LAB – 2

- 14 AUG 2023

1. Write a simple TCP iterative server and client to evaluate the given expression. The client enters an infix expression and sends to the server. There may or may not be spaces before and after the operators. For example, all of the following are valid expressions to enter: “13 + 42*5”, “10+2/4”, “5 + 6 - 3”. The server evaluates the postfix expression and sends the result back to the client. The client displays the result on the screen. It then prompts the user to enter the next expression.

Server.py

```
import socket
import re
import operator

operators = {
    '+': operator.add,
    '-': operator.sub,
    '*': operator.mul,
    '/': operator.truediv
}

def evaluate_expression(expression):
    tokens = re.findall(r'\d+|\S', expression)
    stack = []

    for token in tokens:
        if token.isdigit():
            stack.append(int(token))
        elif token in operators:
            if len(stack) < 2:
                return "Invalid expression"
            else:
                op2 = stack.pop()
                op1 = stack.pop()
                result = operators[token](op1, op2)
                stack.append(result)
        else:
            return "Invalid expression"

    if len(stack) != 1:
        return "Invalid expression"

    return stack[0]

def main():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(('127.0.0.1', 12345))
    server.listen(1)

    print("Server is listening...")

    while True:
        conn, addr = server.accept()
        print(f"Connected by {addr}")

        data = conn.recv(1024).decode()
        if not data:
            break

        result = evaluate_expression(data)
        conn.send(str(result).encode())

        conn.close()

if __name__ == "__main__":
    main()
```

client.py

```
import socket

def main():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('127.0.0.1', 12345))

    while True:
        expression = input("Enter an expression: ")
        client.send(expression.encode())

        result = client.recv(1024).decode()
        print(f"Result: {result}")

    client.close()

if __name__ == "__main__":
    main()
```

2. Implement a mini chat app using socket programming. Clients must be able to chat with each other (as users can in WhatsApp and InstaGram). The server must supervise all these chats and enable communication between them.

Server.py

```
import socket
import threading

clients = []

def handle_client(client_socket):
    while True:
        try:
            message = client_socket.recv(1024).decode()
            if not message:
                remove_client(client_socket)
                break

            broadcast(message, client_socket)
        except:
            remove_client(client_socket)
            break

def broadcast(message, sender_socket):
    for client in clients:
        if client != sender_socket:
            try:
                client.send(message.encode())
            except:
                continue

def remove_client(client_socket):
    if client_socket in clients:
        clients.remove(client_socket)

def main():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(('127.0.0.1', 12345))
    server.listen(5)

    print("Server is listening...")

    while True:
        client_socket, client_addr = server.accept()
        clients.append(client_socket)
        print(f"Connected to {client_addr}")

        client_thread = threading.Thread(target=handle_client, args=(client_socket,))
        client_thread.start()

if __name__ == "__main__":
    main()
```

Client.py

```
import socket
import threading

def receive_messages(client_socket):
    while True:
        try:
            message = client_socket.recv(1024).decode()
            print(message)
        except:
            print("Connection closed.")
            break

def main():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('127.0.0.1', 12345))

    receive_thread = threading.Thread(target=receive_messages, args=(client,))
    receive_thread.start()

    while True:
        message = input()
        client.send(message.encode())

if __name__ == "__main__":
    main()
```

3. Implement a 2 player Tic Tac Toe Game using socket programming. There should be at least 2 programs - client playing the game and the server, and a supervising server conducting the game. It should display proper result on game completion.

Server.py

```
import socket
import threading

# Tic Tac Toe board
board = [" " for _ in range(9)]
current_player = "X"

def send_board(player_socket):
    player_socket.send("".join(board).encode())

def handle_player(player_socket, player_symbol):
    global current_player

    while True:
        try:
            send_board(player_socket)

            if current_player == player_symbol:
                player_socket.send("Your turn. Choose a position (1-9): ".encode())
                position = int(player_socket.recv(1024).decode()) - 1

                if board[position] == " ":
                    board[position] = player_symbol
                    current_player = "O" if player_symbol == "X" else "X"
                else:
                    player_socket.send("Invalid move. Try again.".encode())
            else:
                player_socket.send("Waiting for the opponent's move...".encode())

            # Check for a win or draw
            winner = check_winner()
            if winner:
                send_board(player_socket)
                player_socket.send(f"Player {player_symbol} wins!".encode())
                break
            elif " " not in board:
                send_board(player_socket)
                player_socket.send("It's a draw!".encode())
                break
        except:
            break
    player_socket.close()

def check_winner():
    winning_positions = [(0, 1, 2), (3, 4, 5), (6, 7, 8),
                         (0, 3, 6), (1, 4, 7), (2, 5, 8),
                         (0, 4, 8), (2, 4, 6)]

    for pos1, pos2, pos3 in winning_positions:
        if board[pos1] == board[pos2] == board[pos3] != " ":
            return board[pos1]
    return None

def main():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(('127.0.0.1', 12345))
    server.listen(2)

    print("Server is listening...")

    player_sockets = []

    for _ in range(2):
        player_socket, _ = server.accept()
        player_sockets.append(player_socket)

        player_symbol = "X" if len(player_sockets) == 1 else "O"
        player_socket.send(f"You are Player {player_symbol}".encode())

        player_thread = threading.Thread(target=handle_player, args=(player_socket, player_symbol))
        player_thread.start()

if __name__ == "__main__":
    main()
```

Client.py

```
import socket
import threading

def receive_board(server_socket):
    while True:
        try:
            board = server_socket.recv(1024).decode()
            print_board(board)
        except:
            print("Connection closed.")
            break

def print_board(board_str):
    print("-------------")
    for i in range(0, 9, 3):
        print(f"| {board_str[i]} | {board_str[i+1]} | {board_str[i+2]} |")
        print("-------------")

def main():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('127.0.0.1', 12345))

    player_symbol = client.recv(1024).decode()
    print(player_symbol)

    receive_thread = threading.Thread(target=receive_board, args=(client,))
    receive_thread.start()

    while True:
        try:
            message = client.recv(1024).decode()
            print(message)

            if "win" in message or "draw" in message:
                client.close()
                break

            if "Your turn" in message:
                position = input("Choose a position (1-9): ")
                client.send(position.encode())
        except:
            client.close()
            break

if __name__ == "__main__":
    main()
```