

## WEEK6&7 IT\_LAB : CLOCK SYNCHRONIZATION & MUTUAL EXCLUSION (ELECTION ALGORITHM)

**Name: Sagnik Chatterjee**

**Reg: 180905478**

**Roll No: 61**

**Sem: 6**

**Section :B**

### **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:

```

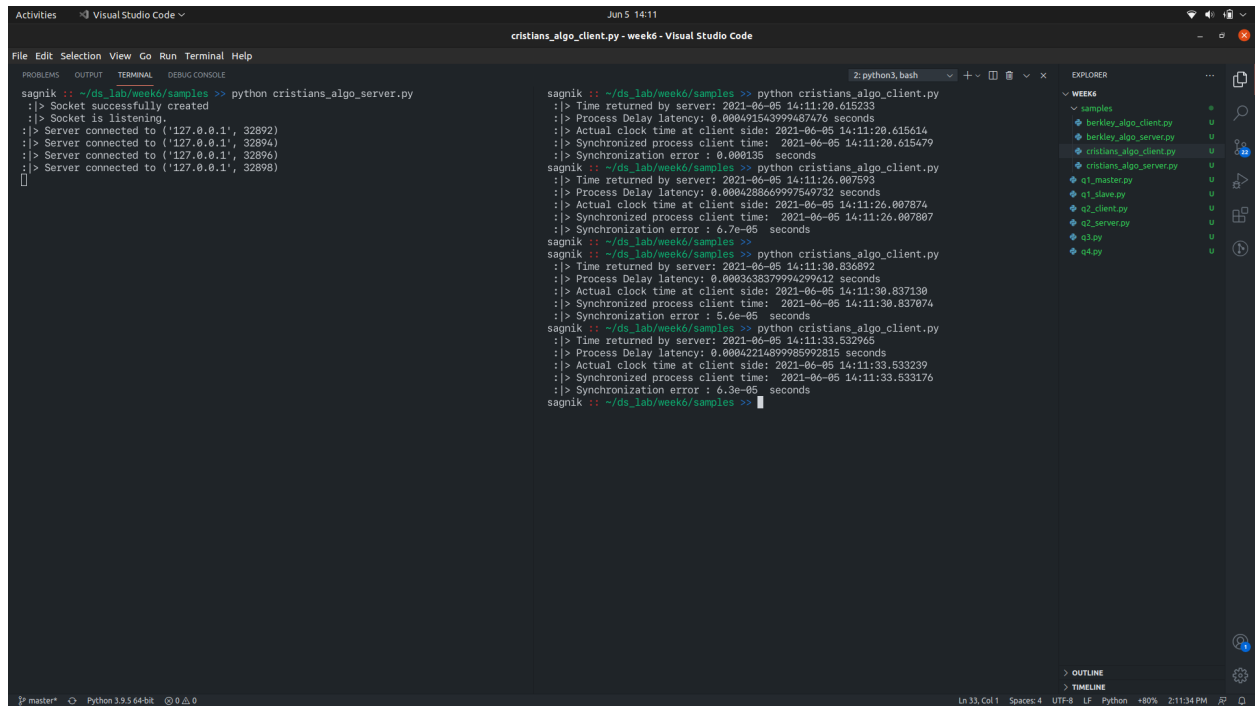
sagnik : ~/ds_lab/week6/samples >> python cristians_algo_server.py
:|> Socket successfully created
:|> Socket is listening.
:|> Server connected to ('127.0.0.1', 32892)
:|> Server connected to ('127.0.0.1', 32894)
[]

sagnik : ~/ds_lab/week6/samples >> python cristians_algo_client.py
:|> Time returned by server: 2021-06-05 14:11:20.615233
:|> Process Delay latency: 0.000491543999487476 seconds
:|> Actual clock time at client side: 2021-06-05 14:11:20.615614
:|> Synchronized process client time: 2021-06-05 14:11:20.615479
:|> Synchronization error : 0.000135 seconds

sagnik : ~/ds_lab/week6/samples >> python cristians_algo_server.py
:|> Time returned by server: 2021-06-05 14:11:26.007593
:|> Process Delay latency: 0.0004288669977549732 seconds
:|> Actual clock time at client side: 2021-06-05 14:11:26.007874
:|> Synchronized process client time: 2021-06-05 14:11:26.007807
:|> Synchronization error : 6.7e-05 seconds

sagnik : ~/ds_lab/week6/samples >>
sagnik : ~/ds_lab/week6/samples >>

```



```
Activities Visual Studio Code Jun 5 14:11 cristians_algo_client.py - week6 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
sagnik :: ~/ds_lab/week6/samples >> python cristians_algo_server.py
:> Socket successfully created
:> Socket is listening.
:> Server connected to ('127.0.0.1', 32892)
:> Server connected to ('127.0.0.1', 32894)
:> Server connected to ('127.0.0.1', 32896)
:> Server connected to ('127.0.0.1', 32898)
sagnik :: ~/ds_lab/week6/samples >> python cristians_algo_client.py
:> Time returned by server: 2021-06-05 14:11:20.615233
:> Process Delay latency: 0.000491543999487476 seconds
:> Actual clock time at client side: 2021-06-05 14:11:20.615614
:> Synchronized process client time: 2021-06-05 14:11:20.615479
:> Synchronization error : 0.000135 seconds
sagnik :: ~/ds_lab/week6/samples >> python cristians_algo_client.py
:> Time returned by server: 2021-06-05 14:11:26.007593
:> Process Delay latency: 0.000428866997549732 seconds
:> Actual clock time at client side: 2021-06-05 14:11:26.007874
:> Synchronized process client time: 2021-06-05 14:11:26.007807
:> Synchronization error : 6.7e-05 seconds
sagnik :: ~/ds_lab/week6/samples >> python cristians_algo_client.py
:> Time returned by server: 2021-06-05 14:11:30.836892
:> Process Delay latency: 0.0003638379994299612 seconds
:> Actual clock time at client side: 2021-06-05 14:11:30.837130
:> Synchronized process client time: 2021-06-05 14:11:30.837074
:> Synchronization error : 5.6e-05 seconds
sagnik :: ~/ds_lab/week6/samples >> python cristians_algo_client.py
:> Time returned by server: 2021-06-05 14:11:33.532965
:> Process Delay latency: 0.00042214899785922815 seconds
:> Actual clock time at client side: 2021-06-05 14:11:33.533239
:> Synchronized process client time: 2021-06-05 14:11:33.533176
:> Synchronization error : 6.3e-05 seconds
sagnik :: ~/ds_lab/week6/samples >>
EXPLORER
samples
  cristians_algo_client.py
  cristians_algo_server.py
  q1_master.py
  q1_slave.py
  q2_client.py
  q2_server.py
  q3.py
  q4.py
OUTLINE
TIMELINE
Ln 33, Col 1 Spaces: 4 UTF-8 LF Python +80% 2:11:34 PM
```

## 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:**



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)

```

## Slave.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__":

    list_clients = ["KMC", "MIT", "TAMPI", "SOLS"]
    for client in list_clients:
        initiateSlaveClient(client, port=8080)

```

## Output:

```

sagnik :: ~/ds_lab/week6 >> python cl_master.py
:|> Socket at master node created successfully.

:|> Clock server started.

:|> Starting to make connection.

:|> Starting synch parallelly.

:|> New synchronization cycle started.
:|> Number of clients to be synchronized: 0
:|> No client data. Synchronization not applicable.
:|> 127.0.0.1:56344 got connected successfully
:|> Client Data updated with: 127.8.6.1:56344
:|> 127.0.0.1:56345 got connected successfully
:|> Client Data updated with: 127.8.6.1:56346
:|> 127.0.0.1:56349 got connected successfully
:|> Client Data updated with: 127.8.6.1:56348
:|> 127.0.0.1:56359 got connected successfully
:|> Client Data updated with: 127.8.6.1:56350
:|> New synchronization cycle started.
:|> Number of clients to be synchronized: 4
:|> Client Data updated with: 127.8.6.1:56344
:|> Client Data updated with: 127.8.6.1:56346
:|> Client Data updated with: 127.8.6.1:56350
:|> Client Data updated with: 127.8.6.1:56348
:|> New synchronization cycle started.
:|> Number of clients to be synchronized: 4
:|> Client Data updated with: 127.8.6.1:56348
:|> Client Data updated with: 127.8.6.1:56344
:|> Client Data updated with: 127.8.6.1:56350
:|> Client Data updated with: 127.8.6.1:56346

Starting to receive time from server
:|> Starting to receiving synchronized time from server
:|> Recent time sent successfully by KMC

Starting to receive time from server
:|> Starting to receiving synchronized time from server
:|> Recent time sent successfully by MIT

Starting to receive time from server
:|> Starting to receiving synchronized time from server
:|> Recent time sent successfully by TAMPI

Starting to receive time from server
:|> Starting to receiving synchronized time from server
:|> Recent time sent successfully by SOLS

:|> Synchronized time at the client is KMC : 2021-06-05 16:34:11.500927
:|> Synchronized time at the client is MIT : 2021-06-05 16:34:11.501102
:|> Synchronized time at the client is SOLS : 2021-06-05 16:34:11.501208
:|> Synchronized time at the client is TAMPI : 2021-06-05 16:34:11.501161
:|> Recent time sent successfully by TAMPI

:|> Recent time sent successfully by KMC
:|> Recent time sent successfully by MIT
:|> Recent time sent successfully by SOLS

:|> Synchronized time at the client is KMC : 2021-06-05 16:34:16.504750
:|> Synchronized time at the client is MIT : 2021-06-05 16:34:16.504889
:|> Synchronized time at the client is SOLS : 2021-06-05 16:34:16.504982
:|> Synchronized time at the client is TAMPI : 2021-06-05 16:34:16.504940
:|> Recent time sent successfully by TAMPI

:|> Recent time sent successfully by MIT
:|> Recent time sent successfully by KMC
:|> Recent time sent successfully by SOLS

```

**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()
```

## Client.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 - DEVICE")
    synchronizeTime("LAPTOP - DEVICE")

```

## Output:

```

sagnik ~ /ds Lab/week4 >> python q2_server.py
:> Socket successfully created
:> Socket is listening
:> Server connected to (127.0.0.1, 33934)
:> Server connected to (127.0.0.1, 33936)
:> Server connected to (127.0.0.1, 33938)
:> Server connected to (127.0.0.1, 33940)
:> Server connected to (127.0.0.1, 33942)
:> Server connected to (127.0.0.1, 33944)
|

sagnik ~ /ds Lab/week4 >> python q2_client.py
:> Synchronising now :- MOBILE - DEVICE
:> Time returned by server: 2021-06-05 16:55:36.674441
:> Process Delay latency: 0.001040847599320249 seconds
:> Actual clock time at client side: 2021-06-05 16:55:36.675321
:> Synchronized process client time: 2021-06-05 16:55:36.674961
:> Synchronization error : 0.00036 seconds.
:> Synchronising now :- LAPTOP - DEVICE
:> Time returned by server: 2021-06-05 16:55:36.679868
:> Process Delay latency: 0.0005021469940728666 seconds
:> Actual clock time at client side: 2021-06-05 16:55:36.676175
:> Synchronized process client time: 2021-06-05 16:55:36.676139
:> Synchronization error : 3.6e-05 seconds.
sagnik ~ /ds Lab/week4 >> python q2_client.py
:> Synchronising now :- MOBILE - DEVICE
:> Time returned by server: 2021-06-05 16:55:39.800704
:> Process Delay latency: 0.00038213108016127666 seconds
:> Actual clock time at client side: 2021-06-05 16:55:39.800964
:> Synchronized process client time: 2021-06-05 16:55:39.800964
:> Synchronization error : 5.9e-05 seconds.
:> Synchronising now :- LAPTOP - DEVICE
:> Time returned by server: 2021-06-05 16:55:39.801179
:> Process Delay latency: 0.00027226480615933905 seconds
:> Actual clock time at client side: 2021-06-05 16:55:39.801311
:> Synchronized process client time: 2021-06-05 16:55:39.801315
:> Synchronization error : -4e-06 seconds.
sagnik ~ /ds Lab/week4 >> python q2_client.py
:> Synchronising now :- MOBILE - DEVICE
:> Time returned by server: 2021-06-05 16:55:42.984165
:> Process Delay latency: 0.00047941680613456587 seconds
:> Actual clock time at client side: 2021-06-05 16:55:42.984461
:> Synchronized process client time: 2021-06-05 16:55:42.984375
:> Synchronization error : 7.6e-05 seconds.
:> Synchronising now :- LAPTOP - DEVICE
:> Time returned by server: 2021-06-05 16:55:42.984609
:> Process Delay latency: 0.0002467679967603333 seconds
:> Actual clock time at client side: 2021-06-05 16:55:42.984793
:> Synchronized process client time: 2021-06-05 16:55:42.984782
:> Synchronization error : 1.1e-05 seconds.
sagnik ~ /ds Lab/week4 >>

```

**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 :**



```

sagnik :: ~/ds_lab/week6 >> python q3.py
:|> Process 5 noticed the crash.
:|> Process 5 is initiating the election.
:|> Message from process 6 to 5 is OK
:|> Message from process 7 to 5 is OK
:|> Message from process 8 to 5 is OK
:|> Message from process 9 to 5 is OK
:|> Message from process 10 to 5 is OK

:|> Message from process 7 to 6 is OK
:|> Message from process 8 to 6 is OK
:|> Message from process 9 to 6 is OK
:|> Message from process 10 to 6 is OK

:|> Message from process 8 to 7 is OK
:|> Message from process 9 to 7 is OK
:|> Message from process 10 to 7 is OK

:|> Message from process 9 to 8 is OK
:|> Message from process 10 to 8 is OK

:|> Message from process 10 to 9 is OK

:|> Process 10 is new coordinator.
sagnik :: ~/ds_lab/week6 >> █

```

**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 getAllProcesses(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 Screenshot:

```
sagnik :: ~/ds_lab/week6 >> python q4.py
:|> Process 5 noticed the crash
:|> Process 5 is initiating the election

:|> Election message is [5, 6, 7, 8]

:|> Process 8 is the new coordinator

sagnik :: ~/ds_lab/week6 >> █
```