2803ICT – Assignment 1

<Petar Vidakovic s5240487>

Contents

1. Problem Statement 1

2. User Requirements 1

3. Software Requirements 2

4. Software Design 2

5. Requirement Acceptance Tests 4

6. Detailed Software Testing 5

7. User Instructions 6

## Problem Statement

In this assignment students were tasked to create a client server connection. To host a game over that network with a list of rules, the protocol, which dictates how clients can communicate with the server and how the server is to handle the client messages. The connection is TCP based run over sockets. The protocol details 4 states, Initiate, Join, Play, and Game over announcing results.

## User Requirements

The user who I assume is playing all the games to test. Will first open the server and lunch it with the specified values, port number, game type and players(number). The user will then open the clients and connect them into the game.

Once the game starts the user can only enter on the client that received the Go message. The user should then enter a state of play entering single digits on their quest to obtain 30 points. The user can quit in response to a go message instead of their turn. That client will shut down and exit out. The user can expect the game to continue with the quit player missing. If all players quit and there’s just 1 left, they are the winner, and the server will exit.

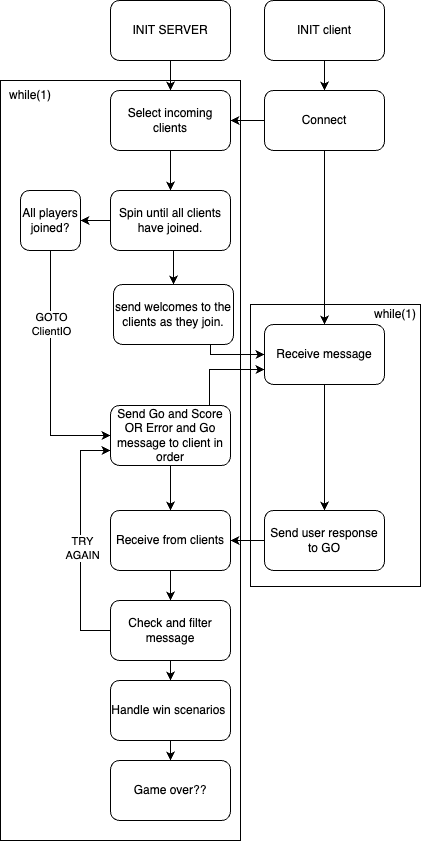
## Software Requirements

Requirements:

1. Establish client server programs.
2. Welcome message sent to clients as they join
3. Server Sends Go and score in the order players join
4. Client can only respond to Go message
5. Client timed out after 30 seconds afk
6. Last player declared winner
7. Quit message exits client
8. Handle game errors
9. Kick client after 5 errors
10. Win loss messages sent and ENDs

## Software Design

### High Level Design – Logical Block Diagram.



List of all functions in the software.

|  |
| --- |
| Function: CreateServerSocket()  Description: This is a custom function which bundles together the creation of a server and initialize a host. This socket can host multiple clients and be reused.  Input params: None  Side effects: None  Returns: An Int which corresponds to the socket on which the host is listening. |

|  |
| --- |
| Function: Socket(AF\_INET, SOCK\_STREAM, 0)  Description: creates a socket file descriptor and acts as an end point.  Input params: AF\_INET, SOCK\_STREAM  Side effects: None  Returns: A int related to the socket file descriptor. |

|  |
| --- |
| Function: setSocketopt  Description: this function allows for socket options to be altered. There are many options for the input params such as allowing for non-blocking or how I have it here reusable address  Input params: The socked fd, option for socket level, reuse local address, none  Side effects: None  Returns: return 0 on successful assignment and -1 on fail. |

|  |
| --- |
| Function: memset  Description: Memory macro that sets all element of a memory construct to another.  Input params: target memory area, amount to set to (0)  Side effects: None  Returns: returns its first argument with the updated elements. |

|  |
| --- |
| Function: Bind  Description: Assigns a name to an unnamed socket, this is connecting the struct sockaddr and the host allowing for clients to connect.  Input params: socket, the memory address that we are binding to, size  Side effects: None  Returns: Returns -1 for unsuccessful call and 0 for success. |

|  |
| --- |
| Function: Listen  Description: Take our host server socket and set it to listen for new connections.  Input params: socket, backlog which refers to how many clients can be waiting to join.  Side effects: None  Returns: -1 on fail, 0 on success |

|  |
| --- |
| Function: Accept  Description: Accepts a new socket trying to join the server.  Input params: socket, memory location from a connecting client, size of that  Side effects:  Returns: -1 on fail, 0 on success. |

|  |
| --- |
| Function: Connect – seen in client  Description: This is an attempt to make a connection with another socket.  Input params: socket, memory location from a connecting client, size of that  Side effects:  Returns: -1 on fail, 0 on success. |

|  |
| --- |
| Function: snprintf  Description: A macro in the same family as printf, allows to transform data in a memory container using pre-allocated variable to inject the string.  Input params: destination memory, the size, the edited string using % operator, % argument  Side effects: creates a new memory construct in destination memory.  Returns: the first argument with the changes. |

|  |
| --- |
| Function: Select and FD macro set  Description: select  FD\_CLR(fd, fd\_set \*fdset)   * This macro clears the current fd set, arg 1, of a specific fd, arg 2.   FD\_COPY(fd\_set \*fdset\_orig, fd\_set \*fdset\_copy);   * This macro copy’s one fd set to another   FD\_ISSET(fd, fd\_set \*fdset);   * This is a check if the fd is in the fd set   FD\_SET(fd, fd\_set \*fdset);   * Assgin a fd to be apart of the fd set with this macro   FD\_ZERO(fd\_set \*fdset);   * Zero out entire fd set specified   Input params: Select its self takes in the size of the fd set we’re working with as arg 1,  The memory location of the fd set stated in our declaration, select has some more flags here for advanced use, and the last argument is a specified time out.  Side effects: Reads the fd set and waits for incoming connections of size fd set  Returns: total number of sockets ready to be used. |

|  |
| --- |
| Function: send  Description: Sends bytes across a connected network.  Input params: A socket to send to, the message or bytes, the size  Side effects:  Returns: a -1 if fail, 0 on success |

|  |
| --- |
| Function: recv  Description: Receives information across connected sockets  Input params: target socket to recv from, the memory the bytes will be stored in, its size  Side effects:  Returns: Returns greater than 0 on successful receive, 0 on connection lost, -1 on connection read error |

|  |
| --- |
| Function: bzero  Description: zeros out a memory construct  Input params: destination memory location, size of dest  Side effects: None  Returns: arg 1 |

### List of all data structures in the software. (eg linked lists, trees, arrays etc)

In order as they appear in the code

|  |  |
| --- | --- |
| Name and type: | fd\_set Fds |
| Type: | Macro (SET) |
| Description: | Fds is the fd set I use in the program to watch the incoming connection for when they would like to join the server. This data structure is a part of select and is used for asynchronous I/O. |
| Data members: | File descriptors that join the server will be listed inside this set |
| Functions: |  |

|  |  |
| --- | --- |
| Name and type: | Buf and score |
| Type: | Char array |
| Description: | Buf and score are use as buffers for transferring data in and out of locations when live data is obtained via the users. Score is used specifically to update the sum relative to all the user’s inputs and send that across the socket |
| Data members: | Buf will hold user input and score will hold a modified string that updates. |
| Functions: |  |

|  |  |
| --- | --- |
| Name and type: | Connections |
| Type: | Int array |
| Description: | Holds the socket file descriptors, I use this array heavily throughout the program. As it is the core storage location of the active and non-active sockets. Initially this array is set to all -1 indicating not active. |
| Data members: | The server socket is on connection[0], new connections join the array at index 1. This is important because we don’t scan the host socket for io. |
| Functions: | Fd\_isset, Chcekmessage, send, recv |

|  |  |
| --- | --- |
| Name and type: | Input – client.c |
| Type: | Char array |
| Description: | The input array for the client programs. Used to filter data in and out of send/recv. If the message is vaild appended a MOVE string to the first 4 chars |
| Data members: | Characters from user |
| Functions: | Recv, strcmp, send, bzero, isdigit |

### Detailed Design – Pseudocode for all non-standard and non-trivial algorithms that operate on data structures

I’ll explain here a bit about my send and receive pattern. We first enter all the active client into the fd\_set. We then use select to poll for incoming connections. Select will return the total number of file descriptors in use. As we accept our clients, we send welcome messages as they join. Once no new client can join the game this section of code is not come to again. And to get path select blocking due to no new incoming connection I use a goto statement to inject us into the else position of the select. I obviously should have used a different way, but I thought initially I could use select to timeout clients however that isn’t how selects time out works. Once into the client communication exchange we start in the order that the sockets joined. The go is sent, and the first player responds. The message goes through a filter to reform various operation on the data the user has given the server.

CheckMessage

Buf[end] = null

Display message

If buf 🡪 “quit” exit player

If buf 🡪 isdigit += sum

else

Anything else typed 🡪 ERROR++

Send ERROR && Send GO

This psudo code can describe the basic structure of the filter. With this model I don’t differentiate between errors. The server only accepts a digit or quit anything else will spawn an error.

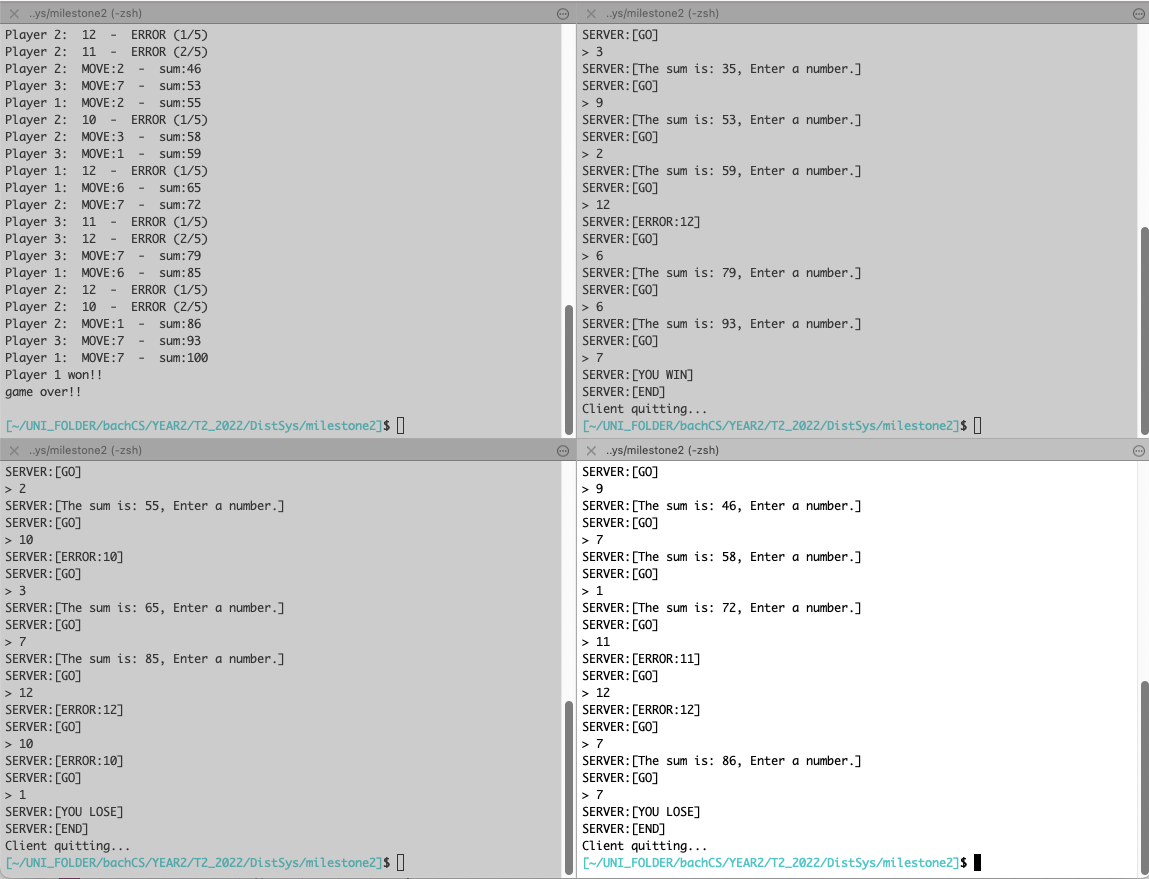
## Requirement Acceptance Tests

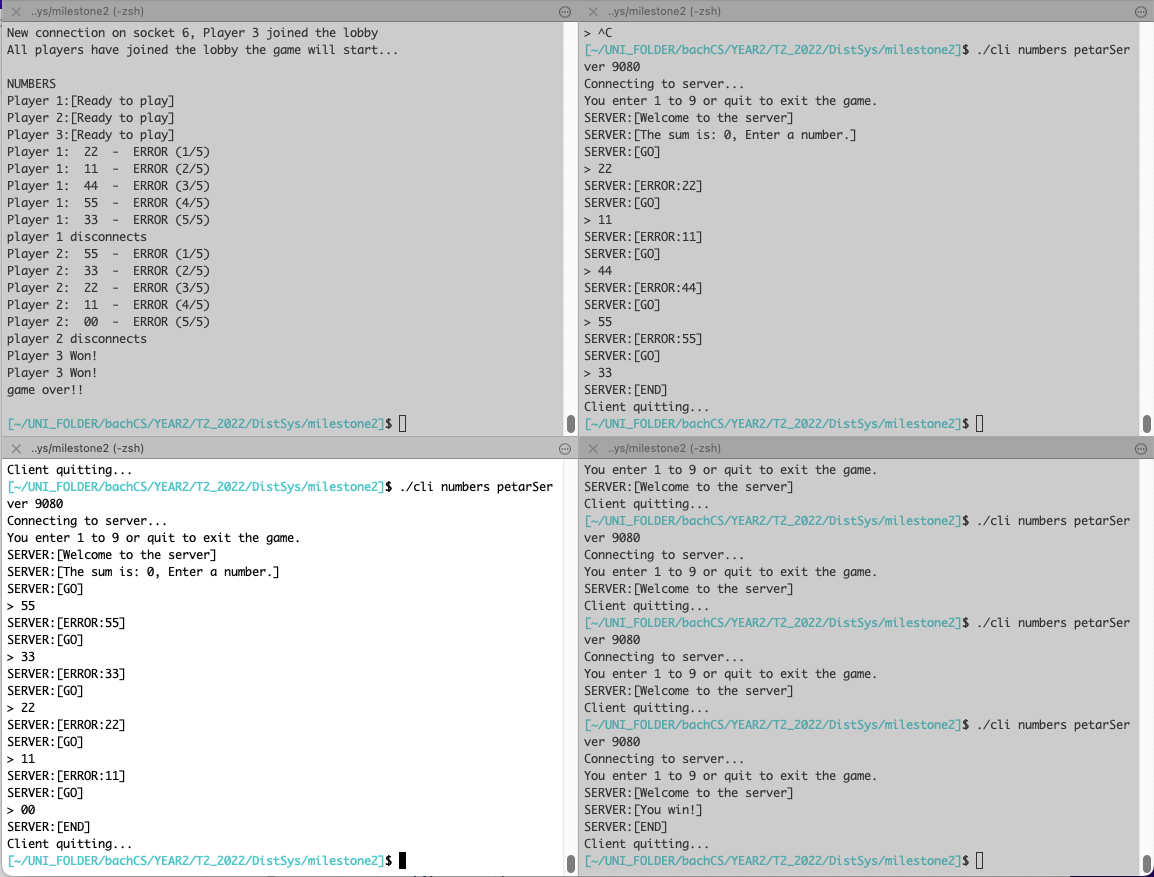
| Software  Requirement No | Test | Implemented (Full /Partial/ None) | Test Results (Pass/ Fail) | Comments (for partial implementation or failed test results) |
| --- | --- | --- | --- | --- |
| 1 | Establish client server programs. | Full | Pass |  |
| 2 | Welcome message sent to clients as they join | Full | Pass |  |
| 3 | Server follows protocol | Full | Pass |  |
| 4 | Client follows protocol | Full | Pass |  |
| 5 | Client timed out after 30 seconds afk | None | Fail | Select didn’t work how I thought it did and I was going to use select to timeout clients as it has a built-in timeout in the last argument. I had developed a lot with select and was reluctant to change but should have. Then tiring to implement a timeout was breaking my blocked sequential order. |
| 6 | Last player declared winner | Full | Pass |  |
| 7 | Quit message exits client | Full | Pass |  |
| 8 | Handle game errors | Full | Pass |  |
| 9 | Handling protocol errors | Partial | Pass | Not properly defined by task sheet, no input besides digits or quits gets through and all error are treated the same. |

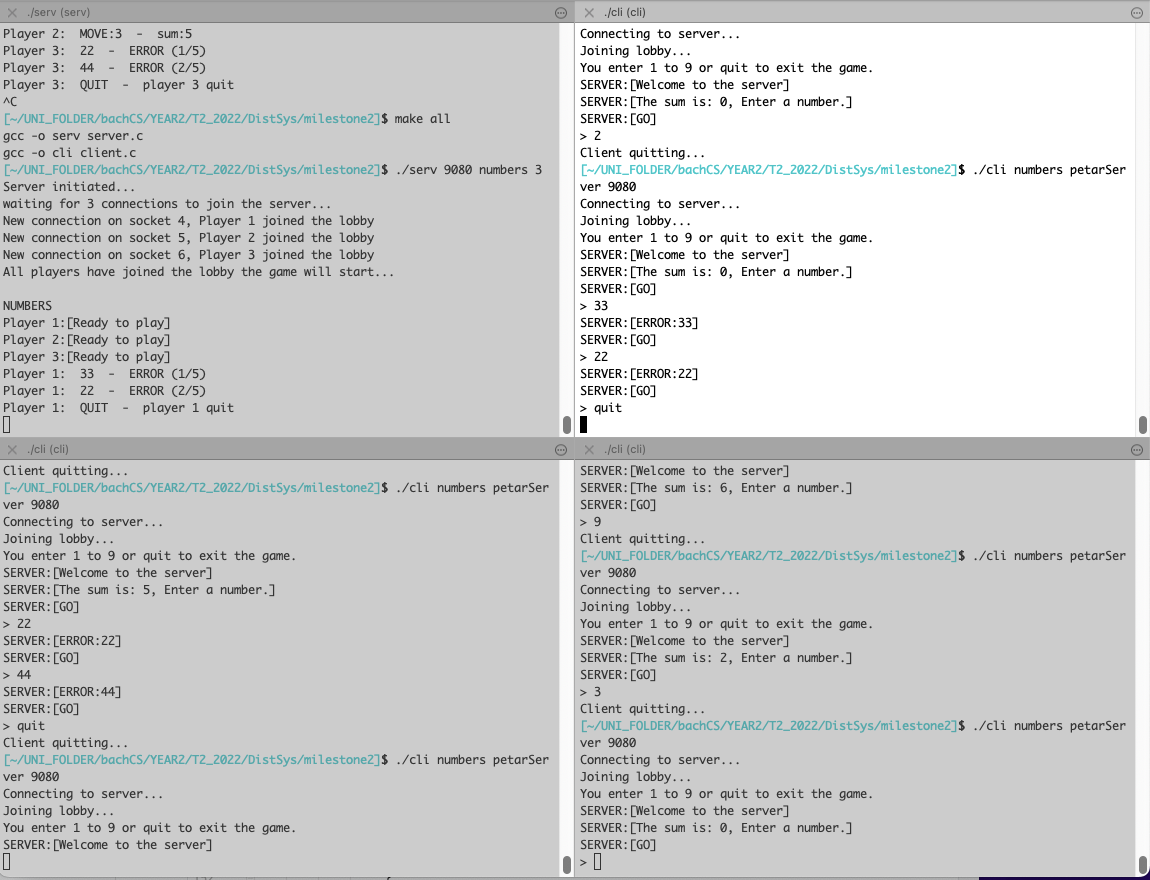
## Detailed Software Testing

| No | Test | Expected Results | Actual Results |
| --- | --- | --- | --- |
| 1 | That any client can quit, and the game will flow until a winner is found. | Any player can leave after receiving a go and the game continues. | True |
| 2 | Players can quit on turn 1 | Pass | True |
| 3 | Players can quit at any stage on play | Socket will close and game will no longer a lot that connection[i] in the if to selecet. | True |
| 4 | From turn 1 player quit until 1 and he is made winner winner, with no pointer summed. | The game will quit out the player and catch the winner before the Go is sent. | True |
| 5 | 5 errors exited out a client and the game continue | The errored-out client will be quit out of the game and the game will continue. | True |
| 6 | Sending any input besides a digit or quit will result in an error tally | The input that isn’t quit or single digits will be identified and sent without MOVE | True |
| 7 | The bot can play the game flawlessly over very large sum numbers, inc when there are no errors and only correct input is sent. | Might be some interaction with how fast the messages get sent so I attached a slow. But when errors are introduced the auto fill will hang at some points, I believe this to be because of the speed. | True |
| 8 | A player’s digit increases the score correctly | True | True |
| 9 | A player can leave the game with only the word ‘quit’ after a GO | True, all io is blocked on the clients until a go message is received. | True |
| 10 | Player can leave after sending errors | It’s a different go but it will close out the connection[i] and set it to -1. The game will continue. | True |

My main use of testing has been automated responses. In my code in the client.c there is a section for “auto fill” which will send input over the sockets. I developed this early as to not waste my time testing by hand. After I had obtained the win state end points it was a great way to ensure my games ended. Once I had developed error handling, I improved the auto fill to send a few none digits in the range of (1, 13) and this was a great was to ensure my program was exiting out players on 5 errors and how errors handled in general on mass.







## User Instructions

First run the makefile with ‘make all’, which will compile both the server and client programs. Then run the server with the specified arguments.   
  
if the makefile does not work complie both programs with the following commands

gcc -o serv server.c

gcc -o cli client.c

Then run the server and client with arguments

the command is:

./server (port number) (numbers) (players)

Then run the clients with,

./client (numbers) (PetarServer) (port number)

The program will prompt you if you get it wrong.

Launch as many clients as (players). The server will wait until all players have joined the server the game will start. During the play cycle of the game players can only enter digits 1 to 9 or the disconnect word “quit”. In the order the players join the server will send the current score and a GO message which signals the corresponding connection to make their turn. If a client is not responsive for 30 second their connection will be closed and exit the game. The game will continue until there is only 1 player. When there is a winner messages will be sent to all connected clients that they either won or lost.