# Lab Report: Distributed Mutual Exclusion using Ricart-Agrawala Algorithm

Course: Distributed Systems

Instructor: [Your Instructor’s Name]

Submitted by: [Your Name]

Date: [Submission Date]

## 1. Objective

To implement and analyze the Ricart-Agrawala algorithm for distributed mutual exclusion using Java socket programming, ensuring:  
✔ Mutual Exclusion (only one process in CS at a time).  
✔ Deadlock Freedom (no indefinite waiting).  
✔ Fairness (requests granted in order of timestamps).

## 2. Problem Statement

Design a multi-process simulation where processes coordinate access to a critical section (CS) using:  
  
- REQUEST messages (before entering CS).  
- REPLY messages (granting permission).  
- Logical Clocks (Lamport timestamps for ordering).

## 3. Tools & Technologies

• Java JDK 8+ (Socket Programming, Multithreading).  
• Terminal / Command Prompt (for execution).  
• Wireshark (Optional) (for network traffic analysis).

## 4. Implementation

### 4.1 Key Components

|  |  |
| --- | --- |
| Component | Description |
| RicartAgrawalaProcess | Main class handling message passing and CS access. |
| ServerSocket | Listens for incoming messages. |
| Lamport Clock | Logical clock for event ordering. |
| REQUEST/REPLY | Message types for mutual exclusion. |
| Deferred Queue | Holds pending requests when CS is occupied. |

### 4.2 Algorithm Steps

Requesting CS:  
- Increment logical clock.  
- Send REQUEST(ts, pid) to all processes.  
  
Granting CS:  
- Wait for REPLY from all processes.  
- Enter CS if all replies received.  
  
Releasing CS:  
- Exit CS and send REPLY to deferred requests.

### 4.3 Code Snippet (Critical Part)

```java  
private void handleRequest(int timestamp, int senderPid) {  
 boolean shouldDefer = wantCS && (timestamp > clock || (timestamp == clock && senderPid > pid));  
 if (shouldDefer) {  
 deferredRequests.add(new Request(timestamp, senderPid)); // Defer reply  
 } else {  
 sendReply(senderPid); // Reply immediately  
 }  
}  
```

## 5. Experiment & Results

### 5.1 Setup

Processes: 3 (P0, P1, P2).  
Ports: 5000, 5001, 5002.  
Test Case: All processes randomly request CS.

### 5.2 Output Observations

Process 0 requesting CS at time 1  
Process 0 ENTERING critical section.  
Process 1 requesting CS at time 2 (Deferred)  
Process 0 EXITING critical section.  
Process 1 ENTERING critical section.  
  
✅ Mutual Exclusion: Only P0 was in CS initially.  
✅ No Starvation: P1 entered CS after P0 exited.

### 5.3 Challenges Faced

Socket Connection Refused: Fixed by ensuring all processes start simultaneously.  
Race Conditions: Resolved using ConcurrentHashMap for thread-safe replies tracking.

## 6. Conclusion

The Ricart-Agrawala algorithm successfully enforces distributed mutual exclusion.  
Logical clocks prevent race conditions.  
Future Work: Extend to handle process failures.

## 7. References

Ricart, G., & Agrawala, A. K. (1981). "An Optimal Algorithm for Mutual Exclusion in Computer Networks."  
Lamport, L. (1978). "Time, Clocks, and the Ordering of Events in a Distributed System."

## Appendix

GitHub Link: [Your Repository URL]  
Execution Video: [Link if recorded]