Impact Factor (SJIF): 3.632

International Journal of Advance Research in Engineering, Science & Technology

e-ISSN: 2393-9877, p-ISSN: 2394-2444

Volume 3, Issue 6, June-2016

Critical Section and Mutual Exclusion Problem in Distributed System

Mr. Niteen Y.Hanchinmani[1], Prof. S.L.Deshpande[2]
[1]*Department of CNE,VTU PG Center, Belagavi*
[2]*Department of CNE,VTU PG Center, Belagavi*

Abstract-- *Distributed system is one which has a number of systems connected to it. Each of them does not share a common memory and they do not have a global clock. And these systems communicate with each other by passing messages. One of the major design consideration is resource sharing. And the problem arises while dealing with the critical section. CS is a part of the program or code which can be accessed by only one process at a time. If two or more processes try to gain access to that critical region then the problems may arise in the system. The major problem due to this is known as deadlock. So to avoid this condition some algorithms are designed. Out of these proposed algorithms this paper analyze two of the algorithms compare the result of the same. Two algorithms taken in this paper are "baker's algorithm" and other one is the "Dekker's algorithm".*

Keywords- Critical section, mutual exclusion, Baker algorithm, Dekker algorithm

I.INTRODUCTION

Distributed system is an infrastructure, which is the group of network, where network includes physically connected devices and processing nodes. Every node holds an unmistakable programming subset of the worldwide OS.. A subset is a composite of unmistakable administration suppliers. Initial one is a widespread negligible bit, or small scale piece, which specifically controls the hub's equipment. Furthermore, the second one is the accumulation of larger amount framework administration peripherals that partners the hub's individual and shared exercises. In general it is a network of computers that do not have a memory or clock and communicate by passing messages over the network

A 'critical section' is a code section that gets to shared variables and must be executed as a atomic activity. In programming, a CS is a part of a multi-process program that need not to be simultaneously processed by greater than one of the project's processes A CS will more often than not end in limited time, and threads, undertaking, or process will need to wait for a particular time to enter it. Some synchronization instrument is required at the passage and way out of the CS to guarantee restrictive use.

In mutual exclusion it provides no simultaneous processes demand the execution at same section of time interval; in order to overcome race case. Amount of time required for process execution using common memory system by shared requirements of system. This basically uses locks and semaphores to prevent other processes to enter into the critical region.

II.PROBLEM STATEMENT

Earlier when distributed systems were designed the problem arrived while there is sharing of resources and memory. As discussed in disadvantages above, one of the major disadvantages of distributed systems is 'mutual exclusion' problem. The exclusion problem directly affects to the reliability of the system. This problem arises when there is sharing of the resources and when there is a shared memory. This condition occurs when any two or more processes try to access the critical section. Exclusion in DS causes deadlock conditions and causing problem in optimal resource sharing hence reflecting in the performance of the system.

2.1 Critical section problem

Casually, a 'critical section' is a code section that gets to shared variables and must be executed as a atomic activity. The CS issue alludes to the issue of how to guarantee that at most one procedure is executing its CS at a given time. CS in various threads are not as a matter of course the same code section.

In programming, a CS is a part of a multi-process program that need not to be simultaneously processed by greater than one of the project's processes. As it were, it is a bit of a program that requires ME of access. Typically, the CS gets to a mutual resource, for example, data structure, or a system connection, that does not permit different simultaneous accesses. A CS may comprise of numerous discontinuous parts of the code. For instance, one programs part may read from a record that another part wishes to adjust. These parts together shape a CS, since concurrent readings and modifications may collide with each other.

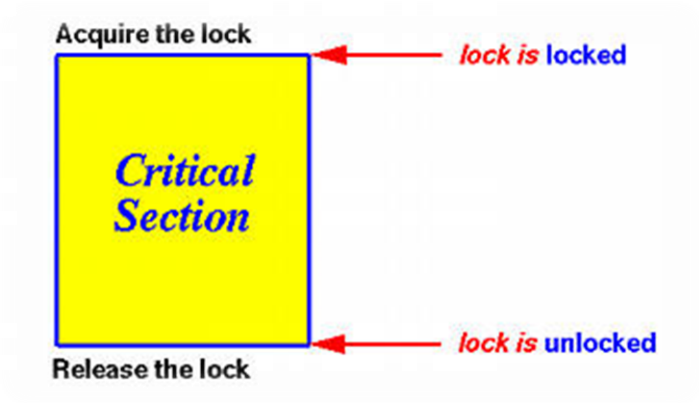

Fig 2.1 CS lock

As shown in the above figure, when a thread wants to enter a CS it must acquire some kind of lock before entering it. And once the work is done it must release the lock so that other processes should get access to the CS.

III .MUTUAL EXCLUSION

In mutual exclusion provides no simultaneous processes demand the execution at same section of time interval; in order to overcome race case. Amount of time required for process execution using common memory system by shared requirements of system. "Singly linked list" is example of 'mutual exclusion'. Preceding node points to next and remove pointer by moving to subsequent end node. Implementation in which list of processes linked is shared among many node processes, simultaneously occurring of two nodes result in problem of system security, consider node *k* along with k + 1 remove because it neither points to head or tail; such that upcoming pointer node k-1 is moved from k+1 which result in pointer to move to next node pointing as k+2. Problem can be overcome by 'mutual exclusion' which needs to update the system requirement.

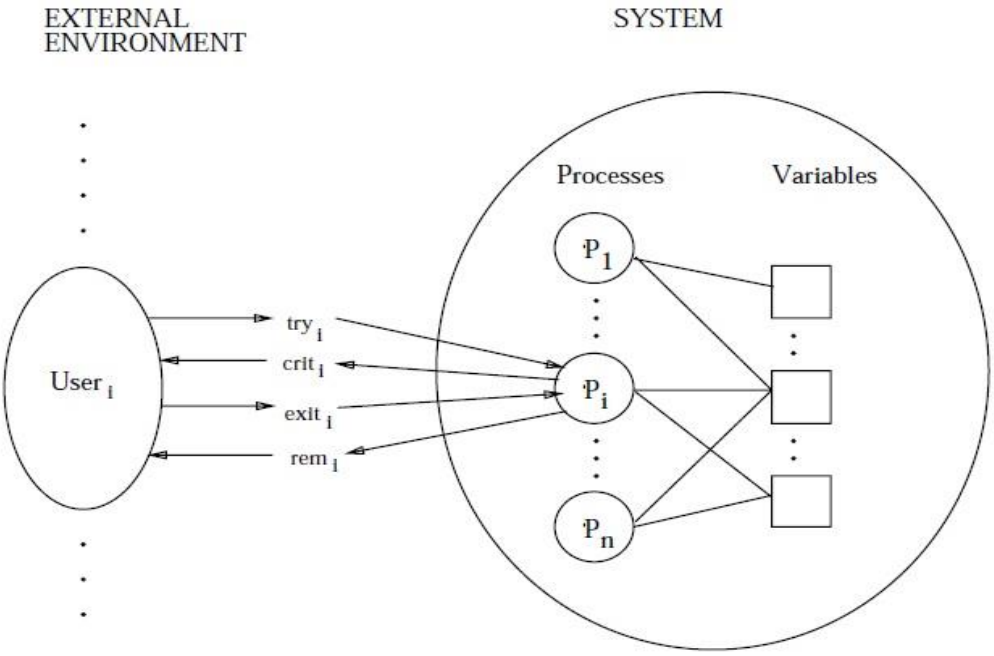

Fig 3.1 Mutual mechanism

Mutual exclusion accomplished both system requirements. Different proposed solutions are provided below,

3.1 H/W Solution

Proposed solution in processor working with single system at demanded time section, in order to overcome 'mutual exclusion'. This prevents process from acquiring interruption services route in executing system, but it determine more problems. In demand of time section every time its impossible to determine the clock control signal when the service is been executed, hence path of time execution lead to interruption of services. Termination of process during

system run time will provide no response to the services system and issued the problem for terminate integrated system. Solution is also obtained by using "busy-wait".

In case of processor executing as multiple or individual system, 'mutual exclusion' provide 'Busy-wait' capability to give solution to issues."Atomic test and set rules" provide solution for exclusion through use of shared system. In processor system executing as individual node the 'test-and instruction' standard is set up only to each process, since system operation is fragmented. In case of issuing the control set standard for processes there can travel to other processes along with using some loop conditions to in order to obtain the set rules for process complete it task and overcome 'mutual exclusion 'problem. Appropriate in use of system halt where process hold lock mechanism to system.

"Compare and swap" data structure is used to operation fragmented system to accommodate exclusion solution. Linked list represent operation to performed using "compare and swap" to obtain solution and accomplish use of "wait free" correlated exclusion to several memory shared system. New code inserted provide to change the list pointer to next node of the linked structure, where individual node is executed at once time in CAS; and deny remaining processes attempting to interrupt execution of the pointer system of linked structure. Exclusion also include the copy of its visited node upon negotiation so that next system process use the copy of node pointer from data linked structure.

3.2 S/W solution

Some issues with hardware, supported to provide solution using application which uses wait and busy system to obtain associated exclusion.

- "Dekker's Algorithm"
- "Peterson's Algorithm"
- "Lamport's bakery Algorithm"
- "Szymanski's Algorithm"
- "Taubenfeld's black-white bakery Algorithm"

As above mentioned data algorithm doesn't execute with system outside the execution mode. Thread concept used by programmer to restrict the operation of memory system.

Application solution is provide to come up solution to issues approached by hardware system, hence application use recur process competence provided by operating system. In case, where lock acquired by other system process and next process want to use for execution by issuing lock approved by process which it holding, the system depend on thread stop the execution using 'context switch' and hand it over to other system process which is waiting in the queue to executed the process system.

IV.ALGORITHMS AND METHADOLOGY

4.1 LAMPORT'S BAKERY ALGORITHM

The "Lamport's bakery algorithm" is proposed algorithm determined by Leslie to is used to provide solution for resources in shared system through exclusion mechanism.

Simultaneous execution of resources is provided by the multiple threads. Writing into memory section which is already initialed will result into data corruption, because memory is utilized before the completion of the execution provided to processes."Lamppost's bakery algorithm is used from several exclusion mechanism to overcome issues of concurrent execution of threads pointing towards time section to deny the problem of data misrepresentation.

4.1.1 Implementation

In Lamport's, *accessing* parameter is considered as *choosing*, following conditions are considered:

- Choosing [i] number [i] included in memory are in initialed to zero.
- The area to number[i] is released.
- The condition of halt in uncritical region resulted in failure of process.
- The range values number [i] is unbounded.
- Process failure occurs at any time which results in halt and entry into critical condition. Arbitrary benefit is provided by going through memory of system. It initials value to zero in memory when read.

The pseudo code for the algorithm is shown below.

```
//Declaration and global variables
        Choosing: array [1..N] bool ={False}
        Num: array [1..N] int={0}
        Lock (int i)
        Choosing[i] true;

        Num[i]= 1+max[num[1]...num[N]);
```

```
For(int j=1; j<=N; j++)
        //wait for thread to recieve number
While (choosing[j])
// wait till all smaller number proceses gets executed
While ((num[j]!=0) && (num[j]) < (num[i] i))
        Thread (int)
        While (true)
        {
                Lock(i)
                // CS
                Unlock(i)
                // non cs
        }
```

4.2 'Dekker's algorithm'

This algorithm is known for its co existence nature which provides solution to exclusion. Provided by mathematician. Some researcher named Dekker given process constant along with its cooperating constant process. Concede different threads to share user requirements without contest memory section for communication. Issues with respect to naive algorithm to determine exclusion mechanism

Critical mechanism is obtained by process outside loop. No process is executing the system then current process entry into system queue. Exclusion mechanism set flag in order to overcome issues of failure system, as well unknowing waiting the process system in queue resulting in critical problem.

Accordingly, if system is provided with variables then it determine which process entry the critical zone, until a process waiting in queue is given priority to execute, where this processes entry into critical zone by breaking loop condition.

Dekker does secure system deadlock, along with starvation. Suppose k0 want to enter the system loop but its halted, there creates the deadlock condition, hence k1 is used for processing in critical zone and set its turn to 0.Actually k0 is broken when turn is not equal to zero. Later time it set enter [0] to correct value along with setting enter [1] to false result. Consequent other process f1 bid to introduce in critical condition so that activity is executed in enter [0] loop. In case, it determines enter [1] to invalid and halt in loop condition. Next process execution will control processing and leave loop condition enter [1] critical zone of system.

```
A: Flag ← 1
While turn! = i do
        If flag[turn] = 0
        Then
                Turn != i
        End
End
Flag ← 2
        For j!= i do
        flag[i] = 2 go to A
End
        //CS
Exit
        Flag[1] ← 0
        // non CS
```

V.RESULTS AND ANALYSIS

Bakery's Algorithm					Dekkar's Algorithm				
Number of Process	Thread	Start Time	End Time	Time Inside Critical Section(ms)	Number of Process	Thread	Start Time	End Time	Time Inside Critical Section
0	0	1.46226E+12	1.46226E+12	0	50	0	1.46607E	1.46607E	1

							+12	+12	
	0	1.46226E+12	1.46226E+12	0		17	1.46607E +12	1.46607E +12	0
	0	1.46226E+12	1.46226E+12	16		0	1.462826 E+12	1.462826 E+12	0
	0	1.46226E+12	1.46226E+12	0					
	0	1.46226E+12	1.46226E+12	96					
	0	1.46226E+12	1.46226E+12	0					
	0	1.46226E+12	1.46226E+12	0					
	0	1.46226E+12	1.46226E+12	38					
	3	1.46226E+12	1.46226E+12	0					
	3	1.46226E+12	1.46226E+12	0					
	3	1.46226E+12	1.46226E+12	30					
	3	1.46226E+12	1.46226E+12	1					
	3	1.46226E+12	1.46226E+12	3					

. Table 5.1. For Result for 50 Process

The analysis and the result of the algorithm is shown in the form of table. We can observe that in bakers the processes are getting faster access to the critical section as compared to the Dekker. The time is in epoch and it is in milliseconds. In ME problem the time the process spends inside the critical section is not under control but the number of processes that gets access to the critical section will give the better performance of the DS.

The study is carries out by taking different number of processes i.e 10, 50, 80, 100 and the result is tabulated above. From the above tables we can observe that the bakery provides much faster access to critical section compared to that of the Dekker. This means access time of CS by other processes is varied observably in both the algorithm.

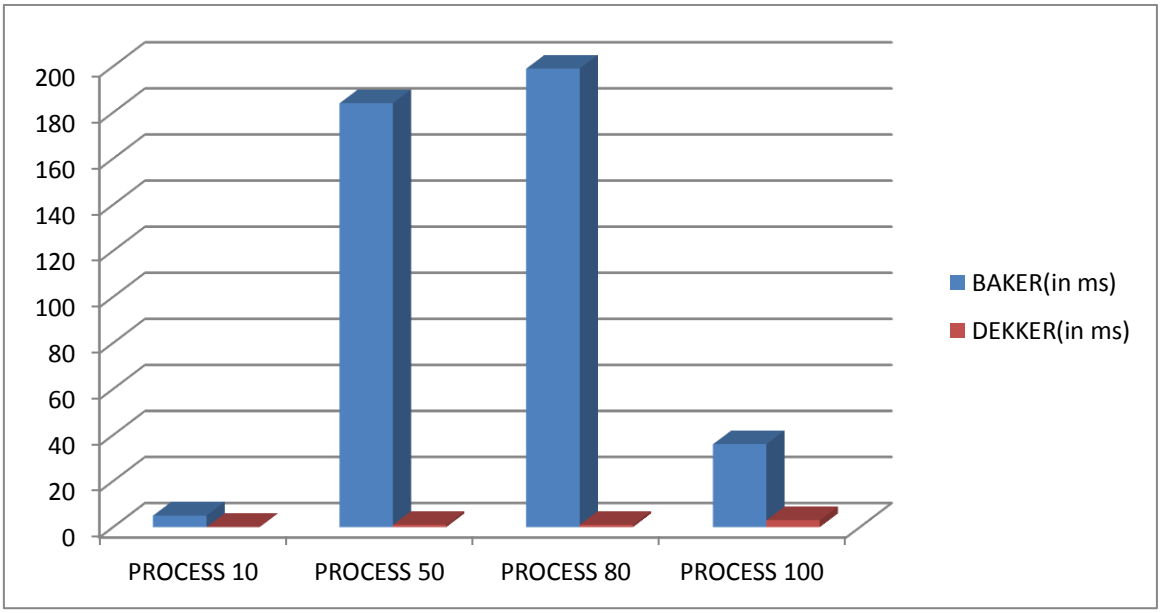

Fig 5.1. bar-graph of analyzed result

Complete table result is summarized in the above graph. Where it is observed that that total amount of time spent in the CS is more in Bakers than the Dekker. Not just the total time but total number of times the process get access to the CS is also more in the Bakers.

VI.CONCULSION

From above analysis and observation of the result it can be stated that bakers algorithm is better when it is compared to the Dekker's algorithm for ME problem. From the result it is observed that the time taken for the next process to enter the CS is much lesser in bakers than the Dekker's program. This makes it much faster and efficient to use. And also the time spent by total processes in CS is higher in bakers.

REFERENCES

[1] Marvin V. Zelkowitz Department of Computer Science, University of Maryland, Coiiege Park, Maryland
[2] Mohammad Rastegari and Amir Masoud Rahmani " Solving Critical Section problem in Distributed system by Entangled Quantum bits1" Department of Computer Engineering, University of Science and Research, Tehran February,2008
[3] HECTOR GARCIA-MOLINA, "Elections in a Distributed Computing System"IEEE TRANSACTIONS ON COMPUTERS, VOL. C-31, NO. 1, JANUARY 1982
[4] Nancy lynch and boaz patt-shamir "distributed algorithms" january 1993..
[5] Jean-Pierre Lozi Florian David Gaël Thomas Julia Lawall Gilles Muller "Remote Core Locking: Migrating Critical-Section Execution to Improve the Performance of Multithreaded Applications" 2012.Stijn Eyerman
[6] Lieven Eeckhout "Modeling Critical Sections in Amdahl's Lawand its Implications for Multicore Design" 2010.