

Data Communication and Computer Network Laboratory

ASSIGNMENT

MCA 1st year 2nd Sem

Name	Roll-no
Dwip Shekhar Mondal	002210503037

Assignment - I

Questions →

1. Write a TCP Day-Time server program that returns the current time and date. Also write a TCP client program that sends requests to the server to get the current time and date. Choose your own formats for the request/reply messages.
2. Write a TCP Math server program that accepts any valid integer arithmetic expression, evaluates it and returns the value of the expression. Also write a TCP client program that accepts an integer arithmetic expression from the user and sends it to the server to get the result of evaluation. Choose your own formats for the request/reply messages.
3. Implement a UDP server program that returns the permanent address of a student upon receiving a request from a client. Assume that a text file that stores the names of students and their permanent addresses is available local to the server. Choose your own formats for the request/reply messages.

Your report should contain at least the following sections →

1. Problem Statement
2. Your design of request/reply protocol
3. Source code (with appropriate comments)
4. Sample run

Question 1:

Problem statement →

Write a TCP Day-Time server program that returns the current time and date. Also write a TCP client program that sends requests to the server to get the current time and date. Choose your own formats for the request/reply messages.

Design of Request/Reply protocol →

- Client sends a request to the server in the format of the of a serialized JSON object with the following structure

```
request = {  
  method: 'GET | POST | PUT | DELETE',  
  msg: "message"  
}
```

The method field represents the type of the request i.e the method of the request that is sent to the server. (The server is designed to only accept a request with method type GET)

The msg field holds the command string that the client sends the server.

- The server responds to a requesting client by →

- ◆ Deserialize the request sent by the client to get the JSON object sent by the client.
- ◆ If the request method is not GET then send an Error message in the format of serialized JSON object of the following structure →

```
response = {  
  status: 500,  
  data: 'unknown command or command not supported'  
}
```

- ◆ Otherwise send the client the current date and time in the format of serialized JSON object of the following structure →

```
response = {  
  status: 200,  
  data: 'date-time string'  
}
```

- ◆ General Response object structure →

```
response = {  
  status: 200 | 500 (for non-error or error response),  
  data: 'date-time string | Error message'  
}
```

Code →

```
#server.py

from datetime import datetime
import socket
import signal
import sys
import json
import threading

class SokServer(object):
    """
    Socket server class

    reject any request with methods other than GET (for now)

    request struct --> {
        method : GET | POST | PUT | DELETE
        msg: "message"
    }

    response struct --> {
        status: NUMBER
        data: response data OR error msg
    }

    """

    def __init__(self, port=9999):
        self.host = socket.gethostname().split('.')[0]
        self.port = port
        print("host: ", self.host, "port: ", self.port)

    def start_server(self):
        """
        Start socket server
        """

        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        try:

            print("setting up server at {host}:{port}".format(host=self.host,
port=self.port))
            self.socket.bind((self.host, self.port))
            print("server started at port: {port}".format(port=self.port))
```

```
except Exception as e:
    print("Error setting up server")
    self.stop_server()
    sys.exit(1)

# start listening for incoming connection
self.listen()

def listen(self):
    """
    listen for incoming connection/request to the server
    """

    self.socket.listen(5)

    while True:

        clientSok, addr = self.socket.accept()
        # clientSok.settimeout(30)
        print(f"Received connection from address: {addr}")

        ## TODO: handle concurrent connection(Threads)

        threading.Thread(target=self.handle_request, args=(clientSok, addr)).start()
        # self.handle_request(clientSok, addr)

def handle_request(self, client, address):
    """
    handle incoming request and send appropriate response
    """
    PACKET = 1024

    try:
        req = client.recv(PACKET)
        decoded_data = json.loads(req.decode())
        print("received data: ", decoded_data)

        # Reject any request other than req with GET method

        if decoded_data["method"] != "GET":
            raise Exception("Unknown request method or method not supported")

        if decoded_data['msg'] != "Get time":
            raise Exception("Unknown request command or command not supported")
```

```
        response = {
            "status" : 200,
            "data": str(datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
        }

        json_response = json.dumps(response)
        client.sendall(json_response.encode())
        client.close()

    except Exception as e:
        print(e)
        err_res = {
            "status" : 500,
            "data": str(e)
        }

        client.sendall(json.dumps(err_res).encode())
        client.close()

def stop_server(self):
    """
    Shutdown server
    """
    try:
        print("Shutting down server ... ")
        self.socket.shutdown(socket.SHUT_RDWR)
        self.socket.close()
        sys.exit(1)

    except Exception as e:
        pass

def shutdownServer(sig, unused):
    """
    Shutdown server from a SIGINT received signal
    """
    server.stop_server()
    sys.exit(1)

signal.signal(signal.SIGINT, shutdownServer)
server = SokServer(7000)
server.start_server()
```

```
#Client.py

import socket
import json

HOST = socket.gethostname().split('.')[0]
PORT = 7000
BUFF_SIZE = 1024

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as skt:

    skt.connect((HOST, PORT))
    req = {}
    req["method"] = "GET"

    print(f"Client connected on {HOST}:{PORT}")

    msg = input("Enter command: ")

    # msg = "What's the current date ?>"

    req["msg"] = msg

    skt.sendall(json.dumps(req).encode())

    print("Client request: ", req)

    data = skt.recv(BUFF_SIZE)

    res = json.loads(data.decode())

    print("response from server: ", res)

    print(f"Server --> status: {res['status']}, data: {res['data']}")

    skt.close()
```

Sample run →

```

_server.py
host: Deep post: 7000
setting up server at Deep:7000
server started at port: 7000
Received connection from address: ('192.168.0.200', 61975)
Received connection from address: ('192.168.0.200', 61976)
received data: {'method': 'GET', 'msg': 'Get time'}
received data: {'method': 'GET', 'msg': 'Get food'}
Unknown request command or command not supported

_client.py
Client connected on Deep:7000
Enter command: Get time
Client request: {'method': 'GET', 'msg': 'Get time'}
response from server: {'status': 200, 'data': '2023-08-17 21:56:10'}
Server --> status: 200, data: 2023-08-17 21:56:10
PS C:\Users\monda\OneDrive\Desktop\net_lab_updated\set_1_sockets\q1>

_client.py
Client connected on Deep:7000
Enter command: Get food
Client request: {'method': 'GET', 'msg': 'Get food'}
response from server: {'status': 500, 'data': 'Unknown request command or command not supported'}
Server --> status: 500, data: Unknown request command or command not supported
PS C:\Users\monda\OneDrive\Desktop\net_lab_updated\set_1_sockets\q1>

```

Question 2:

Problem statement →

Write a TCP Math server program that accepts any valid integer arithmetic expression, evaluates it and returns the value of the expression. Also write a TCP client program that accepts an integer arithmetic expression from the user and sends it to the server to get the result of evaluation. Choose your own formats for the request/reply messages.

Design of Request/Reply protocol →

- Client sends a request to the server in the format of the of a serialized JSON object with the following structure

```

request = {
    method: 'GET | POST | PUT | DELETE',
    msg: "math expression string"
}

```

The method field represents the type of the request i.e the method of the request that is sent to the server. (The server is designed to only accept a request with method type GET)

The msg field holds the mat-expression to be evaluated by the server.

→ The server responds to a requesting client by →

- ◆ Deserialize the request sent by the client to get the JSON object sent by the client.
- ◆ If the request method is not GET then send an Error message in the format of serialized JSON object of the following structure →

```
response = {  
    status: 500,  
    data: 'Error message'  
}
```

- ◆ Otherwise evaluate the math expression and send the client, the result of the operation in the format of serialized JSON object of the following structure →

```
response = {  
    status: 200,  
    data: 'evaluated math expression'  
}
```

- ◆ General Response object structure →

```
response = {  
    status: 200 | 500 (for non-error or error response),  
    data: 'evaluated math expression | Error message'  
}
```

Code →

```
#server  
  
from datetime import datetime  
import socket  
import signal  
import sys  
import json  
import threading  
  
class SokServer(object):  
    """  
  
    Socket server class
```

```
reject any request with methods other than GET (for now)

request struct --> {
    method : GET | POST | PUT | DELETE
    msg: "math expression"
}

response struct --> {
    status: NUMBER
    data: response (math expression) data OR error msg
}

"""
def __init__(self, port=9999):
    self.host = socket.gethostname().split('.')[0]
    self.port = port
    print("host: ", self.host, "port: ", self.port)

def start_server(self):
    """
    Start socket server
    """
    self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    try:
        print("setting up server at {host}:{port}".format(host=self.host,
port=self.port))
        self.socket.bind((self.host, self.port))
        print("server started at port: {port}".format(port=self.port))

    except Exception as e:
        print("Error setting up server")
        self.stop_server()
        sys.exit(1)

    # start listening for incoming connection

    self.listen()

def listen(self):
    """
    listen for incoming connection/request to the server
    """
    self.socket.listen(5)

    while True:

        clientSok, addr = self.socket.accept()
```

```
clientSok.settimeout(30)
print(f"Received connection from address: {addr}")

## TODO: handle concurrent connection(Threads)

threading.Thread(target=self.handle_request, args=(clientSok, addr)).start()
# self.handle_request(clientSok, addr)

def handle_request(self, client, address):
    """
        handle incoming request and send appropriate response
    """

    PACKET = 1024

    try:
        req = client.recv(PACKET)
        decoded_data = json.loads(req.decode())
        print("received data: ", decoded_data)

        # Reject any request other than req with GET method

        if decoded_data["method"] != "GET":
            raise Exception("Unknown request method or method not supported")

        print("client >", decoded_data['msg'])

        res = eval(decoded_data['msg'])

        response = {
            "status" : 200,
            "data": res
        }

        json_response = json.dumps(response)
        client.sendall(json_response.encode())

    except Exception as e:
        print(e)
        err_res = {
            "status" : 500,
            "data": str(e)
        }
        client.sendall(json.dumps(err_res).encode())

def stop_server(self):
    """
        Shutdown server
    """
```

```
    """
    try:
        print("Shutting down server ... ")
        self.socket.shutdown(socket.SHUT_RDWR)
        self.socket.close()
        sys.exit(1)
    except Exception as e:
        pass

def shutdownServer(sig, unused):
    """
    Shutdown server from a SIGINT received signal
    """
    server.stop_server()
    sys.exit(1)

signal.signal(signal.SIGINT, shutdownServer)
server = SokServer(7000)
server.start_server()
```

```
#client

import socket
import json

HOST = socket.gethostname().split('.')[0]
PORT = 7000
BUFF_SIZE = 1024

def validate_first_brackets(expression):
    stack = []

    for char in expression:
        if char in ['[', '{']:
            return False
        elif char == '(':
            stack.append('(')
        elif char == ')':
            if len(stack) == 0 or stack[-1] != '(':
                return False
            stack.pop()

    return len(stack) == 0

def validate_balanced_brackets(expression):
    stack = []
```

```
for char in expression:
    if char == '(':
        stack.append('(')
    elif char == ')':
        if not stack or stack[-1] != '(':
            return False
        stack.pop()

return len(stack) == 0

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as skt:
    """
    request struct = {
        method: GET
        msg: math expression string
    }

    response struct --> {
        status: NUMBER
        data: response (math expression) data OR error msg
    }
    """
    try:
        skt.connect((HOST, PORT))
        req = {}
        req["method"] = "GET"

        print(f"Client connected on {HOST}:{PORT}")
        msg = input("Enter expression: ")

        if not validate_first_brackets(msg):
            raise Exception(f"Please use only first brackets in the expression")

        if not validate_balanced_brackets(msg):
            raise Exception(f"Brackets are not balanced in the expression")
        try:
            res = eval(msg)
            print(res)
        except Exception as e:
            raise Exception("Invalid math Expression")

        req["msg"] = msg
        skt.sendall(json.dumps(req).encode())

        print("Client request: ", req)
```

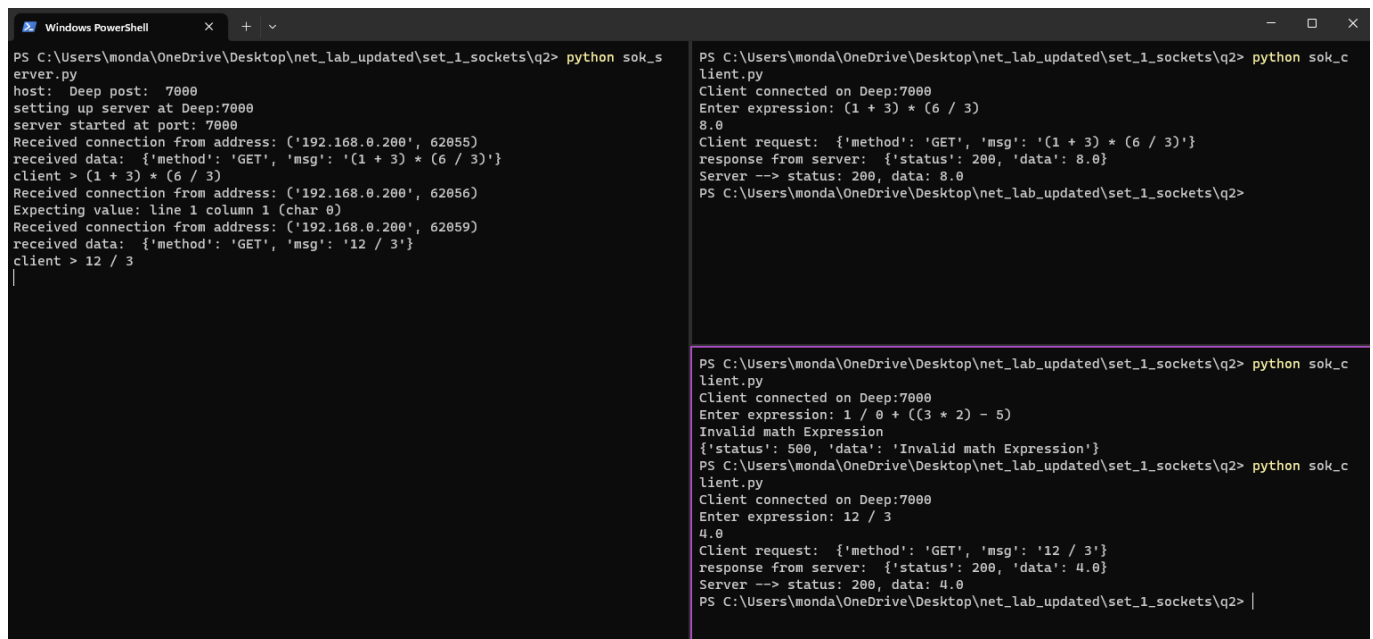
```
data = skt.recv(BUFF_SIZE)
res = json.loads(data.decode())

print("response from server: ", res)

print(f"Server --> status: {res['status']], data: {res['data']}")

skt.close()
except Exception as e:
    print(e)
    err_res = {
        "status" : 500,
        "data": str(e)
    }
    print(err_res)
    skt.close()
```

Sample run →



```
PS C:\Users\monda\OneDrive\Desktop\net_lab_updated\set_1_sockets\q2> python sok_s
server.py
host: Deep post: 7000
setting up server at Deep:7000
server started at port: 7000
Received connection from address: ('192.168.0.200', 62055)
received data: {'method': 'GET', 'msg': '(1 + 3) * (6 / 3)'}
client > (1 + 3) * (6 / 3)
Received connection from address: ('192.168.0.200', 62056)
Expecting value: line 1 column 1 (char 0)
Received connection from address: ('192.168.0.200', 62059)
received data: {'method': 'GET', 'msg': '12 / 3'}
client > 12 / 3
|

PS C:\Users\monda\OneDrive\Desktop\net_lab_updated\set_1_sockets\q2> python sok_c
lient.py
Client connected on Deep:7000
Enter expression: (1 + 3) * (6 / 3)
8.0
Client request: {'method': 'GET', 'msg': '(1 + 3) * (6 / 3)'}
response from server: {'status': 200, 'data': 8.0}
Server --> status: 200, data: 8.0
PS C:\Users\monda\OneDrive\Desktop\net_lab_updated\set_1_sockets\q2>

PS C:\Users\monda\OneDrive\Desktop\net_lab_updated\set_1_sockets\q2> python sok_c
lient.py
Client connected on Deep:7000
Enter expression: 1 / 0 + ((3 * 2) - 5)
Invalid math Expression
{'status': 500, 'data': 'Invalid math Expression'}
PS C:\Users\monda\OneDrive\Desktop\net_lab_updated\set_1_sockets\q2> python sok_c
lient.py
Client connected on Deep:7000
Enter expression: 12 / 3
4.0
Client request: {'method': 'GET', 'msg': '12 / 3'}
response from server: {'status': 200, 'data': 4.0}
Server --> status: 200, data: 4.0
PS C:\Users\monda\OneDrive\Desktop\net_lab_updated\set_1_sockets\q2> |
```

Question 3:

Problem statement →

Implement a UDP server program that returns the permanent address of a student upon receiving a request from a client. Assume that a text file that stores the names of students and their permanent addresses is available local to the server. Choose your own formats for the request/reply messages.

Design of Request/Reply protocol →

- The UDP server has a database of the students in a csv file format with student info such that each entry contains the student name and the student address.
- The UDP client sends the server with a student's name as a message.
- In response, the server,
 - ◆ If finds the student name in the records of the database →
 - It sends the client the student info as a utf-8 encoded string with the following format: `'Name: {student name} Address: {student-address}'`
 - ◆ If does not find the name in the database records →
 - It sends the client generic error message as a utf-8 encoded string with the following format: `'student not found'`

Code →

```
#server

import socket
import threading
import csv

def load_student_data(file_path):
    student_data = {}
    with open(file_path, 'r') as file:
        csvreader = csv.reader(file)
        for row in csvreader:
```

```
        name, address = row
        student_data[name] = address
    return student_data

def find_student_info(file_path, student_name):
    res = "student not found"
    with open(file_path, 'r') as file:
        csvreader = csv.reader(file)
        for row in csvreader:
            name, address = row
            if name == student_name:
                res = f"Name: {name}, Address: {address}"
                break
            # student_data[name] = address
    return res

def handle_UDP_requests(client_address, server_socket, data, file_path):
    student_name = data.decode()
    student_data = find_student_info(file_path, student_name)
    print("client >", student_name)
    reply = student_data.encode()
    server_socket.sendto(reply, client_address)

def main():
    host = '127.0.0.1'
    port = 9999
    file_path = './data.csv'

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_socket.bind((host, port))
    print("UDP server listening on", host, "port", port)

    while True:
        data, address = server_socket.recvfrom(1024)
        client_handler = threading.Thread(target=handle_UDP_requests, args=(address,
server_socket, data, file_path))
        client_handler.start()

if __name__ == "__main__":
    main()
```

```
#client
import socket

def main():
    host = '127.0.0.1'
```



```
port = 9999

client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

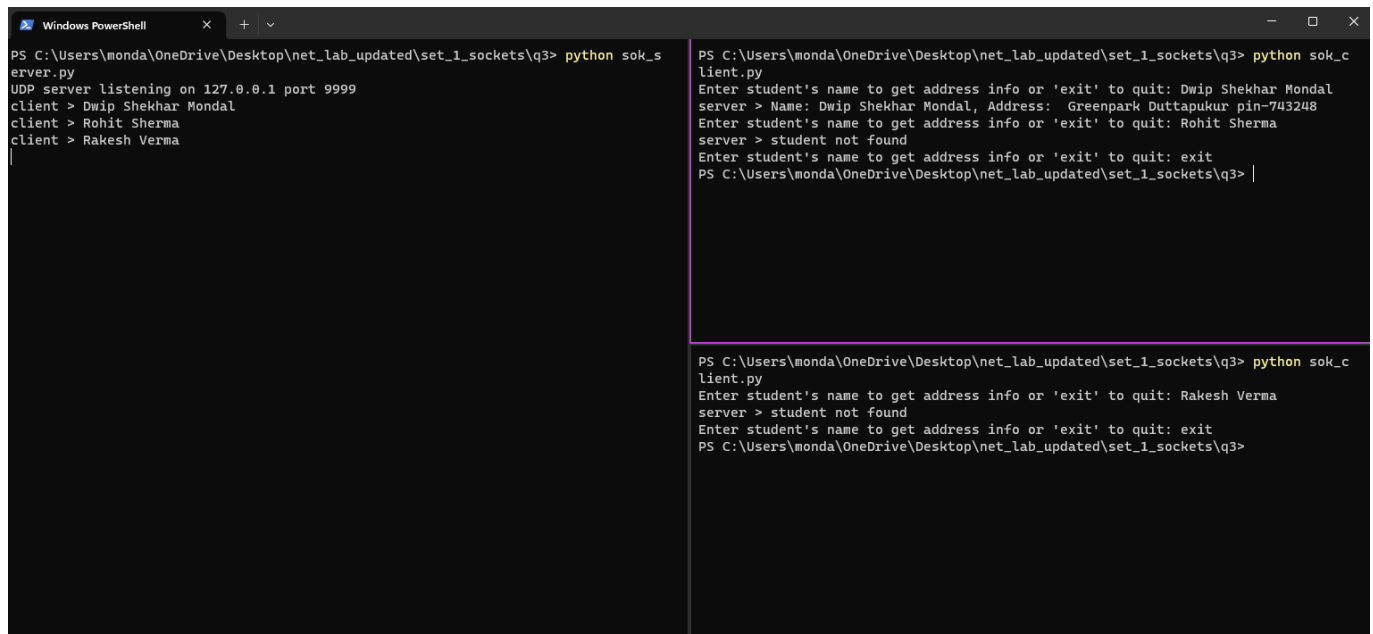
while True:
    student_name = input("Enter student's name to get address info or 'exit' to quit:")
    if student_name.lower() == 'exit':
        break

    client_socket.sendto(student_name.encode(), (host, port))
    data, _ = client_socket.recvfrom(1024)
    print("server >", data.decode())

client_socket.close()

if __name__ == "__main__":
    main()
```

Sample run →



```
PS C:\Users\monda\OneDrive\Desktop\net_lab_updated\set_1_sockets\q3> python sok_server.py
UDP server listening on 127.0.0.1 port 9999
client > Dwip Shekhar Mondal
client > Rohit Sherma
client > Rakesh Verma
|

PS C:\Users\monda\OneDrive\Desktop\net_lab_updated\set_1_sockets\q3> python sok_client.py
Enter student's name to get address info or 'exit' to quit: Dwip Shekhar Mondal
server > Name: Dwip Shekhar Mondal, Address: Greenpark Duttapukur pin-743248
Enter student's name to get address info or 'exit' to quit: Rohit Sherma
server > student not found
Enter student's name to get address info or 'exit' to quit: exit
PS C:\Users\monda\OneDrive\Desktop\net_lab_updated\set_1_sockets\q3> |

PS C:\Users\monda\OneDrive\Desktop\net_lab_updated\set_1_sockets\q3> python sok_client.py
Enter student's name to get address info or 'exit' to quit: Rakesh Verma
server > student not found
Enter student's name to get address info or 'exit' to quit: exit
PS C:\Users\monda\OneDrive\Desktop\net_lab_updated\set_1_sockets\q3>
```