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

Submitted by: Kommuru Jaya Naga Bhavana.

(A04711981)

**Project Aim and Objective:**

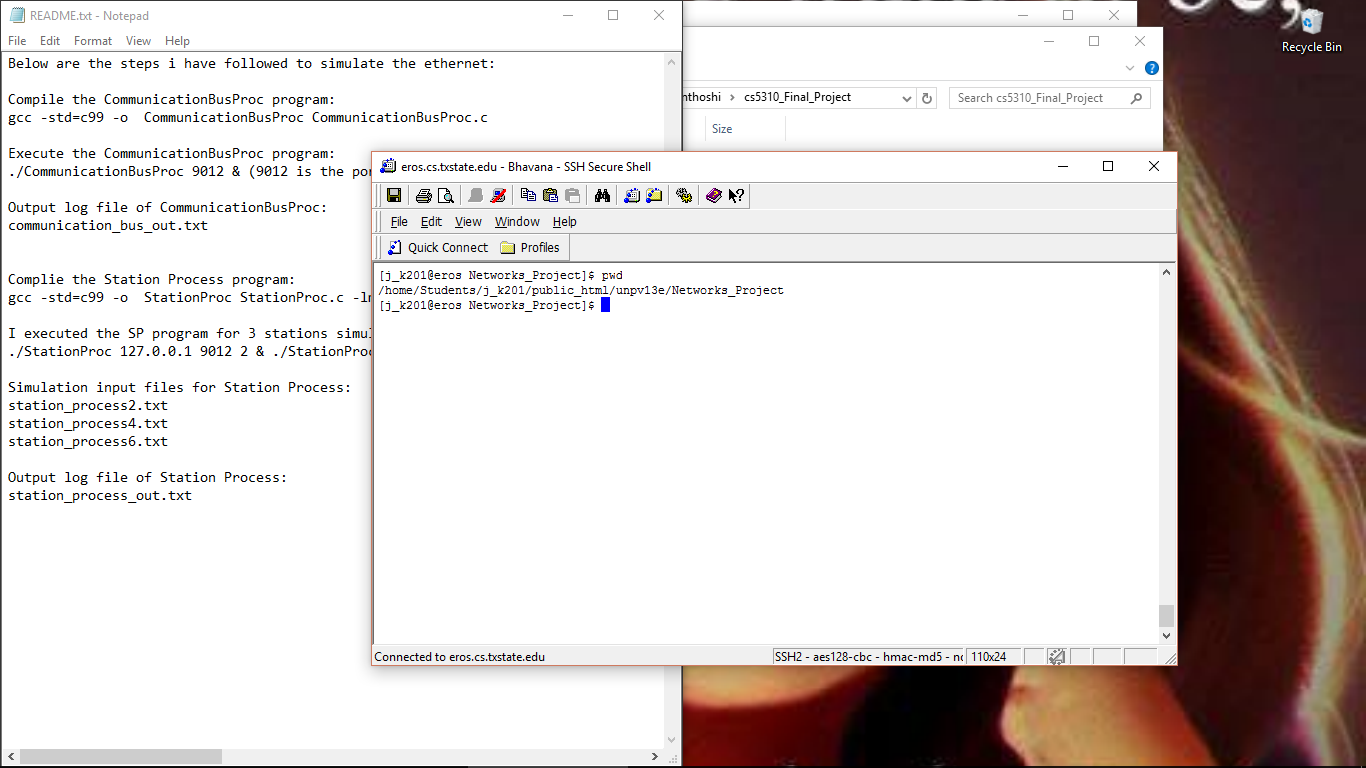
This project is to simulate the classical Ethernet. The main aim of the project is to practice basic socket API programming through a client/server application. It is a simulation in the sense that the Ethernet is simulated by multiple processes on multiple machines. Each station in the Ethernet is simulated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

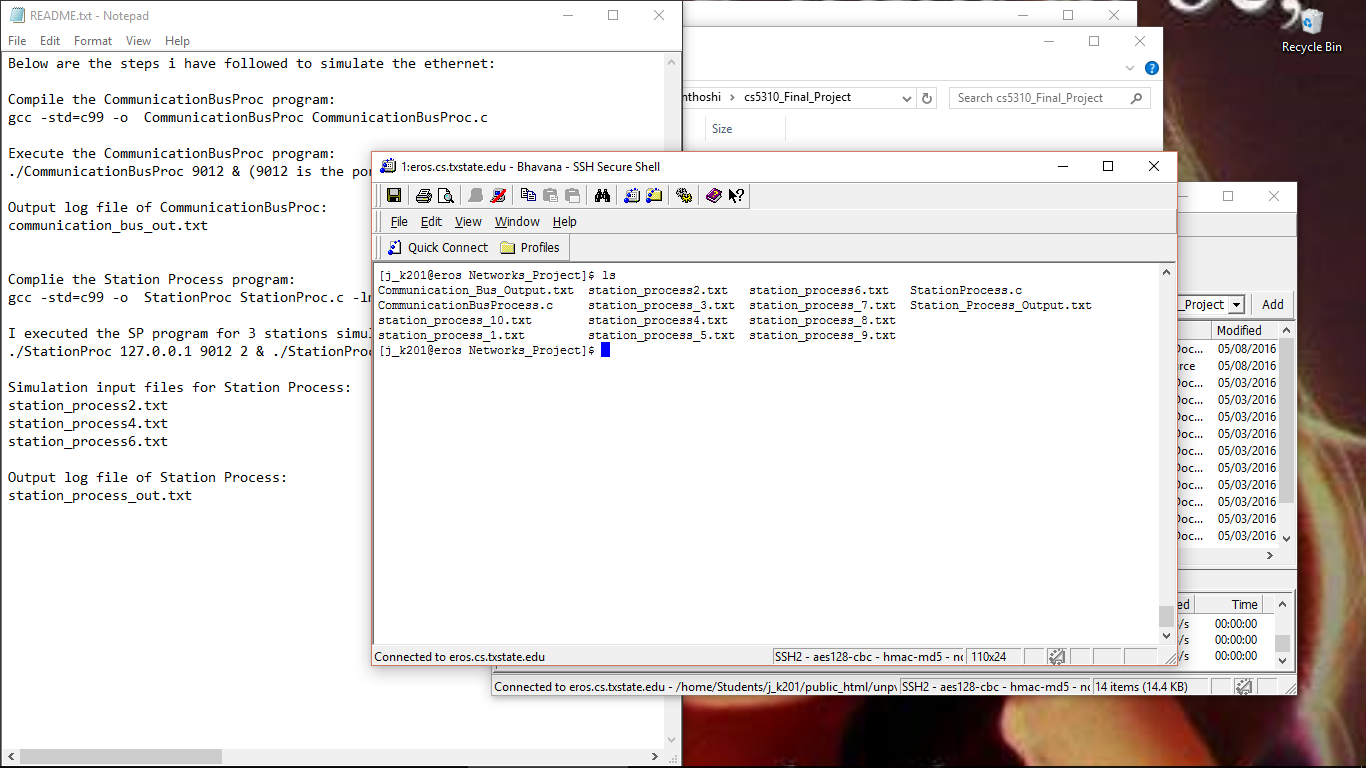
station\_process10.txt

**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);

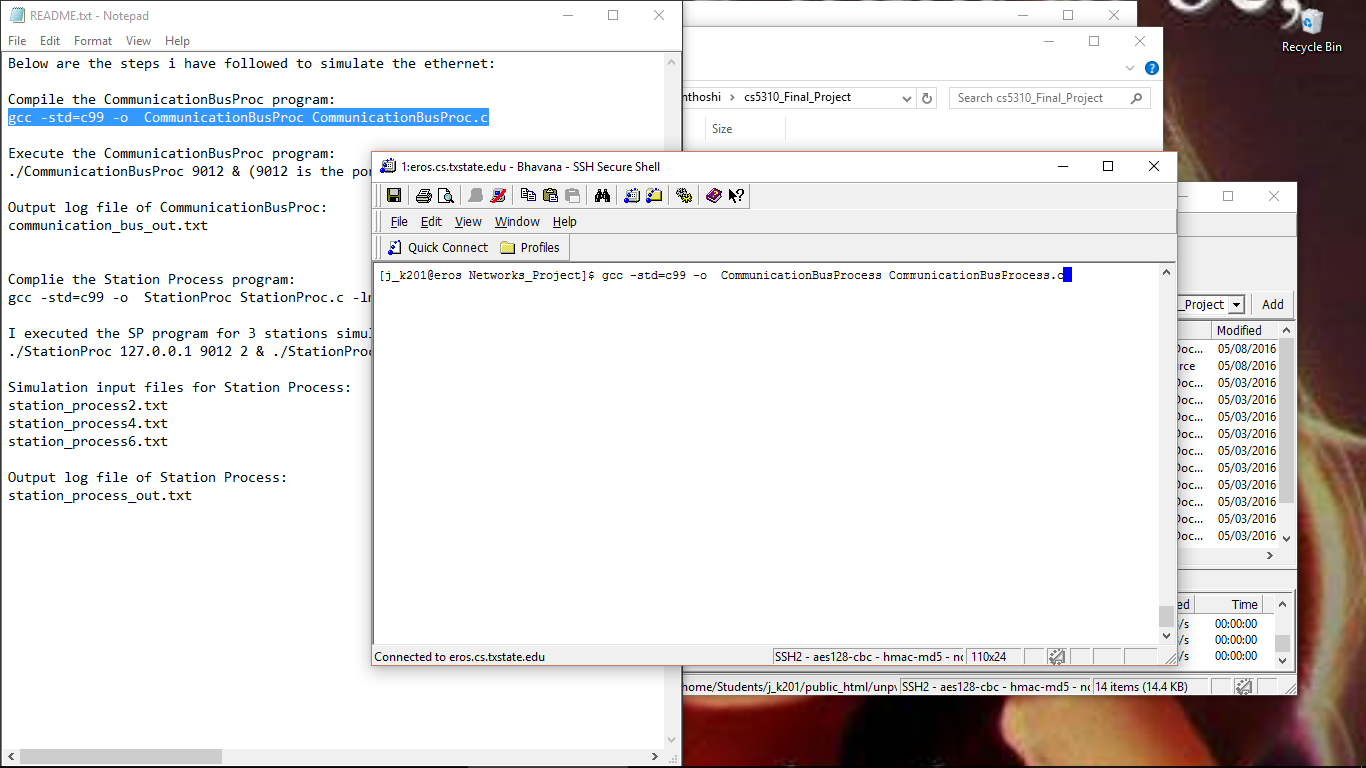
fclose(file\_pointer);

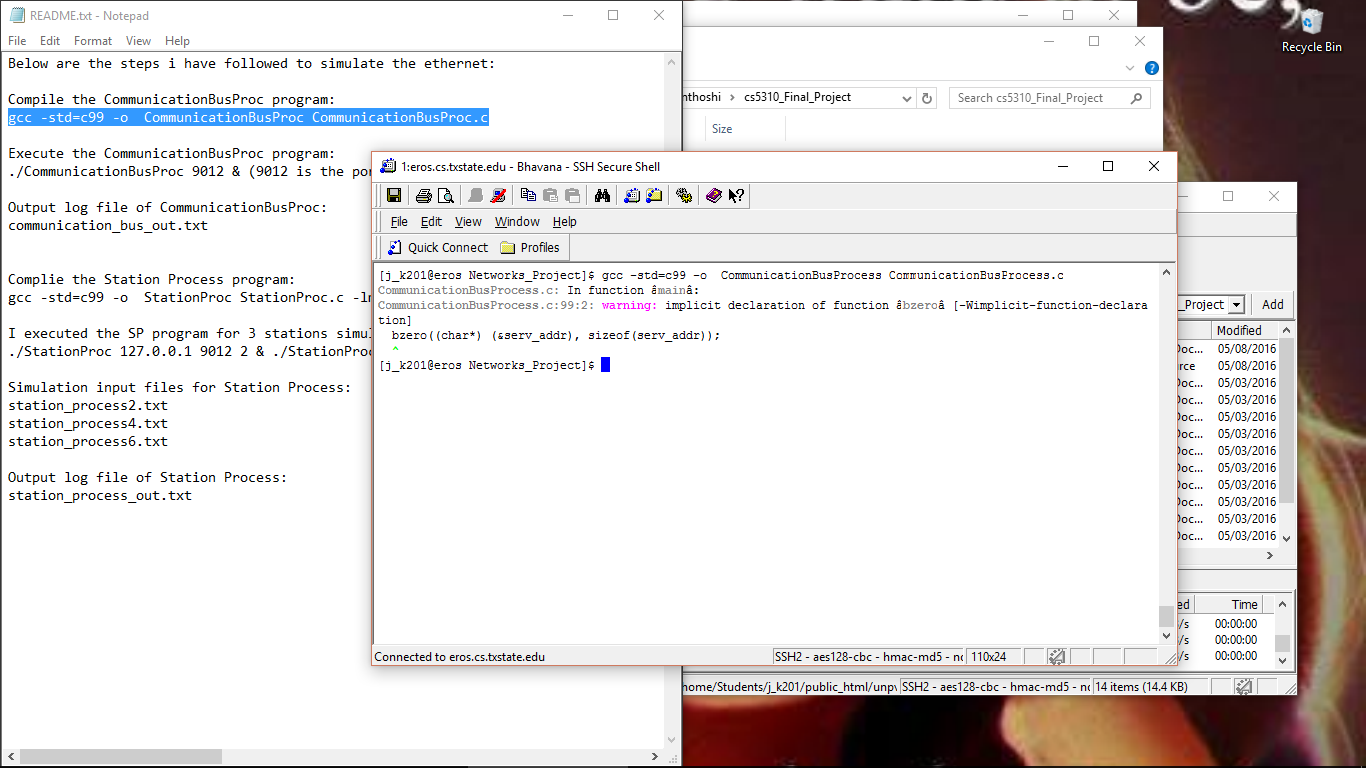
return 0;

}

**Compile the CommunicationBusProcess.c program:**

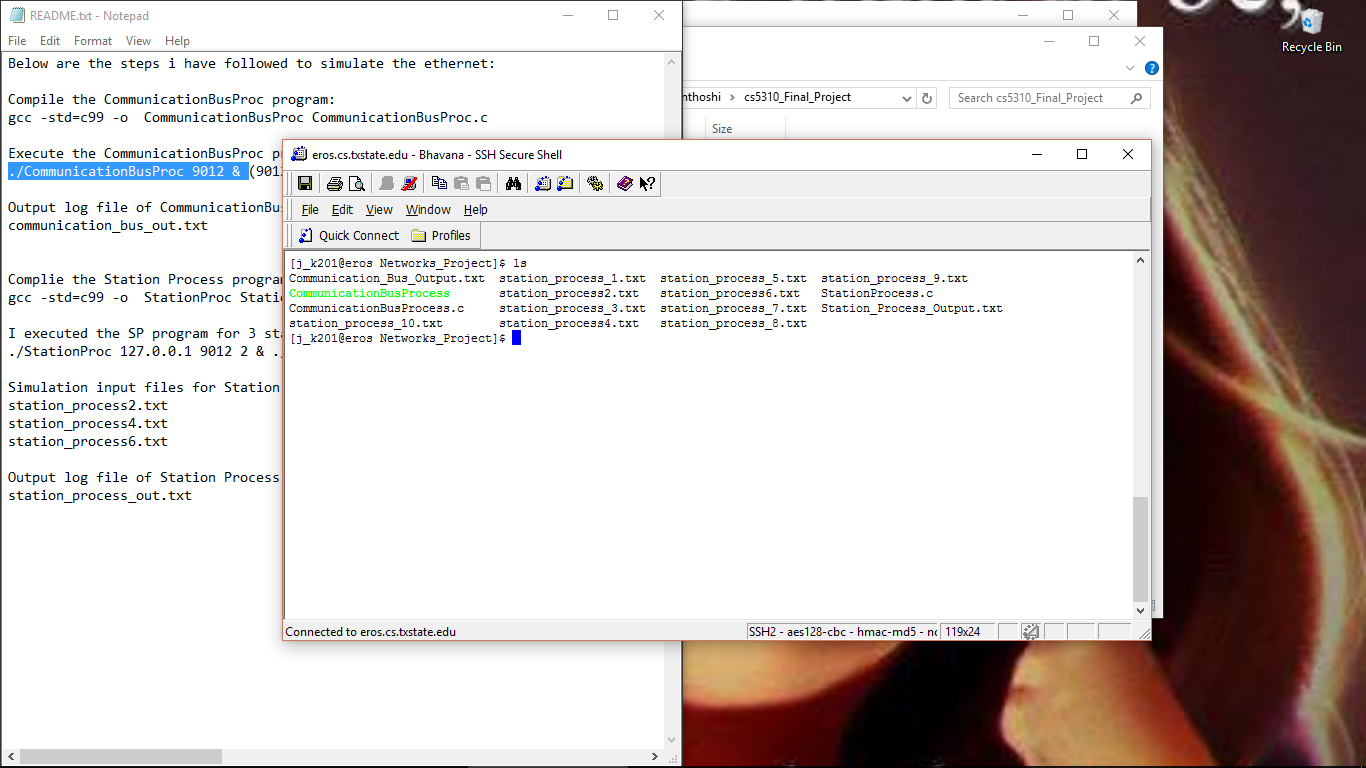
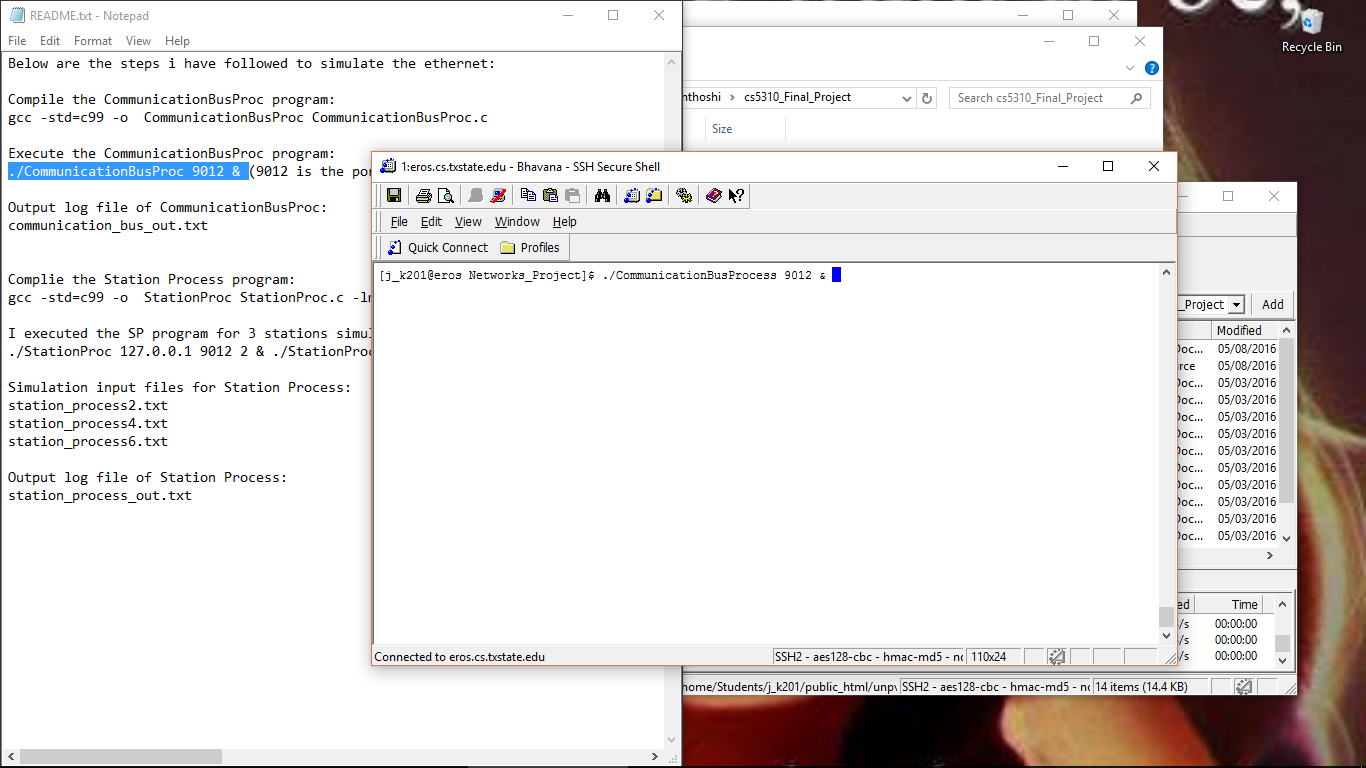
**"gcc -std=c99 -o CommunicationBusProcess CommunicationBusProcess.c"**

****

****

**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);

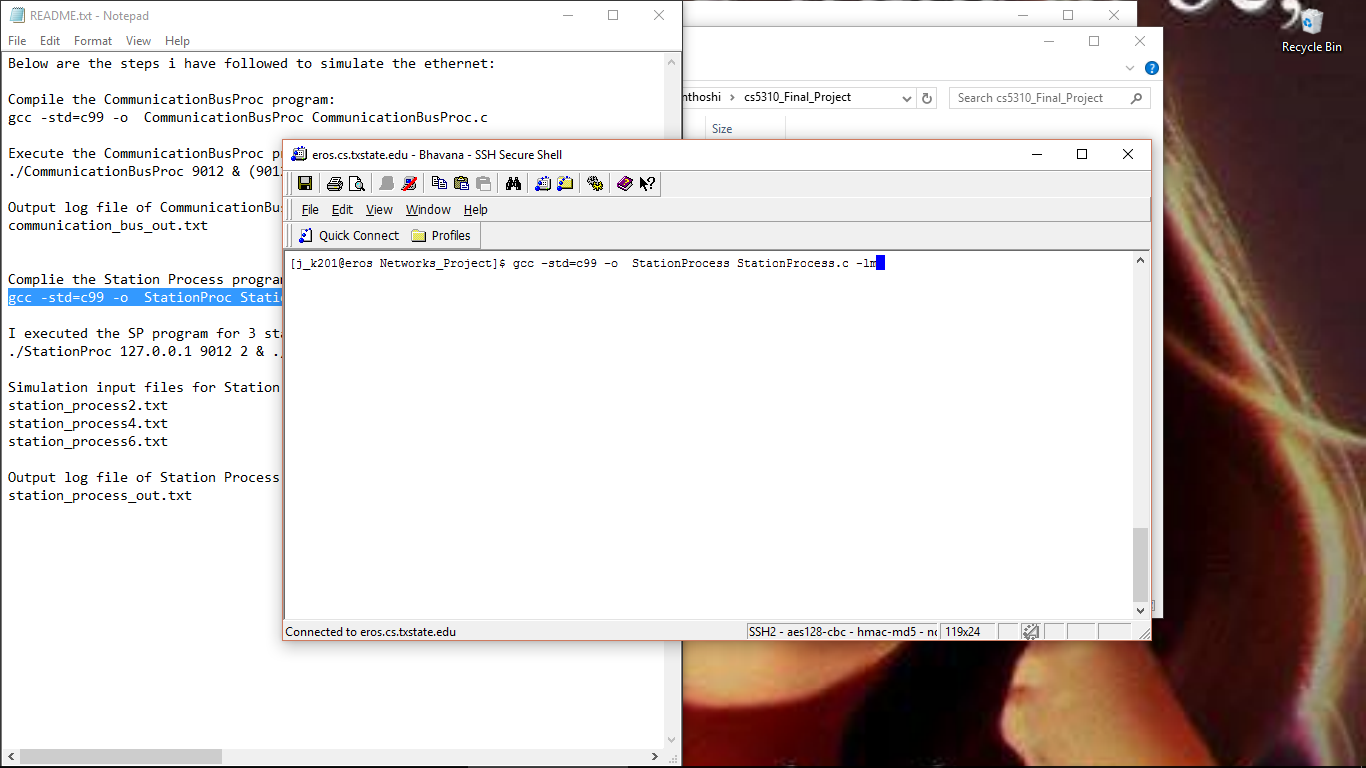
fclose(outfile);

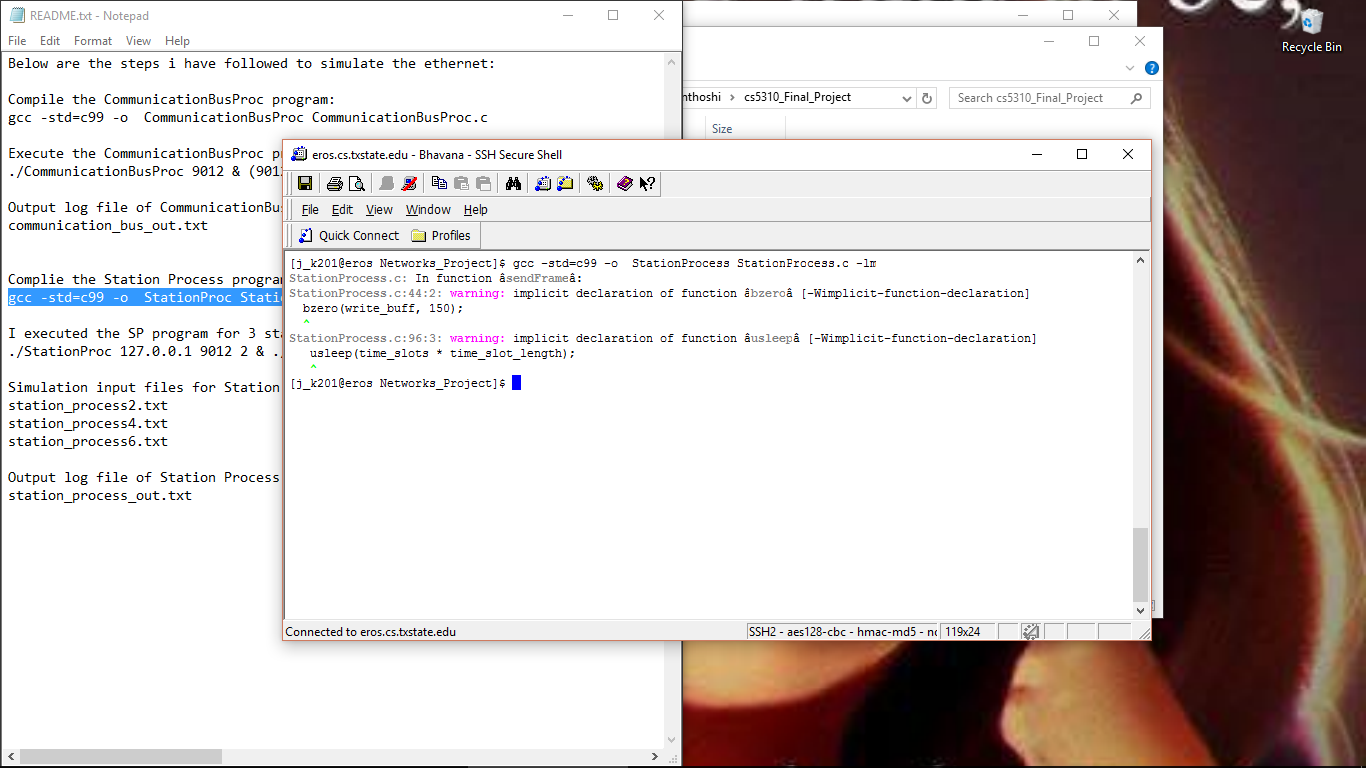
return 0;

}

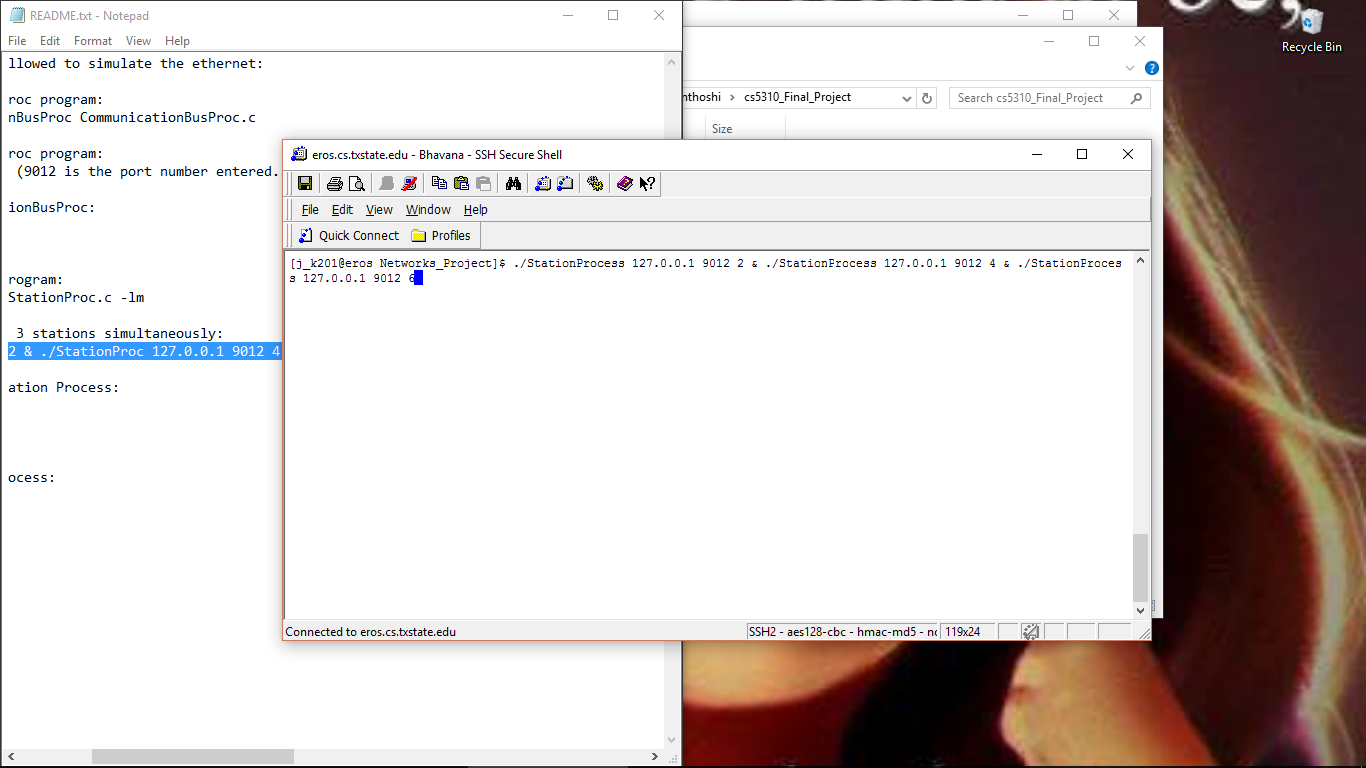
**Complie the StationProcess.c program:**

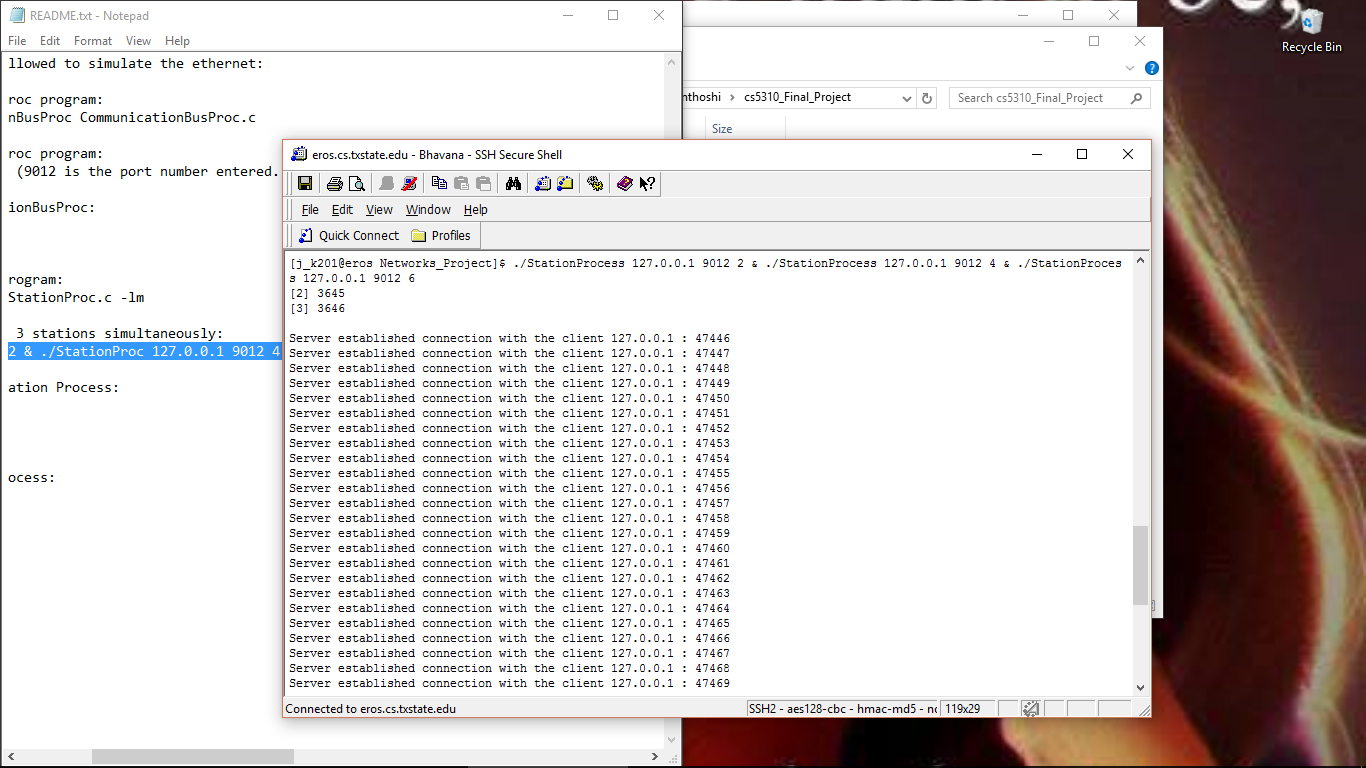
**"gcc -std=c99 -o StationProcess StationProcess.c -lm"**

****

****

**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>"**

****

**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.