

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

18CSC302J-COMPUTER NETWORKS

BATCH – 2 , SEMESTER – 5

YEAR 2019-20

|  |  |
| --- | --- |
| REGISTRATION NUMBER | RA1811028010069 |
| NAME | M. Sai Krishna Teja |

**B.tech-CSE-CC, Third Year (Section: J2)**

**Faculty Incharge: Vaishnavi Moorthy, Asst Prof/Dept of CSE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INDEX** | | | | |
| **Ex.no** | **Topic** | **Date of completion** | **Marks** | **Faculty signature** |
| **Assessment-1** | | | | |
| 1 | **Standard Library for Socket IO** | **05/08/2020** |  |  |
| 2 | **TCP/IP Client-Server Communication** | **13/08/2020** |  |  |
| 3 | **UDP Echo Client-Server Implementation** | **22/08/2020** |  |  |
| 4 | **Concurrent TCP/IP Date-Time Server** | **29/08/2020** |  |  |
| 5 | **Half Duplex Chat using TCP/IP** | **05/09/2020** |  |  |

**Lab Exercise: 1**

**Aim: Study of basic commands for socket programming**

***man*** command in Linux is used to display the user manual of any command that we can run on the terminal.

**Syntax: man [option] [command name]**……option here gives the page we want to view in that command function of the manual.

**1.stdio - standard input/output library functions**

Description:

It contains all the input/output functions in this library manual of library **libc**. Input and output is mapped into logical data streams and the physical I/O characteristics are concealed.

Command: man stdio

**2. string- Contains all the string operations**

Description:

String functions like strcat , strcmp , strchr , etc are included under this command and passed as a function with arguments in it. Performs operations only on null terminated strings (‘\0’) and should have header function “string.h”.

Command: man string

**3. net db- database access methods**

Description:

Dbopen() is the library interface used here and it is used to read / write the files. The files under this are not stored in the disk. These are database files like hashed files , btree format files , etc.

Command: man net db

**4. time- run programs and summarize system resource usage**

Description:

Basically , a time command is given with some arguments like “time wc /etc/hosts” . So now this command gives the information about the resource i.e. (hosts in this ex:). If no arguments or format is used then it returns default time command.

Command: man time

**5. date- print or set the system date and time**

Description:

It just displays the current date and time in the specified format of that argument.

Command: man date

**6. stat- display file or file system status**

Description:

Displays the file with its name , its location ,owner name ,etc. File system status like last modification of file , last access of file , etc.

Command: man stat

**7. ioctl- control device**

Description:

The ioctl() system call manipulates the underlying device parameters of special files i.e. special files are interfaced to that particular device driver and this command manipulates this file of that device to appear as a normal file in that file system.

Command: man ioctl

**8. sys- System interface**

Description:

It performs many system calls like val executable\_name , val is\_directory ,etc as it acts as a system interface to invoke any program or file or directory ,etc according to arguments given.

Command: man sys

**9. errno- look up errno names and descriptions**

Description:

errno looks up errno macro names, errno codes, and the corresponding descriptions. Suppose if we access any file or dir. Not present , it shows “No such file or dir.” . This statement shows that it is an error and errno is responsible for it.

Command: man errno

**10. pcap-filter - packet filter syntax**

Description:

This command compiles the statements into a filter program and turns them into packets of data and performs operations according to arguments given like reaching these filtered packets to destination address from host address, etc.

Command: man pcap-filter

**11. inet- Internet address manipulation routines**

Description:

It converts IPv4 host address into binary , octal , hexadecimal form ,etc and vice versa.

Command: man inet

**12. system- execute a shell command**

Description:

It performs fork and child processes i.e. it creates a fork to execute that child processes in that program of the file.

Command: man system

**13. read- read from a file descriptor**

Description:

It counts bytes from file descriptor (indicator used to access file ,etc) by incrementing the count from initial position in the file.

Command: man read

**14. write- send a message to another user**

Description:

It is used to communicate with other hosts by sending messages. Other hosts identify our message by our host name.

Command: man write

**15. gethostname- get/set hostname**

Description:

It gets others null-terminated hostname by gethostname and can change the hostname by sethostname.

Command: man gethostname

**16. gethostbyname- get network host entry**

Description:

The gethostbyname() function returns a structure of type hostent for the given host name i.e. hostent is known as the structure that describes that internet host and it returns this structure only and gets entry.

Command: man gethostbyname

**17. htons- convert values between host and network byte order**

Description:

The htons() function converts the unsigned short integer hostshort from host byte order to network byte order.

Command: man htons

**18. htonl- convert values between host and network byte order**

Description:

The htonl() function converts the unsigned integer hostlong from host byte order to network byte order.

Command: man htonl

**htons- ‘s’ means short integer value**

**htonl- ‘l’ means long ineger value**

**19. bind- bind a name to a socket**

Description:

When a socket is created it has no address initially. But our file descriptor (i.e. the file) to go to that socket we use bind.

Command: man bind

**20. socket- create an endpoint for communication**

Description:

For a file descriptor containing data packets to reach the end point of the data communication we need a socket and this assigns sockets address.

Command: man socket

**21. ifconfig- configure a network interface**

Description:

Ifconfig is used to configure the kernel-resident network interfaces i.e. it changes the configuration of networks around the central host network.

Command: man ifconfig

**22. send- send a message on a socket**

Description:

Here the message is sent from one socket to another socket ( should be connected). Difference between **write and send** is both communicate by sending messages but **flag** plays an important role.

Command: man send

**23. recv- receive a message from a socket**

Description:

Here message is received from one socket to another. Difference between **read and recv** is also **flags.**

Command: man recv

**24. accept- accepts jobs sent to destination**

Description:

When data packets are sent from source and later when source receives acknowledgment , this command helps to accept that data packet.

Command: man accept

**Result:** Thus , all commands are verified.

**Lab Exercise - 2**

**Aim: TCP/IP Server- Client Implementation**

**Programming Language: C**

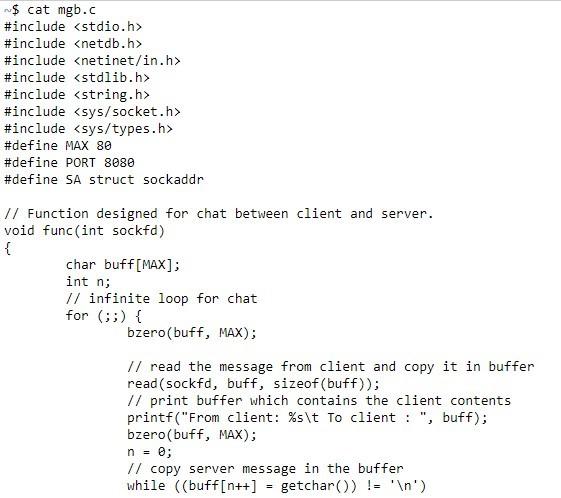
**Algorithm for TCP server connection:**

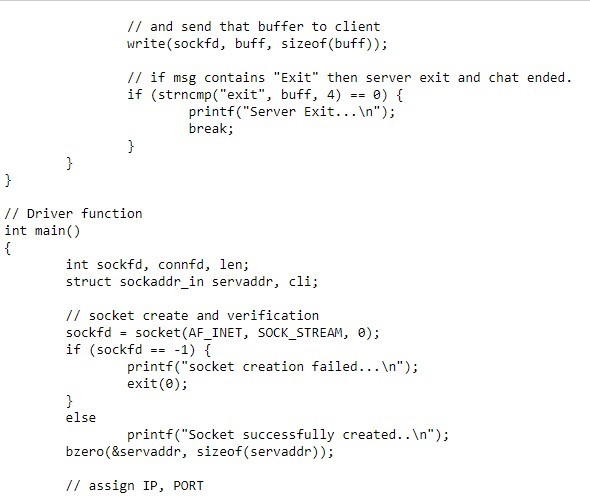
1. with the help of **create()** , a socket is created for the TCP connection.
2. with the help of **bind()**, this socket (initially with no address) gets connected to TCP/IP server address.
3. with the help of **listen()**, a synchronous sequence number is sent to client in passive mode which happens in openend() communication.
4. With help of **accept()**, both synchronous and acknowledgment are received from client side with help of sequence numbers and connection is established between both and transfers data and repeat the listen() function for further process.

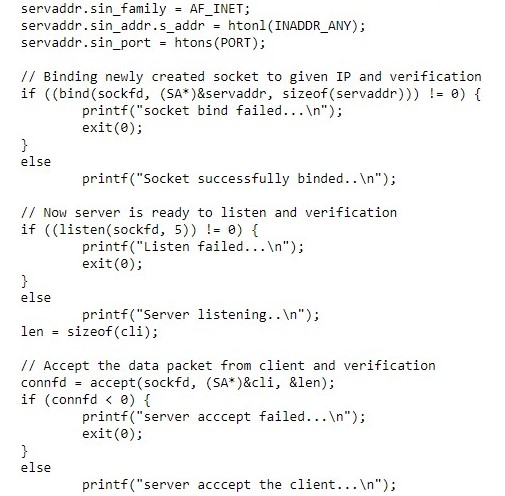
**Algorithm for TCP Client connection:**

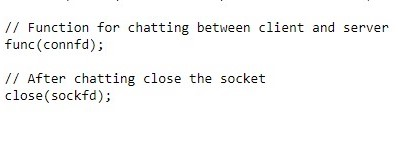
1. with the help of **create()**, create socket for client side connection.
2. with the help of **connect()**, connect with server.

**SERVER CODE:**

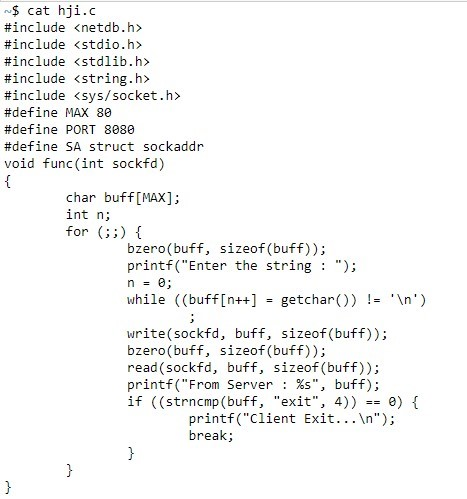


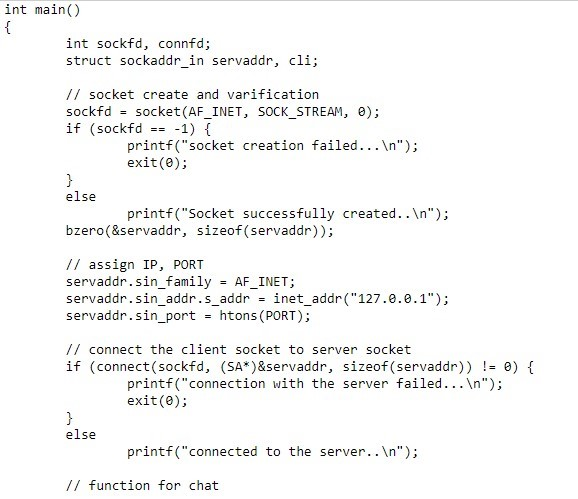


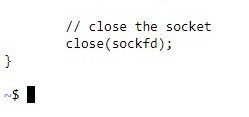




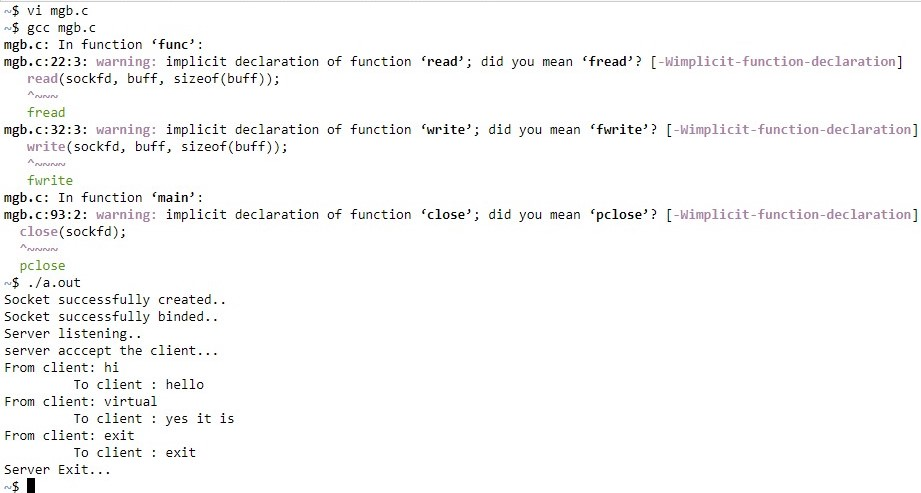
**CLIENT CODE:**



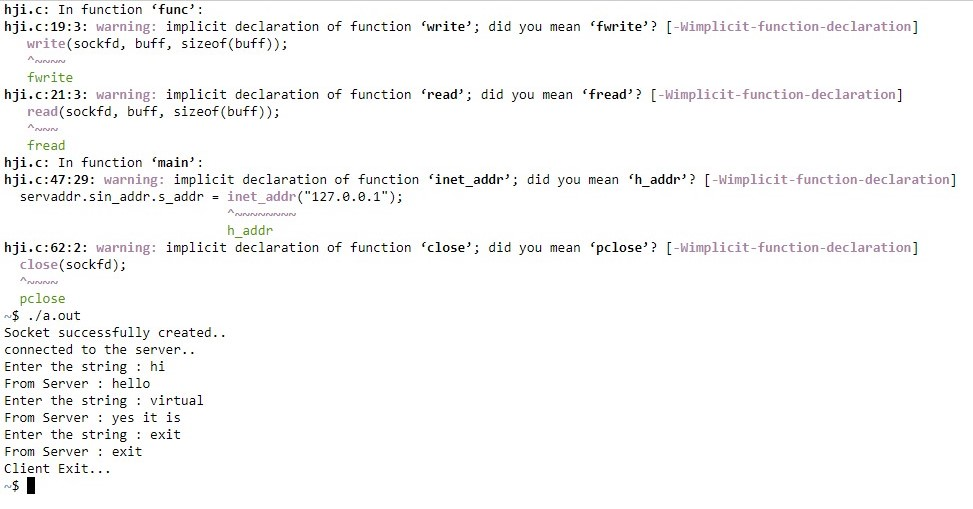




**SERVER OUTPUT:**



**CLIENT OUTPUT:**



**Result:** Thus , TCP/IP client-server communication is obtained.

**Lab Exercise - 3**

**Aim: UDP Echo Server-Client Implementation**

**Programming Language: C**

**Algorithm for Echo Server:**

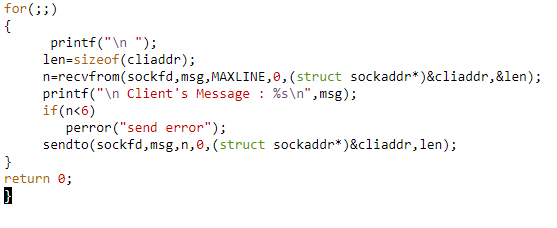
1. Create UDP socket.
2. Bind the socket to the server address using **bind()** function.
3. Wait until the datagram packet arrives from the client using **recv()** function.
4. Process the datagram packet and send a reply to the client as an echo.
5. Repeat the steps from Step 3.

**Algorithm for Echo Client:**

1. Create UDP socket.
2. Send a message to the server using **sendto()** function.
3. Wait until a response from the server is received using **recv()** function.
4. Get the server’s echo back using **recvline()** function and repeat steps from step 2, if necessary.
5. Close socket descriptor and exit.

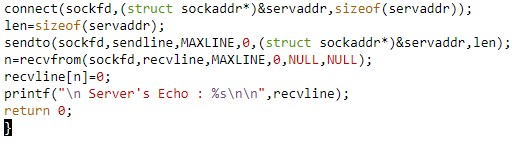
**Echo Server Code:**



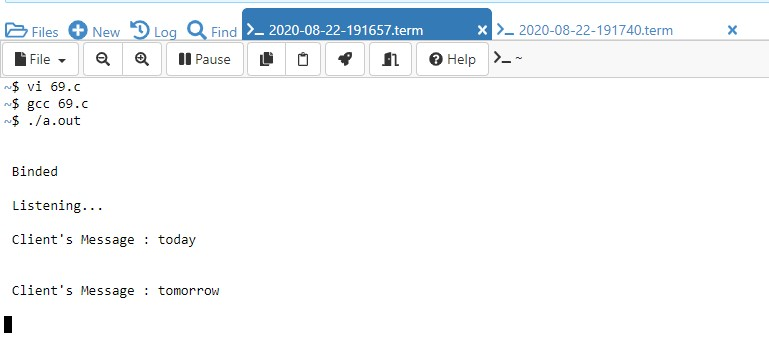


**Echo Client Code:**

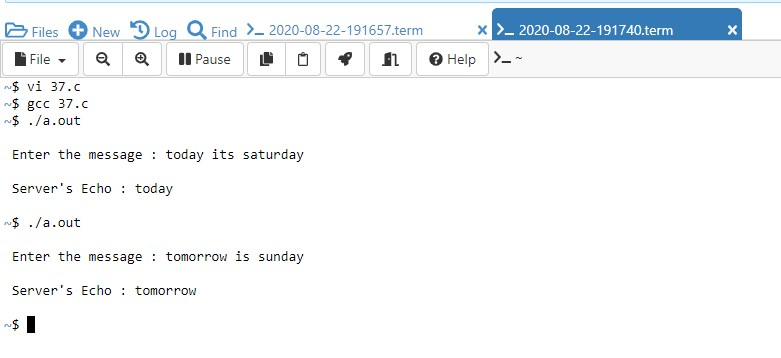




**Echo server Output:**



**Echo Client output:**



**Result:** Thus , UDP echo between client and server is obtained.

**Lab Exercise -4**

**Aim: Concurrent TCP/IP Date-Time Server**

**Programming Language: C**

**Algorithm for Server Date-Time:**

1. Create a TCP socket.
2. Bind the socket to the server address using **bind()** function and listen to client using **listen()** function and waits.
3. After the Server receives the acknowledgement from the client , it accepts the client request using **accept()** function.
4. Server sends the synchronization message initially with **time\_t** set to null pointer. Later it takes the server time and sends the synchronization message to the client and waits for acknowledgement from the client to display in string format in the server using **ctime .**

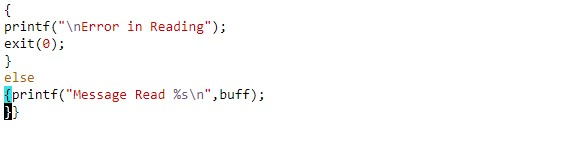
**Algorithm for Client Date-Time:**

1. Create a TCP socket.
2. Connect it to the server socket using the port address with the help of **connect()** function.
3. Read the synchronization message received from the server using **read()** function and send the syn+ack to the server for further process.
4. Later the server accepts the acknowledgment.
5. Close socket descriptor and exit.

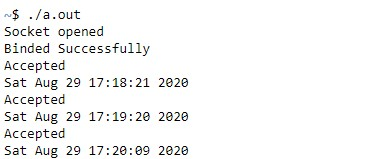
**Server Code:**



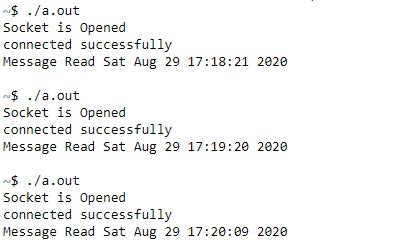
**Client Code:**



**Server Output:**



**Client Output:**



**Result:** Thus , concurrent TCP/IP Date-time server and client are obtained.

**Lab Exercise -5**

**Aim :** To implement a half duplex application, where the Client establishes a connection with the Server. The Client can send and the server well receive messages at the same time.

**Programming Language : C**

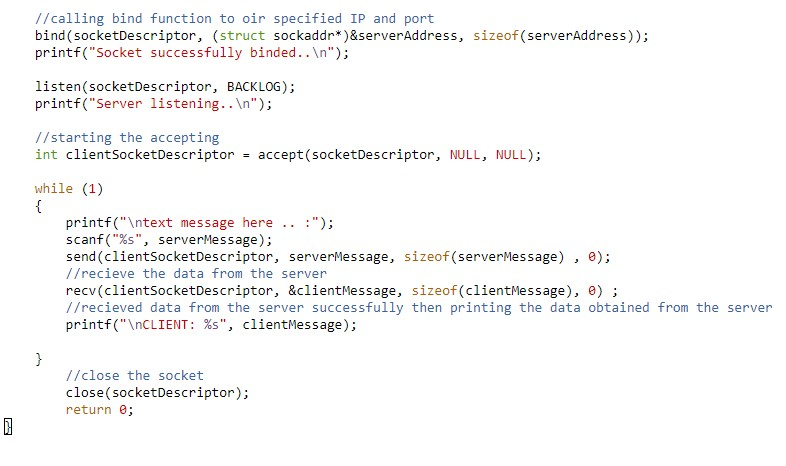
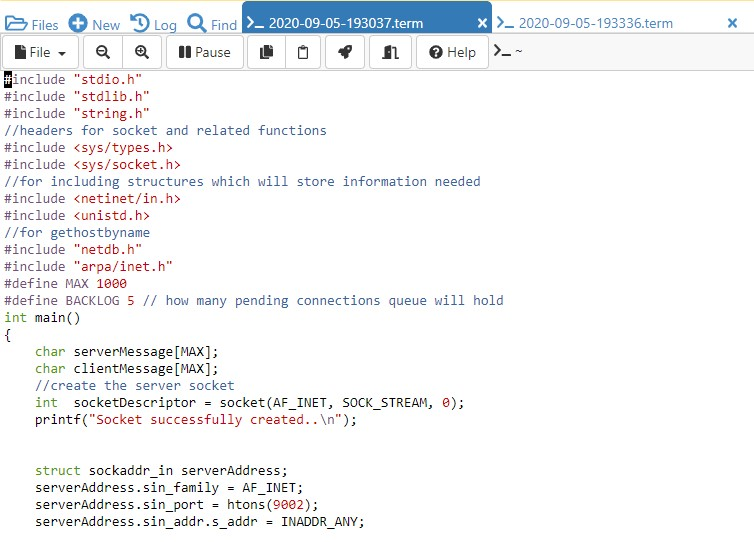
**Algorithm for half duplex communication in server:**

1. Create a TCP socket using a **socket()** function.
2. Bind the socket to the server address using **bind()** and listen and accept client requests or messages using **listen()** and **accept()** functions respectively.
3. As the server initiates the first **syn+ack** , send the message to the client using **send()** function .
4. Server also receives the messages from the client in **syn** manner i.e. sequence numbers using **recv()** function.
5. Then close the server using **close()** function.

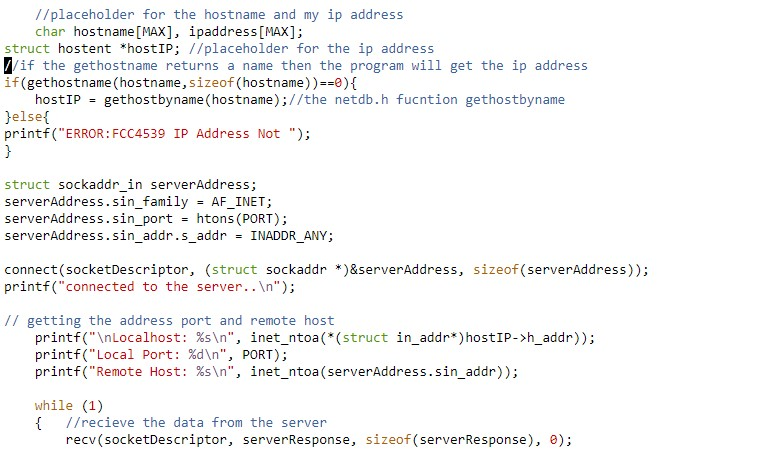
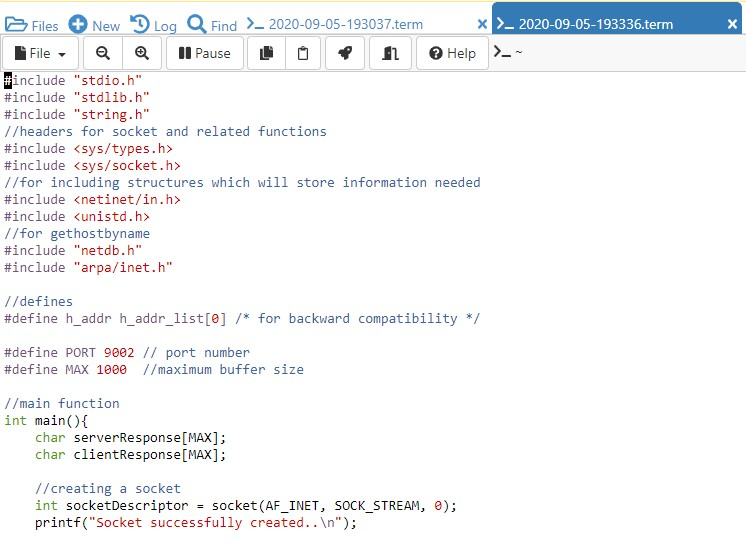
**Algorithm for half duplex communication in client:**

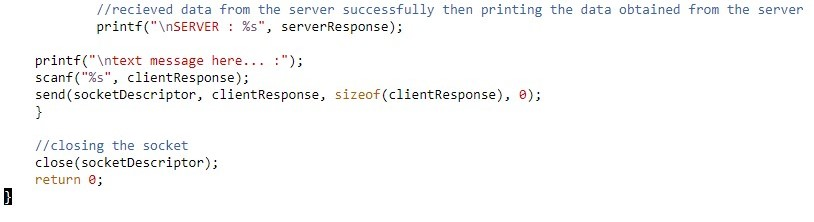
1. Create a TCP socket using **socket()** function.
2. Using the socket address of the server we get the **local host , local port , remote host** of the server side. Here we get the remote host address as 0’s as it only one server and thus gives only local host address using **gethostname()** function.
3. Client gets connected to the server using **connect()** function.
4. Now the client receives the message from the server by **ack** form and sends the message by **syn** form in the form of sequence numbers using **recv()** and **send()** functions respectively.
5. Then close the client using **close()** function.

In half duplex communication , the server and the client can send and receive messages but not at the same time. Example for it is WalkieTalkie.

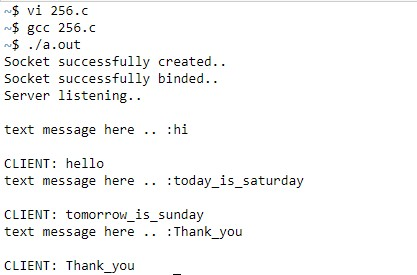
**Server Code:**

**Client Code:**

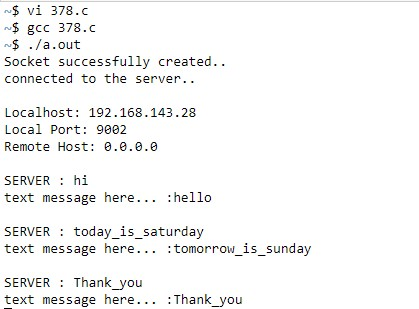




**Server Output:**



**Client output:**



**Result:** Thus , Half Duplex communication between server and client is obtained.