WEEK6&7 DS_LAB : CLOCK SYNCHRONIZATION & MUTUAL EXCLUSION (ELECTION ALGORITHM)

Name: Sagnik Chatterjee
Reg: 180905478
Roll No: 61
Sem: 6
Section :B

To imitate the feel of distributed systems the Berkeley and Cristian's algorithms were performed using the python threads which were mirroring to be independent systems.

Also the different process in case of q3 and q4 for get the sync. Done was done using opening multiple terminals and making multiple clients

Eg : one for MIT, KMC , SOLS ,TAPMI each etc.

Similarly made 2 clients for MOBILE and LAPTOP in case of q4 .

For Bully and Ring Algorithms the processes were treated as classes and various number of objects were declared which acted as independent processes in distributed systems and were given appropriate fields for relevant data to be stored.

Q1. Cristian Algorithm Implementation

Server.py

```
import socket
import datetime
import time


def initiateClockServer():
   s = socket.socket()
   print(f" :|> Socket successfully created")

   port = 8011
   s.bind(('', port))

   s.listen(5)
   print(f" :|> Socket is listening.")

   while True:
       connection, address = s.accept()
       print(f':|> Server connected to {address}')
       connection.send(str(datetime.datetime.now()).encode())

if __name__ == '__main__':

   initiateClockServer()
```

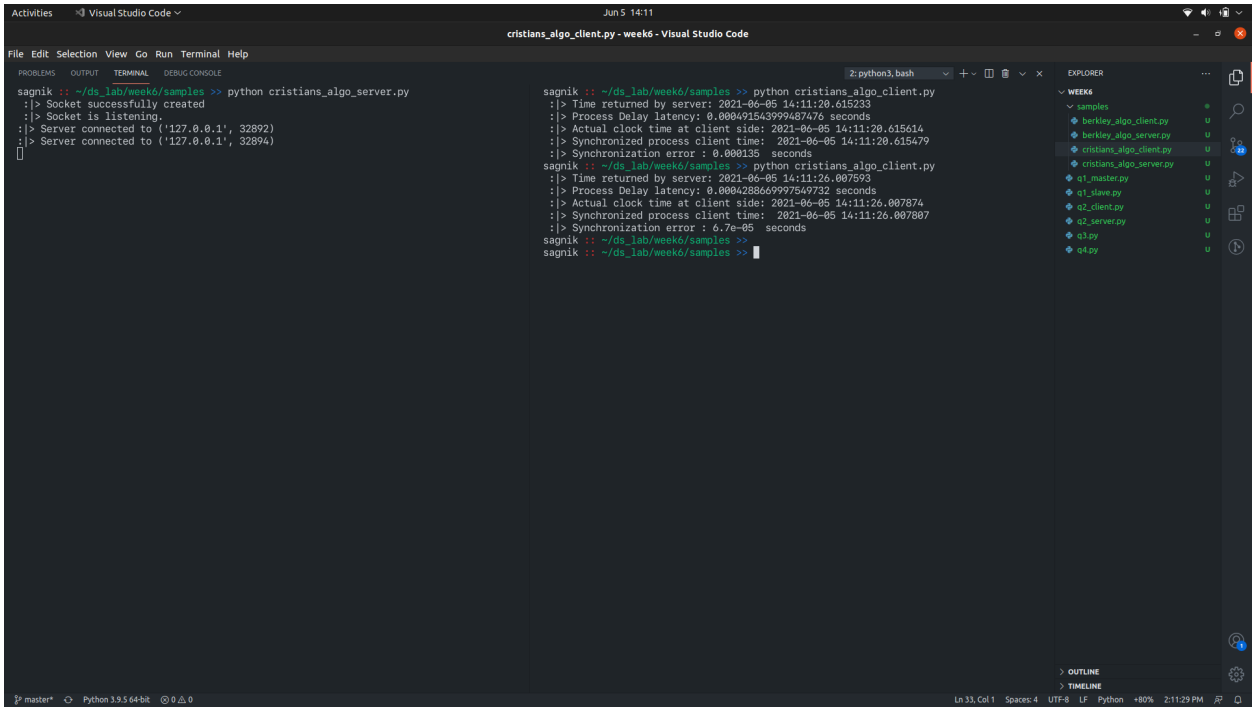
Client.py

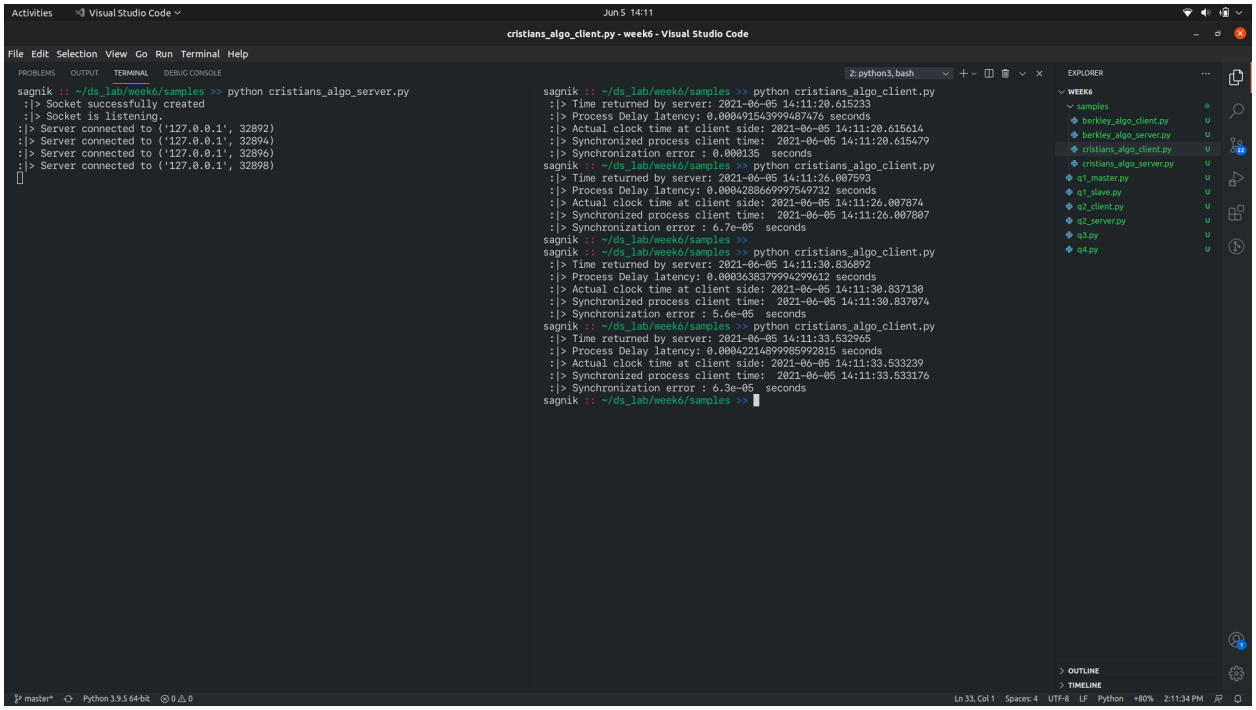
```
import socket
import datetime
```

```
from dateutil import parser
from timeit import default_timer as timer


def synchronizeTime():

   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(f" :|> Time returned by server: {server_time}")

   process_delay_latency = response_time - request_time

   print(f" :|> Process Delay latency: {process_delay_latency}
seconds")

   print(f" :|> Actual clock time at client side:
{actual_time}")

   client_time = server_time +
datetime.timedelta(seconds=(process_delay_latency) / 2)

   print(f" :|> Synchronized process client time:
{client_time}")
```

```
    error = actual_time - client_time
    print(f" :|> Synchronization error : {error.total_seconds()}
seconds")

    s.close()


if __name__ == "__main__":

    synchronizeTime()
```

Output Screenshots:





2.Berkeley Algorithm Implementation

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(f" :|> Client Data updated with: {address}")
        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(f" :|> {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(f" :|> New synchroniztion cycle started.")
        print(f" :|> Number of clients to be synchronized:
{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(
                        f" :|> Something went wrong while sending
synchronized through {client_addr} ,{e}"
                    )

        else:
            print(f" :|> No client data. synchronization not
applicable.")

        time.sleep(5)


def initiateClockServer(port=8080):

    master_server = socket.socket()
    master_server.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)

    print(f" :|> Socket at master node created successfully")

    master_server.bind(("", port))

    master_server.listen(10)
    print(f" :|> Clock server started.")

    print(f" :|> Starting to make connections.")
    master_thread = threading.Thread(target=startConnecting,
args=(master_server,))
    master_thread.start()
```

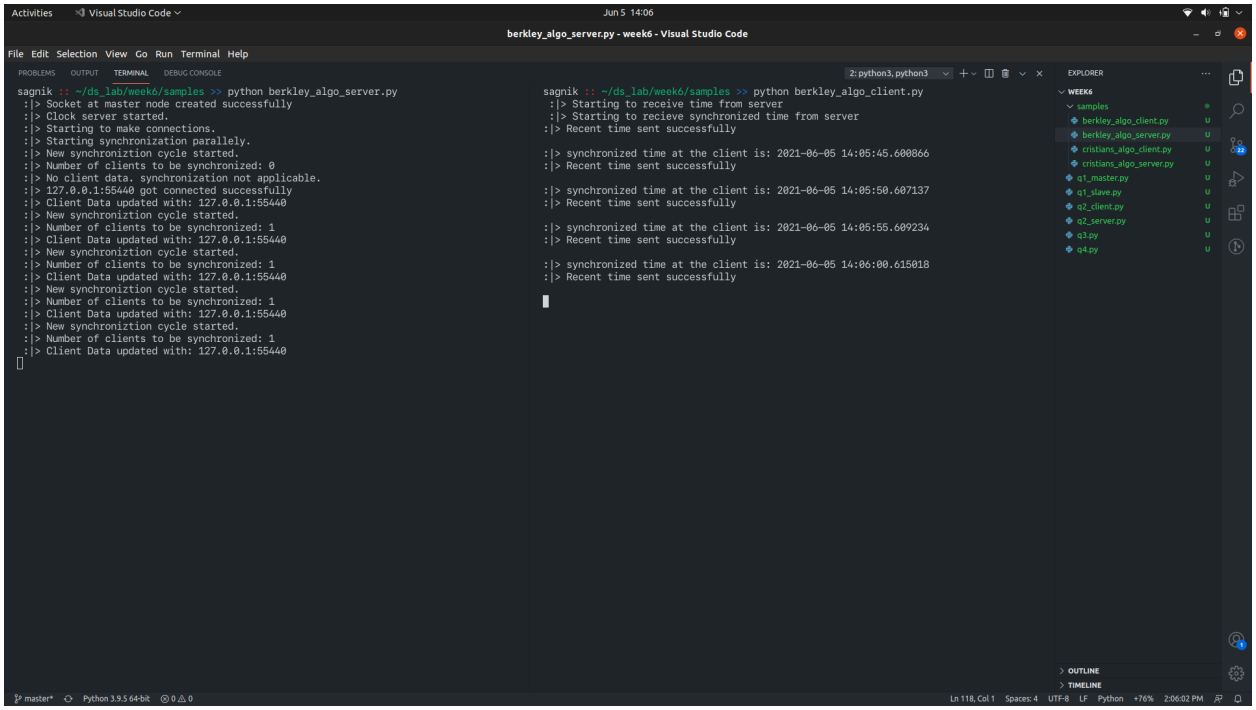
```
    print(f" :|> Starting synchronization parallely.")
    sync_thread = threading.Thread(target=synchronizeAllClocks,
args=())
    sync_thread.start()


if __name__ == "__main__":

    initiateClockServer(port=8080)
```

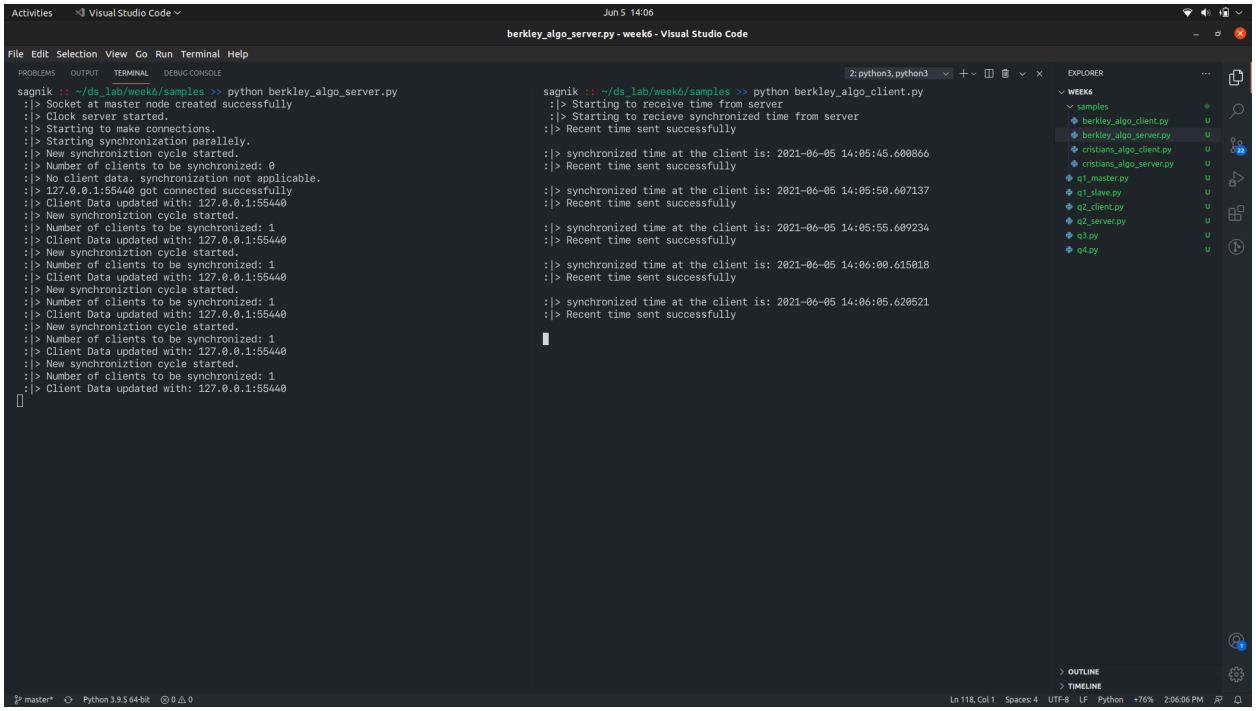
Client.py

```
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(":|> Recent 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(f":|> synchronized time at the client is:
{synchronized_time}")


def initiateSlaveClient(port=8080):

    slave_client = socket.socket()

    slave_client.connect(("127.0.0.1", port))

    print(f" :|> Starting to receive time from server")
    send_time_thread = threading.Thread(target=startSendingTime,
args=(slave_client,))
    send_time_thread.start()

    print(f" :|> Starting to recieve synchronized time from
server")
    receive_time_thread = threading.Thread(
        target=startReceivingTime, args=(slave_client,)
    )
    receive_time_thread.start()


if __name__ == "__main__":

    initiateSlaveClient(port=8080)
```

Output Screenshots:





Q3. The Manipal Foodie is a renowned automated food processing outlet known for its tiffin

service to students. The various processes involved are food production, filling and packing. Every day more than 3000 orders are received on an average from the students in manipal. There are total of 4 production lines for orders received from KMC, MIT, TAPMI and SOLS students, each of them has a digital clock which needs to be in synchronization with the master clock. The master clock mounted in the testing lab controls the entire clock system. Design an appropriate solution using Berkeley's algorithm for the above scenario. Assume that the clocks at the institutes are slave/clients.

Master.py

```
from functools import reduce
from dateutil import parser
import threading
import datetime
import socket
import time


client_data = {}


def startRecieveingClockTime(connector, address):
    """nested thread function used to receive
    clock time from a connected client
    """

    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(f" :|> Client Data updated with: {address} ")
        time.sleep(5)


""" master thread function used to open portal for
   accepting clients over given port """


def startConnecting(master_server):

   while True:

        master_slave_connector, addr = master_server.accept()
        slave_address = str(addr[0]) + ":" + str(addr[1])

        print(f" :|> {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(f" :|> Number of clients to be synchronized:
{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(
                        f"Something went wrong while sending
synchronized time through  {client_addr}"
                    )

        else:
            print(" :|> No client data. Synchronization not
applicable.")

        time.sleep(5)


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))

    master_server.listen(10)
    print(" :|> Clock server started.\n")

    print(" :|> Starting to make connection.\n")
```

```
    master_thread = threading.Thread(target=startConnecting,
args=(master_server,))
    master_thread.start()

    print(" :|> Starting synch parallely.\n")
    sync_thread = threading.Thread(target=synchronizeAllClocks,
args=())
    sync_thread.start()


if __name__ == "__main__":

    initiateClockServer(port=8080)
```

client_MIT.py

```
from timeit import default_timer as timer
from dateutil import parser
import threading
import datetime
import socket
import time


def startSendingTime(name, slave_client):

    while True:

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

        print(f" :|> Recent time sent successfully by {name}",
end="\n\n")
        time.sleep(5)
```

```
def startReceivingTime(name, slave_client):

    while True:

        Synchronized_time =
parser.parse(slave_client.recv(1024).decode())

        print(
            f" :|> Synchronized time at the client is {name} :
{str(Synchronized_time)}",
            end="\n",
        )


def initiateSlaveClient(name, port=8080):

    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=(name, slave_client)
    )
    send_time_thread.start()

    print(f" :|> Starting to recieving synchronized time from
server\n")
    receive_time_thread = threading.Thread(
        target=startReceivingTime, args=(name, slave_client)
    )
    receive_time_thread.start()
```

```
if __name__ == "__main__":

    client = "MIT"
    initiateSlaveClient(client, port=8080)
```

client_TAPMI.py

```
from timeit import default_timer as timer
from dateutil import parser
import threading
import datetime
import socket
import time


def startSendingTime(name, slave_client):

    while True:

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

        print(f" :|> Recent time sent successfully by {name}",
end="\n\n")
        time.sleep(5)


def startReceivingTime(name, slave_client):

    while True:

        Synchronized_time =
parser.parse(slave_client.recv(1024).decode())
```

```
        print(
            f" :|> Synchronized time at the client is {name} :
{str(Synchronized_time)}",
            end="\n",
        )


def initiateSlaveClient(name, port=8080):

    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=(name, slave_client)
    )
    send_time_thread.start()

    print(f" :|> Starting to recieving synchronized time from
server\n")
    receive_time_thread = threading.Thread(
        target=startReceivingTime, args=(name, slave_client)
    )
    receive_time_thread.start()


if __name__ == "__main__":

    client = "TAPMI"
    initiateSlaveClient(client, port=8080)
```

client_KMC.py

```
from timeit import default_timer as timer
from dateutil import parser
import threading
import datetime
import socket
import time


def startSendingTime(name, slave_client):

   while True:

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

       print(f" :|> Recent time sent successfully by {name}",
end="\n\n")
       time.sleep(5)


def startReceivingTime(name, slave_client):

   while True:

       Synchronized_time =
parser.parse(slave_client.recv(1024).decode())

       print(
           f" :|> Synchronized time at the client is {name} :
{str(Synchronized_time)}",
           end="\n",
       )


def initiateSlaveClient(name, port=8080):
```

```
    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=(name, slave_client)
    )
    send_time_thread.start()

    print(f" :|> Starting to recieving synchronized time from
server\n")
    receive_time_thread = threading.Thread(
        target=startReceivingTime, args=(name, slave_client)
    )
    receive_time_thread.start()


if __name__ == "__main__":

    client = "KMC"
    initiateSlaveClient(client, port=8080)
```

client_SOLS.py

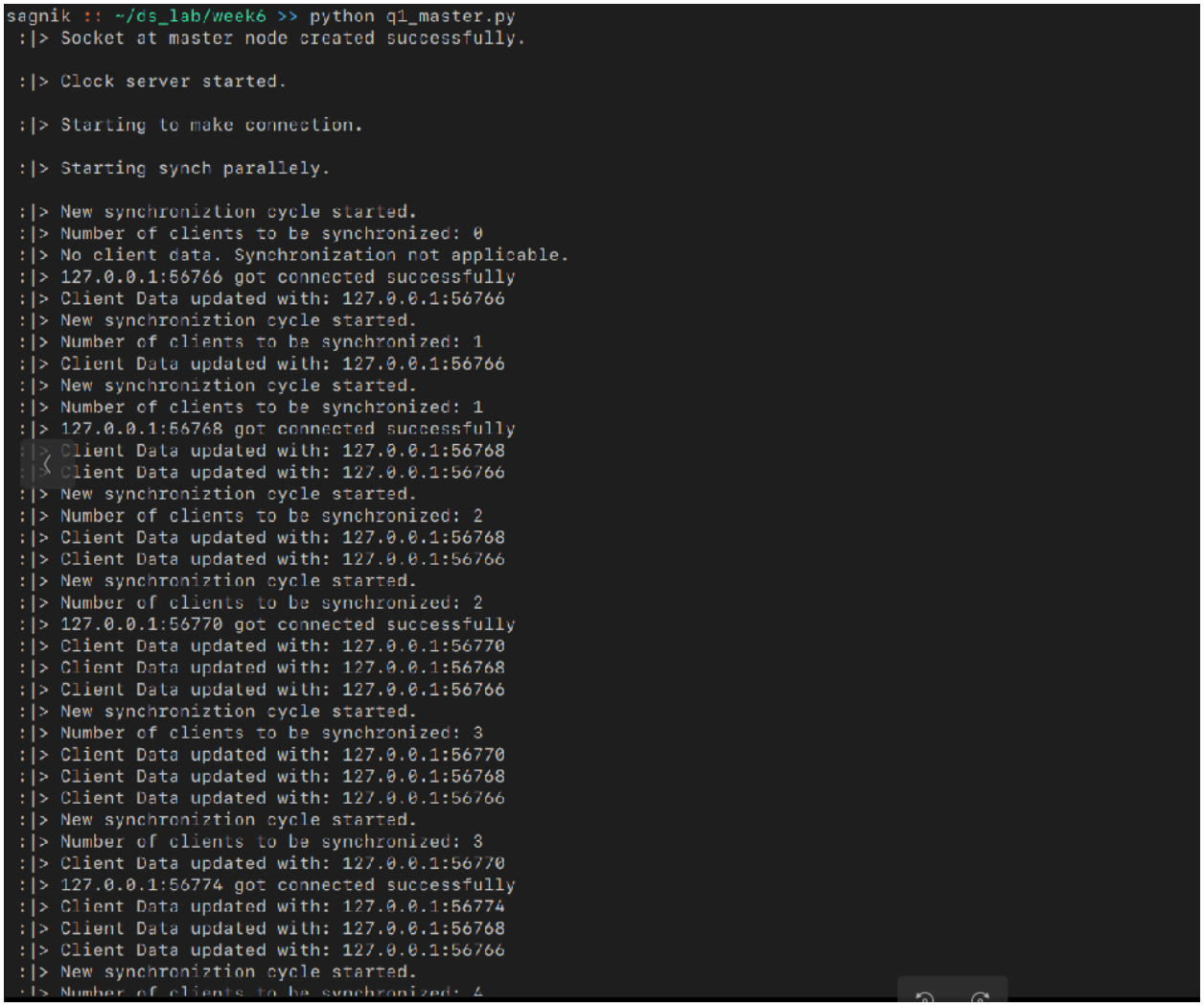
```
from timeit import default_timer as timer
from dateutil import parser
import threading
import datetime
import socket
import time
```

```
def startSendingTime(name, slave_client):

    while True:

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

        print(f" :|> Recent time sent successfully by {name}",
end="\n\n")
        time.sleep(5)


def startReceivingTime(name, slave_client):

    while True:

        Synchronized_time =
parser.parse(slave_client.recv(1024).decode())

        print(
            f" :|> Synchronized time at the client is {name} :
{str(Synchronized_time)}",
            end="\n",
        )


def initiateSlaveClient(name, port=8080):

    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=(name, slave_client)
    )
    send_time_thread.start()

    print(f" :|> Starting to recieving synchronized time from
server\n")
    receive_time_thread = threading.Thread(
        target=startReceivingTime, args=(name, slave_client)
    )
    receive_time_thread.start()


if __name__ == "__main__":

    client = "SOLS"
    initiateSlaveClient(client, port=8080)
```

Output:

q1_master.py


sagnik :: ~/ds_lab/week6 >> python q1_master.py
:|> Socket at master node created successfully.
:|> Clock server started.
:|> Starting to make connection.
:|> Starting synch parallely.
:|> New synchroniztion cycle started.
:|> Number of clients to be synchronized: 0
:|> No client data. Synchronization not applicable.
:|> 127.0.0.1:56766 got connected successfully
:|> Client Data updated with: 127.0.0.1:56766
:|> New synchroniztion cycle started.
:|> Number of clients to be synchronized: 1
:|> Client Data updated with: 127.0.0.1:56766
:|> New synchroniztion cycle started.
:|> Number of clients to be synchronized: 1
:|> 127.0.0.1:56768 got connected successfully
:|> Client Data updated with: 127.0.0.1:56768
:|> Client Data updated with: 127.0.0.1:56766
:|> New synchroniztion cycle started.
:|> Number of clients to be synchronized: 2
:|> Client Data updated with: 127.0.0.1:56768
:|> Client Data updated with: 127.0.0.1:56766
:|> New synchroniztion cycle started.
:|> Number of clients to be synchronized: 2
:|> 127.0.0.1:56770 got connected successfully
:|> Client Data updated with: 127.0.0.1:56770
:|> Client Data updated with: 127.0.0.1:56768
:|> Client Data updated with: 127.0.0.1:56766
:|> New synchroniztion cycle started.
:|> Number of clients to be synchronized: 3
:|> Client Data updated with: 127.0.0.1:56770
:|> Client Data updated with: 127.0.0.1:56768
:|> Client Data updated with: 127.0.0.1:56766
:|> New synchroniztion cycle started.
:|> Number of clients to be synchronized: 3
:|> Client Data updated with: 127.0.0.1:56770
:|> 127.0.0.1:56774 got connected successfully
:|> Client Data updated with: 127.0.0.1:56774
:|> Client Data updated with: 127.0.0.1:56768
:|> Client Data updated with: 127.0.0.1:56766
:|> New synchroniztion cycle started.

```
:|> Number of clients to be synchronized: 1
:|> Client Data updated with: 127.0.0.1:56766
:|> New synchroniztion cycle started.
:|> Number of clients to be synchronized: 1
:|> 127.0.0.1:56768 got connected successfully
:|> Client Data updated with: 127.0.0.1:56768
:|> Client Data updated with: 127.0.0.1:56766
:|> New synchroniztion cycle started.
:|> Number of clients to be synchronized: 2
:|> Client Data updated with: 127.0.0.1:56768
:|> Client Data updated with: 127.0.0.1:56766
:|> New synchroniztion cycle started.
:|> Number of clients to be synchronized: 2
:|> 127.0.0.1:56770 got connected successfully
:|> Client Data updated with: 127.0.0.1:56770
:|> Client Data updated with: 127.0.0.1:56768
:|> Client Data updated with: 127.0.0.1:56766
:|> New synchroniztion cycle started.
:|> Number of clients to be synchronized: 3
:|> Client Data updated with: 127.0.0.1:56770
:|> Client Data updated with: 127.0.0.1:56768
:|> Client Data updated with: 127.0.0.1:56766
:|> New synchroniztion cycle started.
:|> Number of clients to be synchronized: 3
:|> Client Data updated with: 127.0.0.1:56770
:|> 127.0.0.1:56774 got connected successfully
:|> Client Data updated with: 127.0.0.1:56774
:|> Client Data updated with: 127.0.0.1:56768
:|> Client Data updated with: 127.0.0.1:56766
:|> New synchroniztion cycle started.
:|> Number of clients to be synchronized: 4
:|> Client Data updated with: 127.0.0.1:56770
:|> Client Data updated with: 127.0.0.1:56774
:|> Client Data updated with: 127.0.0.1:56768
:|> Client Data updated with: 127.0.0.1:56766
:|> New synchroniztion cycle started.
:|> Number of clients to be synchronized: 4
:|> Client Data updated with: 127.0.0.1:56770
:|> Client Data updated with: 127.0.0.1:56774
:|> Client Data updated with: 127.0.0.1:56768
:|> Client Data updated with: 127.0.0.1:56766
:|> New synchroniztion cycle started.
:|> Number of clients to be synchronized: 4
:|> Client Data updated with: 127.0.0.1:56770
:|> Client Data updated with: 127.0.0.1:56774
:|> Client Data updated with: 127.0.0.1:56768
```

q1clientMIT.py

```
sagnik :: ~/ds_lab/week6 >> python q1clientMIT.py
Starting to receive time from server

 :|> Starting to recieving synchronized time from server

 :|> Recent time sent successfully by MIT

 :|> Synchronized time at the client is MIT : 2021-06-05 17:17:23.907094
 :|> Recent time sent successfully by MIT

 :|> Synchronized time at the client is MIT : 2021-06-05 17:17:28.913964
 :|> Recent time sent successfully by MIT

 :|> Synchronized time at the client is MIT : 2021-06-05 17:17:33.916815
 :|> Recent time sent successfully by MIT

 :|> Synchronized time at the client is MIT : 2021-06-05 17:17:38.923506
 :|> Recent time sent successfully by MIT

 :|> Synchronized time at the client is MIT : 2021-06-05 17:17:43.928421
 :|> Recent time sent successfully by MIT

 :|> Synchronized time at the client is MIT : 2021-06-05 17:17:48.934362
 :|> Recent time sent successfully by MIT

 :|> Synchronized time at the client is MIT : 2021-06-05 17:17:53.940581
 :|> Recent time sent successfully by MIT

 :|> Synchronized time at the client is MIT : 2021-06-05 17:17:58.945715
 :|> Recent time sent successfully by MIT

 :|> Synchronized time at the client is MIT : 2021-06-05 17:18:03.951553
 :|> Recent time sent successfully by MIT

 :|> Synchronized time at the client is MIT : 2021-06-05 17:18:08.957886
```

q1clientKMC.py

```
sagnik :: ~/ds_lab/week6 >> python q1clientKMC.py
Starting to receive time from server

 :|> Starting to recieving synchronized time from server

 :|> Recent time sent successfully by KMC

 :|> Synchronized time at the client is KMC : 2021-06-05 17:17:53.940797
 :|> Recent time sent successfully by KMC

 :|> Synchronized time at the client is KMC : 2021-06-05 17:17:58.945973
 :|> Recent time sent successfully by KMC

 :|> Synchronized time at the client is KMC : 2021-06-05 17:18:03.951782
 :|> Recent time sent successfully by KMC

 :|> Synchronized time at the client is KMC : 2021-06-05 17:18:08.958105
```

q1clientSOLS.py

```
sagnik :: ~/ds_lab/week6 >> python q1clientSOLS.py
Starting to receive time from server

 :|> Starting to recieving synchronized time from server

 :|> Recent time sent successfully by SOLS

 :|> Synchronized time at the client is SOLS : 2021-06-05 17:17:43.928606
 :|> Recent time sent successfully by SOLS

 :|> Synchronized time at the client is SOLS : 2021-06-05 17:17:48.934534
 :|> Recent time sent successfully by SOLS

 :|> Synchronized time at the client is SOLS : 2021-06-05 17:17:53.940758
 :|> Recent time sent successfully by SOLS

 :|> Synchronized time at the client is SOLS : 2021-06-05 17:17:58.945933
 :|> Recent time sent successfully by SOLS

 :|> Synchronized time at the client is SOLS : 2021-06-05 17:18:03.951740
 :|> Recent time sent successfully by SOLS

 :|> Synchronized time at the client is SOLS : 2021-06-05 17:18:08.958065
```

q1cleintTAPMI.py:-

```
sagnik :: ~/ds_lab/week6 >> python q1clientTAPMI.py
Starting to receive time from server

 :|> Starting to recieving synchronized time from server

 :|> Recent time sent successfully by TAPMI

 :|> Synchronized time at the client is TAPMI : 2021-06-05 17:17:33.916932
 :|> Recent time sent successfully by TAPMI

 :|> Synchronized time at the client is TAPMI : 2021-06-05 17:17:38.923639
 :|> Recent time sent successfully by TAPMI

 :|> Synchronized time at the client is TAPMI : 2021-06-05 17:17:43.928556
 :|> Recent time sent successfully by TAPMI

 :|> Synchronized time at the client is TAPMI : 2021-06-05 17:17:48.934489
 :|> Recent time sent successfully by TAPMI

 :|> Synchronized time at the client is TAPMI : 2021-06-05 17:17:53.940709
 :|> Recent time sent successfully by TAPMI

 :|> Synchronized time at the client is TAPMI : 2021-06-05 17:17:58.945871
 :|> Recent time sent successfully by TAPMI

 :|> Synchronized time at the client is TAPMI : 2021-06-05 17:18:03.951689
 :|> Recent time sent successfully by TAPMI

 :|> Synchronized time at the client is TAPMI : 2021-06-05 17:18:08.958013
```

Q4.Manipal Buddy is a banking and education application for the students and staff of MIT,
Manipal. Mr Vinay, a sixth semester student wants to pay the end semester exams fees for
a re-registered course. He simultaneously wishes to register for a course on NPTEL
through the app. To register for exam he uses the mobile app whereas to register for
NPTEL course he uses his laptop to log in. As he needs to finish both the registrations on
the same day, he tries to do both the tasks simultaneously. Analyse and demonstrate using
a program how Cristian's algorithm can be used in the above case to synchronize the
clocks. Assume the relevant parameters.

Server.py

```
import socket
import datetime
import time


def initiateClockServer():
    s = socket.socket()
    print(" :|> Socket successfully created")

    port = 8011
    s.bind(("", port))

    s.listen(5)
    print(":|> Socket is listening")

    while True:
        connection, address = s.accept()
        print(f" :|> Server connected to {address}")
```

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


if __name__ == "__main__":

    initiateClockServer()
```

clientLAPTOP.py

```
import socket
import datetime
from dateutil import parser
from timeit import default_timer as timer


def synchronizeTime(device_type):

    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(f" :|> Synchronising now :- {device_type}")
    print(f" :|> Time returned by server: {server_time} ")

    process_delay_latency = response_time - request_time
```

```
   print(f" :|> Process Delay latency: {process_delay_latency}
seconds")

   print(f" :|> Actual clock time at client side:
{actual_time}")

   client_time = server_time +
datetime.timedelta(seconds=(process_delay_latency) / 2)

   print(f" :|> Synchronized process client time:
{client_time}")

   error = actual_time - client_time
   print(f" :|> Synchronization error : {error.total_seconds()}
seconds.")
   s.close()


if __name__ == "__main__":

   synchronizeTime("LAPTOP")
```

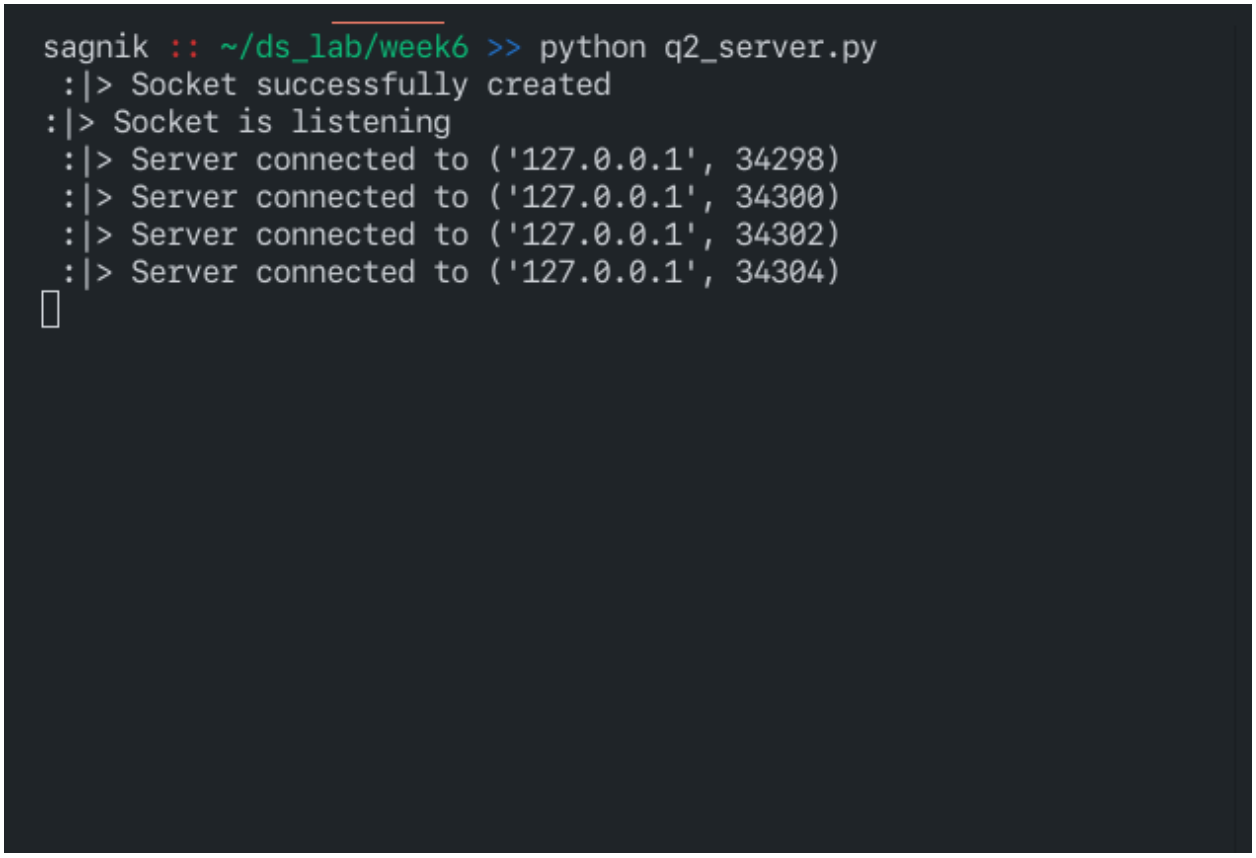
clientMOBILE.py

```
import socket
import datetime
from dateutil import parser
from timeit import default_timer as timer


def synchronizeTime(device_type):

   s = socket.socket()

   port = 8011
```

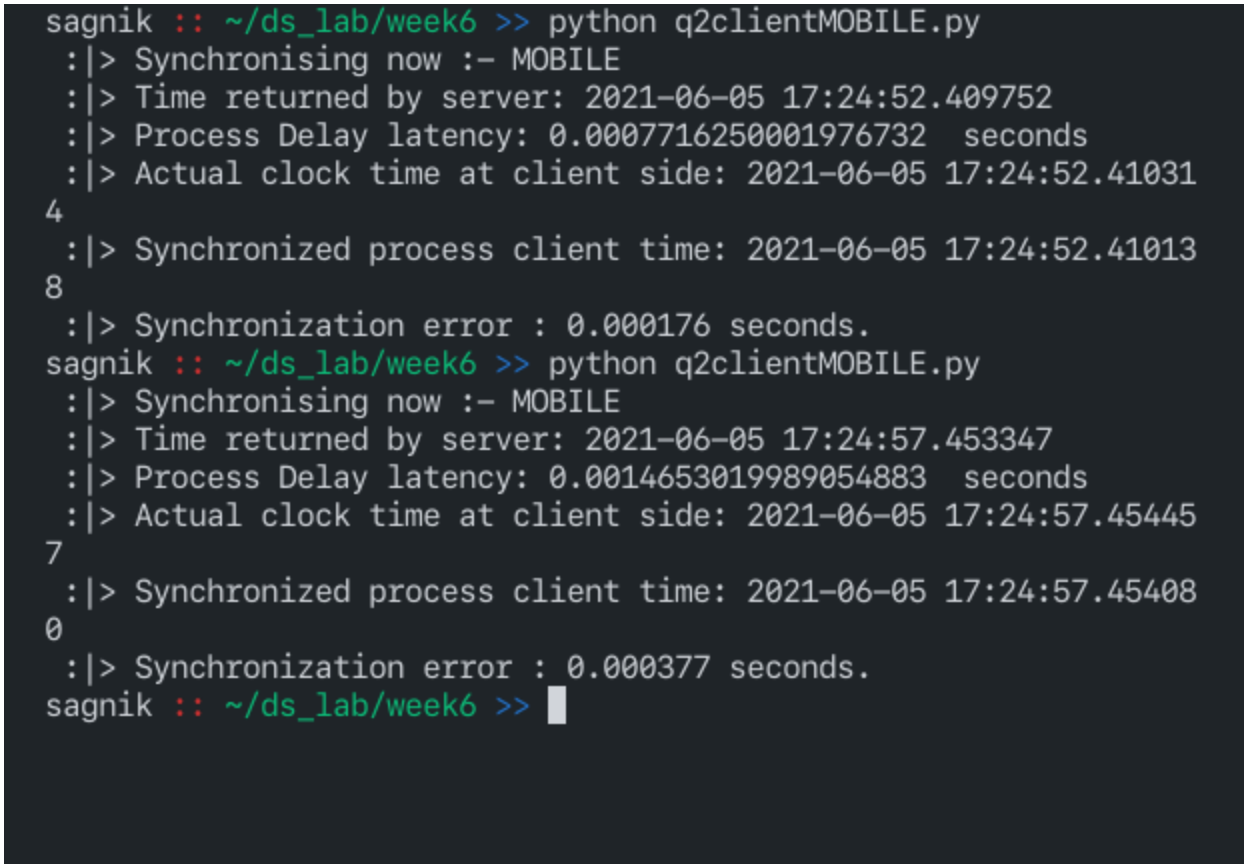
```
    s.connect(("127.0.0.1", port))

    request_time = timer()

    server_time = parser.parse(s.recv(1024).decode())
    response_time = timer()
    actual_time = datetime.datetime.now()
    print(f" :|> Synchronising now :- {device_type}")
    print(f" :|> Time returned by server: {server_time} ")

    process_delay_latency = response_time - request_time

    print(f" :|> Process Delay latency: {process_delay_latency}
seconds")

    print(f" :|> Actual clock time at client side:
{actual_time}")

    client_time = server_time +
datetime.timedelta(seconds=(process_delay_latency) / 2)

    print(f" :|> Synchronized process client time:
{client_time}")

    error = actual_time - client_time
    print(f" :|> Synchronization error : {error.total_seconds()}
seconds.")
    s.close()


if __name__ == "__main__":

    synchronizeTime("MOBILE")
```

Output:

Server.py



clientMOBILE.py



clientLAPTOP.py

```
sagnik :: ~/ds_lab/week6 >> python q2clientLAPTOP.py
 :|> Synchronising now :- LAPTOP
 :|> Time returned by server: 2021-06-05 17:24:51.218912
 :|> Process Delay latency: 0.0013924949998909142  seconds
 :|> Actual clock time at client side: 2021-06-05 17:24:51.22005
7
 :|> Synchronized process client time: 2021-06-05 17:24:51.21960
8
 :|> Synchronization error : 0.000449 seconds.
sagnik :: ~/ds_lab/week6 >>
sagnik :: ~/ds_lab/week6 >> python q2clientLAPTOP.py
 :|> Synchronising now :- LAPTOP
 :|> Time returned by server: 2021-06-05 17:24:55.857081
 :|> Process Delay latency: 0.0011848180001834407  seconds
 :|> Actual clock time at client side: 2021-06-05 17:24:55.85797
3
 :|> Synchronized process client time: 2021-06-05 17:24:55.85767
3
 :|> Synchronization error : 0.0003 seconds.
sagnik :: ~/ds_lab/week6 >>
```

Q5 Simulate a scenario in distributed systems to implement the Bully Algorithm for choosing
a coordinator node amongst the participative nodes of the system after the collapse of the
existing coordinator node in the system

bullyAlgorithm.py

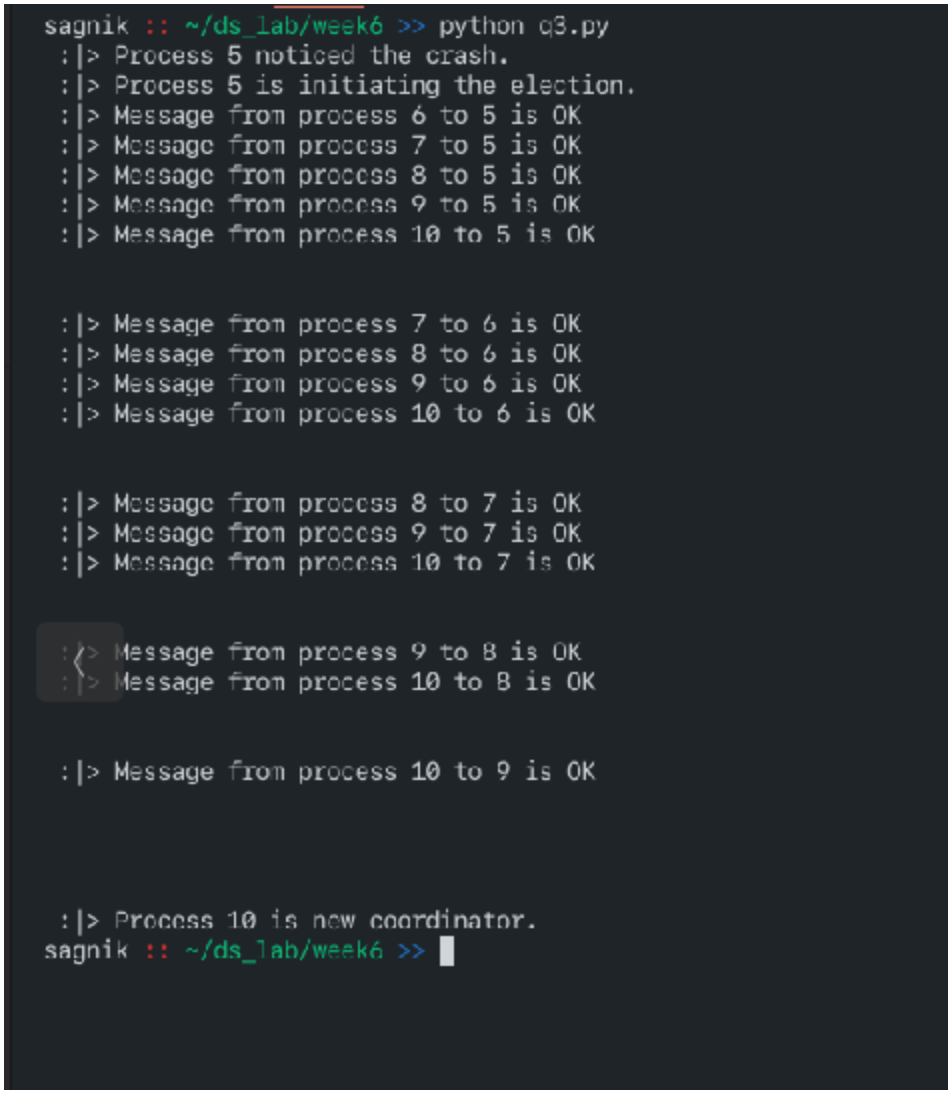
```
class Process:
    def __init__(self, id, alive):
        self.id = id
        self.alive = alive
```

```
        self.cordinator = False
        self.crashNoticer = False

    def knowAllProcesses(self):
        allProcesses = System().getAllProcessess()
        return allProcesses

    def getMessage(self, reqId):
        if reqId < self.id and self.alive:
            return "OK"
        else:
            return "NOT OK"


class System:
    allProcesses = []

    def createProcess(self, id, alive):
        process = Process(id, alive)
        self.allProcesses.append(process)

    def getAllProcessess(self):
        return self.allProcesses

    def processessCount(self):
        return len(self.allProcesses)

    def getHigherIds(self, id):
        processes = []
        for process in self.allProcesses:
            if process.id > id:
                processes.append(process.id)
        return processes
```

```
    def getProcess(self, id):
        for process in self.allProcesses:
            if process.id == id:
                return process


s = System()
for i in range(1, 11):
    s.createProcess(i, True)


p = s.getProcess(5)
p.crashNoticer = True

processes = s.getAllProcessess()
global initiator

for process in processes:
    if process.alive == False:
        print(f" :|> Process {process.id} is crashed.")
    if process.crashNoticer:
        initiator = process
        print(f" :|> Process {process.id} noticed the crash.")

print(f" :|> Process {initiator.id} is initiating the
election.")


def conductElection(id):
    nextAvailable = []
    for i in s.getHigherIds(id):
        if s.getProcess(id).alive:
            message = s.getProcess(i).getMessage(id)
```

```
            print(f" :|> Message from process {i} to {id} is
{message}")
            if message == "OK":
                nextAvailable.append(i)
            if (
                len(s.getHigherIds(id)) == 2
                and s.getProcess(s.getHigherIds(id)[1]).alive ==
False
            ):
                print(f" :|> Process {i} is new coordinator.")
                s.getProcess(i).coordinator = True
                quit()
    print("\n")
    if len(nextAvailable) == 0:
        print(f" :|> Process {id} is new coordinator.")
        s.getProcess(id).coordinator = True
        quit()
    smaller = nextAvailable[0]
    conductElection(smaller)


conductElection(initiator.id)
```

Output Screenshot :



Q6 Simulate a scenario in distributed systems to implement the Ring Algorithm for choosing
a coordinator node amongst the participative nodes of the system after the collapse of the
existing coordinator node in the system

ringAlgorithm.py

```
import random



class Process:
```

```
    def __init__(self, id, alive):
        self.id = id
        self.alive = alive
        self.cordinator = False
        self.crashNoticer = False


class System:
    allProcesses = []

    def createProcess(self, id, alive):
        process = Process(id, alive)
        self.allProcesses.append(process)

    def getAllProcessess(self):
        return self.allProcesses

    def processessCount(self):
        return len(self.allProcesses)

    def getNextProcess(self, id):
        return self.getProcess(id + 1)

    def getProcess(self, id):
        for process in self.allProcesses:
            if process.id == id:
                return process


s = System()
for i in range(1, 9):
    s.createProcess(i, True)
```

```
p = s.getProcess(5)
p.crashNoticer = True

processes = s.getAllProcessess()
global initiator

for process in processes:
   if process.alive == False:
       print(f" :|> Process {process.id} is crashed\n")
   if process.crashNoticer:
       initiator = process
       print(f" :|> Process {process.id} noticed the crash")

print(f":|> Process {initiator.id} is initiating the
election\n")


electionMessage = []


def conductElection(id):
   process = s.getProcess(id)
   if process != None:
       if process.alive:
           electionMessage.append(process.id)
       if s.getNextProcess(id) == None:
           return None
   conductElection(s.getNextProcess(id).id)


conductElection(initiator.id)

print(f" :|>  Election message is {electionMessage} \n")
```

```
s.getProcess(electionMessage[-1]).coordinator = True

print(f":|> Process {electionMessage[-1]} is the new
coordinator\n")
```

Output Screesnhot: