[CS341] Introduction to Computer Networks
Project 3. KENSv3 Part 2
20170504 Juan Lee (Team. 45)

Introduction

This report is about the third programming assignment of CS341 Introduction to Computer Networks. The third assignment is to implement handshaking functions, open, accept, connect, getpeername, and closing mechanism of TCP. The details about this assignment is described on the KENS web site.

Before I start to explain my works, I want to say about my trial, which goes fail for the days I have tried. I couldn't pass any test cases suggested, and I hope I could implement those in following assignment.

Explain: getpeername

First function I implemented is getpeername. The mechanism of getpeername is simple and very similar to getsockname, however, I simply get connected information instead of its own information.

Explain: listen

Then, I implemented listen function, which has a role of changing the state of FSM and store backlogs given as parameters. It checks two conditions before storing backlog, one for checking the existence of socket, and the other for the duplication of listen call.

Explain: connect

Connect function is quite difficult. It checks two conditions, the existence of socket and the previous connection state of given socket. Then, it checks whether it is bound or not, if not, connect function automatically assign random port and local ip to socket, which is a mechanism called implicit bind.

Next step is handshaking. However, the handling of arrived packet is

responsibility of packetArrived function, so connect does only sending initial SYN signal. Of course, as we learned, I turned on the timer and tried to check TIME_OUT.

Note that there is no return when connect is called. I simply blocked by not to call returnSystemCall function, and store the information for that. After handshaking, it will be returned.

Explain: accept

Now, we move to next function, accept. Just like to connect, accept can enter the status of block. If accept is called and there are no established connection sockets, I blocked the function and deliver the information that "accept is called". Otherwise, return one connected socket with creating file descriptor and record it in the context list of TCPAssignment. In second case, we should call returnSystemCall since the function call is finished.

Explain: packetArrived

The problem is packetArrived. Since it has a structure of deterministic finite-state automata, and packetArrived is passively called, the role of packetArrived is very important. It checks the current state of socket, parse the packet, and does some actions needed.

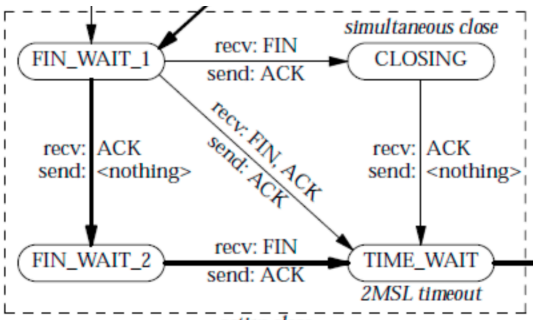
I considered three cases, SYN is coming, ACK is coming, and FIN is coming. Of course, I handled two or more bits are on, such as SYN and ACK are coming. There are two cases that SYN is coming. Firstly, server gets SYN bit on listen state when client wants to connect to the server. In this case, server should store them as pending clients for further connections, and return proper SYN + ACK packet.

```
SocketInfo* p_sock = new SocketInfo;
p_sock->other_base_seq = parsed.seq;
p_sock->setSignals(random_sequence_number++, parsed.seq+1, parsed.seq, parsed.ack);
p_sock->base_seq = p_sock->seq;
p_sock->bind(parsed.other_ip, parsed.other_port);
p_sock->setOtherIPandPort(parsed.ip, parsed.port);
p_sock->state = ST_SYN_RCVD;
w_sock->pending.push_back(p_sock);

// sending syn and ack
Packet* sa_packet = allocatePacket(54);
formPacket(sa_packet, parsed.other_ip, parsed.other_port, parsed.ip, parsed.port, p_sock->base_seq, p_sock->ack, 0x12);
sendPacket("IPv4", clonePacket(sa_packet));
```

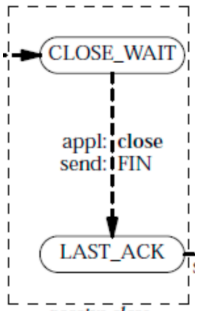
This is what I implemented for server side SYN coming scenario, even if I don't think it works well… Secondly, client gets SYN bit because of the first scenario. In this case, I am now only handling about SYN bit, made the state as SYN_RCVD and move it again to ESTABLISHED when ACK is checked. After SYN received, I have returned ACK packet. This is what I did:

```
// for client
if(w_sock->state == ST_SYN_SENT){
    w_sock->other_base_seq = parsed.seq;
    w_sock->setSignals(w_sock->seq, parsed.seq+1, parsed.seq, parsed.ack);
    w_sock->setIPandPort(parsed.other_ip, parsed.other_port);
    w_sock->setOtherIPandPort(parsed.ip, parsed.port);
    w_sock->state = ST_SYN_RCVD;
}

// re-send syn packet, ack to syn
Packet* packet = allocatePacket(54);
formPacket(packet, parsed.other_ip, parsed.other_port, parsed.ip, parsed.port, w_sock->base_seq + 1, w_sock->ack, 0x10);
sendPacket("IPv4", packet);
```

When ACK is coming, I firstly handle the case of connection process. Make the sockets in pending queue established, and if accept is called before, I returned the result here. Then, I moved to closing process, which frequently changes FIN_WAIT_1, FIN_WAIT_2, and TIME_WAIT states.



I tried to implement two scenarios of closing processes. Well, due to the coding ability of mine, the picture would explain better.



Lastly, FIN is coming. Just like its name, I implemented the states and changes of the states at the closing process. ACK coming scenario and FIN coming scenario together make above images possible.

Explain: timerCallback

Timer is call-backed when give time is expired. So, I put some WHAT_TO_DO, which is a kind of action plan for called. I implemented three cases, SYN and FIN retransmission and Closing process after some delay time.

Explain: And the others…

I, indeed, wrote some code for helping function like 'formPacket', but it does not affect the main cycle of TCP.

Connection Setup and Teardown

This is briefly explained in above, function explanation. When we actively call function, it decides whether it blocks the function or not. When it needs to block the function is that it should receive something from peer, however, only packetArrived function handles the arrived packet. Thus, I implemented packetArrived as a mediator of this FSM, and it checks and changes the states or conditions for those connection setups.

Teardown has somewhat different aspects, since it explicitly uses TIME_WAIT. So the timer is critical for teardown, and I focused on the implementation of change of states and timed handling.

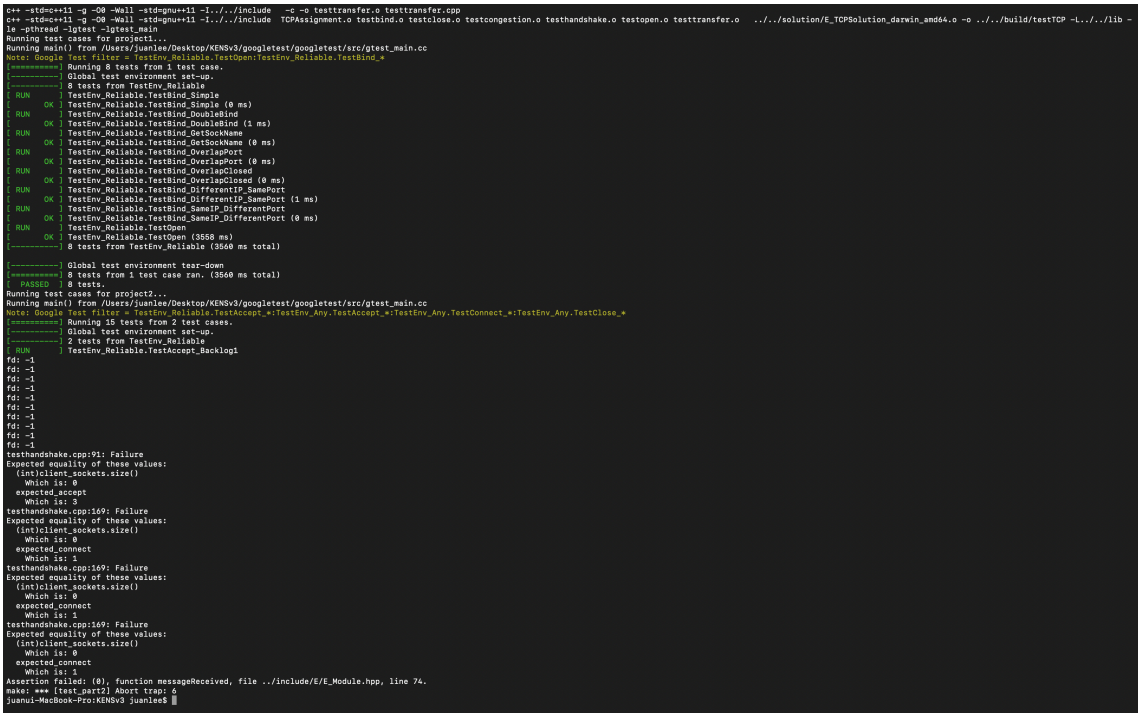
I tried to strictly follow the handshaking model in my code, so easily I can say connect, accept and packetArrived makes handshaking of connection setup, and close, packetArrived makes handshaking of teardown.

TIME_WAIT

Time wait was quite tricky for me. It is implemented with timerCallback function and some built-in timer functions. I made TimerArguments, which is delivered when timed. It has three components, action plan(what), socket information(info), and packet information(packet).

In TIME_WAIT State, it turned on the timer with some time interval, and wait until above callback is called. Then the callback function removes and cleans the socket delivered, and make it closed.

Screen shot



I found the exact position of error occurred, however, I just didn't know how to fix it... I will try it for the fourth assignment!