

Networked Hangman C

Version 1.0.0
4/11/2020 7:32:00 PM

Table of Contents

File Index.....	1
File Documentation	1
D:/workspaces/college/NG_Asgnmt_01/hangman/CMakeLists.txt	1
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/datatypes_all.h	1
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/datatypes_all.h	2
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/datatypes_client.h.....	2
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/datatypes_client.h.....	2
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/datatypes_server.h.....	2
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/datatypes_server.h.....	3
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/datatypes_udp.h	4
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/datatypes_udp.h	4
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/hangclient.h.....	4
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/hangclient.h.....	6
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/hangclient_udp.h.....	6
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/hangclient_udp.h.....	7
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/hangserver.h.....	8
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/hangserver.h.....	9
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/hangserver_udp.h.....	9
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/hangserver_udp.h.....	10
D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangclient.c.....	11
D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangclient.c.....	11
D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangclient_udp.c.....	12
D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangclient_udp.c.....	13
D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangserver.c.....	16
D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangserver.c.....	17
D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangserver_Fork.c	18
D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangserver_Fork.c	19
D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangserver_udp.c.....	21
D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangserver_udp.c.....	23
Index.....	27

File Index

File List

Here is a list of all files with brief descriptions:

D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/ datatypes_all.h	1
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/ datatypes_client.h	2
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/ datatypes_server.h	2
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/ datatypes_udp.h	4
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/ hangclient.h	4
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/ hangclient_udp.h	6
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/ hangserver.h	8
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/ hangserver_udp.h	9
D:/workspaces/college/NG_Asgnmt_01/hangman/src/ hangclient.c	11
D:/workspaces/college/NG_Asgnmt_01/hangman/src/ hangclient_udp.c	12
D:/workspaces/college/NG_Asgnmt_01/hangman/src/ hangserver.c	16
D:/workspaces/college/NG_Asgnmt_01/hangman/src/ hangserver_Fork.c	18
D:/workspaces/college/NG_Asgnmt_01/hangman/src/ hangserver_udp.c	21

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
```

```

00013 #include <stdbool.h>
00014 #include <time.h>
00015
00016 char* word[] = {
00017 #include "../rsc/words"
00018 };
00019
00020 enum Game State {
00021     IN_PROGRESS, WON, LOST
00022 };
00023
00024 #define NUM_OF_WORDS (sizeof(word) / sizeof(word[0]))
00025 #define MAX_PLAYERS 2
00026 #define MAX_LIVES 10
00027 #define MAX_TOTAL_LEN 120
00028
00029 #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

<i>argc</i>	- The count of cmdline arguments
<i>argv</i>	- 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

<i>argc</i>	- The count of cmdline arguments
<i>argv</i>	- 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

```
00001 #ifndef HANGCLIENT_H
00002 #define HANGCLIENT_H
00003
00004 #include <stdio.h>
00005 #include <sys/types.h>
00006 #include <sys/socket.h>
00007 #include <netinet/in.h>
00008 #include <netdb.h>
00009 #include <stdlib.h>
00010 #include <string.h>
00011 #include <unistd.h>
00012
00013 #define LINESIZE 80
00014 #define HANGMAN_TCP_GENERIC_PORT 1066 // Define this in the Specific Header
00015
00016 int main(int argc, char* argv[]);
00017
00018 #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

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

<i>sock</i>	- The Client socket to Send/Receive to/from
<i>serv_addr</i>	- The address of the remote Server
<i>serv_len</i>	- 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

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

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

Parameters

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

Definition at line [134](#) of file [hangclient_udp.c](#).

hangclient_udp.h

```

00001
00010 #ifndef HANGMAN_HANGSERVER_UDP_H
00011 #define HANGMAN_HANGSERVER_UDP_H
00012
00013 #include "datatypes_all.h"
00014 #include "datatypes_client.h"
00015 #include "datatypes_udp.h"
00016
00017 #define GUESS_LEN 2
00018
00019 void play hangman(int sock, struct sockaddr* serv_addr, socklen_t serv_len, char
cli_id[ID_LEN]);
00020
00021 void test_connection(int sock, struct sockaddr* serv_addr, socklen_t serv_len);
00022
00023 void setup_connection(int sock, struct sockaddr* serv_addr, socklen_t serv_len);
00024
00025 #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_addr, 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_addr*, 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

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

<i>sock</i>	- The Client socket to Send/Receive to/from
<i>cli_addr</i>	- The address of the remote Client
<i>cli_len</i>	- The length of the Client Address Structure
<i>cli_count</i>	- 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

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

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

Parameters

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

Definition at line [134](#) of file [hangclient_udp.c](#).

hangserver_udp.h

```
00001
00010 #ifndef HANGMAN_HANGSERVER_UDP_H
00011 #define HANGMAN_HANGSERVER_UDP_H
00012
00013 #include "datatypes_all.h"
00014 #include "datatypes_server.h"
00015 #include "datatypes_udp.h"
00016
00017 void play_hangman(int sock, struct sockaddr_in* cli_addrs, socklen_t cli_len, const
int* connected_clients);
00018
00019 void test_connection(int sock, struct sockaddr* cli_addr, socklen_t cli_len);
00020
00021 void setup_connections(int sock, struct sockaddr* cli_addr, socklen_t cli_len, const
int* cli_count);
00022
00023 #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     */
```

```

00045
00046     printf("Connected to server %s \n", server_name);
00047     while ((count = read(sock, i_line, LINESIZE)) > 0) {
00048         write(1, i_line, count);
00049         count = read(0, o_line, LINESIZE); //0 = STDIN
00050         write(sock, o_line, count);
00051     }
00052 }

```

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

<i>argc</i>	- The count of cmdline arguments
<i>argv</i>	- 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

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

<i>sock</i>	- The Client socket to Send/Receive to/from
-------------	---

<i>serv_addr</i>	- The address of the remote Server
<i>serv_len</i>	- 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

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

Definition at line [134](#) of file [hangclient_udp.c](#).

hangclient_udp.c

```

00001
00010 #include "../hdr/hangclient_udp.h"
00011
00012
00020 void play hangman(int sock, struct sockaddr* serv_addr, socklen_t serv_len, char
cli_id[ID_LEN]) {
00021     ssize_t count;
00022     int round_local;
00023     char hostname[MAX_LEN];
00024     char i_line[MAX_LEN];
00025     char o_guess[GUESS_LEN];
00026     char temp_guess[GUESS_LEN];
00027
00028     // Zero out all data before starting
00029     memset(&count, '\0', sizeof(count));
00030     memset(&round_local, '\0', sizeof(round_local));
00031     memset(&hostname, '\0', sizeof(hostname));
00032     memset(&i_line, '\0', sizeof(i_line));
00033     memset(&o_guess, '\0', sizeof(o_guess));
00034     memset(&temp_guess, '\0', sizeof(temp_guess));
00035
00036     // Get the Human Readable name of this host
00037     gethostname(hostname, MAX_LEN);
00038
00039     fprintf(stdout, "Playing Hangman as Client #s on [%s]\n", cli_id, hostname);
00040
00041     // Receive the Hostname of the Server
00042     count = recvfrom(sock, i_line, MAX_LEN, 0, serv_addr, &serv_len);
00043
00044     // Check the received data for errors
00045     if (count < 0) {
00046         perror("Receiving from Server Socket Failed\n");
00047         exit(4); // Error Condition 04
00048     }
00049
00050     fprintf(stdout, "Connected to Server: %s\n", i_line);
00051
00052     // Receive the initial game state
00053     memset(&i_line, '\0', sizeof(i_line));
00054     count = recvfrom(sock, i_line, MAX_LEN, 0, serv_addr, &serv_len);
00055
00056     // Check the received data for errors
00057     if (count < 0) {
00058         perror("Receiving from Server Socket Failed\n");
00059         exit(4); // Error Condition 04
00060     }
00061
00062     fprintf(stdout, "\nInitial Game State:%s", i_line);
00063
00064     // Play the game
00065     do {
00066         do {
00067             // Receive the current turn from the Server

```



```

00068         memset(&i_line, '\0', sizeof(i_line));
00069         count = recvfrom(sock, i_line, MAX_LEN, 0, serv_addr, &serv_len);
00070
00071         // Check the received data for errors
00072         if (count < 0) {
00073             perror("Receiving from Server Socket Failed\n");
00074             exit(4); // Error Condition 04
00075         }
00076
00077         if (strcmp(i_line, "#GAMEOVER\0") == 0) { exit(0); }
00078     } while (strcmp(i_line, cli_id) != 0);
00079
00080     fprintf(stdout, "\n---\nRound: %d", ++round_local);
00081
00082     // Receive the current game state
00083     memset(&i_line, '\0', sizeof(i_line));
00084     count = recvfrom(sock, i_line, MAX_LEN, 0, serv_addr, &serv_len);
00085
00086     // Check the received data for errors
00087     if (count < 0) {
00088         perror("Receiving from Server Socket Failed\n");
00089         exit(4); // Error Condition 04
00090     }
00091
00092     fprintf(stdout, "\nGame State:%s", i_line);
00093     memset(&i_line, '\0', sizeof(i_line));
00094
00095     // Securely retrieve data from the User
00096     memset(&temp_guess, '\0', sizeof(temp_guess));
00097
00098     // Don't allow bad characters
00099     while (!isalpha((unsigned) temp_guess[0])) { // NOLINT (Bug in CLion)
00100         fprintf(stdout, "\nGuess a LETTER [a-z]\n>>");
00101         temp_guess[0] = (char) fgetc(stdin);
00102     }
00103
00104     // Convert the letter to lowercase
00105     temp_guess[0] = tolower(temp_guess[0]);
00106
00107     // Terminate the String and prepare it for sending
00108     strncat(temp_guess, "\0", GUESS_LEN);
00109     strncpy(o_guess, temp_guess, GUESS_LEN);
00110
00111     // Show the User what they typed
00112     fprintf(stdout, "Guess was: %s", o_guess);
00113
00114     // Send the data to the Server
00115     count = sendto(sock, o_guess, GUESS_LEN, 0, serv_addr, serv_len);
00116
00117     // Check the sent data for errors
00118     if (count < 0) {
00119         perror("Sending to Server Socket Failed\n");
00120         exit(3); // Error Condition 03
00121     }
00122
00123     } while (strcmp(i_line, "#GAMEOVER\0") != 0);
00124 }
00125
00126 void test_connection(int sock, struct sockaddr* serv_addr, socklen_t serv_len) {
00135     ssize_t count;
00136     char i_line[MAX_LEN + 1];
00137     char o_line[MAX_LEN];
00138
00139     // Zero the data out
00140     memset(&i_line, '\0', sizeof(i_line));
00141     memset(&o_line, '\0', sizeof(o_line));
00142
00143     fprintf(stdout, "Testing Connection\n");
00144
00145     fprintf(stdout, ">>>");
00146     while (fgets(o_line, MAX_LEN, stdin) != NULL) {
00147         fprintf(stdout, "Sending: %s\n", o_line);
00148
00149         // Send the data to the Server
00150         sendto(sock, o_line, strlen(o_line), 0, serv_addr, serv_len);
00151

```

```

00152         // Receive a reply from the Server
00153         count = recvfrom(sock, i_line, MAX_LEN, 0, NULL, NULL);
00154
00155         // Check the received data for errors
00156         if (count < 0) {
00157             perror("Receiving from Server Socket Failed\n");
00158             exit(3); // Error Condition 03
00159         }
00160
00161         i_line[count] = 0;
00162         fprintf(stdout, "Received: %s\n---\n", i_line);
00163
00164         memset(&i_line, '\0', sizeof(i_line));
00165         memset(&o_line, '\0', sizeof(o_line));
00166
00167         fprintf(stdout, "\n>>");
00168     }
00169 }
00170
00171
00172 void setup_connection(int sock, struct sockaddr* serv_addr, socklen_t serv_len) {
00173     ssize_t count;
00174     char id_request[ID_LEN + 1];
00175     char id_response[ID_LEN];
00176
00177     // Zero the data out
00178     memset(&id_request, '\0', sizeof(id_request));
00179     memset(&id_response, '\0', sizeof(id_response));
00180
00181     // Assign request character
00182     strncpy(id_request, "-1", ID_LEN);
00183
00184     fprintf(stdout, "Sending: %s\n", id_request);
00185
00186     // Send the data to the Server
00187     count = sendto(sock, id_request, ID_LEN, 0, serv_addr, serv_len);
00188
00189     // Check the sent data for errors
00190     if (count < 0) {
00191         perror("Sending to Server Socket Failed\n");
00192         exit(3); // Error Condition 03
00193     }
00194
00195     // Receive a reply from the Server
00196     count = recvfrom(sock, id_response, ID_LEN, 0, serv_addr, &serv_len);
00197
00198     // Check the received data for errors
00199     if (count < 0) {
00200         perror("Receiving from Server Socket Failed\n");
00201         exit(4); // Error Condition 04
00202     }
00203
00204     fprintf(stdout, "Received: %s\n---\n", id_response);
00205
00206     memset(&id_request, '\0', sizeof(id_request));
00207
00208     play_hangman(sock, (struct sockaddr*) &serv_addr, sizeof(*serv_addr),
00209 id_response);
00210 }
00211
00212
00213 int main(int argc, char* argv[]) {
00214     int udp_sock;
00215     struct sockaddr_in serv_addr;
00216     struct hostent* host_info;
00217     char* server_name;
00218
00219     // Set the Server address to the cmdline option, or LOCALHOST
00220     server_name = (argc == 2) ? argv[1] : "localhost";
00221
00222     // Convert the IP Address to a Human-Readable format
00223     host_info = gethostbyname(server_name);
00224     if (host_info == NULL) {
00225         perror("Unknown Host\n");
00226         exit(1);
00227     }
00228 }
00229
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244

```

```

00245 // Build up the Server Address Structure
00246 memset(&serv_addr, '\0', sizeof(serv_addr));
00247 serv_addr.sin_family = (sa_family_t) host_info->h_addrtype;
00248 memcpy((char*) &serv_addr.sin_addr, host_info->h_addr, (size_t)
host_info->h_length);
00249 serv_addr.sin_port = htons(HANGMAN_UDP_PORT);
00250
00251 // Create the local Socket
00252 udp_sock = socket(AF_INET, SOCK_DGRAM, 0); //0 or IPPROTO_UDP
00253
00254 // Error check The Socket
00255 if (udp_sock < 0) {
00256     perror("Creating Datagram Socket Failed\n");
00257     exit(2);
00258 }
00259
00260 fprintf(stdout, "UDP Client Socket Created\n");
00261
00262 //test_connection(udp_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr));
00263 setup_connection(udp_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr));
00264
00265 // Close the Socket, and exit the program
00266 close(udp_sock);
00267 return (0);
00268 }

```

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

```
00001 /* Network server for hangman game */
00002 /* File: hangserver.c */
00003
00004 #include "../hdr/hangserver.h"
00005
00006 int maxlives = 12;
00007
00008 /* ----- Play_hangman () ----- */
00009
00010 void play_hangman(int in, int out) {
00011     char* whole_word, part_word[MAXLEN],
00012         guess[MAXLEN], outbuf[MAXLEN];
00013
00014     int lives = maxlives;
00015     int game_state = 'I'; // I = Incomplete
00016     int i, good_guess, word_length;
00017     char hostname[MAXLEN];
00018
00019     gethostname(hostname, MAXLEN);
00020     sprintf(outbuf, "Playing hangman on host %s: \n\n", hostname);
00021     write(out, outbuf, strlen(outbuf));
00022
00023     // Pick a word at random from the list
00024     whole_word = word[rand() % NUM_OF_WORDS];
00025     word_length = strlen(whole_word);
00026     syslog(LOG_USER | LOG_INFO, "Server chose hangman word %s", whole_word);
00027
00028     // No letters are guessed Initially
00029     for (i = 0; i < word_length; i++) {
00030         part_word[i] = '-';
00031     }
00032
00033     part_word[i] = '\0';
00034
00035     sprintf(outbuf, "%s %d \n", part_word, lives);
00036     write(out, outbuf, strlen(outbuf));
00037
00038     while (game_state == 'I')
00039         // Get a letter from player guess
00040     {
00041         while (read(in, guess, MAXLEN) < 0) {
00042             if (errno != EINTR) {
00043                 exit(4);
00044             }
00045             printf("re-read the start in \n");
00046         } // Re-start read () if interrupted by signal
00047         good_guess = 0;
00048         for (i = 0; i < word_length; i++) {
00049             if (guess[0] == whole_word[i]) {
00050                 good_guess = 1;
00051                 part_word[i] = whole_word[i];
00052             }
00053         }
00054         if (!good_guess) { lives--; }
00055         if (strcmp(whole_word, part_word) == 0) {
00056             game_state = 'W'; // W ==> User Won
00057         } else if (lives == 0) {
00058             game_state = 'L'; // L ==> User Lost
00059             strcpy(part_word, whole_word); // User Show the word
00060         }
00061         sprintf(outbuf, "%s %d \n", part_word, lives);
00062         write(out, outbuf, strlen(outbuf));
00063     }
00064 }
00065
00066 int main() {
00067     int sock, fd, client_len;
00068     struct sockaddr_in server, client;
00069
00070     srand((int) time((long*) 0)); // randomize the seed
00071 }
```

```

00072     sock = socket(AF_INET, SOCK_STREAM, 0); //0 or IPPROTO_TCP
00073     if (sock < 0) { // This error checking is the code Stevens wraps in his Socket
Function etc
00074         perror("creating stream socket");
00075         exit(1);
00076     }
00077
00078     server.sin_family      = AF_INET;
00079     server.sin_addr.s_addr = htonl(INADDR_ANY);
00080     server.sin_port        = htons(HANGMAN\_TCP\_GENERIC\_PORT);
00081
00082     if (bind(sock, (struct sockaddr*) &server, sizeof(server)) < 0) {
00083         perror("binding socket");
00084         exit(2);
00085     }
00086
00087     listen(sock, 5);
00088
00089     while (1) {
00090         client_len = sizeof(client);
00091         if ((fd = accept(sock, (struct sockaddr*) &client, &client_len)) < 0) {
00092             perror("accepting connection");
00093             exit(3);
00094         }
00095         play\_hangman(fd, fd);
00096         close(fd);
00097     }
00098 }

```

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){
```

```

00019     }
00020     }
00021     exit(0);
00022 }
00023
00024 void testGameZombie(int in,int out){
00025
00026     printf("\nPlaying Test Connection\n\n");
00027
00028     exit(0);
00029 }
00030
00031
00032 void play hangman(int in, int out) {
00033     char* whole_word, part_word[MAXLEN],
00034           guess[MAXLEN], outbuf[MAXLEN];
00035
00036     int  lives      = maxlives;
00037     int  game_state = 'I'; // I = Incomplete
00038     int  i, good_guess, word_length;
00039     char hostname[MAXLEN];
00040
00041     gethostname(hostname, MAXLEN);
00042     sprintf(outbuf, "Playing hangman on host %s: \n\n", hostname);
00043     write(out, outbuf, strlen(outbuf));
00044
00045     // Pick a word at random from the list
00046     whole_word = word[rand() % NUM_OF_WORDS];
00047     word_length = strlen(whole_word);
00048     syslog(LOG_USER | LOG_INFO, "Server chose hangman word %s", whole_word);
00049
00050     // No letters are guessed Initially
00051     for (i = 0; i < word_length; i++) {
00052         part_word[i] = '-';
00053     }
00054
00055     part_word[i] = '\0';
00056
00057     sprintf(outbuf, "%s %d \n", part_word, lives);
00058     write(out, outbuf, strlen(outbuf));
00059
00060     while (game_state == 'I')
00061         // Get a letter from player guess
00062     {
00063         while (read(in, guess, MAXLEN) < 0) {
00064             if (errno != EINTR) {
00065                 exit(4);
00066             }
00067             printf("re-read the start in \n");
00068         } // Re-start read () if interrupted by signal
00069         good_guess = 0;
00070         for (i = 0; i < word_length; i++) {
00071             if (guess[0] == whole_word[i]) {
00072                 good_guess = 1;
00073                 part word[i] = whole word[i];
00074             }
00075         }
00076         if (!good_guess) { lives--; }
00077         if (strcmp(whole_word, part_word) == 0) {
00078             game_state = 'W'; // W ==> User Won
00079         } else if (lives == 0) {
00080             game_state = 'L'; // L ==> User Lost
00081             strcpy(part_word, whole_word); // User Show the word
00082         }
00083         sprintf(outbuf, "%s %d \n", part_word, lives);
00084         write(out, outbuf, strlen(outbuf));
00085     }
00086 }
00087
00088 int main() {
00089     int sock, fd, client_len, childProcCount;
00090     struct sockaddr in server, client;
00091
00092     srand((int) time((long*) 0)); // randomize the seed
00093
00094     sock = socket(AF_INET, SOCK_STREAM, 0); // 0 or IPPROTO_TCP

```

```

00095     if (sock < 0) { // This error checking is the code Stevens wraps in his Socket
Function etc
00096         perror("creating stream socket");
00097         exit(1);
00098     }
00099
00100     server.sin_family      = AF_INET;
00101     server.sin_addr.s_addr = htonl(INADDR_ANY);
00102     server.sin_port        = htons(HANGMAN_TCP_PORT);
00103
00104     if (bind(sock, (struct sockaddr*) &server, sizeof(server)) < 0) {
00105         perror("binding socket");
00106         exit(2);
00107     }
00108
00109     listen(sock, 5);
00110     childProcCount = 0;
00111     while (1) {
00112         client_len = sizeof(client);
00113         //Accept Connection "FORK HERE!!"
00114         if ((fd = accept(sock, (struct sockaddr*) &client, &client_len)) < 0) {
00115             perror("accepting connection");
00116             exit(3);
00117         }
00118         pid_t pid = fork();
00119
00120         if(pid < 0){
00121             perror("Fork() Failed");
00122         }
00123         if(pid==0){ //TODO Gamify with Hangman
00124             perror("Fork Connection Accepted");
00125             play_hangman(fd,fd);
00126             exit(0);
00127         }
00128
00129         //Increment Child Tracker
00130         printf("",pid);
00131         close(fd);
00132         childProcCount++;
00133         //clean up zombies
00134         while(childProcCount){
00135             pid = waitpid((pid_t) -1 ,NULL,WNOHANG);
00136             if(pid < 0){
00137                 perror("waitpid() failed No Zombie Found");
00138             }
00139             else if(pid ==0){
00140                 perror("No Zombies, break");
00141                 break;
00142             }
00143             else{
00144                 perror("Zombie terminated");
00145                 childProcCount--;
00146             }
00147         }
00148
00149         //play_hangman(fd, fd);
00150         //close(fd);
00151     }
00152 }
00153

```

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

<i>argc</i>	- The count of cmdline arguments
<i>argv</i>	- 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

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

<i>sock</i>	- The Client socket to Send/Receive to/from
<i>cli_addr</i>	- The address of the remote Client
<i>cli_len</i>	- The length of the Client Address Structure
<i>cli_count</i>	- 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

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

Definition at line [210](#) of file [hangserver_udp.c](#).

hangserver_udp.c

```
00001
00010 #include "../hdr/hangserver_udp.h"
00011
00012
00021 void play_hangman(int sock, struct sockaddr_in* cli_addrs, socklen_t cli_len, const
int* connected_clients) {
00022     fprintf(stdout, "\n---\nPlaying Hangman\n");
00023
00024     // Set up the game
00025     ssize_t count;
00026     char* whole_word,
00027         part_word[MAX_LEN],
00028         outbuf[MAX_TOTAL_LEN];
00029     bool good_guess;
00030     size_t word_length;
00031     char hostname[MAX_LEN];
00032     char guess[2];
00033     int lives;
00034     int clients_in_play = *connected_clients;
00035     enum Game State game_state = IN_PROGRESS;
00036
00037     // Zero out all data before starting
00038     memset(&count, '\0', sizeof(count));
00039     memset(&whole_word, '\0', sizeof(whole_word));
00040     memset(&part_word, '\0', sizeof(part_word));
00041     memset(&outbuf, '\0', sizeof(outbuf));
00042     memset(&guess, '\0', sizeof(guess));
00043     memset(&lives, '\0', sizeof(lives));
00044     memset(&good_guess, '\0', sizeof(good_guess));
00045     memset(&word_length, '\0', sizeof(word_length));
00046     memset(&hostname, '\0', sizeof(hostname));
00047
00048     fprintf(stdout, "\nThere are %d Clients in play\n", clients_in_play);
00049
00050     // Pick a word at random from the list
00051     whole_word = word[random() % NUM_OF_WORDS];
00052     word_length = strlen(whole_word);
00053     lives = MAX_LIVES;
00054     fprintf(stdout, "\nServer chose hangman word %s", whole_word);
00055
00056     // Ensure no letters are guessed Initially
00057     for (int j = 0; j < (int) word_length; j++) {
00058         part_word[j] = '-';
00059     }
00060
00061     // Null-terminate the String
00062     part_word[word_length] = '\0';
00063     fprintf(stdout, "\nWordWhole: %s", whole_word);
00064     fprintf(stdout, "\nWordPart: %s", part_word);
00065
00066     // Get the Human Readable name of this host
00067     gethostname(hostname, MAX_LEN);
00068
00069     // Client currently being handled
00070     struct sockaddr* cli_addr = NULL;
00071
00072     // Introduce each Client
00073     for (int i = 0; i < clients_in_play; i++) {
00074         // Set the current Client
00075         cli_addr = (struct sockaddr*) &cli_addrs[i];
00076
00077         memset(&outbuf, '\0', sizeof(outbuf));
00078         sprintf(outbuf, "%s", hostname);
00079         sendto(sock, outbuf, strlen(outbuf), 0, cli_addr, cli_len);
00080
00081         // Check that there were no errors with sending the data
00082         if (count < 0) {
00083             perror("Sending to Client Socket Failed\n");
00084             exit(4); // Error Condition 04
00085         }
00086     }
```

```

00087     memset(&outbuf, '\0', sizeof(outbuf));
00088     snprintf(outbuf, MAX TOTAL LEN, "\n\tWord: %s\n\tLives: %hu", part_word,
(unsigned short) lives);
00089     sendto(sock, outbuf, strlen(outbuf), 0, cli_addr, cli_len);
00090
00091     // Check that there were no errors with sending the data
00092     if (count < 0) {
00093         perror("Sending to Client Socket Failed\n");
00094         exit(4); // Error Condition 04
00095     }
00096 }
00097
00098 // Loop until the game is WON
00099 while (game_state == IN PROGRESS) {
00100     good_guess = false;
00101
00102     // Loop for each Client
00103     for (int i = 0; i < clients_in_play; i++) {
00104         fprintf(stdout, "\nServing Client %d", i);
00105
00106         // Set the current Client
00107         cli_addr = (struct sockaddr*) &cli_addrs[i];
00108
00109         // Inform the Client that it's their turn
00110         memset(&outbuf, '\0', sizeof(outbuf));
00111         sprintf(outbuf, "%d", (i + 1));
00112         sendto(sock, outbuf, strlen(outbuf), 0, cli_addr, cli_len);
00113
00114         // Check that there were no errors with sending the data
00115         if (count < 0) {
00116             perror("Sending to Client Socket Failed\n");
00117             exit(4); // Error Condition 04
00118         }
00119
00120         // Send the current state of the game to the Client
00121         memset(&outbuf, '\0', sizeof(outbuf));
00122         snprintf(outbuf, MAX TOTAL LEN, "\n\tWord: %s\n\tLives: %hu",
part word, (unsigned short) lives);
00123         sendto(sock, outbuf, strlen(outbuf), 0, cli_addr, cli_len);
00124
00125         // Check that there were no errors with sending the data
00126         if (count < 0) {
00127             perror("Sending to Client Socket Failed\n");
00128             exit(4); // Error Condition 04
00129         }
00130
00131         // Get a letter from player guess
00132         count = recvfrom(sock, guess, MAX LEN, 0, cli_addr, &cli_len);
00133
00134         // Check the received data for errors
00135         if (count < 0) {
00136             perror("Receiving from Client Socket Failed\n");
00137             exit(3); // Error Condition 03
00138         }
00139
00140         // Evaluate the Client's guess
00141         for (int j = 0; j < (int) word_length; j++) {
00142             if (guess[0] == whole_word[j]) {
00143                 good_guess = true;
00144                 part_word[j] = whole_word[j];
00145             }
00146         }
00147
00148         // If the guess was bad, subtract from the Lives counter
00149         if (!good_guess) {
00150             lives--;
00151         }
00152
00153         // If the whole word has been guessed
00154         if (strcmp(whole_word, part_word) == 0) {
00155             game_state = WON; // User Won
00156
00157             // Let the Client(s) know they WON
00158             memset(&outbuf, '\0', sizeof(outbuf));
00159             sprintf(outbuf, "%s", "#GAMEOVER");
00160             sendto(sock, outbuf, strlen(outbuf), 0, cli_addr, cli_len);
00161

```

```

00162         // Check that there were no errors with sending the data
00163         if (count < 0) {
00164             perror("Sending to Client Socket Failed\n");
00165             exit(4); // Error Condition 04
00166         }
00167
00168     } else if (lives == 0) {
00169         game_state = LOST; // User Lost
00170
00171         // Let the Client(s) know they LOST
00172         memset(&outbuf, '\0', sizeof(outbuf));
00173         sprintf(outbuf, "%s", "#GAMEOVER");
00174         sendto(sock, outbuf, strlen(outbuf), 0, cli_addr, cli_len);
00175
00176         // Check that there were no errors with sending the data
00177         if (count < 0) {
00178             perror("Sending to Client Socket Failed\n");
00179             exit(4); // Error Condition 04
00180         }
00181
00182         strcpy(part_word, whole_word); // User Show the word
00183     }
00184 }
00185 }
00186
00187 // Send ENDGAME message
00188 for (int i = 0; i < clients_in_play; i++) {
00189     // Set the current Client
00190     cli_addr = (struct sockaddr*) &cli_addrs[i];
00191
00192     sendto(sock, outbuf, strlen(outbuf), 0, cli_addr, cli_len);
00193
00194     // Check that there were no errors with sending the data
00195     if (count < 0) {
00196         perror("Sending to Client Socket Failed\n");
00197         exit(4); // Error Condition 04
00198     }
00199 }
00200 }
00201
00202 void test_connection(int sock, struct sockaddr* cli_addr, socklen_t cli_len) {
00203     ssize_t count;
00204     char i_line[MAX_LEN];
00205
00206     fprintf(stdout, "Testing Connection\n");
00207
00208     do {
00209         memset(&i_line, '\0', sizeof(i_line));
00210         fprintf(stdout, "---\nAwaiting data on Socket %d...\n\n", sock);
00211
00212         // Receive data from the Client Socket
00213         count = recvfrom(sock, i_line, MAX_LEN, 0, cli_addr, &cli_len);
00214         i_line[count] = '\0';
00215
00216         // Check the received data for errors
00217         if (count < 0) {
00218             perror("Receiving from Client Socket Failed\n");
00219             exit(3); // Error Condition 03
00220         }
00221
00222         // Print the received message to the screen
00223         fprintf(stdout, "Messg Received: %s", i_line);
00224
00225         // Send data to the Client Socket
00226         count = sendto(sock, i_line, MAX_LEN, 0, cli_addr, cli_len);
00227
00228         // Check that there were no errors with sending the data
00229         if (count < 0) {
00230             perror("Sending to Client Socket Failed\n");
00231             exit(4); // Error Condition 04
00232         }
00233
00234         // Print confirmation of the send to the screen
00235         fprintf(stdout, "Messg Sent: %s", i_line);
00236     } while (strcmp(i_line, "#quit\0") != 0); // ToDo: Create stop condition that
00237     actually works

```

```

00245 }
00246
00247
00255 void setup_connections(int sock, struct sockaddr* cli_addr, socklen_t cli_len, const
int* cli_count) {
00256     ssize_t count;
00257     int client_id;
00258     char id_request[ID_LEN];
00259     char id_response[ID_LEN];
00260
00261     // Zero out data
00262     memset(&count, '\0', sizeof(count));
00263     memset(&client_id, '\0', sizeof(client_id));
00264     memset(&id_request, '\0', sizeof(id_request));
00265     memset(&id_response, '\0', sizeof(id_response));
00266     memset(cli_addr, '\0', sizeof(cli_len));
00267
00268     fprintf(stdout, "\nSetting Up New Client\n");
00269     fprintf(stdout, "\nAwaiting request on Socket %d...\n", sock);
00270
00271     // Receive data from the Client Socket
00272     count = recvfrom(sock, id_request, ID_LEN, 0, cli_addr, &cli_len);
00273
00274     // Check the received data for errors
00275     if (count < 0) {
00276         perror("Receiving from Client Socket Failed\n");
00277         exit(3); // Error Condition 03
00278     }
00279
00280     // Print the received message to the screen
00281     fprintf(stdout, "\nMessg Received: %s", id_request);
00282     client_id = (*cli_count) + 1;
00283
00284     // Assign a Client ID to the Client if it is new
00285     if (strcmp(id_request, "-1\0") == 0) {
00286         fprintf(stdout, "\nClient is new. Assigning ID: %d", client_id);
00287         sprintf(id_response, "%d", client_id); // Convert the int to char*
00288     } else {
00289         fprintf(stdout, "\nClient already assigned ID");
00290     }
00291
00292     // Send data to the Client Socket
00293     count = sendto(sock, id_response, ID_LEN, 0, cli_addr, cli_len);
00294
00295     // Check that there were no errors with sending the data
00296     if (count < 0) {
00297         perror("Sending to Client Socket Failed\n");
00298         exit(4); // Error Condition 04
00299     }
00300
00301     // Print confirmation of the send to the screen
00302     fprintf(stdout, "\nMessg Sent: %s\n", id_response);
00303 }
00304
00305
00315 int main(int argc, char* argv[]) {
00316     // Set the `max_players` to the cmdline option, or MAX_PLAYERS
00317     int max_players = (argc == 2) ? (int) strtol(argv[1], NULL, 10) : MAX_PLAYERS;
00318     int udp_sock;
00319     struct sockaddr_in serv_addr;
00320     struct sockaddr_in cli_addr[max_players];
00321     int connected_clients;
00322
00323     // Zero out Server data
00324     memset(&serv_addr, '\0', sizeof(serv_addr));
00325     memset(&udp_sock, '\0', sizeof(udp_sock));
00326     memset(&connected_clients, '\0', sizeof(connected_clients));
00327
00328     for (int i = 0; i < max_players; i++) {
00329         memset(&cli_addr[i], '\0', sizeof(cli_addr[i]));
00330     }
00331
00332     // Seed the random number generator
00333     srand((unsigned int) time(NULL));
00334
00335     // Create the UDP Socket
00336     udp_sock = socket(AF_INET, SOCK_DGRAM, 0); //0 or IPPROTO_UDP

```

```

00337
00338 // Error check The Socket
00339 if (udp_sock < 0) {
00340     perror("Creating Datagram Socket Failed\n");
00341     exit(1); // Error Condition 01
00342 }
00343
00344 // Build the Server Address manually
00345 serv_addr.sin_family = AF_INET;
00346 serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
00347 serv_addr.sin_port = htons(HANGMAN_UDP_PORT);
00348
00349 // Bind the Server socket to an address
00350 if (bind(udp_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) < 0) {
00351     perror("Binding Datagram Socket Failed\n");
00352     exit(2); // Error Condition 02
00353 }
00354 fprintf(stdout, "UDP Server Socket Created\n");
00355
00356 // Test connections (DEBUG FUNCTION)
00357 //test_connection(udp_sock, (struct sockaddr*) &cli_addr, sizeof(cli_addr));
00358
00359 // Accept Clients until all game slots are full
00360 connected_clients = 0;
00361 for (int i = 0; i < max_players; i++) {
00362     fprintf(stdout, "\n---\nCreating Client #%d", connected_clients);
00363     setup_connections(udp_sock, (struct sockaddr*) &cli_addrs[i],
00364         sizeof(cli_addrs[i]), &connected_clients);
00365     connected_clients++;
00366 }
00367 play_hangman(udp_sock, cli_addrs, sizeof(struct sockaddr),
00368     &connected_clients);
00369
00370 // Close the Socket, and exit the program
00371 close(udp_sock);
00372 return (0);
00373 }

```

Index

D:/workspaces/college/NG_Asgnmt_01/hangman/CMakeLists.txt, 1	D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangserver_Fork.c, 18, 19
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/datatypes_all.h, 1, 2	D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangserver_udp.c, 21, 23
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/datatypes_client.h, 2	datatypes_all.h
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/datatypes_server.h, 2, 3	MAX_LEN, 1
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/datatypes_udp.h, 4	datatypes_server.h
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/hangclient.h, 4, 6	Game_State, 3
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/hangclient_udp.h, 6, 7	IN_PROGRESS, 3
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/hangserver.h, 8, 9	LOST, 3
D:/workspaces/college/NG_Asgnmt_01/hangman/hdr/hangserver_udp.h, 9, 10	MAX_LIVES, 3
D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangclient.c, 11	MAX_PLAYERS, 3
D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangclient_udp.c, 12, 13	MAX_TOTAL_LEN, 3
D:/workspaces/college/NG_Asgnmt_01/hangman/src/hangserver.c, 16, 17	NUM_OF_WORDS, 3
	WON, 3
	word, 3
	datatypes_udp.h
	HANGMAN_UDP_PORT, 4
	ID_LEN, 4
	Game_State
	datatypes_server.h, 3
	GUESS_LEN
	hangclient_udp.h, 6
	hangclient.c
	main, 11

- hangclient.h
 - HANGMAN_TCP_GENERIC_PORT, 5
 - LINESIZE, 5
 - main, 5
- hangclient_udp.c
 - main, 12
 - play_hangman, 12
 - setup_connection, 12
 - test_connection, 13
- hangclient_udp.h
 - GUESS_LEN, 6
 - play_hangman, 6
 - setup_connection, 7
 - test_connection, 7
- HANGMAN_TCP_GENERIC_PORT
 - hangclient.h, 5
 - hangserver.h, 8
- HANGMAN_TCP_PORT
 - hangserver_Fork.c, 18
- HANGMAN_UDP_PORT
 - datatypes_udp.h, 4
- hangserver.c
 - main, 16
 - maxlives, 16
 - play_hangman, 16
- hangserver.h
 - HANGMAN_TCP_GENERIC_PORT, 8
 - MAXLEN, 8
 - NUM_OF_WORDS, 8
 - time, 8
 - word, 9
- hangserver_Fork.c
 - HANGMAN_TCP_PORT, 18
 - main, 19
 - MAXLEN, 18
 - maxlives, 19
 - NUM_OF_WORDS, 19
 - play_hangman, 19
 - testGameNoZombie, 19
 - testGameZombie, 19
 - time, 19
- hangserver_udp.c
 - main, 22
 - play_hangman, 22
 - setup_connections, 22
 - test_connection, 22
- hangserver_udp.h
 - play_hangman, 9
 - setup_connections, 10
 - test_connection, 10
- ID_LEN
 - datatypes_udp.h, 4
- IN_PROGRESS
 - datatypes_server.h, 3
- LINESIZE
 - hangclient.h, 5
- LOST
 - datatypes_server.h, 3
- main
 - hangclient.c, 11
 - hangclient.h, 5
 - hangclient_udp.c, 12
 - hangserver.c, 16
 - hangserver_Fork.c, 19
 - hangserver_udp.c, 22
- MAX_LEN
 - datatypes_all.h, 1
- MAX_LIVES
 - datatypes_server.h, 3
- MAX_PLAYERS
 - datatypes_server.h, 3
- MAX_TOTAL_LEN
 - datatypes_server.h, 3
- MAXLEN
 - hangserver.h, 8
 - hangserver_Fork.c, 18
- maxlives
 - hangserver.c, 16
 - hangserver_Fork.c, 19
- NUM_OF_WORDS
 - datatypes_server.h, 3
 - hangserver.h, 8
 - hangserver_Fork.c, 19
- play_hangman
 - hangclient_udp.c, 12
 - hangclient_udp.h, 6
 - hangserver.c, 16
 - hangserver_Fork.c, 19
 - hangserver_udp.c, 22
 - hangserver_udp.h, 9
- setup_connection
 - hangclient_udp.c, 12
 - hangclient_udp.h, 7
- setup_connections
 - hangserver_udp.c, 22
 - hangserver_udp.h, 10
- test_connection
 - hangclient_udp.c, 13
 - hangclient_udp.h, 7
 - hangserver_udp.c, 22
 - hangserver_udp.h, 10
- testGameNoZombie
 - hangserver_Fork.c, 19
- testGameZombie
 - hangserver_Fork.c, 19
- time
 - hangserver.h, 8
 - hangserver_Fork.c, 19
- WON
 - datatypes_server.h, 3
- word
 - datatypes_server.h, 3
 - hangserver.h, 9