CS 461

Lab Assignment 6

Name: Gandhi Dhruv Vipulkumar

Institute ID: 202151053

Date: 16-11-2024

**Q. Implement Distributed Banking Application**

**Server.py:**

import socket  
import threading  
  
class Server:  
    def \_\_init\_\_(self, host='127.0.0.1', port=9000):  
        self.server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
        self.server\_socket.bind((host, port))  
        self.server\_socket.listen(5)  
        self.clients = {}  
  
        print(f"Server running on {host}:{port}")  
  
    def handle\_client(self, client\_socket, client\_address):  
        try:  
            while True:  
                packet = client\_socket.recv(1024).decode()  
                if not packet:  
                    break  
  
                # Registration  
                if packet.startswith('id\_'):  
                    client\_id = packet[3:]  
                    self.clients[client\_id] = 0  
                    print(f"Registered client ID: {client\_id}")  
  
                # Withdrawal  
                elif packet.startswith('1'):  
                    id, amount = packet[1:].split('.')  
                    amount = int(amount)  
                    if self.clients.get(id, 0) < amount:  
                        # Insufficient funds  
                        client\_socket.sendall("0".encode())  
                    else:  
                        self.clients[id] -= amount  
                        client\_socket.sendall("1".encode())  # Successful  
  
                # Deposit  
                elif packet.startswith('2'):  
                    id, amount = packet[1:].split('.')  
                    amount = int(amount)  
                    self.clients[id] = self.clients.get(id, 0) + amount  
                    client\_socket.sendall("1".encode())  # Successful  
  
                # Transfer  
                elif packet.startswith('3'):  
                    id, id2, amount = packet[1:].split('.')  
                    amount = int(amount)  
                    if self.clients.get(id, 0) < amount:  
                        # Insufficient funds  
                        client\_socket.sendall("0".encode())  
                    elif id2 not in self.clients:  
                        # Invalid recipient  
                        client\_socket.sendall("1".encode())  
                    else:  
                        self.clients[id] -= amount  
                        self.clients[id2] += amount  
                        client\_socket.sendall("2".encode())  # Successful  
  
                # Balance inquiry  
                elif packet.startswith('4'):  
                    balance = self.clients.get(packet[1:], 0)  
                    client\_socket.sendall(str(balance).encode())  
  
                # Exit  
                elif packet.startswith('5'):  
                    print(f"Client {client\_address} disconnected.")  
                    break  
        finally:  
            client\_socket.close()  
  
    def start(self):  
        while True:  
            client\_socket, client\_address = self.server\_socket.accept()  
            print(f"Connection established with {client\_address}")  
            threading.Thread(target=self.handle\_client, args=(  
                client\_socket, client\_address)).start()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    server = Server()  
    server.start()

**Client.py**

import socket  
import sys  
  
class Client:  
    def \_\_init\_\_(self, host='127.0.0.1', port=9000):  
        self.host = host  
        self.port = port  
        self.socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
        self.socket.connect((self.host, self.port))  
  
    def register(self, client\_id):  
        msg = 'id\_' + client\_id  
        self.socket.send(msg.encode())  
  
    def send\_transaction(self, msg):  
        self.socket.send(msg.encode())  
        response = self.socket.recv(1024).decode()  
        return response  
  
    def close(self):  
        self.socket.send('5'.encode())  
        self.socket.close()  
  
def main():  
    client = Client()  
    client\_id = input(  
        "Enter your first name and ID (Ex: John123): ").replace(' ', '')  
    client.register(client\_id)  
  
    while True:  
        print("\nEnter the number according to the option you want:")  
        print("1: Withdraw\n2: Deposit\n3: Transfer\n4: Balance\n5: Exit")  
        option = input("Your choice: ")  
  
        if option == '1':  
            amount = input("Enter the amount to withdraw: ")  
            msg = option + client\_id + '.' + amount  
            response = client.send\_transaction(msg)  
            print("Transaction Successful!" if response ==  
                  '1' else "Insufficient funds.")  
  
        elif option == '2':  
            amount = input("Enter the amount to deposit: ")  
            msg = option + client\_id + '.' + amount  
            response = client.send\_transaction(msg)  
            print("Deposit Successful!" if response ==  
                  '1' else "Deposit Failed.")  
  
        elif option == '3':  
            recipient = input("Enter the recipient's ID: ")  
            amount = input("Enter the amount to transfer: ")  
            msg = option + client\_id + '.' + recipient + '.' + amount  
            response = client.send\_transaction(msg)  
            if response == '2':  
                print("Transfer Successful!")  
            elif response == '1':  
                print("Invalid recipient.")  
            else:  
                print("Insufficient funds.")  
  
        elif option == '4':  
            msg = option + client\_id  
            balance = client.send\_transaction(msg)  
            print("Your balance is:", balance)  
  
        elif option == '5':  
            client.close()  
            break  
  
        else:  
            print("Invalid option. Please try again.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    main()

**Code Explanation:**

**Server Code (server.py):**

* The server acts as a central component that listens for client connections and processes requests related to banking operations such as registering clients, withdrawing, depositing, transferring funds, and checking balances.

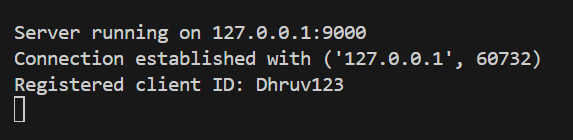
**Client Code (client.py):**

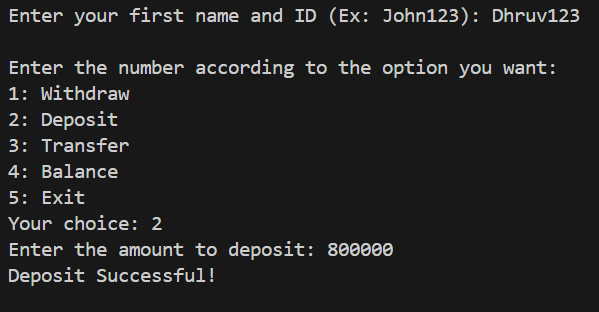
* The client interacts with the server to perform various banking operations by sending requests and receiving responses.

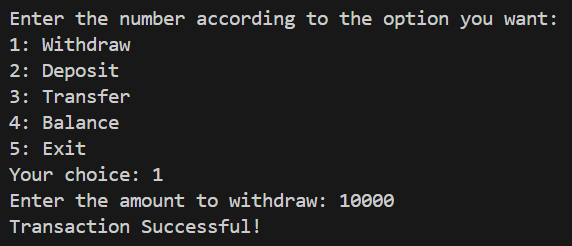
**Key Features:**

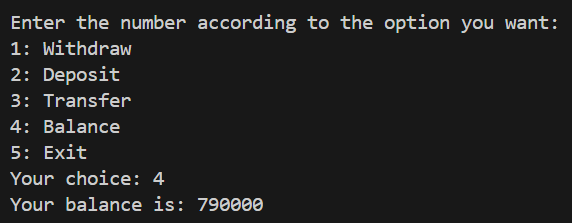
1. Withdraw
2. Deposit
3. Transfer
4. Check Balance

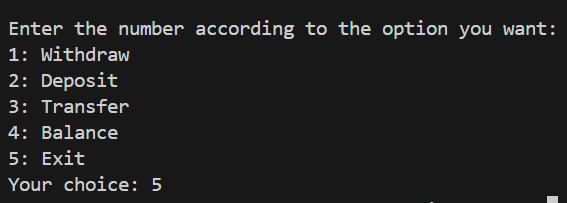
**Testing Phase:**

****

****

****

****

****

**Conclusion:** Successfully implemented distributed banking application.