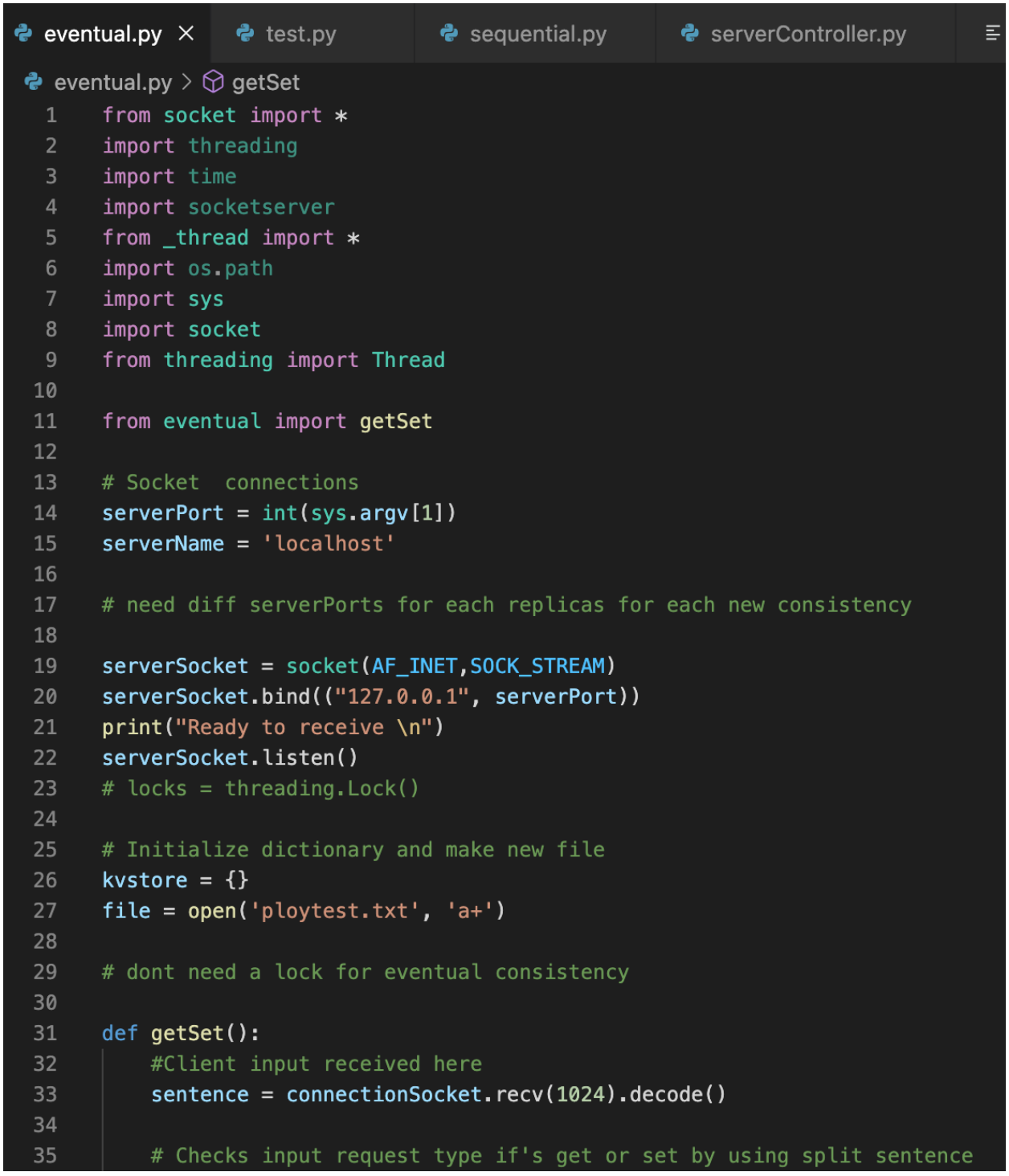
The serverController.py was intended to be the driver code so you can pick between evential, sequential, etc. But that was not finished.

Client.py is almost the same as the first KV store:

client.py > ...

```python
from fileinput import filename
from socket import *
from curses import raw
import sys

# Port and server info
serverName = 'localhost'
serverPort = int(sys.argv[1])

#Connecting
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))

# User input
sentence = input(str(r'Input set/get space key space value:')).encode("utf-8")
# set 8 ploy

#sending it to sever
clientSocket.send(sentence)

#This is where you receive info from server after connecting and sending to server
modifiedSentence = clientSocket.recv(1024)
removeString = str(modifiedSentence, 'utf-8')
print('From Server: ', removeString)
clientSocket.close()
```

Eventual.py is the eventual consistency one. It starts with the socket connections and initializes a dictionary and makes a new file called ploytest.txt to test it.



In the getSet() method, It splits up the request type and writes it into the kvstore and encodes it. To connect it to different servers, there is a for loop that reads each line in serverPorts.txt and sends that key value to the different ports.

```python
def getSet():
    #Client input received here
    sentence = connectionSocket.recv(1024).decode()

    # Checks input request type if's get or set by using split sentence
    request_type = sentence.split()[0]
    print(str(request_type))
    if str(request_type) == "set":

        keyword = sentence.split()[1]
        value = sentence.split()[2]

       # print(keyword, value)
        ## set ops
        #print(str(request_type))
        kvstore[keyword] = value

        # locks.acquire()

        file.write('%s:%s\n' % (keyword, value))

        print(kvstore[keyword])
        connectionSocket.send('successfully saved!'.encode())

        # read server port numbers and send to key value to that port
        f = open('serverPorts.txt', "r")
        for x in f:
            clientSocket = socket(AF_INET, SOCK_STREAM)
            clientSocket.connect((serverName, x))
            print(x)
        f.close

        # locks.release()
```

serverPorts.txt

```
1207
1206
1205
```

Finally, you create a new thread at the very end:

```
# You get connected here
while 1:
    connectionSocket, addr = serverSocket.accept()
    new_thread = Thread(getSet, connectionSocket)
    new_thread.start()
    new_thread.join()
```

Here is one test I did:

test.py > ...

```
import socket
from threading import Thread

from eventual import getSet

HOST = "127.0.0.1"  # Standard loopback interface address (localhost)
PORT = 65432  # Port to listen on (non-privileged ports are > 1023)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    while True:
        connectionSocket, addr = s.accept()
        new_thread = Thread(getSet, connectionSocket)
        new_thread.start()
        new_thread.join()

```