Group CN Project (Single Person):

> Mohammad Awais (BSCS-8-A) (CMS: 242554)

Reliable UDP Sending & Receiving a Video File

Specifications

This project requires you to write code for a sender and receivers that implements video file transfer over

UDP protocol using Linux/GNU C sockets. Sender is required to open a video file, read data chunks from file

and write UDP segments, send these segments on UDP. Receiver must be able to receive, reorder and write

data to a file at the receiving end.

Main Pillars of Project

➔ Sender:

The sender will read the file specified by the filename and transfer it using UDP sockets. On completing the

transfer, the sender should terminate and exit. The sender should bind to listen port to receive

acknowledgments and other signaling from the receiver. You should only use a single UDP socket for both

sending and receiving data. Note that although UDP will allow you to send large packets using IP

fragmentation, but make sure that you restrict your packets to 500 bytes (in payload).

➔ Receiver:

The receiver will need to bind to the UDP port specified on the command line and receive a file from the

sender sent over the port. The file should be saved to a different filename. Note that you should make sure in

your testing that the filename used by the receiver is not the same as the one used by the sender. The receiver

should exit once the transfer is complete.

➔ Implementing reliability in UDP

You will be required to implement following to make UDP reliable:

a) Sequence numbers

b) Retransmission (selective repeat)

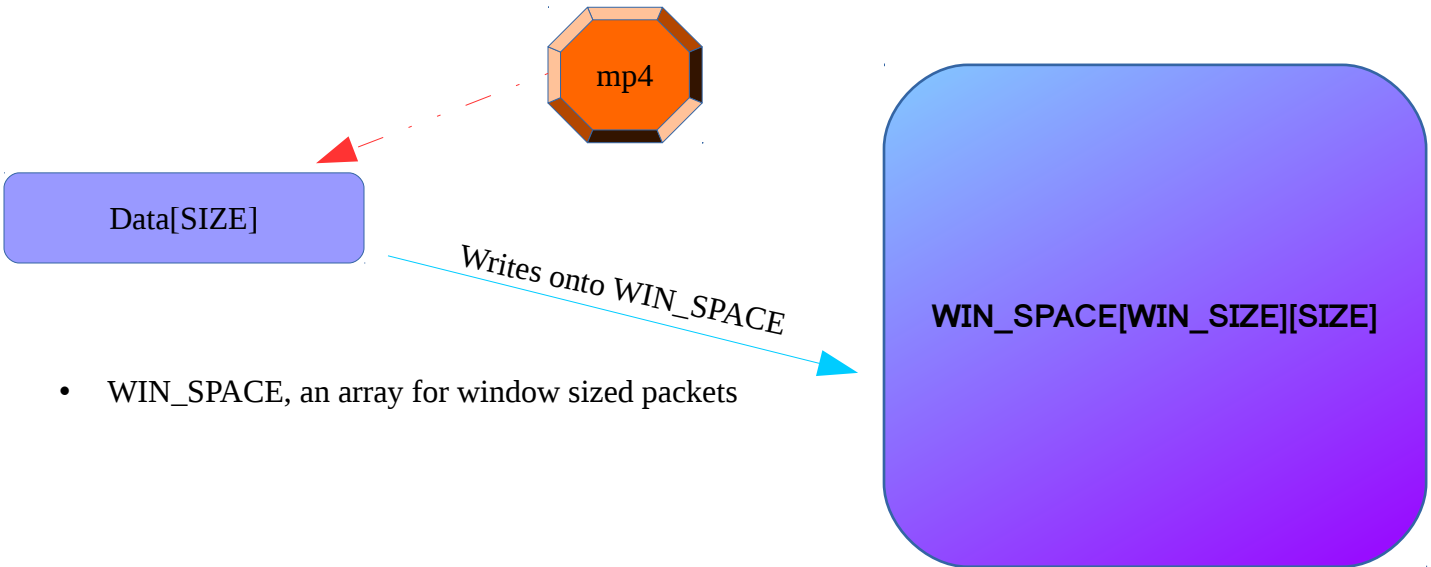
c) Window size of 5-10 UDP segments (stop n wait)

d) Re ordering on receiver side

Basic Concept:-

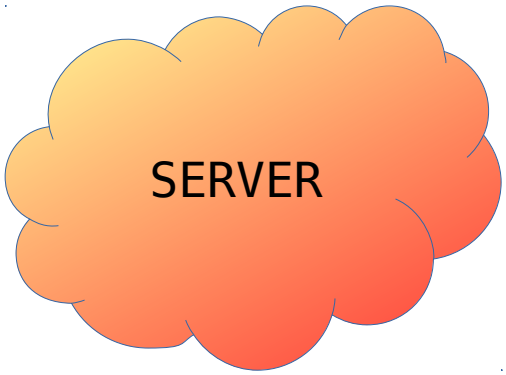
➔ Client's Side

- SIZE → Packet Size
- WIN_SIZE → Window Size (5 or 10)
- "data" array which serves as buffer to read from File



- WIN_SPACE, an array for window sized packets

- It collects Packets for current Window Phase

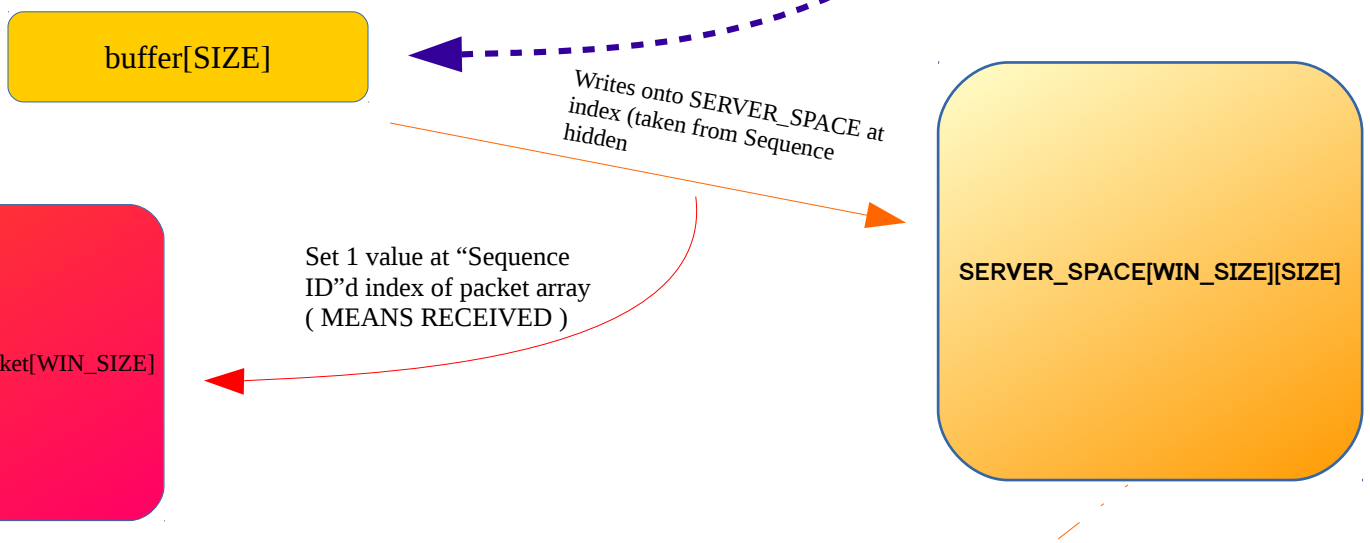


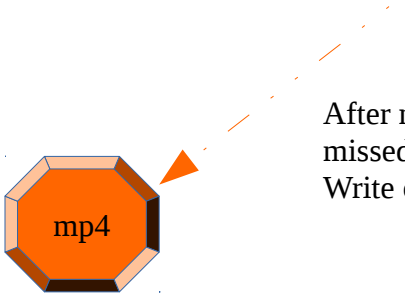
Sending WIN_SIZE-d Packets from Window Space once a loop to Server (along wity Sequence in 0 index

➔ Server's Side

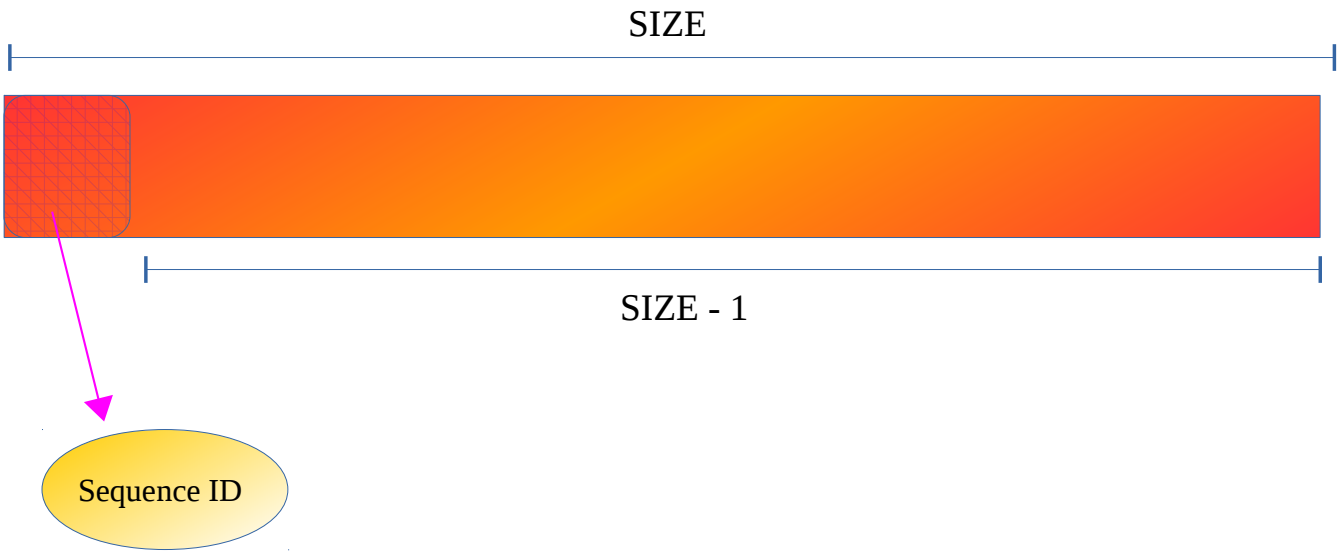
- Client sends packet with embedded packet sequence ID at 0 index
- buffer reads it and writes on Server_Space[SQ ID]
- where SQ ID is at 0 index of buffer

Receiving Packets from Client along with Sequence ID at index 0 of packet





➔ **Packet sent**



Functionalities

- Is able to send any type of file (text file, mp3 file, mp4 file) but was made for purpose of sending video files.
- Acknowledgements gives Both Server and Client programs to get themselves ordered for proper communication.
- Sequence Ids help in arranging it Server_Space array which then write data in ascending order keeping overall data in file in order.
- A rechecking phase for each window cycle is there which makes sure before writing onto file whether all packets sent have been received or not.
- packet[WIN_SIZE] has all enteries set to 0. Whenever a Packet is received with SQ_ID, packet[SQ_ID-1] is set to 1 , which means the sequence associated packet has arrived.

Procedure

- First UDP socket is created for both server and client side.
- Server's side IPv4 (local address) is entered in serverUDP.c program along with the port used.
- Client's side has Server's Side Public IP written along with the Port used to send data from in clientUDP.
- Server's side Port forwarding on the subjected IP and port is established and it's run and it starts receiving.
- Client Side run clientUDP.c program.
- First of all a simple Handshake protocol is established to make sure the communication is done by both side.
- A Loop of communication starts:-
    - Then client reads SIZE by SIZE data from mp4 file (here we'll use "Revolution.mp4").
    - Then it saves it in its WIN_SPACE[WIN_SIZE][SIZE] 2D array which works as an Array of Strings here. It keeps storing till WIN_SIZE.

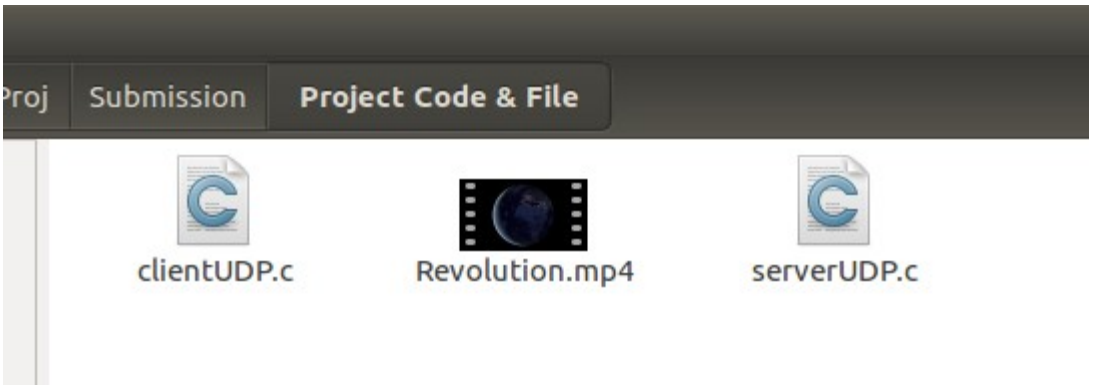
    NOTE: It's to be noted that SQ_ID →{ 0,1,2,….,WIN_SIZE}

    - Then it sends all of its packets from WIN_SPACE to server.
    - Server also on its end receives the packets in buffer[SIZE] and then later on stores data part at SERVER_SPACE[SQ_ID-1]. Also sets packet[SQ_ID-1] = 1.
    - Client then sends "SENT" message as all Window packets sent which server receives and and then asks for "Claim" (total packets sent from WIN_SPACE) from client.
    - Client receives its request and sends another Message but with Packet_count embedded at Index 0.
    - Server receives the message and decipher the Packet_count, stores in claim variable.
    - It checks if Packet it received and the claim variable are same or not:
        - If not then it sends "!RECEIVED" message at which Client ready itself to receive index
        - Server checks packet[WIN_SIZE] to see if under "packet_count" any value not set to 1, Then sends the index of that value through embedding it into a message
        - Which after receiving sends the particular packet along with SENT Message.
        - Server receives and sets value 1 at packet[SQ_ID-1] and saves message at SERVER_SIZE

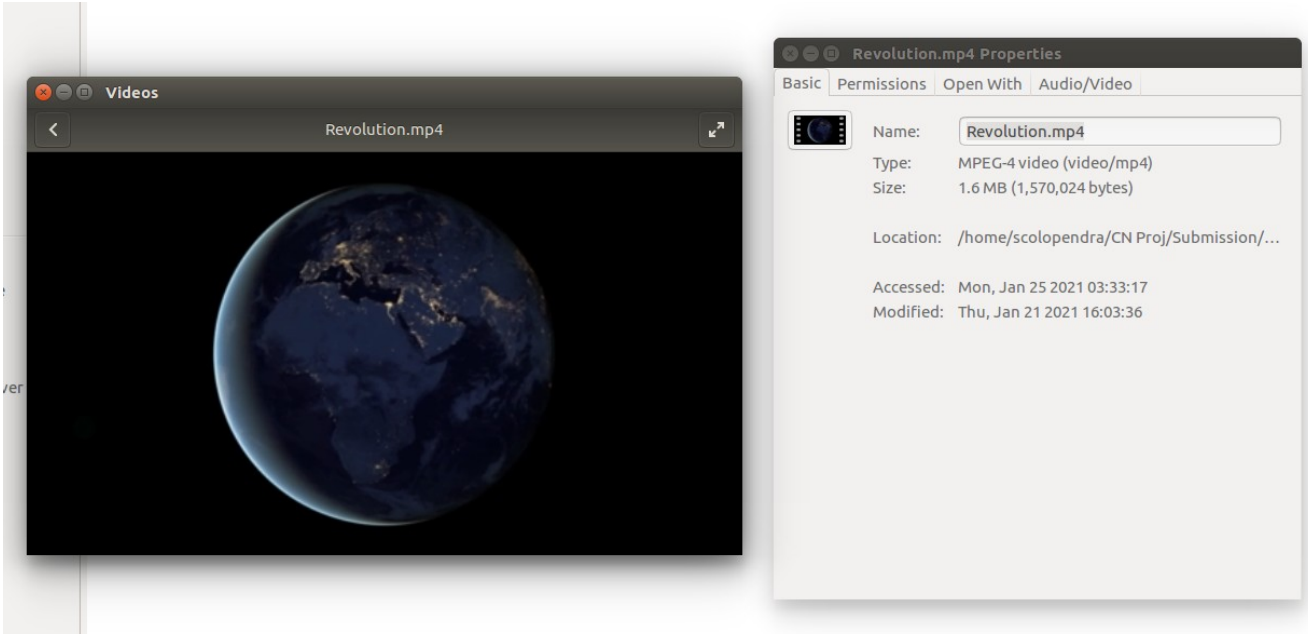
    - This continues till all packets under Window phase are received and then finally “RECEIVED” message is sent and which client then moves on forward and refreshes for next phase.
    - Client then writes the SERVER_SIDE[WIN_SIZE] ascending order wise onto a newly created file and then moves onto next loop.
- If End of file reached, client sends END-OF-FILE message to server and exits.
- Server After receiving that starts writing the current window-received packets onto file and then exits.

Screenshots:-

➔ Folder:-



➔ Revolution.mp4



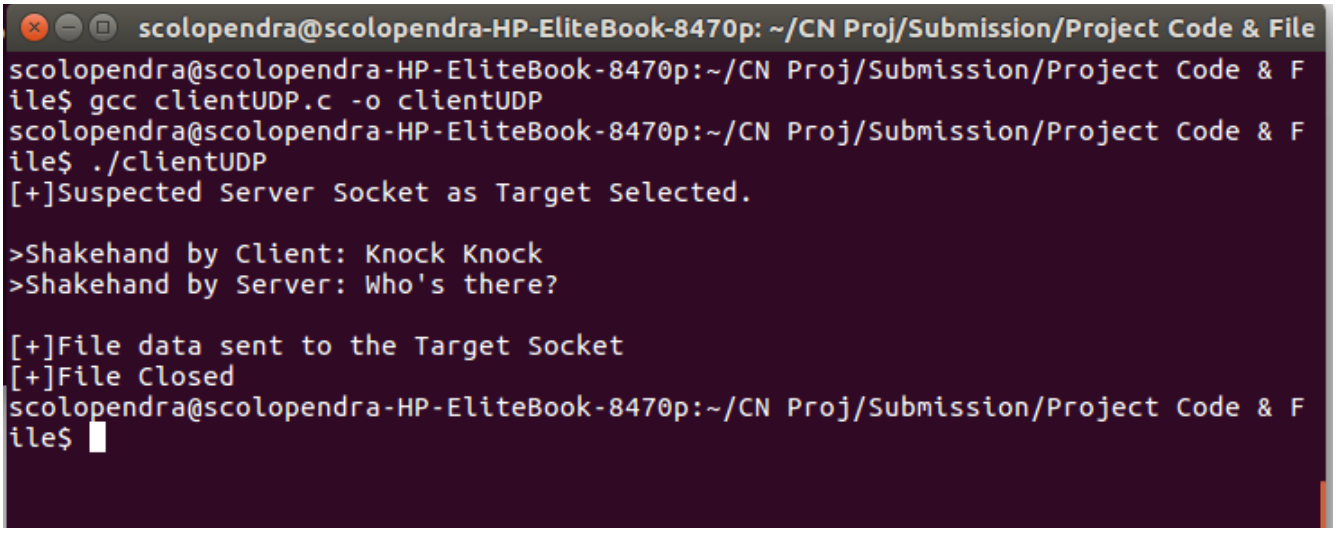
➔ ./serverUDP

Waiting to Recieve...

```
scolopendra@scolopendra-HP-EliteBook-8470p: ~/CN Proj/Submission/Project Code & File
scolopendra@scolopendra-HP-EliteBook-8470p:~/CN Proj/Submission/Project Code & F
ile$ gcc serverUDP.c -o serverUDP
scolopendra@scolopendra-HP-EliteBook-8470p:~/CN Proj/Submission/Project Code & F
ile$ ./serverUDP
[+]Server socket created.
[+]Binding successful.
[+]Receiving Phase Start...
```

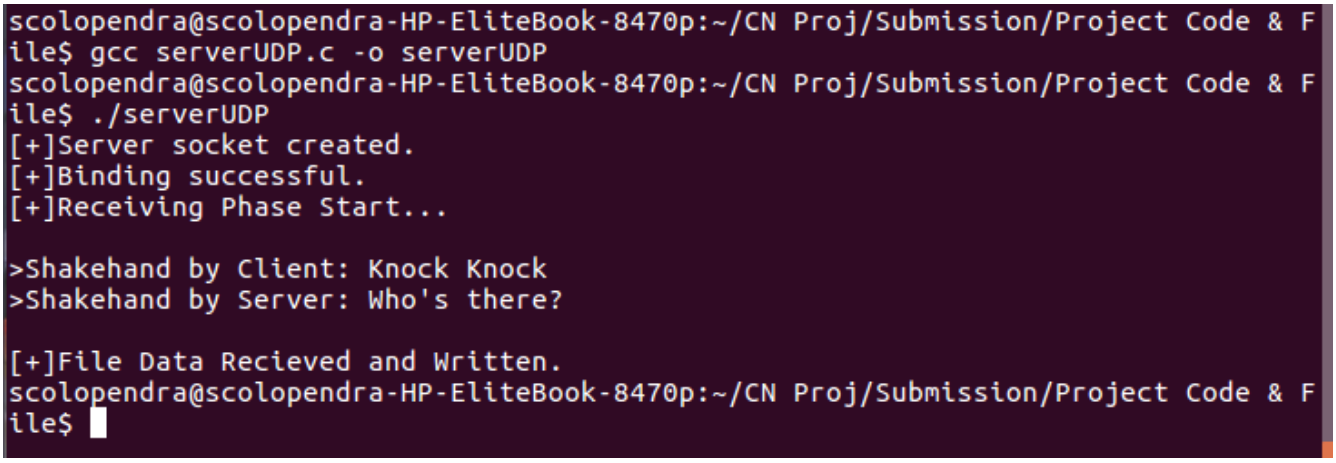
➔ ./clientUDP

Connected & Sent!

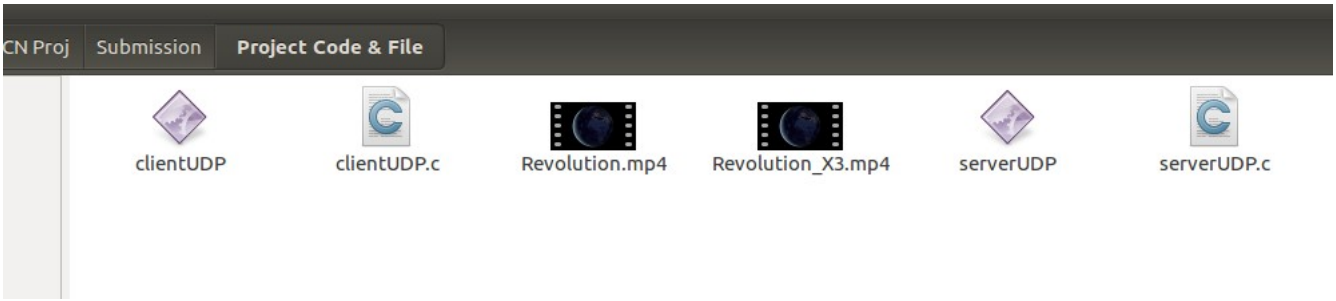


➔ ./serverUDP

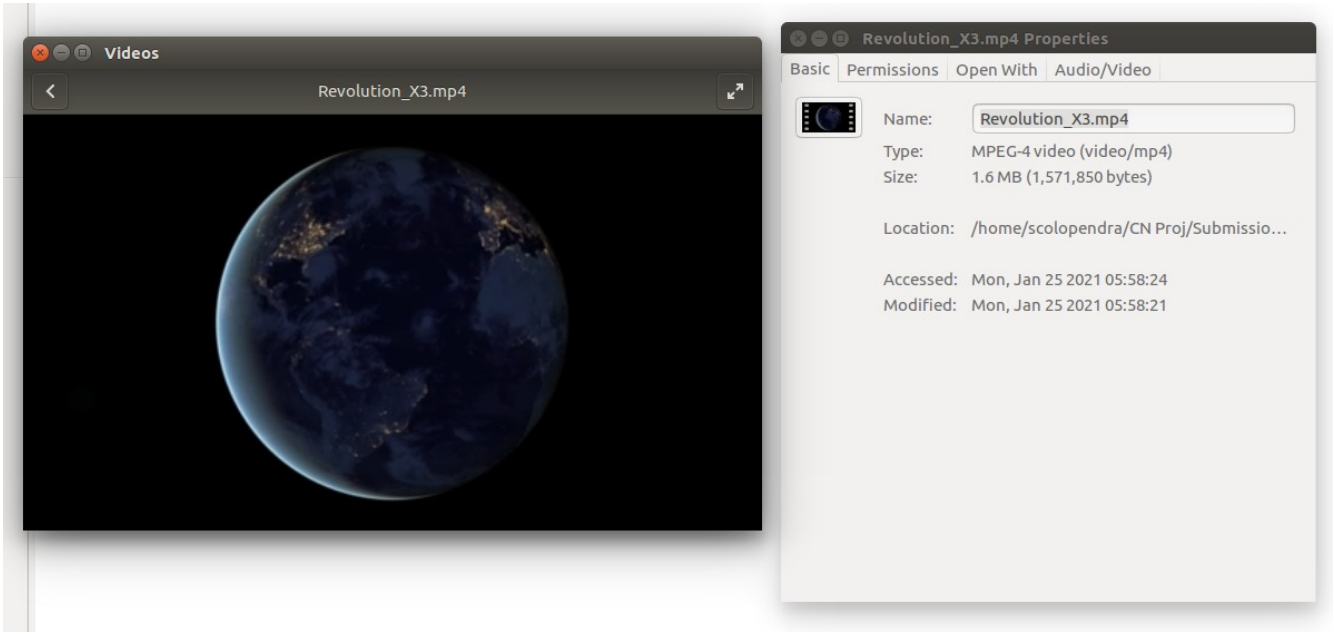
After Receiving



➔ Folder



➔ Revolution_X3.mp4



Program Code:-

➔ serverUDP.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>
#define SIZE 500
#define WIN_SIZE 5

char SERVER_SPACE[WIN_SIZE][SIZE] = {0};
```

```
char *EFLAG = "=====END=====\0";

int main()
{

	char *ip = "192.168.10.11"; // IPv4 of Server Machine
	int port = 25000;
	int e;

	int sockfd;
	struct sockaddr_in server_addr, new_addr;
	socklen_t addr_size;
	char buffer[SIZE];

	sockfd = socket(AF_INET, SOCK_DGRAM, 0);
	if (sockfd < 0)
	{
		perror("[-]Error in socket.");
		exit(1);
	}
	printf("[+]Server socket created.\n");

	server_addr.sin_family = AF_INET;
	server_addr.sin_port = htons(port);
	server_addr.sin_addr.s_addr = inet_addr(ip);


	e = bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));
	if (sockfd < 0)
	{
		perror("[-]Error in socket.");
		exit(1);
	}
	printf("[+]Binding successful.\n");

	// FILE HANDLING

	int n;
	FILE *fp;
	char *filename = "Revolution_X3.mp4";
	addr_size = sizeof(server_addr);
```

```
	fp = fopen(filename, "wb");

	if (fp == NULL)
	{
		perror("[-]Error in Creating File.");
		exit(1);
	}

	printf("[+]Receiving Phase Start...\n");

	// UDP Reliability
	int ack; // not used yet

	int packet[WIN_SIZE] = {0};
	int packet_counter = 0;
	char sent_message[16] = "======SENT=====\0";
	char ack_message[16] = {0};
	int number;
	int exit = 0;
	int claim = WIN_SIZE;

	ack = recvfrom(sockfd, ack_message, sizeof(ack_message), 0, (struct sockaddr
*)&server_addr, &addr_size);
	printf("\n>Shakehand by Client: %s", ack_message);

	sendto(sockfd, "Who's there?", sizeof("Who's there?"), 0, (struct sockaddr
*)&server_addr, sizeof(server_addr));
	printf("\n>Shakehand by Server: %s\n\n", "Who's there?");

	while (1)
	{

		if (exit == 1)
		{ // when End Flag come in last Window space
			break;
		}

		// Receives packet from Client
		n = recvfrom(sockfd, buffer, SIZE, 0, (struct sockaddr *)&server_addr,
&addr_size);
```

```
packet_counter = 0; // For Counting Packets received in particular Window Phase

if (n)
{

        while (strcmp(buffer, sent_message)) //while received packet is not a "Sent
Message"
        { // not same
                if (!(strcmp(buffer, EFLAG)))
                { //Exit on receiving User EOF Flag
                        exit = 1;

                        break;
                }

                number = buffer[0] - '0'; // Separating Sequence no. of Packet
                number = number - 1;

                number++;

                packet[number - 1] = 1; // Setting the Sequence numbered position in
a specific array as 1 (means Received)

                bzero(SERVER_SPACE[number-1],SIZE); // Saving the received
Packet in the Server's Window_Space Array after clearing that space
                for (int i = 1; i < SIZE; i++)
                {

                        SERVER_SPACE[number - 1][i - 1] = buffer[i];
                }

                packet_counter++;

                bzero(buffer, SIZE);

                // Receive Next Packet ( Can be acknowledgement message or Data
packet)
                n = recvfrom(sockfd, buffer, SIZE, 0, (struct sockaddr
*)&server_addr, &addr_size);
```

```
                }

                // Receive total packets sent in window phase from Client if not EOF case by asking
                if (exit != 1)
                {
                        sendto(sockfd, "Claim?\0", sizeof("Claim?\0"), 0, (struct sockaddr *)&server_addr, sizeof(server_addr));
                        bzero(ack_message, SIZE);
                        recvfrom(sockfd, ack_message, 1, 0, (struct sockaddr *)&server_addr, &addr_size);
                        claim = ack_message[0] - '0';


                }
        }

        if (exit == 1) //  *** Iconic case of Not checking data in last Window Phase and just writing and exiting
        {
                for (int i = 0; i < claim; i++)
                {
                        fwrite(SERVER_SPACE[i], sizeof(char), SIZE-1, fp);

                }

                break;
        }

        // If Packet received in WIndow Phase not equal to Packets sent from Client in Window Phase, Sending missing IDs and re-receiving
        if (packet_counter != claim)
        {


                while (1)
                {
                        strcpy(ack_message, "===!RECEIVED===\0");
                        sendto(sockfd, ack_message, sizeof(ack_message), 0, (struct sockaddr *)&server_addr, sizeof(server_addr));
```

```
                        // Checking missing sequence ID marked and sending it, Receiving its relavent data
                        for (int i = 0; i < claim; i++)
                        {

                                // On Missing Packet Sequence ID, Send ID and receive packet relevant to it
                                if (packet[i] == 0)
                                {
                                        snprintf(ack_message, 16, "%d======SUS======", i + 1); // puts string into buffer
                                        sendto(sockfd, ack_message, sizeof(ack_message), 0, (struct sockaddr *)&server_addr, sizeof(server_addr));
                                        bzero(buffer, SIZE);

                                        // Relevant Packet particularly received
                                        n = recvfrom(sockfd, buffer, SIZE, 0, (struct sockaddr *)&server_addr, &addr_size);

                                        number = buffer[0] - '0'; // Converted it's ID

                                        packet[number] = 1; // Setting Packet ID based entry to 1 ( received)

                                        //Saving the Received Packet in Server's Window_Space
                                        for (int i = 1; i < n; i++)
                                        {
                                                SERVER_SPACE[number - 1][i] = buffer[i];
                                        }
                                        packet_counter++;
                                }
                        }

                        // Sending Message to clear that all Packets in Window Phase are checked in current Loop Cycle
                        sendto(sockfd, sent_message, sizeof(sent_message), 0, (struct sockaddr *)&server_addr, sizeof(server_addr));

                        // Client also acknowledges the Current Loop checking
```

```
                        ack = recvfrom(sockfd, ack_message, sizeof(ack_message), 0, (struct
sockaddr *)&server_addr, &addr_size);


                        if ((!strcmp(ack_message, sent_message)) && (packet_counter ==
claim)) //If Client acknowledges and Packet counts is same as claim now then send
"RECEIVED" acknowledgement
                        {

                                strcpy(ack_message, "====RECEIVED===\0");
                                sendto(sockfd, ack_message, sizeof(ack_message), 0, (struct
sockaddr *)&server_addr, sizeof(server_addr));
                        }


                }
        }
        else  // IF NO PACKET MISSED, SEND RECEIVE ACKNOWLEDGEMENT
        {

                strcpy(ack_message, "====RECEIVED===\0");
                sendto(sockfd, ack_message, sizeof(ack_message), 0, (struct sockaddr
*)&server_addr, sizeof(server_addr));
        }

        // Write Window's Space packets into New File Created
        for (int i = 0; i < packet_counter; i++)
        {
                fwrite(SERVER_SPACE[i], sizeof(char), SIZE-1, fp);

        }

        bzero(buffer, SIZE);
        bzero(ack_message, sizeof(ack_message));
        for (int i = 0; i < WIN_SIZE; i++)
        {
                bzero(SERVER_SPACE[i], SIZE);
        }
    }

    fclose(fp);
```

```
        printf("[+]File Data Recieved and Written.\n");
}
```

➔ clientUDP.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>
#define SIZE 500
#define WIN_SIZE 5

char* EFLAG = "=====END=====\0";

int main(){

char *ip = "39.45.57.58"; // Public IP of Server's Router ( ALso enable port forwarding )
int port = 25000;
int e;

int sockfd;
struct sockaddr_in server_addr;
FILE *fp;
char *filename = "Revolution.mp4";
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
if (sockfd < 0){
        perror("[-]Error in socket.");
        exit(1);
}

printf("[+]Suspected Server Socket as Target Selected.\n");


server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(port);
server_addr.sin_addr.s_addr = inet_addr(ip);

// FILE HANDLING
```

```
socklen_t addr_size = sizeof(server_addr);
fp = fopen(filename, "rb");
if (fp == NULL){
perror("[-]Error in Reading File.");
exit(1);
}

char data[SIZE] = {0};

// UDP Reliability
int ack;
char WIN_SPACE[WIN_SIZE][SIZE]={0};
int packet_counter=0;
char sent_message[16] = "======SENT=====\0";
char ack_message[16]={0};

//Confirming connection before starting
printf("\n>Shakehand by Client: Knock Knock");
        sendto(sockfd, "Knock Knock", sizeof("Knock Knock"), 0, (struct sockaddr
*)&server_addr, sizeof(server_addr));

        ack = recvfrom(sockfd, ack_message, sizeof(ack_message), 0, (struct sockaddr
*)&server_addr, &addr_size);
printf("\n>Shakehand by Server: %s\n\n", ack_message);

// Send till END-OF-FILE reached
while(!feof(fp)){ // Overall Sending
        packet_counter = 0;

        // Reading file till 'WIN_SIZE' packets collected in Window of fixed size if data in File
        while(!feof(fp) && packet_counter<WIN_SIZE){ // A window Size

                fread(data,sizeof(char), sizeof(data)-1, fp);

                bzero(WIN_SPACE[packet_counter],SIZE);
                WIN_SPACE[packet_counter][0]=('0'+packet_counter)+1;
                for(int i=1;i<SIZE;i++)
                        {
                        WIN_SPACE[packet_counter][i]=data[i-1];
                        }
```

```
                packet_counter++;
                bzero(data,SIZE);
        }

        // Send packets one at a time from WIN_SPACE Array
        for(int i=0;i<packet_counter;i++){

                sendto(sockfd, WIN_SPACE[i], SIZE, 0, (struct sockaddr *)&server_addr,
sizeof(server_addr));


                }

        // if END-OF-File, Send End Flag
        if(feof(fp)){
                sendto(sockfd, EFLAG, strlen(EFLAG), 0, (struct sockaddr *)&server_addr,
sizeof(server_addr)); //Sending a User created EOF Flag

                }

        // Send "SENT" message to clear that WINDOW SPACE sent
        sendto(sockfd, sent_message, sizeof(sent_message), 0, (struct sockaddr *)&server_addr,
sizeof(server_addr));


        if(feof(fp)){ ////// Exit if File read completely (non-check part added)
                break;
                }

        // Message received from Server (Either Sequence of missing packet or ack-message
        ack = recvfrom(sockfd, ack_message, sizeof(ack_message), 0, (struct sockaddr
*)&server_addr, &addr_size);

        if(!strcmp(ack_message,"Claim?\0")){ // Send Packets count (in WIndow Phase) sent
                ack_message[0] = packet_counter+'0';

                sendto(sockfd, ack_message, 1, 0, (struct sockaddr *)&server_addr,
sizeof(server_addr));
        }
```

```
        // Receiving an Acknowledgement Message
        ack = recvfrom(sockfd, ack_message, sizeof(ack_message), 0, (struct sockaddr
*)&server_addr, &addr_size);



        if(ack){

                // While all WIndows Packets not received by Server, Resend according to IDs
specified by Server
                while(!strcmp(ack_message, "===!RECEIVED===\0")){ // While not same

                        bzero(ack_message, sizeof(ack_message));
                        ack = recvfrom(sockfd, ack_message, sizeof(ack_message), 0, (struct
sockaddr *)&server_addr, &addr_size);

                        while(strcmp(ack_message, sent_message)){ // While not current
Server'cycle's missing packets sequences sent
                                packet_counter = ack_message[0]-'0';

                                sendto(sockfd, WIN_SPACE[packet_counter-1], SIZE, 0, (struct
sockaddr *)&server_addr, sizeof(server_addr));

                                bzero(ack_message, sizeof(ack_message));
                                ack = recvfrom(sockfd, ack_message, sizeof(ack_message), 0, (struct
sockaddr *)&server_addr, &addr_size);
                        }

                        // Client tells server that it's done for current loop
                        sendto(sockfd, sent_message, sizeof(sent_message), 0, (struct sockaddr
*)&server_addr, sizeof(server_addr));

                        // Waiting to know whether any need for next loop checking if "!
RECEIVED" message received
                        ack = recvfrom(sockfd, ack_message, sizeof(ack_message), 0, (struct
sockaddr *)&server_addr, &addr_size);
                }
        }
```

```
        bzero(ack_message, sizeof(ack_message));
        bzero(data, SIZE);
        for(int i=0; i<WIN_SIZE;i++){
                bzero(WIN_SPACE[i],SIZE);
        }



}

// Send END-FLAG as File is read completely
sendto(sockfd, EFLAG, strlen(EFLAG), 0, (struct sockaddr *)&server_addr,
sizeof(server_addr)); //Sending a User created EOF Flag

printf("[+]File data sent to the Target Socket\n");

close(sockfd);

printf("[+]File Closed\n");

return 0;
}
```