Networked Hangman C

Version 1.0.0

4/11/2020 7:32:00 PM

Table of Contents

File Index

File List

Here is a list of all files with brief descriptions:

File Documentation

D:/workspaces/college/NG_Asgnmt_01/hangman/CMakeLists.txt File Reference

D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/datatypes_all.h File Reference

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

Macros

- #define MAX_LEN 80

Macro Definition Documentation

#define MAX_LEN 80

Author: Ciaran Bent [K00221230] Date: 2020/04/02

This is the Header file for the imports and definitions required by ALL versions of the Networked Hangman game.

Definition at line 19 of file datatypes_all.h.

datatypes_all.h

```
00001
00010 #ifndef HANGMAN_DATATYPES_ALL_H
00011 #define HANGMAN_DATATYPES_ALL_H
00012
00013 #include <netdb.h>
00014 #include <stdio.h>
00015 #include <stdlib.h>
00016 #include <string.h>
00017 #include <unistd.h>
00018
00019 #define MAX_LEN 80
00020
00021 #endif //HANGMAN_DATATYPES_ALL_H
```

D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/datatypes_client.h File Reference

```
#include <ctype.h>
```

datatypes_client.h

```
00001
00010 #ifndef HANGMAN_DATATYPES_CLIENT_H
00011 #define HANGMAN_DATATYPES_CLIENT_H
00012
00013 #include <ctype.h>
00014
00015 #endif //HANGMAN_DATATYPES_CLIENT_H
```

D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/datatypes_server.h File Reference

```
#include <stdbool.h>
#include <time.h>
#include "../rsc/words"
```

Macros

- #define NUM_OF_WORDS   (sizeof(word) / sizeof(word[0]))
- #define MAX_PLAYERS   2
- #define MAX_LIVES   10
- #define MAX_TOTAL_LEN   120

Enumerations

- enum Game_State { IN_PROGRESS, WON, LOST }

Variables

- char * word []

Macro Definition Documentation

#define MAX_LIVES   10

Definition at line 26 of file datatypes_server.h.

#define MAX_PLAYERS   2

Definition at line 25 of file datatypes_server.h.

#define MAX_TOTAL_LEN   120

Definition at line 27 of file datatypes_server.h.

#define NUM_OF_WORDS   (sizeof(word) / sizeof(word[0]))

Definition at line 24 of file datatypes_server.h.

Enumeration Type Documentation

enum Game_State

Enumerator:

IN_PROGRESS	
WON	
LOST	

Definition at line 20 of file datatypes_server.h.

Variable Documentation

char* word[]

```
Initial value:= {
}
```

Author: Ciaran Bent [K00221230] Date: 2020/04/02

This is the Header file for the imports and definitions required by ALL versions of the Server side of the Networked Hangman game.

Definition at line 16 of file datatypes_server.h.

datatypes_server.h

```
00001
00010 #ifndef HANGMAN_DATATYPES_SERVER_H
00011 #define HANGMAN_DATATYPES_SERVER_H
00012
```

```
#include <stdbool.h>
#include <time.h>

char* word[] = {
#include "../rsc/words"
};

enum Game_State {
    IN_PROGRESS, WON, LOST
};

#define NUM_OF_WORDS (sizeof(word) / sizeof(word[0]))
#define MAX_PLAYERS 2
#define MAX_LIVES 10
#define MAX_TOTAL_LEN 120

#endif //HANGMAN_DATATYPES_SERVER_H
```

D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/datatypes_udp.h File Reference

Macros

- #define HANGMAN_UDP_PORT 1337
- #define ID_LEN 3

Macro Definition Documentation

#define HANGMAN_UDP_PORT 1337

Author: Ciaran Bent [K00221230] Date: 2020/04/02

This is the Header file for the imports and definitions required by the UDP versions of the Networked Hangman game.

Definition at line 13 of file datatypes_udp.h.

#define ID_LEN 3

Definition at line 14 of file datatypes_udp.h.

datatypes_udp.h

```
00001
00010 #ifndef HANGMAN_DATATYPES_UDP_H
00011 #define HANGMAN_DATATYPES_UDP_H
00012
00013 #define HANGMAN_UDP_PORT 1337
00014 #define ID_LEN 3
00015
00016 #endif //HANGMAN_DATATYPES_UDP_H
```

D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/hangclient.h File Reference

`#include <stdio.h>`

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

Macros

- #define LINESIZE   80
- #define HANGMAN_TCP_GENERIC_PORT   1066

Functions

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

Macro Definition Documentation

#define HANGMAN_TCP_GENERIC_PORT   1066

Definition at line 14 of file hangclient.h.

#define LINESIZE   80

Definition at line 13 of file hangclient.h.

Function Documentation

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

main() function is the main runtime function of the UDP Client. It connects to a Server, and begins processes to commence a game of Networked Hangman.

Parameters

argc	- The count of cmdline arguments
argv	- The cmdline arguments, the address of the remote Server

Returns

- Exit Status

main() function is the main runtime function of the UDP Server. It gathers several Clients and launches the Server for the Networked Hangman game.

Parameters

argc	- The count of cmdline arguments
argv	- The cmdline arguments, the number of Clients to connect in this case.

Returns

- Exit Status

Definition at line 5 of file hangclient.c.

hangclient.h

```
#ifndef HANGCLIENT_H
#define HANGCLIENT_H

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define LINESIZE 80
#define HANGMAN_TCP_GENERIC_PORT 1066 // Define this in the Specific Header

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

#endif // HANGCLIENT_H
```

D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/hangclient_udp.h File Reference

```
#include "datatypes_all.h"
#include "datatypes_client.h"
#include "datatypes_udp.h"
```

Macros

- #define GUESS_LEN 2

Functions

- void play_hangman (int sock, struct sockaddr *serv_addr, socklen_t serv_len, char cli_id[ID_LEN])
- void test_connection (int sock, struct sockaddr *serv_addr, socklen_t serv_len)
- void setup_connection (int sock, struct sockaddr *serv_addr, socklen_t serv_len)

Macro Definition Documentation

#define GUESS_LEN 2

Author: Ciaran Bent [K00221230] Date: 2020/03/09

This is the Header file for the Client side of the UDP version of the Networked Hangman game.

Definition at line 17 of file hangclient_udp.h.

Function Documentation

void play_hangman (int *sock*, struct sockaddr * *serv_addr*, socklen_t *serv_len*, char *cli_id*[ID_LEN])

Author: Ciaran Bent [K00221230] Date: 2020/03/09

This is the Source file for the Client side of the UDP version of the Networked Hangman game. play_hangman() function is used to handle playing the Networked

Parameters

sock	- The Client socket to Send/Receive to/from
serv_addr	- The address of the remote Server
serv_len	- The length of the Server Address Structure
cli_id	- The ID Tag for this Client

Definition at line 20 of file hangclient_udp.c.

void setup_connection (int *sock*, struct sockaddr * *serv_addr*, socklen_t *serv_len*)

setup_connection() function is used to receive a number from the Server to use when sending any data so that the Server knows which Client this is.

Parameters

sock	- The Client socket to Send/Receive to/from
serv_addr	- The address of the remote Server
serv_len	- The length of the Server Address Structure

Definition at line 180 of file hangclient_udp.c.

void test_connection (int *sock*, struct sockaddr * *cli_addr*, socklen_t *cli_len*)

test_connection() function is used to verify that a connection can be made to a Server.

Parameters

sock	- The Client socket to Send/Receive to/from
serv_addr	- The address of the remote Server
serv_len	- The length of the Server Address Structure

test_connection() function is used to verify that a connection can be made to a Client.

Parameters

sock	- The Server socket to Send/Receive to/from
cli_addr	- The address of the remote Client
cli_len	- The length of the Client Address Structure

Definition at line 134 of file hangclient_udp.c.

hangclient_udp.h

```
#ifndef HANGMAN_HANGSERVER_UDP_H
#define HANGMAN_HANGSERVER_UDP_H

#include "datatypes_all.h"
#include "datatypes_client.h"
#include "datatypes_udp.h"

#define GUESS_LEN 2

void play_hangman(int sock, struct sockaddr* serv_addr, socklen_t serv_len, char
cli_id[ID_LEN]);

void test_connection(int sock, struct sockaddr* serv_addr, socklen_t serv_len);

void setup_connection(int sock, struct sockaddr* serv_addr, socklen_t serv_len);

#endif //HANGMAN_HANGSERVER_UDP_H
```

D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/hangserver.h File Reference

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <syslog.h>
#include <signal.h>
#include <errno.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include "../rsc/words"
```

Macros

- #define NUM_OF_WORDS   (sizeof(word) / sizeof(word[0]))
- #define MAXLEN   80
- #define HANGMAN_TCP_GENERIC_PORT   1066

Functions

- time_t time ()

Variables

- char * word []

Macro Definition Documentation

#define HANGMAN_TCP_GENERIC_PORT   1066

Definition at line 25 of file hangserver.h.

#define MAXLEN   80

Definition at line 24 of file hangserver.h.

#define NUM_OF_WORDS   (sizeof(word) / sizeof(word[0]))

Definition at line 23 of file hangserver.h.

Function Documentation

time_t time ()

Variable Documentation

char* word[]

```
Initial value:= {
}
```

Definition at line 19 of file hangserver.h.

hangserver.h

```
00001 #ifndef HANGSERVER_H
00002 #define HANGSERVER_H
00003
00004 #include <sys/types.h>
00005 #include <sys/socket.h>
00006 #include <netinet/in.h>
00007 #include <stdio.h>
00008 #include <syslog.h>
00009 #include <signal.h>
00010 #include <errno.h>
00011 #include <unistd.h>
00012 #include <arpa/inet.h>
00013 #include <stdlib.h>
00014 #include <string.h>
00015 #include <sys/wait.h>
00016
00017 extern time_t time();
00018
00019 char* word[] = {
00020 #include "../rsc/words"
00021 };
00022
00023 #define NUM_OF_WORDS (sizeof(word) / sizeof(word[0]))
00024 #define MAXLEN 80 // Maximum size in the word of any String
00025 #define HANGMAN_TCP_GENERIC_PORT 1066 // Define this in the Specific Header
00026
00027 #endif // HANGSERVER_H
```

D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/hangserver_udp.h File Reference

```
#include "datatypes_all.h"
#include "datatypes_server.h"
#include "datatypes_udp.h"
```

Functions

- void play_hangman (int sock, struct sockaddr_in *cli_addrs, socklen_t cli_len, const int *connected_clients)
- void test_connection (int sock, struct sockaddr *cli_addr, socklen_t cli_len)
- void setup_connections (int sock, struct sockaddr *cli_addr, socklen_t cli_len, const int *cli_count)

Function Documentation

void play_hangman (int *sock*, struct sockaddr_in * *cli_addrs*, socklen_t *cli_len*, const int * *connected_clients*)

Author: Ciaran Bent [K00221230] Date: 2020/03/09

This is the Header file for the Server side of the UDP version of the Networked Hangman game.

Author: Ciaran Bent [K00221230] Date: 2020/03/09

This is the Source file for the Server side of the UDP version of the Networked Hangman game. play_hangman() function is used to handle serving the Networked Hangman game.

Parameters

sock	- The Server socket to Send/Receive to/from
cli_addrs	- The address(es) of the remote Client(s)
cli_len	- The length of the Client Address Structure
connected_clients	- The number of connected Clients

Definition at line 21 of file hangserver_udp.c.

void setup_connections (int *sock*, struct sockaddr * *cli_addr*, socklen_t *cli_len*, const int * *cli_count*)

setup_connections() function is used to add Clients to the game.

Parameters

sock	- The Client socket to Send/Receive to/from
cli_addr	- The address of the remote Client
cli_len	- The length of the Client Address Structure
cli_count	- The numerical identifier for this Client

Definition at line 255 of file hangserver_udp.c.

void test_connection (int *sock*, struct sockaddr * *cli_addr*, socklen_t *cli_len*)

test_connection() function is used to verify that a connection can be made to a Server.

Parameters

sock	- The Client socket to Send/Receive to/from
serv_addr	- The address of the remote Server
serv_len	- The length of the Server Address Structure

test_connection() function is used to verify that a connection can be made to a Client.

Parameters

sock	- The Server socket to Send/Receive to/from
cli_addr	- The address of the remote Client
cli_len	- The length of the Client Address Structure

Definition at line 134 of file hangclient_udp.c.

hangserver_udp.h

```

#ifndef HANGMAN_HANGSERVER_UDP_H
#define HANGMAN_HANGSERVER_UDP_H

#include "datatypes_all.h"
#include "datatypes_server.h"
#include "datatypes_udp.h"

void play_hangman(int sock, struct sockaddr_in* cli_addrs, socklen_t cli_len, const int* connected_clients);

void test_connection(int sock, struct sockaddr* cli_addr, socklen_t cli_len);

void setup_connections(int sock, struct sockaddr* cli_addr, socklen_t cli_len, const int* cli_count);

#endif //HANGMAN_HANGSERVER_UDP_H
```

D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangclient.c File Reference

```
#include "../hdr/hangclient.h"
```

Functions

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

Function Documentation

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

Definition at line 5 of file hangclient.c.

hangclient.c

```
00001 /* Hangclient.c - Client for hangman server.  */
00002 
00003 #include "../hdr/hangclient.h"
00004 
00005 int main(int argc, char* argv[]) {
00006     struct sockaddr_in server; // Server's address assembled here
00007     struct hostent* host_info;
00008     int  sock, count;
00009     char i_line[LINESIZE];
00010     char o_line[LINESIZE];
00011     char* server_name;
00012 
00013     // Get server name from the command line.  If none, use 'localhost'
00014     server_name = (argc == 1) ? argv[1] : "localhost";
00015 
00016     // Create the socket
00017     sock = socket(AF_INET, SOCK_STREAM, 0);
00018     if (sock < 0) {
00019         perror("Creating stream socket");
00020         exit(1);
00021     }
00022 
00023     host_info = gethostbyname(server_name);
00024     if (host_info == NULL) {
00025         fprintf(stderr, "%s: unknown host:%s \n", argv[0], server_name);
00026         exit(2);
00027     }
00028 
00029     // Set up the server's socket address, then connect
00030 
00031     server.sin_family = host_info->h_addrtype;
00032     memcpy((char*) &server.sin_addr, host_info->h_addr, host_info->h_length);
00033     server.sin_port = htons(HANGMAN_TCP_GENERIC_PORT);
00034 
00035     if (connect(sock, (struct sockaddr*) &server, sizeof server) < 0) {
00036         perror("connecting to server");
00037         exit(3);
00038     }
00039 
00040     /* We are connected to the server.
00041         Take a line from the server and show it
00042         Take a line and send the user input to the server.
00043         Repeat until the server terminates the connection.
00044     */
```

```

    printf("Connected to server %s \n", server_name);
    while ((count = read(sock, i_line, LINESIZE)) > 0) {
        write(1, i_line, count);
        count = read(0, o_line, LINESIZE);//0 = STDIN
        write(sock, o_line, count);
    }
}
```

D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangclient_udp.c File Reference

`#include "../hdr/hangclient_udp.h"`

Functions

- void play_hangman (int sock, struct sockaddr *serv_addr, socklen_t serv_len, char cli_id[ID_LEN])
- void test_connection (int sock, struct sockaddr *serv_addr, socklen_t serv_len)
- void setup_connection (int sock, struct sockaddr *serv_addr, socklen_t serv_len)
- int main (int argc, char *argv[])

Function Documentation

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

main() function is the main runtime function of the UDP Client. It connects to a Server, and begins processes to commence a game of Networked Hangman.

Parameters

argc	- The count of cmdline arguments
argv	- The cmdline arguments, the address of the remote Server

Returns

- Exit Status

Definition at line 229 of file hangclient_udp.c.

void play_hangman (int *sock*, struct sockaddr * *serv_addr*, socklen_t *serv_len*, char *cli_id*[ID_LEN])

Author: Ciaran Bent [K00221230] Date: 2020/03/09

This is the Source file for the Client side of the UDP version of the Networked Hangman game. play_hangman() function is used to handle playing the Networked

Parameters

sock	- The Client socket to Send/Receive to/from
serv_addr	- The address of the remote Server
serv_len	- The length of the Server Address Structure
cli_id	- The ID Tag for this Client

Definition at line 20 of file hangclient_udp.c.

void setup_connection (int *sock*, struct sockaddr * *serv_addr*, socklen_t *serv_len*)

setup_connection() function is used to receive a number from the Server to use when sending any data so that the Server knows which Client this is.

Parameters

sock	- The Client socket to Send/Receive to/from

serv_addr	- The address of the remote Server
serv_len	- The length of the Server Address Structure

Definition at line 180 of file hangclient_udp.c.

void test_connection (int *sock*, struct sockaddr * *serv_addr*, socklen_t *serv_len*)

test_connection() function is used to verify that a connection can be made to a Server.

Parameters

sock	- The Client socket to Send/Receive to/from
serv_addr	- The address of the remote Server
serv_len	- The length of the Server Address Structure

Definition at line 134 of file hangclient_udp.c.

hangclient_udp.c

```

#include "../hdr/hangclient_udp.h"


void play_hangman(int sock, struct sockaddr* serv_addr, socklen_t serv_len, char cli_id[ID_LEN]) {
    ssize_t count;
    int round_local;
    char hostname[MAX_LEN];
    char i_line[MAX_LEN];
    char o_guess[GUESS_LEN];
    char temp_guess[GUESS_LEN];

    // Zero out all data before starting
    memset(&count, '\0', sizeof(count));
    memset(&round_local, '\0', sizeof(round_local));
    memset(&hostname, '\0', sizeof(hostname));
    memset(&i_line, '\0', sizeof(i_line));
    memset(&o_guess, '\0', sizeof(o_guess));
    memset(&temp_guess, '\0', sizeof(temp_guess));

    // Get the Human Readable name of this host
    gethostname(hostname, MAX_LEN);

    fprintf(stdout, "Playing Hangman as Client #%s on [%s]\n", cli_id, hostname);

    // Receive the Hostname of the Server
    count = recvfrom(sock, i_line, MAX_LEN, 0, serv_addr, &serv_len);

    // Check the received data for errors
    if (count < 0) {
        perror("Receiving from Server Socket Failed\n");
        exit(4); // Error Condition 04
    }

    fprintf(stdout, "Connected to Server: %s\n", i_line);

    // Receive the initial game state
    memset(&i_line, '\0', sizeof(i_line));
    count = recvfrom(sock, i_line, MAX_LEN, 0, serv_addr, &serv_len);

    // Check the received data for errors
    if (count < 0) {
        perror("Receiving from Server Socket Failed\n");
        exit(4); // Error Condition 04
    }

    fprintf(stdout, "\nInitial Game State:%s", i_line);

    // Play the game
    do {
        do {
            // Receive the current turn from the Server
```

```c
            memset(&i_line, '\0', sizeof(i_line));
            count = recvfrom(sock, i_line, MAX_LEN, 0, serv_addr, &serv_len);

            // Check the received data for errors
            if (count < 0) {
                perror("Receiving from Server Socket Failed\n");
                exit(4); // Error Condition 04
            }

            if (strcmp(i_line, "#GAMEOVER\0") == 0) { exit(0); }
        } while (strcmp(i_line, cli_id) != 0);

        fprintf(stdout, "\n---\nRound: %d", ++round_local);

        // Receive the current game state
        memset(&i_line, '\0', sizeof(i_line));
        count = recvfrom(sock, i_line, MAX_LEN, 0, serv_addr, &serv_len);

        // Check the received data for errors
        if (count < 0) {
            perror("Receiving from Server Socket Failed\n");
            exit(4); // Error Condition 04
        }

        fprintf(stdout, "\nGame State:%s", i_line);
        memset(&i_line, '\0', sizeof(i_line));

        // Securely retrieve data from the User
        memset(&temp_guess, '\0', sizeof(temp_guess));

        // Don't allow bad characters
        while (!isalpha((unsigned) temp_guess[0])) { // NOLINT (Bug in CLion)
            fprintf(stdout, "\nGuess a LETTER [a-z]\n>>");
            temp_guess[0] = (char) fgetc(stdin);
        }

        // Convert the letter to lowercase
        temp_guess[0] = tolower(temp_guess[0]);

        // Terminate the String and prepare it for sending
        strncat(temp_guess, "\0", GUESS_LEN);
        strncpy(o_guess, temp_guess, GUESS_LEN);

        // Show the User what they typed
        fprintf(stdout, "Guess was: %s", o_guess);

        // Send the data to the Server
        count = sendto(sock, o_guess, GUESS_LEN, 0, serv_addr, serv_len);

        // Check the sent data for errors
        if (count < 0) {
            perror("Sending to Server Socket Failed\n");
            exit(3); // Error Condition 03
        }

    } while (strcmp(i_line, "#GAMEOVER\0") != 0);
}


void test_connection(int sock, struct sockaddr* serv_addr, socklen_t serv_len) {
    ssize_t count;
    char i_line[MAX_LEN + 1];
    char o_line[MAX_LEN];

    // Zero the data out
    memset(&i_line, '\0', sizeof(i_line));
    memset(&o_line, '\0', sizeof(o_line));

    fprintf(stdout, "Testing Connection\n");

    fprintf(stdout, ">>");
    while (fgets(o_line, MAX_LEN, stdin) != NULL) {
        fprintf(stdout, "Sending: %s\n", o_line);

        // Send the data to the Server
        sendto(sock, o_line, strlen(o_line), 0, serv_addr, serv_len);

        // Receive a reply from the Server
        count = recvfrom(sock, i_line, MAX_LEN, 0, NULL, NULL);

        // Check the received data for errors
        if (count < 0) {
            perror("Receiving from Server Socket Failed\n");
            exit(3); // Error Condition 03
        }

        i_line[count] = 0;
        fprintf(stdout, "Received: %s\n---\n", i_line);

        memset(&i_line, '\0', sizeof(i_line));
        memset(&o_line, '\0', sizeof(o_line));

        fprintf(stdout, "\n>>");
    }
}


void setup_connection(int sock, struct sockaddr* serv_addr, socklen_t serv_len) {
    ssize_t count;
    char id_request[ID_LEN + 1];
    char id_response[ID_LEN];

    // Zero the data out
    memset(&id_request, '\0', sizeof(id_request));
    memset(&id_response, '\0', sizeof(id_response));

    // Assign request character
    strncpy(id_request, "-1", ID_LEN);

    fprintf(stdout, "Sending: %s\n", id_request);

    // Send the data to the Server
    count = sendto(sock, id_request, ID_LEN, 0, serv_addr, serv_len);

    // Check the sent data for errors
    if (count < 0) {
        perror("Sending to Server Socket Failed\n");
        exit(3); // Error Condition 03
    }

    // Receive a reply from the Server
    count = recvfrom(sock, id_response, ID_LEN, 0, serv_addr, &serv_len);

    // Check the received data for errors
    if (count < 0) {
        perror("Receiving from Server Socket Failed\n");
        exit(4); // Error Condition 04
    }

    fprintf(stdout, "Received: %s\n---\n", id_response);

    memset(&id_request, '\0', sizeof(id_request));

    play_hangman(sock, (struct sockaddr*) &serv_addr, sizeof(*serv_addr),
id_response);
}


int main(int argc, char* argv[]) {
    int udp_sock;
    struct sockaddr_in serv_addr;
    struct hostent* host_info;
    char* server_name;

    // Set the Server address to the cmdline option, or LOCALHOST
    server_name = (argc == 2) ? argv[1] : "localhost";

    // Convert the IP Address to a Human-Readable format
    host_info = gethostbyname(server_name);
    if (host_info == NULL) {
        perror("Unknown Host\n");
        exit(1);
    }

    // Build up the Server Address Structure
    memset(&serv_addr, '\0', sizeof(serv_addr));
    serv_addr.sin_family = (sa_family_t) host_info->h_addrtype;
    memcpy((char*) &serv_addr.sin_addr, host_info->h_addr, (size_t)
host_info->h_length);
    serv_addr.sin_port = htons(HANGMAN_UDP_PORT);

    // Create the local Socket
    udp_sock = socket(AF_INET, SOCK_DGRAM, 0); //0 or IPPROTO_UDP

    // Error check The Socket
    if (udp_sock < 0) {
        perror("Creating Datagram Socket Failed\n");
        exit(2);
    }

    fprintf(stdout, "UDP Client Socket Created\n");

    //test_connection(udp_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr));
    setup_connection(udp_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr));

    // Close the Socket, and exit the program
    close(udp_sock);
    return (0);
}
```

D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangserver.c File Reference

`#include "../hdr/hangserver.h"`

Functions

- void play_hangman (int in, int out)
- int main ()

Variables

- int maxlives = 12

Function Documentation

int main ()

Definition at line 66 of file hangserver.c.

void play_hangman (int *in*, int *out*)

Definition at line 10 of file hangserver.c.

Variable Documentation

int maxlives = 12

Definition at line 6 of file hangserver.c.

hangserver.c

```c
/* Network server for hangman game */
/* File: hangserver.c */

#include "../hdr/hangserver.h"

int maxlives = 12;

/* ---------------- Play_hangman () ---------------------*/

void play_hangman(int in, int out) {
    char* whole_word, part_word[MAXLEN],
                      guess[MAXLEN], outbuf[MAXLEN];

    int  lives      = maxlives;
    int  game_state = 'I';//I = Incomplete
    int  i, good_guess, word_length;
    char hostname[MAXLEN];

    gethostname(hostname, MAXLEN);
    sprintf(outbuf, "Playing hangman on host %s: \n\n", hostname);
    write(out, outbuf, strlen(outbuf));

    // Pick a word at random from the list
    whole_word  = word[rand() % NUM_OF_WORDS];
    word_length = strlen(whole_word);
    syslog(LOG_USER | LOG_INFO, "Server chose hangman word %s", whole_word);

    // No letters are guessed Initially
    for (i = 0; i < word_length; i++) {
        part_word[i] = '-';
    }

    part_word[i] = '\0';

    sprintf(outbuf, "%s %d \n", part_word, lives);
    write(out, outbuf, strlen(outbuf));

    while (game_state == 'I')
        // Get a letter from player guess
    {
        while (read(in, guess, MAXLEN) < 0) {
            if (errno != EINTR) {
                exit(4);
            }
            printf("re-read the start in \n");
        } // Re-start read () if interrupted by signal
        good_guess = 0;
        for (i     = 0; i < word_length; i++) {
            if (guess[0] == whole_word[i]) {
                good_guess = 1;
                part_word[i] = whole_word[i];
            }
        }
        if (!good_guess) { lives--; }
        if (strcmp(whole_word, part_word) == 0) {
            game_state = 'W'; // W ==> User Won
        } else if (lives == 0) {
            game_state = 'L'; // L ==> User Lost
            strcpy(part_word, whole_word); // User Show the word
        }
        sprintf(outbuf, "%s %d \n", part_word, lives);
        write(out, outbuf, strlen(outbuf));
    }
}

int main() {
    int                 sock, fd, client_len;
    struct sockaddr_in server, client;

    srand((int) time((long*) 0)); // randomize the seed

    sock = socket(AF_INET, SOCK_STREAM, 0);//0 or IPPROTO_TCP
    if (sock < 0) { // This error checking is the code Stevens wraps in his Socket Function etc
        perror("creating stream socket");
        exit(1);
    }

    server.sin_family      = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port        = htons(HANGMAN_TCP_GENERIC_PORT);

    if (bind(sock, (struct sockaddr*) &server, sizeof(server)) < 0) {
        perror("binding socket");
        exit(2);
    }

    listen(sock, 5);

    while (1) {
        client_len = sizeof(client);
        if ((fd    = accept(sock, (struct sockaddr*) &client, &client_len)) < 0) {
            perror("accepting connection");
            exit(3);
        }
        play_hangman(fd, fd);
        close(fd);
    }
}
```

D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangserver_Fork.c File Reference

`#include "../hdr/hangserver.h"`

Macros

- #define NUM_OF_WORDS   (sizeof(word) / sizeof(word[0]))
- #define MAXLEN   80
- #define HANGMAN_TCP_PORT   1066

Functions

- time_t time ()
- void testGameNoZombie (int in, int out)
- void testGameZombie (int in, int out)
- void play_hangman (int in, int out)
- int main ()

Variables

- int maxlives = 12

Macro Definition Documentation

#define HANGMAN_TCP_PORT   1066

Definition at line 11 of file hangserver_Fork.c.

#define MAXLEN   80

Definition at line 10 of file hangserver_Fork.c.

#define NUM_OF_WORDS   (sizeof(word) / sizeof(word[0]))

Definition at line 9 of file hangserver_Fork.c.

Function Documentation

int main ()

Definition at line 88 of file hangserver_Fork.c.

void play_hangman (int *in*, int *out*)

Definition at line 32 of file hangserver_Fork.c.

void testGameNoZombie (int *in*, int *out*)

Definition at line 15 of file hangserver_Fork.c.

void testGameZombie (int *in*, int *out*)

Definition at line 24 of file hangserver_Fork.c.

time_t time ()

Variable Documentation

int maxlives = 12

Definition at line 12 of file hangserver_Fork.c.

hangserver_Fork.c

```
00001 /* Network server for hangman game */
00002 /* File: hangserver.c */
00003
00004 #include "../hdr/hangserver.h"
00005
00006 extern time_t time();
00007
00008
00009 # define NUM_OF_WORDS (sizeof(word) / sizeof(word[0]))
00010 # define MAXLEN 80 // Maximum size in the word of any String
00011 # define HANGMAN_TCP_PORT 1066
00012 int maxlives = 12;
00013
00014
00015 void testGameNoZombie(int in,int out){
00016
00017     printf("\nPlaying Test Connection\n\n");
00018     while(1){

    }
    exit(0);
}

void testGameZombie(int in,int out){

    printf("\nPlaying Test Connection\n\n");

    exit(0);
}


void play_hangman(int in, int out) {
    char* whole_word, part_word[MAXLEN],
                      guess[MAXLEN], outbuf[MAXLEN];

    int  lives      = maxlives;
    int  game_state = 'I';//I = Incomplete
    int  i, good_guess, word_length;
    char hostname[MAXLEN];

    gethostname(hostname, MAXLEN);
    sprintf(outbuf, "Playing hangman on host %s: \n\n", hostname);
    write(out, outbuf, strlen(outbuf));

    // Pick a word at random from the list
    whole_word  = word[rand() % NUM_OF_WORDS];
    word_length = strlen(whole_word);
    syslog(LOG_USER | LOG_INFO, "Server chose hangman word %s", whole_word);

    // No letters are guessed Initially
    for (i = 0; i < word_length; i++) {
        part_word[i] = '-';
    }

    part_word[i] = '\0';

    sprintf(outbuf, "%s %d \n", part_word, lives);
    write(out, outbuf, strlen(outbuf));

    while (game_state == 'I')
        // Get a letter from player guess
    {
        while (read(in, guess, MAXLEN) < 0) {
            if (errno != EINTR) {
                exit(4);
            }
            printf("re-read the start in \n");
        } // Re-start read () if interrupted by signal
        good_guess = 0;
        for (i     = 0; i < word_length; i++) {
            if (guess[0] == whole_word[i]) {
                good_guess = 1;
                part_word[i] = whole_word[i];
            }
        }
        if (!good_guess) { lives--; }
        if (strcmp(whole_word, part_word) == 0) {
            game_state = 'W'; // W ==> User Won
        } else if (lives == 0) {
            game_state = 'L'; // L ==> User Lost
            strcpy(part_word, whole_word); // User Show the word
        }
        sprintf(outbuf, "%s %d \n", part_word, lives);
        write(out, outbuf, strlen(outbuf));
    }
}

int main() {
    int sock, fd, client_len, childProcCount;
    struct sockaddr_in server, client;

    srand((int) time((long*) 0)); // randomize the seed

    sock = socket(AF_INET, SOCK_STREAM, 0);//0 or IPPROTO_TCP
    if (sock < 0) { // This error checking is the code Stevens wraps in his Socket
Function etc
        perror("creating stream socket");
        exit(1);
    }

    server.sin_family      = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port        = htons(HANGMAN_TCP_PORT);

    if (bind(sock, (struct sockaddr*) &server, sizeof(server)) < 0) {
        perror("binding socket");
        exit(2);
    }

    listen(sock, 5);
    childProcCount = 0;
    while (1) {
        client_len = sizeof(client);
        //Accept Connection "FORK HERE!!"
        if ((fd    = accept(sock, (struct sockaddr*) &client, &client_len)) < 0) {
            perror("accepting connection");
            exit(3);
        }
        pid_t pid = fork();

        if(pid < 0){
            perror("Fork() Failed");
        }
        if(pid==0){ //TODO Gamify with Hangman
            perror("Fork Connection Accepted");
            play_hangman(fd,fd);
            exit(0);
        }

        //Increment Child Tracker
        printf("",pid);
        close(fd);
        childProcCount++;
        //clean up zombies
        while(childProcCount){
            pid = waitpid((pid_t) -1 ,NULL,WNOHANG);
            if(pid < 0){
                perror("waitpid() failed No Zombie Found");
            }
            else if(pid ==0){
                perror("No Zombies, break");
                break;
            }
            else{
                perror("Zombie terminated");
                childProcCount--;
            }
        }

        //play_hangman(fd, fd);
        //close(fd);
    }
}

```

D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangserver_udp.c File Reference

`#include "../hdr/hangserver_udp.h"`

Functions

- void play_hangman (int sock, struct sockaddr_in *cli_addrs, socklen_t cli_len, const int *connected_clients)

- void test_connection (int sock, struct sockaddr *cli_addr, socklen_t cli_len)
- void setup_connections (int sock, struct sockaddr *cli_addr, socklen_t cli_len, const int *cli_count)
- int main (int argc, char *argv[])

Function Documentation

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

main() function is the main runtime function of the UDP Server. It gathers several Clients and launches the Server for the Networked Hangman game.

Parameters

argc	- The count of cmdline arguments
argv	- The cmdline arguments, the number of Clients to connect in this case.

Returns

- Exit Status

Definition at line 315 of file hangserver_udp.c.

void play_hangman (int *sock*, struct sockaddr_in * *cli_addrs*, socklen_t *cli_len*, const int * *connected_clients*)

Author: Ciaran Bent [K00221230] Date: 2020/03/09

This is the Source file for the Server side of the UDP version of the Networked Hangman game. play_hangman() function is used to handle serving the Networked Hangman game.

Parameters

sock	- The Server socket to Send/Receive to/from
cli_addrs	- The address(es) of the remote Client(s)
cli_len	- The length of the Client Address Structure
connected_clients	- The number of connected Clients

Definition at line 21 of file hangserver_udp.c.

void setup_connections (int *sock*, struct sockaddr * *cli_addr*, socklen_t *cli_len*, const int * *cli_count*)

setup_connections() function is used to add Clients to the game.

Parameters

sock	- The Client socket to Send/Receive to/from
cli_addr	- The address of the remote Client
cli_len	- The length of the Client Address Structure
cli_count	- The numerical identifier for this Client

Definition at line 255 of file hangserver_udp.c.

void test_connection (int *sock*, struct sockaddr * *cli_addr*, socklen_t *cli_len*)

test_connection() function is used to verify that a connection can be made to a Client.

Parameters

sock	- The Server socket to Send/Receive to/from
cli_addr	- The address of the remote Client
cli_len	- The length of the Client Address Structure

Definition at line 210 of file hangserver_udp.c.

hangserver_udp.c

```
#include "../hdr/hangserver_udp.h"


void play_hangman(int sock, struct sockaddr_in* cli_addrs, socklen_t cli_len, const int* connected_clients) {
    fprintf(stdout, "\n---\nPlaying Hangman\n");

    // Set up the game
    ssize_t count;
    char* whole_word,
            part_word[MAX_LEN],
            outbuf[MAX_TOTAL_LEN];
    bool good_guess;
    size_t word_length;
    char hostname[MAX_LEN];
    char guess[2];
    int lives;
    int clients_in_play = *connected_clients;
    enum Game_State game_state = IN_PROGRESS;

    // Zero out all data before starting
    memset(&count, '\0', sizeof(count));
    memset(&whole_word, '\0', sizeof(whole_word));
    memset(&part_word, '\0', sizeof(part_word));
    memset(&outbuf, '\0', sizeof(outbuf));
    memset(&guess, '\0', sizeof(guess));
    memset(&lives, '\0', sizeof(lives));
    memset(&good_guess, '\0', sizeof(good_guess));
    memset(&word_length, '\0', sizeof(word_length));
    memset(&hostname, '\0', sizeof(hostname));

    fprintf(stdout, "\nThere are %d Clients in play\n", clients_in_play);

    // Pick a word at random from the list
    whole_word = word[random() % NUM_OF_WORDS];
    word_length = strlen(whole_word);
    lives = MAX_LIVES;
    fprintf(stdout, "\nServer chose hangman word %s", whole_word);

    // Ensure no letters are guessed Initially
    for (int j = 0; j < (int) word_length; j++) {
        part_word[j] = '-';
    }

    // Null-terminate the String
    part_word[word_length] = '\0';
    fprintf(stdout, "\nWordWhle: %s", whole_word);
    fprintf(stdout, "\nWordPart: %s", part_word);

    // Get the Human Readable name of this host
    gethostname(hostname, MAX_LEN);

    // Client currently being handled
    struct sockaddr* cli_addr = NULL;

    // Introduce each Client
    for (int i = 0; i < clients_in_play; i++) {
        // Set the current Client
        cli_addr = (struct sockaddr*) &cli_addrs[i];

        memset(&outbuf, '\0', sizeof(outbuf));
        sprintf(outbuf, "%s", hostname);
        sendto(sock, outbuf, strlen(outbuf), 0, cli_addr, cli_len);

        // Check that there were no errors with sending the data
        if (count < 0) {
            perror("Sending to Client Socket Failed\n");
            exit(4); // Error Condition 04
        }

        memset(&outbuf, '\0', sizeof(outbuf));
        snprintf(outbuf, MAX_TOTAL_LEN, "\n\tWord: %s\n\tLives: %hu", part_word, (unsigned short) lives);
        sendto(sock, outbuf, strlen(outbuf), 0, cli_addr, cli_len);

        // Check that there were no errors with sending the data
        if (count < 0) {
            perror("Sending to Client Socket Failed\n");
            exit(4); // Error Condition 04
        }
    }

    // Loop until the game is WON
    while (game_state == IN_PROGRESS) {
        good_guess = false;

        // Loop for each Client
        for (int i = 0; i < clients_in_play; i++) {
            fprintf(stdout, "\nServing Client %d", i);

            // Set the current Client
            cli_addr = (struct sockaddr*) &cli_addrs[i];

            // Inform the Client that it's their turn
            memset(&outbuf, '\0', sizeof(outbuf));
            sprintf(outbuf, "%d", (i + 1));
            sendto(sock, outbuf, strlen(outbuf), 0, cli_addr, cli_len);

            // Check that there were no errors with sending the data
            if (count < 0) {
                perror("Sending to Client Socket Failed\n");
                exit(4); // Error Condition 04
            }

            // Send the current state of the game to the Client
            memset(&outbuf, '\0', sizeof(outbuf));
            snprintf(outbuf, MAX_TOTAL_LEN, "\n\tWord: %s\n\tLives: %hu", part_word, (unsigned short) lives);
            sendto(sock, outbuf, strlen(outbuf), 0, cli_addr, cli_len);

            // Check that there were no errors with sending the data
            if (count < 0) {
                perror("Sending to Client Socket Failed\n");
                exit(4); // Error Condition 04
            }

            // Get a letter from player guess
            count = recvfrom(sock, guess, MAX_LEN, 0, cli_addr, &cli_len);

            // Check the received data for errors
            if (count < 0) {
                perror("Receiving from Client Socket Failed\n");
                exit(3); // Error Condition 03
            }

            // Evaluate the Client's guess
            for (int j = 0; j < (int) word_length; j++) {
                if (guess[0] == whole_word[j]) {
                    good_guess = true;
                    part_word[j] = whole_word[j];
                }
            }

            // If the guess was bad, subtract from the Lives counter
            if (!good_guess) {
                lives--;
            }

            // If the whole word has been guessed
            if (strcmp(whole_word, part_word) == 0) {
                game_state = WON; // User Won

                // Let the Client(s) know they WON
                memset(&outbuf, '\0', sizeof(outbuf));
                sprintf(outbuf, "%s", "#GAMEOVER");
                sendto(sock, outbuf, strlen(outbuf), 0, cli_addr, cli_len);

                // Check that there were no errors with sending the data
                if (count < 0) {
                    perror("Sending to Client Socket Failed\n");
                    exit(4); // Error Condition 04
                }

            } else if (lives == 0) {
                game_state = LOST; // User Lost

                // Let the Client(s) know they LOST
                memset(&outbuf, '\0', sizeof(outbuf));
                sprintf(outbuf, "%s", "#GAMEOVER");
                sendto(sock, outbuf, strlen(outbuf), 0, cli_addr, cli_len);

                // Check that there were no errors with sending the data
                if (count < 0) {
                    perror("Sending to Client Socket Failed\n");
                    exit(4); // Error Condition 04
                }

                strcpy(part_word, whole_word); // User Show the word
            }
        }
    }

    // Send ENDGAME message
    for (int i = 0; i < clients_in_play; i++) {
        // Set the current Client
        cli_addr = (struct sockaddr*) &cli_addrs[i];

        sendto(sock, outbuf, strlen(outbuf), 0, cli_addr, cli_len);

        // Check that there were no errors with sending the data
        if (count < 0) {
            perror("Sending to Client Socket Failed\n");
            exit(4); // Error Condition 04
        }
    }
}


void test_connection(int sock, struct sockaddr* cli_addr, socklen_t cli_len) {
    ssize_t count;
    char i_line[MAX_LEN];

    fprintf(stdout, "Testing Connection\n");

    do {
        memset(&i_line, '\0', sizeof(i_line));
        fprintf(stdout, "---\nAwaiting data on Socket %d...\n\n", sock);

        // Receive data from the Client Socket
        count = recvfrom(sock, i_line, MAX_LEN, 0, cli_addr, &cli_len);
        i_line[count] = '\0';

        // Check the received data for errors
        if (count < 0) {
            perror("Receiving from Client Socket Failed\n");
            exit(3); // Error Condition 03
        }

        // Print the received message to the screen
        fprintf(stdout, "Messg Received: %s", i_line);

        // Send data to the Client Socket
        count = sendto(sock, i_line, MAX_LEN, 0, cli_addr, cli_len);

        // Check that there were no errors with sending the data
        if (count < 0) {
            perror("Sending to Client Socket Failed\n");
            exit(4); // Error Condition 04
        }

        // Print confirmation of the send to the screen
        fprintf(stdout, "Messg Sent: %s", i_line);
    } while (strcmp(i_line, "#quit\0") != 0); // ToDo: Create stop condition that
actually works
```

```c
}


void setup_connections(int sock, struct sockaddr* cli_addr, socklen_t cli_len, const int* cli_count) {
    ssize_t count;
    int client_id;
    char id_request[ID_LEN];
    char id_response[ID_LEN];

    // Zero out data
    memset(&count, '\0', sizeof(count));
    memset(&client_id, '\0', sizeof(client_id));
    memset(&id_request, '\0', sizeof(id_request));
    memset(&id_response, '\0', sizeof(id_response));
    memset(cli_addr, '\0', sizeof(cli_len));

    fprintf(stdout, "\nSetting Up New Client\n");
    fprintf(stdout, "\nAwaiting request on Socket %d...\n", sock);

    // Receive data from the Client Socket
    count = recvfrom(sock, id_request, ID_LEN, 0, cli_addr, &cli_len);

    // Check the received data for errors
    if (count < 0) {
        perror("Receiving from Client Socket Failed\n");
        exit(3); // Error Condition 03
    }

    // Print the received message to the screen
    fprintf(stdout, "\nMessg Received: %s", id_request);
    client_id = (*cli_count) + 1;

    // Assign a Client ID to the Client if it is new
    if (strcmp(id_request, "-1\0") == 0) {
        fprintf(stdout, "\nClient is new.  Assigning ID: %d", client_id);
        sprintf(id_response, "%d", client_id); // Convert the int to char*
    } else {
        fprintf(stdout, "\nClient already assigned ID");
    }

    // Send data to the Client Socket
    count = sendto(sock, id_response, ID_LEN, 0, cli_addr, cli_len);

    // Check that there were no errors with sending the data
    if (count < 0) {
        perror("Sending to Client Socket Failed\n");
        exit(4); // Error Condition 04
    }

    // Print confirmation of the send to the screen
    fprintf(stdout, "\nMessg Sent: %s\n", id_response);
}


int main(int argc, char* argv[]) {
    // Set the `max_players` to the cmdline option, or MAX_PLAYERS
    int max_players = (argc == 2) ? (int) strtol(argv[1], NULL, 10) : MAX_PLAYERS;
    int udp_sock;
    struct sockaddr_in serv_addr;
    struct sockaddr_in cli_addrs[max_players];
    int connected_clients;

    // Zero out Server data
    memset(&serv_addr, '\0', sizeof(serv_addr));
    memset(&udp_sock, '\0', sizeof(udp_sock));
    memset(&connected_clients, '\0', sizeof(connected_clients));

    for (int i = 0; i < max_players; i++) {
        memset(&cli_addrs[i], '\0', sizeof(cli_addrs[i]));
    }

    // Seed the random number generator
    srandom((unsigned int) time(NULL));

    // Create the UDP Socket
    udp_sock = socket(AF_INET, SOCK_DGRAM, 0); //0 or IPPROTO_UDP

    // Error check The Socket
    if (udp_sock < 0) {
        perror("Creating Datagram Socket Failed\n");
        exit(1); // Error Condition 01
    }

    // Build the Server Address manually
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(HANGMAN_UDP_PORT);

    // Bind the Server socket to an address
    if (bind(udp_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) < 0) {
        perror("Binding Datagram Socket Failed\n");
        exit(2); // Error Condition 02
    }
    fprintf(stdout, "UDP Server Socket Created\n");

    // Test connections (DEBUG FUNCTION)
    //test_connection(udp_sock, (struct sockaddr*) &cli_addr, sizeof(cli_addr));

    // Accept Clients until all game slots are full
    connected_clients = 0;
    for (int i = 0; i < max_players; i++) {
        fprintf(stdout, "\n---\nCreating Client #%d", connected_clients);
        setup_connections(udp_sock, (struct sockaddr*) &cli_addrs[i], sizeof(cli_addrs[i]), &connected_clients);
        connected_clients++;
    }

    play_hangman(udp_sock, cli_addrs, sizeof(struct sockaddr), &connected_clients);

    // Close the Socket, and exit the program
    close(udp_sock);
    return (0);
}
```

Index