Network Lab1: Socket Programming

V Venkataraman 106118106

**TCP Implementation**

Server\_tcp.c

#include <unistd.h>

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <string.h>

#define PORT 8080

typedef enum{ DATA,ACK }   MSGKIND;

struct timeval timeout;

struct MESSAGE

{

    MSGKIND type;

    int seq;

    unsigned int len;

    char msg[100];

    int parity;

};

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

{

    timeout.tv\_sec = 1;

    timeout.tv\_usec = 0;

    int server\_fd, new\_socket, valread;

    struct sockaddr\_in address;

    int opt = 1;

    int addrlen = sizeof(address);

    char buffer[1024] = {0};

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

    {

        perror("socket failed");

        exit(EXIT\_FAILURE);

    }

    if (setsockopt(server\_fd, SOL\_SOCKET, SO\_RCVTIMEO, (char \*)&timeout, sizeof(timeout)) < 0)

    {

        perror("setsockopt");

        exit(EXIT\_FAILURE);

    }

    address.sin\_family = AF\_INET;

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

    address.sin\_port = htons( PORT );

    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);

    }

    if (setsockopt(new\_socket, SOL\_SOCKET, SO\_RCVTIMEO, (char \*)&timeout, sizeof(timeout)) < 0)

    {

        perror("setsockopt");

        exit(EXIT\_FAILURE);

    }

    // Declare the file pointer

    FILE \*filePointer ;

    // Declare the variable for the data to be read from file

    char dataToBeRead[100];

    filePointer = fopen("read.txt", "r") ;

    if ( filePointer == NULL )

    {

        perror( "file failed to open." ) ;

    }

    else

    {

        // Read the dataToBeRead from the file

        // using fgets() method

        int flag=1, s=0;

        int count=0;

        while( fgets ( dataToBeRead, 100, filePointer ) != NULL )

        {

            while(1)

            {

                count++;

                struct MESSAGE\* Message = (struct MESSAGE\*) malloc(sizeof(struct MESSAGE));

                struct MESSAGE\* Acknowledge = (struct MESSAGE\*) malloc(sizeof(struct MESSAGE));

                Message->type = DATA;

                Message->len = strlen(dataToBeRead);

                strcpy(Message->msg, dataToBeRead);

                Message->seq = s;

                int sum=0;

                for(int j=0;j<strlen(dataToBeRead);j++)

                {

                    sum+=dataToBeRead[j];

                }

                    Message->parity =  sum%2;

                send(new\_socket,(void\*)Message, sizeof(struct MESSAGE), 0);

                printf("MSG: %d-%s\n",s,dataToBeRead);

                if(recv(new\_socket, Acknowledge, sizeof(struct MESSAGE), 0) > 0)

                {

                    printf("ACK: %d\n",Acknowledge->seq);

                    if(Acknowledge->type == ACK && Acknowledge->seq == s)

                    {

                        sleep(1);

                        // if(flag==0) flag=1;

                        // else flag=0;

                        break;

                    }

                }

            }

            if(s==0) s=1;

            else s=0;

         }

        // Closing the file using fclose()

        fclose(filePointer) ;

    }

    close(new\_socket);

    return 0;

}

Client\_tcp.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#define PORT 8080

typedef enum{ DATA,ACK }   MSGKIND;

struct MESSAGE

{

    MSGKIND type;

    int seq;

    unsigned int len;

    char msg[100];

    int parity;

};

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

{

    int sock = 0, valread;

    struct sockaddr\_in serv\_addr;

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

        return -1;

    }

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

    {

        printf("\nConnection Failed \n");

        return -1;

    }

    // Declare the file pointer

    FILE \*filePointer ;

    // Get the data to be written in file

    char dataToBeWritten[100] ;

    filePointer = fopen("write.txt", "w") ;

    if ( filePointer == NULL )

    {

        printf( "file failed to open." ) ;

    }

    else

    {

            int s=1;

            char null[1]={'N'};

            int i=0;

            while(i<5)

            {

                while(1)

                {

                    struct MESSAGE\* Message = (struct MESSAGE\*) malloc(sizeof(struct MESSAGE));

                    struct MESSAGE\* Acknowledge = (struct MESSAGE\*) malloc(sizeof(struct MESSAGE));

                    Acknowledge->type = ACK;

                    Acknowledge->len = 0;

                    strcpy(Acknowledge->msg, null);

                    Acknowledge->seq=s;

                    send(sock,(void\*)Acknowledge, sizeof(struct MESSAGE), 0);

                    if(recv(sock, Message, sizeof(struct MESSAGE), 0) > 0)

                    {

                        if(Message->type == DATA && Message->seq + s == 1)

                        {

                            int sum=0;

                            for(int j=0;j<strlen(Message->msg);j++)

                            {

                                sum+=Message->msg[j];

                            }

                            if(Message->parity!=sum%2)

                                continue;

                            if(s==0) s=1;

                            else s=0;

                            strcpy(dataToBeWritten, Message->msg);

                            fputs(dataToBeWritten, filePointer) ;

                            break;

                        }

                    }

                }

            i++;

            }

        // Closing the file using fclose()

        fclose(filePointer) ;

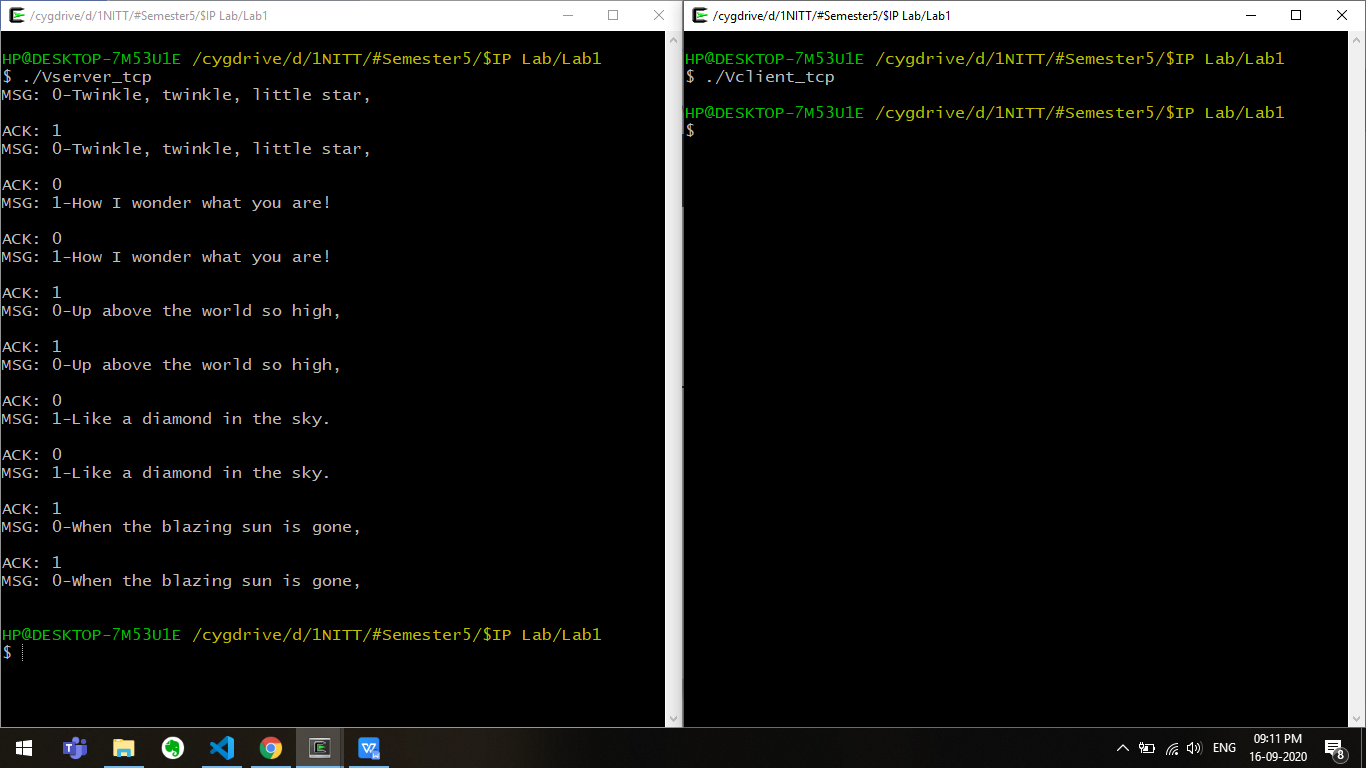
    }

    close(sock);

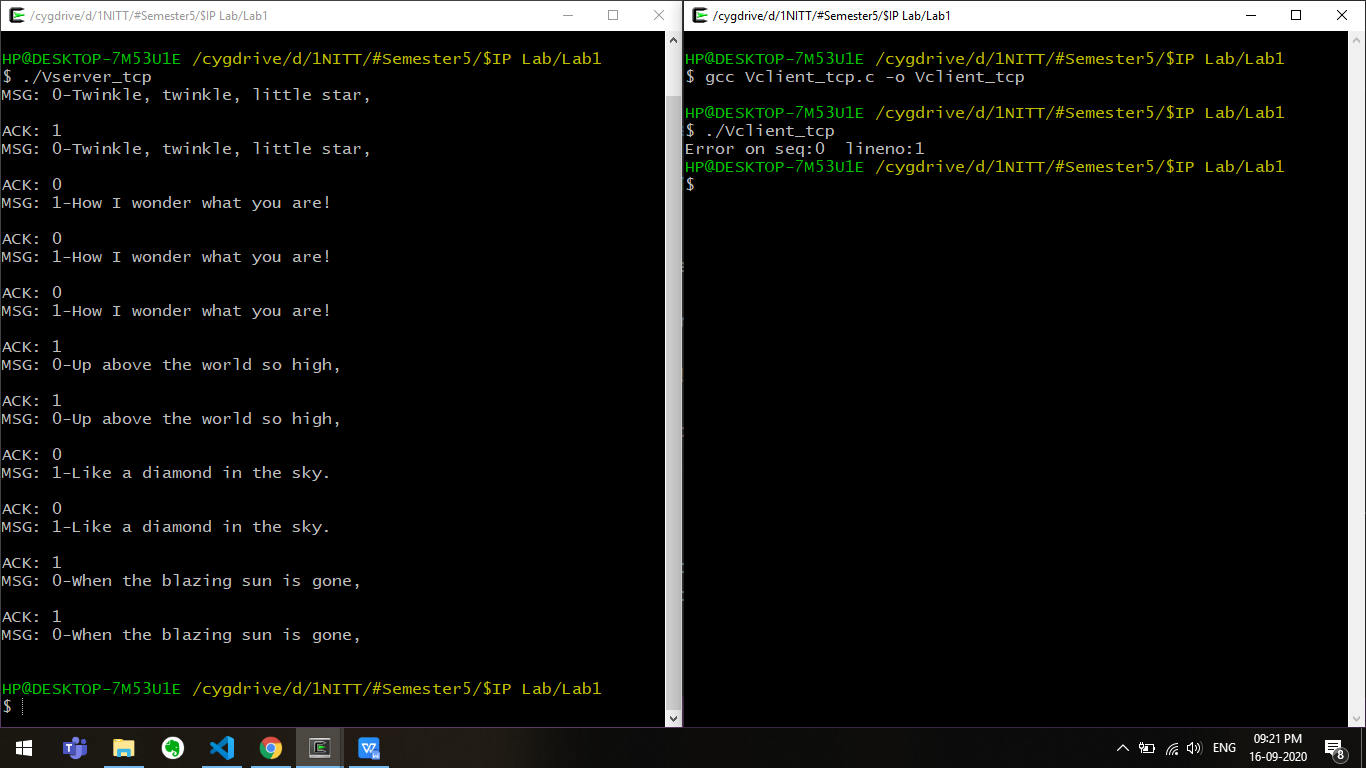
    return 0;

}

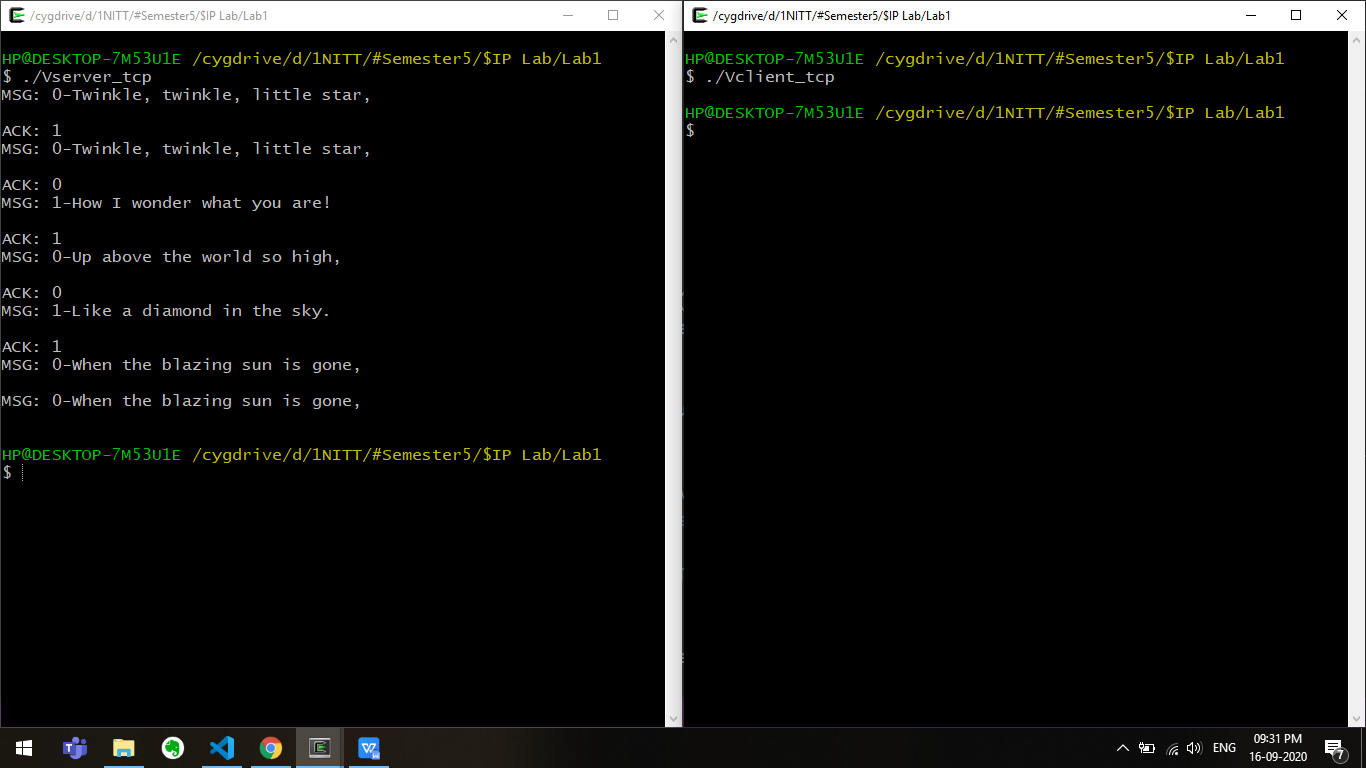
Normal Transmission



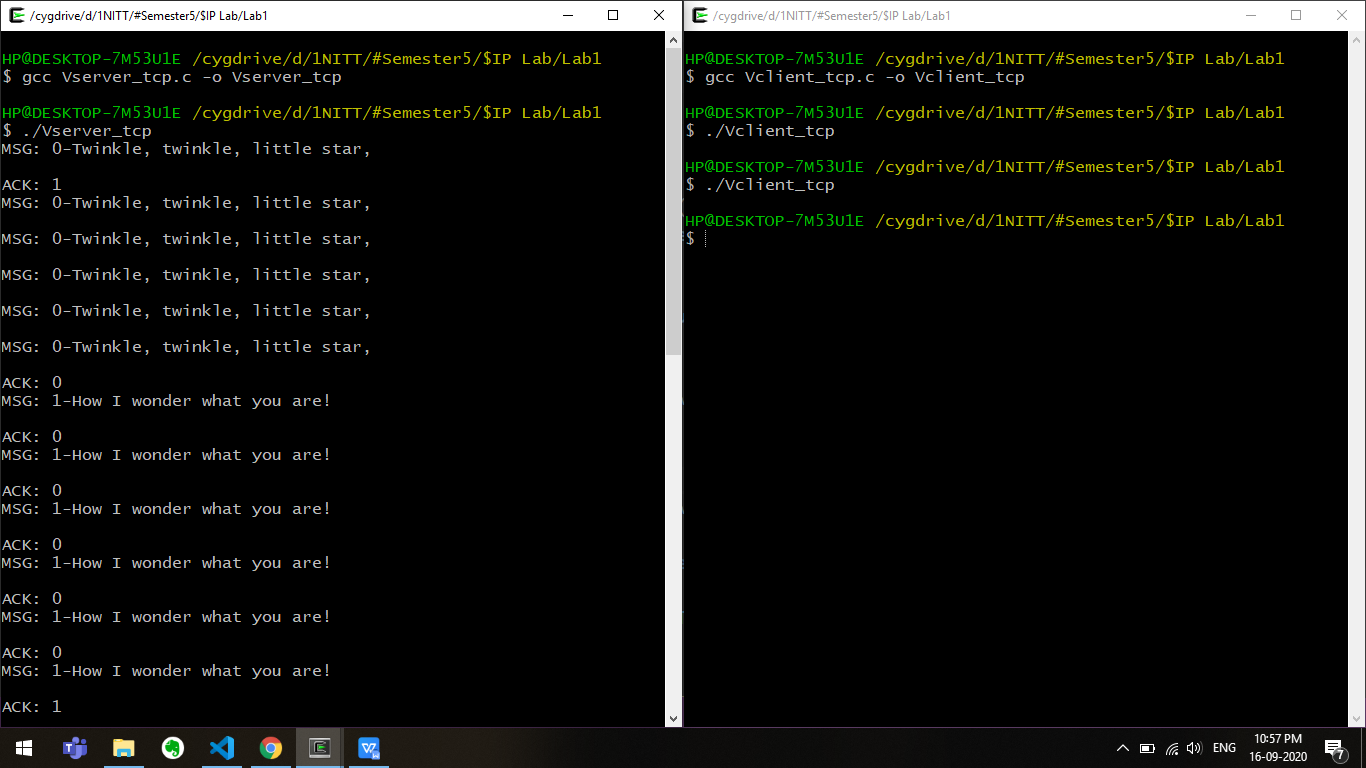
Damaged Message(change parity of random msg)



Lost Acknowledgement(don’t send random ack)



Delayed Acknowledgement(sleep() random ack)



**UDP Implementation**

Server\_udp.c

#include <unistd.h>

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <string.h>

#define PORT 8080

typedef enum{ DATA,ACK }   MSGKIND;

struct timeval timeout;

struct MESSAGE

{

    MSGKIND type;

    int seq;

    unsigned int len;

    char msg[100];

    int parity;

};

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

{

    timeout.tv\_sec = 1;

    timeout.tv\_usec = 0;

    int server\_fd;

    struct sockaddr\_in servaddr, cliaddr;

    char buffer[1024] = {0};

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

    {

        perror("socket failed");

        exit(EXIT\_FAILURE);

    }

    memset(&servaddr, 0, sizeof(servaddr));

    memset(&cliaddr, 0, sizeof(cliaddr));

    // if (setsockopt(server\_fd, SOL\_SOCKET, SO\_RCVTIMEO, (char \*)&timeout, sizeof(timeout)) < 0)

    // {

    //     perror("setsockopt");

    //     exit(EXIT\_FAILURE);

    // }

    servaddr.sin\_family = AF\_INET;

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

    servaddr.sin\_port = htons( PORT );

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

                                 sizeof(servaddr))<0)

    {

        perror("bind failed");

        exit(EXIT\_FAILURE);

    }

    int len, n;

    len = sizeof(cliaddr);

    // Declare the file pointer

    FILE \*filePointer ;

    // Declare the variable for the data to be read from file

    char dataToBeRead[100];

    filePointer = fopen("read.txt", "r") ;

    if ( filePointer == NULL )

    {

        perror( "file failed to open." ) ;

    }

    else

    {

        // Read the dataToBeRead from the file

        // using fgets() method

        int flag=1, s=0;

        recvfrom(server\_fd, (void \*)buffer, sizeof(struct MESSAGE), 0, ( struct sockaddr \*) &cliaddr, &len);

        while( fgets ( dataToBeRead, 100, filePointer ) != NULL )

        {

            while(1)

            {

                struct MESSAGE\* Message = (struct MESSAGE\*) malloc(sizeof(struct MESSAGE));

                struct MESSAGE\* Acknowledge = (struct MESSAGE\*) malloc(sizeof(struct MESSAGE));

                Message->type = DATA;

                Message->len = strlen(dataToBeRead);

                strcpy(Message->msg, dataToBeRead);

                Message->seq = s;

                int sum=0;

                for(int j=0;j<strlen(dataToBeRead);j++)

                {

                    sum+=dataToBeRead[j];

                }

                Message->parity =  sum%2;

                sendto(server\_fd, (void\*)Message, sizeof(struct MESSAGE), 0, ( struct sockaddr \*) &cliaddr, len);

                printf("MSG: %d-%s\n",s,dataToBeRead);

                if( recvfrom(server\_fd, (void \*)Acknowledge, sizeof(struct MESSAGE), 0, ( struct sockaddr \*) &cliaddr, &len)> 0)

                {

                    printf("ACK: %d\n",Acknowledge->seq);

                    if(Acknowledge->type == ACK && Acknowledge->seq == s)

                    {

                        sleep(1);

                        break;

                    }

                }

            }

            if(s==0) s=1;

            else s=0;

         }

        // Closing the file using fclose()

        fclose(filePointer) ;

    }

    close(server\_fd);

    return 0;

}

client\_udp.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#define PORT 8080

typedef enum{ DATA,ACK }   MSGKIND;

struct MESSAGE

{

    MSGKIND type;

    int seq;

    unsigned int len;

    char msg[100];

    int parity;

};

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

{

    int sock = 0, valread;

    struct sockaddr\_in serv\_addr;

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

    {

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

        return -1;

    }

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

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

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

    int len, n;

    len = sizeof(serv\_addr);

    // Declare the file pointer

    FILE \*filePointer ;

    // Get the data to be written in file

    char dataToBeWritten[100] ;

    filePointer = fopen("write.txt", "w") ;

    if ( filePointer == NULL )

    {

        printf( "file failed to open." ) ;

    }

    else

    {

            int s=1;

            char null[1]={'N'};

            int i=0;

            int count=0;

            while(i<5)

            {

                while(1)

                {

                    count++;

                    struct MESSAGE\* Message = (struct MESSAGE\*) malloc(sizeof(struct MESSAGE));

                    struct MESSAGE\* Acknowledge = (struct MESSAGE\*) malloc(sizeof(struct MESSAGE));

                    Acknowledge->type = ACK;

                    Acknowledge->len = 0;

                    strcpy(Acknowledge->msg, null);

                    Acknowledge->seq=s;

                    sendto(sock, (void\*)Acknowledge, sizeof(struct MESSAGE), 0, ( struct sockaddr \*) &serv\_addr, len);

                    if(recvfrom(sock, (void \*)Message, sizeof(struct MESSAGE), 0, (  struct sockaddr \*) &serv\_addr, &len)> 0)

                    {

                        if(Message->type == DATA && Message->seq + s == 1)

                        {

                            int sum=0;

                            for(int j=0;j<strlen(Message->msg);j++)

                            {

                                sum+=Message->msg[j];

                            }

                            if(Message->parity!=sum%2)

                                continue;

                            if(s==0) s=1;

                            else s=0;

                            strcpy(dataToBeWritten, Message->msg);

                            fputs(dataToBeWritten, filePointer) ;

                            break;

                        }

                    }

                }

            i++;

            }

        // Closing the file using fclose()

        fclose(filePointer) ;

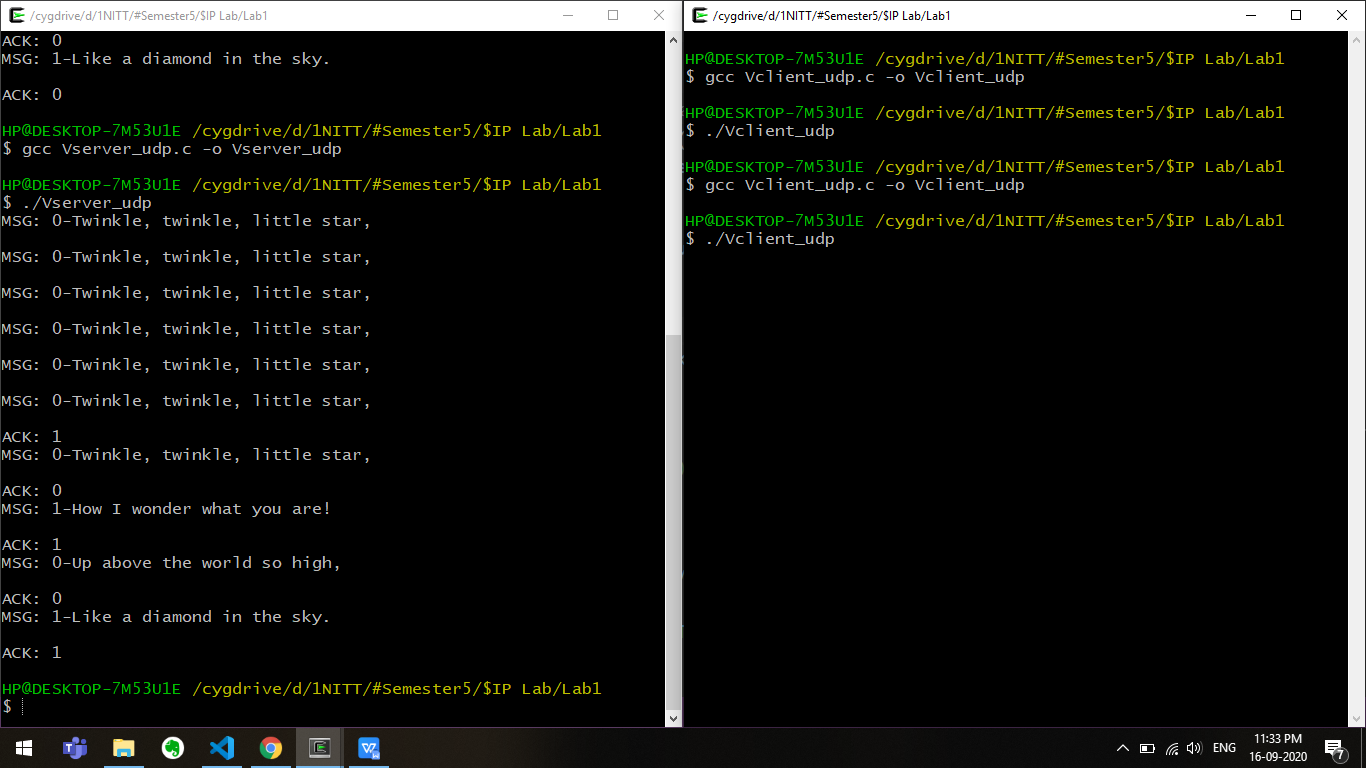
    }

    close(sock);

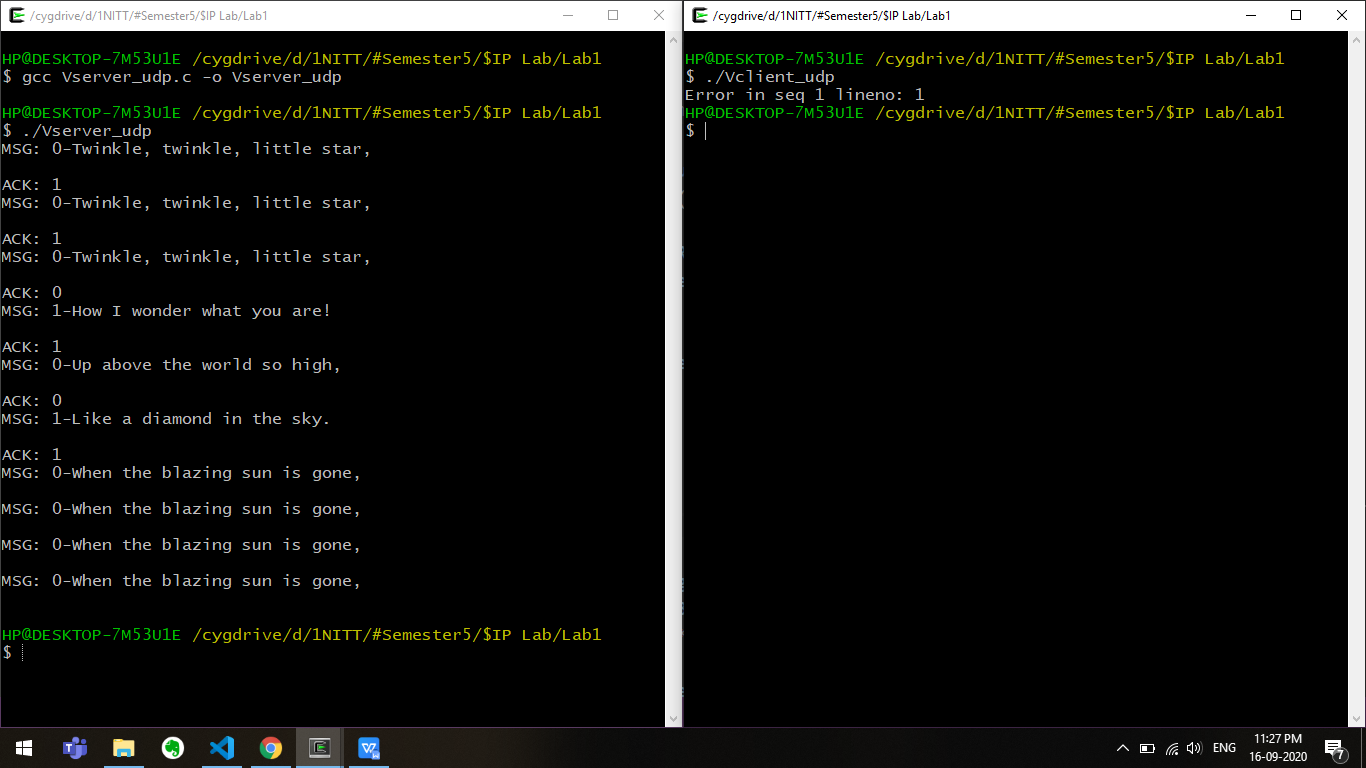
    return 0;

}

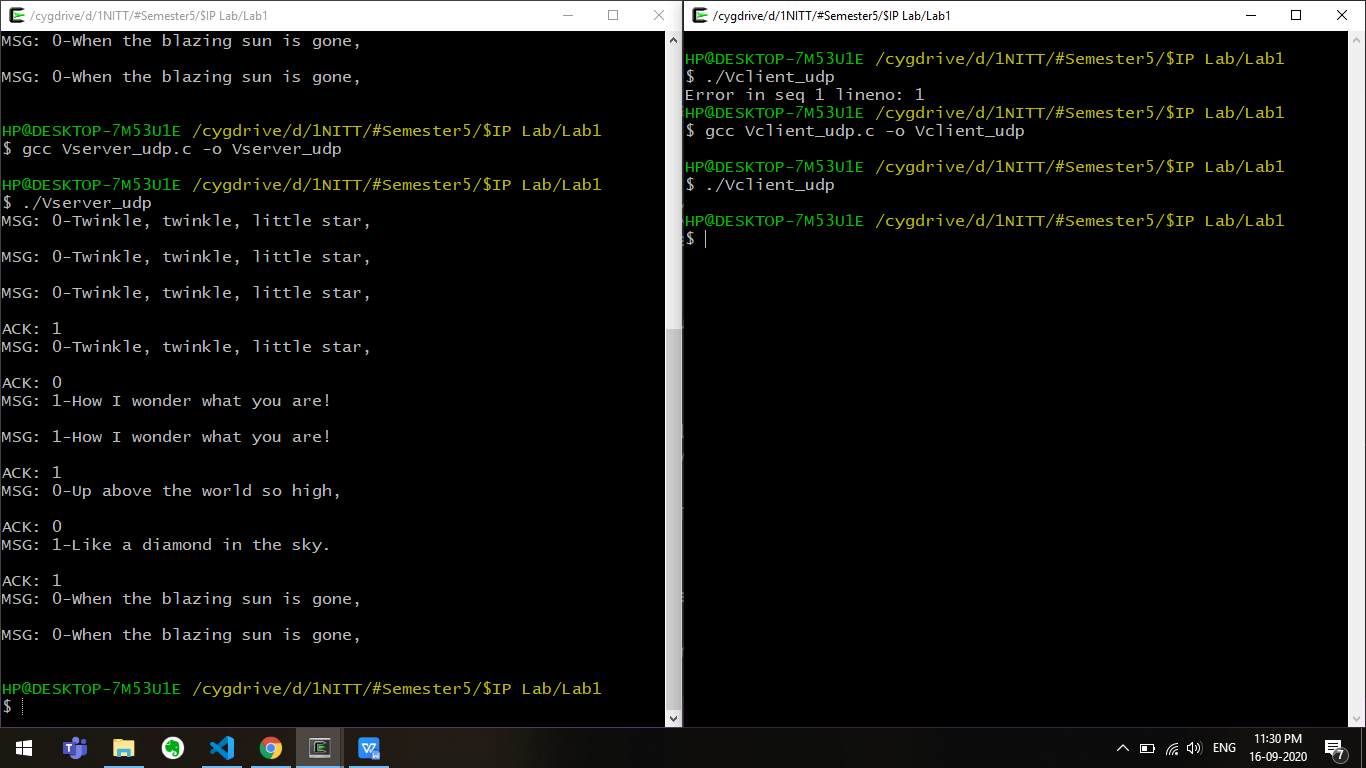
Normal Transmission



Damaged Message(change parity random msg)



Lost Acknowledgement(don’t send random ack)



Delayed Acknowledgement(sleep() random ack)

