Experiment 8– Non-Token Based Algorithm

Learning Objective: Student should be able to design a program to illustrate non-token based algorithm.

Tools : Java

Theory:

Non-token based approach:

- A site communicates with other sites in order to determine which sites should execute critical section next. This requires exchange of two or more successive round of messages among sites.
- This approach use timestamps instead of sequence number to order requests for the critical section.
- When ever a site make request for critical section, it gets a timestamp. Timestamp is also used to resolve any conflict between critical section requests.
- All algorithm which follows non-token based approach maintains a logical clock. Logical clocks get updated according to Lamport's scheme

Example:

- Lamport's algorithm, Ricart–Agrawala algorithm

Ricart–Agrawala algorithm

Ricart–Agrawala algorithm is an algorithm to for mutual exclusion in a distributed system proposed by Glenn Ricart and Ashok Agrawala. This algorithm is an extension and optimization of Lamport's Distributed Mutual Exclusion Algorithm. Like Lamport's Algorithm, it also follows permission based approach to ensure mutual exclusion.

In this algorithm:

- Two type of messages (**REQUEST** and **REPLY**) are used and communication channels are assumed to follow FIFO order.
- A site send a **REQUEST** message to all other site to get their permission to enter critical section.
- A site send a **REPLY** message to other site to give its permission to enter the critical section.
- A timestamp is given to each critical section request using Lamport's logical clock.
- Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section request is always in the order of their timestamp.

- **To enter Critical section:**

- When a site Si wants to enter the critical section, it send a timestamped **REQUEST** message to all other sites.
- When a site Sj receives a **REQUEST** message from site Si, It sends a **REPLY** message to site Si if and only if
  - Site Sj is neither requesting nor currently executing the critical section.
  - In case Site Sj is requesting, the timestamp of Site Si's request is smaller than its own request.
- Otherwise the request is deferred by site Sj.

- **To execute the critical section:**
  - Site Si enters the critical section if it has received the **REPLY** message from all other sites.
- **To release the critical section:**
  - Upon exiting site Si sends **REPLY** message to all the deferred requests.

Message Complexity:
Ricart–Agrawala algorithm requires invocation of 2(N – 1) messages per critical section execution. These 2(N – 1) messages involves

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

Drawbacks of Ricart–Agrawala algorithm:

- **Unreliable approach:** failure of any one of node in the system can halt the progress of the system. In this situation, the process will starve forever. The problem of failure of node can be solved by detecting failure after some timeout.

Result and Discussion:

Code:

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class Message {
    String messageType;
    int timestamp;
    int siteId;

    Message(String messageType, int timestamp, int siteId) {
        this.messageType = messageType;
        this.timestamp = timestamp;
        this.siteId = siteId;
```

```
    }
}

class Site {
    int siteId;
    boolean requesting;
    boolean executing;
    int timestamp;
    List<Message> deferredQueue;

    Site(int siteId) {
        this.siteId = siteId;
        this.requesting = false;
        this.executing = false;
        this.timestamp = 0;
        this.deferredQueue = new ArrayList<>();
    }

    void requestCriticalSection(List<Site> sites) {
        this.requesting = true;
        this.timestamp++;
        for (Site site : sites) {
            if (site.siteId != this.siteId) {
                Message requestMessage = new Message("REQUEST", this.timestamp, this.siteId);
                sendMessage(requestMessage, site);
            }
        }
        waitForReplies(sites);
    }

    void sendMessage(Message message, Site destination) {
        System.out.println("Site " + this.siteId + " sends " + message.messageType + " message
to Site " + destination.siteId);
        destination.receiveMessage(message, this);
    }

    void receiveMessage(Message message, Site sender) {
        System.out.println("Site " + this.siteId + " receives " + message.messageType + " message
from Site " + sender.siteId);
        if (message.messageType.equals("REQUEST")) {
            if (!this.requesting && !this.executing) {
                this.sendMessage(new Message("REPLY", 0, this.siteId), sender);
            } else if (this.requesting && message.timestamp < this.timestamp) {
                this.deferredQueue.add(message);
            }
        } else if (message.messageType.equals("REPLY")) {
            if (this.requesting) {
```

```
            this.deferredQueue.removeIf(m -> m.siteId == sender.siteId);
            if (this.deferredQueue.isEmpty()) {
                this.executing = true;
                System.out.println("Site " + this.siteId + " enters critical section.");
            }
        }
    }
}

void waitForReplies(List<Site> sites) {
    int repliesExpected = sites.size() - 1;
    int repliesReceived = 0;
    while (repliesReceived < repliesExpected) {
        // Wait for replies
    }
}

void releaseCriticalSection(List<Site> sites) {
    this.requesting = false;
    this.executing = false;
    for (Site site : sites) {
        if (site.siteId != this.siteId) {
            for (Message message : this.deferredQueue) {
                this.sendMessage(new Message("REPLY", 0, this.siteId), site);
            }
        }
    }
    this.deferredQueue.clear();
    System.out.println("Site " + this.siteId + " releases critical section.");
}
}

public class RicartAgrawalaAlgorithm {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of sites: ");
        int numberOfSites = scanner.nextInt();

        List<Site> sites = new ArrayList<>();
        for (int i = 0; i < numberOfSites; i++) {
            sites.add(new Site(i+1));
        }

        for (Site site : sites) {
            site.requestCriticalSection(sites);
            site.releaseCriticalSection(sites);
        }
```

```
  }
}
```

Output:

```
Enter the number of sites:
3
Site 1 sends REQUEST message to Site 2
Site 2 receives REQUEST message from Site 1
Site 2 sends REPLY message to Site 1
Site 1 receives REPLY message from Site 2
Site 1 enters critical section.
Site 1 sends REQUEST message to Site 3
Site 3 receives REQUEST message from Site 1
Site 3 sends REPLY message to Site 1
Site 1 receives REPLY message from Site 3
Site 1 enters critical section.
PS C:\Users\tiwar\OneDrive\Desktop>
```

Learning Outcomes: The student should have the ability to

LO1: Recall the non token based algorithm.

LO2: Analyze the different non token based algorithm

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

Conclusion:

For Faculty Use

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