**Network Lab: Sliding Window Protocol**

V Venkataraman 106118106

server.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;

    int msg;

};

int main(int argc, char \*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);

    }

        int serv\_win = 3;

        int flag=1, s=0;

        int count=0;

        int i=0;

        int max=10;

        int num=0;

        if(argc>2)

        exit(0);

        printf(argv[1]);

        if(strcmp(argv[1],"1")==0)

        {

        //Go back N

        printf("Go back n\n");

        int count=0;

        while(count<max)

        {

            int right= s+serv\_win;

            for(int count=s;count<right;count++)

            {

                //Send all messages in window

                num++;

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

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

                Message->type = DATA;

                Message->len = 1;

                Message->msg=count;

                Message->seq = count;

                // if(num!=2)

                {

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

                    printf("MSG: %d\n",count);

                }

                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);

                        //move winddow left limit

                        s++;

                        if(count>max)

                        break;

                    }

                }

            }

        }

        }

        else if(strcmp(argv[1],"2")==0)

        //Selective Repeat

        {

            printf("Selective Repeat\n");

            int buffer[serv\_win];

            memset(buffer,0,sizeof(buffer));

            int num=0;

            int right= s+serv\_win;

            while(count<max)

            {

            for(int count=s;count<right;count++)

            {

                num++;

                if(count>max)

                break;

                //Send message only if not acknowledged

                if( count>=s && buffer[count-s]==0)

                {

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

                Message->type = DATA;

                Message->len = 1;

                Message->msg=count;

                Message->seq = count;

                    // if(num!=2)

                    {

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

                    printf("MSG: %d s=%d\n",count,s);

                    }

                }

                //Receive ACK

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

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

                {

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

                    if(Acknowledge->type == ACK && Acknowledge->seq >=s && Acknowledge->seq < s+serv\_win && buffer[Acknowledge->seq-s]==0)

                    {

                        sleep(1);

                        //Window element acknowledged

                        buffer[Acknowledge->seq-s]=1;

                    }

                }

                int flag=1;

                for(int p=0;p<serv\_win;p++)

                {

                    if(buffer[p]==0)

                    {

                        flag=-1;

                        break;

                    }

                }

                if(flag==1)

                {

                    s+=serv\_win;

                    right+=serv\_win;

                    memset(buffer,0,sizeof(buffer));

                }

            }

            }

        }

    close(new\_socket);

    return 0;

}

client.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;

    int msg;

};

int main(int argc, char \*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;

    }

            int s=0;

            char null[1]={'N'};

            int i=0;

            int count=0;

            int max=10;

            int recv\_win=3;

    if(argc>2)

        exit(0);

    printf(argv[1]);

    if(strcmp(argv[1],"1")==0)

    //Go Back n

    {

        printf("Goback n\n");

        int num=0;

        int temp;

            while(s<max)

            {

                num++;

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

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

                    {

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

                        {

                            printf("Recvd: %d Acc\n",Message->seq);

                            s++;

                            temp=s;

                        }

                        else

                        {

                            printf("Recvd: %d Disc expecting %d\n",Message->seq,s);

                            if(Message->seq<s)

                            {

                                temp=s;

                                s=Message->seq;

                            }

                        }

                    }

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

                Acknowledge->type = ACK;

                Acknowledge->len = 0;

                Acknowledge->msg=-1;

                Acknowledge->seq=s-1;

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

                if(temp!=s)

                s=temp;

            }

    }

    else if(strcmp(argv[1],"2")==0)

    {

    // Selective repeat

    printf("Selective Repeat\n");

        int ind=0;

        int temp;

        count=0;

        int buffer[recv\_win];

        memset(buffer,0,sizeof(buffer));

        //for(int count=s;count<s+recv\_win;s++)

        int left=0;

        int right=recv\_win-1;

        int num=0;

        while(left<max)

        {

                num++;

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

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

                    {

                        if(Message->type == DATA && Message->seq >= left && Message->seq <= right && buffer[Message->seq-left]==0)

                        {

                            printf("Recvd: %d Acc L:%d R:%d\n",Message->seq,left,right);

                            buffer[Message->seq-left]=1;

                            ind=Message->seq;

                            temp=ind;

                        }

                        else

                        {

                            printf("Recvd: %d Disc L:%d R:%d\n",Message->seq,left,right);

                            if(Message->seq<=left)

                            {

                                temp=ind;

                                ind=Message->seq;

                            }

                        }

                    }

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

                    Acknowledge->type = ACK;

                    Acknowledge->len = 0;

                    Acknowledge->msg=-1;

                    Acknowledge->seq=ind;

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

                    if(ind!=temp)

                        ind=temp;

                int flag=1;

                for(int p=left;p<=right;p++)

                {

                    if(buffer[p-left]==0)

                    {

                        flag=-1;

                        //printf("%d",p);

                        break;

                    }

                }

                if(flag==1)

                {

                    left+=recv\_win;

                    right+=recv\_win;

                    memset(buffer,0,sizeof(buffer));

                }

        }

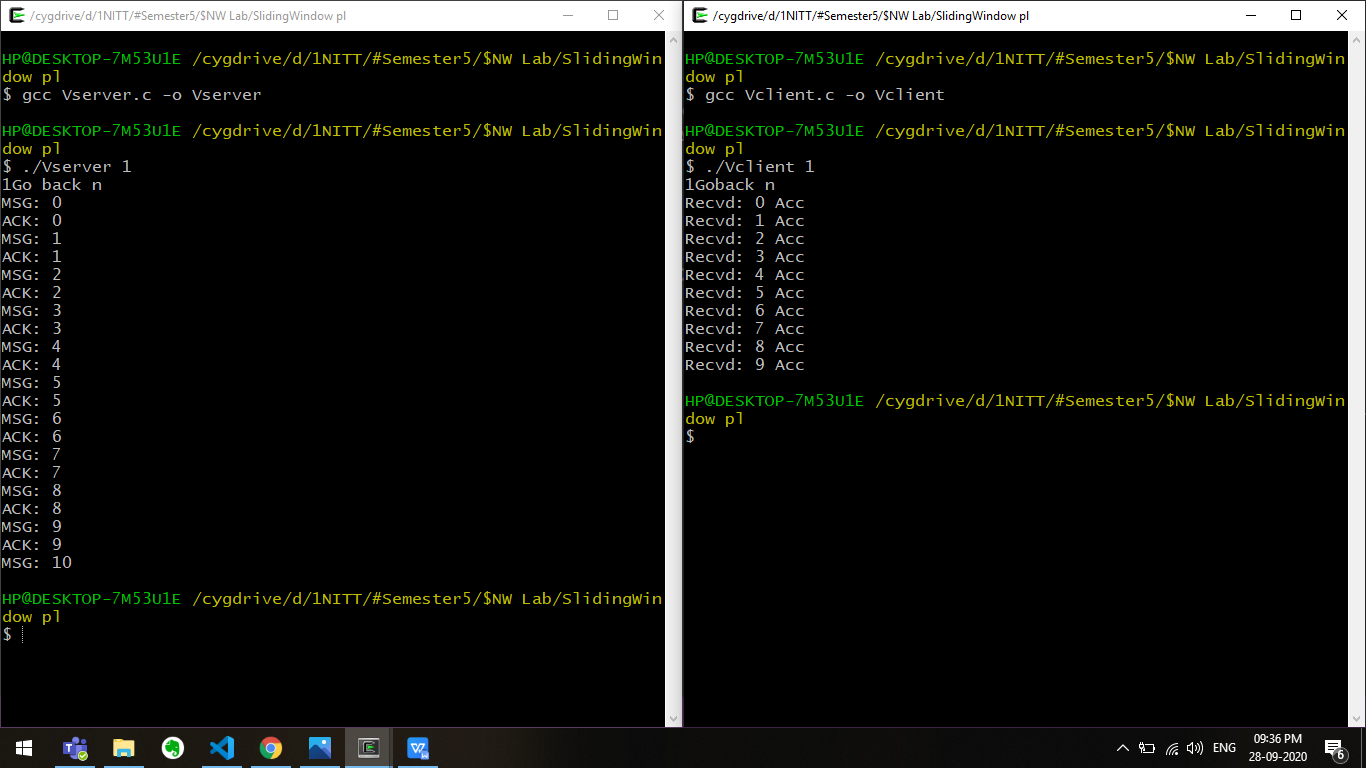
    }

    close(sock);

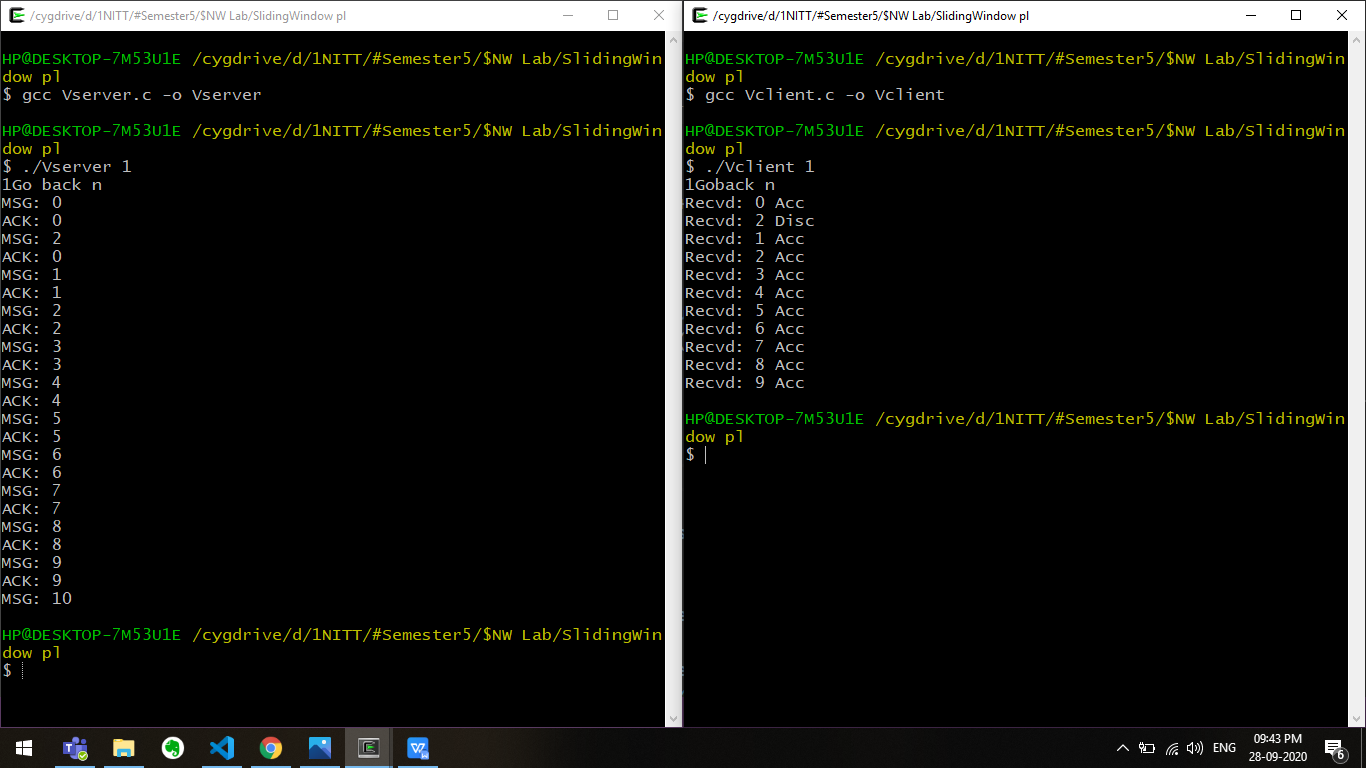
    return 0;

}

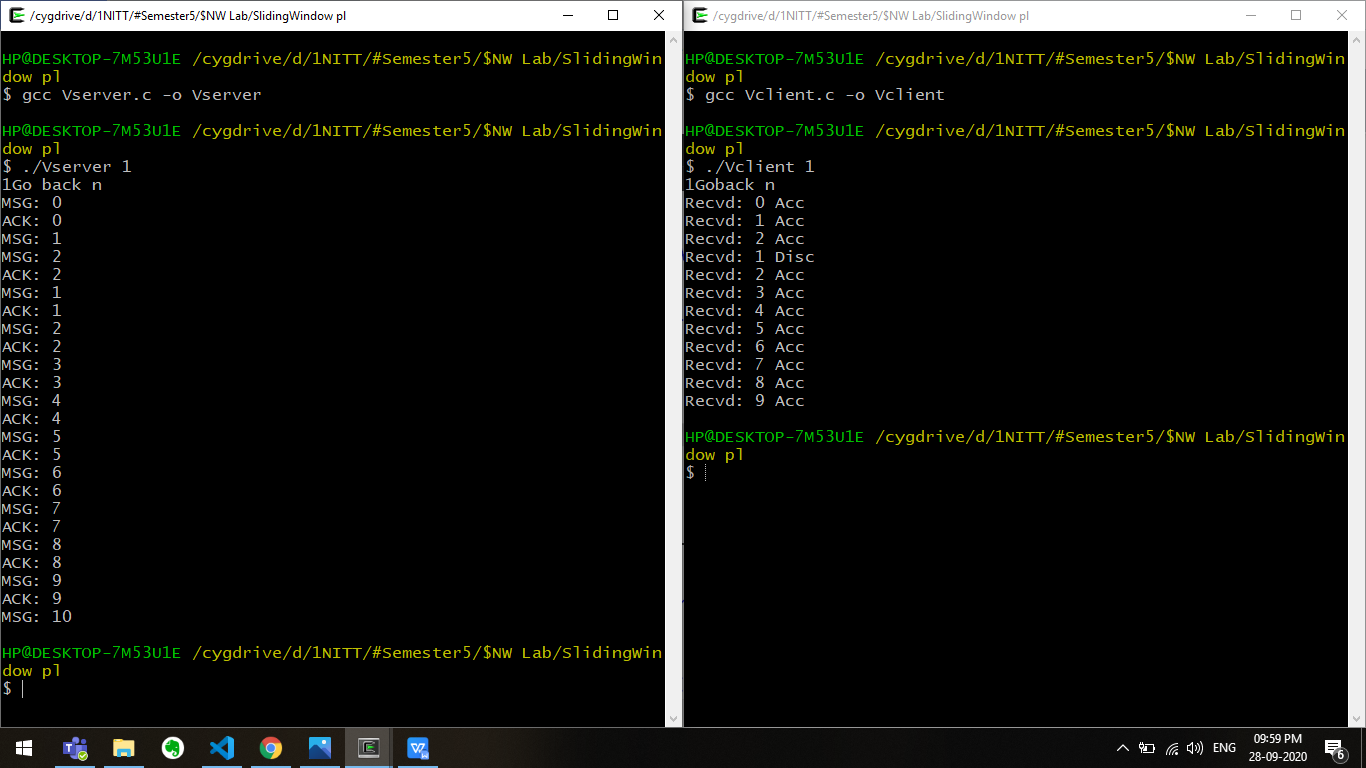
Go Back n Normal Transmission



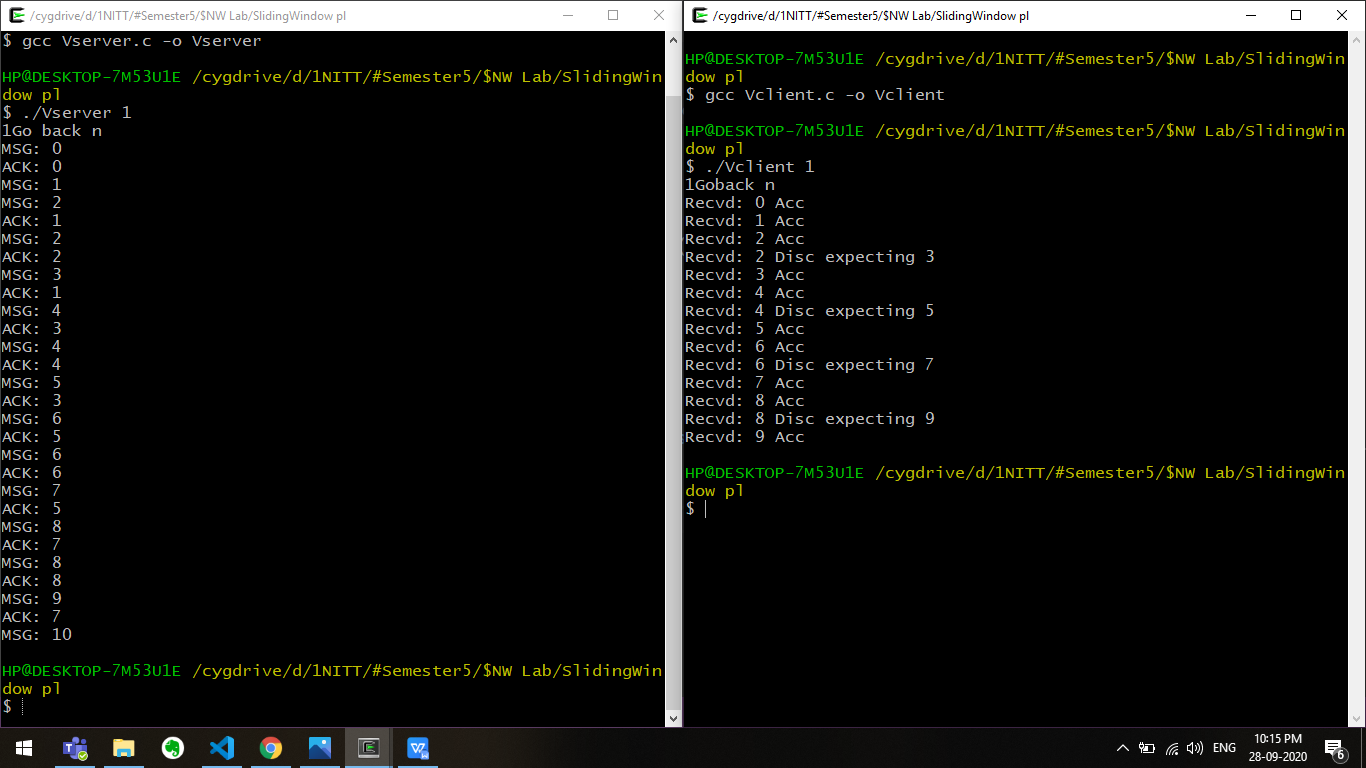
Go Back n Lost Packet(msg 1)



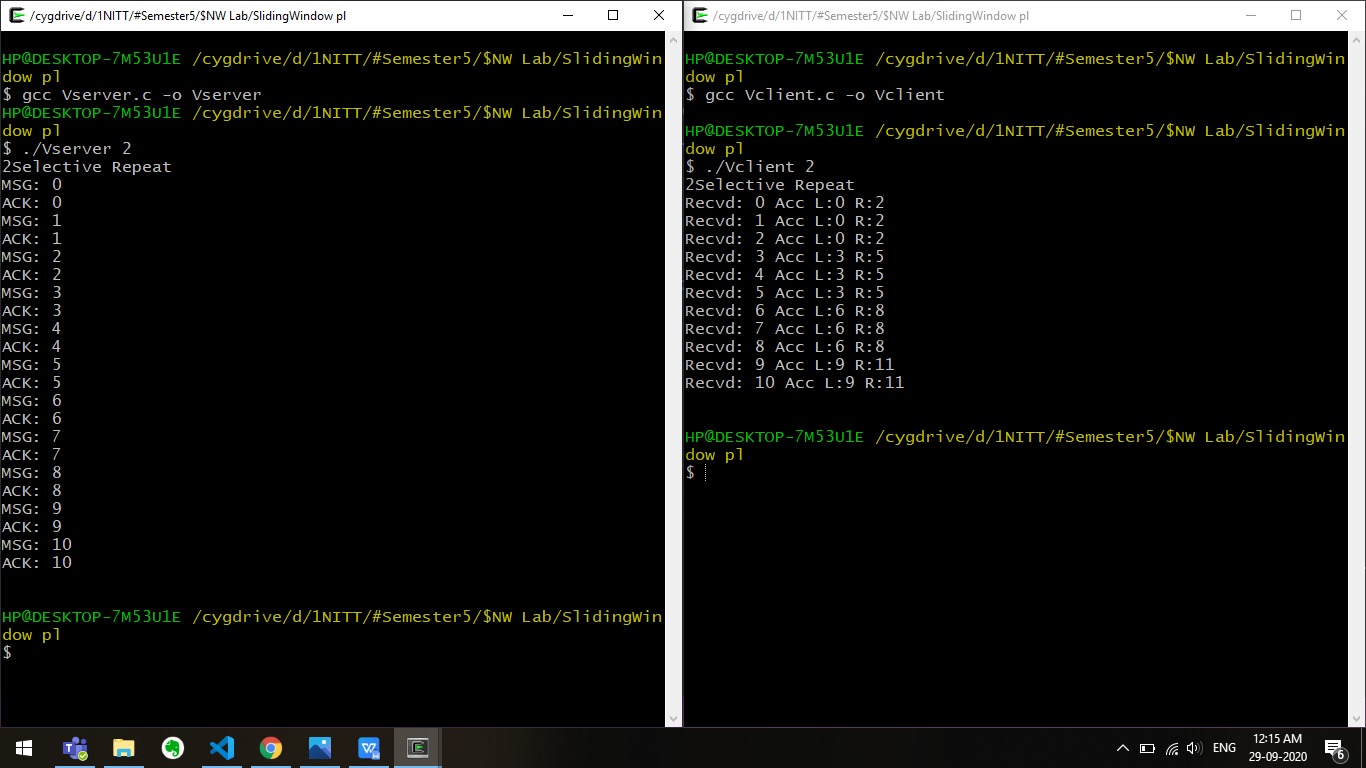
Go Back n Lost Acknowledgement(ack 1)



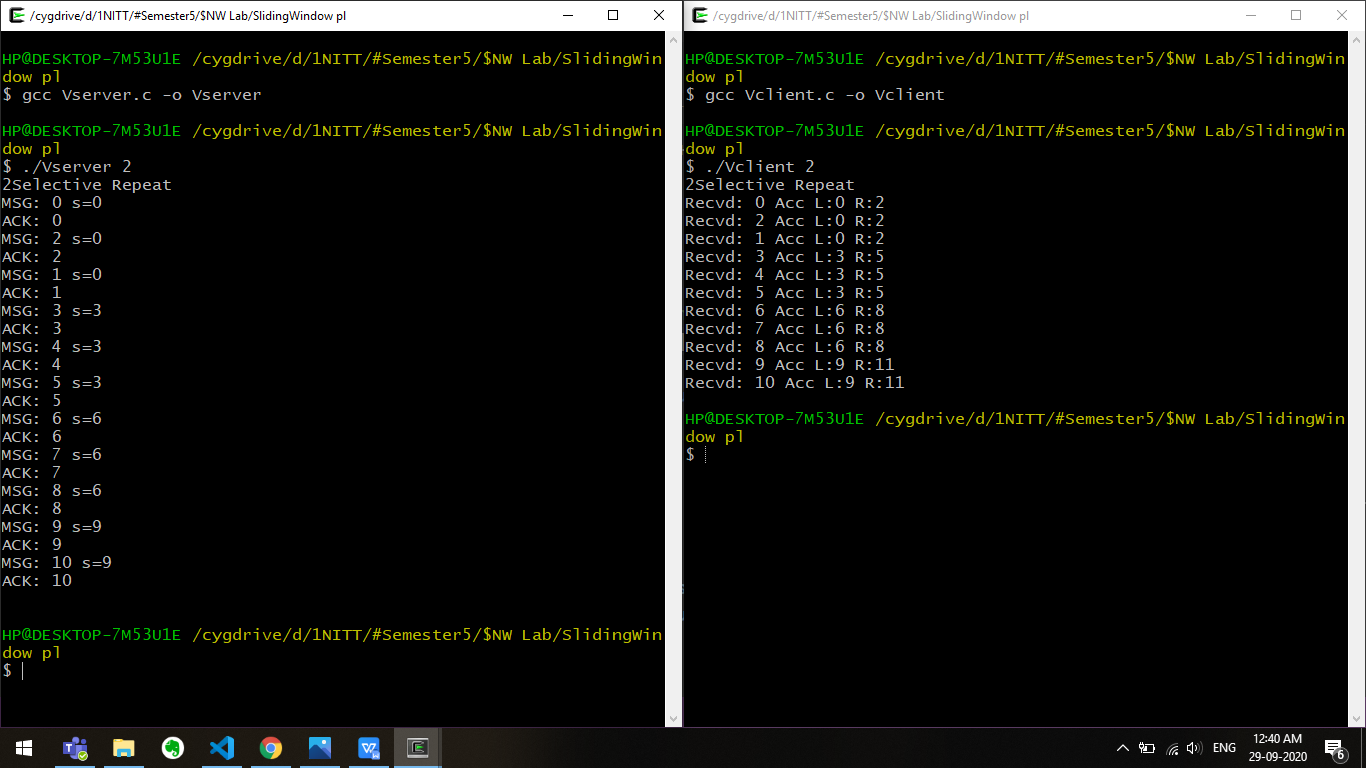
Go Back n Timeout (ack 1)



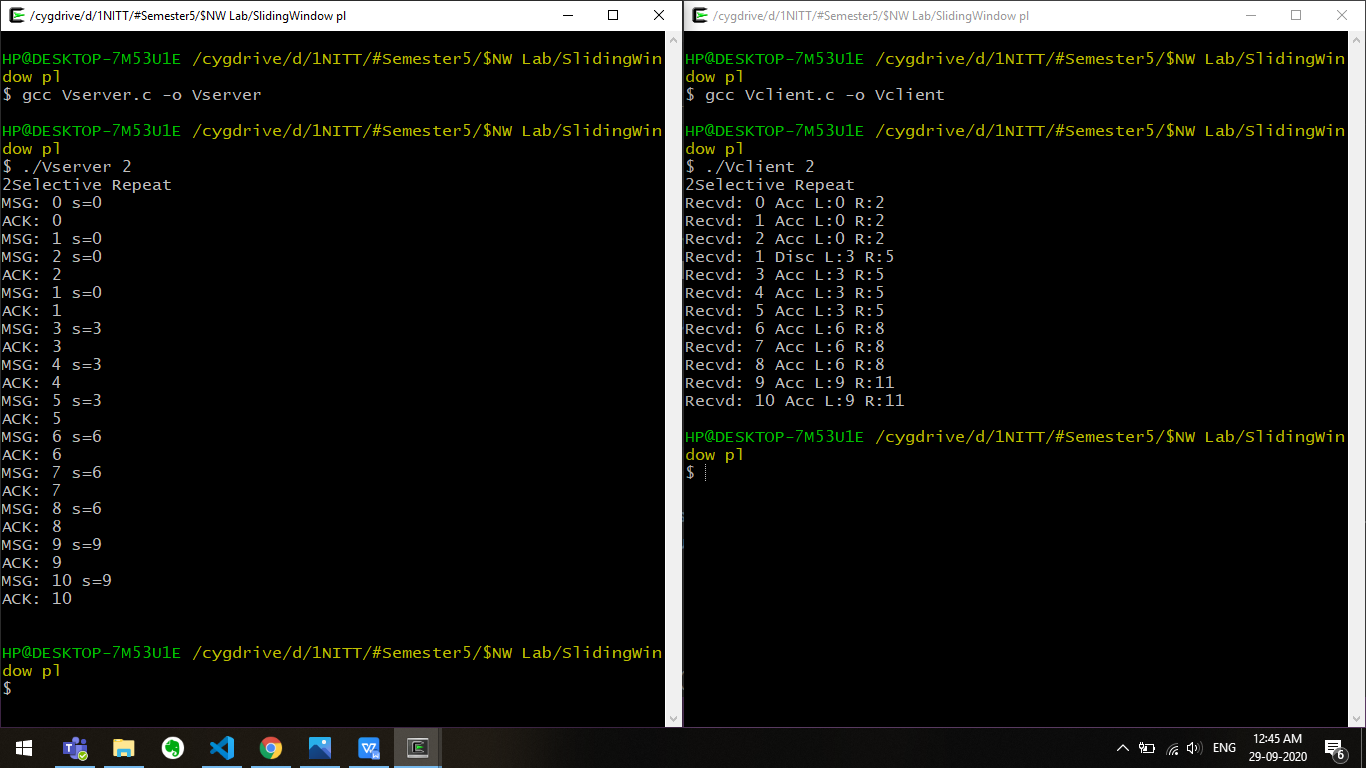
Selective Repeat Normal Transmission



Selective Repeat Lost Packet(msg 1)



Selective Repeat Lost Acknowledgement(ack 1)



Selective Repeat Timeout(ack 1)

