Project Setup

• Sign up for a GitHub account if you don’t have one yet.
• Get the invitation to access the skeleton code. https://classroom.github.com/a/t7ljk0Rw. It is also in the Piazza and project description document.
• Once accept the invitation, you will get a private repo.
• Clone the repo and do it yourself.
• If you are a freshman to Github, some tips for you to start with: https://guides.github.com/activities/hello-world/
Overview

• GOAL: Simulate the Link Layer network protocol
• 1. Implement communication between two or more hosts
• 2. Hosts implemented as Threads
• 3. 2 types of Hosts: Senders and Receivers
• 4. Practice implementing frame and error handling. (Lecture 2-4)
• 5. Messages can be corrupted and dropped.
• 6. Implement reliable transmission (Lecture 5)
Code Structure
Code Structure

Initiation

main.c

input.c
Code Structure

common.h
sender.c
Code Structure

Initiation

common.h
receiver.c
Code Structure

input.c
**Sender_t** (common.h)

```c
struct Sender_t {
    // DO NOT CHANGE:
    // 1) buffer_mutex
    // 2) buffer_cv
    // 3) input_cmdlist_head
    // 4) input_framelist_head
    // 5) send_id
    pthread_mutex_t buffer_mutex;
    pthread_cond_t buffer_cv;
    LLnode* input_cmdlist_head;
    LLnode* input_framelist_head;
    int send_id;
};
```

**run_sender** (sender.c)

1. Determine the next time the thread should wake up
2. Grab the mutex protecting the input_cmd/inframe queues
3. Dequeues messages from the input queue and adds them to the outgoing_frames list
4. Releases the lock
5. Sends out the messages
communicate.c

- Implements transporting of messages between the sender and receiver threads.
- Two main methods:
  - send_msg_to_receivers
  - Send_msg_to_senders
- Supports buffers of at most 64B
- Communication is broadcast based!
- Messages can be dropped!
- Messages can be corrupted!
Code Structure

Receive

common.h
receiver.c
Receiver_t (common.h)

```c
struct Receiver_t {
    // DO NOT CHANGE:
    // 1) buffer_mutex
    // 2) buffer_cv
    // 3) input_framelist_head
    // 4) recv_id
    pthread_mutex_t buffer_mutex;
    pthread_cond_t buffer_cv;
    LLnode* input_framelist_head;
    int recv_id;
};
```

run_receiver (receiver.c)

1. Determine the next time the thread should wake up if there is nothing in the incoming queue(s)
2. Grab the mutex protecting the input_msg queue
3. Dequeues messages from the input_msg queue and prints them
4. Releases the lock
5. Sends out any outgoing messages
Code Structure

 Acknowledge & Retransmission
How Thread Work

• What is Thread:

  • https://www.tutorialspoint.com/operating_system/os_multi_threading.htm#:~:text=A%20thread%20is%20a%20flow,which%20contains%20the%20execution%20history.&text=Each%20thread%20represents%20a%20separate%20flow%20of%20control.

• How Thread wait/signal function work in C:

  • https://www.geeksforgeeks.org/condition-wait-signal-multi-threading/
  • https://pubs.opengroup.org/onlinepubs/009696699/functions/pthread_cond_timedwait.html
Tasks

• Framing
• Error handling
• Acknowledgements
• Retransmission
Frame

• Envelop message into frame:
  • 1. Error detection
  • 2. Frame header: Who should get the message as the Communication is broadcast based

• Note: Leave at least 48 bytes for FRAME_PAYLOAD_SIZE, and the input message will not be longer than 48 bytes for this project. Meaning you do need to consider about splitting the message.

• Sender.c: handle_input_cmds(Sender* sender, LLnode** outgoing_frames_head_ptr)

• Common.h:

```
define MAX_FRAME_SIZE 64

// TODO: You should change this!
// Remember, your frame can be AT MOST 64 bytes!
define FRAME_PAYLOAD_SIZE 64
struct Frame_t {
    char data[FRAME_PAYLOAD_SIZE];
};
typedef struct Frame_t Frame;
```
Tasks

• Framing
• Error handling
• Acknowledgements
• Retransmission
Error handling

• Checksum function:
  • Lecture 3-4
  • Etc. Apply CRC bit

• Sender.c:
  • void handle_input_cmds(Sender* sender, LLnode** outgoing_frames_head_ptr)

• Receiver.c:
  • void handle_incoming_msgs(Receiver* receiver, LLnode** outgoing_frames_head_ptr)
Tasks

• Framing
• Error handling
• Acknowledgements
• Retransmission
Acknowledgements

• When the receiver get a correct message, tell the sender.

• Receiver.c:
  • void handle_incoming_msgs(Receiver* receiver, LLnode** outgoing_frames_head_ptr)

• Sender.c:
  • void handle_incoming_acks(Sender* sender, LLnode** outgoing_frames_head_ptr)
  • How to receive message?
    • handle_incomingmsgs() tells you
Tasks

• Framing
• Error handling
• Acknowledgements
• Retransmission
Retransmission

• Side-and-wait scheme.
  • Lecture 5
  • Do not use slide-window protocol!

• 1. `pthread_cond_timedwait(&sender->buffer_cv, &sender->buffer_mutex, &time_spec);` -> get the input from user/Receiver send the ack/ Time out

• 2. If Timeout still not get ack -> `handle_timedout_frames(sender, &outgoing_frames_head);` -> retransmit the packets.
  • a. If a message is lost in transit, your senders should retransmit it after waiting more than 0.085 seconds but less than 0.1 seconds. We recommend **0.09 seconds**.
Test

• Compile the skeleton code by typing: make
• Helping: ./tritontalk –h
• ./tritontalk -s 1 -r 2 start one sender two receivers with 0 message to be dropped and 0 message to be corrupted.
• ./tritontalk -s 1 -r 2 -d 0.3 will cause 30% of messages to be dropped.
• ./tritontalk -s 1 -r 2 -c 0.3 will cause 30% of messages to be corrupted.
• You are responsible for modifying the sender.c and receiver.c and common.h files

• You are not supposed to modify input.h, input.c, communicate.h, and communicate.c files. We will overwrite them after you have submitted your project.

• You may modify any of the other files in the repo, and add any additional files as necessary (also taking care to change the Makefile)