Two Fault Tolerant Token Based Algorithms with Logical Ring for Mutual Exclusion in Distributed Systems

H. Maraghi[a], A. Parhizkari[a], A.T. Haghighat[a,b]
[a]Department of Electrical, Computer & IT, Islamic Azad University, Qazvin Branch, Qazvin, Iran
[b]Atomic Energy Organization of Iran (AEOI), NPPD, Tehran, Iran
I.R.IRAN

Abstract: - In this paper by using logical ring in network we have presented two algorithms for mutual exclusion in distributed systems. In the first algorithm the token always travels in the ring and each processes going to enter the critical section should wait for token. In the other algorithm the place of token is fixed until not one process is going to enter in critical section; and any process going to enter the critical section, by using a message searches the ring for the token. These algorithms are fault tolerant against crashing any process or losing token.

Key-Words: - Ring, Token, Process, Distributed System, Critical Section, Mutual Exclusion, Token Base Algorithm.

1 Introduction

Algorithms that are used for mutual exclusion should not at all let more than one process enter in critical section together. Furthermore in distributed systems the algorithm had better to be distributed and not centralized on special point. Some of these algorithms known as *token-based* algorithms which are based on using a token. In this kind of algorithms every process which have token can enter in the critical section. In this paper we studied token-based algorithms one of them *token ring algorithms* which are distributed. The mentioned algorithms face weak points which will be mentioned further. The presented algorithms, the previous problems are in solved.

2 Mutual Exclusion Algorithms

Distributed mutual exclusion algorithms can be divided into two classes: (1) *permission-based* algorithms, where all involved sites vote to select one which receives the permission to enter the critical section, and (2) *token-based* algorithms, in which only the site with the token may enter the critical section. In general, a permission-based algorithm involves higher communication traffic overhead than a token-based algorithm.

2.1 Raymond Algorithm

The Raymond algorithm [1] determines and maintains a *static* logical structure. The logical structure (for example, a spanning tree) is kept unchanged throughout its lifetime, but the directions of edges in the structure change dynamically as the token migrates among sites, in order to point toward the possible token holder. The directions of edges in the structure always point to the possible token holder, making the token holder a sink node in the structure.

Each site has a local queue to hold requests coming from its neighbors and itself, and has only one outstanding request at any given time, resulting in the local queue length no more than the node degree of the embedding structure.

Each site wishing to enter the critical section inserts its local request to the rear of its local queue, so that all requests appeared at that site in a first-come-first-served order. While it is possible to get better performance by inserting a locally generated request at the front of the local queue, referred to as the eager Raymond algorithm (because the local site is then allowed to enter the critical section immediately when the token reaches the site), this tends to pose a concern on the fairness of requests and is not considered here.

2.2 Modified Raymond algorithm

In the Raymond Algorithm described above, a token request always follows the token from an intermediate site whose local queue contains more than one element. This situation happens more frequently as the critical section request rate grows. We introduce a simple modification to lower communication traffic by eliminating the token request from a site whose local queue contains multiple elements. Instead of sending a separate token request, the site marks in the token message the situation that the token has to come back later on. A marked token causes an enqueuein operation at the receiving site, recording that the token will be sent back along the link from which it gets to the site. This combines the token message with a subsequent token request message at every site whose local queue length is greater than 1, effectively lowering mutual exclusion traffic and thus improving performance.

2.3 Star Algorithm

Instead of passing the token step by step through intermediate sites in the logic structure to the token requestor as in the Raymond algorithm [1], Neilsen and Mizuno proposed an algorithm where the token holder can send the token directly to the requesting site with one message [2]. This is made possible by attaching the requestor's ID in the request message so that the token holder knows, on receiving the message, who is the requestor.

One special case of this algorithm is that the logical structure can be a fixed star topology (called the Star algorithm). Under such a situation, any site ready to enter the critical section always sends a request message attached with its own ID directly to the root node. The root node makes it possible to establish a distributed waiting queue (of all requesting sites) by recording the site which has most recently requested the token (and is the tail site in the distributed waiting queue). When receiving a request message, the root forwards the message to the tail site (of the queue) and updates its record, unless the root itself holds the token. On receiving a request message, the token holder, if not in need of the token, forwards the privilege to the requestor directly using a token message. A very attractive property of the Star algorithm is that it always takes three (3) exchange messages for a requestor to get the token, if the root does not own the token, and only two (2) messages if the root holds the token.

2.4 CSL Algorithm

Chang, Singhal, and Liu's algorithm [5] maintains a list which links all requesting sites (i.e., a distributed queue), such that each requesting site records (using variable **Next**) only the identifier of its next requesting site, thereby simplifying the data structure of token message [5]. The logical structure in the CSL algorithm is a star topology initially, and it changes dynamically as the algorithm proceeds. A site is the tail in the distributed queue, if it is waiting for the token and its **Next** is **NIL**. If its **Next** is not **NIL**, its successor site in the distributed queue is pointed by **Next**. As a result, when a request message arrives at a site which is the tail in the distributed queue, the site simply sets its **Next** to the requesting site. If a request message arrives at a site which neither holds nor is requesting the token, or which is requesting the token but its **Next** is not **NIL**, the request message is forwarded to the possible token holder (pointed by variable **NewRoot**) to form a distributed queue; **NewRoot** is then set to point to the current requestor because it will eventually becomes the new token holder. On sending the token message to the 'next' site, the variable **NewRoot** is piggybacked in the token message so that the 'next' site can update its **NewRoot** accordingly.

The NME complexity of this algorithm depends on the height of the logical tree, and it is $O(\log N)$ per critical section entry, where N is the system size. Because its logical structure changes dynamically, the CSL algorithm fails to exhibit as good performance as the Star algorithm, where the structure is kept unchanged.

2.5 Token Ring Algorithm

A approach to achieving mutual exclusion in distributed system is illustrated in Fig.1. We have a network of processing. A logical ring is constructed in which each process is assigned a position in the ring, as shown in Fig.1.

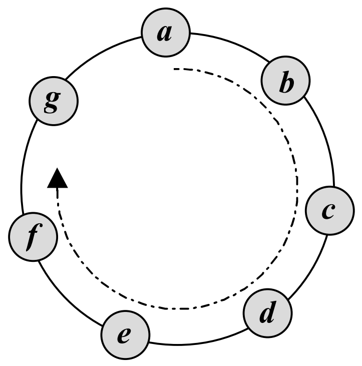


Fig.1. a logical ring

When the ring is initialized, process ***a*** given a token. The token circulates around the ring. It is passed from a process to next process (from ***a*** to ***b***, from ***b*** to ***c*** ,…) in point-to-point messages. When a process acquires the token from its neighbor, it checks to see if it is attempting to enter a critical section. If so, the process enters the section, does all the work it needs to, and leaves the section. After it has exited, it passes the token along the ring. It is not permitted to enter a second critical section using the same token. If a process is handed the token by its neighbor and is not interested in entering a critical section, it just passes it along. When no processes want to enter any critical section, the token circulates at high speed around the ring.

Token ring algorithm's problems:

1. If the token is ever lost, it must be regenerated. In fact, detecting that it is lost is difficult, since the amount of time between successive appearances of token on the network is unbounded. The fact that the token has not been spotted for a hour does not mean that it has been lost; somebody may still be using it.
2. The algorithm also runs into trouble if a processor crashes. If we require a process receiving the token to acknowledge receipt, a dead process will be detected when its neighbor tries to give it the token and fails.

2.6 Fixed Token Ring Algorithm

The algorithm proposed in [3,4] establishes a static logical ring over all sites and allows the token to move along a fixed direction, in response to a token request traveling along an opposite direction. The logic ring and the direction of its links are all kept unchanged. When ready to enter the critical section, a site without the token, say S_w, must send a request messages to its successor, site $S_{(w+1) \bmod N}$, and then goes to WAIT state until it receives the token, where N is the system size. If $S_{(w+1) \bmod N}$ is not the token holder, it sends a request message to its successor, site $S_{(w+2) \bmod N}$. This process repeats until the site with the token, say S_h, receives a request message from its predecessor. All sites within S_w and S_h (along the direction of the request message traversals) are all at the SUBS (short for substitute) state. On receiving a request message, the token holder, if not in need of the token, forwards the privilege to its predecessor using a token message. The token is then forwarded by the SUBS sites in sequence to site S_w (along the reverse direction of the request message). If the number of SUBS sites is α, $0 \le \alpha \le N-1$, the total number of messages exchanged for S_w to get the privilege of entering the critical section equals $2\times(\alpha + 1)$.

3 Fault Tolerant Token Ring Algorithm

As we mentioned before token ring algorithm has some problems that are related to loss of token. When a process is going to enter the critical section, waits for the token; if it does not receive the token for a long time, cannot distinguish whether there is another process in critical section or it has crashed or the token has been lost. To distinguish whether there is still another process in critical section or another problem has happened we change the algorithm as below.

In new algorithm like former algorithm there is a token at first which swirl in the ring and the process that is going to enter the critical section waits for it and then enters the critical section as soon as receiving the token. In old algorithm the process that entered the critical section kept the token. But in new algorithm the process creates a new token that is called second type token, now this token travels in the ring. If this process is going to exit from the critical section waits for the second type token and after receiving and discarding it will release the first type token.

By the change in the algorithm it can distinguish whether really there is any process on going in the critical section or the token has been lost or the process in the critical section has crashed.

Suppose the process ***A*** is in critical section and process ***B*** is going to enter the critical section. At this time process ***B*** sets a timer and waits for the token. If it receives before timeout the second type token finds out that there is another process in critical section. But if in this duration it would not receive the token it finds out that there is a problem and send a message in ring. This message means that process ***B*** is going to enter critical section and has not received the token. Now if the process that is in critical section (process ***A***) receive this message find out that the second type token has been lost so it would get the message and send the second type token. Then process ***B*** by receiving the token will find out that another process is in critical section and the second type token has been lost. But if process ***B*** would receive its own message it finds out that another process is in critical section or the process in critical section had crashed. In both cases it can enter in critical section and after entering, it creates a second type token and sends in the ring.

So by using this new algorithm in case of losing token or crashing of any process, there would be no problem in system and this algorithm would be a fault tolerant against those problems.

4 Fault Tolerant Fixed Token Ring Algorithm

In the algorithm we presented in the last section, there is a token which always swirl in the ring and each process which is going to enter the critical section waits for the first type token and enters in the critical section after receiving the token. In this algorithm the problem is that the token always swirl in the ring.

Another algorithm pointed in 2.6 that is run in a physical ring network will have problem in case of crashing a process or losing the token. We can run this algorithm in a logical ring and solve the problems by some changes that is presented here.

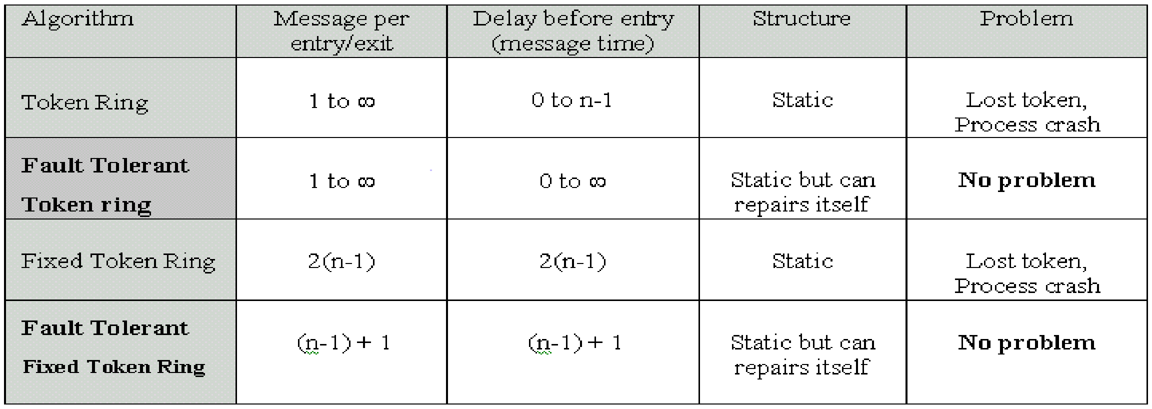
In this algorithm the token is located in a process at the first. Suppose a process not having the token and is going to enter the critical section, it sends a message in the ring. This message travels in the ring until going to the process which it has the token. If this process is not in critical section and not going to enter in it, sends the token to the process which it has requested the token. Other wise it sends the message, which it is in critical section and will send the token after exiting from critical section.

As commented before the problem in this algorithm is crashing one of the processes or losing the token. In case of losing the token the message that the requesting process sends will go to itself, which means none of the ring processes have the token. To solve the problem the requesting process should create a token and remain and then enter the critical section. But in case of crashing one of the processes the requesting process never receives the message, so if it would not receive the message after a certain time, it means that one of the processes has crashed, and by sending the repairing message, the nodes will repair the ring then after repairing the ring the requesting process will repeat the request. By these changes the algorithm will be fault tolerant against the problems.

5 Conclusion

In this paper, we presented two solutions for distributed mutual exclusion that use ring algorithms. We improved the algorithms for fault tolerance. The Comparison of these algorithms is presented in Table 1.

Table 1. Comparison of algorithms



Algorithm	Message per entry/exit	Delay before entry (message time)	Structure	Problem
Token Ring	1 to ∞	0 to n-1	Static	Lost token, Process crash
Fault Tolerant Token ring	1 to ∞	0 to ∞	Static but can repairs itself	**No problem**
Fixed Token Ring	2(n-1)	2(n-1)	Static	Lost token, Process crash
Fault Tolerant Fixed Token Ring	(n-1) + 1	(n-1) + 1	Static but can repairs itself	**No problem**

References:

[1] K. Raymond, "A Tree Based Algorithm for Distributed Mutual Exclusion," *ACM Trans. on Computer Systems*, vol.7, No. 1, pp. 61–77, February 1989.

[2] M. L. Neilsen and M. Mizuno, "A Dag-Based Algorithm for Distributed Mutual Exclusion," *Proc. 11th Int. Conf. Distributed Computer Systems,* pp. 354–360, May 1991.

[3] A. J. Martin, "Distributed Mutual Exclusion on a Ring of Processes," *Science of Computer Programming*, vol. 5, pp. 265–276, February 1985.

[4] S. S. Fu and N.-F. Tzeng, "Efficient Token-Based Approach to Mutual Exclusion in Distributed Memory Systems," Tech. Rep. TR-95-8-1, CACS, Univ. Southwestern Louisiana, Lafayette, LA 70504, July 1995.

[5] Y. I. Chang, M. Singhal, and M. T. Liu, "An Improved $O(\log N)$ Mutual Exclusion Algorithm for Distributed Systems," *Proc. 1990 Int. Conf. Parallel Processing,* vol. III, pp. 295–302, August 1990.

[6] A. S. Tanenbaum, " Distributed Systems", Prentice Hall, 2002.

[7] Shiwa S.Fu, Nian-Feng Tzeng, and Zhiyuan Li, "Empirical Evaluation of Distributed Mutual Exclusion Algorithms," Institute of Electrical and Electronics Engineers,1997.