Name : Rajvardhan Reddy
Reg no : 180905093
Sec : B , Batch – B1
Roll No : 19

DS Lab - Week 6 : CLOCK SYNCHRONIZATION & MUTUAL EXCLUSION (ELECTION ALGORITHM)

EXAMPLE PROGRAMS :

A. Cristian's algorithm

I) To initiate a prototype of a clock server on local machine:

Server:

```
# Python3 program imitating a clock server

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 = 8012

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()
```

Output Sever Side:

```
user@user-VirtualBox:~/Desktop$ python3 server.py
Socket successfully created
Socket is listening...
Server connected to ('127.0.0.1', 55998)
```

II) Code below is used to initiate a prototype of a client process on local machine:

```
# Python3 program imitating a client process
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 = 8012
# 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 Client Side:

```
user@user-VirtualBox:~/Desktop$ python3 client.py
Time returned by server: 2021-06-02 12:52:36.716402
Process Delay latency: 0.0006133880015113391 seconds
Actual clock time at client side: 2021-06-02 12:52:36.716862
Synchronized process client time: 2021-06-02 12:52:36.716709
Synchronization error : 0.000153 seconds
user@user-VirtualBox:~/Desktop$
```

B. Berkeley's algorithm:

Server Side:

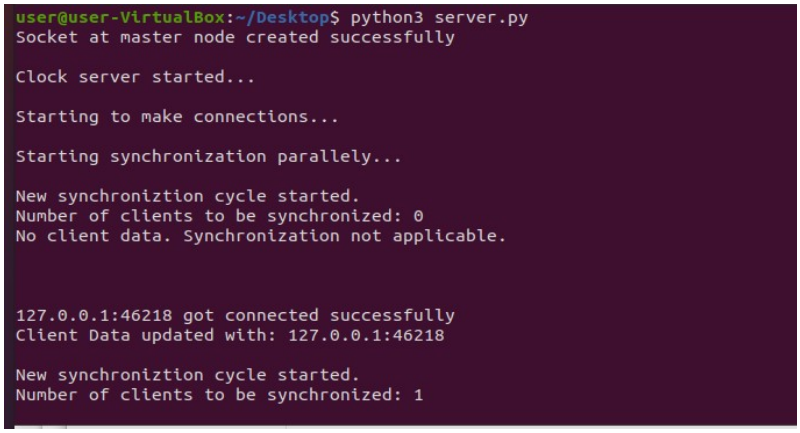
```
# Python3 program imitating a clock server
from functools import reduce
from dateutil import parser
import threading
import datetime
import socket
import time
# datastructure used to store client address and clock data
client_data = {}
''' nested thread function used to receive
clock time from a connected client '''
def 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)
```

Server_Output:



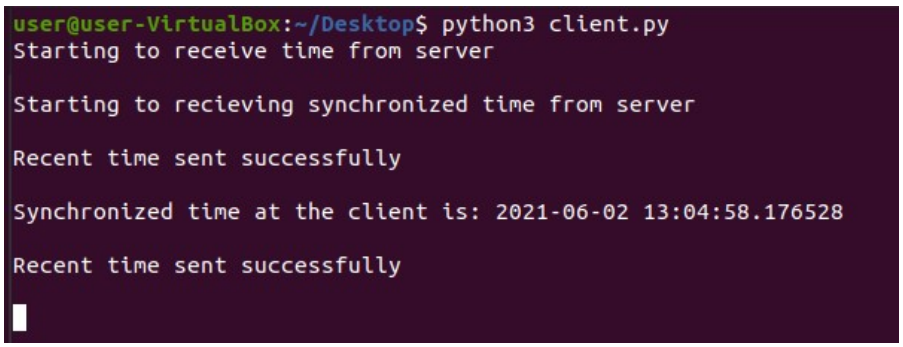
Client Side :

```
# 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 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)
```

Client_Output:



Lab Exercises:

P1)

q1_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 = 8080):
    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 = 8080)
```

KMC:

```
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 = 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 = (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 = 8080)
```

MIT:

```
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 = 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 = (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 = 8080)
```

TAPMI:

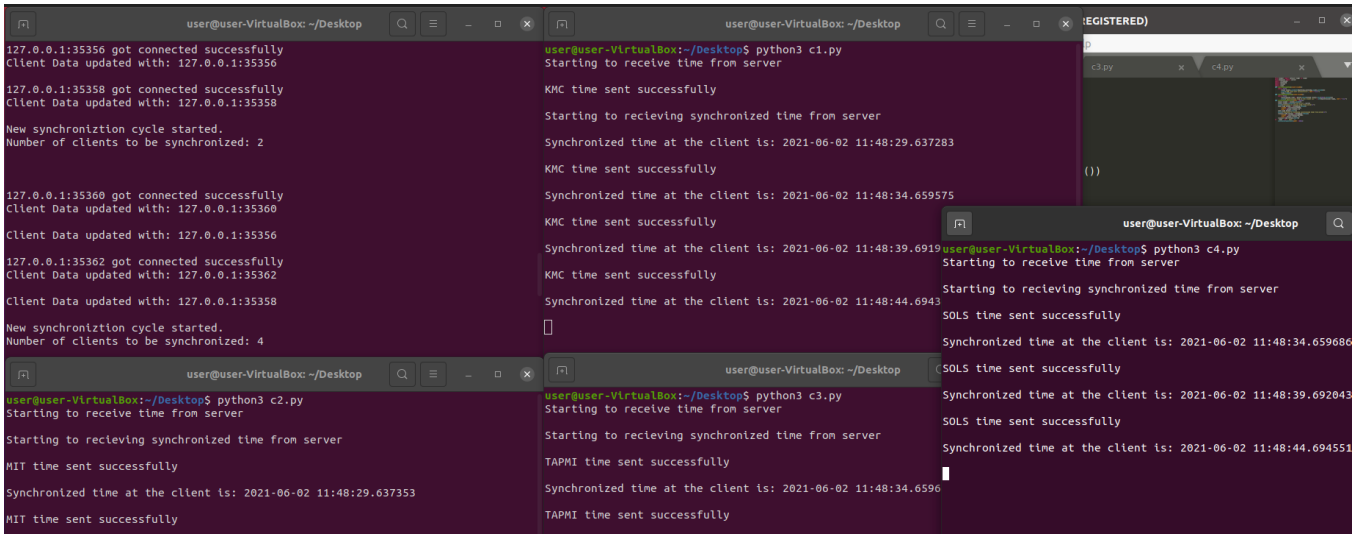
```
from timeit import default_timer as timer
from dateutil import parser
import threading
import datetime
import socket
import time
def startSendingTime(slave_client):
	while True:
		slave_client.send(str(datetime.datetime.now()).encode())
		print("TAPMI time sent successfully", end = "\n\n")
		time.sleep(5)
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 = 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 = (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 = 8080)
```

SOLS:

```
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 = 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 = (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 = 8080)
```

OUTPUT :



P2)

Server:

```
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()
```

mobile app – exam fees (client 1):

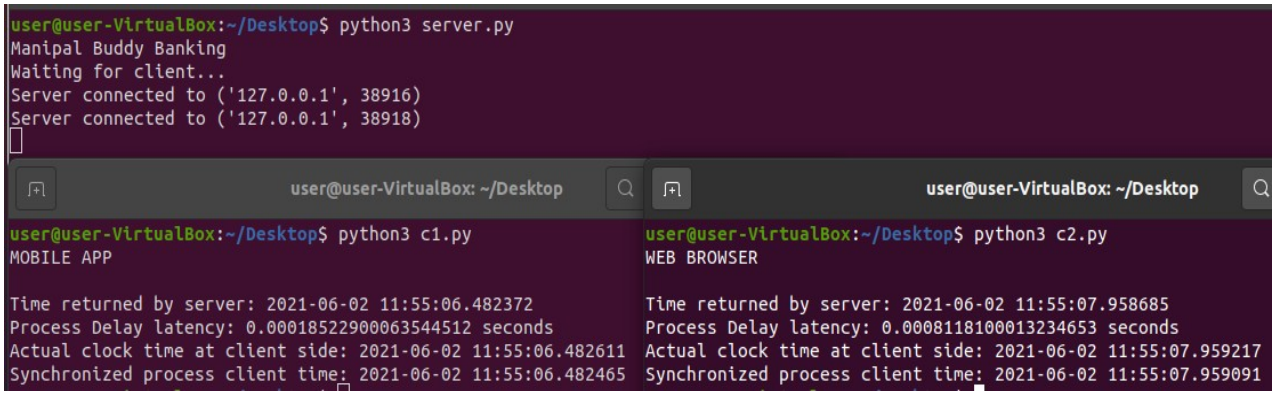
```
import socket
import datetime
import time
from dateutil import parser
from timeit import default_timer as timer
def synchronizeTime():
	print("MOBILE APP\n")
	s = socket.socket()
	port = 8011
	s.connect(('127.0.0.1', port))
	request_time = timer()
	server_time = parser.parse(s.recv(1024).decode())
	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()
```

web browser - NPTEL (client 2):

```
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 :



P3)

Bully.py:

```
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()
```

OUTPUT:

```
user@user-VirtualBox:~/Desktop$ python3 bully.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
```

P4)

Server:

```
import sys
import threading
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = socket.gethostname()
port = 7777
try:
    s.bind((host, port))
except socket.error as msg:
    print("bind failed" + str(msg))
    sys.exit()
s.listen(10)
process_sockets_list = []
process_list = []
```

```
neighbor_list = []
msg_token = ""
def recv_message(conn):
    while True:
        try:
            received = conn.recv(1024)
            msg_token = received.decode('utf-8')
            print("received token: " + msg_token)
        except:
            continue
        if "Coordinator: " in msg_token :
            le=msg_token.split()
            leader=le[1]
        process_index = process_sockets_list.index(conn)
        if len(process_sockets_list)==process_index+1 :
            to_process=0
        else :
            to_process=process_index+1
        try :
            process_sockets_list[to_process].send(received)
            print("sending :" + received.decode('utf-8'))
        except :
            if process_list[to_process]!=leader :
                process_sockets_list[to_process+1].send(received)
                print("sending :" + received.decode('utf-8'))
            process_sockets_list[to_process].close()
            process_sockets_list.remove(process_sockets_list[to_process])
            process_list.remove(process_list[to_process])
            continue
while True:
    try:
        connection, addr = s.accept()
        process_sockets_list.append(connection)
        recv_process_id = connection.recv(1024)
        from_to_process = recv_process_id.decode('utf-8')
        process_list.append(from_to_process)
        print("Process: " + from_to_process)
        start_thread = threading.Thread(target=recv_message, args=(connection,))
        start_thread.start()
    except socket.error as msg:
        print("thread failed"+msg)
connection.close()
s.close()
```

client 1:

```
import socket
import threading
import time
import select
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = socket.gethostname()
to_port = 7777
```

```
s.connect((host, to_port))
my_id = "0"
s.send(my_id.encode('utf-8'))
leader="-1"

def initiate_election(s):
    time.sleep(1)
    s.send(my_id.encode('utf-8'))
    print("token sent: " + my_id)
    print("Election initiated")
def Ring_Election_Algorithm(s):

    while True:
        global leader
        try:
            s.settimeout(15)
            received = s.recv(1024)
            s.settimeout(None)
            received_token_list = received.decode('utf-8')
        except socket.timeout:
            leader = "0"
            initiate_election(s)
            continue
        if my_id in received_token_list and "Coordinator: " not in received_token_list and "hello" not
in received_token_list:
            leader = max(received_token_list)
            forwarding_leader = "Coordinator: " + leader
            time.sleep(1)
            s.send(forwarding_leader.encode('utf-8'))
        elif my_id not in received_token_list and "Coordinator: " not in received_token_list and
"hello" not in received_token_list :

            print("rec tok: " + received_token_list)

            leader = "0"
            received_token_list = received_token_list + " " + my_id

            time.sleep(1)
            s.send(received_token_list.encode('utf-8'))
            print("adding token: " + received_token_list)
        elif ("hello" in received_token_list or "Coordinator: " in received_token_list )and leader=="-1"
:
                leader="0"
                initiate_election(s)
        elif "Coordinator: " in received_token_list and leader not in received_token_list :
            print(received_token_list)
            le=received_token_list.split()
            leader=le[1]
            time.sleep(1)
            s.send(received_token_list.encode('utf-8'))
        else :
            if leader=="-1" or leader=="0":
```

```
            continue
        else :
            print(received_token_list)
            communicate = "hello" + " from " + my_id
            time.sleep(1)
            s.send(communicate.encode('utf-8'))
            continue
recv_thread = threading.Thread(target=Ring_Election_Algorithm, args=(s,))
recv_thread.start()
recv_thread.join()
s.close()
```

client 2:

```
import socket
import threading
import time
import select
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = socket.gethostname()
to_port = 7777
s.connect((host, to_port))
my_id = "1"
s.send(my_id.encode('utf-8'))
leader="-1"
def initiate_election(s):
    time.sleep(1)
    s.send(my_id.encode('utf-8'))
    print("token sent: " + my_id)
    print("Election initiated")
def Ring_Election_Algorithm(s):

    while True:
        global leader
        try:
            s.settimeout(15)
            received = s.recv(1024)
            s.settimeout(None)
            received_token_list = received.decode('utf-8')
        except socket.timeout:
            leader = "0"
            initiate_election(s)
            continue
        if my_id in received_token_list and "Coordinator: " not in received_token_list and "hello" not
in received_token_list:
            leader = max(received_token_list)
            forwarding_leader = "Coordinator: " + leader
            time.sleep(1)
            s.send(forwarding_leader.encode('utf-8'))
        elif my_id not in received_token_list and "Coordinator: " not in received_token_list and
"hello" not in received_token_list :

            print("rec tok: " + received_token_list)
            leader = "0"
```

```
            received_token_list = received_token_list + " " + my_id
            time.sleep(1)
            s.send(received_token_list.encode('utf-8'))
            print("adding token: " + received_token_list)
        elif ("hello" in received_token_list or "Coordinator: " in received_token_list )and leader=="-1"
:
                leader="0"
                initiate_election(s)
        elif "Coordinator: " in received_token_list and leader not in received_token_list :
            print(received_token_list)
            le=received_token_list.split()
            leader=le[1]
            time.sleep(1)
            s.send(received_token_list.encode('utf-8'))
        else :
            if leader=="-1" or leader=="0":
                continue
            else :
                print(received_token_list)
                communicate = "hello" + " from " + my_id
                time.sleep(1)
                s.send(communicate.encode('utf-8'))
                continue
recv_thread = threading.Thread(target=Ring_Election_Algorithm, args=(s,))
recv_thread.start()
recv_thread.join()
s.close()
```

client 3:

```
import socket
import threading
import time
import select
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = socket.gethostname()
to_port = 7777
s.connect((host, to_port))
my_id = "2"
s.send(my_id.encode('utf-8'))
leader="-1"
def initiate_election(s):
    time.sleep(1)
    s.send(my_id.encode('utf-8'))
    print("token sent: " + my_id)
    print("Election initiated")
def Ring_Election_Algorithm(s):

    while True:
        global leader
        try:
            s.settimeout(15)
            received = s.recv(1024)
```

```
            s.settimeout(None)
            received_token_list = received.decode('utf-8')
        except socket.timeout:
            leader = "0"
            initiate_election(s)
            continue
        if my_id in received_token_list and "Coordinator: " not in received_token_list and "hello" not
in received_token_list:
            leader = max(received_token_list)
            forwarding_leader = "Coordinator: " + leader
            time.sleep(1)
            s.send(forwarding_leader.encode('utf-8'))
        elif my_id not in received_token_list and "Coordinator: " not in received_token_list and
"hello" not in received_token_list :
            print("rec tok: " + received_token_list)
            leader = "0"
            received_token_list = received_token_list + " " + my_id
            time.sleep(1)
            s.send(received_token_list.encode('utf-8'))
            print("adding token: " + received_token_list)
        elif ("hello" in received_token_list or "Coordinator: " in received_token_list )and leader=="-1"
:
                leader="0"
                initiate_election(s)
        elif "Coordinator: " in received_token_list and leader not in received_token_list :
            print(received_token_list)
            le=received_token_list.split()
            leader=le[1]
            time.sleep(1)
            s.send(received_token_list.encode('utf-8'))
        else :
            if leader=="-1" or leader=="0":
                continue
            else :
                print(received_token_list)
                communicate = "hello" + " from " + my_id
                time.sleep(1)
                s.send(communicate.encode('utf-8'))
                continue

recv_thread = threading.Thread(target=Ring_Election_Algorithm, args=(s,))
recv_thread.start()
recv_thread.join()
s.close()
```

OUTPUT:

```
user@user-VirtualBox:~/Desktop$ python3 server.py
Process: 0
Process: 1
Process: 2
received token: 0
sending :0
received token: 0 1
sending :0 1
received token: 0 1 2
sending :0 1 2
received token: Coordinator: 2
sending :Coordinator: 2
received token: Coordinator: 2
sending :Coordinator: 2
received token: Coordinator: 2
sending :Coordinator: 2
received token: hello from 0
sending :hello from 0
received token: hello from 1
sending :hello from 1
received token: hello from 2
sending :hello from 2
received token: hello from 0
sending :hello from 0
```

```
user@user-VirtualBox:~/Desktop$ python3 c1.py
token sent: 0
Election initiated
Coordinator: 2
hello from 2
hello from 2
hello from 2
hello from 2
hello from 2
hello from 2
hello from 2
hello from 2
hello from 2
```

user@user-VirtualBox: ~/Desktop

```
user@user-VirtualBox:~/Desktop$ python3 c2.py
rec tok: 0
adding token: 0 1
Coordinator: 2
hello from 0
hello from 0
hello from 0
hello from 0
hello from 0
```

user@user-VirtualBox:

```
user@user-VirtualBox:~/Desktop$ python3 c3.py
rec tok: 0 1
adding token: 0 1 2
Coordinator: 2
hello from 1
hello from 1
hello from 1
hello from 1
hello from 1
```