**2803ICT – Assignment 1**

Harry Rowe – S5166434

Contents

[1. Problem Statement 1](#_Toc81504782)

[2. User Requirements 2](#_Toc81504783)

[3. Software Requirements 2](#_Toc81504784)

[4. Software Design 3](#_Toc81504785)

[High Level Design – Logical Block Diagram 3](#_Toc81504786)

[Software Functions 3](#_Toc81504787)

[Data Structures 3](#_Toc81504788)

[Detailed Design 3](#_Toc81504789)

## Problem Statement

The objective of this assignment is to write a remote execution system in C which consists of two programs, a server and a client. Both programs will follow a game protocol, which will be implemented through a simple game called Numbers. The game will firstly be initiated through executing a game server, which will define how many clients can connect before starting the game. Following this, a join stage will be activated, where clients are able to connect to the server. Once all clients have connected, the game will play, where each client will get a chance to select a number between 1 and 9. The sum of all client’s numbers is stored on the server. Once a client reaches a sum of 30 or more, the game is over (that client has won, the remaining clients have lost). Post-game, the clients will be removed from the server with a message stating the results of the game.

Throughout the game, the server and client will communicate through a series of messages (protocol – this is defined in the assignment requirement documentation). Protocol infringements must be handled correctly (disconnecting client, whilst also handling game errors appropriately (request valid input – disconnect client after 5 incorrect moves). Clients should also be timed out 30 seconds after not responding to the server.

## User Requirements

The programs have been designed to run on a Linux/UNIX environment. If the user does not have access to these operating systems, it is recommended to install and run Cygwin (Linux terminal environment). The first step is to ensure the executables are up to date by running ‘make’. The user will have access to 2 different executables, a server, and a client (game\_server.exe and game\_client.exe). Before running the client, the server has to be set up.

Running the server:

The server executable takes in 3 command line arguments, and should follow the below structure:

./game\_server <Port Number> <Game Type> <Game Arguments>

Firstly, the port number is an address (integer value) which the server will used to bind to. A common available port address is 80 (works adequately for this assignment). The clients will need to use the same port address that the server has bound to.

The game type is ‘numbers’ which is the game that the clients are playing.

The game arguments are the number of clients that can connect to the server. The game will not start until this number is reached. Once all clients have connected the server will notify each client that the game has started. The number of clients can be any integer amount, but the game cannot be played with 1 or less people.

Running the client:

The client executable takes in 3 command line arguments, and should follow the below structure:

./game\_client <Game Type> <Server Name> <Port Number>

The game type is ‘numbers’ which is the game that the clients will play.

The server name is the hostname or IP address of the computer that the server is running on. For instance, if the client was connecting to a PC with a hostname of ‘my pc’ and an IP address of ‘192.168.1.10’, the user can enter either of these into this command line argument and connect to the server, providing it has already been set up.

The port number is the same integer address that the server has bound to. Failing to enter the same port number will result in the client failing to bind to the specified port.

Once all clients have connected to the server, the game will commence, where the server will select one client at a time (in joining order) to choose a number between 1 and 9. The client may also quit the game by typing ‘quit’, removing them from the server. The game will continue until the sum of all previous numbers selected has reached 30 or above, at which the last client who has entered a number will win the game and the remaining clients.

## Software Requirements

The program requires the following to be able to function as per the assignment specification:

|  |  |
| --- | --- |
| **Software Requirement ID** | **Software Requirement** |
| **REQ-01** | The software should be written for Linux/UNIX systems, where each program file is written in C. |
| **REQ-02** | A make file should be used to compile the source code files into two different executables: game\_server.exe and game\_client.exe. |
| **REQ-03** | The program should ensure that the command line argument input is entered correctly (no more or less than 4 arguments for both server and client). |
| **REQ-04** | The server program should ensure that the game argument number is greater than 1 so that the server can expect more than 1 user to connect to the game. |
| **REQ-05** | The server program should detect any errors whilst initiating the server. If it detects any errors, it should terminate itself immediately. |
| **REQ-06** | The client should detect any errors whilst trying to connect/bind to the server address. If it detects any errors, it should terminate itself immediately. |
| **REQ-07** | The server will wait and open a socket connection during the joining stage, where clients are able to connect to the server (receiving a welcome message on arrival) until the specified number of clients (game arguments) have successfully joined. Once this number has been reached, the server will send a message to all of the clients, declaring that the game has commenced. The server will then be in the play stage. |
| **REQ-08** | Once the server is in the play stage, no clients will be allowed the join the game. If a client does connect during this stage, it will be sent an error message, and will be automatically terminated. |
| **REQ-09** | During the play stage, both the server and the client will agree and abide by the communication protocol by sending messages using the socket connection. |
| **REQ-10** | The server may message a client a text message which is structured using: ‘TEXT <message>’, where message is to be displayed to the clients screen. |
| **REQ-11** | The server may message a client ‘GO’, which indicates that the client needs to send some input back to the server. |
| **REQ-12** | The server may message a client ‘END’, which indicates the client must terminate itself from the server immediately. |
| **REQ-13** | The server may message a client ‘ERROR’, which indicates an error message. |
| **REQ-14** | The client may message the server ‘MOVE <move>’, where move is the clients response to a GO message. |
| **REQ-15** | The client may message the server ‘QUIT’, which informs the server the client wishes to leave the game. This results in the server replying with an END message. |
| **REQ-16** | Once a client has received a GO message, the server waits for a reply to be sent back to the server. If the blocking period lasts more than 30 seconds (server has not received input in 30 seconds), the client will automatically be disconnected from the server. |
| **REQ-17** | If the client replies back to a GO message with just a single number, the client is responsible for appending ‘MOVE’ to the front of the string, prior to sending the message back to the server. |
| **REQ-18** | The server is responsible for handling any game errors that the client sends as MOVE messages. In the case that the client has sent a MOVE message that exceeds the bounds of 1 and 9, the server should respond with a TEXT message declaring that the client has made a game error. |
| **REQ-19** | If the client has sent a MOVE message that is eligible (move is followed by a number between 1 and 9), the number is added to the total sum of all previously chosen numbers, which is stored on the server. |
| **REQ-20** | If a client has made a game error, they are allowed to resend another MOVE message back to the server. If the same client makes 5 game errors in a row, the server is responsible for terminating them from the game. |
| **REQ-21** | The server is responsible for terminating a client if they have received a message which is not included in the client sending protocol. |
| **REQ-22** | The client is responsible for terminating themselves if they have received a message which is not included in the server sending protocol. |
| **REQ-23** | The server is responsible for keeping track of the order that the clients have joined, so that the next persons turn can be chosen appropriately. |
| **REQ-24** | If a client leaves the game (through quitting or a time out), the game continues with the remaining active clients. The server will keep track of who has left, so that the disconnected clients do not get another turn. |
| **REQ-25** | If the number of active clients in the game drops to 1, the client that is remaining will receive a TEXT message from the server letting them know they have won. |
| **REQ-26** | If the total sum stored on the server exceeds 30 (or is equal to 30), the game is over. The server is responsible for announcing the results (client whose turn it was last wins – remaining clients lose) and terminating them appropriately. |
| **REQ-27** | Once the game has been announced that it is over, the server should ensure that all resources such as sockets and processes are fully closed. |

## Software Design

## High Level Design – Logical Block Diagram

## Software Functions

For each function in the list the following information is provided:

1. a brief description of what it does (1 or 2 sentences);
2. a list of the input parameters, and their data types, and what they are used for;
3. a list of any side effects caused by the function (ie change global or member variables, changes data passed by reference from calling function etc)
4. a description of the function’s return value

## Data Structures

For each data structure in the list the following information is provided:

1. Type of structure (tree, list etc),
2. Description of where and how it is used
3. List of data members, and what each one is for do
4. List of functions that use it

## Detailed Design

– Pseudocode for all non-standard and non-trivial algorithms that operate on datastructures