**TITLE: - MULTIPLAYER TIC TAC TOE GAME USING SOCKET PROGRAMMING**

**PROJECT**

**COMPUTER NETWORK (IT2008)**

**S. Y. B. Tech**

**Submitted by**

**1.Aryan Gupta ;06; (11811285)**

**2. Kaustubh Bhale;08; (11810687)**

**3.Divsehaj Singh Anand;16;(11810037)**

**May 2020**

**ABSTRACT**

In this course project we have implemented a multiplayer tic tac toe game using socket programming in C in which a server creates the game in local area network. The players in LAN can connect to the server by using the IP address if the server.

We have used the concept of thread.When a new connection arrives ,new thread will be created. To avoid race condition taking place between threads we have implemented the concept of mutex.

**Rules of the game:One by one the players have to enter a key ,‘0’ or ‘X’.A player can win the game when either of the diagonals have the same key i.e. ‘0’ or ‘x’ or any of the rows or the columns have the same key otherwise the game will result in a draw.**

**Code of the Course project:**

**CLIENT SIDE:**

#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 recv\_msg(int sockfd, char \* msg)

{

memset(msg, 0, 4);

int n = read(sockfd, msg, 3);

if (n < 0 || n != 3)

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;

}

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);

}

void error(const char \*msg)

{

perror(msg);

printf("Either the server shut down or the other player disconnected.\nGame over.\n");

exit(0);

}

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;

}

{

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 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("\n");

write\_server\_int(sockfd, move);

break;

}

else

printf("\nInvalid input. Try again.\n");

}

}

void get\_update(int sockfd, char board[][3])

{

int player\_id = recv\_int(sockfd);

int move = recv\_int(sockfd);

board[move/3][move%3] = player\_id ? 'X' : 'O';

}

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);

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")) {

printf("Waiting for other players move...\n");

}

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.");

}

printf("Game over.\n");

close(sockfd);

return 0;

}

**SERVER SIDE:**

#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 player\_count = 0;

pthread\_mutex\_t mutexcount;

void error(const char \*msg)

{

perror(msg);

pthread\_exit(NULL);

}

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.");

#ifdef DEBUG

printf("[DEBUG] Listener set.\n");

#endif

return sockfd;

}

int recv\_int(int cli\_sockfd)

{

int msg = 0;

int n = read(cli\_sockfd, &msg, sizeof(int));

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

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

return msg;

}

void write\_client\_msg(int cli\_sockfd, char \* msg)

{

int n = write(cli\_sockfd, msg, strlen(msg));

if (n < 0)

error("ERROR writing msg to client socket");

}

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");

#endif

}

num\_conn++;

}

}

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

}

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

}

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;

}

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");

}

}

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]);

pthread\_mutex\_lock(&mutexcount);

player\_count--;

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 <= 252) {

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");

}

}

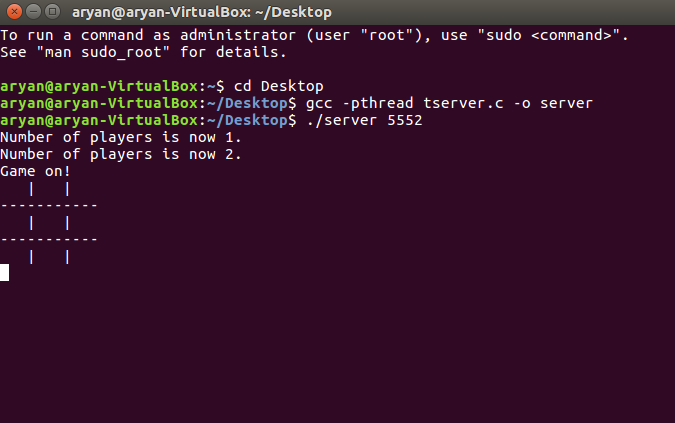
close(lis\_sockfd);

pthread\_mutex\_destroy(&mutexcount);

pthread\_exit(NULL);

}

**OUTPUTS :**

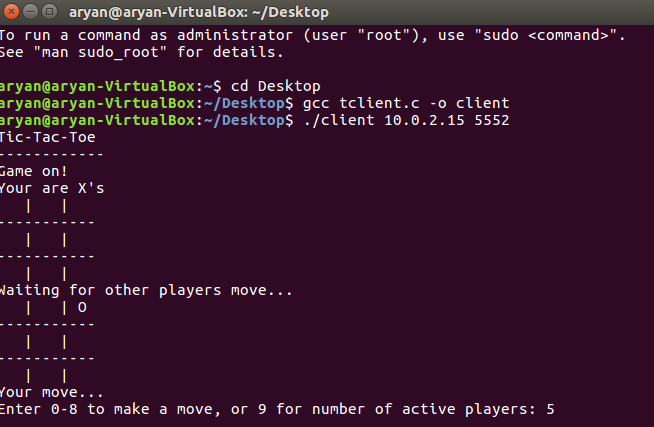


Here we are setting up the server by invoking pthread concept and creating a port known as 5552.

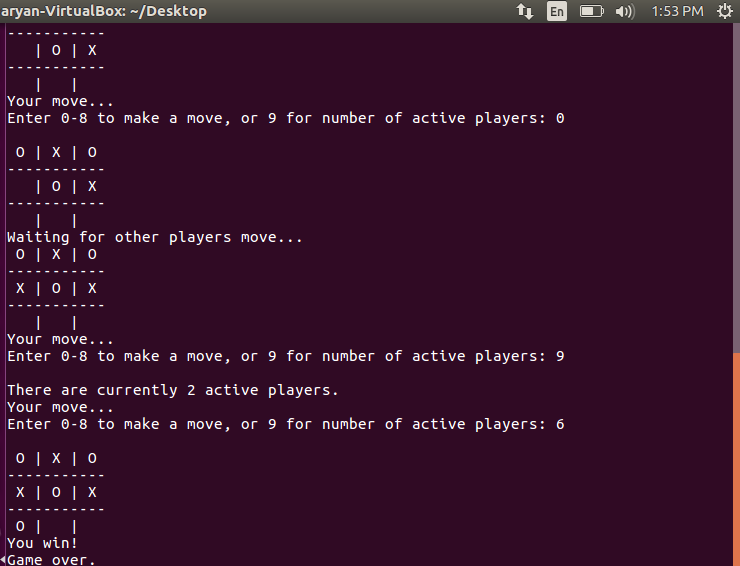
A screenshot of a cell phone

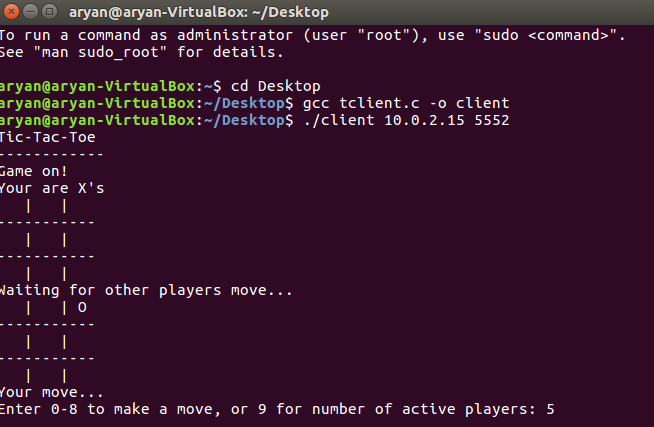
Description automatically generated

Here we are connecting to server using the same port number and internal IP address which can be calculate vy using the command ifconfig.

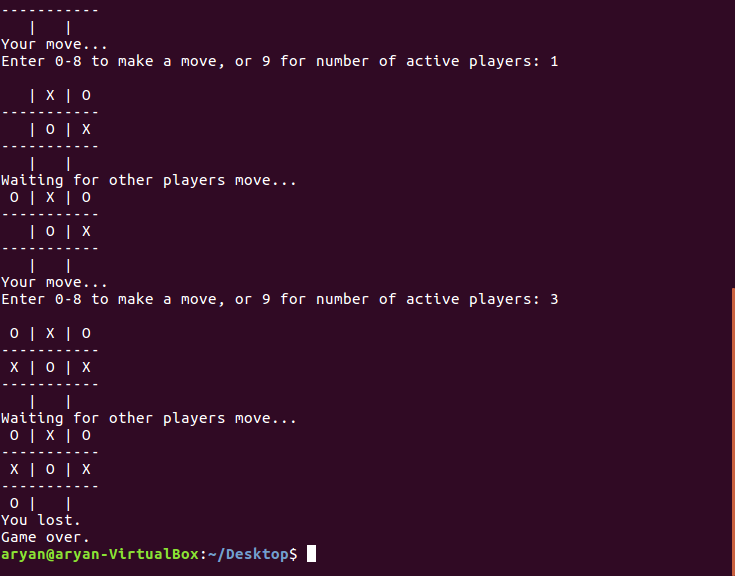


Now,the second player got connected and game has started.





Game goes on..

When player a lost the game.A screenshot of a cell phone

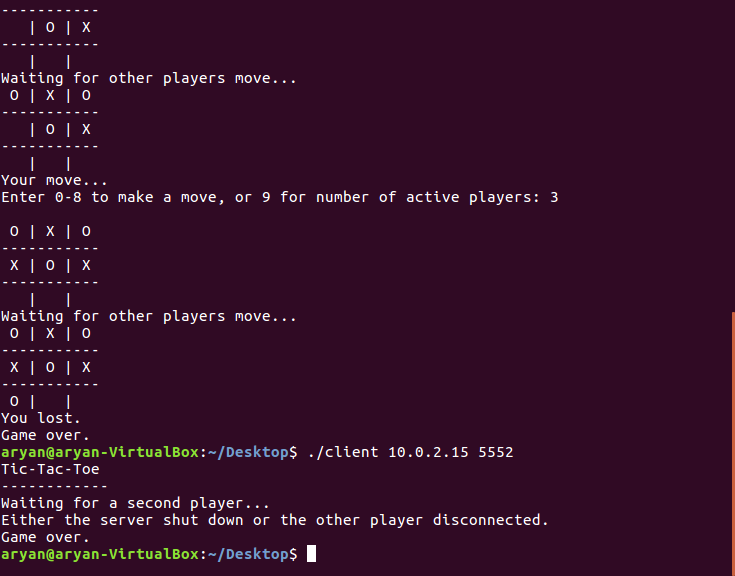
Description automatically generated

Client side when another PLAYER wins the game at the same time.

A screenshot of a cell phone

Description automatically generated

Server side when PLAYER 0 WON the GAME.

Server after getting disconnected!!

**THANK YOU**