IT205 Midsem Assignment

October 2, 2023

Nithin S

221IT085

Tic-Tac-Toe Server-Client Application

The below C code implements a client-server application for playing the game of Tic-Tac-Toe between two players.

Server-Side Code

Header Files Overview

`pthread.h`

- **Purpose**: Provides support for multithreading.
- **Function**: Allows the code to create and manage threads, essential for handling multiple clients concurrently.

`stdio.h`

- **Purpose**: Standard Input and Output functions.
- **Function**: Provides functions for reading and writing to the console, used for debugging and displaying game information.

`stdlib.h`

- **Purpose**: Standard Library functions.
- **Function**: Provides general-purpose functions like memory allocation and program termination (`exit`).

`string.h`

- **Purpose**: String handling functions.
- **Function**: Offers functions for string manipulation, including string comparison (`strcmp`) and memory clearing (`memset`).

`unistd.h`

- **Purpose**: POSIX Operating System API.
- **Function**: Provides access to POSIX system calls, including file and socket operations.

`sys/types.h`

- **Purpose**: Data types for system calls.
- **Function**: Defines various data types used in system calls, such as `socklen_t`.

`sys/socket.h`

- **Purpose**: Socket programming functions and structures.
- **Function**: Contains socket-related functions and structures used for network communication.

`netinet/in.h`

- **Purpose**: Internet address structures.
- **Function**: Defines structures for handling Internet addresses and port numbers, crucial for socket programming.

These header files are essential for including the necessary functions and data structures required to implement the server and client sides of a two-player Tic-Tac-Toe game.

Function Descriptions

`reportError(const char *message)`

- **Description**: This function is responsible for handling errors by printing an error message and terminating the thread.
- **Usage**: It is called when any error condition is encountered during the execution of the server code.

`sendClientInterger(int clientSock, int value)`

- **Description**: Writes an integer value to the specified client socket.
- **Usage**: Used to send various integer messages to clients, including game moves, player counts, and other updates.

`sendClientMessage(int *clientSock, char *message)`

- **Description**: Writes a message (string) to both client sockets.
- **Usage**: Used to send messages to both clients simultaneously, such as game start, updates, and notifications.

`sendClientsInteger(int *clientSocks, int value)`

- **Description**: Writes an integer to both client sockets.
- **Usage**: Used to send integer updates to both clients simultaneously, like player IDs, moves, and player counts.

`setupListener(int portNumber)`

- **Description**: Initializes and sets up a socket for the server to listen for incoming connections.
- **Usage**: Called in `main` to create a socket, set up socket parameters, bind it to a specific port, and return the socket file descriptor.

`receiveInteger(int clientSock)`

- **Description**: Receives an integer from a client socket.
- **Usage**: Used to read integer messages sent by clients, like moves or player counts.

`sendClientMessage(int clientSock, char *message)`

- **Description**: Writes a string message to a specified client socket.
- **Usage**: Used to send string-based messages to individual clients, such as turn notifications and invalid move alerts.

`getClientSockets(int listenerSock, int *clientSocks)`

- **Description**: Accepts incoming client connections and assigns client socket file descriptors.
- **Usage**: Called in `main` to listen for and accept connections from two clients. It assigns unique IDs to the clients and updates the player count.

`getPlayerMove(int clientSock)`

- **Description**: Initiates the process of getting a player's move by sending a "TRN" (turn) message to the client.
- **Usage**: Used to signal a player to make a move and expects a move response from the client.

`isValidMove(char board[][3], int move, int playerId)`

- **Description**: Checks if a move made by a player is valid.
- **Usage**: Ensures that a move is within the board boundaries and is not already occupied by another player's symbol.

`updateGameBoard(char board[][3], int move, int playerId)`

- **Description**: Updates the game board with the player's move.
- **Usage**: Changes the value at the specified position on the board to 'X' or 'O' based on the player's ID.

`drawGameBoard(char board[][3])`

- **Description**: Prints the current game board to the console.
- **Usage**: Used to display the current state of the Tic-Tac-Toe board.

`sendGameUpdate(int *clientSocks, int move, int playerId)`

- **Description**: Sends a game update message to both clients, including the player's ID and move.
- **Usage**: Used to inform both players of the latest move and its outcome.

`sendPlayerCount(int clientSock))`

- **Description**: Sends the current player count to a specific client.
- **Usage**: Used to inform a client about the total number of active players.

`checkGameBoard(char board[][3], int lastMove)`

- **Description**: Checks the game board for a win condition.
- **Usage**: Determines if a player has won by checking rows, columns, and diagonals on the board.

`runGame(void *threadData)`

- **Description**: A thread function that manages the execution of a single game session between two clients.
- **Usage**: Coordinates the flow of the game, including turn management, move validation, board updates, and game result checks.

`main(int argc, char *argv[])`

- **Description**: The main function of the server application.
- **Usage**: Initializes server components, accepts client connections, and creates game threads for each session.

[ ]:
```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int playerCount = 0;
pthread_mutex_t playerMutex;

void reportError(const char *message);
void sendClientMessage(int clientSock, char *message);
void sendClientInteger(int clientSock, int value);
void sendClientsMessage(int *clientSocks, char *message);
void sendClientsInteger(int *clientSocks, int value);
int setupListener(int portNumber);
int receiveInteger(int clientSock);
void getClientSockets(int listenerSock, int *clientSocks);
int getPlayerMove(int clientSock);
int isValidMove(char board[][3], int move, int playerId);
void updateGameBoard(char board[][3], int move, int playerId);
void drawGameBoard(char board[][3]);
void sendGameUpdate(int *clientSocks, int move, int playerId);
```

```
void sendPlayerCount(int clientSock);
int checkGameBoard(char board[][3], int lastMove);

int setupListener(int portNumber)
{
    int listenerSock;
    struct sockaddr_in serverAddress;

    listenerSock = socket(AF_INET, SOCK_STREAM, 0);
    if (listenerSock < 0)
        reportError("Error opening listener socket.");

    memset(&serverAddress, 0, sizeof(serverAddress));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = INADDR_ANY;
    serverAddress.sin_port = htons(portNumber);

    if (bind(listenerSock, (struct sockaddr *)&serverAddress,␣
 ↪sizeof(serverAddress)) < 0)
        reportError("Error binding listener socket.");

    return listenerSock;
}

void sendClientInteger(int clientSock, int value)
{
    int n = write(clientSock, &value, sizeof(int));
    if (n < 0)
        reportError("Error writing integer to client socket");
}

void sendClientsMessage(int *clientSocks, char *message)
{
    sendClientMessage(clientSocks[0], message);
    sendClientMessage(clientSocks[1], message);
}

void sendClientsInteger(int *clientSocks, int value)
{
    sendClientInteger(clientSocks[0], value);
    sendClientInteger(clientSocks[1], value);
}

int receiveInteger(int clientSock)
{
    int value = 0;
    int n = read(clientSock, &value, sizeof(int));
```

```
    if (n < 0 || n != sizeof(int))
        return -1;

    return value;
}

void sendClientMessage(int clientSock, char *message)
{
    int n = write(clientSock, message, strlen(message));
    if (n < 0)
        reportError("Error writing message to client socket");
}

void reportError(const char *message)
{
    perror(message);
    pthread_exit(NULL);
}

void getClientSockets(int listenerSock, int *clientSocks)
{
    socklen_t clientAddressLength;
    struct sockaddr_in serverAddress, clientAddress;
    int numConnections = 0;

    while (numConnections < 2)
    {
        listen(listenerSock, 253 - playerCount);

        memset(&clientAddress, 0, sizeof(clientAddress));
        clientAddressLength = sizeof(clientAddress);

        clientSocks[numConnections] = accept(listenerSock, (struct sockaddr␣
↪*)&clientAddress, &clientAddressLength);

        if (clientSocks[numConnections] < 0)
            reportError("Error accepting a connection from a client.");

        write(clientSocks[numConnections], &numConnections, sizeof(int));

        pthread_mutex_lock(&playerMutex);
        playerCount++;
        printf("Number of players is now %d.\n", playerCount);
        pthread_mutex_unlock(&playerMutex);

        if (numConnections == 0)
```

```
        {
            sendClientMessage(clientSocks[0], "HLD");
        }

        numConnections++;
    }
}

int getPlayerMove(int clientSock)
{
    sendClientMessage(clientSock, "TRN");
    return receiveInteger(clientSock);
}

int isValidMove(char board[][3], int move, int playerId)
{
    if ((move == 9) || (board[move / 3][move % 3] == ' '))
        return 1;
    else
        return 0;
}

void updateGameBoard(char board[][3], int move, int playerId)
{
    board[move / 3][move % 3] = playerId ? 'X' : 'O';
}

void drawGameBoard(char board[][3])
{
    printf("\n");
    printf(" %c | %c | %c \n", board[0][0], board[0][1], board[0][2]);
    printf("-----------\n");
    printf(" %c | %c | %c \n", board[1][0], board[1][1], board[1][2]);
    printf("-----------\n");
    printf(" %c | %c | %c \n", board[2][0], board[2][1], board[2][2]);
}

void sendGameUpdate(int *clientSocks, int move, int playerId)
{
    sendClientsMessage(clientSocks, "UPD");
    sendClientsInteger(clientSocks, playerId);
    sendClientsInteger(clientSocks, move);
}

void sendPlayerCount(int clientSock)
{
    sendClientMessage(clientSock, "CNT");
```

```
    sendClientInteger(clientSock, playerCount);
}

int checkGameBoard(char board[][3], int lastMove)
{
    int row = lastMove / 3;
    int col = lastMove % 3;

    if (board[row][0] == board[row][1] && board[row][1] == board[row][2])
    {
        return 1;
    }
    else if (board[0][col] == board[1][col] && board[1][col] == board[2][col])
    {
        return 1;
    }
    else if (!(lastMove % 2))
    {
        if ((lastMove == 0 || lastMove == 4 || lastMove == 8) && (board[1][1] ==␣
 ↪board[0][0] && board[1][1] == board[2][2]))
        {
            return 1;
        }
        if ((lastMove == 2 || lastMove == 4 || lastMove == 6) && (board[1][1] ==␣
 ↪board[0][2] && board[1][1] == board[2][0]))
        {
            return 1;
        }
    }

    return 0;
}

void *runGame(void *threadData)
{
    int *clientSocks = (int *)threadData;
    char board[3][3] = {{' ', ' ', ' '},
                        {' ', ' ', ' '},
                        {' ', ' ', ' '}};

    printf("Game on!\n");

    sendClientsMessage(clientSocks, "SRT");

    drawGameBoard(board);

    int prevPlayerTurn = 1;
```

```
int playerTurn = 0;
int gameOver = 0;
int turnCount = 0;

while (!gameOver)
{
    if (prevPlayerTurn != playerTurn)
        sendClientMessage(clientSocks[(playerTurn + 1) % 2], "WAT");

    int valid = 0;
    int move = 0;

    while (!valid)
    {
        move = getPlayerMove(clientSocks[playerTurn]);
        if (move == -1)
            break;

        printf("Player %d played position %d\n", playerTurn, move);

        valid = isValidMove(board, move, playerTurn);
        if (!valid)
        {
            printf("Move was invalid. Let's try this again...\n");
            sendClientMessage(clientSocks[playerTurn], "INV");
        }
    }

    if (move == -1)
    {
        printf("Player disconnected.\n");
        break;
    }
    else if (move == 9)
    {
        prevPlayerTurn = playerTurn;
        sendPlayerCount(clientSocks[playerTurn]);
    }
    else
    {
        updateGameBoard(board, move, playerTurn);
        sendGameUpdate(clientSocks, move, playerTurn);

        drawGameBoard(board);

        gameOver = checkGameBoard(board, move);
```

```
            if (gameOver == 1)
            {
                sendClientMessage(clientSocks[playerTurn], "WIN");
                sendClientMessage(clientSocks[(playerTurn + 1) % 2], "LSE");
                printf("Player %d won.\n", playerTurn);
            }
            else if (turnCount == 8)
            {
                printf("Draw.\n");
                sendClientsMessage(clientSocks, "DRW");
                gameOver = 1;
            }

            prevPlayerTurn = playerTurn;
            playerTurn = (playerTurn + 1) % 2;
            turnCount++;
        }
    }

    printf("Game over.\n");

    close(clientSocks[0]);
    close(clientSocks[1]);

    pthread_mutex_lock(&playerMutex);
    playerCount--;
    printf("Number of players is now %d.\n", playerCount);
    pthread_mutex_unlock(&playerMutex);

    free(clientSocks);

    pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
    if (argc < 2)
    {
        fprintf(stderr, "Error, no port provided\n");
        exit(1);
    }
    printf("Server is up and running, waiting for clients to connect...\n");
    fflush(stdout); // Flush the output buffer

    int listenerSock = setupListener(atoi(argv[1]));
    pthread_mutex_init(&playerMutex, NULL);
```

```
    while (1)
    {
        if (playerCount <= 252)
        {
            int *clientSocks = (int *)malloc(2 * sizeof(int));
            memset(clientSocks, 0, 2 * sizeof(int));

            getClientSockets(listenerSock, clientSocks);

            pthread_t thread;
            int result = pthread_create(&thread, NULL, runGame, (void␣
↪*)clientSocks);
            if (result)
            {
                printf("Thread creation failed with return code %d\n", result);
                exit(-1);
            }

            printf("New game thread started.\n");
        }
    }

    close(listenerSock);
    pthread_mutex_destroy(&playerMutex);
    pthread_exit(NULL);
}
```

Client-Side Code

The below C code implements a client for a Tic-Tac-Toe game that connects to a server.

Function Descriptions

`receiveMessage(int sockfd, char *msg)`

- **Description**: Receives and processes game updates from the server, including the opponent's moves.
- **Usage**: Used to read messages from the server and handle different game states.

`receiveInteger(int sockfd)`

- **Description**: Reads an integer value from the server.
- **Usage**: Used to receive integer messages sent by the server, such as game moves and player counts.

`sendServerInteger(int sockfd, int msg)`

- **Description**: Writes an integer value to the server.

- **Usage**: Used to send integer messages to the server, including the player's moves.

`reportError(const char *msg)`

- **Description**: Handles errors by printing an error message and terminating the client application.
- **Usage**: Called when any error condition is encountered during the execution of the client code.

`connectToServer(char *hostname, int portno)`

- **Description**: Establishes a connection to the server using the provided hostname and port number.
- **Usage**: Called at the beginning of the `main` function to connect to the server.

`displayGameBoard(char board[][3])`

- **Description**: Prints the current state of the Tic-Tac-Toe game board.
- **Usage**: Used to display the game board to the client.

`takeTurn(int sockfd)`

- **Description**: Initiates the process of taking a turn by sending a move to the server.
- **Usage**: Prompts the player for input and sends the chosen move to the server.

`getGameUpdate(int sockfd, char board[][3])`

- **Description**: Updates the client's game board based on updates received from the server.
- **Usage**: Used to keep the client's game board synchronized with the server's game state.

`main(int argc, char *argv[])`

- **Description**: The main function of the client application.
- **Usage**: Initializes client components, establishes a connection to the server, and manages the client's interaction with the game server.

[ ]:
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void receiveMessage(int sockfd, char *msg);
int receiveInteger(int sockfd);
void sendServerInteger(int sockfd, int msg);
void sendServerInteger(int sockfd, int msg);
void reportError(const char *msg);
```

```c
int connectToServer(char *hostname, int portno);
void displayGameBoard(char board[][3]);
void takeTurn(int sockfd);
void getGameUpdate(int sockfd, char board[][3]);

int connectToServer(char *hostname, int portno)
{
    struct sockaddr_in serverAddress;
    struct hostent *server;

    int sockfd = socket(AF_INET, SOCK_STREAM, 0);

    if (sockfd < 0)
        perror("Error opening socket for server!!");

    server = gethostbyname(hostname);

    if (server == NULL)
    {
        fprintf(stderr, "Error, no such host!!\n");
        exit(0);
    }

    memset(&serverAddress, 0, sizeof(serverAddress));

    serverAddress.sin_family = AF_INET;
    memmove(server->h_addr, &serverAddress.sin_addr.s_addr, server->h_length);
    serverAddress.sin_port = htons(portno);

    if (connect(sockfd, (struct sockaddr *)&serverAddress,␣
 ↪sizeof(serverAddress)) < 0)
        perror("Error Connecting to server!!");

    printf("Connected to server.\n");
    return sockfd;
}

void receiveMessage(int sockfd, char *msg)
{
    memset(msg, 0, 4);
    int n = read(sockfd, msg, 3);

    if (n < 0 || n != 3)
        perror("Error reading message from server socket.");
}

int receiveInteger(int sockfd)
```

```
{
    int msg = 0;
    int n = read(sockfd, &msg, sizeof(int));

    if (n < 0 || n != sizeof(int))
        perror("Error reading integer from server socket");
    return msg;
}

void sendServerInteger(int sockfd, int msg)
{
    int n = write(sockfd, &msg, sizeof(int));
    if (n < 0)
        perror("Error writing integer to server socket");
}

void reportError(const char *msg)
{
    perror(msg);
    printf("Either the server shut down or the other player disconnected.\nGame␣
 ↪over.\n");
    exit(0);
}


void displayGameBoard(char board[][3])
{
    printf("\n");
    for (int row = 0; row < 3; row++)
    {
        for (int col = 0; col < 3; col++)
        {
            char symbol = board[row][col];
            if (symbol == ' ')
                printf(" %d ", (row * 3) + col); // Display the position number
            else
                printf(" %c ", symbol);

            if (col < 2)
                printf("|");
        }
        printf("\n");
        if (row < 2)
            printf("-----------\n");
    }
}
```

```
void takeTurn(int sockfd)
{
    char buffer[10];

    while (1)
    {
        printf("Enter 0-8 to make a move: ");
        fgets(buffer, 10, stdin);
        int move = buffer[0] - '0';
        if (move < 9 && move >= 0)
        {
            printf("\n");
            sendServerInteger(sockfd, move);
            break;
        }
        else
            printf("\nInvalid input. Try again.\n");
    }
}

void getGameUpdate(int sockfd, char board[][3])
{
    int playerId = receiveInteger(sockfd);
    int move = receiveInteger(sockfd);
    board[move / 3][move % 3] = playerId ? 'X' : 'O';
}

int main(int argc, char *argv[])
{
    if (argc < 3)
    {
        fprintf(stderr, "usage %s hostname port\n", argv[0]);
        exit(0);
    }

    int sockfd = connectToServer(argv[1], atoi(argv[2]));

    int id = receiveInteger(sockfd);

    char msg[4];
    char board[3][3] = {{' ', ' ', ' '},
                        {' ', ' ', ' '},
                        {' ', ' ', ' '}};

    printf("\n"
           " _______ _        _______           _______ \n"
           "|__   __(_)      |__   __|         |__   __| \n"
```

```
        "   | |   _  ___     | | __ _  ___     | | ___   ___ \n"
        "   | |  | |/ __|    | |/ _` |/ __|    | |/ _ \\ / _ \\ \n"
        "   | |  | | (__     | | (_| | (__     | | (_) |  __/ \n"
        "   |_|  |_|\\___|    |_|\\__,_|\\___|    |_|\\___/ \\___| \n\n\n");

    do
    {
        receiveMessage(sockfd, msg);
        if (!strcmp(msg, "HLD"))
            printf("Waiting for a second player...\n");
    } while (strcmp(msg, "SRT"));

    /* The game has begun. */
    printf("Game on!\n");
    printf("You are %c's\n", id ? 'X' : 'O');

    displayGameBoard(board);

    while (1)
    {
        receiveMessage(sockfd, msg);

        if (!strcmp(msg, "TRN"))
        {
            printf("Your move...\n");
            takeTurn(sockfd);
        }
        else if (!strcmp(msg, "INV"))
        {
            printf("That position has already been played. Try again.\n");
        }
        else if (!strcmp(msg, "CNT"))
        {
            int numPlayers = receiveInteger(sockfd);
            printf("There are currently %d active players.\n", numPlayers);
        }
        else if (!strcmp(msg, "UPD"))
        {
            getGameUpdate(sockfd, board);
            displayGameBoard(board);
        }
        else if (!strcmp(msg, "WAT"))
        {
            printf("Waiting for the other player's move...\n");
        }
        else if (!strcmp(msg, "WIN"))
        {
```

```
            printf("__   __           __        ___       \n");
            printf("\\ \\ / /__  _   _  \\ \\      / (_)_ __  \n");
            printf(" \\ V / _ \\| | | |  \\ \\ /\\ / /| | '_ \\ \n");
            printf("  | | (_) | |_| |   \\ V  V / | | | | |\n");
            printf("  |_|\\___/ \\__,_|    \\_/\\_/  |_|_| |_|\n");

            break;
        }
        else if (!strcmp(msg, "LSE"))
        {
            printf("__   __            _              _   \n");
            printf("\\ \\ / /__  _   _  | |    ___  ___| |_ \n");
            printf(" \\ V / _ \\| | | | | |   / _ \\/ __| __|\n");
            printf("  | | (_) | |_| | | |__| (_) \\__ \\ |_ \n");
            printf("  |_|\\___/ \\__,_| |_____\\___/|___/\\__|\n");
            break;
        }
        else if (!strcmp(msg, "DRW"))
        {
            printf(" ___ _   _                 _____ _      \n");
            printf("|_ _| |_( )___    __ _  |_   _(_) ___ \n");
            printf(" | || __|// __|  / _` |   | | | |/ _ \\\\n");
            printf(" | || |_  \\__ \\ | (_| |   | | | |  __/\n");
            printf("|___|\\__| |___/  \\__,_|   |_| |_|\\___|\n");

            break;
        }
        else
        {
            reportError("Unknown message.");
        }
    }

    printf("Game over.\n");
    close(sockfd);
    return 0;
}
```

Server Code Analysis

The server code implements the server-side logic for a two-player Tic-Tac-Toe game. It uses multi-threading to handle multiple client connections concurrently.

Server Initialization and Setup

- The server initializes a listener socket using the `setup_listener` function, which binds to a specified port and waits for incoming client connections.

- Multithreading is used to handle client connections concurrently, and a mutex (`mutexcount`) is used to protect the `player_count` variable, which keeps track of the number of connected players.

Client Connection Handling

- The `get_clients` function accepts incoming client connections and assigns unique IDs (0 and 1) to each client. It also sends an initial "HLD" message to the first client, indicating that it should hold until the second player joins.
- The server spawns a new thread for each game session using the `pthread_create` function, passing the client socket file descriptors as thread data.
- The `run_game` function manages the gameplay for two connected clients, updating the game board, checking for a win, loss, or draw, and sending updates to clients.

Game Logic

- The game logic is handled within the `run_game` function, where two players take turns making moves.
- Players communicate with the server using messages like "TRN" (turn), "INV" (invalid move), "UPD" (update board), "WIN" (win), "LSE" (lose), and "DRW" (draw).
- The server maintains the game state, checks the validity of moves, updates the game board, and determines the winner or a draw.
- The game continues until a win, loss, or draw condition is met.

Thread Management

- The server creates a new thread for each game session to handle concurrent gameplay for multiple clients.
- After a game session is completed, the server releases resources and decrements the `player_count`.

Client Code Analysis

The provided C code implements the client-side logic for a two-player Tic-Tac-Toe game. It connects to the server and communicates with it during gameplay. Here's an analysis of the client code:

Client Initialization

- The client initializes a socket and connects to a specified server using the `connect_to_server` function.
- It receives an ID (0 or 1) from the server, indicating the player's role (X or O) in the game.

Gameplay Interaction

- The client communicates with the server using messages like "TRN" (turn), "INV" (invalid move), "UPD" (update board), "WIN" (win), "LSE" (lose), and "DRW" (draw).
- The `take_turn` function allows the player to input their move (0-8) and sends it to the server.
- The `get_update` function receives updates from the server, including the opponent's moves, and updates the local game board accordingly.

Game Loop

- The client enters a game loop, where it continuously receives and processes messages from the server.
- Messages from the server guide the player's actions, such as making moves, handling invalid moves, updating the board, and waiting for the opponent.
- The game loop continues until a win, loss, or draw condition is met.

Display

- The client uses the `draw_board` function to display the current state of the Tic-Tac-Toe game board.
- The game board is printed to the console, showing the positions of X, O, or empty spaces.

Termination

- When the game ends, the client displays the game outcome and closes the socket connection to the server.

Overall, the client-side code enables players to connect to the server, participate in the Tic-Tac-Toe game, and interact with the game state through console messages.

Output

Server

```
(base) nithin@astralanguish:~/Desktop/IT205/MidSemProject$ gcc server.c -lpthrea
d -o ser
(base) nithin@astralanguish:~/Desktop/IT205/MidSemProject$ ./ser 8080
Server is up and running, waiting for clients to connect...
Number of players is now 1.
Number of players is now 2.
[DEBUG] New game thread started.
Game on!

   |   |
-----------
   |   |
-----------
   |   |
```

Clients

```
(base) nithin@astralanguish:~/Desktop/IT205/MidSemProject$ gcc client.c -o cli
(base) nithin@astralanguish:~/Desktop/IT205/MidSemProject$ ./cli 192.168.119.169
 8080
Connected to server.

 _____ _        _____            _____
|_   _(_) ___  |_   _|_ _  ___  |_   _|__   ___
  | | | |/ __|   | |/ _` |/ __|   | |/ _ \ / _ \
  | | | | (__    | | (_| | (__    | | (_) |  __/
  |_| |_|\___|   |_|\__,_|\___|   |_|\___/ \___|


Waiting for a second player...
Game on!
You are O's

 0 | 1 | 2
-----------
 3 | 4 | 5
-----------
 6 | 7 | 8
Your move...
Enter 0-8 to make a move:
```

```
(base) nithin@astralanguish:~/Desktop/IT205/MidSemProject$ gcc client.c -o cli
(base) nithin@astralanguish:~/Desktop/IT205/MidSemProject$ ./cli 192.168.119.169
 8080
Connected to server.

 _____ _        _____            _____
|_   _(_) ___  |_   _|_ _  ___  |_   _|__   ___
  | | | |/ __|   | |/ _` |/ __|   | |/ _ \ / _ \
  | | | | (__    | | (_| | (__    | | (_) |  __/
  |_| |_|\___|   |_|\__,_|\___|   |_|\___/ \___|


Game on!
You are X's

 0 | 1 | 2
-----------
 3 | 4 | 5
-----------
 6 | 7 | 8
Waiting for the other player's move...
```

```
Number of players is now 2.
[DEBUG] New game thread started.
Game on!

   |   |
-----------
   |   |
-----------
   |   |
Player 0 played position 0

 O |   |
-----------
   |   |
-----------
   |   |
Player 1 played position 1

 O | X |
-----------
   |   |
-----------
   |   |
```

```
 0 | 1 | 2
-----------
 3 | 4 | 5
-----------
 6 | 7 | 8
Your move...
Enter 0-8 to make a move: 0

Wrote int to server: 0

 O | 1 | 2
-----------
 3 | 4 | 5
-----------
 6 | 7 | 8
Waiting for the other player's move...

 O | X | 2
-----------
 3 | 4 | 5
-----------
 6 | 7 | 8
Your move...
Enter 0-8 to make a move:
```

```
 0 | 1 | 2
-----------
 3 | 4 | 5
-----------
 6 | 7 | 8
Waiting for the other player's move...

 O | 1 | 2
-----------
 3 | 4 | 5
-----------
 6 | 7 | 8
Your move...
Enter 0-8 to make a move: 1

Wrote int to server: 1

 O | X | 2
-----------
 3 | 4 | 5
-----------
 6 | 7 | 8
Waiting for the other player's move...
```

```
-----------
   |   |
Player 1 played position 1

 O | X |
-----------
   |   |
-----------
   |   |
Player 0 played position 4

 O | X |
-----------
   | O |
-----------
   |   |
Player 1 played position 5

 O | X |
-----------
   | O | X
-----------
   |   |
```

```
 O | X | 2
-----------
 3 | 4 | 5
-----------
 6 | 7 | 8
Your move...
Enter 0-8 to make a move: 4

Wrote int to server: 4

 O | X | 2
-----------
 3 | O | 5
-----------
 6 | 7 | 8
Waiting for the other player's move...

 O | X | 2
-----------
 3 | O | X
-----------
 6 | 7 | 8
Your move...
Enter 0-8 to make a move:
```

```
 O | X | 2
-----------
 3 | 4 | 5
-----------
 6 | 7 | 8
Waiting for the other player's move...

 O | X | 2
-----------
 3 | O | 5
-----------
 6 | 7 | 8
Your move...
Enter 0-8 to make a move: 5

Wrote int to server: 5

 O | X | 2
-----------
 3 | O | X
-----------
 6 | 7 | 8
Waiting for the other player's move...
```

```

 O | X |
-----------
   | O |
-----------
   |   |
Player 1 played position 5

 O | X |
-----------
   | O | X
-----------
   |   |
Player 0 played position 8

 O | X |
-----------
   | O | X
-----------
   |   | O
Player 0 won.
Game over.
Number of players is now 1.
```

```
Waiting for the other player's move...

 O | X | 2
-----------
 3 | O | X
-----------
 6 | 7 | 8
Your move...
Enter 0-8 to make a move: 8

Wrote int to server: 8

 O | X | 2
-----------
 3 | O | X
-----------
 6 | 7 | O
__   __            __        ___
\ \ / /__  _   _   \ \      / (_)_ __
 \ V / _ \| | | |   \ \ /\ / /| | '_ \
  | | (_) | |_| |    \ V  V / | | | | |
  |_|\___/ \__,_|     \_/\_/  |_|_| |_|
Game over.
(base) nithin@astralanguish:~/Desktop/IT205/MidSemProject$
```

```
Your move...
Enter 0-8 to make a move: 5

Wrote int to server: 5

 O | X | 2
-----------
 3 | O | X
-----------
 6 | 7 | 8
Waiting for the other player's move...

 O | X | 2
-----------
 3 | O | X
-----------
 6 | 7 | O
__   __            _              _
\ \ / /__  _   _  | |    ___  ___| |_
 \ V / _ \| | | | | |   / _ \/ __| __|
  | | (_) | |_| | | |__| (_) \__ \ |_
  |_|\___/ \__,_| |_____\___/|___/\__|
Game over.
(base) nithin@astralanguish:~/Desktop/IT205/MidSemProject$
```