LAB – 3

- 21 AUG 2023

1. Implement an auction marketplace. An auctioneer has a set of items to be auctioned and users can buy an item. Before an item is sold, it should be properly put out for auction - the auctioneer starts with the basic price, and users keep quoting higher and higher prices till no one agrees for a higher price and one of them decides to buy it.

Server.py

```
import socket
import threading

class AuctionItem:
    def __init__(self, name, initial_price):
        self.name = name
        self.current_price = initial_price
        self.highest_bidder = None

    def place_bid(self, bidder, bid_amount):
        if bid_amount > self.current_price:
            self.current_price = bid_amount
            self.highest_bidder = bidder
            return True
        return False

    def __str__(self):
        return f"{self.name}: Current Price - {self.current_price}, Highest Bidder -
{self.highest_bidder}"

class AuctionServer:
    def __init__(self):
        self.auction_items = {}
        self.lock = threading.Lock()

    def add_item(self, item):
        self.auction_items[item.name] = item

    def handle_client(self, client_socket):
        while True:
            client_socket.send("1. List Auction Items\n2. Bid for an Item\n3. Exit\n".encode())
            choice = client_socket.recv(1024).decode()

            if choice == "1":
                items_list = "\n".join([item.name for item in self.auction_items.values()])
                client_socket.send(items_list.encode())
            elif choice == "2":
                items_list = "\n".join([item.name for item in self.auction_items.values()])
                client_socket.send(items_list.encode())
                item_name = client_socket.recv(1024).decode()
                if item_name in self.auction_items:
                    self.handle_auction(item_name, client_socket)
                else:
                    client_socket.send("Item not found.".encode())
            elif choice == "3":
                client_socket.send("Goodbye!".encode())
                break
            else:
                client_socket.send("Invalid choice. Please select again.".encode())

    def handle_auction(self, item_name, client_socket):
        auction_item = self.auction_items[item_name]
        client_socket.send(f"Auction for '{item_name}' started with initial price: 
{auction_item.current_price}".encode())

        while True:
            client_socket.send("Enter your bid: ".encode())
            bid_amount = int(client_socket.recv(1024).decode())
            if auction_item.place_bid(client_socket.getpeername(), bid_amount):
                client_socket.send(f"Bid accepted! Current highest bid: {auction_item.current_price} by 
{auction_item.highest_bidder}".encode())
            else:
                client_socket.send("Your bid was not accepted. Please bid a higher amount.".encode())
                break

def main():
    auction_server = AuctionServer()

    item1 = AuctionItem("Painting", 100)
    item2 = AuctionItem("Antique Watch", 200)
    item3 = AuctionItem("Rare Coin", 150)

    auction_server.add_item(item1)
    auction_server.add_item(item2)
    auction_server.add_item(item3)

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('127.0.0.1', 12345))
    server_socket.listen(5)

    print("Auction Server is listening...")

    while True:
        client_socket, client_addr = server_socket.accept()
        print(f"Client connected: {client_addr}")
        threading.Thread(target=auction_server.handle_client, args=(client_socket,)).start()

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

client.py

```
import socket

def main():
    host = '127.0.0.1'
    port = 12345

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((host, port))

    while True:
        print("1. List Auction Items\n2. Bid for an Item\n3. Exit")
        choice = input("Enter your choice: ")
        client_socket.send(choice.encode())

        if choice == "1":
            items_list = client_socket.recv(1024).decode()
            print("Auction Items:")
            print(items_list)
        elif choice == "2":
            items_list = client_socket.recv(1024).decode()
            print("Auction Items:")
            print(items_list)
            item_name = input("Enter the name of the item for auction: ")
            client_socket.send(item_name.encode())
            response = client_socket.recv(1024).decode()
            print(response)
            if "started" in response:
                while True:
                    bid_amount = int(input("Enter your bid amount: "))
                    client_socket.send(str(bid_amount).encode())
                    response = client_socket.recv(1024).decode()
                    print(response)
                    if "accepted" not in response:
                        break
        elif choice == "3":
            print("Goodbye!")
            break
        else:
            print("Invalid choice. Please select again.")

    client_socket.close()

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

2. Implement an auth service using sockets. Clients can create / sign up an account. Once signed in, you can store key value pairs which you can return as per demand. Implement some form of hashing of password and encryption for kay value paid using a library. Also allow clients to login and upload files in the server and set proper access - either to the whole public or only to certain specific users.

Server.py

```
import socket
import threading

clients = []

def handle_client(client_socket):
    while True:
        try:
            message = client_socket.recv(1024).decode()
            if not message:
                remove_client(client_socket)
                break

            broadcast(message, client_socket)
        except:
            remove_client(client_socket)
            break

def broadcast(message, sender_socket):
    for client in clients:
        if client != sender_socket:
            try:
                client.send(message.encode())
            except:
                continue

def remove_client(client_socket):
    if client_socket in clients:
        clients.remove(client_socket)

def main():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(('127.0.0.1', 12345))
    server.listen(5)

    print("Server is listening...")

    while True:
        client_socket, client_addr = server.accept()
        clients.append(client_socket)
        print(f"Connected to {client_addr}")

        client_thread = threading.Thread(target=handle_client, args=(client_socket,))
        client_thread.start()

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

Client.py

```
import socket
import threading

def receive_messages(client_socket):
    while True:
        try:
            message = client_socket.recv(1024).decode()
            print(message)
        except:
            print("Connection closed.")
            break

def main():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('127.0.0.1', 12345))

    receive_thread = threading.Thread(target=receive_messages, args=(client,))
    receive_thread.start()

    while True:
        message = input()
        client.send(message.encode())

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