University of Dhaka

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 2 : Introduction to Socket Programming

Submitted By:

Name: Meherun Farzana

Roll No : 05

Name: Mohd. Jamal Uddin Mallick

Roll No : 07

Submitted On :

February 1, 2024

Submitted To :

Dr. Md. Abdur Razzaque

Dr. Md. Mamun Or Rashid

Dr. Muhammad Ibrahim

Redwan Ahmed Rizvee

1 Introduction

Socket programming is a method for establishing a connection between two nodes on a network in order to facilitate communication between them. One socket, also known as a node, is responsible for listening on a specific port at an IP address. The other socket, on the other hand, is used to establish a connection with the first socket. The server establishes the listener socket, while the client initiates a connection to the server. The client-server model is a prevalent architecture employed in several networked systems, such as the World Wide Web, email, and file sharing.

1.1 Objectives

- Attain a thorough understanding of how processes communicate over networks through the utilization of sockets.
- Develop practical expertise in creating, binding, and managing sockets for both client and server applications.
- Enhance skills in handling errors and exceptions during socket communication, and learn effective troubleshooting methods.
- Grasp the fundamental principles of the client-server paradigm and create programs that demonstrate successful interaction between clients and servers.

2 Theory

The theory behind the working principle of socket programming revolves around the concept of communication endpoints, known as sockets, which facilitate data exchange between processes across a network. Sockets function based on a client-server concept, where one entity starts a connection (client) and the other responds (server). Two prominent communication protocols, TCP and UDP, present different trade-offs in terms of dependability and speed. Sockets leverage a collection of functions, including as creation, binding, listening, connecting, sending, and receiving, to build and manage connections between processes. The server-side sockets passively listen for incoming connections, while the client-side sockets actively create connections to servers. Through these principles, socket programming promotes strong and adaptable communication, constituting the backbone of networked applications by allowing data to flow effortlessly across diverse computer units.

3 Methodology

3.1 Steps

1. We used socket programming to create a TCP connection between a server process on host A and a client process on host B. This entails generating sockets, binding them to specified addresses, and beginning the connection.

2. Implemented communication protocols between the client and server processes to make it easier to send and receive requests. This comprises sending operation requests from the client to the server and getting the related responses.

3. Developed functionality within the server process to conduct the specified operation, specifically the conversion of capital letters to tiny letters for a given line of text. This requires processing the text, finding capital letters, and converting them.

4. For problem A, we improved the server's ability to handle queries including an integer and an operation name ('prime' or 'palindrome'). Implement algorithms that identify whether the provided integer is a prime number or a palindrome, as per the defined procedure.

5. For problem B, we set an ATM booth as the client and the bank's main server as the server. We implemented the basic features of such a system that includes: checking balance, correct withdrawal and deposition. We created some error deliberately to display the possible errors that might occur in such a real life system and handled them logically.

6. Ensured that the server returns acceptable responses to the client after completing the specified tasks. Responses should convey the outcomes of letter conversion or the outcome of the prime/palindrome check.

7. In order to handle possible problems that may arise during communication, like network outages or erroneous requests, we provided strong error handling procedures.

8. To ensure that the deployed system is functioning as intended, thoroughly test it. We checked to make sure the server converts letters correctly and performs prime/palindrome operations. It also processes client requests accordingly.

3.2 Problem A

3.2.1 Code : Server

```python
import socket
import sys
import sympy

def create_socket():
    try:
        global host
        global port
        global sock
        host = socket.gethostbyname(socket.gethostname())
        port = 9005
        sock = socket.socket()
    except socket.error as msg:
        print("Socket creation error " + str(msg))

def bind_socket():
    try:
        global host
        global port
        global sock

        print("binding the port "+str(port))
        sock.bind((host, port))
        sock.listen(5)
    except socket.error as msg:
        print("Socket binding error " +
        str(msg) + "\n" + "Retrying...")
        bind_socket()

def isPalindrome(s):
    return s == s[::-1]

def socket_accept():
    conn, address = sock.accept()
    print("Connection has been established! "+ "\nIP : "
    + address[0] + "\nPort : " + str(address[1]))

    while True:
        text = conn.recv(1024).decode()

        if(text[0].isdigit()):
            num, operation = text.split()
            if operation == 'prime':
                if(sympy.isprime(int(num))):
                    conn.send('It is a prime number.'.encode())
                else:
                    conn.send('It is not a prime number.'.encode())

            elif operation == 'palindrome':
                if(isPalindrome(num)):
                    conn.send('It is palindrome.'.encode())
                else:
                    conn.send('It is not palindrome.'.encode())

            else:
                conn.send('Operation not available'.encode())

        else:
            conn.send(text.lower().encode())

    conn.close()

def main():
    create_socket()
    bind_socket()
    socket_accept()

main()
```

3.2.2 Code : Client

```
import socket

HOST = socket.gethostbyname(socket.gethostname())
PORT = 9005
client_socket: socket.socket

def client_init():
    client_socket = socket.socket(socket.AF_INET,
        socket.SOCK_STREAM)
    client_socket.connect((HOST,PORT))

    while True:
        text = input('Enter text or number with an operation:')
        client_socket.send(text.encode())

        print(client_socket.recv(1024).decode())
```

```
client_init()
```

3.3 Problem B

3.3.1 Code : Server

```
import socket
import sys #to implement command line and terminal commands into python file
from users import users
import threading
import random
import uuid
import pickle

ERROR_RATE = 3

conn: socket.socket

def is_float(string):
    if string.replace(".", "").isnumeric():
        return True
    else:
        return False

def create_socket():
    try:
        global host
        global port
        global sock
        host = socket.gethostbyname(socket.gethostname())
        port = 9005
        sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    except socket.error as msg:
        print("Socket creation error " + str(msg))

def bind_socket():
    try:
        global host
        global port
        global sock

        sock.bind((host, port))
        print("binding the port "+str(port))
        sock.listen()
    except socket.error as msg:
        print("Socket binding error " + str(msg) + "\n" + "Retrying...")
        sock.close()
        bind_socket()

def create_transID():
    random_uuid = uuid.uuid4()
    print(f'Transaction ID: {random_uuid}')
    return random_uuid

def socket_accept():
    while True:
        global conn
        conn, address = sock.accept()
        print("Connection has been established! "+
        "\nIP : " + address[0] + "\nAddress : " + str(address[1]))

        while True:
            name = conn.recv(1024).decode()
            password= conn.recv(1024).decode()
            print(name,password)

            logged_user = None
            for user in users:
                if name in user['name'] and password in user['pass']:
                    logged_user = user
                    break

            print(logged_user)
            if logged_user == None:
                conn.send('WRONG_CRED'.encode())
                continue

            conn.send('LOGGED_IN'.encode())
            status = conn.recv(1024).decode()

            if status != 'LOGGED_IN':
                continue

            while True:
                available_options = '''\nSelect one
                1. Check Balance
                2. Withdraw
                3. Deposit'''
                conn.send(available_options.encode())
                option = conn.recv(1024).decode()
                print(f'option 1= {option}')

                if not option.isdigit() or int(option) not in [1,2,3]:
                    conn.send('INVALID_OPTION'.encode())
                    acknowledge = conn.recv(1024).decode()
                    continue
                else:
                    conn.send('VALID_OPTION'.encode())
                    print('VALID')
                    acknowledge = conn.recv(1024)
                    print(acknowledge.decode())

                print(f'option = {option}')

                if option=='1':
                    check_balance(logged_user)

                elif option == '2':
                    withdraw(logged_user)

                elif option == '3':
                    deposit(logged_user)

def check_balance(logged_user):
    print('selected 1')
    print(f'Balance: {logged_user['balance']}')
    conn.send(str(logged_user['balance']).encode())
    ack = conn.recv(1024).decode()

def withdraw(logged_user):
    conn.send('AMOUNT'.encode())
    withdrawal_amount = conn.recv(1024).decode()
    print(f'with_amount = {float(withdrawal_amount)}')

    if not is_float(withdrawal_amount):
        data = {
            'status':'ERROR',
            'message': 'INVALID_IN'
        }
        message = pickle.dumps(data)
        conn.send(message)
        stat = conn.recv(1024).decode()
        print(f'stat = {stat}')
        return
    elif float(withdrawal_amount)>logged_user['balance']:
        data = {
            'status':'ERROR',
            'message': 'NO_BLNC'
        }
        message = pickle.dumps(data)
        conn.send(message)
        stat = conn.recv(1024).decode()
        print(f'stat = {stat}')
        return

    logged_user['balance'] -= float(withdrawal_amount)
    tranXID = f'WITH_{create_transID()}'
    data = {
        'status': 'OK',
        'tid': tranXID,
        'withdraw': float(withdrawal_amount),
        'balance': logged_user['balance']
    }
    message = pickle.dumps(data)
    conn.send(message)
    status = conn.recv(1024).decode()
    print(f'status = {status}')

    if status == 'ERROR_WITHDRAW':
        logged_user['balance'] += float(withdrawal_amount)
        send_msg = 'ROLLED_BACK'
        conn.send(send_msg.encode())
        conn.recv(1024).decode()

    elif status == 'WITHDRAW_RECV':
        print('WITHDRAW RECEIVED')

def deposit(logged_user):
    conn.send('AMOUNT'.encode())
    deposit_amount = conn.recv(1024).decode()
    print(deposit_amount)

    if not is_float(deposit_amount):
        data = {
            'status':'ERROR',
            'message': 'INVALID_IN'
        }
        message = pickle.dumps(data)
        conn.send(message)
        return
    logged_user['balance'] += float(deposit_amount)
    tranXID = f'DEP_{create_transID()}'
    data = {
        'status': 'OK',
        'tid': tranXID,
        'deposit': float(deposit_amount),
        'balance': logged_user['balance']
    }
    message = pickle.dumps(data)
    conn.send(message)
    status = conn.recv(1024).decode()

def main():
    create_socket()
    bind_socket()
    socket_accept()

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

3.3.2 Code : Client

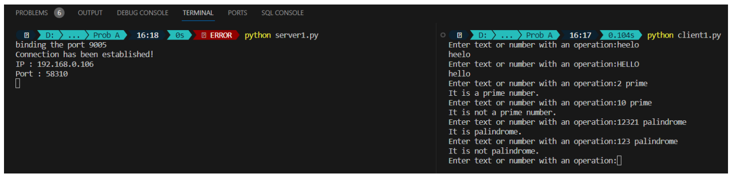
```
import socket
import users
import random
import time
import pickle
import numpy as np
import matplotlib.pyplot as plt

HOST = socket.gethostbyname(socket.gethostname())
PORT = 9005
ERROR_RATE = 0
ITERATIONS = 100
logged_in = False
avg_times = []
times = []
client_socket: socket.socket
count = 0

def authenticate():
    name = input('Enter name: ')
    client_socket.send(name.encode())
    password = input('Enter password: ')
    client_socket.send(password.encode())

    message= client_socket.recv(1024).decode()

    logged_in = message=='LOGGED_IN'

    if logged_in:
        client_socket.send('LOGGED_IN'.encode())

    return message == 'LOGGED_IN'

def select_comm(client_socket):
    options = client_socket.recv(1024).decode()
    print(options)

    selected_comm = input('Enter: ')

    client_socket.send(selected_comm.encode())
    select_response = client_socket.recv(1024).decode()
    print(f'option response={select_response}')

    return select_response=='VALID_OPTION',selected_comm

def respond(client_socket):
    global logged_in
    global count
    global start_time
    global end_time
    selected_comm:int
    iteration = 0

    while iteration<ITERATIONS:
        options = client_socket.recv(1024).decode()
        print(options)
        chance = -1
        while chance<=ERROR_RATE:
            chance = random.randint(1,100)
        print(chance)
        selected_comm = random.randint(1,3)
        print(f'selected option={selected_comm}')
        client_socket.send(f'{selected_comm}'.encode())
        is_valid = True

        select_response = client_socket.recv(1024).decode()
        is_valid= (select_response == 'VALID_OPTION')
        start = time.time()

        if not is_valid:
            print('Invalid option. Try again.')
            client_socket.send('INVALID_OPTION'.encode())
            continue
        else:
            client_socket.send('VALID_OPTION'.encode())
        selected_comm = int(selected_comm)

        match selected_comm:
            case 1:
                balance_check(client_socket)
            case 2:
                withdrawal(client_socket)
            case 3:
                deposit(client_socket)

        end = time.time()
        iteration+=1
        times.append(end-start)
        print(f'Time to send = {end-start}')

def client_init():
    global client_socket
    global ERROR_RATE
    global logged_in
    global avg_times

    client_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    client_socket.connect((HOST,PORT))

    while not logged_in:
        if not authenticate():
            print('Wrong credentials. Try again.')
        else:
            print('Logged in')
            logged_in = True

    while ERROR_RATE<10:
        respond(client_socket)

        time_array = np.array(times)
        mean = time_array.mean()*1000
        print(f'Avg time = {mean} ms')
```

```
            avg_times.append(mean)
            ERROR_RATE+=1
        print(avg_times)

def balance_check(client_socket):
    balance = client_socket.recv(1024).decode()
    client_socket.send('BALANCE'.encode())
    print(f'Balance: {balance}')

def withdrawal(client_socket):
    print(client_socket.recv(1024).decode())
    amount = random.randint(1,5000)
    print(f'with_amount: {amount}')
    client_socket.send(f'{float(amount)}'.encode())
    response_pickle = client_socket.recv(1024)
    response = pickle.loads(response_pickle)

    chance = random.randint(1,10)
    print(chance)
    if chance <= ERROR_RATE:
        client_socket.send('ERROR_WITHDRAW'.encode())
        response = client_socket.recv(1024).decode()
        print(response)
        if response == 'ROLLED_BACK':
            print('An error occurred. Try again')
            client_socket.send('ROLLBACK_RECV'.encode())
            return
    else:
        client_socket.send('WITHDRAW_RECV'.encode())
        if response['status'] == 'ERROR' and response['message'] == 'INVALID_IN':
            print('Not a valid input. Try again.')
            return
        elif response['status'] == 'ERROR' and response['message'] == 'NO_BLNC':
            print('Insufficient balance. Try again. ')
            return
    print(response)
    withdraw=response['withdraw']
    balance = response['balance']

    print(f'{withdraw} withdrawn. Total balance: {balance}')

def deposit(client_socket):
    print(client_socket.recv(1024).decode())
    amount = random.randint(1,5000)
    print(f'dep_amount={amount}')
    client_socket.send(str(amount).encode())
    response_pickle = client_socket.recv(1024)
    response = pickle.loads(response_pickle)

    if response['status'] == 'ERROR' and response['message'] == 'NO_IN':
        print('Not a valid input. Try again.')
        return

    print(response)
    client_socket.send('DEPOSIT_RECV'.encode())
    deposit=response['deposit']
    balance = response['balance']

    print(f'{str(deposit)} deposited. New balance: {balance}')

if __name__=='__main__':
    client_init()

    x = [ i for i in np.arange(start=0,stop=100,step=10)]
    y = avg_times

    plt.plot(x, y, marker='o', linestyle='-')
    plt.title('Average Times Over X')
    plt.xlabel('Error Rate(%)')
    plt.ylabel('Average Times (ms)')
    plt.xticks(np.arange(0, 101, step=10))
    plt.grid(True)

    plt.show()
```

4 Experimental Result

4.1 Problem A

The server receives either a string or an integer from the client. If the received text is a string, it transforms the string into lowercase and returns it to the client. Similarly, if the text is an integer, it examines the integer to determine if it is a prime or palindrome, sending the outcome back to the client. The client gathers input from the user and transmits it to the server. Subsequently, it receives the message from the server and displays it in the console.



```
D: > ... > Prob A  16:18  0s  ERROR  python server1.py
binding the port 9005
Connection has been established!
IP : 192.168.0.106
Port : 58310
```

```
D: > ... > Prob A  16:17  0.104s  python client1.py
Enter text or number with an operation:heelo
heelo
Enter text or number with an operation:HELLO
hello
Enter text or number with an operation:2 prime
It is a prime number.
Enter text or number with an operation:10 prime
It is not a prime number.
Enter text or number with an operation:12321 palindrome
It is palindrome.
Enter text or number with an operation:123 palindrome
It is not palindrome.
Enter text or number with an operation:
```

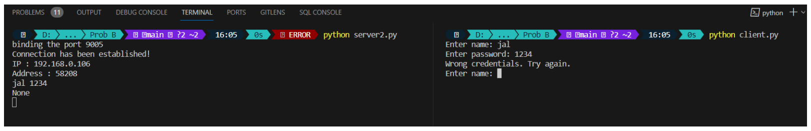
Figure 1: Output for Problem A

4.2 Problem B

It represents the basic scenario of an ATM Booth acting as a client and a bank's main server acting as a server. We also displayed the time consumed by the following process.

4.2.1 Authentication

As there are multiple users, one user can log in with his/her correct credentials. The user acts as the client here. After logging in, the user gets 3 options to choose. Any other option will be considered "INVALID".



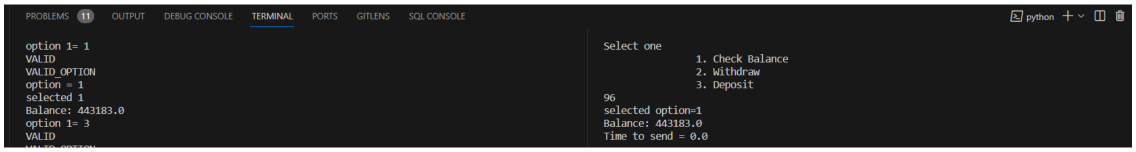
```
D: > ... > Prob B  main ?2 ~2  16:05  0s  ERROR  python server2.py
binding the port 9005
Connection has been established!
IP : 192.168.0.106
Address : 58208
jal 1234
None
```

```
D: > ... > Prob B  main ?2 ~2  16:05  0s  python client.py
Enter name: jal
Enter password: 1234
Wrong credentials. Try again.
Enter name:
```

Figure 2: Authentication

4.2.2 Checking Balance

Checking balance is the first option. The user/client can check balance by choosing 1 (sending 1 to the server). Then the server sends the current balance of that specific user to the client.



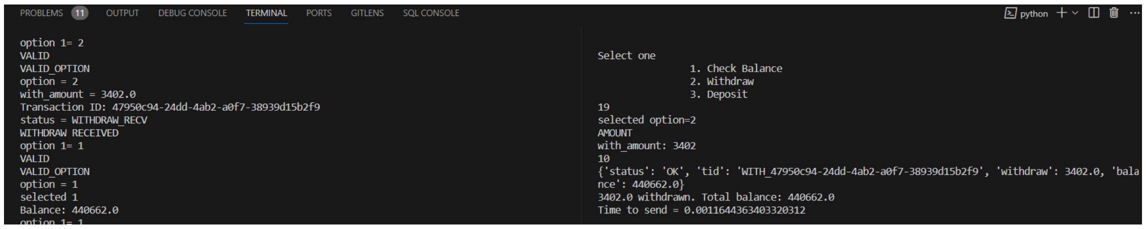
```
option 1= 1
VALID
VALID_OPTION
option = 1
selected 1
Balance: 443183.0
option 1= 3
VALID
```

```
Select one
            1. Check Balance
            2. Withdraw
            3. Deposit
96
selected option=1
Balance: 443183.0
Time to send = 0.0
```

Figure 3: Checking Balance

4.2.3 Withdrawal

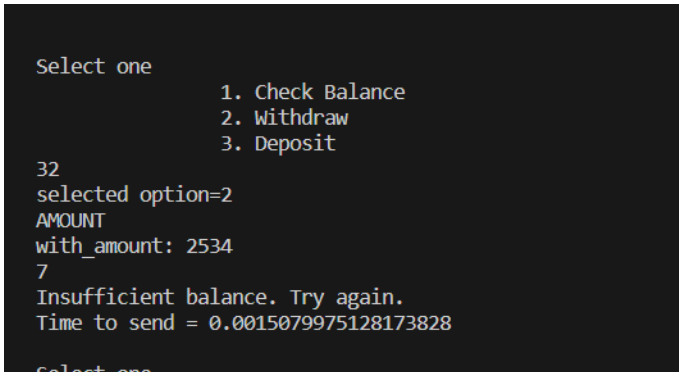
The user can request to withdraw a certain amount not greater than the current balance by choosing option 2. If any amount greater than the current balance is requested for withdrawal, an error message from the server will be sent to the client. After a successful withdrawal, the server will update the current balance according that and inform the client.



```
option 1= 2
VALID
VALID_OPTION
option = 2
with_amount = 3402.0
Transaction ID: 47950c94-24dd-4ab2-a0f7-38939d15b2f9
status = WITHDRAW_RECV
WITHDRAW RECEIVED
option 1= 1
VALID
VALID_OPTION
option = 1
selected 1
Balance: 440662.0
```

```
Select one
                1. Check Balance
                2. Withdraw
                3. Deposit
19
selected option=2
AMOUNT
with_amount: 3402
10
{'status': 'OK', 'tid': 'WITH_47950c94-24dd-4ab2-a0f7-38939d15b2f9', 'withdraw': 3402.0, 'bala
nce': 440662.0}
3402.0 withdrawn. Total balance: 440662.0
Time to send = 0.0011644363403320312
```

Figure 4: Correct withdrawal

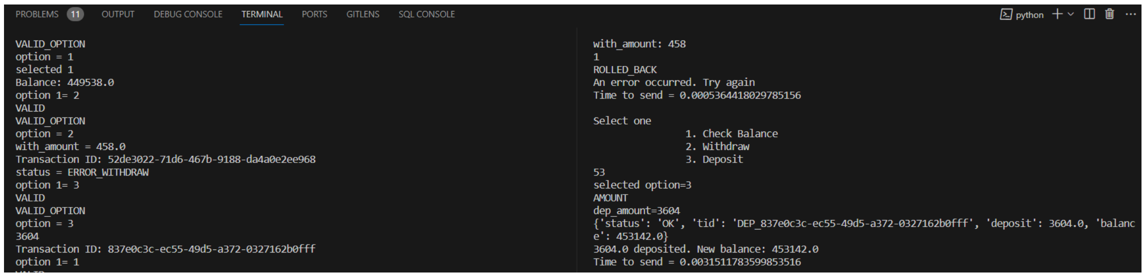


```
Select one
                1. Check Balance
                2. Withdraw
                3. Deposit
32
selected option=2
AMOUNT
with_amount: 2534
7
Insufficient balance. Try again.
Time to send = 0.0015079975128173828
```

Figure 5: Insufficient Balance

4.2.4 Deposit

By choosing option 3, the user can deposit a certain amount of money into his or her account. After a successful deposit, the server will update the current balance according to that and inform the client.



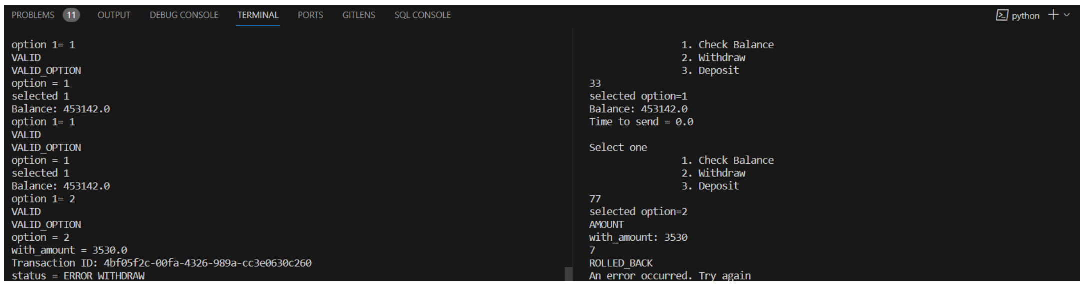
```
VALID_OPTION
option = 1
selected 1
Balance: 449538.0
option 1= 2
VALID
VALID_OPTION
option = 2
with_amount = 458.0
Transaction ID: 52de3022-71d6-467b-9188-da4a0e2ee968
status = ERROR_WITHDRAW
option 1= 3
VALID
VALID_OPTION
option = 3
3604
Transaction ID: 837e0c3c-ec55-49d5-a372-0327162b0fff
option 1= 1
```

```
with_amount: 458
1
ROLLED_BACK
An error occurred. Try again
Time to send = 0.0005364418029785156

Select one
                1. Check Balance
                2. Withdraw
                3. Deposit
53
selected option=3
AMOUNT
dep_amount=3604
{'status': 'OK', 'tid': 'DEP_837e0c3c-ec55-49d5-a372-0327162b0fff', 'deposit': 3604.0, 'balanc
e': 453142.0}
3604.0 deposited. New balance: 453142.0
Time to send = 0.0031511783599853516
```

Figure 6: Deposition

4.2.5 Error Handling : Rollback

To manifest how the bank server and the client would react in case of an erroneous situation, we have implemented a rollback error. A series of random numbers are generated on basis of which sometimes some error will occur. The withdrawal will be successful from the server side, and the current balance will also be updated, but the client won't receive any success messages as well as the money. Hence, the client will request to rollback to the previous state, and the server will do so.



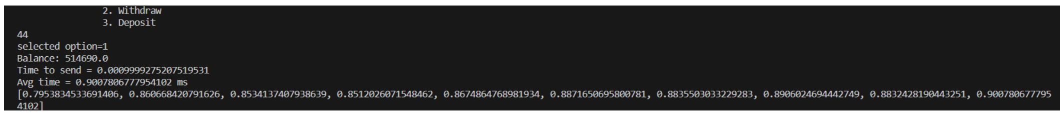
```
option 1= 1
VALID
VALID_OPTION
option = 1
selected 1
Balance: 453142.0
option 1= 1
VALID
VALID_OPTION
option = 1
selected 1
Balance: 453142.0
option 1= 2
VALID
VALID_OPTION
option = 2
with_amount = 3530.0
Transaction ID: 4bf05f2c-00fa-4326-989a-cc3e0630c260
status = ERROR WITHDRAW
```

```
                1. Check Balance
                2. Withdraw
                3. Deposit
33
selected option=1
Balance: 453142.0
Time to send = 0.0

Select one
                1. Check Balance
                2. Withdraw
                3. Deposit
77
selected option=2
AMOUNT
with_amount: 3530
7
ROLLED_BACK
An error occurred. Try again
```

Figure 7: Rollback done when client doesn't receive the withdrawn money

4.2.6 Error handling : Packet Loss

For testing the average time for request handling, 100 requests are sent for each error rate from 0% to 90% with a step of 10. Then, for each case, the time is tracked, and the average time for each error rate is calculated.
When a packet is lost, it is sent again until it is successfully received. This act of repeated sending of a packet increases the response time, eventually increasing the average response time.



```
                2. Withdraw
                3. Deposit
44
selected option=1
Balance: 514690.0
Time to send = 0.0009999275207519531
Avg time = 0.9007806777954102 ms
[0.7953834533691406, 0.860668420791626, 0.8534137407938639, 0.8512026071548462, 0.8674864768981934, 0.8871650695800781, 0.8835503033229283, 0.8906024694442749, 0.8832428190443251, 0.900780677795
4102]
```

Figure 8: Average Time

4.2.7 Graph (Average Time vs Error Rate)

The graph gives a better picture of the relationship between error rate and average response time. It is visible that the average time tends to increase with the increase in error rate, reaching the maximum time of 0.90 ms. In conclusion, error rate has a significant effect on response times.

We plotted and displayed the graph using the **matplotlib** package of Python.

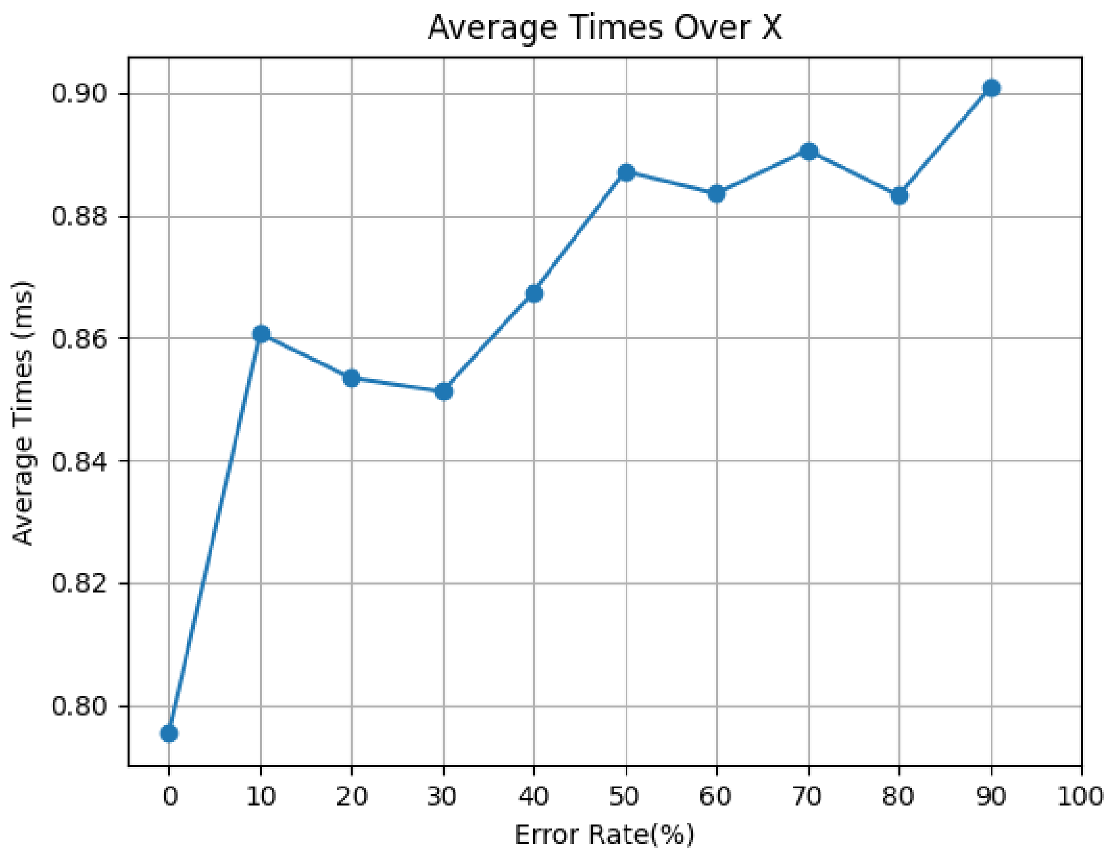


Figure 9: Graph

5 Experience

1. Through the experiment, we acquired an enhanced understanding of the intricate nature of socket connections, data transfer intricacies, and the dynamics of inter-process communication across networks during the course of this experiment.
2. We encountered challenges and were provided with opportunities to refine our troubleshooting and debugging proficiency.
3. The resolution of socket programming issues contributed to the development of adept problem-solving capacities.
4. We were engaged collaboratively in socket programming projects, particularly within a group context. Our effort left us with valuable experience in project management, effective communication, and collaborative coding practices.

References

[1] Socket programming in python. *GeeksforGeeks*, jun 20 2017. [Online; accessed 2023-01-25].

[2] IBM Corporation. Socket programming - ibm. aug 31 2021.

[3] Pankaj. Python socket programming - Server, client example. *DigitalOcean*, aug 3 2022. [Online; accessed 2023-01-25].