Using Event –B, Distribution of Messages By Mutual Exclusion Lamport's Algorithm

Meena Verma
Computer Science and Engineering Department
Maharana Pratap Engineering College, Mandhana,
Kanpur, India

***Abstract*- Event - B is a formal technique to develop models in distributed system .This model consists the problem in abstract model, introducing solutions or the design details in refinement steps .The refinement step "Mutual exclusion algorithm for distributed system is proposed in which processes communicate by asynchronous message passing. The algorithm requires between (N-1) and 2(N-1) messages per critical section access, where N is the number of processes in the system. This technique requires the discharge of proof obligation for consistency checking & refinement checking . B- Tool provide automated proof . In the refinement steps we show how distribution of messages by Mutual Exclusion can correctly be implemented by a system of vector clocks . In this paper, we develop model using Event-B by Lamport Algorithm in distributed system. For discharge proof obligations, we search some invariants that describes some points between Mutual Exclusion, global state, vector clock. By using Event-B tool. We will get proof.**

Keywords*: *Mutual-Exclusion, Distributed System, Multicast system , Event-B, Ordering properties

I. INTRODUCTION

A mutual Exclusion broadcast is uses Non-Token Based Algorithms .The fundamental principle in all non-token based algorithms lies in processes communicating with each other to determine which one of them should execute the critical section next. The distributed mutual exclusion algorithms uses timestamps to order requests for the critical section. It is also used to resolve conflicts when two processes simultaneously request the critical section [10,13,17]. When we consider a system with two processes and a disk. The processes send messages to each other, and also send messages to the disk requesting access. The disk grants access in the order the messages were sent. Now, imagine process 1 sends a message to the disk asking for access to write, and then sends a message to process 2 asking it to read. Process 2 receives the message, and as a result sends its own message to the disk. Now, due to some timing delay(d), the disk receives both messages at the same time: how does it determine which message happened-before the other? (*A* happens-before *B* if one can get from *A* to *B* by a sequence of moves of two types: moving forward while remaining in the same process, and following a message from its sending to its reception.) A logical clock algorithm [10] provides a mechanism to determine facts about the order of such events. Leslie Lamport invented a simple mechanism by which the happened-before ordering can be captured numerically. A Lamport logical clock is a monotonically incrementing software counter maintained in each process.

In this , there are some non-token based algorithms which are used to maintain logical clocks in accordance with Lamport 's scheme . Every request for the critical section gets a timestamp, with the priority lying with processes which have smaller timestamps .A distributed mutual exclusion algorithm was developed by Lamport's , in response to the naive centralized approach that existed in present time. In the centralized solution, it consisted of a control site, which received requests for the critical section from all the processes. It queued all the requests, and granted the processes permission one by one. Lamport's identified some drawbacks [13] of this centralized solution to the distributed mutual exclusion problem.

First - that the control site might fail, and that would mean failure of the entire system.

Secondly, there is a lot of pressure on one "supervisor" to coordinate the entry of all processes into the critical section and the release of the critical section by the processes. There was also the case when many processes request the critical section at one time, which would lead to congestion at the communication links near the control site. Many researchers further came up with more drawbacks of this centralized approach, such as this approach did not support the existence of a fast algorithm. Lamport's algorithm is a very elegant algorithm which has formed the basis of most of the other algorithms in the open literature.

The causal ordering of messages are maintaining the same causal relationship that holds among the "send message" and "receive message" events. That mean, if send (msg1), send (msg2), (where send(msg1) and send(msg2) are the events of sending messages msg1 and msg2), then every process that receives both msg1 and msg2, receives msg1 before msg2.

Schemes that achieve the causal ordering of messages are very useful, and simplify the algorithms for distributed mutual exclusion to a great extent [13]. Two protocols which achieve this objective, the Birman-Schiper-Stephenson protocol, and the Schiper-Eggli-Sandoz protocol. The essence of both protocols is the same, with the main difference between the two being that the Birman - Schiper-Stephenson protocol [6,8,17] relies on the ability of processes to communicate using broadcast messages, while the second protocol does not require that the processes only communicate through broadcast messages. The main idea in both protocols is to deliver a message only if the message preceding it has been delivered to the process. In this case that the message preceding it has not been delivered, the message is not delivered immediately, but it is buffered until the message immediately preceding it has been delivered. Both protocols necessitate the existence of a vector accompanying each message that carries all the necessary information for a process to decide whether there is a message preceding it, or it is free to deliver its message [13]. The first protocol is like fig(1):

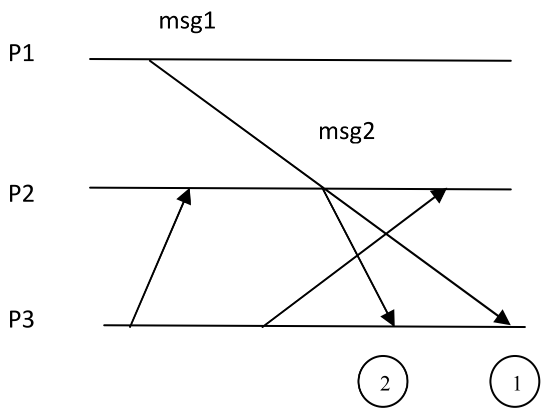

Fig. 1: Violation of Causal Ordering of messages

Requesting the Critical Section:When a process Pi wishes to enter in its critical section, it sends a REQUEST (ts,i) message to all the other processes in the system (ts is its timestamp). It then places its own request in its request-q. which is shown in fig. 2.In this ,there are two processes P1 and P2 create requests for the critical section, and send messages to the other two processes with their timestamps. At the same time, they enter this request in their respective queues which is shown in fig. 2 [13] .

VC-Vector time of Process,VCq –Vector time of Message,pp-Process,mm-Message,d-delay,N-number of process

a) VCi(i) : = VCi(i) + d , where d = {1}

b) VCq(k) := VCi(k), for all k = (1..N)

In Event-B [19]

a) LET nVC BE nVC = VC(pp) <+ { pp |-> VC(pp)(pp)+ 1 }

b) VCq(mm) := nVC || VC(pp) := nVC

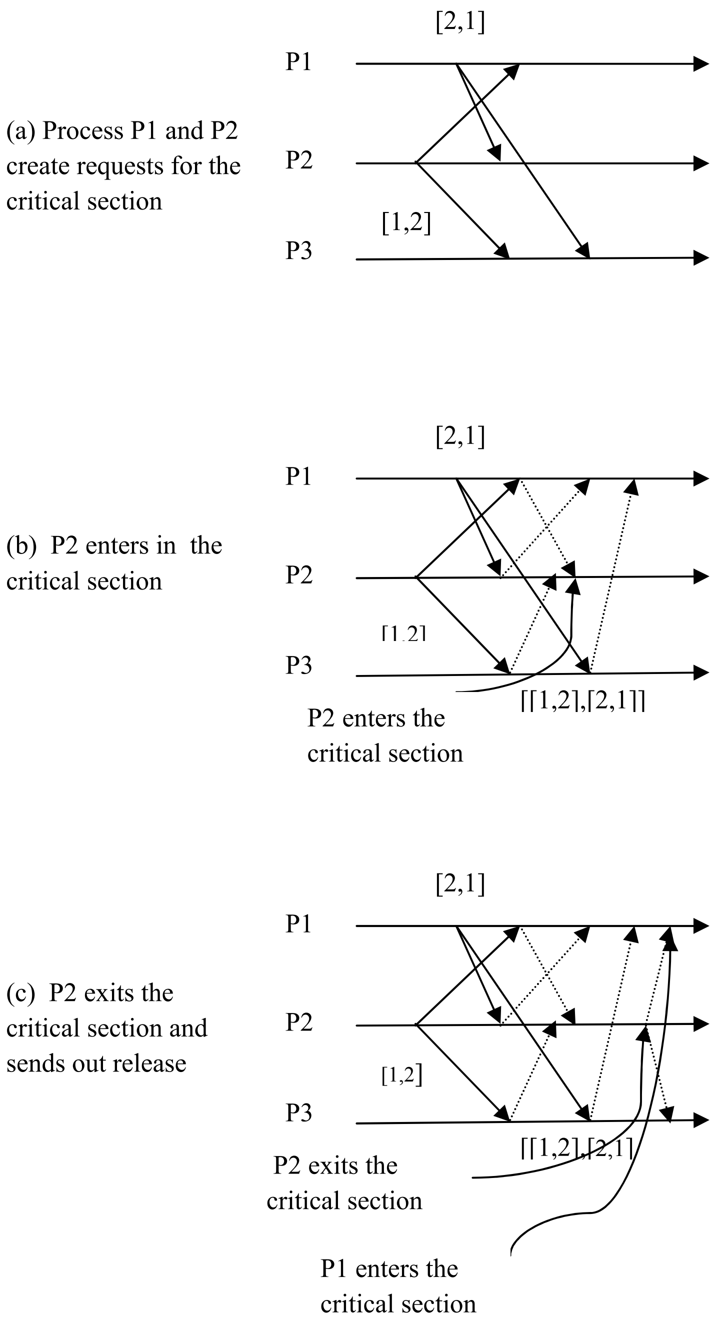
[2,1]
P1
(a) Process P1 and P2 create requests for the critical section
P2
[1,2]
P3
[2,1]
P1
(b) P2 enters in the critical section
P2
[1,2]
P3
[[1,2],[2,1]]
P2 enters the critical section
[2,1]
P1
(c) P2 exits the critical section and sends out release
P2
[1,2]
P3
[[1,2],[2,1]]
P2 exits the critical section
P1 enters the critical section

Fig. 2: Lamport's Algorithm

Reception of Process:When a process Pj receives a request, it returns ANSWER message with its own timestamp, and places the process request in its own request-q.

a) VCj(i):=VCq(i) -1

b) VCj(k) ≥ VCq(k) ,for all k= (1..N) &(k/=i)

In Event-B [19]

a) VC(pp)(sender(mm)) = VCq(mm)(sender(mm)) -1

b) !p.(p:PROCESS & p/=sender(mm)=>VC(pp)(p) >= VCq(mm)(p))

For example ,this translate to processes P1, P2 and P3 sending their timestamps to P1 and P2 [13]. Which is shown in fig 2(b). According to Lamport's algorithm, a process Pi can only enter its critical section if two conditions are satisfied:
[C1] Pi has received ANSWER message from all the other processes in the system with a larger timestamp than its own.
[C2] Pi's request is at the top of its own request-q.

a) VCj(k):=max(VCj(k),VCq(k),d>0

where VCq(k)-no.of messages delivered to Pj that were sent by Pi.

In Event-B [19]

a) VC(pp) := VC(pp) <+ ({r|r:PROCESS & VC(pp)(r) < VCq(mm)(r)} <|VCq(mm))

In this <+(override operator) updates the values in the vector clock of a process pp by corresponding values in the vector timestamp of message,where VC(pp)(r) are less than corresponding values in the message timestamp(VCq(mm)(r)).

The second condition[C2] is just restates the fact that if Pi received a REQUEST message from some other process, Pj before it sent out its own REQUEST message, it cannot possible have a lower timestamp than process Pj .shown in fig 2(b), P2 receives clearance from the other two processes that its timestamp is lesser than both P1's and P3's. Once execution of the process Pi finishes in its critical section, then it exits the critical section, and sends out time stamped RELEASE messages to the other processes in the system. Along with this, it also removes its own request from the top of its request-q.When a process Pj receives the time stamped RELEASE message from Pi, it removes Pi's entry from the top its request-q, and if its own request is on the top of its request-q and it satisfies condition [C1] of Lamport's algorithm, then it can enter its critical section. This is shown in fig[2(c)], after getting process P2's time stamped RELEASE message, process P1 enters the critical section. As far as performance goes, Lamport's algorithm requires 3(N-1) messages, every time a critical section is executed. This follows from the fact that (N-1) messages are required for the request of the critical section, (N-1) messages for the answer, and (N-1) messages for the release. In order to reduce the number of messages involved in this algorithm, it is possible to suppress certain ANSWER messages. For example if a process Pj receives a request from another process Pi, after it has sent out a REQUEST with a higher timestamp than Pi, it need not send an ANSWER message to Pi confirming that Pi has a lower timestamp than it. Because, Pi’s upon receiving Pj's REQUEST will realize it has a lower timestamp than Pj , and assume that it is free to enter the critical section. This is just one of the optimizations suggested on the Lamport algorithm. Lamport's provides distributed mutual exclusion algorithm, and a lot of the current algorithms are modifications "optimizations of Lamport's distributed mutual exclusion algorithm” [13].

II. Event-B

The abstract machine and refinement is central to Event-B [14,15,18,19,21]. An abstract machine consists of constants , sets and variable clauses modeled as set theoretic constructs. The properties and invariants are defined as first order predicates. The event system is defined by its *state* and contains a number of *events*. The state of the system is defined by the variables and the event consists of guarded actions defined on the variables. The invariants state the properties that must be *satisfied* by the variables and *maintained* by the activation of the events. In refinement steps, guards may be strengthened, variables may add or removed and new events may be introduced . Abstract and concrete variables are related through *gluing invariants*. This technique requires the discharge of the proof obligations for *refinement checking and consistency checking* . Refinement checking involves showing that the specifications at each refinement step are valid. Consistency checking involves showing that a machine preserves the invariants when events are invoked. Event-B notation is based on set theory and most of it is self explanatory. Some of the frequently used notations in our models are explained here to enhance the readability.

III.CONCLUSIONS

In this paper , using Lamport's distributed mutual exclusion algorithm, it consisted of a control site, which received requests for the critical section from all the processes. It queued all the requests, and granted the processes permission one by one. When a process Pi wishes to enter in its critical section, it sends a REQUEST (ts, i) message to all the other processes in the system (ts is its timestamp). It then places its own request in its request-q using Event –B and create model . In a future work , focuses on multiple critical section for multiple processes for distribution of messages through Mutual Exclusion Lamport algorithm using Event-B .We believe that the methodology and the models presented in this paper may be extended to enhance our understanding of other related techniques .

REFERENCES

[1] Schwartz, J.T. *Relativity in lllustrations.* New York U. Press, New York, 1962.

[2] Taylor, E.F., and Wheeler, J.A. *Space-Time Physics,* W.H.Freeman, San Francisco, 1966.

[3] Lamport, L. The implementation of reliable distributed multiprocess systems. To appear in *Computer Networks.*

[4] Ellingson, C, and Kulpinski, R.J. Dissemination of system-time. *1EEE Trans. Comm. Com-23,* 5 (May 1973), 605-624.

[5] K.M. Chandy and L. Lamport. Distributed snapshots:Determining global states of distributed systems. *ACM Trans. Comput. Systems*, 3(1):63–75,1985.

[6] Birman K and Joseph T 1987 Reliable communication in presence of failures *ACM Trans. Compur. Systems 3* 47-76

[7] Helary I-M 1989 Observing global states of asynchronous distributed applications *Proc. 3rd Inr. Workhop on Disrribured A/gorirhnu, LNCS* 392 (Berlin: Springer) pp 124-34

[8] A. Schiper, J. Eggli, and A. Sandoz. A new algorithm to implement causal ordering. In *Proc. 3rd Intl .Workshop on Distributed Algorithms*, pages 219– 232. Springer Verlag LNCS 392, 1989.

[9] T.H. Lai and T.H. Yang. On distributed snapshots. *Information Processing Letters*, 25(3):153–158, 1987.

[10] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Comm. ACM*, 21(7):558–565, 1978.

[11] L. L. Peterson, N.C. Bucholz, and R. D. Schlichting. Preserving and using context information in interprocess communication. *ACM Trans. Comput. Systems*, 7(3), 1989.

[12]. DIJKSTRA, E. W. The distributed snapshot of K. M. Chandy and L. Lamport. Tech. Rep. EWD864a, Univ. of Texas, Austin, Tex., 1984.

[13]. Joydeep Ganguly and Michael D.Lemmon,Theory of Clock Synchronization and Mutual Exclusion in Networked Control Systems, University of Notre Dame, ISIS-99-007 August, 1999

[14] C. Metayer, J. R. Abrial, and L. Voison, "Event-B language," RODIN deliverables 3.2,http://rodin.cs.ncl.ac.uk/deliverables/D7.pdf, 2005.

[15] D. Yadav and M. Butler, "Rigorous design of fault-tolerant transactions for replicated database systems using Event B." in *Rigorous Development of Complex Fault-Tolerant Systems*, volume 4157 of Lecture Notes in Computer Science, Springer,2006, pp. 343–363.

[16] M. Leuschel and M. Butler, "ProB: A model checker for B," in *FME*, ser. Lecture Notes in Computer Science, K. Araki,S. Gnesi, and D. Mandrioli, Eds., vol. 2805. Springer, 2003, pp. 855–874.

[17] K. P. Birman, A. Schiper, and P. Stephenson, "Lightweigt causal and atomic group multicast." *ACM Trans. Comput. Syst.*, vol. 9, no. 3, pp. 272–314, 1991.

[18] J.-R. Abrial. The B-Book: Assigning programs to meanings. Cambridge UniversityPress, 1996.

[19] D Yadav, M Butler. Application of Event B to Global Causal Ordering for Fault Tolerant Transactions. Proc. of REFT 2005, Newcastle upon Tyne, pp 93-103, 2005.

[20] J.-R. Abrial . A system development process with Event-B and the Rodin platform . In M. Butler, Hinchey, Larrondo-Petrie, ICFEM 2007. LNCS, vol.4789,pp.1- 3.Springer(2007)

[21] Butler, M., Walden, M. 1996 Distributed System Development in B. In Proceedings of Ist Conf. in B Method, Nantes, pp.155-168.

[22] Mukesh Singhal and Niranjan Shivaratri, Advanced Concepts in Operating Systems, McGraw Hill, 1994