



University of Dhaka

Department of Computer Science and Engineering

CSE-3111:Computer Networking Lab

Lab Report 2:

Introduction to Socket Programming

Exercise of Simple Client -Server Communication

Submitted By:

1. Diponker Roy Roll:28
2. Abdullah Ashik Roll:32

Submitted To:

1. Dr. Md. Abdur Razzaque
2. Dr. Md. Mamun or Rashid
3. Dr. Muhhamad Ibrahim
4. Mr. Md. Redwan Ahmed Rizvee

1 Introduction

Socket programming is a way of enabling communication between processes running on different networked computers. It allows processes to communicate with each other, either on the same machine or across a network, by using sockets, which are communication endpoints.

1.1 Objectives

The objective of socket programming is to facilitate communication between processes running on different networked computers or within the same computer. It enables data exchange, allowing programs to send and receive information over a network using sockets as communication endpoints. Socket programming aims to establish reliable and efficient communication channels between client and server applications, facilitating tasks such as file transfer, remote execution, real-time messaging, and distributed computing.

2 Theory

Socket programming is based on the concept of sockets, which are communication endpoints that allow processes to communicate with each other over a network. Here's a brief explanation of the theory behind socket programming:

Sockets:

Sockets are software abstractions representing communication endpoints in a network. Each socket is identified by an IP address and a port number, allowing processes to establish connections and communicate with each other. Client-Server Model:

Socket programming typically follows the client-server model, where one program (the server) listens for incoming connections, and another program (the client) initiates connections to the server. The server binds a socket to a specific IP address and port, listens for incoming connections, and accepts them. The client connects to the server's IP address and port, establishing a communication channel. Communication Protocols:

TCP sockets provide reliable, connection-oriented communication, ensuring data integrity and in-order delivery. UDP sockets provide connectionless communication, offering lower overhead but without reliability guarantees. Data Exchange:

Once a connection is established between client and server, they can exchange data using the socket's `send()` and `recv()` functions (for TCP) or `sendto()` and `recvfrom()` functions (for UDP). TCP sockets transmit data as a continuous stream, ensuring reliability and in-order delivery. UDP sockets transmit data as individual datagrams, which may arrive out of order or be lost. Error Handling and Exception Handling:

Socket programming involves error handling mechanisms to deal with network-related errors, such as connection failures or data transmission errors. Exception handling is used to gracefully handle exceptions that may occur during socket operations, ensuring the robustness of the communication system. Socket programming enables a wide range of networked applications, including web servers, chat applications, file transfer protocols, and more. It provides a flexible and powerful mechanism for processes to communicate and collaborate over a network.

3 Methodology

3.1 Server

A server is a computer or system that provides resources, data, or services to other computers or clients over a network. On the server side when we turn it on it will wait for any client request. If it gets any request then it will establish a connection. After setting up the connection, it receives a query corresponding to which file client requested. In this section Our task was Client sends Small letter. Server receives it and converts it to Capital letter and checks whether a number is prime or not and sends the result to client. We also implemented a Bank server. When a client requests for checking balance, cash deposit, cash withdrawal server does the operation and send back the result to the client. It can be said that server accepts all valid requests and rejects the requests that are invalid.

3.2 Client

A client is a computer or system that requests and uses the resources, data, or services provided by a server. Here our client side is any web browser. We will enter the IP address of our server and the port number. Then a request will be sent from client to the server. In our lab task we had to implement an ATM booth machine as a client. We will see some answers to the query that the server provided.

3.3 Problem A

Establishing 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. Small letter to capital conversion for a line of text
2. Checking whether a number is prime or not and also have to check whether the number

SERVER

for conversion small letter to capital letter

#(a) 1. Capital letter to Small letter conversion for a line of \\ text. This problem

```
import socket
import threading
```

```
HEADER = 64
PORT = 5050
FORMAT = 'utf-8'
DISCONNECTMESSAGE = "!DISCONNECT"
```

```
SERVER = socket.gethostbyname(socket.gethostname())
ADDR = (SERVER, PORT)
```

```
print (SERVER)
```

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(ADDR)
```

```
def handleClient(conn, addr):
    print(f"[NEW CONNECTION] {addr} connected.")

    connected = True
    while connected:
        msgLength = conn.recv(HEADER).decode(FORMAT)
        if msgLength:
            msgLength = int(msgLength)
```

```

        msg = conn.recv(msgLength).decode(FORMAT)
        print("Message received: ", msg)
        if msg == DISCONNECTMESSAGE:
            connected = False

        print(f"[{addr}] {msg.upper()}")

        response = f"[Lower Case] {msg.lower()} [Upper Case] {msg.upper()}"

        conn.send(response.encode(FORMAT))
    conn.close()

def start():
    server.listen()
    print(f"[LISTENING] Server is listening on {SERVER}")
    while True:
        conn, addr = server.accept()
        thread = threading.Thread(target=handleClient, args=(conn, addr))
        thread.start()
        print(f"[ACTIVE CONNECTIONS] {threading.activeCount() - 1}")

print("[STARTING] server is starting...")

start()

```

Client

```

import socket

HEADER = 64
PORT = 5050
FORMAT = "utf-8"
DISCONNECT_MESSAGE = "!DISCONNECT"

```

```

SERVER = socket.gethostbyname(socket.gethostname())

ADDR = (SERVER, PORT)
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDR)
def send(msg:str):
    message = msg.encode(FORMAT)
    msgLength = len(message)
    sendLength = str(msgLength).encode(FORMAT)
    sendLength += b' ' * (HEADER - len(sendLength))
    client.send(sendLength)
    client.send(message)

while True:
    msg = input("Enter message: ")
    send(msg)
    print(client.recv(2048).decode(FORMAT))
    if msg == DISCONNECT_MESSAGE:
        break

```

Server(for checking pallindrome and Prime)

(a) 2. Capital letter to Small letter conversion for a line of text
 # ii. Send an integer and operation name (either 'prime' or 'palindrome') to the server
 # check whether it's a prime (or palindrome) or not . This problem interacts with client

```

import socket
import threading
import utils

HEADER = 64
PORT = 5050
FORMAT = 'utf-8'
DISCONNECTMESSAGE = "!DISCONNECT"

SERVER = socket.gethostbyname(socket.gethostname())

```

```

ADDR = (SERVER, PORT)

print (SERVER)

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(ADDR)

def handleClient(conn, addr):
    print(f"[NEW CONNECTION] {addr} connected.")

    connected = True
    while connected:
        msgLength = conn.recv(HEADER).decode(FORMAT)
        if msgLength:
            msgLength = int(msgLength)
            msg = conn.recv(msgLength).decode(FORMAT)
            print("Message received: ", msg)
            if msg == DISCONNECTMESSAGE:
                connected = False

            isPalidrome = utils.isPalidrome(msg)

            msg = int(msg)

            response = f"[Prime] {utils.isPrime(msg)} [Palidrome] {isPalidrome}"

            conn.send(response.encode(FORMAT))

    conn.close()

def start():
    server.listen()
    print(f"[LISTENING] Server is listening on {SERVER}")
    while True:
        conn, addr = server.accept()

```

```

        thread = threading.Thread(target=handleClient, args=(conn, addr))
        thread.start()
        print(f"[ACTIVE CONNECTIONS] {threading.activeCount() - 1}")

print("[STARTING] server is starting...")

start()

```

Client(for checking pallindrome and Prime)

```

import socket

HEADER = 64
PORT = 5050
FORMAT = "utf-8"
DISCONNECT_MESSAGE = "!DISCONNECT"
SERVER = socket.gethostbyname(socket.gethostname())

ADDR = (SERVER, PORT)
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDR)

def send(msg:str):
    message = msg.encode(FORMAT)
    msgLength = len(message)
    sendLength = str(msgLength).encode(FORMAT)
    sendLength += b' ' * (HEADER - len(sendLength))
    client.send(sendLength)
    client.send(message)

while True:
    msg = input("Enter message: ")
    send(msg)
    print(client.recv(2048).decode(FORMAT))
    if msg == DISCONNECT_MESSAGE:

```



```
break
```

3.4 Problem B

Create a TCP connection and design and implement a non-idempotent operation between bank server and ATM booth (client).

Server(for ATM Booth)

```
import socket
import threading
import utils
import random

HEADER = 64
PORT = 5050
FORMAT = 'utf-8'
DISCONNECT_MESSAGE = "!DISCONNECT"

SERVER = socket.gethostbyname(socket.gethostname())
ADDR = (SERVER, PORT)

print(SERVER)

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(ADDR)

# Initialize counters for successful and total transactions
total_transactions = 0
successful_transactions = 0

def handleClient(conn, addr):
    global total_transactions, successful_transactions # Declare global variables

    print(f"[NEW CONNECTION] {addr} connected.")

    connected = True
```

```

while connected:
    msgLength = conn.recv(HEADER).decode(FORMAT)
    if msgLength:
        msgLength = int(msgLength)
        msg = conn.recv(msgLength).decode(FORMAT)

        if msg == DISCONNECT_MESSAGE:
            break
        if utils.getName(int(msg[0]), int(msg[1:5])) == False:
            str = "Invalid ID or Password\n"
            conn.send(str.encode(FORMAT))
            continue
        name = utils.getName(int(msg[0]), int(msg[1:5]))

        str = f"Welcome {name}!\n"
        str += "1. Check Balance\n"
        str += "2. Deposit\n"
        str += "3. Withdraw\n"
        str += "4. Exit\n"
        conn.send(str.encode(FORMAT))

    msgLength = conn.recv(HEADER).decode(FORMAT)
    if msgLength:
        msgLength = int(msgLength)
        msg = conn.recv(msgLength).decode(FORMAT)

        if msg == DISCONNECT_MESSAGE:
            break

        total_transactions += 1 # Increment total transactions count

        if msg[5] == "1":
            str = f"Your balance is {utils.getBalance(int(msg[0]), int(msg[1:5]))}\n"
            str += f"Total transactions: {total_transactions}\n"
            str += f"Success rate: {successful_transactions/total_transactions}\n"
            str += f"Error rate: {(total_transactions-successful_transactions)/total_transactions}\n"
            conn.send(str.encode(FORMAT))

        elif msg[5] == "2":

```

```

rand = random.randint(0, 10)
str = "Enter amount to deposit: "

conn.send(str.encode(FORMAT))
msgLength = conn.recv(HEADER).decode(FORMAT)
if msgLength:
    msgLength = int(msgLength)
    msg = conn.recv(msgLength).decode(FORMAT)

    if msg == DISCONNECT_MESSAGE:
        break
    if rand > 5:
        utils.deposit(int(msg[0]), int(msg[1:5]), int(msg[5:]))
        successful_transactions += 1 # Increment successful transactions
        str = f"Your balance is {utils.getBalance(int(msg[0]), int(msg[1:5]))}\n"
        str += f"Successful transactions: {successful_transactions}\n"
        str += f"Total transactions: {total_transactions}\n"
        str += f"Success rate: {successful_transactions/total_transactions}\n"
        str += f"Error rate: {(total_transactions-successful_transactions)/total_transactions}\n"
        conn.send(str.encode(FORMAT))
    else:
        str = "Transaction failed\n"
        str += f"Your balance is {utils.getBalance(int(msg[0]), int(msg[1:5]))}\n"
        str += "Please try again\n"
        str += f"Total transactions: {total_transactions}\n"
        str += f"Success rate: {successful_transactions/total_transactions}\n"
        str += f"Error rate: {(total_transactions-successful_transactions)/total_transactions}\n"
        conn.send(str.encode(FORMAT))

elif msg[5] == "3":
    str = "Enter amount to withdraw: "
    rand = random.randint(0, 10)

    conn.send(str.encode(FORMAT))
    msgLength = conn.recv(HEADER).decode(FORMAT)
    if msgLength:
        msgLength = int(msgLength)
        msg = conn.recv(msgLength).decode(FORMAT)

```

```

        if msg == DISCONNECT_MESSAGE:
            break
        if rand > 5:
            utils.withdraw(int(msg[0]), int(msg[1:5]), int(msg[5:]))
            successful_transactions += 1 # Increment successful transactions
            str = f"Your balance is {utils.getBalance(int(msg[0])), :.2f}\n"
            str += f"Successful transactions: {successful_transactions}\n"
            str += f"Total transactions: {total_transactions}\n"
            str += f"Success rate: {successful_transactions/total_transactions}\n"
            str += f"Error rate: {(total_transactions-successful_transactions)/total_transactions}\n"
            conn.send(str.encode(FORMAT))
        else:
            str = "Transaction failed\n"
            str += f"Your balance is {utils.getBalance(int(msg[0])), :.2f}\n"
            str += "Please try again\n"
            str += f"Total transactions: {total_transactions}\n"
            str += f"Success rate: {successful_transactions/total_transactions}\n"
            str += f"Error rate: {(total_transactions-successful_transactions)/total_transactions}\n"
            conn.send(str.encode(FORMAT))

    elif msg[5] == "4":
        break

    conn.close()

def start():
    server.listen()
    print(f"[LISTENING] Server is listening on {SERVER}")
    while True:
        conn, addr = server.accept()
        thread = threading.Thread(target=handleClient, args=(conn, addr))
        thread.start()
        print(f"[ACTIVE CONNECTIONS] {threading.activeCount() - 1}")

start()

```

Client(for ATM Booth)

```
import socket
HEADER = 64
PORT = 5050
FORMAT = "utf-8"
DISCONNECT_MESSAGE = "!DISCONNECT"
SERVER = socket.gethostname(socket.gethostname())

ADDR = (SERVER, PORT)
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDR)
def send(id:str, password:str, choice:str = "", amount:str = ""):
    msg = id + password + choice+ amount
    message = msg.encode(FORMAT)
    msgLength = len(message)
    sendLength = str(msgLength).encode(FORMAT)
    sendLength += b' ' * (HEADER - len(sendLength))
    client.send(sendLength)
    client.send(message)

while True:
    id = input("Enter ID: ")
    password = input("Enter Password: ")
    send(id, password)

    print(client.recv(2048).decode(FORMAT))

    choice = input("Enter choice: ")
    send(id, password, choice)

    if choice == "1":
        print(client.recv(2048).decode(FORMAT))

    elif choice == "2":
```

```
print(client.recv(2048).decode(FORMAT))
amount = input("Enter amount: ")
send(id, password, amount)
print(client.recv(2048).decode(FORMAT))

elif choice == "3":
    print(client.recv(2048).decode(FORMAT))
    amount = input("Enter amount: ")
    send(id, password, amount)
    print(client.recv(2048).decode(FORMAT))
```

Utils.py file

```
def isPrime(n):

    if n == 1:
        return False

    for i in range(2, n):
        if n % i == 0:
            return False

    return True

def isPalidrome(n):
    return n == n[::-1]

user = [
    [
        {
            "id" : 1,
            "password" : 1234,
            "balance" : 10000,
            "name" : "Dibbyo Roy"
```

```

    },
    {
        "id" : 2,
        "password" : 1234,
        "balance" : 10000,
        "name" : "Abdullah Ashik"

    },
    {
        "id" : 3,
        "password" : 1234,
        "balance" : 10000,
        "name" : "Abir Hasan"
    }
]
]

```

```

def getName(id, password):
    for i in range(len(user)):
        for j in range(len(user[i])):
            if user[i][j]["id"] == id and user[i][j]["password"] == password:
                return user[i][j]["name"]
    return False

def getBalance(id, password):
    for i in range(len(user)):
        for j in range(len(user[i])):
            if user[i][j]["id"] == id and user[i][j]["password"] == password:
                return user[i][j]["balance"]
    return -1

def deposit(id, password, amount):
    for i in range(len(user)):
        for j in range(len(user[i])):
            if user[i][j]["id"] == id and user[i][j]["password"] == password:
                user[i][j]["balance"] += amount
                return user[i][j]["balance"]
    return -1

```

```

def withdraw(id, password, amount):
    for i in range(len(user)):
        for j in range(len(user[i])):
            if user[i][j]["id"] == id and user[i][j]["password"] == password:
                user[i][j]["balance"] -= amount
                return user[i][j]["balance"]
    return -1

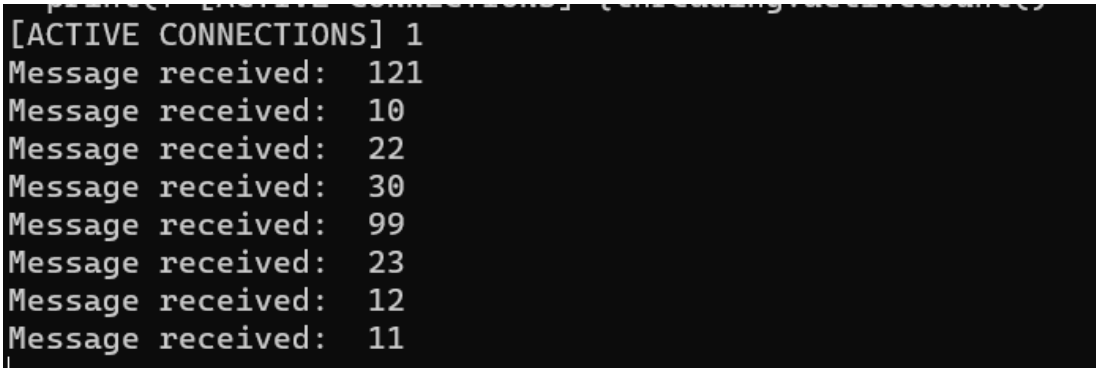
```

4 Experimental Result

4.1 Problem A

Server :

Server takes a string or an integer from the client. It converts the string to an upper case string and send it back to the client. On the other hand, it check the integer if its a prime or not and send the result to the client.



```

[ACTIVE CONNECTIONS] 1
Message received: 121
Message received: 10
Message received: 22
Message received: 30
Message received: 99
Message received: 23
Message received: 12
Message received: 11

```

Figure 1: Content Of Server Problem A(Checking Pallindrome and Prime)


```

[ACTIVE CONNECTIONS] 1
Message received:  hello world
[('192.168.0.103', 49891)] HELLO WORLD
Message received:  ashik and dibbyo
[('192.168.0.103', 49891)] ASHIK AND DIBBYO
Message received:  this is our second networking lab
[('192.168.0.103', 49891)] THIS IS OUR SECOND NETWORKING LAB
Message received:  and the lab is about socket programming
[('192.168.0.103', 49891)] AND THE LAB IS ABOUT SOCKET PROGRAMMING
Message received:  hi
[('192.168.0.103', 49891)] HI

```

Figure 2: Content Of Server Problem A(for sending Uppercase letter to client)

Client :

The client takes the input from the user and send it to the server. Then it takes the message form the server and show it in the console..

```

[ACTIVE CONNECTIONS] 1
Message received:  121
Message received:  10
Message received:  22
Message received:  30
Message received:  99
Message received:  23
Message received:  12
Message received:  11

```

Figure 3: Content Of Client Problem A(Checking Pallindrome and Prime)

```

Enter message: hello world
[Lower Case] hello world [Upper Case] HELLO WORLD
Enter message: ashik and dibbyo
[Lower Case] ashik and dibbyo [Upper Case] ASHIK AND DIBBYO
Enter message: this is our second networking lab
[Lower Case] this is our second networking lab [Upper Case] THIS IS OUR SECOND NETWORKING LAB
Enter message: and the lab is about socket programming
[Lower Case] and the lab is about socket programming [Upper Case] AND THE LAB IS ABOUT SOCKET PROGRAMMING
Enter message: hi
[Lower Case] hi [Upper Case] HI
Enter message: 

```

Figure 4: Content Of Client Problem A(for sending Uppercase letter to client)

4.2 Problem B

This is for ATM Machine Problem we set four option for client Checking balance , withdraw , deposit and exit if client choose exit he will be exit from connection

otherwise we have shown her/his successful transaction.

```
2. Deposit
3. Withdraw
4. Exit

Enter choice: 2
Enter amount to deposit:
Enter amount: 2
Your balance is 10002
Successful transactions: 1
Total transactions: 1
Success rate: 1.00
Error rate: 0.00

Enter ID: 1
Enter Password: 1234
Welcome Dibbyo Roy!
1. Check Balance
2. Deposit
3. Withdraw
4. Exit

Enter choice: 2
Enter amount to deposit:
Enter amount: 2
Transaction failed
Your balance is 10000
Please try again
Total transactions: 2
Success rate: 0.50
Error rate: 0.50
```

Figure 4: Content Of Client Problem A(for sending Uppercase letter to client)



Figure 5: Total Transaction vs error graph

5 Experience

1. We had a wonderful experience with how servers and clients work in real life.
2. Experienced how socket programming work and establish connection between client and server.

References

1. Socket programming in python. GeeksforGeeks, jun 20 2017. [Online; accessed 2023-01-25].
2. Pankaj. Python socket programming - Server, client example. DigitalOcean, aug 3 2022. [Online;accessed 2023-01-25].
3. Real Python. Socket programming in python (guide). Real Python, feb 21 2022. [Online; accessed 2023-01-25].