters a wait state and may never execute. This paper aims to present a resource allocation algorithm that ensures that the requested resources are made available to the processes while incurring lower overhead for deciding about the resource allocation. In this paper we show that the overhead incurred by the existing Banker's algorithm is of the $\boldsymbol{O(mn^2)}$ *while that of the proposed threshold based resource allocation technique is* $\boldsymbol{O(m)}$*.The simulation results indicates that the average turnaround time of the proposed technique is approximately 9% lower than the existing technique leading to overall improvement in the performance of the system.*

KEYWORDS: *Deadlock, Deadlock avoidance, Deadlock Recovery, Banker's Algorithm*.

I. INTRODUCTION

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, is referred to as deadlock [1, 2, 3, 8]. Deadlock processes never terminate their executions and the resources held by them are not available to any other process, hence, there is no progress in the system and circular wait cycles are formed. These cycles are incapable of terminating by themselves. This in turn leads to low throughput, degraded performance and poor resource utilization. Deadlock is a common problem in multiprocessing systems, parallel computing and distributed systems, which handle shared resources and implement process synchronization [4, 8, 9].

Deadlock is a well-known problem and highly undesirable, because for systems such as automated manufacturing systems, distributed systems etc. it may lead to catastrophic losses. Coffman [1, 8] studied and suggested necessary conditions for a deadlock to occur. These conditions are: Mutual exclusion, Hold and wait, No pre-emption and Circular wait condition. The authors [1], suggested the following strategies for deadlock handling: deadlock prevention, deadlock avoidance, deadlock detection.

The deadlock prevention technique can prevent deadlock if and only if, one of the four necessary conditions stated above fails to hold. However, Mutual exclusion, Hold and wait, and No pre-emption conditions are system dependant and may not be prevented [1, 12]. Thus, preventing circular wait from occurring is the best way for preventing deadlock. This can be achieved by using a hierarchy to determine a partial ordering of resources [8, 10].

Deadlock can be avoided, if some prior knowledge about the resource requirement is available. A well-known deadlock avoidance algorithm used in operating systems is the banker's algorithm which was proposed by Dijkstra to handle a single resource type [4, 5], and later extended by Habermann to handle multiple resource types [4, 5, 6, 7]. Banker's algorithm takes into consideration the maximum resource requirement at any given time by a process in the system. It would grant the request if the

resulting system remains in a "safe" state. That is, even in the worst case that all processes request their maximum claims, there is still a schedule of process execution so that all the requests will be granted eventually. However, both the prevention and avoidance strategies, lead to lower device utilization and throughput. Hence, authors [8, 9] suggested using deadlock detection strategies, where corrective measures (deadlock recovery) are taken only when the deadlock actually occurs. When a deadlock actually occurs no process is capable of completing, hence, the system throughput and resource utilization tends to zero. The deadlock detection strategy is most efficient till a deadlock does not occur, but, incurs overhead for deadlock recovery and for some finite amount of time the system does not respond to the user requests.

This paper presents a new threshold based resource allocation technique that will allocate the resources to the requesting process. The proposed technique reduces the overhead incurred by the deadlock avoidance but do not guarantee that a deadlock will never occur. However, as compared to the deadlock detection strategy, it reduces the frequency of deadlock occurrence in the system. The proposed threshold based technique increases the system performance by reducing the overhead incurred during both the deadlock avoidance and recovery. Further, banker's deadlock avoidance technique does not consider the arrival and burst time of the process while doing the resource allocation leading to poorer performance. The proposed technique also takes in account the arrival and burst time for the processes while granting the resources.

The rest of the paper is organized as follows: section II, illustrates a motivational example while section III describes the system model. Section IV and V elaborates our proposed approach and simulation results. Finally, paper concludes with section VI.

II. MOTIVATIONAL EXAMPLE

In this section we present a motivational example which will illustrate the limitations of existing techniques.

Example [9]: Consider a system consisting of five processes P_0 through P_4 and three resource types R_1, R_2 and R_3 with instances 10, 5 and 7 respectively. Suppose that, at time t_0, the snapshot of the system has been taken as given in table1.

Table 1: Snapshot of the system at time t_0

	Allocation			Maximum			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	0	1	0	7	5	3	7	4	3	3	3	2
P_1	2	0	0	3	2	2	1	2	2			
P_2	3	0	2	9	0	2	6	0	0			
P_3	2	1	1	2	2	2	0	1	1			
P_4	0	0	2	4	3	3	4	3	1			

Where Allocation R_j, represents the number of instances of a resource type R_j allocated to a process P_i;

Maximum R_j, represents the at most demand for resource type R_j by process P_i during its entire execution;

Need R_j, represents the remaining need for resource type R_j by the process P_i;

Available R_j, is the number of resource type R_j available in the system at time t_0.

Suppose at time t_0 a process P_1 request for one additional resource type A, and two instances of resource type C, i.e., (1, 0, 2), the decision that whether this request can be granted immediately by the existing techniques is done as follows:

Table 2: Snapshot of the system, after the probable allocation to P_1 (1, 0, 2)

	Allocation			Maximum			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	0	1	0	7	5	3	7	4	3	2	3	0
P_1	3	0	2	3	2	2	0	2	0			

P_2	3	0	2	9	0	2	6	0	0	
P_3	2	1	1	2	2	2	0	1	1	
P_4	0	0	2	4	3	3	4	3	1	

Deadlock Avoidance (Banker's Algorithm) [4, 5, 9]: At time t_0, when P_1 request for (1, 0, 2) resources, then the Banker's algorithm will calculate the safety sequence, i.e., a sequence with worst case resource allocation that will lead to completion of all processes. Thus, in case the requested resources are granted the snapshot of the system can be seen in the table 2. Banker's algorithm will estimate the safety sequence (refer section 3) as $\langle P_1, P_3, P_4, P_0, P_2 \rangle$. Thus, it will allocate the requested resources to process P_1 at time t_1, causing an overhead of $t_1 - t_0$. Suppose at time t_2, process P_0 request for (0, 2, 0) resources then the snapshot can be seen in the table 3. Banker's algorithm will again estimate the safety sequence causing overhead. However, this time it is unable to find a single process whose need is less than available, hence, no process will be able to complete itself as the need for all the process as can be seen in the table 3 is greater than the available. Thus, the Banker's algorithm declines this request.

Deadlock Recovery: Deadlock recovery technique does not do any pretesting, it will simply grant the requests as and when made by processes P_1 and P_0. However, immediately the system may not be in deadlock state, but over time all the processes will eventually ask for the resources mentioned in their need column of table 3 without releasing any resources up to their completion. Thus, a deadlock will eventually occur. Hence, the preemption of the resources or process needs to be done causing an overhead and degraded performance.

Table 3: Snapshot of the system, after the probable allocation to P_0 (0, 2, 0)

	Allocation			Maximum			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	0	3	0	7	5	3	7	2	3	2	1	0
P_1	3	0	2	3	2	2	0	2	0			
P_2	3	0	2	9	0	2	6	0	0			
P_3	2	1	1	2	2	2	0	1	1			
P_4	0	0	2	4	3	3	4	3	1			

The motivational example clearly demonstrate that the above approaches either perform rigorous testing or no testing leading to no or frequent deadlocks. This paper strives to balance between the two and suggest an approach which will produce higher performance by performing a lower cost test. The following section presents the assumption and the terminologies used.

III. SYSTEM MODEL

This paper deals with resources allocation technique that allocates the resources to the requesting processes. The system is assumed to have m resource types, i.e.,R_1, R_2, $R_3 \ldots R_m$, with α_1, α_2, $\alpha_3 \ldots \alpha_m$ instances of each type. Further, it consists of n independent processes P_1, P_2, $P_3 \ldots P_n$ where, each process P_i has the attributes(a_i, e_i) that is the arrival and worst-case execution time respectively. The processes are assigned priority based on smallest execution time (Shortest job first, SJF). Several data structures are needed for maintaining the state of the resource allocation in the system; they can be defined as follows:

- **Available:** An array of m elements, indicating the number of instances available for each resource type. Thus, Available(R_j), is the number of resource type R_j available in the system.
- **Maximum:** A two dimensional array $n \times m$, defining the maximum resource demand of each process. If Max[i][j] equal k, then process P_i may request at most k instances of resources type R_j in its life time.
- **Allocation:** A two dimensional array $n \times m$, defines the number of resources of each type currently allocated to each process. If Allocation[i][j] equals k, then process P_i is currently allocated k instances of resources type R_j.

- **Need:** A two dimensional array n × m, indicates the remaining resource need of each process. If Need[i][j] equals k, then process P_i may need k more instances of resources type R_j to complete its execution. It can estimated as $\text{Need}[i][j] = \text{Maximum}[i][j] - \text{Allocation}[i][j]$
- **Threshold:** An array of m elements, indicating the minimum number of remaining resource at least one process will require to complete. If threshold[j] equals k, then process at least one process P_i will require at most k instances of resource type R_j to complete its execution. It can be estimated as $\text{Threshold}[j] = \lceil \lfloor \text{need}[i][j] \forall i = 1, 2 \ldots n \rfloor, 0 \rceil$
- **Request:** A two dimensional array n × m, indicating the number of resource requested by process P_i during its execution. If Request[i][j] equals k, then process P_i request k instances of resource type R_j for current execution.
- **Safe State[9]:** The system is said to be in Safe State, if allocation to each process can be made in some order (Safety Sequence) and still avoid a deadlock.
- **Safety Sequence[9]:** A sequence of process $\langle P_1, P_2, P_3 \ldots P_n \rangle$ is safe sequence for the current allocation state if, for each P_i, the resource requests that P_i can still make can be satisfied by the currently available resources plus resources held by all P_j, $j < i$. It can be estimated as suggested in the Banker's Algorithm as follows:

Safety Sequence Algorithm [9]:
Begin

1. ***Let Work and Finish be vectors of length m and n respectively. Initialize Work=Available and Finish[i]=false for i=1,2, … n***

2. ***For x= 1 to n***
   do
   a. ***For i= 1 to n***
      do
      i. ***If (Finish[i] = true) then goto step 2 a.***
      ii. ***For j= 1 to m***
          do
          1. ***If (Need[i][j] ≤ Work[j]) then***
             a. ***Can_exe=1;***
             Else
             b. ***Can_exe=0;***
             c. ***Goto step 2 a.***
          End for

      iii. ***For j= 1 to m***
          do
          1. ***Work[j]=Work[j] + Allocation[i][j]***
          2. ***Finish[i]=true;***
          3. ***Goto step 2***
   b. ***If (Finish[x] =true) for all x, then system is in safe state***

End

The following section illustrates the proposed threshold based technique for effective resource allocation.

IV. PROPOSED THRESHOLD BASED RESOURCE ALLOCATION TECHNIQUE

The motivational example in section 2, demonstrate that Banker's Algorithm will always estimate the safety sequence by considering the requirement for all the processes in the system. However, this cross checking incurs as overhead $O(mn^2)$. This paper proposes a threshold based technique in which the system reserves a pool of threshold number of resources to ensure that at least one process will always complete and relieves all the resources it is holding. The proposed technique only considers

the requesting process detail along with the system data structure to take the decision of granting or not granting of the resources incurring an overhead of mere $O(m)$. The proposed Threshold based Resource Allocation (TRA) technique can be summarized as follows:

Table 4: Process P_1 data structures when requesting for (1, 0, 2) resource

Resources type	Request	Need	Available	Threshold	Can_grant
R_1	1	1	3	1	1
R_2	0	2	3	1	1
R_3	2	2	2	1	0

TRA Algorithm :
Input: Process Priority Queue
begin

1. ***Initially***
   a. $\boldsymbol{Need = Maximum}$ */*whenever a process arrives it has no resource allocated to itself hence, the need is same as the maximum*/*
   b. $\boldsymbol{Available[j] = \alpha_j \forall j = 1, 2 \ldots m}$
   c. $\boldsymbol{Threshold[j] = \lceil [need[i][j] \forall i = 1, 2 \ldots n], 0 \rceil}$
   d. ***Allocation=0***
   e. ***Can_grant=0***

2. ***Till no request***
   a. ***Wait*** *// Do nothing*

3. ***If a process request Request[i], a vector of size m for each resource then***
   a. ***For j =1 1 to m***
      do
      i. ***If (Need[i][j] = threshold[j] and Request[i][j] ≤ Available[i][j]) then***
         1. ***Can_grant=1;***
         Else
         2. ***If (Available[i][j] – Request[i][j] ≥ Threshold[j]) then***
            a. ***Can_grant=1***
            Else
            a. ***Can_grant=0;***
               //decline the request
            b. ***Goto step 2***
   End for

4. ***For j=1 to m***
   do
   a. ***Allocation[i][j]= Allocation[i][j] + Request[i][j]***
      // grant the request
   b. ***Available[i][j]=Available[i][j] – Request[i][j]***
   // since the resources are allocated hence, they are no more //available
   c. ***Need[i][j]=Maximum[i][j] –Allocation[i][j]***
   End for

5. ***If a process i completes then***
   a. ***For j= 1 to m***
      do
      i. ***Available [i][j] = Available[i][j] + Allocation[i][j]***
      End for
   Remove P_i from the queue, goto step 2.

End

Table 5: Process P_0 data structures when requesting for (0, 2, 0) resource

Resources type	Request	Need	Available	Threshold	Can_grant
R_1	0	7	3	1	1
R_2	2	4	3	1	1
R_3	0	3	2	1	1

The effectiveness of proposed Threshold based Resource Allocation Technique can be seen by the motivational example in section 2. Here, for the same example resource allocation is done using the proposed approach. Thus, considering the system snapshot as illustrated in table 1. At time t_0, when P_1 request for (1, 0, 2) resources then Request[1][]={1, 0, 2}, entering the for loop as in step 3 of the TRA algorithm. The values of data structures can be seen in the table 4. The request for the resource type R_1 can be granted because the need for R_1 is equal to its threshold (step 3.a.i. of TRA algorithm), similarly as per the step 3.a.i.2, the allocation for the resource type R_2 can be done. However, if two instances of R_3 are granted then the available number of instance of R_3 in the system will be zero, i.e., lower than its threshold value, which is not permissible, hence, this request cannot be granted, therefore the snapshot of the system remains as in table1. Suppose at time t_2, process P_0 request for (0, 2, 0) resources as assumed in the example then the data structures can be seen in the table 5. This request can be granted and the snap shot of the system can be seen in the table 6.
The Banker's algorithm allocates the resources (1, 0, 2) to process P_1 and rejects the request of (0, 2, 0) to the process P_0 leaving the system in the safe state with safety sequence as $\langle P_1, P_3, P_4, P_0, P_2 \rangle$. On the other hand, the resource recovery algorithm will grant both the resources, leading the system to be in unsafe state. However, the proposed threshold based resource allocation technique (TRA) will reject the request made by P_1 but will accept the request of P_0, still the system is in safe state and safety sequence can be found as $\langle P_3, P_1, P_0, P_2, P_4 \rangle$. However, the safety sequence estimation is not required for deciding to accept or reject a resource request. Thus, the overhead involved in the proposed technique is considerable lower than that presented by the Banker's Algorithm.

Table 6: Snapshot of the system after granting the request made by P_2

	Allocation			Maximum			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	0	3	0	7	5	3	7	2	3	3	1	2
P_1	2	0	0	3	2	2	1	2	2			
P_2	3	0	2	9	0	2	6	0	0			
P_3	2	1	1	2	2	2	0	1	1			
P_4	0	0	2	4	3	3	4	3	1			

V. SIMULATION RESULTS

In this section simulation on synthesized process sets are performed to evaluate the performance of the proposed techniques: Threshold based Resource Allocation Technique (TRA). Comparison is done with the Banker's Algorithm (BA). The key parameters used for comparison is the Average Turnaround time, which is the difference in time between the submission of a process to its completion.
Processes were generated using an exponential distribution with inter arrival time (1/λ) and service time (1/μ) with parameters λ and μ, simulation is run for 1000 processes. The resources are also picked from a pool, randomly generated at the beginning.

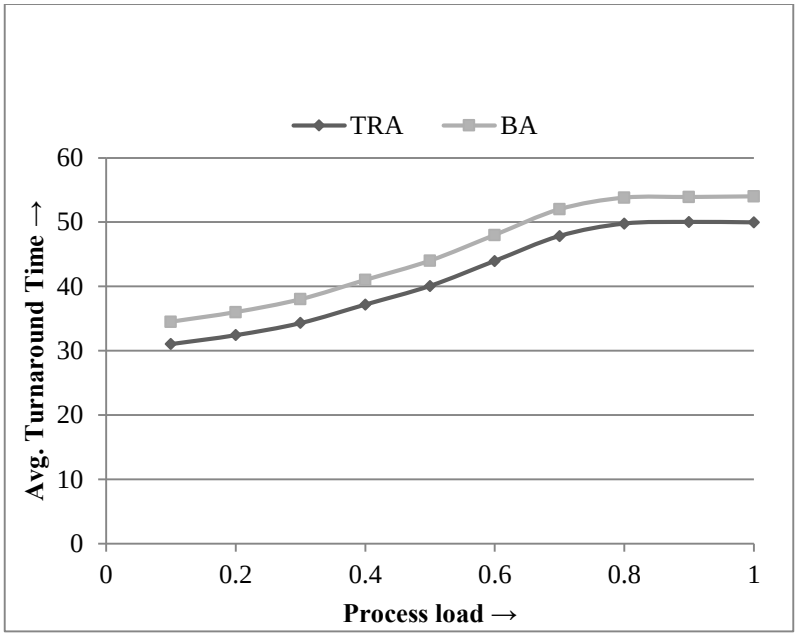

Figure 1: Process Load Vs. Turnaround Time

The effect of the process load increasing over the average turnaround time can be seen in the figure 1. The average turnaround time increases for all the techniques as the load increases. This is because, more loads 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%.

VI. CONCLUSION

In this paper we presented a threshold based resource allocation technique that focused on deadlock avoidance by ensuring that at least one process always has the requisite number of resources to complete. The motivational example illustrate that without performing the safety sequence check also the proposed algorithm is capable of allocating the resources and maintain the system in safe state with an overhead of mere $O(m)$ as compared to Banker's algorithm of the $O(mn^2)$.The average improvement in the turnaround time is approximately 9%.

VII. FUTURE WORK

The proposed technique reduces the frequency of deadlocks and hence, the average turnaround time improves. However, it does not guarantee that a deadlock will never occur. This work can be refined further to ensure that a deadlock will never occur while still maintaining the low overhead proposed by this technique.

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]. U. Kapasi, W. Dally, S. Rixner, J. Owens, and B. Khailany, "The Imagine stream processor", Proc. International Conference of Computer Design, 2002, 282–288.
[3]. D. Zobel, "The Deadlock problem: a classifying bibliography", ACM SIGOPS Operating Systems Review, vol. 17, October 1983.
[4]. Sheau-Dong Lang, "An Extended Banker's Algorithm for Deadlock Avoidance", IEEE Transactions on Software Engineering, vol. 25, no. 3, May/June 1999.
[5]. E.W. Dijkstra, "Cooperating Sequential Processes," Programming Languages, F. Genuys, ed., pp. 103-110, New York: Academic Press,1968.

[6]. A. N. Habermann, "Prevention of System Deadlocks," Comm. ACM, vol. 12, no. 7, pp. 373-377, 385, July 1969.
[7]. R.C. Holt, "Some Deadlock Properties of Computer Systems", ACM Computing Surveys, vol. 4, no. 3, pp. 179-196, Sept. 1972.
[8]. http://en.wikipedia.org/wiki/Deadlock
[9]. Silberschatz, P. B. Galvin and G. Gagne, "Operating System Principle", Seventh Edition, Wiley India.
[10]. Ran Yang, Sandjai Bhulai, Rob van der Mei, Frank Seinstra , "Optimal resource allocation for time-reservation systems", Performance Evaluation, vol. 68,no.5, pp.414-428, 2011
[11]. Y in-Fu Huang, Bo-Wei Chao," A priority -based resource allocation strategy in distributed Computing Networks", The journal of Systems and Software , vol. 58, pp. 221-233,2001
[12]. Hesuan Hu, Zhiwu Li, "Local and global deadlock prevention policies for resource allocation systems using partially generated reachability graphs", Computers & Industrial Engineering vol. 57, pp.1168-1181,2009

AUTHORS

B. Madhavi Devi received her Masters Degree from Vellore Institute of Technology,Vellore, India and Bachelor Degree from Sri Venkateswara University, Tirupati, India. She is presently, working with JB Institute of Engineering and Technology, Hyderabad, India as Associate Professor. Her research interest includes Cloud Computing, parallel programming, Grid Computing and performance evaluation in Operating Systems.

Smriti Agrawal was awarded PhD in 2009 by Motilal Nehru National Institute of Technology, Allahabad, India. She also received her BTech (hons) degree from VBS Purvanchal University in 2003. She is presently, working with JB Institute of Engineering and Technology, Hyderabad, India as Associate Professor. She has previously worked in Jaypee University of Information Technology, Solan, and Motilal Nehru National Institute of Technology, Allahabad, India. Smriti has around 17 publications in various international journals and conferences of repute. She has reviewed number a paper for various international journals and conferences. She is also the editor and Program Committee Member for various journals and conferences.

Ch Srinivasulu has obtained his B.Tech Degree from SV University and M.Tech (CSE) from JNT University, Hyderabad, India. He is having nearly 15 years' experience in Industry as well as a faculty of Computer Science and Information Technology departments. He is pursuing his PhD from JNTU Kakinada. His area of research includes Computer Architecture, Parallel Computing, Software Engineering. Presently he is working as Associate Professor in JB Institute of Engineering Technology, Hyderabad.