**DS Lab 6-7 – CLOCK SYNCHRONIZATION & MUTUAL EXCLUSION (ELECTION ALGORITHM)**

**Name:** GP Anirudh

**Registration Number:** 180905452

**Section:** B

**Batch:** B2

**Roll Number:** 59

**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

import socket

# 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 Server:**

**Text

Description automatically generated**

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

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:**

**Graphical user interface, text

Description automatically generated**

**B. Berkeley’s algorithm:**

**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)

**Output Server:**

**Text

Description automatically generatedGraphical user interface, text

Description automatically generated with medium confidence**

**Client:**

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 Client:**

**A screenshot of a computer

Description automatically generated with medium confidenceGraphical user interface, text, application

Description automatically generated**

**Lab Exercises:**

**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 = 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.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 = 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.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 = 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.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 = 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.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 = 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 (Client and Server):**

**Text

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generated**

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()

**mobappexamfees.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()

**webbrownptel.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 (Client + Server):**

**Text

Description automatically generated**

**Graphical user interface, text

Description automatically generatedGraphical user interface, text

Description automatically generated**

3.)

**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:**

**Text

Description automatically generated**

**Graphical user interface, text

Description automatically generated**

4.)

**server.py:**

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])

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()

**cl1.py:**

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()

**cl2.py:**

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:

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()

**cl3.py:**

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:

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:**

**Text

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generated**