Volume 4, Issue 4, April 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

A Distributed Rapid Mutual Exclusion

Vivek kumar Gaur
ISM Department
DIT,University, India

Rohan Verma
Asst prof,Dept. of info technology
Uttaranchal College of Science & technology, India

Kushal Gupta
Asst Prof,Dept. of info Technology
DIT,University, India

Abstract—we are present a new solution of Distributed fast mutual Exclusion to access memory of a process. In Distributed Mutual Exclusion shared resources use concurrent operations. In this paper we use a unique Token for execute our process in Critical Section. If any process wants to enter the critical section then process demands a token. A standard multiprocessor has been work of every hardware of system. Read and Write operations are check shared resources that resources are completely read and write and we are trying our process deadlock free.

Key Words and Phrases: Critical Section, process, Token, Multiprocessor, Deadlock

I. INTRODUCTION

Our main goal is solve the mutual exclusion problem. Firstly we should know the main problem of mutual exclusion that at a time only one process can enter in critical section. Basically firstly mutual exclusion problem solve by "Dijkstra". But it did not give the fair solution. This algorithm is used for find the shortest path for sharing resources. Mainly Distributed Mutual Exclusion is used for message passing and sharing resources in a group of computers. Groups of computer are connected by a network and they are sharing resources.

When any process demand for critical section then process send a request all other process. When all process sends a reply message to the request process then request process enters the critical section. For example In a office only one printer is use for printing. Many user are connected by a single printer. If any user want to print some text it give the command for printing text then mutual exclusion required.

Now a days we are using a single standard Multiprocessor for change a small modification hardware in a office network. In the multiprocessor and memories we do not use the test and set mythology. We are widely studied the process of Distributed Mutual Exclusion of shared memory and decide fast but when we are developing algorithm then we have decide any process of does not wait a outside of the critical section. If any process wait outside of the critical section it is called starvation. But sometimes a process without wait enters the critical section. Then starvation does not occur. Most of the time some process are wait forever outside of the critical section then starvation occur.

If read and write operation access the shared memory with a latest and fast speed processors then it takes a lot of time in a local network. Much number of processes of shared memories is a good achievement of execution of algorithm. In this algorithm we have use 0(N) process for shared memories but many publication work on N process. In this paper we present two reads and four writes of shared memory in this case. This algorithm gives the solution of distributed fast mutual exclusion and it requires how much size of process can enter in critical section.

II. ALGORITHM

In this algorithm we suppose of each and every process have a unique number of identity that is totally different of other process. These process have integer type value the condition of algorithm these integer type value must be positive. In the memory Read and Write operation are permitted to a single word and imagine a number of process are hold a long time according to the condition of deadlock. All other unique process does not change any integer value of algorithm.

According to the algorithm we take a one of the simple way by Michael Fisher, a number of process pi give this following value, in our algorithm c is a word of shared memory in the brackets and work on read and write operation and wait w for a short time while it has -w skip:

```
repeat    wait{ c = 0};
          {c:= pi};
         {delay}
 until { c = pi};
critical section;
c:= 0
```

When process enter the critical section in average time then delay operation performed and process does not wait a long time for demanding critical section. Then another process pj read the value of word c and wait the executing of process pi. When c:= pi value is release the critical section then process pj will complete this process same as of process pi. Now we are discuss about the process speeds when we are use the read and write operations then processor take the responsibility of process delay. How a processor can handle the action taking of power responsibility with a low level operating system and how ignore the hardware. In our algorithm we have a condition of process wait if a process is running in then other

process does not wait if process pi could want to wait then it give the permission of other process in this condition pi could wait otherwise not.

Our algorithm appearing to access in the memory only four times. When process delay in a single memory or process demand to enter critical section then we give time of process to enter in the critical section. If process does not grant critical section at that time it has gone into the worst case. In this (N-1) process demand for access the memory and the best worst case 0(N) times access the memory.

Now let us start to make a good algorithm but firstly we know that how much memory we have to need for starting mutual exclusion in the first stage of the system. Our main goal is make a fast mutual exclusion algorithm so we have requires a static memory access to N number of process in the absence of contention if our algorithm conclusion could be wrong then argument solve this problem to make a good algorithm.

When a process delays a long time and it has watched other process have done his work in the critical section to 0(N) time because of sufficient memory contention. So in this algorithm we assume that no delay operation performed in this we access only memory .Let Ri is the series of read and write operations and i is a process is entering in the critical section and read operation is returns the starting value and written in the series Ri.

If any process write a number and that process does not read by another process that is no point in our algorithm. and the Ri memory word not accessed and Rj play if any both process i and j want to enter the critical section at the same time then we will use a minimal number of memory and Ri access the memory at that time. If number of memory is fixed then N is independent and increasing N then many process of i from Ri give the identical sequence of read and write operation. and the actual values are return which is depend upon process i. Our main focus is those process that we may assume these process have loss generality and each process access the memory words and order.

The first operation in Ri read, and all other process can execute and find the initial value before execute any process. Operation Ri write few variable in C and the second operation gives no sense to another write to C and another variable D ,two write operations change by a single write to a longer word. Now the second operation of Ri is read and the second operation of every process execute at that time when first operation finished. Other process read correct and written.

Start (C := Pi);
If (D ⊨ 0) then goto start A;
(D:= Pi);
If (C ≠ Pi) then delay;
If (D ≠ Pi) then goto start A A;
Critical Section ;
(D := 0)

Every process will do work read and write operations. If any process writes C then another process read D variable. So Ri do read C and write D. The final operation Ri before entering the critical section when contention is not present and does not write because write operation cannot decide that which process enter the critical section or not. So in this algorithm Ri make a sequence write C, read D, write D, read C and this sequence D has initial value then firstly write D and read C according to algorithm. If find the value of C then firstly enter C in the critical section.

Critical section executing a process and at least one write operation is executed if critical section is free then process enters late and realize there is no contention. When every process give the writes of C then process C access the shared memory and after that process write D and reset the value of D after leave critical section. The sequence of memory access is w-C, r-D, w-D, r-C in the critical section w-D is shown the figure 1 and D initial value is 0 and goto statement work when conflict is not present and the process shown Pi of program.

When another process Pj firstly read D= 0 then If statement D= Pi and Pj will enter the second number or execute the critical section. Pi finishes his executing delay statement then D reset 0. So in this case our algorithm make deadlock free and every process may be starved for execute the critical section. In this algorithm only upper bound not time required all other operation is take time to execute. The most case of the algorithm we have not need upper bound. Our main focus is how to access a lot of memory when contention is not present in the algorithm.

start (x[Pi] :=true);
(C := Pi)
If (D ≠ 0) then (x[Pi] :=false);
await (D=0);
goto start A;
(D := Pi);
If (C≠ Pi) then (x[Pi] :=false);
for Pj :=1 to N do await (-x[Pj]) od;
if If (D =Pi) then await (D=0);
goto start A A;
critical section;
(D:= 0);
(x[Pi] :=false);

In the critical section a few protocol enters in the form of w-C ,r-D ,r-C and the operation sequence number performed 1,2 and 3.When process perform the operation these scripts denote the process sequence.

w2-C, w1-C, r1-D, r2-D, w1-D, w2-D, r1-C, w3-C, r2-C

III. PROPOSED WORK

In the previous algorithm we were using read & write operations and we are take only four read and two write operations for access memory and main problem of the previous algorithm we have limit of access the process because we used only four read and two write operations to execute critical section. Other problem of this algorithm it has taken a lot of time to execute a process because firstly it has read the then it allow to write the process to enter the critical section so in this process it has taken time. After discuss about these problem we proposed a new solution of mutual exclusion for sharing resources. Our new scheme we work on token based algorithm. In this algorithm we are using a new technique we can create nodes in different places and create a network. After this a unique token give a process which is want to enter the critical section. Now let us know about the total brief summary of this algorithm.

In our new scheme of token based algorithm we divide distributed mutual exclusion in two category one is Distributed algorithm and second one is Centralized algorithm. Distributed algorithm is two types Token based and Non Token based. Our new scheme is based on Token based algorithm. In the Token based algorithm we are using a unique token for sharing resources

Token Based Algorithm

In this scheme we are using a simple property to execute a critical section at a time only one process can enter in the critical section. We create a special object to enter the critical section which is called token. This token helps us for sharing resources or executes a process in the critical section. Only one process uses one unique token at a time. When token finish his work then we will give this token to another process which process want to attempt the critical section. Movement of token is based on two methods in the Distributed Mutual exclusion. Token asking method and perpetual mobility are two method uses for movement of token.

Perpetual Mobility

Perpetual mobility of token give the permission of token to enter the critical section when process demand the token if any process does not want to enter the critical section then token pass to another nodes or process. In the perpetual mobility token moves in a logical ring and clockwise direction and we cannot leave the order of token. In this method we use used a unidirectional ring. but main problem in this method when number of process are increasing the average waiting time of process taken token is increase. So in short we can it does not have stability property.

Token -asking

In token asking method a process wants to enter the critical section when process takes a request
Of token and execute critical section. But token holding process wait outside the critical when process does not execute. Then token holding process sends a request to another process. In Suzuki & kasmi algorithm token holding process does not need the token then token holding process does not send token away. In this method using N process if any process wants to enter the critical section it send automatically request message (N-1) process and token send directly to process Pr. In token asking method we are using new techniques of logical tree structures. Logical tree structure is divided into two categories static and dynamic.

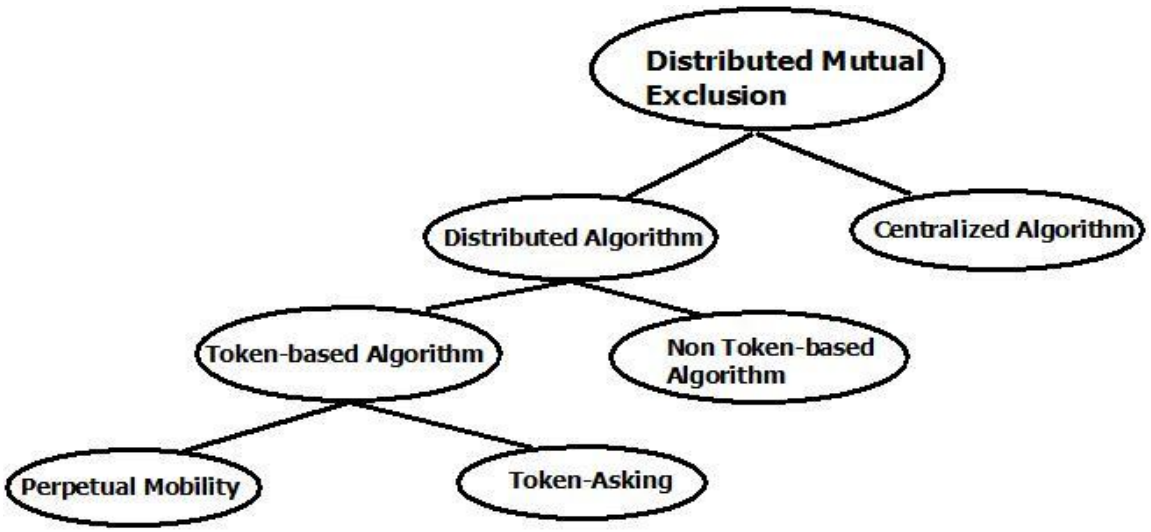

Fig. 1 Overview of the distributed mutual exclusion

IV. CONCLUSIONS

In this paper, we have proposed an efficient to access memory of a process. Our algorithm is especially suited for applications or processes to share resources and use concurrent operation. Our algorithm has much better performance with the unique token for the execution of our process in critical section. With this algorithm the process enter in the critical section and does not wait a long time for demanding critical section, which reduce the deadlock possibility. We have also described several modifications to the basic token-based algorithm which help in process deadlock free.

References

[1] N. G. DEBRUIJN, *"Additional comments on a problem in concurrent programming control"*. Communication ACM 8, 9 (Mar. 1967), 137-138..

[2] E. W. DIJKSTRA, "*Solution of a problem in concurrent programming control*". Communication ACM 8, 9 (Sept. 1965), 569.

[3] D. Agrawal, A. El Abbadi , " *An efficient solution to the distributed mutual exclusion problem* " , in proc. 8th ACM Symposium on Principles of Distributed Computing , pp. 193-200 , 1989 .

[4] L. Lamport; Time, "*A Fast Mutual Exclusion Algorithm*". ACM Transactions on Computer Systems, Vol. 5, No. 1, February 1987.

[5] J. M. Helary, N. Plouzeau, M. Raynal, *" A distributed algorithm for mutual exclusion in an arbitrary network*", volume 31 of Computer Journal,pp. 289-295, 1988..

[6] Y.-I. Chang, "*A Simulation Study on Distributed Mutual Exclusion*," J. Parallel and Distributed Computing, vol. 33, pp. 107-121, 1996.

[7] L. Lamport; " *Time , Clocks and Ordering of Events in a Distributed System*". Communications of the ACM, vol. 21, no. 1, pp. 558-565, July. 1978

[8] S. Lodha and A. Kshemkalyani, "*A Fair Distributed Mutual Exclusion Algorithm*,"IEEE Transactions On Parallel And Distributed Systems, Vol. 11, No. 6, June 2000,pp. 537-549

