CS5310.251/252, Spring 2016, Programming Project Simulation of the Ethernet

Submitted by: Kommuru Jaya Naga Bhavana.

(A04711981)

Project Aim and Objective:

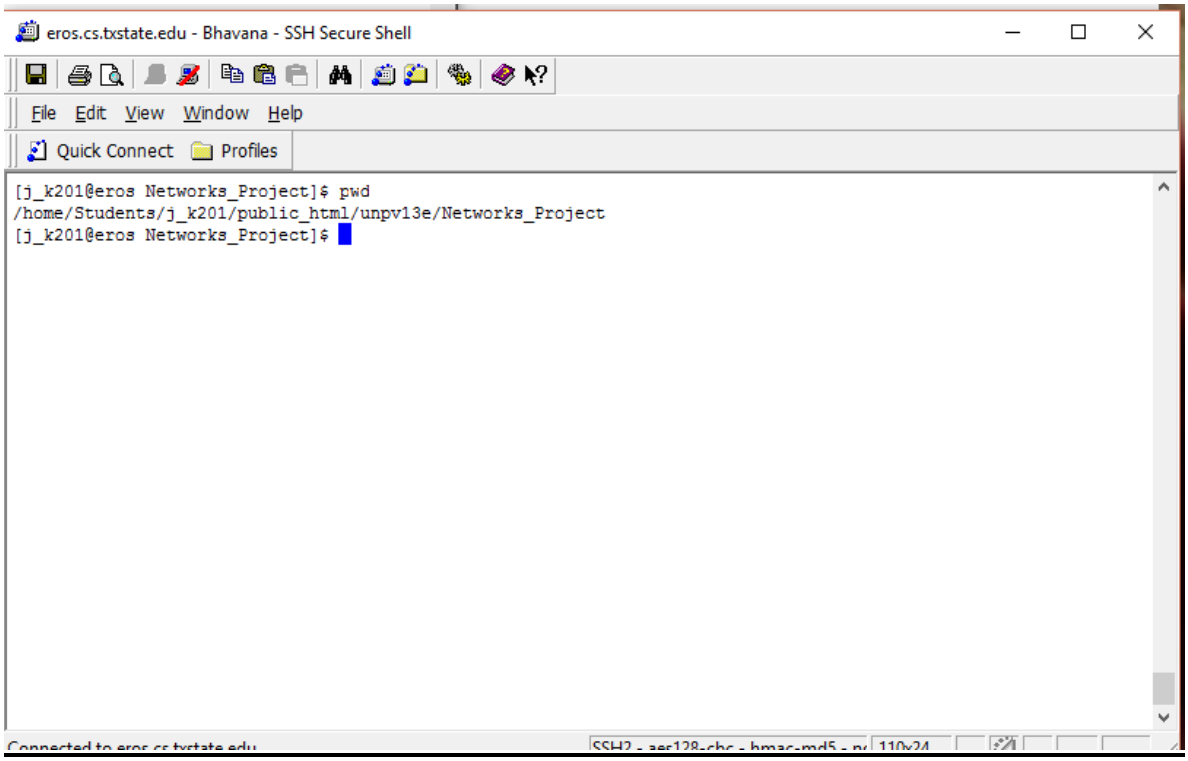
This project is to simulate the classical Ethernet. The main aim of the project is to practice basic socket API programming through a client/server application. It is a simulation in the sense that the Ethernet is simulated by multiple processes on multiple machines. Each station in the Ethernet is simulated by a process running on one of the workstations and the common bus is also simulated by a process.

Implementation:

The project is implemented using C programming language. The project contains a program 'CommunicationBusProcess.c' that implements the functionalities of a Communication Bus process and it acts as server in the client/server implementation. It contains 10 station process 'Station_Process1.txt', 'Station_Process2.txt', 'Station_Process3.txt', 'Station_Process4.txt', 'Station_Process5.txt', 'Station_Process6.txt', 'Station_Process7.txt', 'Station_Process8.txt', 'Station_Process9.txt', 'Station_Process10.txt'' which acts as input files. Hence we can access at most 10 station process for the communication bus process. TCP.IP protocol is chosen as the underlying protocol of the communication in this project.

All the Program files are stored at the path:

/home/Students/j_k201/public_html/unpv13e/Networks_Project.



List of Files :

Program files:

CommunicationBusProcess.c

StationProcess.c

Simulation input files for Station Process:

station_process1.txt

station_process2.txt

station_process3.txt

station_process4.txt

station_process5.txt

station_process6.txt

station_process7.txt

station_process8.txt

station_process9.txt

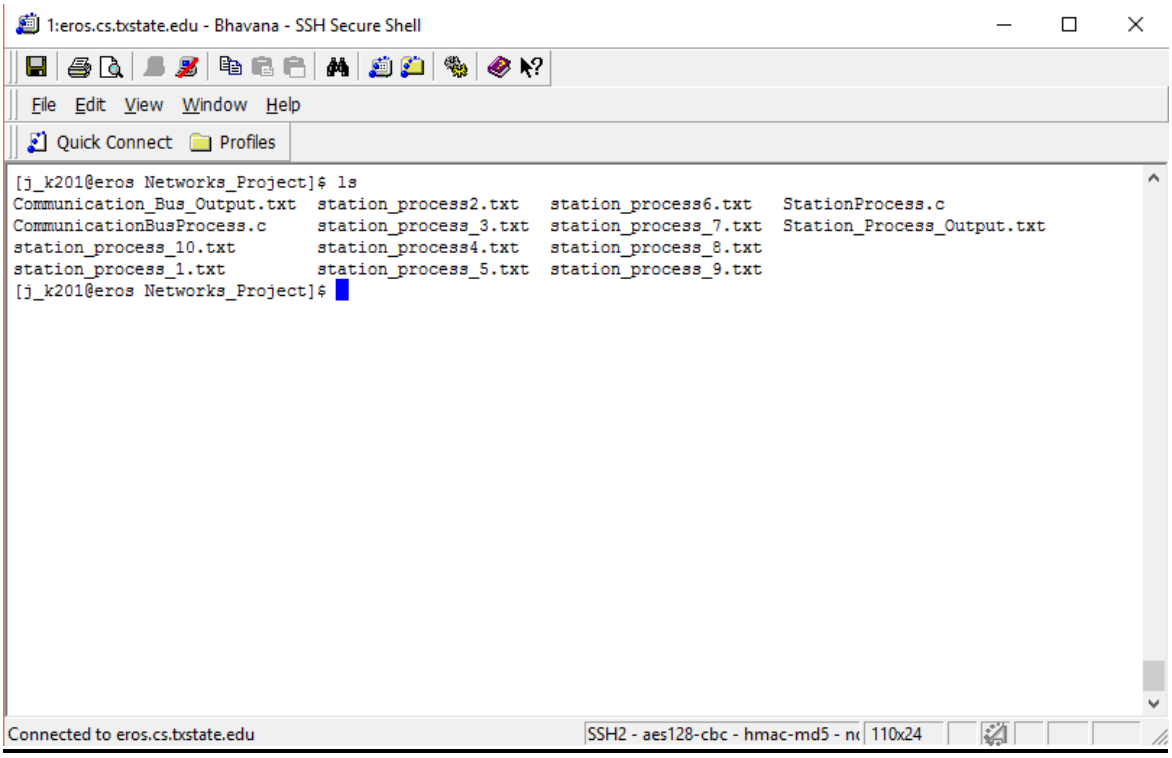
station_process10.txt

Output log file of CommunicationBusProcess:

Communication_Bus_Output.txt

Output log file of StationProcess:

Station_Process_Output.txt



Communication Bus Process (CommunicationBusProcess.c):

- The main duty of CommunicationBusProcess is to function as a single communication bus shared by all stations in an Ethernet.
- A data structure named **sockaddr_in** is used to store the addresses of the client and server.
- **sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) :** The function socket creates a socket. AF_INET specifies the namespace, SOCK_STREAM specifies the communication style and IPPROTO_TCP is the protocol. The return value from socket is the file descriptor for the new socket named sockfd.
- **bind(sockfd, (struct sockaddr*) (&serv_addr), sizeof(serv_addr)) :** The bind function assigns an address to the socket sockfd. The serv_addr and sizeof(serv_addr) arguments specify the address. The return value is 0 on success and -1 on failure.
- **listen(sockfd, 10) :** The listen function enables the socket sockfd to accept connections, thus making it a server socket. The argument 10 specifies the length of the queue for pending connections.

```
#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <sys/select.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <signal.h>


#define bool int

#define true 1

#define false 0


//File pointer to open and close the output log file.

FILE *file_pointer;
```

```
//variable port_no is used to store the port number given by the user.
int port_no;
int sock_fd;

//an array of stations that are connected.
bool connectedStations_ar[10];
bool stations_coll_ar[10];

//exits if there is any error
void error(const char *message)
{
        perror(message);
        exit(1);
}

//Is_there_a_Collision function checks if there is a collision
int Is_there_a_Collision(int station)
{
        int collision_occured = 0;
        for (int i = 0; i < 10; i++)
        {
      //if the current station is not equal to the array index
                if ((station != i))
      {
        //if the station with index number is already connected to the CommunicationBusProcess
        if(connectedStations_ar[i] == 1)
                {
                        collision_occured = 1;
```

```
                                        stations_coll_ar[i] = 1;
                }
                        }
                }
                return collision_occured;
}

//main function
int main(int argc, char * argv[])
{
   if (argc != 2)
        {
                printf("\n Please enter : ./CommunicationBusProc <server port number>");
                exit(0);
        }

        port_no = atoi(argv[1]);
        file_pointer = fopen("Communication_Bus_Output.txt" , "w");

   if(file_pointer == NULL)
        {
                perror("Error in opening output file");
                return(-1);
   }

   //Initializing the connectedStations_ar and stations_coll_ar arrays.
   for (int i = 0; i < 10; i++)
        {
```

```
        connectedStations_ar[i] = false;
        stations_coll_ar[i] = false;
}

int new_Sockfd;
socklen_t client_length;
struct sockaddr_in serv_addr, cli_addr;
int n = 0;

//Creating an internet stream TCP socket.
sock_fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

if (sock_fd < 0)
        error("ERROR opening socket");

int yes = 1;

//setting the SO_REUSEADDR socket option before calling bind function.
if (setsockopt(sock_fd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1)
{
        perror("setsockopt");
        exit(1);
}

//bzero function sets the entire structure to zero.
bzero((char*) (&serv_addr), sizeof(serv_addr));
```

```
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
//port_no is the port number which user gives as input.
serv_addr.sin_port = htons(port_no);

if (bind(sock_fd, (struct sockaddr*) (&serv_addr), sizeof(serv_addr)) < 0)
{
        error("ERROR on binding");
}

//the socket is converted into a listening socket with a queue of 10 where the incoming connections will be accepted by the kernel.
listen(sock_fd, 10);

client_length = sizeof(cli_addr);

fd_set rset;

//Initializes the file descriptor set to contain no file decriptors.
FD_ZERO(&rset);

while (true)
{
        FD_SET(sock_fd, &rset);

        //waiting for connection from any station
        int nready = select(sock_fd + 1, &rset, NULL, NULL, NULL);

        if (FD_ISSET(sock_fd, &rset))
```

```
		{
			new_Sockfd = accept(sock_fd, (struct sockaddr*) (&cli_addr), &client_length);
			if (new_Sockfd < 0)
			{
				error("ERROR on accept");
			}

			printf("\nServer established connection with the client %s : %d",inet_ntoa(cli_addr.sin_addr),ntohs(cli_addr.sin_port) );

			fflush(stdout);

			int childpid;

			if ((childpid = fork()) == 0)
			{
				//spawn a child process to handle the station frame part
				char buffer1[151];
				bzero(buffer1, 151);

				//Read from station process
				n = read(new_Sockfd, buffer1, 150);

				if (n < 0)
					error("ERROR reading from socket");

				int fromStation, toStation, frameid, partno;
```

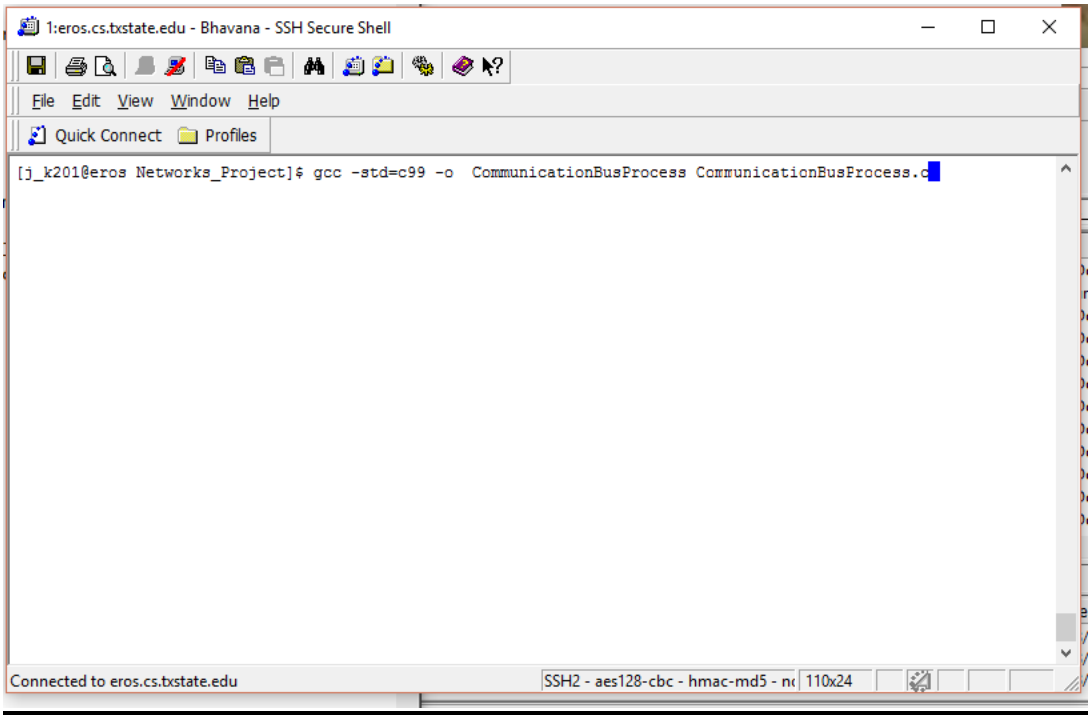
```
                    //frame id, part no, station from, station to are stored in the buffer1
array.
                    sscanf(buffer1, "%d %d %d %d", &frameid, &partno,
&fromStation,&toStation);


                    //Log received message to bus log
                    fprintf(file_pointer,"Receive part %d  of frame %d from Station
%d , to Station %d \r\n", partno,frameid,fromStation,toStation);


                    //Set the flag to indicate connected station
                    if (partno == 1)
                            connectedStations_ar[fromStation - 1] = true;


                    char reply[10] = "success";
                    //checks for collision


                    int collision = 0;


                    if(Is_there_a_Collision(fromStation-1) == 1)
                    {
                            collision = 1;
            }
            if(stations_coll_ar[fromStation - 1] == 1)
                    {
                            collision = 1;
            }
                    if (collision == 1)
                    {
                            //prepare to send collision message back to station and reset
flags
```

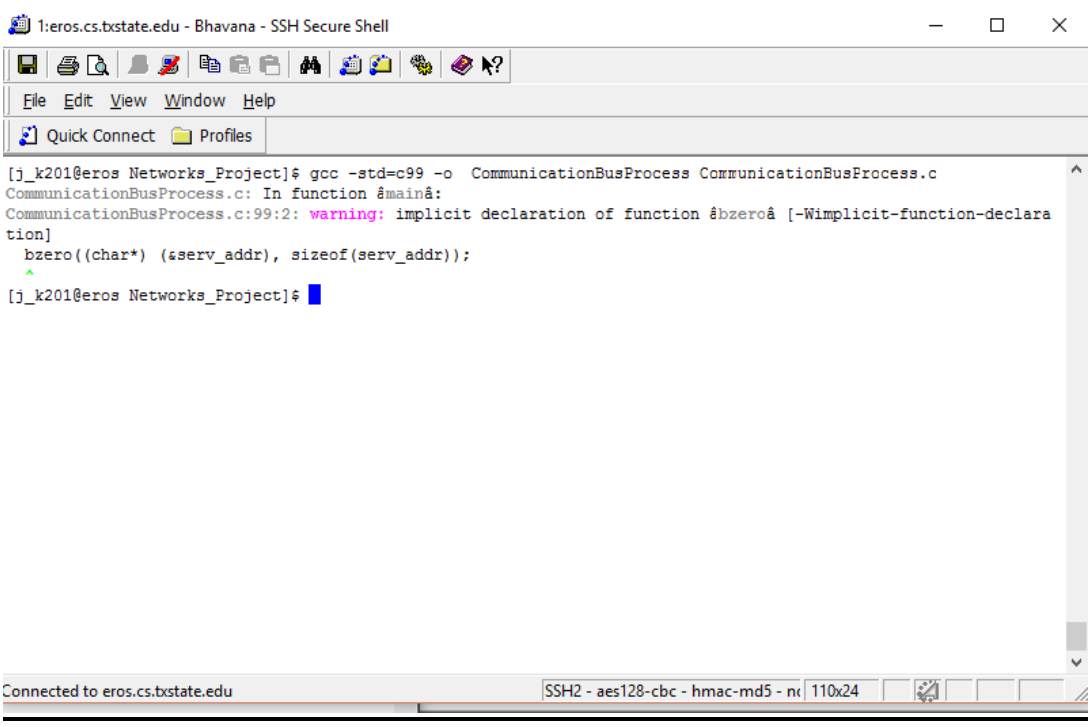
```
                                        strcpy(reply, "collision");
                                        fprintf(file_pointer,"Inform station  %d  a collision \r\n",
fromStation);
                                }
                                else
                                {
                                        if (partno == 2)
                                        {
                                                fprintf(file_pointer,"Transfer part 1 of frame   %d
from Station %d , to station %d\r\n",frameid, fromStation,toStation);

                                                //check whether there is a collision after sending
first part to destination
                                                if(Is_there_a_Collision(fromStation-1) == 1)
                                        {
                                                collision = 1;
                }
                if(stations_coll_ar[fromStation - 1] == 1)
                                        {
                                                collision = 1;
                }
                                                if (collision == 1)
                                                {
                                                //prepare to send collision message back to
station and reset flags
                                                strcpy(reply, "collision");

                                                fprintf(file_pointer,"Inform station  %d  a
collision \r\n", fromStation);
                                                }
```

```
                                        else
                                        {
                                                //prepare for success reply and reset flags
                                                fprintf(file_pointer,"Transfer part 2 of frame %d  from Station %d , to station %d\r\n",frameid, fromStation,toStation);
                                        }
                                }
                                }
                                connectedStations_ar[fromStation - 1] = false;
                                stations_coll_ar[fromStation - 1] = false;

                                //send the reply back to station
                                write(new_Sockfd, reply, sizeof(reply));
                                exit(0);
                        }
                        //parent closes connected socket
                        close(new_Sockfd);
                }
        }
        close(sock_fd);
   fclose(file_pointer);

        return 0;
}
```

Compile the CommunicationBusProcess.c program:

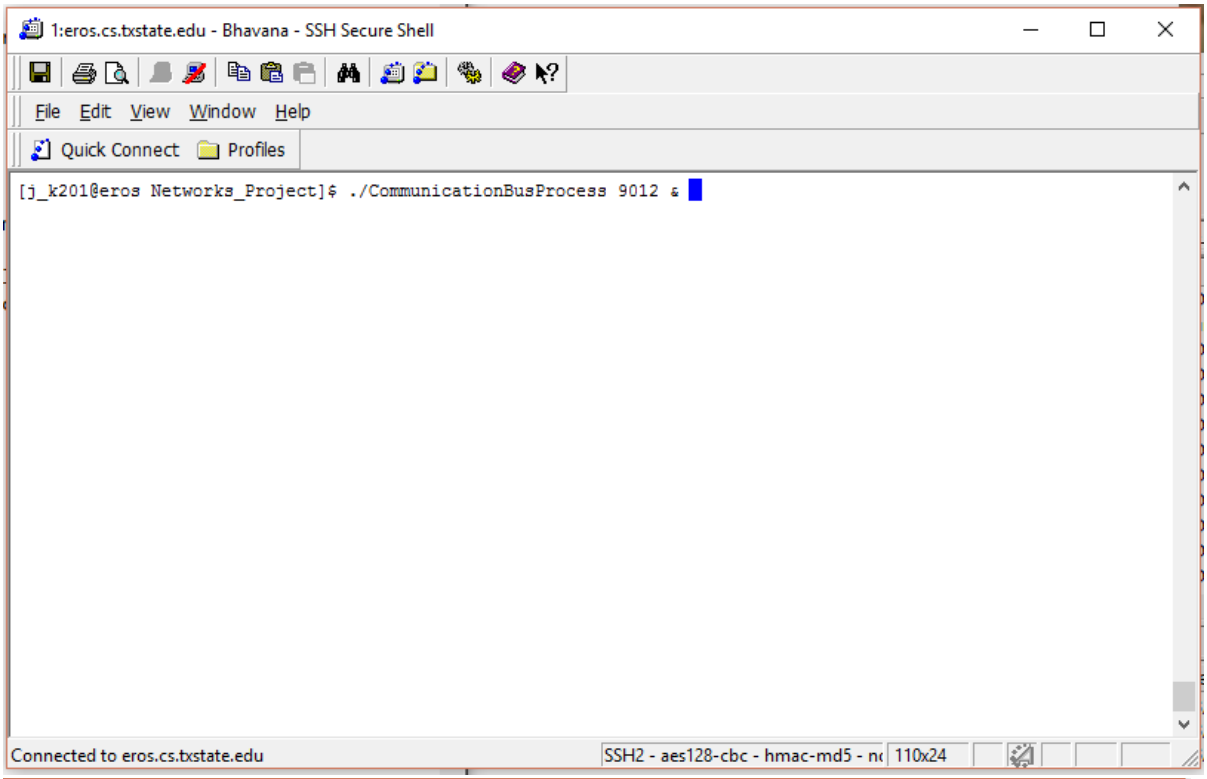
"gcc -std=c99 -o CommunicationBusProcess CommunicationBusProcess.c"

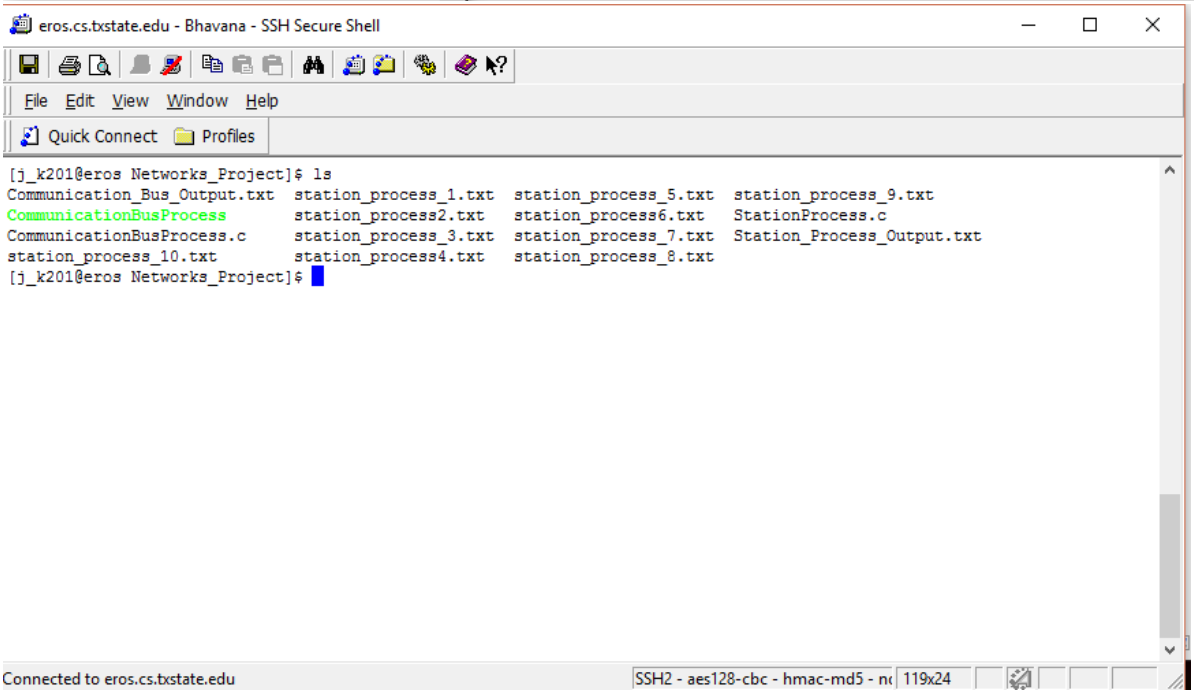




Execute the CommunicationBusProcess.c program:

"./CommunicationBusProcess <ANY PORT NUMBER> &"





Station Process (StationProcess.c):

- StationProcess.c reads the simulation input data file based on the users Input.
- It sends the first message representing the first part of the frame to the CommunicationBusProcess.
- StationProcess.c program will simulate the station process. Based on the user input station, it will simulate those corresponding stations.
- The contents of the input file are read into a buffer line by line. The input file specifies which frame needs to be sent to which destination station. The **socket** created using the **Socket Function** once the buffer has a line form the input file. The **connect** function initiates a connection from the socket with file descriptor sockfd to the socket whose address is specified by the serv_addr and sizeof(serv_addr) arguments. Then writes the part of frame into the socket sockfd.
- **FUNCTION: sendFrame(char *rdbuff, int part, int i)**

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

#define bool int
#define true 1
#define false 0

int sock_fd;
struct sockaddr_in serv_addr;
int portno;
```

```
int time_slot_length = 100000;
int station_numb;
FILE *input_file;
FILE *outfile;

//exits when there is any error occured.
void error(const char *message)
{
        perror(message);
        exit(0);
}

//Sends a part of the frame to the Communication bus.
bool sendFrame(char *rdbuff, int part, int i)
{
        int n;
        int frame_id, to_Station;
        char s1[10], s2[10], s3[10], s4[10];
   char reply_buffer[51];
   char write_buff[150];

        //reads the values.
        sscanf(rdbuff, "%s %d %s %s %s %d", s1, &frame_id, s2, s3, s4,&to_Station);

        //sets all the values in the array to zero in write_buff.
        bzero(write_buff, 150);

        //prepare the buffer to write to socket
```

```
        sprintf(write_buff, "%d %d %d %d", frame_id, part, station_numb, to_Station);

        //create socket descriptor sock_fd using socket function
        sock_fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

        //outputs if there is an error in opening the socket.
        if (sock_fd < 0)
        {
                error("ERROR in opening socket");
}

        //outputs the error if there is an error in connecting.
        if (connect(sock_fd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
        {
                error("ERROR in connecting");
}

         //write to socket using socket descriptor sock_fd
        if (write(sock_fd, write_buff, strlen(write_buff)) < 0)
        {
                error("ERROR writing to socket");
}

        //write log to file
        fprintf(outfile,"\nSend part  %d  of  %s \n", part,rdbuff);

        //sets zero to all values in reply_buffer array.
        bzero(reply_buffer, 51);
```

```
        //read from socket using socket descriptor sock_fd
        if (read(sock_fd, reply_buffer, 50) < 0)
        {
                error("ERROR reading from socket");
    }

        //check whether there is a collision at the bus
        if (strcmp(reply_buffer, "success") != 0)
        {
                if (i == 16)
                {
                        error("Transmission failure after 16 attempts");
                }
                n = i > 10 ? 10 : i;
           n = pow(2, n);

                //calculate the time slots to wait before the next attempt
                int time_slots = rand() % n;

      fprintf(outfile,"A collision informed, wait for %d time slots\r\n", time_slots);
                usleep(time_slots * time_slot_length);
                close(sock_fd);
                return false;
        }
        close(sock_fd);
        return true;
}
```

```
int main(int argc, char *argv[])
{
	//check arguments to start the station process
	if (argc != 4)
	{
		printf("\n Please enter in this format : ./StationProc <server name> <port> <station number>");
		exit(0);
	}

   char readstationip[101];

	//open output file and writes on it.
   outfile = fopen("Station_Process_Output.txt" , "w");

   if(outfile == NULL)
	{
		perror("Error opening output file");
		return(-1);
   }

	//hostname store the host address.
	char * hostname = argv[1];

	//portno stores the port number given by the user.
	portno = atoi(argv[2]);

	//station_numb stores the number of the station entered by the user.
```

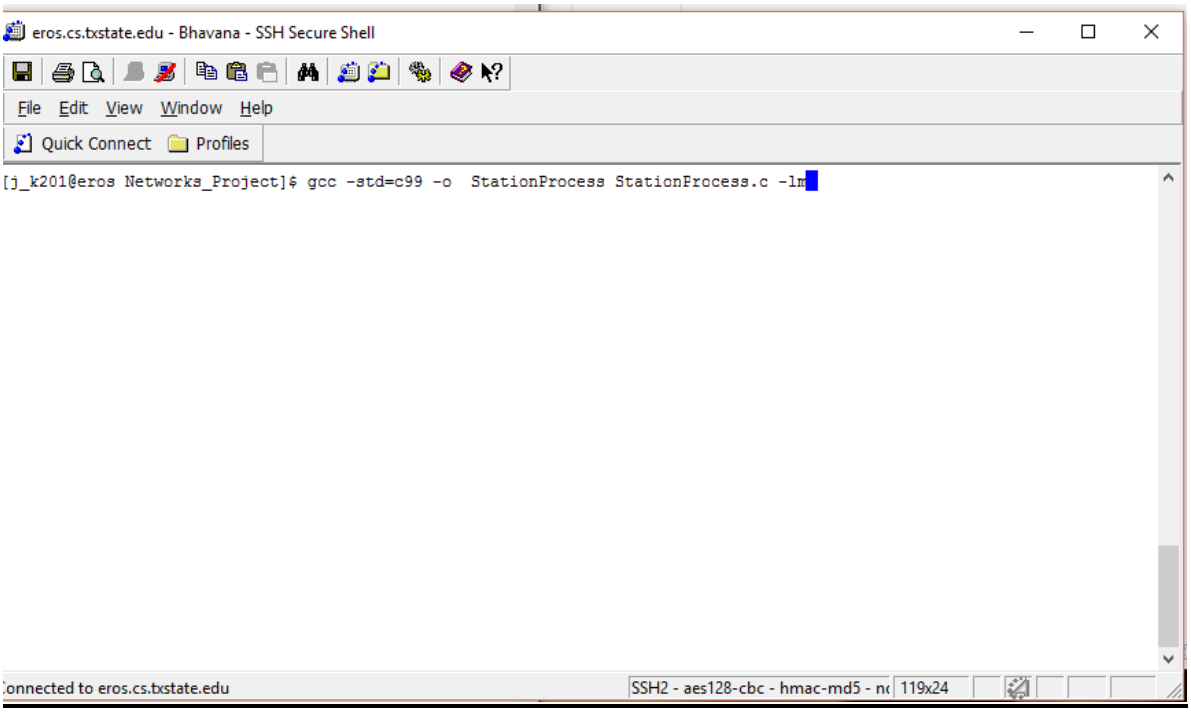
```
        station_numb = atoi(argv[3]);

        //open the input file based on the station number
switch(station_numb)
        {
 case 1:
      input_file  = fopen("station_process1.txt" , "r");
      break;
 case 2:
      input_file  = fopen("station_process2.txt" , "r");
      break;
 case 3:
      input_file  = fopen("station_process3.txt" , "r");
      break;
 case 4:
      input_file  = fopen("station_process4.txt" , "r");
      break;
 case 5:
      input_file  = fopen("station_process5.txt" , "r");
      break;
 case 6:
      input_file  = fopen("station_process6.txt" , "r");
      break;
 case 7:
      input_file  = fopen("station_process7.txt" , "r");
      break;
 case 8:
      input_file  = fopen("station_process8.txt" , "r");
```

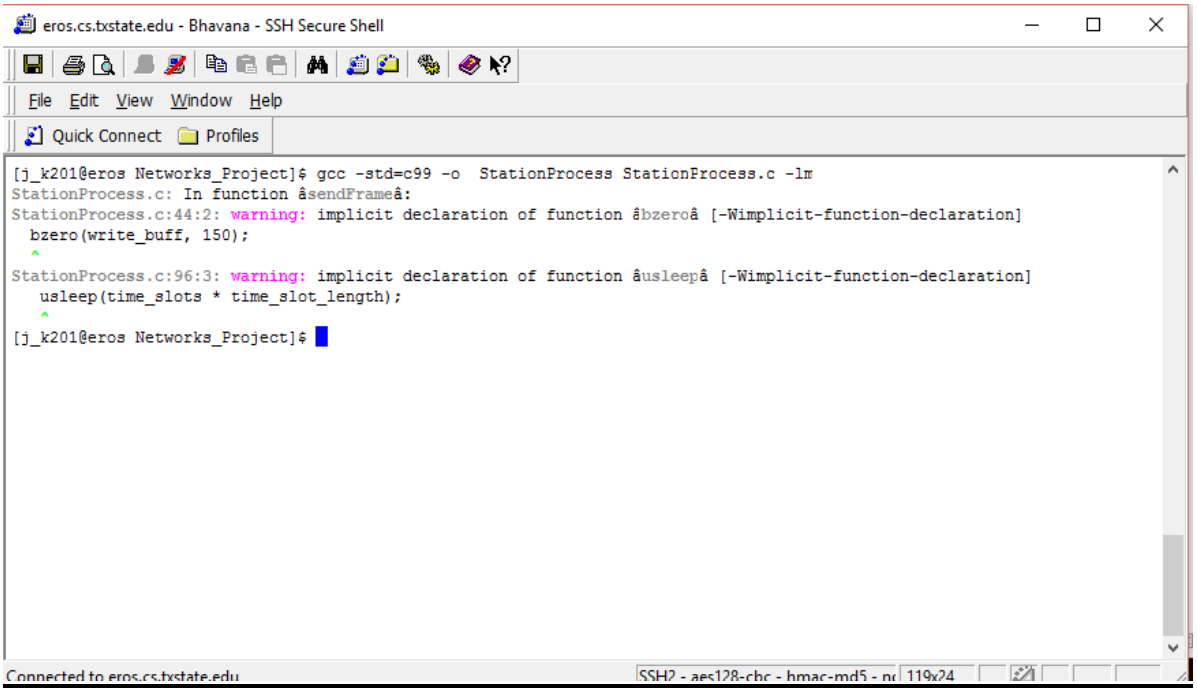
```
        break;
    case 9:
        input_file  = fopen("station_process9.txt" , "r");
        break;
    case 10:
        input_file  = fopen("station_process10.txt" , "r");
        break;
}

//check if input file is opened without any errors.
if(input_file == NULL)
        {
                perror("Error opening input file");
                return(-1);
}

        //sets all the values to zero.
        bzero((char *) &serv_addr, sizeof(serv_addr));

        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr = INADDR_ANY;
        serv_addr.sin_port = htons(portno);

        //read from input file
        while (true)
        {
                bzero(readstationip, 101);
                if(feof(input_file))
```

```
                break;
            if(fgets(readstationip, 100,input_file)!=NULL)
            {
                //send frame represented by the current line in the simulation file
                int i = 0;
                while (true)
                {
                    i++;
                    //send first part of the frame
                    if (!sendFrame(readstationip, 1, i))
                        continue;
                    //send second part of the frame
                    if (sendFrame(readstationip, 2, i))
                        break;
                }
            }
        }
        fclose(input_file);
        fclose(outfile);
        return 0;
}
```

Complie the StationProcess.c program:

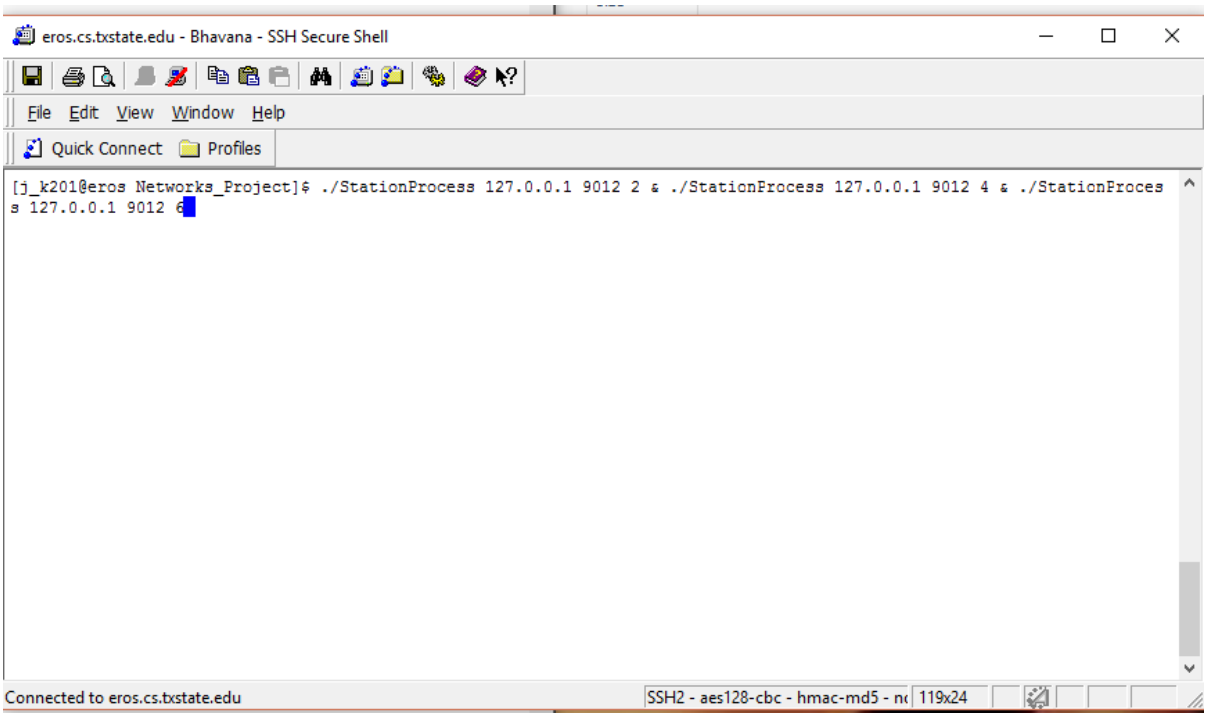
"gcc -std=c99 -o StationProcess StationProcess.c -lm"

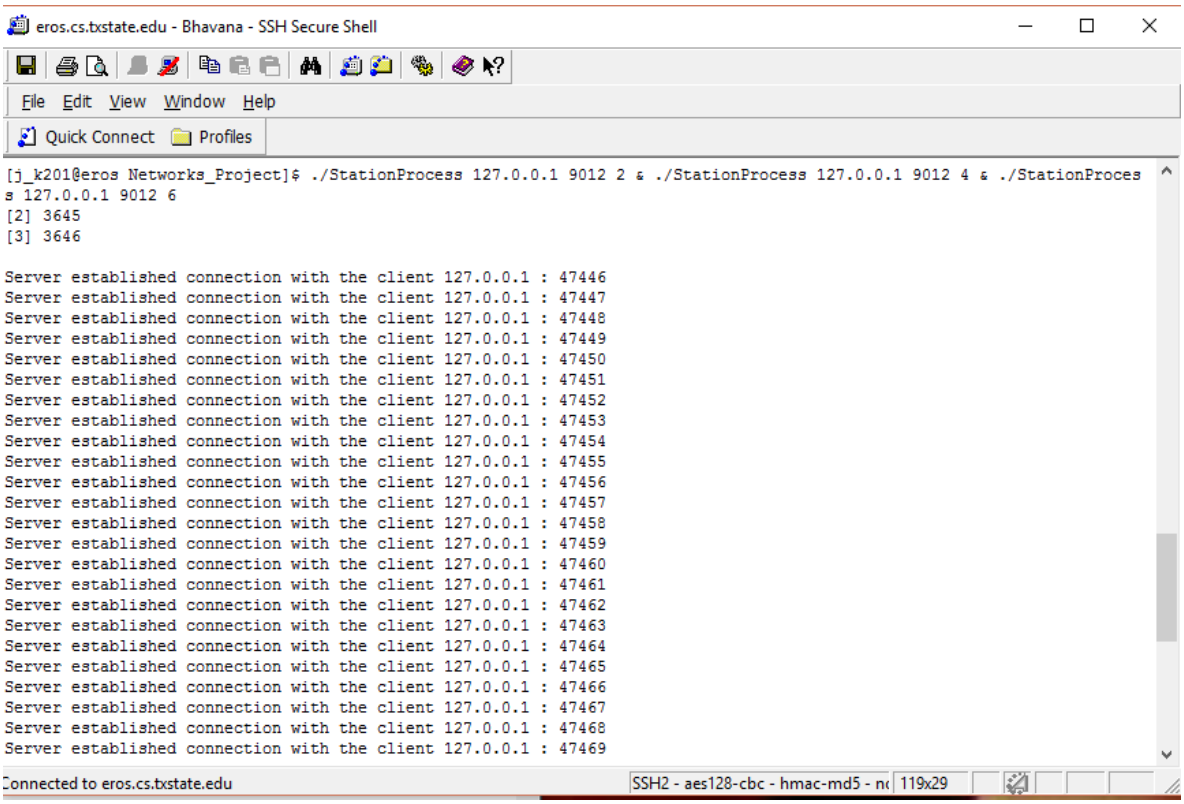




Execute the StationProcess program for different stations simultaneously:

"./StationProcess 127.0.0.1 <PORT NO> <STATION NUMBER> & ./StationProcess 127.0.0.1 <PORT NO> <STATION NUMBER> & ./StationProcess 127.0.0.1 <PORT NO> <STATION NUMBER>"




eros.cs.txstate.edu - Bhavana - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles
[j_k201@eros Networks_Project]$ ./StationProcess 127.0.0.1 9012 2 & ./StationProcess 127.0.0.1 9012 4 & ./StationProcess 127.0.0.1 9012 6
[2] 3645
[3] 3646
Server established connection with the client 127.0.0.1 : 47446
Server established connection with the client 127.0.0.1 : 47447
Server established connection with the client 127.0.0.1 : 47448
Server established connection with the client 127.0.0.1 : 47449
Server established connection with the client 127.0.0.1 : 47450
Server established connection with the client 127.0.0.1 : 47451
Server established connection with the client 127.0.0.1 : 47452
Server established connection with the client 127.0.0.1 : 47453
Server established connection with the client 127.0.0.1 : 47454
Server established connection with the client 127.0.0.1 : 47455
Server established connection with the client 127.0.0.1 : 47456
Server established connection with the client 127.0.0.1 : 47457
Server established connection with the client 127.0.0.1 : 47458
Server established connection with the client 127.0.0.1 : 47459
Server established connection with the client 127.0.0.1 : 47460
Server established connection with the client 127.0.0.1 : 47461
Server established connection with the client 127.0.0.1 : 47462
Server established connection with the client 127.0.0.1 : 47463
Server established connection with the client 127.0.0.1 : 47464
Server established connection with the client 127.0.0.1 : 47465
Server established connection with the client 127.0.0.1 : 47466
Server established connection with the client 127.0.0.1 : 47467
Server established connection with the client 127.0.0.1 : 47468
Server established connection with the client 127.0.0.1 : 47469
Connected to eros.cs.txstate.edu
SSH2 - aes128-cbc - hmac-md5 - n
119x29

BEBO Algorithm:

It checks whether there is collision at CommunicationBusProcess. If collision occurs, it will calculate the binary exponential backoff time and sleep for the calculated amount of time. The sendFrame function returns false if collision occurs, otherwise the function return true.

Communication_Bus_Process.txt

Receive part 1 of frame 1 from Station 6 , to Station 3

Receive part 1 of frame 1 from Station 4 , to Station 3

Receive part 2 of frame 1 from Station 6 , to Station 3

Transfer part 1 of frame 1 from Station 6 , to station 3

Transfer part 2 of frame 1 from Station 6 , to station 3

Receive part 1 of frame 1 from Station 2 , to Station 3

Receive part 1 of frame 2 from Station 6 , to Station 4

Receive part 2 of frame 1 from Station 2 , to Station 3

Transfer part 1 of frame 1 from Station 2 , to station 3

Transfer part 2 of frame 1 from Station 2 , to station 3

Receive part 2 of frame 2 from Station 6 , to Station 4

Transfer part 1 of frame 2 from Station 6 , to station 4

Transfer part 2 of frame 2 from Station 6 , to station 4

Receive part 1 of frame 3 from Station 6 , to Station 3

Receive part 2 of frame 1 from Station 4 , to Station 3

Transfer part 1 of frame 1 from Station 4 , to station 3

Transfer part 2 of frame 1 from Station 4 , to station 3

Receive part 1 of frame 2 from Station 4 , to Station 4

Receive part 2 of frame 3 from Station 6 , to Station 3

Transfer part 1 of frame 3 from Station 6 , to station 3

Transfer part 2 of frame 3 from Station 6 , to station 3

Receive part 1 of frame 2 from Station 2 , to Station 4

Receive part 2 of frame 2 from Station 4 , to Station 4

Transfer part 1 of frame 2 from Station 4 , to station 4

Transfer part 2 of frame 2 from Station 4 , to station 4

Receive part 2 of frame 2 from Station 2 , to Station 4

Transfer part 1 of frame 2 from Station 2 , to station 4

Transfer part 2 of frame 2 from Station 2 , to station 4

Receive part 1 of frame 3 from Station 4 , to Station 3

Receive part 2 of frame 3 from Station 4 , to Station 3

Transfer part 1 of frame 3 from Station 4 , to station 3

Transfer part 2 of frame 3 from Station 4 , to station 3

Receive part 1 of frame 3 from Station 2 , to Station 3

Receive part 2 of frame 3 from Station 2 , to Station 3

Transfer part 1 of frame 3 from Station 2 , to station 3

Transfer part 2 of frame 3 from Station 2 , to station 3

Receive part 1 of frame 4 from Station 6 , to Station 2

Receive part 2 of frame 4 from Station 6 , to Station 2

Transfer part 1 of frame 4 from Station 6 , to station 2

Transfer part 2 of frame 4 from Station 6 , to station 2

Receive part 1 of frame 4 from Station 4 , to Station 2

Receive part 1 of frame 1 from Station 6 , to Station 3

Receive part 1 of frame 4 from Station 2 , to Station 2

Receive part 2 of frame 1 from Station 6 , to Station 3

Transfer part 1 of frame 1 from Station 6 , to station 3

Transfer part 2 of frame 1 from Station 6 , to station 3

Receive part 2 of frame 4 from Station 4 , to Station 2

Transfer part 1 of frame 4 from Station 4 , to station 2

Transfer part 2 of frame 4 from Station 4 , to station 2

Receive part 2 of frame 4 from Station 2 , to Station 2

Transfer part 1 of frame 4 from Station 2 , to station 2

Transfer part 2 of frame 4 from Station 2 , to station 2

Receive part 1 of frame 2 from Station 6 , to Station 4

Receive part 1 of frame 1 from Station 4 , to Station 3

Receive part 1 of frame 1 from Station 2 , to Station 3

Receive part 2 of frame 1 from Station 2 , to Station 3

Transfer part 1 of frame 1 from Station 2 , to station 3

Transfer part 2 of frame 1 from Station 2 , to station 3

Receive part 2 of frame 1 from Station 4 , to Station 3

Transfer part 1 of frame 1 from Station 4 , to station 3

Transfer part 2 of frame 1 from Station 4 , to station 3

Receive part 1 of frame 2 from Station 2 , to Station 4

Receive part 1 of frame 2 from Station 4 , to Station 4

Receive part 2 of frame 2 from Station 6 , to Station 4

Transfer part 1 of frame 2 from Station 6 , to station 4

Transfer part 2 of frame 2 from Station 6 , to station 4

Receive part 2 of frame 2 from Station 2 , to Station 4

Transfer part 1 of frame 2 from Station 2 , to station 4

Transfer part 2 of frame 2 from Station 2 , to station 4

Receive part 2 of frame 3 from Station 2 , to Station 3

Transfer part 1 of frame 3 from Station 2 , to station 3

Transfer part 2 of frame 3 from Station 2 , to station 3

Receive part 1 of frame 3 from Station 6 , to Station 3

Receive part 1 of frame 3 from Station 2 , to Station 3

Receive part 2 of frame 2 from Station 4 , to Station 4

Transfer part 1 of frame 2 from Station 4 , to station 4

Transfer part 2 of frame 2 from Station 4 , to station 4

Receive part 2 of frame 3 from Station 6 , to Station 3

Transfer part 1 of frame 3 from Station 6 , to station 3

Transfer part 2 of frame 3 from Station 6 , to station 3

Receive part 1 of frame 3 from Station 4 , to Station 3

Receive part 2 of frame 3 from Station 4 , to Station 3

Transfer part 1 of frame 3 from Station 4 , to station 3

Transfer part 2 of frame 3 from Station 4 , to station 3

Receive part 1 of frame 4 from Station 6 , to Station 2

Receive part 1 of frame 4 from Station 2 , to Station 2

Station_Process_Output.txt:

Send part 1 of Frame 1, To Station 3

Send part 2 of Frame 1, To Station 3

Send part 1 of Frame 2, To Station 4

Send part 2 of Frame 2, To Station 4

Send part 1 of Frame 3, To Station 3

Send part 2 of Frame 3, To Station 3

Send part 1 of Frame 4, To Station 2

Send part 2 of Frame 4, To Station 2

Send part 1 of Frame 1, To Station 3

Send part 2 of Frame 1, To Station 3

Send part 1 of Frame 2, To Station 4

Send part 2 of Frame 2, To Station 4

Send part 1 of Frame 3, To Station 3

Send part 2 of Frame 3, To Station 3

Send part 1 of Frame 4, To Station 2

Send part 2 of Frame 4, To Station 2

Collision detecting:

I tried to implement collision detection using the function **Is_there_a_Collision.** While testing I was unable simulate a collision scenario. So I was unable to test this part of implementation. I tried to run two or more stations simultaneously. But I was not able to simulate a collision even in this scenario.

Interrupts:

Interrupts is not implemented.

Observations during testing:

1. The program is successfully sending the frames from station process to bus process. The output file related to the bus file is showing the received and transferred frames.

2. The program is handling multiple stations simultaneously and simulating the frame transfer for all the intended stations (given by the user). This can be verified in the output files supplied as part of the deliverables.

3. Collision is not being detected by the program or my understanding of collision simulation is wrong.