Lab3-Part1 report (Stop and Wait cTCP & Sliding Winodws and Network Services)

Joohan Lee(joohanl@usc.edu)

1. Program Structure and Design

(1) High-level structure

a) `ctcp_read()`

- Reads 1440bytes(MAX_SEG_DATA_SIZE) of STDIN data at each call.
- Create and sends segment including data.
    - If send window(a.k.a receive window) is not enough to send new data, stores the segment in waiting_segments(buffer in transmitter of linked_list_t). These segments will be sent after ACK is received and know receiver buffer is now available.

b) `ctcp_receive()`

- First, checkes if checksum of received segment is valid. If not, drop the segment.
- If the segment is FIN segment, transitions to the next TCP termination sequence and sends ACK for the received FIN or terminates if required.
- If the segment is ACK segment, acknowledge all already sent segments(in-flight segments) in transmission buffer(state->segments) by removing segments whose seqno is less than received ackno from state->segments. Also, send waiting_segments that were pending due to lack of send window.
- If the segment has new data(but not FIN), add the segment in the proper location of receiver buffer(state->received_segments) to follow in-order rule by sequence number.

c) `ctcp_output()`

- Output consecutive data(in sequence number) in receiver buffer. If there is hole in receiver buffer, stop output. After output, send ACK.

d) `ctcp_timer()`

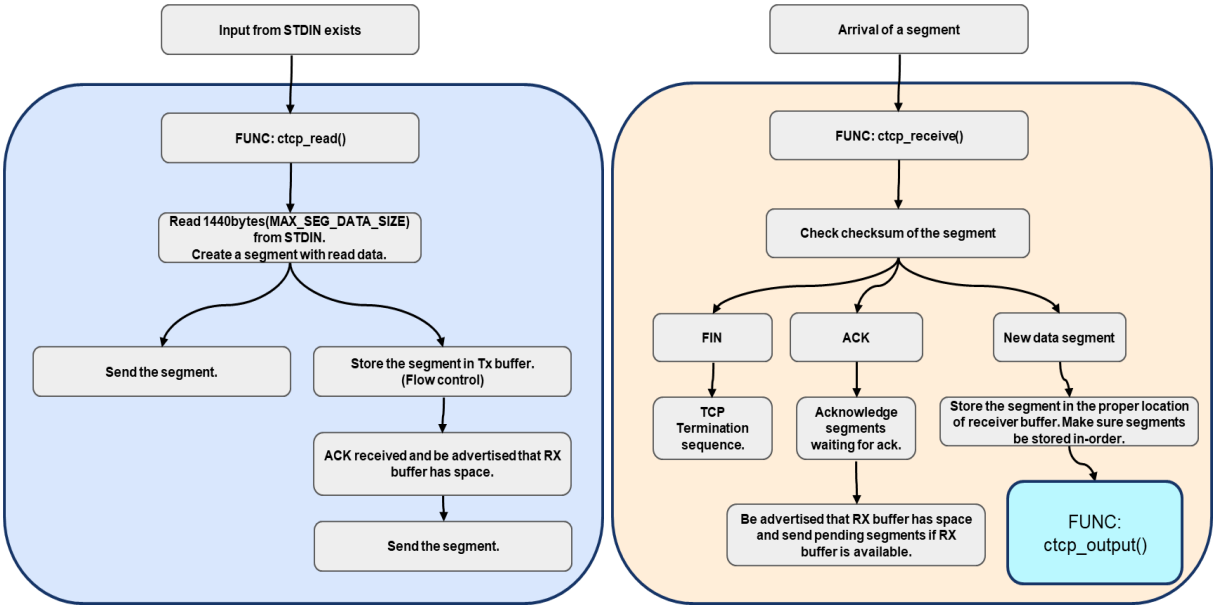
- If there are one or more connections, go through all connection states(state_list) to retransmit segments or tear down connections.
- While iterating through,
    - Tear down.(TIME_WAIT -> CLOSED)
        - increases elapesd time in TIME_WAIT state by config.timer(40ms). (=>time_wait_in_ms += config.timer)
        - after updating timer if time elapsed more than double MSL after starting TIME_WAIT state or there is no more segments to be sent in the host, tear down(= call ctcp_destroy).
    - Retransmission.
        - go through all sent transmissions in the current connection state.

- then, increases timer of each transmission for segment by config.timer(40ms).
- if it is time to retransmit(every 200(rt_timer) ms), retransmit the segment.
- if it is already sent up to 6 times in total, immediately tear down.

e) ctcp_init() and ctcp_destroy()

- `ctcp_init()`: Initialize required variables and data structures. For data structure details, refer to the (2) Used data structures.
- `ctcp_destroy()`: Free all dynamically used memory such as linked list and its objects. Ends client.

FLOW CHART OF ctcp_read() and ctcp_receive()



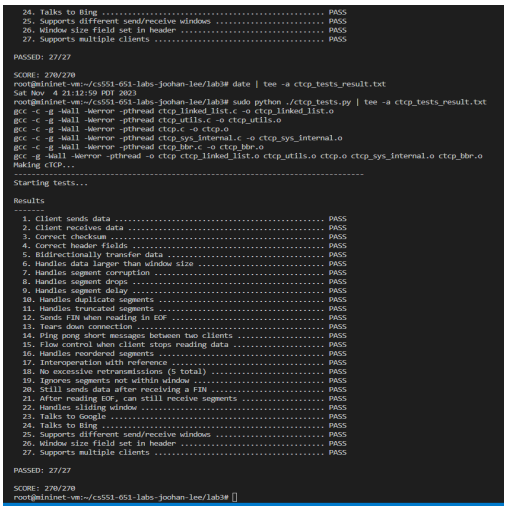
(2) Used data structures

- linked_list_t segments(w/ transmission info), waiting_segments(w/ transmission info), received_segments(ctcp_segment_t)
- ctcp_transmission_info_t includes
  - time elapsed from last transmission of the segment.
  - the number of transmission of this segment.
  - actual sent segment.
- segments
  - linked list that contains ctcp_transmission_info_t as its nodes. This includes what segments were sent, how many times the segment was sent, and how long it elapsed after the last transmission.
- waiting_segments
  - linked list that contains ctcp_transmission_info_t as its nodes. This includes segments that were created to be sent but was not sent due to lack of receiver buffer.
- received_segments
  - linked list that contains all received segments but were not pulled to application layer. Here, even though segments are received, they are not acked until they are be outputted.

2. Implementation Challenges

- Sending happens in a few functions, so it is easy to mistake missing counting up the number of transmissions. I used timer for tear down after 5 retransmissions as well as counts of transmissions to complement this possibility of mistakes.
- TCP termination state changes consecutively in multiple functions as well. So, it is hard to debug and understand the whole process.

3. Testing

- I tried all 27 test cases in ctcp_tests.py and passed all of them. Sometimes it fails, but in most case, I could pass all of them including basic, advanced, and hidden. Here is the snapshot of output for ctcp_tests.py:
  - 

- Since some of test cases randomly fail when keep testing, I analyzed the statistics of test cases in terms of pass or fail rate.
# of test sets	# of total test cases	# of passing test cases	# of failed test cases	Avaerage score
30	810	801	9	**98.8889%**
    - I ran ctcp_tests.py 30 times. Each test set means one running of ctcp_tests.py and has 27 test cases such as 'Client sends data', 'Handles segment drops', and 'Flow control when client stops reading data'. There is one unique test set.
  - The above results were made by parsing test logs. If the grader wants to refer, feel free to look at 'ctcp_tests_result.txt'.

4. Remaining Bugs

- When running ctcp_tests.py, one or two test cases very occasionally and randomly fail. I could not figure out if it is matter of my ctcp code, lack of timeout, or test code.
- I tried my best to fix bugs once I could notice them.