



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 : Abdullah Al Mahmud

Roll No : 15

Name : Zisan Mahmud

Roll No : 23

Submitted On:

January 25, 2024

Submitted To:

Dr. Md. Abdur Razzaque

Dr. Md Mamunur Rashid

Dr. Muhammad Ibrahim

Mr. Md. Redwan Ahmed Rizvee

1 Introduction

In this lab, the main goal is to become acquainted with socket programming, a fundamental aspect of network communication. Within this controlled environment, the central aim is to gain a nuanced understanding of the principles and practices associated with establishing connections between clients and servers.

2 Objectives

- We have designed and implemented a client-server communication system
- Test the reliability and efficiency of the implemented client-server communication system, to analyze network traffic, identify potential bottlenecks, and ensure seamless data exchange.
- Evaluate the performance metrics of the client-server communication system, including error rate.

3 Theory

A socket is a mechanism for allowing communication between processes, for example, programs running on the same machine or different computers connected on a network. Internet sockets offer an operating system-managed programming access to the network protocol stack. Using Internet Sockets, it is easy to identify a host with an IP address and port number. An IP address is the unique address that identifies a machine uniquely. The port number defines the application the sender wants to connect to that machine. Port numbers from 0 to 1023 are reserved for special purposes. So, we can use port number greater than 1023 till the maximum available port.

Fundamentally, a socket may be divided into two categories: the client socket, which starts communication with the server, and the server socket, which waits for and accepts incoming client connections. In this client-server model, both client and server participate in bidirectional communication. This interaction is facilitated by the use of unique port numbers and IP addresses, enabling precise identification of endpoints.

In summary, this theoretical framework, combined with practical exercises, lays the groundwork for building robust networked applications, fostering a deeper comprehension of socket programming in the broader context of computer networking.

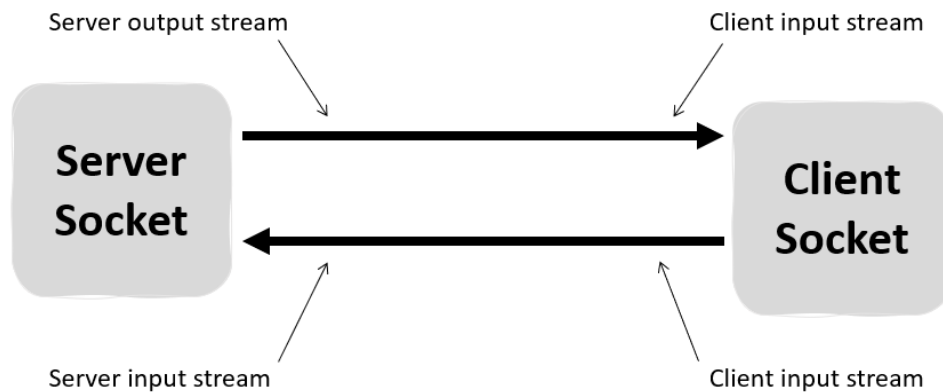


Figure 1: Client Server communication

4 Methodology

4.1 Server

A server is a program that runs on a machine providing services to the clients. When a client requests for a service, the server receives the request and perform the desired task, but it never initiates the service.

In our first task, we have implemented a server that takes connection requests from a client. If the connection is established, then the server receives a request to perform some tasks from the client, here the queries are to convert a line of text from capital letters to small letters, to check an integer if it is prime or palindrome based on the requested operation. Server get these requests and sends a response to the client. If the operation is to convert small to capital letters, the server receives a line of text as a string from the client, converts it to a small letter using `lower()` function, and send it to the client through the socket. If the operation is to check if a

number is prime or palindrome, the server gets "integer prime" or "integer palindrome" and based on the operation using a dedicated function finds if the number is prime or palindrome and send a response to the client.

In our second task we have implemented an ATM machine that simulates communication between bank server and client. Here the server first is requested to set up a connection. If the connection is established, then the server accepts requests from the client. First the server authenticates the client using the id, if not matched then the client is rejected to request any queries. When a valid client requests for balance check, money withdrawal, or deposit money, then the server performs the operations and send response back. We have also implemented error handling for the server operation. Any type of invalid request will be rejected by the server. All transaction is atomic, that is the change in information is done both in the client side and server side. If any error occurs during transaction, then the server revokes the operation and sends response to the client that the operation had not been performed by the server. If same requests are made consecutively, then the server also shows error and revokes the operation.

As the communication is tested within small distance and TCP/IP protocol is working behind the model, there is approximately no chance of errors occurring. So we have integrated some errors manually and tested the performance of the server.

4.2 Client

Client is a user interface or application that initiates requests for services or resources from a server. Here our client is any web browser that sends requests to the server with the provided IP address of the server and the port number. In our first task, we implemented a client program that sends connection requests to the server. When the connection is established, the client requests the server to convert a line of text from capital letter to small letter or to check a number is prime or not, and gets a response from the server. In our second task, we implemented a client that sends queries to the bank server through ATM machine to check the account balance, withdraw money or to deposit money through a valid account. In both the task we have included error handling mechanism so that every communication remains atomic.

4.3 Performance statistics

For both our server and client we find out the time required for a successful communication and calculated the error percentages of the communication model.

5 Experimental result

5.1 Problem A

Server:

Server takes a text line from client and convert it to lowercase letter. It also checks if an integer is prime or palindrome.

```
araf@Mahmuds-MacBook-Air Lab2 % python3 Server.py
Server started on port 8080
New client connected: ('127.0.0.1', 50955)
capital to small converting HELOW WORLD
capital to small converting SendINg FrOM cliENT
capital to small converting OK, SeRvER iS woRKING fInE
checking prime 67
checking prime 78
checking prime 567
checking palindrome 4545
checking palindrome 9009
checking palindrome 788887
█
```

Figure 2: Server

Client:

The client takes input from the user(text line or a number with desired operation) and sends it to the server. It also shows the response from the server.

```
araf@Mahmuds-MacBook-Air Lab2 % python3 Client.py
Valid operations:
  1.String (to lowercase the string)
  2.Integer Operation(to check prime or palindrome.Operation can be either 'prime' or 'palindrome')

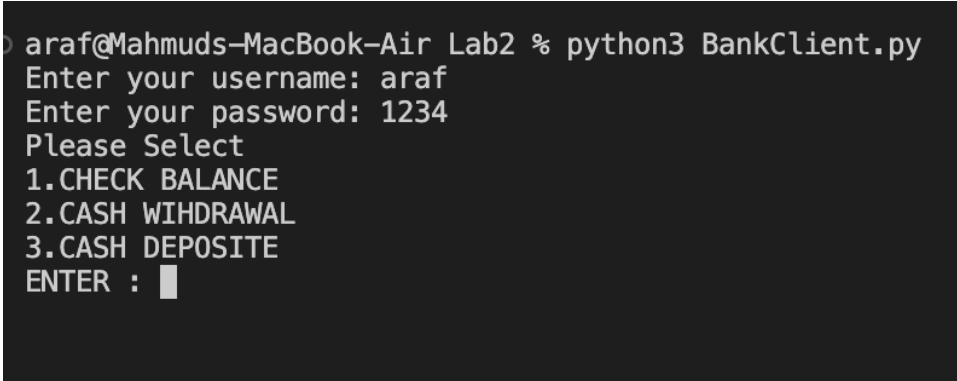
HELOW WORLD
helow world
SendIng FrOm cliENT
sending from client
OK, SeRvER iS woRKING fInE
ok, server is working fine
67 prime
Yes
78 prime
No
567 prime
No
4545 palindrome
No
9009 palindrome
Yes
788887 palindrome
Yes
█
```

Figure 3: Client

5.2 Problem B

Authorizing user:

User enters their credentials in the client side and it is then verified in the server side.

A terminal window with a dark background and light gray text. The prompt is 'araf@Mahmuds-MacBook-Air Lab2 %'. The user has run 'python3 BankClient.py'. The program prompts for a username, which is 'araf', and a password, which is '1234'. It then displays a menu with three options: '1.CHECK BALANCE', '2.CASH WIHDRAWAL', and '3.CASH DEPOSITE'. The prompt 'ENTER : ' is followed by a cursor.

```
araf@Mahmuds-MacBook-Air Lab2 % python3 BankClient.py
Enter your username: araf
Enter your password: 1234
Please Select
1.CHECK BALANCE
2.CASH WIHDRAWAL
3.CASH DEPOSITE
ENTER : █
```

Figure 4: Authorizing user

Checking balance:

Client request for current balance of the user and the server returns the account information.

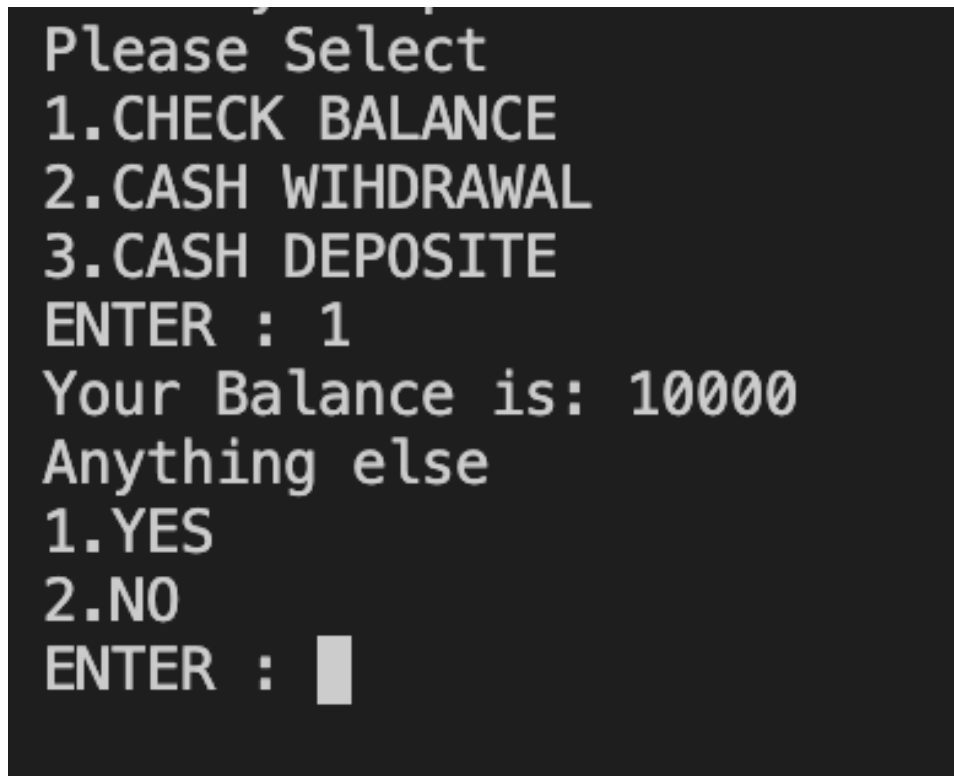


Figure 5: Checking balance

Withdrawing Money:

A user can withdraw money after selecting option 2 and giving an amount.
Successful withdrawn is shown below:

```
Please Select
1.CHECK BALANCE
2.CASH WIHDRAWAL
3.CASH DEPOSITE
ENTER : 2
Enter amount :4000
4000
Withdrawn successful. Your Balance is: 6000
Anything else
1.YES
2.NO
ENTER : █
```

Figure 6: Successful withdraw

Transition Failure Transition can fail if the random number generator value is of greater than 70. That is there is 30% chance of transition failure. In this case the user can again try for a successful transition.

```
ENTER : █  
Please Select  
1.CHECK BALANCE  
2.CASH WIHDRAWAL  
3.CASH DEPOSITE  
ENTER : 2  
Enter amount :500  
500  
555  
Transaction Failed  
Try Again?  
1.YES  
2.NO  
ENTER : █
```

Figure 7: Transition fail

Try again: Until the transition is successful which has a 70% chance, try again prompt will appear. One can cancel requesting for the failed transition by selecting No.

```
Please Select
1.CHECK BALANCE
2.CASH WIHDRAWAL
3.CASH DEPOSITE
ENTER : 2
Enter amount :500
500
555
Transaction Failed
Try Again?
1.YES
2.NO
ENTER : 1
555
Transaction Failed
Try Again?
1.YES
2.NO
ENTER : 1
Withdrawn successful. Your Balance is: 9500
Anything else
1.YES
2.NO
ENTER : █
```

Figure 8: Try Again after Transition failure

Insufficient Balance: In case of money withdraw, if the given amount is greater than the user's balance. Transition will not happen and user will receive an Insufficient Balance verdict.

```
Please Select
1.CHECK BALANCE
2.CASH WIHDRAWAL
3.CASH DEPOSITE
ENTER : 2
Enter amount :13000
13000
501
Insufficient Balance
Anything else
1.YES
2.NO
ENTER : █
```

Figure 9: Transition failed after giving withdraw amount more than balance

Giving Negative Amount: In case of giving negative amount on both withdraw and deposit transition will not happen.

```
Please Select
1.CHECK BALANCE
2.CASH WIHDRAWAL
3.CASH DEPOSITE
ENTER : 2
Enter amount :-700
-700
Invalid amount
Anything else
1.YES
2.NO
ENTER : █
```

Figure 10: Error on giving negative amount

Successful Deposit: the below prompt will be shown on a successful deposit.

```
Please Select
1.CHECK BALANCE
2.CASH WIHDRAWAL
3.CASH DEPOSITE
ENTER : 3
Enter amount :5000
New amount added. Your Balance is: 15000
Anything else
1.YES
2.NO
ENTER : █
```

Figure 11: Successful Deposit

Client side Error: Every transition is marked by a unique ID. If someone tries requesting transitions that are already executed, an Already requested prompt will generate.

```
Please Select
1.CHECK BALANCE
2.CASH WIHDRAWAL
3.CASH DEPOSITE
ENTER : 2
Enter amount :5000
5000
502
Already requested
Anything else
1.YES
2.NO
ENTER : █
```

Figure 12: Error on giving the same Transition that was executed before

Time Delay Graph: The graph below shows the time delay or latency of 100 successful requests on different error percentages. As the percentage of error increases the time to complete 100 random requests also increases gradually.

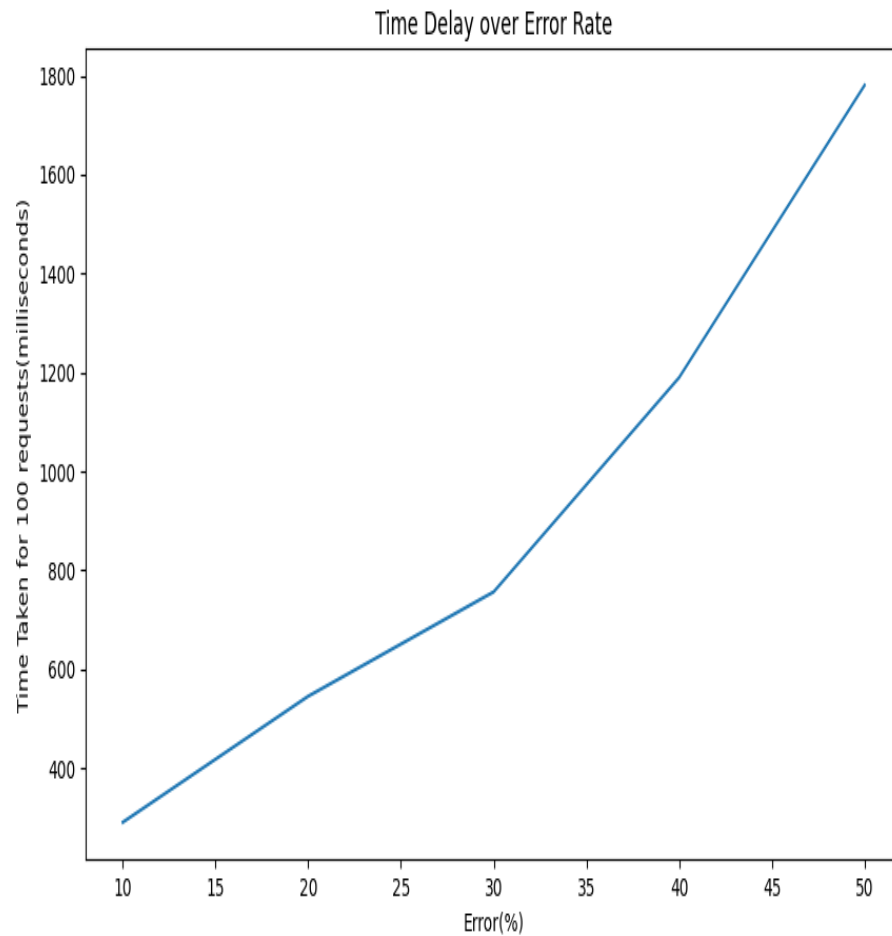


Figure 13: Time Delay Graph

6 Experience

1. We had learned how the socket is established between a client and a server.
2. We had gained hands-on experience in crafting and sending requests from the client to the server for further processing.
3. We encountered some real world problems during this experiment which provide an opportunity to debug and troubleshoot issues.
4. Finally, we gained insights into the complexities involved in managing connections and data flow between clients and servers.

References

- [1] Socket Programming: <https://research.ncl.ac.uk/game/mastersdegree/workshops/networkintroduction/intro.pdf>
- [2] IBM: <https://www.ibm.com/docs/en/i/7.1?topic=communications-socket-programming>
- [3] Client-Server Model: <https://www.geeksforgeeks.org/client-server-model/>
- [4] Socket Programming in Python (Guide): <https://realpython.com/python-sockets/>
- [5] Socket Programming in Python: <https://www.geeksforgeeks.org/socket-programming-python/>

A Code

A.1 Problem A

Establish a TCP connection in between a server process, running on host A and a client process, running on host B and then perform some operation by the server process requested by the client and send responses from the server.

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

Server

```
import socket
from threading import Thread

class Server:
    # Constructor to initialize the server socket using a port
    def __init__(self, port):
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.server.bind(('localhost', port))
        self.server.listen(5) #limiting client connections to 5
        print(f"Server started on port {port}")

    # Function to convert capital to small
    def small(self, sentence):
        return sentence.lower()

    # Function to check if a number is prime or not
    def is_prime(self, a):
        try:
            a = int(a)
        except:
            return "Not a number"
        if a < 2:
            return "No"
        for i in range(2, int(a**0.5) + 1):
            if a % i == 0:
                return "No"
        return "Yes"

    # Function to check if a string is palindrome or not
    def is_palindrome(self, a):
        return "Yes" if a == a[::-1] else "No"

    # Function to start the server
    def start(self):
        try:
            while True:
                client_socket, addr = self.server.accept()
                print(f"New client connected: {addr}")
                #starting a new thread for each client
                Thread(target=self.handle_client, args=(client_socket,)).start()
        except KeyboardInterrupt:
            print("Server stopped by user")
```

```

        finally:
            self.close()

# Function to close the server
def close(self):
    self.server.close()

# Function to handle each client
def handle_client(self, client_socket):
    with client_socket:
        while True:
            data = client_socket.recv(1024)
            if not data:
                client_socket.send("Invalid input".encode())
                continue
            str = data.decode()

            # Checking for the type of operation
            parts = str.split(" ")
            if len(parts) != 2:
                print("capital to small converting", str)
                rstr = self.small(str)
            elif parts[1]=="prime":
                # condition for checking if a number is prime or not
                print("checking prime", parts[0])
                rstr = self.is_prime(parts[0])
            elif parts[1]=="palindrome":
                # condition for checking if a string is palindrome or not
                print("checking palindrome", parts[0])
                rstr = self.is_palindrome(parts[0])
            else:
                # condition for lowercasing a string
                print("capital to small converting", str)
                rstr = self.small(str)

            client_socket.sendall(rstr.encode())

if __name__ == "__main__":
    port = 8080
    server = Server(port)
    server.start()

```

Client

```

import socket
from threading import Thread

class Client:
    # Constructor to initialize the client socket using a server ip and port
    def __init__(self, server_ip, server_port):
        self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.client_socket.connect((server_ip, server_port))

    # Function to send 3 different type of messages to the server
    def send_message(self):
        while True:
            message_to_send = input()
            self.client_socket.sendall(message_to_send.encode())

    # Function to listen for messages from the server
    def listen_for_message(self):
        while True:
            #receiving message size set to 1024 bytes.
            data = self.client_socket.recv(1024)
            if not data:
                break
            print(data.decode())

```

```

# function to start the server both for sending and receiving messages
def start(self):
    Thread(target=self.listen_for_message).start()
    self.send_message()

# Function to close the client
def close(self):
    self.client_socket.close()

if __name__ == "__main__":
    server_ip = "localhost"
    server_port = 8080
    client = Client(server_ip, server_port)
    print("Valid operations:\n 1.String (to lowercase the string)")
    print(" 2.Integer Operation(to check prime or palindrome.)")
    print(" Operation can be either 'prime' or 'palindrome')\n")
    client.start()
    client.close()

```

A.2 Problem B

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.

Server

```

import socket
import random

class BankServer:
    def __init__(self):
        self.users = {
            "araf": {
                "password": "1234",
                "balance": 10000
            },
            "zisan": {
                "password": "1234",
                "balance": 10000
            }
        }
        self.dict = list()
        self.hostname = socket.gethostname()
        self.host = socket.gethostbyname(self.hostname)
        self.port = 4532
        self.server_socket = socket.socket()
        self.server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.server_socket.bind(('localhost', self.port))
        print(f"Server started on port {self.port}")
        self.server_socket.listen(5)
        self.conn, self.address = self.server_socket.accept()
        print("Connection from: " + str(self.address))

    def run(self):
        while True:
            data = self.conn.recv(1024).decode()
            if not data:
                break
            if data in self.users:
                user = data
                data = self.conn.recv(1024).decode()
                if data == self.users[user]['password']:

```

```

self.conn.send(('40').encode())
while True:
    choice = self.conn.recv(1024).decode()
    print('choice', choice)
    if choice == 'close':
        return
    if int(choice) == 1: # choice 1 for checking balance
        print('choice:', choice)
        self.conn.send(('Your Balance is: '
                        + str(self.users[user]['balance']))
                        .encode())
    elif int(choice) == 3: # choice3 for cash deposit
        amnt = self.conn.recv(1024).decode()
        amnt = int(amnt)
        print('deposited amount: ', amnt)
        rand = random.randint(0, 100)
        print('rand', rand)
        if rand > 70:
            self.conn.send(('555').encode())
        else:
            self.users[user]['balance'] += amnt
            self.conn.send(('New amount added.
                            Your Balance is: '
                            + str(self.users[user]['balance']))
                            .encode())
    else:
        # for cash withdrawal
        amnt = self.conn.recv(1024).decode()
        amnt = int(amnt)
        print(amnt)
        id = self.conn.recv(1024).decode()
        print(f"id={id}")
        print('requested withdrawn amount: ', amnt)
        if id in self.dict:
            self.conn.send(('502').encode())
        elif amnt > self.users[user]['balance']:
            # if requested amount is greater than acc balance
            self.conn.send(('501').encode())
        else:
            rand = random.randint(0, 100)
            print('rand', rand)
            if rand > 70:
                self.conn.send(('555').encode())
            else:
                self.dict.append(id)
                self.users[user]['balance'] -= amnt
                self.conn.send(('Withdrawn successful. Your Balance is: '
                                + str(self.users[user]['balance']))
                                .encode())

        print(self.dict)
    else:
        print('Invalid Password') # if password is not match
        self.conn.send(('404').encode())
    else:
        print('Invalid User') # if user id is not match
        self.conn.send(('404').encode())
self.conn.close() # close the connection

server = BankServer()
server.run()

```

Client

```
import socket
import time
import random

class BankClient:
    def __init__(self):
        self.host = 'localhost'
        self.port = 4532
        self.client_socket = socket.socket()
        self.client_socket.connect((self.host, self.port))
        self.id = 0

    def send_withdraw_req(self, wit):
        self.client_socket.send(wit.encode())
        time.sleep(1)
        strid = str(self.id)
        self.client_socket.send(strid.encode())

    def run(self):
        us = input('Enter your username: ')
        self.client_socket.send(us.encode())
        pas = input('Enter your password: ')
        self.client_socket.send(pas.encode())
        cc = self.client_socket.recv(1024).decode()
        if cc == '404':
            print('invalid Data')
            self.client_socket.close()
            return
        while True:
            print('Please Select ') # ATM booth menu option
            print('1.CHECK BALANCE')
            print('2.CASH WITHDRAWAL')
            print('3.CASH DEPOSITE')
            choose = input('ENTER : ') # enter desired option
            self.client_socket.send(choose.encode())
            if choose == '1':
                print(self.client_socket.recv(1024).decode())
            elif choose == '3':
                dep = input('Enter amount :') # enter deposit amount
                if int(dep) <= 0:
                    print('Invalid amount')
                else:
                    while True:
                        self.client_socket.send(dep.encode())
                        dd = self.client_socket.recv(1024).decode()
                        print(dd)
                        if dd == "555":
                            print("Transaction Failed")
                            print('Try Again?')
                            print('1.YES')
                            print('2.NO')
                            try_again = input('ENTER : ')
                            if try_again == '1': # press 1 for again go back to the menu
                                self.client_socket.send(('3').encode()) # press 2 for closing connection
                                time.sleep(0.5)
                                continue
                            else:
                                break
                        else:
                            break
                    else:
                        break
            elif choose == '2':
                self.id = random.randint(0, 10)
                wit = input('Enter amount :') # enter withdrawal amount
                print(wit)
                wi = int(wit)
                if wi <= 0:
                    print('Invalid amount')
                else:
                    while True:
                        self.send_withdraw_req(wit) # send request id, withdrawal amount
                        rr = self.client_socket.recv(1024).decode()
                        print(rr)
```

```

        if rr == '501':
            print('Insufficient Balance')
            break
        if rr == '502':
            print('Already requested')
            break
        elif rr == '555':
            print("Transaction Failed")
            print('Try Again?')
            print('1.YES')
            print('2.NO')
            try_again = input('ENTER : ')
            if try_again == '1': # press 1 for again go back to the menu
                self.client_socket.send(('2').encode()) # press 2 for closing connection
                time.sleep(0.5)
                continue
            else:
                break
        else:
            break
    print('Anything else') # after executing one request
    print('1.YES')
    print('2.NO')
    a = input('ENTER : ') # enter the desired option
    if a == '2':
        self.client_socket.send(('close').encode())
        break
    self.client_socket.close() # close the connection

client = BankClient()
client.run()

```