Name: Haowei Huang
Student id: z5247672

project report

1. Environment

Development environment: Python 3.7 on Mac by using Pycharm
Test environment: Python 3.7 on CSE VLab

2. How to run my code

To run the code ‘peer.py’, please simply run the script ‘test.sh’ in ‘assign.tar’ where content in the script is shown below:

```
xterm -hold -title "Peer 1" -e "python3 peer.py init 2 4 5 30" &
xterm -hold -title "Peer 3" -e "python3 peer.py  init 4 5 8 30" &
xterm -hold -title "Peer 4" -e "python3 peer.py  init 5 8 9 30" &
xterm -hold -title "Peer 5" -e "python3 peer.py  init 8 9 14 30" &
xterm -hold -title "Peer 8" -e "python3 peer.py init 9 14 19 30" &
xterm -hold -title "Peer 10" -e "python3 peer.py  init 14 19 2 30" &
xterm -hold -title "Peer 12" -e "python3 peer.py  init 19 2 4 30" &
```

Steps:

1. cd to the directory and type ‘chmod u+x test.sh’

2. run the script by typing ‘./test.sh’, then a DHT network is built

3. to join another peer, cd to the directory and type:

   ‘python3 peer.py join <PEER_ID> <KNOWN_PEER> <PING_INTERVAL>’

4. to kill a peer, simply press ctrl + c, or type ‘Quit’ for graceful leave

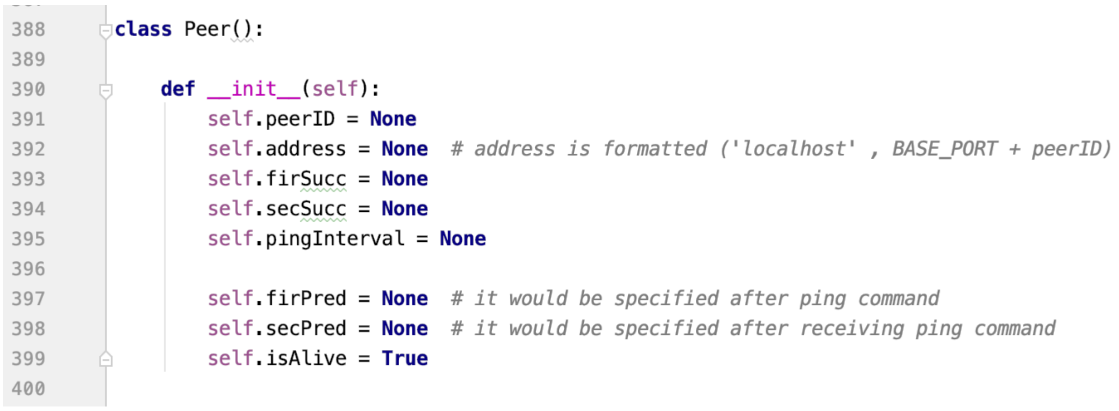
5. to store file, simply type ‘Store <FILE>’

6. to request file, simply type ‘Request <FILE>’

3. Program Structure

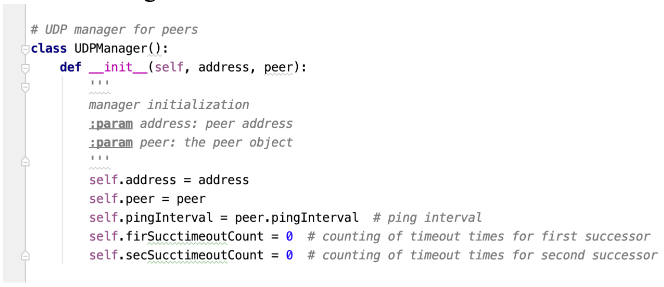
The coding style of the program is object oriented. There are 4 classes in total, Peer, TCPManager, UDPManager and GlobalParameter. To achieve various logic operation, the peer object call threads to process requestat at the backend. Screenshots below are the ‘def __init__(self)’ function for each class.

1. Peer



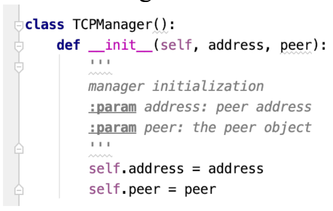
```
class Peer():

    def __init__(self):
        self.peerID = None
        self.address = None  # address is formatted ('localhost' , BASE_PORT + peerID)
        self.firSucc = None
        self.secSucc = None
        self.pingInterval = None

        self.firPred = None  # it would be specified after ping command
        self.secPred = None  # it would be specified after receiving ping command
        self.isAlive = True

```

2. UDPManager



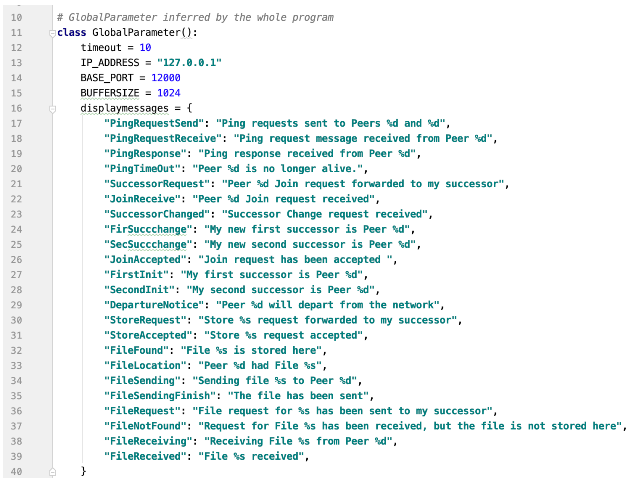
```python
# UDP manager for peers
class UDPManager():
    def __init__(self, address, peer):
        '''
        manager initialization
        :param address: peer address
        :param peer: the peer object
        '''
        self.address = address
        self.peer = peer
        self.pingInterval = peer.pingInterval  # ping interval
        self.firSucctimeoutCount = 0  # counting of timeout times for first successor
        self.secSucctimeoutCount = 0  # counting of timeout times for second successor
```

3. TCPManager



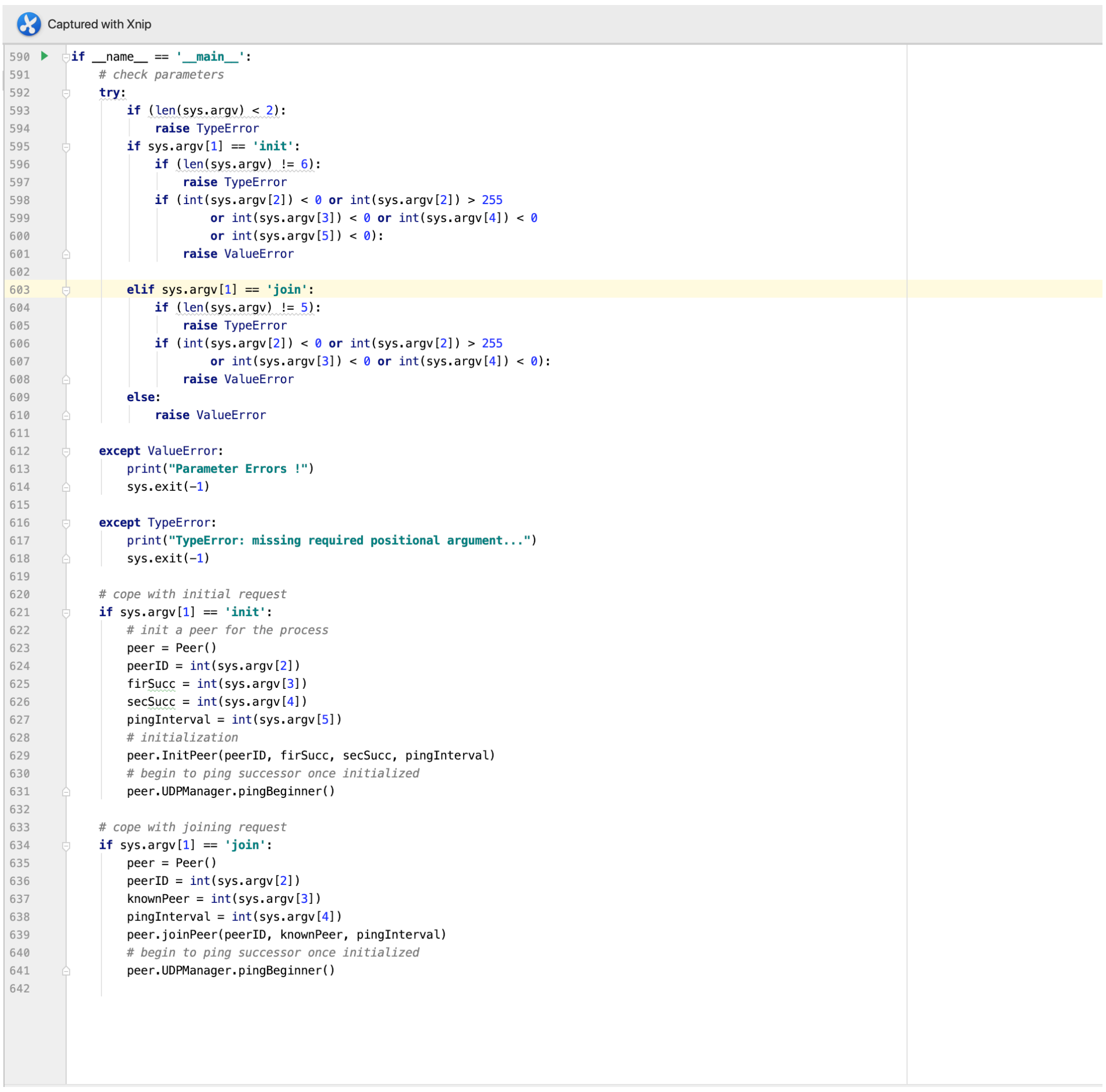
```python
class TCPManager():
    def __init__(self, address, peer):
        '''
        manager initialization
        :param address: peer address
        :param peer: the peer object
        '''
        self.address = address
        self.peer = peer
```

4. GlobalParameter



```python
# GlobalParameter inferred by the whole program
class GlobalParameter():
    timeout = 10
    IP_ADDRESS = "127.0.0.1"
    BASE_PORT = 12000
    BUFFERSIZE = 1024
    displaymessages = {
        "PingRequestSend": "Ping requests sent to Peers %d and %d",
        "PingRequestReceive": "Ping request message received from Peer %d",
        "PingResponse": "Ping response received from Peer %d",
        "PingTimeOut": "Peer %d is no longer alive.",
        "SuccessorRequest": "Peer %d Join request forwarded to my successor",
        "JoinReceive": "Peer %d Join request received",
        "SuccessorChanged": "Successor Change request received",
        "FirSuccchange": "My new first successor is Peer %d",
        "SecSuccchange": "My new second successor is Peer %d",
        "JoinAccepted": "Join request has been accepted ",
        "FirstInit": "My first successor is Peer %d",
        "SecondInit": "My second successor is Peer %d",
        "DepartureNotice": "Peer %d will depart from the network",
        "StoreRequest": "Store %s request forwarded to my successor",
        "StoreAccepted": "Store %s request accepted",
        "FileFound": "File %s is stored here",
        "FileLocation": "Peer %d had File %s",
        "FileSending": "Sending file %s to Peer %d",
        "FileSendingFinish": "The file has been sent",
        "FileRequest": "File request for %s has been sent to my successor",
        "FileNotFound": "Request for File %s has been received, but the file is not stored here",
        "FileReceiving": "Receiving File %s from Peer %d",
        "FileReceived": "File %s received",
    }
```

Also, there are two entries for the code, for ‘init’ and ‘join’ repectively. Before that I have already verify the validity of the parameters entered by users.


Captured with Xnip

```python
if __name__ == '__main__':
    # check parameters
    try:
        if (len(sys.argv) < 2):
            raise TypeError
        if sys.argv[1] == 'init':
            if (len(sys.argv) != 6):
                raise TypeError
            if (int(sys.argv[2]) < 0 or int(sys.argv[2]) > 255
                    or int(sys.argv[3]) < 0 or int(sys.argv[4]) < 0
                    or int(sys.argv[5]) < 0):
                raise ValueError

        elif sys.argv[1] == 'join':
            if (len(sys.argv) != 5):
                raise TypeError
            if (int(sys.argv[2]) < 0 or int(sys.argv[2]) > 255
                    or int(sys.argv[3]) < 0 or int(sys.argv[4]) < 0):
                raise ValueError
        else:
            raise ValueError

    except ValueError:
        print("Parameter Errors !")
        sys.exit(-1)

    except TypeError:
        print("TypeError: missing required positional argument...")
        sys.exit(-1)

    # cope with initial request
    if sys.argv[1] == 'init':
        # init a peer for the process
        peer = Peer()
        peerID = int(sys.argv[2])
        firSucc = int(sys.argv[3])
        secSucc = int(sys.argv[4])
        pingInterval = int(sys.argv[5])
        # initialization
        peer.InitPeer(peerID, firSucc, secSucc, pingInterval)
        # begin to ping successor once initialized
        peer.UDPManager.pingBeginner()

    # cope with joining request
    if sys.argv[1] == 'join':
        peer = Peer()
        peerID = int(sys.argv[2])
        knownPeer = int(sys.argv[3])
        pingInterval = int(sys.argv[4])
        peer.joinPeer(peerID, knownPeer, pingInterval)
        # begin to ping successor once initialized
        peer.UDPManager.pingBeginner()
```

4. Possible Improvement

1. In the code, I use a lot of text process to cope with the inter communication among peers. Thanks to the convenience and powerful built-in functions of python, I finish all tasks. However, it seems not wise to do that. For instead, I can simply use code to imply different messages. That would promote the whole program.

2. Also, as I'm not that familiar with object-oriented programming. This program doesn't strictly follow the object-oriented style. I know that this is essential for python(java) programming and engineering project. So I would try to adjust the structure of my program to be more efficient and reader-friendly if I have more time.

3. For convenience, I simply use the library in python. To get more familiar with TCP and UDP protocols. I should have tried to stimulated the mechanisms in them, like various flags(FIN, SYN), three way handshake and stop-and-wait and etc. Of course, python have done everything for me.