Computer Networks and Communication Systems (CS 5310)

Project

Vasudha Gangireddy (A04071215)

December 10, 2017

**Project Description:**

The project aims at the simulation of an **Fast Ethernet** network for a client/server application using programming API. The Fast Ethernet is simulated by multiple processes and multiple clients using multiplexing concepts.

**Technical Stack:**

For implementing the project, the following were used.

Operating system: UNIX

Programming Language : C

**Station Process**

• SP.c reads the simulation input data ﬁle based on the users Input.

• SP.c program will simulate the station process based on the input station entered by user.

• The contents from input ﬁle are read into buﬀer line by line. The input ﬁle contains the host and destination Service Process numbers.

* Methods/ functions used:

The following functions have been created to implement the functionality of the Station Process:

* *void errHandle(const char \*msg)* : This function helps is handling errors and exiting gracefully.
* *void sendRFrame(char \*rbuff)* : Used to send request frame to the server.
* *void sendDFrame(char \*rdbuff)*: Used to read the server response and if the server has accepted the request, a data frame is created and sent out.

The data in the data frame is read from the *DataFile.txt* file.

If a Reject is read from the CSP, it writes an appropriate message to the output file and the request is resend if request has not been rejected thrice.

* *bool waitFrames(char \*rbuff)*: This function checks if a Reject has been sent 3 times. If a request has been rejected 3 times this function will return True else False.
* *void processInput(char input\_line[])*: This function determines the frame type or action to be taken after a line is read from the input file.

If the read line is to send a data to another SP then sendRFrame() function is called, else waitFrames() function is called to tell the client to wait for the specified number of frames in the input file.

* *Main function*:
  + - The user is asked to input <server name> <port> <station number>.

./SP <server name> <port> <station number>

* + - Based on the station number entered by the client the respective Input file is opened and data read from the file one at a time.
    - The **select** function is called here to handle multiple processes on the client side. A file descriptor and a socket descriptor are used to read data from a file and read and write to the CSP respectively.
* **Communication Switch Process:**

Structures used:

Frame :

struct frame

{

int seqno;

int sadd;

int dadd;

char data[MAX];

};

used to create data\_frame queue and request\_frame. This is where the data frames are stored that have been received from the client.

If the data frame queue is full then the requests are stored in the request queue and processed later.

struct cliaddr

{

int descriptor;

int station\_num;

};

This data structure is used to store the client socket descriptors and their corresponding station numbers. A map between the clients and their addresses (i.e station numbers entered by the user.

* Methods/Functions used:
* *void errHandler(const char \*message,int exitCode):* Handles errors gracefully.
* *void processRequest(char \*rbuff, int i):* This function processes the request frames received from the SP’s.
  + - This function is used both for processing the requests coming from the client as well as the request frames in the request queue.
    - If the frame is a request frame, then the data frame queue is checked.
    - If the data frame queue has space in it then an accept frame is sent to the SP.
    - If the data frame queue is full, then the request queue is checked and the frame is placed in the request queue.
    - If the request queue is full as well then a Reject frame is sent to the client.
* *int main(int argc, char \* argv[]):*
* This function takes an input from the client.

./CSP <SERVER PORT NUMBER>

* The data frame queue and the request frame queues are initialized to -1.
* Select function is used to handle the various clients – sending as well as receiving from the clients.
* All the clients are listened to and then the data from the clients is read and processed.
* If it’s a request frame –

And it’s the first request from the client then the descriptor as well as the station number from the client are saved in the client address data structure so that the CSP can retrieve it later and send the data frames to the corresponding SP’s.

* The variable nready stores the output from the select statement. If the value in the variable is less than or equal to zero (which means there are no clients ready for either read or write) then the data frame queue and the request queues are processed.
* The data frames from the data queues are read and the appropriate descriptor is selected from the client address array structure and the frame is sent. Then the frame is removed from the queue.
* The request queues are processed using the processRequest function.

**Fastether.h**

This file contains some releveant API’s included that are used by both the SP.c and CSP.c. A structure called frame is defined which is the basic structure of a frame.

**Observations:**

* Code compiles successfully without any errors.
* Server and clients run on different systems successfully in the Linux environment.
* There is no output generated as there is a bug in the code which I could not resolve.

I used select function in the SP code to read file and then write to the socket. I could figure out that the socket descriptor was not adding to rset though I couldn’t figure out why? That is the reason though the file is being read there is nothing written onto the output files.