Project Report

Computer Networks

Instructor: Sir Dr. Muhammad Zeeshan

Submitted by

Abdul Ghaffar Kalhoro: 194699

Hamad Nasir: 120312

Introduction:

Socket Programming:

Socket programming is very important for communication between network applications. Socket programming enables communication between network software applications using standard mechanisms, which are built in to network hardware and operating system. A socket represents point-to-point communication between exactly two pieces of software. Multiple sockets are used for communication between more than two network software applications with client/server or distributed system. Socket based communication done on two different applications on the network but they can also be used for local communication on the same computer. Socket communication on the network is bi-directional means application on each end can act as either sender or a receiver. Conventionally the one that initiates communication is termed as client and the one that responds to request as server.

Transmission Control Protocol (TCP):

TCP communication is connection oriented, which means that first connection is established between client and server. The connection is maintained until all the messages are exchanged between both ends. It breaks the application data in packets that the network can send. It then sends packets to and accepts from network layer. It is used in situations where we want error free data transmission, so it has to deal with retransmission of dropped packets and with acknowledgement that all the packets have reached their destination. In an open system interconnection (OSI) communication model, TCP deals with transport layer and parts of session layer.

User Data Protocol (UDP):

UDP is an alternative communication protocol to TCP. It is used in situations where reliability, data integrity and order is not required. It does not cater for the dropped packets and allows for them to be received in a different order rather than the one in which they were sent for better performance Therefore UDP is used to send short messages called datagram and it is a connectionless, unreliable protocol. . In an open system interconnection (OSI) communication model, UDP deals with session layer and some parts of transport layer.

Problem Statement:

This statement of problem is to write a sender-receiver application that implements audio file transfer over UDP protocol using Linux/GNU C sockets. Sender is required to open an audio 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. The main objective is to implement reliability functionality of TCP in UDP.

Solution:

The traditional UDP protocol does not provides reliability as it does not cater loss of the data. The problem is to implement TCP like reliability in UDP and transfer an audio file from sender to receiver. The solution to this problem is achieved by implementing following functionalities in UDP.

Heading	Approach
Retransmission	In case the loss occurs and receiver does not acknowledge the arrival of packets, then packets are re-transmitted until acknowledgement received.
Sequence Number	Each packet contains a sequence number determining the sequence number of packets.
Re-ordering on receiver side	The packets are re-ordered on the receiver side depending on the sequence numbers. In this way, the packets are received in the same order in which they were sent.
Window size	The window size of five segment is maintained thus implementing stop n wait.

Code Explanation:

Sending Side:

- First all the required libraries are included.
- In the main () function we have created server socket. Programutiltiy () function is called with parameter passing of socket status. In this function whole logic of sender is implemented.
- In a variable the audio file which was to be sent t receiver was defined along with buffer size which in our case was 500.
- A structure of packet was defined containing sequence number, a character array for holding data and array size.
- Many variables were defined each related to particular functionality in the programutiltiy () function. The important ones are as follows.
    - **Value_time.tv_usec:** Time out of interval 0.2 seconds is set so that if the packet is not received the program does not stuck and after this interval the packet is resend.
    - **arrayAck[]:** This array keeps the record of all the acknowledgements received of the packets.
    - **pkts_structVar[]:** This array stores all the packets at specific index values. These index value are pivotal in defining the sequence number of packets.
- A SOCK_DGRAM type socket is created which shows UDP connection is to be established and error checking if condition is also implemented.
- Then server address was initialized and memset was used to append with zeroes. Using the address and the port number we created aforementioned.
- Then the audio file was opened and error checking was implemented.
- The fstat() was implemented to give the statistical information of the audio file also error checking was also implemented.
- The time out interval is set to 0.2 seconds.
- Now a series of while loops are used to transmit the audio file.
- The first while loop is used to read the file in packets in the buffer array and sequence numbers are attached to each packet. It also updates the length of packet array.
- The next while loop is the main loop in which contains the reset tag which implements the selective repeat functionality.
- Inside main while loop is a nested while loop which uses in build sendto () function which is used to send the packets stored in packet array to the receiver.
- There is another nested while loop inside the main while loop. Inside it an arrayAck[] array is maintained which was initialized before. It maintains confirmations whether an ack has been received or not. If a packet is not received then on that specific index of arrayAck [] array an empty slot is created, this is what selective repeat will be using. In build strcmp() function will be used which checks for a string comparison with string “ack”. If no comparison is found then it is considered as data loss and packet associated to that ack is resend. The first nested loop in main while loop after altering the RESEND tag, it checks

whether arrayAck [i]==0, if it is then ack is not received and then selective retransmission of packet takes place.

- The last nested loop simply breaks the file in to packets, assign them sequence numbers and update the length according to the condition.
- Finally the packets are send to receiver and after completion the socket is closed.

Receiver Side:

- In the main () function we have created server socket. Programutiltiy () function is called with parameter passing of socket. In this function whole logic of receiver is implemented.
- The socket is initialized by passing the port number and server address.
- The function bind () tells the binding of the port number and socket address. With that there is error checking provoked.
- In integer type variable fileDescriptor we are opening the audio file and checking whether file exists or not. If not then it creates a file of this name.
- To declare the instance of packet struct_pkt is initialized. This instance is used for each arriving packet.
- For packet receiving a nested while loop is implemented in a while loop which is conditional to the length of the variable. This loop provides the functionality of receiving all the sent packets where they have same order that they sent before. The packets are then stored in pkts_structVar[] in the same order in which they were sent. The indication to complete packet transfer is when the packet size becomes zero and control is shifted to end tag.
- Then another loop is built which will send acks of received packets to the sender also receiver will maintain an array of acks called acks_structVar[] to keep track of all the packets received.
- The final while loop is implementing the functionality of writing the receiving packets in to file. All the received packets are now written in the fileDescriptor variable that was defined above.
- There is another function recFromsendr () which gives the integer value of remaining data that is to be send to the receiver. It is called inside the programUtility () function.
- After this the socket is now closed.

Functionalities Implementation Explanation:

The explanation of different functionalities that were implemented in order to introduce reliability is as follows.

Functionality	Code Approach
Sequence Number	The functionality of sequence number was introduced by implementing the packet structure, which includes the introduction of instance variable in each struct packet. This instance variable value points to the sequence number of that packet that provides ordering of packets.
Selective Repeat through retransmission:	When the receiver successfully receives a packet, it sends an acknowledgement or ack to the sender. To keep track of all the acks received an array ack[] is maintained. If there is no ack received for a particular packet at its index, then that packet will have a particular index value in the acks array. String comparison is done to check whether the ack is received or not. In case the ack is not received the control is shifted to RESEND. If ack[i]==0 the loop only resends those packets whose acknowledgement is not yet received.
Window Size	The audio file which was to be sent to the receiver is portioned in to 5 segments as per the requirement of the problem, this is how the window size is implemented. This was implemented by using various loops including those which were used to send, partition, receive or write the packets.
Window size	By using indexes and arrays as pivots we implemented the ordering of packets at the both sender and receiver side. These indexes determines the sequence numbers and this is how we avoided un-ordering of the packets. When a packet has been received by the receiver its sequence number is determined by x + 1.

Conclusion:

In this project, we implemented the fundamental concepts of socket programming and introduced reliability in UDP, which traditionally does not provide it. This is done by implementing TCP like functionalities including retransmission when packet loss occurs, sequence number, window size for stop n wait and reordering of packets depending of sequence number. We learned GNU C coding on Linux Ubuntu system and its powerful use.

GNU C Code:

Sender Side Code:

```
#include <stdio.h>
#include <time.h>
#include <sys/types.h>

#include <stdlib.h>
#include <errno.h>

#include <fcntl.h>
#include <string.h>

#include <sys/socket.h>
#include <sys/stat.h>
#include <unistd.h>

#include <netinet/in.h>
#include <sys/time.h>
```

```
#include <sys/sendfile.h>


#include <arpa/inet.h>
#include <sys/types.h>

//function declaration
void programUtility(int rec_sokt);
//packat structure
struct struct_pkt{
        int sequenceNo; //Sequence number of the packet
        char data[500]; //Data of packet in bytes
        int payLoadSIze; //Size of payload in data variable
};
/* Ending identifier */
char *END_FLAG = "~~~~~~flag value for ending~~~~";

//main function
int main(int argc, char **argv)
{
      int send_sokt;
      /* Creating the sender socket */
      send_sokt = socket(AF_INET, SOCK_DGRAM, 0);
        //calling function program Utility.
        programUtility(send_sokt);
```

```
        return 0;
}
void programUtility(int send_sokt){
          ssize_t ssize;
        struct sockaddr_in      sendAdrs;
        int fileDescriptr;
//To specify a time in nano-seconds and seconds timespec structure is used.
        struct timespec tSpecVar1, tSpecVar2;
        tSpecVar1.tv_sec = 0;
        tSpecVar1.tv_nsec = 500000L;
        int ACK;
        int arrayAck[5];
          int tempInt;
        socklen_t     sock_length;
        off_t size_file;
        struct stat statFile;
        struct struct_pkt pkts_structVar[5];
        struct timeval Value_time;
                Value_time.tv_sec = 0;
                Value_time.tv_usec = 1000000;

        int pkt_countVar;
          int counter = 0;
```

```
/* assigning the sendAdrs struct to zero */
memset(&sendAdrs, 0, sizeof(sendAdrs));
/* Constructing sendAdrs struct */
        bzero(&sendAdrs, sizeof(sendAdrs));
sendAdrs.sin_family = AF_INET;
inet_pton(AF_INET, "127.0.0.1", &sendAdrs.sin_addr);
sendAdrs.sin_port = htons(29314);
sock_length = sizeof(struct sockaddr_in);
/* Opening audio file as read-only*/
fileDescriptr = open("audiosend.mp3", O_RDONLY);
/* Getting the statistics of file */
fstat(fileDescriptr, &statFile);
/* Printing payLoadSIze of file to be sent*/
fprintf(stdout, "File Size: \n%ld bytes\n", statFile.st_size);
/* Sending file payLoadSIze to receiver */
size_file = statFile.st_size;
sendto(send_sokt, &size_file, sizeof(off_t), 0, (struct sockaddr *) &sendAdrs,
sock_length);
/* Setting timeout for send_sokt */
        setsockopt(send_sokt, SOL_SOCKET, SO_RCVTIMEO, &Value_time,
sizeof(Value_time));
/* Packet defined for receiving Acks */
struct struct_pkt pkt;
/* Reading file data into pkts_structVar in a buffer array */
        tempInt = 1;
```

```
ssize = 1;
pkt_countVar = 0;
  while ( counter < 5){
  ssize = read(fileDescriptr, pkts_structVar[counter].data, 500);
  pkts_structVar[counter].sequenceNo = tempInt;
  pkts_structVar[counter].payLoadSIze = ssize;
  tempInt += 1;
   pkt_countVar++;
        counter++;
  }
//////////////////////////////////////to use func here///////////////
/* while loop for sending data receiving Acks and reading further data */
while (ssize > 0 && tempInt > 1)
{
  /* Resend tag */
  RESEND:
  counter = 0;
  /* Loop for sending pkts_structVar in the buffer array */
  while ( counter < 5){
    /* Check if ACK for that struct_pkt is received or not */
    if (arrayAck[counter]== 0){
      printf("Sending packet no: %d\n", pkts_structVar[counter].sequenceNo);
      sendto(send_sokt, &pkts_structVar[counter], sizeof(struct struct_pkt), 0, (struct sockaddr *) &sendAdrs, sock_length);
```

```
            nanosleep(&tSpecVar1, &tSpecVar2);
        }
    counter++;
    }
    printf("\n-------------------------------------------------\n");
    counter = 0;
    /* Loop for receiving Acks */
    while (counter < 5){
            ACK = recvfrom(send_sokt, &pkt, sizeof(struct struct_pkt), 0, (struct
sockaddr *)&sendAdrs , &sock_length);
        if (strcmp(pkt.data, "Ack") == 0 && ACK > 0){
                printf("Ack number received: %d\n", pkt.sequenceNo);
            arrayAck[pkt.sequenceNo - 1] = 1;
            }else{
            /* If Ack is not received */
            goto RESEND;
        }
            counter++;
    }
            printf("\n-------------------------------------------------\n");
    /* Zeroing array of Acks */
    memset(arrayAck, 0, sizeof(arrayAck));
    /* Zeroing array of pkts_structVar */
    memset(pkts_structVar, 0, sizeof(pkts_structVar));
    /* Loop for reading next 5 pkts_structVar */
```

```
        tempInt = 1;
                counter = 0;
                while (counter < 5){
                ssize = read(fileDescriptr, pkts_structVar[counter].data, 500);
            pkts_structVar[counter].sequenceNo = tempInt;
            pkts_structVar[counter].payLoadSIze = ssize;
            tempInt += 1;
            if (ssize > 0){
               pkt_countVar++;
            }
                counter++;
                }
                }
    printf("Total packets: %d\n", pkt_countVar);
    /* Sending the end flag  */
                struct struct_pkt end_pack;
    strcpy(end_pack.data, END_FLAG);
                printf("Sending the end flag\n");
                sendto(send_sokt, &end_pack, sizeof(struct struct_pkt), 0,  (struct sockaddr *) &sendAdrs, sock_length);

    /* Closing the socket */
    close(send_sokt);
return;
}
```

Receiver Side Code:

```
#include <sys/socket.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/sendfile.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <sys/stat.h>
#include <fcntl.h>
void programUtility(int rec_sokt);
int recFromsendr(int rec_sokt,  struct sockaddr_in recv_adres, socklen_t skt_len);
/*identifier for ending*/
char *END_FLAG = "~~~~~~flag value for ending~~~~";
/*this is the structure for the packet that contains 3 things:
        1)      sequence number.
```

```
        2)      data
        3)      payload
*/
struct struct_pkt{
        int sequenceNo; //shows Sequence number for struct_pkt
        char data[500]; //Data of size 500
        int payLoadSIze; //total payload size
};
int main(int argc, char **argv)
{
  int rec_sokt;
      /* Create server socket */
      rec_sokt = socket(AF_INET, SOCK_DGRAM, 0);

        //program utility function calling.
        programUtility(rec_sokt);

       return 0;
}       //ending main function

/*this funciton receives input integer value of
receiver socket as parameter and returns nothing. */
void programUtility(int rec_sokt){

        int dataRemaining;
        socklen_t    skt_len;              //socket length variable
```

```
ssize_t ssize;                    //socket size variable
struct sockaddr_in     recv_adres;
int rcv_data_byte = 0;

   int fileDescriptr;

struct struct_pkt pkts_structVar[5];
struct struct_pkt acks_structVar[5];

int NAcks;
int totalACKs;
int size_send;
   int counter;
   printf("open to receive....\n\n");
// appending Zero to recv_adres struct
memset(&recv_adres, 0, sizeof(recv_adres));
// making recv_adres structure
            bzero(&recv_adres, sizeof(recv_adres));
recv_adres.sin_family = AF_INET;
recv_adres.sin_addr.s_addr=htonl(INADDR_ANY);
recv_adres.sin_port = htons(29314);     //here port number used is 29314
skt_len = sizeof(struct sockaddr_in);

// Binding receiver socket
bind(rec_sokt, (struct sockaddr *)&recv_adres, sizeof(struct sockaddr));
```

```
//opening file if not exist otherwise create new file.
fileDescriptr = open("audioreceive.mp3", O_RDWR | O_CREAT, 0666);
  //calling function to receiving from sender.
  dataRemaining = recFromsendr(rec_sokt, recv_adres,skt_len);
// Creating struct_pkt for data receiving
struct struct_pkt pkt;
totalACKs = 0;
  while(ssize != -1 && dataRemaining > 0)
        {
  label1:
        counter = 0;
              while(counter < 5 ){
    ssize = recvfrom(rec_sokt, &pkt, sizeof(struct struct_pkt), 0, (struct sockaddr *)&recv_adres , &skt_len);
       if (ssize > 0){
                              if (strcmp(pkt.data, END_FLAG) == 0){
                    ssize = -1;
          break;
                    }
              pkts_structVar[pkt.sequenceNo - 1] = pkt;
              }
              counter++;
           }

  NAcks = 0;
              counter = 0;
```

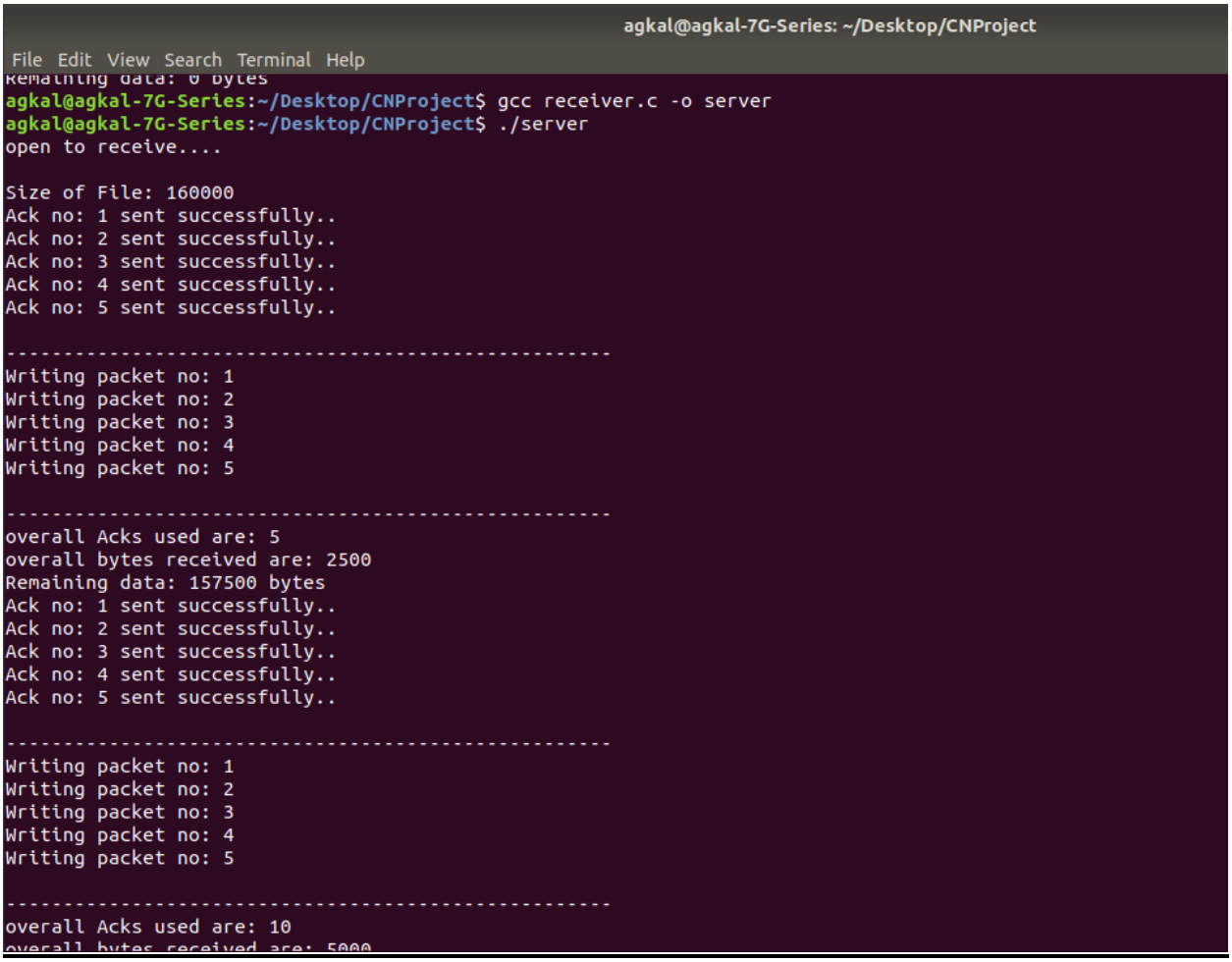
```
                while(counter < 5){
                        strcpy(acks_structVar[counter].data, "Ack");
                        acks_structVar[counter].sequenceNo = counter + 1;
                        size_send = sendto(rec_sokt, &acks_structVar[counter],
sizeof(acks_structVar[counter]), 0,  (struct sockaddr *) &recv_adres, skt_len);
            if (size_send > 0){
               NAcks++;
               totalACKs++;
               printf("Ack no: %d sent successfully..\n",
acks_structVar[counter].sequenceNo);
            }
                    counter++;
                }

        Printf("\n--------------------------------------------------------------------------\n");

        if (NAcks < 5){
           goto label1;
        }
                    counter = 0;
                    while(counter < 5 && NAcks == 5 ){
              /* Check for data into struct_pkt */
                        if (pkts_structVar[counter].payLoadSIze != 0){
                            printf("Writing packet no: %d\n",
pkts_structVar[counter].sequenceNo);
                            write(fileDescriptr, pkts_structVar[counter].data,
pkts_structVar[counter].payLoadSIze);
```

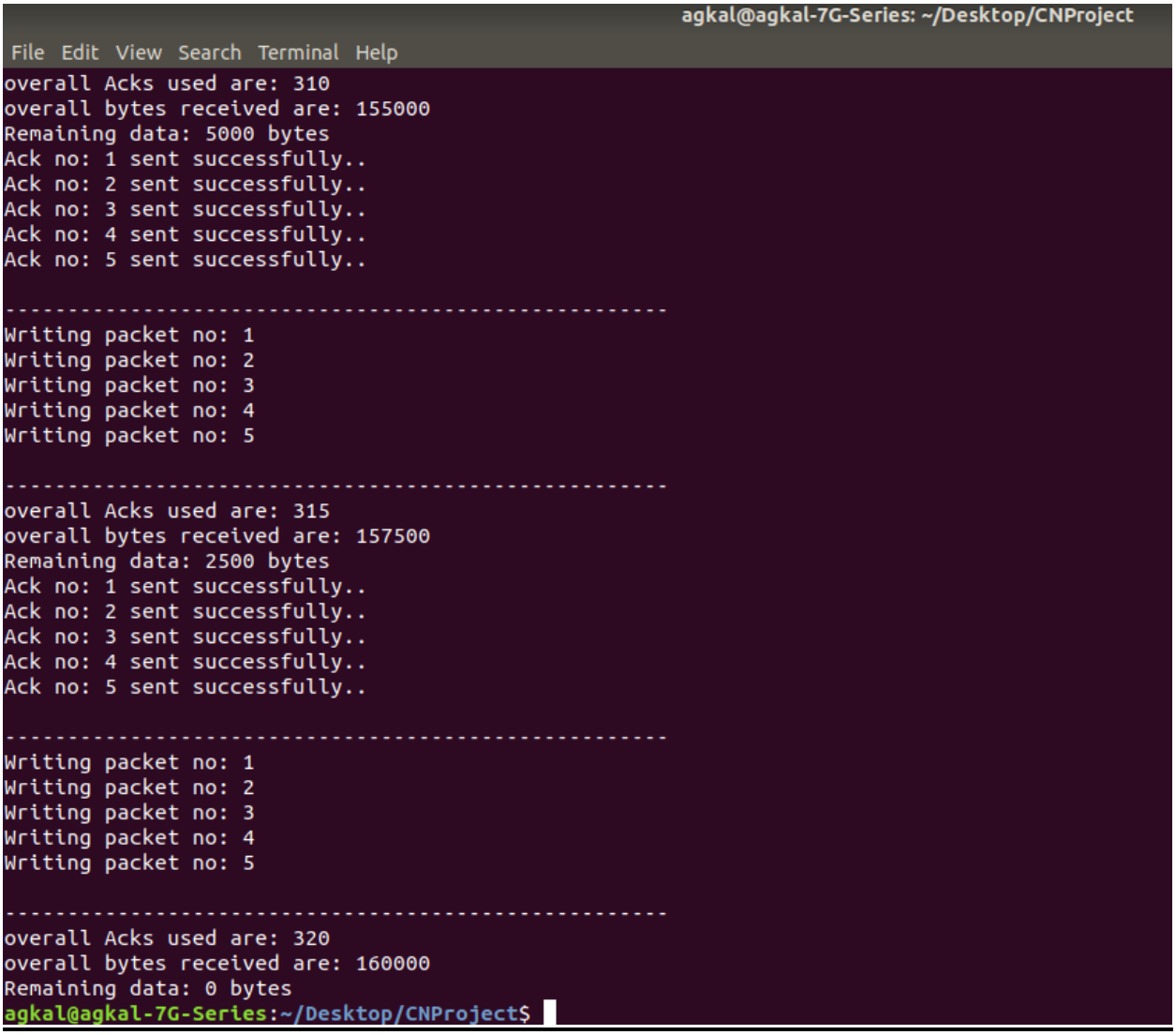
```
            rcv_data_byte += pkts_structVar[counter].payLoadSIze;
            dataRemaining -= pkts_structVar[counter].payLoadSIze;
                        }
                        counter++;
                    }
        Printf("\n--------------------------------------------------------------------------------\n");
        /* append zeros to pkts_structVar */
        memset(pkts_structVar, 0, sizeof(pkts_structVar));

        printf("overall Acks used are: %d\n", totalACKs);
        printf("overall bytes received are: %d\nRemaining data: %d bytes\n", rcv_data_byte, dataRemaining);
    }

    close(rec_sokt);
    close(fileDescriptr);
return;
}

//this function returns the integer value of dataRemaining
int recFromsendr(int rec_sokt,  struct sockaddr_in recv_adres, socklen_t skt_len){
        int dataRemaining;
        off_t size_file;
    struct timeval Value_time;
            Value_time.tv_sec = 0;
```

```
            Value_time.tv_usec = 1000000;

// Receive size_file from sender
    recvfrom(rec_sokt, &size_file, sizeof(off_t), 0, (struct sockaddr *)&recv_adres , &skt_len);
            dataRemaining = size_file;
    printf("Size of File: %ld\n", size_file);

    // Setting timeout for sender_socket
      /*setsockopt() function used in setting options that are related with a socket. now these Options can exist at multiple  levels of protocol*/
            setsockopt(rec_sokt, SOL_SOCKET, SO_RCVTIMEO, &Value_time, sizeof(Value_time));

return      dataRemaining;

};
```

Output Screenshots:

Receiver:

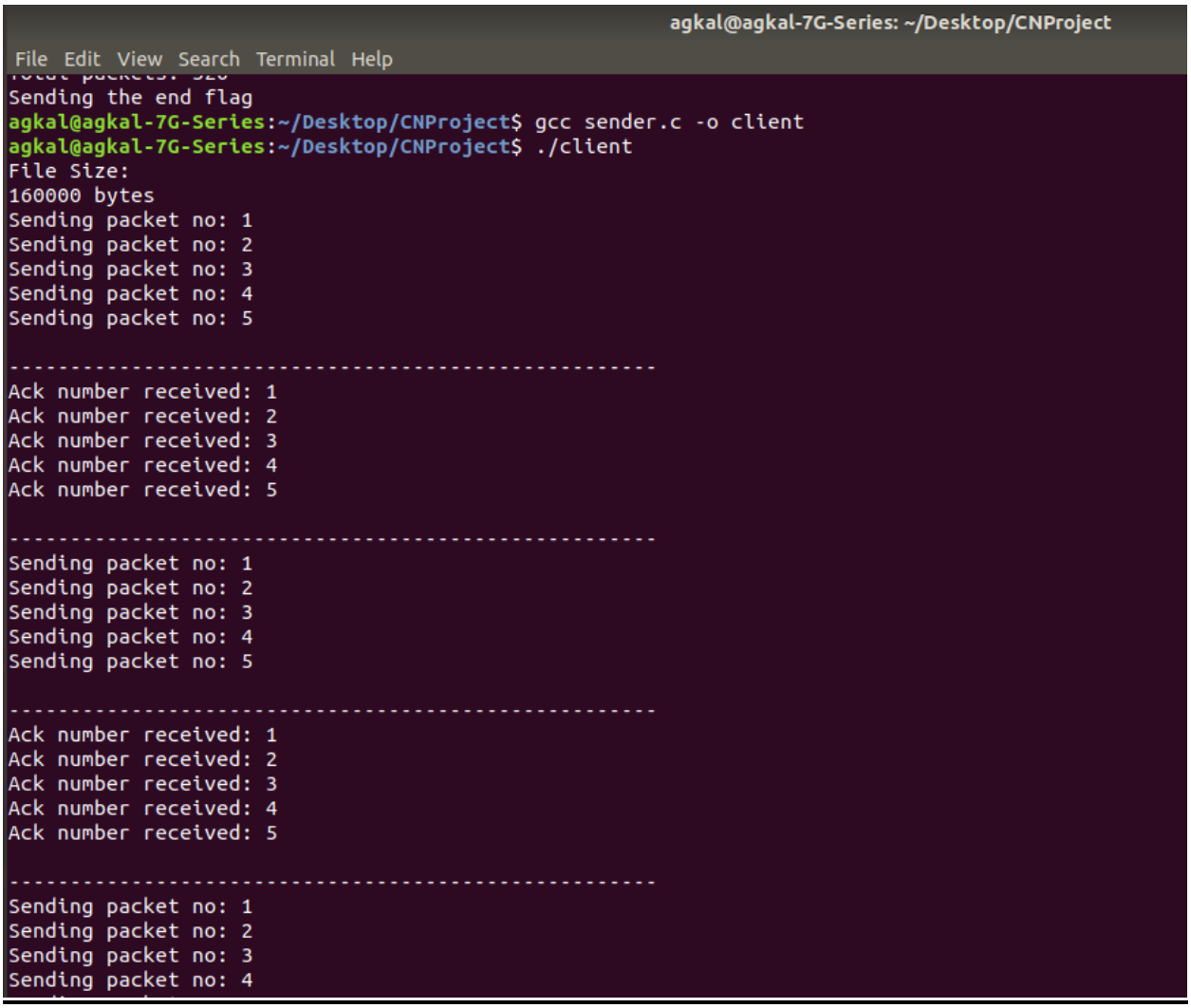

agkal@agkal-7G-Series: ~/Desktop/CNProject
File Edit View Search Terminal Help
Remaining data: 0 bytes
agkal@agkal-7G-Series:~/Desktop/CNProject$ gcc receiver.c -o server
agkal@agkal-7G-Series:~/Desktop/CNProject$ ./server
open to receive....
Size of File: 160000
Ack no: 1 sent successfully..
Ack no: 2 sent successfully..
Ack no: 3 sent successfully..
Ack no: 4 sent successfully..
Ack no: 5 sent successfully..
Writing packet no: 1
Writing packet no: 2
Writing packet no: 3
Writing packet no: 4
Writing packet no: 5
overall Acks used are: 5
overall bytes received are: 2500
Remaining data: 157500 bytes
Ack no: 1 sent successfully..
Ack no: 2 sent successfully..
Ack no: 3 sent successfully..
Ack no: 4 sent successfully..
Ack no: 5 sent successfully..
Writing packet no: 1
Writing packet no: 2
Writing packet no: 3
Writing packet no: 4
Writing packet no: 5
overall Acks used are: 10

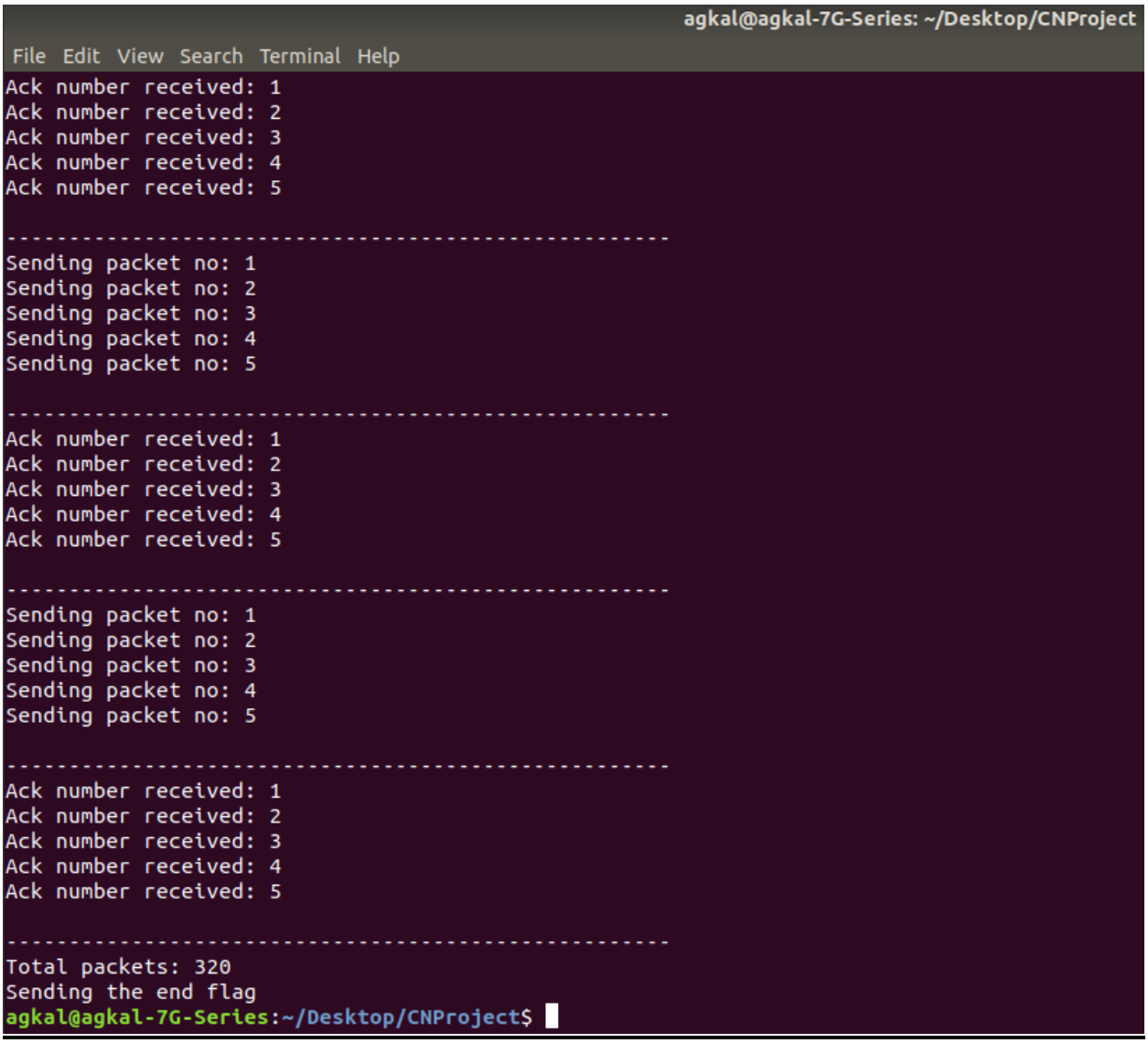

agkal@agkal-7G-Series: ~/Desktop/CNProject
File Edit View Search Terminal Help
overall Acks used are: 310
overall bytes received are: 155000
Remaining data: 5000 bytes
Ack no: 1 sent successfully..
Ack no: 2 sent successfully..
Ack no: 3 sent successfully..
Ack no: 4 sent successfully..
Ack no: 5 sent successfully..

Writing packet no: 1
Writing packet no: 2
Writing packet no: 3
Writing packet no: 4
Writing packet no: 5
----
overall Acks used are: 315
overall bytes received are: 157500
Remaining data: 2500 bytes
Ack no: 1 sent successfully..
Ack no: 2 sent successfully..
Ack no: 3 sent successfully..
Ack no: 4 sent successfully..
Ack no: 5 sent successfully..

Writing packet no: 1
Writing packet no: 2
Writing packet no: 3
Writing packet no: 4
Writing packet no: 5
----
overall Acks used are: 320
overall bytes received are: 160000
Remaining data: 0 bytes
agkal@agkal-7G-Series:~/Desktop/CNProject$

Sender:


agkal@agkal-7G-Series: ~/Desktop/CNProject
File Edit View Search Terminal Help
Sending the end flag
agkal@agkal-7G-Series:~/Desktop/CNProject$ gcc sender.c -o client
agkal@agkal-7G-Series:~/Desktop/CNProject$ ./client
File Size:
160000 bytes
Sending packet no: 1
Sending packet no: 2
Sending packet no: 3
Sending packet no: 4
Sending packet no: 5
Ack number received: 1
Ack number received: 2
Ack number received: 3
Ack number received: 4
Ack number received: 5
Sending packet no: 1
Sending packet no: 2
Sending packet no: 3
Sending packet no: 4
Sending packet no: 5
Ack number received: 1
Ack number received: 2
Ack number received: 3
Ack number received: 4
Ack number received: 5
Sending packet no: 1
Sending packet no: 2
Sending packet no: 3
Sending packet no: 4


agkal@agkal-7G-Series: ~/Desktop/CNProject
File Edit View Search Terminal Help
Ack number received: 1
Ack number received: 2
Ack number received: 3
Ack number received: 4
Ack number received: 5
Sending packet no: 1
Sending packet no: 2
Sending packet no: 3
Sending packet no: 4
Sending packet no: 5
Ack number received: 1
Ack number received: 2
Ack number received: 3
Ack number received: 4
Ack number received: 5
Sending packet no: 1
Sending packet no: 2
Sending packet no: 3
Sending packet no: 4
Sending packet no: 5
Ack number received: 1
Ack number received: 2
Ack number received: 3
Ack number received: 4
Ack number received: 5
Total packets: 320
Sending the end flag
agkal@agkal-7G-Series:~/Desktop/CNProject$