Readme and Project Description

Simulation of Fast Ethernet

CS5310 / Fall 2018

Student: Liang Wang

Contents

Instruction to compile and run

The project is implemented in C and source files are as following:

common.h

CSP.c

SP.c

To compile

make CSP

make SP

To run

./CSP listen_address listen_port queue_size(optional)

./SP server_address server_port input_file

Design and implementation

General Consideration

There are several design options to implement the project, for example, SP/CSP could be both server/client and both could have multiple threads, which might be better satisfying the requirement "**send data/request and receive data frame at the same time**". Based on lecture 9, multiplex using select function is used as main structure in both SP and CSP. While technically speaking, in this approach, reading from and writing into the socket happen sequentially instead of "at the same time."

Another simplification in the implementation is that the station number is assigned automatically by CSP according to the connecting sequence and available station spot. For example, the first connecting SP is assigned as SP 1, the second is SP 2, and the third as SP 3. At this moment if SP1 quits, and a new SP comes in, it becomes new SP 1 instead of SP 4.

Station Process (SP)

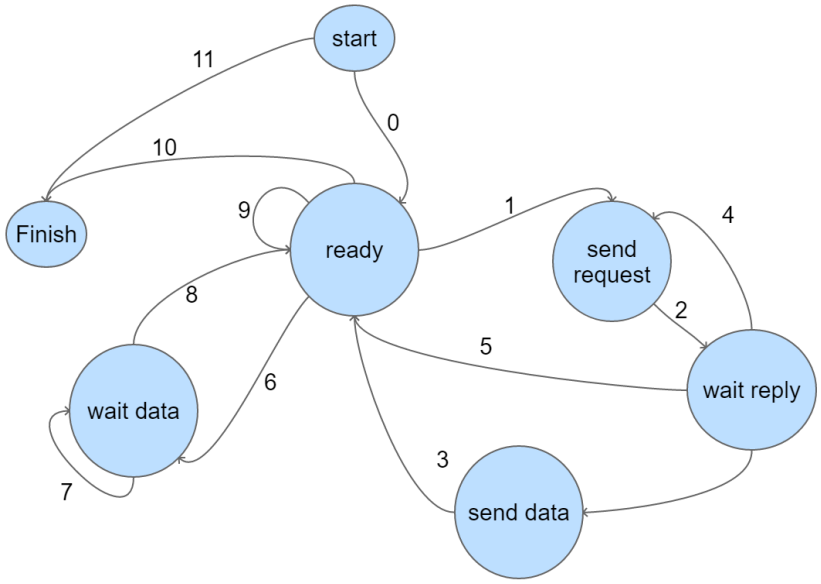
SP takes three parameters (server address, server port and input files) and starts the process with the following steps,

- Connect to server at specified address/port and gets the socket file descriptor,
- Open input file using standard library and gets File pointer and file descriptor,
- Use read set to monitor these two file descriptors,
    - When socket file descriptor is readable, read a frame from it.
    - When input file descriptor is readable, read a line/command from it and execute the command.
        - For Sending command, send request to CSP first and check for response,
            - For positive response, send the frame,
            - For negative response, wait 1 second and retransmit the request,
            - If no response(request is put in CSP request queue), then wait on reading from the socket.
        - For waiting command, just wait and read from the socket.

The input file is always readable, so the file handling logic controls the process, and waiting for response for CSP, sending frame to CSP, waiting for data frame from CSP are all inside the file handling logic. The socket file descriptor handling logic is basically used when the connection is established and when the station number is assigned by CSP. The implementation basically ensures that during the file handle logic, SP is still able to receive data from CSP.

Technically there is a risk in this implementation where if the file handling logic gets blocked in writing data frame to CSP, it might not have chance to keep reading. There is another approach, which is to monitor socket writable event as well and use state diagram to control the overall logic. The steps can be as following,

- Use state diagram to describe SP, and for each state, there are different actions and different monitor set,



Status	Description of status (action to do)	File descriptor Read set	File descriptor write set
Start	Initialize	-	-
Ready	Ready to read from file and socket,	Socket, File	-
Send request	Send request frame, stop reading file, ready to read from and write to Socket	Socket	Socket
Send data	send data frame, stop reading file, ready to read from and write to Socket	Socket	Socket
Wait reply	Wait to receive frame, stop reading file, ready to read from socket	Socket	-
Wait data	Wait to receive data frame, stop reading input file,	Socket	
Finish	End of process	-	-

- File readable event handler only reads the command, change state of SP accordingly, stop monitoring the input file until finishing this command,
- Socket writable event handler writes request and data frame to socket,
- Socket readable event handler only reads the response from CSP.

This approach is not chosen for the consideration of simplicity and readability.

Communication Switch Process (CSP)

The main structure of CSP is

- Start the server according to address and port in the parameter,
- Monitor readable event for server socket and all client sockets,
- When server socket is readable, assign the first available station number to the client, from 1 to 10 (CSP itself is considered as station 0) and add the new client socket into monitor set,
- When client socket is readable, receive the request frame or data frame and process them according to the queue status
- After either server socket or client socket read, reprocess the data queue and request queue.

According the requirement, the CSP maintains two queues, data queue and request queue, which are implemented as array of frames. By default, the number of frames each queue can hold (Actual queue size) is specified as a small number (4), so that it is relatively easier to trigger the 'reject' action with limited number of station (5). While CSP takes an optional parameter, which specifies the Actual queue size when required.

The socket readable event handler receives frames from each readable station. If this is a data frame, it is put into data queue. At this point, CSP dose not check the availability of the station, meaning even if the target station of the data frame is not connected to SCP, the data frame still goes into the data queue. If this is request frame, CSP sends response according to the status of two queues.

- If the data queue can still accommodate date frame, send Positive response directly. Note: one empty spot in data queue can only justify one Positive response, just like for one empty table, only one invitation can be issued. If multiple positive responses are sent for one empty spot, only the first coming data frame gets the spot. In this implementation, if the difference between the number of positive responses issued (invitation) and number of data frame received (guest already arrived) is less than the empty spots in data queue, then send positive response.
- If the data queue is full/fully reserved and there are empty spots in the request queue, just put the request into request queue. In this case, SP is blocked until getting the response from CSP.
- If both data queue and request queue are full, send negative response.

Queue processing logic is invoked in after handling each readable event on server or client. For each data frame in date queue, if the destination of the data frame is connected to CSP, send the data frame to the destination and remove the data frame from date queue. Otherwise the data frames stay in the queue. For each request in request queue, use the same logic above to check if the data queue can accommodate new data frame, if yes, send the positive reply to the sender of the request and remove the request from request queue.

Test Case 1

This test case is designed to test the basic functions.

It requires 3 SPs plus CSP. The steps are as following:

1. Start CSP (./CSP address port)
2. Start SP1 with input1.txt (./SP server port input1A.txt);
3. When SP1 gets blocked, start SP2 (./SP server port input2A.txt);
4. When SP2 get blocked, start SP3 (./SP server port input3A.txt);

SP1 (input1A.txt) Wait 100 frames	SP2 (input2A.txt) Send 50 frames to SP1, Wait 50 frames	SP3 (input3A.txt) Send 50 frames to SP1 And send 50 frames to SP2 alternatively
Wait for receiving 100 frames	Frame 1, To SP 1 Frame 2, To SP 1 … Frame 48, To SP 1 Frame 49, To SP 1 Frame 50, To SP 1 Wait for receiving 50 frames	Frame 1, To SP 1 Frame 2, To SP 2 Frame 3, To SP 1 Frame 4, To SP 2 .. Frame 99, To SP 1 Frame 100, To SP 2

Expected Behaviors

At the end of the test, the log of SPs shows

SP 1 receives 100 frames,

SP2 sends 50 frames, receives 50 frames,

SP3 sends 100 frames.

Test Case 2

This test case is designed to trigger the “reject” response from CSP.

It requires 6 SPs plus CSP. The steps are as following:

5. Start CSP (./CSP address port)
6. Start SP1 with input1.txt (./SP server port input1.txt);
7. When SP1 gets blocked, start SP2 (./SP server port input2.txt);
8. When SP1/2 get blocked, start SP3 (./SP server port input3.txt);
9. When SP1/2/3 get blocked, start SP4 (./SP server port input4.txt);
10. When SP1/2/3/4 get blocked, start SP5 (./SP server port input5.txt);
11. When SP1/2/3/4/5 get blocked, start SP6 (./SP server port input6.txt);

Contents of input files and blocking points

SP1 (input1.txt)	**SP2 (input2.txt)**	**SP3 (input3.txt)**	**SP4 (input4.txt)**	**SP5 (input5.txt)**	**SP6 (input6.txt)**
Frame 1, To SP 2 **Frame 2, To SP 2** **Frame 3, To SP 3** **Frame 4, To SP 3**					
Frame 5, To SP 2 **Frame 6, To SP 2** **Frame 7, To SP 3**	**Wait 4 frames** **Frame 1, To SP 6** **Frame 2, To SP 6** **Frame 3, To SP 6**	Wait 4 frames			
Frame 8, To SP 3 **Frame 9, To SP 6**	**Frame 4, To SP 6** **Wait 1 frames**	Frame 1, To SP 6 Wait f 1 frames	Frame 1, To SP 6 Wait 1 frames	Frame 1, To SP 6 Frame 2, To SP 6 Frame 3, To SP 6	
Frame 10, To SP 6 **Wait 1 frames**				Wait 1 frames	Frame 1, To SP 1 Frame 2, To SP 2 Frame 3, To SP 3 Frame 4, To SP 4 Frame 5, To SP 5

Expected Behaviors

Block Position	Description (Notes: 1->2 means SP 1 sends a frame to SP 2)
	After SP 1 starts, it sends 2 frames to SP 2 and 2 frames to SP 3, both of which are not online, and gets blocked after sending the request to send next frame to SP 2. Content of Data queue (Max size 4, count 4) (1->2, 1->2, 1->3, 1->3) Content of Request queue (Max size 4, count 1) (1->2)
	When SP 2 starts, CSP sends the frames in data queue to SP 2 and SP 1 gets unblocked and starts sending 2 more frames to SP 2 and 1 frame to SP3, after that gets blocked again. SP 2 receives 4 frames from SP 1 and starts sending 1 frame to SP 6, then gets blocked after that. Content of Data queue (Max size 4, count 4) (1->3, 1->3, 1->3, 2->6) Content of Request queue (Max size 4, count 2) (1->3, 2->6)
	When SP 3 kicks in, it consumes all the frames to SP 3 in the queue, and unblocks SP 1 and SP 2, both of which then starts sending frame to SP 6, which is offline, so they get blocked again at this position. Content of Data queue (Max size 4, count 4) (1->6, 2->6, 2->6, 2->6) Content of Request queue (Max size 4, count 3) (1->6, 2->6, 3->6)
	SP 4 kicks in by sending a frame request to SP 6 and gets blocked too. At this moment, all 4 SPs are blocked at sending request to SP 6, and both data queue and request queue of CSP are full. Content of Data queue (Max size 4, count 4) (1->6, 2->6, 2->6, 2->6) Content of Request queue (Max size 4, count 4) (1->6, 2->6, 3->6, 4->6)
	SP 5 gets in by trying to send 3 frames to SP 1, 2 and 3. Because all both data queue and request queue are full, SP 5 gets Reject response to the request. After retrying 3 times, SP 5 drops these 3 frames and stops at the last command waiting 1 frame, and this does not change the status of CSP and status of the other 4 SPs.
SP 6	When SP 6 kicks in, it first consumes all the frames in data queue and request queue and unblocks the other 5 SPs, each of which goes to its last command, waiting for 1 frame. Then SP 6 sends 1 frame to each of the 5 SPs, which brings all the SPs to the end and concludes the test case.

Test cases are tested on zeus.cs.txstate.edu and eros.cs.txstate.edu