Name: Adwait Purao

UID: 2021300101

Batch: B2

Experiment no.: 5

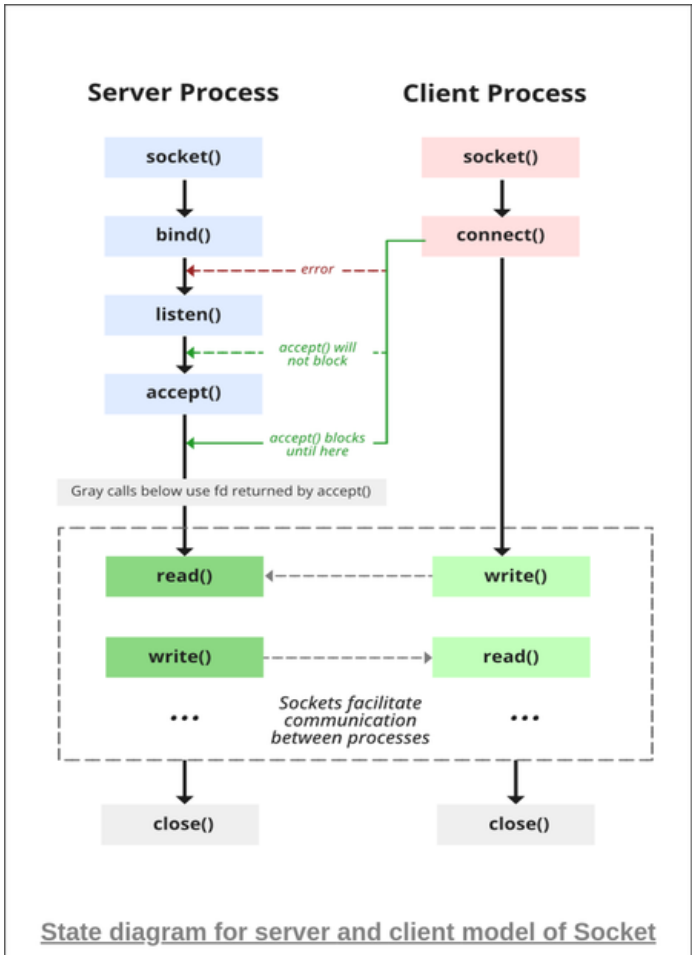
Aim: To learn how to create sockets for communication between processes, computers, or devices.

Theory:

What is socket programming?

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.

State diagram for server and client model



State diagram for server and client model of Socket

Stages for server

1. Socket creation:

int sockfd = socket(domain, type, protocol)

- **sockfd:** socket descriptor, an integer (like a file-handle)
- **domain:** integer, specifies communication domain. We use AF_ LOCAL as defined in the POSIX standard for communication between processes on the same host. For communicating between processes on different hosts connected by IPV4, we use AF_INET and AF_I NET 6 for processes connected by IPV6.
- **type:** communication type
  SOCK_STREAM: TCP(reliable, connection oriented)
  SOCK_DGRAM: UDP(unreliable, connectionless)
- **protocol:** Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)

2. Setsockopt:

This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: “address already in use”.

*int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen);*

3. Bind:

*int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);*

After the creation of the socket, the bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.

4. Listen:

int listen(int sockfd, int backlog);

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

5. Accept:

*int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);*

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, the connection is established between client and server, and they are ready to transfer data.

Stages for Client

- **Socket connection:** Exactly same as that of server’s socket creation
- **Connect:** The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server’s address and port is specified in addr.

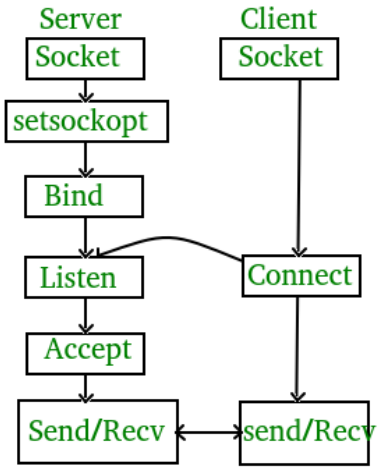
*int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);*

TCP Server Client

If we are creating a connection between client and server using TCP then it has a few functionalities like, TCP is suited for applications that require high reliability, and transmission time is relatively less critical. It is used by other protocols like HTTP, HTTPs, FTP, SMTP, Telnet. TCP rearranges data packets in the order specified. There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. TCP does Flow Control and requires three packets to set up a socket connection before any user data can be sent. TCP handles reliability and

congestion control. It also does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination.

The entire process can be broken down into the following steps:



The entire process can be broken down into following steps:

TCP Server –

1. using create(), Create TCP socket.
2. using bind(), Bind the socket to server address.
3. using listen(), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection
4. using accept(), At this point, connection is established between client and server, and they are ready to transfer data.
5. Go back to Step 3.

TCP Client –

1. Create TCP socket.
2. connect newly created client socket to server.

SocketServer.c

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
```

```c
#include <sys/types.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

// Function designed for chat between client and server.
void func(int connfd)
{
    char buff[MAX];
    int n;
    // infinite loop for chat
    for (;;) {
        bzero(buff, MAX);

        // read the message from client and copy it in buffer
        read(connfd, buff, sizeof(buff));
        // print buffer which contains the client contents
        printf("From client: %s\t To client : ", buff);
        bzero(buff, MAX);
        n = 0;
        // copy server message in the buffer
        while ((buff[n++] = getchar()) != '\n')
            ;

        // and send that buffer to client
        write(connfd, buff, sizeof(buff));

        // if msg contains "Exit" then server exit and chat ended.
        if (strncmp("exit", buff, 4) == 0) {
            printf("Server Exit...\n");
            break;
        }
    }
}

// Driver function
int main()
{
    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
```

```
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // Binding newly created socket to given IP and verification
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("socket bind failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully binded..\n");

    // Now server is ready to listen and verification
    if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
    }
    else
        printf("Server listening..\n");
    len = sizeof(cli);

    // Accept the data packet from client and verification
    connfd = accept(sockfd, (SA*)&cli, &len);
    if (connfd < 0) {
        printf("server accept failed...\n");
        exit(0);
    }
    else
        printf("server accept the client...\n");

    // Function for chatting between client and server
    func(connfd);

    // After chatting close the socket
    close(sockfd);
}
```

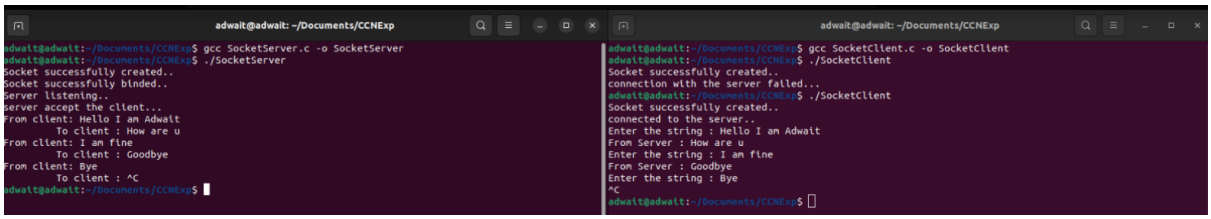
SocketClient.c

```
#include <arpa/inet.h> // inet_addr()
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <string.h>
#include <strings.h> // bzero()
#include <sys/socket.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
{
    char buff[MAX];
    int n;
    for (;;) {
        bzero(buff, sizeof(buff));
        printf("Enter the string : ");
        n = 0;
        while ((buff[n++] = getchar()) != '\n')
            ;
        write(sockfd, buff, sizeof(buff));
        bzero(buff, sizeof(buff));
        read(sockfd, buff, sizeof(buff));
        printf("From Server : %s", buff);
        if ((strncmp(buff, "exit", 4)) == 0) {
            printf("Client Exit...\n");
            break;
        }
    }
}

int main()
{
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("10.0.2.15");
    servaddr.sin_port = htons(PORT);
```

```
    // connect the client socket to server socket
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr))
        != 0) {
        printf("connection with the server failed...\n");
        exit(0);
    }
    else
        printf("connected to the server..\n");

    // function for chat
    func(sockfd);

    // close the socket
    close(sockfd);
}
```

Output:



Server.c

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

// Function designed for chat between client and server.
void func(int connfd)
{
        char buff[MAX];
        int n;
        // infinite loop for chat
        for (;;) {
                bzero(buff, MAX);

                // read the message from client and copy it in buffer
                read(connfd, buff, sizeof(buff));
                // print buffer which contains the client contents
                printf("From client: %s\t To client : ", buff);
                bzero(buff, MAX);
```

```
                n = 0;
                // copy server message in the buffer
                while ((buff[n++] = getchar()) != '\n')
                        ;

                // and send that buffer to client
                write(connfd, buff, sizeof(buff));

                // if msg contains "Exit" then server exit and chat
ended.
                if (strncmp("exit", buff, 4) == 0) {
                        printf("Server Exit...\n");
                        break;
                }
        }
}

// Driver function
int main()
{
        int sockfd, connfd, len;
        struct sockaddr_in servaddr, cli;

        // socket create and verification
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd == -1) {
                printf("socket creation failed...\n");
                exit(0);
        }
        else
                printf("Socket successfully created..\n");
        bzero(&servaddr, sizeof(servaddr));

        // assign IP, PORT
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
        servaddr.sin_port = htons(PORT);

        // Binding newly created socket to given IP and verification
        if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
                printf("socket bind failed...\n");
                exit(0);
        }
        else
                printf("Socket successfully binded..\n");

        // Now server is ready to listen and verification
        if ((listen(sockfd, 5)) != 0) {
                printf("Listen failed...\n");
                exit(0);
        }
        else
                printf("Server listening..\n");
        len = sizeof(cli);

        // Accept the data packet from client and verification
        connfd = accept(sockfd, (SA*)&cli, &len);
        if (connfd < 0) {
```

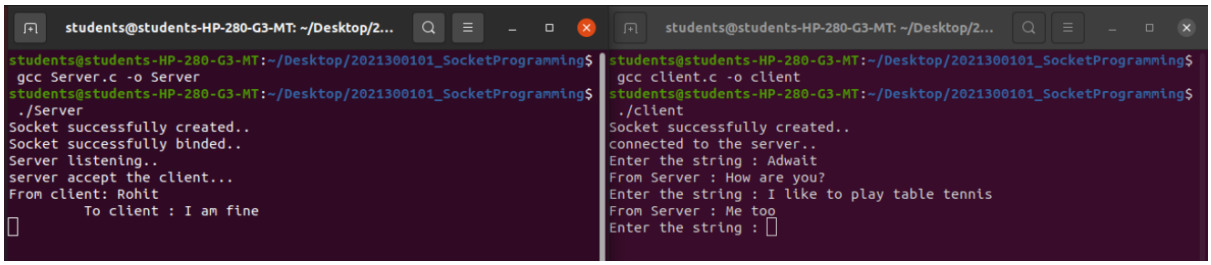
```
		printf("server accept failed...\n");
		exit(0);
	}
	else
		printf("server accept the client...\n");

	// Function for chatting between client and server
	func(connfd);

	// After chatting close the socket
	close(sockfd);
}
```

Client.c

```
#include <arpa/inet.h> // inet_addr()
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h> // bzero()
#include <sys/socket.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
{
	char buff[MAX];
	int n;
	for (;;) {
		bzero(buff, sizeof(buff));
		printf("Enter the string : ");
		n = 0;
		while ((buff[n++] = getchar()) != '\n')
			;
		write(sockfd, buff, sizeof(buff));
		bzero(buff, sizeof(buff));
		read(sockfd, buff, sizeof(buff));
		printf("From Server : %s", buff);
		if ((strncmp(buff, "exit", 4)) == 0) {
			printf("Client Exit...\n");
			break;
		}
	}
}

int main()
{
	int sockfd, connfd;
	struct sockaddr_in servaddr, cli;

	// socket create and verification
	sockfd = socket(AF_INET, SOCK_STREAM, 0);
	if (sockfd == -1) {
		printf("socket creation failed...\n");
		exit(0);
```

```
        }
        else
                printf("Socket successfully created..\n");
        bzero(&servaddr, sizeof(servaddr));

        // assign IP, PORT
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = inet_addr("172.16.31.189");
        servaddr.sin_port = htons(PORT);

        // connect the client socket to server socket
        if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr))
                != 0) {
                printf("connection with the server failed...\n");
                exit(0);
        }
        else
                printf("connected to the server..\n");

        // function for chat
        func(sockfd);

        // close the socket
        close(sockfd);
}
```

Output:



TCP Socket Programming

Within the same PC

TCPServer.c

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
```

```c
#define SA struct sockaddr

// Function designed for chat between client and server.
void func(int connfd)
{
    char buff[MAX];
    int n;
    // infinite loop for chat
    for (;;) {
        bzero(buff, MAX);

        // read the message from client and copy it in buffer
        read(connfd, buff, sizeof(buff));
        // print buffer which contains the client contents
        printf("From client: %s\t To client : ", buff);
        bzero(buff, MAX);
        n = 0;
        // copy server message in the buffer
        while ((buff[n++] = getchar()) != '\n')
            ;

        // and send that buffer to client
        write(connfd, buff, sizeof(buff));

        // if msg contains "Exit" then server exit and chat ended.
        if (strncmp("exit", buff, 4) == 0) {
            printf("Server Exit...\n");
            break;
        }
    }
}

// Driver function
int main()
{
    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));
```

```
    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // Binding newly created socket to given IP and verification
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("socket bind failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully binded..\n");

    // Now server is ready to listen and verification
    if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
    }
    else
        printf("Server listening..\n");
    len = sizeof(cli);

    // Accept the data packet from client and verification
    connfd = accept(sockfd, (SA*)&cli, &len);
    if (connfd < 0) {
        printf("server accept failed...\n");
        exit(0);
    }
    else
        printf("server accept the client...\n");

    // Function for chatting between client and server
    func(connfd);

    // After chatting close the socket
    close(sockfd);
}
```

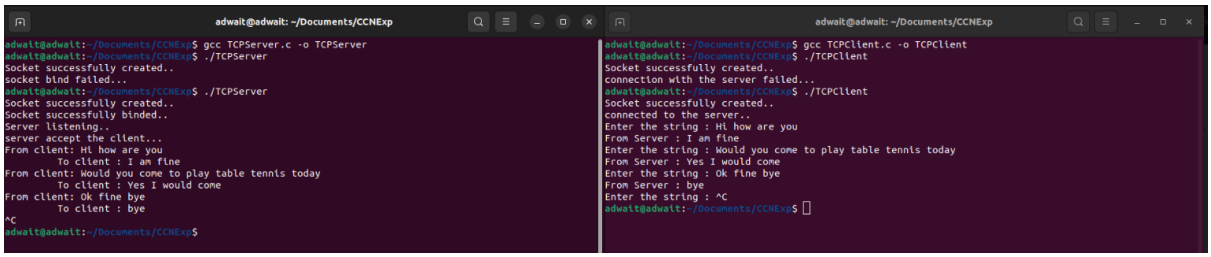
TCPClient.c

```
#include <arpa/inet.h> // inet_addr()
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h> // bzero()
```

```c
#include <sys/socket.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
{
    char buff[MAX];
    int n;
    for (;;) {
        bzero(buff, sizeof(buff));
        printf("Enter the string : ");
        n = 0;
        while ((buff[n++] = getchar()) != '\n')
            ;
        write(sockfd, buff, sizeof(buff));
        bzero(buff, sizeof(buff));
        read(sockfd, buff, sizeof(buff));
        printf("From Server : %s", buff);
        if ((strncmp(buff, "exit", 4)) == 0) {
            printf("Client Exit...\n");
            break;
        }
    }
}

int main()
{
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("10.0.2.15");
    servaddr.sin_port = htons(PORT);

    // connect the client socket to server socket
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr))
```

```
        != 0) {
        printf("connection with the server failed...\n");
        exit(0);
    }
    else
        printf("connected to the server..\n");

    // function for chat
    func(sockfd);

    // close the socket
    close(sockfd);
}
```

Output:



UDP Socket Programming

Within one PC

UDPServer.c

```
// server program for udp connection
#include <stdio.h>
#include <strings.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include<netinet/in.h>
#define PORT 5000
#define MAXLINE 1000

// Driver code
int main()
{
    char buffer[100];
    char *message = "Hello Client";
    int listenfd, len;
    struct sockaddr_in servaddr, cliaddr;
```

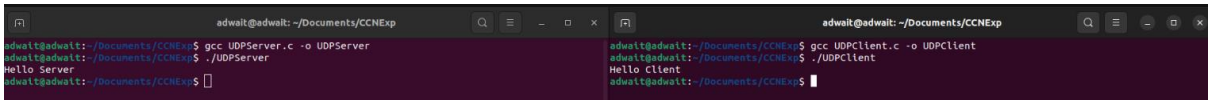
```
    bzero(&servaddr, sizeof(servaddr));

    // Create a UDP Socket
    listenfd = socket(AF_INET, SOCK_DGRAM, 0);
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);
    servaddr.sin_family = AF_INET;

    // bind server address to socket descriptor
    bind(listenfd, (struct sockaddr*)&servaddr, sizeof(servaddr));

    //receive the datagram
    len = sizeof(cliaddr);
    int n = recvfrom(listenfd, buffer, sizeof(buffer),
            0, (struct sockaddr*)&cliaddr,&len); //receive message from
server
    buffer[n] = '\0';
    puts(buffer);

    // send the response
    sendto(listenfd, message, MAXLINE, 0,
        (struct sockaddr*)&cliaddr, sizeof(cliaddr));
}
```

UDPClient.c

```
// udp client driver program
#include <stdio.h>
#include <strings.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include<netinet/in.h>
#include<unistd.h>
#include<stdlib.h>

#define PORT 5000
#define MAXLINE 1000

// Driver code
int main()
{
    char buffer[100];
    char *message = "Hello Server";
    int sockfd, n;
    struct sockaddr_in servaddr;
```

```
    // clear servaddr
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_addr.s_addr = inet_addr("10.0.2.15");
    servaddr.sin_port = htons(PORT);
    servaddr.sin_family = AF_INET;

    // create datagram socket
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    // connect to server
    if(connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr))
< 0)
    {
        printf("\n Error : Connect Failed \n");
        exit(0);
    }

    // request to send datagram
    // no need to specify server address in sendto
    // connect stores the peers IP and port
    sendto(sockfd, message, MAXLINE, 0, (struct sockaddr*)NULL,
sizeof(servaddr));

    // waiting for response
    recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr*)NULL,
NULL);
    puts(buffer);

    // close the descriptor
    close(sockfd);
}
```

Output:



Conclusion:

During an experiment on socket programming, I learned how to create socket objects, bind them to a specific address and port, listen for incoming connections, and send and receive data over those connections. I also learned about different socket types and protocols, such as TCP and UDP, and how to handle errors and exceptions.