#server

# Python3 program imitating a clock server

from functools import reduce

from dateutil import parser

import threading

import datetime

import socket

import time

#server

# Python3 program imitating a clock server

from functools import reduce

Name: Ghodake Omkar Atul

Roll No: 30

PRN No: 72036196G

BE IT

### Assignment 1

**CODE:**

#server

# Python3 program imitating a clock server

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 startReceivingClockTime(connector, address):

while True:

# receive 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

}

# datastructure used to store client address and clock data

client\_data = {}

''' nested thread function used to receive

clock time from a connected client '''

def startReceivingClockTime(connector, address):

while True:

# receive 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

}

# datastructure used to store client address and clock data

client\_data = {}

''' nested thread function used to receive

clock time from a connected client '''

def startReceivingClockTime(connector, address):

while True:

# receive 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 '''

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 '''

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 = startReceivingClockTime,

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

# 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

''' master sync thread function used to generate

cycles of clock synchronization in the network '''

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("\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 parallelly...\n")

sync\_thread = threading.Thread(

target = synchronizeAllClocks,

args = ())

sync\_thread.start()

# Driver function

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

# Trigger the Clock Server

initiateClockServer(port = 8080)

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 parallelly...\n")

sync\_thread = threading.Thread(

target = synchronizeAllClocks,

args = ())

sync\_thread.start()

# Driver function

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

# Trigger the Clock Server

initiateClockServer(port = 8080)

#Client

# Python3 program imitating a client process

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 receiving synchronized from server

print("Starting to receiving " + \

"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)

# Driver function

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

# initialize the Slave / Client

initiateSlaveClient(port = 8080)

**OUTPUT:**

