COMPUTER NETWORKS

Name: Vinayak Sethi **Roll No:** COE18B061

Assignment 8 ***Date: 30th September 2020***

FLOW CONTROL PROTOCOLS

Stop and Wait Protocol

It is the simplest flow control protocol.

It works under the following assumptions

- Communication channel is perfect.
- No errors occur during transmission.

How does it work?

- Sender sends a data packet to the receiver.
- Sender stops and waits for the acknowledgment for the sent packet from the receiver.
- Receiver receives and processes the data packet.
- Receiver sends an acknowledgment to the sender.
- After receiving the acknowledgement, the sender sends the next data packet to the receiver.

How code works?

Client Side

Initially we create a client socket, which connects to the server using **connect()** function. Here client is a sender.
Then the client sends a message to the server. At every iteration (an iteration per character of the string), the sender sends a variable called **reached_stringend** to notify the receiver whether we have reached the

string end or not. A character is sent and acknowledgement is received.If acknowledge is ‘0’ the character is sent again.This happens until the string end is reached.

Server Side
Initially a server socket is created, which binds with the client, listens to its request and accepts. Here the server is the receiver.
At every iteration until we reach the end of the string, we receive a character from the sender and send an acknowledgement corresponding to it **(1 -> successfully received, 0 -> unsuccessful)**. After receiving the complete message, it displays the message.

Codes

Filename: Client.c

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<unistd.h>
#include<string.h>

int main()
{
   int client_socket,sin_size;
   struct sockaddr_in server_address;

   //create a socket
   client_socket = socket(AF_INET,SOCK_STREAM,0);
   if(client_socket == -1)
   {
       printf("\nSocket Creation Failure\n");
       exit(EXIT_FAILURE);
   }

   //specify an address for the socket
```

```c
   server_address.sin_family = AF_INET;
   server_address.sin_port = htons(9009);
   server_address.sin_addr.s_addr = INADDR_ANY;

   sin_size = sizeof(struct sockaddr_in);

   //connect to server
   if(connect(client_socket,(struct sockaddr *)&server_address, sin_size) == 0)
       printf("Connect Successful\n");

   printf("****************************STOP AND WAIT SENDER
SIDE***************************\n");

   char message[150];
   printf("\nEnter the message to send: ");
   gets(message);

   char buffer[2]; //to send the data packet
   buffer[0] = '1';
   buffer[1] = '\0';

   char acknowledge[2]; //successful acknowledgement received or not
   acknowledge[0] = '1';
   acknowledge[1] = '\0';

   char reached_stringend[2]; //reached end of string or not
   reached_stringend[0] = '0';
   reached_stringend[1] = '\0';

   //sending packet byte by byte
   for(int i=0; i<strlen(message); i++)
   {
       printf("\nSending data -> %c", message[i]);
       buffer[0] = message[i];
       reached_stringend[0] = '0';

       send(client_socket, reached_stringend, sizeof(reached_stringend), 0);
//check if reached string end or not
       send(client_socket, buffer, sizeof(buffer), 0); //sending the data
packet
       recv(client_socket, acknowledge, sizeof(acknowledge), 0); //receiving
the acknowledgement
```

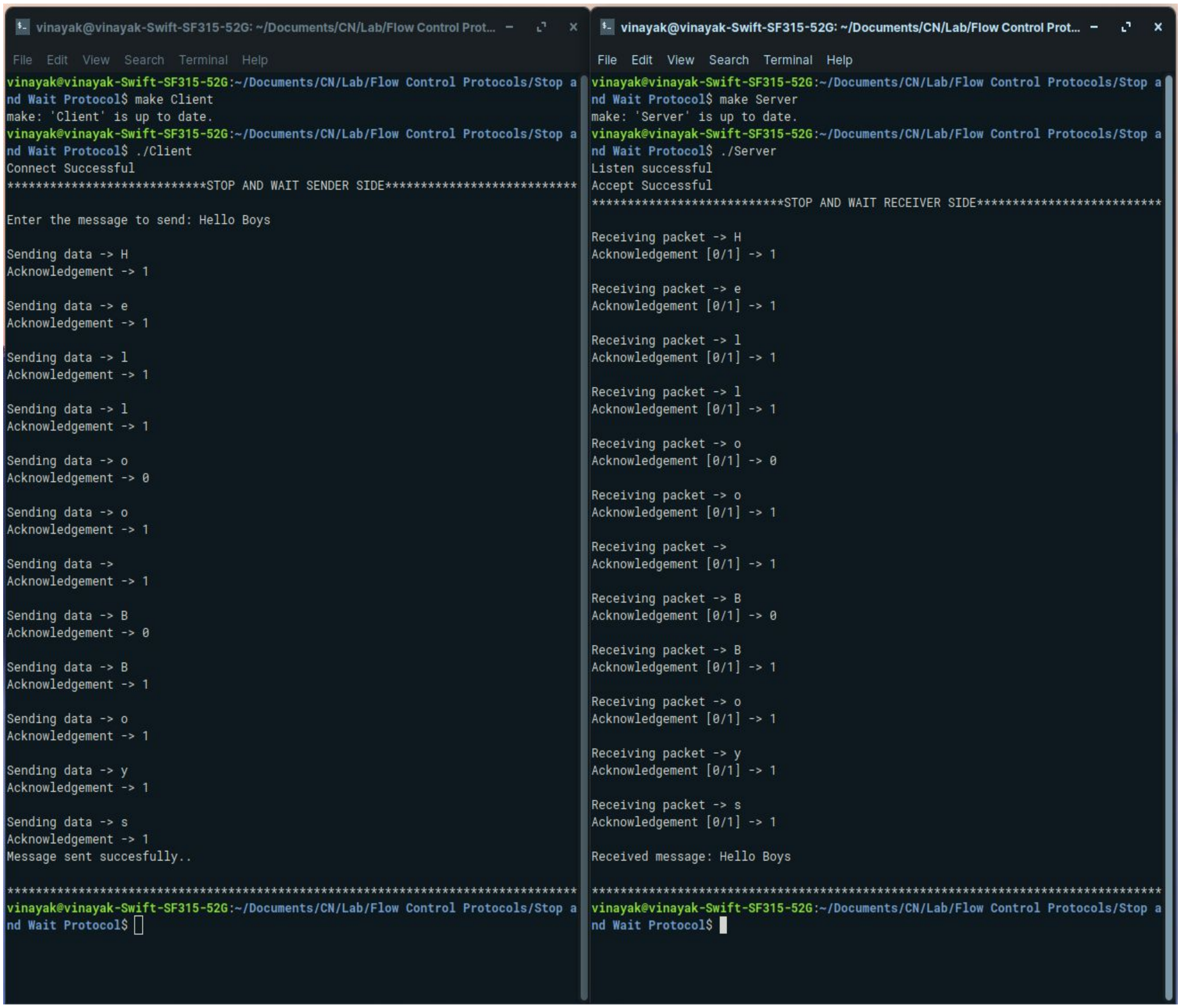
```
        printf("\nAcknowledgement -> %s\n", acknowledge);

        if(acknowledge[0] == '0') //Failed Acknowledgement
            i--;
    }

    reached_stringend[0] = '1'; //end of string
    send(client_socket, reached_stringend, sizeof(reached_stringend), 0);

    printf("Message sent succesfully..\n");

printf("\n*********************************************************************
***********\n");

    //close the socket
    close(client_socket);
    return 0;
}
```

Filename: Server.c

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<unistd.h>
#include<string.h>

int main()
{
    int server_socket,client_socket,sin_size;
    struct sockaddr_in server_address, client_address;

    //create a socket
    server_socket = socket(AF_INET,SOCK_STREAM,0);
    if(server_socket == -1)
    {
        printf("\nSocket Creation Failure\n");
        exit(EXIT_FAILURE);
    }

    //specify an address for the socket
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(9009);
    server_address.sin_addr.s_addr = INADDR_ANY;

    //bind with the client
    if( bind(server_socket, (const struct sockaddr *)&server_address,
sizeof(server_address)) < 0)
    {
        printf("Could not bind to Client\n");
        exit(EXIT_FAILURE);
    }

    //listen to the incoming client request
    if(listen(server_socket, 10) == 0)
        printf("Listen successful\n");

    //accept a connection request from client
    sin_size = sizeof(struct sockaddr_in);
    if((client_socket = accept(server_socket, (struct sockaddr
*)&client_address,&sin_size)) > 0)
        printf("Accept Successful\n");


    printf("***************************STOP AND WAIT RECEIVER
SIDE**************************\n");
    char message[150];

    char acknowledge[2]; //successful acknowledgement sent or not
    acknowledge[0] = '1';
    acknowledge[1] = '\0';

    char reached_stringend[2], buffer[2];

    int message_len = 0;
    recv(client_socket, reached_stringend, sizeof(reached_stringend), 0);

    //Receiving Packet byte by byte
    while(reached_stringend[0] == '0')
```

```c
    {
        recv(client_socket, buffer, sizeof(buffer), 0); //receiving packet

        printf("\nReceiving packet -> %s", buffer);
        printf("\nAcknowledgement [0/1] -> ");
        scanf("%s", acknowledge);

        send(client_socket, acknowledge, sizeof(acknowledge), 0); //send the
acknowledgement corresponding to packet
        if(acknowledge[0] == '1') //Positive acknowlegment
            message[message_len++] = buffer[0];

        recv(client_socket, reached_stringend, sizeof(reached_stringend), 0);
        if(reached_stringend[0] == '1')
            break;
    }

    message[message_len++] = '\0'; //end of message
    printf("\nReceived message: ");
    for(int i=0; i<message_len; i++)
        printf("%c", message[i]);

    printf("\n");


printf("\n*********************************************************************
***********\n");

    //close the socket
    close(client_socket);
    close(server_socket);

    return 0;
}
```

Output:



```
vinayak@vinayak-Swift-SF315-52G:~/Documents/CN/Lab/Flow Control Protocols/Stop a
nd Wait Protocol$ make Client
make: 'Client' is up to date.
vinayak@vinayak-Swift-SF315-52G:~/Documents/CN/Lab/Flow Control Protocols/Stop a
nd Wait Protocol$ ./Client
Connect Successful
****************************STOP AND WAIT SENDER SIDE***************************

Enter the message to send: Hello Boys

Sending data -> H
Acknowledgement -> 1

Sending data -> e
Acknowledgement -> 1

Sending data -> l
Acknowledgement -> 1

Sending data -> l
Acknowledgement -> 1

Sending data -> o
Acknowledgement -> 0

Sending data -> o
Acknowledgement -> 1

Sending data ->
Acknowledgement -> 1

Sending data -> B
Acknowledgement -> 0

Sending data -> B
Acknowledgement -> 1

Sending data -> o
Acknowledgement -> 1

Sending data -> y
Acknowledgement -> 1

Sending data -> s
Acknowledgement -> 1
Message sent succesfully..

********************************************************************************
vinayak@vinayak-Swift-SF315-52G:~/Documents/CN/Lab/Flow Control Protocols/Stop a
nd Wait Protocol$
```

```
vinayak@vinayak-Swift-SF315-52G:~/Documents/CN/Lab/Flow Control Protocols/Stop a
nd Wait Protocol$ make Server
make: 'Server' is up to date.
vinayak@vinayak-Swift-SF315-52G:~/Documents/CN/Lab/Flow Control Protocols/Stop a
nd Wait Protocol$ ./Server
Listen successful
Accept Successful
***************************STOP AND WAIT RECEIVER SIDE**************************

Receiving packet -> H
Acknowledgement [0/1] -> 1

Receiving packet -> e
Acknowledgement [0/1] -> 1

Receiving packet -> l
Acknowledgement [0/1] -> 1

Receiving packet -> l
Acknowledgement [0/1] -> 1

Receiving packet -> o
Acknowledgement [0/1] -> 0

Receiving packet -> o
Acknowledgement [0/1] -> 1

Receiving packet ->
Acknowledgement [0/1] -> 1

Receiving packet -> B
Acknowledgement [0/1] -> 0

Receiving packet -> B
Acknowledgement [0/1] -> 1

Receiving packet -> o
Acknowledgement [0/1] -> 1

Receiving packet -> y
Acknowledgement [0/1] -> 1

Receiving packet -> s
Acknowledgement [0/1] -> 1

Received message: Hello Boys

********************************************************************************
vinayak@vinayak-Swift-SF315-52G:~/Documents/CN/Lab/Flow Control Protocols/Stop a
nd Wait Protocol$
```

Sliding Window Protocol

The two well known implementations of sliding window protocol are-

- Go back N Protocol
- Selective Repeat Protocol

So, I have implemented the Selective Repeat Protocol.

Why Selective Repeat Protocol?
The go-back-n protocol works well if errors are less, but if the line is poor it wastes a lot of bandwidth on retransmitted frames. An alternative strategy, the selective repeat protocol, is to allow the receiver to accept and buffer the frames following a damaged or lost one.
Selective Repeat attempts to retransmit only those packets that are actually lost (due to errors) :

- Receiver must be able to accept packets out of order.
- Since the receiver must release packets to a higher layer in order, the receiver must be able to buffer some packets.

How does code work?
The main condition for this protocol is that **Sender window size == Receiver window size.**
The sender sends the message in the given window size and waits for the acknowledgement from the receiver side for the window sent. If the sender receives the negative acknowledgment for a given character, then the sender sends the new window from that character to the receiver.
If the timeout happens from the receiver side while sending the acknowledgement the sender resends the complete window, and this happens until the receiver receives the complete message successfully.

Codes

Filename: Client.c

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<unistd.h>
#include<string.h>
#include<time.h>
```

```
double timeout = 3; //timeout time of acknowledgement

int main()
{
    int client_socket,sin_size;
    struct sockaddr_in server_address;

    //create a socket
    client_socket = socket(AF_INET,SOCK_STREAM,0);
    if(client_socket == -1)
    {
        printf("\nSocket Creation Failure\n");
        exit(EXIT_FAILURE);
    }

    //specify an address for the socket
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(9009);
    server_address.sin_addr.s_addr = INADDR_ANY;

    sin_size = sizeof(struct sockaddr_in);

    //connect to server
    if(connect(client_socket,(struct sockaddr *)&server_address, sin_size) == 0)
        printf("Connect Successful\n");

    printf("**************************SELECTIVE REPEAT SENDER
SIDE**************************\n");

    int windowsize;
    printf("Enter the window size: ");
    scanf("%d", &windowsize);

    int win_size[] = {windowsize};
    send(client_socket, win_size, sizeof(win_size), 0); //send the window size
to the receiving end

    char ch;
    scanf("%c", &ch);
    char message[150];
    printf("\nEnter the message to send: ");
```

```
   gets(message);

   char buffer[2]; //to send the data packet
   buffer[0] = '1';
   buffer[1] = '\0';

   char acknowledge[2]; //successful acknowledgement received or not
   acknowledge[0] = '1';
   acknowledge[1] = '\0';

   char reached_stringend[2]; //reached end of string or not
   reached_stringend[0] = '0';
   reached_stringend[1] = '\0';

   //sending continuous packet which can fit in a window
   for(int i=0; i<strlen(message); i++)
   {
       send(client_socket, reached_stringend, sizeof(reached_stringend), 0);
//check if reached string end or not

       for(int j=0; j<windowsize; j++) //sending packets for a given window
size
       {
           printf("\nSending data -> %c", message[i+j]);
           buffer[0] = message[i+j];
           send(client_socket, buffer, sizeof(buffer), 0); //sending the data
packet
       }

       int ack_check = -1;
       printf("\n");

       for(int j=0; j<windowsize; j++)
       {

           recv(client_socket, acknowledge, sizeof(acknowledge), 0);
//receiving the acknowledgement

           double time_taken[1]; //time taken by receiver to send the
acknowledgement
           recv(client_socket, time_taken, sizeof(time_taken), 0);
           if(timeout < time_taken[0])
               printf("Acknowledgement -> TIMEOUT");
           else
               printf("Acknowledgement -> %s", acknowledge);

           if(acknowledge[0] == '0' || timeout < time_taken[0]) //Failed
Acknowledgement
           {
               ack_check = j;
               j = windowsize;
           }
           printf("\n");
       }

       if(ack_check == -1)
           i = i + windowsize - 1; // no failed acknowedgement received
       else
           i = i + ack_check - 1; //any failed acknowledgement received
   }

   reached_stringend[0] = '1'; //end of string
   send(client_socket, reached_stringend, sizeof(reached_stringend), 0);

   printf("\nMessage sent succesfully..\n");

printf("\n********************************************************************
***********\n");

   //close the socket
   close(client_socket);
   return 0;
}
```

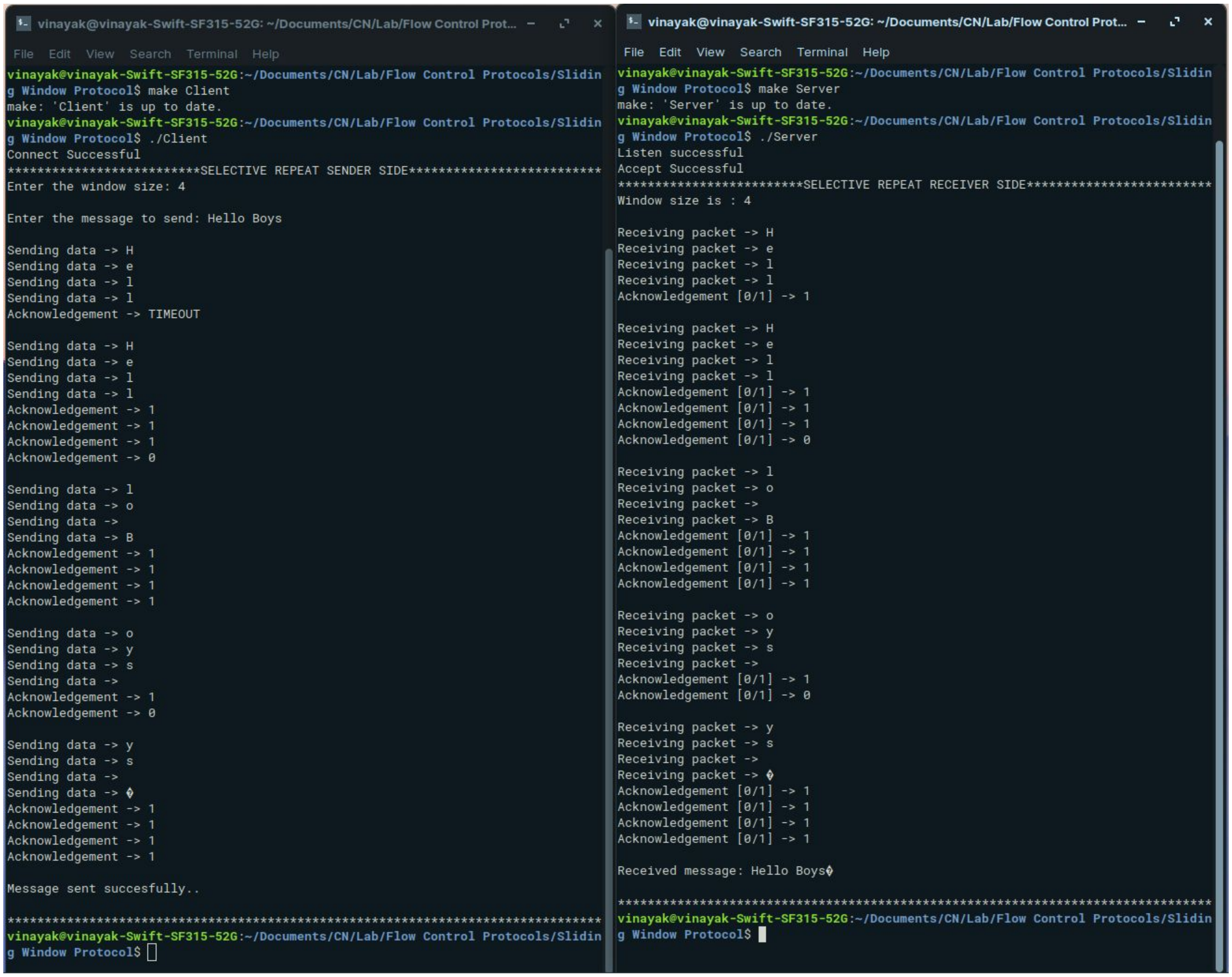
Filename: Server.c

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<unistd.h>
```

```
#include<string.h>
#include<sys/time.h>

double timeout = 3; //timeout time of acknowledgement

int main()
{
   int server_socket,client_socket,sin_size;
   struct sockaddr_in server_address, client_address;
   struct timeval start, end;

   //create a socket
   server_socket = socket(AF_INET,SOCK_STREAM,0);
   if(server_socket == -1)
   {
       printf("\nSocket Creation Failure\n");
       exit(EXIT_FAILURE);
   }

   //specify an address for the socket
   server_address.sin_family = AF_INET;
   server_address.sin_port = htons(9009);
   server_address.sin_addr.s_addr = INADDR_ANY;

   //bind with the client
   if( bind(server_socket, (const struct sockaddr *)&server_address,
sizeof(server_address)) < 0)
   {
       printf("Could not bind to Client\n");
       exit(EXIT_FAILURE);
   }

   //listen to the incoming client request
   if(listen(server_socket, 10) == 0)
       printf("Listen successful\n");

   //accept a connection request from client
   sin_size = sizeof(struct sockaddr_in);
   if((client_socket = accept(server_socket, (struct sockaddr
*)&client_address,&sin_size)) > 0)
       printf("Accept Successful\n");
   printf("**************************SELECTIVE REPEAT RECEIVER
SIDE*************************\n");
   char message[150];

   char acknowledge[2]; //successful acknowledgement sent or not
   acknowledge[0] = '1';
   acknowledge[1] = '\0';

   char reached_stringend[2], buffer[2];

   int win_size[1];
   recv(client_socket, win_size, sizeof(win_size), 0);//receiving the window
size from sender

   int windowsize = 0;
   windowsize = win_size[0];
   printf("Window size is : %d\n", windowsize);

   int message_len = 0;
   recv(client_socket, reached_stringend, sizeof(reached_stringend), 0); //if
reached string end or not

   //sending continuous acknowledgement for received packets in a given window
size
   while(reached_stringend[0] == '0')
   {
       for(int j=0; j<windowsize; j++)//receiving packets for a given window
size
       {
           recv(client_socket, buffer, sizeof(buffer), 0);
           printf("\nReceiving packet -> %s", buffer);
           message[message_len++] = buffer[0];
       }

       printf("\n");
       for(int j=0; j<windowsize; j++) //sending acknowledgements for received
packets
       {
           printf("Acknowledgement [0/1] -> ");
           gettimeofday(&start, NULL); //starting time
           scanf("%s", acknowledge);
           send(client_socket, acknowledge, sizeof(acknowledge), 0); //send the
acknowledgement corresponding to packet

           gettimeofday(&end, NULL); //ending time
           double duration = (double)(end.tv_usec - start.tv_usec) / 1000000 +
(double)(end.tv_sec - start.tv_sec);

           double time_taken[] = {duration};

           send(client_socket, time_taken, sizeof(time_taken), 0); //send the
time taken to give acknowledgement for received packet

           if(acknowledge[0] == '0' || timeout < duration) //Negative
acknowlegment
           {
               message_len = message_len - windowsize + j;
               j = windowsize;
           }
       }

       recv(client_socket, reached_stringend, sizeof(reached_stringend), 0);
       if(reached_stringend[0] == '1')
           break;
   }

   message[message_len++] = '\0'; //end of message

   printf("\nReceived message: ");
   for(int i=0; i<message_len; i++)
       printf("%c", message[i]);

   printf("\n");


printf("\n*********************************************************************
***********\n");

   //close the socket
   close(client_socket);
   close(server_socket);
```

```
    return 0;
}
```

Output:



```
vinayak@vinayak-Swift-SF315-52G: ~/Documents/CN/Lab/Flow Control Prot...
vinayak@vinayak-Swift-SF315-52G:~/Documents/CN/Lab/Flow Control Protocols/Slidin
g Window Protocol$ make Client
make: 'Client' is up to date.
vinayak@vinayak-Swift-SF315-52G:~/Documents/CN/Lab/Flow Control Protocols/Slidin
g Window Protocol$ ./Client
Connect Successful
**************************SELECTIVE REPEAT SENDER SIDE***************************
Enter the window size: 4

Enter the message to send: Hello Boys

Sending data -> H
Sending data -> e
Sending data -> l
Sending data -> l
Acknowledgement -> TIMEOUT

Sending data -> H
Sending data -> e
Sending data -> l
Sending data -> l
Acknowledgement -> 1
Acknowledgement -> 1
Acknowledgement -> 1
Acknowledgement -> 0

Sending data -> l
Sending data -> o
Sending data ->
Sending data -> B
Acknowledgement -> 1
Acknowledgement -> 1
Acknowledgement -> 1
Acknowledgement -> 1

Sending data -> o
Sending data -> y
Sending data -> s
Sending data ->
Acknowledgement -> 1
Acknowledgement -> 0

Sending data -> y
Sending data -> s
Sending data ->
Sending data -> �
Acknowledgement -> 1
Acknowledgement -> 1
Acknowledgement -> 1
Acknowledgement -> 1

Message sent succesfully..

*********************************************************************************
vinayak@vinayak-Swift-SF315-52G:~/Documents/CN/Lab/Flow Control Protocols/Slidin
g Window Protocol$
```

```
vinayak@vinayak-Swift-SF315-52G: ~/Documents/CN/Lab/Flow Control Prot...
vinayak@vinayak-Swift-SF315-52G:~/Documents/CN/Lab/Flow Control Protocols/Slidin
g Window Protocol$ make Server
make: 'Server' is up to date.
vinayak@vinayak-Swift-SF315-52G:~/Documents/CN/Lab/Flow Control Protocols/Slidin
g Window Protocol$ ./Server
Listen successful
Accept Successful
*************************SELECTIVE REPEAT RECEIVER SIDE*************************
Window size is : 4

Receiving packet -> H
Receiving packet -> e
Receiving packet -> l
Receiving packet -> l
Acknowledgement [0/1] -> 1

Receiving packet -> H
Receiving packet -> e
Receiving packet -> l
Receiving packet -> l
Acknowledgement [0/1] -> 1
Acknowledgement [0/1] -> 1
Acknowledgement [0/1] -> 1
Acknowledgement [0/1] -> 0

Receiving packet -> l
Receiving packet -> o
Receiving packet ->
Receiving packet -> B
Acknowledgement [0/1] -> 1
Acknowledgement [0/1] -> 1
Acknowledgement [0/1] -> 1
Acknowledgement [0/1] -> 1

Receiving packet -> o
Receiving packet -> y
Receiving packet -> s
Receiving packet ->
Acknowledgement [0/1] -> 1
Acknowledgement [0/1] -> 0

Receiving packet -> y
Receiving packet -> s
Receiving packet ->
Receiving packet -> �
Acknowledgement [0/1] -> 1
Acknowledgement [0/1] -> 1
Acknowledgement [0/1] -> 1
Acknowledgement [0/1] -> 1

Received message: Hello Boys�

*********************************************************************************
vinayak@vinayak-Swift-SF315-52G:~/Documents/CN/Lab/Flow Control Protocols/Slidin
g Window Protocol$
```