

Guess The Number

A Random Number Guessing Game For Multiple Clients Using Socket Programming

Khushi Suresh Muddi
dept. of Information Technology
National Institute of Technology
Karnataka
Surathkal, India

khushisureshmuddi.211it032@nitk.edu.in

Harsha S
dept. of Information Technology
National Institute of Technology
Karnataka
Surathkal, India

harshas.211it024@nitk.edu.in

Mohammed Fahad
Altamash
dept. of Information Technology
National Institute of Technology
Karnataka
Surathkal, India
mohammedfahadaltamash.2
11it041@nitk.edu.in

Abstract—Socket programming is a method to connect two nodes over a network to establish a means of communication between those two nodes. In this report we explain the design and implementation of Guess the Number: A multiplayer/multi-client number guessing game build using socket programming in C/C++.

server using the read() system call. The server-side socket program would create a socket and bind it to a specific port number and IP address, and then listen for incoming connections. When a connection is accepted, the server can send and receive data with the client using the same system calls (write() and read()). The key thing to remember is that s

I. INTRODUCTION

In a Multiplayer/Multi-client number guessing game using socket programming, multiple clients connect to a server via sockets and attempt to guess a randomly generated number. The server keeps track of the clients guesses and sends messages to each client indicating whether their guess is too high, too low, or correct. The first client to guess the correct number wins the game. The game can be implemented using various programming languages such as C/C++

II. SOCKET PROGRAMMING

Socket programming is a method of inter-process communication (IPC) used to allow separate processes to communicate with each other on the same or different machines. It allows for the creation of network-based applications and is a key technology for distributed systems. Sockets provide a common interface for network communication and are based on the concept of ports, which are virtual channels for data transmission. A socket is an end-point for sending or receiving data across a computer network. They are created using the socket system call and are used to send and receive data using the read and write system calls.

a client-side socket program is being implemented that connects to a server using socket programming. The client program takes the server's hostname and port number as command-line arguments and creates a socket using the socket() system call. The hostname is used to resolve the IP address of the server, and the port number is used to specify the service or application that the client wants to connect to on the server. The client then uses the connect() system call to establish a connection to the server. Once the connection is established, the client can send data to the server using the write() system call and receive data from the

III. COMPONENTS

A multi-client number guessing game using socket programming involves several components:

Server: The server generates a random number and waits for clients to connect. It also maintains a list of connected clients and their current guesses.

Client: Clients connect to the server, make a guess, and receive feedback from the server. The feedback will indicate whether the guess is too high, too low, or correct.

Socket: The server and clients communicate using sockets. Sockets are a way for programs to send and receive data over a network. Here is a more detailed breakdown of the process: The server generates a random number and starts listening for incoming connections. Clients connect to the server via sockets.

IV. IDEA USED

In a multiplayer/multi-client number guessing game using socket programming, multiple clients connect to a server via sockets and attempt to guess a randomly generated number. The server keeps track of the clients' guesses and sends messages to each client indicating whether their guess is too high, too low, or correct. The first client to guess the correct number wins the game. The game can be implemented using various programming languages such as C/C++,

A. Algorithm

Server side:

1. Generate a random number between 1 and 100 (inclusive).
2. Create a socket and bind it to a specific address and port.

- Listen for incoming connections on the socket.
- Initialize an empty dictionary to keep track of connected clients and their current guesses.
- Enter a loop to continuously listen for new connections.
- Receive the client's guess Compare the client's guess to the randomly generated number.
- If the guess is incorrect, send a message to the client indicating whether their guess was too high or too low.
- If the guess is correct, send a message to the client indicating that they have won the game, disconnect the client and break the loop.
- Continue the loop to listen for new connections.

B. Client side:

- Create a socket and connect it to the server's address and port.
- Continuously prompt the user for a guess. Send the user's guess to the server.
- Receive feedback from the server indicating whether the guess was too high, too low, or correct
- If the guess was correct, print a message indicating that the user has won the game or if somebody else has won the game and exit the program.
- If the guess was incorrect, print the feedback received from the server and prompt the user for another guess.
- Repeat steps 2 to 6 until the user guesses the correct number or the connection to the server is closed

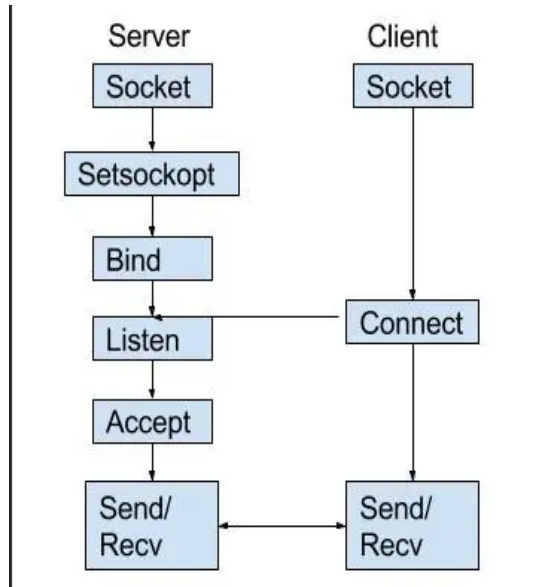


Fig. 1. Basic Server Client connection.

V. IMPLEMENTATION

The Server and Client Side Codes were implemented using C/C++ .The code sizes of important components are shown in Table I.

Concepts used in Implementation:

A. Multi-threading in Socket programming:

In socket programming, multithreading refers to the use of multiple threads to handle multiple clients or connections simultaneously. This allows the server to handle multiple requests at the same time, rather than having to handle them one at a time. This can improve the performance and scalability of the server. Each thread can handle a separate client connection, allowing the server to handle multiple connections simultaneously without blocking other connections.

TABLE I. Sizes of Important Code Components

Code Component	Code Size
Server.c	3.92 KB
Client.c	2.06 KB

Usage:

In a multiplayer number guessing game implemented using socket programming, threading can be used to handle multiple player connections simultaneously. Each player can be handled by a separate thread, allowing the game server to handle multiple players at the same time without blocking other players. For example, when a player connects to the server, a new thread can be spawned to handle that player's connection. This thread can then handle all communication with that player, such as sending and receiving messages, updating the player's score, and checking if the player has guessed the correct number. This allows the server to handle multiple players at the same time, rather than having to handle them one at a time, improving the performance and scalability of the game server. Additionally, it allows for each player to have their own separate game state, without interfering with the other players, even if the game is played simultaneously. Each thread can also handle the process of number generation, and number guessing independently, this allows for each player to play the game without waiting for other players.

Mutex:

A mutex (short for "mutual exclusion") is a synchronization object that is used to protect shared resources from concurrent access by multiple threads. The `pthread_mutex_lock()` function is used to acquire a lock on a mutex. When a thread calls this function, it will block (i.e. wait) until the lock is available. Once the lock is acquired, the thread can safely access the shared resource without fear of interference from other threads. When the thread is finished accessing the shared resource, it should call `pthread_mutex_unlock()` to release the lock, allowing other threads to acquire it.

In the given code, the lock is used to ensure that only one thread can access the "winner" variable at a time, preventing race conditions where multiple threads may try to update the variable simultaneously.

B.Header files used in the code:

`<stdio.h>` - This header file is used for input and output operations. It includes functions like `printf`, `scanf`, etc.

`<stdlib.h>` - This header file includes functions related to memory allocation, process control, and conversions. It includes functions like `malloc`, `exit`, `atoi`, etc.

`<string.h>` - This header file includes functions related to manipulating strings such as `strlen`, `strcpy`, `strcmp`, etc.

`<unistd.h>` - This header file includes functions that provide access to the operating system, such as `read`, `write`, `close`, `fork`, etc .

`<sys/types.h>` - This header file includes definitions of various data types used in system calls, such as `pid_t`, `size_t`, etc.

`<sys/socket.h>` - This header file is used for socket programming in C and includes functions like `socket`, `bind`, `listen`, `accept`, etc.

`<netinet/in.h>` - This header file includes constants and structures needed for internet domain addresses.

`<pthread.h>` - This header file includes functions for creating and manipulating threads.

C.Server Implementation:

Start the program.

Initialize a mutex variable (lock).

Initialize a variable to keep track of the winner (winner) and set it to 0.

Generate a random number (random_number).

Create a function to handle clients (handle_client).

In the `handle_client` function, create a loop to read in a client's guess.

Check if the client's guess is equal to the random number.

If it is, check if a winner has already been declared.

If not, set the winner variable to 1 and send a message to the client that they have won.

If a winner has already been declared, send a message to the client that they have guessed correctly but someone else has already won.

If the client's guess is not equal to the random number, check if it is higher or lower and send a message to the client accordingly.

In the main function, create a socket and bind it to a port number.

Start listening for incoming connections.

In a loop, accept incoming connections and create a new thread to handle each client using the `handle_client` function.

Clean up and exit the program when all clients have disconnected or the program is closed.

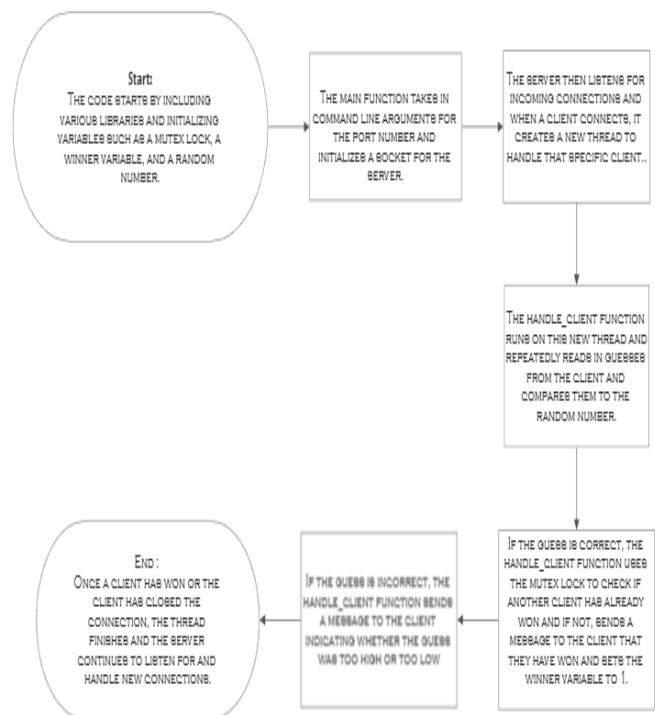


Fig. 2.Server Side Steps Involved.

D.Client Implementation:

The code starts by including several libraries and header files such as `stdio.h`, `stdlib.h`, `string.h`, `unistd.h`, `sys/types.h`, `sys/socket.h`, `netinet/in.h`, and `netdb.h`.

The main function takes in two arguments, the hostname and port number, and assigns the port number to a variable called `portno`.

The `socket` function is called to create a socket and the returned value is stored in the `sockfd` variable.

The `gethostbyname` function is used to get the server's address and the returned value is stored in the server variable.

The `bzero` function is used to set all values in the `serv_addr` variable to 0.

The server address and port number are set in the `serv_addr` variable.

The `connect` function is called to connect to the server using the `sockfd` and `serv_addr` variables.

The program enters a while loop that prompts the user to guess a number between 1 and 100, and the user's input is stored in the buffer variable.

The `write` function is called to send the user's input to the server and the returned value is stored in the `n` variable.

The `read` function is called to receive the server's response and the returned value is stored in the `n` variable.

The server's response is printed to the screen.

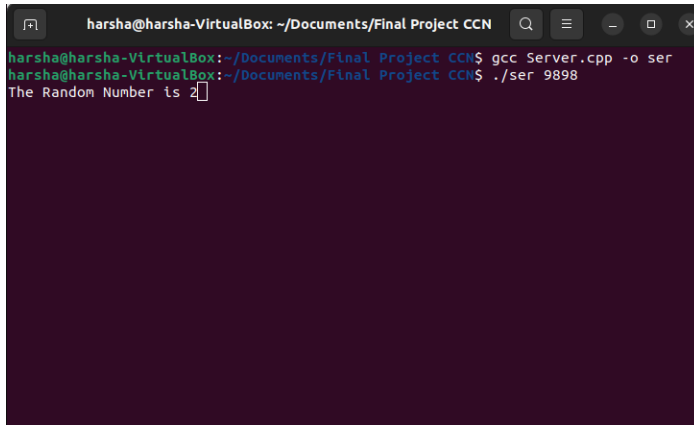
The `strcmp` function is used to compare the server's response to the string "Correct!" and the program exits the while loop if the strings match.

The number of tries the user took to guess the correct number

is printed and the program exits.

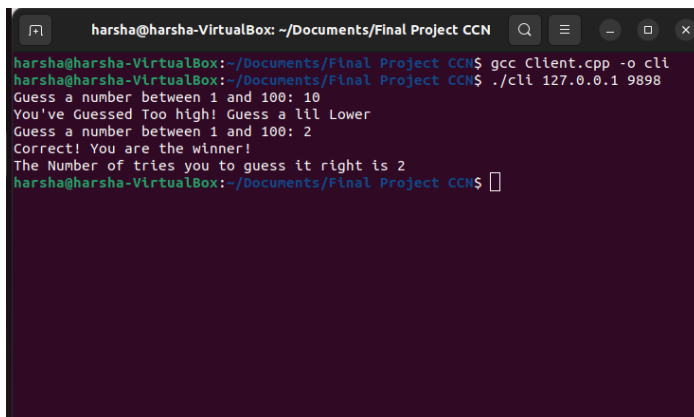
VI. TESTING AND EVALUATION

The implementation was tested in the same computer terminal of the Linux operating system with a terminal dedicated to the server and few other terminals for clients which connect to the server through the loopback address and the port number

A terminal window titled 'harsha@harsha-VirtualBox: ~/Documents/Final Project CCN'. The user has compiled 'Server.cpp' and executed it. The output shows a random number '2' and a prompt for a client to connect.

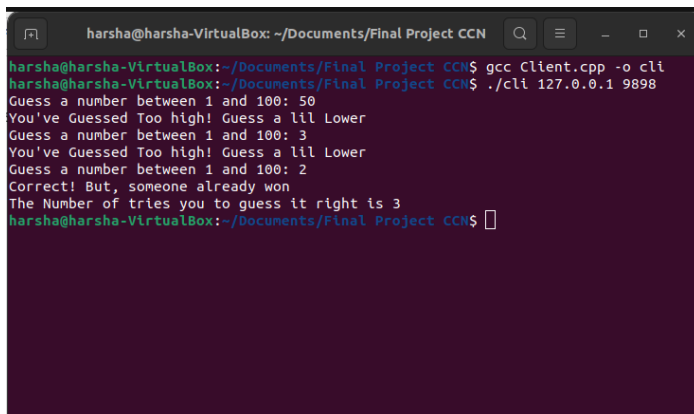
```
harsha@harsha-VirtualBox:~/Documents/Final Project CCN$ gcc Server.cpp -o ser
harsha@harsha-VirtualBox:~/Documents/Final Project CCN$ ./ser 9898
The Random Number is 2
```

Fig. 3.Server Side Terminal after execution.

A terminal window titled 'harsha@harsha-VirtualBox: ~/Documents/Final Project CCN'. The user has compiled 'Client.cpp' and executed it. The output shows a series of guesses (10, 3, 2) and the server's responses, eventually leading to a win with 2 tries.

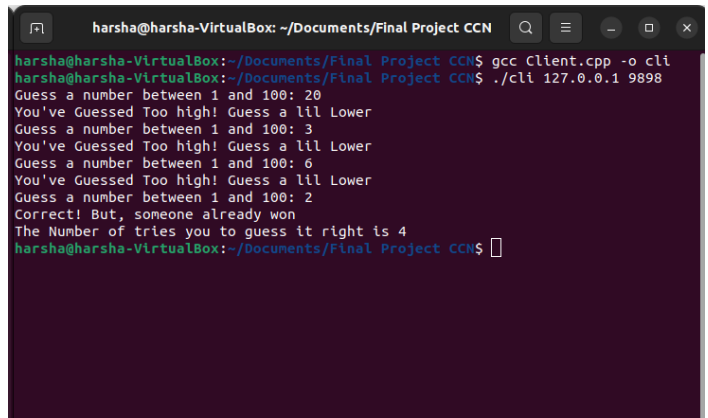
```
harsha@harsha-VirtualBox:~/Documents/Final Project CCN$ gcc Client.cpp -o cli
harsha@harsha-VirtualBox:~/Documents/Final Project CCN$ ./cli 127.0.0.1 9898
Guess a number between 1 and 100: 10
You've Guessed Too high! Guess a lil Lower
Guess a number between 1 and 100: 3
You've Guessed Too high! Guess a lil Lower
Guess a number between 1 and 100: 2
Correct! You are the winner!
The Number of tries you to guess it right is 2
```

Fig. 4.Winning Client terminal after guessing the number right.

A terminal window titled 'harsha@harsha-VirtualBox: ~/Documents/Final Project CCN'. The user has compiled 'Client.cpp' and executed it. The output shows a series of guesses (50, 3, 2) and the server's responses, eventually leading to a loss with 3 tries.

```
harsha@harsha-VirtualBox:~/Documents/Final Project CCN$ gcc Client.cpp -o cli
harsha@harsha-VirtualBox:~/Documents/Final Project CCN$ ./cli 127.0.0.1 9898
Guess a number between 1 and 100: 50
You've Guessed Too high! Guess a lil Lower
Guess a number between 1 and 100: 3
You've Guessed Too high! Guess a lil Lower
Guess a number between 1 and 100: 2
Correct! But, someone already won
The Number of tries you to guess it right is 3
```

Fig. 5.Client 2 side guessing number after a client winning

A terminal window titled 'harsha@harsha-VirtualBox: ~/Documents/Final Project CCN'. The user has compiled 'Client.cpp' and executed it. The output shows a series of guesses (20, 3, 6, 2) and the server's responses, eventually leading to a loss with 4 tries.

```
harsha@harsha-VirtualBox:~/Documents/Final Project CCN$ gcc Client.cpp -o cli
harsha@harsha-VirtualBox:~/Documents/Final Project CCN$ ./cli 127.0.0.1 9898
Guess a number between 1 and 100: 20
You've Guessed Too high! Guess a lil Lower
Guess a number between 1 and 100: 3
You've Guessed Too high! Guess a lil Lower
Guess a number between 1 and 100: 6
You've Guessed Too high! Guess a lil Lower
Guess a number between 1 and 100: 2
Correct! But, someone already won
The Number of tries you to guess it right is 4
```

Fig. 6.Client 3 side guessing the number after the winning client

VII. CONCLUSION

In conclusion, the implementation of a multi-client number guessing game using socket programming demonstrates the ability to create a simple yet functional network-based application. The game allows multiple clients to connect to the server and participate in a number guessing game simultaneously. The use of sockets enables real-time communication between the clients and the server, allowing for an interactive and engaging experience for the users. The use of threads ensures that each client's connection is handled separately, allowing for multiple games to be played simultaneously without interference. Overall, the application serves as a useful example of the power and flexibility of socket programming in creating network-based applications.

REFERENCES

- [1] Brian Hall, "Beej's Guide to Network Programming:Using Internet Sockets".
- [2] B. A. Forouzan, "TCP/IP Protocol Suite," McGraw-Hill, 2000
- [3] Lewis Van Winkle,"Hands-On Network Programming with C: Learn socket programming".