Network Lab
Assignment 1

Problem Statements

---

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.

   Answer :

   Design of request/reply protocol

   The TCP client is required to send a valid command (e.g., 'What is the time?') and takes input within a loop. The TCP server is expected to receive a valid command, acknowledge it with an appropriate response, and handle invalid commands accordingly(By sending 'Invalid Command' message). Finally the response is printed into the console. The Server Socket and the Client Socket are never closed to enforce uninterrupted connection between them.

   Source code (with appropriate comments)

1_client.py

```
import socket

# Define the TCP port and IP address to connect to
TCP_PORT = 5000
IP_ADDR = '127.0.0.1'

# Define the buffer size for receiving data
BUF_SIZE = 1024

while True:
    # Create a new client socket using IPv4 and TCP
    Client_Socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Connect the client socket to the server
    Client_Socket.connect((IP_ADDR, TCP_PORT))

    # Prompt the user to enter a command
    Command = input('Enter a command: ')

    # Send the entered command to the server after encoding it
    Client_Socket.send(Command.encode())
```

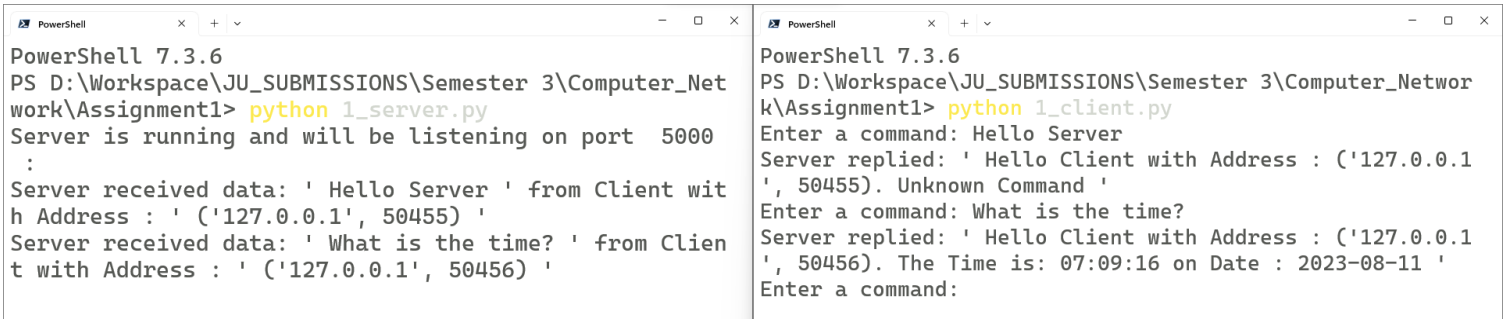
```
    # Receive the server's reply and decode it
    Reply = Client_Socket.recv(BUF_SIZE).decode()

    # Print the server's reply
    print("Server replied: '", Reply, "'")

# Note: This loop runs indefinitely, allowing the client to repeatedly
# connect to the server, send a command, receive a reply, and display it.
# To stop the client, you may need to manually terminate the program.
```

1_server.py

```
import socket
import datetime

# Define the TCP port and IP address to bind the server socket
TCP_PORT = 5000
IP_ADDR = '127.0.0.1'

# Define the buffer size for receiving data
BUF_SIZE = 1024

# Response prefix
Reset = 'Hello Client with Address : '

# Create a server socket using IPv4 and TCP
Server_Socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the server socket to the specified IP address and port
Server_Socket.bind((IP_ADDR, TCP_PORT))

# Print server information
print("Server is running and will be listening on port ", TCP_PORT, " : ")

while True:
    # Start listening for incoming connections
    Server_Socket.listen(1)
```

```
    # Accept an incoming connection from a client
    Client_Socket, Client_Address = Server_Socket.accept()

    # Receive data from the client and decode it
    Incoming_Data = Client_Socket.recv(BUF_SIZE).decode()

    # Print received data and client address
     print("Server received data: '", Incoming_Data, "' from Client with
Address : '", Client_Address, "'")

    # Check the received command and generate a reply
    if Incoming_Data == 'What is the time?':
        # Get the current date and time
        Current = str(datetime.datetime.now())
        time = Current.split(' ')[1].split('.')[0]
        date = Current.split(' ')[0]
        # Create a reply with the current time and date
        Reply = Reset + str(Client_Address) + '. The Time is: ' + time + '
on Date : ' + date
        # Send the reply back to the client after encoding
        Client_Socket.send(Reply.encode())
    else:
            # If the command is not recognized, send an "Unknown Command"
reply
        Reply = Reset + str(Client_Address) + '. Unknown Command'
        Client_Socket.send(Reply.encode())
```

Sample Run



---

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.

   Answer :

   Design of request/reply protocol

   The TCP client is required to send a valid infix expression (which is more appropriate for humans to send) (e.g., '1+2+3-10/1') and takes input within a loop. The TCP server is expected to receive a valid infix expression, acknowledge it with a calculated reply, and handle invalid expressions accordingly (By sending an 'Invalid Expression' message). Finally the response is printed into the console. The Server Socket and the Client Socket are never closed to enforce uninterrupted connection between them.
   Furthermore, I have created an additional python package named 'Additional Functions' in which I have created a function that tokenizes the operands and operators for further conversions. And then I have created functions for converting those infix expressions to postfix expressions. Since a Computer is more reliable in solving a Postfix expression rather than an infix expression. Then I have created a function that evaluates the converted Postfix expression and gives the correct output.
   A try-except block is added to capture any error while the process of conversions, If any error is captured server sends a Invalid Expression message to client.

   Source code (with appropriate comments)

Additional_Functions.py

```
# Function to convert an arithmetic expression to postfix notation
def Arithmatic_Expression_to_PostFix(arg):
    Stack = []  # Initialize a stack for operators
    PostFix = []  # Initialize a list to store postfix notation
     Precedence = lambda x: 1 if x in ['+', '-'] else 2 if x in ['*', '/']
else 0  # Define operator precedence
    for element in arg:
        if element in ['+', '-', '*', '/']:
            while Stack and Precedence(Stack[-1]) >= Precedence(element):
                PostFix.append(Stack.pop())
            Stack.append(element)
        else:
            PostFix.append(element)
    while Stack:
        PostFix.append(Stack.pop())
    return PostFix
```

```python
# Function to evaluate a postfix expression
def PostFix_Expression_Evaluation(PostFix):
    Stack = []  # Initialize a stack for operands
    for char in PostFix:
        if char in ['+', '-', '*', '/']:
            Operand1 = Stack.pop()
            Operand2 = Stack.pop()
            if char == '+':
                Stack.append(Operand2 + Operand1)
            elif char == '-':
                Stack.append(Operand2 - Operand1)
            elif char == '*':
                Stack.append(Operand2 * Operand1)
            elif char == '/':
                Stack.append(Operand2 / Operand1)
        else:
            Stack.append(int(char))
    return Stack.pop()

# Function to tokenize an expression into operands and operators
def Tokenizing_Operands(arg):
    temp = ''  # Initialize a temporary string for storing operands
    res = []   # Initialize a list for tokenized result
    for char in arg:
        if char in ['+', '-', '*', '/'] and temp == '':
            res.append(char)  # Append operator directly if no operand yet
        elif char in ['+', '-', '*', '/'] and temp != '':
            res.append(temp)  # Append operand
            res.append(char)  # Append operator
            temp = ''  # Reset temporary operand string
        else:
            temp += char  # Continue building the operand string
    res.append(temp)  # Append the last operand
    return res
```

2_client.py

```python
import socket

# Define the TCP port and IP address to connect to
TCP_PORT = 5000
IP_ADDR = '127.0.0.1'

# Define the buffer size for receiving data
BUF_SIZE = 1024

while True:
    # Create a new client socket using IPv4 and TCP
    Client_Socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Connect the client socket to the server
    Client_Socket.connect((IP_ADDR, TCP_PORT))

    # Prompt the user to enter an expression
    Command = input('Enter an Expression: ')

    # Send the entered expression to the server after encoding it
    Client_Socket.send(Command.encode())

    # Receive the server's reply and decode it
    Reply = Client_Socket.recv(BUF_SIZE).decode()

    # Print the server's reply
    print("Server replied: '", Reply, "'")

    # Note: This loop runs indefinitely, allowing the client to repeatedly
     # connect to the server, send an expression, receive a reply, and
display it.
    # To stop the client, you may need to manually terminate the program.
```

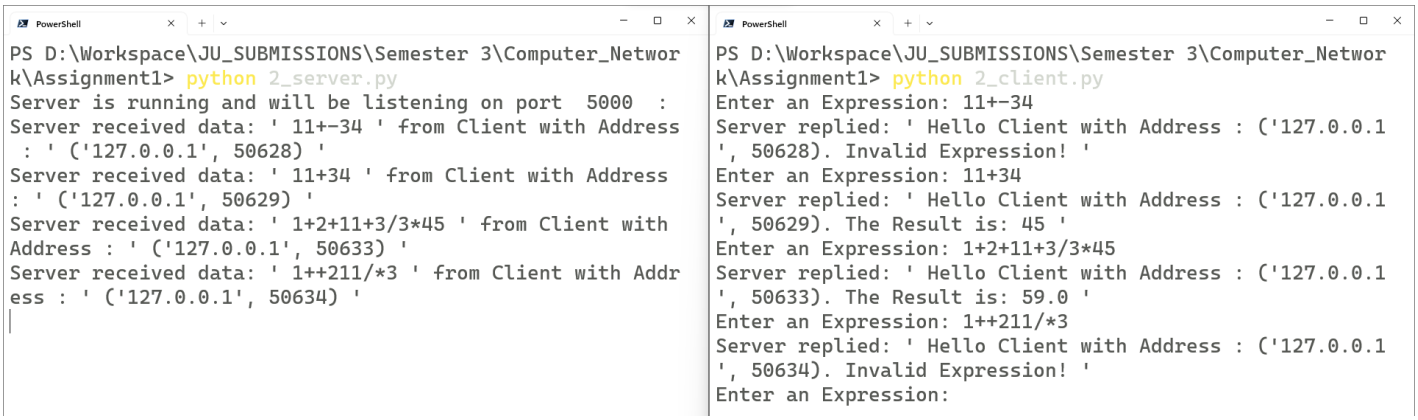
2_server.py

```python
import socket
import Additional_Functions as AF    # Assuming this module contains
necessary functions
```

```
# Define the TCP port and IP address to bind the server socket
TCP_PORT = 5000
IP_ADDR = '127.0.0.1'

# Define the buffer size for receiving data
BUF_SIZE = 1024

# Response prefix
Reset = 'Hello Client with Address : '

# Create a server socket using IPv4 and TCP
Server_Socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the server socket to the specified IP address and port
Server_Socket.bind((IP_ADDR, TCP_PORT))

# Print server information
print("Server is running and will be listening on port ", TCP_PORT, " : ")

while True:
    # Start listening for incoming connections
    Server_Socket.listen(1)

    # Accept an incoming connection from a client
    Client_Socket, Client_Address = Server_Socket.accept()

    # Receive data from the client and decode it
    Incoming_Data = Client_Socket.recv(BUF_SIZE).decode()

    # Print received data and client address
     print("Server received data: '", Incoming_Data, "' from Client with
Address : '", Client_Address, "'")

    try:
            # Attempt to tokenize and convert the incoming arithmetic
expression to postfix
        Incoming_Data = AF.Tokenizing_Operands(Incoming_Data)
        Incoming_Data = AF.Arithmetic_Expression_to_PostFix(Incoming_Data)
    except:
```

```
            # Handle any exceptions (e.g., invalid expression)
            Incoming_Data = 'Invalid Expression'

        if Incoming_Data != 'Invalid Expression':
            # Evaluate the postfix expression and generate a reply
            Result = AF.PostFix_Expression_Evaluation(Incoming_Data)
                Reply = Reset + str(Client_Address) + '. The Result is: ' +
str(Result)
            # Send the reply back to the client after encoding
            Client_Socket.send(Reply.encode())
        else:
            # Send an "Invalid Expression" reply if necessary
            Reply = Reset + str(Client_Address) + '. Invalid Expression!'
            Client_Socket.send(Reply.encode())
```

Sample Run



```
PS D:\Workspace\JU_SUBMISSIONS\Semester 3\Computer_Networ
k\Assignment1> python 2_server.py
Server is running and will be listening on port  5000  :
Server received data: ' 11+-34 ' from Client with Address
 : ' ('127.0.0.1', 50628) '
Server received data: ' 11+34 ' from Client with Address
: ' ('127.0.0.1', 50629) '
Server received data: ' 1+2+11+3/3*45 ' from Client with
Address : ' ('127.0.0.1', 50633) '
Server received data: ' 1++211/*3 ' from Client with Addr
ess : ' ('127.0.0.1', 50634) '
```

```
PS D:\Workspace\JU_SUBMISSIONS\Semester 3\Computer_Networ
k\Assignment1> python 2_client.py
Enter an Expression: 11+-34
Server replied: ' Hello Client with Address : ('127.0.0.1
', 50628). Invalid Expression! '
Enter an Expression: 11+34
Server replied: ' Hello Client with Address : ('127.0.0.1
', 50629). The Result is: 45 '
Enter an Expression: 1+2+11+3/3*45
Server replied: ' Hello Client with Address : ('127.0.0.1
', 50633). The Result is: 59.0 '
Enter an Expression: 1++211/*3
Server replied: ' Hello Client with Address : ('127.0.0.1
', 50634). Invalid Expression! '
Enter an Expression:
```

---

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 locally to the server. Choose your own formats for the request/reply messages.

   Answer :

   Design of request/reply protocol

   The UDP client is required to send a valid name (which is present into the text file) and takes input within a loop. The UDP server is expected to receive a valid name (which is present in the text file), acknowledge it with an appropriate Address, and handle invalid commands accordingly(By sending an 'Invalid Name' message). Finally the response is printed into the console. The Server Socket and the Client Socket are never closed to enforce uninterrupted connection between them.
   Furthermore, I have created an additional python package named 'Additional Functions' in which I have created a function that Reads the file and returns a dictionary that contains the addresses of the students corresponding to their name. Key == Name, Value == Address
   A try-except block is added to capture any error while the process of reading, If any error is captured server sends a Invalid Name message to client.

Source code (with appropriate comments)

Additional_Functions.py

```
# Function to convert an arithmetic expression to postfix notation
def Arithmatic_Expression_to_PostFix(arg):
    Stack = []  # Initialize a stack for operators
    PostFix = []  # Initialize a list to store postfix notation
     Precedence = lambda x: 1 if x in ['+', '-'] else 2 if x in ['*', '/']
else 0  # Define operator precedence
    for element in arg:
        if element in ['+', '-', '*', '/']:
            while Stack and Precedence(Stack[-1]) >= Precedence(element):
                PostFix.append(Stack.pop())
            Stack.append(element)
        else:
            PostFix.append(element)
    while Stack:
        PostFix.append(Stack.pop())
    return PostFix

# Function to evaluate a postfix expression
def PostFix_Expression_Evaluation(PostFix):
```

```
    Stack = []  # Initialize a stack for operands
    for char in PostFix:
        if char in ['+', '-', '*', '/']:
            Operand1 = Stack.pop()
            Operand2 = Stack.pop()
            if char == '+':
                Stack.append(Operand2 + Operand1)
            elif char == '-':
                Stack.append(Operand2 - Operand1)
            elif char == '*':
                Stack.append(Operand2 * Operand1)
            elif char == '/':
                Stack.append(Operand2 / Operand1)
        else:
            Stack.append(int(char))
    return Stack.pop()

# Function to tokenize an expression into operands and operators
def Tokenizing_Operands(arg):
    temp = ''  # Initialize a temporary string for storing operands
    res = []   # Initialize a list for tokenized result
    for char in arg:
        if char in ['+', '-', '*', '/'] and temp == '':
            res.append(char)  # Append operator directly if no operand yet
        elif char in ['+', '-', '*', '/'] and temp != '':
            res.append(temp)  # Append operand
            res.append(char)  # Append operator
            temp = ''  # Reset temporary operand string
        else:
            temp += char  # Continue building the operand string
    res.append(temp)  # Append the last operand
    return res

# Function to read data from a file and create a dictionary
def Read_File():
    Collective_Info = {}  # Initialize an empty dictionary
    name = ''  # Initialize a variable to store the name
    with open('./Files/input.txt', 'r') as file:
        data = file.read()  # Read data from the file
    for line in data.split('\n'):
```

```
            name = line.split(' ')[0] + ' ' + line.split(' ')[1]  # Extract
the name from the line
            Collective_Info[name] = ' '.join(line.split(' ')[2:])  # Store the
associated info in the dictionary
    return Collective_Info
```

3_client.py

```
import socket

# Define local IP address and port to communicate with the server
LOCAL_IP = '127.0.0.1'
LOCAL_PORT = 20001

# Define the buffer size for receiving data
BUF_SIZE = 1024

# Create a tuple representing the server socket address
Server_Socket_Address = (LOCAL_IP, LOCAL_PORT)

while True:
    # Create a new client socket using UDP
                Client_Socket    =    socket.socket(family=socket.AF_INET,
type=socket.SOCK_DGRAM)

    # Prompt the user to enter a name
    MyName = input("Enter Name : ")

    # Encode the name to bytes and send it to the server socket address
    MyName = MyName.encode()
    Client_Socket.sendto(MyName, Server_Socket_Address)

    # Receive incoming data from the server
    Incoming_Packet = Client_Socket.recvfrom(BUF_SIZE)
    Incoming_Data = Incoming_Packet[0].decode()

    # Print the server's sent address
    print("Server sent Address: '", Incoming_Data, "'")

# Note: This loop runs indefinitely, allowing the client to repeatedly
```

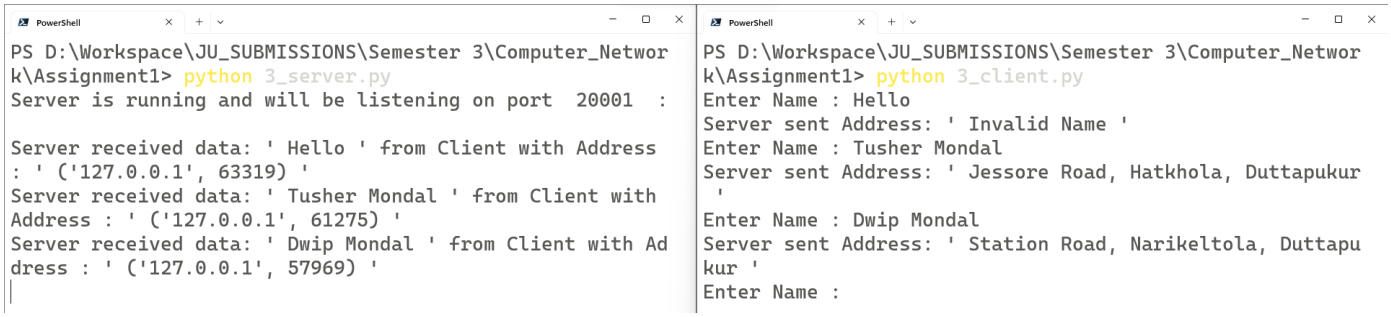
```
# send a name to the server, receive an address, and display it.
# To stop the client, you may need to manually terminate the program.
```

3_server.py

```
import socket
import Additional_Functions as AF    # Assuming this module contains
necessary functions

# Define local IP address and port for the server socket
LOCAL_IP = "127.0.0.1"
LOCAL_PORT = 20001

# Define the buffer size for receiving data
BUF_SIZE = 1024

# Response prefix
Reset = 'Hello Client with Address : '

# Create a server socket using UDP
Server_Socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind the server socket to the specified local IP address and port
Server_Socket.bind((LOCAL_IP, LOCAL_PORT))

# Print server information
print("Server is running and will be listening on port ", LOCAL_PORT, " :
")

while True:
          #  Read  collective  information  from  a  file  using  the
Additional_Functions module
    #try block to handle any exceptions
    try:
        Collective_Info = AF.Read_File()
    except:
        print("Error reading file!")
        break

    # Receive incoming packet from the client
```

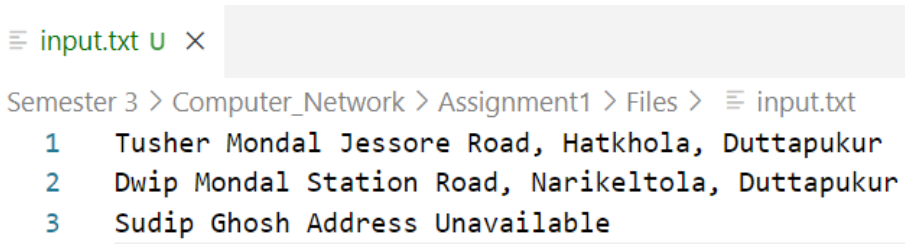
```
    Incoming_Packet = Server_Socket.recvfrom(BUF_SIZE)
    Incoming_Data, Client_Address = Incoming_Packet

    # Decode the received data and print client information
    Incoming_Data = Incoming_Data.decode()
     print("Server received data: '", Incoming_Data, "' from Client with
Address : '", Client_Address, "'")

    if Incoming_Data in Collective_Info:
         # If the incoming data (name) is in the collective info, send the
corresponding info back
        Info = Collective_Info[Incoming_Data]
        Server_Socket.sendto(Info.encode(), Client_Address)
    else:
        # If the name is not found, send an "Invalid Name" reply
        Server_Socket.sendto("Invalid Name".encode(), Client_Address)
```

Sample Run



```
PS D:\Workspace\JU_SUBMISSIONS\Semester 3\Computer_Networ
k\Assignment1> python 3_server.py
Server is running and will be listening on port  20001  :

Server received data: ' Hello ' from Client with Address
: ' ('127.0.0.1', 63319) '
Server received data: ' Tusher Mondal ' from Client with
Address : ' ('127.0.0.1', 61275) '
Server received data: ' Dwip Mondal ' from Client with Ad
dress : ' ('127.0.0.1', 57969) '
```

```
PS D:\Workspace\JU_SUBMISSIONS\Semester 3\Computer_Networ
k\Assignment1> python 3_client.py
Enter Name : Hello
Server sent Address: ' Invalid Name '
Enter Name : Tusher Mondal
Server sent Address: ' Jessore Road, Hatkhola, Duttapukur
 '
Enter Name : Dwip Mondal
Server sent Address: ' Station Road, Narikeltola, Duttapu
kur '
Enter Name :
```



input.txt U

Semester 3 > Computer_Network > Assignment1 > Files > input.txt

```
1   Tusher Mondal Jessore Road, Hatkhola, Duttapukur
2   Dwip Mondal Station Road, Narikeltola, Duttapukur
3   Sudip Ghosh Address Unavailable
```
