**9331 program design**

Xiaowei Zhu z5102903

**Language: Python**

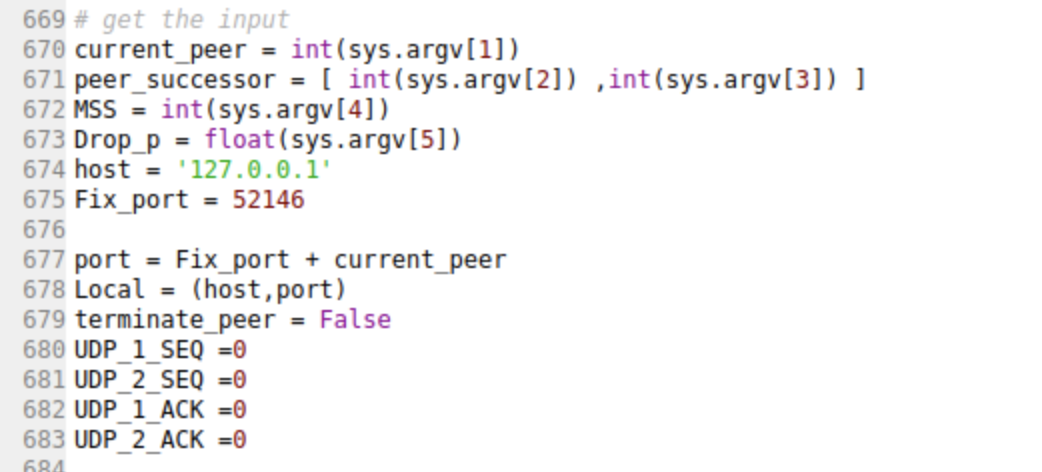
**Python version: 3.7.2**

**Youtube Demo**

<https://www.youtube.com/watch?v=WLdwb9Q2KYM>

**Port Selection:**

I choose 52146 to test my program, Frankly, it’s really hard for me to find that 50000 is available. So if you want to test the port 50000, please change it in 675 line, change the Fix\_port to 50000

****

**Whole Idea:**

Sys: used to get the input from terminal command

Socket: used to create UDP TCP sockets, send and reply relatively message(buyes)

Global variable: some global variables are used to identify the process, such as sequence, Tranfer\_process, successors, predecessors

Multi-threading: use Thread packet to create multi-threading in order to avoid confliction

Pickle package: used to package Packets and be transmitted through UDP

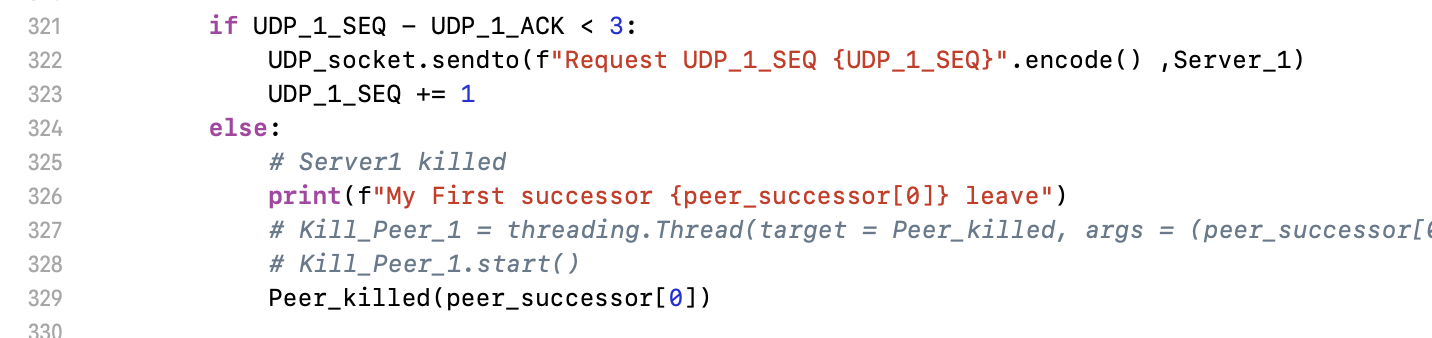
Time package: used to deal with the time out situation

Random package: used to simulated packet lost during UDP file transmission

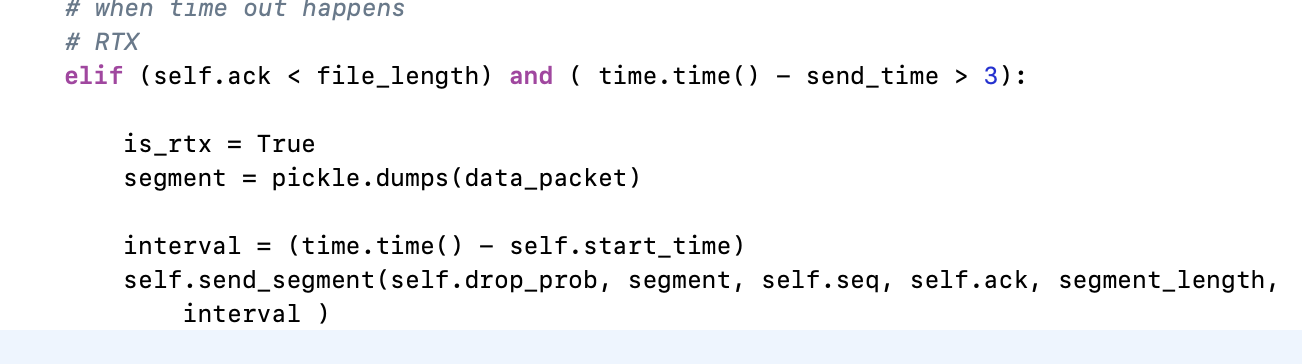
**Key variables:**

One UDP Ping message was sent per 10 seconds ( use time.sleep(10) )

During peer killed process, if peer doesn’t get 3 continuous ack from its successors, it will execute Peer\_killed() function in order to update successor



During UDP sending file, if Sender peer doesn’t get the response ack from request peer for 3 seconds, which is been set as a timeout period. This means the previous packet lost, it will be resent.



**Whole structure:**

At the beginning, I use normal functions to deal with peer ping message, peer departure and peer killed, but I found that make my program complex and hard to identify. Then I start to think about use Python Class to make the structure clear. So, I use class (def attributes and action for the file transfer part)

The whole key functions are below:

**UDP\_Send\_to( )** # used to send UDP ping message continuously

**UDP\_Recieve( )** # receive UDP message and reply and execute different program according to different message continuously

**Construct\_predecessor( )** # we need to use two predecessors during file transfer period, so here, we construct the two predecessors according to the UDP ping message.

**Moniter\_input( )** #this is used to monitor the input from terminal command, the quit message and the request file message

**Peer\_departure( )** #this is used to deal with the peer gracefully leaving, before it leaving, it sends messages to its predecessors and successors

**Peer\_killed( )** # this is used to deal with the peer leaving without any message, UDP ping can test with sequence and acknowledge numbers

**TCP\_Recieve( )** #this is used to receive the TCP messages, as requirement, we need to deal with some situation in TCP transmission. This function also includes the transferring file part. It receives, reply and execute different program according to different message

**Class W\_content** # used to accumulate the file segments from reply peer

**Class Packet** # package the whole information and use pickle function to deliver it with UDP

**Log( )** # used to record and write log information

**Class Send\_File\_Requester:** # this is used to send file request and identify is that file in my location, if not forward it to my successor.

**Class UDP\_Transfer\_File:** # as requirement, we use UDP to transfer file, this Class includes sequence, ack address, filename, threading and so on. It also contains open\_file, read\_fille, send\_segment, send\_file, receive\_ack actions. All these actions are used to deal with file transmission.

**Class UDP\_Recieve\_File:** # in requester part, we also need to build a socket to receive file transferring and return ack message relatively. Write\_in\_file action is also defined inside which is used to write file into a PDF document.