Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и физика»

Дисциплина: «Операционные системы»

Курсовой проект.

Тема:

«Tic-Tac-Toe.Sockets»

Группа: 8О-204Б

Студент: Утенкков

Вариант: №

Оценка:

Подпись:

Москва

2016

Сервер

#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);

}

/\*

\* Socket Read Functions

\*/

/\* Reads an int from a client socket. \*/

int recv\_int(int cli\_sockfd)

{

int msg = 0;

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

if (n < 0 || n != sizeof(int)) /\* Not what we were expecting. Client likely disconnected. \*/

return -1;

#ifdef DEBUG

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

#endif

return msg;

}

/\*

\* Socket Write Functions

\*/

/\* Writes a message to a client socket. \*/

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");

}

/\* Writes an int to a client socket. \*/

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");

}

/\* Writes a message to both client sockets. \*/

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

{

write\_client\_msg(cli\_sockfd[0], msg);

write\_client\_msg(cli\_sockfd[1], msg);

}

/\* Writes an int to both client sockets. \*/

void write\_clients\_int(int \* cli\_sockfd, int msg)

{

write\_client\_int(cli\_sockfd[0], msg);

write\_client\_int(cli\_sockfd[1], msg);

}

/\*

\* Connect Functions

\*/

/\* Sets up the listener socket. \*/

int setup\_listener(int portno)

{

int sockfd;

struct sockaddr\_in serv\_addr;

/\* Get a socket to listen on \*/

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd < 0)

error("ERROR opening listener socket.");

/\* Zero out the memory for the server information \*/

memset(&serv\_addr, 0, sizeof(serv\_addr));

/\* set up the server info \*/

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = INADDR\_ANY;

serv\_addr.sin\_port = htons(portno);

/\* Bind the server info to the listener socket. \*/

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 the socket number. \*/

return sockfd;

}

/\* Sets up the client sockets and client connections. \*/

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

/\* Listen for two clients. \*/

int num\_conn = 0;

while(num\_conn < 2)

{

/\* Listen for clients. \*/

listen(lis\_sockfd, 253 - player\_count);

/\* Zero out memory for the client information. \*/

memset(&cli\_addr, 0, sizeof(cli\_addr));

clilen = sizeof(cli\_addr);

/\* Accept the connection from the client. \*/

cli\_sockfd[num\_conn] = accept(lis\_sockfd, (struct sockaddr \*) &cli\_addr, &clilen);

if (cli\_sockfd[num\_conn] < 0)

/\* Horrible things have happened. \*/

error("ERROR accepting a connection from a client.");

#ifdef DEBUG

printf("[DEBUG] Accepted connection from client %d\n", num\_conn);

#endif

/\* Send the client it's ID. \*/

write(cli\_sockfd[num\_conn], &num\_conn, sizeof(int));

#ifdef DEBUG

printf("[DEBUG] Sent client %d it's ID.\n", num\_conn);

#endif

/\* Increment the player count. \*/

pthread\_mutex\_lock(&mutexcount);

player\_count++;

printf("Number of players is now %d.\n", player\_count);

pthread\_mutex\_unlock(&mutexcount);

if (num\_conn == 0) {

/\* Send "HLD" to first client to let the user know the server is waiting on a second client. \*/

write\_client\_msg(cli\_sockfd[0],"HLD");

#ifdef DEBUG

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

#endif

}

num\_conn++;

}

}

/\*

\* Game Functions

\*/

/\* Gets a move from a client. \*/

int get\_player\_move(int cli\_sockfd)

{

#ifdef DEBUG

printf("[DEBUG] Getting player move...\n");

#endif

/\* Tell player to make a move. \*/

write\_client\_msg(cli\_sockfd, "TRN");

/\* Get players move. \*/

return recv\_int(cli\_sockfd);

}

/\* Checks that a players move is valid. \*/

int check\_move(char board[][3], int move, int player\_id)

{

if ((move == 9) || (board[move/3][move%3] == ' ')) { /\* Move is valid. \*/

#ifdef DEBUG

printf("[DEBUG] Player %d's move was valid.\n", player\_id);

#endif

return 1;

}

else { /\* Move is invalid. \*/

#ifdef DEBUG

printf("[DEBUG] Player %d's move was invalid.\n", player\_id);

#endif

return 0;

}

}

/\* Updates the board with a new move. \*/

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

}

/\* Draws the game board to stdout. \*/

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]);

}

/\* Sends a board update to both clients. \*/

void send\_update(int \* cli\_sockfd, int move, int player\_id)

{

#ifdef DEBUG

printf("[DEBUG] Sending update...\n");

#endif

/\* Signal an update \*/

write\_clients\_msg(cli\_sockfd, "UPD");

/\* Send the id of the player that made the move. \*/

write\_clients\_int(cli\_sockfd, player\_id);

/\* Send the move. \*/

write\_clients\_int(cli\_sockfd, move);

#ifdef DEBUG

printf("[DEBUG] Update sent.\n");

#endif

}

/\* Sends the number of active players to a client. \*/

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

}

/\* Checks the board to determine if there is a 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] ) { /\* Check the row for a win. \*/

#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] ) { /\* Check the column for a win. \*/

#ifdef DEBUG

printf("[DEBUG] Win by column %d.\n", col);

#endif

return 1;

}

else if (!(last\_move % 2)) { /\* If the last move was at an even numbered position we have to check the diagonal(s) as well. \*/

if ( (last\_move == 0 || last\_move == 4 || last\_move == 8) && (board[1][1] == board[0][0] && board[1][1] == board[2][2]) ) { /\* Check backslash diagonal. \*/

#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]) ) { /\* Check frontslash diagonal. \*/

#ifdef DEBUG

printf("[DEBUG] Win by frontslash diagonal.\n");

#endif

return 1;

}

}

#ifdef DEBUG

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

#endif

/\* No winner, yet. \*/

return 0;

}

/\* Runs a game between two clients. \*/

void \*run\_game(void \*thread\_data)

{

int \*cli\_sockfd = (int\*)thread\_data; /\* Client sockets. \*/

char board[3][3] = { {' ', ' ', ' '}, /\* Game Board \*/

{' ', ' ', ' '},

{' ', ' ', ' '} };

printf("Game on!\n");

/\* Send the start message. \*/

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) {

/\* Tell other player to wait, if necessary. \*/

if (prev\_player\_turn != player\_turn)

write\_client\_msg(cli\_sockfd[(player\_turn + 1) % 2], "WAT");

int valid = 0;

int move = 0;

while(!valid) { /\* We need to keep asking for a move until the player's move is valid. \*/

move = get\_player\_move(cli\_sockfd[player\_turn]);

if (move == -1) break; /\* Error reading client socket. \*/

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

valid = check\_move(board, move, player\_turn);

if (!valid) { /\* Move was invalid. \*/

printf("Move was invalid. Let's try this again...\n");

write\_client\_msg(cli\_sockfd[player\_turn], "INV");

}

}

if (move == -1) { /\* Error reading from client. \*/

printf("Player disconnected.\n");

break;

}

else if (move == 9) { /\* Send the client the number of active players. \*/

prev\_player\_turn = player\_turn;

send\_player\_count(cli\_sockfd[player\_turn]);

}

else {

/\* Update the board and send the update. \*/

update\_board(board, move, player\_turn);

send\_update( cli\_sockfd, move, player\_turn );

/\* Re-draw the board. \*/

draw\_board(board);

/\* Check for a winner/loser. \*/

game\_over = check\_board(board, move);

if (game\_over == 1) { /\* We have a winner. \*/

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) { /\* There have been nine valid moves and no winner, game is a draw. \*/

printf("Draw.\n");

write\_clients\_msg(cli\_sockfd, "DRW");

game\_over = 1;

}

/\* Move to next player. \*/

prev\_player\_turn = player\_turn;

player\_turn = (player\_turn + 1) % 2;

turn\_count++;

}

}

printf("Game over.\n");

/\* Close client sockets and decrement player counter. \*/

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);

}

/\*

\* Main Program

\*/

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

{

/\* Make sure a port was specified. \*/

if (argc < 2) {

fprintf(stderr,"ERROR, no port provided\n");

exit(1);

}

int lis\_sockfd = setup\_listener(atoi(argv[1])); /\* Listener socket. \*/

pthread\_mutex\_init(&mutexcount, NULL);

while (1) {

if (player\_count <= 252) { /\* Only launch a new game if we have room. Otherwise, just spin. \*/

int \*cli\_sockfd = (int\*)malloc(2\*sizeof(int)); /\* Client sockets \*/

memset(cli\_sockfd, 0, 2\*sizeof(int));

/\* Get two clients connected. \*/

get\_clients(lis\_sockfd, cli\_sockfd);

#ifdef DEBUG

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

#endif

pthread\_t thread; /\* Don't really need the thread id for anything in this case, but here it is anyway. \*/

int result = pthread\_create(&thread, NULL, run\_game, (void \*)cli\_sockfd); /\* Start a new thread for this game. \*/

if (result){

printf("Thread creation failed with return code %d\n", result);

exit(-1);

}

#ifdef DEBUG

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

#endif

}

}

close(lis\_sockfd);

pthread\_mutex\_destroy(&mutexcount);

pthread\_exit(NULL);

}

Клиент

#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 error(const char \*msg)

{

#ifdef DEBUG

perror(msg);

#else

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

#endif

exit(0);

}

/\*

\* Socket Read Functions

\*/

/\* Reads a message from the server socket. \*/

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

{

/\* All messages are 3 bytes. \*/

memset(msg, 0, 4);

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

if (n < 0 || n != 3) /\* Not what we were expecting. Server got killed or the other client disconnected. \*/

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

#ifdef DEBUG

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

#endif

}

/\* Reads an int from the server socket. \*/

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");

#ifdef DEBUG

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

#endif

return msg;

}

/\*

\* Socket Write Functions

\*/

/\* Writes an int to the server socket. \*/

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");

#ifdef DEBUG

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

#endif

}

/\*

\* Connect Functions

\*/

/\* Sets up the connection to the server. \*/

int connect\_to\_server(char \* hostname, int portno)

{

struct sockaddr\_in serv\_addr;

struct hostent \*server;

/\* Get a socket. \*/

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd < 0)

error("ERROR opening socket for server.");

/\* Get the address of the server. \*/

server = gethostbyname(hostname);

if (server == NULL) {

fprintf(stderr,"ERROR, no such host\n");

exit(0);

}

/\* Zero out memory for server info. \*/

memset(&serv\_addr, 0, sizeof(serv\_addr));

/\* Set up the server info. \*/

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);

/\* Make the connection. \*/

if (connect(sockfd, (struct sockaddr \*) &serv\_addr, sizeof(serv\_addr)) < 0)

error("ERROR connecting to server");

#ifdef DEBUG

printf("[DEBUG] Connected to server.\n");

#endif

return sockfd;

}

/\*

\* Game Functions

\*/

/\* Draws the game board to stdout. \*/

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]);

}

/\* Get's the players turn and sends it to the server. \*/

void take\_turn(int sockfd)

{

char buffer[10];

while (1) { /\* Ask until we receive. \*/

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");

/\* Send players move to the server. \*/

write\_server\_int(sockfd, move);

break;

}

else

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

}

}

/\* Gets a board update from the server. \*/

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

{

/\* Get the update. \*/

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

int move = recv\_int(sockfd);

/\* Update the game board. \*/

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

}

/\*

\* Main Program

\*/

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

{

/\* Make sure host and port are specified. \*/

if (argc < 3) {

fprintf(stderr,"usage %s hostname port\n", argv[0]);

exit(0);

}

/\* Connect to the server. \*/

int sockfd = connect\_to\_server(argv[1], atoi(argv[2]));

/\* The client ID is the first thing we receive after connecting. \*/

int id = recv\_int(sockfd);

#ifdef DEBUG

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

#endif

char msg[4];

char board[3][3] = { {' ', ' ', ' '}, /\* Game board \*/

{' ', ' ', ' '},

{' ', ' ', ' '} };

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

/\* Wait for the game to start. \*/

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")) { /\* Take a turn. \*/

printf("Your move...\n");

take\_turn(sockfd);

}

else if (!strcmp(msg, "INV")) { /\* Move was invalid. Note that a "TRN" message will always follow an "INV" message, so we will end up at the above case in the next iteration. \*/

printf("That position has already been played. Try again.\n");

}

else if (!strcmp(msg, "CNT")) { /\* Server is sending the number of active players. Note that a "TRN" message will always follow a "CNT" message. \*/

int num\_players = recv\_int(sockfd);

printf("There are currently %d active players.\n", num\_players);

}

else if (!strcmp(msg, "UPD")) { /\* Server is sending a game board update. \*/

get\_update(sockfd, board);

draw\_board(board);

}

else if (!strcmp(msg, "WAT")) { /\* Wait for other player to take a turn. \*/

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

}

else if (!strcmp(msg, "WIN")) { /\* Winner. \*/

printf("You win!\n");

break;

}

else if (!strcmp(msg, "LSE")) { /\* Loser. \*/

printf("You lost.\n");

break;

}

else if (!strcmp(msg, "DRW")) { /\* Game is a draw. \*/

printf("Draw.\n");

break;

}

else /\* Weird... \*/

error("Unknown message.");

}

printf("Game over.\n");

/\* Close server socket and exit. \*/

close(sockfd);

return 0;

}

Пример

Сервер

xpunter97:~/workspace $ cd kp

xpunter97:~/workspace/kp $ ./server 127.0.0.1 4400

ERROR binding listener socket.: Permission denied

xpunter97:~/workspace/kp $ ./server 4400

Number of players is now 1.

Number of players is now 2.

Game on!

| |

-----------

| |

-----------

| |

Player 0 played position 2

| | O

-----------

| |

-----------

| |

Player 1 played position 3

| | O

-----------

X | |

-----------

| |

Player 0 played position 1

| O | O

-----------

X | |

-----------

| |

Player 1 played position 5

| O | O

-----------

X | | X

-----------

| |

Player 0 played position 0

O | O | O

-----------

X | | X

-----------

| |

Player 0 won.

Game over.

Клиент 1

xpunter97:~/workspace $ cd kp

xpunter97:~/workspace/kp $ ./client 1 4400

Tic-Tac-Toe

------------

Waiting for a second player...

Game on!

Your are O's

| |

-----------

| |

-----------

| |

Your move...

Enter 0-8 to make a move, or 9 for number of active players: 2

| | O

-----------

| |

-----------

| |

Waiting for other players move...

| | O

-----------

X | |

-----------

| |

Your move...

Enter 0-8 to make a move, or 9 for number of active players: 1

| O | O

-----------

X | |

-----------

| |

Waiting for other players move...

| O | O

-----------

X | | X

-----------

| |

Your move...

Enter 0-8 to make a move, or 9 for number of active players: 0

O | O | O

-----------

X | | X

-----------

| |

You win!

Game over.

Клиент 2

xpunter97:~/workspace/kp $ ./client 2 4400

Tic-Tac-Toe

------------

Game on!

Your are X's

| |

-----------

| |

-----------

| |

Waiting for other players move...

| | O

-----------

| |

-----------

| |

Your move...

Enter 0-8 to make a move, or 9 for number of active players: 3

| | O

-----------

X | |

-----------

| |

Waiting for other players move...

| O | O

-----------

X | |

-----------

| |

Your move...

Enter 0-8 to make a move, or 9 for number of active players: 5

| O | O

-----------

X | | X

-----------

| |

Waiting for other players move...

O | O | O

-----------

X | | X

-----------

| |

You lost.

Game over.

Вывод.

Для сервера создаем сокет и дает ему адрес порта, по которому к нему будут подключаться игроки.

Так же будем использовать мьютекс, чтобы соединять двоих игроков, и игра не начиналась, пока игроков нечетное количество.

Так как игроков может больше двух – для отдельной игры будем создавать собственный поток, чтобы не происходил ненужный обмен данными между парами.