Experiment 7– Token Based Algorithm

Learning Objective: Student should be able to design a program to illustrate token based algorithms.

Tools : Java

Theory:

Mutual Exclusion

There are several resources in a system that must not be used simultaneously by multiple processes if program operation is to be correct. For example, a file must not be simultaneously updated by multiple processes. Similarly, use of unit record peripherals such as tape drives or printers must be restricted to a single process at a time. Therefore, exclusive access to such a. shared resource by a process must be ensured. This exclusiveness of access is called mutual exclusion between processes. The sections of a program that need exclusive access to shared resources are referred to as critical sections. For mutual exclusion, means are introduced to prevent processes from executing concurrently within their associated critical sections.
An algorithm for implementing mutual exclusion must satisfy the following requirements:

1. Mutual exclusion. Given a shared resource accessed by multiple concurrent processes, at any time only one process should access the resource. That is, a process that has been granted the resource must release it before it can be granted to another process.

2. No starvation. If every process that is granted the resource eventually releases it, every request must be eventually granted.

In single-processor systems, mutual exclusion is implemented using semaphores, monitors, and similar constructs.

Suzuki–Kasami Algorithm

Suzuki–Kasami algorithm is a token-based algorithm for achieving mutual exclusion in distributed systems.This is modification of Ricart–Agrawala algorithm, a permission based (Non-token based) algorithm which uses **REQUEST** and **REPLY** messages to ensure mutual exclusion.

In token-based algorithms, A site is allowed to enter its critical section if it possesses the unique token. Non-token based algorithms uses timestamp to order requests for the critical section where as sequence number is used in token based algorithms.

Data structure and Notations:

- An array of integers **RN[1…N]**

  A site Si keeps **RNi[1…N]**, where **RNi[j]** is the largest sequence number received so far through **REQUEST** message from site Si.

- An array of integer **LN[1…N]**

  This array is used by the token.**LN[J]** is the sequence number of the request that is recently executed by site Sj.

- A queue **Q**
  This data structure is used by the token to keep a record of ID of sites waiting for the token

Algorithm:

- **To enter Critical section:**
  - When a site Si wants to enter the critical section and it does not have the token then it increments its sequence number **RNi[i]** and sends a request message **REQUEST(i, sn)** to all other sites in order to request the token. Here **sn** is update value of **RNi[i]**
  - When a site Sj receives the request message **REQUEST(i, sn)** from site Si, it sets **RNj[i]** to maximum of **RNj[i]** and **sn** i.e **RNj[i]** = max(**RNj[i]**, **sn**).
  - After updating **RNj[i]**, Site Sj sends the token to site Si if it has token and **RNj[i]** = **LN[i]** + 1
- **To execute the critical section:**
  - Site Si executes the critical section if it has acquired the token.
- **To release the critical section:**

  After finishing the execution Site Si exits the critical section and does following:
  - sets **LN[i]** = **RNi[i]** to indicate that its critical section request **RNi[i]** has been executed
  - For every site Sj, whose ID is not prsent in the token queue **Q**, it appends its ID to **Q** if **RNi[j]** = **LN[j]** + 1 to indicate that site Sj has an outstanding request.

- After above updation, if the Queue **Q** is non-empty, it pops a site ID from the **Q** and sends the token to site indicated by popped ID.
- If the queue **Q** is empty, it keeps the token

Message Complexity:

The algorithm requires 0 message invocation if the site already holds the idle token at the time of critical section request or maximum of N message per critical section execution. This N messages involves

- (N – 1) request messages
- 1 reply message

Drawbacks of Suzuki–Kasami Algorithm:

- **Non-symmetric Algorithm:** A site retains the token even if it does not have requested for critical section. According to definition of symmetric algorithm "No site possesses the right to access its critical section when it has not been requested."

Result and Discussion:

Code:

```
import java.util.concurrent.Semaphore;

public class SuzukiKasami {
    private static final int NUM_NODES = 5; // Number of nodes in the distributed system
    private static final int NUM_REQUESTS = 10; // Number of requests for each node

    private static int[] request = new int[NUM_NODES];
    private static int[] token = new int[NUM_NODES];
    private static int[][] quorum = {
        {1, 2, 4},  // Quorum for node 0
        {0, 2, 3},  // Quorum for node 1
        {0, 1, 3},  // Quorum for node 2
        {1, 2, 4},  // Quorum for node 3
        {0, 3, 4}   // Quorum for node 4
    };

    private static Semaphore mutex = new Semaphore(1);

    public static void main(String[] args) {
        for (int i = 0; i < NUM_NODES; i++) {
            final int nodeId = i;
            new Thread(() -> {
```

```
            for (int j = 0; j < NUM_REQUESTS; j++)
{
                requestCriticalSection(nodeId);
                // Simulate critical section execution
                System.out.println("Node " + nodeId + "
is in critical section");
                try {
                    Thread.sleep((long) (Math.random()
* 1000));
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                releaseCriticalSection(nodeId);
            }
        }).start();
    }
}


    private static void requestCriticalSection(int
nodeId) {
        try {
            mutex.acquire();
            request[nodeId] = 1;
            for (int i : quorum[nodeId]) {
                if (i != nodeId) {
                    while (request[i] == 1) {
                        mutex.release();
                        Thread.sleep(10); // Adjust as needed
                        mutex.acquire();
                    }
                }
            }
            token[nodeId] = 1;
            mutex.release();
            for (int i : quorum[nodeId]) {
                if (i != nodeId) {
                    while (token[i] == 0) {
                        Thread.sleep(10); // Adjust as needed
                    }
                }
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    private static void releaseCriticalSection(int
nodeId) {
        try {
            mutex.acquire();
            token[nodeId] = 0;
            request[nodeId] = 0;
            mutex.release();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

The output will consist of lines indicating when each node enters and exits the critical section.

Node 0 is in critical section

Node 1 is in critical section

Node 2 is in critical section

Node 0 is in critical section

Node 3 is in critical section

Node 1 is in critical section

Node 4 is in critical section

Node 2 is in critical section

Node 0 is in critical section

...

The Suzuki-Kasami Algorithm ensures mutual exclusion in a distributed system by coordinating access to a critical section among multiple nodes. It achieves this by following these key steps Request phase, Token phase, Critical Section Execution, Release phase.

Learning Outcomes: The student should have the ability to

LO1: Recall the different token based algorithm.

LO2: Apply the different token based algorithm.

Course Outcomes: Upon completion of the course students will be able to understand token based Algorithm.

Conclusion:

The Suzuki-Kasami Algorithm ensures mutual exclusion in a distributed system by coordinating access to a critical section among multiple nodes. It achieves this by following these key steps Request phase, Token phase, Critical Section Execution, Release phase as implemented in the code.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				