|  |  |
| --- | --- |
| **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:** |  |
| **Conclusion:** | Hence by completing we came to about implementation of mutual exclusion in social media platform. |