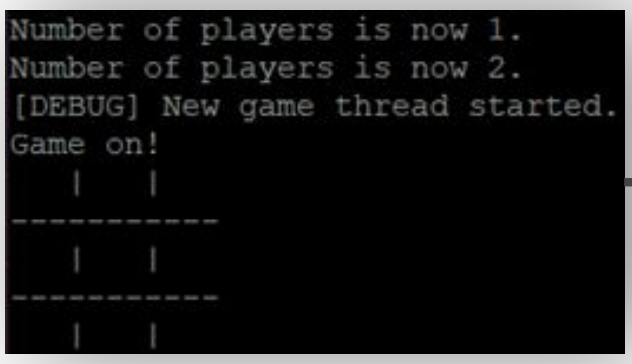
# Multi User Tic-Tac-Toe

As Created by: Caleb Kahn, Luke Alcazar, Noah Trillizio

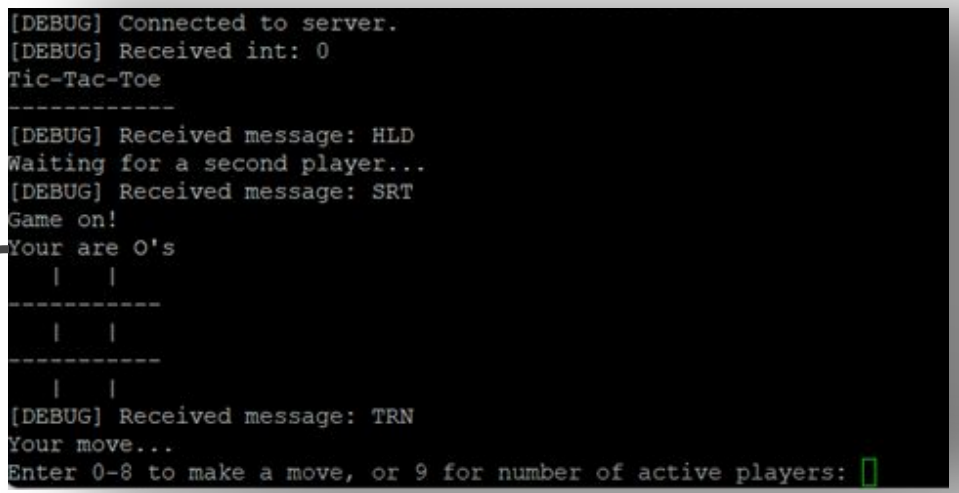
# Gameplay

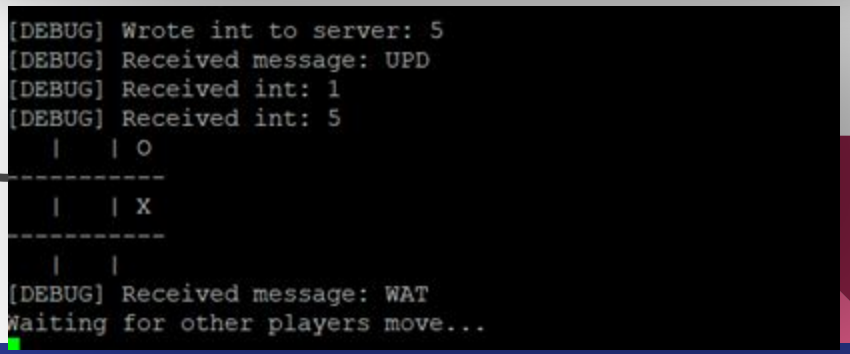
# Gameplay Example

## Server




## Clients



# Client Code

# Client Code - Functions

## Connect To Server

```
int connect_to_server(char * hostname, int portno)
{
        struct sockaddr_in serv_addr;
        struct hostent *server;

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

        if (sockfd < 0)
        {
                error("ERROR opening socket for server.");
        }
        server = gethostbyname(hostname);
        if (server == NULL)
        {
                fprintf(stderr, "ERROR, no such host\n");
                exit(0);
        }

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

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

        if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
        {
                error("ERROR connecting to server");
        }

        printf("[DEBUG] Connected to server.\n");
        return sockfd;
}
```

## Receive Message

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

        if (
        {
                error("ERROR reading message from server socket.");
        }

        printf("[DEBUG] Received message: %s\n", msg);
}

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

        if (n < 0 || n != sizeof(int))
        {
                error("ERROR reading int from server socket");
        }

        printf("[DEBUG] Received int: %d\n", msg);

        return msg;
}
```

## Send Message

```
void write_server_int(int sockfd, int msg)
{
        int n = write(sockfd, &msg, sizeof(int));
        if (n < 0)
        {
                error("ERROR writing int to server socket");
        }

        printf("[DEBUG] Wrote int to server: %d\n", msg);
}
```

## Draw Game

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

## Take Turn

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

        while (1)
        {
                printf("Enter 0-8 to make a move, or 9 for number of active players: ");
                fgets(buffer, 10, stdin);
                int move = buffer[0] - '0';
                if (move <= 9 && move >= 0)
                {
                        printf("
                        write_server_int(sockfd, move);
                        break;

                }
                else
                {
                        printf("\nInvalid input. Try again.\n");
                }
        }
}
```

# Client Code – Main Game

Part 2

## Set Up Game

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

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


    int id = recv_int(sockfd);

#ifdef DEBUG
    printf("[DEBUG] Client ID: %d\n", id);
#endif

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

    printf("Tic-Tac-Toe\n------------\n");

    do
    {
        recv_msg(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("Your are %c's\n", id ? 'X' : 'O');

    draw_board(board);
```

## Looped Game

```c
while (1)
{
        recv_msg(sockfd, msg);

        if (!strcmp(msg, "TRN"))
        {
                printf("Your move...\n");
                take_turn(sockfd);
        }
        else if (!strcmp(msg, "INV"))
        {
                printf("That position has already been played. Try again.\n");
        }
        else if (!strcmp(msg, "CNT"))
        {
                int num_players = recv_int(sockfd);
                printf("There are currently %d active players.\n", num_players);
        }
        else if (!strcmp(msg, "UPD"))
        {
                get_update(sockfd, board);
                draw_board(board);
        }
        else if (!strcmp(msg, "WAT"))
        {
                [illegible]
        }
        else if (!strcmp(msg, "WIN"))
        {
                printf("You win!\n");
                break;
        }
        else if (!strcmp(msg, "LSE"))
        {
                printf("You lost.\n");
                break;
        }
        else if (!strcmp(msg, "DRW"))
        {
                printf("Draw.\n");
                break;
        }
        else
        {
                error("Unknown message.");
        }
```

# Server Code

# Server Code – Clients

Part 1

```
void get_clients(int lis_sockfd, int * cli_sockfd)
{
        socklen_t clilen;
        struct sockaddr_in serv_addr, cli_addr;

#ifdef DEBUG
        printf("[DEBUG] Listening for clients...\n");
#endif

        int num_conn = 0;
        while (num_conn < 2)
        {
                listen(lis_sockfd, 253 - player_count);

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

                clilen = sizeof(cli_addr);

                cli_sockfd[num_conn] = accept(lis_sockfd, (struct sockaddr *) &cli_addr, &clilen);
                if (cli_sockfd[num_conn] < 0)
                {
                        error("ERROR accepting a connection from a client.");
                }

#ifdef DEBUG
                printf("[DEBUG] Accepted connection from client %d\n", num_conn);
#endif

                write(cli_sockfd[num_conn], &num_conn, sizeof(int));

#ifdef DEBUG
                printf("[DEBUG] Sent client %d it's ID.\n", num_conn);
#endif

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

                if (num_conn == 0)
                {
                        write_client_msg(cli_sockfd[0], "HLD");

#ifdef DEBUG
                        printf("[DEBUG] Told client 0 to hold.\n");
```

Connect Clients

```
void write_client_int(int cli_sockfd, int msg)
{
        int n = write(cli_sockfd, &msg, sizeof(int));
        if (n < 0)
        {
                error("ERROR writing int to client socket");
        }
}


void write_clients_msg(int * cli_sockfd, char * msg)
{
        write_client_msg(cli_sockfd[0], msg);
        write_client_msg(cli_sockfd[1], msg);
}
void write_clients_int(int * cli_sockfd, int msg)
{
        write_client_int(cli_sockfd[0], msg);
        write_client_int(cli_sockfd[1], msg);
}

int setup_listener(int portno)
{
        int sockfd;
        struct sockaddr_in serv_addr;

        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd < 0)
        {
                error("ERROR opening listener socket.");
        }

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

        if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
        {
                error("ERROR binding listener socket.");
        }
```

Send Messages

# Server Code – Functions

Part 2

## Get Move

```
int get_player_move(int cli_sockfd)
{
#ifdef DEBUG
        printf("[DEBUG] Getting player move...\n");
#endif

        write_client_msg(cli_sockfd, "TRN");

        return recv_int(cli_sockfd);
}

int check_move(char board[][3], int move, int player_id)
{
        if ((move == 9) || (board[move / 3][move % 3] == ' '))
        {

#ifdef DEBUG
                printf("[DEBUG] Player %d's move was valid.\n", player_id);
#endif

                return 1;
        }
        else
        {
#ifdef DEBUG
                printf("[DEBUG] Player %d's move was invalid.\n", player_id);
#endif
                return 0;
        }
}

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

#ifdef DEBUG
        printf("[DEBUG] Board updated.\n");
#endif
}
```

## Update Game

## Draw Game

```
void draw_board(char board[][3])
{
        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 send_update(int * cli_sockfd, int move, int player_id)
{
#ifdef DEBUG
        printf("[DEBUG] Sending update...\n");
#endif

        write_clients_msg(cli_sockfd, "UPD");

        write_clients_int(cli_sockfd, player_id);

        write_clients_int(cli_sockfd, move);

#ifdef DEBUG
        printf("[DEBUG] Update sent.\n");
#endif
}

void send_player_count(int cli_sockfd)
{
        write_client_msg(cli_sockfd, "CNT");
        write_client_int(cli_sockfd, player_count);

#ifdef DEBUG
        printf("[DEBUG] Player Count Sent.\n");
#endif
}
```

## Send Updates

## Check For Winner

```
int check_board(char board[][3], int last_move)
{
#ifdef DEBUG
        printf("[DEBUG] Checking for a winner...\n");
#endif

        int row = last_move / 3;
        int col = last_move % 3;

        if (board[row][0] == board[row][1] && board[row][1] == board[row][2])
        {
#ifdef DEBUG
                printf("[DEBUG] Win by row %d.\n", row);
#endif
                return 1;
        }
        else if (board[0][col] == board[1][col] && board[1][col] == board[2][col]) {
#ifdef DEBUG
                printf("[DEBUG] Win by column %d.\n", col);
#endif
                return 1;
        }
        else if (!(last_move % 2))
        {
                if ((last_move == 0 || last_move == 4 || last_move == 8) && (board[1][1] == board[0][0] && board[1][1] == board[2][2]))
                {
#ifdef DEBUG
                        printf("[DEBUG] Win by backslash diagonal.\n");
#endif
                        return 1;
                }
                if ((last_move == 2 || last_move == 4 || last_move == 6) && (board[1][1] == board[0][2] && board[1][1] == board[2][0]))
                {
#ifdef DEBUG
                        printf("[DEBUG] Win by frontslash diagonal.\n");
#endif
                        return 1;
                }
        }

#ifdef DEBUG
        printf("[DEBUG] No winner, yet.\n");
#endif

        return 0;
}
```

# Server Code – Game Implementation

Part 3

## Setup Game

```
void *run_game(void *thread_data)
{
        int *cli_sockfd = (int*)thread_data;
        char board[3][3] = { {' ', ' ', ' '},
                             {' ', ' ', ' '},
                             {' ', ' ', ' '} };

        printf("Game on!\n");

        write_clients_msg(cli_sockfd, "SRT");

#ifdef DEBUG
        printf("[DEBUG] Sent start message.\n");
#endif

        draw_board(board);

        int prev_player_turn = 1;
        int player_turn = 0;
        int game_over = 0;
        int turn_count = 0;
        while (!game_over)
        {
                if (prev_player_turn != player_turn)
                {
                        write_client_msg(cli_sockfd[(player_turn + 1) % 2], "WAT");
                }

                int valid = 0;
                int move = 0;
                while (!valid)
                {
                        move = get_player_move(cli_sockfd[player_turn]);
                        if (move == -1) break;
                        printf("Player %d played position %d\n", player_turn, move);

                        valid = check_move(board, move, player_turn);
                        if (!valid)
                        {
                                printf("Move was invalid. Let's try this again...\n");
                                write_client_msg(cli_sockfd[player_turn], "INV");
                        }
                }
```

## Loop Game

```
        if (move == -1)
        {
                printf("Player disconnected.\n");
                break;
        }
        else if (move == 9)
        {
                prev_player_turn = player_turn;
                send_player_count(cli_sockfd[player_turn]);
        }
        else
        {
                update_board(board, move, player_turn);
                send_update(cli_sockfd, move, player_turn);


                draw_board(board);

                game_over = check_board(board, move);

                if (game_over == 1)
                {
                        write_client_msg(cli_sockfd[player_turn], "WIN");
                        write_client_msg(cli_sockfd[(player_turn + 1) % 2], "LSE");
                        printf("Player %d won.\n", player_turn);
                }
                else if (turn_count == 8)
                {
                        printf("Draw.\n");
                        write_clients_msg(cli_sockfd, "DRW");
                        game_over = 1;
                }

                prev_player_turn = player_turn;
                player_turn = (player_turn + 1) % 2;
                turn_count++;
        }
}

printf("Game over.\n");

close(cli_sockfd[0]);
close(cli_sockfd[1]);
```

## Prepare Multiple Games

```
        printf("Number of players is now %d.", player_count);
        player_count--;
        printf("Number of players is now %d.", player_count);
        pthread_mutex_unlock(&mutexcount);

        free(cli_sockfd);

        pthread_exit(NULL);
}


int main(int argc, char *argv[])
{
        if (argc < 2)
        {
                fprintf(stderr, "ERROR, no port provided\n");
                exit(1);
        }

        int lis_sockfd = setup_listener(atoi(argv[1]));
        pthread_mutex_init(&mutexcount, NULL);

        while (1)
        {
                if (player_count <= 2)
                {
                        int *cli_sockfd = (int *) malloc(2 * sizeof(int));
                        memset(cli_sockfd, 0, 2 * sizeof(int));

                        get_clients(lis_sockfd, cli_sockfd);

#ifdef DEBUG
                        printf("[DEBUG] Starting new game thread...\n");
#endif

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

                        printf("[DEBUG] New game thread started.\n");
                }
        }
```

# End/Final Points

Thank You!