Token Ring Algorithm To Achieve Mutual Exclusion In Distributed System – A Centralized Approach

Sandipan Basu
Post Graduate Department of Computer Science, St. Xavier's College, University of Calcutta
Kolkata-700016, INDIA

Abstract
This paper presents an algorithm for achieving mutual exclusion in Distributed System. The proposed algorithm is a betterment of the already existing Token Ring Algorithm, used to handle mutual exclusion in Distributed system. In the already existing algorithm, there are few problems, which, if occur during process execution, then the distributed system will not be able to ensure mutual exclusion among the processes and consequently unwanted situations may arise. The proposed algorithm will overcome all the problems in the existing algorithm, and ensures mutual exclusion among the processes, when they want to execute in their critical section of code.

Keywords: *TIMESTAMP_OF_REQUEST_GENERATION, PID, NEW, REQUEST_QUEUE, EXISTS, COORDINATOR, UPDATE.*

1. Introduction

In most of the distributed systems it is very common that, resources are being shared among various processes, with the condition that a single resource can be allocated to a single process at a time. Therefore, mutual exclusion is a fundamental problem in any distributed computing system. So, the goal is to find a solution that will synchronize the access among shared resources in order to maintain their consistency and integrity.

In this paper, the proposed algorithm is able to handle the problems of mutual exclusion in a distributed system. It is also able to handle all other problems that may arise, while a process is executing in its critical section.

2. Existing Work

Till now, several token-based algorithms have been proposed. Some of them are –

1. Ricart - Agrawala algorithm.
2. Suzuki - Kazami algorithm.
3. Mizuno - Neilsen - Rao algorithm.
4. Neilson – Mizuno algorithm.
5. Helary – Plouzeau – Raynal algorithm.
6. Raymon'd algorithm.
7. Singhal's algorithm.
8. Maimi – Trehel algorithm.
9. Misra- Srimani algorithm.
10. Nishio – Li – Manning algorithm.

3. Preconditions

The effectiveness of an algorithm depends on the validity of the assumptions that are made. In distributed mutual exclusion environment certain assumptions must be considered to make the work successful. The following assumptions are made for this algorithm:-

1. All nodes in the system are assigned unique identification numbers from 1 to N.
2. There is only one requesting process executing at each node. Mutual exclusion is implemented at the node level.
3. Processes are competing for a single resource.
4. At any time, each process initiates at most one outstanding request for mutual exclusion.
5. All the nodes in the system are fully connected.

6. A process can enter only one critical section when it receives the token. If it wants to enter another critical section, the process must send another request to the coordinator.

7. Instead of using globally synchronized physical clocks, Lamport's concept of logical clock is used in distributed system that we are considering. The concept of logical clock is the way to associate a timestamp (simply a number independent of any clock time) with each system event so that events that are related to each other by the ***happened-before*** relation (directly or indirectly) can be properly ordered in that sequence. To ensure that all events that occur in a distributed system can be totally ordered, the use of **any arbitrary total ordering of events**, proposed by Lamport, is applied in the distributed system considered in this algorithm. For instance, if events a and b happen respectively in processes P1 and P2, and both events will have a timestamp of (say) 40, then according to Lamport's proposal the timestamp associated with events a and b will be 40.1 and 40.2 respectively, where the process identity number (PID) of processes P1 and P2 are 1 and 2 respectively. Using this technique, we will be able to assign unique timestamp to each event in a distributed system to provide a total ordering of all events in the system.

As we are considering distributed systems, some assumptions also need to make about the communications network. This is very important because nodes communicate only by exchanging messages between them. The following aspects about the reliability of the distributed communications network should be considered.

1. Messages are not lost or altered and are correctly delivered to their destination in a finite amount of time.

2. Messages reach their destination in a finite amount of time, but the time of arrival is variable.

3. Nodes know the physical layout of all nodes in the system and know the path to reach each other.

4. Algorithm

In this algorithm, we consider that a set of processes are logically organized in a ring structure. Between several processes, one process acts as a coordinator. It is the coordinator's task to generate a token and circulate the token around the ring, as needed. In this algorithm, we consider that the token can move in any direction as per the necessity. In the ring structure, every process maintains the current ring configuration of the system. If a process is removed or added into the system, then the updating must be reflected to all the processes' ring configuration table. A ring configuration table is something that contains information about process id (PID), state of the processes and their status in the current system.

The algorithm works as follows: -
Consider, the processes are numbered as P1, P2, P3…Pn. For simplicity of our discussion, consider process P1 wants to enter in its critical section. P1 will send a request [REQUEST (PID, TIMESTAMP_OF_REQUEST_GENERATION)] to the coordinator to acquire the token. A REQUEST message contains two parameters- a) PID (i.e. process id of the requesting process), b) Time Stamp of request generation. Here, two cases may appear:

Case 1: -

At this particular time, if no other process is executing in its critical section, and the coordinator retains the token, then the coordinator will send the token to the requesting process (P1). After acquiring the token, the process keeps the token and enters in its critical section. After exiting from the critical section the process returns the token to the coordinator.

Case 2:-

But, if some other process P_i (i != 1) is executing in its critical section, then the coordinator sends a WAIT signal, to the requesting process (P1) and the request from process P1 is stored in the REQUEST QUEUE, maintained by the coordinator only. In between, if any other processes want to enter into critical section and send request to the coordinator, then those requests will also be stored in the REQUEST QUEUE. When the coordinator gets back the token, then it sends the token to one of the waiting process in the REQUEST_QUEUE, which has smallest TIMESTAMP_OF_REQUEST_GENERATION.

The algorithm will overcome the drawbacks of the general token ring algorithm, in the following manner.

Loss of Token:-

When a process (say) P1 wants to enter into its critical section, it sends request to the coordinator. If the coordinator retains the token, it then sends the token to the requesting process (P1). After getting the token, the process will send an acknowledgment to the coordinator, and enters the critical section. During the execution of the critical section, P1 will continually send an EXISTS signal to the coordinator at certain time interval, so that, the coordinator becomes acquainted that the token is alive and it has not lost. As a reply to every EXISTS signal, the coordinator sends back an OK signal to that particular process (P1), so that the process that is executing in its critical section (P1), gets to know that the coordinator is alive also.

Now, suppose, the coordinator is not receiving the EXISTS signal from that process P1. Here two cases may appear:-

Case 1:-

The coordinator assumes that the token has lost. Then the coordinator will regenerate a new token and sends it to that process (P1) and again it starts executing its critical section.

Case 2:-

The process P1 may crash or fail while executing in it critical section and consequently, the coordinator does not receiving any EXISTS signal from P1. Hence, the coordinator will identify it as a crashed process and update the ring configuration table and send the UPDATE signal with update information to other processes to update their own ring configuration tables.

Again it may be the case, that the process P1 (which is currently executing in its critical section), is not receiving the OK signal form the coordinator. So, P1 would assume that the coordinator is somehow crashed. At this moment, the process P1 will become the new coordinator and complete the critical section execution. The new coordinator will send a message [COORDINATOR (PID)] to every other process, that it becomes the new coordinator and send the UPDATE signal with update information, to update the ring configuration tables maintained by all other processes.

The algorithm also overcomes the overhead of token circulation in the ring. If no processes in the ring want to enter in its critical section, then there is no meaning of circulating the token throughout the ring. Rather, in this approach, the coordinator will keep the token, until any other process requests it.

- The algorithm guarantees mutual exclusion, because at any instance of time only one process can hold the token and can enter into the critical section.

- Since a process is permitted to enter one critical section at a time, starvation cannot occur.

5. Performance Analysis

The performance of the algorithm will be evaluated in terms of the total number of messages required for a node to enter into a critical section. The number of messages exchanged for an entry into a critical section to take effect will be used as a complexity measure. In this algorithm the number of messages per critical section entry varies from 3 (when the coordinator possesses the token and no other process is executing in its critical section. REQUEST->GRANT->RELEASE) to 4 (when some other process is already executing in its critical section. REQUEST->WAIT->GRANT->RELEASE).

For a total no. of n processes in the ring, the waiting time from the moment a process wants to enter a critical section until its actual entry, may vary from 0 to n-1; 0, in the case, when a process wants to enter a critical section and acquires the token immediately form the coordinator; (n-1), in the case, when the process sends the request after all other process has already requested for the token.

6. Illustration

The above algorithm can best be explained by an example.

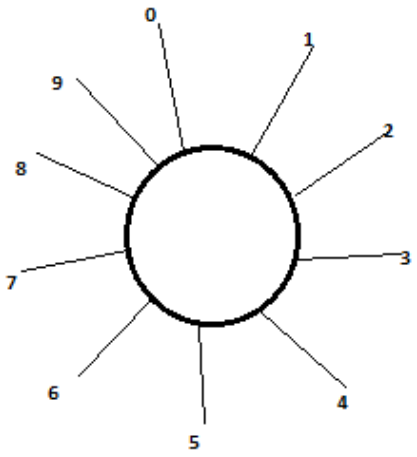

Figure 1: Ring Configuration

Consider a distributed system consists of 10 processes (P0-P9). Between these processes, one process is selected as a coordinator. Suppose when the ring is initialized, process P0 is selected as the coordinator. As soon as P0 is selected as the coordinator, it generates a token and retains the token. Now, process P5 wants to enter in its critical section. So, P5 sends a request [REQUEST (PID, TIMESTAMP_OF_REQUEST_GENERATION)] to the coordinator (P0).

Table 1: Ring Configuration Table

Process ID	***State***	***Status***
P0	Alive	**Coordinator**
P1	Alive	Normal
P2	Alive	Normal
P3	Alive	Normal
P4	Alive	Normal
P5	Alive	Normal
P6	Alive	Normal
P7	Alive	Normal
P8	Alive	Normal
P9	Alive	Normal

Now, if no other process is executing in its critical section, and the token has been kept by P0, then immediately the token is send to process P5. After completing the execution of its critical section, process P5 releases the token and gives it back to the coordinator.

In this case, the situation may appear that, while process P5 is executing in its critical section, process P7 wants to enter its critical section and sends a request to the coordinator[REQUEST(PID,TIMESTAMP_OF_REQUEST_GENERATION.)]. As process P5 possesses the token and executing in its critical section, the coordinator sends a WAIT signal to process P7, and stores the request in the REQUEST_QUEUE. Now suppose, immediately after, process P2 also wants to enter in its critical section, and sends a request [REQUEST (PID, TIMESTAMP_OF_REQUEST_GENERATION] to the coordinator. As process P5 is still executing in its critical section, so the coordinator sends a WAIT signal to process P2, and stores the request in the REQUEST_QUEUE. After process P5 has exited from its critical section, it releases the token and sends it back to the coordinator. Then the coordinator selects the process with smallest TIMESTAMP_OF_REQUEST_GENERATION and sends the token to the corresponding process.

	P7	**P2**	

Figure 2: REQUEST QUEUE

One thing that we have to keep in mind, while a process is executing in is critical section, during that, with a certain time interval the process continually sends an EXISTS signal to the coordinator, to indicate that the token is alive. In reply of each EXISTS signal, the coordinator sends an OK signal to that process, indicating, the coordinator is alive also.

In the previous case, when process P5 is executing in its critical section and the processes P7 and P2 (or any other processes), send request to the coordinator and wait to acquire the token, then it could happen that the coordinator might crash. After this incident, when process P5 sends the EXISTS signal but and does not receive the OK signal within a fixed timeout period, then process P5 assumes that the coordinator has crashed, hence it becomes the new coordinator, and process P5 announces this by sending a message [COORDINATOR (PID)] to the rest of the alive processes in the ring. Then all the other processes update their ring configuration table to the current state of the ring.

Table 2: Ring Configuration Table

Process ID	***State***	***Status***
P0	Dead	***Unknown***
P1	Alive	Normal
P2	Alive	Normal
P3	Alive	Normal
P4	Alive	Normal
P5	Alive	**Coordinator**
P6	Alive	Normal
P7	Alive	Normal
P8	Alive	Normal
P9	Alive	Normal

Consequently, each process including P7 and P2 get to know that the process P5 became the new coordinator. As processes P7 and P2 are still not being able to enter their critical section, they both send requests to the new coordinator P5, to acquire the token. Now, there is a very significant point to notice that, previously P7 was the first one to sends its request and then P2 sends its request. But, when, again they send their requests to the coordinator, it may happen, that the request from process P2 reaches to the coordinator prior process P7's request. In this situation one may think that, as the request form process P2 reaches first to the coordinator (P5), so, the coordinator may send the token to process P2 and not process P7.But, it is not the case. As previously mentioned, when a coordinator selects a process from its REQUEST QUEUE, (when multiple requests are in the queue) it always makes the selection based on the smallest 'time stamp of request generation'. So accordingly, process P7 will get the chance to acquire the token.

After a while, it may be the case, process P0 has restarted. Then it will send a message NEW to every other process in the ring. Hence, every other process will update their corresponding ring configuration table. In this situation the present coordinator gets to know that a new entry has been done. As a result the coordinator will send a message [COORDINATOR (PID)] and also send the current ring configuration table to the revived process P0; so that revived process (P0) gets to know who the current coordinator is and also maintains the ring configuration table.

Another situation may appear, that when no other processes are executing in its critical section, then somehow the coordinator has crashed. In this case, the process that will notice it first, it will be the new coordinator in the ring and announces it by sending the [COORDINATOR (PID)] message to every other process and consequently every other process updates their ring configuration table.

7. Conclusions

In this paper, the proposed algorithm does not allow the circulation of the token along the ring, when there is no need (i.e. when no process wants to enter in its critical section). Loss of a token in the ring can easily be detected, and regeneration of token can be done easily in this algorithm. And process crash and recovery of crashed process can easily be managed using this algorithm. And there is no chance of creation of duplicate tokens in the ring.

Hence, the proposed algorithm overcomes all the drawbacks that may appear in the existing Token Ring Algorithm for handling Mutual Exclusion in Distributed System.

References

[1] Andrew S. Tanenbaum, Distributed Operating System, Pearson Education, 2007.
[2] Pradeep K. Sinha, Distributed Operating Systems Concepts and Design, Prentice-Hall of India private Limited, 2008.
[3] H.Attiya and J.Welch, Distributed Computing Fundamentals, Simulation and Advanced Topics, Second Edition, A John Wiley & Sons, Inc., Publication, 2004.
[4]Martin G. Velazquez, "A survey of Distributed Mutual Exclusion Algorithms", Department of Computer Science, Colorado State University.
[5].Ajay D. Kshemkalyani and Mukesh Singhal, Distributed Computing principles, Algorithms, and Systems, Cambridge University Press,2008.

Sandipan Basu is final year student of M.Sc. Computer Science, St. Xavier's College, University of Calcutta. He completed B.Sc (Honours) degree in Computer Science from Asutosh College, University of Calcutta. His research interests include Distributed Systems, Networking, Operating System and Cryptography.