CSE 1004

NETWORK AND COMMUNICATION

Lab FAT, June 3rd

L23+L24 | PLBG17

WINTER SEMESTER 2020-21

by

SHARADINDU ADHIKARI

19BCE2105

FAT LAB EXAM - 03-06-21

Points
No points

Due today at 1:10 PM ● Closes today at 1:10 PM

Instructions

1. Write a Linux command to remove the empty directory.
2. Write a Network command which works with the Network subsystem.
3. Implement the concept of Hamming code for 7 bit data.

   Input format: 7 bit data

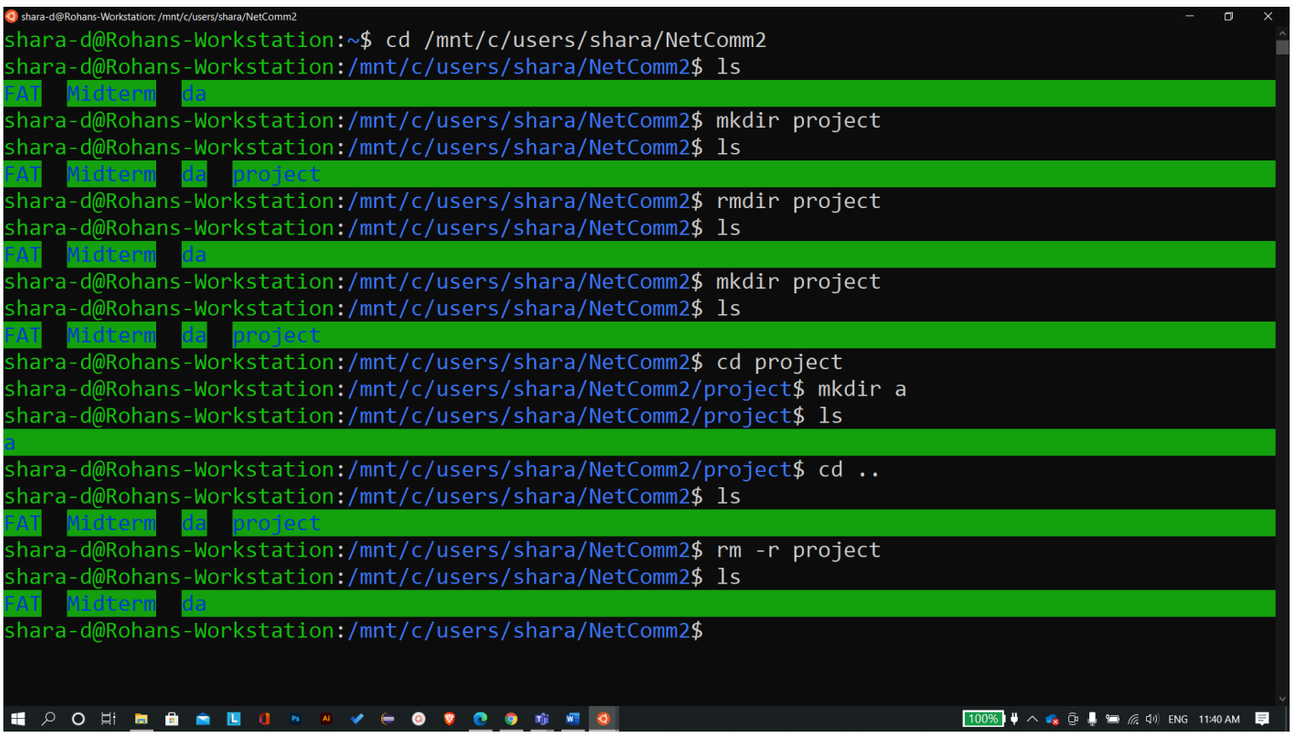
   Output format: Number of redundant bits, Code word
4. Create a Simple chat between client and server using stream connection and analyse the following

Suppose you run the client without starting the server. What happens exactly?
What happens if you start the server after the client but before typing any input?

shara-d@Rohans-Workstation

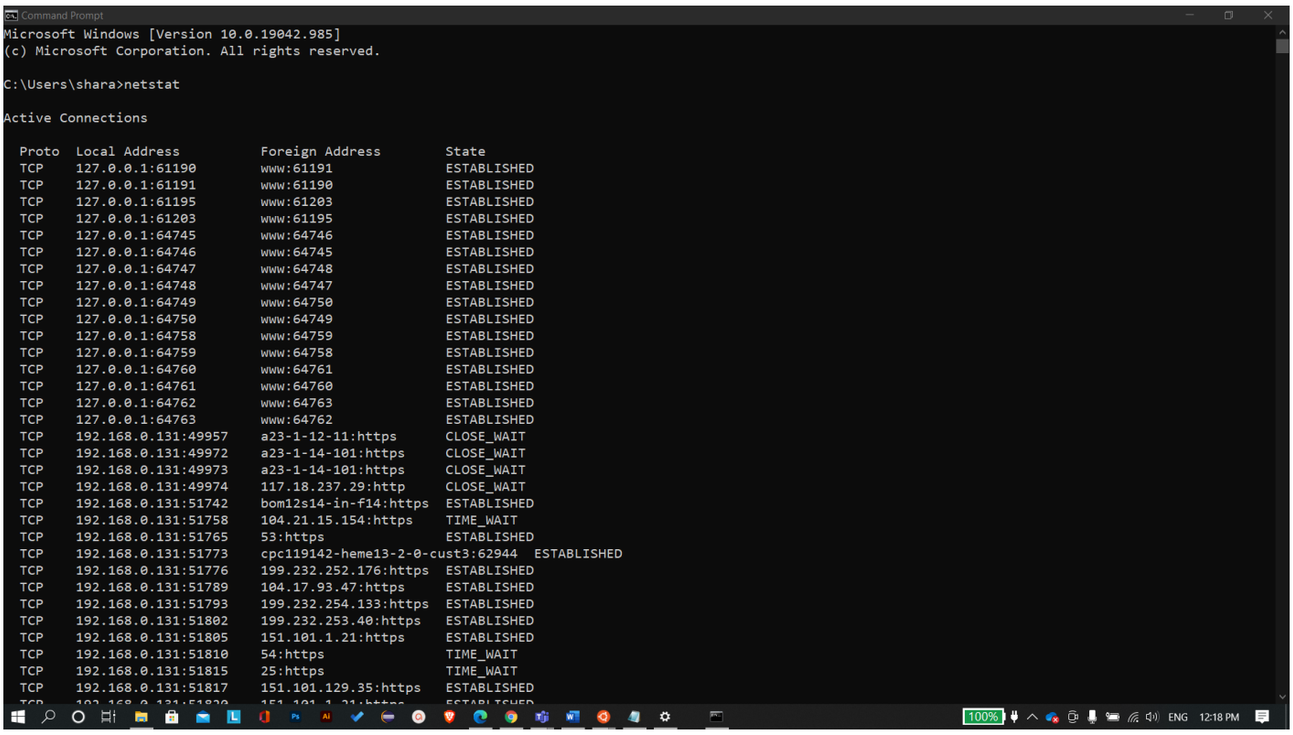
Q1. Write a Linux command to remove the empty directory.

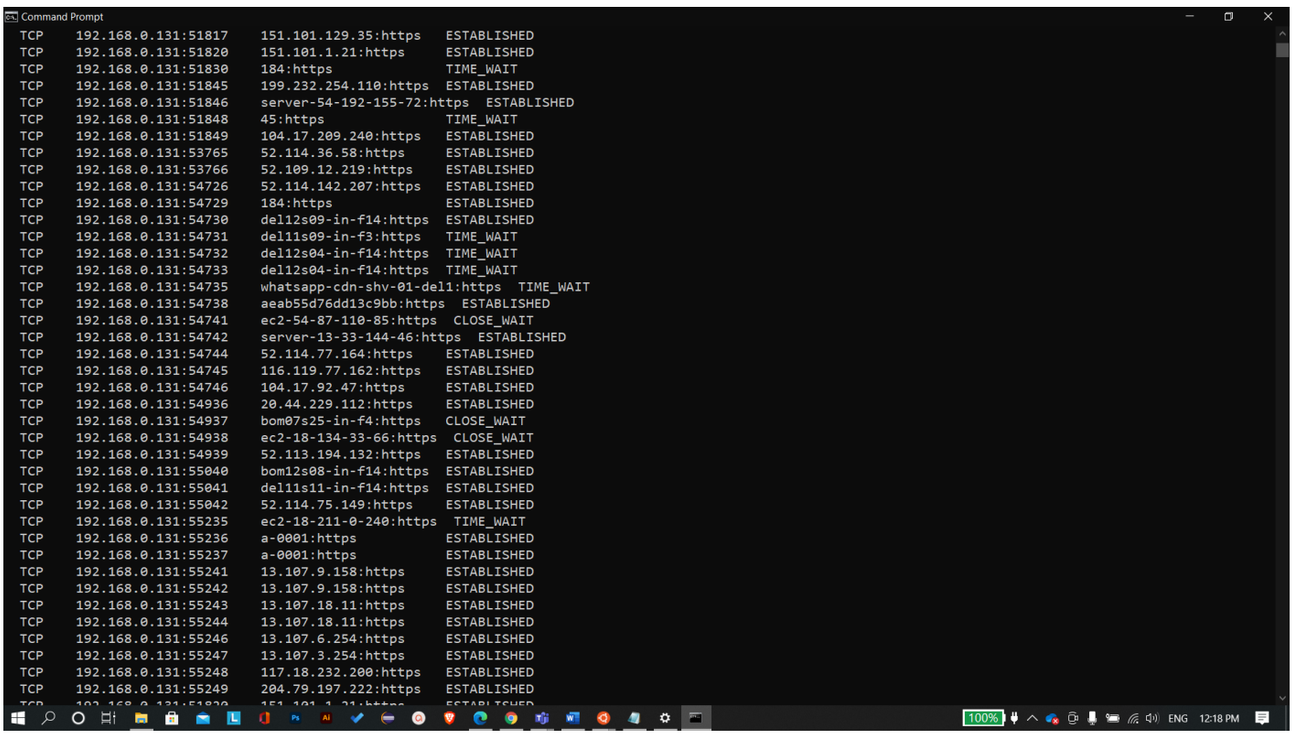
To remove an empty directory, I'm first creating one and will subsequently show the content, followed by removing it as well.

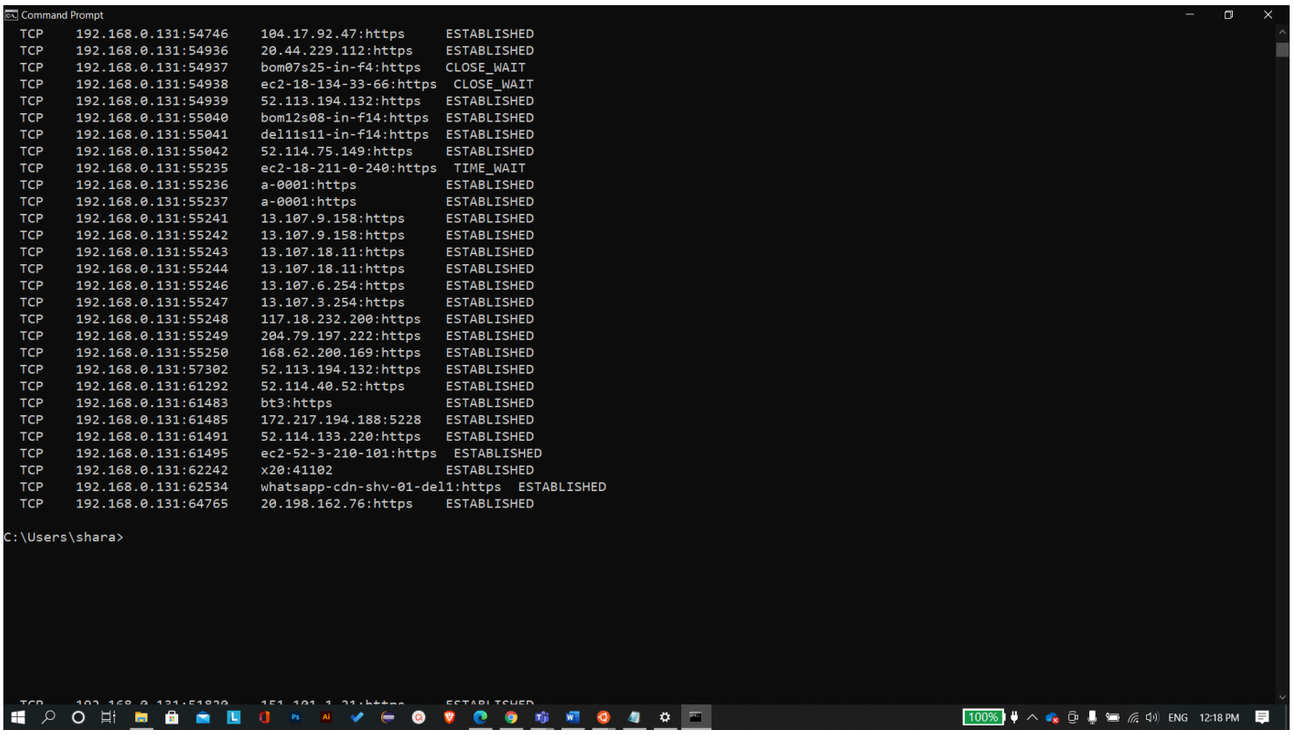


Q2. Write a Network command which works with the Network subsystem.

1 network command which works with the Network subsystem is `netstat`. It displays contents of /proc/net files.


Command Prompt
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.
C:\Users\shara>netstat
Active Connections
Proto Local Address Foreign Address State
TCP 127.0.0.1:61190 www:61191 ESTABLISHED
TCP 127.0.0.1:61191 www:61190 ESTABLISHED
TCP 127.0.0.1:61195 www:61203 ESTABLISHED
TCP 127.0.0.1:61203 www:61195 ESTABLISHED
TCP 127.0.0.1:64745 www:64746 ESTABLISHED
TCP 127.0.0.1:64746 www:64745 ESTABLISHED
TCP 127.0.0.1:64747 www:64748 ESTABLISHED
TCP 127.0.0.1:64748 www:64747 ESTABLISHED
TCP 127.0.0.1:64749 www:64750 ESTABLISHED
TCP 127.0.0.1:64750 www:64749 ESTABLISHED
TCP 127.0.0.1:64758 www:64759 ESTABLISHED
TCP 127.0.0.1:64759 www:64758 ESTABLISHED
TCP 127.0.0.1:64760 www:64761 ESTABLISHED
TCP 127.0.0.1:64761 www:64760 ESTABLISHED
TCP 127.0.0.1:64762 www:64763 ESTABLISHED
TCP 127.0.0.1:64763 www:64762 ESTABLISHED
TCP 192.168.0.131:49957 a23-1-12-11:https CLOSE_WAIT
TCP 192.168.0.131:49972 a23-1-14-101:https CLOSE_WAIT
TCP 192.168.0.131:49973 a23-1-14-101:https CLOSE_WAIT
TCP 192.168.0.131:49974 117.18.237.29:http CLOSE_WAIT
TCP 192.168.0.131:51742 bom12s14-in-f14:https ESTABLISHED
TCP 192.168.0.131:51758 104.21.15.154:https TIME_WAIT
TCP 192.168.0.131:51765 53:https ESTABLISHED
TCP 192.168.0.131:51773 cpc119142-heme13-2-0-cust3:62944 ESTABLISHED
TCP 192.168.0.131:51776 199.232.252.176:https ESTABLISHED
TCP 192.168.0.131:51789 104.17.93.47:https ESTABLISHED
TCP 192.168.0.131:51793 199.232.254.133:https ESTABLISHED
TCP 192.168.0.131:51802 199.232.253.40:https ESTABLISHED
TCP 192.168.0.131:51805 151.101.1.21:https ESTABLISHED
TCP 192.168.0.131:51810 54:https TIME_WAIT
TCP 192.168.0.131:51815 25:https TIME_WAIT
TCP 192.168.0.131:51817 151.101.129.35:https ESTABLISHED
100%
ENG
12:18 PM


Command Prompt
TCP 192.168.0.131:51817 151.101.129.35:https ESTABLISHED
TCP 192.168.0.131:51820 151.101.1.21:https ESTABLISHED
TCP 192.168.0.131:51830 184:https TIME_WAIT
TCP 192.168.0.131:51845 199.232.254.110:https ESTABLISHED
TCP 192.168.0.131:51846 server-54-192-155-72:https ESTABLISHED
TCP 192.168.0.131:51848 45:https TIME_WAIT
TCP 192.168.0.131:51849 104.17.209.240:https ESTABLISHED
TCP 192.168.0.131:53765 52.114.36.58:https ESTABLISHED
TCP 192.168.0.131:53766 52.109.12.219:https ESTABLISHED
TCP 192.168.0.131:54726 52.114.142.207:https ESTABLISHED
TCP 192.168.0.131:54729 184:https ESTABLISHED
TCP 192.168.0.131:54730 del12s09-in-f14:https ESTABLISHED
TCP 192.168.0.131:54731 del11s09-in-f3:https TIME_WAIT
TCP 192.168.0.131:54732 del12s04-in-f14:https TIME_WAIT
TCP 192.168.0.131:54733 del12s04-in-f14:https TIME_WAIT
TCP 192.168.0.131:54735 whatsapp-cdn-shv-01-del1:https TIME_WAIT
TCP 192.168.0.131:54738 aeab55d76dd13c9bb:https ESTABLISHED
TCP 192.168.0.131:54741 ec2-54-87-110-85:https CLOSE_WAIT
TCP 192.168.0.131:54742 server-13-33-144-46:https ESTABLISHED
TCP 192.168.0.131:54744 52.114.77.164:https ESTABLISHED
TCP 192.168.0.131:54745 116.119.77.162:https ESTABLISHED
TCP 192.168.0.131:54746 104.17.92.47:https ESTABLISHED
TCP 192.168.0.131:54936 20.44.229.112:https ESTABLISHED
TCP 192.168.0.131:54937 bom07s25-in-f4:https CLOSE_WAIT
TCP 192.168.0.131:54938 ec2-18-134-33-66:https CLOSE_WAIT
TCP 192.168.0.131:54939 52.113.194.132:https ESTABLISHED
TCP 192.168.0.131:55040 bom12s08-in-f14:https ESTABLISHED
TCP 192.168.0.131:55041 del11s11-in-f14:https ESTABLISHED
TCP 192.168.0.131:55042 52.114.75.149:https ESTABLISHED
TCP 192.168.0.131:55235 ec2-18-211-0-240:https TIME_WAIT
TCP 192.168.0.131:55236 a-0001:https ESTABLISHED
TCP 192.168.0.131:55237 a-0001:https ESTABLISHED
TCP 192.168.0.131:55241 13.107.9.158:https ESTABLISHED
TCP 192.168.0.131:55242 13.107.9.158:https ESTABLISHED
TCP 192.168.0.131:55243 13.107.18.11:https ESTABLISHED
TCP 192.168.0.131:55244 13.107.18.11:https ESTABLISHED
TCP 192.168.0.131:55246 13.107.6.254:https ESTABLISHED
TCP 192.168.0.131:55247 13.107.3.254:https ESTABLISHED
TCP 192.168.0.131:55248 117.18.232.200:https ESTABLISHED
TCP 192.168.0.131:55249 204.79.197.222:https ESTABLISHED
100%
ENG
12:18 PM


Command Prompt
TCP 192.168.0.131:54746 104.17.92.47:https ESTABLISHED
TCP 192.168.0.131:54936 20.44.229.112:https ESTABLISHED
TCP 192.168.0.131:54937 bom07s25-in-f4:https CLOSE_WAIT
TCP 192.168.0.131:54938 ec2-18-134-33-66:https CLOSE_WAIT
TCP 192.168.0.131:54939 52.113.194.132:https ESTABLISHED
TCP 192.168.0.131:55040 bom12s08-in-f14:https ESTABLISHED
TCP 192.168.0.131:55041 del11s11-in-f14:https ESTABLISHED
TCP 192.168.0.131:55042 52.114.75.149:https ESTABLISHED
TCP 192.168.0.131:55235 ec2-18-211-0-240:https TIME_WAIT
TCP 192.168.0.131:55236 a-0001:https ESTABLISHED
TCP 192.168.0.131:55237 a-0001:https ESTABLISHED
TCP 192.168.0.131:55241 13.107.9.158:https ESTABLISHED
TCP 192.168.0.131:55242 13.107.9.158:https ESTABLISHED
TCP 192.168.0.131:55243 13.107.18.11:https ESTABLISHED
TCP 192.168.0.131:55244 13.107.18.11:https ESTABLISHED
TCP 192.168.0.131:55246 13.107.6.254:https ESTABLISHED
TCP 192.168.0.131:55247 13.107.3.254:https ESTABLISHED
TCP 192.168.0.131:55248 117.18.232.200:https ESTABLISHED
TCP 192.168.0.131:55249 204.79.197.222:https ESTABLISHED
TCP 192.168.0.131:55250 168.62.200.169:https ESTABLISHED
TCP 192.168.0.131:57302 52.113.194.132:https ESTABLISHED
TCP 192.168.0.131:61292 52.114.40.52:https ESTABLISHED
TCP 192.168.0.131:61483 bt3:https ESTABLISHED
TCP 192.168.0.131:61485 172.217.194.188:5228 ESTABLISHED
TCP 192.168.0.131:61491 52.114.133.220:https ESTABLISHED
TCP 192.168.0.131:61495 ec2-52-3-210-101:https ESTABLISHED
TCP 192.168.0.131:62242 x20:41102 ESTABLISHED
TCP 192.168.0.131:62534 whatsapp-cdn-shv-01-del1:https ESTABLISHED
TCP 192.168.0.131:64765 20.198.162.76:https ESTABLISHED
C:\Users\shara>
100%
ENG
12:18 PM

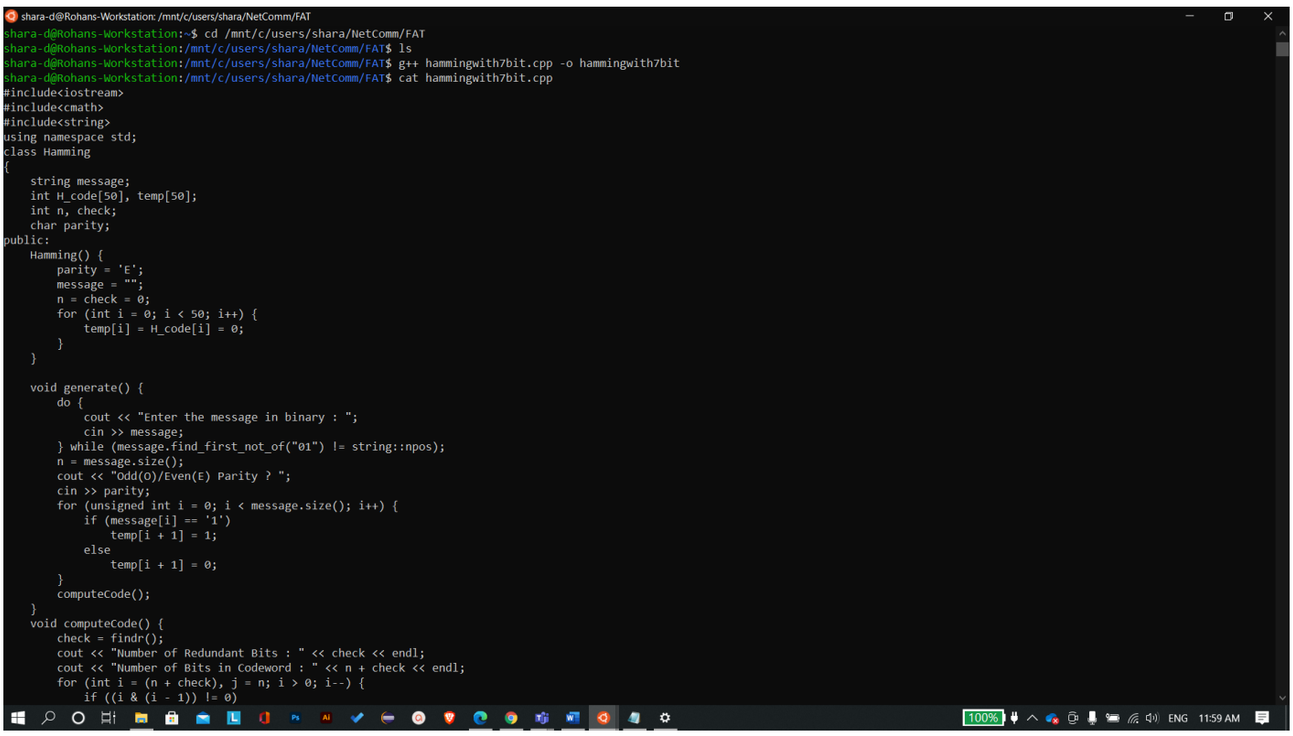
Q3. Implement the concept of Hamming code for 7-bit data. Input format: 7-bit data

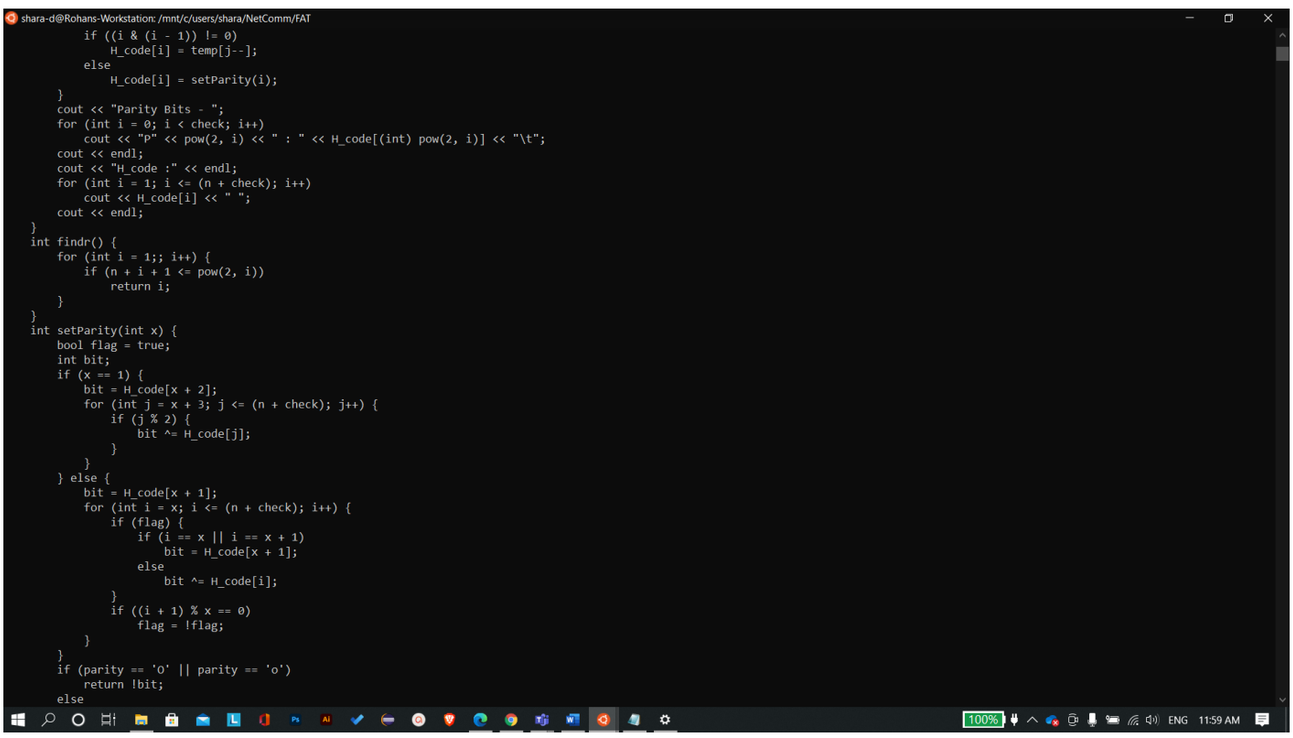
Code:

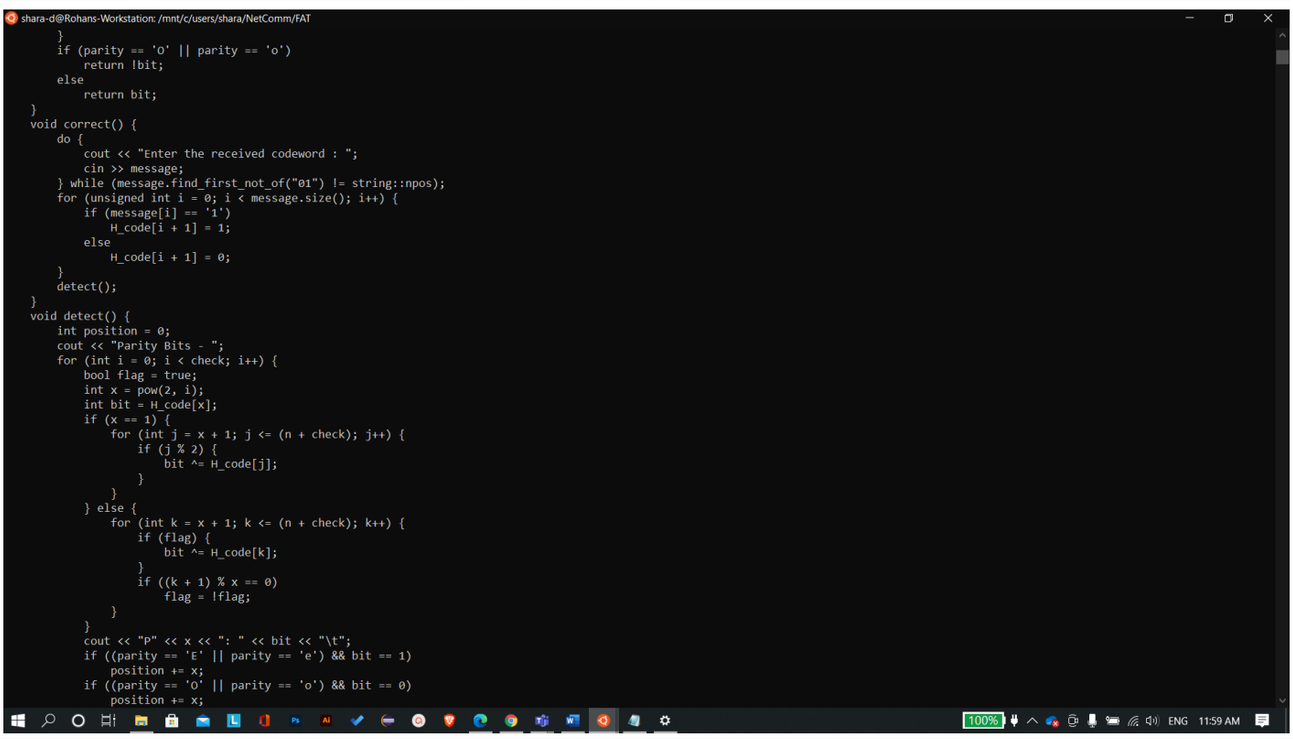
```
#include<iostream>
#include<cmath>
#include<string>
using namespace std;
class Hamming
{
    string message;
    int H_code[50], temp[50];
    int n, check;
    char parity;
public:
    Hamming() {
        parity = 'E';
        message = "";
        n = check = 0;
        for (int i = 0; i < 50; i++) {
            temp[i] = H_code[i] = 0;
        }
    }

    void generate() {
        do {
            cout << "Enter the message in binary : ";
            cin >> message;
        } while (message.find_first_not_of("01") != string::npos);
        n = message.size();
        cout << "Odd(O)/Even(E) Parity ? ";
        cin >> parity;
        for (unsigned int i = 0; i < message.size(); i++) {
            if (message[i] == '1')
                temp[i + 1] = 1;
            else
                temp[i + 1] = 0;
        }
        computeCode();
    }
    void computeCode() {
        check = findr();
        cout << "Number of Redundant Bits : " << check << endl;
        cout << "Number of Bits in Codeword : " << n + check << endl;
        for (int i = (n + check), j = n; i > 0; i--) {
            if ((i & (i - 1)) != 0)
                H_code[i] = temp[j--];
            else
                H_code[i] = setParity(i);
        }
        cout << "Parity Bits - ";
        for (int i = 0; i < check; i++)
            cout << "P" << pow(2, i) << " : " << H_code[(int) pow(2, i)] << "\t";
        cout << endl;
        cout << "H_code :" << endl;
        for (int i = 1; i <= (n + check); i++)
            cout << H_code[i] << " ";
        cout << endl;
    }
    int findr() {
        for (int i = 1;; i++) {
            if (n + i + 1 <= pow(2, i))
                return i;
        }
    }
    int setParity(int x) {
        bool flag = true;
        int bit;
        if (x == 1) {
            bit = H_code[x + 2];
            for (int j = x + 3; j <= (n + check); j++) {
                if (j % 2) {
                    bit ^= H_code[j];
                }
            }
        } else {
            bit = H_code[x + 1];
            for (int i = x; i <= (n + check); i++) {
                if (flag) {
                    if (i == x || i == x + 1)
                        bit = H_code[x + 1];
                    else
                        bit ^= H_code[i];
```

```
                }
                if ((i + 1) % x == 0)
                    flag = !flag;
            }
        }
        if (parity == 'O' || parity == 'o')
            return !bit;
        else
            return bit;
    }
    void correct() {
        do {
            cout << "Enter the received codeword : ";
            cin >> message;
        } while (message.find_first_not_of("01") != string::npos);
        for (unsigned int i = 0; i < message.size(); i++) {
            if (message[i] == '1')
                H_code[i + 1] = 1;
            else
                H_code[i + 1] = 0;
        }
        detect();
    }
    void detect() {
        int position = 0;
        cout << "Parity Bits - ";
        for (int i = 0; i < check; i++) {
            bool flag = true;
            int x = pow(2, i);
            int bit = H_code[x];
            if (x == 1) {
                for (int j = x + 1; j <= (n + check); j++) {
                    if (j % 2) {
                        bit ^= H_code[j];
                    }
                }
            } else {
                for (int k = x + 1; k <= (n + check); k++) {
                    if (flag) {
                        bit ^= H_code[k];
                    }
                    if ((k + 1) % x == 0)
                        flag = !flag;
                }
            }
            cout << "P" << x << ": " << bit << "\t";
            if ((parity == 'E' || parity == 'e') && bit == 1)
                position += x;
            if ((parity == 'O' || parity == 'o') && bit == 0)
                position += x;
        }
        cout << endl << "Received Codeword :" << endl;
        for (int i = 1; i <= (n + check); i++)
            cout << H_code[i] << " ";
        cout << endl;
        if (position != 0) {
            cout << "Error at bit : " << position << endl;
            H_code[position] = !H_code[position];
            cout << "Corrected Codeword : " << endl;
            for (int i = 1; i <= (n + check); i++)
                cout << H_code[i] << " ";
            cout << endl;
        } else
            cout << "No Error in Received code." << endl;
        cout << "Received Message is : ";
        for (int i = 1; i <= (n + check); i++)
            if ((i & (i - 1)) != 0)
                cout << H_code[i] << " ";
        cout << endl;
    }
};
int main() {
    char choice;
    do {
        Hamming a;
        cout << "At Sender's side : " << endl;
        a.generate();
        cout << endl << "At Receiver's Side : " << endl;
        a.correct();
        cout << endl << "Enter another code ? (Y/N) : ";
        cin >> choice;
        cout << endl;
    } while (choice == 'y' || choice == 'Y');
    return 0;
}
```

Screenshots of Input:







```
shara-d@Rohans-Workstation: /mnt/c/users/shara/NetComm/FAT
            }
        }
        cout << "P" << x << ": " << bit << "\t";
        if ((parity == 'E' || parity == 'e') && bit == 1)
            position += x;
        if ((parity == 'O' || parity == 'o') && bit == 0)
            position += x;
    }
    cout << endl << "Received Codeword :" << endl;
    for (int i = 1; i <= (n + check); i++)
        cout << H_code[i] << " ";
    cout << endl;
    if (position != 0) {
        cout << "Error at bit : " << position << endl;
        H_code[position] = !H_code[position];
        cout << "Corrected Codeword : " << endl;
        for (int i = 1; i <= (n + check); i++)
            cout << H_code[i] << " ";
        cout << endl;
    } else
        cout << "No Error in Received code." << endl;
    cout << "Received Message is : ";
    for (int i = 1; i <= (n + check); i++)
        if ((i & (i - 1)) != 0)
            cout << H_code[i] << " ";
    cout << endl;
  }
};
int main() {
  char choice;
  do {
    Hamming a;
    cout << "At Sender's side : " << endl;
    a.generate();
    cout << endl << "At Receiver's Side : " << endl;
    a.correct();
    cout << endl << "Enter another code ? (Y/N) : ";
    cin >> choice;
    cout << endl;
  } while (choice == 'y' || choice == 'Y');
  return 0;
}
shara-d@Rohans-Workstation:/mnt/c/users/shara/NetComm/FAT$ ./hammingwith7bit
At Sender's side :
Enter the message in binary : 1011010
Odd(O)/Even(E) Parity ? O
```

OUTPUT:

```
shara-d@Rohans-Workstation: /mnt/c/users/shara/NetComm/FAT
  } while (choice == 'y' || choice == 'Y');
  return 0;
}
shara-d@Rohans-Workstation:/mnt/c/users/shara/NetComm/FAT$ ./hammingwith7bit
At Sender's side :
Enter the message in binary : 1011010
Odd(O)/Even(E) Parity ? O
Number of Redundant Bits : 4
Number of Bits in Codeword : 11
Parity Bits - P1 : 1    P2 : 1  P4 : 1  P8 : 0
H_code :
1 1 1 1 0 1 1 0 0 1 0

At Receiver's Side :
Enter the received codeword : 11110110010
Parity Bits - P1: 1     P2: 1   P4: 1   P8: 1
Received Codeword :
1 1 1 1 0 1 1 0 0 1 0
No Error in Received code.
Received Message is : 1 0 1 1 0 1 0

Enter another code ? (Y/N) : Y

At Sender's side :
Enter the message in binary : 0100101
Odd(O)/Even(E) Parity ? E
Number of Redundant Bits : 4
Number of Bits in Codeword : 11
Parity Bits - P1 : 1    P2 : 1  P4 : 1  P8 : 0
H_code :
1 1 0 1 1 0 0 0 1 0 1

At Receiver's Side :
Enter the received codeword : 11011000101
Parity Bits - P1: 0     P2: 0   P4: 0   P8: 0
Received Codeword :
1 1 0 1 1 0 0 0 1 0 1
No Error in Received code.
Received Message is : 0 1 0 0 1 0 1

Enter another code ? (Y/N) : N

shara-d@Rohans-Workstation:/mnt/c/users/shara/NetComm/FAT$
```

Q4. Create a Simple chat between client and server using stream connection and analyse the following:

Suppose you run the client without starting the server. What happens exactly? What happens if you start the server after the client but before typing any input?

Since it's not specifically mentioned in the question about which client to choose, I'm going ahead with TCP here.

PART A:

SERVER:

```
#include<stdio.h>
#include<netinet/in.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netdb.h>
#include<stdlib.h>
#include<string.h>
#define MAX 80
#define PORT 43454
#define SA struct sockaddr
void func(int sockfd)
{
    char buff[MAX];
    int n;
    for(;;)
    {
        bzero(buff,MAX);
        read(sockfd,buff,sizeof(buff));
        printf("From client: %s\t To client : ",buff);
        bzero(buff,MAX);
        n=0;
        while((buff[n++]=getchar())!='\n');
        write(sockfd,buff,sizeof(buff));
        if(strncmp("exit",buff,4)==0)
        {
            printf("Server Exit...\n");break;
        }
    }
}
int main()
{
    int sockfd,connfd,len;
    struct sockaddr_in servaddr,cli;
    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));
    servaddr.sin_family=AF_INET;
    servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
    servaddr.sin_port=htons(PORT);
    if((bind(sockfd,(SA*)&servaddr, sizeof(servaddr)))!=0)
    {
        printf("socket bind failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully binded..\n");
    if((listen(sockfd,5))!=0)
    {
        printf("Listen failed...\n");exit(0);
    }
    else
        printf("Server listening..\n");
    len=sizeof(cli);
    connfd=accept(sockfd,(SA *)&cli,&len);
    if(connfd<0)
    {
        printf("server acccept failed...\n");
        exit(0);
    }
    else
        printf("server acccept the client...\n");
    func(connfd);
    close(sockfd);
}
```

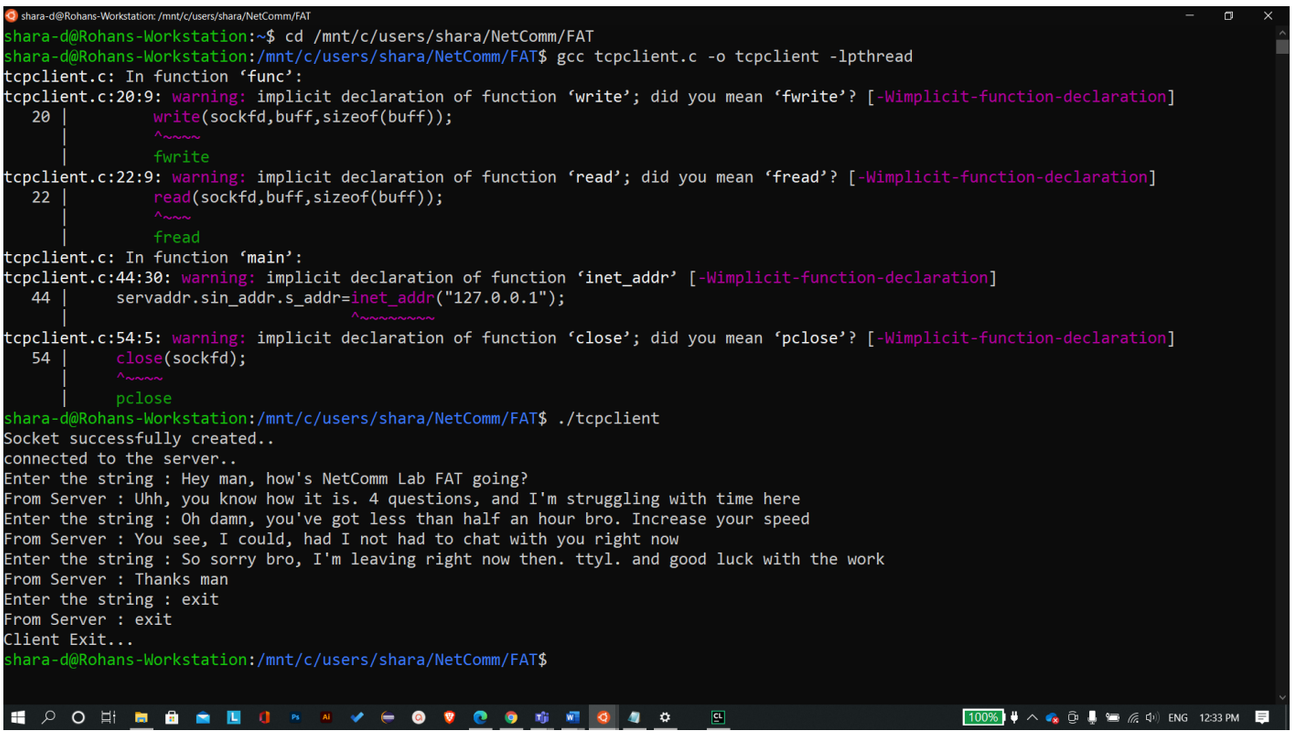
CLIENT:

```
#include<stdio.h>
#include<netinet/in.h>
#include<sys/types.h>
#include<sys/socket.h>
```

```c
#include<netdb.h>
#include<string.h>
#include<stdlib.h>
#define MAX 80
#define PORT 43454
#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;
    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));
    servaddr.sin_family=AF_INET;
    servaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
    servaddr.sin_port=htons(PORT);
    if(connect(sockfd,(SA *)&servaddr,sizeof(servaddr))!=0)
    {
        printf("connection with the server failed...\n");
        exit(0);
    }
    else
        printf("connected to the server..\n");
    func(sockfd);
    close(sockfd);
}
```

OUTPUT:

```
shara-d@Rohans-Workstation: /mnt/c/users/shara/NetComm/FAT
shara-d@Rohans-Workstation:~$ cd /mnt/c/users/shara/NetComm/FAT
shara-d@Rohans-Workstation:/mnt/c/users/shara/NetComm/FAT$ gcc tcpserver.c -o tcpserver -lpthread
tcpserver.c: In function ‘func’:
tcpserver.c:18:9: warning: implicit declaration of function ‘read’; did you mean ‘fread’? [-Wimplicit-function-declaration]
   18 |         read(sockfd,buff,sizeof(buff));
      |         ^~~~
      |         fread
tcpserver.c:23:9: warning: implicit declaration of function ‘write’; did you mean ‘fwrite’? [-Wimplicit-function-declaration]
   23 |         write(sockfd,buff,sizeof(buff));
      |         ^~~~~
      |         fwrite
tcpserver.c: In function ‘main’:
tcpserver.c:69:5: warning: implicit declaration of function ‘close’; did you mean ‘pclose’? [-Wimplicit-function-declaration]
   69 |     close(sockfd);
      |     ^~~~~
      |     pclose
shara-d@Rohans-Workstation:/mnt/c/users/shara/NetComm/FAT$ ./tcpserver
Socket successfully created..
Socket successfully binded..
Server listening..
server acccept the client...
From client: Hey man, how's NetComm Lab FAT going?
        To client : Uhh, you know how it is. 4 questions, and I'm struggling with time here
From client: Oh damn, you've got less than half an hour bro. Increase your speed
        To client : You see, I could, had I not had to chat with you right now
From client: So sorry bro, I'm leaving right now then. ttyl. and good luck with the work
        To client : Thanks man
exit
From client: exit
        To client : Server Exit...
shara-d@Rohans-Workstation:/mnt/c/users/shara/NetComm/FAT$
```



PART B

For the TCP application, as soon as the client is executed, it attempts to initiate a TCP connection with the server. If the TCP server is not running, then the client will fail to make a connection.
For the UDP application, the client does not initiate connections (since UDP is connectionless) with the UDP server immediately upon execution.

If the server is started after the client but before typing any input, the connection is established normally after running client and code executes normally without throwing any errors.