MUTUAL EXCLUSION

PROJECT REPORT

Version 1.0

Contents

Objective

To study, implement and compare the two token based mutual exclusion algorithms for Advanced Operating Systems coursework

1. Suzuki Kazami's algorithm
2. Raymond's tree algorithm

Team

1. Dhwani Raval
2. Vincy Shrine

Algorithms

Raymond's tree algorithm and Suzuki Kazami's algorithm are token-based algorithms for achieving mutual exclusion in distributed systems. They have a similar concept for achieving mutual exclusion. For a process to enter its critical section, it must have a token. When a process wants to enter in its critical section, but it does not have token, it will send a message to other processes asking for token. The process that has the token, if it is not currently in a critical section, will then send the token to the requesting process. Both the algorithms differ in the way the requesting process sends the message to other processes.

1. Suzuki Kazami's Algorithm

In this algorithm, if a process wants to enter its critical section and it does not have the token, it broadcasts a request message to all other processes in the system.

The algorithm uses three data structures – Requesting Array (RN), Satisfied Token Request Array (LN) and a token queue. LN and token queue are an integral part of the token that is being transferred from one process to another. For a process having the token and is idle, if the arrived request from a process is not a stale request and it is on the top of the token queue, then the token would be passed to that process. The algorithm makes use of increasing Request Numbers to also allow messages to arrive out-of-order.

2. Raymond's Tree Algorithm

In this algorithm, the topology of the processes in the distributed system is of the form of a directed tree. The root node has the token. Initially the coordinator acts as a root node. Each process will maintain a holder which will point to its neighbor on the path towards the root node.

This algorithm uses a token queue as its data structure to keep the track of the requesting processes. For a process having the token and is idle, if the arrived request from a process is on the top of the queue, then the token would be passed to that process. The process which possess the token becomes the root node whereas the other processes makes a request to that node.

Project Requirements

After initialization the steps executed by each process are as follows.

1. Wait for a period of uniformly distributed over [t1, t2] msecs. Ensure that you seed each process with different values for the random function to be truly random.

2. Request to enter the critical section using the appropriate algorithm. Print that the process has requested to enter the CS with the process id and time.

3. Enter the critical section. Print that the process has entered the CS with the process id and time. Wait for t3 msecs and then exit the CS.

4. Repeat the steps 1 to 3 several times randomly chosen over the interval [20, 40]. Again, seed appropriately.

5. Send a completed message to the coordinator along with the data that were collected.

6. If you are a coordinator wait for completed messages from all the processes. Then send a terminate message to all the processes. Evaluate the performance of the algorithm and print the appropriate output, and then terminate.

7. All processes terminate on receipt of terminate message

Algorithm Performance

Both the algorithms were executed on dc machines of the University of Texas at Dallas. Following are the factors taken into consideration while recording the outputs:

- Both the algorithms were run for n= 5, 10, 15, 20, 25; where n is the number of processes in the distributed system.
- Every process in both the algorithms makes 20 requests to enter in its critical section. (The number of requests is maintained uniformed for all processes for an accurate comparison between the algorithms).
- The values of T1, T2 and T3 are specified in the config file which are T1 = 500, T2 = 700 and T3 = 1000. (The values are kept same for both algorithms for an accurate comparison between the algorithms).

Every process on completion sends the data to the coordinator, which evaluates average message count, average synchronization delay and average waiting time based on the data received.

The calculations for the following parameters are made as follow:

- **Average Message Count**
  The number of messages sent by a process to other processes to get token is the number of request messages required for a process to enter in its critical section. Every process computes the average messages required per request which is sent to the coordinator. The coordinator takes a sum of all these average messages per request and divides it by the total number of processes.
- **Average Synchronization Delay**

Synchronization Delay is the parameter used to evaluate the performance in the critical section. It is calculated by measuring the time when a process leaves its critical section and another process enters the critical section, i.e. It measures the time wherein the critical section is idle.
Once a process completes CS execution and exits for a request, it sends a packet to coordinator. Coordinator computes the difference between exit time and entry time of consecutive requests and divides by total requests to get average SD.

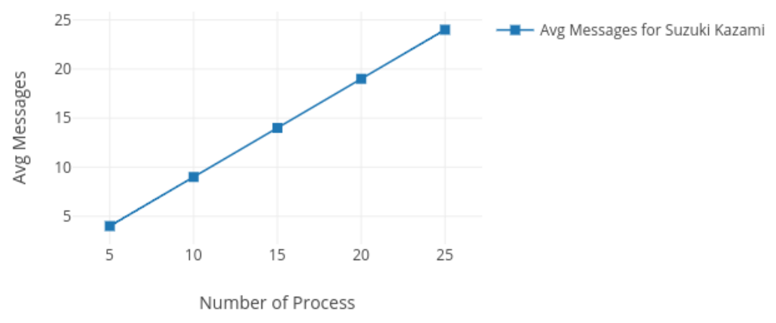
- **Average Waiting Time**
  Waiting Time for a process is the time for which it waits to enter in its critical section. It is calculated by measuring the time difference between the times when the process has requested for a token to enter in its critical section with when the process has received the token. Every process calculates its average waiting time per request and then sends it to the coordinator, which takes the sum of all these average value and divides it by the total number of processes.

The observations made for the asked parameters are as follow:

Suzuki Kazami's Algorithm

1. **Average Number of Messages**

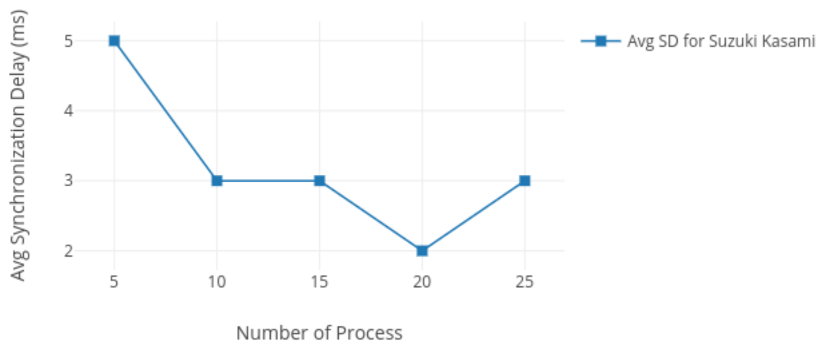
Number of Processes	Average Message Count
5	4
10	9
15	14
20	19
25	24



2. **Average Synchronization Delay**

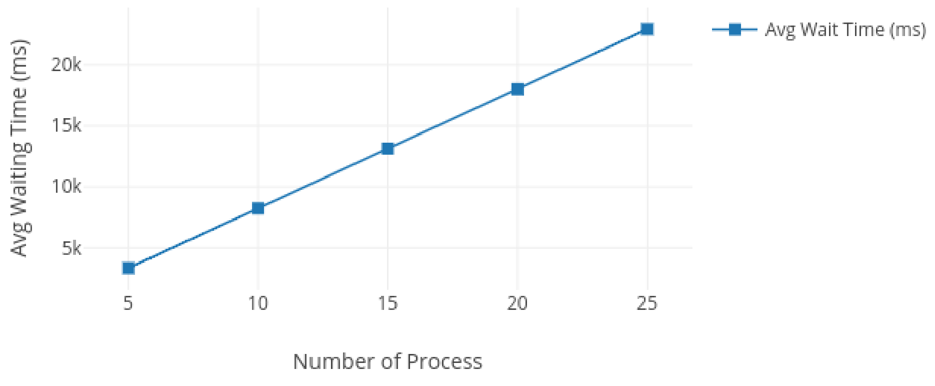
Number of Processes	Average SD (ms)
5	5
10	3
15	3
20	2

25	3



3. Average Waiting Time

Number of Processes	Average Waiting Time (ms)
5	3352
10	8245
15	13136
20	18011
25	22922

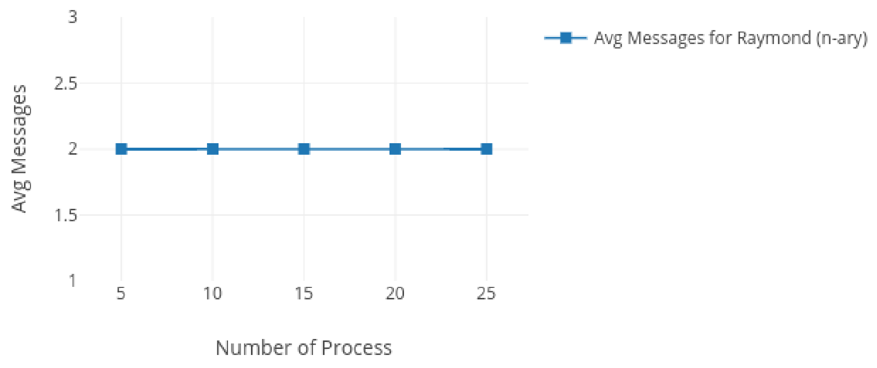


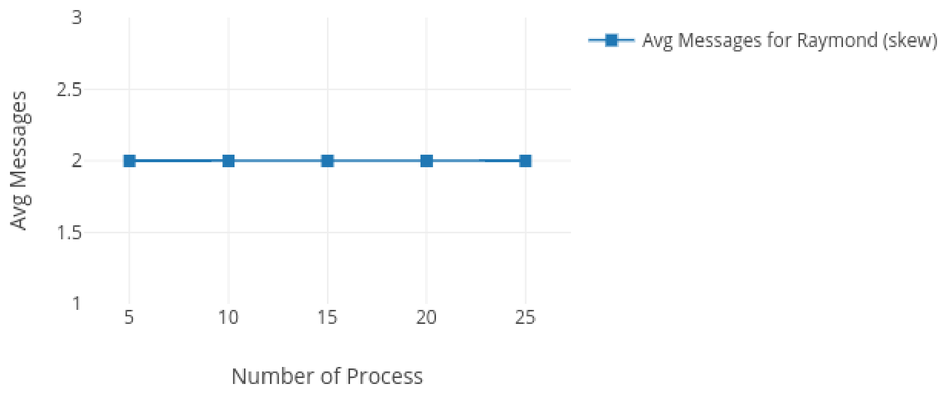
Raymond Tree's Algorithm

For Raymond's tree algorithm, the observations are performed taking two types of initial tree configuration into consideration: n-ary trees and skew tree

1. Average Number of Messages

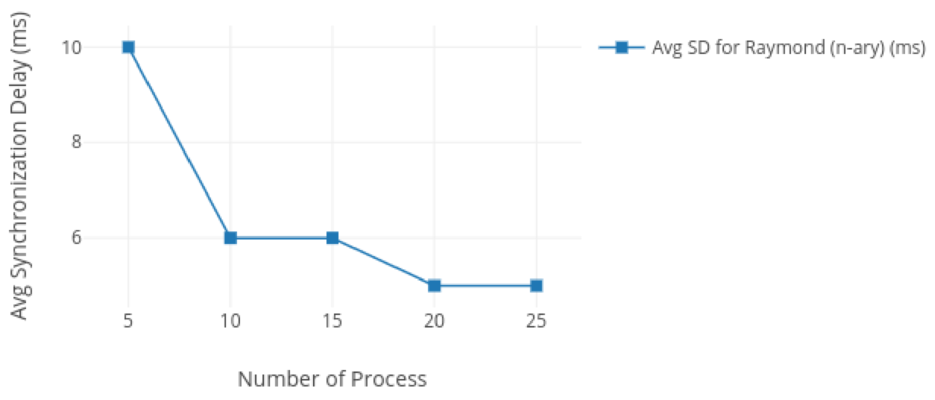
Number of Processes	Average Messages	
	N- ary tree	Skew Tree
5	2	2
10	2	2
15	2	2
20	2	2
25	2	2

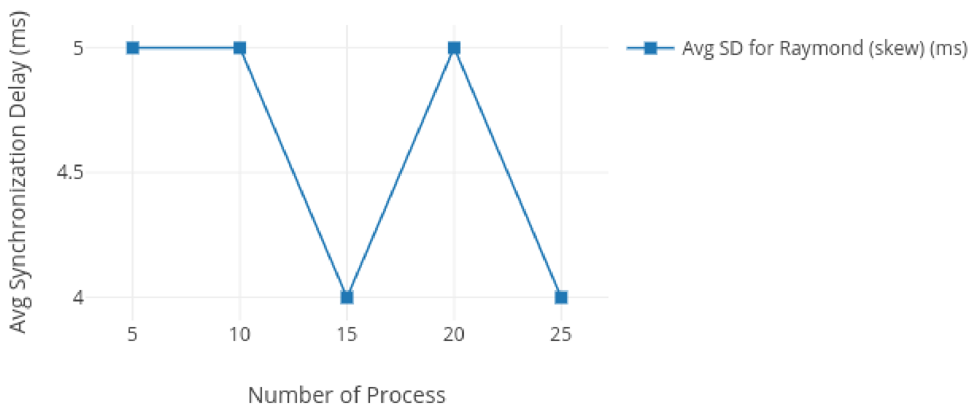




2. Average Synchronization Delay

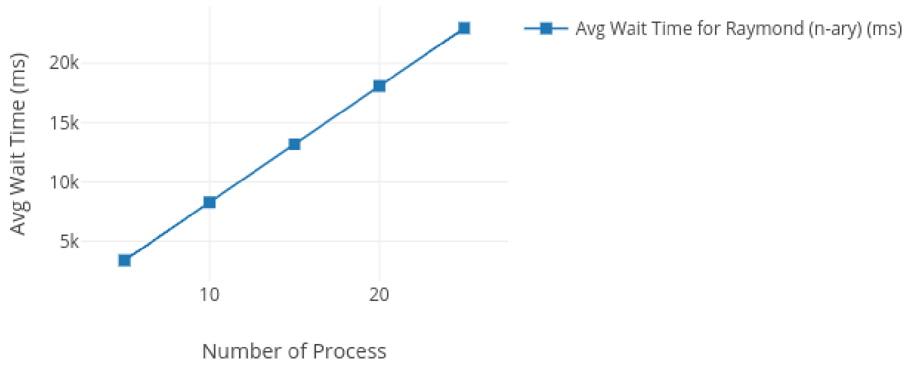
Number of Processes	Average SD (ms)	
	N- ary tree	Skew Tree
5	10	5
10	6	5
15	6	4
20	5	5
25	5	4

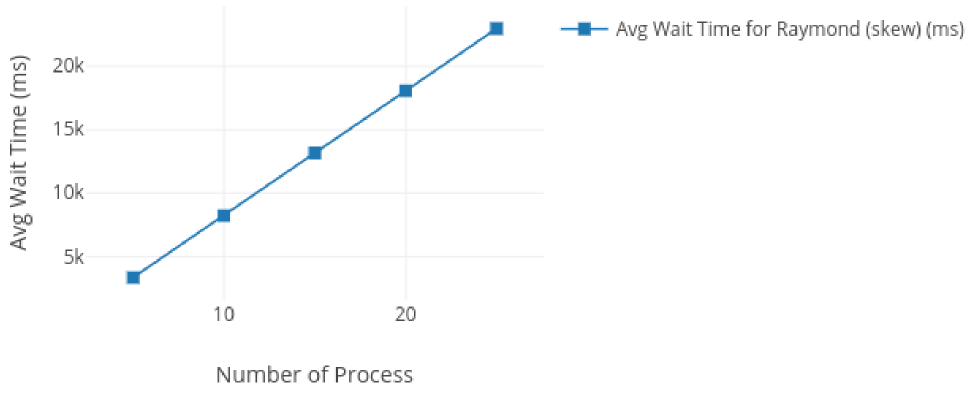




3. Average Waiting Time

Number of Processes	Average Messages (ms)	
	N- ary tree	Skew Tree
5	3380	3354
10	8280	8216
15	13171	13162
20	18084	18066
25	22971	22967





Performance Comparison

Number of messages

The average number of messages sent is less in Raymond’s tree algorithm compared to Suzuki Kazami’s algorithm.

For Raymond’s tree algorithm it is O (log N).

For Suzuki Kazami’s algorithm it is O (N).

Average Synchronization Delay

The average synchronization delay is less in Suzuki Kazami’s algorithm compared to Raymond’s tree algorithm.

For Raymond’s tree algorithm it is O (T log N)/2.

For Suzuki Kazami’s algorithm it is 0 or one message propagation delay.

Average Waiting Time

The average wait time for different number of processes for both the algorithms is nearly same.

Revision History

Date	Version	Changes	Authors
11/02/2017	1.0	Report Draft	Dhwani Raval, Vincy Shrine