Networked Hangman C

Version 1.0.0
4/14/2020 5:57:00 PM

Table of Contents

File Index

File List

Here is a list of all files with brief descriptions:

File Documentation

NG_Asgnmt_01/hangman/CMakeLists.txt File Reference

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
- #define GUESS_LEN 2

Macro Definition Documentation

#define GUESS_LEN 2

Definition at line 20 of file datatypes_all.h.

#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 #define GUESS_LEN 2
00021
00022 #endif //HANGMAN_DATATYPES_ALL_H
```

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
```

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 28 of file datatypes_server.h.

#define MAX_PLAYERS 2

Definition at line 27 of file datatypes_server.h.

#define MAX_TOTAL_LEN 120

Definition at line 29 of file datatypes_server.h.

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

Definition at line 26 of file datatypes_server.h.

Enumeration Type Documentation

enum Game_State

Enumerator:

IN_PROGRESS	
WON	
LOST	

Definition at line 22 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 18 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
00017
00018 char* word[] = {
00019 #include "../rsc/words"
00020 };
00021
00022 enum Game_State {
00023     IN_PROGRESS, WON, LOST
00024 };
00025
00026 #define NUM_OF_WORDS (sizeof(word) / sizeof(word[0]))
00027 #define MAX_PLAYERS 2
00028 #define MAX_LIVES 10
00029 #define MAX_TOTAL_LEN 120
00030
00031 #endif //HANGMAN_DATATYPES_SERVER_H
```

NG_Asgnmt_01/hangman/hdr/datatypes_tcp.h File Reference

```
#include <unistd.h>
#include <stdarg.h>
```

Macros

- #define HANGMAN_TCP_PORT 1168
- #define HANGMAN_TCP_FORK_PORT 1268

Macro Definition Documentation

#define HANGMAN_TCP_FORK_PORT 1268

Definition at line 9 of file datatypes_tcp.h.

#define HANGMAN_TCP_PORT 1168

Definition at line 8 of file datatypes_tcp.h.

datatypes_tcp.h

```
00001
00002 #ifndef HANGMAN_DATATYPES_TCP_H
00003 #define HANGMAN_DATATYPES_TCP_H
```

```
00004
00005 #include <unistd.h>
00006 #include <stdarg.h>
00007
00008 #define HANGMAN_TCP_PORT 1168
00009 #define HANGMAN_TCP_FORK_PORT 1268
00010
00011 #endif //HANGMAN_DATATYPES_TCP_H
```

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
```

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*[])

Author: Rory Ryan [K00218864] Date: 2020/03/09

This is the Source file for the Client side of the TCP version of the Networked Hangman game. Main Client program for online Hangman game served on forking server

Parameters

argc	- Number of arguments provided to the terminal
argv	- Arguments provided

Returns

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

```
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
```

NG_Asgnmt_01/hangman/hdr/hangclient_tcp.h File Reference

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

hangclient_tcp.h

```
00001 #ifndef HANGMAN_HANGCLIENT_TCP_H
00002 #define HANGMAN_HANGCLIENT_TCP_H
00003
00004 #include "datatypes_all.h"
00005 #include "datatypes_client.h"
00006 #include "datatypes_tcp.h"
00007
00008
00009 #endif //HANGMAN_HANGCLIENT_TCP_H
```

NG_Asgnmt_01/hangman/hdr/hangclient_tcp_fork.h File Reference

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

Functions

- 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 PassiveTCPClient (const char *, const char *)

Function Documentation

int PassiveTCPClient (const char * *server*, const char * *service*)

Constructs & assigns a socket for a TCP Client

Parameters

server	- The server IP address the socket will connect to
service	- The Socket this client program will communicate through

Returns

Definition at line 70 of file hangclient_tcp_fork.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 183 of file hangclient_udp.c.

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

Author: Rory Ryan [K00218864] Date: 2020/03/09

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

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_tcp_fork.h

```
00001
00010 #ifndef HANGMAN_HANGSERVER_FORK_H
00011 #define HANGMAN_HANGSERVER_FORK_H
00012
00013 #include "datatypes_all.h"
00014 #include "datatypes_client.h"
00015 #include "datatypes_tcp.h"
00016
00017
00018 void test_connection(int sock, struct sockaddr* serv_addr, socklen_t serv_len);
00019
00020 void setup_connection(int sock, struct sockaddr* serv_addr, socklen_t serv_len);
00021
00022 int PassiveTCPClient(const char *,const char *);
```

```
00023
00024
00025
00026 #endif //HANGMAN_HANGSERVER_FORK_H
```

NG_Asgnmt_01/hangman/hdr/hangclient_udp.h File Reference

```
#include "datatypes_all.h"
#include "datatypes_client.h"
#include "datatypes_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)

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 Header file for the Client 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 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 183 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

```
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 void play_hangman(int sock, struct sockaddr* serv_addr, socklen_t serv_len, char
cli_id[ID_LEN]);
00018
00019 void test_connection(int sock, struct sockaddr* serv_addr, socklen_t serv_len);
00020
00021 void setup_connection(int sock, struct sockaddr* serv_addr, socklen_t serv_len);
00022
00023 #endif //HANGMAN_HANGSERVER_UDP_H
```

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
```

NG_Asgnmt_01/hangman/hdr/hangserver_tcp.h File Reference

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

hangserver_tcp.h

```
00001 #ifndef HANGMAN_HANGSERVER_TCP_H
00002 #define HANGMAN_HANGSERVER_TCP_H
00003
00004 #include "datatypes_all.h"
00005 #include "datatypes_server.h"
00006 #include "datatypes_tcp.h"
00007
00008
00009 #endif //HANGMAN_HANGSERVER_TCP_H
```

NG_Asgnmt_01/hangman/hdr/hangserver_tcp_fork.h File Reference

```
#include <sys/wait.h>
#include "datatypes_all.h"
#include "datatypes_server.h"
#include "datatypes_tcp.h"
```

Typedefs

- typedef unsigned short u_short

Functions

- void reaper ()
- int passivesock (int service, const char *transport, int qlen)
- int passiveTCP (int, int)
- int errexit (const char *format,...)

Typedef Documentation

typedef unsigned short u_short

Definition at line 24 of file hangserver_tcp_fork.h.

Function Documentation

int errexit (const char * *format*, ...)

int passivesock (int *service*, const char * *transport*, int *qlen*)

Allocate & bind server socket

Parameters

service	- The Socket/Service to be used
transport	- The Transport Protocol being used by this socket
qlen	- The number of Clients in queue to be connected

Returns

Definition at line 134 of file hangserver_Fork.c.

int passiveTCP (int *service*, int *qlen*)

Call a sokcket build structure based on the protocol provided Currently only calls TCP

Parameters

service	- The Socket/Service to be used
qlen	- The number of Clients in queue to be connected

Returns

Definition at line 122 of file hangserver_Fork.c.

void reaper ()

Author: Rory Ryan [K00218864] Date: 2020/03/09

This is the Source file for the Client side of the TCP version of the Networked Hangman game.

The Signal Function designed to search for and Terminate Zombie Processses in the main program

Definition at line 96 of file hangserver_Fork.c.

hangserver_tcp_fork.h

```
00001
00010 #ifndef HANGMAN_HANGSERVER_TCP_FORK_H
00011 #define HANGMAN_HANGSERVER_TCP_FORK_H
00012
00013
00014 #include <sys/wait.h>
00015 #include "datatypes_all.h"
00016 #include "datatypes_server.h"
00017 #include "datatypes_tcp.h"
00018
00019 void reaper();
00020 int passivesock(int service,const char *transport,int qlen);
00021 int passiveTCP(int , int);
00022
00023 //extern int errno;
00024 typedef unsigned short u_short;
00025
00026 int errexit(const char *format, ...);
00027
00028 #endif //HANGMAN_HANGSERVER_TCP_FORK_H
```

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

```
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
```

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 }
```

NG_Asgnmt_01/hangman/src/hangclient_tcp_fork.c File Reference

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

Functions

- int main (int argc, char *argv[])
- int PassiveTCPClient (const char *server, const char *service)

Function Documentation

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

Author: Rory Ryan [K00218864] Date: 2020/03/09

This is the Source file for the Client side of the TCP version of the Networked Hangman game. Main Client program for online Hangman game served on forking server

Parameters

argc	- Number of arguments provided to the terminal
argv	- Arguments provided

Returns

Definition at line 19 of file hangclient_tcp_fork.c.

int PassiveTCPClient (const char * *server*, const char * *service*)

Constructs & assigns a socket for a TCP Client

Parameters

server	- The server IP address the socket will connect to
service	- The Socket this client program will communicate through

Returns

Definition at line 70 of file hangclient_tcp_fork.c.

hangclient_tcp_fork.c

```
00001
00010 #include "../hdr/hangclient_tcp_fork.h"
00011
00012
00019 int main(int argc, char* argv[]) {
00020     int sock, count;
00021     char buffer[MAX_LEN];
00022
00023     char *server = (argc >= 2) ? argv[1] : "localhost";
00024     //char *service = (argc == 3) ? argv[2] : (char *)HANGMAN_TCP_FORK_PORT;
00025     char *service = "1268";
00026
00027     fprintf(stdout, "\n---\nPassivetTCPClient(server: %s, service:
%s)\n---\n",server,service);
00028     // Create a connected TCP socket
00029
00030     sock = PassiveTCPClient(server, service);
00031     if (sock < 0){
00032         perror("\nSetupTCPClientSocket() failed");
00033         exit(1);
00034     }
00035
00036
00037
00038     printf("Please enter the message(Single Letter This is Hangman :) ): ");
00039     memset(buffer,'\0', sizeof(buffer));
00040     buffer[0] = (char) fgetc(stdin);
00041
00042     /* Send message to the server */
00043     count = write(sock, buffer, strlen(buffer));
00044
00045     if (count < 0) {
00046         perror("ERROR writing to socket");
00047         exit(1);
00048     }
00049
00050     /* Now read server response */
00051     memset(buffer,'\0',256);
00052     count = read(sock, buffer, 255);
00053
00054     if (count < 0) {
00055         perror("ERROR reading from socket");
00056         exit(1);
00057     }
00058
00059     printf("%s\n",buffer);
00060     return 0;
00061 }
00062
00063
00070 int PassiveTCPClient(const char *server,const char *service){
00071     printf("Passive TCP Client");
00072     struct addrinfo addrCriteria;                   // Criteria for address match
00073     memset(&addrCriteria, 0, sizeof(addrCriteria)); // Zero out structure
00074     addrCriteria.ai_family = AF_UNSPEC;             // v4 or v6 is OK
00075     addrCriteria.ai_socktype = SOCK_STREAM;         // Only streaming sockets
00076     addrCriteria.ai_protocol = IPPROTO_TCP;         // Only TCP protocol
```

```
00077
00078     // Get address(es)
00079     struct addrinfo *servAddr; // Holder for returned list of server addrs
00080     int rtnVal = getaddrinfo(server,service, &addrCriteria, &servAddr);
00081     if (rtnVal != 0) {
00082         perror("GetAddrInfo() Failed: ");
00083         exit(2);
00084     }
00085
00086     int sock = -1;
00087     for (struct addrinfo *addr = servAddr; addr != NULL; addr = addr->ai_next) {
00088         printf("\n---\nTesting:\n\nsock = socket(addr->ai_family: %d,
addr->ai_socktype: %d, addr->ai_protocol: %d)\n\n"
00089                 ,addr->ai_family,addr->ai_socktype,addr->ai_protocol);
00090         sock = socket(addr->ai_family, addr->ai_socktype, addr->ai_protocol);
00091         if (sock < 0){  //Failed
00092             perror("\nSocket Failed\n");
00093             continue;
00094         }
00095
00096         printf("connect(sock: %d, addr->ai_addr: %d, addr->ai_addrlen:
%d)\n---\n",sock,addr->ai_addr,addr->ai_addrlen);
00097         // Establish the connection to the echo server
00098         if (connect(sock, addr->ai_addr, addr->ai_addrlen) == 0){   //Success
00099             break;
00100         }
00101
00102         close(sock); // Failed
00103         sock = -1;
00104     }
00105
00106     freeaddrinfo(servAddr); // Free addrinfo allocated in getaddrinfo()
00107     return sock;
00108 }
```

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 232 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 183 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

```
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
00134 void test_connection(int sock, struct sockaddr* serv_addr, socklen_t serv_len) {
00135
00136     ssize_t count;
00137     char i_line[MAX_LEN + 1];
00138     char o_line[MAX_LEN];
00139
00140     // Zero the data out
00141     memset(&i_line, '\0', sizeof(i_line));
00142     memset(&o_line, '\0', sizeof(o_line));
00143
00144
00145     fprintf(stdout, "Testing Connection\n");
00146
00147     fprintf(stdout, ">>");
00148     while (fgets(o_line, MAX_LEN, stdin) != NULL) {
00149         fprintf(stdout, "Sending: %s\n", o_line);
00150
00151         // Send the data to the Server
00152         sendto(sock, o_line, strlen(o_line), 0, serv_addr, serv_len);
00153
00154         // Receive a reply from the Server
00155         count = recvfrom(sock, i_line, MAX_LEN, 0, NULL, NULL);
00156
00157
00158         // Check the received data for errors
00159         if (count < 0) {
00160             perror("Receiving from Server Socket Failed\n");
00161             exit(3); // Error Condition 03
00162         }
00163
00164         i_line[count] = 0;
00165         fprintf(stdout, "Received: %s\n---\n", i_line);
00166
00167         memset(&i_line, '\0', sizeof(i_line));
00168         memset(&o_line, '\0', sizeof(o_line));
00169
00170         fprintf(stdout, "\n>>");
00171     }
00172 }
00173
00174
00183 void setup_connection(int sock, struct sockaddr* serv_addr, socklen_t serv_len) {
00184     ssize_t count;
00185     char id_request[ID_LEN + 1];
00186     char id_response[ID_LEN];
00187
00188     // Zero the data out
00189     memset(&id_request, '\0', sizeof(id_request));
00190     memset(&id_response, '\0', sizeof(id_response));
00191
00192     // Assign request character
00193     strncpy(id_request, "-1", ID_LEN);
00194
00195     fprintf(stdout, "Sending: %s\n", id_request);
00196
00197     // Send the data to the Server
00198     count = sendto(sock, id_request, ID_LEN, 0, serv_addr, serv_len);
00199
00200     // Check the sent data for errors
00201     if (count < 0) {
00202         perror("Sending to Server Socket Failed\n");
00203         exit(3); // Error Condition 03
00204     }
00205
00206     // Receive a reply from the Server
00207     count = recvfrom(sock, id_response, ID_LEN, 0, serv_addr, &serv_len);
00208
00209     // Check the received data for errors
00210     if (count < 0) {
```

```
00211         perror("Receiving from Server Socket Failed\n");
00212         exit(4); // Error Condition 04
00213     }
00214
00215     fprintf(stdout, "Received: %s\n---\n", id_response);
00216
00217     memset(&id_request, '\0', sizeof(id_request));
00218
00219     play_hangman(sock, (struct sockaddr*) &serv_addr, sizeof(*serv_addr),
id_response);
00220 }
00221
00222
00232 int main(int argc, char* argv[]) {
00233     int udp_sock;
00234     struct sockaddr_in serv_addr;
00235     struct hostent* host_info;
00236     char* server_name;
00237
00238     // Set the Server address to the cmdline option, or LOCALHOST
00239     server_name = (argc == 2) ? argv[1] : "localhost";
00240
00241     // Convert the IP Address to a Human-Readable format
00242     host_info = gethostbyname(server_name);
00243     if (host_info == NULL) {
00244         perror("Unknown Host\n");
00245         exit(1);
00246     }
00247
00248     // Build up the Server Address Structure
00249     memset(&serv_addr, '\0', sizeof(serv_addr));
00250     serv_addr.sin_family = (sa_family_t) host_info->h_addrtype;
00251     memcpy((char*) &serv_addr.sin_addr, host_info->h_addr, (size_t)
host_info->h_length);
00252     serv_addr.sin_port = htons(HANGMAN_UDP_PORT);
00253
00254     // Create the local Socket
00255     udp_sock = socket(AF_INET, SOCK_DGRAM, 0); //0 or IPPROTO_UDP
00256
00257     // Error check The Socket
00258     if (udp_sock < 0) {
00259         perror("Creating Datagram Socket Failed\n");
00260         exit(2);
00261     }
00262
00263     fprintf(stdout, "UDP Client Socket Created\n");
00264
00265     //test_connection(udp_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr));
00266     setup_connection(udp_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr));
00267
00268     // Close the Socket, and exit the program
00269     close(udp_sock);
00270     return (0);
00271 }
```

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 }
```

NG_Asgnmt_01/hangman/src/hangserver_Fork.c File Reference

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

Functions

- void testGameNoZombie ()
- void testGameZombie ()
- void play_hangman (int sock, struct sockaddr *cli_adr, socklen_t cli_len)
- int test (int sock)
- void reaper ()
- int passiveTCP (int service, int qlen)

- int passivesock (int service, const char *transport, int qlen)
- int main ()

Function Documentation

int main ()

Main function of hangserver_tcp_fork(), Builds and deploys server from variables set in Header files using TCp transport protocol for the socket Fork() each incoming connection to run on a seperate thread when received Deploys Signal Function reaper() to check for Orphaned Processes

Returns

Definition at line 196 of file hangserver_Fork.c.

int passivesock (int *service*, const char * *transport*, int *qlen*)

Allocate & bind server socket

Parameters

service	- The Socket/Service to be used
transport	- The Transport Protocol being used by this socket
qlen	- The number of Clients in queue to be connected

Returns

Definition at line 134 of file hangserver_Fork.c.

int passiveTCP (int *service*, int *qlen*)

Call a sokcket build structure based on the protocol provided Currently only calls TCP

Parameters

service	- The Socket/Service to be used
qlen	- The number of Clients in queue to be connected

Returns

Definition at line 122 of file hangserver_Fork.c.

void play_hangman (int *sock*, struct sockaddr * *cli_adr*, socklen_t *cli_len*)

The m,ain hangman game

Parameters

sock	- What Socket the hangman game service will be run through
cli_adr	- The client playing Hangman
cli_len	- The Length of the Client address

Definition at line 45 of file hangserver_Fork.c.

void reaper ()

The Signal Function designed to search for and Terminate Zombie Processses in the main program

Definition at line 96 of file hangserver_Fork.c.

int test (int *sock*)

Definition at line 73 of file hangserver_Fork.c.

void testGameNoZombie ()

Author: Rory Ryan [K00218864] Date: 2020/03/09

This is the Source file for the Client side of the TCP version of the Networked Hangman game. A Function Designed to create a thread that does not become a Zombie for testing purposes

Definition at line 19 of file hangserver_Fork.c.

void testGameZombie ()

A Function Designed to create a thread that does become a Zombie for testing purposes

Definition at line 32 of file hangserver_Fork.c.

hangserver_Fork.c

```
00001
00010 #include "../hdr/hangserver_tcp_fork.h"
00011
00012
00013
00014
00019 void testGameNoZombie(){
00020
00021     printf("\n--Playing Test Connection--\n\n");
00022     while(1){
00023
00024     }
00025     exit(0);
00026 }
00027
00032 void testGameZombie(){
00033
00034     printf("\n--Playing Test Connection--\n\n");
00035
00036     exit(0);
00037 }
00038
00045 void play_hangman(int sock,struct sockaddr* cli_adr,socklen_t cli_len) {
00046     fprintf(stdout, "\n--Playing Hangman--");
00047     fprintf(stdout, "\n--sock: %d", sock);
00048     fprintf(stdout, "\n--cli addr: %d", cli adr->sa family);
00049     fprintf(stdout, "\n--c0li_len: %d", cli_len);
00050     bool endGame = false;
00051     int count;
00052     char i_line[MAX_LEN];
00053
00054     //test Transmission from user
00055     do{
00056         //fprintf(stdout,"---\nAwaiting Data%d...\n\n", sock);
00057
00058         count = recv(sock,i_line,MAX_LEN,0);
00059         i line[count]= '\0';
00060         //Did Message Receive Correctly
00061         if(count<0){
00062             perror("\n---\nReceive Failed!\n---\n");
00063             exit(3);
00064         }
00065
00066         fprintf(stdout,"\nMessage Reads: %s",i_line);
00067
00068
00069         count = sendto(sock,i_line,count,0,cli_adr,cli_len);
```

```
00070
00071     }while(!endGame);
00072 }
00073 int test(int sock){
00074     int count;
00075     char buffer[MAX_LEN];
00076     while(1) {
00077         memset(&buffer, '\0', sizeof(buffer));
00078
00079         printf("\nSocket: %d, Waiting for Message: ",sock);
00080         count = read(sock, buffer, sizeof(buffer));
00081
00082         if(count<=0){
00083             perror("\nRead operation failed");
00084             exit(2);
00085         }
00086         printf("\n\nMessage Reads: %s",buffer);
00087     }
00088 }
00089
00090
00091
00096 void reaper(){
00097
00098     int status;
00099     memset(&status,'\0', sizeof(status));
00100
00101     while (wait3(&status,WNOHANG,(struct rusage *)0)>=0){
00102         if(status < 0){
00103             perror("\n--waitpid() failed No Zombie Found--");
00104         } else if(status == 0){
00105             perror("\n--No Zombies, break--");
00106             break;
00107         }
00108         else{
00109             perror("\n--Zombie terminated--");
00110         }
00111     }
00112
00113 }
00114
00122 int passiveTCP(int service, int qlen){
00123     printf("---\nPassiveTCP");
00124     return passivesock(service,"tcp",qlen);
00125 }
00126
00134 int passivesock(int service,const char *transport,int qlen) {
00135     printf("\n---\nPassiveSock");
00136     //u_short portbase = 0;
00137     //struct servent *pse;        //Pointer to service information entry;
00138     struct protoent *ppe;        //pointer to protocol information entry
00139     struct sockaddr_in sin;     //an internet endpoint address
00140     int s, type;                //socket Descriptor and socket type
00141
00142     memset(&sin, '\0', sizeof(sin));
00143     sin.sin_family = AF_INET;
00144     sin.sin_addr.s_addr = INADDR_ANY;
00145     sin.sin_port = htons(service);
00146     printf("\nTransport: \n");
00147     /* Map service name to port number */
00148     /*if ((pse = getservbyname((const char *)service, transport))) {
00149         sin.sin_port = htons(ntohs((u_short) pse->s_port) + portbase);
00150     }
00151     else if ((sin.sin_port = htons((u_short) atoi(service))) == 0) {
00152         perror("Cant get service entry");
00153     }*/
00154
00155     /* Map protocol name to protocol number */
00156     if ((ppe = getprotobyname(transport)) == 0) {
00157         perror("Cant get protocol entry");
00158     }
00159
00160
00161     /* Use protocol to choose a socket type */
00162     type = SOCK_STREAM;
00163
00164
```

```
00165     /* Allocate a socket */
00166     s = socket(PF_INET, type, ppe->p_proto);
00167     if (s < 0) {
00168         //errexit("can't create socket: %s\n", strerror(errno));
00169         perror("cant create socket");
00170         exit(1);
00171     }
00172     /* Bind the socket */
00173     if (bind(s, (struct sockaddr *) &sin, sizeof(sin)) < 0) {
00174         //errexit("can't bind to %s port: %s\n", service, strerror(errno));
00175         perror("cant bind socket");
00176         exit(1);
00177     }
00178
00179     if (type == SOCK_STREAM && listen(s, qlen) < 0) {
00180         //errexit("can't listen on %s port: %s\n", service,strerror(errno));
00181         perror("cant listen on port");
00182         exit(1);
00183     }
00184     return s;
00185 }
00186
00187
00188
00196 int main() {
00197     printf("\nStarted");
00198     int sock, ssock, fd, client_len, childProcCount, numOfClients;
00199     struct sockaddr_in server, client[MAX_PLAYERS];
00200     int service;
00201
00202     printf("\n---\nZeroing");
00203     memset(&server,'\0', sizeof(server));
00204     memset(&sock,'\0', sizeof(sock));
00205     memset(&ssock,'\0', sizeof(ssock));
00206     memset(&fd,'\0', sizeof(fd));
00207     memset(&client_len,'\0', sizeof(client_len));
00208     memset(&childProcCount,'\0', sizeof(childProcCount));
00209     memset(&numOfClients,'\0',sizeof(numOfClients));
00210     memset(&service,'\0',sizeof(service));
00211
00212     printf("\n---\npost Zeroing\n---\n");
00213     for(int i = 0; i < MAX_PLAYERS; i++) {
00214         memset(&client[i],'\0', sizeof(client[i]));
00215     }
00216     service = (HANGMAN_TCP_FORK_PORT);
00217
00218     printf("Creating Socket \n");
00219     sock = passiveTCP(service,5);
00220     perror("\n---\nCreated Socket");
00221     signal(SIGCHLD, reaper);
00222
00223
00224     if (sock < 0) { // This error checking is the code Stevens wraps in his Socket
Function etc
00225         perror("\n--creating stream socket--");
00226         exit(1);
00227     }
00228
00229     for (int i =0;i < MAX_PLAYERS; i++) {
00230
00231         client len = sizeof(client[0]);
00232         printf("\n---\nAccepting?\n\n");
00233         ssock = accept(sock,(struct sockaddr *)&server,(socklen_t *)&client_len);
00234
00235         printf("Sock: %d produced child ssock: %d",sock,ssock);
00236         if(ssock < 0){
00237             perror("Accept Failed \n---\n");
00238             printf("Failure\n---\n");
00239         }
00240         else{
00241             perror("Accept Succeeded\n---\n");
00242         }
00243
00244
00245         switch(fork()){
00246             case 0: //Child
00247                 printf("\n---\nForked new Child Process\nParent: %d\n"
```

```
00248                         "Created Child Process\n New Child:
%d",getppid(),getpid());
00249                 close(sock);//Child Doesnt Need Listener Port
00250                 test(ssock);
00251                 break;
00252             default:
00253                 printf("Failed to Fork new Child Process\n---\n");
00254                 close(ssock);
00255                 break;
00256         }
00257
00258         close(ssock);
00259     }
00260 }
```

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

```
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[GUESS_LEN];
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, "\nWordWhle: %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 
00210 void test_connection(int sock, struct sockaddr* cli_addr, socklen_t cli_len) {
00211     ssize_t count;
00212     char i_line[MAX_LEN];
00213 
00214     fprintf(stdout, "Testing Connection\n");
00215 
00216     do {
00217         memset(&i_line, '\0', sizeof(i_line));
00218         fprintf(stdout, "---\nAwaiting data on Socket %d...\n\n", sock);
```

```
00219
00220         // Receive data from the Client Socket
00221         count = recvfrom(sock, i_line, MAX LEN, 0, cli_addr, &cli_len);
00222         i_line[count] = '\0';
00223
00224         // Check the received data for errors
00225         if (count < 0) {
00226             perror("Receiving from Client Socket Failed\n");
00227             exit(3); // Error Condition 03
00228         }
00229
00230         // Print the received message to the screen
00231         fprintf(stdout, "Messg Received: %s", i_line);
00232
00233         // Send data to the Client Socket
00234         count = sendto(sock, i_line, MAX LEN, 0, cli_addr, cli_len);
00235
00236         // Check that there were no errors with sending the data
00237         if (count < 0) {
00238             perror("Sending to Client Socket Failed\n");
00239             exit(4); // Error Condition 04
00240         }
00241
00242         // Print confirmation of the send to the screen
00243         fprintf(stdout, "Messg Sent: %s", i_line);
00244     } while (strcmp(i_line, "#quit\0") != 0); // ToDo: Create stop condition that
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_addrs[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_addrs[i], '\0', sizeof(cli_addrs[i]));
00330     }
00331
00332     // Seed the random number generator
00333     srandom((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],
sizeof(cli_addrs[i]), &connected_clients);
00364         connected_clients++;
00365     }
00366
00367     play_hangman(udp_sock, cli_addrs, sizeof(struct sockaddr),
&connected_clients);
00368
00369     // Close the Socket, and exit the program
00370     close(udp_sock);
00371     return (0);
00372 }
```

Index