2803ICT ASSIGNMENT 2

Sebastian perry

S5132483

Contents

Problem Statement

This assignment requires the construction of code which can produce 2 separate programs, a server and client. The server should be able to facilitate the connection and communication with multiple clients and through the use of a pre-determined set of protocols and rules set by the task sheet. The sever should also be able to use the aforementioned rules as a way for the client to interact with a game known as numbers which the server will run, the client in turn will be a way in which the average end user would be able to interact with the server and furthermore the game in question.

User Requirements

The following will be a collection of dot point outlining the user requirements stated by the task sheet:

1. The user should be able to connect to the server running on a different pc
2. The user should be able freely type commands when it's their turn
3. They should be able to leave the game via the quit command
4. They should be able to play a move by either using the MOVE command or just simply typing a number
5. If they input a number outside the allowed range they should be able to try 4 more times before they are kicked from the game
6. The user running the server should be able correctly run the server from command line and have it autonomously run the game from there

Software Requirements

The following will be a collection of dot point outlining the software requirements stated by the task sheet:

1. Both the client and server programs should be able to be run with arguments from command line
2. The server accepts connections from users during the joining phase until the require game size is reached however once it is the server should stop accepting new connections.
3. The server and client both run off a set of predetermined protocols
4. If either the server or the client receive a command out of protocol they will disconnect from the offender and protocol
5. If a player does not respond for 30 seconds on their turn they are removed from the server
6. If the score reaches 30 the last player should be considered the winner and the game should end

Software Design

Shared

Includes

<stdio.h>, <stdlib.h>, <string.h>, <time.h>, <sys/socket.h>, <unistd.h>, <netinet/in.h>, <arpa/inet.h>

Functions

- void myTime ();

  Summary: this function prints the current time to console

- char* popFront (char * str);

  Summary: this function replaces the first space it finds with a \0 and return a char* pointer to the character after it

  "str": the string which this function concerns

  Returns: a char pointer to the string after the first

- void unPopFront (char * str);

  Summary: first of all, frankly this feels like bad design but this function reverses the effect of the popFront function. upon giving it the orginal string it will remove the \0 which seperated the command from the args

  "str": the string which this function concerns

- void fgetsClean (char * str);
  Summary: this function cleans up after fget's removing it's tailing newline character

  "str": the string which this function concerns

Global constants

- MAXIN = 1001

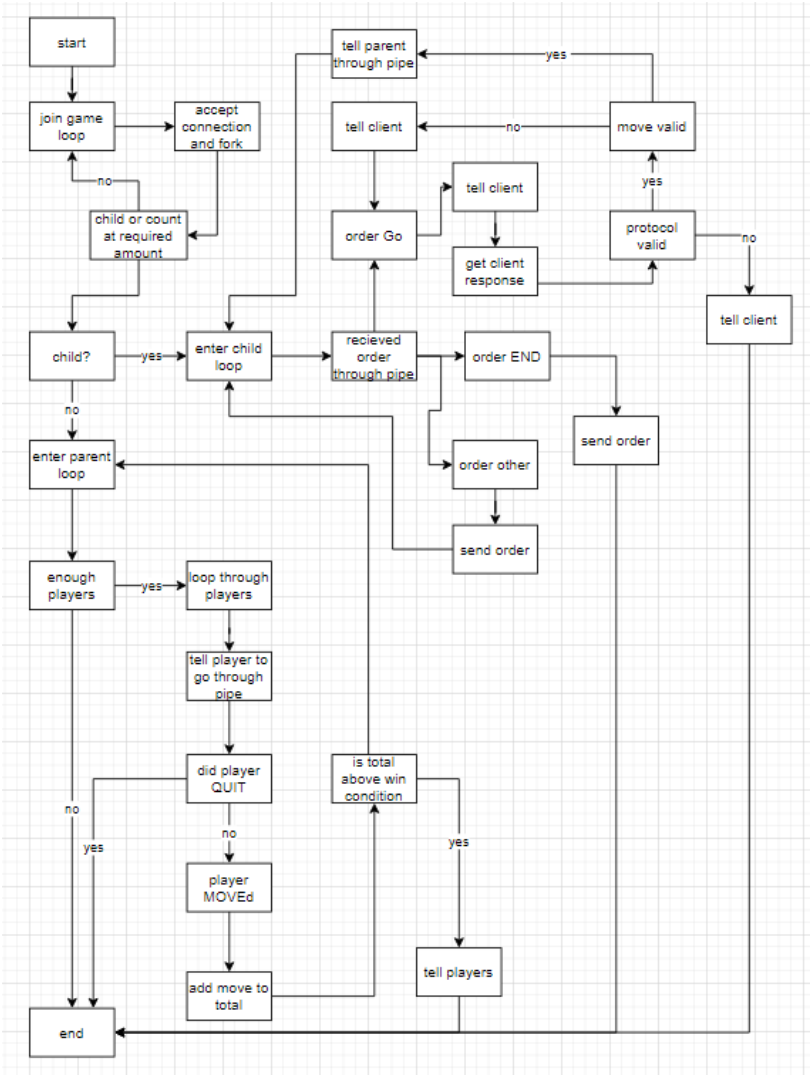
  Description: a global constant contains as an integer the maximum number of characters accepted in all input applications

Server

Includes

<sys/types.h>, <sys/wait.h>

Logical Block Diagram



Global constants

- BACK_LOG = 10
  Description: a global constant contains as an integer of the maximum number of clients that can attempt to connect at the same time
- GAME_GOAL
  Description: a global constant contains as an integer of the goal number to be reached in the game
- TIMEOUT_LIMIT
  Description: the amount of time the server will wait for a client answer after giving them the go command

Loose Pseudocode description

```
initialise variables, and setup network connections

while player count is below the desired amount

	wait for a client to attempt a connection

	switch the connection socket to the connect with the client

	fork the process

	if this process is a child break the loop early

if this process is a child:

	loop through forever:

		if an order has been sent through the pipe:

			if the order is GO

				loop through forever

					send the GO protocol to the client

					wait for a response

					if the response is the MOVE protocol

						if the move is valid:

							write the move to the score pipe

						else if the move is not valid:

							inform the user of the invalid input

							if there hasn't be 5 invalid inputs:

								let the user try again
```

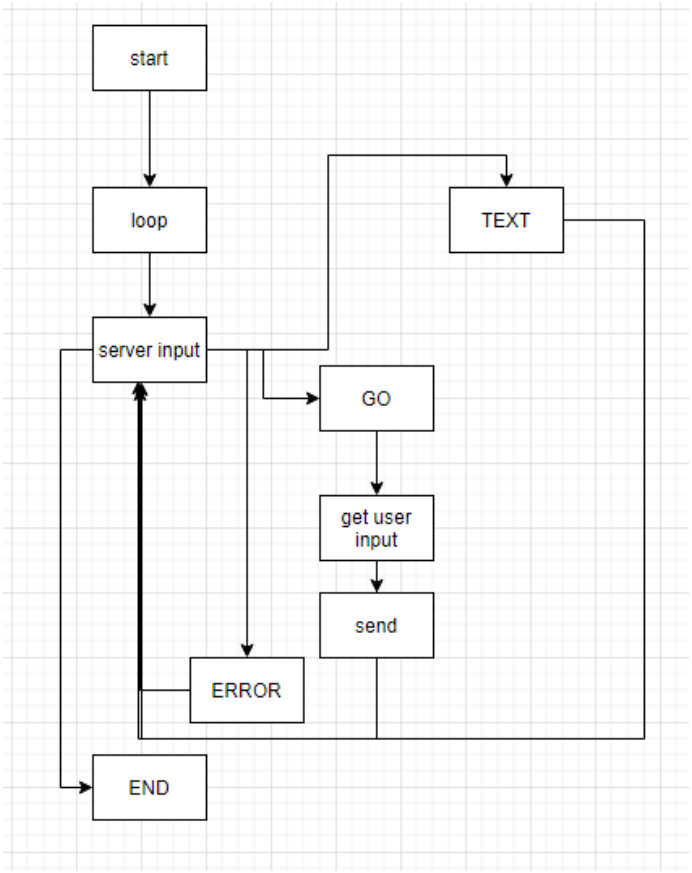
```
                                else if the response isn't the QUIT protocol:

                                        inform the user that they have used a command outside of protocol

                                // if the code has gotten here then 1 of a few things have happened the client // timed out,
                                the command was invalid, or the client sent quit

                                tell the user to disconnect

                                close the connection

                                kill the process

                        begin the loop from the beginning

                if the order is end:

                        tell the user to disconnect

                        close the connection

                        kill the process


                // if the code has gotten here the protocol is
                // either text or error

                send the message

else if this process is the parent

        loop through forever:

                loop through the players in the order they joined

                        if the player count falls to 1:

                                loop throught the players:
                                        if the player is still connected:
                                                through the pipe tell the players that the game has ended due to a lack of players
```

```
        loop throught the players:
                if the player is still connected:
                        through the pipe tell the players to disconnect

        break the loop

if the current player is still in the game

        through the pipe tell the player to GO

        loop through forever:

                if the child process sends a reply through the pipe:

                        if the player left

                                constuct leave message

                        else if they made a move

                                add the move to the total

                                construct the move message

                        loop through the players:

                                        if the player is still connected:

                                        send the player the message through the pipe

                        if the score is now above the goal:

                                construct the win message

                                loop through the players:

                                        if the player is still connected:

                                                through the pipe send the message
```

```
                                        loop through the players:
                                               if the player is still connected:
                                                      through the pipe tell the player to disconnect
                                        close and deallocate rescourse
                                        wait for any zombies
                                        exit the application
                                 break the loop
            if the current player count is less than 1:
                  break the loop
wait for any zombies
close and deallocate rescourse
```

Client

Logical Block Diagram



Loose Pseudocode description

```
initialise variables, and setup network connections

loop through forever:

	get input from server

	if input is of TEXT protocol:

		then display it's args to console

		begin the loop again

	if input is of GO protocol:

		get users input

		if their input isn't one of the recognised protocols and it has a number:
			add the MOVE prefix

		begin the loop again

	if the input is of END protocol:

		break the loop

	if the input is of the ERROR protocol:

		then display it's args to the console
		begin the loop again

	// if this has been reached then the command sent is not protocol

	quit the server

	break the loop

close and deallocate resources
```

Requirement Acceptance Tests

Requirement No.	Test	Implemented (Full/ Partial/ Not)	Test Result (Pass / Fail)	Comment
U1	While I didn't have another pc on hand, I did forward the chosen port and connect to my public IP address which should prove the validity of the result	FULL	Pass	
U2	2 protocol commands were typed and 2 random strings were typed in	FULL	Pass	
U3	Typed QUIT on client turn	FULL	Pass	
U4	A move was taken with the command prefix, and by just typing the number	FULL	Pass	
U5	First make and incorrect input 6 times, then another time make an incorrect input followed by a valid one	FULL	Pass	
U6	Run the server	FULL	Pass	
S1	Run the applications under user conditions	FULL	Pass	
S2	Connect the correct number of clients then attempt to connect one more	FULL	Pass	
S4	During development make the server send an invalid protocol, and after as a client send an invalid protocol	FULL	Pass	
S5	On the clients turn, don't respond for 30 seconds	FULL	Pass	If the player does nothing before 30 seconds ends the application will wait for input forever, upon the next enter however it will display a message and end as desired
S6	A game was played to completion	FULL	Pass	

Software Testing

No.	Test	Expected result	Actual result
1	Input negative number into the move protocol	The server would remove the negative sign and read it as a positive	As expected
2	Input only negative number into the input	The client would not add the prefix and as such the server would indicate it was outside protocol and disconnect	As expected
3	A random string was entered into the input	The server would indicate it was outside protocol and disconnect	As expected

User Instructions

Below will be a list of instructions that are necessary to run the project which was created for this assignment:

Preparations

This application is designed to solely work on Linux based operating systems and as such is inoperable on windows without the use of bash emulation software such as Cygwin; such running conditions must be obtained. Furthermore, if the intention is to run this with clients connecting from different networks on the internet, some extra steps will have to be taken to forward the port from your router to the computer running the server; in such a situation the server runner should forward the desired port and the clients should instead use the public IP of the network rather than the local IP of the server computer on its network.

Server

In order to run the server, you should open the system terminal (or Cygwin terminal on windows), navigate to the directory of the server program and run it with a line similar to the one shown below, filling in the angle bracketed sections with relevant variables:

./game_server.exe <port number> <game type> <game arguments>

Port number – is the number of the port on which the server is listen to for incoming users, in instance where this game is being played over the internet, this will also be the port which will have been forwarded from the router to the computer running the server.

Game type – this is the name of the game which will be played, currently numbers is the only game which has been developed and as such it is the only acceptable input.

Game arguments – this is a section for arguments and variables required to inform the game on how it should run. For numbers this is used to dictate the number of players the following game will hold.

Client

In order to run the client, you should open the system terminal (or Cygwin terminal on windows), navigate to the directory of the server program and run it with a line similar to the one shown below, filling in the angle bracketed sections with relevant variables:

./game_client.exe <game type> <server IP> <port number>

Game type – this is the name of the game which will be played, currently numbers is the only game which has been developed and as such it is the only acceptable input.

Server IP – this is the IP of the computer running the server, if being played on the same network the local IPV4 should be used. If the game is instead being played over the internet, then the public IP address of the network should be used.

Port number – is the number of the port on which the server is listen to for incoming users, in instance where this game is being played over the internet, this will also be the port which will have been forwarded from the router to the computer running the server.

Game rules

Phases

The game is split up into 2 sections: the joining phase, and the game phase. In the Joining phase all the clients should connect to the server until the specified number of players have joined the game, at this point the game phase will begin.

Controls

During the game phase the server will call on the users one at a time, in the order in which they joined the game, when called upon to 'please take you turn' the user running the client has 30 seconds (if exceeded with no response the user will be removed from the game) to use one of two commands: the user can choose to QUIT which will remove them from the game, or they can choose to MOVE which should be followed by a number; in addition to that a user can also simply input a number which the client will interpreted as a MOVE command and the prefix automatically; furthermore the number sent has to be within a given range, if a players sends off a number outside the given range (which is currently between 1-9) the server will give the player 5 additional chances to send a valid input before they are removed from the game. Any command other than the commands mentioned will result in a server error which will result in that client's removal from the game.

Win conditions

For each turn the players will use the move command you send a number to the server which will be added to a running total which will be displayed to the users at the end of each turn, the aim of the game is to make the total reach 30 on your turn at which point you will win and the game will be over.