BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Empowered Autonomous Institute Affiliated to Mumbai University)
Department Of Computer Engineering

Name	Mayur Solankar, Manish Jadhav, Vishesh Savani, Shreyansh Salvi
UID	2023301018, 2023301005, 2022300100, 2022300091
Subject	Distributed Computing
Experiment No.	6
Project title	Social Media System
Problem Statement	To implement mutual exclusion algorithm
Objectives	Ensures only one user or process accesses a shared resource at a time, preventing conflicts and maintaining data consistency in a social media platform.
Theory	**MUTUAL EXCLUSION** What is a Mutual Exclusion ? In the context of a social media platform, mutual exclusion refers to a synchronization mechanism that ensures only one user or process can access and modify shared resources, such as posts, comments, or user profiles, at a given time. This prevents conflicts and inconsistencies that might arise if multiple users attempt to modify the same data simultaneously. By using mutual exclusion, the social media platform maintains data integrity, preserves the order of user interactions, and prevents race conditions, ensuring a seamless and coherent user experience. Mutual exclusion is a fundamental problem in distributed computing systems. Mutual exclusion ensures that concurrent access of processes to a shared resource or data is serialized, that is, executed in a mutually exclusive manner. Mutual exclusion in a distributed system states that only one process is allowed to execute the critical section (CS) at any given time. In a distributed system, shared variables (semaphores) or a local kernel cannot be used to implement mutual exclusion. Message passing is the sole means for implementing distributed mutual exclusion. The decision as to which process is allowed access to the CS next is arrived at by message passing, in which each process learns about the state of all other processes in some consistent way. The design of distributed mutual exclusion algorithms is complex because these algorithms must deal with unpredictable message delays and incomplete knowledge of the system state. There are three basic approaches for implementing distributed mutual exclusion:

1. Token-based approach.
2. Non-token-based approach.
3. Quorum-based approach.

In the token-based approach, a unique token (also known as the PRIVILEGE message) is shared among the sites. A site is allowed to enter its CS if it possesses the token, and it continues to hold the token until the execution of the CS is over. Mutual exclusion is ensured because the token is unique.

Requirements of Mutual exclusion Algorithm:
•No Deadlock: Two or more sites should not endlessly wait for any message that will never arrive.
• No Starvation: Every site who wants to execute critical section should get an opportunity to execute it in finite time. Any site should not wait indefinitely to execute critical section while other site are repeatedly executing critical section
•Fairness: Each site should get a fair chance to execute critical section. Any request to execute critical section must be executed in the order they are made i.e. Critical section execution requests should be executed in the order of their arrival in the system.
•Fault Tolerance: In case of failure, it should be able to recognize it by itself to continue functioning without any disruption.

Code

Server Code :-

```
import socket
import threading

friend_requests = []
connections = []
logical_clock = 0
requesting = False

def handle_client(client_socket, client_logical_clock):
    global logical_clock, requesting

    while True:
        request = client_socket.recv(1024).decode()

        if not request:
            break  # Client disconnected

        # Update logical clocks
        client_logical_clock = max(client_logical_clock,
int(request.split(":")[1]))
```

```
        logical_clock = max(logical_clock, client_logical_clock) + 1

        if request.startswith("request:"):
            sender_clock = int(request.split(":")[1])
            friend_requests.append((client_socket, sender_clock))
            print("Received request from client at logical clock
{}".format(sender_clock))

            # If this node is also requesting and should process its request
first
            if requesting and (sender_clock, client_socket) < (logical_clock,
connections[0]):
                friend_requests.pop(0)
                friend_requests.append((client_socket, sender_clock))
            else:
                client_socket.send("ack:{}".format(logical_clock).encode())

        elif request.startswith("release:"):
            sender_clock = int(request.split(":")[1])
            print("Received release from client at logical clock
{}".format(sender_clock))

            if (client_socket, sender_clock) in friend_requests:
                friend_requests.remove((client_socket, sender_clock))
            else:
                print("Release not found in friend_requests")

        elif request.startswith("exit:"):
            client_socket.close()
            connections.remove(client_socket)
            print("Client " + str(client_socket) + " disconnected")
            break


def accept_connections():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(("0.0.0.0", 8080))
    server.listen(5)

    while True:
        client_socket, addr = server.accept()
        connections.append(client_socket)
        print("Accepted connection from " + str(addr))
```

```
        client_logical_clock = 0  # Reset client_logical_clock for each new
client
        client_handler = threading.Thread(target=handle_client,
args=(client_socket, client_logical_clock))
        client_handler.start()



accept_connections()
```

Client :-

```
import socket

logical_clock = 0  # Initialize logical clock here

def send_request_to_server(request_type, server_address="127.0.0.1",
server_port=8080):
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((server_address, server_port))

    global logical_clock  # Declare logical_clock as global

    if request_type == "request_friend":
        logical_clock += 1  # Increment logical clock when making a request
        request_message = "request:" + str(logical_clock)

    elif request_type == "release":
        logical_clock += 1  # Increment logical clock when releasing
        request_message = "release:" + str(logical_clock)

    elif request_type == "exit":
        request_message = "exit:" + str(logical_clock)

    else:
        print("Invalid request type.")
        return

    client.send(request_message.encode())

    if request_type == "request_friend":
        print("Friend request sent to the server")
        response = client.recv(1024).decode()
        sender_clock = int(response.split(":")[1])
```

```
        print("Received acknowledgment from server at logical clock " +
str(sender_clock))

    elif request_type == "exit":
        print("Exiting...")
        client.close()


while True:
    print("1. Send Friend Request")
    print("2. Release")
    print("3. Exit")

    choice = input("Enter your choice: ")

    if choice == "1":
            send_request_to_server("request_friend")
    elif choice == "2":
            send_request_to_server("release")
    elif choice == "3":
            send_request_to_server("exit")
            break
    else:
            print("Invalid choice. Please try again.")
```

Output:

```
PS C:\Users\vishe\OneDrive\Desktop\DC Codes\Experiment 6> python mutual_exclusion_client.py
1. Send Friend Request
2. Release
3. Exit
Enter your choice: 1
Friend request sent to the server
Received acknowledgment from server at logical clock 2
1. Send Friend Request
2. Release
3. Exit
Enter your choice: 1
Friend request sent to the server
Received acknowledgment from server at logical clock 3
1. Send Friend Request
2. Release
3. Exit
Enter your choice: 1
Friend request sent to the server
Received acknowledgment from server at logical clock 4
1. Send Friend Request
2. Release
3. Exit
Enter your choice: 2
1. Send Friend Request
2. Release
3. Exit
Enter your choice: 1
Friend request sent to the server
Received acknowledgment from server at logical clock 6
1. Send Friend Request
2. Release
3. Exit
Enter your choice: 3
```

Conclusion:

Hence by completing we came to about implementation of mutual exclusion in social media platform.