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 Machine Operation

Submitted By :

Name: Md. Emon Khan

Roll No : 30

Name: Mahmudul Hasan

Roll No : 60

Submitted To :

Dr. Md. Abdur Razzaque

Dr. Md. Mamun Or Rashid

Dr. Muhammad Ibrahim

Md. Redwan Ahmed Rizvee

Submitted On : February 1, 2024

1 Introduction

The main objective of Lab Experiment-2 is to employ socket programming for the implementation of client-server communication using the TCP protocol. Socket programming involves creating a server socket and specifying a port for communication. The client can then utilize this socket to establish a connection with the server and exchange data.

1.1 Objectives

The specific objectives of the lab experiment include:

- Establishing client-server communication using the TCP protocol using socket programming as the fundamental goal.
- Testing the communication by exchanging data and performing specific operations:
  - Converting a provided string from uppercase to lowercase on the client side.
  - Sending a number to the server and instructing it to determine whether the number is prime or not, or whether it's a palindrome or not.
- Implementing an ATM server, where the client sends a money exchange request.
- Ensuring the ATM server is capable of handling errors effectively.

2 Theory

A socket serves as a software endpoint facilitating the connection between two devices for data exchange. Socket programming involves creating a listener socket (server-side) and a client socket, establishing a connection with the server for communication. This interaction can take place using protocols like TCP or UDP.

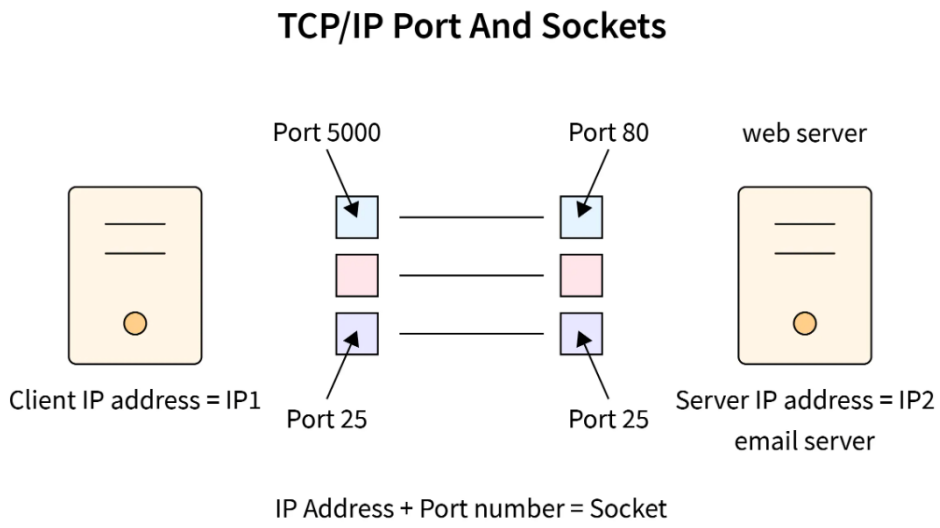


Figure 1: Socket Programming

3 Methodology

3.1 Establishing a TCP connection

3.1.1 Server

On our server side, we initiate the creation of a server socket using the TCP protocol and bind it to a specific host address along with an available port. Subsequently, we set our server in listening mode, awaiting any incoming connection requests. Once a connection request is received, we proceed to establish a connection with the client by accepting the request.

3.1.2 Client

On the client side, we initiate the creation of a client socket, also using the TCP protocol. After this, we send a connection request to the previously established server, using the host address and port.

3.2 Capital to Small Letter Conversion

3.2.1 Server

The server awaits a request from the client. Upon reception, it decodes the expected message, which is anticipated to be a string in uppercase Latin letters. It convert the message to lowercase, encode it, and send the modified message back to the client.

3.2.2 Client

The client transmits a string of uppercase Latin letters to the server after encoding the message, and then awaits a reply. Upon receiving the server's response, the client proceeds to decode the message.

3.3 Prime or Palindrome Test

3.3.1 Server

The server awaits two requests from the client. The initial request is anticipated to be a number, and the following request specifies a particular test, such as a Prime test or a Palindrome test. Upon decoding these requests, the server comprehends the specified operation, sending the number to the appropriate function. Following the function's execution, the server processes the output, and sends the relevant result encoded as a reply to the client.

3.3.2 Client

The client sends two requests to the server. The first request is expected to be a number, and the latter specifies a particular test, such as a Prime test or a Palindrome test. These requests are encoded and transmitted to the server. Then the client awaits a reply. Upon receiving the reply, it proceeds to decode the response for further processing.

3.4 ATM Machine

3.4.1 Server

The server awaits two requests from the client. The initial request is anticipated to be a number, and the following request specifies a particular request, such as a Withdrawal or a Deposition. Upon decoding these requests, the server comprehends the specified operation, sending the number

to the appropriate function. Following the function's execution, the server processes the output, and sends the relevant result encoded as a reply to the client.

3.4.2 Client

The client sends two requests to the server. The first request is expected to be a number, and the latter specifies a particular request, such as a Withdrawal or a Deposition. These requests are encoded and transmitted to the server. Then the client awaits a reply. Upon receiving the reply, it proceeds to decode the response for further processing.

3.5 Error Handling

3.5.1 Server

The server issues a pseudo-error. When it produces an error, it doesn't get the client's request. So the server keeps on listening until it gets the client request. The client on the other hand, calculated the elapsed time from the start of the session, to the server's reply.

3.5.2 Client

The client issues a pseudo-error and halts for some time. Meanwhile the server keeps on listening. The client keeps note of the elapsed time during the whole transaction.

4 Experimental result

Some Snapshots of the terminal output for each of these tools.

4.1 Capital to Small Conversion

4.1.1 Server

```
E:\Codes\py>python -u "e:\Codes\py\lab2\CapitalSmallServer.py"
Server is running...

Client: ALL IN CAPITAL
Sent: all in capital
Client: I LOVE BANGLADESH
Sent: i love bangladesh

Ending the Server... goodbye!

E:\Codes\py>
```

4.1.2 Client

```
E:\Codes\py\lab2>python CapitalSmallClient.py
Server: You are connected to the server...

Client: ALL IN CAPITAL
Server: all in capital

Client: I LOVE BANGLADESH
Server: i love bangladesh

Client: quit

Ending the server... goodbye!

E:\Codes\py\lab2>
```

4.2 Prime or Palindrome

4.2.1 Server

```
E:\Codes\py\lab2>python ServerPrimePalindrome.py
Server is running...

Received number:  10
Received request:  Prime
sent response:  No
Received number:  11
Received request:  Prime
sent response:  Yes
Received number:  10
Received request:  Palindrome
sent response:  No
Received number:  1123211
Received request:  Palindrome
sent response:  Yes
Received number:  1
Received request:  quit

Ending the Server... goodbye!
```

4.2.2 Client

```
E:\Codes\py\lab2>python ClientPrimePalindrome.py

Server: You are connected to the server...

Enter a number: 10
Type in either 'Prime' or 'Palindrome': Prime

Server: No

Enter a number: 11
Type in either 'Prime' or 'Palindrome': Prime

Server: Yes

Enter a number: 10
Type in either 'Prime' or 'Palindrome': Palindrome

Server: No

Enter a number: 1123211
Type in either 'Prime' or 'Palindrome': Palindrome

Server: Yes

Enter a number: 1
Type in either 'Prime' or 'Palindrome': quit

Ending the Server... goodbye!

E:\Codes\py\lab2>
```

4.3 ATM Machine

4.3.1 Server

```
E:\Codes\py\lab2>python ServerATM.py
Server is running...


Client Request Received:

Amount: 10000
Requested operation: wd

Successful withdrawal responded


Client Request Received:

Amount: 50000
Requested operation: wd

Insufficient fund responded


Client Request Received:

Amount: 20000
Requested operation: quit


Ending the server... goodbye!

E:\Codes\py\lab2>
```

4.3.2 Client

```
E:\Codes\py\lab2>python ClientATM.py

Server Response:
You are connected to the server...

Amount: 10000
Type in wd or dp for withdrawal or deposit: wd

Server Response:
Amount withdrawn: 10000
Balance: 40000

Amount: 50000
Type in wd or dp for withdrawal or deposit: wd

Server Response:
You have insufficient funds!!

Amount: 20000
Type in wd or dp for withdrawal or deposit: quit

Ending the server... goodbye!

E:\Codes\py\lab2>
```

4.4 ATM Machine: Error Handling

4.4.1 Server

```
E:\Codes\py\lab2>python ErrorServerATM.py
Server is running...

Error:  0.05798243999127839

Error has occurred!! Waiting for client...
Error:  0.25808549175749673

Error has occurred!! Waiting for client...
Error:  0.6146715284425768


Client Request Received:

Amount: 10000
Requested operation: wd

Successful withdrawal responded

Error:  0.11413931674804922

Error has occurred!! Waiting for client...
Error:  0.6740058260942822


Client Request Received:

Amount: 50000
Requested operation: wd

Insufficient fund responded

Error:  0.5802537892093477
```

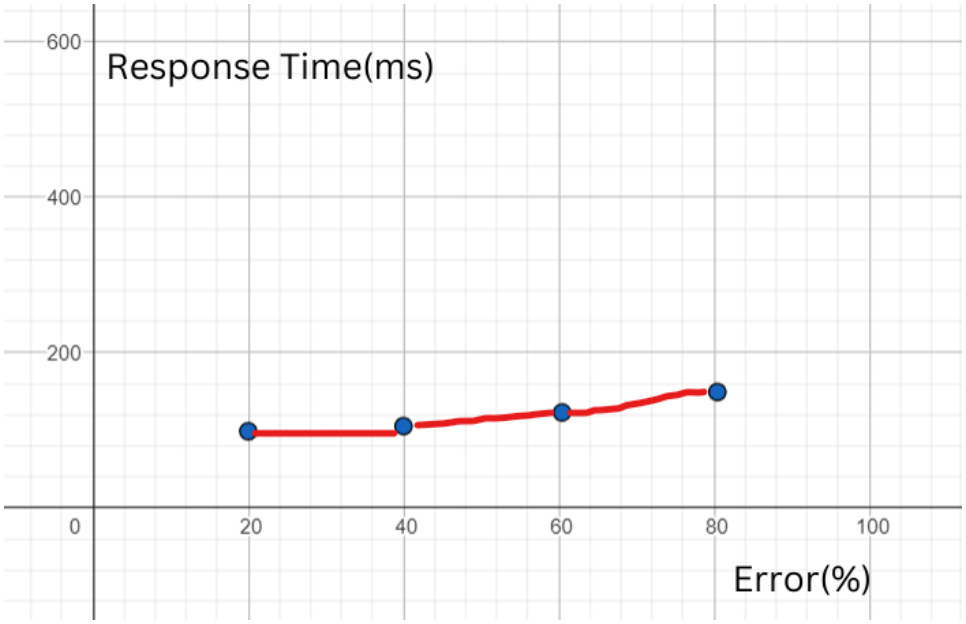
```
Error:  0.5802537892093477


Client Request Received:

Amount: 10000
Requested operation: dp

Successful diposition responded

Error:  0.5318490219057249


Client Request Received:

Amount: 1
Requested operation: quit


Ending the server... goodbye!

E:\Codes\py\lab2>1
```

4.4.2 Client

```
E:\Codes\py\lab2>python ErrorClientATM.py

Server Response:
You are connected to the server...

Amount: 10000
Type in wd or dp for withdrawal or deposit: wd

Server Response:
Amount withdrawn: 10000
Balance: 40000
Elapsed: 0.00596928596496582

Amount: 50000
Type in wd or dp for withdrawal or deposit: wd

Server Response:
You have insufficient funds!!
Elapsed: 0.0011832714080810547

Amount: 10000
Type in wd or dp for withdrawal or deposit: dp

Server Response:
Amount diposited: 10000
Balance: 50000
Elapsed: 0.003518342971801758

Amount: 1
Type in wd or dp for withdrawal or deposit: quit

Ending the server... goodbye!

E:\Codes\py\lab2>
```

4.5 Error Analysis

Now in this Error handling, we used different percentages of error occurrence. Here's a graph of the response time of the server with respect to the error percentages



5 Experience

1. We had to look up the server device IP to establish the connection
2. We established a client-server communication between two different devices using socket programming for the first time
3. We learned how errors are handled in a TCP connection

Appendix

All the codes are provided in Python programming language

Code for Capital to Small Conversion

Server

```
import socket

# HOST_IP = "10.33.2.94"
HOST_IP = socket.gethostbyname(socket.gethostname())
HOST_PORT = 12348
```

```
ENCODER = "utf-8"
BYTESIZE = 1024

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()

client_socket.send("You are connected to the server...".encode(ENCODER))

while True:
    message = client_socket.recv(BYTESIZE).decode(ENCODER)

    if message == "quit":
        client_socket.send("quit".encode(ENCODER))
        print("\nEnding the Server... goodbye!")
        break
    else:
        print(f"Client: {message}")
        client_socket.send((message.lower()).encode(ENCODER))
        print(f"Sent: {message.lower()}")

server_socket.close()
```

Client

```
import socket

#DEST_IP = "10.33.2.94"

DEST_IP = socket.gethostbyname(socket.gethostname())
DEST_PORT = 12348
ENCODER = "utf-8"
BYTESIZE = 1024

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((DEST_IP, DEST_PORT))
```

```
while True:
    message = client_socket.recv(BYTESIZE).decode(ENCODER)

    if message == "quit":
        client_socket.send("quit".encode(ENCODER))
        print("\nEnding the server... goodbye!")
        break
    else:
        print(f"Server: {message}")
        user_input = input(f"\nClient: ")
        client_socket.send(user_input.encode(ENCODER))

client_socket.close()
```

Code for Prime and Palindrome Test

Server

```
import socket

# HOST_IP = '10.33.2.94'
HOST_IP = socket.gethostbyname(socket.gethostname())
HOST_PORT = 12349
ENCODER = "utf-8"
BYTESIZE = 1024

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()

client_socket.send("You are connected to the server...".encode(ENCODER))

def isPrime(n):
    if n < 2:
        return "No"
    i = 2
    while i*i <= n:
```

```
        if n%i == 0:
            return "No"
        i += 1
    return "Yes"

def isPalindrome(n):
    if n == n[::-1]:
        return "Yes"
    return "No"

while True:
    number = client_socket.recv(BYTESIZE).decode(ENCODER)
    print("Received number: ", number)
    op = client_socket.recv(BYTESIZE).decode(ENCODER)
    print("Received request: ", op)

    if op == "quit":
        client_socket.send("quit".encode(ENCODER))
        print("\nEnding the Server... goodbye!")
        break
    elif op == 'Prime':
        message = isPrime(int(number))
        client_socket.send(message.encode(ENCODER))
        print("sent response: ", message)
    elif op == "Palindrome":
        message = isPalindrome(number)
        client_socket.send(message.encode(ENCODER))
        print("sent response: ", message)

server_socket.close()
```

Client

```
import socket

# HOST_IP = '10.33.2.94'
DEST_IP = socket.gethostbyname(socket.gethostname())
DEST_PORT = 12349
ENCODER = "utf-8"
BYTESIZE = 1024
```

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((DEST_IP, DEST_PORT))

while True:
    message = client_socket.recv(BYTESIZE).decode(ENCODER)

    if message == "quit":
        client_socket.send("quit".encode(ENCODER))
        print("\nEnding the Server... goodbye!")
        break
    else:
        print(f"\nServer: {message}")
        user_input = input(f"\nEnter a number: ")
        client_socket.send(user_input.encode(ENCODER))
        user_input = input(f"Type in either 'Prime' or 'Palindrome': ")
        client_socket.send(user_input.encode(ENCODER))

client_socket.close()
```

Code for ATM Server Implementation

Server

```
import socket

# HOST_IP = '10.33.2.94'
HOST_IP = socket.gethostbyname(socket.gethostname())
HOST_PORT = 12348
ENCODER = "utf-8"
BYTESIZE = 1024

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()

client_socket.send("You are connected to the server...".encode(ENCODER))
```

```
balance = 50000

while True:
    amount = client_socket.recv(BYTESIZE).decode(ENCODER)
    print('\nClient Request Received:\n')
    print(f'Amount: {amount}')
    op = client_socket.recv(BYTESIZE).decode(ENCODER)
    print(f'Requested operation: {op}\n')

    if op == "quit":
        client_socket.send("quit".encode(ENCODER))
        print("\nEnding the server... goodbye!")
        break
    elif op == 'wd':
        if balance < int(amount):
            client_socket.send("You have insufficient funds!!".encode(ENCODER))
            print('Insufficient fund responded\n')
        else:
            balance -= int(amount)
            response = "Amount withdrawn: " + str(amount) + "\nBalance: " + str(balan
            client_socket.send(response.encode(ENCODER))
            print('Successful withdrawal responded\n')
    elif op == 'dp':
        balance += int(amount)
        response = "Amount diposited: " + str(amount) + "\nBalance: " + str(balance)
        client_socket.send(response.encode(ENCODER))
        print('Successful diposition responded\n')
server_socket.close()
```

Client

```
import socket

DEST_IP = socket.gethostbyname(socket.gethostname())

# DEST_IP = '10.33.2.94'
DEST_PORT = 12348
ENCODER = "utf-8"
BYTESIZE = 1024
```

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((DEST_IP, DEST_PORT))

while True:
    message = client_socket.recv(BYTESIZE).decode(ENCODER)

    if message == "quit":
        client_socket.send("quit".encode(ENCODER))
        print("\nEnding the server... goodbye!")
        break
    else:
        print(f"\nServer Response:\n{message}")
        user_input = input("\nAmount: ")
        client_socket.send(user_input.encode(ENCODER))
        user_input = input("Type in wd or dp for withdrawal or deposit: ")
        client_socket.send(user_input.encode(ENCODER))

client_socket.close()
```

Code for Error Handling in ATM server

Server

```
import socket
import random
import time

# HOST_IP = '10.33.2.94'
HOST_IP = socket.gethostbyname(socket.gethostname())
HOST_PORT = 12348
ENCODER = "utf-8"
BYTESIZE = 1024

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()
```

```
client_socket.send("You are connected to the server...".encode(ENCODER))

errorPercentage = 0.2           #or whatever error percetage
balance = 50000

while True:
    error = random.random()
    print("Error: ", error, "\n")

    if error <= errorPercentage:
        print('Error has occurred!! Waiting for client...')
    else:
        amount = client_socket.recv(BYTESIZE).decode(ENCODER)
        print('\nClient Request Received:\n')
        print(f'Amount: {amount}')
        op = client_socket.recv(BYTESIZE).decode(ENCODER)
        print(f'Requested operation: {op}\n')
        if op == 'quit':
            client_socket.send('quit'.encode(ENCODER))
            print("\nEnding the server... goodbye!")
            break
        elif op == 'wd':
            if balance < int(amount):
                client_socket.send("You have insufficient funds!!".encode(ENCODER))
                print('Insufficient fund responded\n')
            else:
                balance -= int(amount)
                response = "Amount withdrawn: " + str(amount) + "\nBalance: " + str(b
                client_socket.send(response.encode(ENCODER))
                print('Successful withdrawal responded\n')
        elif op == 'dp':
            balance += int(amount)
            response = "Amount diposited: " + str(amount) + "\nBalance: " + str(balan
            client_socket.send(response.encode(ENCODER))
            print('Successful diposition responded\n')

server_socket.close()
```

Client

```
import socket
import time

DEST_IP = socket.gethostbyname(socket.gethostname())

# DEST_IP = '10.33.2.94'
DEST_PORT = 12348
ENCODER = "utf-8"
BYTESIZE = 1024

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((DEST_IP, DEST_PORT))

amount = "0"
opcode = "dp"

start_time = time.time()
end_time = time.time()

requested = False

while True:
    message = client_socket.recv(BYTESIZE).decode(ENCODER)
    if message.lower() == "quit":
        client_socket.send("quit".encode(ENCODER))
        print("\nEnding the server... goodbye!")
        break
    else:
        print(f"\nServer Response:\n{message}")

        if requested:
            end_time = time.time()
            elapsed_time = end_time - start_time
            print(f"Elapsed: {elapsed_time}")

        amount = input("\nAmount: ")
        client_socket.send(amount.encode(ENCODER))
        opcode = input("Type in wd or dp for withdrawal or deposit: ")
```

```
        client_socket.send(opcode.encode(ENCODER))
        requested = True
        start_time = time.time()

client_socket.close()
```

References

[1] https://realpython.com/

[2] https://www.geeksforgeeks.org/

[3] https://morioh.com/