University of Dhaka

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 2 :

Introduction to Client-Server Communication using Socket Programming — Simulating an ATM Machine Operation.

Submitted By:

Name : MD. Farhan Tanvir

Roll No : 45

Name : Asef Sami Chowdhury

Roll No : 53

Submitted On :

February 01, 2024

Submitted To :

Dr. Md. Abdur Razzaque

Dr. Md. Mamun Or Rashid

Dr. Md. Muhammad Ibrahim

Md. Redwan Ahmed Rizvee

1 Introduction

In the realm of distributed computing, the client-server model stands as a foundational architecture, facilitating seamless communication between entities on a network. This report delves into the intricacies of client-server communication using Socket Programming, with a focus on simulating an ATM machine operation. The use of sockets enables the establishment of robust connections between a server process (running on machine A) and a client process (running on machine B).

The primary objectives of this endeavor include creating TCP connections and implementing non-idempotent operations with exactly-once semantics. The simulated ATM machine operation involves diverse functionalities, such as converting text case, checking for prime or palindrome numbers, and conducting financial transactions.

Through the application of Socket Programming, we explore the intricacies of data exchange between the client and server, investigating the challenges posed by potential errors in messages, responses, and process execution. The report delves into the development of an application-level protocol for communication between an ATM and a bank's centralized server, emphasizing robust error handling mechanisms to ensure the integrity and reliability of the communication channel.

This experiment provides a comprehensive exploration of the methodologies employed for both server-side and client-side operations, shedding light on the intricacies of the implemented code. The subsequent sections delve into the detailed steps of creating TCP connections, executing non-idempotent operations, and enhancing error handling for a seamless client-server communication experience.

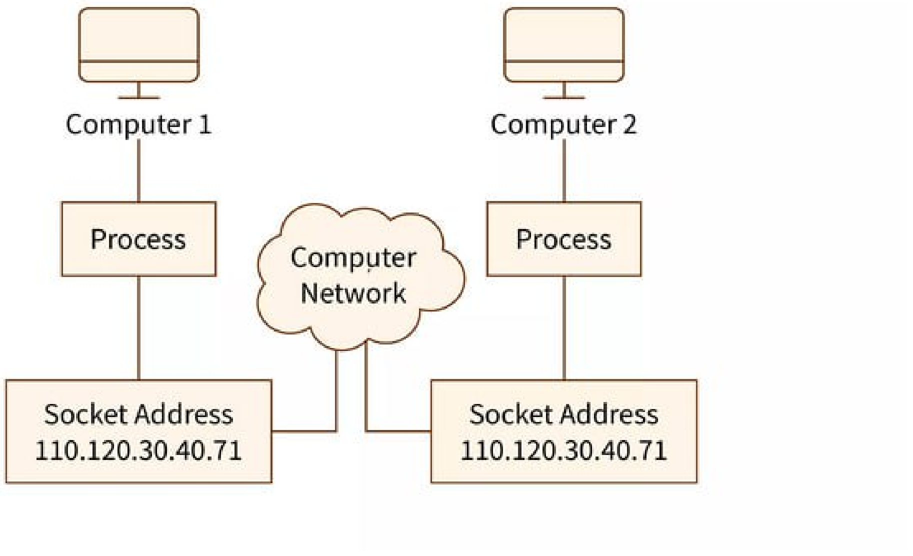


Figure 1: Client-Server Communication Flow Chart

1.1 Objectives

- Understand and implement Socket Programming for establishing a TCP connection between a server process on host A and a client process on host B.
- Execute operations on the server in response to client requests, including converting a line of text from capital to small letters and checking if a given integer is prime or a palindrome. Use testing tools like *name of the tool* to validate the functionality.
- Design and implement a non-idempotent operation with exactly-once semantics, focusing on handling failures in request messages, response messages, and process execution. Analyze and evaluate the system's reliability and fault tolerance.
- Simulate an authentic ATM experience by enabling users to conduct banking transactions from their own computers.
- Seamlessly integrate the Python programming language with Transmission Control Protocol (TCP) to establish a reliable connection between the server and client computers.
- Seamlessly integrate the Python programming language with Transmission Control Protocol (TCP) to establish a reliable connection between the server and client computers.

2 Theory

2.1 Sockets:

Sockets act as communication endpoints, enabling processes on different devices to exchange data seamlessly. They are a software interface allowing programs to send and receive data over a network, fostering communication across diverse devices regardless of underlying hardware and operating system.

2.1.1 Role of Sockets:

Sockets operate on a client-server model. The server, waiting for connections, binds to a specific port, while the client initiates a connection by specifying the server's IP address and port number. This model empowers the development of distributed applications, facilitating communication and information sharing across the network.

Sockets offer a programming interface for network communication, abstracting the complexities of underlying protocols. This abstraction lets developers focus on application logic without delving into the intricacies of networking.

2.2 TCP:

Transmission Control Protocol (TCP) is integral to the Internet Protocol suite, operating at the transport layer. It provides reliable, connection-oriented communication, ensuring data delivery in the correct order and handling retransmission of lost or corrupted packets. TCP is suitable for applications requiring high reliability, like file transfer and web browsing.

TCP establishes a connection using a three-way handshake, and once established, facilitates stream-oriented data exchange. This connection-oriented approach maintains data integrity, making TCP a preferred choice for applications where accurate and ordered data delivery is crucial.

2.3 Error Handling:

Error handling in network communication is crucial for maintaining the reliability and integrity of data exchange between clients and servers. Networks are prone to various uncertainties, such as packet loss, hardware failures, or unexpected interruptions. Effective error handling ensures that these uncertainties do not compromise the functionality and security of the communication process.

2.4 Client-Server Communication:

TCP has a few functionalities like, TCP is suited for applications that require high reliability, and transmission time is relatively less critical. It is used by other protocols like HTTP, HTTPs, FTP, SMTP, Telnet. TCP rearranges data packets in the order specified. There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. TCP does Flow Control and requires three packets to set up a socket connection before any user data can be sent. TCP handles reliability and congestion control. It also does error checking and error recovery.

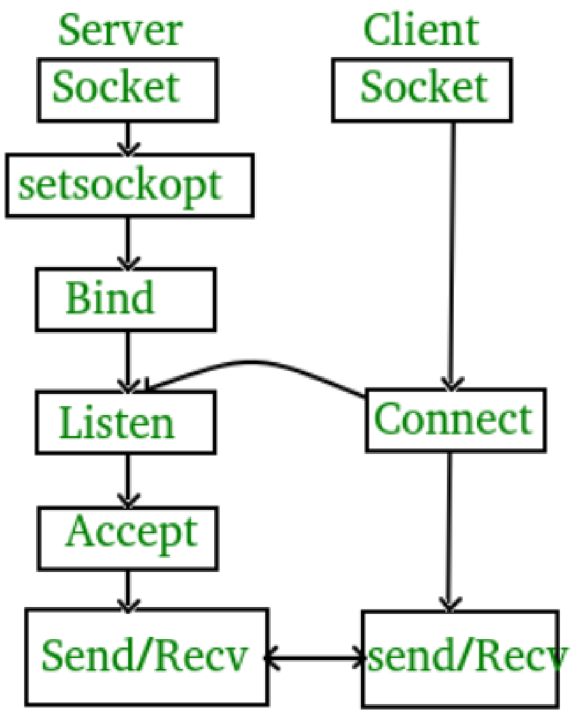


Figure 2: Client-Server Communication Flow Chart

3 Methodology

3.1 Server-side Operations

In the context of simulating an ATM machine operation through client-server communication using Socket Programming, the methodology encompasses two main aspects: creating TCP connections and implementing non-idempotent operations with exactly-once semantics.

3.1.1 Creating TCP Connections

a) Establishing a TCP Connection: The server process, hosted on machine A, and the client process, running on machine B, initiate a TCP connection using Socket Programming. Upon connection, the server performs operations based on client requests and sends responses accordingly.

i. Capital Letter to Small Letter Conversion:

- The client requests the server to convert a line of text from capital letters to small letters.
- The server processes the request, performs the conversion, and sends the modified text back to the client.

ii. Prime or Palindrome Check:

- The client sends an integer along with an operation name ('prime' or 'palindrome') to the server.
- The server checks whether the integer is prime or a palindrome and communicates the result to the client.

b) Non-Idempotent Operation with Exactly-Once Semantics: Design and implement a non-idempotent operation ensuring exactly-once semantics. Handle potential failures, including request message failures, response message failures, and process execution failures.

i. Application-level Protocol for ATM-Bank Communication:

- Develop a protocol for communication between an ATM and a bank's centralized server.
- Include operations for user card and password verification, account balance inquiry, and account withdrawal.
- Specify how the protocol entities handle cases where there's insufficient account balance for withdrawal.
- List messages exchanged and actions taken by the ATM and the bank's centralized computer during transmission and receipt of messages.
- Provide a sketch illustrating the protocol's operation in a scenario of a simple withdrawal with no errors.

ii. Enhancement for Error Handling: To fortify the communication protocol's robustness, we introduce enhancements to adeptly manage errors stemming from both the client and server sides. The approach involves the following measures:

- Comprehensive Error Handling: - The protocol is augmented to detect and handle various error scenarios, ensuring a resilient system.
- Packet Dropout Mechanism: - In the event of an error, be it on the server or client side, the protocol employs a packet dropout strategy. When an error occurs, the affected packet is intelligently discarded to prevent any potential compromise to the transaction integrity.
- Automatic Packet Resend: - Following a detected error and subsequent packet dropout, the client machine initiates an automatic resend of the data to the server machine. This mechanism ensures the completion of the transaction without manual intervention, enhancing the overall reliability of the communication process.
- Transaction Continuity: - The designed error-handling mechanism guarantees transaction continuity. By automatically reinitiating the data exchange process, the protocol minimizes disruptions caused by errors, fostering a seamless and dependable communication channel between the ATM and the bank's centralized server.

3.2 Client-side Operations

In the client-side operations, the focus is on the interaction with the server, acting as the client, for simulating ATM machine operations. This involves creating TCP connections and implementing non-idempotent operations with exactly-once semantics.

3.2.1 Creating TCP Connections

a) Establishing a TCP Connection: The client-side operation begins by initiating a TCP connection to the server, hosted on machine A, from the client process running on machine B. This connection is established through Socket Programming. Once connected, the client is ready to send requests and receive responses from the server.

Capital Letter to Small Letter Conversion:

- The client inputs a line of text in lowercase and sends a request to the server for conversion.
- The server processes the request, converts the text from capital letters to small letters, and sends the modified text back to the client for display or further processing.

Prime or Palindrome Check:

- The client inputs an integer and specifies an operation name ('prime' or 'palindrome') to be performed by the server.
- The server checks whether the given integer satisfies the specified operation and communicates the result back to the client for display or further action.

b) Non-Idempotent Operation with Exactly-Once Semantics: Similar to the server-side, the client also participates in non-idempotent operations with exactly-once semantics. This ensures robust handling of potential failures in requests, responses, and process execution.

Enhanced Error Handling in Client-Server Communication: To fortify the communication protocol's robustness on the client side, we introduce enhancements adept at managing errors originating from both the client and server sides. The measures include comprehensive error handling, a packet dropout mechanism, automatic packet resend, and transaction continuity.

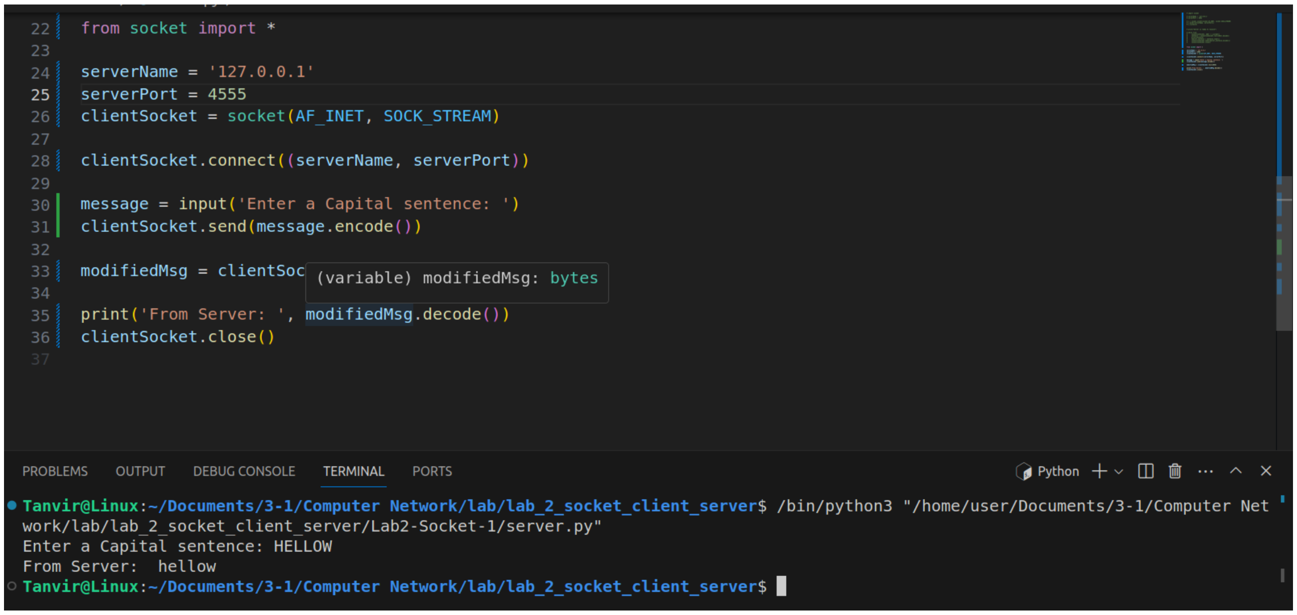
The client-side operations ensure effective and resilient communication with the server, supporting the simulation of ATM machine operations through Socket Programming.

4 Experimental result

Presented here are the experimental results of our Capital letter to Small letter conversion,operation for (either 'prime' or 'palindrome') to the server and check whether it's a prime (or palindrome) or not and ATM client-server system, with dedicated screenshots provided for each specific task to offer a comprehensive visual overview

4.1 Capital letter to Small letter Conversion

4.1.1 Client Screen Shot

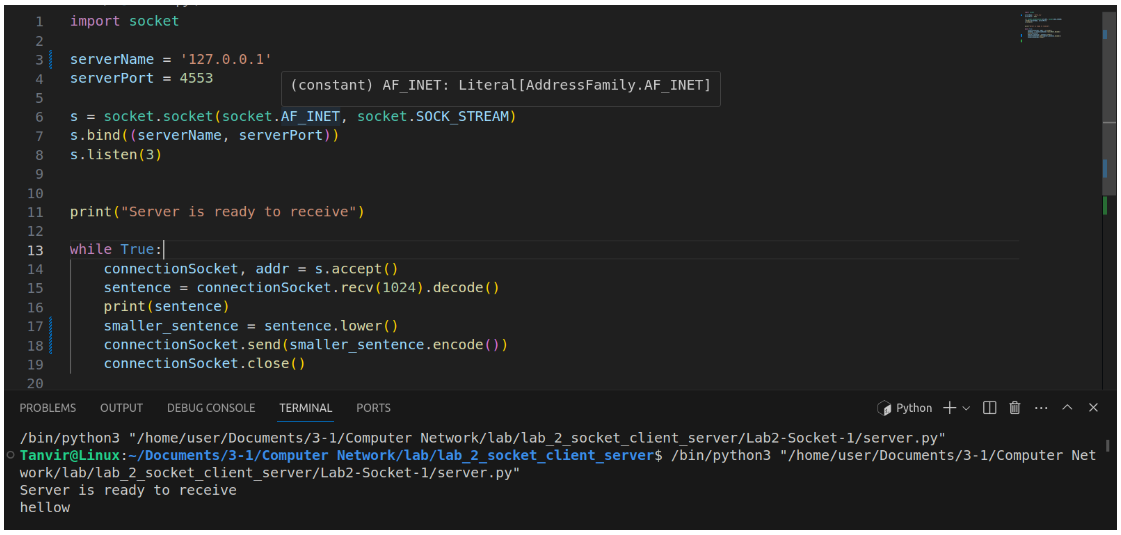


```
from socket import *

serverName = '127.0.0.1'
serverPort = 4555
clientSocket = socket(AF_INET, SOCK_STREAM)

clientSocket.connect((serverName, serverPort))

message = input('Enter a Capital sentence: ')
clientSocket.send(message.encode())

modifiedMsg = clientSoc (variable) modifiedMsg: bytes

print('From Server: ', modifiedMsg.decode())
clientSocket.close()
```

```
Tanvir@Linux:~/Documents/3-1/Computer Network/lab/lab_2_socket_client_server$ /bin/python3 "/home/user/Documents/3-1/Computer Net
work/lab/lab_2_socket_client_server/Lab2-Socket-1/server.py"
Enter a Capital sentence: HELLOW
From Server:  hellow
Tanvir@Linux:~/Documents/3-1/Computer Network/lab/lab_2_socket_client_server$ 
```

Figure 3: Input from client HELLOW , Output From Server hellow

4.1.2 Server Screen Shot



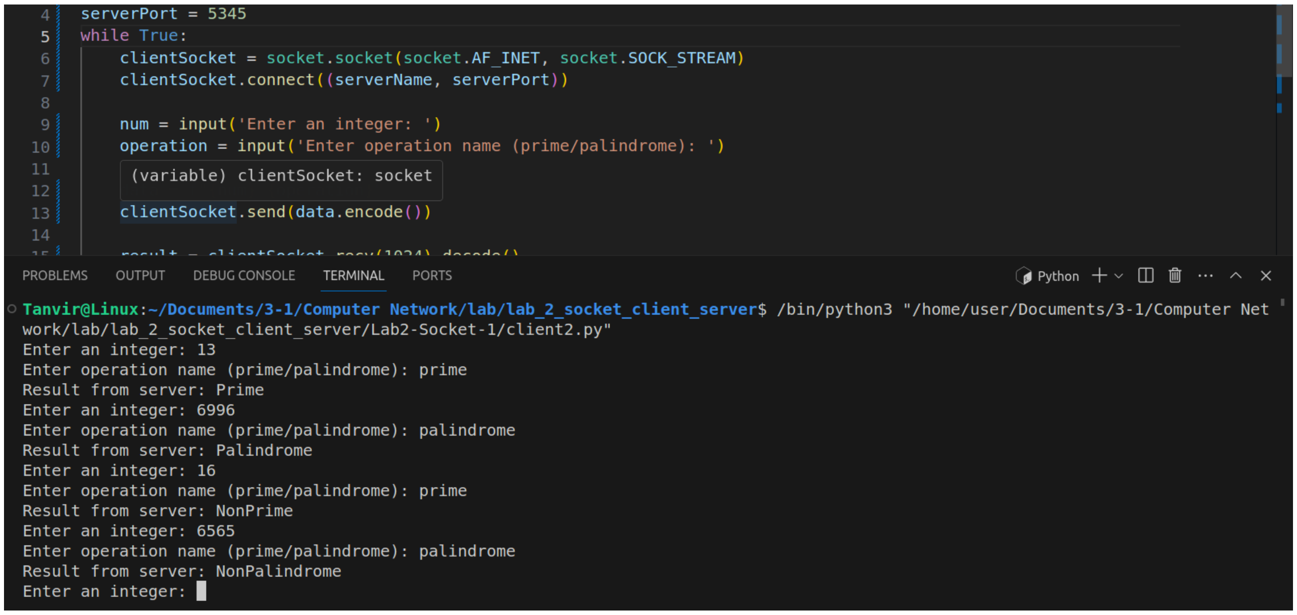
```
import socket

serverName = '127.0.0.1'
serverPort = 4553                    (constant) AF_INET: Literal[AddressFamily.AF_INET]

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((serverName, serverPort))
s.listen(3)


print("Server is ready to receive")

while True:
    connectionSocket, addr = s.accept()
    sentence = connectionSocket.recv(1024).decode()
    print(sentence)
    smaller_sentence = sentence.lower()
    connectionSocket.send(smaller_sentence.encode())
    connectionSocket.close()
```

```
/bin/python3 "/home/user/Documents/3-1/Computer Network/lab/lab_2_socket_client_server/Lab2-Socket-1/server.py"
Tanvir@Linux:~/Documents/3-1/Computer Network/lab/lab_2_socket_client_server$ /bin/python3 "/home/user/Documents/3-1/Computer Net
work/lab/lab_2_socket_client_server/Lab2-Socket-1/server.py"
Server is ready to receive
hellow
```

Figure 4: Server Received HELLOW,and make it hellow, I print it to the server to show

4.2 'prime' or 'palindrome' Check

4.2.1 Client Screen Shot

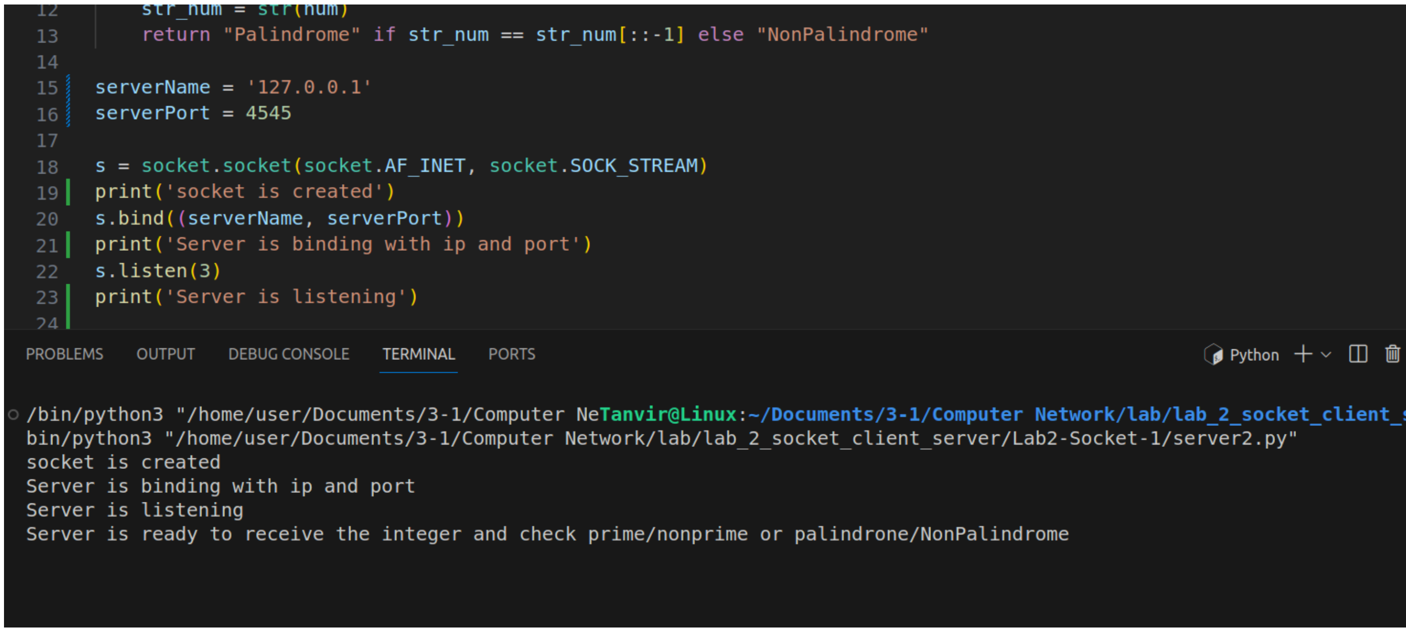


```
serverPort = 5345
while True:
    clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    clientSocket.connect((serverName, serverPort))

    num = input('Enter an integer: ')
    operation = input('Enter operation name (prime/palindrome): ')
    (variable) clientSocket: socket

    clientSocket.send(data.encode())

```

```
Tanvir@Linux:~/Documents/3-1/Computer Network/lab/lab_2_socket_client_server$ /bin/python3 "/home/user/Documents/3-1/Computer Net
work/lab/lab_2_socket_client_server/Lab2-Socket-1/client2.py"
Enter an integer: 13
Enter operation name (prime/palindrome): prime
Result from server: Prime
Enter an integer: 6996
Enter operation name (prime/palindrome): palindrome
Result from server: Palindrome
Enter an integer: 16
Enter operation name (prime/palindrome): prime
Result from server: NonPrime
Enter an integer: 6565
Enter operation name (prime/palindrome): palindrome
Result from server: NonPalindrome
Enter an integer:
```

Figure 5: Checking Either the integer is prime/palindrome or not

4.2.2 Server Screen Shot



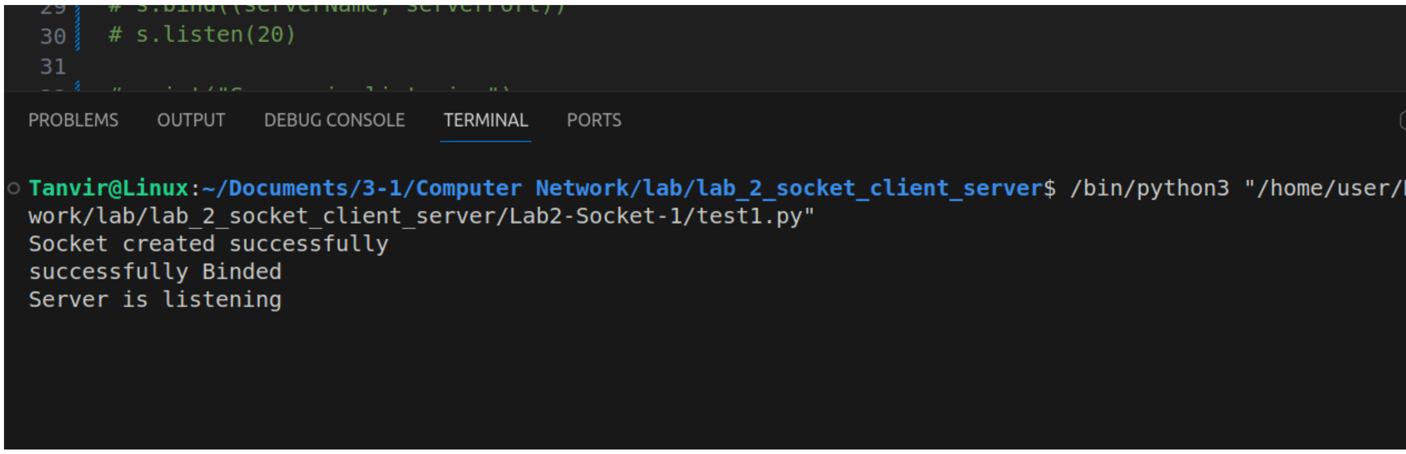
```
    str_num = str(num)
    return "Palindrome" if str_num == str_num[::-1] else "NonPalindrome"

serverName = '127.0.0.1'
serverPort = 4545

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print('socket is created')
s.bind((serverName, serverPort))
print('Server is binding with ip and port')
s.listen(3)
print('Server is listening')
```

```
/bin/python3 "/home/user/Documents/3-1/Computer NeTanvir@Linux:~/Documents/3-1/Computer Network/lab/lab_2_socket_client_s
bin/python3 "/home/user/Documents/3-1/Computer Network/lab/lab_2_socket_client_server/Lab2-Socket-1/server2.py"
socket is created
Server is binding with ip and port
Server is listening
Server is ready to receive the integer and check prime/nonprime or palindrone/NonPalindrome
```

Figure 6: Prime And Palindrime Checker Server

4.3 ATM Server Screen Shot

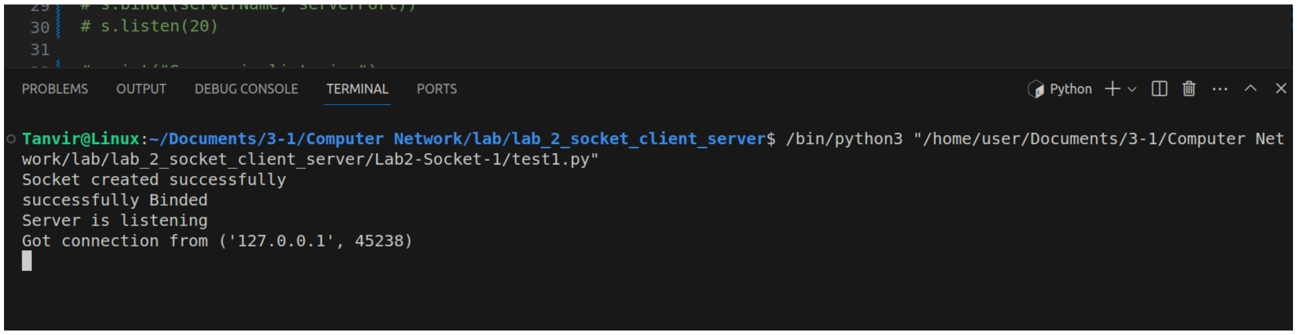
4.3.1 Socket, Bind,Listening



```
    # s.bind((serverName, serverPort))
    # s.listen(20)
```

```
Tanvir@Linux:~/Documents/3-1/Computer Network/lab/lab_2_socket_client_server$ /bin/python3 "/home/user/
work/lab/lab_2_socket_client_server/Lab2-Socket-1/test1.py"
Socket created successfully
successfully Binded
Server is listening
```

Figure 7: Socket Created,Binding with ip and port,Server Listening



```
# s.listen(20)
```

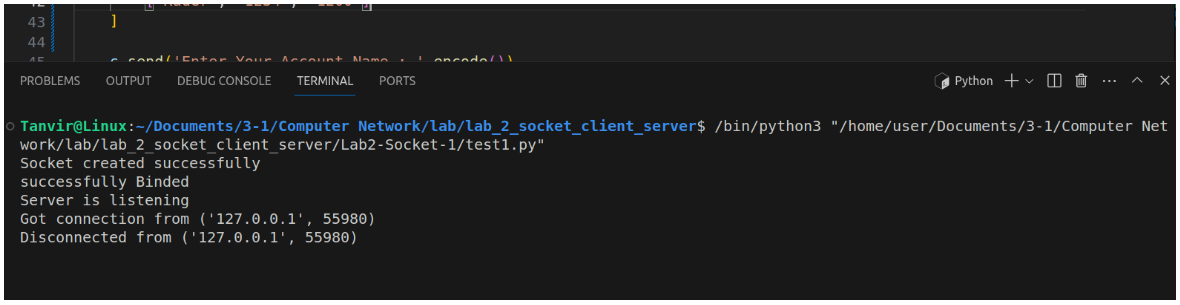
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Tanvir@Linux:~/Documents/3-1/Computer Network/lab/lab_2_socket_client_server$ /bin/python3 "/home/user/Documents/3-1/Computer Net
work/lab/lab_2_socket_client_server/Lab2-Socket-1/test1.py"
Socket created successfully
successfully Binded
Server is listening
Got connection from ('127.0.0.1', 45238)
```

Figure 8: Got connection from client address

4.3.2 Connected To client

4.3.3 Server Disconnected When client Exit



```
]
```

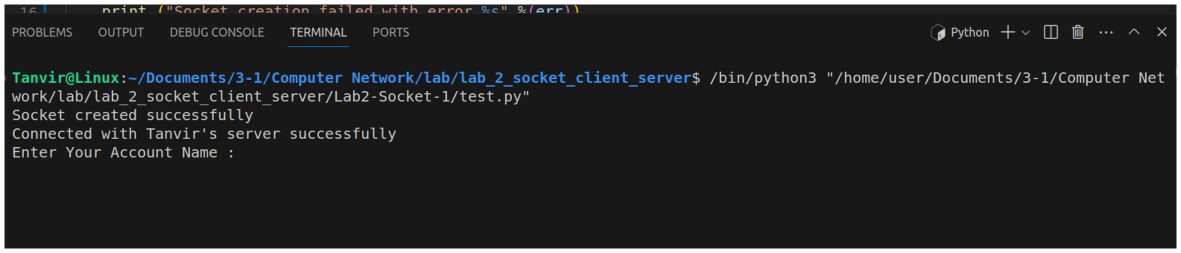
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Tanvir@Linux:~/Documents/3-1/Computer Network/lab/lab_2_socket_client_server$ /bin/python3 "/home/user/Documents/3-1/Computer Net
work/lab/lab_2_socket_client_server/Lab2-Socket-1/test1.py"
Socket created successfully
successfully Binded
Server is listening
Got connection from ('127.0.0.1', 55980)
Disconnected from ('127.0.0.1', 55980)
```

Figure 9: Disconnected from this Client When client Exit

4.4 ATM Client Screen Shot

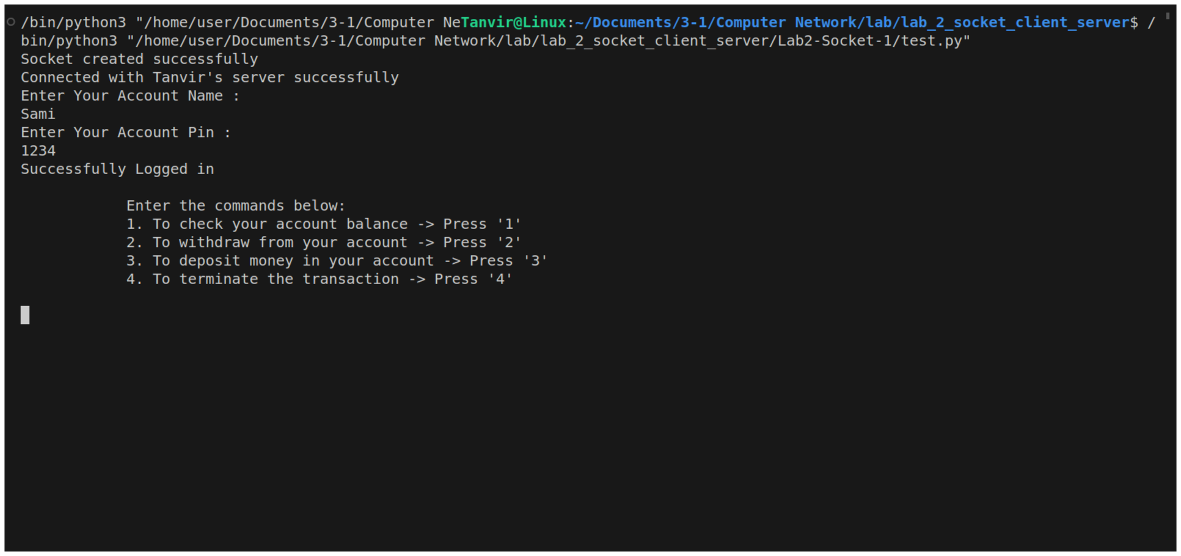
4.4.1 Client Socket Creation And Client To Server Connection



```
Tanvir@Linux:~/Documents/3-1/Computer Network/lab/lab_2_socket_client_server$ /bin/python3 "/home/user/Documents/3-1/Computer Net
work/lab/lab_2_socket_client_server/Lab2-Socket-1/test.py"
Socket created successfully
Connected with Tanvir's server successfully
Enter Your Account Name :
```

Figure 10: Client Socket Created and This CLient is now Connected to The ATM Server

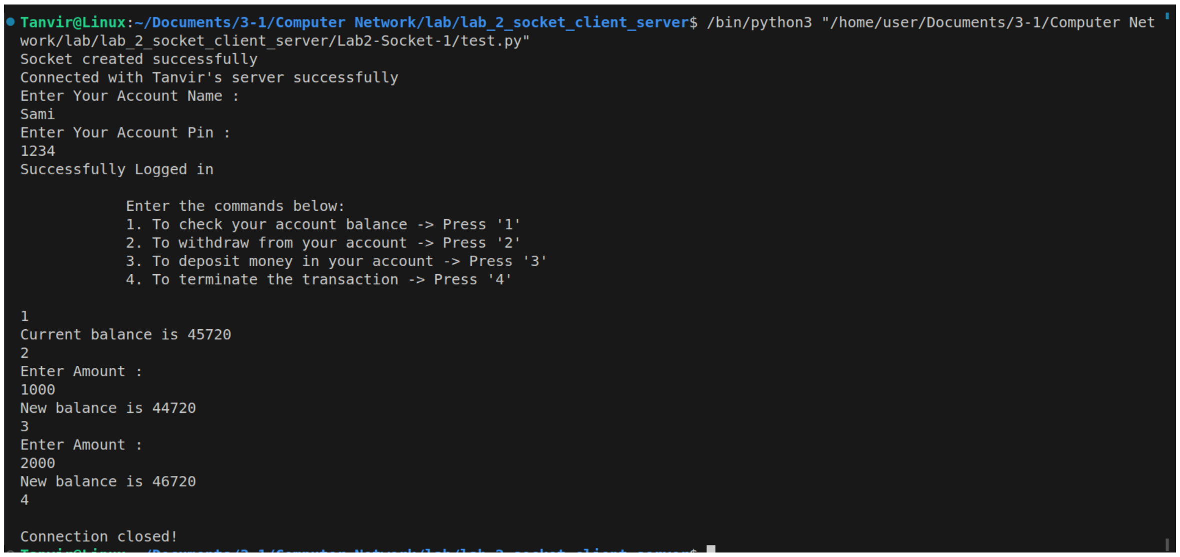
4.4.2 Client Login



```
/bin/python3 "/home/user/Documents/3-1/Computer NeTanvir@Linux:~/Documents/3-1/Computer Network/lab/lab_2_socket_client_server$ /
bin/python3 "/home/user/Documents/3-1/Computer Network/lab/lab_2_socket_client_server/Lab2-Socket-1/test.py"
Socket created successfully
Connected with Tanvir's server successfully
Enter Your Account Name :
Sami
Enter Your Account Pin :
1234
Successfully Logged in

            Enter the commands below:
            1. To check your account balance -> Press '1'
            2. To withdraw from your account -> Press '2'
            3. To deposit money in your account -> Press '3'
            4. To terminate the transaction -> Press '4'
```

Figure 11: Client Successfully Loged in by giving Name And Pin

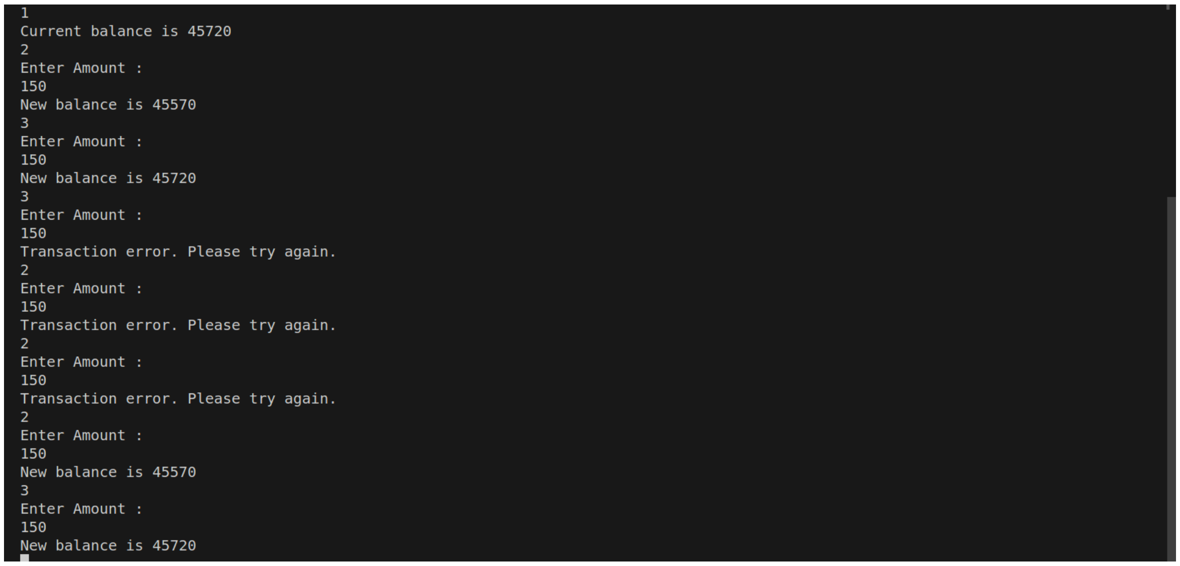
4.4.3 Withdrawal,Deposit Without Error



```
Tanvir@Linux:~/Documents/3-1/Computer Network/lab/lab_2_socket_client_server$ /bin/python3 "/home/user/Documents/3-1/Computer Net
work/lab/lab_2_socket_client_server/Lab2-Socket-1/test.py"
Socket created successfully
Connected with Tanvir's server successfully
Enter Your Account Name :
Sami
Enter Your Account Pin :
1234
Successfully Logged in

            Enter the commands below:
            1. To check your account balance -> Press '1'
            2. To withdraw from your account -> Press '2'
            3. To deposit money in your account -> Press '3'
            4. To terminate the transaction -> Press '4'

1
Current balance is 45720
2
Enter Amount :
1000
New balance is 44720
3
Enter Amount :
2000
New balance is 46720
4

Connection closed!
```

Figure 12: Client Can Withdraw and Deposit after Log in

4.4.4 Withdrawal,Deposit With Error



```
1
Current balance is 45720
2
Enter Amount :
150
New balance is 45570
3
Enter Amount :
150
New balance is 45720
3
Enter Amount :
150
Transaction error. Please try again.
2
Enter Amount :
150
Transaction error. Please try again.
2
Enter Amount :
150
Transaction error. Please try again.
2
Enter Amount :
150
New balance is 45570
3
Enter Amount :
150
New balance is 45720
```

Figure 13: Withdraw and Deposit After Adding Error

4.5 Graphical Representation Of Client and Server Error

4.5.1 Server Side Error

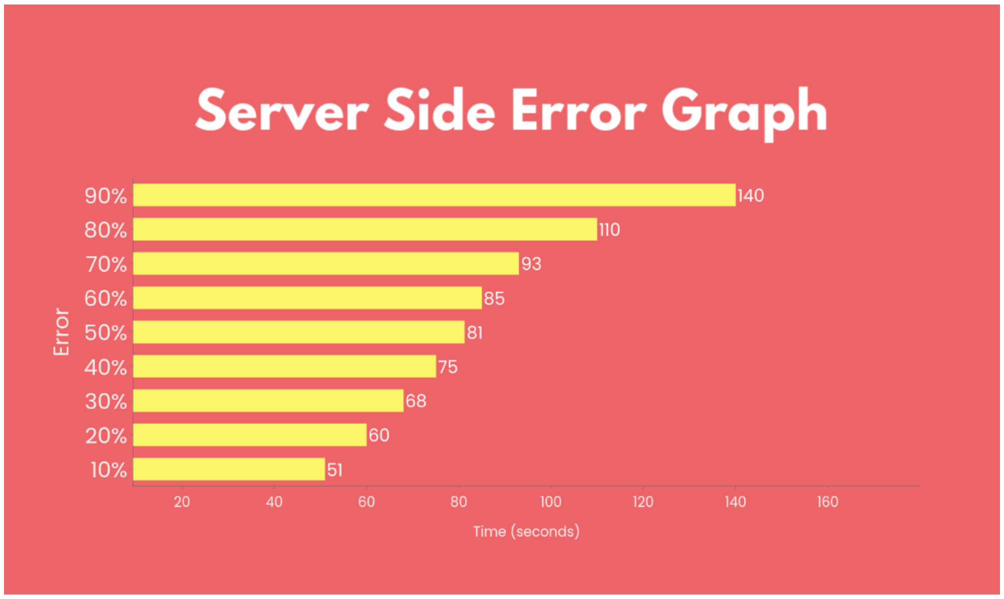


Figure 14: Server Error

4.5.2 Client Side Error

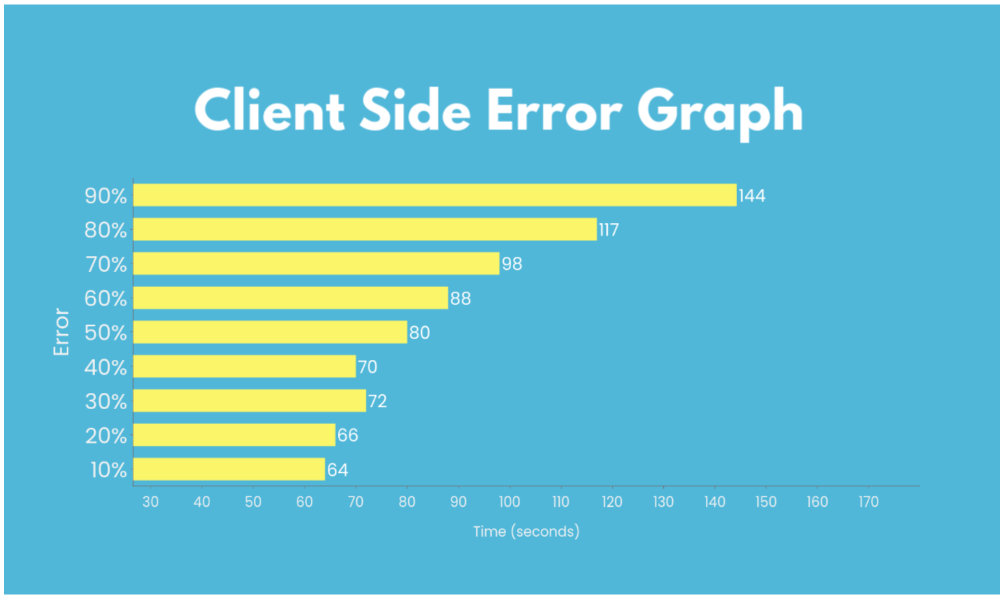


Figure 15: Server Error

5 Client Code Examples:

Written by Asef Sami Chowdhury.

5.1 Capital letter to Small letter conversion for a line of text.

```
from socket import *

serverName = '10.42.0.111'
serverPort = 4553
clientSocket = socket(AF_INET, SOCK_STREAM)

clientSocket.connect((serverName, serverPort))

message = input('Enter a uppercase sentence: ')
clientSocket.send(message.encode())

modifiedMsg = clientSocket.recv(2048)
print('From Server: ', modifiedMsg.decode())

clientSocket.close()
```

5.2 Send an integer and operation name (either 'prime' or 'palindrome') to the server and check whether it's a prime (or palindrome) or not.

```
from socket import *

serverName = '10.42.0.111'
serverPort = 5345
#creating socket
clientSocket = socket(AF_INET, SOCK_STREAM)

clientSocket.connect((serverName, serverPort))

integernum = input('Enter a integer number: ')
operation = input('Enter operation name (prime/palindrome): ')

#sending integer and operation to the server
data = f"{integernum} {operation}"
clientSocket.send(data.encode())

result = clientSocket.recv(1024).decode()
print(f"Result from server: {result}")

clientSocket.close()
```

5.3 Design and implement a non-idempotent operation using exactly-once semantics that can handle the failure of request messages, failure of response messages and process execution failures.Without error handling.

```
# client code with time
import socket
import time

# Server’s IP address
serverName = ’10.33.3.9’

# Port for communication
serverPort = 5353

# Creating Client Socket
try:
    clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print("Socket created successfully")
except socket.error as err:
    print("Socket creation failed with error %s" % (err))

# Record the start time
start_time = time.time()

# Connecting to the server
try:
    clientSocket.connect((serverName, serverPort))
    print("Connected with Tanvir’s server successfully1")
except Exception as err:
    print("Connection with server failed! Error is: %s" % (err))

# Record the connection time
connection_time = time.time() - start_time
#print(f"Connection time: {connection_time} seconds")

while True:
    print(clientSocket.recv(1024).decode())

    # send account name
    name = input()
    clientSocket.send(name.encode())

    print(clientSocket.recv(1024).decode())

    # send account pin
    pin = input()
    clientSocket.send(pin.encode())

    print(clientSocket.recv(1024).decode())

    while True:
        command = input()
        clientSocket.send(command.encode())
        print(clientSocket.recv(1024).decode())

        if command == '4':  # exit
            break
        elif command == '3' or command == '2':  # deposit / withdraw
            amount = input()
            clientSocket.send(amount.encode())
            print(clientSocket.recv(1024).decode())
    print("Connection closed!")

    # Record the end time
    end_time = time.time()
    total_time = end_time - start_time
    print(f"Total time: {total_time} seconds")
    break

clientSocket.close()
```

5.4 Design and implement a non-idempotent operation using exactly-once semantics that can handle the failure of request messages, failure of response messages and process execution failures.With error handling.

```
# for socket
import socket

# for random error
import random

#Server's IP address
serverName = '10.42.0.111'

#Port for communication
serverPort = 4553

# Creating Client Socket
try:
    clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print ("Socket created successfully")
except socket.error as err:
    print ("Socket creation failed with error %s" %(err))


# Connecting to the server
try:
    clientSocket.connect((serverName, serverPort))
    print ("Connected with Tanvir’s server successfully1")
except Exception as err:
    print("Connection with server failed! Error is: %s" %(err))


while True:

    print(clientSocket.recv(1024).decode())

    # send account name
    name = input()
    clientSocket.send(name.encode())

    print(clientSocket.recv(1024).decode())

    # send account pin
    pin = input()
    clientSocket.send(pin.encode())

    print(clientSocket.recv(1024).decode())

    while True:
        #print(’Select a command to continue: ’)
        command = input()
        clientSocket.send(command.encode())
        print(clientSocket.recv(1024).decode())

        if command == ’4’: #exit
            break
        elif command == ’3’ or command == ’2’: #deposit / withdraw
            amount = input()
            clientSocket.send(amount.encode())
            print(clientSocket.recv(1024).decode())
    print("Connection closed!")
    break

clientSocket.close()
```

6 Server Code Examples:

Written by Farhan Tanvir.

6.1 Capital letter to Small letter conversion for a line of text.

```
import socket

serverName = '10.42.0.111'
serverPort = 4553

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((serverName, serverPort))
s.listen(3)


print("Server is ready to receive")

while True:
    connectionSocket, addr = s.accept()
    sentence = connectionSocket.recv(1024).decode()
    print(sentence)
    smaller_sentence = sentence.lower()
    connectionSocket.send(smaller_sentence.encode())
    connectionSocket.close()
```

6.2 Send an integer and operation name (either 'prime' or 'palindrome') to the server and check whether it's a prime (or palindrome) or not.

```
import socket

def is_prime(num):
    if num < 2:
        return "NonPrime"
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return "NonPrime"
    return "Prime"

def is_palindrome(num):
    str_num = str(num)
    return "Palindrome" if str_num == str_num[::-1] else "NonPalindrome"

serverName = '10.42.0.111'
serverPort = 4545
```

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print('socket is created')
s.bind((serverName, serverPort))
print('Server is binding with ip and port')
s.listen(3)
print('Server is listening')


print("Server is ready to receive the integer and check prime/nonprime or
        palindrone/NonPalindrome")

while True:
    connectionSocket, addr = s.accept()
    data = connectionSocket.recv(1024).decode().split()

    if len(data) == 2:
        try:
            num = int(data[0])
            operation = data[1].lower()

            result = None

            if operation == 'prime':
                result = is_prime(num)

            elif operation == 'palindrome':
                result = is_palindrome(num)

            connectionSocket.send(result.encode())
        except ValueError:
            connectionSocket.send("Invalid input.
            Please send an integer and operation.".encode())
    else:
        connectionSocket.send("Invalid input.
        Please send an integer and operation.".encode())

    connectionSocket.close()
```

6.3 Design and implement a non-idempotent operation using exactly-once semantics that can handle the failure of request messages, failure of response messages and process execution failures.Without error handling.

```
import socket

serverName = '127.0.0.1'
serverPort = 4553

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((serverName, serverPort))
s.listen(20)

print("Server is listening")
c, addr = s.accept()
print('Got connection from', addr)


while True:

    account_list = [
        ['Farhan', '1234', '1169'],
        ['Sami', '1234', '45720'],
        ['Moyeed', '1234', '70000'],
        ['Kader', '1234', '1200']
    ]

    c.send('Enter Your Account Name : '.encode())
    account_name = c.recv(1024).decode()
    c.send('Enter Your Account Pin : '.encode())
    pincode = c.recv(1024).decode()

    verification_status = False
    account_info = []

    for acc in account_list:
        if acc[0] == account_name and acc[1] == pincode:
            verification_status = True
            account_info = acc
            break

    if verification_status:
        menu_prompt = """
            Enter the commands below:
            1. To check your account balance -> Press '1'
            2. To withdraw from your account -> Press '2'
            3. To deposit money in your account -> Press '3'
            4. To terminate the transaction -> Press '4'
        """
        c.send(f"Successfully Logged in\n{menu_prompt}".encode())

        while True:
            command = c.recv(1024).decode()

            if command == '1':
                c.send(f'Current balance is {account_info[2]}'.encode())

            elif command == '4':
                break

            elif command == '3' or command == '2':
                c.send("Enter Amount : ".encode())
                amount = c.recv(1024).decode()

                if command == '3':
                    new_amt = int(account_info[2]) + int(amount)
                    account_info[2] = str(new_amt)
                    c.send(f'New balance is {account_info[2]}'.encode())

                elif command == '2':
                    cur_balance = int(account_info[2])

                    if cur_balance >= int(amount):
                        cur_balance -= int(amount)
                        account_info[2] = str(cur_balance)
                        c.send(f'New balance is {account_info[2]}'.encode())

                    else:
                        c.send('Insufficient Balance'.encode())

            else:
                c.send('Enter a valid command'.encode())

    else:
        c.send("Incorrect account name/pin\n".encode())

    c.close()
```

6.4 Design and implement a non-idempotent operation using exactly-once semantics that can handle the failure of request messages, failure of response messages and process execution failures.With error handling.

```
import socket
import random

serverName = '10.42.0.111'
serverPort = 5353

try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print ("Socket created successfully")
except socket.error as err:
    print ("Socket creation failed with error %s" %(err))


# connecting to the server
try:
    s.bind((serverName, serverPort))
    print ("successfully Binded")
except Exception as err:
    print("Binding failed! Error is: %s" %(err))

try:
    s.listen(20)
    print("Server is listening")
except Exception as err:
    print("Listening failed! Error is: %s" %(err))

# s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# s.bind((serverName, serverPort))
# s.listen(20)

# print("Server is listening")

while True:
    c, addr = s.accept()
    print('Got connection from', addr)

    account_list = [
        ['Farhan', '1234', '1169'],
        ['Sami', '1234', '45720'],
        ['Moyeed', '1234', '70000'],
        ['Kader', '1234', '1200']
    ]
```

```
c.send('Enter Your Account Name : '.encode())
account_name = c.recv(1024).decode()
c.send('Enter Your Account Pin : '.encode())
pincode = c.recv(1024).decode()

verification_status = False
account_info = []

for acc in account_list:
    if acc[0] == account_name and acc[1] == pincode:
        verification_status = True
        account_info = acc
        break

if verification_status:
    command_promt = """
        Enter the commands below:
        1. To check your account balance -> Press '1'
        2. To withdraw from your account -> Press '2'
        3. To deposit money in your account -> Press '3'
        4. To terminate the transaction -> Press '4'
    """
    c.send(f"Successfully Logged in\n{command_promt}".encode())

    transaction_count = 0

    while True:
        command = c.recv(1024).decode()

        if command == '1':
            c.send(f'Current balance is {account_info[2]}'.encode())

        elif command == '4':
            print('Disconnected from', addr)
            c.close()
            break

        elif command == '3' or command == '2':
            c.send("Enter Amount : ".encode())
            amount = c.recv(1024).decode()

            # Simulating a random transaction error
            random_number = random.randint(1, 10)
            if random_number <= 3:
                c.send('Transaction error. Please try again.'.encode())
            else:
                if command == '3':
                    new_amt = int(account_info[2]) + int(amount)
                    account_info[2] = str(new_amt)
                    c.send(f'New balance is {account_info[2]}'.encode())
                elif command == '2':
                    cur_balance = int(account_info[2])
                    if cur_balance >= int(amount):
                        cur_balance -= int(amount)
                        account_info[2] = str(cur_balance)
                        c.send(f'New balance is {account_info[2]}'.encode())
                    else:
                        c.send('Insufficient Balance'.encode())

                if transaction_count == 10:
                    c.send('10 Succsessfil transaction done')
                    break
        else:
            c.send('Enter a valid command'.encode())

else:
    c.send("Incorrect account name/pin\n".encode())

c.close()
```

7 Experience

1. Developed a TCP protocol enabling communication between client and server.
2. Explored the seamless data exchange between client and server components.
3. EImplemented algorithms on the server to determine whether the received number is prime and/or a palindrome.
4. Developed features allowing users to check balances, withdraw money, and deposit funds.
5. Delved into socket programming intricacies, understanding the communication flow between client and server.
6. We learnt a lot about socket ,TCP/UDP protocall,Python

References

[1] Computer networking : A top-down approach 8th ed.

[2] GeeksforGeeks : `https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c`

[3] Scaler: `https://www.scaler.com/topics/tcp-server-client-implementation`

[4] Educative: `https://www.educative.io/answers/what-are-sockets-in-computer-networks`

[5] Tutorialspoint: `https://www.tutorialspoint.com/what-is-a-network-socket-computer-networks`

[6] TeachTarget: `https://www.techtarget.com/searchnetworking/definition/port-number`

[7] Cloudflare: `https://www.cloudflare.com/en-ca/learning/network-layer/what-is-a-computer-port/`