Assignment 1
Computer Network

Program: MScIT Sem-2

Group ID : 28

Student Name	Student ID
Dev Adnani	202212012
Saif Saiyed	202212083

202212012

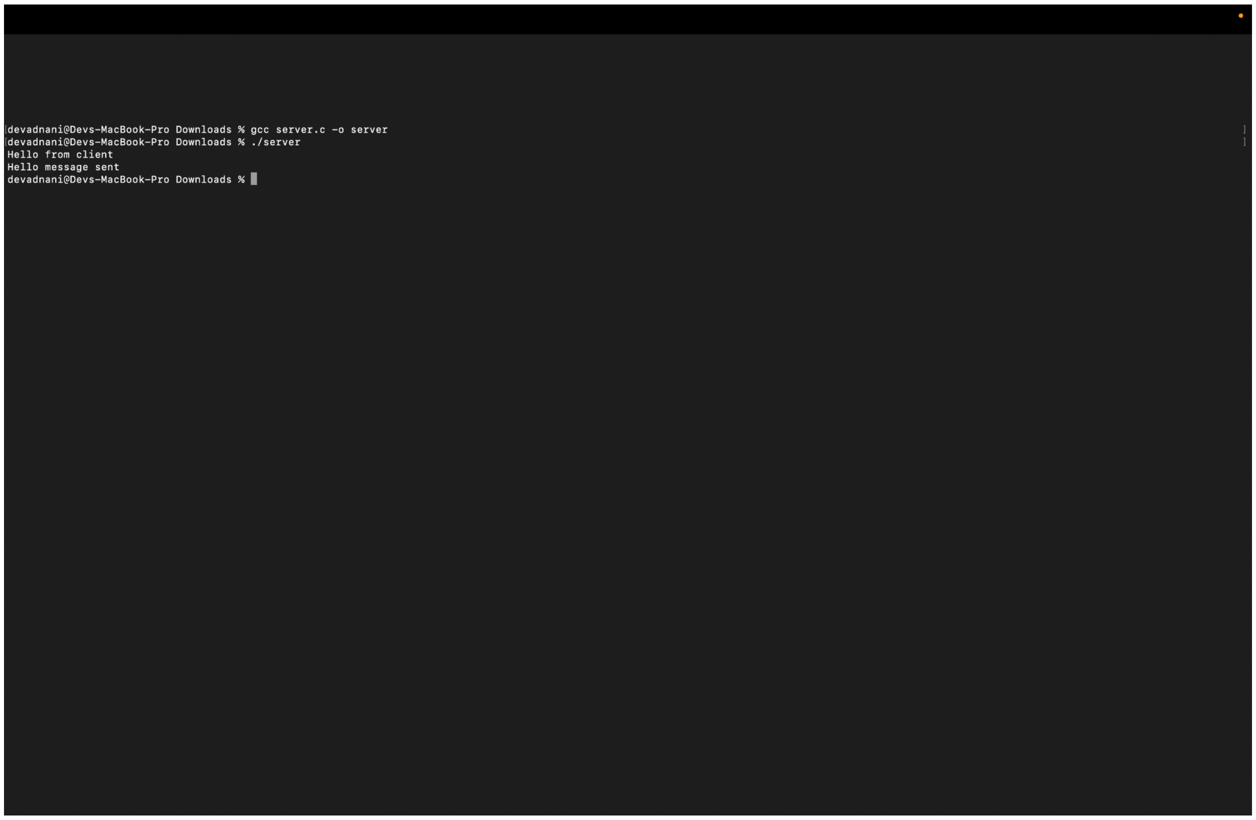
2.6 Exercise

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:



```
[devadnani@Devs-MacBook-Pro Downloads % gcc server.c -o server
[devadnani@Devs-MacBook-Pro Downloads % ./server
Hello from client
Hello message sent
devadnani@Devs-MacBook-Pro Downloads % 
```

Explanation:

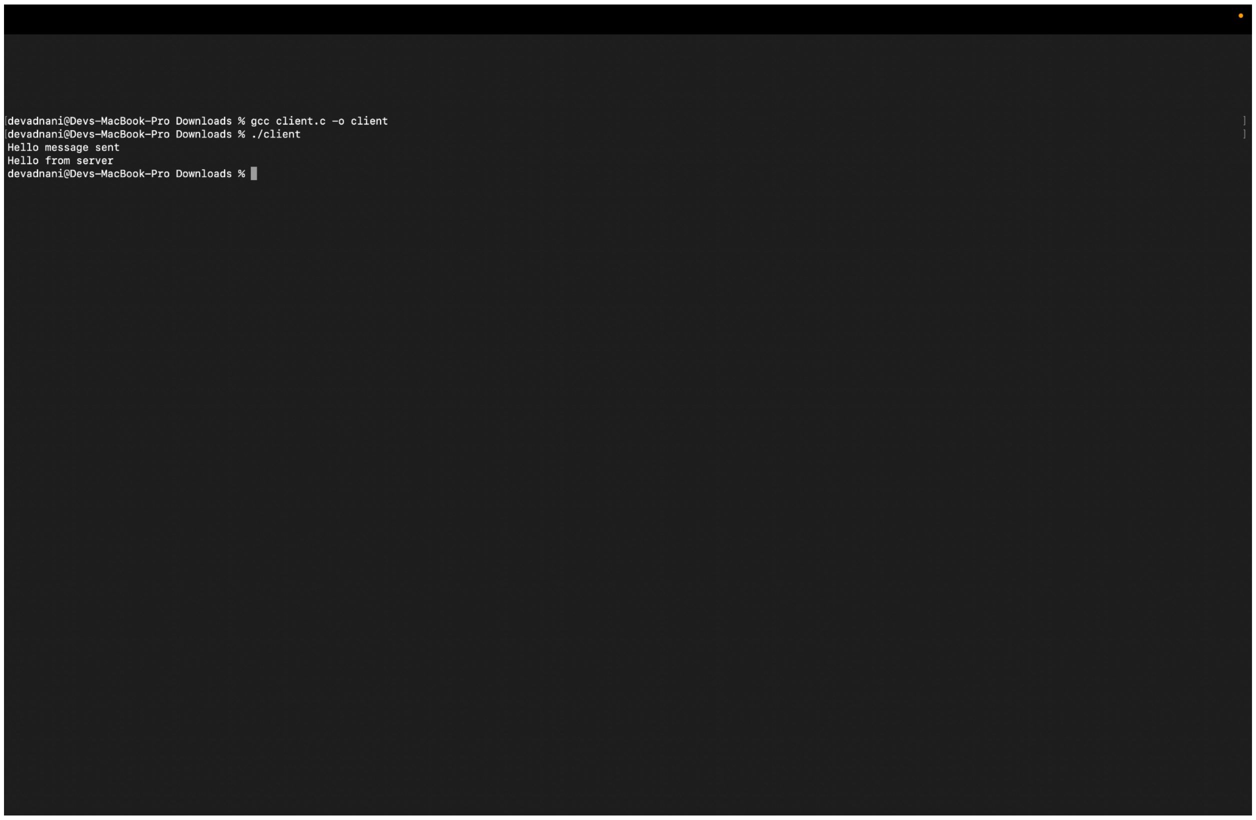
- This is a basic server code in C for a socket application. The server creates a socket and binds it to IP address INADDR_ANY and port number 8080.
- Once a connection is established, it reads data sent by the client and prints it to the console. It then sends a message "Hello from server" to the client 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;
}
```

<u>Output:</u>

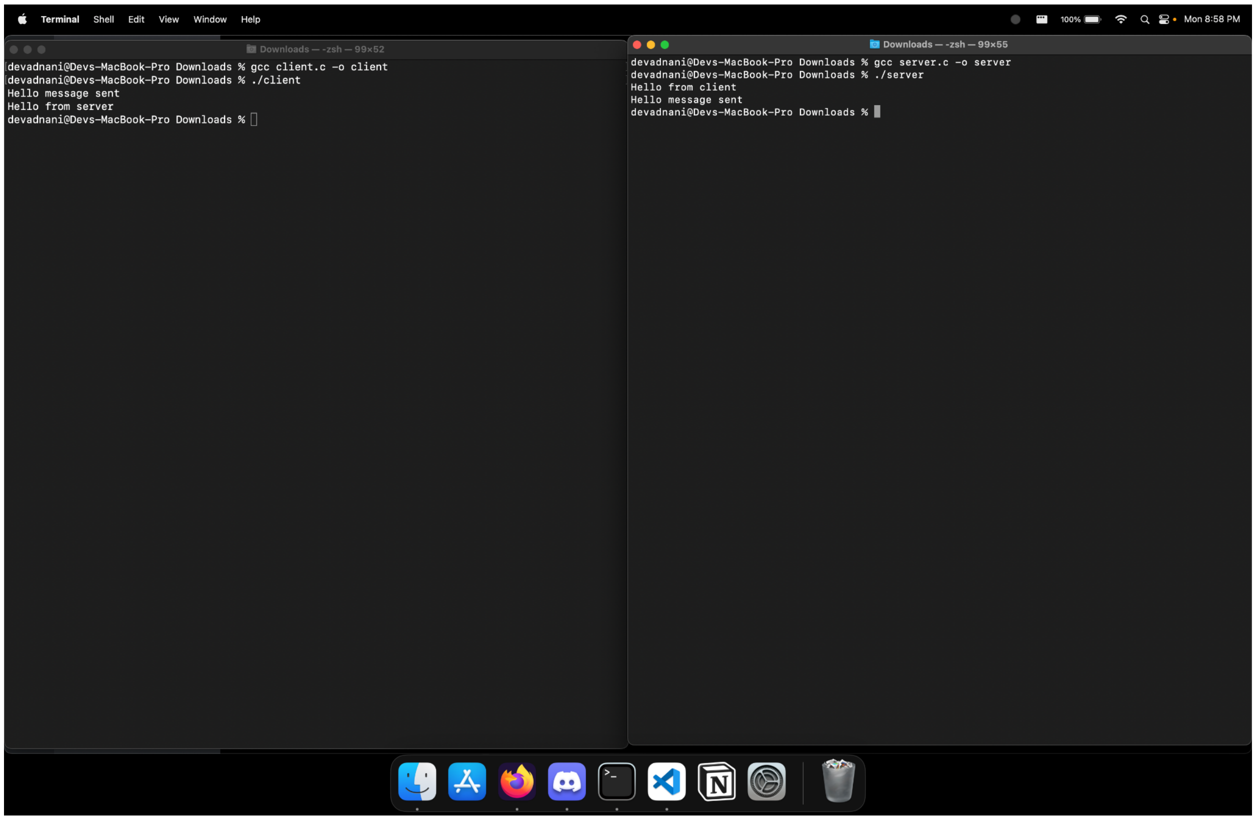


```
devadnani@Devs-MacBook-Pro Downloads % gcc client.c -o client
devadnani@Devs-MacBook-Pro Downloads % ./client
Hello message sent
Hello from server
devadnani@Devs-MacBook-Pro Downloads % 
```

<u>Explanation:</u>

- 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:



```
devadnani@Devs-MacBook-Pro Downloads % gcc client.c -o client
devadnani@Devs-MacBook-Pro Downloads % ./client
Hello message sent
Hello from server
devadnani@Devs-MacBook-Pro Downloads % 
```

```
devadnani@Devs-MacBook-Pro Downloads % gcc server.c -o server
devadnani@Devs-MacBook-Pro Downloads % ./server
Hello from client
Hello message sent
devadnani@Devs-MacBook-Pro Downloads % 
```

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.

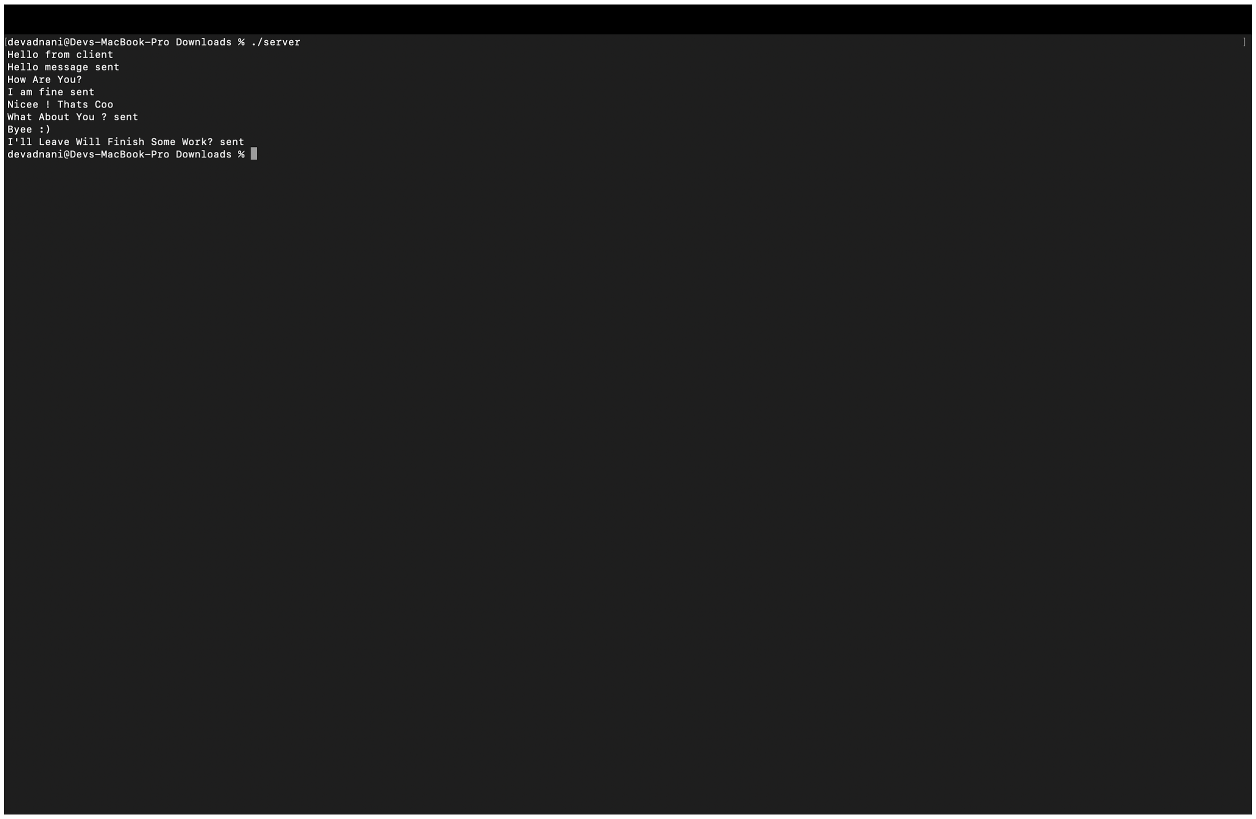
gcc server.c -o server -> This command is used to compile the server file. After running this command a new executable file is created.

./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.



```
devadnani@Devs-MacBook-Pro Downloads % ./server
Hello from client
Hello message sent
How Are You?
I am fine sent
Nicee ! Thats Coo
What About You ? sent
Byee :)
I'll Leave Will Finish Some Work? sent
devadnani@Devs-MacBook-Pro Downloads %
```

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;
}
```

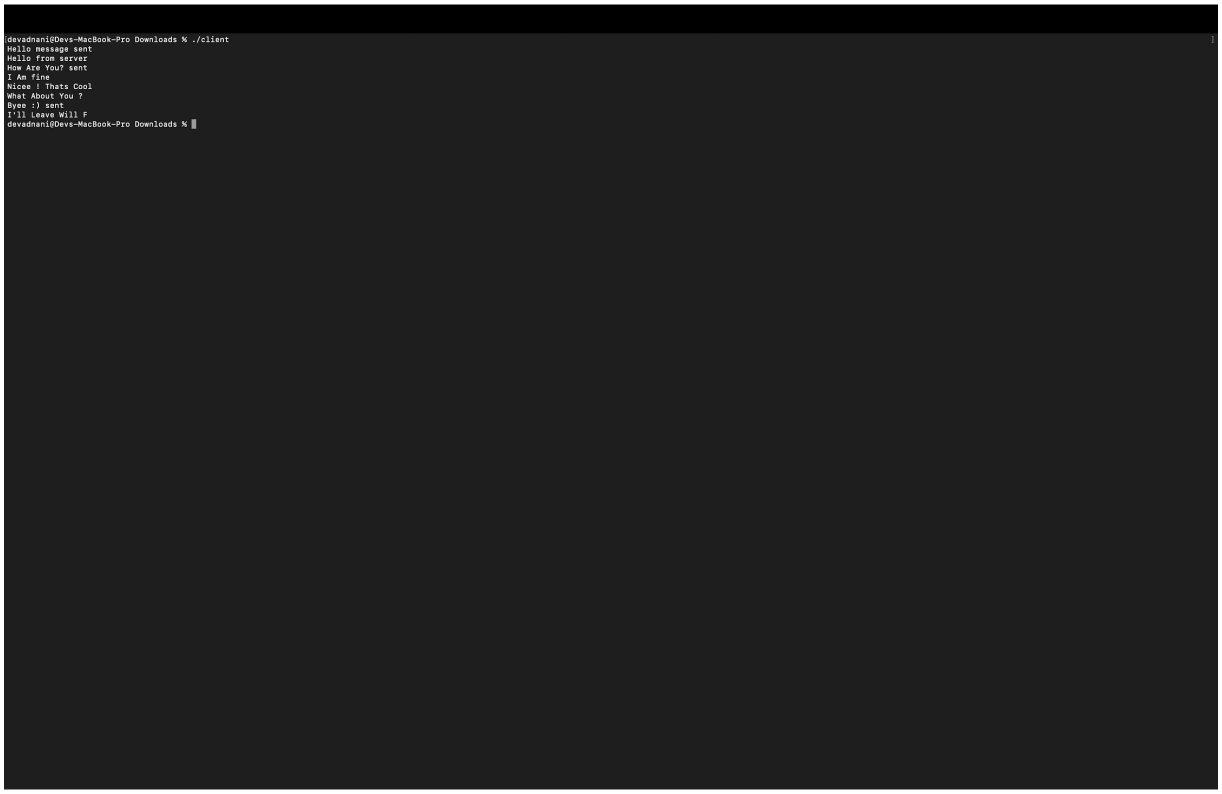
Output :
Open another terminal and execute the client's command.

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.

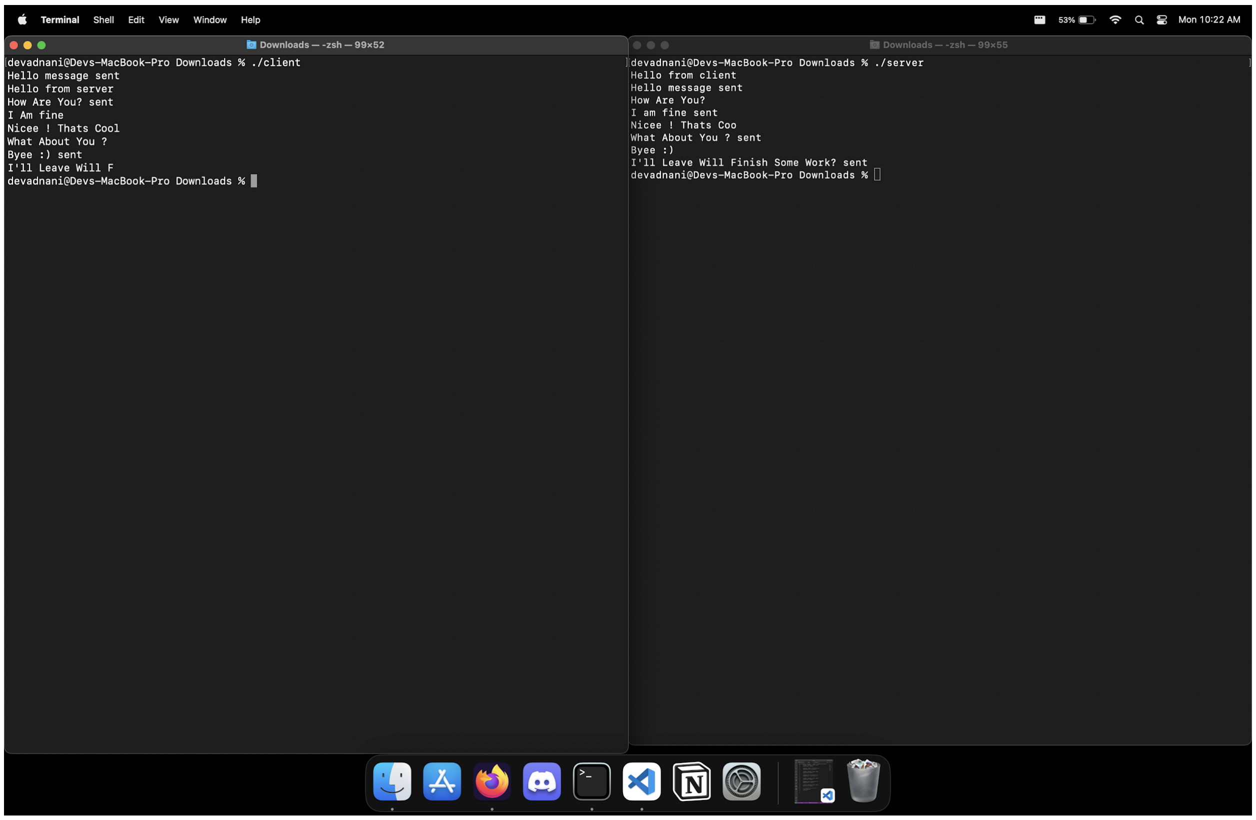
After this, client send connection request to the server by using connect
Function.

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



```
[devadnani@Devs-MacBook-Pro Downloads % ./client
Hello message sent
Hello from server
How Are You? sent
I Am fine
Nicee ! Thats Cool
What About You ?
Byee :) sent
I'll Leave Will F
devadnani@Devs-MacBook-Pro Downloads %
```

Client - Server View



```
devadnani@Devs-MacBook-Pro Downloads % ./client
Hello message sent
Hello from server
How Are You? sent
I Am fine
Nicee ! Thats Cool
What About You ?
Byee :) sent
I'll Leave Will F
devadnani@Devs-MacBook-Pro Downloads %
```

```
devadnani@Devs-MacBook-Pro Downloads % ./server
Hello from client
Hello message sent
How Are You?
I am fine sent
Nicee ! Thats Coo
What About You ? sent
Byee :)
I'll Leave Will Finish Some Work? sent
devadnani@Devs-MacBook-Pro Downloads %
```

202212083

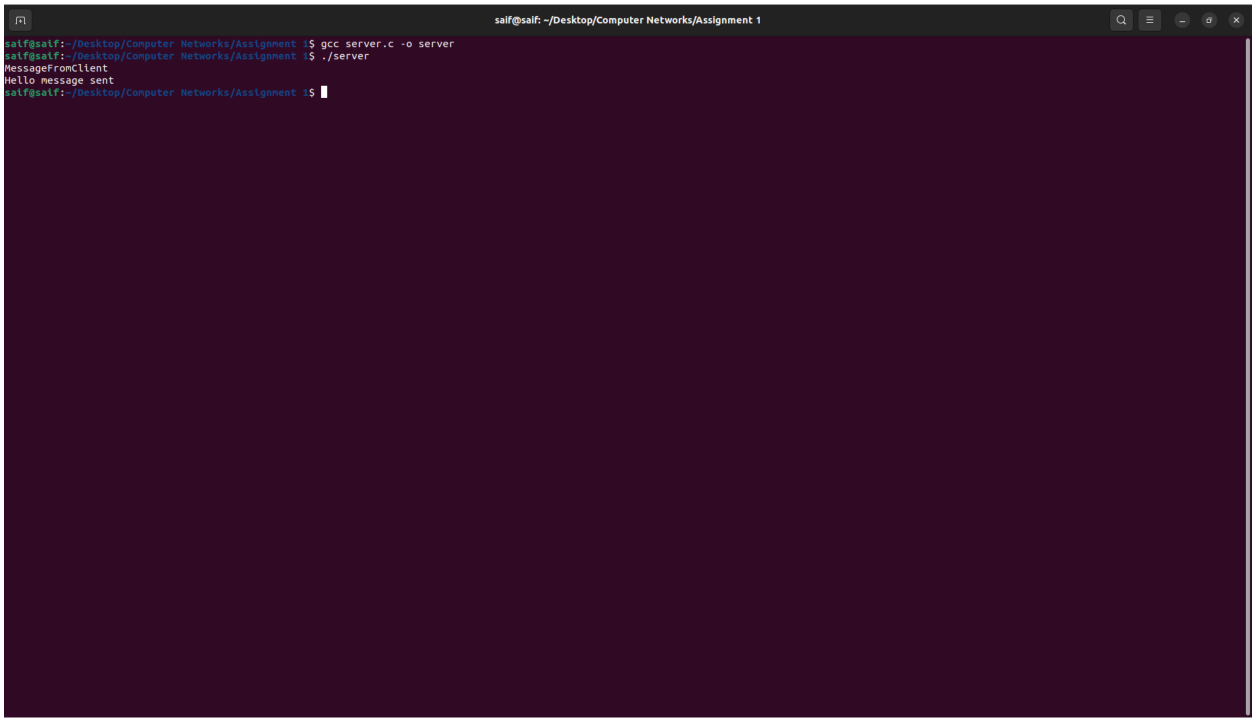
2.6 Exercise

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:



```
saif@saif: ~/Desktop/Computer Networks/Assignment 1
saif@saif:~/Desktop/Computer Networks/Assignment 1$ gcc server.c -o server
saif@saif:~/Desktop/Computer Networks/Assignment 1$ ./server
MessageFromClient
Hello message sent
saif@saif:~/Desktop/Computer Networks/Assignment 1$ 
```

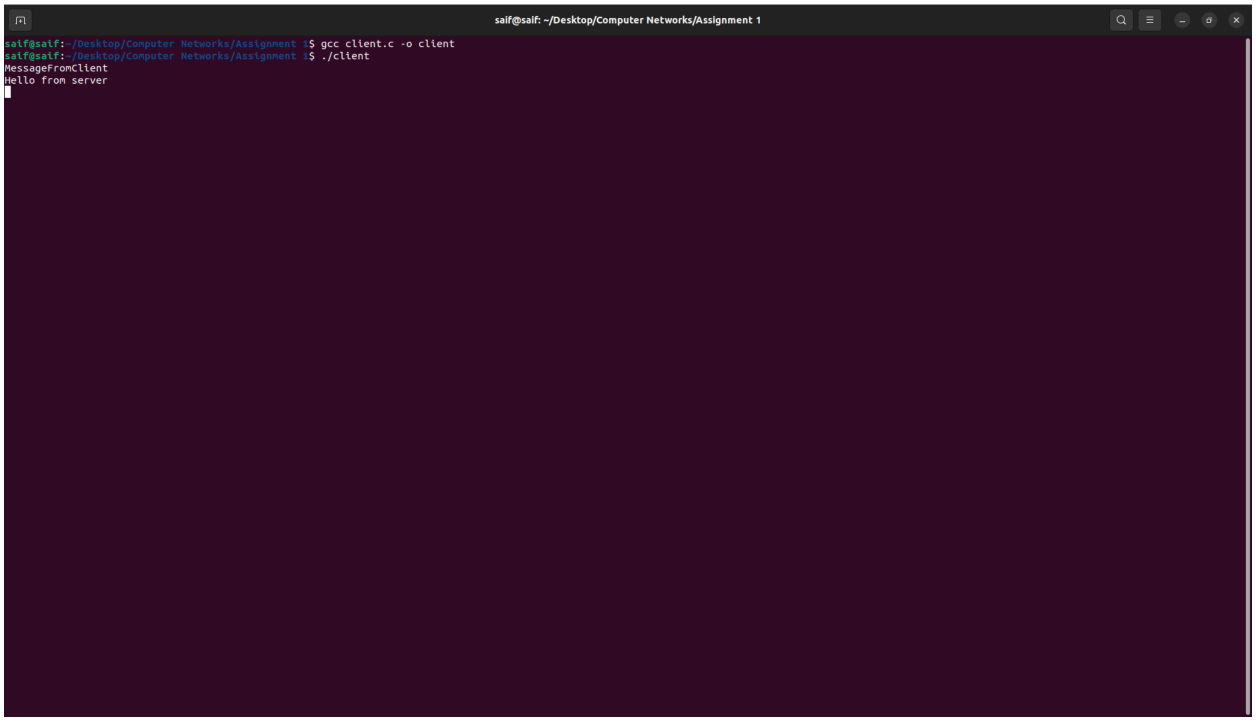
Explanation:

- 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:

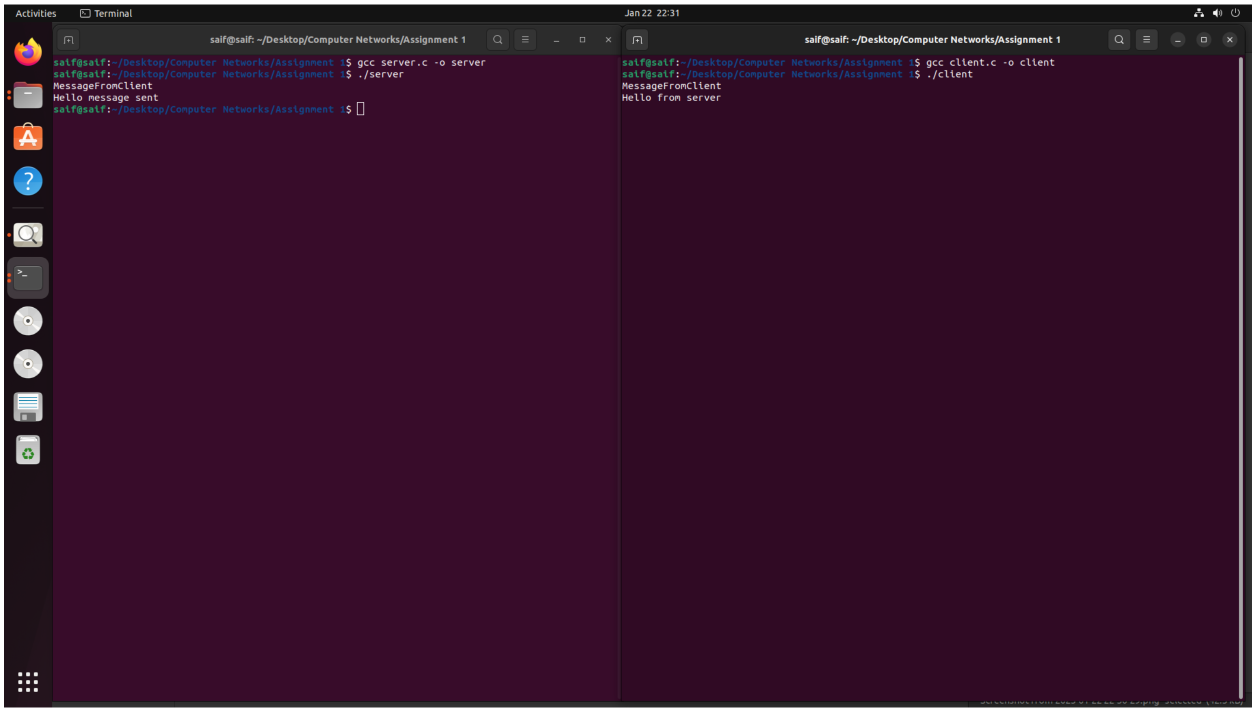


```
saif@saif:~/Desktop/Computer Networks/Assignment 1$ gcc client.c -o client
saif@saif:~/Desktop/Computer Networks/Assignment 1$ ./client
MessageFromClient
Hello from server
```

<u>Explanation:</u>

- 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:


Activities
Terminal
Jan 22 22:31
saif@saif: ~/Desktop/Computer Networks/Assignment 1
saif@saif: ~/Desktop/Computer Networks/Assignment 1

```
saif@saif:~/Desktop/Computer Networks/Assignment 1$ gcc server.c -o server
saif@saif:~/Desktop/Computer Networks/Assignment 1$ ./server
MessageFromClient
Hello message sent
saif@saif:~/Desktop/Computer Networks/Assignment 1$
```

```
saif@saif:~/Desktop/Computer Networks/Assignment 1$ gcc client.c -o client
saif@saif:~/Desktop/Computer Networks/Assignment 1$ ./client
MessageFromClient
Hello from server
```

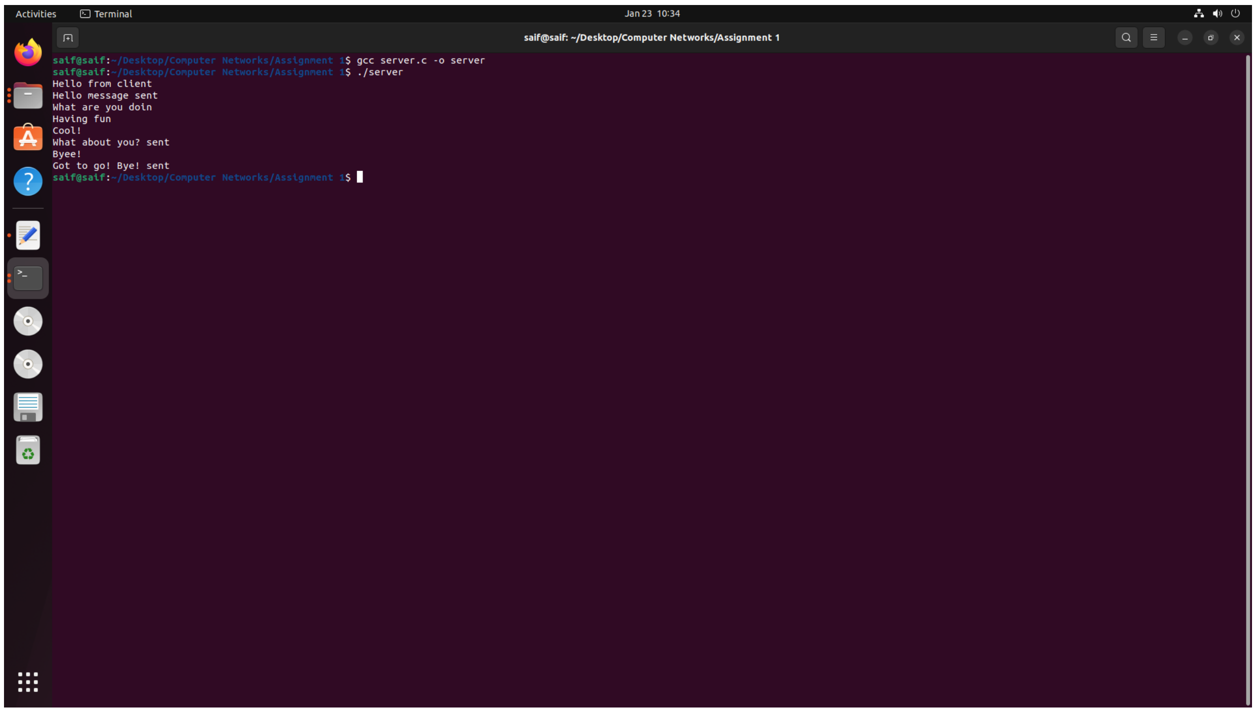
3.3 Exercise

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:



```
saif@saif:~/Desktop/Computer Networks/Assignment 1$ gcc server.c -o server
saif@saif:~/Desktop/Computer Networks/Assignment 1$ ./server
Hello from client
Hello message sent
What are you doin
Having fun
Cool!
What about you? sent
Byee!
Got to go! Bye! sent
saif@saif:~/Desktop/Computer Networks/Assignment 1$
```

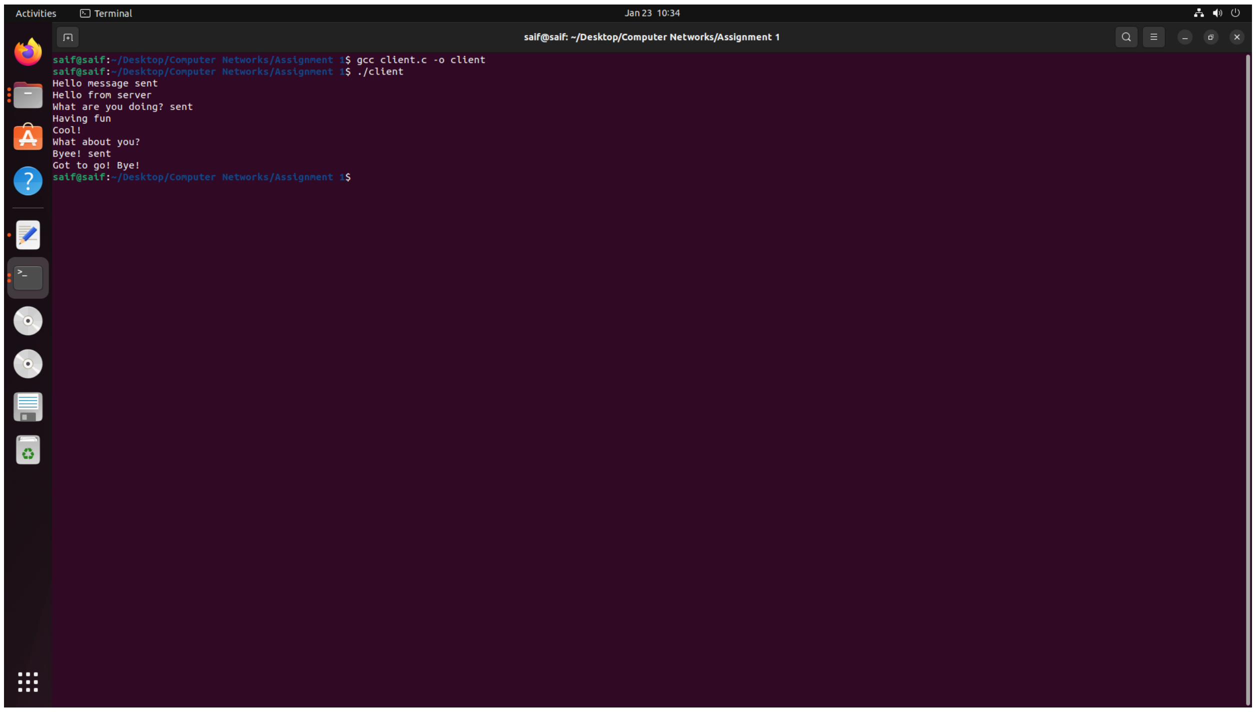
Client Code (client.c):

```
// 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);
```

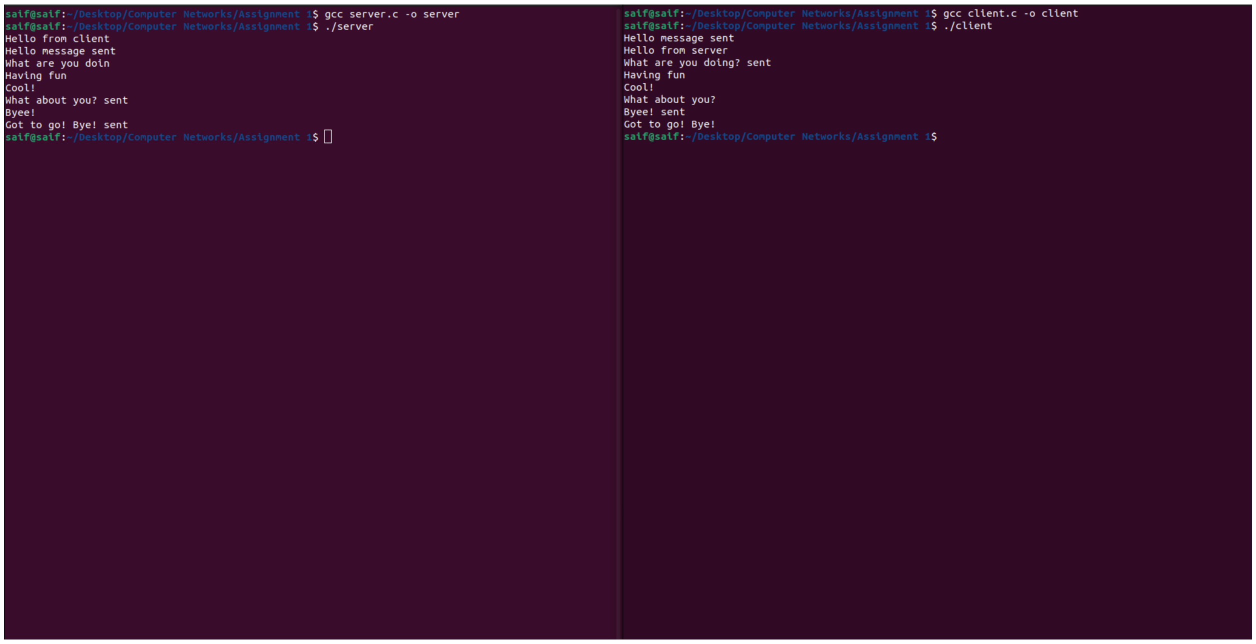
```
        printf("%s\n", buffer);

        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;
}
```

<u>Output:</u>



```
saif@saif:~/Desktop/Computer Networks/Assignment 1$ gcc client.c -o client
saif@saif:~/Desktop/Computer Networks/Assignment 1$ ./client
Hello message sent
Hello from server
What are you doing? sent
Having fun
Cool!
What about you?
Byee! sent
Got to go! Bye!
saif@saif:~/Desktop/Computer Networks/Assignment 1$
```

Client - Server Sending & Receiving Messages



```
saif@saif:~/Desktop/Computer Networks/Assignment 1$ gcc server.c -o server
saif@saif:~/Desktop/Computer Networks/Assignment 1$ ./server
Hello from client
Hello message sent
What are you doin
Having fun
Cool!
What about you? sent
Byee!
Got to go! Bye! sent
saif@saif:~/Desktop/Computer Networks/Assignment 1$
```

```
saif@saif:~/Desktop/Computer Networks/Assignment 1$ gcc client.c -o client
saif@saif:~/Desktop/Computer Networks/Assignment 1$ ./client
Hello message sent
Hello from server
What are you doing? sent
Having fun
Cool!
What about you?
Byee! sent
Got to go! Bye!
saif@saif:~/Desktop/Computer Networks/Assignment 1$
```