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:

Diptajoy Mistry

Roll No : 34

Md. Mushiur Rahman

Roll No : 58

Submitted On :

January 24, 2023

Submitted To :

Dr. Md. Abdur Razzaque

Dr. Muhammad Ibrahim

Md. Fahim Arefin

1 Introduction

This lab report describes the exercises performed on simple client-server communication using socket programming. Socket programming is a method of creating a communication link between two devices over a network using sockets. The exercises in this lab report focused on creating a simple client-server application using the Python programming language.

1.1 Objectives

- To understand the principles of network communication and how data is transmitted over a network.
- To learn how to create a client-server application using socket programming and the basics of programming with sockets.
- To learn how to use socket programming to create applications that can connect to different types of networks and protocols, such as TCP, UDP, IPv4, and IPv6.
- To learn how to use socket programming as a way to implement network security by implementing secure communication protocols such as SSL or TLS.
- To understand the use of socket programming in various real-world applications such as chat applications, file transfer, remote access, and more.
- To develop the ability to troubleshoot and debug network communication issues and to have knowledge on how to optimize network communication.
- To have a good understanding of networking concepts and to be able to apply that knowledge to create different types of networked applications.
- To have a fundamental understanding of the low-level details of network communication and use that knowledge to design and implement efficient and reliable networked systems.

2 Theory

Socket programming is a method of creating a communication link between two devices over a network using sockets. Sockets are the endpoints of a bidirectional communication link between two programs running on the network. A socket has an IP address and a port number. The lab report covers exercises that demonstrate the basic principles of socket programming and simple client-server communication using the Python programming language. The exercises involve creating a simple server that listens for incoming connections on a specified port and creating a simple client that connects to the server. In exercise A(i), the server listens to the client and in response, the server returns the incoming message converting it into uppercase letters. In exercise A(ii), the client requests the server to tell whether the sent number is "Prime" or "Not Prime" and the server returns the response by calculating the numbers type as "true" or "false". In exercise B(i), we've implemented a protocol between the ATM and the Bank's centralized server and extended version of exercise B(i), in exercise B(ii), we handled errors in client-server communication. The lab report also covers the concepts of IP addresses and ports, which are used to uniquely identify a specific process running on a device on a network. IP addresses are used to identify a device on the network, while ports are used to identify a specific process running on that device. Overall, the lab report provides a hands-on introduction to the basics of socket programming and simple client-server communication, providing a foundation for further exploration of socket programming and its applica- tions in network communication.

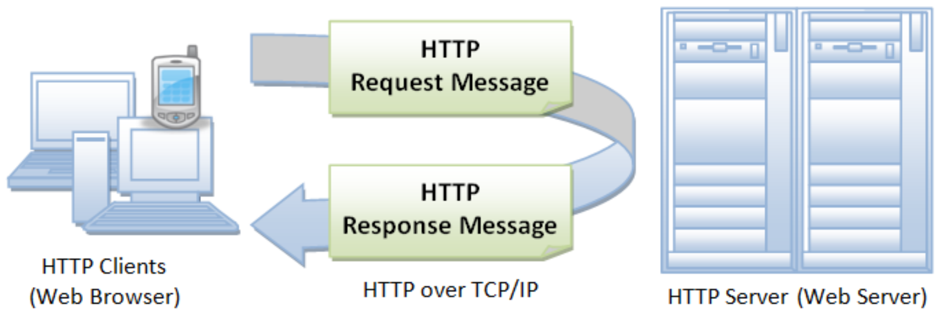


Figure 1: HyperText Transfer Protocol

3 Methodology

3.1 Server

In the server side when we turn it on it will wait for any HTTP request. If it gets any request then it will establish a connection. After setting up the connection, it receives an http query corresponding to which file client requested. Then it will read bytes from that file and send it to the client.

3.2 Client

Here our client side is any web browser. We will enter IP address of our server and the port number. Then an HTTP request will be sent from our browser to the server. We will see links to some files that the server provided. If we click any of them we can download them in our pc.

3.3 server for capitalization,prime and palindrome check

Listing 1: Server Code

```python


import socket
import math

HOST_IP = '192.168.0.194'
HOST_PORT = 50004
ENCODER = "utf-8"
BYTESIZE = 1024


def capitalize_text(text):
    return text.upper()

def check_prime(num):
    if num < 2:
        return False
    for i in range(2, int(math.sqrt(num)) + 1):
        if num % i == 0:
            return False
    return True

def check_palindrome(string):
    return string == string[::-1]
```

```

# Create a server socket, bind it to an IP/port, and listen
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((HOST_IP, HOST_PORT))
server_socket.listen()

# Accept any incoming connection and let them know they are connected
print("Server is running... \n")
client_socket, client_address = server_socket.accept()

# Send a welcome message to the connected client
client_socket.send("You are connected to the server...".encode(ENCODER))

while True:
    # Receive information from the client
    message = client_socket.recv(BYTESIZE).decode(ENCODER)

    # Quit if the client socket wants to quit, else display the message
    if message.lower() == "quit":
        client_socket.send("quit".encode(ENCODER))
        print("\nEnding the chat... goodbye!")
        break
    elif message.startswith('CHECK'):
        _, num, operation = message.split(' ')
        num = int(num)
        if operation == 'prime':
            response = str(check_prime(num))
        elif operation == 'palindrome':
            response = str(check_palindrome(str(num)))
        else:
            response = 'Invalid operation'
        client_socket.send(response.encode(ENCODER))
    elif message.startswith('CAPITALIZE'):
        text_to_capitalize = message.split(' ', 1)[1]
        response = capitalize_text(text_to_capitalize)
        client_socket.send(response.encode(ENCODER))
    else:
        print(f"\n{message}")
        user_input = message
        client_socket.send(user_input.encode(ENCODER))

client_socket.close()
```

3.4 Client for capitalization,prime and palindrome check

Listing 2: Client Code

```
import socket

DEST_IP = '192.168.0.194'
DEST_PORT = 50008
ENCODER = "utf-8"
BYTESIZE = 1024

# Create a client socket and connect to the server
client_socket = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
client_socket.connect((DEST_IP, DEST_PORT))
#It creates a new socket (client_socket) using IPv4 (
    socket.AF_INET) and
# TCP (socket.SOCK_STREAM). It then connects to the
    server using the specified IP address and port.

while True:
    # Receive information from the server
    message = client_socket.recv(BYTESIZE).decode(ENCODER
        )

    # Quit if the connected server wants to quit, else
        keep sending messages
    if message.lower() == "quit":
        client_socket.send("quit".encode(ENCODER))
        print("\nEnding the chat... goodbye!")
        break
    else:
        print(f"\n{message}")
        user_input = input("Message: ")
        client_socket.send(user_input.encode(ENCODER))

client_socket.close()


# Message: Hello
```

```


#Message: CAPITALIZE Make it


#Message: CHECK 23 prime


#Message: CHECK 121 palindrome


#Message: quit
```

3.5 Server Code for ATM machine with error handling

Listing 3: Server Code

```python
import random
import socket

HOST_IP = '192.168.0.194'
HOST_PORT = 50001
ENCODER = "utf-8"
BYTESIZE = 1024
DEFAULT_PIN = '1234'  # Set a default PIN
DEFAULT_USERNAME = 'user123'  # Set a default username

server_socket = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
server_socket.bind((HOST_IP, HOST_PORT))
server_socket.listen()

print("Server is running... \n")
client_socket, client_address = server_socket.accept()

# Set a username and PIN for the server
server_username = DEFAULT_USERNAME
server_pin = DEFAULT_PIN

# Send a message to the client to provide username and
    PIN
client_socket.send("provide_credentials".encode(ENCODER))

while True:
    # Receive username and PIN from the client
    credentials = client_socket.recv(BYTESIZE).decode(ENCODER)
    entered_username, entered_pin = credentials.split()

    if entered_username == server_username and entered_pin == server_pin:
        client_socket.send("Credentials verified. You can now perform transactions.".encode(ENCODER))
        break
    else:
        client_socket.send("Invalid credentials. Please try again.".encode(ENCODER))

balance = 50000

while True:
    op = client_socket.recv(BYTESIZE).decode(ENCODER)
    print('Requested operation: ', op)

    error = random.randint(1, 10)

    if op == "quit":
        client_socket.send("quit".encode(ENCODER))
        print("\nThanks for taking our service.")
        break
    elif error <= 5:
        print('Error Generated!!\n')
        client_socket.send("Error".encode(ENCODER))
        break
    else:
        amount = client_socket.recv(BYTESIZE).decode(ENCODER)
        print('Amount: ', amount)
        if op == 'wd':
            if balance < int(amount):
                client_socket.send("You have insufficient funds!!".encode(ENCODER))
                print('Insufficient fund responded')
            else:
                balance -= int(amount)
                response = "Amount withdrawn: " + str(amount) + "\nBalance: " + str(balance)
                client_socket.send(response.encode(
                        ENCODER))
                print('Successful withdrawal responded')
        elif op == 'dp':
            balance += int(amount)
            response = "Amount deposited: " + str(amount)
                + "\nBalance: " + str(balance)
            client_socket.send(response.encode(ENCODER))
            print('Successful deposition responded')

server_socket.close()
```

3.6 Client Code for ATM machine with error handling

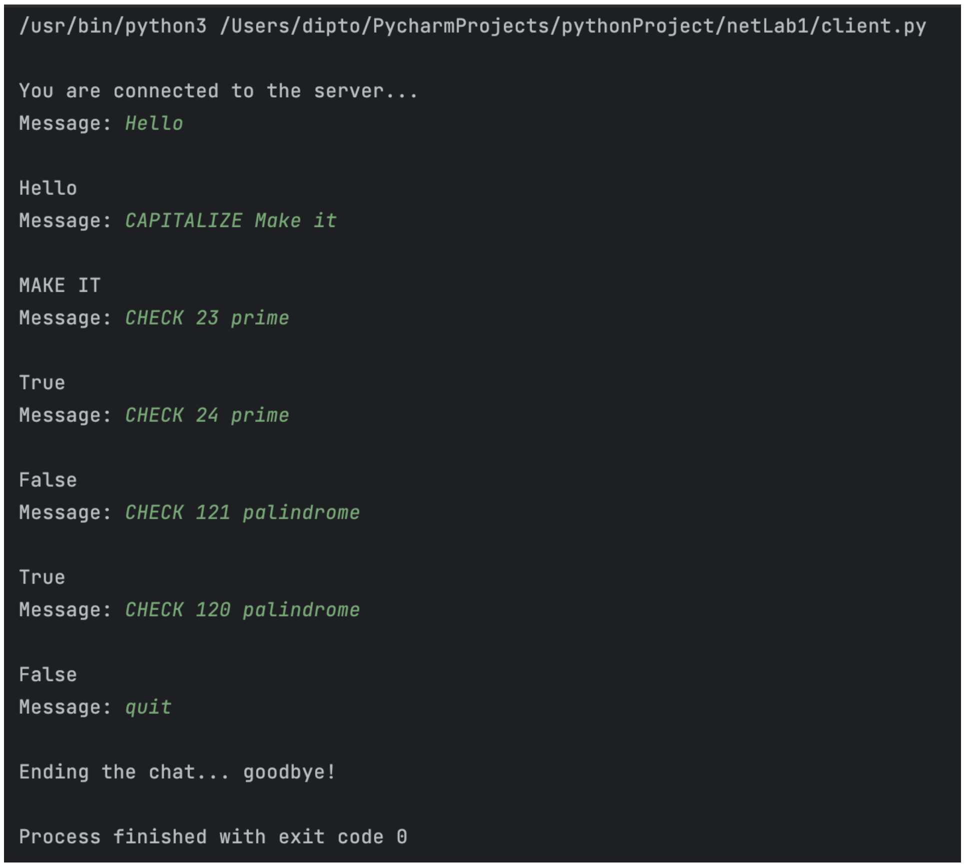
Listing 4: Client Code

```
import socket

DEST_IP = '192.168.0.194'
DEST_PORT = 50001
ENCODER = "utf-8"
BYTESIZE = 1024

# Create a client socket and connect to the server
client_socket = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
client_socket.connect((DEST_IP, DEST_PORT))

while True:
    # Receive information from the server
    message = client_socket.recv(BYTESIZE).decode(ENCODER
        )

    if message == "provide_credentials" or message == "
        Invalid credentials. Please try again.":
        # If the server requests credentials, get
            username and PIN from the user
        if message == "Invalid credentials. Please try
            again.":
            credentials_input = input("Invalid
                credentials. Try again. \nEnter username
                and PIN separated by space: ")
        else:
            credentials_input = input("Enter username and
                PIN separated by space: ")
```

```python
        client_socket.send(credentials_input.encode(ENCODER))

    elif message == "Credentials verified. You can now perform transactions.":
        print(f"\n{message}")
        while True:
            # Add operation input (withdrawal/deposit)
            operation_input = input("Enter operation (wd/dp or quit): ")

            client_socket.send(operation_input.encode(ENCODER))

            if operation_input.lower() == 'quit':
                break

            user_input = input("Enter amount: ")
            client_socket.send(user_input.encode(ENCODER))

            message = client_socket.recv(BYTESIZE).decode(ENCODER)
            print(f"\n{message}")
            if message == "Error":
                break
            else:

                # Check if the user wants to perform another operation
                another_operation = input("Do you want to perform another operation? (yes/no): ")
                if another_operation.lower() != 'yes':
                    client_socket.send("quit".encode(ENCODER))
                    break

# Close the client socket
client_socket.close()
```

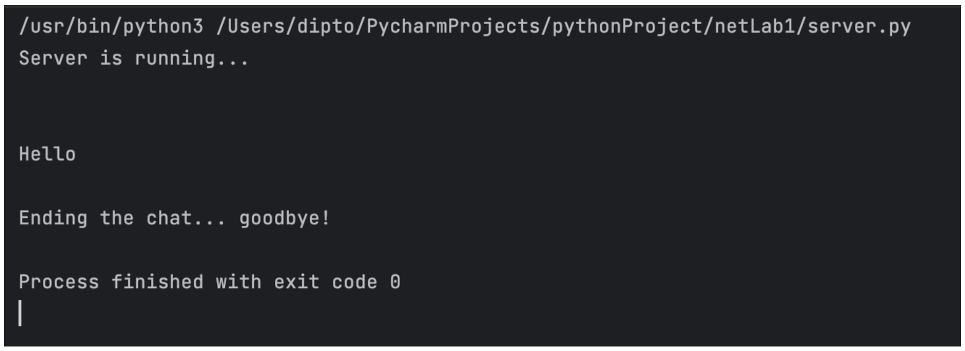
4 Experimental result

Some Snapshots of the Client Side queries can be seen in the following figures:



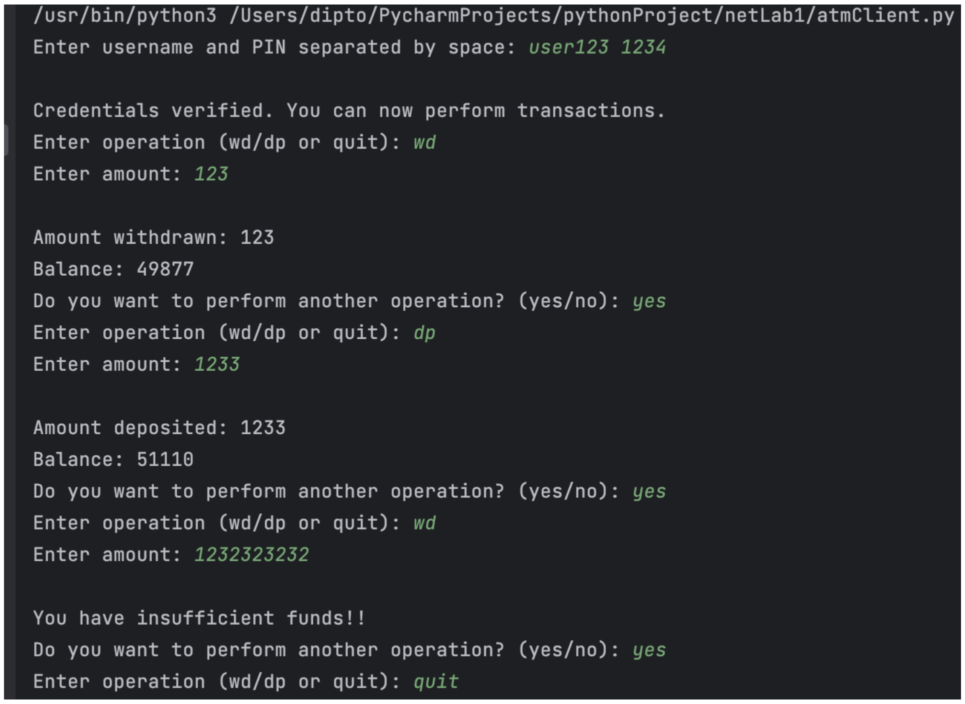
```
/usr/bin/python3 /Users/dipto/PycharmProjects/pythonProject/netLab1/client.py

You are connected to the server...
Message: Hello

Hello
Message: CAPITALIZE Make it

MAKE IT
Message: CHECK 23 prime

True
Message: CHECK 24 prime

False
Message: CHECK 121 palindrome

True
Message: CHECK 120 palindrome

False
Message: quit

Ending the chat... goodbye!

Process finished with exit code 0
```

Figure 2: Capitalize,prime and palindrome check(client)



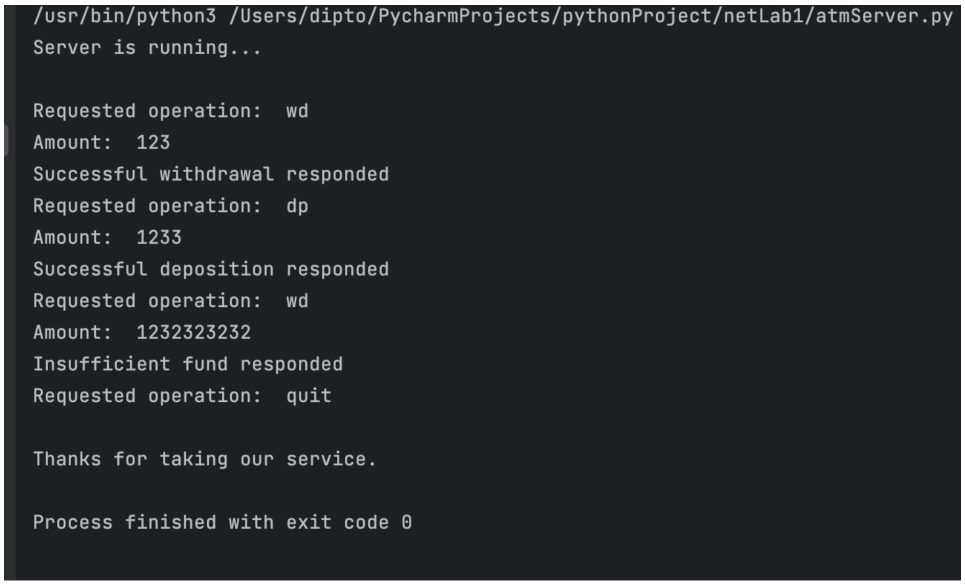
```
/usr/bin/python3 /Users/dipto/PycharmProjects/pythonProject/netLab1/server.py
Server is running...


Hello

Ending the chat... goodbye!

Process finished with exit code 0
```

Figure 3: Capitalize,prime and palindrome check(server)



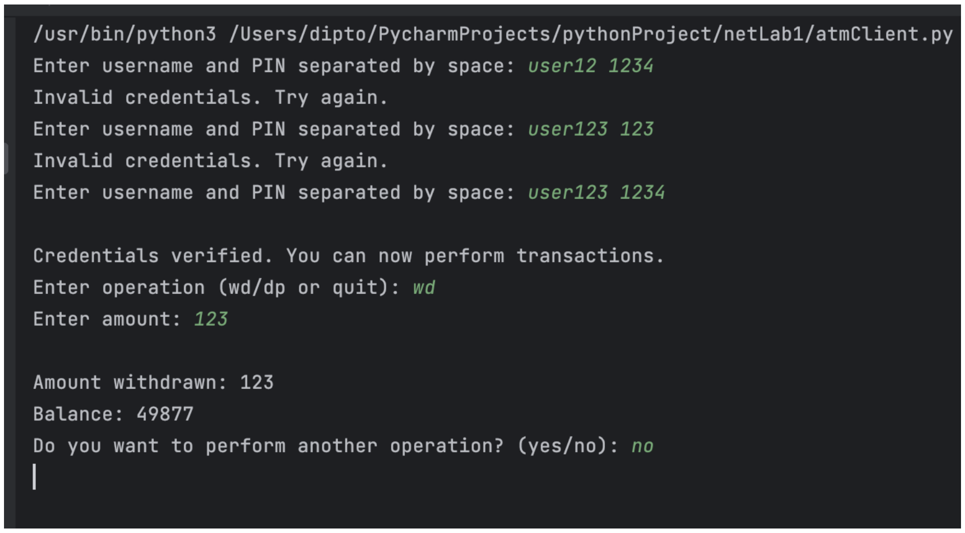
```
/usr/bin/python3 /Users/dipto/PycharmProjects/pythonProject/netLab1/atmClient.py
Enter username and PIN separated by space: user123 1234

Credentials verified. You can now perform transactions.
Enter operation (wd/dp or quit): wd
Enter amount: 123

Amount withdrawn: 123
Balance: 49877
Do you want to perform another operation? (yes/no): yes
Enter operation (wd/dp or quit): dp
Enter amount: 1233

Amount deposited: 1233
Balance: 51110
Do you want to perform another operation? (yes/no): yes
Enter operation (wd/dp or quit): wd
Enter amount: 1232323232

You have insufficient funds!!
Do you want to perform another operation? (yes/no): yes
Enter operation (wd/dp or quit): quit
```

Figure 4: ATM(client)



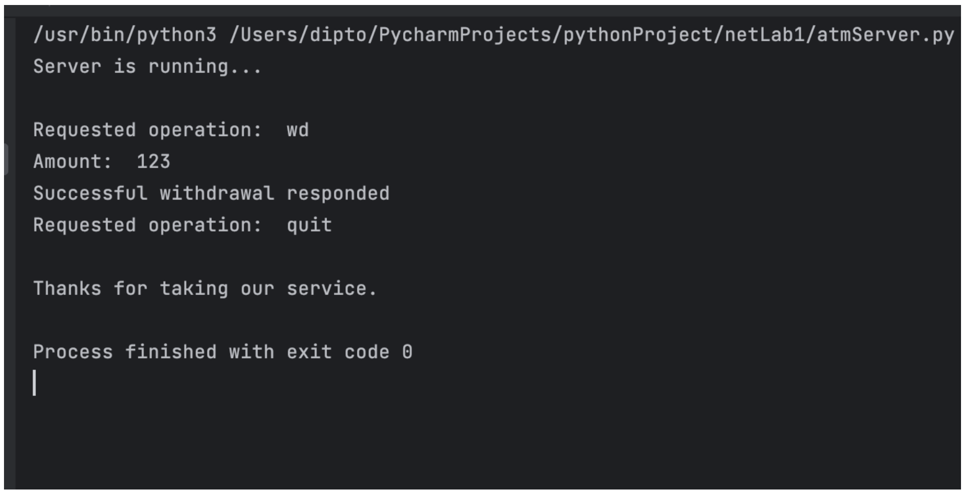
```
/usr/bin/python3 /Users/dipto/PycharmProjects/pythonProject/netLab1/atmServer.py
Server is running...

Requested operation:  wd
Amount:  123
Successful withdrawal responded
Requested operation:  dp
Amount:  1233
Successful deposition responded
Requested operation:  wd
Amount:  1232323232
Insufficient fund responded
Requested operation:  quit

Thanks for taking our service.

Process finished with exit code 0
```

Figure 5: ATM(server)



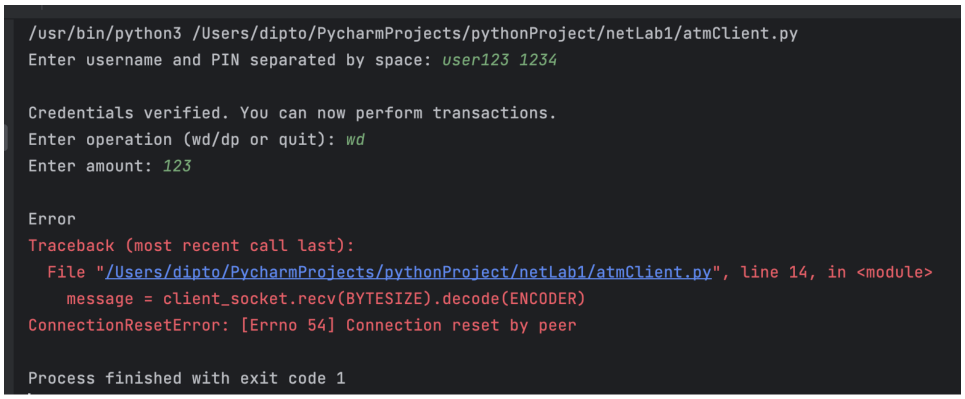
```
/usr/bin/python3 /Users/dipto/PycharmProjects/pythonProject/netLab1/atmClient.py
Enter username and PIN separated by space: user12 1234
Invalid credentials. Try again.
Enter username and PIN separated by space: user123 123
Invalid credentials. Try again.
Enter username and PIN separated by space: user123 1234

Credentials verified. You can now perform transactions.
Enter operation (wd/dp or quit): wd
Enter amount: 123

Amount withdrawn: 123
Balance: 49877
Do you want to perform another operation? (yes/no): no
```

Figure 6: ATM(client)



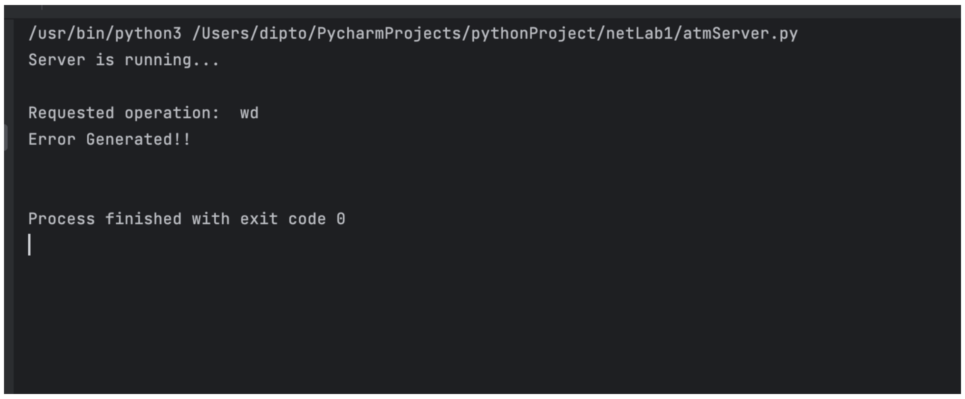
```
/usr/bin/python3 /Users/dipto/PycharmProjects/pythonProject/netLab1/atmServer.py
Server is running...

Requested operation:  wd
Amount:  123
Successful withdrawal responded
Requested operation:  quit

Thanks for taking our service.

Process finished with exit code 0
```

Figure 7: ATM(server)



```
/usr/bin/python3 /Users/dipto/PycharmProjects/pythonProject/netLab1/atmClient.py
Enter username and PIN separated by space: user123 1234

Credentials verified. You can now perform transactions.
Enter operation (wd/dp or quit): wd
Enter amount: 123

Error
Traceback (most recent call last):
  File "/Users/dipto/PycharmProjects/pythonProject/netLab1/atmClient.py", line 14, in <module>
    message = client_socket.recv(BYTESIZE).decode(ENCODER)
ConnectionResetError: [Errno 54] Connection reset by peer

Process finished with exit code 1
```

Figure 8: ATM error(client)



```
/usr/bin/python3 /Users/dipto/PycharmProjects/pythonProject/netLab1/atmServer.py
Server is running...

Requested operation:  wd
Error Generated!!


Process finished with exit code 0
```

Figure 9: ATM error(server)

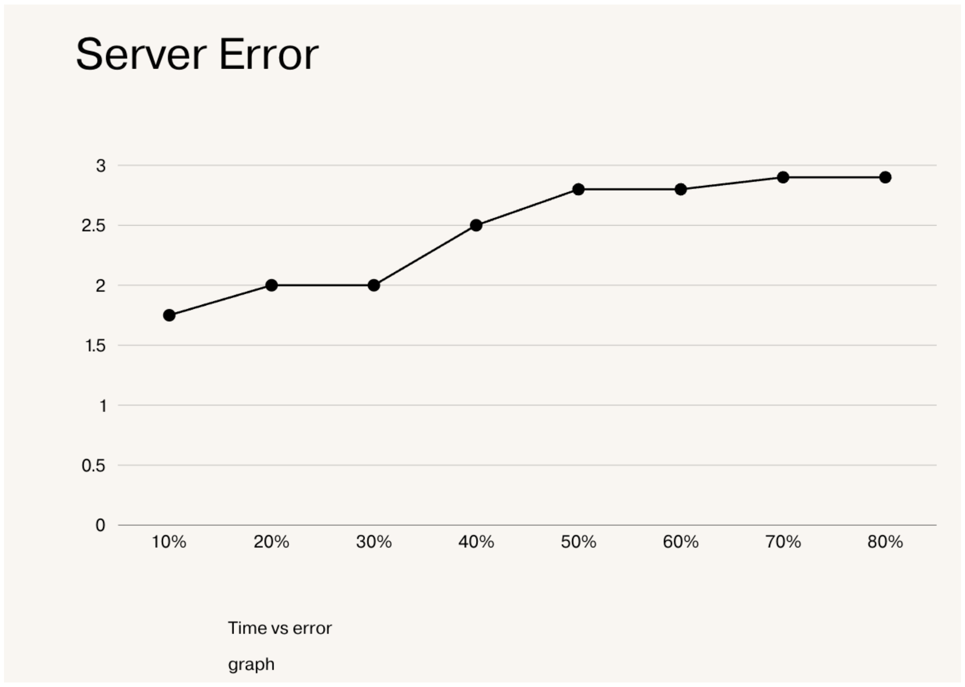


Figure 10: Time vs Error Percentage graph

5 Experience

1. Hands-on experience in writing Python code for creating a simple server and client.
2. Understanding of the basics of network communication and how data is transmitted over a network using sockets.
3. Knowledge on how to use socket programming as a tool for creating distributed systems, where multiple devices work together to perform a task.
4. Understanding of the use of socket programming in various real-world applications such as chat applications, file transfer, remote access, and more.
5. Experience in troubleshooting and debugging network communication issues and knowledge of how to optimize network communication.
6. Understanding of the low-level details of network communication and the ability to design and implement efficient and reliable networked systems.
7. Familiarity with the concepts of IP addresses and ports, and how they are used to uniquely identify a specific process running on a device on a network.

References

[1] geeksforgeeks: `https://www.geeksforgeeks.org/socket-programming-python/`