**DS Lab Week #6**

Angad Sandhu

190905494

1/04/2022

**solved questions**

**1)**

**#server.py**

import socket

import datetime

import time

# function used to initiate the Clock Server

def initiateClockServer():

s = socket.socket()

print("Socket successfully created")

# Server port

port = 8011

s.bind(('', port))

# Start listening to requests

s.listen(5)

print("Socket is listening...")

# Clock Server Running forever

while True:

# Establish connection with client

connection, address = s.accept()

print('Server connected to', address)

# Respond the client with server clock time

connection.send(str(datetime.datetime.now()).encode())

# Close the connection with the client process

connection.close()

# Driver function

if \_\_name\_\_ == '\_\_main\_\_':

# Trigger the Clock Server

initiateClockServer()

**#client.py**

import socket

import datetime

from dateutil import parser

from timeit import default\_timer as timer

# function used to Synchronize client process time

def synchronizeTime():

s = socket.socket()

# Server port

port = 8011

# connect to the clock server on local computer

s.connect(('127.0.0.1', port))

request\_time = timer()

# receive data from the server

server\_time = parser.parse(s.recv(1024).decode())

response\_time = timer()

actual\_time = datetime.datetime.now()

print("Time returned by server: " + str(server\_time))

process\_delay\_latency = response\_time - request\_time

print("Process Delay latency: " + str(process\_delay\_latency) + " seconds")

print("Actual clock time at client side: " + str(actual\_time))

# synchronize process client clock time

client\_time = server\_time + datetime.timedelta(seconds = (process\_delay\_latency) / 2)

print("Synchronized process client time: " + str(client\_time))

# calculate synchronization error

error = actual\_time - client\_time

print("Synchronization error : " + str(error.total\_seconds()) + " seconds")

s.close()

# Driver function

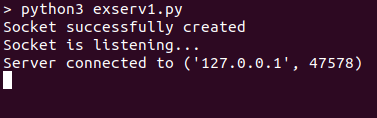
if \_\_name\_\_ == '\_\_main\_\_':

# synchronize time using clock server

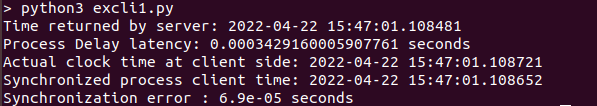
synchronizeTime()

**Output:**

**Server**



**Client**



**2)**

**#server.py**

from functools import reduce

from dateutil import parser

import threading

import datetime

import socket

import time

# datastructure used to store client address and clock data

client\_data = {}

''' nested thread function used to receive

clock time from a connected client '''

def startRecieveingClockTime(connector, address):

while True:

# recieve clock time

clock\_time\_string = connector.recv(1024).decode()

clock\_time = parser.parse(clock\_time\_string)

clock\_time\_diff = datetime.datetime.now() - \

clock\_time

client\_data[address] = {

"clock\_time" : clock\_time,

"time\_difference" : clock\_time\_diff,

"connector" : connector

}

print("Client Data updated with: "+ str(address),

end = "\n\n")

time.sleep(5)

''' master thread function used to open portal for

accepting clients over given port '''

def startConnecting(master\_server):

# fetch clock time at slaves / clients

while True:

# accepting a client / slave clock client

master\_slave\_connector, addr = master\_server.accept()

slave\_address = str(addr[0]) + ":" + str(addr[1])

print(slave\_address + " got connected successfully")

current\_thread = threading.Thread(

target = startRecieveingClockTime,

args = (master\_slave\_connector,

slave\_address, ))

current\_thread.start()

# subroutine function used to fetch average clock difference

def getAverageClockDiff():

current\_client\_data = client\_data.copy()

time\_difference\_list = list(client['time\_difference']

for client\_addr, client

in client\_data.items())

sum\_of\_clock\_difference = sum(time\_difference\_list, \

datetime.timedelta(0, 0))

average\_clock\_difference = sum\_of\_clock\_difference \

/ len(client\_data)

return average\_clock\_difference

def synchronizeAllClocks():

while True:

print("New synchroniztion cycle started.")

print("Number of clients to be synchronized: " + \

str(len(client\_data)))

if len(client\_data) > 0:

average\_clock\_difference = getAverageClockDiff()

for client\_addr, client in client\_data.items():

try:

synchronized\_time = \

datetime.datetime.now() + \

average\_clock\_difference

client['connector'].send(str(

synchronized\_time).encode())

except Exception as e:

print("Something went wrong while " + \

"sending synchronized time " + \

"through " + str(client\_addr))

else :

print("No client data." + \

" Synchronization not applicable.")

print("\n\n")

time.sleep(5)

# function used to initiate the Clock Server / Master Node

def initiateClockServer(port = 8080):

master\_server = socket.socket()

master\_server.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

print("Socket at master node created successfully\n")

master\_server.bind(('', port))

# Start listening to requests

master\_server.listen(10)

print("Clock server started...\n")

# start making connections

print("Starting to make connections...\n")

master\_thread = threading.Thread(

target = startConnecting,

args = (master\_server, ))

master\_thread.start()

# start synchroniztion

print("Starting synchronization parallely...\n")

sync\_thread = threading.Thread(

target = synchronizeAllClocks,

args = ())

sync\_thread.start()

# Driver function

if \_\_name\_\_ == '\_\_main\_\_':

# Trigger the Clock Server

initiateClockServer(port = 8080)

**#client.py**

from timeit import default\_timer as timer

from dateutil import parser

import threading

import datetime

import socket

import time

# client thread function used to send time at client side

def startSendingTime(slave\_client):

while True:

# provide server with clock time at the client

slave\_client.send(str(

datetime.datetime.now()).encode())

print("Recent time sent successfully",

end = "\n\n")

time.sleep(5)

# client thread function used to receive synchronized time

def startReceivingTime(slave\_client):

while True:

# receive data from the server

Synchronized\_time = parser.parse(

slave\_client.recv(1024).decode())

print("Synchronized time at the client is: " + \

str(Synchronized\_time),

end = "\n\n")

# function used to Synchronize client process time

def initiateSlaveClient(port = 8080):

slave\_client = socket.socket()

# connect to the clock server on local computer

slave\_client.connect(('127.0.0.1', port))

# start sending time to server

print("Starting to receive time from server\n")

send\_time\_thread = threading.Thread(

target = startSendingTime,

args = (slave\_client, ))

send\_time\_thread.start()

# start recieving synchronized from server

print("Starting to recieving " + \

"synchronized time from server\n")

receive\_time\_thread = threading.Thread(

target = startReceivingTime,

args = (slave\_client, ))

receive\_time\_thread.start()

# Driver function

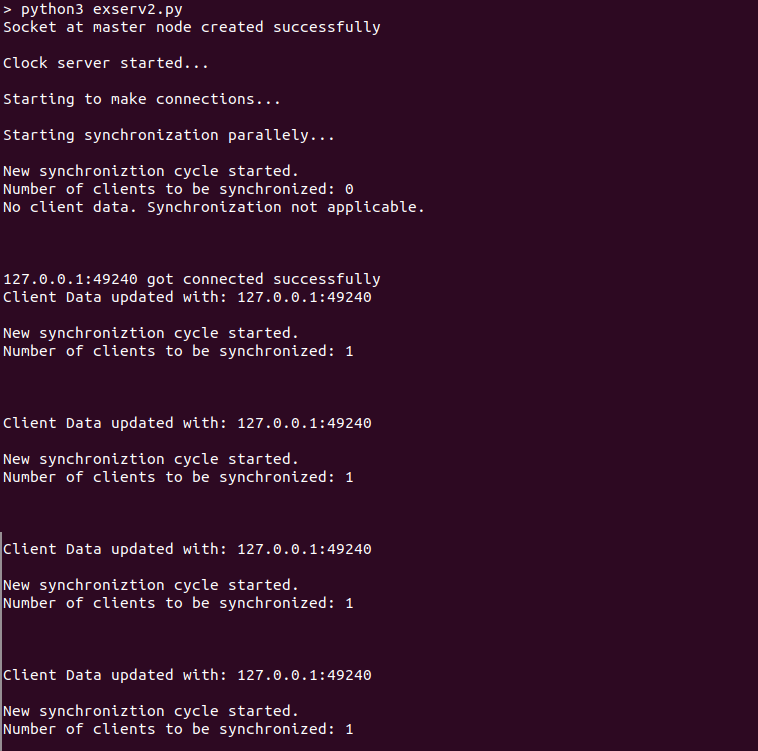
if \_\_name\_\_ == '\_\_main\_\_':

# initialize the Slave / Client

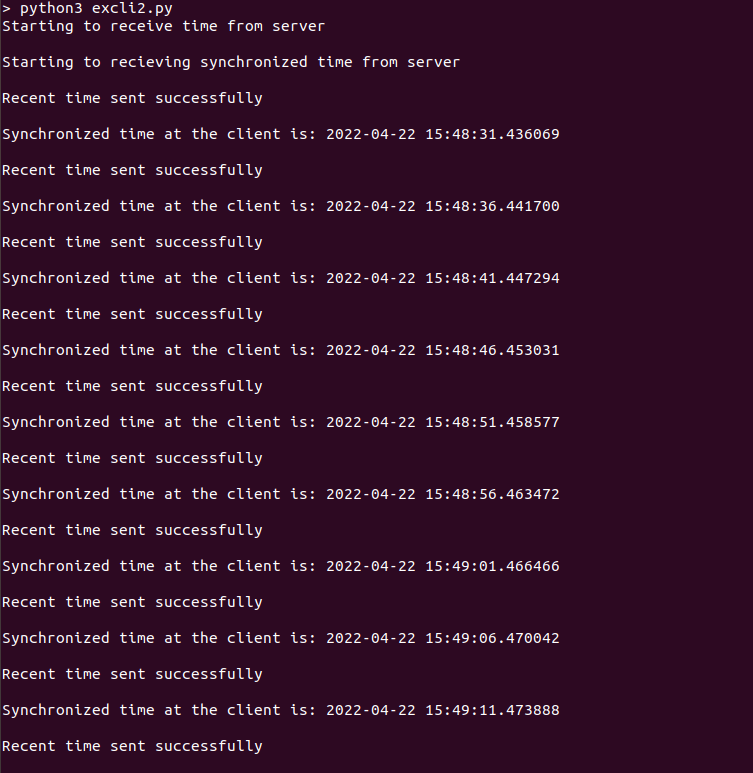
initiateSlaveClient(port = 8080)

**Output:**

**Server**



**Client**



**exersize questions**

**1)**

**#server.py**

from functools import reduce

from dateutil import parser

import threading

import datetime

import socket

import time

client\_data = {}

def startRecieveingClockTime(connector, address):

while True:

clock\_time\_string = connector.recv(1024).decode()

clock\_time = parser.parse(clock\_time\_string)

clock\_time\_diff = datetime.datetime.now() - clock\_time

client\_data[address] = {

"clock\_time": clock\_time,

"time\_difference": clock\_time\_diff,

"connector": connector

}

print("Client Data updated with: " + str(address), end="\n\n")

time.sleep(5)

def startConnecting(master\_server):

while True:

master\_slave\_connector, addr = master\_server.accept()

slave\_address = str(addr[0]) + ":" + str(addr[1])

print(slave\_address + " got connected successfully")

current\_thread = threading.Thread(

target=startRecieveingClockTime,

args=(master\_slave\_connector, slave\_address, ))

current\_thread.start()

def getAverageClockDiff():

current\_client\_data = client\_data.copy()

time\_difference\_list = list(client['time\_difference']

for client\_addr, client in client\_data.items())

sum\_of\_clock\_difference = sum(

time\_difference\_list, datetime.timedelta(0, 0))

average\_clock\_difference = sum\_of\_clock\_difference / len(client\_data)

return average\_clock\_difference

def synchronizeAllClocks():

while True:

print("New synchroniztion cycle started.")

print("Number of clients to be synchronized: " + str(len(client\_data)))

if len(client\_data) > 0:

average\_clock\_difference = getAverageClockDiff()

for client\_addr, client in client\_data.items():

try:

synchronized\_time = datetime.datetime.now() + average\_clock\_difference

client['connector'].send(str(synchronized\_time).encode())

except Exception as e:

print("Something went wrong while sending synchronized time through" + str(client\_addr))

else:

print("No client data. Synchronization not applicable.")

print("\n\n")

time.sleep(5)

def initiateClockServer(port=8059):

master\_server = socket.socket()

master\_server.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

print("The Manipal Foodie\n")

master\_server.bind(('', port))

master\_server.listen(10)

print("Clock server print\n")

print("Connecitng to production lines...\n")

master\_thread = threading.Thread(

target=startConnecting,

args=(master\_server, ))

master\_thread.start()

print("Starting synchronization parallely...\n")

sync\_thread = threading.Thread(

target=synchronizeAllClocks,

args=())

sync\_thread.start()

if \_\_name\_\_ == '\_\_main\_\_':

initiateClockServer(port=8059)

**#client1.py**

from timeit import default\_timer as timer

from dateutil import parser

import threading

import datetime

import socket

import time

def startSendingTime(slave\_client):

while True:

slave\_client.send(str(datetime.datetime.now()).encode())

print("KMC time sent successfully", end="\n\n")

time.sleep(5)

def startReceivingTime(slave\_client):

while True:

Synchronized\_time = parser.parse(slave\_client.recv(1024).decode())

print("Synchronized time at the client is: " +

str(Synchronized\_time), end="\n\n")

def initiateSlaveClient(port=8059):

slave\_client = socket.socket()

slave\_client.connect(('127.0.0.1', port))

print("Starting to receive time from server\n")

send\_time\_thread = threading.Thread(

target=startSendingTime,

args=(slave\_client, ))

send\_time\_thread.start()

print("Starting to recieving synchronized time from server\n")

receive\_time\_thread = threading.Thread(

target=startReceivingTime,

args=(slave\_client, ))

receive\_time\_thread.start()

if \_\_name\_\_ == '\_\_main\_\_':

initiateSlaveClient(port=8059)

**#client2.py**

from timeit import default\_timer as timer

from dateutil import parser

import threading

import datetime

import socket

import time

def startSendingTime(slave\_client):

while True:

slave\_client.send(str(datetime.datetime.now()).encode())

print("MIT time sent successfully", end="\n\n")

time.sleep(5)

def startReceivingTime(slave\_client):

while True:

Synchronized\_time = parser.parse(slave\_client.recv(1024).decode())

print("Synchronized time at the client is: " +

str(Synchronized\_time), end="\n\n")

def initiateSlaveClient(port=8059):

slave\_client = socket.socket()

slave\_client.connect(('127.0.0.1', port))

print("Starting to receive time from server\n")

send\_time\_thread = threading.Thread(

target=startSendingTime,

args=(slave\_client, ))

send\_time\_thread.start()

print("Starting to recieving synchronized time from server\n")

receive\_time\_thread = threading.Thread(

target=startReceivingTime,

args=(slave\_client, ))

receive\_time\_thread.start()

if \_\_name\_\_ == '\_\_main\_\_':

initiateSlaveClient(port=8059)

**#client3.py**

from timeit import default\_timer as timer

from dateutil import parser

import threading

import datetime

import socket

import time

def startSendingTime(slave\_client):

while True:

slave\_client.send(str(datetime.datetime.now()).encode())

print("TAPMI time sent successfully", end="\n\n")

time.sleep(5)

def startReceivingTime(slave\_client):

while True:

Synchronized\_time = parser.parse(slave\_client.recv(1024).decode())

print("Synchronized time at the client is: " +

str(Synchronized\_time), end="\n\n")

def initiateSlaveClient(port=8059):

slave\_client = socket.socket()

slave\_client.connect(('127.0.0.1', port))

print("Starting to receive time from server\n")

send\_time\_thread = threading.Thread(

target=startSendingTime,

args=(slave\_client, ))

send\_time\_thread.start()

print("Starting to recieving synchronized time from server\n")

receive\_time\_thread = threading.Thread(

target=startReceivingTime,

args=(slave\_client, ))

receive\_time\_thread.start()

if \_\_name\_\_ == '\_\_main\_\_':

initiateSlaveClient(port=8059)

**#client4.py**

from timeit import default\_timer as timer

from dateutil import parser

import threading

import datetime

import socket

import time

def startSendingTime(slave\_client):

while True:

slave\_client.send(str(datetime.datetime.now()).encode())

print("SOLS time sent successfully", end="\n\n")

time.sleep(5)

def startReceivingTime(slave\_client):

while True:

Synchronized\_time = parser.parse(slave\_client.recv(1024).decode())

print("Synchronized time at the client is: " + str(Synchronized\_time), end="\n\n")

def initiateSlaveClient(port=8059):

slave\_client = socket.socket()

slave\_client.connect(('127.0.0.1', port))

print("Starting to receive time from server\n")

send\_time\_thread = threading.Thread(

target=startSendingTime,

args=(slave\_client, ))

send\_time\_thread.start()

print("Starting to recieving synchronized time from server\n")

receive\_time\_thread = threading.Thread(

target=startReceivingTime,

args=(slave\_client, ))

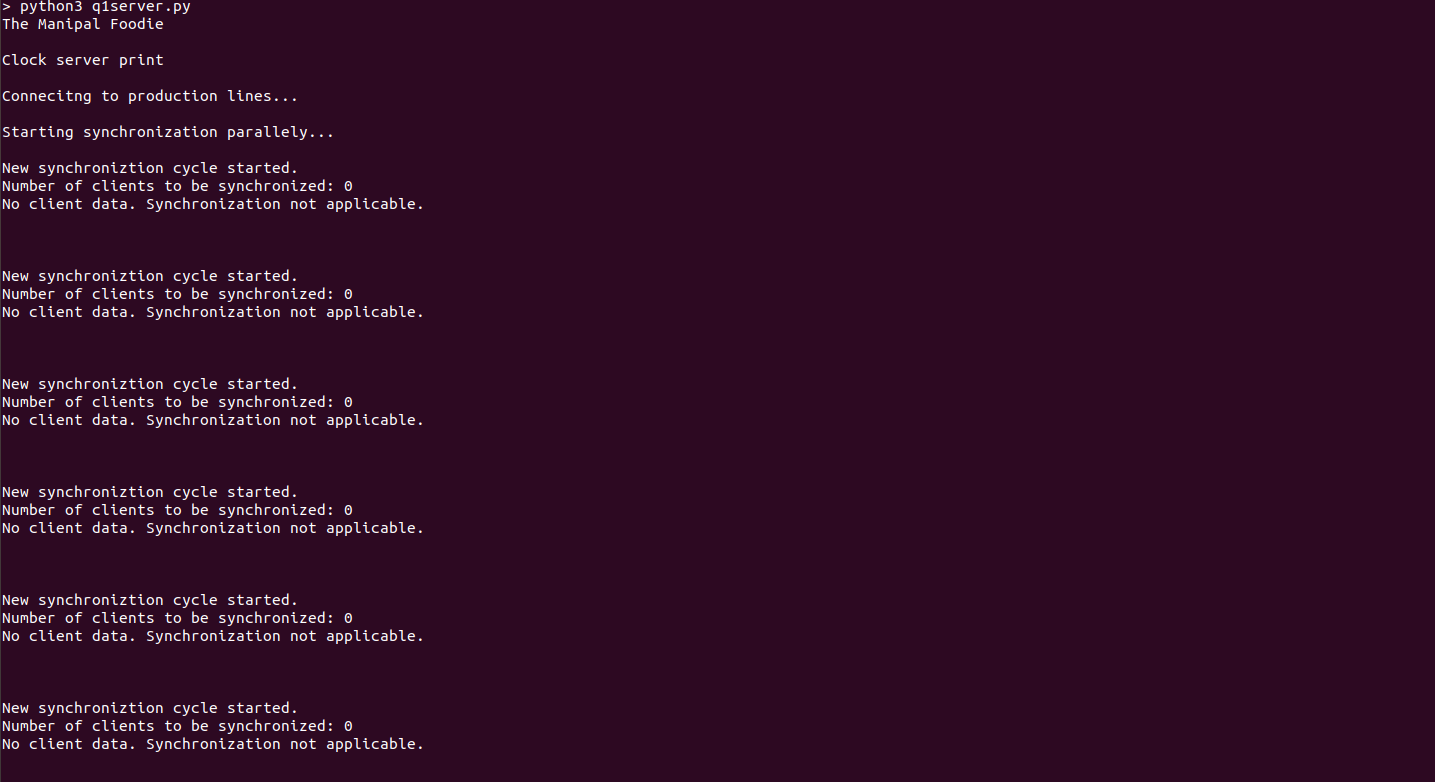
receive\_time\_thread.start()

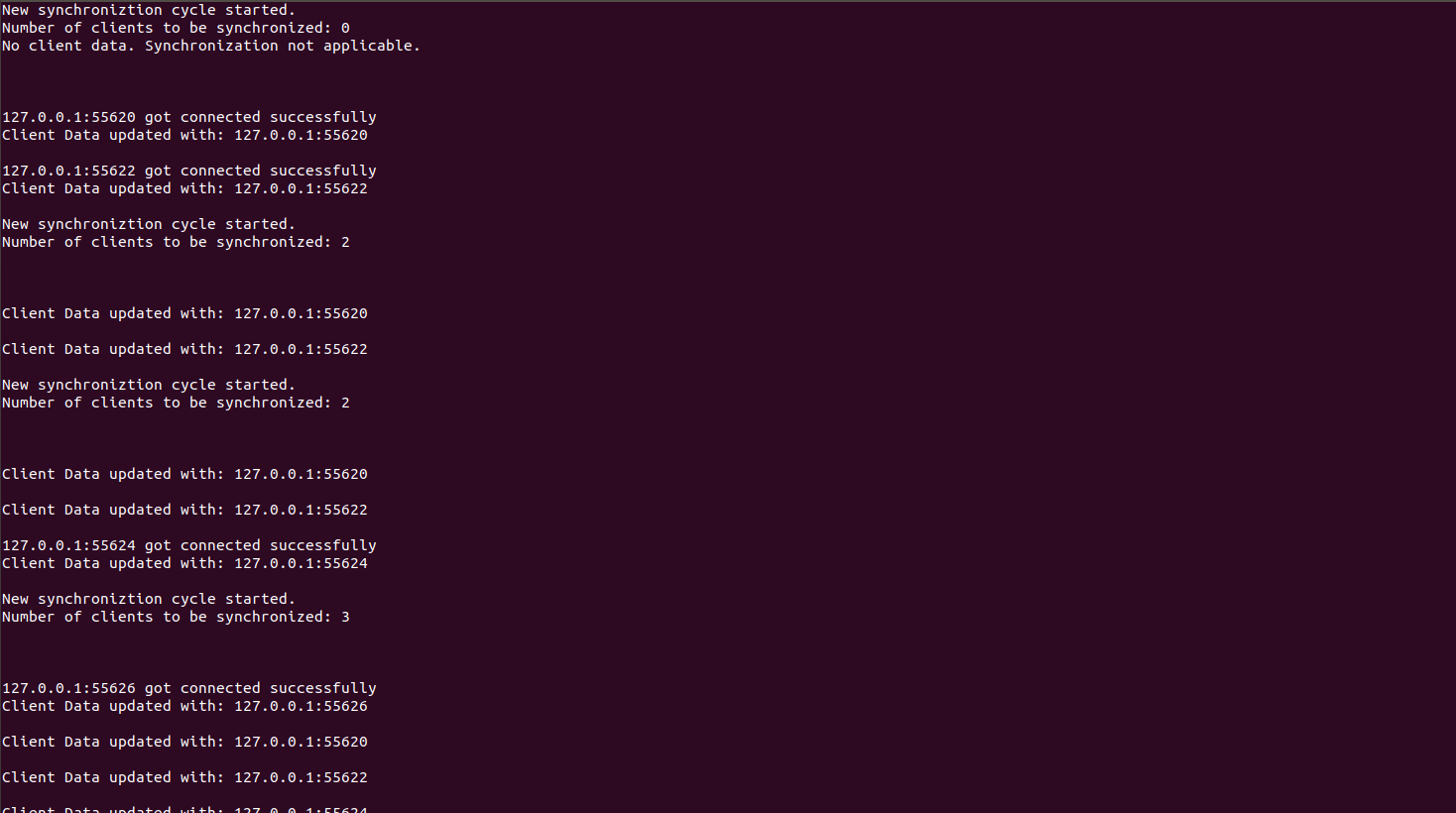
if \_\_name\_\_ == '\_\_main\_\_':

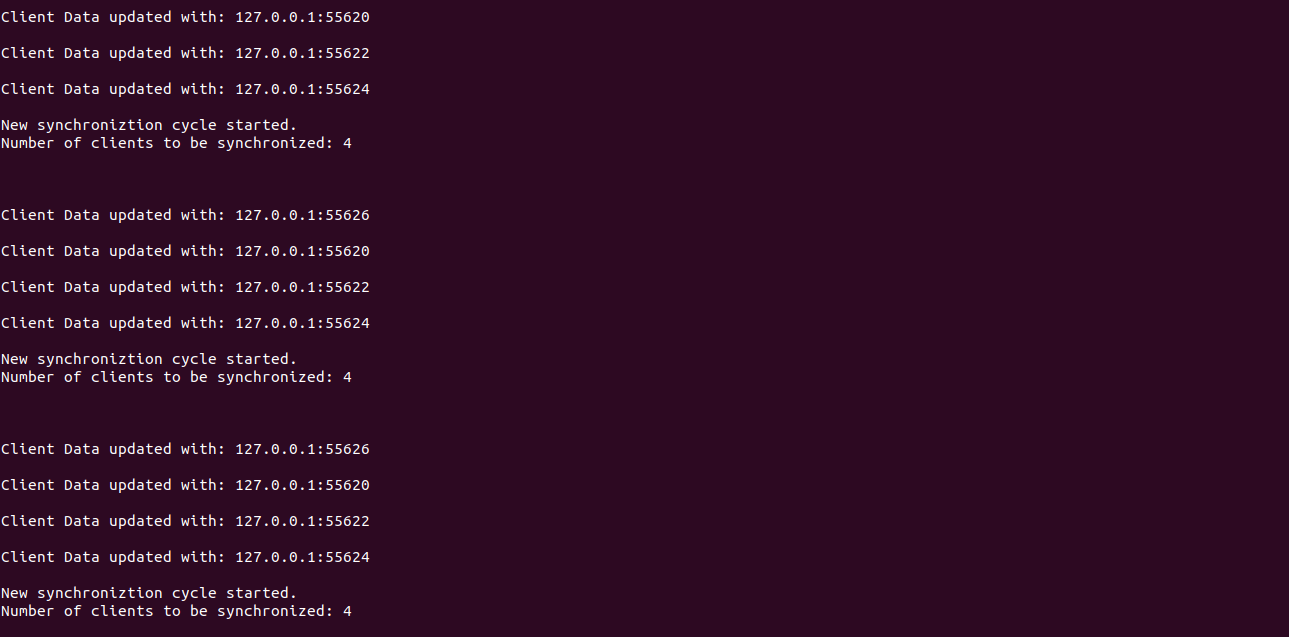
initiateSlaveClient(port=8059)

**Output:**

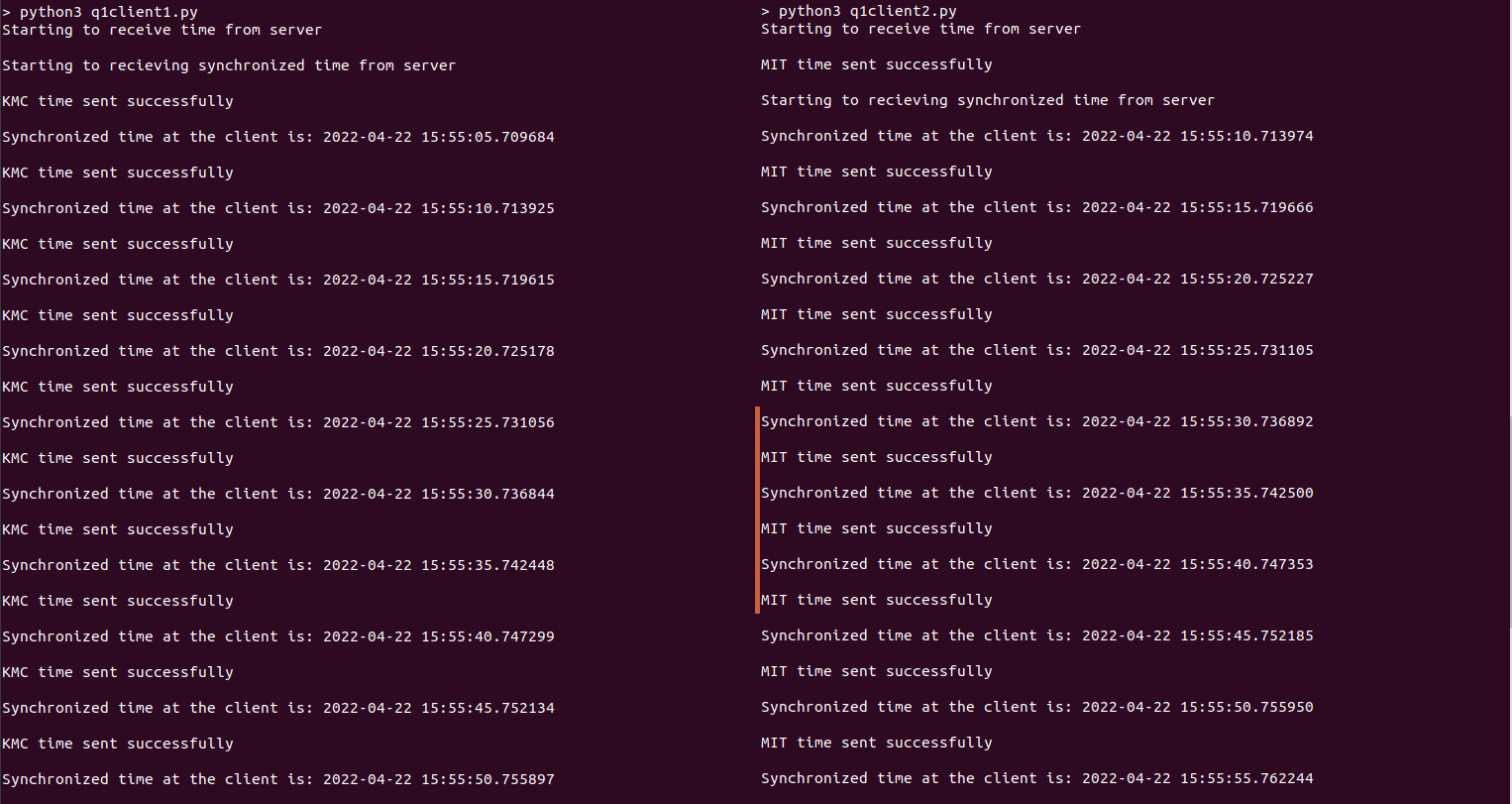
**Server:**



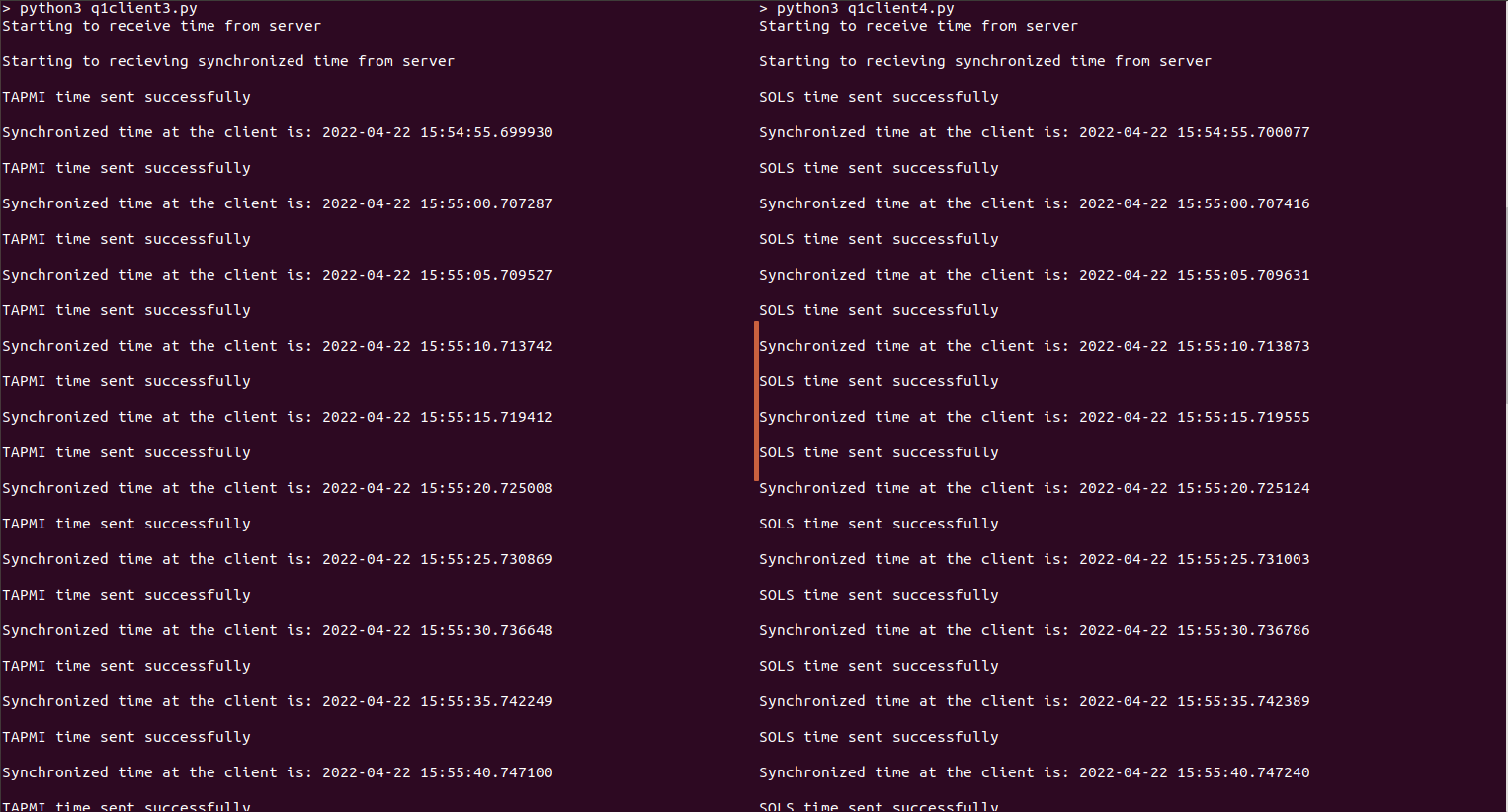




**Client 1 & Client 2**



**Client 3 & Client 4**



**2)**

**#server.py**

import socket

import datetime

import time

def initiateClockServer():

s = socket.socket()

print("Manipal Buddy Banking")

port = 8011

s.bind(('', port))

s.listen(5)

print("Waiting for client...")

while True:

connection, address = s.accept()

print('Server connected to', address)

connection.send(str(datetime.datetime.now()).encode())

connection.close()

if \_\_name\_\_ == '\_\_main\_\_':

initiateClockServer()

**#client1.py**

import socket

import datetime

import time

from dateutil import parser

from timeit import default\_timer as timer

def synchronizeTime():

print("MOBILE APP\n")

s = socket.socket()

port = 8011

s.connect(('127.0.0.1', port))

request\_time = timer()

server\_time = parser.parse(s.recv(1024).decode())

response\_time = timer()

actual\_time = datetime.datetime.now()

print("Time returned by server: " + str(server\_time))

process\_delay\_latency = response\_time - request\_time

print("Process Delay latency: " + str(process\_delay\_latency) + " seconds")

print("Actual clock time at client side: " + str(actual\_time))

client\_time = server\_time + \

datetime.timedelta(seconds=(process\_delay\_latency) / 2)

print("Synchronized process client time: " + str(client\_time))

time.sleep(10)

s.close()

if \_\_name\_\_ == '\_\_main\_\_':

synchronizeTime()

**#client2.py**

import socket

import datetime

import time

from dateutil import parser

from timeit import default\_timer as timer

def synchronizeTime():

print("WEB BROWSER\n")

s = socket.socket()

port = 8011

s.connect(('127.0.0.1', port))

request\_time = timer()

server\_time = parser.parse(s.recv(1024).decode())

response\_time = timer()

actual\_time = datetime.datetime.now()

print("Time returned by server: " + str(server\_time))

process\_delay\_latency = response\_time - request\_time

print("Process Delay latency: " + str(process\_delay\_latency) + " seconds")

print("Actual clock time at client side: " + str(actual\_time))

client\_time = server\_time + datetime.timedelta(seconds = (process\_delay\_latency) /2)

print("Synchronized process client time: " + str(client\_time))

time.sleep(10)

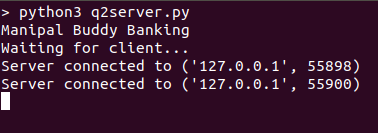
s.close()

if \_\_name\_\_ == '\_\_main\_\_':

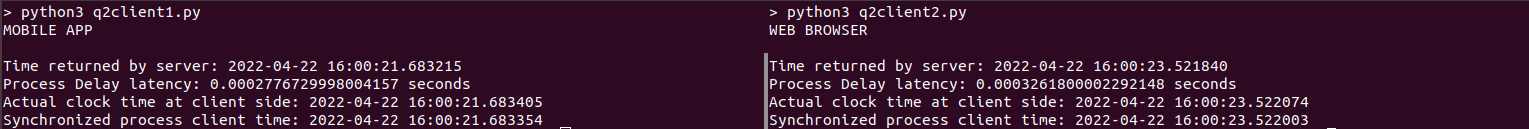
synchronizeTime()

**Output:**

**Server**

****

**Client1**



**Client2**

