LAB 6&7

Clock Synchronisation and Mutual Exclusion

AKSHAT KANSAL

180905206

Section – B,29

# Example Problems

1. Christians algorithm

Server

```
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('Server connected to', address)
        connection.send(str(datetime.datetime.now()).encode())
```

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

client

```
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()
```

```
Socket successfully created
Socket is listening...
Server connected to ('127.0.0.1', 57282)
```

```
Time returned by server: 2021-06-01 13:48:42.102021
Process Delay latency: 0.00033354999868606683 seconds
Actual clock time at client side: 2021-06-01 13:48:42.102263
Synchronized process client time: 2021-06-01 13:48:42.102188
Synchronization error : 7.5e-05 seconds
```

2. Berkeley algorithm

Server

```
from functools import reduce
from dateutil import parser
import threading
import datetime
import socket
import time
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):
    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("Socket at master node created successfully\n")
    master_server.bind(('', port))
    master_server.listen(10)
    print("Clock server started...\n")

    print("Starting to make connections...\n")
    master_thread = threading.Thread(
        target=startConnecting,
```

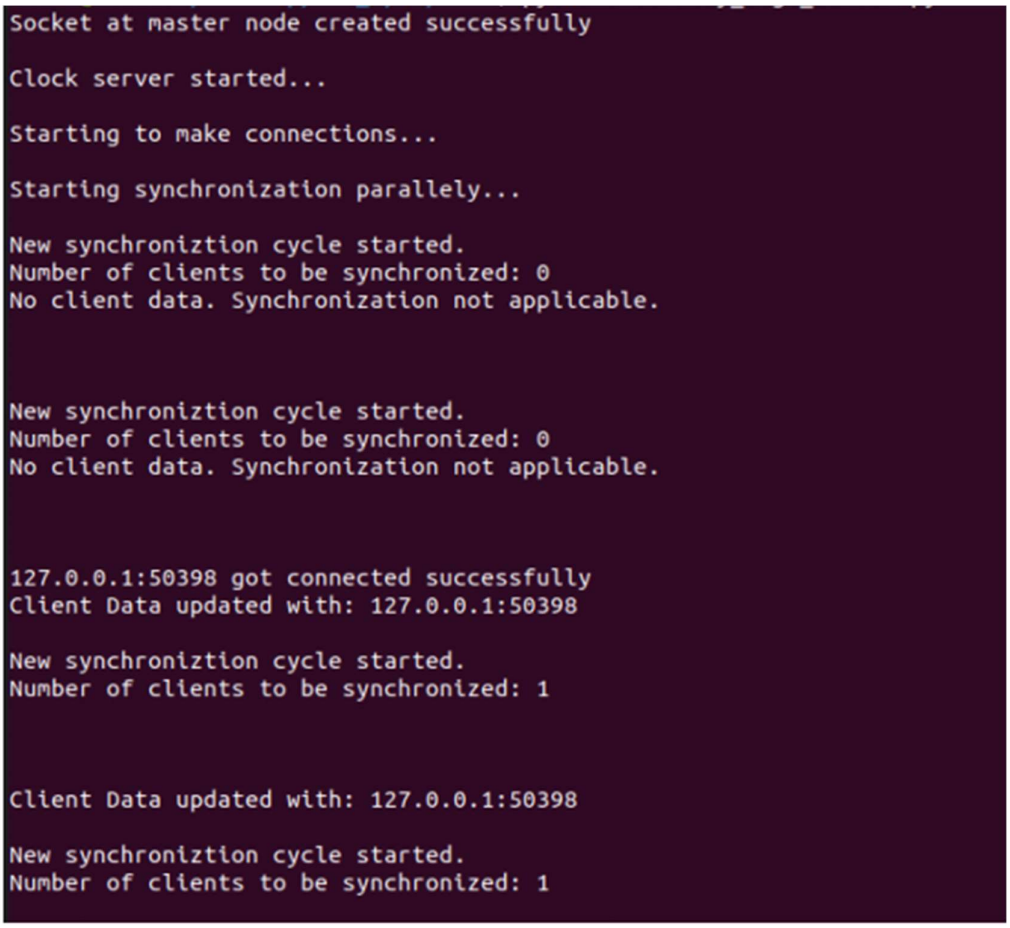
```
        args=(master_server, ))
    master_thread.start()

    print("Starting synchronization parallely...\n")
    sync_thread = threading.Thread(
        target=synchronizeAllClocks,
        args=())
    sync_thread.start()


if __name__ == '__main__':
    initiateClockServer(port=8080)
```

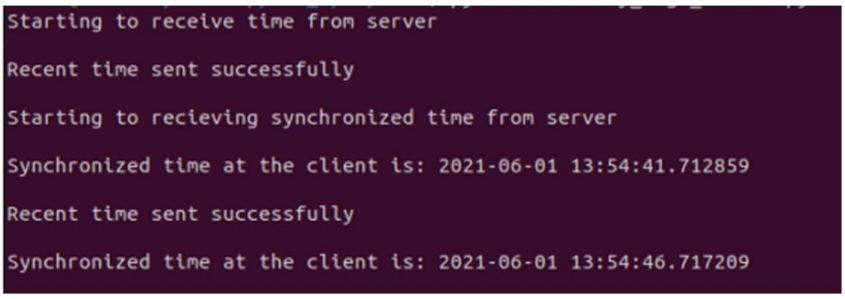
Client

```
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("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)
```



1.

Master time server

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

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):
    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("Socket at master node created successfully\n")
    master_server.bind(('', port))
    master_server.listen(10)
    print("Clock server started...\n")

    print("Starting to make connections...\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)
```

Slaves

```
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("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("Synchronized time at the client is "+name+": " +
              str(Synchronized_time),
              end="\n\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("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","TAPMI","SOLS"]
    for client in list_clients:
        initiateSlaveClient(client,port=8080)
```

```
Socket at master node created successfully

Clock server started...

Starting to make connections...

Starting synchronization parallely...

New synchroniztion cycle started.
Number of clients to be synchronized: 0
No client data. Synchronization not applicable.



127.0.0.1:47974 got connected successfully
Client Data updated with: 127.0.0.1:47974

127.0.0.1:47976 got connected successfully
Client Data updated with: 127.0.0.1:47976

127.0.0.1:47978 got connected successfully
Client Data updated with: 127.0.0.1:47978

127.0.0.1:47980 got connected successfully
Client Data updated with: 127.0.0.1:47980

New synchroniztion cycle started.
Number of clients to be synchronized: 4



Client Data updated with: 127.0.0.1:47976

Client Data updated with: 127.0.0.1:47974

Client Data updated with: 127.0.0.1:47980

Client Data updated with: 127.0.0.1:47978
```

```
Starting to receive time from server

Starting to recieving synchronized time from server

Recent time sent successfully by KMC

Starting to receive time from server

Recent time sent successfully by MIT

Starting to recieving synchronized time from server

Starting to receive time from server

Starting to recieving synchronized time from server

Recent time sent successfully by TAPMI

Starting to receive time from server

Recent time sent successfully by SOLS

Starting to recieving synchronized time from server

Synchronized time at the client is KMC: 2021-06-01 22:52:50.267287

Synchronized time at the client is MIT: 2021-06-01 22:52:50.267482

Synchronized time at the client is SOLS: 2021-06-01 22:52:50.267591

Synchronized time at the client is TAPMI: 2021-06-01 22:52:50.267543

Recent time sent successfully by MIT

Recent time sent successfully by KMC

Recent time sent successfully by SOLS

Recent time sent successfully by TAPMI
```

2.

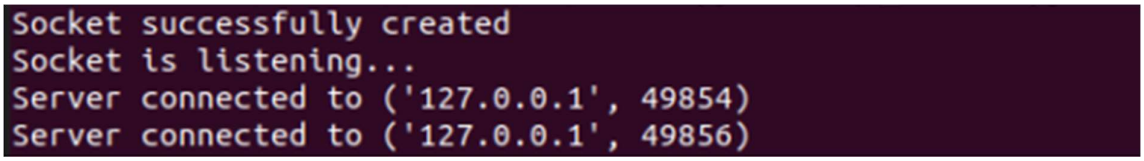
Master server

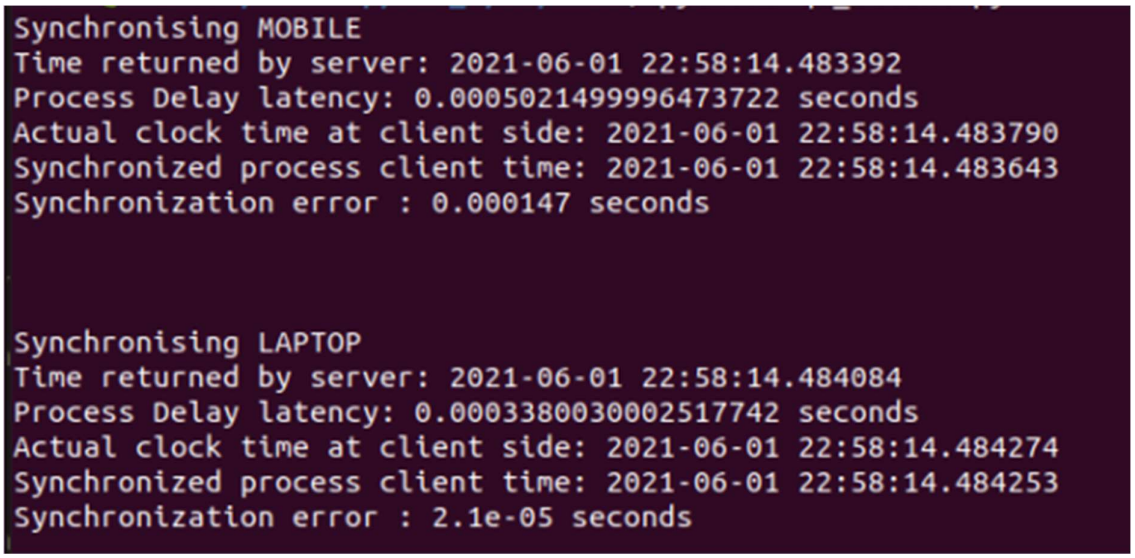
```
import socket
import datetime
import time
def initiateClockServer():
    s = socket.socket()
    print("Socket successfully created")

    port = 8011
    s.bind(('', port))
    s.listen(5)
    print("Socket is listening...")
    while True:
        connection, address = s.accept()
        print('Server connected to', address)
        connection.send(str(datetime.datetime.now()).encode())

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

Clients

```
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("Synchronising " + device_type)
    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")
    print("\n\n")
    s.close()
if __name__ == '__main__':
    synchronizeTime("MOBILE")
    synchronizeTime("LAPTOP")
```




3.

```
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 "NIL"

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()
s.createProcess(1, True)
s.createProcess(2, True)
s.createProcess(3, True)
s.createProcess(4, True)
s.createProcess(5, True)
s.createProcess(6, True)
s.createProcess(7, True)
s.createProcess(8, True)
s.createProcess(9, False)
s.createProcess(10, True)



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

processes = s.getAllProcessess()
global initiator

for process in processes:
    if process.alive == False:
```

```
        print("Process %s is crashed\n" % (process.id))
    if process.crashNoticer:
        initiator = process
        print("Process %s noticed the crash" % (process.id))

print("Process %s is initiating the election\n" % (initiator.id))
def conductElection(id):
    nextAvailable = []
    for i in s.getHigherIds(id):
        if s.getProcess(id).alive:
            message = s.getProcess(i).getMessage(id)
            print('Message from process %d to %d is %s' % (i, id, message))
            if message == "OK":
                nextAvailable.append(i)
            if len(s.getHigherIds(id)) == 2 and
s.getProcess(s.getHigherIds(id)[1]).alive == False:
                print("Process %d is new coordinator\n\n" % i)
                s.getProcess(i).coordinator = True
                quit()
    print("\n")
    if len(nextAvailable) == 0:
        print("Process %d is new coordinator\n\n" % id)
        s.getProcess(id).coordinator = True
        quit()
    smaller = nextAvailable[0]
    conductElection(smaller)
```

conductElection(initiator.id)

Output

```
Process 5 noticed the crash
Process 10 is crashed

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 NIL


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 NIL


Message from process 8 to 7 is OK
Message from process 9 to 7 is OK
Message from process 10 to 7 is NIL


Message from process 9 to 8 is OK
Process 9 is new coordinator
```

```
Process 5 noticed the crash
Process 9 is crashed

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 NIL
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 NIL
Message from process 10 to 6 is OK


Message from process 8 to 7 is OK
Message from process 9 to 7 is NIL
Message from process 10 to 7 is OK


Message from process 9 to 8 is NIL
Message from process 10 to 8 is OK




Process 10 is new coordinator
```

4.

```
import random
class Process:
    def __init__(self, id, alive):
        self.id = id
        self.alive = alive
        self.cordinator = False
        self.crashNoticer = False
class System:
    allProcesses = []

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

    def getAllProcessess(self):
        return self.allProcesses

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

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

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

```
            if process.id == id:
                return process
s = System()
s.createProcess(1, True)
s.createProcess(2, True)
s.createProcess(3, True)
s.createProcess(4, True)
s.createProcess(5, True)
s.createProcess(6, False)
s.createProcess(7, False)
s.createProcess(8, False)
p = s.getProcess(5)
p.crashNoticer = True
processes = s.getAllProcessess()
global initiator
for process in processes:
    if process.alive == False:
        print("process %s is crashed\n" % (process.id))
    if process.crashNoticer:
        initiator = process
        print("process %s noticed the crash" % (process.id))

print("process %s is initiating the election\n" % (initiator.id))
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("Election message is ", electionMessage, "\n")
s.getProcess(electionMessage[-1]).coordinator = True
print("Process %d is the new coordinator\n" % (electionMessage[-1]))
```

```
process 5 noticed the crash
process 8 is crashed

process 5 is initiating the election

Election message is  [5, 6, 7]

Process 7 is the new coordinator
```

```
process 5 noticed the crash
process 6 is crashed

process 7 is crashed

process 8 is crashed

process 5 is initiating the election

Election message is  [5]

Process 5 is the new coordinator
```