UDP Tic-Tac-Toe

Gerado por Doxygen 1.8.13

Sumário

Capítulo 1

Índice das Estruturas de Dados

1.1 Estruturas de Dados

Aqui estão as estruturas de dados, uniões e suas respectivas descrições:

Capítulo 2

Índice dos Arquivos

2.1 Lista de Arquivos

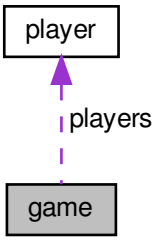
Esta é a lista de todos os arquivos documentados e suas respectivas descrições:

Capítulo 3

Estruturas

3.1 Referência da Estrutura game

Diagrama de colaboração para game:



Campos de Dados

- Player players [2]
- unsigned short number_of_players
- int first_player

3.1.1 Campos

3.1.1.1 first_player

```
int game::first_player
```

Armazena o id do jogador que começará/começou jogando essa partida.

3.1.1.2 number_of_players

```
unsigned short game::number_of_players
```

Armazena o número de jogadores conectados a essa partida.

3.1.1.3 players

```
Player game::players[2]
```

Array contendo os jogadores atualmente nessa partida.

A documentação para esta estrutura foi gerada a partir do seguinte arquivo:

- server/include/server.h

3.2 Referência da Estrutura messagem

Campos de Dados

- char data [TAM_MSG]
- struct sockaddr_in client_addr

3.2.1 Campos

3.2.1.1 client_addr

```
struct sockaddr_in messagem::client_addr
```

Endereço do remetente da mensagem

3.2.1.2 data

```
char messagem::data[TAM_MSG]
```

Conteúdo da mensagem recebida.

A documentação para esta estrutura foi gerada a partir do seguinte arquivo:

- client/include/client.h

3.3 Referência da Estrutura player

Campos de Dados

- unsigned short id
- char name [64]
- Sockaddr addr

3.3.1 Campos

3.3.1.1 addr

```
Sockaddr player::addr
```

Endereço remoto do jogador.

3.3.1.2 id

```
unsigned short player::id
```

id do jogador dentro de uma partida, podendo ser 1 ou 0.

3.3.1.3 name

```
char player::name[64]
```

Nome do jogador.

A documentação para esta estrutura foi gerada a partir do seguinte arquivo:

- server/include/server.h

3.4 Referência da Estrutura received_message

Campos de Dados

- char data [TAM_MSG]
- Sockaddr client_addr

3.4.1 Campos

3.4.1.1 client_addr

```
Sockaddr received_message::client_addr
```

Endereço do remetente da mensagem

3.4.1.2 data

```
char received_message::data[TAM_MSG]
```

Conteúdo da mensagem recebida.

A documentação para esta estrutura foi gerada a partir do seguinte arquivo:

- server/include/server.h

Capítulo 4

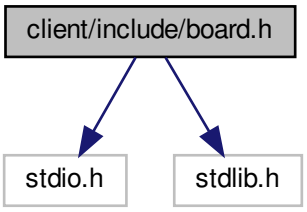
Arquivos

4.1 Referência do Arquivo client/include/board.h

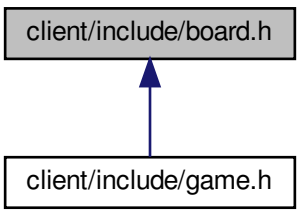
Arquivo contendo as estruturas e cabeçalhos de funções do tabuleiro.

```
#include <stdio.h>
#include <stdlib.h>
```

Gráfico de dependência de inclusões para board.h:



Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com este arquivo:



Funções

- void DrawBoard (char gameBoard[9])

  Constroi o tabuleiro do jogo.
- void DrawBoardWithNames (char gameBoard[9], char ∗player1Name, char ∗player2Name, char ∗player↩ Character)

  Constroi o tabuleiro do jogo.
- int CheckPlayerWin (char gameBoard[9])

  Constroi o tabuleiro do jogo.
- int checkValidPlay (char ∗gameBoard, int position)

  Verifica a jogada.
- void clear_screen (void)

  Limpa a tela.

4.1.1 Descrição Detalhada

Arquivo contendo as estruturas e cabeçalhos de funções do tabuleiro.

Autor

Vitor Correa da Silva

Data

29 May 2020

Esse arquivo contém as definições das funções que são implementadas em board.c. Essas funções tem como objetivo implementar o tabuleiro do jogo e suas especificações.

4.1.2 Funções

4.1.2.1 CheckPlayerWin()

```
int CheckPlayerWin (
            char gameBoard[9] )
```

Constroi o tabuleiro do jogo.

Verifica se houve uma vitoria (1), empate (2) ou se o jogo continua (0)

Parâmetros

gameBoard	Array de 9 posições que guarda as informações do tabuleiro

Retorna

`int` Verifica se a jogada foi a vencedora

4.1.2.2 checkValidPlay()

```
int checkValidPlay (
            char * gameBoard,
            int position )
```

Verifica a jogada.

Verifica se a jogada feita é valida.

Parâmetros

gameBoard	Array de 9 posições que guarda as informações do tabuleiro
position	Valor da possivel jogada

Retorna

`int` retorna a jogada feita se for valida

4.1.2.3 clear_screen()

```
void clear_screen (
            void  )
```

Limpa a tela.

Implementa função para lipar a tela.

4.1.2.4 DrawBoard()

```
void DrawBoard (
            char gameBoard[9] )
```

Constroi o tabuleiro do jogo.

Limpa a tela e desenha o tabuleiro de jogo.

Parâmetros

gameBoard	Array de 9 posições que guarda as informações do tabuleiro

4.1.2.5 DrawBoardWithNames()

```
void DrawBoardWithNames (
            char gameBoard[9],
            char * player1Name,
            char * player2Name,
            char * playerCharacter )
```

Constroi o tabuleiro do jogo.

Limpa a tela, desenha o tabuleiro e informa de qual jogador é a vez.

Parâmetros

gameBoard	Array de 9 posições que guarda as informações do tabuleiro
player1Name	Nome do jogador numero 1
player2Name	Nome do jogador numero 2
playerCharacter	Caractere do jogador atual

4.2 Referência do Arquivo client/include/client.h

Arquivo contendo as estruturas e cabeçalhos de funções do cliente.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/time.h>
#include <stdlib.h>
#include <memory.h>
#include <ifaddrs.h>
#include <net/if.h>
#include <stdarg.h>
#include <time.h>
#include <math.h>
#include <sys/termios.h>
#include <menu.h>
```

Gráfico de dependência de inclusões para client.h:

Estruturas de Dados

- struct messagem

Definições e Macros

- #define **TAM_MSG** 1024

Definições de Tipos

- typedef struct messagem Mensagem

  Mensagem recebida.

Funções

- int create_client_socket (void)

  Cria um socket udp.
- int rand_range (int min, int max)

  Gerador de número aleatório dentro de um intervalo.
- Mensagem receive_message (int sockfd)

  Espera pela chegada de uma mensagem em um socket.
- int envia_mensagem (int socketfd, char ∗msg, char ∗host, int port)

  Envia uma mensagem para um endereço via socket udp.
- int login (int socket, char ∗host, int port)

  Realiza o login no servidor.
- int play_tictactoe (int socket, char ∗host, int port)

  Implementa com o uso das outras funções o jogo.
- void clear_stdin (void)

  Limpa caracteres.
- void disconnect (int socket, char ∗host, int port)

  Desconecta do socket.

4.2.1 Descrição Detalhada

Arquivo contendo as estruturas e cabeçalhos de funções do cliente.

Autor

Vitor Correa da Silva

Data

29 May 2020

Esse arquivo contém as definições das funções que são implementadas em client.c. Essas funções tem como objetivo implementar um client udp que faz login no servidor, envia e recebe mensagens.

4.2.2 Definições dos tipos

4.2.2.1 Mensagem

`Mensagem`

Mensagem recebida.

Mensagem guarda o conteúdo e informações referentes a uma mensagem recebida via socket UDP.

4.2.3 Funções

4.2.3.1 clear_stdin()

```
void clear_stdin (
             void  )
```

Limpa caracteres.

Limpa as entradas de teclado desnecessarias que ficam salvas depois que é realizado o scanf

Parâmetros

void	Não há parametros

Retorna

Não há retorno

4.2.3.2 create_client_socket()

```
int create_client_socket (
             void  )
```

Cria um socket udp.

Faz a criação de um socket do tipo UDP e o associa a porta indicada no argumento `port` .

Parâmetros

void	Não há parametros.

Retorna

`int` contendo o id do descritor desse socket. Retorna **-1** em caso de erro.

Exemplo de uso:

```
int socket;
socket = create_socket(80);
if (socket != -1) {
   printf("socket aberto com sucesso.\n");
}
```

4.2.3.3 disconnect()

```
void disconnect (
             int socket,
             char * host,
             int port )
```

Desconecta do socket.

Envia mensagem para o servidor para se desconectar

Parâmetros

socket	id do descritor do socket que realizará o envio da mensagem.
host	Host de conexao com o servidor
port	porta de conexão com o servidor

Retorna

Não há retorno

4.2.3.4 envia_mensagem()

```
int envia_mensagem (
             int socketfd,
             char * msg,
             char * host,
             int port )
```

Envia uma mensagem para um endereço via socket udp.

Função que realiza o envio de uma mensagem de texto via udp. Não há confirmação de envio, ela realiza o envio e segue a execução do programa. É utilizada para a comunicação com o servidor.

Parâmetros

socketfd	id do descritor do socket que realizará o envio da mensagem.
msg	texto da mensagem a ser enviada.
host	Endereço do cliente que será enviada a mensagem.
host	Porta de conexão com o cliente que receberá a mensagem

Retorna

Não há retorno.

Exemplo de uso:

```
int socket;
char msg[1024];
ReceivedMessage m;

socket = create_socket(80);
if (socket == -1)
{
    exit();
}
// espera pelo contato de um cliente.
m = receive_message(socket);

// envia "HELLO" de volta para o cliente.
strcpy(msg, "HELLO");
send_message(socket, msg, m.addr);
```

Aviso

O socket enviado por parâmetro deve ter sido criado previamente através da função create_socket()

4.2.3.5 login()

```
int login (
              int socket,
              char * host,
              int port )
```

Realiza o login no servidor.

Envia um pedido de login para o servidor e espera pela porta que será usada para começar o jogo.

Parâmetros

socket	id do descritor do socket que realizará o envio da mensagem.
host	Host de conexao com o servidor
port	porta de conexão com o servidor

Retorna

retorna o numero da porta que sera utilizadao para jogar

4.2.3.6 rand_range()

```
int rand_range (
              int min,
              int max )
```

Gerador de número aleatório dentro de um intervalo.

Gera um número aleatório dentro do intervalo [`min` , `max`]

Parâmetros

min	Valor mínimo do intervalo
max	Valor máximo do intervalo

Retorna

`int` Valor aleatório gerado.

Exemplo de uso:

```
int nro_aleatorio;

nro_aleatorio = rand_range(1,100)

printf("Valor aleatório entre 1 e 100: %d\n", nro_aleatorio);
```

4.2.3.7 receive_message()

```
Mensagem receive_message (
              int sockfd )
```

Espera pela chegada de uma mensagem em um socket.

Essa função inicia a espera de uma mensagem.

Parâmetros

sockfd	id do descritor do socket que aguardará por uma mensagem.

Retorna

Estrutura `Mensagem` contendo informações da mensagem que o socket recebeu.

Exemplo de uso:

```
int socket;
Mensagem m;

socket = create_socket(80);
if (socket == -1)
{
    exit();
}
m = receive_message(socket);

printf("Mensagem recebida: %s\n", m.data);
```

Aviso

O socket enviado por parâmetro deve ter sido criado previamente através da função create_socket()

Essa função inicia a espera de uma mensagem e **bloqueia** o programa nesse estado. O programa só é desbloqueado com a chegada de uma mensagem.

Parâmetros

sockfd	id do descritor do socket que aguardará por uma mensagem.

Retorna

Estrutura `ReceivedMessage` contendo informações da mensagem que o socket recebeu.

Exemplo de uso:

```
int socket;
ReceivedMessage m;

socket = create_socket(80);
if (socket == -1)
{
    exit();
}
m = receive_message(socket);

printf("Mensagem recebida: %s\n", m.data);
```

Aviso

O socket enviado por parâmetro deve ter sido criado previamente através da função create_socket()

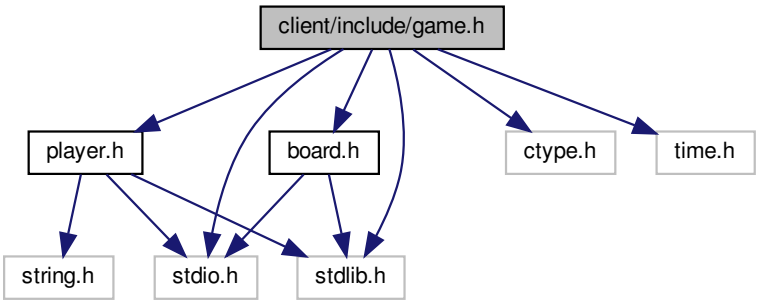
4.3 Referência do Arquivo client/include/game.h

Arquivo contendo as estruturas e cabeçalhos de funções do jogo.

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <ctype.h>
#include <time.h>
#include <player.h>
#include <board.h>
```

Gráfico de dependência de inclusões para game.h:



Funções

- void PrintScore (int player1Wins, int player2Wins, int draws, char ∗player1Name, char ∗player2Name)

  Mostrar a pontuação.
- void PrintWinner (char ∗playerCharacter, char ∗player1Name, char ∗player2Name, int check)

  Mostrar a nome do vencedor.
- void PlayAgain (int ∗newGame)

  Verificar nova partida.
- void StartGame (int numberOfPlayers)

  Começar uma partida.
- int AIPlay (char gameBoard[9], int ∗chosenNumber)

  Jogada do computador.
- void PlayerStart (char ∗playerCharacter, int ∗playerTurn, int numberOfPlayers, char ∗player1Name, char ∗player2Name)

  Definir primeiro jogador.
- void UpdateScores (int check, char ∗playerCharacter, int ∗player1Wins, int ∗player2Wins, int ∗draws, char gameBoard[9])

  Atualiza o marcador de pontos.
- void PlayGame (char ∗player1Name, char ∗player2Name, int numberOfPlayers)

  Implementa o jogo e suas funções.

4.3.1 Descrição Detalhada

Arquivo contendo as estruturas e cabeçalhos de funções do jogo.

Autor

Vitor Correa da Silva

Data

29 May 2020

Esse arquivo contém as definições das funções que são implementadas em game.c. Essas funções tem como objetivo implementar o jogo tic-tac-toe (jogo da velha).

4.3.2 Funções

4.3.2.1 AIPlay()

```
int AIPlay (
            char gameBoard[9],
            int * chosenNumber )
```

Jogada do computador.

Implementa e simula a jogada do computador. Só é utilizada esta função quando apenas um jogador esta na partida.

Parâmetros

gameBoard	Array de 9 posições que guarda as informações do tabuleiro
chosenNumber	Valor da possivel jogada

Retorna

`int` Valor escolhido pelo jogador

4.3.2.2 PlayAgain()

```
void PlayAgain (
            int * newGame )
```

Verificar nova partida.

Verifica se o usuario deseja iniciar uma nova partida, caso nao queira, retorna para o menu.

Parâmetros

newGame	Verficador de novo jogo. **1** para começar uma nova partida

4.3.2.3 PlayerStart()

```
void PlayerStart (
            char * playerCharacter,
            int * playerTurn,
            int numberOfPlayers,
```

```
            char * player1Name,
            char * player2Name )
```

Definir primeiro jogador.

Define e mostra na tela qual jogador ira comecar a partida.

Parâmetros

playerCharacter	Caractere do jogador atual
playerTurn	Define o turno do jogador
numberOfPlayers	Numero de jogadores na partida
player1Name	Nome do jogador numero 1
player2Name	Nome do jogador numero 2

4.3.2.4 PlayGame()

```
void PlayGame (
            char * player1Name,
            char * player2Name,
            int numberOfPlayers )
```

Implementa o jogo e suas funções.

Funcao que controla o fluxo do jogo.

Parâmetros

player1Name	Nome do jogador numero 1
player2Name	Nome do jogador numero 2
numberOfPlayers	Numero de jogadores na partida

4.3.2.5 PrintScore()

```
void PrintScore (
            int player1Wins,
            int player2Wins,
            int draws,
            char * player1Name,
            char * player2Name )
```

Mostrar a pontuação.

Escreve a pontuacao atual dos jogadores na tela.

Parâmetros

player1Wins	Numero de vitorias do jogador 1
player2Wins	Numero de vitorias do jogador 2
draws	Numero de empates entre os dois jogadores
player1Name	Nome do jogador numero 1
player2Name	Nome do jogador numero 2

4.3.2.6 PrintWinner()

```
void PrintWinner (
            char * playerCharacter,
            char * player1Name,
            char * player2Name,
            int check )
```

Mostrar a nome do vencedor.

Escreve na tela o nome do vencedor, se houve algum, ou se houve empate

Parâmetros

playerCharacter	Caractere do joogador. Pode ser **X** ou **O**
player1Name	Nome do jogador 1
player2Name	Nome do jogador 2
check	Verificador de vitoria ou empate. **1** se houver vencedo ou **2** se for um empate

4.3.2.7 StartGame()

```
void StartGame (
            int numberOfPlayers )
```

Começar uma partida.

Define do nome dos jogadores e chama a funcao de inicio do jogo.

Parâmetros

numberOfPlayers	Numero de jogadores na partida

4.3.2.8 UpdateScores()

```
void UpdateScores (
```

```
int check,
char * playerCharacter,
int * player1Wins,
int * player2Wins,
int * draws,
char gameBoard[9] )
```

Atualiza o marcador de pontos.

Atualiza os placares e altera o valor da variavel que controla o loop (win).

Parâmetros

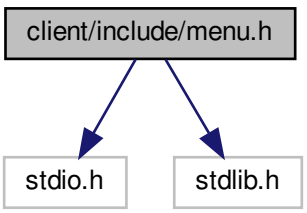
check	Verificador de vitoria ou empate. **1** se houver vencedo ou **2** se for um empate
playerCharacter	Caractere do joogador. Pode ser **X** ou **O**
player1Wins	Numero de vitorias do jogador 1
player2Wins	Numero de vitorias do jogador 2
draws	Numero de empates entre os dois jogadores
gameBoard	Array de 9 posições que guarda as informações do tabuleiro

4.4 Referência do Arquivo client/include/menu.h

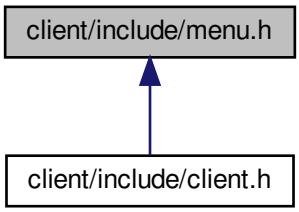
Arquivo contendo as estruturas e cabeçalhos de funções do menu.

```
#include <stdio.h>
#include <stdlib.h>
```

Gráfico de dependência de inclusões para menu.h:



Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com este arquivo:



Funções

- void ShowMenu ()

  Mostra o menu.
- void MenuChoice (int choice, int ∗numberOfPlayers)

  Chama a função escolhida pelo jogador.

4.4.1 Descrição Detalhada

Arquivo contendo as estruturas e cabeçalhos de funções do menu.

Autor

Vitor Correa da Silva

Data

29 May 2020

Esse arquivo contém as definições das funções que são implementadas em menu.c. Essas funções tem como objetivo implementar um menu interativo.

4.4.2 Funções

4.4.2.1 MenuChoice()

```
void MenuChoice (
            int choice,
            int * numberOfPlayers )
```

Chama a função escolhida pelo jogador.

Espera um **int** para definir a escolha do jogador e depois faz a chamada da função escolhida.

Parâmetros

choice	Escolha do jogador
numberOfPlayers	Numero de jogadores

4.4.2.2 ShowMenu()

```
void ShowMenu ( )
```

Mostra o menu.

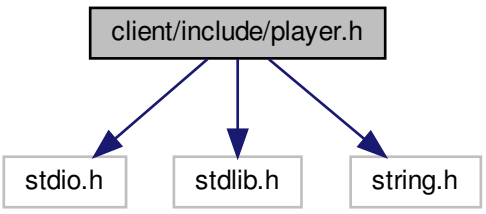
Implemtação do texto que é mostrado no menu

4.5 Referência do Arquivo client/include/player.h

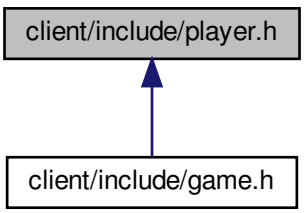
Arquivo contendo cabeçalhos de funções que implementam as funções do player.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Gráfico de dependência de inclusões para player.h:



Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com este arquivo:



Funções

- void ChangePlayer (char *playerCharacter)

  Muda o caractere do jogador atual.
- int GetNumberOfPlayers ()

  Retorna o numero de jogadores.
- char * GetPlayerName (int playerNumber)

  Define o nome do jogador.

4.5.1 Descrição Detalhada

Arquivo contendo cabeçalhos de funções que implementam as funções do player.

Autor

Vitor Correa da Silva

Data

29 May 2020

Esse arquivo contém as definições das funções que são implementadas em player.c. Essas funções tem como objetivo implementar as operações de escolha de caracter para os jogadores, escolha do numero de jogadores e define o nome do jogador.

4.5.2 Funções

4.5.2.1 ChangePlayer()

```
void ChangePlayer (
            char * playerCharacter )
```

Muda o caractere do jogador atual.

Decide o caractere a ser usado pelo jogador.

Parâmetros

playerCharacter	Ponteiro que guarda a informação de caractere do jogador.

4.5.2.2 GetNumberOfPlayers()

```
int GetNumberOfPlayers ( )
```

Retorna o numero de jogadores.

Implementa a escolha do numero de jogadores 1 ou 2.

Retorna

`int` contendo o numero de jogadores da partida. Retorna **1** ou **2** jogadores.

4.5.2.3 GetPlayerName()

```
char* GetPlayerName (
            int playerNumber )
```

Define o nome do jogador.

Define o nome do jogador que é retornornado como ponteiro `*char`

Parâmetros

playerNumber	Contem o numero do jogador que deve alterar o nome. **0** para computador **1** ou **2** para os jogadores.

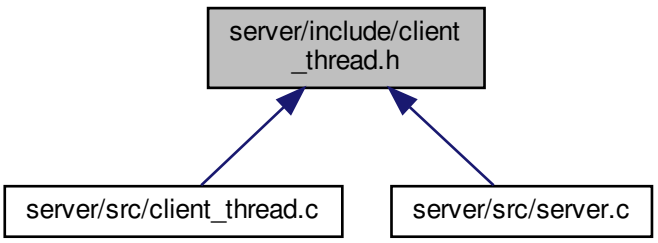
Retorna

`*char` contendo o nome do jogador.

4.6 Referência do Arquivo server/include/client_thread.h

Arquivo contendo cabeçalhos de funções que implementam as interações do servidor com os clientes conectados.

Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com este arquivo:



Definições e Macros

- #define **TAM_JOGADA** 1089

Funções

- int init_shared_variables (void)

  Inicializa as variáveis compartilhadas.
- void ∗ client_connection_thread (void ∗client_address)

  Thread de conexão dedicada a um cliente.
- void start_remote_tictactoe_game (int socket, Player p, unsigned short game_id)

  Inicia uma partida de tic-tac-toe com o jogador indicado.
- int search_for_available_game (void)

  Procura por um jogo com vagas abertas.
- int insert_player_in_game (Player p, int game_id)

  Insere um jogador em uma partida.
- void wait_all_players_to_connect (int game_id)

  Espera até que todos os jogadores se conectem em uma partida.
- void write_in_file_ranking (char ∗msg)

  Cria um arquivo com o nome dos vencedores.
- void leave_game (int game_id)

  Abandona uma partida.

4.6.1 Descrição Detalhada

Arquivo contendo cabeçalhos de funções que implementam as interações do servidor com os clientes conectados.

Autor

Vitor Correa da Silva

Data

29 May 2020

Esse arquivo contém as definições das funções que são implementadas em client_thread.c. Essas funções tem como objetivo implementar o funcionamento da troca de mensagens entre servidor e clientes quando estes já estão conectados.

4.6.2 Funções

4.6.2.1 client_connection_thread()

```
void* client_connection_thread (
            void * client_address )
```

Thread de conexão dedicada a um cliente.

Essa função é chamada em forma de thread sempre que um usuário realiza login no servidor, recebendo o endereço `client_address` do client como parâmetro. Essa função faz as seguintes ações:

- Realizar a abertura de um socket dedicado para o cliente, e envia o endereço do socket para o cliente.
- Receber e armazenar o nome do cliente conectado.
- Chamar a função `search_for_available_game` para encontrar uma partida para esse jogador.
- Chamar a função `insert_player_in_game` para inserir esse jogador na partida encontrada.
- Caso o jogador for o primeiro a se conectar a partida, espera até que o segundo se conecte chamando a função `wait_all_players_to_connect`.
- Caso o jogador for o segundo a se conectar na partida, realiza o sorteio de quem é o primeiro a jogar.
- Avisa os jogadores se são os primeiros ou segundos a jogar.
- Inicia a partida dos jogadores com a função `start_remote_tictactoe_game`.
- Fecha os socket abertos, ao fim da partida.

Parâmetros

client_address	Ponteiro para o endereço do cliente conectado.

Retorna

Não há retorno.

4.6.2.2 init_shared_variables()

```
int init_shared_variables (
            void  )
```

Inicializa as variáveis compartilhadas.

Essa função inicializa as variáveis que são compartilhadas entre as threads de conexão com os clientes. Inicializa os seguintes recursos:

- pthread_mutex_t lock
- Game tictactoe[NRO_PARTIDAS_SIMULTANEAS]

`lock` é o mutex utilizado para controlar o acesso à variável `tictactoe` , realizando a exclusão mútua de acesso à mesma. Fazendo com que não mais que uma thread possa adentrar uma região de código trancada por essa variável.

`tictactoe` é um array que contém todos os jogos do servidor, e é utilizado pelas várias threads de clientes para ler as informações da partida, entrar em partidas, etc.

Retorna

resultado da inicialização: 0 em caso de sucesso e 1 em caso de erro.

4.6.2.3 insert_player_in_game()

```
int insert_player_in_game (
            Player p,
            int game_id )
```

Insere um jogador em uma partida.

Insere o jogador `p` dentro da partida `game_id`, na primeira ou segunda posição.

Parâmetros

p	Jogador que será inserido.
game_id	Partida que terá um jogador inserido.

Retorna

`int` id do jogador dentro dessa partida, sendo 0 ou 1

Exemplo de uso:

```
Player p;
int game_id;

game_id = search_for_available_game();
if (game_id == -1)
{
    printf("Não há partidas com vagas.\n");
}
else
{
    p.id = insert_player_in_game(p, game_id);
}
```

Aviso

É recomendável que essa função seja chamada dentro do bloqueio do mutex. Para impedir que mais de 2 jogadores tentem entrar na mesma partida ao mesmo tempo e quebre o limite de jogadores por partida.

4.6.2.4 leave_game()

```
void leave_game (
            int game_id )
```

Abandona uma partida.

Decrementa em um o número de jogadores de uma partida;

Parâmetros

game↩ _id	id da partida.

Retorna

Não há retorno.

4.6.2.5 search_for_available_game()

```
int search_for_available_game (
            void  )
```

Procura por um jogo com vagas abertas.

Faz a busca por uma partida com vagas abertas no servidor, o procedimento de busca é o seguinte:

- Se houver uma partida com um player esperando um oponente, dá preferência a essa partida.
- Se não houver partida com um player esperando, procura pela primeira partida com duas vagas abertas.
- Se todas as partidas estiverem cheias, retorna **-1** , indicando que não há vagas.

Retorna

Id da partida encontrada.

Exemplo de uso:

```
Player p;
int game_id;

game_id = search_for_available_game();
if (game_id == -1)
{
    printf("Não há partidas com vagas.\n");
}
else
{
    p.id = insert_player_in_game(p, game_id);
}
```

Aviso

É recomendável que essa partida seja chamada dentro do bloqueio do mutex, para evitar que as informações das partidas sejam alteradas durante a busca.

4.6.2.6 start_remote_tictactoe_game()

```
void start_remote_tictactoe_game (
            int socket,
            Player p,
            unsigned short game_id )
```

Inicia uma partida de tic-tac-toe com o jogador indicado.

Essa função implementa a troca de mensagens de um jogador com a partida, e deve ser chamada pelos dois jogadores. Um é o jogador que foi sorteado para começar, o outro para ser o segundo a jogar. A razão para isso é que a conexão dos dois clientes é tratada independentemente em duas threads, e a função é chamada quase que simultaneamente pelas duas threads ao fim da função `wait_all_players_to_connect`.

Quando a partida acaba, é chamada a função `leave_game`.

Parâmetros

socket	Socket que o servidor esperará por mensagens desse jogador e enviará jogadas a seu oponente.
p	Jogador que participará da partida.
game_id	id da partida que está iniciando.

Retorna

Não há retorno.

4.6.2.7 wait_all_players_to_connect()

```
void wait_all_players_to_connect (
            int game_id )
```

Espera até que todos os jogadores se conectem em uma partida.

Bloqueia a execução do programa até que a partida de id `game_id` esteja cheia.

Parâmetros

game_id	id da partida

Retorna

Não há retorno.

Aviso

Se essa função for chamada dentro do bloqueio do mutex, ela nunca será finalizada.

4.6.2.8 write_in_file_ranking()

```
void write_in_file_ranking (
            char * msg )
```

Cria um arquivo com o nome dos vencedores.

Cria um novo arquivo ou abre um ja existente e escreve o nome do vencedor da partida.

Parâmetros

msg	nome do jogador vencedor

Retorna

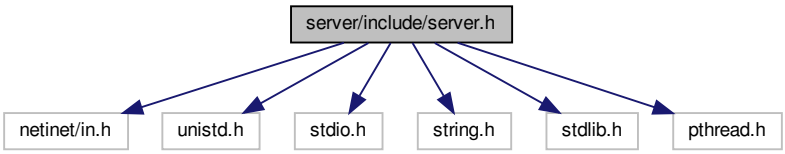
Não há retorno.

4.7 Referência do Arquivo server/include/server.h

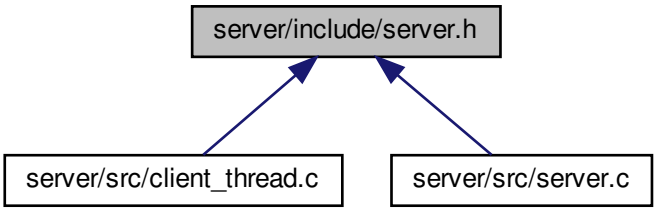
Arquivo contendo as estruturas e cabeçalhos de funções do servidor.

```
#include <netinet/in.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <pthread.h>
```

Gráfico de dependência de inclusões para server.h:



Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com este arquivo:



Estruturas de Dados

- struct received_message
- struct player
- struct game

Definições e Macros

- #define **TAM_MSG** 1024
- #define **NRO_PARTIDAS_SIMULTANEAS** 1000

Definições de Tipos

- typedef struct sockaddr_in Sockaddr

  Endereço Remoto.
- typedef struct received_message ReceivedMessage

  Mensagem recebida.
- typedef struct player Player

  Jogador conectado.
- typedef struct game Game

  Partida de tic-tac-toe.

Funções

- int create_socket (int port)

  Cria um socket udp.
- ReceivedMessage receive_message (int sockfd)

  Espera pela chegada de uma mensagem em um socket.
- int rand_range (int min, int max)

  Gerador de número aleatório dentro de um intervalo.
- void send_message (int socket, char ∗msg, Sockaddr client_addr)

  Envia uma mensagem para um endereço via socket udp.
- void wait_for_login (int socket)

  Realiza a espera por logins no servidor.
- int init_server (void)

  Realiza a inicialização do servidor.

4.7.1 Descrição Detalhada

Arquivo contendo as estruturas e cabeçalhos de funções do servidor.

Autor

Vitor Correa da Silva

Data

29 May 2020

Esse arquivo contém as definições das funções que são implementadas em server.c. Essas funções tem como objetivo implementar um servidor udp que possibilita o login e gerenciamento de múltiplos jogadores para a realização de partidas de tic-tac-toe (jogo da velha) de forma remota.

4.7.2 Definições dos tipos

4.7.2.1 Game

`Game`

Partida de tic-tac-toe.

Estrutura que armazena as informações de uma partida de tic-tac-toe que esteja acontecendo no servidor.

4.7.2.2 Player

`Player`

Jogador conectado.

Estrutura que armazena as informações de um jogador conectado ao servidor.

4.7.2.3 ReceivedMessage

`ReceivedMessage`

Mensagem recebida.

ReceivedMessage guarda o conteúdo e informações referentes a uma mensagem recebida via socket UDP.

4.7.2.4 Sockaddr

`Sockaddr`

Endereço Remoto.

É utilizado como alias à estrutura sockaddr_in da biblioteca netinet/in.h. Essa estrutura guarda as informações de endereço remoto de outros sockets.

4.7.3 Funções

4.7.3.1 create_socket()

```
int create_socket (
            int port )
```

Cria um socket udp.

Faz a criação de um socket do tipo UDP e o associa a porta indicada no argumento `port` .

Parâmetros

port	Número da porta UDP que será associada a esse socket, deve ser um valor entre 1 e 65536.

Retorna

`int` contendo o id do descritor desse socket. Retorna **-1** em caso de erro.

Exemplo de uso:

```
int socket;
socket = create_socket(80);
if (socket != -1) {
   printf("socket aberto com sucesso.\n");
}
```

4.7.3.2 init_server()

```
int init_server (
             void  )
```

Realiza a inicialização do servidor.

Essa função é destinada a guardar todas as instruções que devem ser executadas no inicio do programa. Atualmente, ela inicializa o sistema de geração de números aleatórios, para que cada execução do servidor disponha de valores aleatórios diferentes.

Retorna

Não há retorno.

4.7.3.3 rand_range()

```
int rand_range (
             int min,
             int max )
```

Gerador de número aleatório dentro de um intervalo.

Gera um número aleatório dentro do intervalo [`min` , `max`]

Parâmetros

min	Valor mínimo do intervalo
max	Valor máximo do intervalo

Retorna

`int` Valor aleatório gerado.

Exemplo de uso:

```
int nro_aleatorio;

nro_aleatorio = rand_range(1,100)

printf("Valor aleatório entre 1 e 100: %d\n", nro_aleatorio);
```

4.7.3.4 receive_message()

```
ReceivedMessage receive_message (
            int sockfd )
```

Espera pela chegada de uma mensagem em um socket.

Essa função inicia a espera de uma mensagem e **bloqueia** o programa nesse estado. O programa só é desbloqueado com a chegada de uma mensagem.

Parâmetros

sockfd	id do descritor do socket que aguardará por uma mensagem.

Retorna

Estrutura `ReceivedMessage` contendo informações da mensagem que o socket recebeu.

Exemplo de uso:

```
int socket;
ReceivedMessage m;

socket = create_socket(80);
if (socket == -1)
{
    exit();
}
m = receive_message(socket);

printf("Mensagem recebida: %s\n", m.data);
```

Aviso

O socket enviado por parâmetro deve ter sido criado previamente através da função create_socket()

4.7.3.5 send_message()

```
void send_message (
            int socket,
            char * msg,
            Sockaddr client_addr )
```

Envia uma mensagem para um endereço via socket udp.

Função que realiza o envio de uma mensagem de texto via udp. Essa função, diferentemente de `receive_message` , é não bloqueante. O que significa que não há confirmação de envio, ela realiza o envio e segue a execução do programa. É utilizada para a comunicação com os clientes conectados.

Parâmetros

socket	id do descritor do socket que realizará o envio da mensagem.
msg	texto da mensagem a ser enviada.
client_addr	Endereço do cliente que será enviada a mensagem.

Retorna

Não há retorno.

Exemplo de uso:

```
int socket;
char msg[1024];
ReceivedMessage m;

socket = create_socket(80);
if (socket == -1)
{
    exit();
}
// espera pelo contato de um cliente.
m = receive_message(socket);

// envia "HELLO" de volta para o cliente.
strcpy(msg, "HELLO");
send_message(socket, msg, m.addr);
```

Aviso

O socket enviado por parâmetro deve ter sido criado previamente através da função create_socket()

4.7.3.6 wait_for_login()

```
void wait_for_login (
            int socket )
```

Realiza a espera por logins no servidor.

Essa é a função principal do servidor, fazendo a espera por novas conexões e realizando a alocação de recursos para lidar com os diversos clientes conectados.

Quando um cliente se conecta no servidor, enviando um "LOGIN" para o mesmo, o servidor responde criando uma thread para tratamento dedicado desse cliente, essa lógica está implementada nessa função.

Parâmetros

socket	Socket que receberá o login de clientes.

Retorna

Não há retorno.

Exemplo de uso:

```
int socket;

socket = create_socket(80);
if (socket == -1)
{
    exit();
}

wait_for_login(socket);
```

Aviso

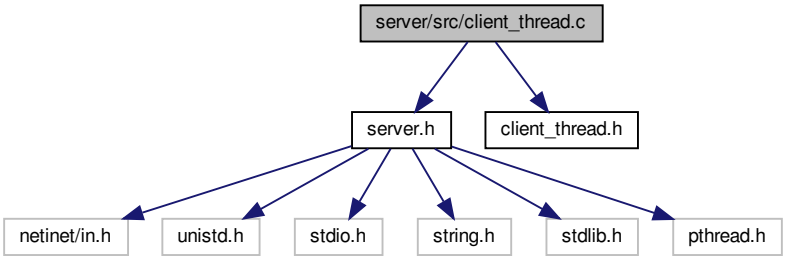
O socket enviado por parâmetro deve ter sido criado previamente através da função create_socket()

4.8 Referência do Arquivo server/src/client_thread.c

Implementação das funções de conexão via thread dedicada.

```
#include <server.h>
#include <client_thread.h>
```

Gráfico de dependência de inclusões para client_thread.c:



Variáveis

- Game **tictactoe** [NRO_PARTIDAS_SIMULTANEAS]
- pthread_mutex_t **lock**

4.8.1 Descrição Detalhada

Implementação das funções de conexão via thread dedicada.

Autor

Vitor Correa da Silva

Data

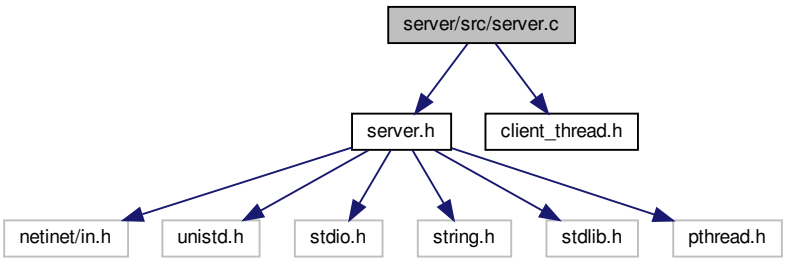
29 May 2020

4.9 Referência do Arquivo server/src/server.c

Implementação das funções do servidor.

```
#include <server.h>
#include <client_thread.h>
```

Gráfico de dependência de inclusões para server.c:



Funções

- void show_usage (char *bin_name)

  Exibe mensagem de utilização.
- int main (int argc, char **argv)

  Função main do servidor.

4.9.1 Descrição Detalhada

Implementação das funções do servidor.

Autor

Vitor Correa da Silva

Data

29 May 2020

4.9.2 Funções

4.9.2.1 main()

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

Função main do servidor.

Realiza a leitura dos parâmetros de linha de comando, e valida-os. Caso os parâmetros sejam incorretos, exibe a mensagem de utilização chamando a função `show_usage`

Após isso, a função main realiza a inicialização do servidor chamando as funções `init_server` e `init_shared_variables` , cria um socket e inicia a espera por logins através da função

Caso os parâmetros sejam válidos

Parâmetros

argc	Número de argumentos passados por parâmetro
argv	Array de argumentos do tipo ∗char

Retorna

int

4.9.2.2 show_usage()

```
void show_usage (
            char * bin_name )
```

Exibe mensagem de utilização.

Exibe a mensagem de utilização:

```
usage: ./server <int: port>
```

Se o nome do binário passado por argumento for "./server", obviamente.

Parâmetros

bin_name	Nome do binário que foi executado

Índice Remissivo