**Assignment 1**

**Computer Network**

**Program: MScIT Sem-2**

**Group ID : 28**

**Student Name Student ID**

Dev Adnani 202212012

Saif Saiyed 202212083

**202212012**

2.6 Exercise

1. Open a terminal

2. Type “gcc server.c -o server”, and press enter.

3. Type “./server”, and press enter.

4. Open 2nd terminal

5. Type “gcc client.c -o client”, and press enter.

6. Type “./client”, and press enter.

7. Both the server and client are running simultaneously.

8. Note: Always run the server first.

Server Code (server.c)

// Server side C/C++ program to demonstrate Socket programming

#include <unistd.h>

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <string.h>

#define PORT 8080

int main(int argc, char const \*argv[])

{

int server\_fd, new\_socket, valread;

struct sockaddr\_in address;

int opt = 1;

int addrlen = sizeof(address);

char buffer[1024] = {0};

char \*hello = "Hello from server";

// Creating socket file descriptor

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0)

{

perror("socket failed");

exit(EXIT\_FAILURE);

}

// Forcefully attaching socket to the port 8080

if (setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT, &opt, sizeof(opt)))

{

perror("setsockopt");

exit(EXIT\_FAILURE);

}

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons( PORT );

// Forcefully attaching socket to the port 8080

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address))<0)

{

perror("bind failed");

exit(EXIT\_FAILURE);

}

if (listen(server\_fd, 3) < 0)

{

perror("listen");

exit(EXIT\_FAILURE);

}

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen))<0)

{

perror("accept");

exit(EXIT\_FAILURE);

}

valread = read( new\_socket , buffer, 1024);

printf("%s\n",buffer );

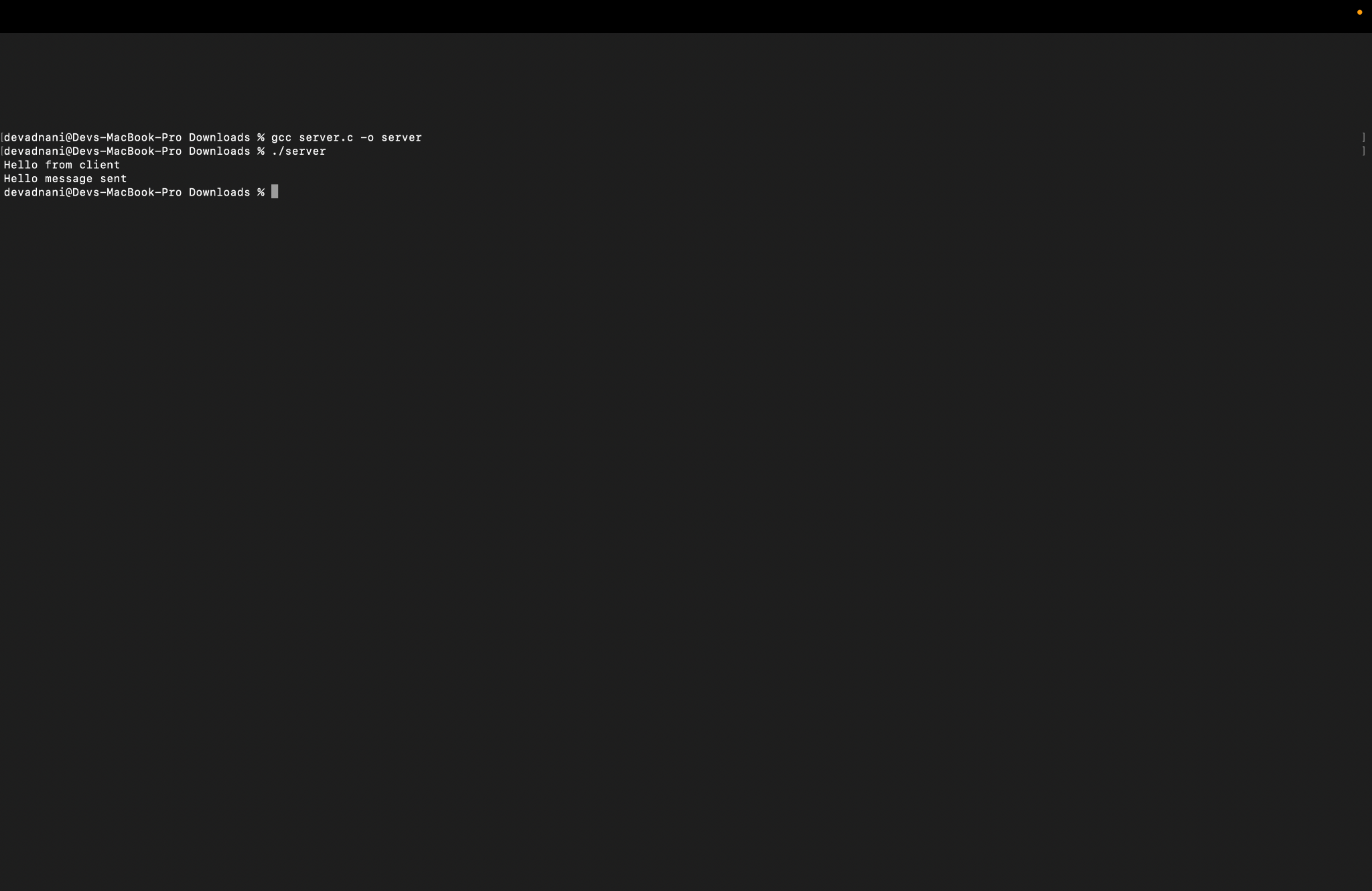
send(new\_socket , hello , strlen(hello) , 0 );

printf("Hello message sent\n");

return 0;

}

Output:



Explanation:

* This is a basic server code in C for a socket application. The server creates a socket and binds it to IP address INADDR\_ANY and port number 8080.
* Once a connection is established, it reads data sent by the client and prints it to the console. It then sends a message "Hello from server" to the client and prints "Hello message sent" to the console.
* The read() and send() functions are used to read data from and send data to the connected client, respectively.
* It uses standard C libraries such as stdio.h, unistd.h, sys/socket.h, stdlib.h, and netinet/in.h. It uses the socket() function to create a socket and bind() function to bind the socket to the IP address and port.

Client Code (client.c)

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#define PORT 8080

int main(int argc , char const \*argv[])

{

int sock = 0, valread;

struct sockaddr\_in serv\_addr;

char \*exit\_msg = "exit" , \*msg;

char buffer[1024]={0};

if((sock = socket(AF\_INET, SOCK\_STREAM , 0))<0)

{

printf("\n Socket creation error\n");

return -1;

}

serv\_addr.sin\_family =AF\_INET;

serv\_addr.sin\_port = htons(PORT);

if(inet\_pton(AF\_INET,"127.0.0.1", &serv\_addr.sin\_addr)<=0)

{

printf("\n Invalid address / address not supported \n");

return -1;

}

if(connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr))<0)

{

printf("\n connection Failed \n");

return -1;

}

while(1){

scanf("%s",msg);

if(!strcmp(msg, exit\_msg)){

close(sock);

return 0;

}

send(sock, msg , strlen(msg) , 0);

valread = read(sock, buffer, 1024);

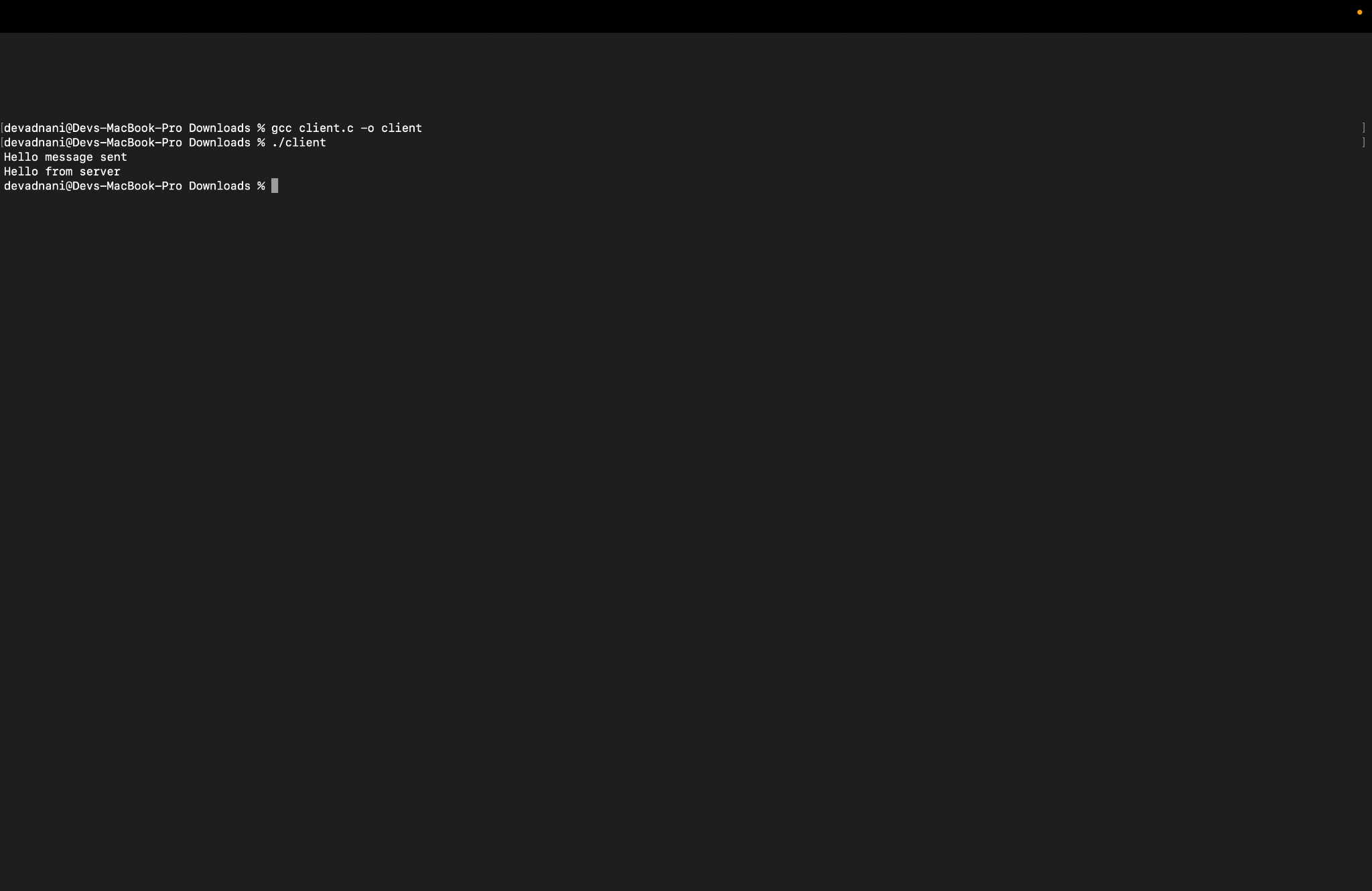
printf("%s\n",buffer);

}

return 0;

}

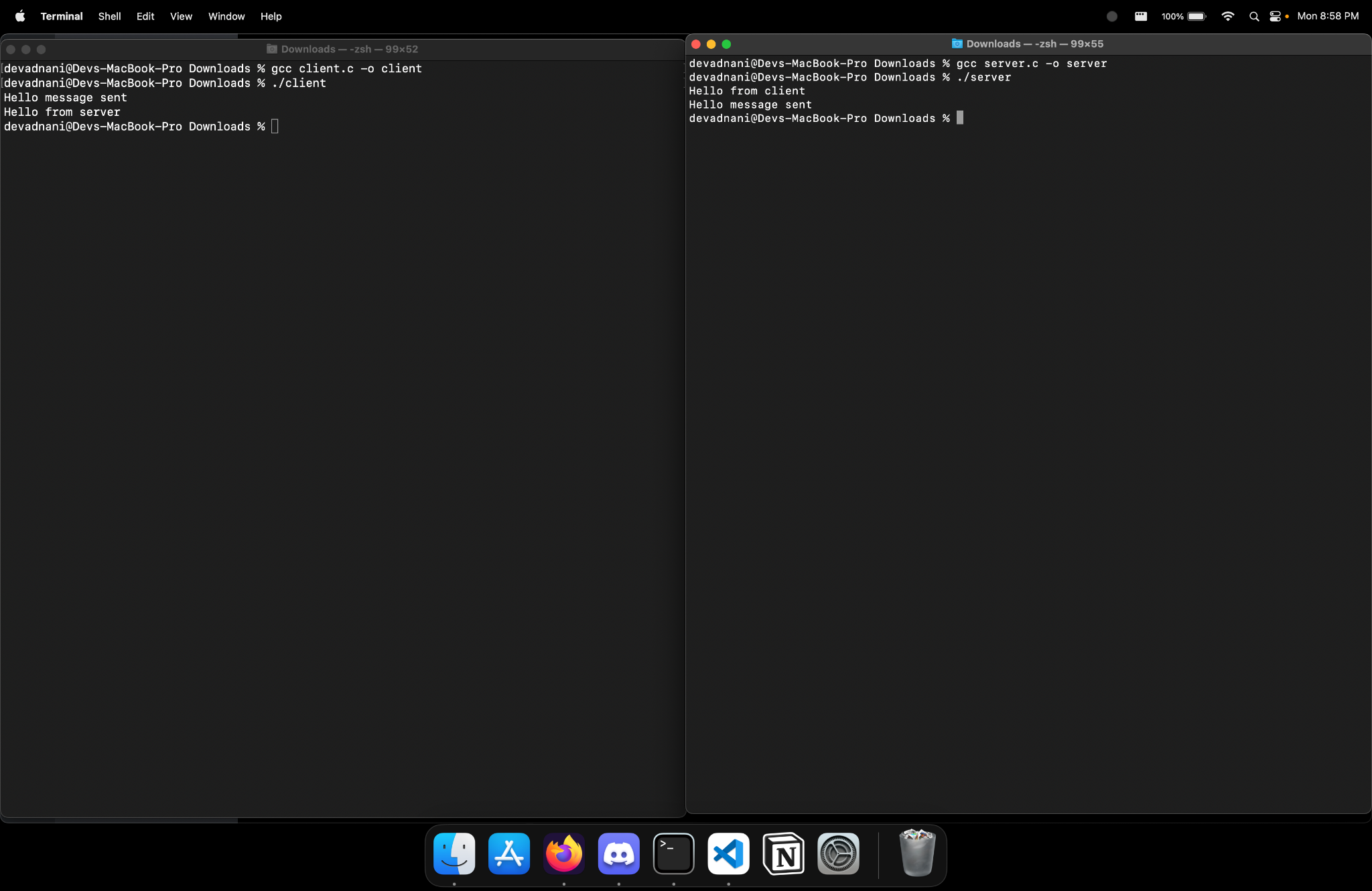
Output:



Explanation:

* This is a basic client code in C for a socket application. The client uses the socket API to create a socket and connect to a server at IP address "127.0.0.1" (localhost) and port number 8080.
* The client then enters into a loop where it prompts the user for input, sends that input to the server, and receives a response from the server. If the user enters "exit", the loop breaks and the program terminates.
* The socket is closed before the program exits.

Client & Server:



3.3 Exercise

1. Create TCP server and client using socket programming. Make them communicate with each other by making a question and answer system between them. (At Least 4 different questions should be there)

This application enables a client to connect to a server and communicate with each other by sending packets between them using TCP server. If the message contains **‘exit’** then server exit and chat ended.

server.c

// Server side C/C++ program to demonstrate Socket

// programming

#include <netinet/in.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#define PORT 8080

int main(int argc, char const\* argv[])

{

int server\_fd, new\_socket, valread;

struct sockaddr\_in address;

int opt = 1;

int addrlen = sizeof(address);

char buffer[1024] = { 0 };

char\* hello = "Hello from server";

char\* middle = "I Am fine";

char\* bye = "What About You ?";

char\* end = "I'll Leave Will Finish Some Work";

// Creating socket file descriptor

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("socket failed");

exit(EXIT\_FAILURE);

}

if (setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR , &opt, sizeof(opt))) {

perror("setsockopt");

exit(EXIT\_FAILURE);

}

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

// Forcefully attaching socket to the port 8080

if (bind(server\_fd, (struct sockaddr\*)&address,

sizeof(address))

< 0) {

perror("bind failed");

exit(EXIT\_FAILURE);

}

if (listen(server\_fd, 3) < 0) {

perror("listen");

exit(EXIT\_FAILURE);

}

if ((new\_socket

= accept(server\_fd, (struct sockaddr\*)&address,

(socklen\_t\*)&addrlen))

< 0) {

perror("accept");

exit(EXIT\_FAILURE);

}

valread = read(new\_socket, buffer, 1024);

printf("%s\n", buffer);

send(new\_socket, hello, strlen(hello), 0);

printf("Hello message sent\n");

valread = read(new\_socket, buffer, 1024);

printf("%s\n", buffer);

send(new\_socket, middle, strlen(hello), 0);

printf("I am fine sent\n");

valread = read(new\_socket, buffer, 1024);

printf("%s\n", buffer);

send(new\_socket, bye, strlen(hello), 0);

printf("What About You ? sent\n");

valread = read(new\_socket, buffer, 1024);

printf("%s\n", buffer);

send(new\_socket, end, strlen(hello), 0);

printf("I'll Leave Will Finish Some Work? sent\n");

// closing the connected socket

close(new\_socket);

// closing the listening socket

shutdown(server\_fd, SHUT\_RDWR);

return 0;

}

Output :

Go to the terminal and execute the command for the server.

gcc server.c -o server -> This command is used to compile the server file. After running this command a new executable file is created.

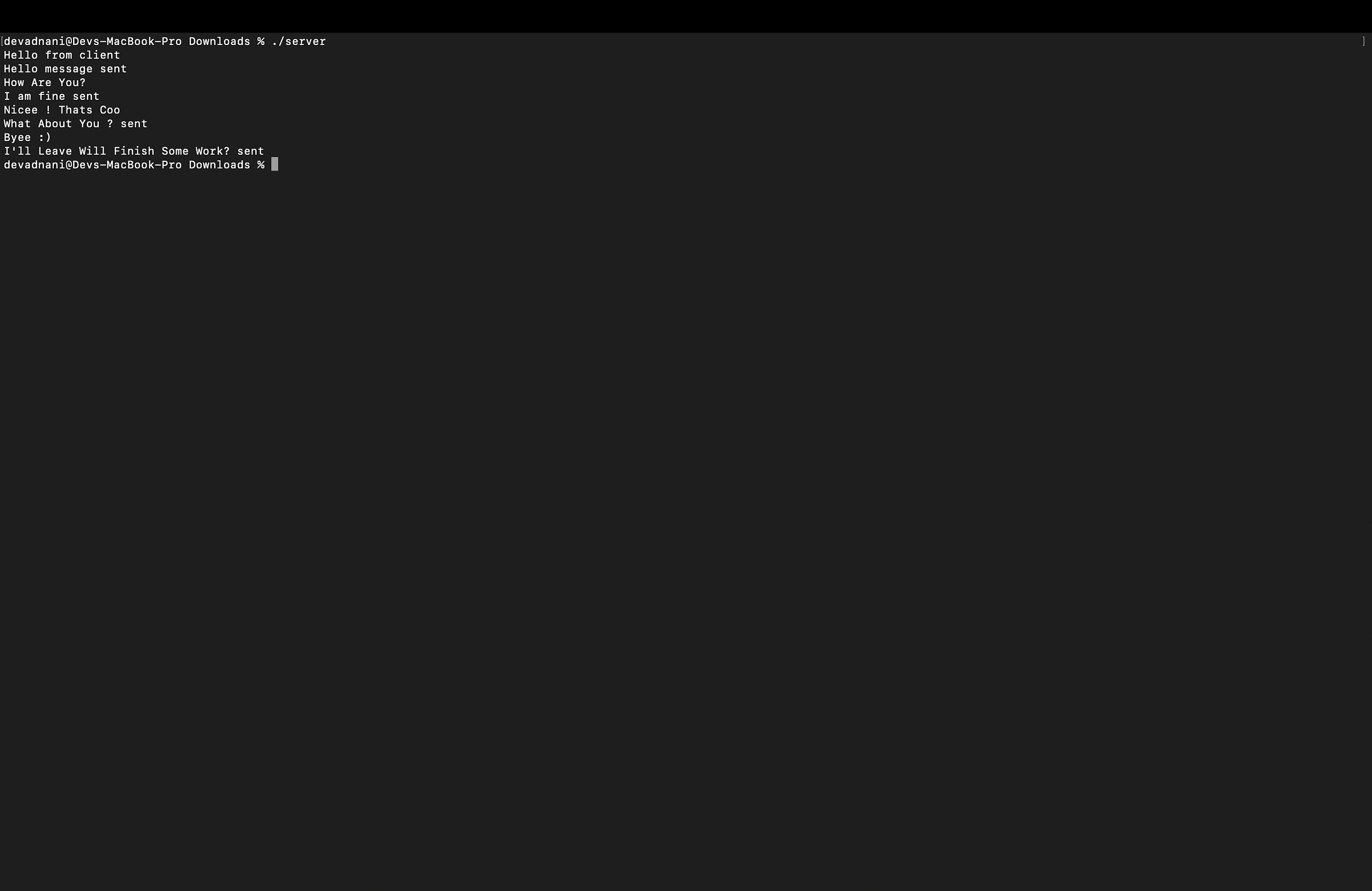
./server -> This command is used to run executable file.

After this, a socket is created using the socket function.

Using bind function we can assign a unique address to the newly created socket.

To accept the client request, it uses a listen function.

When client sent connection request, the server accept that connection request using accept function.After accepting client request message can be transferred between server and client.



Client.c

// Client side C/C++ program to demonstrate Socket

// programming

#include <arpa/inet.h>

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#define PORT 8080

int main(int argc, char const\* argv[])

{

int sock = 0, valread, client\_fd;

struct sockaddr\_in serv\_addr;

char\* hello = "Hello from client";

char\* middle = "How Are You?";

char\* bye = "Nicee ! Thats Cool";

char\* end = "Byee :)";

char buffer[1024] = { 0 };

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

printf("\n Socket creation error \n");

return -1;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

// Convert IPv4 and IPv6 addresses from text to binary

// form

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr)

<= 0) {

printf(

"\nInvalid address/ Address not supported \n");

return -1;

}

if ((client\_fd

= connect(sock, (struct sockaddr\*)&serv\_addr,

sizeof(serv\_addr)))

< 0) {

printf("\nConnection Failed \n");

return -1;

}

send(sock, hello, strlen(hello), 0);

printf("Hello message sent\n");

valread = read(sock, buffer, 1024);

printf("%s\n", buffer);

send(sock, middle, strlen(hello), 0);

printf("How Are You? sent\n");

valread = read(sock, buffer, 1024);

printf("%s\n", buffer);

send(sock, bye, strlen(hello), 0);

printf("Nicee ! Thats Cool \n");

valread = read(sock, buffer, 1024);

printf("%s\n", buffer);

send(sock, end, strlen(hello), 0);

printf("Byee :) sent\n");

valread = read(sock, buffer, 1024);

printf("%s\n", buffer);

// closing the connected socket

close(client\_fd);

return 0;

}

Output :

Open another terminal and execute the client’s command.

gcc client.c -o client -> This command is used to compile the client file. After running this command a new executable file is created.

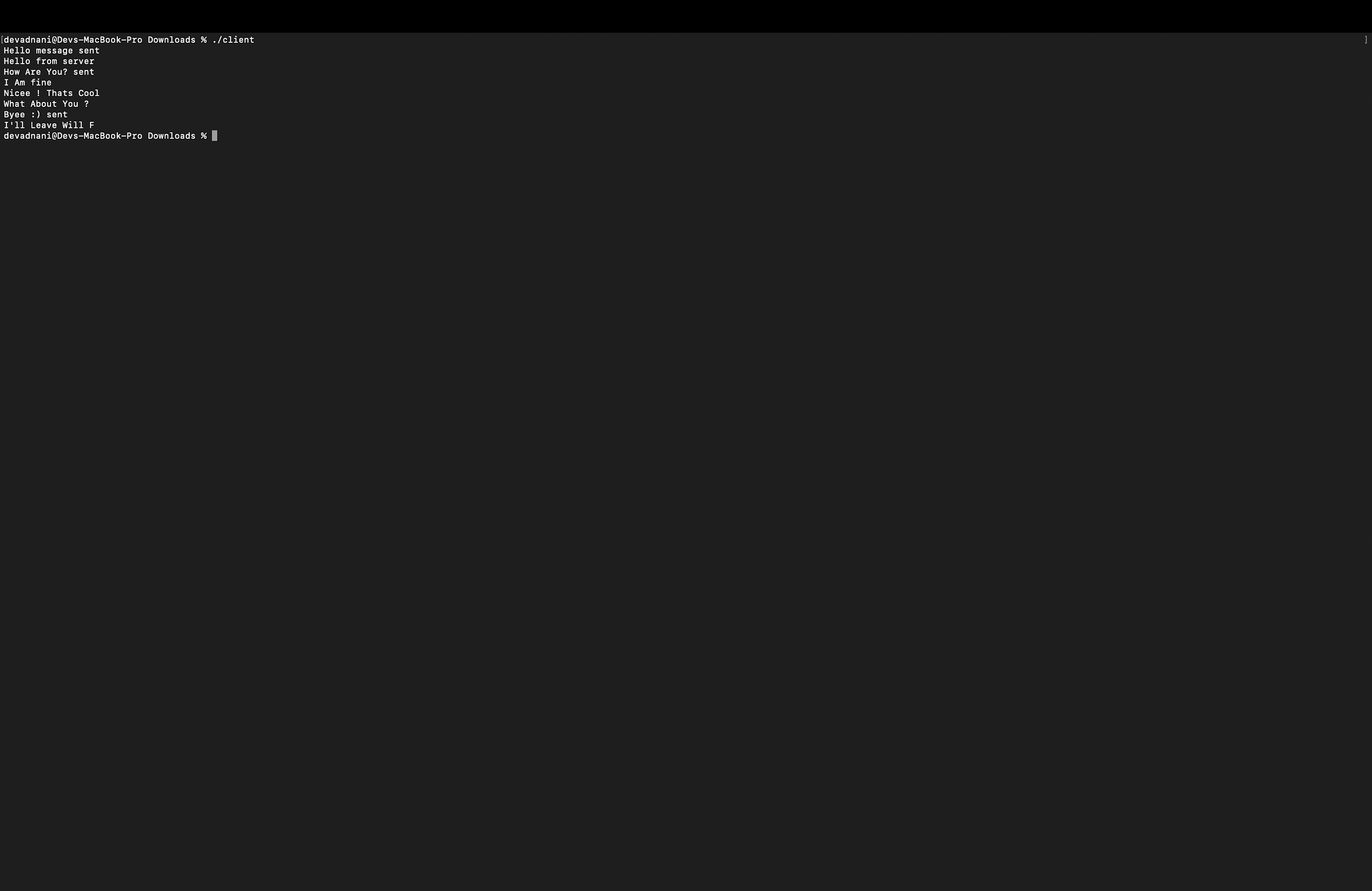
./client -> This command is used to run executable files.

After this, client send connection request to the server by using connect

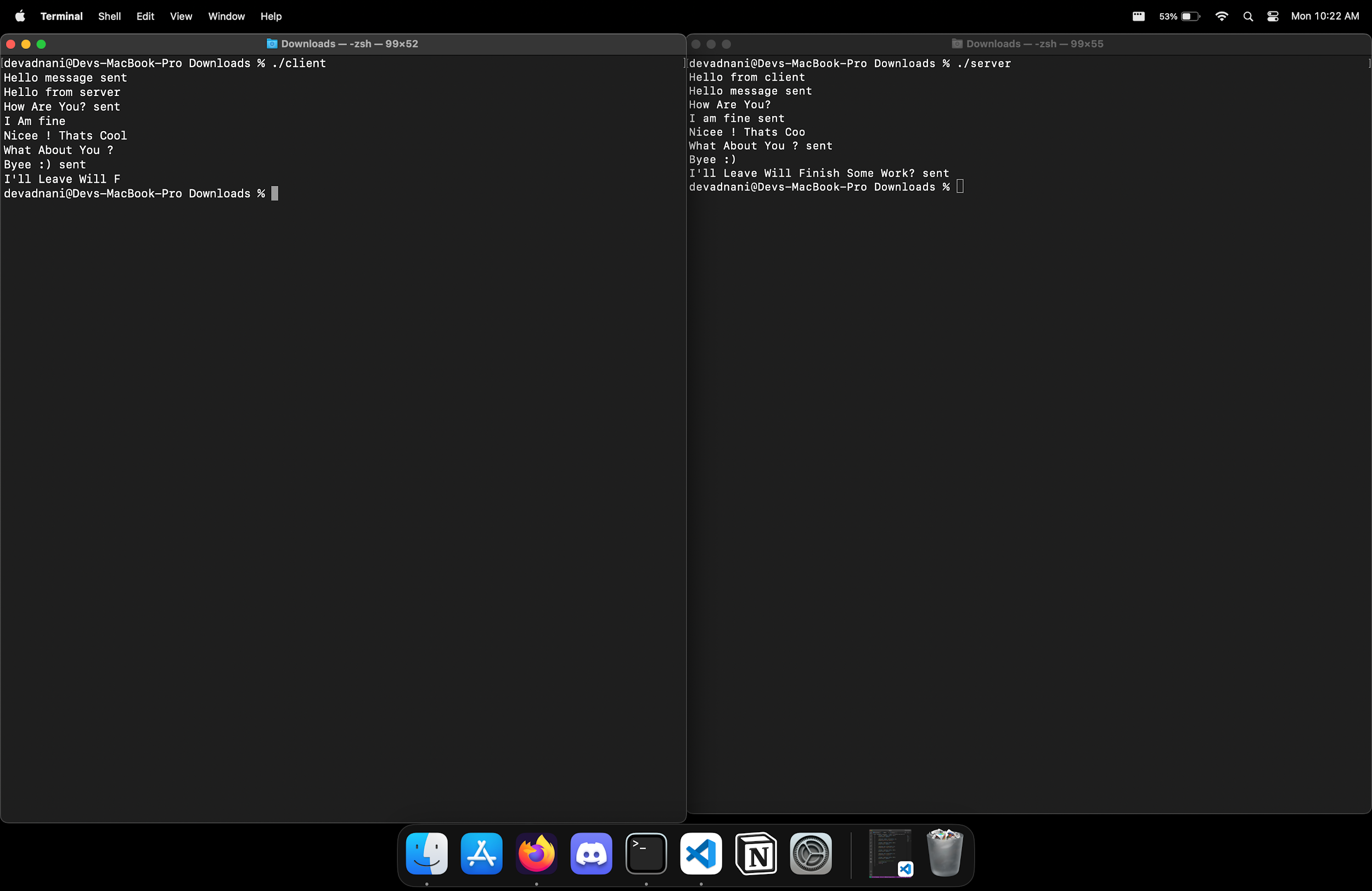
Function.

Using the accept function server accepts the connection request of the client.

After accepting the request, the message can be transferred between the client and server



Client - Server View



**202212083**

2.6 Exercise

1. Open a terminal

2. Type “gcc server.c -o server”, and press enter.

3. Type “./server”, and press enter.

4. Open 2nd terminal

5. Type “gcc client.c -o client”, and press enter.

6. Type “./client”, and press enter.

7. Both the server and client are running simultaneously.

8. Note: Always run the server first.

Server Code (server.c)

// Server side C/C++ program to demonstrate Socket programming

#include <unistd.h>

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <string.h>

#define PORT 8080

int main(int argc, char const \*argv[])

{

int server\_fd, new\_socket, valread;

struct sockaddr\_in address;

int opt = 1;

int addrlen = sizeof(address);

char buffer[1024] = {0};

char \*hello = "Hello from server";

// Creating socket file descriptor

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0)

{

perror("socket failed");

exit(EXIT\_FAILURE);

}

// Forcefully attaching socket to the port 8080

if (setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT, &opt, sizeof(opt)))

{

perror("setsockopt");

exit(EXIT\_FAILURE);

}

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons( PORT );

// Forcefully attaching socket to the port 8080

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address))<0)

{

perror("bind failed");

exit(EXIT\_FAILURE);

}

if (listen(server\_fd, 3) < 0)

{

perror("listen");

exit(EXIT\_FAILURE);

}

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen))<0)

{

perror("accept");

exit(EXIT\_FAILURE);

}

valread = read( new\_socket , buffer, 1024);

printf("%s\n",buffer );

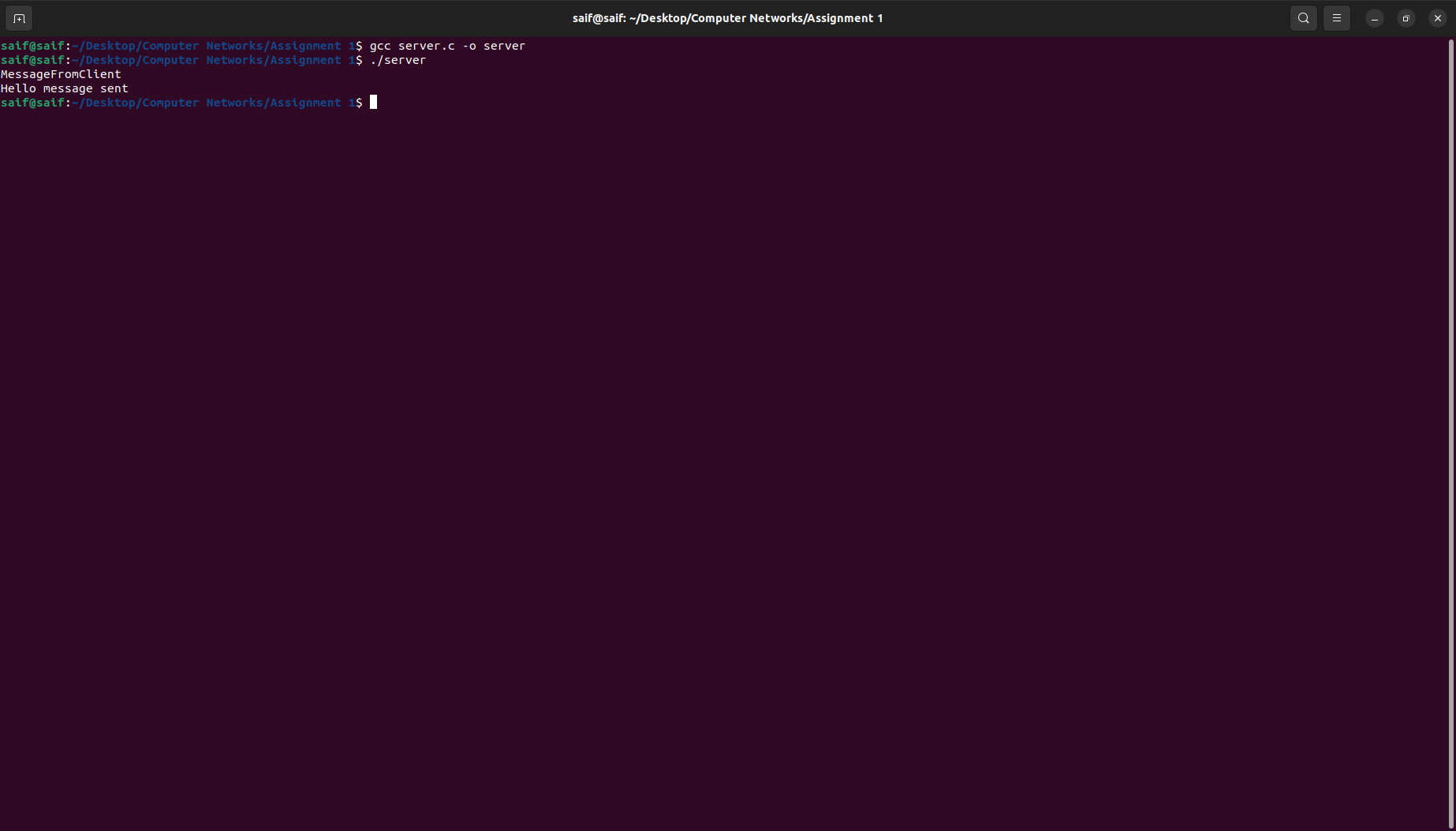
send(new\_socket , hello , strlen(hello) , 0 );

printf("Hello message sent\n");

return 0;

}

Output:



Explanation:

* This is a basic server code in C for a socket application. The server creates a socket and binds it to IP address INADDR\_ANY and port number 8080.
* Once a connection is established, it reads data sent by the client and prints it to the console. It then sends a message "Hello from server" to the client and prints "Hello message sent" to the console.
* The read() and send() functions are used to read data from and send data to the connected client, respectively.
* It uses standard C libraries such as stdio.h, unistd.h, sys/socket.h, stdlib.h, and netinet/in.h. It uses the socket() function to create a socket and bind() function to bind the socket to the IP address and port.

Client Code (client.c)

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#define PORT 8080

int main(int argc , char const \*argv[])

{

int sock = 0, valread;

struct sockaddr\_in serv\_addr;

char \*exit\_msg = "exit" , \*msg;

char buffer[1024]={0};

if((sock = socket(AF\_INET, SOCK\_STREAM , 0))<0)

{

printf("\n Socket creation error\n");

return -1;

}

serv\_addr.sin\_family =AF\_INET;

serv\_addr.sin\_port = htons(PORT);

if(inet\_pton(AF\_INET,"127.0.0.1", &serv\_addr.sin\_addr)<=0)

{

printf("\n Invalid address / address not supported \n");

return -1;

}

if(connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr))<0)

{

printf("\n connection Failed \n");

return -1;

}

while(1){

scanf("%s",msg);

if(!strcmp(msg, exit\_msg)){

close(sock);

return 0;

}

send(sock, msg , strlen(msg) , 0);

valread = read(sock, buffer, 1024);

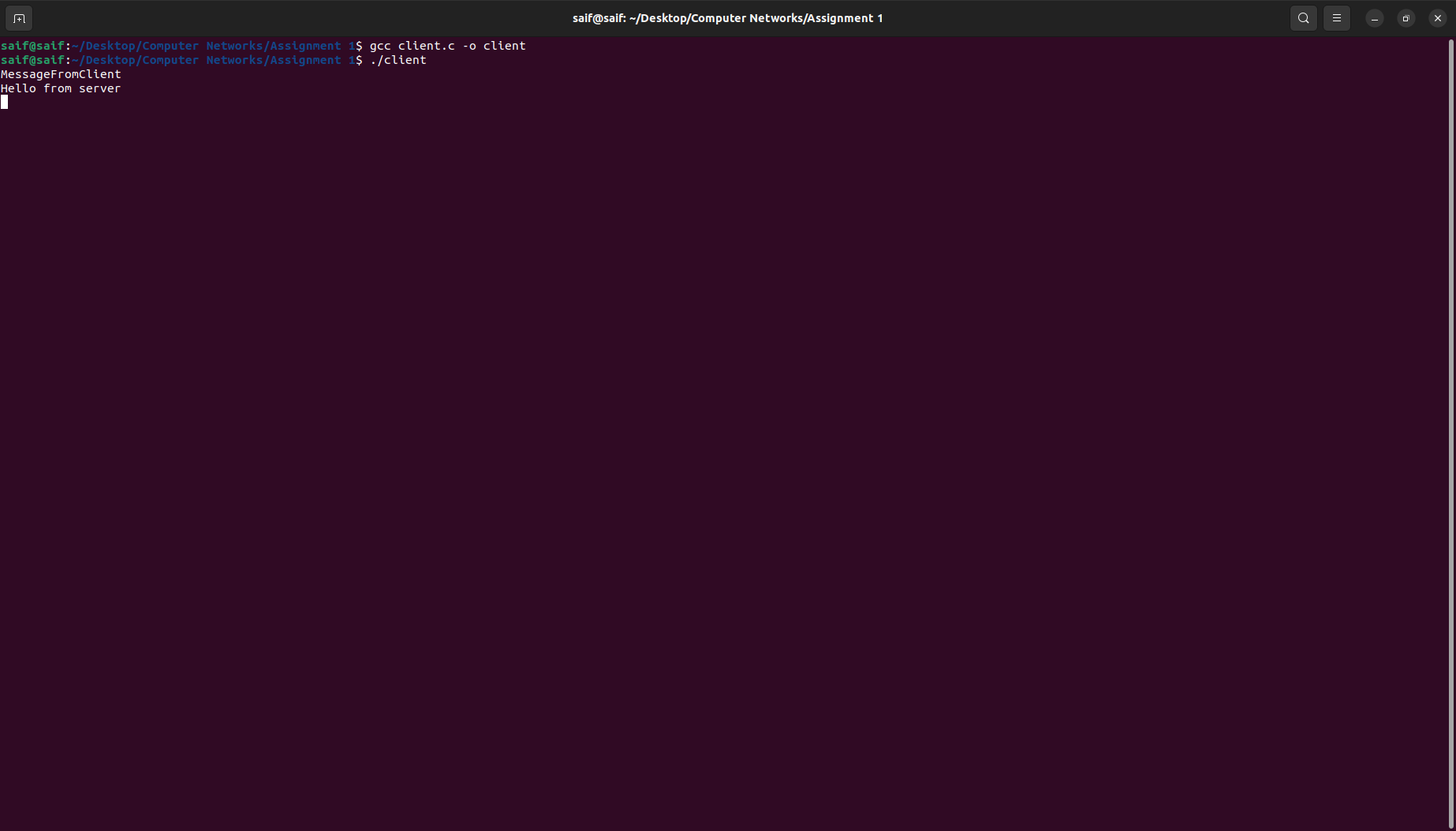
printf("%s\n",buffer);

}

return 0;

}

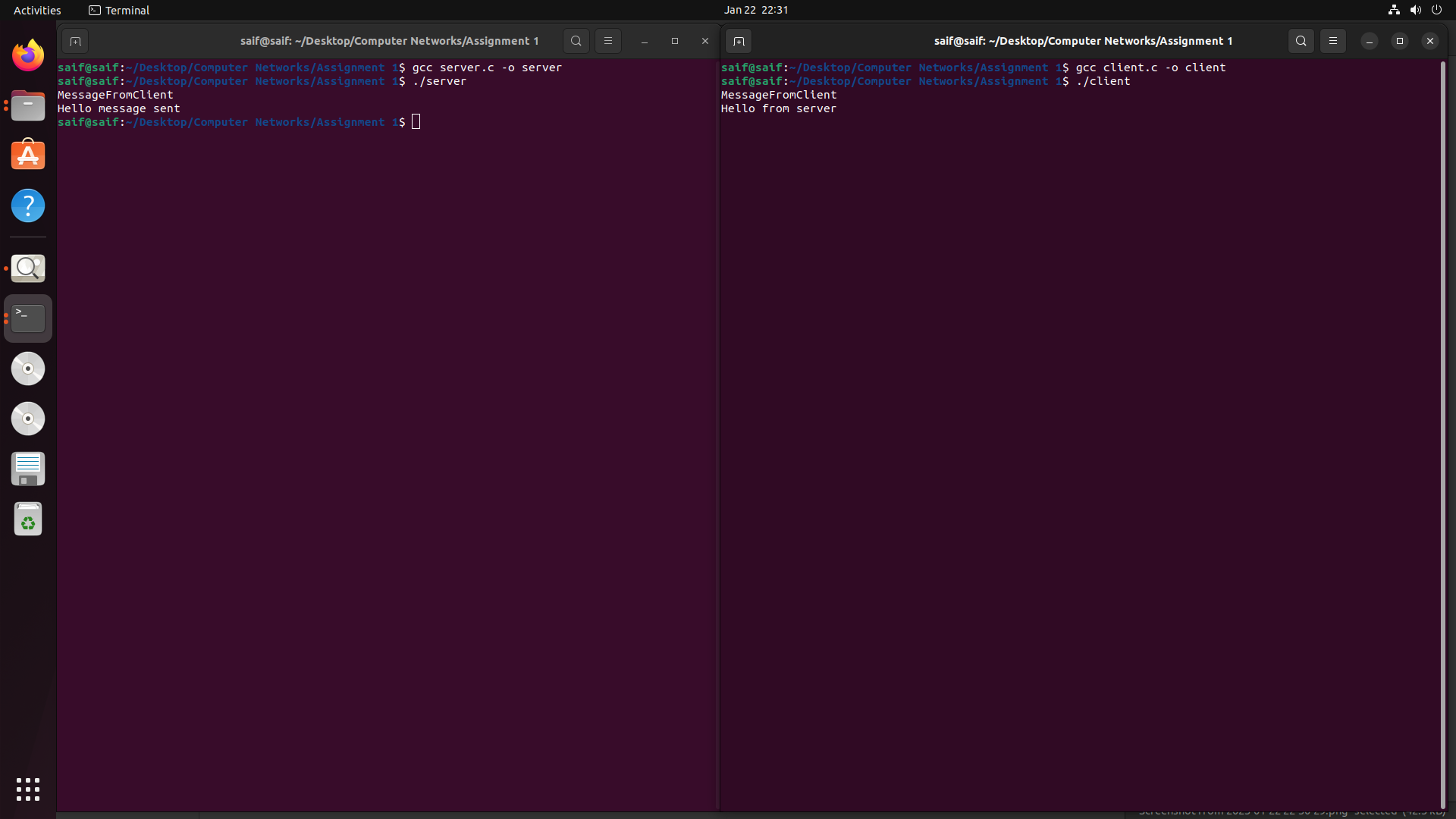
Output:



Explanation:

* This is a basic client code in C for a socket application. The client uses the socket API to create a socket and connect to a server at IP address "127.0.0.1" (localhost) and port number 8080.
* The client then enters into a loop where it prompts the user for input, sends that input to the server, and receives a response from the server. If the user enters "exit", the loop breaks and the program terminates.
* The socket is closed before the program exits.

Client & Server:



3.3 Exercise

1. Create TCP server and client using socket programming. Make them communicate with each other by making a question and answer system between them. (At Least 4 different questions should be there).

Server Code (server.c):

// Server side C/C++ program to demonstrate Socket

// programming

#include <netinet/in.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#define PORT 8080

int main(int argc, char const\* argv[])

{

int server\_fd, new\_socket, valread;

struct sockaddr\_in address;

int opt = 1;

int addrlen = sizeof(address);

char buffer[1024] = { 0 };

char\* hello = "Hello from server";

char\* middle = "Having fun";

char\* bye = "What about you?";

char\* end = "Got to go! Bye!";

// Creating socket file descriptor

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("socket failed");

exit(EXIT\_FAILURE);

}

if (setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR , &opt, sizeof(opt))) {

perror("setsockopt");

exit(EXIT\_FAILURE);

}

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

// Forcefully attaching socket to the port 8080

if (bind(server\_fd, (struct sockaddr\*)&address,

sizeof(address))

< 0) {

perror("bind failed");

exit(EXIT\_FAILURE);

}

if (listen(server\_fd, 3) < 0) {

perror("listen");

exit(EXIT\_FAILURE);

}

if ((new\_socket

= accept(server\_fd, (struct sockaddr\*)&address,

(socklen\_t\*)&addrlen))

< 0) {

perror("accept");

exit(EXIT\_FAILURE);

}

valread = read(new\_socket, buffer, 1024);

printf("%s\n", buffer);

send(new\_socket, hello, strlen(hello), 0);

printf("Hello message sent\n");

valread = read(new\_socket, buffer, 1024);

printf("%s\n", buffer);

send(new\_socket, middle, strlen(hello), 0);

printf("Having fun\n");

valread = read(new\_socket, buffer, 1024);

printf("%s\n", buffer);

send(new\_socket, bye, strlen(hello), 0);

printf("What about you? sent\n");

valread = read(new\_socket, buffer, 1024);

printf("%s\n", buffer);

send(new\_socket, end, strlen(hello), 0);

printf("Got to go! Bye! sent\n");

// closing the connected socket

close(new\_socket);

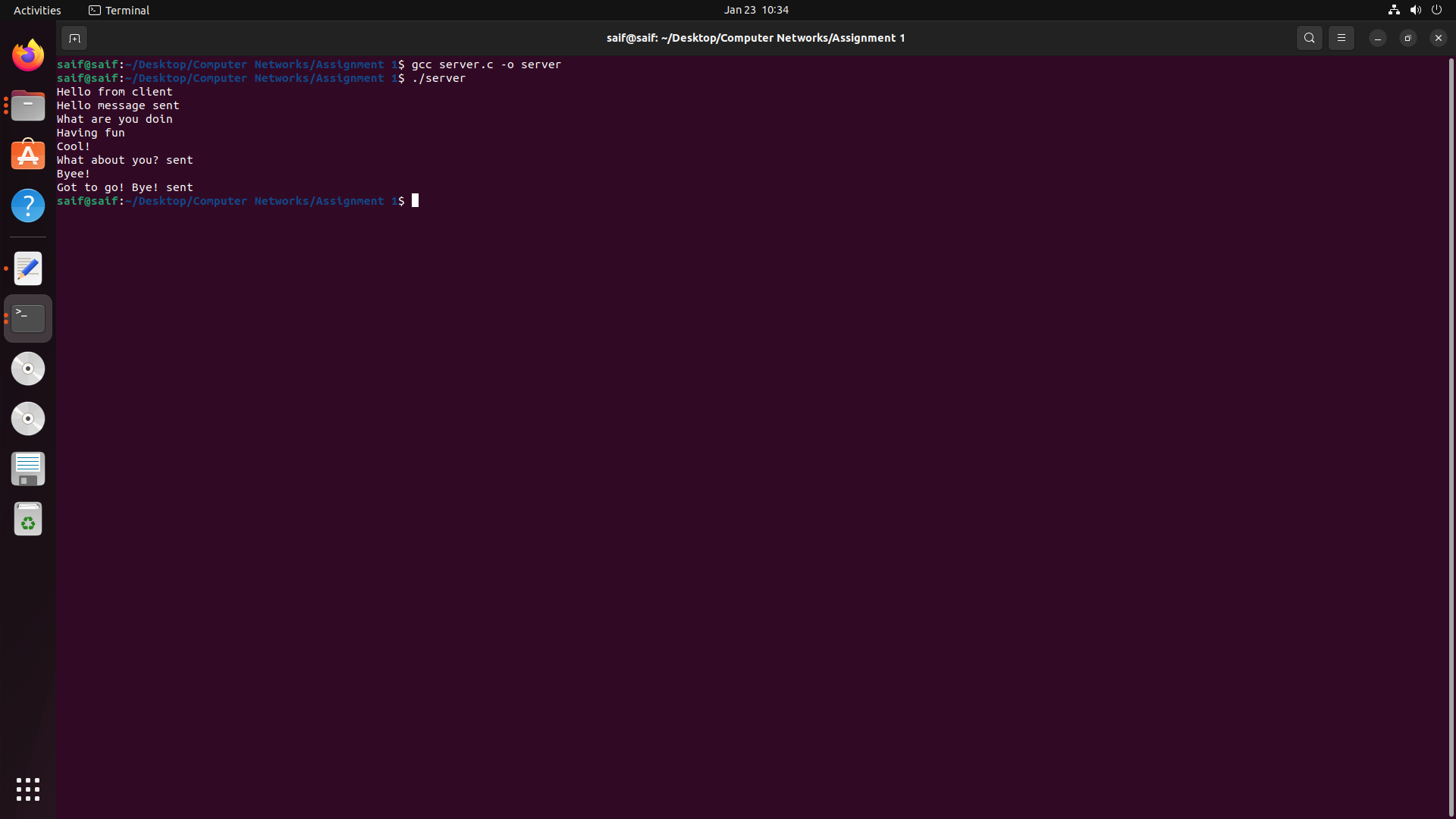
// closing the listening socket

shutdown(server\_fd, SHUT\_RDWR);

return 0;

}

Output:



Client Code (client.c):

// Client side C/C++ program to demonstrate Socket

// programming

#include <arpa/inet.h>

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#define PORT 8080

int main(int argc, char const\* argv[])

{

int sock = 0, valread, client\_fd;

struct sockaddr\_in serv\_addr;

char\* hello = "Hello from client";

char\* middle = "What are you doing?";

char\* bye = "Cool!";

char\* end = "Byee!";

char buffer[1024] = { 0 };

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

printf("\n Socket creation error \n");

return -1;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

// Convert IPv4 and IPv6 addresses from text to binary

// form

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr)

<= 0) {

printf(

"\nInvalid address/ Address not supported \n");

return -1;

}

if ((client\_fd

= connect(sock, (struct sockaddr\*)&serv\_addr,

sizeof(serv\_addr)))

< 0) {

printf("\nConnection Failed \n");

return -1;

}

send(sock, hello, strlen(hello), 0);

printf("Hello message sent\n");

valread = read(sock, buffer, 1024);

printf("%s\n", buffer);

send(sock, middle, strlen(hello), 0);

printf("What are you doing? sent\n");

valread = read(sock, buffer, 1024);

printf("%s\n", buffer);

send(sock, bye, strlen(hello), 0);

printf("Cool! \n");

valread = read(sock, buffer, 1024);

printf("%s\n", buffer);

send(sock, end, strlen(hello), 0);

printf("Byee! sent\n");

valread = read(sock, buffer, 1024);

printf("%s\n", buffer);

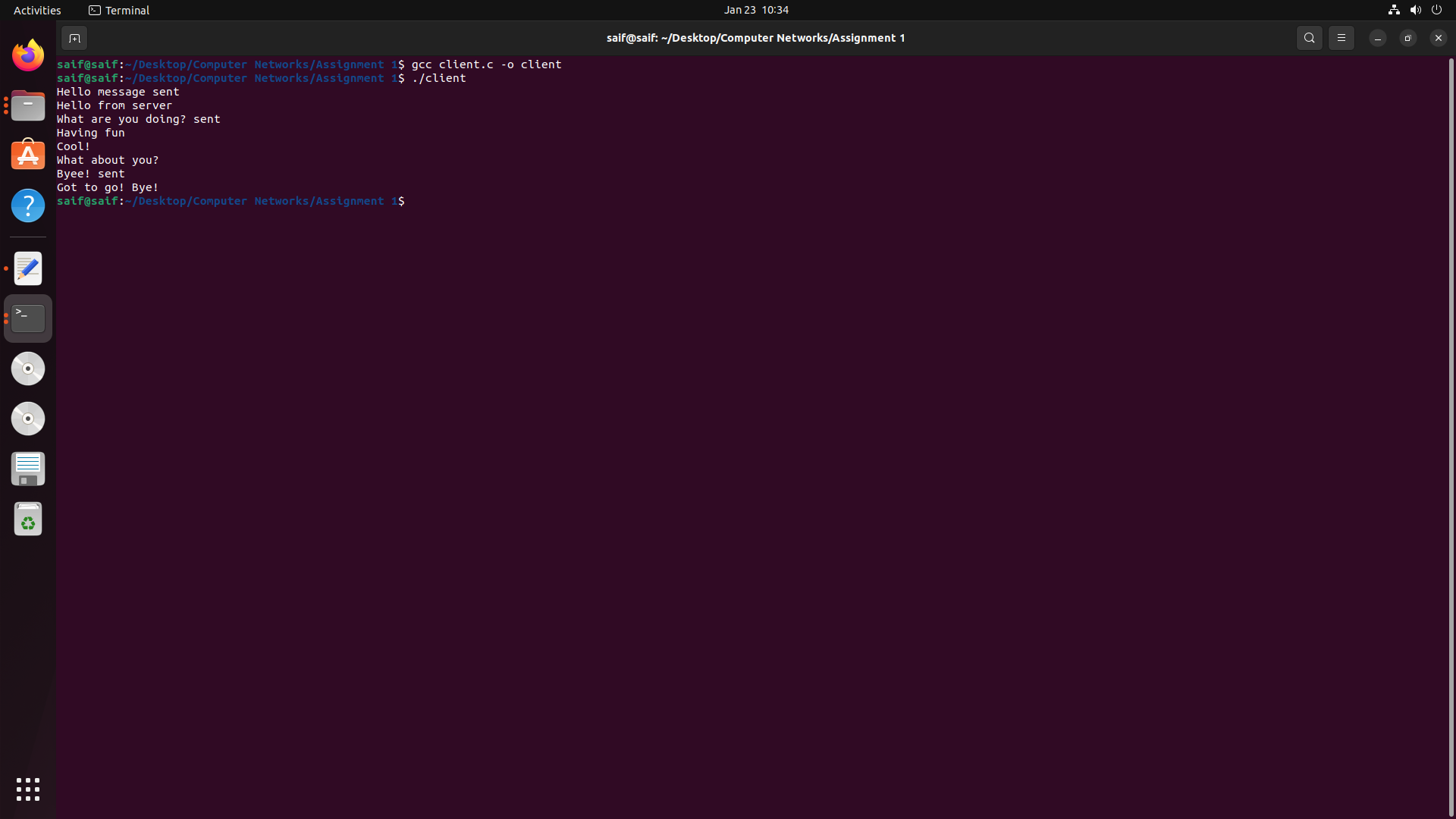
// closing the connected socket

close(client\_fd);

return 0;

}

Output:



Client - Server Sending & Receiving Messages

