Documentation: Multi-user Chat System

Computer Architecture and Operating System

Code Description - Server

A socket is created for within the machine communication efficiently(AF_UNIX). Then bind() and listen() function are used to set the socket for use. Then a while loop is run and in the while loop, it awaits connection from clients. A new coming client is provided with a new socket using the accept() function. This is followed by a thread creation, whereby, the server is set to receive and send data to a particular socket of a client.

Important functions and structs used :

1. socket(): The *socket*() function shall create an unbound socket in a communications domain, and return a file descriptor that can be used in later function calls that operate on sockets.
```
int serverSocket = socket(AF_UNIX, SOCK_STREAM, 0);
```
2. struct sockaddr_un: It is struct for UNIX domain socket address related to the AF_UNIX socket family that is used to provide family and path to the struct.
```
struct sockaddr_un
{
  sa_family_t sun_family;          /* AF_UNIX */
  char sun_path[108];              /* Pathname */
};
```
3. bind(): The *bind*() function shall assign a local socket address *address* to a socket identified by descriptor *socket* that has no local socket address assigned.
```
if(bind(serverSocket,(struct sockaddr *) &serverAddr , len+1) == -1)
{
    perror("Bind failed.\nError ");
    return 0;
}
```
4. listen(): The *listen*() function shall mark a connection-mode socket, specified by the *socket* argument, as accepting connections.

Navya Aggarwal | 2018349|navya18349@iiitd.ac.in

```
if(listen(serverSocket,1024) == -1)
{
    perror("Listen failed.\nError ");
    return 0;
}
```

5. accept(): The *accept*() function shall extract the first connection on the queue of pending connections, create a new socket with the same socket type protocol and address family as the specified socket, and allocate a new file descriptor for that socket.
   int accept(int *socket*, struct sockaddr *restrict *address*, socklen_t *restrict *address_len*);

```
Client[clientCount].sockID = accept(serverSocket, (struct sockaddr*) &Client[clientCount].clientAddr, &Client[clientCount].len);
```

6. recv(): The *recv*() function shall receive a message from a connection-mode or connectionless-mode socket. It is normally used with connected sockets because it does not permit the application to retrieve the source address of received data.

```
int read = recv(clientSocket,data,1024,0);
if (read == -1)
{
    perror("Data recieve failed.\nError ");
    return 0;
}
```

7. send(): The *send*() function shall initiate transmission of a message from the specified socket to its peer. The *send*() function shall send a message only when the socket is connected (including when the peer of a connectionless socket has been set via *connect()*).

```
if (send(Client[i].sockID,data,1024,0) < 0)
    {
        printf("Unable to send Message \n");
        continue;
    }
```

Code Description - Client

Similar to Server, a socket is created. Then, an attempt is made by the accept() function to connect to the Server socket. Then, the program moves into a loop where data for communication is entered and transmitted using the send function.

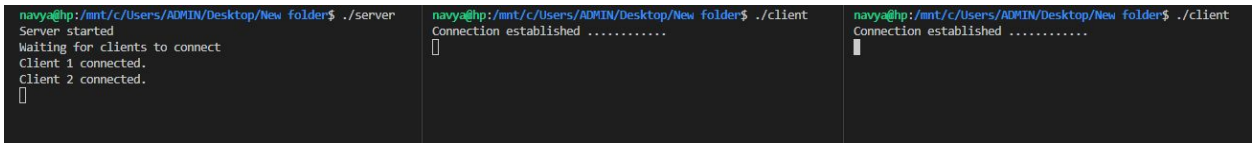
Important function used:
connect() : The *connect*() function shall attempt to make a connection on a socket.

```
int con = connect(clientSocket,(struct sockaddr*) &serverAddr, sizeof(serverAddr));
if(con == -1)
{
    perror("Connection failed.\nError ");
    return 0;
}
```

Compilation

The file is compiled using a Makefile and the executables formed can be executed using the commands mentioned below.

We require multiple terminals for executing the code. One of the terminals works as a server terminal. Other multiple terminals are used as multiple clients.

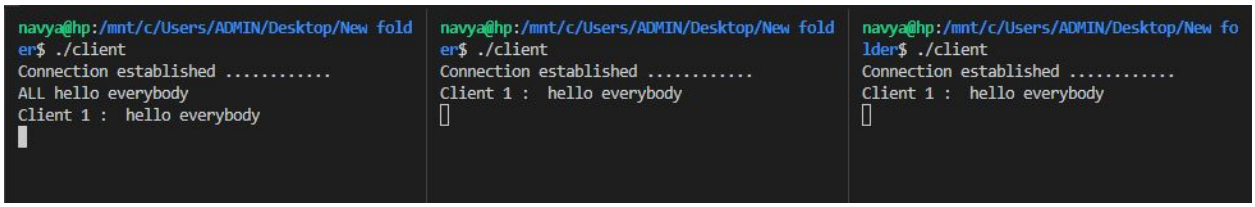


Commands to Test

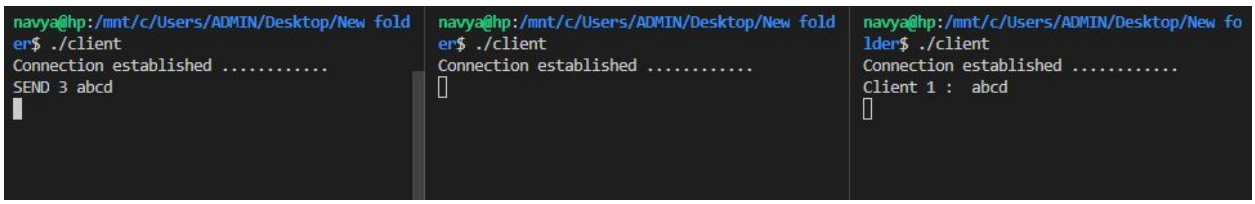
1. SEND <Client_index> <message> : sends message to a particular client. If the Client is not present, an error is reported by the Server.
2. ALL <message>: sends a message to all the connected clients and itself.
3. CONNECTED: shows on the working client, the list of all connected and disconnected Clients and their assigned socket numbers.
4. EXIT: terminates the connection of a client with the server.

Input and Output

1. ALL hello everybody

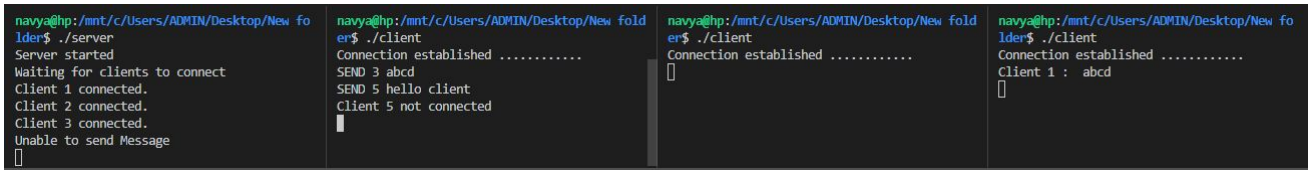


2. SEND 3 hello abcd

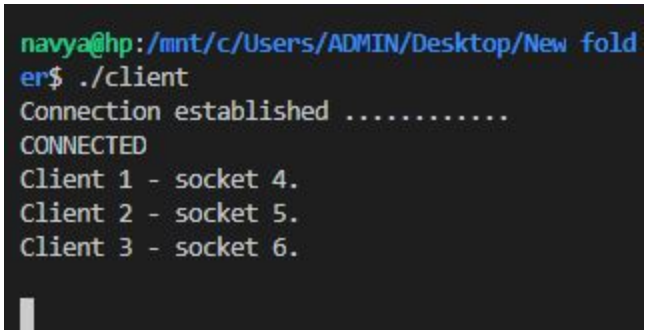


Navya Aggarwal | 2018349|navya18349@iiitd.ac.in

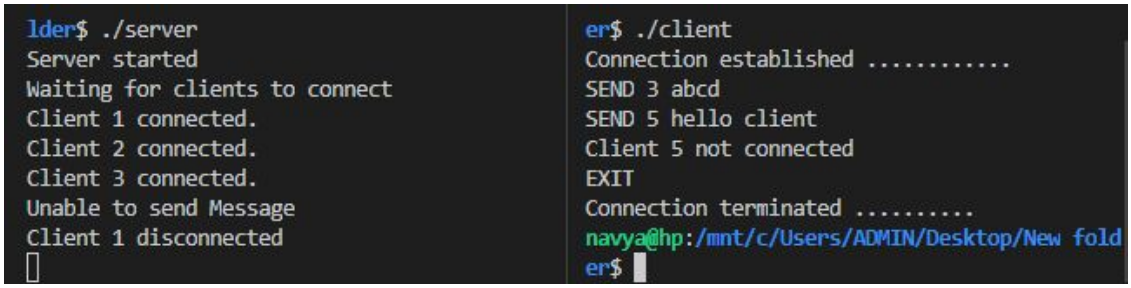
3. SEND 5 hello client



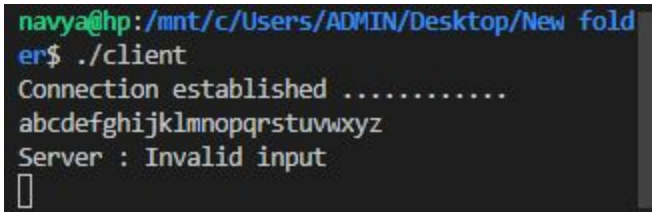
4. CONNECTED



5. EXIT



6. abcdefghijklmnopqrstuvwxyz



Errors Handled

1. When both Server and Client send or receive data, create sockets or threads, use functions such as connect, bind, listen or accept checks are performed to ensure that the program exits safely if something fails.
2. Strict check is present to check that entered messages from any Client is correctly written and any invalid input is reported by the Server.
3. If a particular Client doesn't exist or it has been disconnected, then an error is sent by the Server if we try to send a one-to-one message to it.

References

1. Code: https://gist.github.com/Abhey/47e09377a527acfc2480dbc5515df872
2. Documentation: https://pubs.opengroup.org/onlinepubs/009695399/functions/*.html