Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam
603 110
(An Autonomous Institution, Affiliated to Anna University, Chennai)
Department of CSE

UCS 2501 Computer Networks

ASSIGNMENT 2

Simulation of allocating segment number based on the number of bytes transmitted and its acknowledgment number (Q. no – 12)

TEAM MEMBERS :

S.NO.	NAME	REG. NO.
1)	Akila R	3122215001006
2)	Apurva Narayan	3122215001011
3)	Arvind Minjur	3122215001012
4)	Ashwini Parvatha G S	3122215001016
5)	Bhavana A	3122215001019
6)	Charumathi P	3122215001020

INDEX

PROBLEM STATEMENT

The problem statement is to simulate the allocation of the segment number based on the number of bytes transmitted through the data packet by the sender and its corresponding acknowledgment number sent by the receiver in response to the data packet.

In network communication, data is divided into packets for transmission, and each packet is assigned a sequence number for identification. The sender is responsible for packaging the data into these packets, while the receiver acknowledges the receipt of each packet by specifying an acknowledgment number. This project simulates the dynamic allocation of sequence numbers based on the size of the transmitted data packets and displaying the segment number of the data packet and the acknowledgment numbers provided by the receiver. This simulation would help in understanding and optimizing the communication process, ensuring reliable and efficient data transfer between sender and receiver in a networked environment.

TERMINOLOGIES :

Sequence Number : A unique identifier assigned to each packet or segment of data transmitted over a network.

Segment Number : Segments are portions of a message or data stream that are divided for transmission over a network by the transport layer. Each segment is assigned a unique identifier, typically a sequence number, to facilitate the proper assembly and ordering of the data at the receiver end.

Acknowledgment Number : It is a number sent by receiver indicating the next sequence number the receiver is expecting to receive.

METHOD IMPLEMENTED

To simulate the required segment and acknowledgment numbers, **Transmission Control Protocol (TCP)** is used which is a Connection – oriented protocol which is reliable of the packet delivery. It was implementer using socket programming.

In the solution implemented, the **sender** initiates the communication by sending a connection request packet to the receiver. The sender assigns a sequence number to this packet, and upon receiving an acknowledgment from the receiver with the acknowledgment number for the sent sequence number, the sender considers the connection established and sends a final acknowledgment to the receiver with the MSS.

The segment number of each packet is determined based on the Maximum Segment Size (MSS) parameter. The sender sends the data packets with the sequence number entered by the user. The segment number of the respective packet is determined as below :

$$\textbf{Segment Number} = \frac{\textbf{Sequence Number} - \textbf{Starting sequence number}}{\textbf{Maximum Segment Size (MSS)}}$$

The sender awaits acknowledgment from the receiver, and in case of a timeout, the packet is resent.

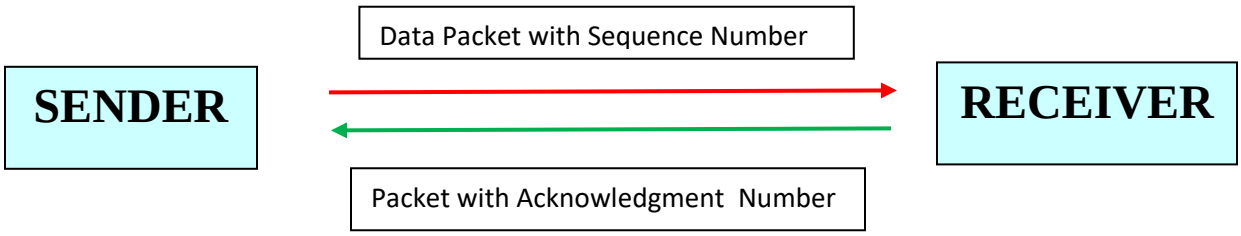
On the **receiver** side, the acknowledgment number is set in response to the received data packet, indicating the next expected sequence number i.e. :

$$\textbf{Acknowledgment Number} = \textbf{Sequence Number of Packet received} + 1$$

The acknowledgment is sent back to the sender, confirming the successful receipt of the data packet and need of the packets. The receiver also has a mechanism to handle timeouts and resends acknowledgments when necessary. This dynamic interaction between sender and receiver, along with the allocation of segment numbers, emulates a basic data transfer protocol with error handling and reliable communication.

TOPOLOGY

For this problem there is no specific topology. It just aims at simulating the transmission of data packets between a sender and a receiver and based on the number of bytes transmitted, the segment number and acknowledgment numbers are determined.



Example of Segment Numbers :

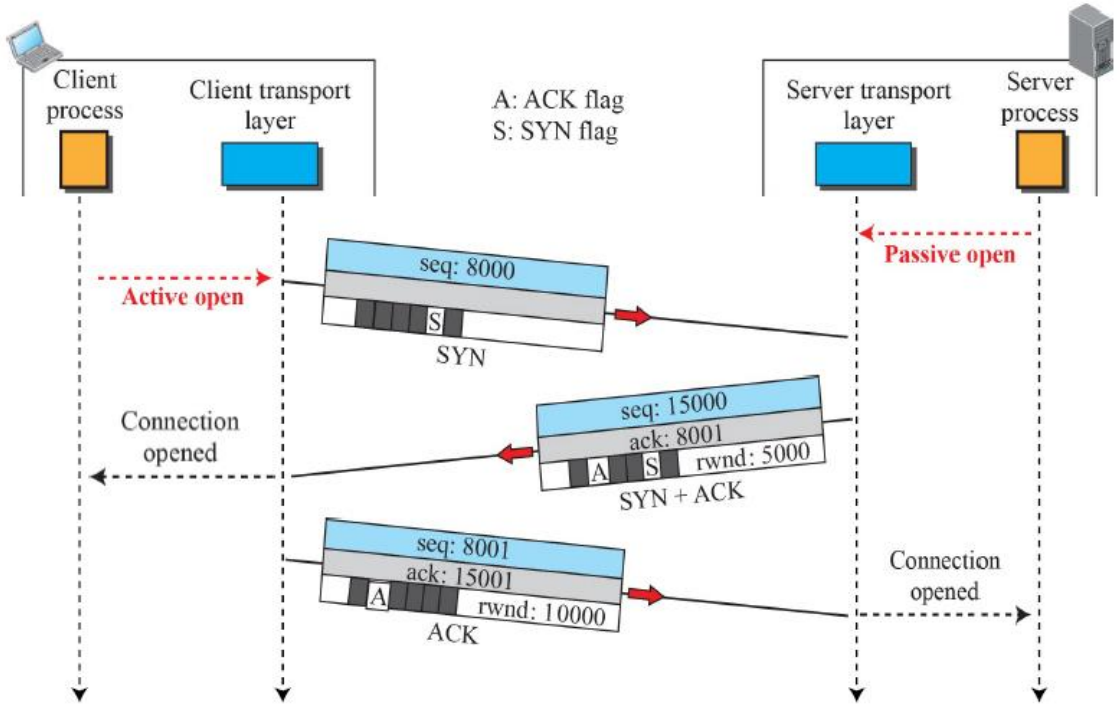
Segment 1 ==> sequence number: 10,010 (range: 10,010 to 11,009)

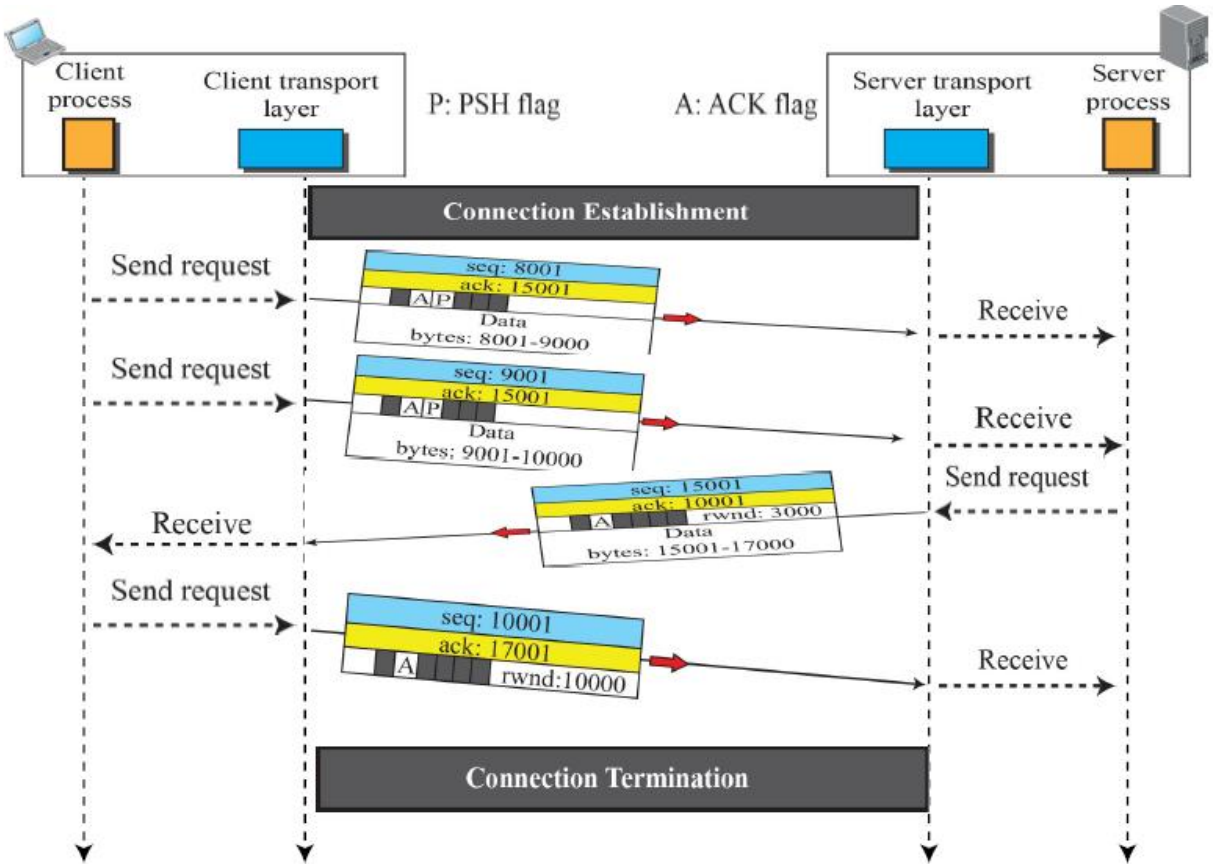
Segment 2 ==> sequence number: 11,010 (range: 11,010 to 12,009)

Segment 3 ==> sequence number: 12,010 (range: 12,010 to 13,009)

Example of a TCP Transmission with sequence numbers and Acknowledgment numbers :

CONNECTION ESTABLISHMENT





Example :

Maximum Segment Size : 1000

Sequence Number : 4001 (Sent 4001 to 5000 bytes)

Acknowlegment Number : 5001

SOURCE CODE

SENDER FILE :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <netinet/in.h>
#include <poll.h>

#define TIME_OUT 10
#define PORT 8080
#define MAX_BUFFER_SIZE 1024


struct DataPacket
{
        int MSS;                        // Maximum segment size
        int seq_no;
        int ack_no;
        int connected;
        char type;
};


int main(int argc, char *argv[])
{
    int clientSocket, seg_no = 0,seg_size = 0, seq_sent = 0, time = 0,
starting_seq = 0;
    struct sockaddr_in serverAddr, clientAddr;
    struct DataPacket Packet;
    socklen_t addrSize = sizeof(clientAddr);
    ssize_t bytes_received = 0, bytes_sent = 0;

    // Creating the socket :

    clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (clientSocket == -1)
    {
        perror("Error in creating socket.");
        exit(EXIT_FAILURE);
    }

    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = inet_addr(argv[1]);
    serverAddr.sin_port = htons(PORT);

    // Connecting to the receiver :
```

```
    connect(clientSocket,(struct sockaddr
*)&serverAddr,sizeof(serverAddr));
    printf("Connected to the receiver...\n");

    // Sending and Receiving the HandShake packet for Connection
establishment for Data Transfer :

    printf("\nEnter the Segment size : ");
    scanf("%d",&Packet.MSS);
    seg_size = Packet.MSS;
    printf("Enter the sequence number of handshaking : ");
    scanf("%d", &Packet.seq_no);
    seq_sent = Packet.seq_no;
    Packet.type = 'E';
    bytes_sent = write(clientSocket, &Packet, sizeof(Packet));
                    // Reqesting for data transfer
    printf("\nSequence number : %d\n",seq_sent);
    Packet.connected = 0;

    /*------------------------------------------------- CONNECTION
ESTABLISHMENT -----------------------------------------------------------
*/

    while (!Packet.connected)
    {
              struct pollfd poll_fd;
              poll_fd.fd = clientSocket;
              poll_fd.events = POLLIN;
              time = TIME_OUT * 1000;
              int poll_result = poll(&poll_fd, 1, time);

              if (poll_result == -1)
              {
                     perror("Poll error");
                     exit(EXIT_FAILURE);
              }
              else if (poll_result == 0)
              {
                     printf("Timeout occurred.\n");
                     exit(-1);
              }
              else
              {
                     if (poll_fd.revents & POLLIN)
                     {
                         bytes_received = read(clientSocket, &Packet,
sizeof(Packet));
                         alarm(0);
                     }
              }

               if (Packet.type == 'E' && Packet.ack_no == seq_sent + 1)
       // Received acknowlegment for the request for transfer
                  {
```

```
                                printf("Acknowlegment received : %d\n",
Packet.ack_no);
                                Packet.ack_no = Packet.seq_no + 1;
                                Packet.seq_no = seq_sent + 1;
                                bytes_sent = write(clientSocket, &Packet,
sizeof(Packet));                        // Sending the acknowlegment for
receiver's packet
                                printf("Acknowlegment sent :
%d\n",Packet.ack_no);


                                struct pollfd poll_fd;
                                poll_fd.fd = clientSocket;
                                poll_fd.events = POLLIN;

                                int poll_result = poll(&poll_fd, 1, TIME_OUT
* 1000);


                                if (poll_result == -1)
                                {
                                        perror("Poll error");
                                        exit(EXIT_FAILURE);
                                }
                                else if (poll_result == 0)
                                {
                                        printf("Timeout occurred.\n");
                                        exit(-1);
                                }
                                else
                                {
                                        if (poll_fd.revents & POLLIN)
                                        {
                                                bytes_received =
read(clientSocket, &Packet, sizeof(Packet));
                                                if (Packet.connected == 1)
                                                {
                                                        printf("\nConnection
Established for Data Transfer....\n");
                                                        break;
                                                }
                                        }
                                }
                        }
    }

   /*-------------------------------------------------------------- DATA
TRANSFER --------------------------------------------------------------------
-----------*/

   // Displaying the segment number after getting packet number from user
:

   printf("\nEnter the starting packet's sequence number : ");
   scanf("%d",&starting_seq);
```

```
   Packet.seq_no = starting_seq;
   seq_sent = Packet.seq_no;
   Packet.type = 'D';
   seg_no = ((Packet.seq_no - starting_seq)/seg_size)+1;
   printf("\nSequence number : %d.\n",Packet.seq_no);
   printf("Segment Number : %d.\n",seg_no);
   bytes_sent = write(clientSocket, &Packet, sizeof(Packet));
       // Sending a data Packet
   sleep(2);

    while (1)
    {
           struct pollfd poll_fd;
           poll_fd.fd = clientSocket;
           poll_fd.events = POLLIN;
           time = TIME_OUT * 1000;
           int poll_result = poll(&poll_fd, 1, time);

           if (poll_result == -1)
           {
              perror("Poll error");
              exit(EXIT_FAILURE);
           }
           else if (poll_result == 0)
           {
              printf("\nTimeout occurred. Resending the packet.\n");
              printf("Sequence number : %d\n", seq_sent);
              bytes_sent = write(clientSocket, &Packet, sizeof(Packet));
              continue;
           }
           else
           {
              if (poll_fd.revents & POLLIN)
              {
                     bytes_received = read(clientSocket, &Packet,
sizeof(Packet));
                     alarm(0);
                     if (bytes_received <= 0)
                     {
                         printf("Connection closed by receiver.\n");
                         break;
                     }

                     if (Packet.type != 'D')
                     {
                            printf("Connection terminated by
receiver.\n");
                            exit(-1);
                     }

                     if (Packet.ack_no == seq_sent + seg_size)
       // Acknowlegment received
                     {
                            Packet.type = 'D';
```

```
                                printf("\nAcknowlegment number :
%d.\n",Packet.ack_no);
                                printf("\nEnter the data to be sent : ");
                                scanf("%d",&Packet.seq_no);
                                seq_sent = Packet.seq_no;
                                seg_no = ((Packet.seq_no -
starting_seq)/seg_size)+1;
                                printf("\nSequence number :
%d.\n",Packet.seq_no);
                                printf("Segment Number : %d.\n",seg_no);
                                bytes_sent = write(clientSocket, &Packet,
sizeof(Packet));                // Sending the data packet
                        }
                        else
                // Send requested packet
                        {
                                Packet.type = 'D';
                                printf("\nAcknowlegment number :
%d.\n",Packet.ack_no);
                                seq_sent = Packet.ack_no;
                                Packet.seq_no = seq_sent;
                                seg_no = ((Packet.seq_no -
starting_seq)/seg_size)+1;
                                printf("\nSequence number :
%d.\n",Packet.seq_no);
                                printf("Segment Number : %d.\n",seg_no);
                                bytes_sent = write(clientSocket, &Packet,
sizeof(Packet));
                        }
                }
            }

        }

    // Closing the socket :

    close(clientSocket);

    return 0;
}
```

RECEIVER FILE :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <signal.h>
#include <poll.h>

#define TIME_OUT 10
#define PORT 8080
#define MAX_BUFFER_SIZE 1024


struct DataPacket
{
        int MSS;                        // Maximum segment size
        int seq_no;
        int ack_no;
        int connected;
        char type;
};


int main()
{
    int serverSocket, clientSocket, seq_no = 0, ack_no = 0,seg_size = 0,
ack_sent = 0, time = 0, connected = 0;
    struct sockaddr_in serverAddr, clientAddr;
    struct DataPacket Packet;
    socklen_t addrSize = sizeof(clientAddr);
    ssize_t bytes_received = 0, bytes_sent = 0;

    // Creating the socket :

    serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == -1)
    {
        perror("Error in creating socket.");
        exit(EXIT_FAILURE);
    }

    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(PORT);

    // Binding of the socket :
```

```
    if (bind(serverSocket, (struct sockaddr*)&serverAddr,
sizeof(serverAddr)) == -1)
    {
        perror("Error in binding the socket.");
        exit(-1);
    }

    // Listen for incoming connections :

    if (listen(serverSocket, 5) == -1)
    {
        perror("Error in listening for senders.");
        exit(-1);
    }

    printf("Server listening on port %d for the senders ...\n", PORT);

    // Accept the connection from the client :

    clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddr,
&addrSize);
    if (clientSocket == -1)
    {
        perror("Error in accepting connection.");
        exit(EXIT_FAILURE);
    }

    printf("Connection accepted.\n");

    // Receiving and Sending the HandShake packet for Connection
establishment for Data Transfer :

    bytes_received = read(clientSocket, &Packet, sizeof(Packet));

    if (bytes_received <= 0)
    {
       printf("Connection closed by sender.\n");
        exit(-1);
    }

    Packet.connected = 0;

    /*------------------------------------------------- CONNECTION
ESTABLISHMENT ----------------------------------------------------------
*/

    while (!Packet.connected)
    {
           if (Packet.type == 'E')
              // Received request for transfer
           {
              printf("Sequence number received : %d\n", Packet.seq_no);
              Packet.ack_no = Packet.seq_no + 1;
              ack_sent = Packet.ack_no;
```

```
                printf("Enter a sequence number for handshaking : ");
                scanf("%d",&Packet.seq_no);
                seg_size = Packet.MSS;
                bytes_sent = write(clientSocket, &Packet, sizeof(Packet));
                // Sending acknowlegment for transfer
                printf("Acknowlegement sent : %d\n", ack_sent);
                struct pollfd poll_fd;
                poll_fd.fd = clientSocket;
                poll_fd.events = POLLIN;
                time = TIME_OUT * 1000;
                int poll_result = poll(&poll_fd, 1, time);

                if (poll_result == -1)
                {
                        perror("Poll error");
                        exit(EXIT_FAILURE);
                }
                 else if (poll_result == 0)
                {
                        printf("\nTimeout occurred. NO acknowlegment
Received.\n");
                       exit(-1);
                }
                else
                {
                        if (poll_fd.revents & POLLIN)
                       {
                               bytes_received = read(clientSocket, &Packet,
sizeof(Packet));
                               time = TIME_OUT * 1000;
                               if (bytes_received <= 0)
                               {
                                       printf("Connection closed by
sender.\n");
                                       exit(-1);
                               }

                               if (Packet.type == 'E')
                                       // Checking if acknowleged ?
                               {
                                       if (Packet.seq_no == ack_sent)
                                               // YES
                                       {
                                               printf("\nAcknowlegement
received : %d\n", Packet.ack_no);
                                               printf("\nConnection
Established for Data Transfer....\n");
                                               Packet.connected = 1;
                                               bytes_sent =
write(clientSocket, &Packet, sizeof(Packet));
                                               sleep(5);
                                               break;
                                       }
                               }
```

```
                        }
                }
        }

    }


   /*------------------------------------------------------------- DATA
TRANSFER ------------------------------------------------------------------
-----------*/

   // Displaying the acknowlegement number after receiving the data :

    struct pollfd poll_fd;
    poll_fd.fd = clientSocket;
    poll_fd.events = POLLIN;
    time = TIME_OUT * 1000;
    int poll_result = poll(&poll_fd, 1, time);

    if (poll_result == -1)
    {
        perror("Poll error");
        exit(EXIT_FAILURE);
    }
    else if (poll_result == 0)
    {
        printf("Timeout occurred.\n");
       exit(-1);
    }
    else
    {
        if (poll_fd.revents & POLLIN)
        {
               bytes_received = read(clientSocket, &Packet,
sizeof(Packet));
               if (bytes_received <= 0)
               {
                      printf("Connection closed by sender.\n");
                      exit(-1);
               }

               Packet.ack_no = Packet.seq_no + seg_size;
               ack_sent = Packet.ack_no;
               printf("\nSequence number : %d\n",Packet.seq_no);
               printf("Acknowlegment sent : %d\n",Packet.ack_no);
               bytes_sent = write(clientSocket, &Packet, sizeof(Packet));
               sleep(2);
// Waiting for the sender to send the data
        }
    }

    while (1)
    {
       struct pollfd poll_fd;
```

```
    poll_fd.fd = clientSocket;
    poll_fd.events = POLLIN;
    time = TIME_OUT * 1000;

    int poll_result = poll(&poll_fd, 1, time );

    if (poll_result == -1)
    {
        perror("Poll error");
        exit(EXIT_FAILURE);
    }
    else if (poll_result == 0)
    {
        printf("\nTimeout occurred. Resending the Acknowlegment.\n");
       printf("Acknowlegment sent : %d\n", ack_sent);
       bytes_sent = write(clientSocket, &Packet, sizeof(Packet));
    }
    else
    {
        if (poll_fd.revents & POLLIN)
        {
              bytes_received = read(clientSocket, &Packet,
sizeof(Packet));
              alarm(0);
              if (bytes_received <= 0)
              {
                  printf("Connection closed by sender.\n");
                  break;
              }

              if (Packet.type != 'D')
              {
                     printf("Connection terminated by sender.\n");
                     exit(-1);
              }

              if (Packet.seq_no == ack_sent)                    //
Received the packet
              {
                     Packet.ack_no = Packet.seq_no + seg_size;
                     ack_sent = Packet.ack_no;
                     Packet.type = 'D';
                     printf("\nSequence number : %d\n",Packet.seq_no);
                     printf("Acknowlegment sent : %d\n", Packet.ack_no);
                     bytes_sent = write(clientSocket, &Packet,
sizeof(Packet));
              }
       }
    }


    }

    // Closing the sockets :
```

```
    close(clientSocket);
    close(serverSocket);

    return 0;
}
```

OUTPUT

SENDER

```
ashu@root:~/Desktop$ ./c 127.0.0.1
Connected to the receiver...

Enter the Segment size : 1000
Enter the sequence number of handshaking : 100

Sequence number : 100
Acknowlegment received : 101
Acknowlegment sent : 201

Connection Established for Data Transfer....

Enter the starting packet's sequence number : 1001

Sequence number : 1001.
Segment Number : 1.

Acknowlegment number : 2001.

Enter the data to be sent : 2001

Sequence number : 2001.
Segment Number : 2.

Acknowlegment number : 3001.

Enter the data to be sent : 3001

Sequence number : 3001.
Segment Number : 3.

Acknowlegment number : 3001.

Sequence number : 3001.
Segment Number : 3.

Acknowlegment number : 4001.

Enter the data to be sent : 8001

Sequence number : 8001.
Segment Number : 8.

Timeout occurred. Resending the packet.
Sequence number : 8001

Acknowlegment number : 4001.

Sequence number : 4001.
```

```
Sequence number : 4001.
Segment Number : 4.

Acknowlegment number : 5001.

Enter the data to be sent : 5001

Sequence number : 5001.
Segment Number : 5.

Acknowlegment number : 6001.

Enter the data to be sent : ^C
ashu@root:~/Desktop$
```

RECEIVER

```
ashu@root:~/Desktop$ ./s
Server listening on port 8080 for the senders ...
Connection accepted.
Sequence number received : 100
Enter a sequence number for handshaking : 200
Acknowlegement sent : 101

Acknowlegement received : 201

Connection Established for Data Transfer....

Sequence number : 1001
Acknowlegment sent : 2001

Sequence number : 2001
Acknowlegment sent : 3001

Timeout occurred. Resending the Acknowlegment.
Acknowlegment sent : 3001

Sequence number : 3001
Acknowlegment sent : 4001

Timeout occurred. Resending the Acknowlegment.
Acknowlegment sent : 4001

Sequence number : 4001
Acknowlegment sent : 5001

Sequence number : 5001
Acknowlegment sent : 6001
Connection closed by sender.
ashu@root:~/Desktop$ S
```

LEARNING OUTCOME

- Learnt about sequence numbers, segment numbers, acknowledgment numbers.
- Learnt to simulate the transmission of data packets with sequence number from sender and with acknowledgment number from the receiver using Socket programming.
- Learnt to set timer for the TCP transmission.
- Learnt how TCP ensures reliability using the numbers for packets and the timers.
- Learnt about retransmission mechanism in case of timeout.

README FILE

https://github.com/charu210703/CN_project_Q12_segment_acknowledgement_number/commit/951af82060edeed27d07a9de50bb57e879e85c50

GITHUB LINK

https://github.com/charu210703/CN_project_Q12_segment_acknowledgement_number