Systems & Distributed Computing (2802ICT)

Assignment Report – Milestone 2

Matthew Prendergast (s5283740)

CONTENTS

PROBLEM FORMULATION

GENERAL OVERVIEW

The problem requires two programs to be written that can run independently from each other and connect over a remote network. The first – a game server to manage the game, and the second – a game client which players can use to connect to the game server.

USER REQUIREMENTS

The following user requirements were extracted from the clients brief:

1. The game will be called "Numbers". Each game client will be given a turn to increase a communal game total to a predetermined number. The first player to increase the game sum to the winning total will win the game.
2. When it is their turn, a game client can submit their move by typing a number between 1 - 9.
3. For each individual turn, a game client will be given a set number of times it can submit an invalid response (5). If the number of invalid responses exceed this number, the client will be disconnected from the game.
4. When it is their turn, if the game client wishes to quit the game, they can do so by submitting "QUIT". This will disconnect the game client from the game server.
5. If the game client does not respond to the game server within a set amount of time (30), that client will be disconnected from the game.

SOFTWARE REQUIREMENTS

The following software requirements were extracted from the clients brief:

1. The game server and the client server must be able to run independently of each other.
2. The game server and the client server must be able to run on either the same computer, or on separate computers.
3. There will be an open port on the game server which listens for connection requests.
4. When a client connects to the game server, a separate communication channel will begin.
5. The game server will commence with the following command line:
   - *./game_server <Port Number: 4444> <Game Type: numbers> <Game Arguments: 3>*
6. The game client will commence with the following command line:
   - *./game_client <Game Type: numbers> <Server Name: mypc> <Port Number: 4444>*
7. When a client connects to the game server, they will be greeted with a welcome message.
8. The game will only begin when the required number of clients have connected to the server.
9. No new players can join the game once it has begun.
10. The game server will iterate through the connected clients according to the game rules, and in the order that they have joined the game.
11. If an invalid message protocol is received by the game server from a client, this will indicate a faulty client. As such, said client will be disconnected from the game.
12. If the number of connected clients drops below a required level (2), the game will be terminated, and the remaining client will be deemed the winner.
13. When the game has completed – by way of a standard winner, or game termination – the winner and all remaining losers will be notified whether they won or lost.

SOFTWARE DESIGN

HIGH-LEVEL DESIGN

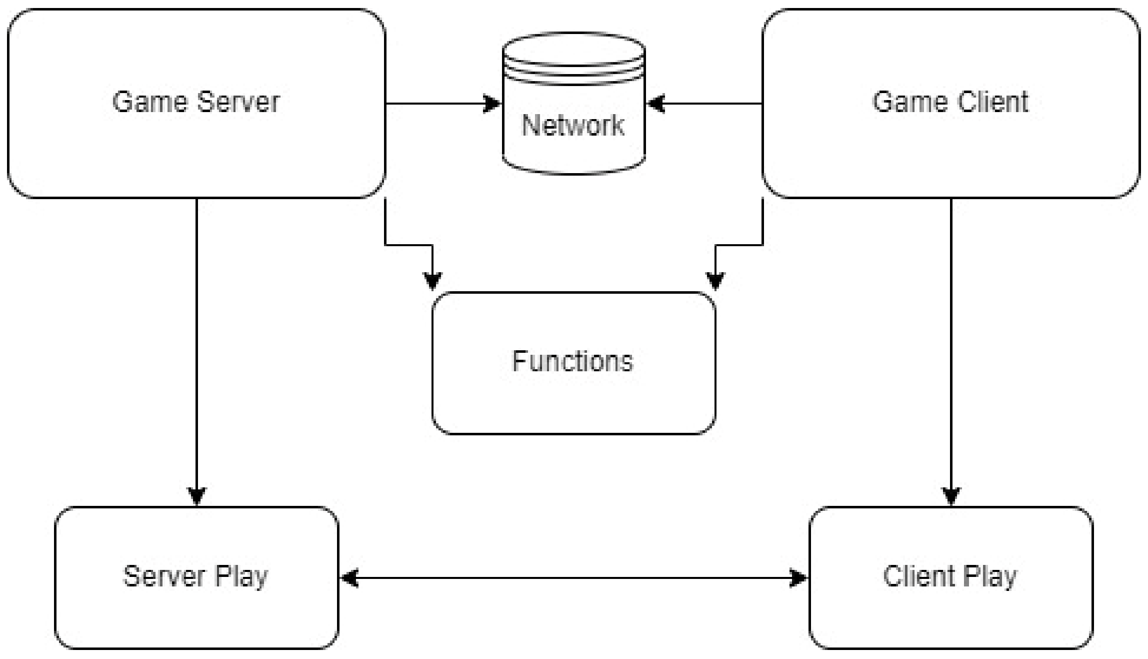

Figure 1.0 – Functional Flow Diagram

PROGRAM DATA

BOOL

connected	Used to run game client loop and break on disconnection.
playing	Used to run game server game loop and break on termination.

BOOL*

connections	An array which is used to monitor the connection status of players.

CHAR*

client_buffer	Used to store client messages to the game server.
game_type	Used to store the name of the game entered from the command line.
hostname	Used to store the name of the game server computer on the network.
server_message	Used to store game server messages to the clients.

INT

client_size	Used to store the sizeof(client) variable.
client_socket	Used to store the client socket ID.
connected_players	Used to monitor the current count of clients connected to the game server.
game_sum	Used to monitor the current game sum, as entered by the players.
move_value	Used to store the current move of a client when converted to an integer.
player_count	Used to store the number of players required for a game to begin, as entered on game server the command line.
port	Used to store the port number for the connection, as entered on the game server and game client command lines.
result	Used to store the return value of functions used in the programs.
server_socket	Used to store the server socket ID.

INT*

cs_ptr	A pointer which is used to point to the client socket ID that the game server is awaiting a response from.

#DEFINE

BACKLOG	Used to set the number of clients that can queue for a connection.
BUFFER	Used to set the buffer size for game server and client messages.
HOST_SIZE	Used to set the buffer size for the hostname variable.
MAX_ERRORS	Used to set the total number of consecutive errors a game client can make.
MINIMUM_PLAYERS	Used to set the minimum number of players to keep a game running.
TIMEOUT	Used to set the number of seconds before a game client triggers a timeout.
WINNING_TOTAL	Used to set the games winning total which is required to win the game.

ENUM

message_types	Enumeration of the message types that can be sent by the server or clients.

STRUCT ADDRINFO

address_struct	Structure required to resolve the IP address of the game server from the hostname provided on the command line.
***server_info**	Structure required to resolve the IP address of the game server from the hostname provided on the command line.

STRUCT SOCKADDR_IN

client	Structure required to map the socket connection details of the game client.
server	Structure required to map the socket connection details of the game server.

PROGRAM FUNCTIONS

main () – Game Server

INPUT: int argc, char *argv[]

RETURN: None.

When initialised, the main() function of the game server will find it's hostname to provide to game clients, and then call the socket(), bind() and listen() function to allow for connections from game clients. Once the required number of connections have been made, the game server will call the server_play() function to initiate a game.

MAIN () – GAME CLIENT

INPUT: int argc, char *argv[]

RETURN: None.

When initialised, the main() function of the game client will begin by resolving the IP address of the game server, using the hostname it has been provided. It will then proceed to call the socket() function and make a connection() with the game server. When a connection has been made with the game server, the game client will call the client_play() function and wait for a game to be initiated.

ALARM_HANDLER()

INPUT: int signum

RETURN: None.

When initialised by the alarm() trigger, the alarm_handler() will send a QUIT request from the current game client that has triggered the alarm(), to the game server. The function will then proceed to terminate the current game client.

CLIENT_PLAY()

INPUT: int client_socket

RETURN: None.

When initialised, the client_play() function will wait until it has received confirmation from the game server that a game has begun. Once a game has begun, the function will continuously loop and wait to receive messages from the game server, at which point it will respond accordingly, and allow the player to interact with the game.

ARGUMENT_COUNT()

INPUT: char* user_input

RETURN: int count;

When initialised, the argument_count() function will take a string input and count the number of arguments it contains. The count is made using " " characters between each word in the string.

CLIENT_SEND()

INPUT: int message_types, char* message, int client_socket

RETURN: None.

When initialised, the client_send() function will initiate the sending of messages from a game client to the game server, based on the arguments given to the function by the game client.

IS_NUMBER()

INPUT: char *text_message

RETURN: true / false

When initialised, the is_number() function will take a string input and determine if it is an integer.

PRINT_HOSTNAME()

INPUT: None.

RETURN: None.

When initialised, the print_hostname() function will determine the hostname of the game server computer and print it to the game server shell.

SERVER_SEND()

INPUT: int message_types, char* message, int client_socket, int i

RETURN: None.

When initialised, the server_send() function will initiate the sending of messages from the game server to a game client, based on the arguments given to the function by the game server.

SERVER_PLAY()

INPUT: int* client_socket, int player_count

RETURN: None.

When initialised, the server_play() function will continuously loop through the connected game clients, sending instructions and waiting to receive responses. When responses are received from a game client, the game server will respond according to the game rules and requirements.

FILE COMPILATION

To compile these files, please follow the instructions below, using GCC in the program directory.

COMPLIATION INSTRUCTIONS: make all

TEST RESULTS

Requirement Acceptance Testing

Software Requirement	**Test Description**	**Implemented** • **Full** • **Partial** • **None**	**Result** • **Pass** • **Fail**	**Comments** **(If Test Failed)**
1	The game server and the client server must be able to run independently of each other.	Full	Pass	None
2	The game server and the client server must be able to run on either the same computer, or on separate computers.	Full	Pass	Connected over the same network.
3	There will be an open port on the game server which listens for connection requests.	Full	Pass	None
4	When a client connects to the game server, a separate communication channel will begin.	Full	Pass	None
5	When a client connects to the game server, they will be greeted with a welcome message.	Full	Pass	None
6	The game will only begin when the required number of clients have connected to the server.	Full	Pass	None
7	The game server will iterate through the connected clients according to the game rules, and in the order that they have joined the game.	Full	Pass	None
8	If an invalid message protocol is received by the game server from a client, this will indicate a faulty client. As such, said client will be disconnected from the game.	Full	N/A	Implemented, however, unable to replicate a faulty message protocol.

Software Requirement	**Test Description**	**Implemented** • **Full** • **Partial** • **None**	**Result** • **Pass** • **Fail**	**Comments** **(If Test Failed)**
9	No new players can join the game once it has begun.	Full	Pass	None
10	If the number of connected clients drops below a required level (2), the game will be terminated, and the remaining client will be deemed the winner.	Full	Pass	None
11	When the game has completed – by way of a standard winner, or game termination – the winner and all remaining losers will be notified whether they won or lost.	Full	Pass	None
12	When it is their turn, a game client can submit their move by typing a number between 1 - 9.	Full	Pass	None
13	For each individual turn, a game client will be given a set number of times it can submit an invalid response (5). If the number of invalid responses exceed this number, the client will be disconnected from the game.	Full	Pass	None
14	When it is their turn, if the game client wishes to quit the game, they can do so by submitting "QUIT". This will disconnect the game client from the game server.	Full	Pass	None
15	If the game client does not respond to the game server within a set amount of time (30), that client will be disconnected from the game.	Full	Pass	None

DETAILED SOFTWARE TESTING

No	**Test**	**Expected Results**	**Actual Results**
1.0	**Run Program**		
1.1	Run Game Server ./game_server 8080 numbers 3	Hostname: LAPTOP-L5VKF6H5 Socket initialised... Bind completed... Waiting for incoming connections...	As expected.
1.2	Run Game Client ./game_client numbers LAPTOP-L5VKF6H5 8080	Socket initialised... Connection established... Player 1 - Welcome to the game! Please wait for all players to connect...	As expected.
2.0	**Connection**		
2.1	Connection – Same Network, Same Computer	As per 1.1 and 1.2	As expected.
2.2	Connection – Same Network, Different Computers	As per 1.1 and 1.2	As expected.
2.3	Connection – Correct Hostname	As per 1.1 and 1.2	As expected.
2.4	Connection – Correct Port	As per 1.1 and 1.2	As expected.
2.5	Connection – Incorrect Hostname	ERROR: Unable to locate the server.	As expected.
2.6	Connection – Incorrect Port	ERROR: Unable to locate the server.	As expected.
2.7	Connection – Separate Channels	Separate Identities PLAYER 1: Connection established… PLAYER 2: Connection established...	As expected.

No	Test	Expected Results	Actual Results
2.8	Connection – Welcome Message	Player 1 - Welcome to the game! Please wait for all players to connect...	As expected.
2.9	Connection – No New Players (Game Started)	ERROR: Unable to locate the server.	As expected.
3.0	**Game Play**		
3.1	Commence Game – Only when all required players connect.	ALL PLAYERS HAVE CONNECTED - THE GAME WILL NOW BEGIN SHORTLY PLEASE WAIT FOR YOUR TURN...	As expected.
3.2	Game Iteration – Iterate through players in the order they connect, when it is their turn.	------TYPE 'QUIT' TO LEAVE THE GAME------ It's your turn... The Game Total is 0. Enter a number:	As expected.
3.3	Player Disconnection	Please Wait - Disconnecting you now...	As expected.
3.3.1	Remaining Player Deemed Winner	You Won! Please Wait - Disconnecting you now...	As expected.
3.4	Game Completion – Player Won	You Won! Please Wait - Disconnecting you now...	As expected.
3.5	Game Completion – Player Lost	Sorry, You Lost... Please Wait - Disconnecting you now...	As expected.
4.0	**Errors**		
4.1	Errors – Invalid Move (Non-Digit: X)	ERROR: Usage - <Number: (1 to 9)>	As expected.

No	Test	Expected Results	Actual Results
4.2	Errors – Invalid Move (Outside Range: 50)	ERROR: Usage - <Number: (1 to 9)>	As expected.
4.3	Errors – Invalid Move (Too Many Arguments: 5 5)	ERROR: Usage - <Number: (1 to 9)>	As expected.
4.4	Errors – Max Errors Exceeded	ERROR: Usage - <Number: (1 to 9)> Please Wait - Disconnecting you now...	As expected.
4.5	Errors – Faulty Client	ERROR: Protocol Infringement (Faulty Client) Please Wait - Disconnecting you now...	Unable to replicate a faulty client.
5.0	**Game Termination**		
5.1	Game Termination – Client Quit	Please Wait - Disconnecting you now...	As expected.
5.2	Game Termination – Client Timeout	Client Timeout - Disconnecting you now...	As expected.