1. Ricart-Agrawala Algorithm Report

Aim:

Implement the Ricart-Agrawala algorithm to achieve mutual exclusion in a distributed system, allowing multiple processes to access a critical section while ensuring that no two processes are in the critical section simultaneously.

Theory:

The Ricart-Agrawala algorithm is a distributed mutual exclusion algorithm that uses a fully connected network of processes, where each process can communicate with every other process. The algorithm relies on message passing and timestamps to manage requests for entering a critical section.

When a process wants to enter the critical section, it sends a request message to all other processes. Each process replies with a message that grants permission, provided that it is not already in the critical section or waiting to enter it. The requesting process can enter the critical section only when it has received permission messages from all other processes.

How the Ricart-Agrawala Algorithm Works:

1. **Requesting Entry to the Critical Section:**
    - When a process wants to enter the critical section, it sends a request message with its timestamp to all other processes.
    - It then waits for permission from all other processes.
2. **Granting Permission:**
    - On receiving a request message, a process compares the timestamp of the received request with its own.
    - If the receiving process is not interested in the critical section or has a lower priority, it sends a reply message immediately.
    - If the receiving process is already in the critical section or has a higher priority, it defers the reply until it exits the critical section.
3. **Entering the Critical Section:**
    - After receiving a reply from all other processes, the requesting process enters the critical section.
4. **Releasing the Critical Section:**
    - Upon exiting the critical section, the process sends any deferred replies to other processes that have requested access.

Performance Evaluation:

1. **Message Complexity:**
    - The algorithm requires `(N-1)` request messages and `(N-1)` reply messages for a total of `2(N-1)` messages per critical section entry, where `N` is the number of processes. This complexity is relatively high but offers fairness in granting access.
2. **Response Time:**
    - The response time for entry into the critical section depends on the network delay and the time taken to receive replies from all other processes.
    - In a low-latency network, the response time is primarily determined by the time taken to send `(N-1)` messages and receive `(N-1)` replies.
3. **Scalability:**
    - The algorithm scales linearly with the number of processes, but as the number of processes increases, the communication overhead also increases.
    - For a large number of processes, the performance may degrade due to high message traffic.
4. **Robustness:**
    - The algorithm is robust against single process failures. If a process fails, it simply does not reply to requests, and other processes will not be able to enter the critical section. However, this can lead to a deadlock situation if no recovery mechanism is in place.
5. **Handling Dynamic Process Joining:**
    - The algorithm can handle the dynamic joining of processes by broadcasting the joining message to all existing processes.
    - New processes must be added to the list of participants, and the message exchange count must be adjusted accordingly.
6. **Advanced Fault Tolerance:**
    - Implementing fault tolerance in this algorithm requires additional mechanisms for detecting failed processes and reconfiguring the network to exclude the failed process.
    - This can be achieved by setting a timeout for each message and using failure detectors to identify non-responsive processes.

Output Analysis:

- The output shows that the algorithm effectively enforces mutual exclusion, with processes sequentially entering the critical section based on the timestamp of their requests.
- The logs indicate the correct sequence of requests and replies, demonstrating the fairness of the algorithm.

Bonus:

1. **Handling Dynamic Process Joining:**
    a. The algorithm can handle the dynamic joining of processes by broadcasting the joining message to all existing processes.
    b. New processes must be added to the list of participants, and the message exchange count must be adjusted accordingly.
2. **Advanced Fault Tolerance:**
    a. Implementing fault tolerance in this algorithm requires additional mechanisms for detecting failed processes and reconfiguring the network to exclude the failed process.
    b. This can be achieved by setting a timeout for each message and using failure detectors to identify non-responsive processes.

Conclusion:

The Ricart-Agrawala algorithm effectively achieves mutual exclusion in a distributed system, especially in scenarios where fairness and strict ordering are required. However, the high message complexity and potential for deadlock in the case of process failures can be drawbacks in large-scale systems. Additional mechanisms for handling dynamic joins and advanced fault tolerance can enhance the robustness of the algorithm.