2803ICT – Assignment 1

Himath Ratnayake

Contents

1. Problem Statement

The goal of this project was to implement a basic remote execution system consisting of a client and server. These two independent programs communicate through a network to run a game of “Numbers”, which is a multiplayer game in which clients will connect to the server and take turns choosing numbers until a player attains a total score of 30 or more.

The game client and server programs must be run in the command line with relevant parameters such as port and number of players, and both programs communicate with each other on a pre-existing protocol.

2. User Requirements

The following outlines the user requirements for the program:

1. When setting up the server, the user must specify port number, game type and the number of players
2. When running the client, each user must specify the game type, server name and the port number
3. Each client enters an input once prompted by the server, and must enter this input within 30 seconds before they are disconnected
4. Each user client is able to run a “move” command which gives the user their move for the turn or “quit” command which will allow the user to leave the game
5. For the move command, the user must only enter a number between 1 and 9
6. If the user enters a number greater than 9 or less than 1 five times in a row, they will be disconnected
7. If the user triggers a protocol infringement, such as entering an invalid message (neither “quit” or a number between 1 and 9), then they will be immediately disconnected

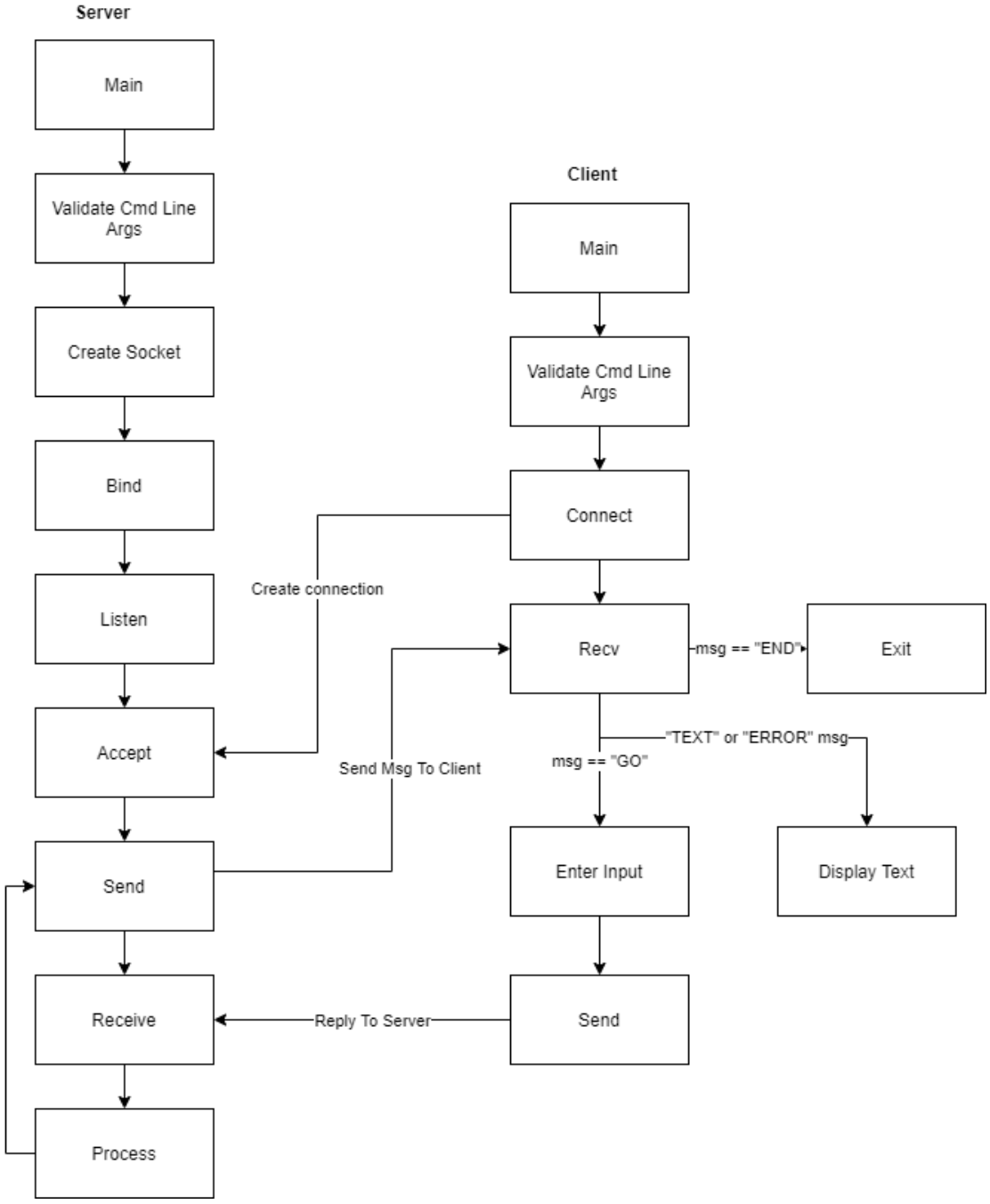
3. Software Requirements

The following outlines the software requirements for the program:

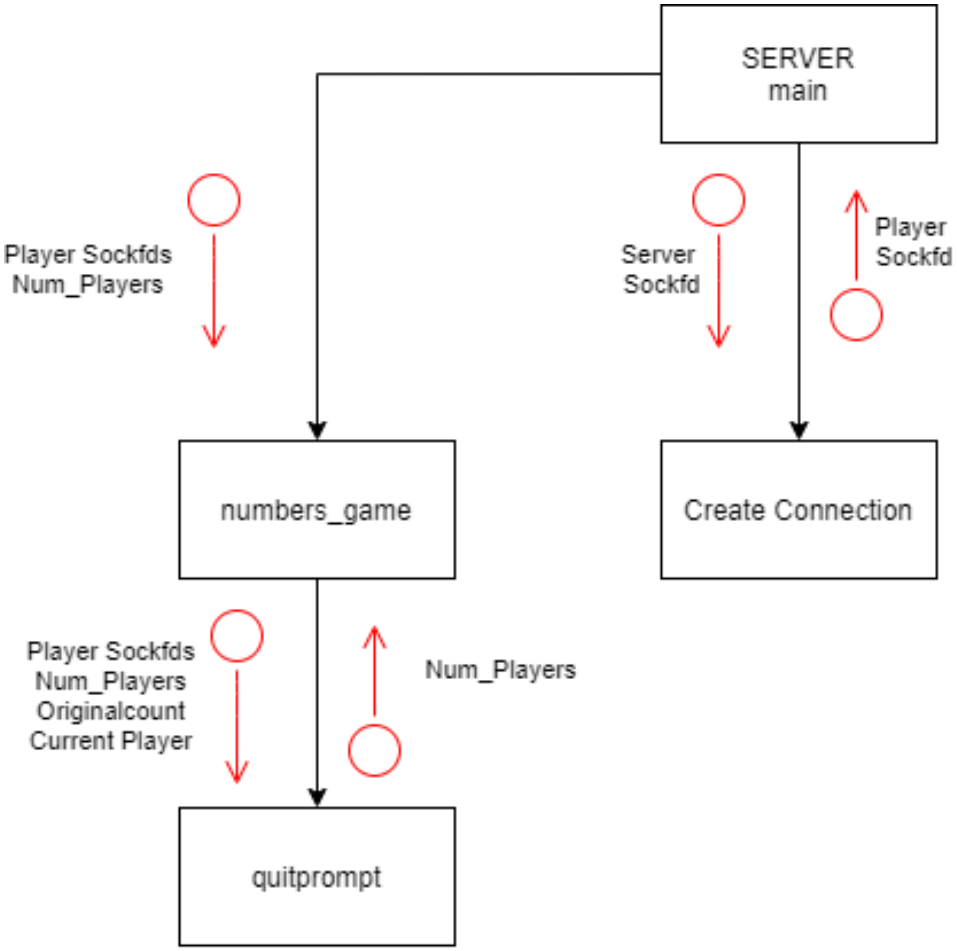
1. Both the client and server programs are run through the command line, with key arguments such as the port number, which must be same for both client and server
2. When a client joins the game, the server sends a “Welcome to the game” message to them
3. The server only starts once the required number of clients join, as specified in the command line arguments when starting the server
4. The server program allows for 4 commands to be send to the client program: TEXT, GO, END, and ERROR
5. Before each “GO” prompt, the server must send the client the current sum of the game.
6. The client program will only accept input once the server program sends a “GO” message
7. The client program must not show the 4 commands to the client’s terminal. For example, for a TEXT message, the word “TEXT” is not shown to the client.
8. The client program can enter the word “quit” to disconnect itself, which will send a “QUIT” message to the server
9. The client program can enter a number between 1 and 9 after a “GO” prompt from the server. A number less than or greater than this range will result in a game error, and the user will be asked to re-input a valid number.
10. For each move the client does, the client will send back the input number with the word “MOVE” before the given number
11. The server will disconnect any faulty clients. For example
    a. Any client that does 5 game errors in a row
    b. Any client that does one protocol error (any non-numeric input that isn’t “quit”)
    c. Any client that takes longer than 30 seconds to respond to a “GO” prompt
12. If there is only one player left, the server program will stop the game as it can no longer continue, and the remaining player is declared the winner.
13. If a client reaches the score of 30 or above first, the server will then send a message announcing the game is over, as well as a “you lost” to the losing clients and “you won” to the winning client
14. Any time a client or the server terminates, all resources must be closed successfully.

4. Software Design

High Level Design – Logical Block Diagram



Structure Chart



Software Functions (Server)

numbers_game	
Description	The main driver function that initiates the numbers game loop.
Information	The two input parameters of this function are the players_fd socket (passed in by reference) and the num_players variable. Both are integers. The num_players variable keeps track of the amount of players in the game, as specified in the command line and gets decremented every time a player drops out of connection. The main function loops through the num_players, sending each player various commands and receiving inputs back from them, as detailed in the pseudocode below. It is also the parent function of the quitprompt function, which is responsible for disconnecting players.
Return value	The return type of this function is void

Create_connection	
Description	This function is responsible for accepting and creating the connection with a client.
Information	This function has one input parameter, which is the server's socket descriptor. This is used within the accept() library function. A connector's address information is stored in a struct. An int new_fd is initalised for new connections, and the accept() library function is used to establish the connection.
Return value	The return type of this function is an int, and is a socket descriptor of the incoming connection.

quitprompt	
Description	This function is used for terminating connections with players.
Information	This function has 4 input paramters – the number of players, the current_player, the original player count when the game started and the socket descriptor array players_fd. It sends the current player an "END" message, closes the player socket of the player who is being removed, and then decrements the number of players field. As the player sockets are stored in an array, the socket of the current player is set as "0". In the main function, this then allows for the code to skip over iterations that involve this player, who has already quit. It then checks to see if the number of players now falls below two – if this is the case the game cannot be continue and a text message and "END" function is sent to the remaining player, after which the program exits.
Return value	The return type of this function is int – it will return the number of players that are present after one leaves back to the numbers_game function, so that it can be updated.

Main (server)	
Description	The driver function for the game server.
Information	First, the number of command line arguments is verified. The num_players int is initialised from the command line arguments, as well as the port number. The player socket descriptors are stored within an int array that is initialised to the size of num_players. The server socket descriptor is also initialised, being an integer. Then, a socket is created with socket(). A socket address is then prepared, which is stored in a sockaddr_in struct. An address is binded to the socket using bind(), and listen() is used to look for incoming connections. They are then accepted using the create_connection function, which is called “num_players” amount of times through a for loop with the server_sockfd socket descriptor being passed into said function. Then, the numbers_game function is invoked to initiate the numbers game loop once enough players are found.
Return value	The main function returns an int (1 for premature exit or 0 for completion of program)

Main (client)	
Description	The driver function for the client server.
Information	Similar to the main function, the command line arugments are verified. A char array of the length BUFFERLEN as defined by the program is also initialised to store incoming messages. A sockaddr_in struct is then created and initialised with the port, IP address, and AF_INET family. The client socket descriptor, which is an int is also initialised using socket(). Then a connection to the server program is established with connect(). In a while loop, the client receives a message, processes it’s contents and sends a message back to the server as per the pseudocode detailed below.
Return value	The main function returns an int (1 for premature exit or 0 for completion of program)

Detailed Design

SERVER main function

Check to ensure enough arguments are given
Create TCP socket
Create Socket Address
Bind address to socket
Listen for incoming connections
Iterate through all players
Accept incoming connections
Start Game Loop
Close the server socket
Run the numbers game
Iterate through all players
    Close player sockets
Finish

CLIENT main function

Create individual client socket
Verify client socket works
S→C: Game is starting
While:
    S→C: msg
    If msg has TEXT:
        Print text minus the "TEXT" (this accounts for ERROR commands too)
    If msg is "GO":
        Enter Input
        If input is quit
            C→S: "QUIT"
        Else
            Check to make sure input is valid
            Msg = Strcat MOVE + Input
            C→S: Msg
    If msg is END:
        The server should terminate with faulty client
        Exit

SERVER quitprompt function

Send player in question an "END" message
Close player socket descriptor
Set player's socket descriptor to 0
Decrement Number of Players
If the number of players is less than 2
    Iterate through sockfds
        For player with valid sockfd still (not 0)
            Send them a message saying they won, and game has ended
            Close their sockfd
        Exit Program
Return num_players

SERVER numbers_game function

Print that the game is starting on server
Iterate through all players
    S→C: TEXT Game is starting
In While Loop:
Iterate through all players. At each iteration, "C" is a new player:
    If current client socket descriptor invalid:
        Skip iteration
    Print "Round _, _'s Turn"
    S→C: TEXT It's your turn! Sum is __. Enter a Number:
    S→C: GO
    Start 30 sec timer.
    If >30 secs passed:
        S→C: END
        Go to new player
    C→S: Input
    If str has the word MOVE:
        Tokenize and get input number
        Errors = 0
        While input number is less than 0, greater than 10:
            Error += 1
            If Error > 5:
                S→C: Quitprompt
                Go to new player
                Continue
            S→C: TEXT ERROR Bad input.
            S→C: GO
            C←S: Input
            Tokenise and get input number
        Total += chosen number
        If total is now greater than 30:
            Print the player who won the game
            Iterate through all players:
                If current player is winner
                    S→C: TEXT You won!
                    S→C: END
                Else
                    S→C: You lost!
                    S→C: END
            Return (FINISHED)
        If total is less than 30:
            S→C: TEXT Number Recorded!
    Otherwise Invalid Client MSG or QUIT (both have same outcome)
            S→C: Quit

5. Requirement Acceptance Tests

Software Requirement No	Test	Implemented (Full /Partial/ None)	Test Results (Pass/ Fail)	Comments (for partial implementation or failed test results)
1	Client program runs on command line, and connects to the server correctly given the correct arguments	Full	Pass	
2	Server program runs on command line, and can receive clients successfully given the correct arguments	Full	Pass	
3	When not enough command line parameters are given the server or client program will print an error message	Full	Pass	
4	When a client joins the game, the server sends a “Welcome to the game” message to them	Full	Pass	
5	The server only starts once the required number of clients join, as specified in the command line arguments when starting the server	Full	Pass	
6	The server program can send “TEXT” commands with a given message to the client	Full	Pass	
7	Before each “GO” prompt, the server sends the client the current sum of the game.	Full	Pass	
8	The client program only accepts input once the server prompts client and sends a “GO” message	Full	Pass	
9	The client program must not show the 4 commands to the client’s terminal. For example, for a TEXT message, the word “TEXT” is not shown to the client.	Full	Pass	
10	Client program enters the word “quit” to disconnect itself, a “QUIT” message is sent to the server, and the server responds with an “END” message.	Full	Pass	
11	The client program can enter a number between 1 and 9 after a “GO” prompt from the server. A number less than or greater than this range will result in a game error.	Full	Pass	

Software Requirement No	Test	Implemented (Full /Partial/ None)	Test Results (Pass/ Fail)	Comments (for partial implementation or failed test results)
12	When a game error occurs, the program will send the client a “TEXT ERROR” command with the error message	Full	Pass	
13	For each move the client does, the client will send back the input number with the word “MOVE” before the given number	Full	Pass	
14	Any client that does 5 game errors will be disconnected	Full	Pass	
15	Any client that does one protocol error (any non-numeric input that isn’t “quit”) will be disconnected	Full	Pass	
16	Any client that takes longer than 30 seconds to respond to a “GO” prompt will be disconnected	Full	Pass	
17	If there is only one player left, the server program will stop the game as it can no longer continue, and the remaining player is sent a message saying they won.	Full	Pass	
18	If a client reaches the score of 30 or above first, the server will then send a message announcing the game is over, as well as a “you lost” to the losing clients and “you won” to the winning client	Full	Pass	
19	Any time a client or the server terminates, all resources must be closed successfully.	Full	Pass	

6. Detailed Software Testing

No	Test	Expected Results	Actual Results
1.0	**Initiation Phase**		
	Run from command line for both:		
1.1	> The expected arguments with format	1.1: The server and client are created successfully	All As Expected
1.2	> Without enough arguments (<3)	1.2: Server or client terminates immediately	
1.3	> With invalid port number	1.3: Prints out warning message and terminates	
2.0	**Join Phase**		
	After initiation, observe on client and server terminals:		All As Expected
2.1	> Server waits for clients to join	2.1: Prints to stdout on server console that it's awaiting connections	
2.2	> Server sends welcome message to new clients	2.2: "Welcome to the game" printed on client screen as soon as they join	
2.3	> Server moves to the next stage after the required player number has joined	2.3: When the playercount arg is reached the server prints message showing that to stdout	
3.0	**Play Phase**		
	On Client Terminal, observe that		
3.1	> Text messages are printed out without the "TEXT" before it	3.1: Look in client terminal to see that the word TEXT isn't printed	
3.2	> Current sum is printed out to the screen before being asked for input	3.2: Current sum printed is the correct sum	

No	Test	Expected Results	Actual Results
3.3: 3.4: 3.5 3.6 3.7 3.8	Inputs: > Enter a number greater than 9 > Enter a number between 1 and 9 > Enter an invalid non-numeric phrase (e.g. HAV 123) > Enter a number greater than 9 five times in a row > Wait 30 seconds when input prompt is given > Enter the word “quit”	3.3: Error message and client must reenter an input 3.4: Successfully added number to current sum and moved on to next player 3.5-3.8: Terminates the current client. If current player number less than two, the remaining player gets winner’s message.	All As Expected
4.0	**Termination Phase**		
4.1	> Play game until sum of 30 is reached. Observe client and server screens for win/lose message.	> When sum of 30 is reached the winner and losers get notified from server and all server and client resources closed succesfully without error	All As Expected

7. User Instructions

For The Server:
Compile the program and run it in the command line with the following arguments, following the syntax below.

- Compiling Syntax: gcc game_server.c -o game_server
- Arguments: Game_server <port number> <game type> <player number>
- Execution Syntax Example: ./game_server 4444 numbers 2

The server will then start assuming all parameters are correct, and will print out information in response to the behaviors of the client and progression of the numbers game.

For The Client:
In a different device or another terminal, compile the client program, and run it in the command line, following the syntax below.

- Compiling Syntax: gcc game_client.c -o game_client
- Arguments: game_client <game type> <server name> <port number>
- Execution Syntax Example: game_client numbers mypc 4444

NOTE: Ensure that the server is set up beforehand, and that the port number specified is the same as the one used by the server.
Once connected, the user must wait for the specified number of players needed for the game to begin to join.
Once the sufficient amount of players is connected, the server will prompt the user when it is their turn.
The user can then input “quit” to leave the game or a number between 1 and 9. A lack of input for more than 30 seconds will result in the termination of the current client
If the current client, or another client reaches the goal total score greater than 30 first, or there are no longer enough players to continue the game, the client will be disconnected, ending the program.

Compiler Notes:
This program was coded in Windows subsystem for linux, and verified to be working in Cygwin64.