

DELHI TECHNOLOGICAL UNIVERSITY



Mathematics And Computing

COMPUTER NETWORKS (MC-308)

Midterm Project 1st Review

SUBMITTED BY:

Aiman Siddiqua - 2K18/MC/008

Apoorva - 2K18/MC/019

SUBMITTED TO:

Sumedha Seniaray

Client-Server File Transfer Console Application using TCP socket programming.

INTRODUCTION

WHAT IS TCP?

TCP refers to the Transmission Control Protocol, which is a highly efficient and reliable protocol designed for end-to-end data transmission over an unreliable network.

A TCP connection uses a three-way handshake to connect the client and the server. It is a process that requires both the client and the server to exchange synchronization (SYN) and acknowledge (ACK) packet before the data transfer takes place.

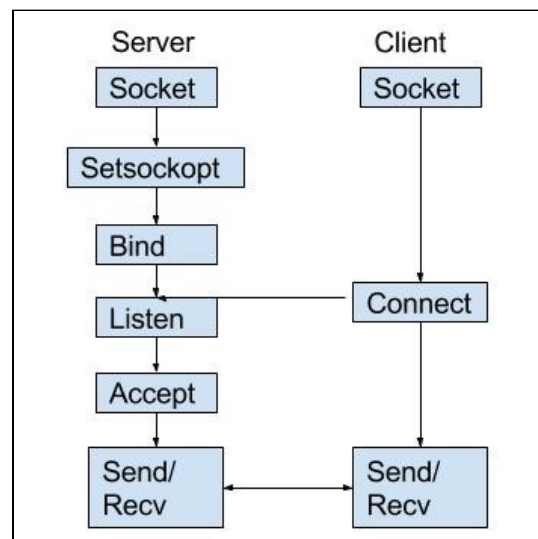
WHAT IS SOCKET PROGRAMMING?

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.

WORK DONE

Understood and implemented the basics of Socket programming in C.

Basic Socket programming involves the various stages as illustrated in the diagram below.



IMPLEMENTATION:

We implemented a basic server-client application as a demonstration. This was an educational exercise in understanding how Socket Programming works by creating a server/client model wherein the server and client exchange a simple Hello message.

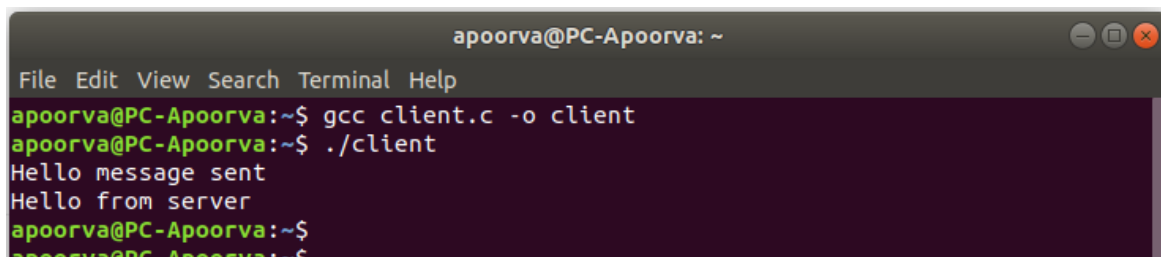
This was accomplished by coding two files in C language in a LINUX environment.

1. server.c
2. client.c

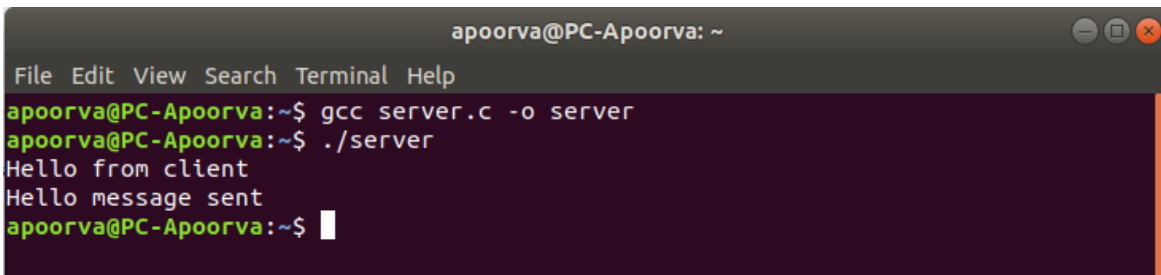
(Both files are given in the appendix)

Both these files were then compiled and executed in separate terminals.

OUTPUT OBTAINED

A terminal window titled 'apoorva@PC-Apoorva: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
apoorva@PC-Apoorva:~$ gcc client.c -o client
apoorva@PC-Apoorva:~$ ./client
Hello message sent
Hello from server
apoorva@PC-Apoorva:~$
```

A terminal window titled 'apoorva@PC-Apoorva: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
apoorva@PC-Apoorva:~$ gcc server.c -o server
apoorva@PC-Apoorva:~$ ./server
Hello from client
Hello message sent
apoorva@PC-Apoorva:~$
```

WORK TO BE DONE

1. Further study on Socket Programming esp. File Handling
2. Coding the main project i.e Transferring a text file from client to server.
3. Preparing a project report which includes all the theory of Socket Programming learnt and the Code and Output of the project.

CONTRIBUTION OF EACH MEMBER

Both members of the team collaborated equally and all work was done together.

APPENDIX: CODE

SERVER SIDE

```
// 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;
}

```

CLIENT SIDE

// 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;
}

```