International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-6 Issue-4, April 2017

Implementing Fair Resource Synchronizer Algorithm for Distributed Mutual Exclusion in Mobile Computing Environment

Ahmed Sharieh, Raja Masadeh

Abstract*: Mutual exclusion in distributed systems is a critical feature required to coordinate access to shared resources. It is highly needed to be employed in distributed systems including mobile computing environments. Dynamic Resource Synchronizer algorithm (DRS) works on decreasing the amount of messages that transferred in the system by minimizing the amount of sites that are included in the mutual exclusion. In this paper, a DRS algorithm is presented with a simulation study for distributed mutual exclusion that could be used in mobile environments in which nodes communicate with each other based onto specific conditions. Also, ring topology is used, all nodes have a unique identifier, a node failure doesn't occur, communication links are bi-directional, and First In First Out (FIFO) priority and a partition in a network doesn't occur. In addition, decreasing the amount of storage which is needed at various sites on the system. The DRS algorithm proved that the mutual exclusion is achieved. Whereas, deadlock and starvation are impossible to occur. Thus development mutual exclusion algorithm is one of the most appropriate for mobile computer systems.

Index Terms*: Distributed systems, synchronization, mutual exclusion, mobile computing.

I. INTRODUCTION

Mutual exclusion algorithm in distributed is a main feature which is required to coordinate access to shared resource. Many distributed algorithms were proposed for mutual exclusion last 30 years [3]. In addition, several classifications of algorithms have been published. Algorithms were mainly classified into two types, nontoken-based and token-based [4]. In nontoken- based mutual exclusion algorithm, messages are alternated in order to define the node that could be the next access to the critical section (CS). The CS is the portion of process program that transacts with global resources at the same time [4, 11]. One example is Lamport's algorithm [12] which requires (3(N – 1) messages). Another example is Ricart-Agrawala's algorithm [5] that requires (2(N – 1) messages) [11]. However, Token-based mutual exclusion algorithm has only a unique token in the whole system and only the processor that holds the token could access the protected resources [4,11]. Token-based mutual exclusion algorithm examples include Suzuki-Kasami's algorithm [11,17] (which requires N messages, Singhal's heuristic algorithm [11,18] which requires (N/2, N) messages and Raymond's tree-based algorithm [11,19] which requires ((log N) messages).The distributed mutual exclusion's performance is measured by two metrics: message complexity and the synchronization delay. Message complexity indicates "the number of messages necessary per critical section (CS) invocation". The synchronization delay indicates to "the time required after a process leaves the CS and before the next process enters the CS". The proposed algorithm by [1] belongs to the second type (the token-based). It minimizes the message complexity and reduces the amount of storage, and reduces the communication between nodes by messages and shared memory.

Fairness is one of the most critical criteria for almost every aspect of real life provided solutions. On the other hand, fairness in mutual exclusion means that requests to enter CS is satisfied with the arrangement of their logical timestamp [12]. During the last years, many distributed mutual exclusion algorithms have been suggested to satisfy the fairness criterion. Jani [13] presented the best performance of the improved fair algorithm that suggested by Lodha-Kshemkalyani (LK) over Ricart-Agrawala algorithm (RA) by using comprehensive simulation for several CS access patterns and several levels of network loads. While Kanrar[14] suggested a new token-based mutual exclusion algorithm for distributed systems which keeps a relatively high level of fairness, taking into consideration the dynamic process priority. In addition; Kanrar[14] had compared his results with FAPP (Fairness Algorithm for Priority Process), and he got better results in case of link failure.

Lejeune [15] presented an effective starvation-free priority-based mutual exclusion algorithm that based on Kanrar–Chaki algorithm. However, this paper provided fairness only in case of starvation. Priorities are related to the demands which could be dynamically raising for assuring that demands with minimum priority are satisfied within a restricted time. Thus, starvation is avoided. Kundu [16] presented a simple sufficient condition "triangle-free property of the interference graph G". In addition to that, the paper presented a procedure for choosing the request-sets Ri to obtain distributed generalized relaxed mutual exclusion. In the other words, no deadlock occurs and fairness is defined in terms of satisfying the processing load between the nodes.

Itriq [20] applied the DRS mutual exclusion algorithm [1] on the infrastructure machines which connected immediately with the mobile hosts (MSS) through a few modifications.

Manuscript published on 30 April 2017.
***** Correspondence Author (s)
Ahmed Sharieh*, Computer Science, The University of Jordan, Faculty of Information Technology, Amman, Jordan, E-mail: sharieh@ju.edu.jo.
Raja'a Masa'deh, Software Engineering, The World Islamic Science and Education University, Faculty of Information Technology, Amman, Jordan, E-mail: raja.masadeh@wise.edu.jo.

Adaptive Dynamic Resource Synchronizer (ADRS) enhanced the algorithm when it decreases the number of messages that will be transferred and reduces the traffic load in the network. The rest of the paper is organized as follows: Section II contains the Dynamic Resource Synchronizer (DRS) Algorithm and how it works. Assertions are presented in section III. Section IV includes the analysis and discussion of implementation of the algorithm. Finally, Section V draws the conclusion.

II. DRS ALGORITHM

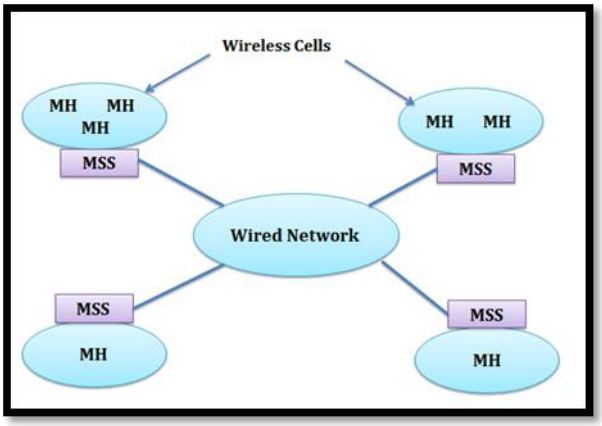

Fig. 1: System Model [2]

The proposed algorithm by [1] is based on the system that used in Fig. 1 [2]. In addition, it takes into considerations the memory limitations, limited battery life, and working under low bandwidth. Mobile Host (MH) is a host that can move while holding its network connection. The infrastructure machine that communicates directly with the mobile host is called Mobile Support Station (MSS). The logical or geographical coverage area under the (MSS) is called a cell. MHs that have identified themselves with a particular MSS are local to the MSS. MHs can directly communicate with a MSS (and vice versa) only if the MH is physically located within the cell serviced by the MSS, and each MH belongs to only one cell at a time.

A. Description

Based on the system model [2], the researchers suggested their novel algorithm for distributed mutual exclusion. That could be employed in mobile computing environments. They mentioned it as "Dynamic Resource Synchronizer (DRS) Algorithm", for the reason that the node which administers the CS is dynamically changed according to specific conditions that decreases the message traffic between the rest of nodes. As well, the dynamic synchronizer holds the information about each node wishes to enter the CS rather than each node holds the information about all nodes.

B. Assumptions

Suppose that the system consists of n autonomous mobile hosts (nodes). These nodes communicate to each other's by message passing through wireless networks. Assumptions on the mobile nodes are [1]:

- All nodes have a unique identifier.
- A node failure doesn't occur
- Communication links are bi-directional First In First Out (FIFO) priority.
- Communication link failures are predictable, providing a reliable communication.
- A partition in a network doesn't occur.

C. Specification

Each node in the system is assumed to be running an application whose status is partitioned into four states:

1. WAITING: the node has requested access to the CS. When the node is in the WAITING state, it can't send another request to enter CS.
2. CRITICAL: the node is executing the CS.
3. SYNCHRONIZER: the node is currently responsible for handling mutual exclusion access to the CS.
4. REMINDER: the node is neither requesting nor executing the CS.

D. Types of Messages

1. REQUEST message: a message that sent by a REMINDER node which wishes to enter the CS.
2. GRANT message: a message that sent by the SYNCHRONIZER to the next node in the queue.
3. RELEASE message: a message sent to the SYNCHRONIZER by CRITICAL node when exiting the CS.
4. YAS message: a message sent to the new SYNCHRONIZER from the current SYNCHRONIZER to transmit the synchronization state.
5. ADD message: a message sent by the SYNCHRONIZER to add the new node to the queue.
6. CHANGE message: a message sent to the CRITICAL by the SYNCHRONIZER to tell it of the SYNCHRONIZER status transmits.

E. Rules

The first rule is "demanding the CS": When the node desires to enter the CS, at the beginning, it checks the status. If the status is REMINDER, it provides a REQUEST message that includes its address and sends it to the next node until reaching the SYNCHRONIZER node and changes the status to WAITING. However, if its state is SYNCHRONIZER and busy is False, it sends YAS message and changes its state to CRITICAL then enters the CS. If its status is SYNCHRONIZER and busy is TRUE, it adds itself at the end of the queue because there is another node at the CS. As well as, it sends YAS message to the next and changes its state to WAITING.The second rule is" Processing a REQUEST message". When the REQUEST message reaches at the node, it checks its status. If its status is SYNCHRONIZER and busy is TRUE, the SYNCHRONIZER adds the requesting node to the end of the queue and checks its pointer; if it isn't nil, the requesting node's address is added to the message and send it to the node in the queue. If status is SYNCHRONIZER and busy is FALSE, a GRANT message is sent to the requesting node and changes the status to CRITICAL and changes busy to TRUE. However, if its status isn't SYNCHRONIZER, it transmits the message to the next node.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

The third rule is" Processing a GRANT message". When a GRANT message reaches the requesting node which was sent from SYNCHRONIZER, it enters the CS and changes its status to CRITICAL. In addition to that, it saves the YNCHRONIZER's address in synch.

The fourth rule is" exiting the CS": This function is called by a node when it decides to exit the CS. It sends RELEASE message to the SYNCHRONIZER and changes its status to REMINDER.

The fifth rule is" Processing a RELEASE message". When the SYNCHRONIZER receipts a RELEASE message, it sets its status to REMINDER, changes busy to FALSE and sends a YAS message to the node which has been completed to tell it that it is the current SYNCHRONIZER.

The sixth rule is" Processing a YAS message". When YAS message reaches the node, if its status is not REMINDER, it transmits the message to the next node and sends CHANGE message to critical section. However, if its status is REMINDER, it sets its status to SYNCHRONIZER and the node checks the queue's contents. If it is not nil, a GRANT message is sent to the node that is in the queue and changes busy to TRUE.

The seventh rule is" Processing an ADD message". When an ADD message reaches the node, it saves the address in the queue.

The eighth rule is" Processing a CHANGE message". When a CHANGE message reaches the node, it overwrites its synch to the address in the message.

III. ASSERTIONS

A. Mutual Exclusion

Mutual exclusion is obtained when two or more nodes never execute simultaneously their CS [5]. Which means one node must leave the CS before the other enters. This will be proved by contradiction.

Assume there are two nodes are executing the CS simultaneously. So, those two nodes have received a GRANT message from the synchronizer that sent in two cases. First; according to Rule (2) it sent by the synchronizer when it receives a REQUEST message and busy is false. Second; according to Rules 4, 5 and 6, it sent by a node that receives a YAS message after release the CS. Thus, in both cases only one GRANT message is sent when the busy is false. So we conclude that the mutual exclusion is reserved.

ASSERTION: Mutual Exclusion is achieved.

B. Deadlock

Deadlock occurs when no node in its CS and no demanding node can ever progress to the CS [5]. This could happen if there is no synchronizer or if no node knows there is a synchronizer. Thus DRS algorithm assumed that a dynamic node should be a synchronizer. When the algorithm is working, YAS message is used to transfer the synchronization status from node to another node. So, we conclude that there is only one CS in the system.

As supposed in DRS algorithm, only one node is initiated as a SYNCHRONIZER. During the running time of the algorithm, Rules 1, 2 and 6 administer YAS message which is employed to transmit the synchronization state from node to another. As well, Rule 2 presents how the SYNCHRONIZER becomes always standby for the other nodes that demand the GRANT messages to enter the CS. It saves the requesting node's address in its queue. In other case, if the SYNCHRONIZER in the busy situation, it sends the address to the node's queue. However, according to Rules 6, 7 and 8, it is serves the waiting nodes.

ASSERTION: Deadlock is impossible.

C. Starvation

Starvation happens when few nodes are entering and executing the CS while another node waits indefinitely to enter the CS [5]. That means there is a node can enter the CS many times while other nodes are waiting their turn to do that. To avoid this problem, the proposed algorithm changes the status of the node to REMINDER when leaves the CS according to Rule (5). When it wishes to enter the CS again, it must send a REQUEST message. According to Rule (2), it will add to the end of the queue when another node in the WAITING status.

ASSERTION: Starvation is impossible.

IV. ANALYSIS AND DISCUSSION

In order to assess the performance of a dynamic resource synchronizer mutual exclusion algorithm, a multithreaded Java based simulation program was developed. This implementation proved that the three conditions are satisfied.

The algorithm uses a ring topology. In addition, the communication between nodes is TCP/IP which includes the three portions that added to the message format: original source, original destination and the command.

A. Scenario Presentation

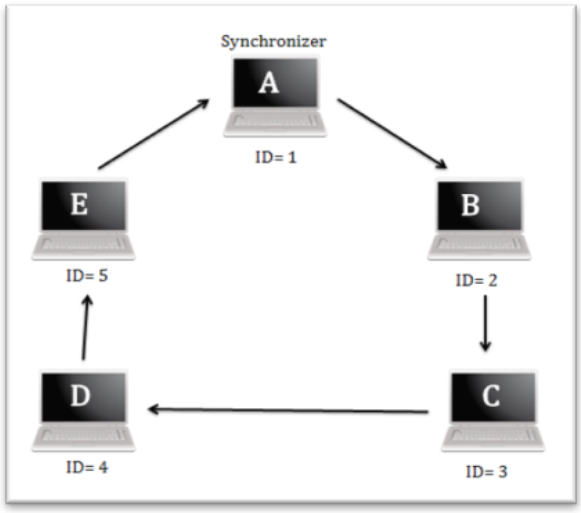

Fig. 2: Scenario Presentation

In this case, node A is initially the SYNCHRONIZER and the other nodes are in REMINDER state as shown in Table 1.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Table 1: The Initial Status of all Nodes

Node	Status
A	SYNCHRONIZER
B	REMINDER
C	REMINDER
D	REMINDER
E	REMINDER

At the beginning, node D wishes to enter the CS. So, it sends REQUEST message to the SYNCHRONIZER. When the SYNCHRONIZER receives the message, it checks the busy if it is false or true. If busy is false, it sends GRANT message to node D that will follow the ring from node A to B then C and finally to the requested node D which changes its status to CRITICAL and enters the CS as shown in Table 2.

Table 2: The First Updated Status of all Nodes

Node	Status
A	SYNCHRONIZER
B	REMINDER
C	REMINDER
D	CRITICAL
E	REMINDER

While the node D is in the CS, node B sends REQUEST message to the SYNCHRONIZER. Since the Synchronizer is busy, it adds node B to the queue and sends ADD message containing its address to node D. Then node E sends REQUEST message to the SYNCHRONIZER which adds it to the end of the queue. Also node C sends REQUEST message as shown in Fig. 3. The statuses of all nodes are presented in Table 3.

Table 3: The Second Updated Status of all Nodes

Node	Status
A	SYNCHRONIZER
B	WAITING
C	WAITING
D	CRITICAL
E	WAITING

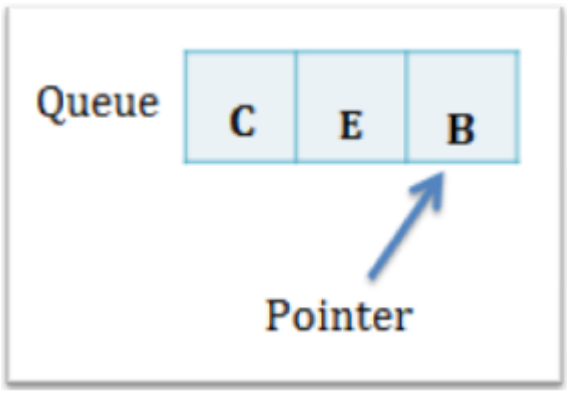

Fig. 3: The Current Queue in the SYNCHRONIZER

When node D finishes its running, it sends RELEASE message to the SYNCHRONIZER and changes its status to REMINDER. When the SYNCHRONIZER receives the RELEASE message, it sends YAS message to node D and changes its status to REMINDER. Then, the current SYNCHRONIZER which is node D sends GRANT message to node B to enter the CS which changes its status to CRITICAL shown in Table 4. The same scenario will be repeated in tables 5, 6, and 7.

Table 4: The Third Updated Status of all Nodes

Node	Status
A	REMINDER
B	CRITICAL
C	WAITING
D	SYNCHRONIZER
C	WAITING

Table 5: The Fourth Updated Status of all Nodes

Node	Status
A	REMINDER
B	SYNCHRONIZER
C	WAITING
D	REMINDER
E	CRITICAL

Table 6: The Fifth Updated Status of all Nodes

Node	Status
A	REMINDER
B	REMINDER
C	CRITICAL
D	REMINDER
E	SYNCHRONIZER

Table 7: The Sixth Updated Status of all Nodes

Node	Status
A	REMINDER
B	REMINDER
C	SYNCHRONIZER
D	REMINDER
E	REMINDER

B. Mathematical Analysis

According to the main paper [1], the best case is constant value that equal 3 as shown in Fig. 2. However the worst case is O (n^2) which is (1 REQUEST * (n/2) + (n-1) GRANT * (n-1) /2 + (n-1) RELEASE *(n-1)/2) as presented in Fig. 3.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

Fig. 4 illustrated the best case for Dynamic Resource Synchronizer algorithm that includes three types of messages (1 REQUEST + 1 GRANT + 1 RELEASE) to enter and exit the CS. However, Fig. 5 shows the worst case for the algorithm and it is clear that when the numbers of nodes increases, the number of messages increases too, because the synchronizer may in the middle of the ring and all nodes wish to enter the CS. Thus, they send a REQUEST message to the synchronizer and each one of them will use the resource as much time as possible. Fig. 6 depicts the worst case for various numbers of nodes started from 50 nodes up to 10000 nodes. It is obvious from the Fig. that the complexity is n^2.

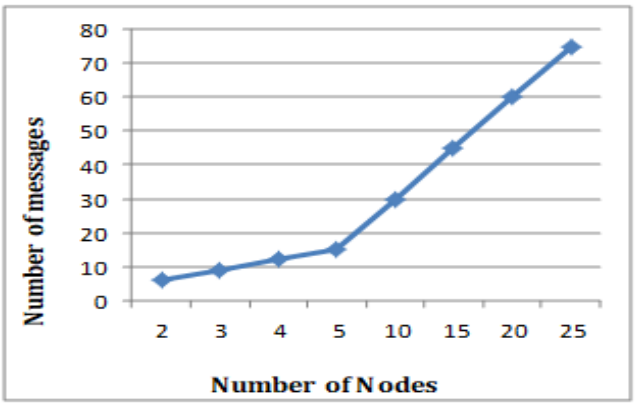

Fig 4: Best Case (Number of Messages to Enter and Exit the CS)

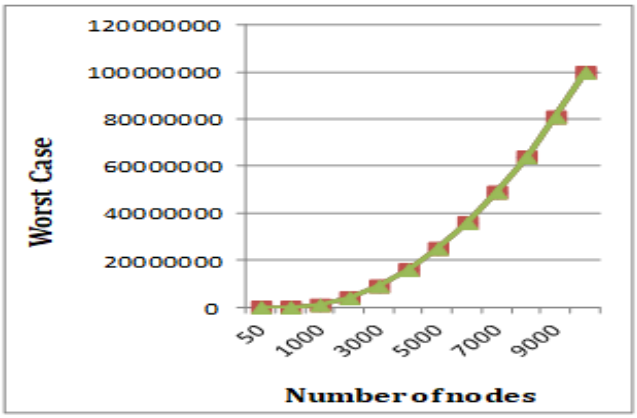

Fig 5: Worst Case (Number of Messages to Enter and Exit the CS)

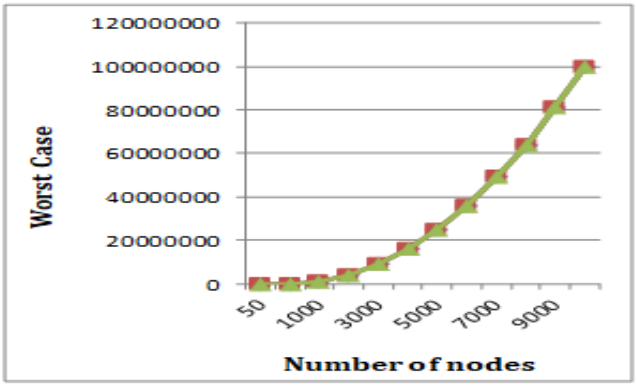

Fig 6: Worst case (Number of Messages to Enter and Exit the CS)

C. Experimental and Time Complexity

In this study, for each scenario we have run 10 experiments by using at most 25 devices which has the same specification; 8 GB RAM, Core I5 and 500 GB HD.

It is clear from Fig. 7 that the time complexity increases with increasing the number of nodes.

Indeed, the number of messages that generated per CS invocation is used to evaluate the performance of the distributed mutual exclusion algorithm. Thus, the best case performance occurs when the request node is a neighbor to the synchronizer and no node is waiting. In this case, the number of messages = 3 (1 REQUEST + 1 GRANT + 1 RELEASE) such as the first scenario, when node B sent a REQUEST message to node B which is a SYNCHRONIZER then it got GRANT message to enter the CS and finally sent a RELEASE message to release the CS. In addition, the waiting time in this case = 0. However, the worst case happens when the synchronizer in the far middle of the ring, the other nodes are waiting for their turns to enter the CS and each node wants to use the resource for the longest possible time. In this case the number of messages= 1 REQUEST * (n/2) + (n-1) (GRANT) * (n-1) /2 + (n-1) (RELEASE) * (n-1) /2 such as the scenario that presented in Section 4.1, when node B sent a REQUEST message to the SYNCHRONIZER (node A), then got GRANT message to enter the CS while other nodes (C, D and E) sent REQUEST message to the SYNCHRONIZER and wait for their turns to enter the CS.

Fig. 8 depicted the comparison between the experimental and mathematical study in term of number of messages under various numbers of nodes (at most 25 nodes). It is obvious from Fig. 8 that the number of messages in real experimental is less than the number of messages in mathematical one.

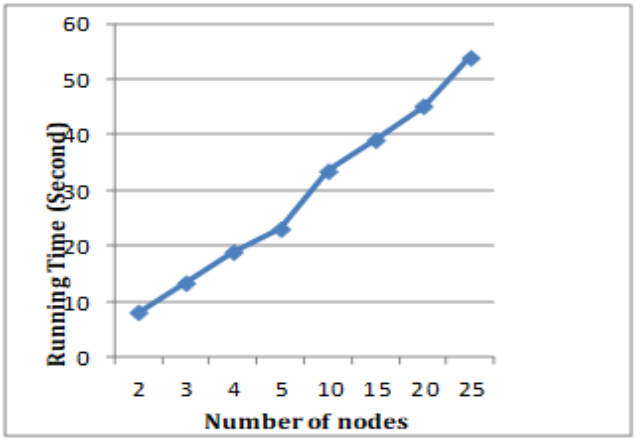

Fig. 7: Running Time under Various Numbers of Nodes.

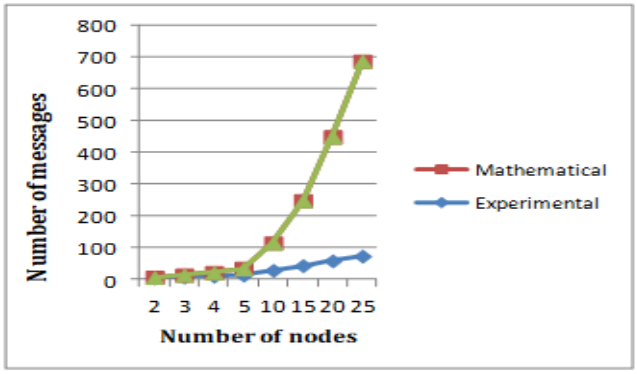

Fig. 8: Number of Messages under Various Numbers of Nodes

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

V. CONCLUSION

In this paper, we presented the implementation of the DRS algorithm which is a Fair Dynamic Resource Synchronizer Algorithm for Distributed Mutual Exclusion. Fairness is determined in terms of satisfying demands for CS access. In other words, the DRS algorithm proved that the mutual exclusion is satisfied. In addition; deadlock and starvation are impossible to occur. Thus, development mutual exclusion algorithm is appropriate for mobile computer systems that resort to reduce the number of messages needed to transmit between nodes and minimize the amount of storages that are needed at various sites. However, since institutional communication professions have been continually exploring effective measurement metrics for their communication initiatives, focusing on how communication practices can be effectively linked to improved performance [21-24], further research is required.

REFERENCES

1. Sharieh, A., Itriq, M., & Dbabat, W. (2008). A dynamic resource synchronizer mutual exclusion algorithm for wired/wireless distributed systems. American Journal of Applied Sciences, 5(7), 829-834.
2. Badrinath, B. R., Acharya, A., & Imielinski, T. (1994, June). Structuring distributed algorithms for mobile hosts. In Distributed Computing Systems, 1994., Proceedings of the 14th International Conference on (pp. 21-28). IEEE.
3. Liu, D., Liu, X., Qiu, Z., & Yan, G. (2003). A high efficiency Distributed Mutual Exclusion algorithm. In Advanced Parallel Processing Technologies (pp. 75-84). Springer Berlin Heidelberg.
4. Erciyes, K. (2004). Distributed mutual exclusion algorithms on a ring of clusters.In Computational Science and Its Applications–ICCSA 2004 (pp. 518-527). Springer Berlin Heidelberg.
5. Ricart, G., & Agrawala, A. K. (1981). An optimal algorithm for mutual exclusion in computer networks. Communications of the ACM, 24(1), 9-17.
6. Lejeune, J., Arantes, L., Sopena, J., & Sens, P. (2015). A fair starvation-free prioritized mutual exclusion algorithm for distributed systems. Journal of Parallel and Distributed Computing, 83, 13-29.
7. Tamhane, S. A., & Kumar, M. (2012). A token based distributed algorithm for supporting mutual exclusion in opportunistic networks. Pervasive and Mobile Computing, 8(5), 795-809.
8. Lodha, S., & Kshemkalyani, A. (2000). A fair distributed mutual exclusion algorithm. Parallel and Distributed Systems, IEEE Transactions on, 11(6), 537-549.
9. Ding, Z., Zhou, M., & Wang, S. (2014). Ordinary Differential Equation-Based Deadlock Detection. Systems, Man, and Cybernetics: Systems, IEEE Transactions on, 44(10), 1435-1454.
10. Du, Y., & Gu, N. (2015, December). Accelerating Reachability Analysis on Petri Net for Mutual Exclusion-Based Deadlock Detection. In 2015 Third International Symposium on Computing and Networking (CANDAR) (pp. 75-81). IEEE.
11. Lodha, S., & Kshemkalyani, A. (2000). A fair distributed mutual exclusion algorithm. IEEE Transactions on Parallel and Distributed Systems, 11(6), 537-549.
12. Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 21(7), 558-565.
13. Jani, K., & Kshemkalyani, A. D. (2004, December). Performance of fair distributed mutual exclusion algorithms. In International Workshop on Distributed Computing (pp. 2-15). Springer Berlin Heidelberg.
14. Kanrar, S., Chattopadhyay, S., & Chaki, N. (2013). A New Link Failure Resilient Priority Based Fair Mutual Exclusion Algorithm for Distributed Systems. Journal of network and systems management, 21(1), 1-24.
15. Lejeune, J., Arantes, L., Sopena, J., & Sens, P. (2015). A fair starvation-free prioritized mutual exclusion algorithm for distributed systems. Journal of Parallel and Distributed Computing, 83, 13-29.
16. Kundu, S. (2005, December). Deadlock-Free distributed relaxed mutual-exclusion without revoke-messages. In International Workshop on Distributed Computing (pp. 463-474). Springer Berlin Heidelberg.
17. Suzuki, I., & Kasami, T. (1985). A distributed mutual exclusion algorithm. ACM Transactions on Computer Systems (TOCS), 3(4), 344-349.
18. Singhal, M. (1989). A heuristically-aided algorithm for mutual exclusion in distributed systems. IEEE transactions on computers, 38(5), 651-662.
19. Raymond, K. (1989). A tree-based algorithm for distributed mutual exclusion. ACM Transactions on Computer Systems (TOCS), 7(1), 61-77.
20. Itriq, M., Dbabat, W., & Sharieh, P. (2013). Adaptive Dynamic Resource Synchronization Distributed Mutual Exclusion Algorithm (ADRS). Journal of Theoretical & Applied Information Technology, 53(3).
21. Altamony, H., Alshurideh, M., & Obeidat, B. (2012). Information Systems for Competitive Advantage: Implementation of an Organisational Strategic Management Process. Proceedings of the 18th IBIMA Conference on Innovation and Sustainable Economic Competitive Advantage: From Regional Development to World Economic, Istanbul, Turkey, 9th-10th May.
22. Alkalha, Z., Al-Zu'bi, Z., Al-Dmour, H., & Alshurideh, M. (2012). Investigating the effects of human resource policies on organizational performance: An empirical study on commercial banks operating in Jordan. European Journal of Economics, Finance and Administrative Sciences, 51, 44-64.
23. Masa'deh, R., Tayeh, M., & Al-Jarrah, I. M. (2015). Accounting vs. Market-based Measures of Firm Performance Related to Information Technology Investments. International Review of Social Sciences and Humanities, 9 (1), 129-145.
24. Shannak, R., Obeidat, B., & Almajali, D. (2010). Information Technology Investments: A Literature Review. Proceedings of the 14th IBIMA Conference on Global Business Transformation through Innovation and Knowledge Management: An Academic Perspective, Istanbul-Turkey, 23rd-24th June, pp.1356-1368.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)