2803ICT – Assignment 1

Harry Rowe – S5166434

Contents

1. 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.

2. User Requirements

From the assignment brief, it can be deduced that the user requires:

- Entering in 3 command line arguments for connecting to the server (order is specified in user instructions).
- To select port number for server setup (use this same port number for client connection).
- Knowledge of server hostname or IP address.
- To be notified from the server when it is their turn.
- To be notified from the server when the game has started.
- To be notified that they cannot join the game once it has already started.
- To be notified from the server if they have won or lost the game.
- To be removed from the server upon committing a protocol infringement.
- Responding back to go messages with a number between 1 and 9.
- Ability to leave the game by entering a ‘quit’ command upon their turn.

3. Software Requirements

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

Table 1: Software Requirements 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 a 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 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-20	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-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.

4. Software Design

High Level Design – Logical Block Diagram

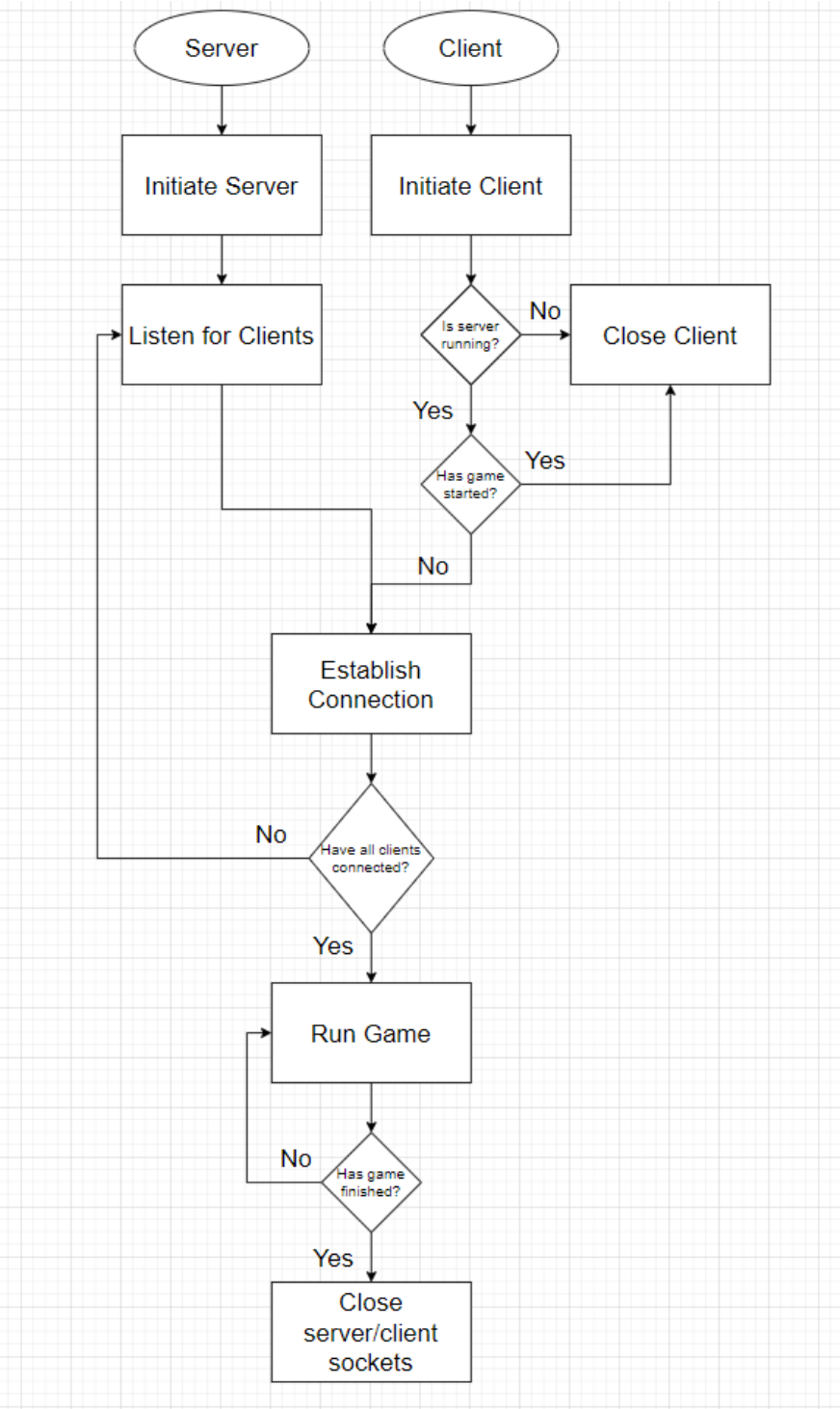

Figure 1: High level logical block diagram

Structure Chart

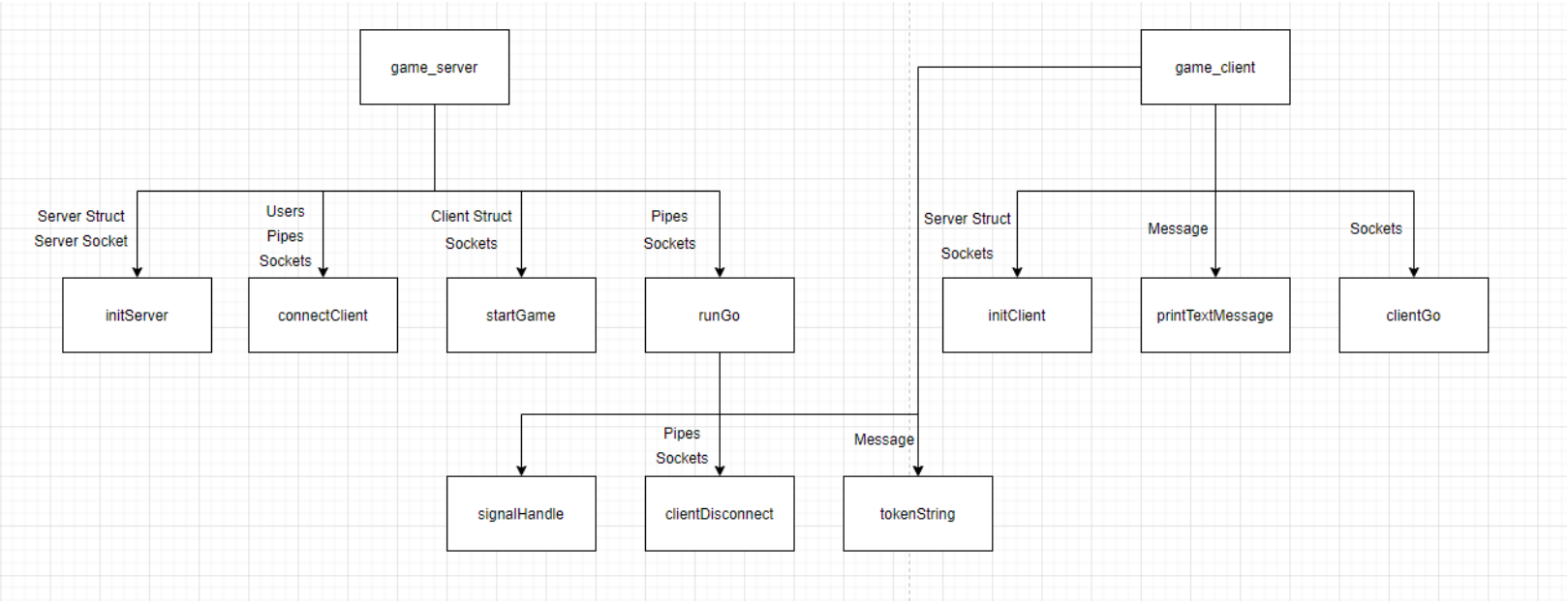

Figure 2: Software function structure chart

Software Functions

Server Functions

initServer

- This function initiates the server socket address structure and binds the server socket to the specified port. It also sets the socket to listen to incoming client connections.
- The input parameters consist of:
  - Server socket address pointer – preparing socket address structure for server socket
  - Server socket integer pointer – holds integer address of the server socket
  - Return value integer pointer – holds error handling value from socket preparation functions
  - Port number as an integer – pass to bind function for socket address
- Considering the server and server socket integer value are passed by reference, these functions are altered during the function by preparing the socket structure. Internal functions are called, such as socket(), bind(), and listen(), which creates the server

socket and causes it to listen to incoming clients who are attempting to connect to the server.

- The functions return value is void.

connectClient

- This function uses a client socket address to accept a connection from a client and connect them to the server. Child processes are also a product of this function, as fork is called once a new client has connected to the server.
- The input parameters consist of:
  - Users integer array as a pointer – keeps track of the place in queue of the current client
  - Active users array as a pointer – keeps track of each child process ID
  - Child to parent pipe integer array as a pointer – Pipe to parent process
  - Parent to child pipe integer array as a pointer – Pipe to child process
  - User number integer as a pointer – Keep track of number of clients connected
  - Client socket address structure as a pointer – Used to accept client to server
  - Client socket integer as a pointer – Holds integer address for client
  - Server socket integer as a pointer - Holds integer address for server
  - Child process ID as a pointer – Holds child process ID
- Primarily, this function accepts the connection of a client, and stores it's child process information in the applicable integer arrays so that the server knows in which order the clients connected. This function is also responsible for piping the correct index to the child process, so that it can communicate to the parent process with no unexpected issues.
- The functions return value is void.

startGame

- This function is called once all clients have connect to the server and the game is ready to be commenced. It sets up a listen ID process to accept clients who connect after the game has started (and immediately terminate them).
- The input parameters consist of:
  - Game started integer passed as a pointer – let the server know the game has started globally.
  - Number of users as an integer – stores current number of clients connected
  - Client socket address structure as a pointer – accepts clients once game has started
  - Client socket integer – store address of client socket
  - Server socket integer – store address of server socket
  - Listen child ID as pointer – holds process ID for listening once the game has commenced

- This function creates the child process for handling clients who connect outside of the game once it has started. It informs them that the game has started and immediately terminates them from the server.
- The functions return value is void.

runGo

- This function is called when a child process receives a request to send a GO message to a client. It handles the passing of data between pipes and sockets.
- The input parameters consist of:
  - Child to parent pipe as integer array pointer – pipe to parent process
  - Parent to child pipe as integer array pointer – pipe to child process
  - Client message as a character array – holds client response to GO message
  - Client socket integer – holds integer address of client socket
  - Number of users as integer – holds number of clients connected to the server
- The focus of this function is to handle writing and receiving to the client from the server. Multiple instances are handled if the client decides to quit or commits a protocol error, where the client will be led to a disconnection outside of the function. The biggest side effect of this function is the communication handle between pipes from the parent process (reads and writes messages back – such as the current sum value, whether the client has quit or not, and the clients response back to the GO protocol).
- The functions return value is void.

signalHandle

- This function handles the instance of when the server sends a previously disconnected client a message. It sets a signal action to continue, ignoring the broken pipe formed by send() or write() to a disconnected client.
- There are no input parameters for this function.
- The focus of this function is to ensure that the program does not get stuck in a broken pipe after sending a message to a disconnected client. If it has already disconnected, it allows send() or write() to return -1, indicating that the message has failed to send. If the server has received this, the client will be disconnected immediately.
- The functions return value is void.

clientDisconnect

- This function handles the processes of removing a client from the game upon either using QUIT, protocol infringements, game errors or leaving the game prior to the turn.
- The input parameters consist of:
  - Pipe to parent process as an integer array pointer – write to parent that client has disconnected

    - Number of users as an integer – hold number of active clients
    - Client socket as integer – hold client socket address as integer
- The focus of this function is to write to the parent process that the current client that is being dealt with no longer needs to be in the game. This communicates this using a -1, which indicates that this child process should no longer be involved in the active users array.
- The functions return value is void.

Client Functions

initClient

- This function initiates a clients connection with a server and provides error handling if any connection issues occur.
- The input parameters consist of:
    - Server socket address structure pointer – used to connect to the server once structure is prepared
    - Client socket integer pointer – used to create and hold client socket address on socket() function call
    - Return value integer pointer – used to handle errors to pass back to main function if any occur whilst connecting
    - Port number as integer – used for server structure to bind its address to
    - Host as a character pointer – used to point to hostname provided by client (this will be the server/host name they are trying to connect to)
- The focus of this function is to establish a connection with the server that is specified by host. If a connection is successful, the server and client socket variables will be updated (as passed by reference), which will allow communication between the server and the client.
- The functions return value is void.

printTextMessage

- This function is used to handle the response to a server when a client receives the TEXT message. It prints out the message from the server excluding the word TEXT.
- The input parameters consist of:
    - Message which is a character pointer to a pointer to a pointer – points to the two-dimensional array that the message will be stored in
    - Size as an integer – holds the number of words in the given message

- The focus of this function is to print each word to the clients screen, which has been passed as a message from the server. It does not change anything outside of the function itself, just provides the client with information from the server.
- The functions return value is void.

clientGo

- This function is used a response to when the client has received a GO message. It handles receiving the server message, as well as the clients response, whilst also running a 30 second timeout.
- The input parameters consist of:
  - Client socket integer – holds the integer address of the client socket
  - Return value integer – holds the return value of any errors found in function
- The focus of this function is to scan the clients input buffer to pass the information back to the server (which deciphers the clients response). The 30 second timeout is also used to identify if the client has become inactive/unresponsive, which sends a client disconnection request back to the server.
- The functions return value is void.

tokenString

- This function is used to convert a string of characters into a two-dimensional ragged array of characters, which uses each i^{th} index as a new word (tokenising each word in original array by spaces).
- The input parameters consist of:
  - Message which is a character pointer to a pointer to a pointer – points to the two-dimensional array that the message will be stored in
  - Original message as a character array – holds original message
  - Size as an integer pointer – holds number of words in original character array
- The focus of this function is to store each word in the original string into a new message array which is passed by reference. Each word is reallocated space in runtime, which allows for the original message to be of variable length.
- The functions return value is void.

Data Structures

One of the main data structures that is used for communication within processes is a pipe. When the program calls the function fork(), the child process creates a copy of its parents data (includes variables, structures, etc.). As a result of the copy, data that is altered in a child process is not reflected amongst the parent or other child processes. A way to overcome this issue is to use inter process communication (pipe). The data structure itself is an integer array which utilises reading and writing functions to communicate between the parent and child processes. A standard one-way pipe uses the following structure:

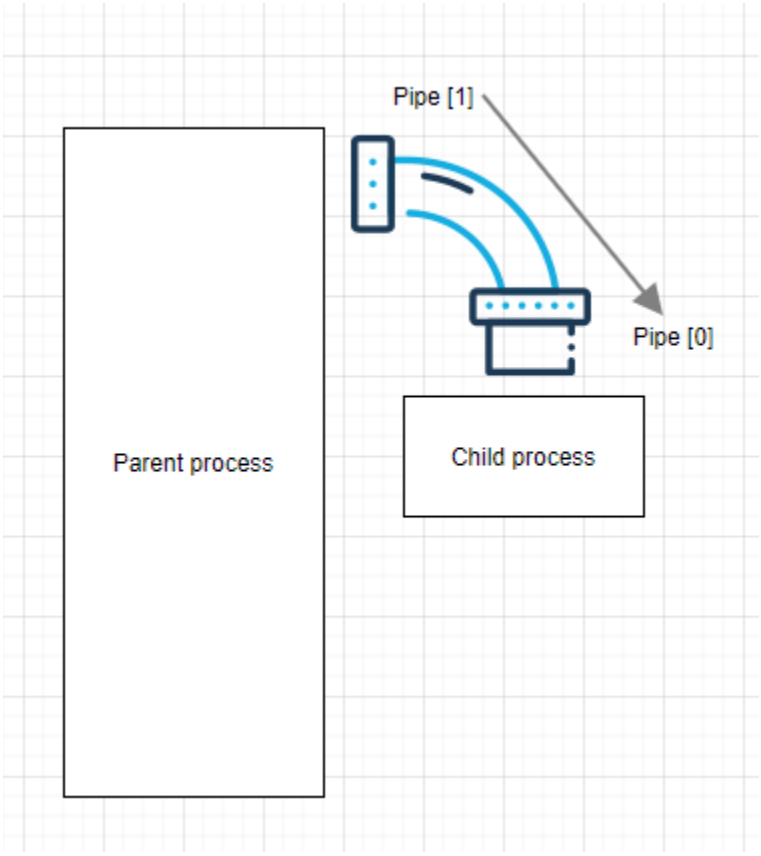

Figure 3: One-way pipe where a parent process writes (using index 1 to a child process (reading using index 0).

For two-way communication, two pipes are needed for the child to write back to the parent process:

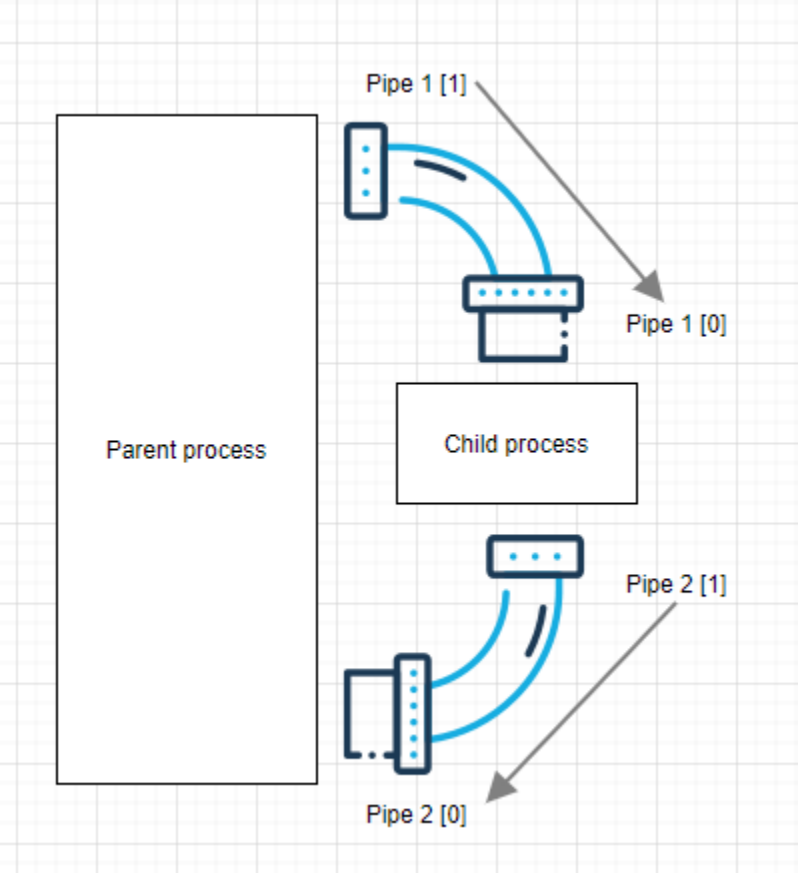

Figure 4: Two-way pipe where a parent process writes using pipe 1, and the child process writes using pipe 2

Due to the nature of the assignment, the main process had to communicate with multiple processes (clients), which required the pipes to use different indexes to control what process to read and write to. Each child process was allocated a specific read and write index to listen and send to when it was proposed to write to that specific process. The number of clients connected at the time of the new client arrival will determine at what position the child process can read/write. The following diagram discusses the process to pipe management:

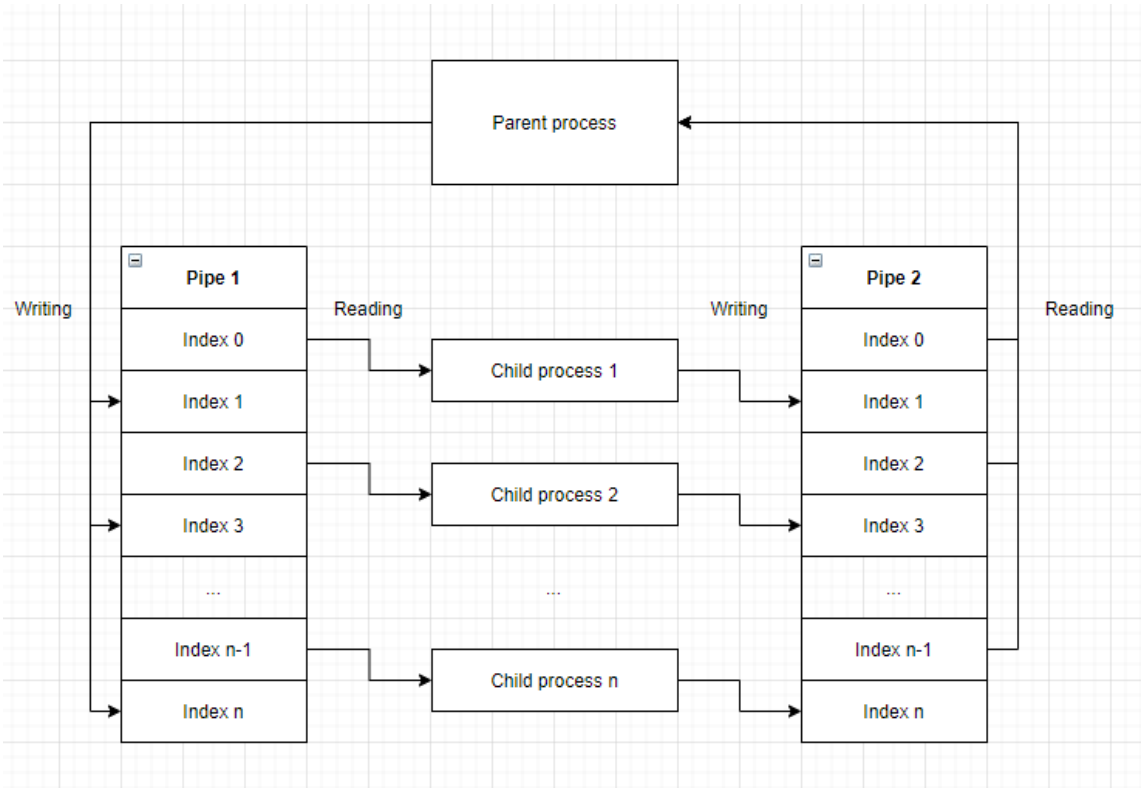

Figure 5: How child processes interact with pipe 1 and pipe 2

The primary functions that interact with pipes are:

- connectClient
- clientDisconnect
- runGo

One of the other structures that is used in the server/client model is the sockaddr_in structure, which is included using the “netinet/in.h” library. This structure is primarily used for handling internet addresses and is responsible for connecting clients to the specified server/hostname that they chose upon initiating the client executable. The data members of this structure include:

- sin_family (short)
- sin_port (unsigned short)
- sin_addr (struct in_addr)
- sin_zero (character)

Each of these values is assigned prior to binding/connecting (whilst preparing the socket structure). The list of functions which utilise this structure are:

- initServer/initClient

- connectClient
- startGame

Detailed Design

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

```
initServer(server struct, server socket, port){
        // Prepare server socket and server data structure
        server socket = socket(AF_INET, SOCK_STREAM)
        server struct.sin_familyu = AF_INET
        server.sin_addr.s_adr = IP Address
        server.sin_port = port
        bind(server socket, server struct)
        listen(serverSocket, BACKLOG)
}

initClient(server struct, client socket, port, host){
        // Prepare server socket
        server.sin_addr.s_addr = IP Address
        server.sin_family = AF_INET
        server.sin_port = port
        client socket = socket(AF_INET, SOCK_STREAM)
        connect(client socket, server struct)
}
```

```
connectClient(users, pipe1, pipe2, client struct, server socket, client socket, child process){
        client socket = accept(server socket, client struct)
        if (client socket < 0)
                print(error)
                exit(1)
        else
                pipe(&pipe1[2*users.size()])
                pipe(&pipe2[2*users.size()])
                child process = fork()
                users[users.size()] = child process
}

runGo(pipe1, pipe2, client socket, user number){
        signalHandle()
        // Read sum from parent process
        read(pipe2[2*user number], sum)
        if (send(client socket, "TEXT" + sum, SIZE) < 0)
                disconnectClient(pipe1, client socket)
        while (client attempts < 5)
                write(client socket, "GO", SIZE)
                recv(client socket, client message, SIZE)
                tokenString(client message)
                if (client message[0] == "MOVE" && client message[1] is between 1 and 9)
                        // Write number back to process
                        write(pipe1[2*user number + 1, client message[1], SIZE)
                else if (client message[0] == "MOVE" && client message[1] is not between 1 and 9)
                        send(client socket, "TEXT Try again", SIZE)
```

```
                else if (client message[0] == “QUIT”)
                        // client quits, or client message is protocol infringement
                        clientDisconnect(pipe1, client socket)
}

clientDisconnect(pipe1, client socket){
        write(p1[2*user number + 1], disconnect (-1))
        close(client socket)
        exit(0)
}
```

5. Requirement Acceptance Tests

Table 2: Requirement Acceptance Tests

Software Requirement ID	Test	Implemented (Full/Partial / None)	Test Results (Pass/ Fail)	Comments
REQ-01	Programs can compile in a Cygwin environment.	Full	Pass	
REQ-02	Make file compiles into two executables in Cygwin environment.	Full	Pass	Compiles with no errors
REQ-03	Enter 5 command line arguments on both client and server executables and expect an error.	Full	Pass	Works where argc = 4
REQ-04	Enter 1 game argument and expect server to respond with 'Enter more game arguments'.	Full	Pass	Works where game arguments exceed 1
REQ-05	Enter an invalid port number on command line argument ("ABC").	Full	Pass	
REQ-06	Enter a port number that differs from the server port number.	Full	Pass	
REQ-07	Establish a connection between 3 clients when game argument is set to 3. Expect game to start immediately after the 3rd client connects.	Full	Pass	Welcomed with message upon arrival
REQ-08	Attempt to connect a client once the game has commenced.	Full	Pass	Error message received
REQ-09	Connect two clients to the server and send a 'MOVE 7' message from the 1st client when server sends GO.	Full	Pass	Completed – sent '7' to 2nd clients screen
REQ-10	Connect a client to the server and expect a welcome message from the server.	Full	Pass	
REQ-11	Connect two clients to the server and expect the 1st client connected to receive a GO message first.	Full	Pass	

REQ-12	Send “MOV 3” from a client as a response to a GO message.	Full	Pass	Removed immediately from server
REQ-13	Enforce the client terminal to close by responding with an unknown command back to a GO message.	Full	Pass	Used an error message to display to server + client once error occurs
REQ-14	Send “MOVE 3” from a client as a response to a GO message.	Full	Pass	
REQ-15	Send “quit” from a client as a response to a GO message	Full	Pass	Removed immediately from server
REQ-16	Upon a client receiving a GO message, do not respond for a 30 second period.	Full	Pass	Kicked from game session
REQ-17	Client sends ‘7’ back to a GO message from server.	Full	Pass	Server received as a ‘MOVE 7’ message
REQ-18	Client sends ‘22’ back to a GO message from a server.	Full	Pass	Asked to retry
REQ-19	Client sends ‘22’ 5 times in a row to a GO message from a server.	Full	Pass	Asked to retry 5 times before disconnecting
REQ-20	With the sum being 15 currently, the client sends ‘3’ back to a GO message from server.	Full	Pass	The next person in queue received 18 as current sum
REQ-21	Client sends ‘HAVE 789’ to a GO message from a server.	Full	Pass	Disconnected immediately
REQ-22	Added a send message in server code to replace welcome TEXT message. Sent client “TEST”.	Full	Pass	Client did not understand ‘TEST’,

				therefore disconnected immediately.
REQ-23	Establish connection between 3 clients, play 1 turn each from the clients, ensure each turn taken is in joining order.	Full	Pass	Followed logical queue joining order
REQ-24	Establish connection between 3 clients, play 1st turn on player one, quit on player two, play 1st turn on player 3. Play second round, ensuring that player one and player three are only involved.	Full	Pass	Disconnecte d player two immediately after 'quit' was sent from client.
REQ-25	Establish connection between 2 clients, play 2 rounds, disconnect client one. Ensure client two receives a winning TEXT message.	Full	Pass	Client two received 'you win!'
REQ-26	Establish connection between 3 clients, play 3 rounds. Get score total to 25 on client one's 4th turn. Client one responds with '5' as a response to GO. Ensure client one receives winning TEXT message and clients two and three receive losing TEXT messages.	Full	Pass	
REQ-27	Ensure once a client has received a winning TEXT message, the server has logged a 'server closing' message.	Full	Pass	

6. Detailed Software Testing

Table 3: Detailed Software Testing

No.	Test	Expected Results	Actual results
1.0	**Setting up server**		
1.1	Enter no command line arguments	Display “Error: insufficient command line arguments”.	As expected.
1.2	Enter 5 command line arguments	Display “Error: insufficient command line arguments”.	As expected.
1.3	Enter 1 as a game argument	Display “Error: Please enter more than 1 client” .	As expected.
1.4	Enter “xyz” as port number	Display “Error: please enter a valid port address”.	As expected.
1.5	Enter port number as 80 with game type as “numbers” and game arguments as 3.	Display “waiting for clients”.	As expected.
2.0	**Setting up client**		
2.1	Enter no command line arguments.	Display “Error: insufficient command line arguments”.	As expected.
2.2	Enter 5 command line arguments.	Display “Error: insufficient command line arguments”.	As expected.
2.3	Enter a server name which does not exist (“computer111”).	Error with connecting to a server.	As expected.
2.4	Enter port number that differs from server port number.	Error connecting to a server.	As expected.
2.5	Enter “numbers” with correct server name (“HARRY-PC2”) and correct port number.	Expect to connect to the server with no errors.	As expected.
3.0	**Connecting clients to server**		
3.1	On server expecting 3 clients, connect 1 client using correct command line arguments.	Welcome message where the client will have to wait for two more users to join. Server also logs that 1 user has joined.	As expected.

3.2	On server expected 3 clients, connect 3 clients using the correct command line arguments.	Welcome message to appear on each clients screen upon connection. Once all three connect, send a game commence message to all clients. The first client who connected should receive a GO message.	As expected.
3.3	Once all clients have connected, attempt to connect another client to the server.	Error message to appear on clients screen (game already started).	As expected.
3.4	On server expecting 3 clients, connect one client successfully, connect a second but immediately terminate the window, connect a third client afterwards.	The game should start as 3 clients have previously connected. Client one should take its turn. Upon client twos turn, immediately disconnect as they are no longer in the game, move straight onto client three.	As expected.
4.0	**Playing & closing the game**		
4.1	On a clients response back to a GO message, reply back with '1'.	Expect next clients turn to display sum + 1 as the total sum.	As expected.
4.2	On a clients response back to a GO message, reply back with a '10'.	Expect another turn.	As expected.
4.3	On a clients response back to a GO message where 4 previous game errors have been committed, reply back with '11'	Expect client to be disconnected, next players turn.	As expected.
4.4	On a clients response back to a GO message, reply back with 'quit'.	Expect client to be disconnected, next players turn.	As expected.

4.5	On a clients response back to a GO message, reply back with "MOV 4"	Expect client to be disconnected, next players turn.	As expected.
4.6	On a clients response back to a GO message, do not reply for 30 seconds.	Expect client to be disconnected, next players turn.	As expected.
4.7	On a server where there are 3 clients connected and client 1 has just received GO; disconnect client 2 unexpectedly.	Client 1 responds with correct protocol; GO gets sent to client 2 but is irresponsive. Disconnect client 2 immediately, GO gets sent to client 3. On the following turn, only client 1 and client 3 receive messages.	As expected.
4.8	On a server where there are 3 clients connected and the sum is currently 22, GO gets sent to client 2 and responds back with '8'.	Client 2 receives a winning message and clients 1 and 3 receive losing messages. Server closes.	As expected.
4.9	On a server where there are 2 clients connected and GO gets sent to client 2. They respond with 'quit'.	Client 2 receives a quit message and is disconnected. As there is only 1 client left, they are sent a win message and disconnected from the game. Server closes afterwards.	As expected.

7. User Instructions

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.