Page No.
Date / /

Assignment No. -04

Aim :- Implement Berkeley algorithm for clock synchronization

Objective :- To understand Berkeley algorithm for clock Synchronization and its implementation.

Infrastructure :- Python environment

Software Requirements :- Python 3.0

Theory :-

Berkeley Algorithm :-

Berkeley algorithm is a distributed algorithm for computing the correct time in a network of computers. The algorithm is designed to work in a network where clocks may be running at slightly different rates, and some computers may experience intermittent communication failures.

The basic idea behind Berkeley's algorithm is that each computer in the network periodically sends its local time to a designated "master" computer, which then computes the correct time for the network based on the received timestamps. The master computer then sends the correct time back to all the computers in the network, & each computer sends its clock to the received time.

There are several variations of Berkeley's algorithm that have been proposed, but a common version of the algorithm is as follows :-

Page No.
Date / /

- Each computer starts with its own local time, p
cally sends its time to master computer.
- The master computer receives timestamps from
computers in the network.
- The master computer computes the average time
the timestamps it has received & sends average t
to all the computers in the network.
- Each computer sets its clock to the time it receiv
the master computer.
- The process is repeated periodically, so that over
the clocks of all the computers in the network
coverage to the correct time.

Benefit:- It is relatively simple to implement & und

Limitation:- The time computed by the algorithm is
on the network conditions & time of sending & rece
timestamps which can't be very accurate & also it
requirement of a master computer which if failed
cause the algorithm to stop working.

More advance algorithm such as Network Time
(NTP) which use a more complex algorithm & also
consider the network delay & clock drift to get a
accurate time.

- Scope of Improvement :-

There are several areas where Berkeley's Algorithm
be improved:

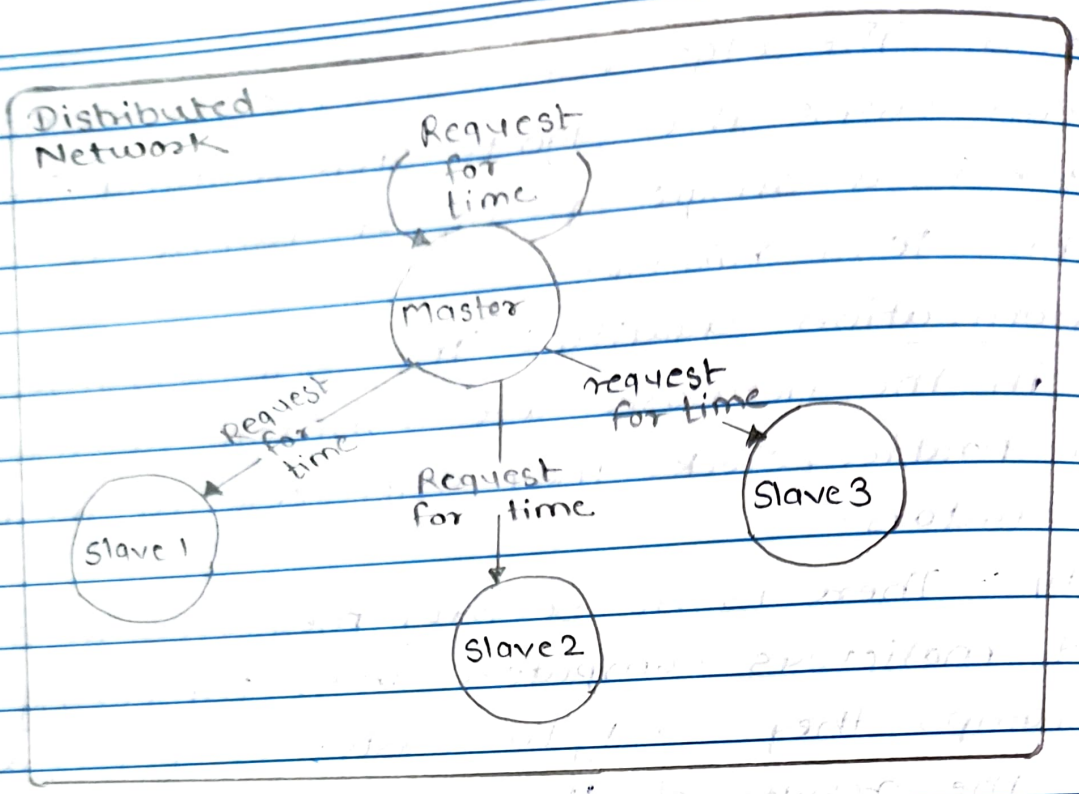
- Accuracy :- The algo. calculates the average time base
timestamps received from all the computers in the
which can lead to inaccuracies especially if the net
has a high degree of jitter or delay.

- **Robustness:-** The algo. requires a master comp, which if it fails, can cause the algo to stop working. If the master computer is a single point of failure, it can make the algorithm less robust.
- **Synchronization Quality:-** The algo. assumes that all the clocks in the network are running at the same rate, but in pratice, clocks may drift due to temp, aging or other factors.
- **Security:-** There is no security measures in the algo. to prevent malicious computers from tampering with the timestamps they send to master computer, which can skew the results of the algorithm.
- **Adaptability:-** The algo. doesn't adapt well to changes in network, for e.g, if a new computer is added to network

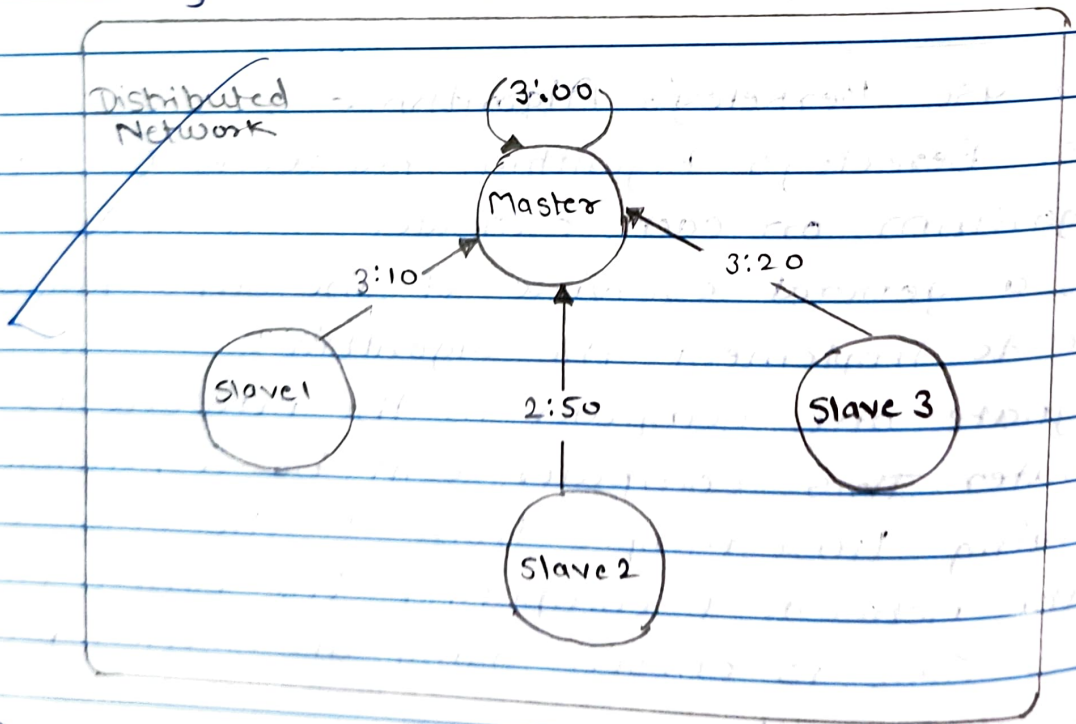
- **How to use Berkeley's Algorithm:-**

To use Berkeley's Algorithm, you would need to implement the algorithm on each computer in a network of computers. Here is a general overview of the steps you would need to take to implement the algorithm :-

1) Designate one computer in the network as the master computer. This computer will be responsible for receiving timestamps from all the other computers in the network & computing the correct time. The master node is chosen using an election process / leader election algorithm.
2) On each computer in the network, set up a timer to periodically send the computer's local time to the master computer.



3) On the master computer, set up a mechanism receiving timestamps from all the computers in

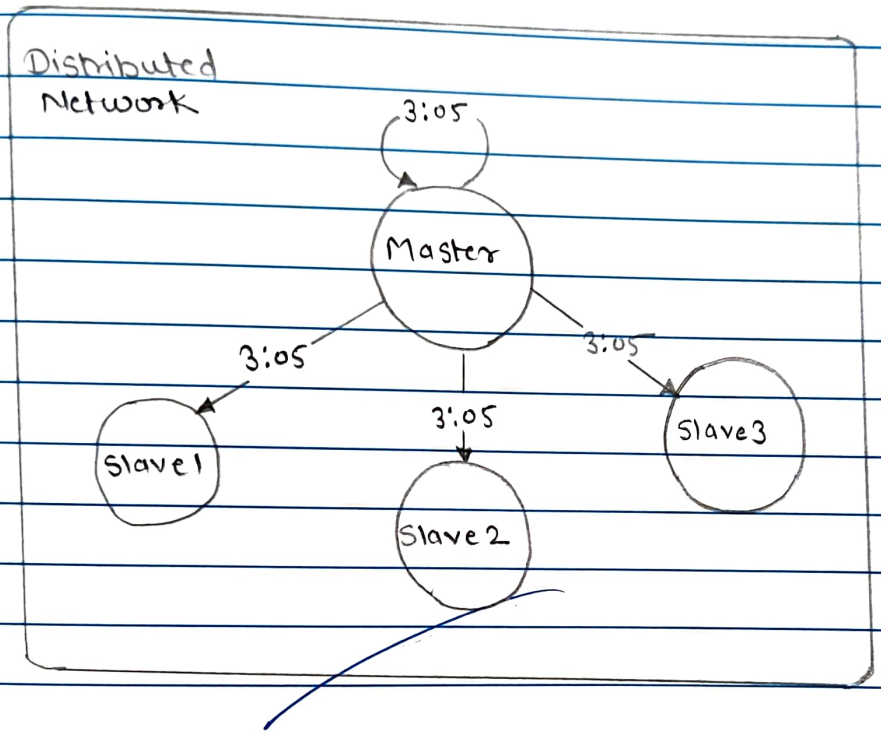


4) On the master computer, implement the logic for ting the average time based on the received ti

5) On the master computer, set up a mechanism f the calculated average time back to all the compu

Page No.
Date / /

6) On each computer in the network, set up a mechanism for receiving the time from the master computer and setting the computer's clock to that time.



7) Repeat the process periodically, for example, every 30 seconds or 1 minute, to ensure that the clocks in the network stay synchronized.

- **Conclusion:-**

We learnt about Berkeley's algorithm for clock synchronization and its implementation.

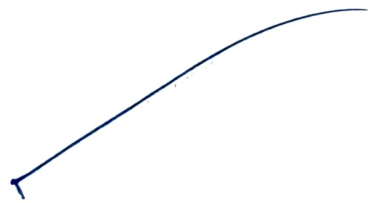
NAME: PALLAVI K. CHOPADE
ROLL NO.: 14
PRN NO.: 72036169K
BE (IT)
LP - V

#Server.py

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

# Client data to be stored in the variable client_data
client_data = {}

# Nested thread function used to receive clock time from a connected client
def startReceivingClockTime(connector, address):
    while True:
        # receive clock time
        clock_time_string = connector.recv(1023).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)

# This opens up the master server to accept clients over a given port
def startConnecting(master_server):
    # Fetching clock time at slaves/clients
    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=startReceivingClockTime,
            args=(master_slave_connector, slave_address,)
        )
        current_thread.start()

# Used to fetch average clock difference
def getAverageClockDiff():
    current_client_data = client_data.copy()
    time_difference_list = list(client['time_difference'] for clientadd, client in
    current_client_data.items())
    sum_of_clock_difference = sum(time_difference_list, datetime.timedelta(0, 0))
    average_clock_difference = sum_of_clock_difference / len(current_client_data)
    return average_clock_difference

# Master sync thread function used to generate cycles of clock synchronization in the
network
def synchronizeAllClocks():
    while True:
        print("New synchronization 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")

    HOST = "127.0.0.1"
    master_server.bind((HOST, port))

    # Starts listening to requests
    master_server.listen(10)
    print("Clock server started...")

    # Start making connections
    print("Starting to make connections...")
    master_thread = threading.Thread(
        target=startConnecting,
        args=(master_server,)
    )
    master_thread.start()

    # Start synchronization
    print("Starting synchronization in parallel...")
    sync_thread = threading.Thread(
        target=synchronizeAllClocks,
        args=()
    )
    sync_thread.start()

# Driver function
if __name__ == '__main__':
    # Trigger the Clock Server
    initiateClockServer(port=8080)

#Client.py

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 the 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 the local computer
    slave_client.connect(('127.0.0.1', port))

    # Start sending time to the server
    print("Starting to receive time from the server\n")
    send_time_thread = threading.Thread(
        target=startSendingTime,
        args=(slave_client,))
    send_time_thread.start()

    # Start receiving synchronized time from the server
    print("Starting to receive synchronized time from the 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)
```

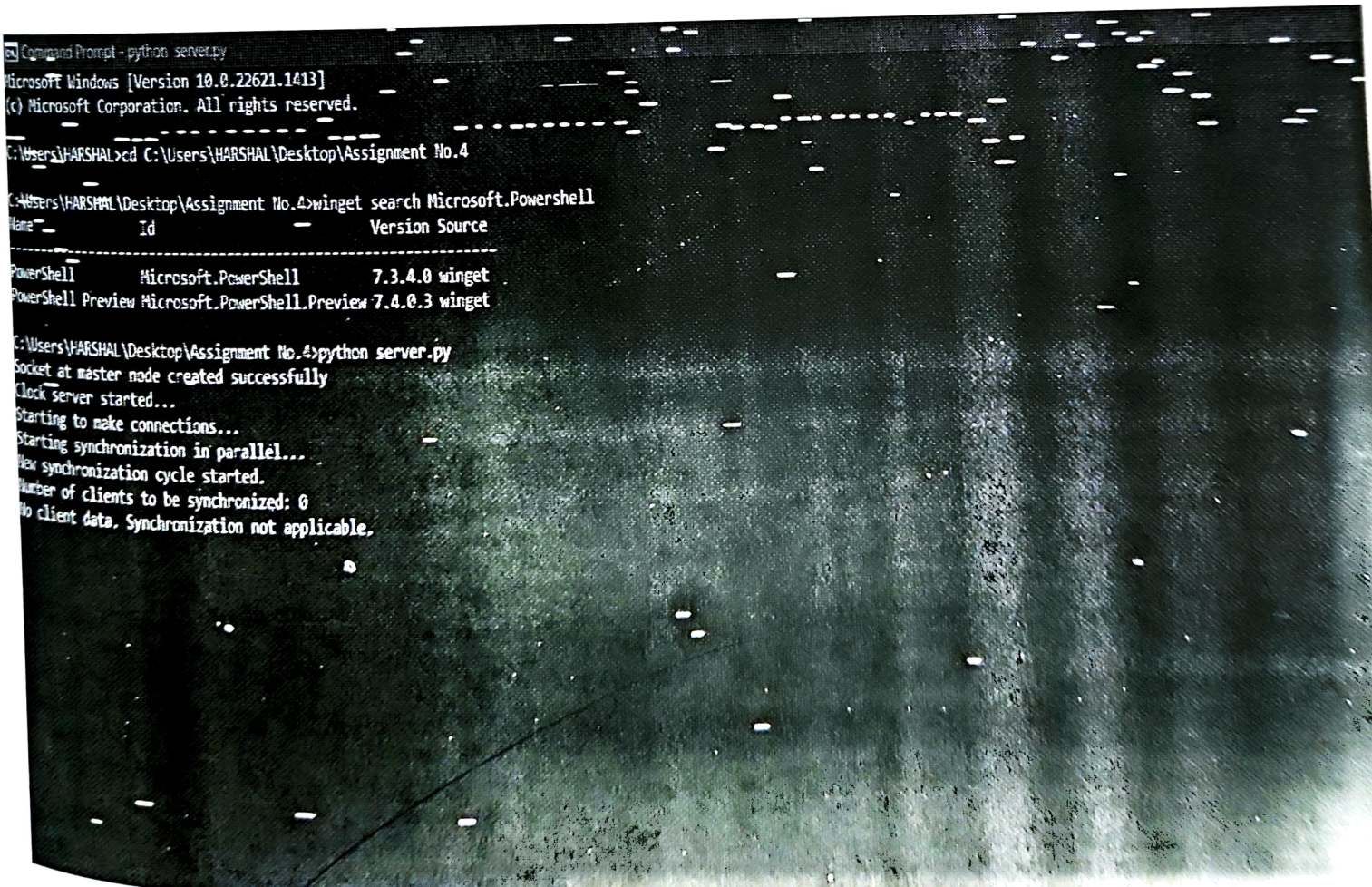


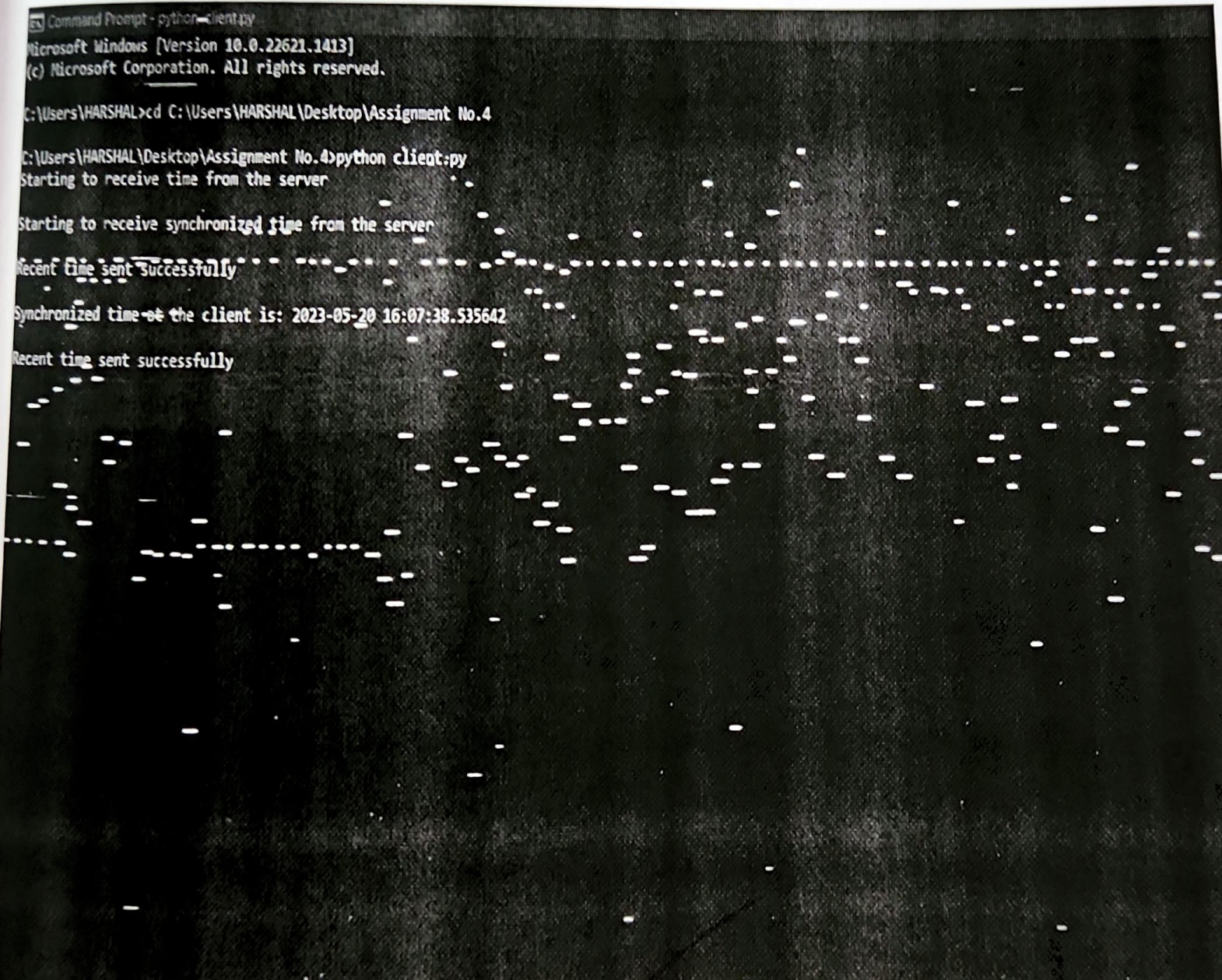
Roll No.: - 14

PRN No.: - 72036169K

Subject: - Distributed Systems

Class: - BE(IT)




Command Prompt - python client.py
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.
C:\Users\HARSHAL>cd C:\Users\HARSHAL\Desktop\Assignment No.4
C:\Users\HARSHAL\Desktop\Assignment No.4>python client.py
Starting to receive time from the server
Starting to receive synchronized time from the server
Recent time sent successfully
Synchronized time at the client is: 2023-05-20 16:07:38.535642
Recent time sent successfully