Readme Structure

Problem1

How to Run the program

- Run `sh script.sh` in one terminal, this will create the executables and run the server
- Run `./client` in another, this will run the client
- Make sure you have an `input.txt` file in the current directory
- After completion, following files will be created
  - destination_file.txt - File where server writes the output
  - client.log - Logs generated by client
  - server.log - logs generated by server
  - To view logs in sorted order run on a terminal following command
    - `sort *.log >> combinedLogs.log`
    - This will generate combined logs in sorted order of time in `combinedLogs.log`

Notes for running the code

- Change following parameters if required while testing
  - In server.h
    - Packet drop rate (PDR)
    - Number of out of order packets buffered (BUFFERSIZE)
    - server port number (SERVER_PORT)
    - server log file (SERVER_LOG_FILE)
    - destination_file (DESTINATION_FILE)
  - In client.h
    - input file (INPUT_FILE)
    - payload size (CHUNK_SIZE)
    - server IP (SERVER_IP)
    - server port number (SERVER_PORT)
    - timeout value (TIMEOUT_S)
    - max number of tries in case of not receiving ack (MAX_TRIES)
    - client log file (CLIENT_LOG_FILE)

Methadology

- **Client side**
  - Sender creates a child process and the parent and child create a TCP connection => open a channel
  - Both share file descriptor for the input file and hence if one reads some chunk of the file, the offsets automatically change in the other
  - They send the chunk read and now enter an infinite loop to wait for an ACK
  - They monitor the readability of the socket using select call
    - If select returns
      - 0 : the timeout has occured => ACK is lost and hence send the packet again
      - >0 : ACK has arrived
        - If ACK is corresponding to the packet sent, construct next packet to forward
        - Else out of order ACK has arrived, ignore this ACK and go in select loop again
- **Server side**
  - The server is a polling one.
  - Check readability on listening socket and adds the incoming connections to an array.
  - Also, check readability of the connections and when they are readable,
    - Receive the data packet
      - If the incoming sequence number is not expected one
        - Buffer the packet if space is available in queue
        - Else, reject the packet
      - Else,
        - Write this data to the file,
        - If there are any outstanding packets contiguous to this, transfer them to the file
    - Send an ACK back if packet is not rejected to same channel

Problem 2

How to run the program

- Run `sh script.sh` in one terminal, this will create the executables
  - Run `./server` in one terminal to run the server
  - Run `./relay 2` to run the relay number 2
  - Run `./relay 1` to run the relay number 1
  - Run `./client` in another, this will run the client
  - Make sure you have an `input.txt` file in the current directory
  - After completion, following files will be created
    - destination_file.txt - File where server writes the output
    - client.log - Logs generated by client
    - server.log - logs generated by server
    - relay1.log - Logs generated by relay1
    - relay2.log - Logs generated by relay2
  - To view logs in sorted order run on a terminal following command

        - `sort *.log >> combinedLogs.log`
        - This will generate combined logs in sorted order of time in `combinedLogs.log`

Notes for running it

- Change following parameters if required while testing
    - In server.h
        - Packet drop rate (PDR)
        - Number of out of order packets buffered (BUFFERSIZE)
        - server log file (SERVER_LOG_FILE)
        - destination_file (DESTINATION_FILE)
    - In client.h
        - input file (INPUT_FILE)
        - timeout value (TIMEOUT_MS)
        - client log file (CLIENT_LOG_FILE)
        - window size(WINDOW_SIZE)
    - In pktInfo.h
        - payload size (CHUNK_SIZE)
    - In common.h
        - IP and port numbers for
            - server (SERVER_PORT, SERVER_IP)
            - client (CLIENT_PORT, CLIENT_IP)
            - relay1 (RELAY1_PORT, RELAY1_IP)
            - relay2 (RELAY2_PORT, RELAY2_IP)
    - In relay.h
        - packet drop rate (PDR)
        - random delay upper limit (DELAY_UPPER_LIMIT_MS)
        - timeout value (TIMEOUT_S)
        - log files for relays (RELAY1_LOG_FILE, RELAY2_LOG_FILE)

Methodology

- **Client side**

    - Client creates packets for all elements of window
    - Sends all of them towards relays (odd numbered to relay2 and even numbered to relay1)
    - Waits for all ACKs in window to arrive using select. `select()` returns
        - >0 => some ACK has arrived, receive it and mark as received in the array
        - = 0 => timeout occured, resend all the packets whose ACKs were not received yet

- **Relay side**

    - Receive packet from c lient
    - Generate a random floating point number between 0-2
    - sleep for that time to introduce delay
    - randomly ignore packet (don't send ACK => drop it) according to PDR value
    - use a timed receive call to receive from server as server could reject the packet
    - send the received ack packet to client

- **Server side**
    - Receive data packet from relay
        - If the incoming sequence number is not expected one
            - Buffer the packet if space is available in queue
            - Else, reject the packet
        - Else,
            - Write this data to the file,
            - If there are any outstanding packets contiguous to this, transfer them to the file
    - Send an ACK back if packet is not rejected to same relay