J. Basic. Appl. Sci. Res., 2(9)9216-9223, 2012

ISSN 2090-4304
Journal of Basic and Applied Scientific Research
www.textroad.com

Accurate and Fast Mutual Exclusion Algorithm

Aasim Khurshid[1], Tabassam Nawaz[2], M. Younas Javed[3], Aamir Khan[4], Farman Ullah[5]

[1, 2] Software Engineering Department University of Engineering and Tech. Taxila, Pakistan
[3]College of Electrical and Mechanical Engineering , NUST, Islamabad Pakistan
[4,5]Electrical Engineering Department, COMSATS Institute of IT Wah Campus, Wah Cantt. Pakistan

ABSTRACT

Multiprocessing systems are programmed cleanly using Critical sections. When a process desires to access some shared data it first gets mutual exclusive access to critical sections for reliable outcome as processes may possibly manipulate the data. This paper presents an algorithm that can solve the problem in single processing, multiprocessing and distributed systems efficiently with minimal changes. For distributed systems we introduce message passing service while keeping rest of the mechanism same works faster than many other algorithms for distributed systems. The algorithm compares its efficiency with bakery's algorithm and performs much better with the liberty of introduction of multiple critical sections for dissimilar shared data. Due to this multiple processes can execute in different critical sections concurrently.

KEYWORDS: Mutual exclusion, Synchronization, Distributed systems, Operating systems, Algorithms, concurrency.

1. INTRODUCTION

Cooperating processes can effect or be affected by other processes in the system. Cooperating processes either share address space (that is both code and data) or be allowed to share data only through files or messages. The former can be implemented through multithreading. Concurrent access to shared data may result in data inconsistency [5]. The critical section problem is available almost in every text book of operating systems. Consider there are n numbers of processes which are competing to use some shared data. Each process has a code segment, called critical section, in which it can access and manipulate shared data. If concurrent processes accessing the shared common resource are not synchronized such that only one process can access this shared resource, then it will lead to integrity violations. CS Problem is to guarantee that when one process is executing in its critical section, no other process is allowed to execute in its critical section.

In distributed systems, cooperating processes share both local and remote resources. Chance is very high that multiple processes make simultaneous requests to the same resource. If the resource requires mutually exclusive access (critical section – CS), then some regulation is needed to access it for ensuring synchronized access of the resource so that only one process could use the resource at a given time. This is the distributed mutual exclusion problem [1]. The problem of coordinating the execution of critical sections by each process is solved by providing mutually exclusive access to the CS. Mutual exclusion ensures that concurrent processes make a serialized access to shared resources. Neither shared variables (semaphores) nor a local kernel can be used in distributed system, in order to implement mutual exclusion. Thus, it has to be implemented on message passing, in the context of impulsive message delays and incomplete knowledge of the state of the system. Requirements for a mutual exclusion mechanism are:

- **Safety:** Not more than one process is allowed to enter critical section simultaneously.
- **Fairness:** The requests for entering CS must be executed in the order in which the requests are made
- **Freedom from deadlocks:** Any process interested in entering CS must be allowed to do so within a finite amount of time.

***Corresponding Author:** Aasim Khurshid, Software Engineering Department University of Engineering and Tech. Taxila, Pakistan. Email: engr.aasimkhurshid@gmail.com

• **Freedom from starvation:** If a process made request for entering into CS, then there must be a limit on maximum number of process that are allowed to enter CS before it so that indefinite blocking of a process(s) can be ruled out.
• **Fault-tolerance:** In the wake of a failure, it is enviable that the algorithm reorganizes itself so that it can continue to function without any prolonged disruptions.

To study the mutual exclusion in single processing, multi-processing and distributed systems is the main intention of this thesis. Testing and analyzing the results obtained will give fair idea of how solution to critical section problem be improved. Moreover, adjustment can be made to some solutions to minimize the cost and improve efficiency. Finally we will try to formulate an efficient and optimal mutual exclusion algorithm that satisfies all the requirements of mutual exclusion.

To achieve the objective of the thesis we started from the single processing system, then multiprocessing system and then moved to distributed systems. Algorithms of mutual exclusion are excessively reviewed and systems were examined using the same software under different loading conditions to understand the concepts related to critical section problem.

After reviewing the literature and simulation results for the above mentioned systems made a way for studying the performance matrices and the improvement factors, which ultimately lead us to a solution.

2. Related work

Consider a system having n processes (P0, P1, P2…… Pn). All processes are competing to access some shared data. Each process has a code segment called Critical Section, in which it can access and manipulate shared data e.g. changing common variables, update tables, writing to files etc. The significant attribute is that only one process can be in its critical section at any point of time. Critical section problem is to design a protocol that these processes can use to cooperate. The section of code implementing this request is Entry section may be followed by an exit section having code to exit from critical section may involve updating of some common data structure of the algorithm that can be used to allow other process to enter into their CS. The remaining code which is not associated to critical section of the process is organized in remainder Section.
The general structure of a process is given in figure below

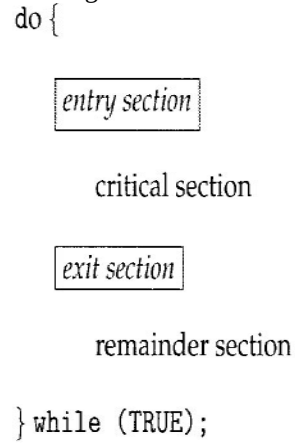

Figure 2.1: General structure of a typical process

A solution to critical section problem must conform to these three basic requirements.

2.1.1 Mutual Exclusion:

Only one process can execute in its critical section at a time i.e. when one process say Pi is executing in its CS than no other process is allowed to execute in its critical section.

2.1.2 Progress*:*

When no process is executing in its critical section than only those processes which are not executing in their remainder section are allowed to compete in the decision to enter into their critical sections and this selection cannot be delayed indefinitely.

2.1.3 Bounded waiting:

When a process made request to enter into its CS than there should be a limit on number of process that are allowed to enter into their critical section before this process's request is approved.

We assume that each process is executing at non zero speed. Yet no assumption regarding relative speed of the processes is possible.

2.2 Bakery Algorithm:

Lamport's bakery algorithm is a computer algorithm by computer scientist Leslie Lamport, which is the simplest solution for critical section problem. In computers multiple threads may try to access simultaneously the same resources. Integrity violation can occur if two or more threads at the same time try to write into the same memory location, or if one thread reads a memory location before another has finished writing into it. Lamport's bakery algorithm is one of many mutual exclusion algorithms designed to eliminate the concurrent access to multiple threads to enter into their critical section at the same time in order to prevent data corruption. Bakery algorithm is use to solve critical section problem for n processes. Before entering into critical section each process receives a number in increasing order. Process with the smallest number enters the critical section. If two processes Pi and Pj receives the same number than process names will be used to serve the request, i.e. if i<j than Pi will be served first otherwise Pj.

The algorithm uses the following shared data:

```
boolean choosing[n];
int number[n];
```

number[n] is the integer array of n length that stores the identification number given by the algorithm when process wants to enter into critical section initialized to 0. Choosing[n] is the Boolean array of length n initialized to false.

Considering this shared data the algorithm is as below.

```
do {
1. choosing[i] = true;
2. number[i] = max(number[0], number[1], …, number [n – 1])+1;
3. choosing[i] = false;
for (j = 0; j < n; j++) {
4. while (choosing[j]) ;
5. while ((number[j] != 0) && (number[j,j] < number[i,i])) ;
}
Critical Section
6. number[i] = 0;
Remainder section
} while (1);
```

The algorithm satisfies all the three properties of critical section problem.

4. Proposed Algorithm

The core algorithm receives requests from processes and gives them mutual exclusive access to some shared data. To achieve this job processes need to be organized in some identical fashion as they have to perform some tasks in common, for instance request CS in the start and while exiting notify the main algorithm by some mean that it made its way out of its CS. For this process organization is as below.

3.1.1 Process organization

There can be number of processes in the system. The general structure of the process is as

```
do {
start section
critical section
exit section
remainder section
} while(true)
```

In the start section process will have a code that makes request to enter into critical section and to do code which is required to enter into its critical section. In our case the start section of the code is just to make a request for its CS. No shared variables are needed to set in the start section.

In the critical section when the algorithm allows this process to execute in its CS. In this section the process may access and manipulate the shared data.

When a process makes its exit from critical section it has to reset a shared variable flag=true; which means other process may enter into its own CS now on. Queuing module is continuously watching this variable so that it can allow other process to make progress.

Process executing in remainder section is either done up with its critical section or it don't want to enter into its CS. A process executing in this section is not allowed to make request for its CS.

3.1.2 Algorithm Description

The algorithm is designed to solve the critical section problem for n processes. When a process enters the system it receives an identity number from identity generation module. When a process make request to enter into its critical section it will wait for 5 milliseconds on average to get response from the main module if it didn't get any it will change its state to waiting and will wait in the waiting queue. Through this we can allow the scheduler to take another process from the ready queue to utilize CPU time. In contrast if it gets response within this time (called GRACE period) it will make its entry to critical section. This grace period changes considering the process flow and average context switching time overhead.

The core algorithm is continuously receiving processes who want to enter into their critical sections, storing them in the Queue. In parallel Queuing module removing a process from the top of the queue and allowing it access to enter into its critical section to use shared data.

When a process completes its execution in the critical section it reset a shared Boolean variable flag to true and when the value of this variable gets true, queuing module allows another process to enter into its critical section.

The general form of algorithm is like

Shared data:

Boolean flag (= true initially)
int ID[ ] (array of length n)

Algorithm Program

```
/********************** Thread1*****************/
        Add-To_Waiting-Queue()
        {
        While(Request)
        {
        Queue.Add-this-ID;
        }//end of loop
        }// end of function
/********************** Thread2*****************/
        Allow-To-CS()
        {
        While(Queue not Empty){
        Flag=false;
        Queue.allow(process on top);
        while(flag==false);
        }//end of loop
        }//end of function
/********************** Process exit section*****************/
Critical section;
        Flag=true; //exit section
        Remainder section;
```

3.2 Correctness Proof

Correctness proof for n processes solution is organized as Mutual exclusion, Progress and bounded waiting.

3.2.1 Mutual exclusion

Clearly algorithm takes a process from the queue and allows it to enter into its critical section. When a process is allowed to enter into its CS algorithm resets the shared variable falg=false. Which means processes should have to wait outside their critical section as a process is already executing in its CS. Through this only one process at a time is allowed to enter into its CS at any point of time. If time expires or process exits its critical section it updates a shared variable flag to True. Only than a new process is allowed to enter when the value of the flag gets false.

3.2.2 Progress

Algorithm continuously investigates the shared variable flag in Thread2 whenever it resets by any means the new process is allowed which makes progress. While flag=false this thread is in

busy waiting in the same statement whenever the flag variable is rest to true in the process's exit section this condition gets false and the algorithm moves to the next iteration in the loop and if queue not empty it allows another process to its CS which illustrate progress.

3.2.3 Bounded waiting

Algorithm is fair it works on first come first serve basis so the bound is the process Pi is allowed to enter into its CS on its turn i.e. after the total number of processes which made request before it. So bounded waiting is preserved. When a new process wants to enter into its critical section it sends request and when a request is received it adds this request to the end of the queue. When all the other processes that made request before this process are done up with their critical section it gets permission to enter into its CS.

3.3 Block Diagram:

The block diagram of the system is presented in this section. The block diagram demonstrates the major components of the system and the flow of the system. It also shows how processes are coming and receiving identities as well as how they get permission when they make request for the critical section.

The block diagram of the complete process is as below.

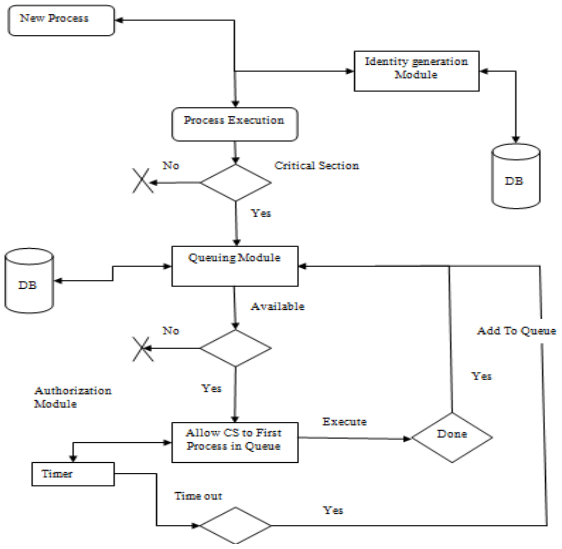

Figure 4.2: Block diagram

4. RESULTS AND DISCUSSION

The performance in terms of time of the proposed algorithm is compared with the bakery algorithm for n processes in multi processing systems. The proposed algorithm performs well in low and high loading conditions in terms of time it takes on average. The result statistics are as blow for number of processes graphically. When the mutual exclusion system was simulation at no of process requests 10 the result obtained of the bakery algorithm vs. proposed algorithm can be seen best by the figure 5.1 (No. of Requests = 10), 5.2 (No. of Requests = 100), 5.3 (No. of Requests = 1000), 5.4 (No. of Requests =10,000) and 5.5 (No. of Requests = 100,000).

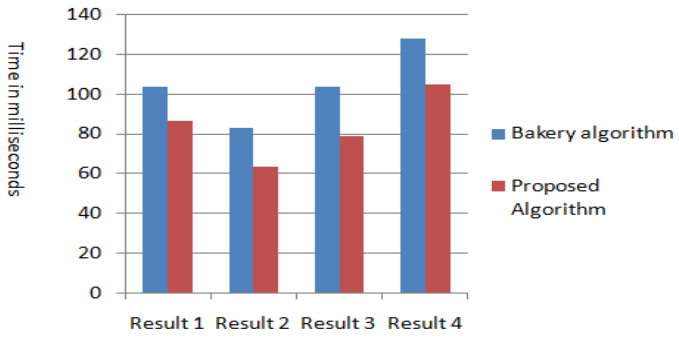

Figure 5.3 : No of requests=10

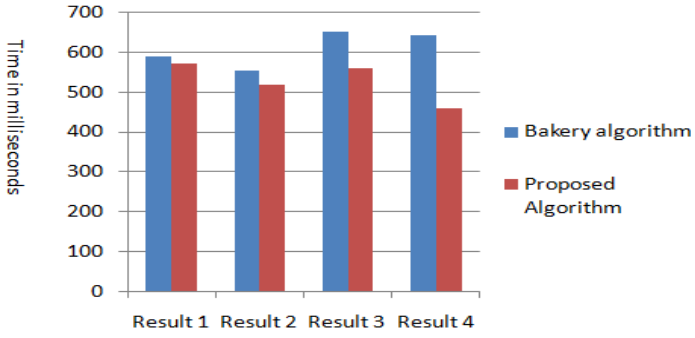

Figure 5.4 : No of requests=100

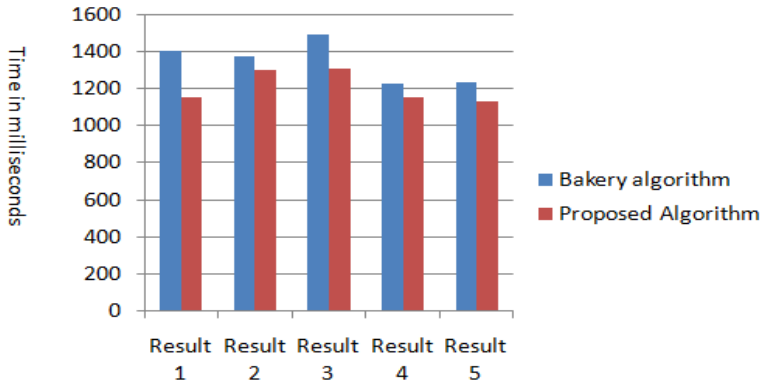

Figure 5.5: No of requests=1000

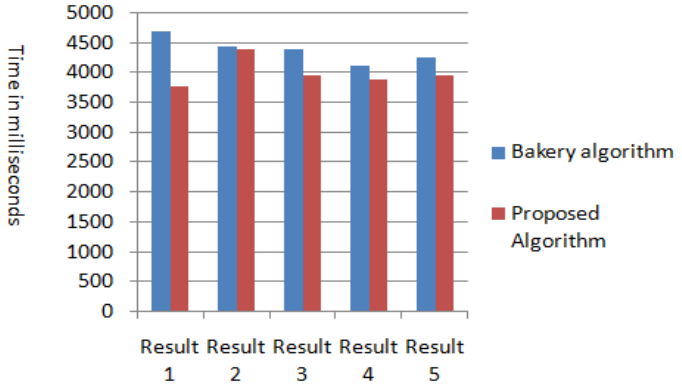

Figure 5.6 : No of requests=10000

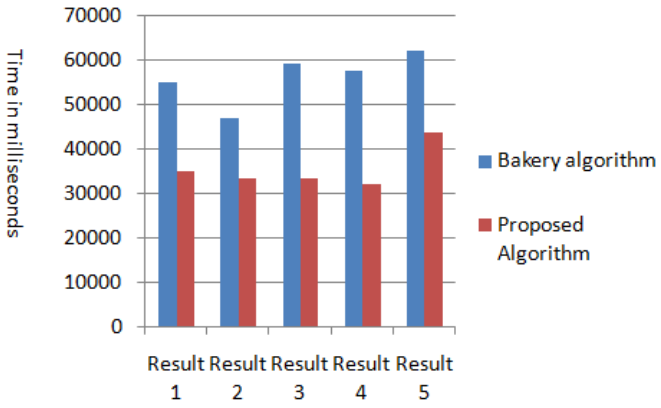

Figure 5.7: No of requests=100000

3.3.1 NOPR Vs Time

Average results for the comparison of Bakery and the proposed algorithm are simulated in figure 5.6. This shows the overall comparison of time taken for both the algorithms at different loads starting from number of processes to be 10 to 100000. Graph illustrates the comparison at many different points which evidence the improved efficiency of the proposed algorithm.

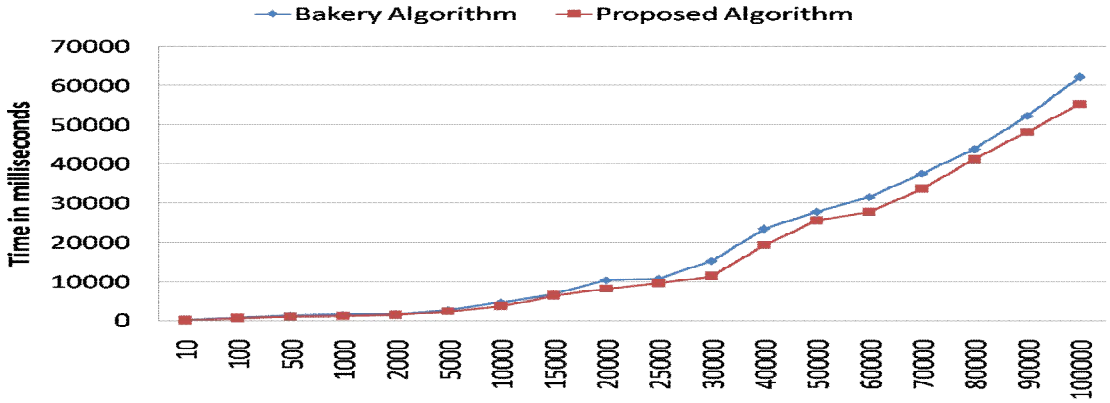

Figure 5.8: Comparison graph

3.4 Distributed Systems Algorithm

In distributed systems, the proposed algorithm serves as centralized algorithm. The algorithm works in central and all the sites sends request to this central site through message passing. Assumptions are that for any two processes pi and pj, the messages sent from pi to pj are received in the same order in which they are sent. Furthermore, we assume that every message is eventually received. We assume that every process can send message directly to every other process. For generality we consider every site has one process.

3.4.1 Working

All the processes which want to enter into their CS sends request to the central site which they know by sending a message. Message contains the site ID. The central site receives that request and adds this to the end of the end of the queue. Whereas in parallel it removes a process ID from the queue and allows the process on top the entry authority to its critical section.

When site completes its execution in critical section it sends finish message to the central site which means it is no more in critical section. Than the algorithm removes another process ID from the queue and sends it ok message mean that it can enter into its CS.

The core algorithm works in similar for distributed systems while message passing system is introduced as in distributed systems sites can be at remote locations.

5. Conclusion

In this paper we analyzed the performance of mutual exclusion algorithms. We successful in devising an algorithm that performs with high throughput than many others while satisfying all the necessary requirements of mutual exclusion algorithms.

It is clear from the results that isolating the receiving request module from the authorization module which allows the processes to execute in their critical sections, and parallel running of these two modules increases the efficiency of the algorithm.

Yet another improvement in the busy waiting which many of the algorithms do while a process waiting for the CS also makes the algorithm efficient. We introduced variable busy waiting system which in low load conditions waits for some period of time which saves the context switching overhead while in high load conditions don't wait and returns immediate control to CPU scheduler as there is no chance of getting permission in short time which ultimately saves busy waiting time.

Queuing mechanism limits the shared data structure used by algorithm. This also allows the algorithm to be used in single processing, multi-processing and distributed environment with minimal change.

6. REFERENCES

[1] E.W. Dijkstra; Solution of a Problem in Concurrent Programming Control, Communication ACM, vol. 8, no. 9, Sept. 1965

[2] Ricart G. and Agrawala A.: An Optimal Algorithm for Mutual Exclusion in Computer Networks. Communications of the ACM, vol. 24, no. 1,pp. 9-17, Jan. 1981

[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 .

Leslie Lamport; Time , Clocks and Ordering of Events in a Distributed System. Communications of the ACM, vol. 21, no. 1,pp. 558-565, July. 1978

[5] Md. Abdur Razzaque Choong Seon Hong, "Multi-Token Distributed Mutual Exclusion Algorithm," in 22nd IEEE International Conference on Advanced Information Networking and Applications, 1550-445X/08, AINA, 2008, pp. 963–970.

[6] A S Silberschatz, P.B.Galvin, G Gangne, *Operating System Concepts*, USA, John Wiley & Sons, 2005, ISBN: 0-471-69466-5.

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

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

[9] O. Carvalho and G. Roucairol, ªOn Mutual Exclusion in Computer Networks, Technical Correspondence,º Comm. ACM, vol. 26, no. 2, pp. 146-147, Feb. 1983.

[10] Ricciuti, Mike (July 20, 2007). "Next version of Windows: Call it 7". CNET News. Available online at http://www.news.com/2100-1016_3-6197943.html.

[11] Nash, Mike (October 28, 2008). "Windows 7 Unveiled Today at PDC 2008". *Windows Team Blog*. Microsoft. Available online at http://windowsteamblog.com/blogs/windows7/archive/2008/10/28/windows-7-unveiled-today-at-pdc-2008.aspx. Retrieved November 11, 2008.

[12] Sadegh Firoozandeh and Abolfazl Toroghi Haghighat," Reducing Coordinator Failures in Centralized Algorithm to Guarantee Mutual Exclusion Using a Backup Site" in Second IEEE International Conference on Future Networks, 2010,pp. 124-128

[13] 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.