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 MESSA
GE));
                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

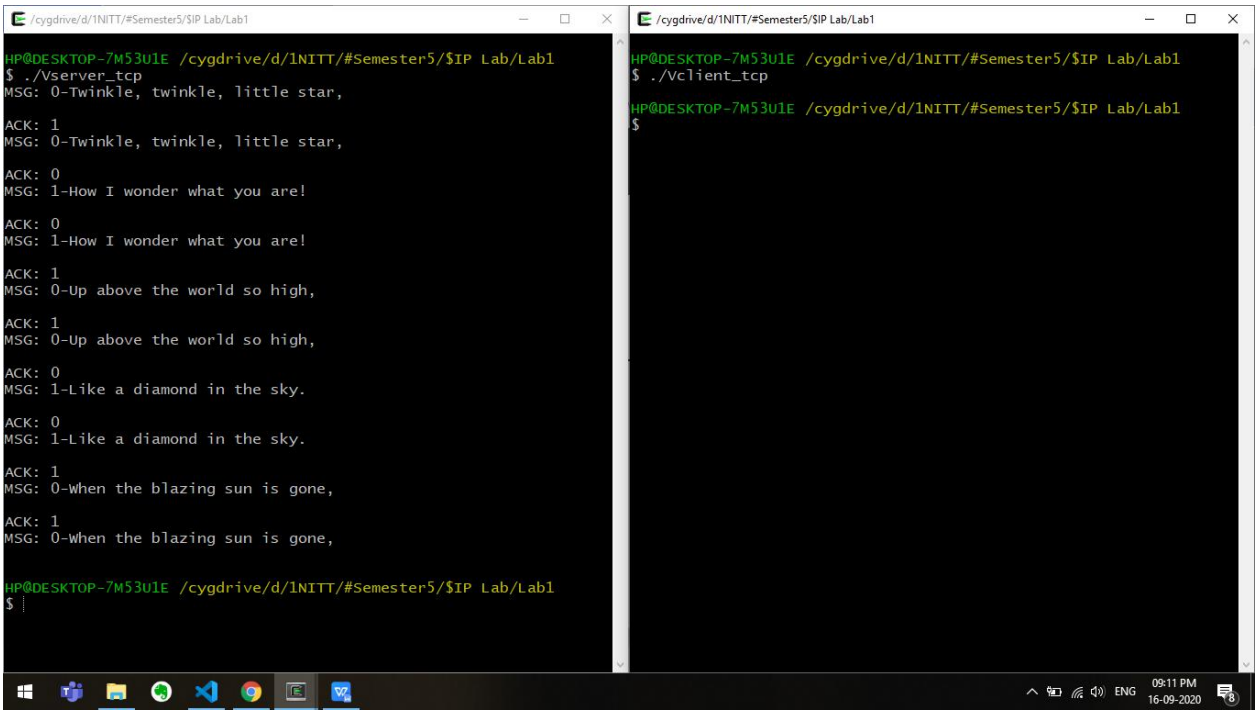
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#define PORT 8080
```

```
typedef enum{ DATA,ACK }   MSGKIND;

struct MESSAGE
{
    MSGKIND type;
    int seq;
    unsigned int len;
    char msg[100];
    int parity;
};
```

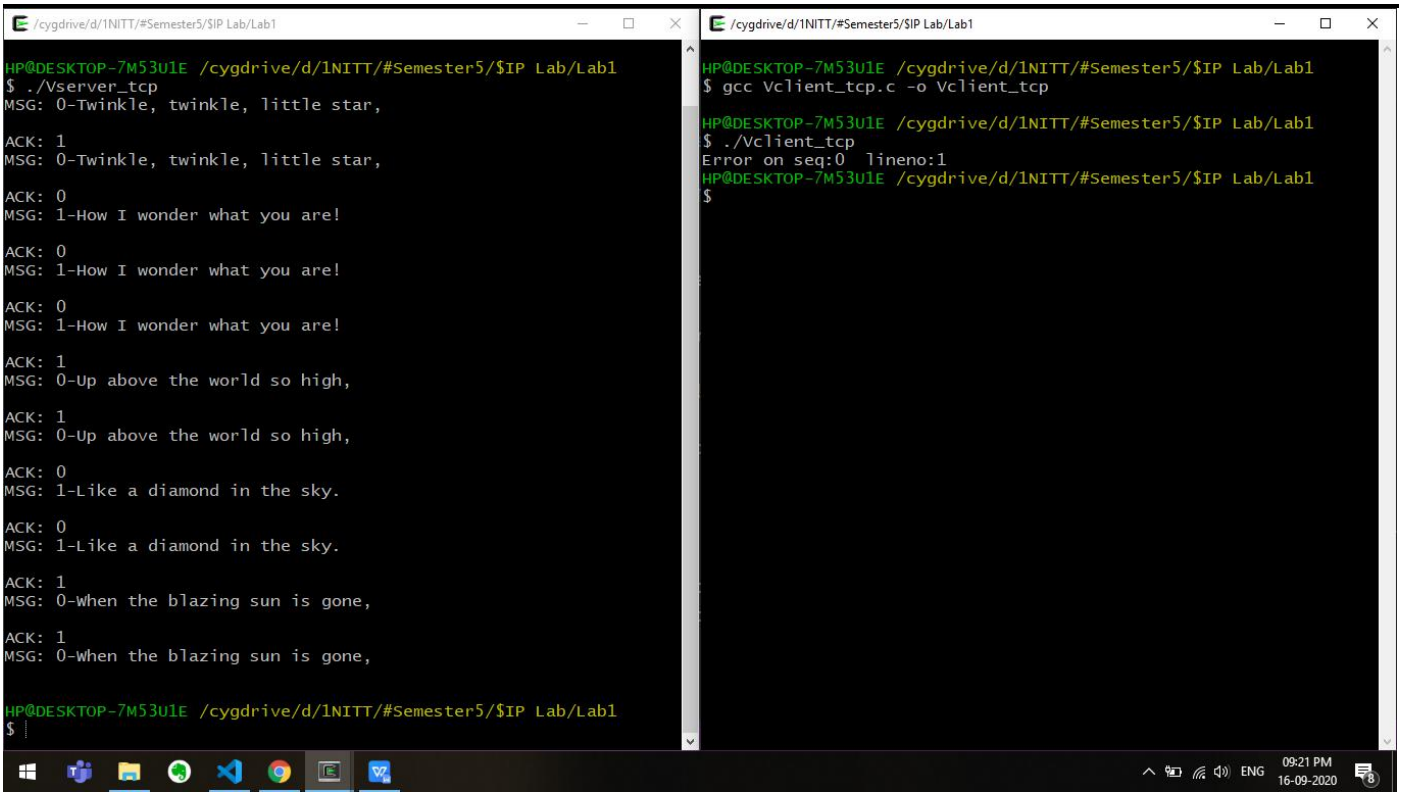
```
int main(int argc, char const *argv[])
{
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char buffer[1024] = {0};
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        return -1;
    }

    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 MESSA
GE));
                    struct MESSAGE* Acknowledge = (struct MESSAGE*) malloc(sizeof(struct M
ESSAGE));
                    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;
}
```

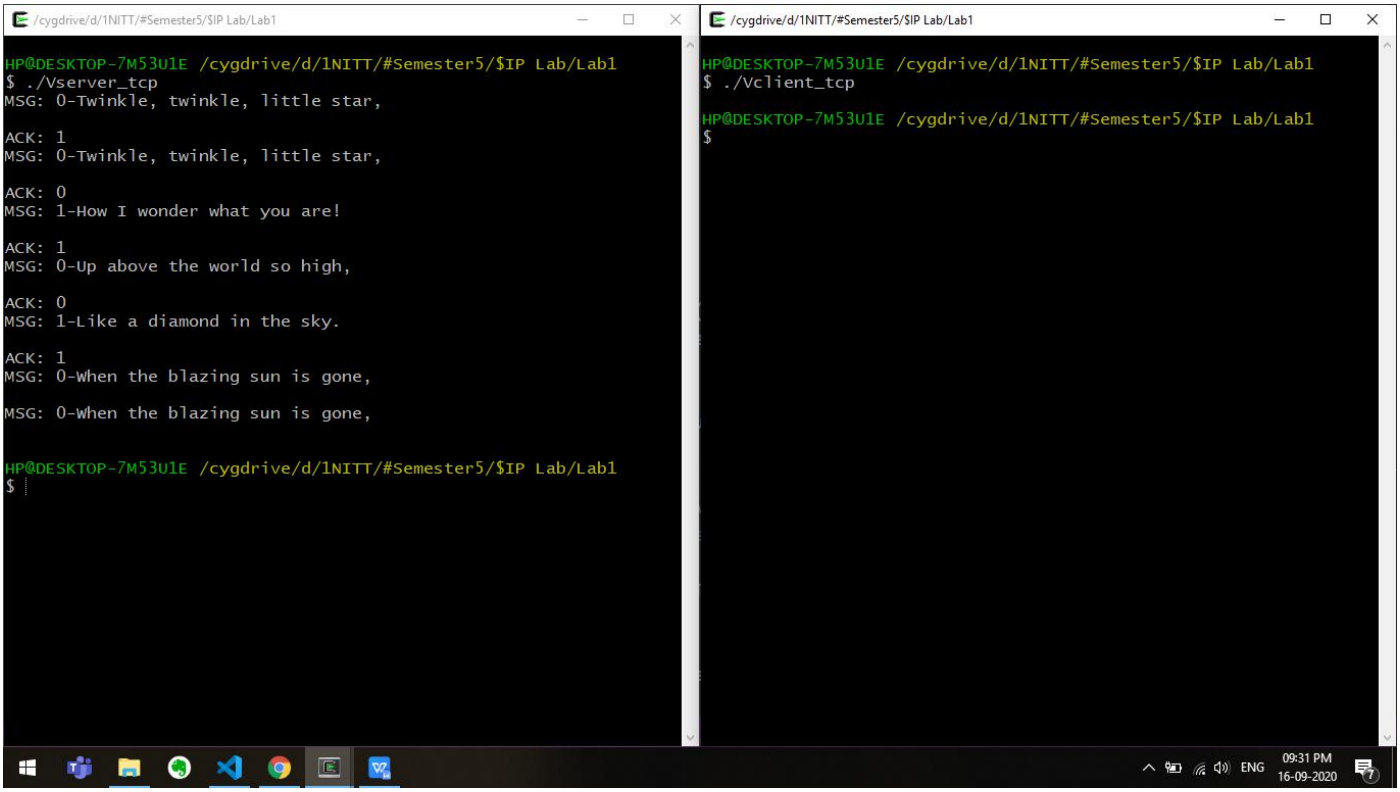
Normal Transmission



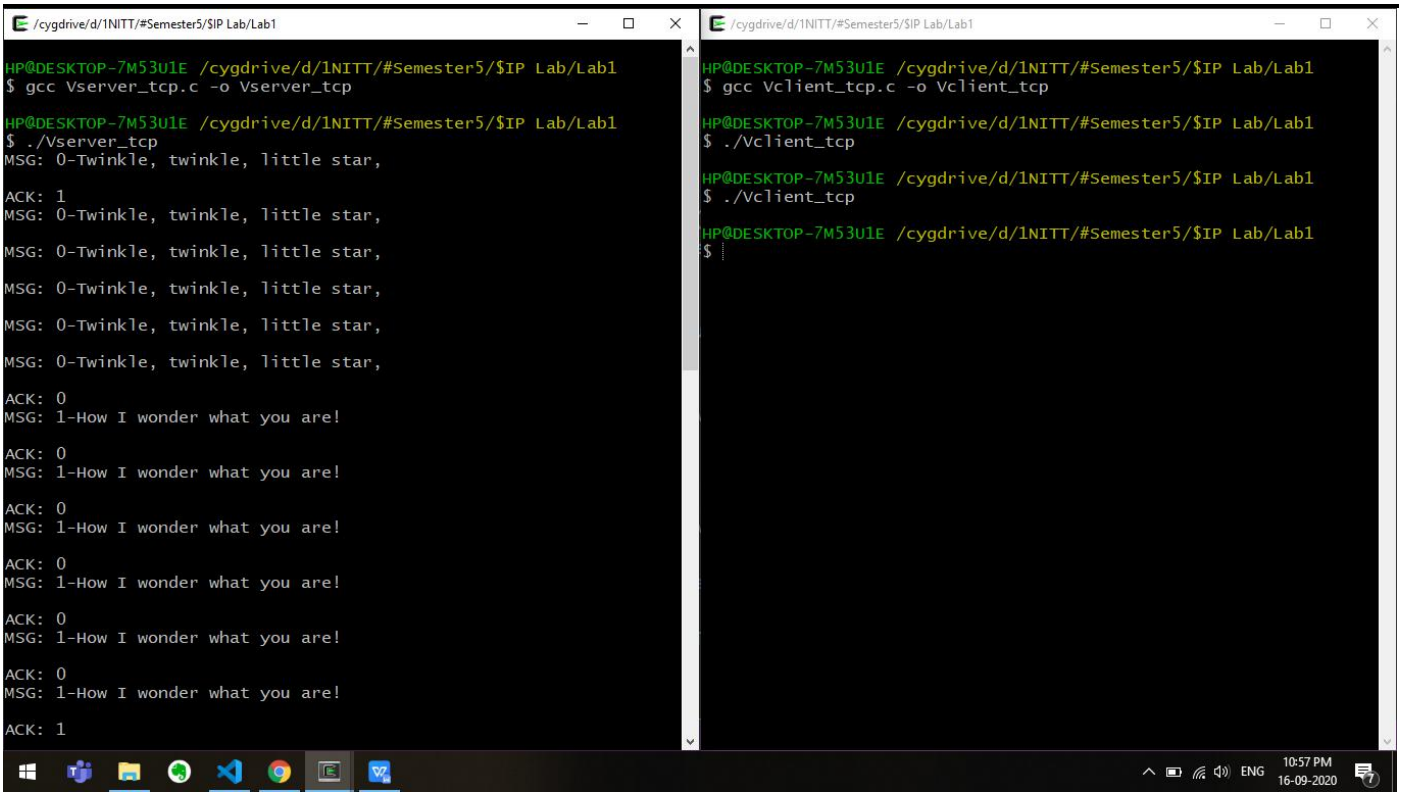
Damaged Message(change parity of random msg)



Lost Acknowledgement(don’t send random ack)



Delayed Acknowledgement(sleep() random ack)



UDP Implementation

Server_udp.c

```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;
};
```

```c
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 MESSA
GE));
                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 sock
addr *) &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;
};
```

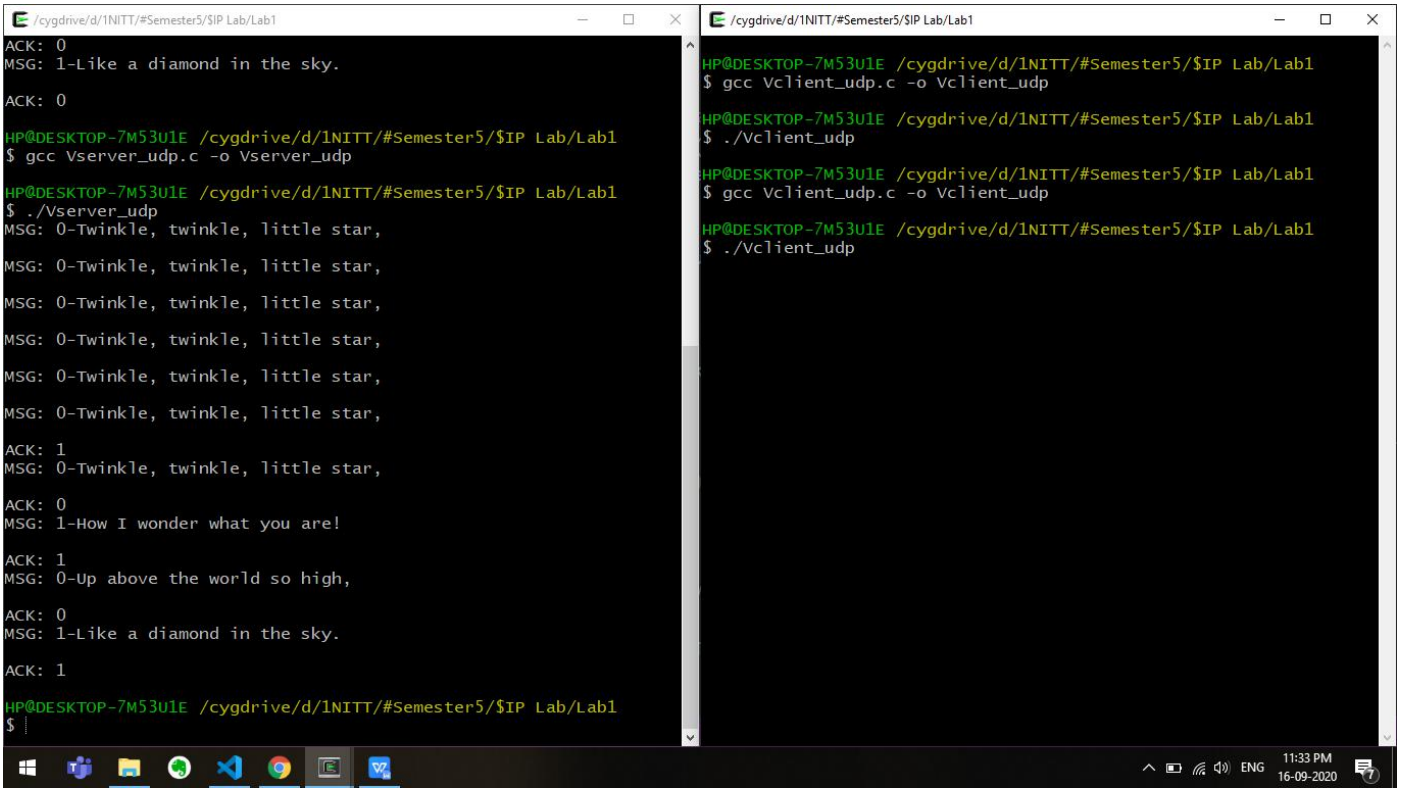
```
int main(int argc, char const *argv[])
{
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        return -1;
    }

    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);
```

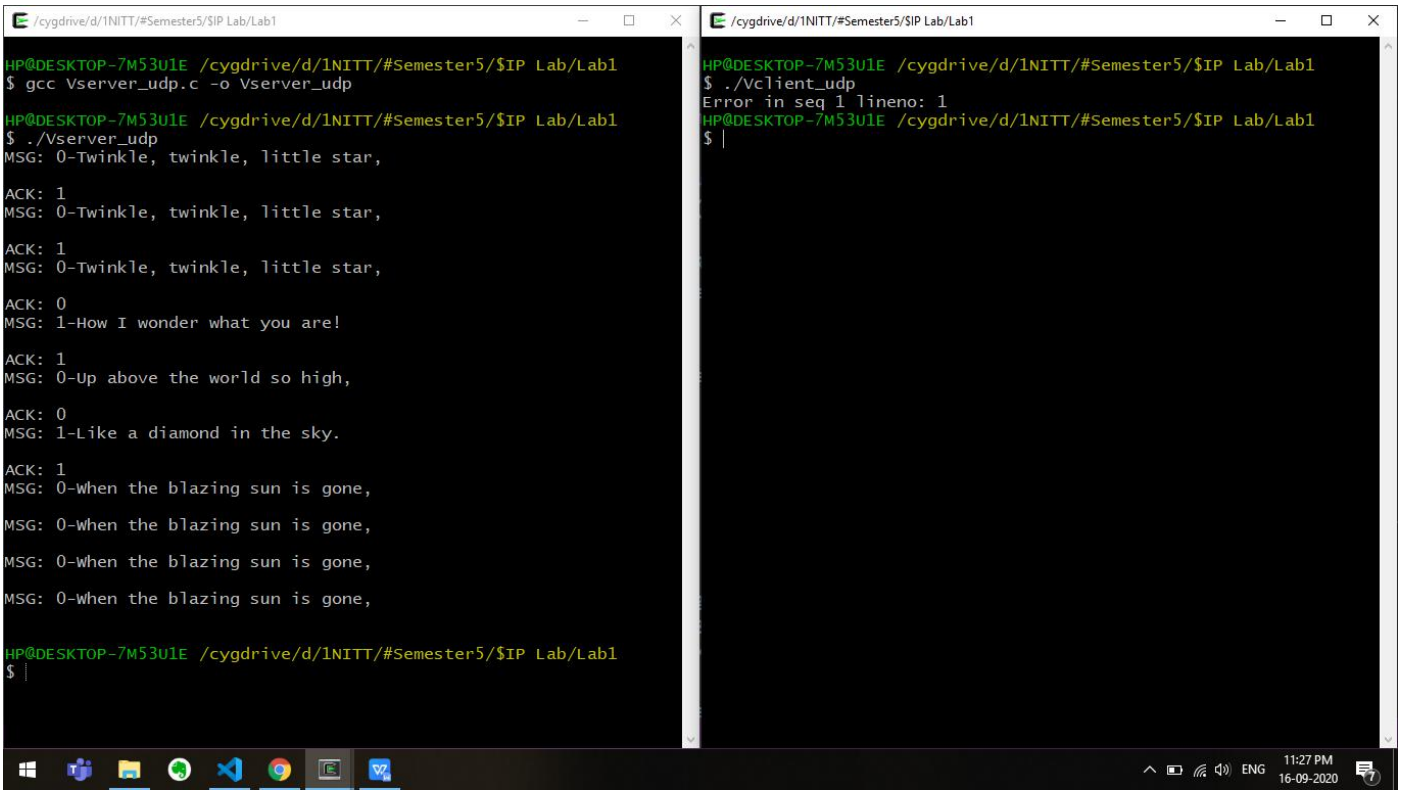
```
    // Declare the file pointer
    FILE *filePointer ;

    // 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 MESSA
GE));
                    struct MESSAGE* Acknowledge = (struct MESSAGE*) malloc(sizeof(struct M
ESSAGE));
                    Acknowledge->type = ACK;
                    Acknowledge->len = 0;
                    strcpy(Acknowledge->msg, null);
                    Acknowledge->seq=s;
                    sendto(sock, (void*)Acknowledge, sizeof(struct MESSAGE), 0, ( struct s
ockaddr *) &serv_addr, len);
                    if(recvfrom(sock, (void *)Message, sizeof(struct MESSAGE), 0, (  struc
t 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;
}
```

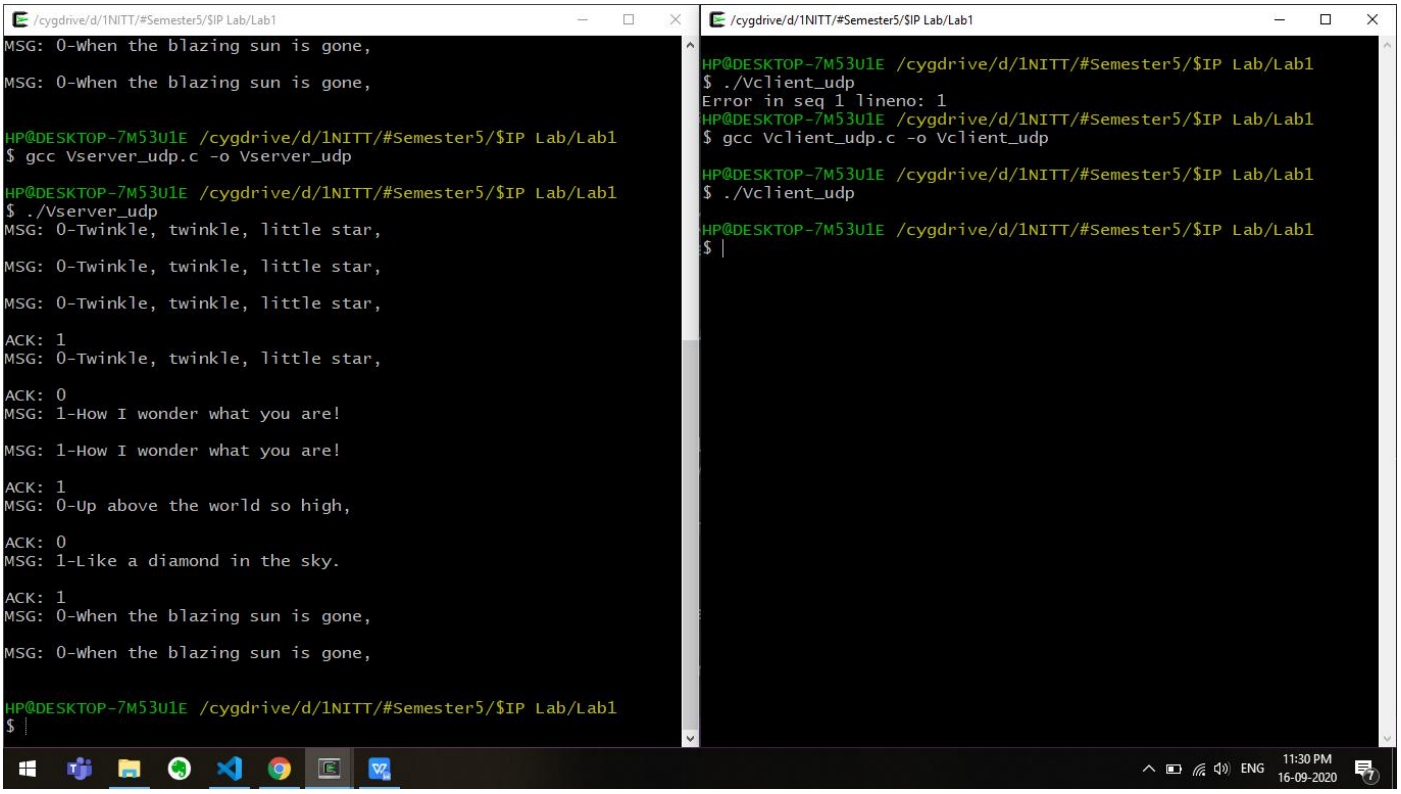
Normal Transmission



Damaged Message(change parity random msg)



Lost Acknowledgement(don’t send random ack)



Delayed Acknowledgement(sleep() random ack)