ARUNIMA SINGH THAKUR
180905218 CSE C-31
DS LAB 6&7
CLOCK SYNCHRONIZATION & MUTUAL EXCLUSION (ELECTION ALGORITHM)

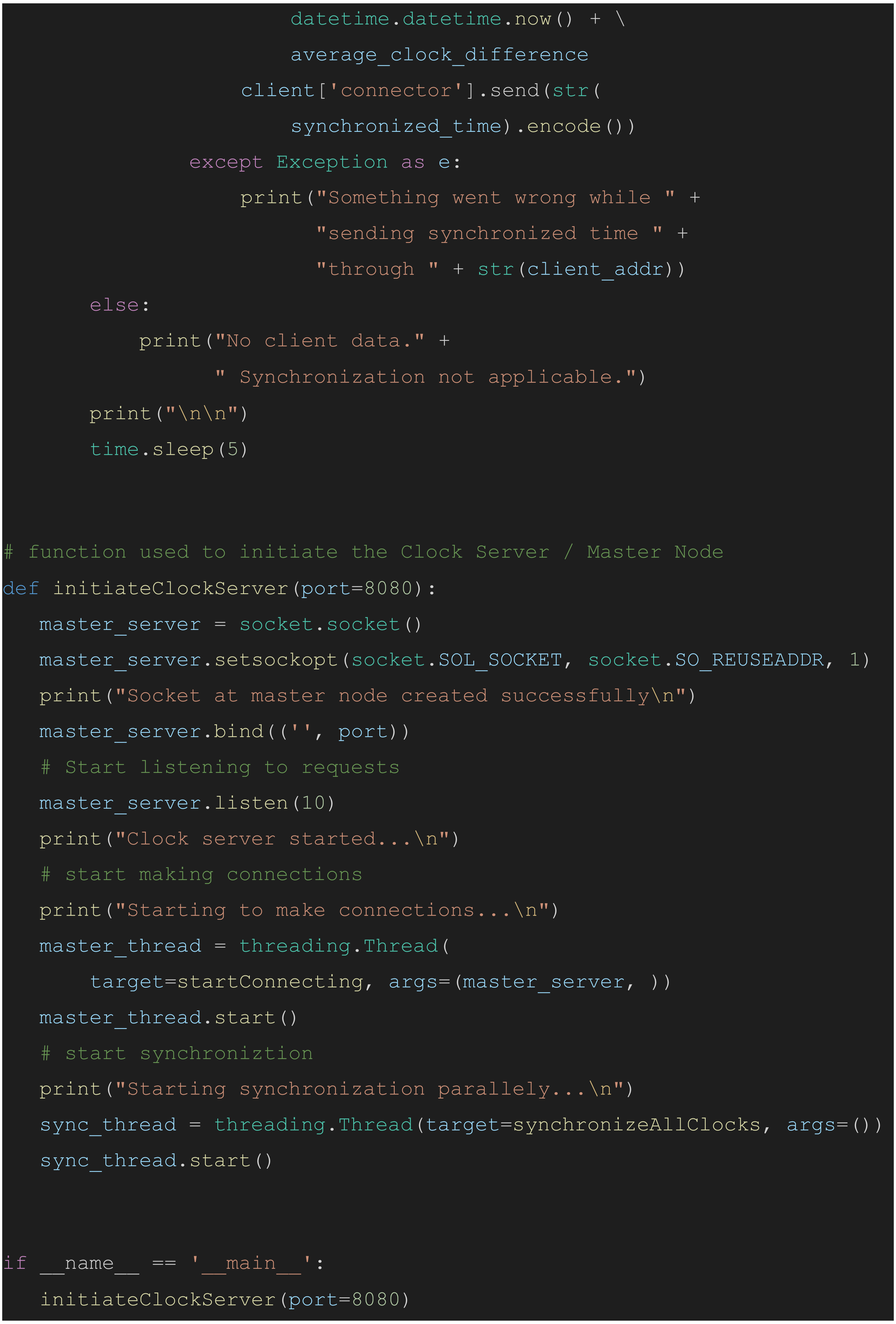
1. **Berkeley’s Algorithm**
   4 clients - KMC, MIT, TAPMI, SOLS institute clocks
   1 master clock server

server.py

```
from dateutil import parser
import threading
import datetime
import socket
import time
# data structure 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:
        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()


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 = \
```

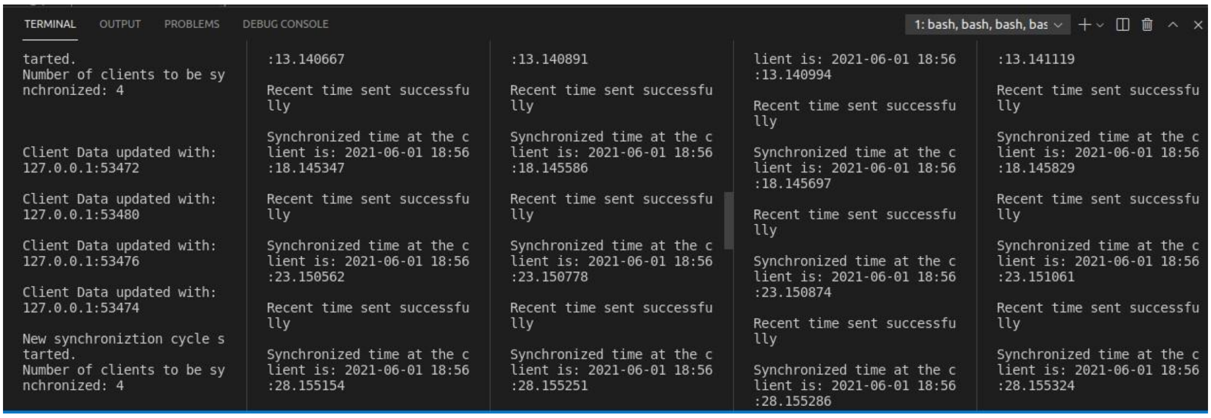


```
                       datetime.datetime.now() + \
                       average_clock_difference
                   client['connector'].send(str(
                       synchronized_time).encode())
               except Exception as e:
                   print("Something went wrong while " +
                         "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()


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
import sys
# 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)


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")


def initiateSlaveClient(port=8080):
   slave_client = socket.socket()
   # connect to the clock server on local computer
   slave_client.connect(('127.0.0.1', port))
   print(sys.argv[1]+" digital clock")
   # 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()


if __name__ == '__main__':
   # initialize the Slave / Client
   initiateSlaveClient(port=8080)
```

Screenshot:



```
tarted.
Number of clients to be sy
nchronized: 4


Client Data updated with:
127.0.0.1:53472

Client Data updated with:
127.0.0.1:53480

Client Data updated with:
127.0.0.1:53476

Client Data updated with:
127.0.0.1:53474

New synchroniztion cycle s
tarted.
Number of clients to be sy
nchronized: 4
```

```
:13.140667

Recent time sent successfu
lly

Synchronized time at the c
lient is: 2021-06-01 18:56
:18.145347

Recent time sent successfu
lly

Synchronized time at the c
lient is: 2021-06-01 18:56
:23.150562

Recent time sent successfu
lly

Synchronized time at the c
lient is: 2021-06-01 18:56
:28.155154
```

```
:13.140891

Recent time sent successfu
lly

Synchronized time at the c
lient is: 2021-06-01 18:56
:18.145586

Recent time sent successfu
lly

Synchronized time at the c
lient is: 2021-06-01 18:56
:23.150778

Recent time sent successfu
lly

Synchronized time at the c
lient is: 2021-06-01 18:56
:28.155251
```

```
lient is: 2021-06-01 18:56
:13.140994

Recent time sent successfu
lly

Synchronized time at the c
lient is: 2021-06-01 18:56
:18.145697

Recent time sent successfu
lly

Synchronized time at the c
lient is: 2021-06-01 18:56
:23.150874

Recent time sent successfu
lly

Synchronized time at the c
lient is: 2021-06-01 18:56
:28.155286
```

```
:13.141119

Recent time sent successfu
lly

Synchronized time at the c
lient is: 2021-06-01 18:56
:18.145829

Recent time sent successfu
lly

Synchronized time at the c
lient is: 2021-06-01 18:56
:23.151061

Recent time sent successfu
lly

Synchronized time at the c
lient is: 2021-06-01 18:56
:28.155324
```

2. **Cristian’s Algorithm**

Taking Laptop as server (UTC reciever)
Mobile as a client for synchronizing

server.py

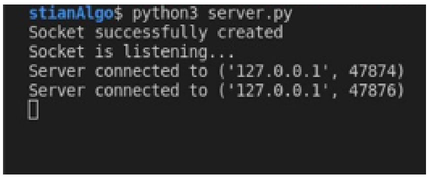
```
import datetime
import time
import socket


def initiateClockServer():
    s = socket.socket()
    print("Socket successfully created")
    port = 8011
    s.bind(('127.0.0.1', port))
    s.listen(5)
    print("Socket is listening...")
    while True:
        connection, address = s.accept()
        print('Server connected to', address)
        serverDateTime = str(datetime.datetime.now())
        connection.send(serverDateTime.encode())
        connection.close()


if __name__ == '__main__':
    initiateClockServer()
```

client.py

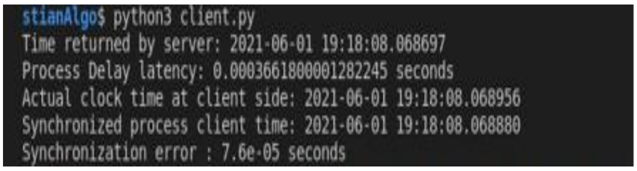
```
import socket
import datetime
from dateutil import parser
from timeit import default_timer as timer


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("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))
```

```
   error = actual_time - client_time
   print("Synchronization error : " + str(error.total_seconds()) + "
seconds")
   s.close()


if __name__ == '__main__':
   synchronizeTime()
```

Screenshot:



```
stianAlgo$ python3 server.py
Socket successfully created
Socket is listening...
Server connected to ('127.0.0.1', 47874)
Server connected to ('127.0.0.1', 47876)
```



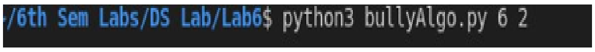
```
stianAlgo$ python3 client.py
Time returned by server: 2021-06-01 19:18:08.068697
Process Delay latency: 0.0003661800001282245 seconds
Actual clock time at client side: 2021-06-01 19:18:08.068956
Synchronized process client time: 2021-06-01 19:18:08.068880
Synchronization error : 7.6e-05 seconds
```

3. **Bully Algorithm CODE:**

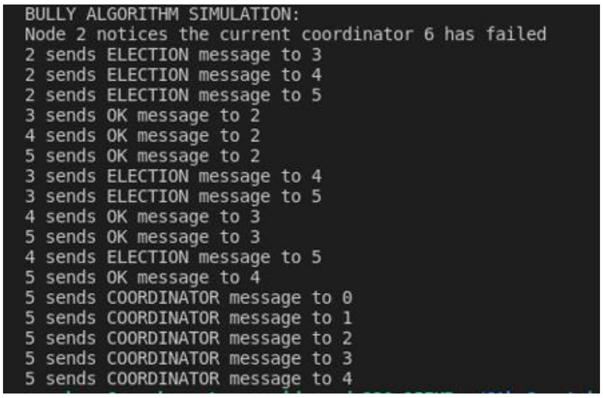
```
import sys


noOfNodes = int(sys.argv[1])
initiatorNode = int(sys.argv[2])


def bully_algorithm():
   print("BULLY ALGORITHM SIMULATION:")
   print('Node %s notices the current coordinator %s has failed'
%
         (initiatorNode, noOfNodes))
   biggerNodes = []
   for i in range(initiatorNode+1, noOfNodes):
       print("%s sends ELECTION message to %s" % (initiatorNode,
i))
       biggerNodes.append(i)
   for i in biggerNodes:
       print("%s sends OK message to %s" % (i, initiatorNode))
```

```
    while len(biggerNodes) != 1:
        i = biggerNodes[0]
        for j in range(i+1, noOfNodes):
            print("%s sends ELECTION message to %s" % (i, j))
        for k in range(i+1, noOfNodes):
            print("%s sends OK message to %s" % (k, i))
        biggerNodes.remove(i)
    newCoordinatorNode = biggerNodes[0]
    for i in range(0, newCoordinatorNode):
        print("%s sends COORDINATOR message to %s" %
(newCoordinatorNode, i))


if __name__ == '__main__':
    bully_algorithm()
```

Screenshot:



```
~/6th Sem Labs/DS Lab/Lab6$ python3 bullyAlgo.py 6 2
```



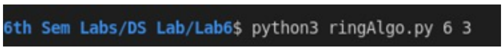
```
BULLY ALGORITHM SIMULATION:
Node 2 notices the current coordinator 6 has failed
2 sends ELECTION message to 3
2 sends ELECTION message to 4
2 sends ELECTION message to 5
3 sends OK message to 2
4 sends OK message to 2
5 sends OK message to 2
3 sends ELECTION message to 4
3 sends ELECTION message to 5
4 sends OK message to 3
5 sends OK message to 3
4 sends ELECTION message to 5
5 sends OK message to 4
5 sends COORDINATOR message to 0
5 sends COORDINATOR message to 1
5 sends COORDINATOR message to 2
5 sends COORDINATOR message to 3
5 sends COORDINATOR message to 4
```

4. **Ring Algorithm CODE:**

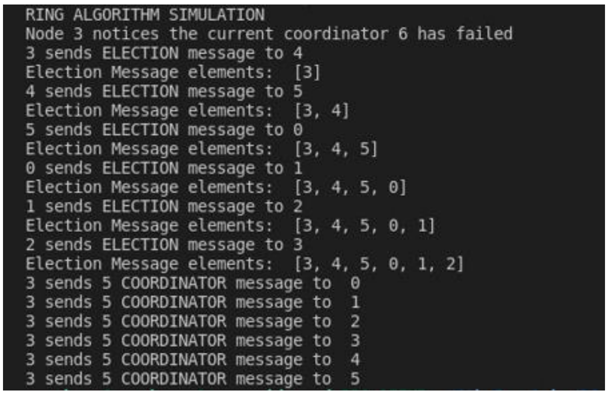
```
import sys


noOfNodes = int(sys.argv[1])
initiatorNode = int(sys.argv[2])
```

```
def ring_algorithm():
   print("RING ALGORITHM")
   print('Node %s notices the current coordinator %s has failed'
%
         (initiatorNode, noOfNodes))
   ELECTION = []

   i = initiatorNode
   while True:
       print("%s sends ELECTION message to %s" % (i, (i+1) %
noOfNodes))
       ELECTION.append(i)
       print("Election Message elements: ", ELECTION)
       i = (i+1) % noOfNodes
       if i == initiatorNode:
           break
   max_ele = max(ELECTION)
   for i in range(0, noOfNodes):
       print("%s sends %s COORDINATOR message to %s" %
             (initiatorNode, max_ele, i))


if __name__ == '__main__':
   ring_algorithm()
```

Screenshot:



```
6th Sem Labs/DS Lab/Lab6$ python3 ringAlgo.py 6 3
```



```
RING ALGORITHM SIMULATION
Node 3 notices the current coordinator 6 has failed
3 sends ELECTION message to 4
Election Message elements:  [3]
4 sends ELECTION message to 5
Election Message elements:  [3, 4]
5 sends ELECTION message to 0
Election Message elements:  [3, 4, 5]
0 sends ELECTION message to 1
Election Message elements:  [3, 4, 5, 0]
1 sends ELECTION message to 2
Election Message elements:  [3, 4, 5, 0, 1]
2 sends ELECTION message to 3
Election Message elements:  [3, 4, 5, 0, 1, 2]
3 sends 5 COORDINATOR message to  0
3 sends 5 COORDINATOR message to  1
3 sends 5 COORDINATOR message to  2
3 sends 5 COORDINATOR message to  3
3 sends 5 COORDINATOR message to  4
3 sends 5 COORDINATOR message to  5
```