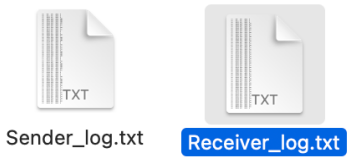
I used python to do this assignment. All the code is receiver.py and sender.py, and to make debug more visible, I add the 'print' commands after some key operations, which may be not required in the assignment.

The sender has three stages: sending the SYN and waiting for the SYN ack, sending the data, sending the FIN, and waiting for the FIN ack. The recvfrom method of the socket is in blocking mode, and in order to retransmit SYN,FIN, and data, I use Thread to handle concurrent operations such as retransmissions (retransmit_syn_thread, retransmit_data_thread, retransmit_fin_thread). These threads manage retransmission logic based on timeouts and state conditions, allowing the sender to operate different operation at the same time.

In the sending data phase, arguments flp and rlp is designed to control the segements if should be sent or received. For the sender, in the first step, we need to hand shake with the receiver. During the task , we need to open the receiver first, and then sender send a SYN to receiver and wait for a SYNACK. If random num smaller than flp, we will resend SYN(also in the DATA send and FIN send) . And if random num smaller than rlp, the sender will drop the ACK and wait for the receiver to send the ACK again(also in the DATA ACK and FIN ACK). Once the connection established, we start transfer the data. We set the MSS as 1000, so every time in a data segment we can store max to 1000 data chunk. Besides, I designed a sliding window. The sliding window is first filled and then sent or retransmitted. Until the data in front of the window is acknowledged, the data is removed, and the new data is filled in. When there is no data left to fill and the sliding window is empty, the data is transferred. After data send completely, we will send a FIN to receiver and wait for FINACK, and then we close the connection. And to record the log, after every step, the operation on the sender side will be write into the logfile.

Here are two Logfiles, and there is a part of logfile.



The screen shot of the result is as follow:

Sender.py

```
z5468081@vx14:~/Desktop$ python3 sender.py 59606 56007 asyoulik.txt 5000 100 0.1 0.1
SYN DOES NOT SEND
SYN SEND
SYN ACK
Established Connection.
Starting Send Data
Data Send Over.
FIN SEND
FINACK
```

```
1   snd 0.00    SYN 3719    0
2   rcv 0.29    ACK 3720    0
3   drp 0.31    DATA    3720    1000
4   snd 0.36    DATA    4720    1000
5   snd 0.37    DATA    5720    1000
6   snd 0.38    DATA    6720    1000
7   snd 0.39    DATA    7720    1000
8   rcv 0.48    ACK 5720    0
9   rcv 0.49    ACK 6720    0
```

The receiver records the start seqno of the current sliding window. First, we will start receiver earlier than sender in order to listen to the segments. In the start_receive() function, we will identify the type of segments we received. After we receive the SYN, we will send back a SYNACK and establish connection.

And then we will listen continuously. Normally we will receive a group of data segments. And every time we will send a ACK back. When the data is received, the position of the data should be calculated according to the seqno of the data and the start seqno of the sliding window. Once the sliding window has data in its first position, it is moved into an ordered array. Finally, the data in the ordered array is written to a file.

Finally, we will receive a FIN sent by sender, which means sender wants to end the connection. And then we will send a FINACK back.

And also in order to record the log, we have a write operation after each step of the operation.

The screen shot of the result is as follow:

Receiver.py

```
z5468081@vx14:~/Desktop$ python3 receiver.py 56007 59606 FileToReceive.txt 5000
Start Receive.
```

Receiver_log.txt

1	rcv 0.00	SYN 3719	0	
2	snd 0.04	ACK 3720	0	
3	rcv 0.40	DATA	4720	1000
4	snd 0.41	ACK 5720	0	
5	rcv 0.44	DATA	5720	1000
6	snd 0.46	ACK 6720	0	
7	rcv 0.48	DATA	6720	1000
8	snd 0.49	ACK 7720	0	
9	rcv 0.50	DATA	7720	1000
10	snd 0.51	ACK 8720	0	
11	rcv 100.46	DATA	3720	1000

Besides, we use ‘diff’ command to check the file which sender sent and the receiver received. We can find there is no difference between two files.

```
z5468081@vx14:~/Desktop$ diff asyoulik.txt FileToReceive.txt
z5468081@vx14:~/Desktop$
```

In this assignment, in the sender side most of function is implemented with thread, so that it can handle with the multiple tasks at the same time, and it make easier to control the timeout and other conditions.