Abstract - In a distributed system, mutual exclusion is a condition which ensures that a shared resource in the system can be accessed by one and only one process at a time. Mutual exclusion ensures that only one process can be in its critical section at a time. Many distributed mutual exclusion algorithms have been proposed and verified [1][2][3][4][5]. In this paper, we have modeled and verified the dynamic information-structure mutual exclusion algorithm for distributed systems [1] using SPIN as the model checker. The algorithm is modeled in the Promela language which is interpreted using SPIN and the sequence diagrams corresponding to the model generated by SPIN are used to verify the correctness of the model.

Keywords – Promela; Spin Checker

I. INTRODUCTION

In the systems involving multiple processes, when a process has to read or update certain shared data structures, it first enters a critical region to achieve mutual exclusion and ensures that no other process will use the shared data structures at the same time. The mutual exclusion problem states that only a single process can be allowed access to a protected resource, also termed as a critical section (CS), at any time. Mutual exclusion is a form of synchronization and is one of the most fundamental paradigms in computing systems.

A good distributed mutual exclusion algorithm must meet following requirements:

1. Safety: At any instant, only one process can execute the critical section.

2. Progress: This property states the absence of deadlock and starvation. Two or more sites should not endlessly wait for messages which will never arrive. In addition, a site must not wait indefinitely to execute the CS while other sites are repeatedly executing the CS. That is, every requesting site should get an opportunity to execute the CS in finite time.

3. Fairness: Each process gets a fair chance to execute the CS. Fairness property generally means the CS execution requests are executed in the order of their arrival (time is determined by a logical clock) in the system.

In this paper, we have discussed the Singhal’s Dynamic Information Structure Distributed Mutual Exclusion Algorithm and its modeling in the ProMeLa and verified using the SPIN model checker.

II. THE SPIN MODEL CHECKER

The Spin model checker [6] is a powerful, but lightweight analysis tool that has been used to model and verify both hardware and software systems. Originally developed to model computer and network protocols, the adoption of Spin has found its way into many application domains [6][7][8][9][10][11][12][13]. One of the novelties of Spin is its relatively simple specification language, Promela. [7]

SPIN [6] is a tool for analyzing the logical consistency of concurrent systems, specifically of data communication protocols. The system is described in a modeling language called PROMELA (Process Meta Language). PROMELA is a verification modeling language. It provides a vehicle for making abstractions of protocols (or distributed systems in general) that suppress details that are unrelated to process interaction. It also allows for the dynamic creation of concurrent processes. Communication via message channels can be defined to be synchronous (i.e., rendezvous), asynchronous (i.e., buffered). Given a model system specified in Promela, Spin performs random simulations of the system's execution. During simulation and verification Spin checks for the absence of deadlocks, unspecified receptions, and unexecutable code. The verifier can also be used to verify the correctness of system invariants, it can find non-progress execution cycles, and it can verify correctness properties expressed in next-time free linear temporal logic formulae.
The SPIN model checker is a system that has been used to model and analyze a large number of applications in several domains including the aerospace industry. One of the novelties of Spin is its relatively simple specification language, Promela, as well as the powerful abilities of the model checker [6].

In particular, conventional static- and dynamic-testing techniques quickly break down if they are applied to distributed systems software. The bugs in such systems are usually triggered by irreproducible event sequences that can make debugging a nightmare. Tools such as the SPIN model checker, however, can help build reliable systems.

III. THE DYNAMIC INFORMATION STRUCTURE DISTRIBUTED MUTUAL EXCLUSION ALGORITHM [1]

In the distributed mutual exclusion algorithm [1] that has been modeled and verified in this paper, every node maintains an information structure which changes dynamically as the inter-node communication takes place.

A. Algorithm Description

Information structure is maintained at every site. This structure specifies the sites to which request message is needed to be send before CS execution. It also specifies the sites to which reply messages are needed to be sent after CS execution.

B. Data Structure

Information structure at a site $S_i$ consists of request set and inform set. Request set or $R_i$ contains sites from which permission is to be acquired before executing CS. Inform set or $I_i$ contains sites to which permission is needed to be send after CS execution is finished. A Logical clock $C_i$ is maintained for every site which gets updated according to Lamport’s rule. A timestamp is assigned to every request. Lower is the timestamp higher is the priority. The state of the site is denoted by three Boolean variables – Requesting, Executing and My_priority. Requesting and Executing are true iff site is requesting or executing respectively. My_priority is true if pending request has higher priority over the current incoming request.

C. Initialization

Initial state Site $S_i$ ( $i=1$ to $n$ ),

$R_i = \{ S_1, S_2, \ldots, S_n \}$

$I_i = \{ S_i \}$

$C_i = 0$

Requesting = Executing = False

Arranging sites $S_n$ to $S_1$ from left to right initial system state follows two properties:

1. Permissions are requested from all sites on the right and none from the left.
2. A staircase pattern is achieved as the cardinality of $R_i$ decreases in stepwise manner.

It satisfies Mutual Exclusion as whenever two sites request for CS, one of them will always ask the permission of the other site. It also avoids deadlock as a unique timestamp ordering is used to decide request priority and only a higher priority request blocks another request.

D. Algorithm

Site $S_i$ executes the following three steps to invoke mutual exclusion:

Step 1: (Request Critical Section)

Requesting = true;

$C_i = C_i + 1$;

Send REQUEST $(C_i, i)$ message to all the sites in $R_i$;

Wait until $R_i = \Phi$;

Requesting = false;

Step 2: (Execute Critical Section)

Executing = true;

Execute CS;

Executing = false;

Step 3: (Release Critical Section)

For every site $S_k$, in $I_i$ (except $S_i$) do

begin

$I_i = I_i - \{ S_k \}$;

send REPLY$(C_i, i)$ message to $S_k$;

$R_i = R_i + \{ S_k \}$;

end;

REQUEST Message Handler

The REQUEST message handler at a site processes incoming REQUEST messages. It takes actions such as updating information structure and sending REQUEST/REPLY messages to other sites. The REQUEST message handler at site $S_i$ is given below:

/* Site $S_i$ is handling message REQUEST (c,j) */

$C_i = \max \{ C_i, c \}$;

Case

Requesting = true: begin if My_priority then $I_i = I_i + \{ S_j \}$

else

send REPLY$(C_i, i)$ message to $S_j$;

if not $(S_j \in R_i)$ then

begin

$R_i = R_i + \{ S_j \}$;

send REQUEST$(C_i, i)$ message to $S_j$;

end;

end;
begin;
Executing = True:  I_j = I_j + [S_j];
Executing = False □ Requesting = False:
begin
R_i = R_i + [S_j];
send REPLY(C_i, j) message to S_j;
end;
REPLY Message Handler
The REPLY message handler at a site processes incoming REPLY messages. It updates the information-structure. REPLY message handler at site S_i is given below:
/* Site S_i is handling a message REPLY (c, j) */
begin
    C_i = max{ C_i, c};
    R_i = R_i - [S_j];
end;

Note that REQUEST/REPLY message handlers and the steps of the algorithm access shared data structures, viz., C_i , R_i and I_i. To guarantee the correctness, we require that execution of REQUEST/REPLY message handlers and all three steps of the algorithm (except “wait for R_i = ɸ to hold” at step 1) mutually exclude each other.

E. Explanation
Step 1: S_i requests all sites in the request set R_i.
Step 2: S_i executes CS.
Step 3: S_i releases CS by sending REPLY messages to all sites in the inform set I_i.

If a higher priority REQUEST message arrives at a site S_j from another site S_i then it sends a REPLY message to S_j and includes it in R_j if S_j not in R_i . Otherwise S_i places it into I_i . S_i puts S_j in I_i if S_j receives a REQUEST message from S_i during the CS execution. This means that it can sent a REPLY message when it has finished the execution of CS. S_i puts S_j in R_i and sends a REPLY message to S_j if S_j receives a REQUEST message from S_i when it is neither requesting nor executing the CS.

F. Modeling in Promela
The algorithm has been specified in the Promela language in following manner:-

For every node, there are three processes, the main process to execute and request the critical section (CS), the request handler (RqH) and the reply handler (RpH).

Whenever a node (suppose node 1) wants to request critical section, the CS process of a node sends a REQUEST message to Request Message Handler of node 2. The REQUEST message is shown in fig 1.

If Requesting = true and My_priority = true then node 2 adds node 1 into its inform set.

If Requesting = true, My_priority = False and node 1 is in request set R_2 then it sends a REPLY message to the Request Message Handler of node 1. The same is the case, when the other node (node 2 in this case) receiving the request is neither requesting nor executing i.e has its boolean Requesting = False and Executing = False. This case is shown in fig 2.

If node 2 is requesting i.e Requesting = true , My_priority = False and node 1 is not in request set R_2, then the Request handler of node 2 sends a REQUEST message to the Request Message Handler along with the REPLY message to the Reply Message Handler of node 1. This case is shown in fig 3.
Reply message handler, on receiving the reply from a node, removes the corresponding node entry from the request set and when the request set becomes empty, the waiting CS process in that node is activated to enter into CS. This case is shown in fig 4.

IV. RESULTS

The proposed model of the Singhal’s Distributed Mutual Algorithm under study, when verified using SPIN model checker produces the snapshots in figure 5. As shown in the snapshots, the first line resembles the process of initialization of the nodes. Each node constitute three processes i.e the main process to execute and request the critical section (CS), the request handler (RqH) and the reply handler (RpH), which are indicated respectively by three consecutive lines. The included snapshots are taken for the simulation of the algorithm for only 3 nodes but the algorithm has been verified for the large systems consisting of 100 nodes and it also produces successful simulation results.
Figure 5. Snapshots of the simulation run for a system of 3 nodes.
This algorithm has been checked for two different simulation styles provided by the spin model checker (version 5.1.7 and above).

Figure 6.

These options or simulation styles are:

1. The random simulation in which the SPIN executes the statements (that can be executed at any given point of time according to the model) for the concurrent processes randomly according to a seed value. This seed value is set as 1 by default but can be changed. The seed value changes the sequence of instructions in which they are executed and thus changes the output every time the seed value is changed. Thus, by changing the seed value every time, the sequence in which the request and reply messages are sent are changed and thus the simulation for a different scenario occurs every time. For all the seed values from 1 to 100, the model for the algorithm under study i.e. the Singhal distributed mutual exclusion algorithm produces the correct simulation results.

Figure 5 shows four instances of a sequence chart output with the random simulation style with seed value 5. First sequence chart instance is the initiation of the simulation run where 3 processes for each of the 3 nodes are created. These 3 processes are the main process within a node which requests for the Critical Section, the request handler and the reply handler. The other three instances show that the simulation occurs continuously with different sequences of requests and replies from the processes.

2. The other simulation style is the interactive way of simulating the model. In this way, the simulation sequence is decided manually. The model checker displays every statement corresponding to different processes that can be executed as next statement according to the model. Thus, different particular scenarios can be produced manually for checking the validity of the model.

Figure 7-11 show one interactive simulation run of the model.
All the three processes have been created for each node. The third node has requested to execute the CS and it has therefore sent the requests to the nodes in its request set i.e. nodes 0 and 1. As is shown in the sequence chart the nodes 0 and 1 were neither requesting nor executing, so they have replied to the requesting node. And thus at this instance, the node 2 is executing CS. So the statements that can be executed next corresponding to node 2 is to leave the CS, and corresponding to nodes 0 and 1 are that they can request for CS execution.

Now a scenario that can develop from is that both the nodes 0 and 1 want to execute the CS. So node 1 will request the nodes in its request set for execution. So it has sent the request messages to nodes 0 and 2. The node 0 will send the message only to node 2 as only node 2 was in its request set. However at this instance, neither node 0 has received the message from node 1 nor has node 2 received the message from node 0 and 1. So spin is displaying the options to make these nodes receive this option.
Now suppose all these sent messages are made to be received by the corresponding nodes. But node 0 has received all its messages except from the node currently executing. So it declares its priority. The thing to note here is that only those options have been displayed which can be taken from this instance according to the given model.
After the node 2 has been made to exit the CS, it sends the reply messages to the waiting nodes. After the message has been sent to all the waiting nodes, the options that are present are only 3:

1. The node 2 can request to enter the CS only.

2. The node 1 can check whether it has received the replies from all the nodes in its request set to execute its CS (which is not true as it has not received its reply from the node 0.

The point to be noted here is that no option corresponding to node 0 for sending the reply to node 1 has been displayed by the model which validates its correctness.

3. The node 0 can also check whether it has received the replies from all the nodes in its request set to execute its CS (which is true. As shown in Fig 11 when the last option of Fig 10 is selected, the node 0 can now declare that it has received all the replies.

Now once node 0 exits CS it replies to node 1 and then node 1 also enters CS and the Fig 12 shows that the options displayed according to the model are:

1. Node 2 can request the CS execution.

2. The reply handler has just received all the replies and can be terminated now with the 2nd option displayed.

3. Node 1, which is currently executing the CS can exit the CS.

4. Node 0 can request for CS execution (proc = proc + 1; is a Promela statement in the CS request
procedure of the model which eventually leads to the CS request from the process

V. CONCLUSION AND FUTURE WORK

This paper has modeled and verified the Singhal dynamic information-structure mutual exclusion algorithm for distributed system using spin. Thus the correctness of the algorithm has been proved.

REFERENCES


