Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

**09**

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| **01** | **Basic Concurrent Account Operations** |
| **02** | **Ensuring Thread Safety in a Messaging System** |
| 03 | GUI-Based Bidirectional Chat System with Socket Programming |

Submitted On:

**Date: 13/Dec/2023**

**TASK # 1: Basic Concurrent Account Operations**

import threading

import time

import random

class BankAccount:

    def \_\_init\_\_(self, balance=0):

        self.balance = balance

        self.lock = threading.Lock()

    def deposit(self, amount):

        with self.lock:

            current\_balance = self.balance

            time.sleep(0.1)

            self.balance = current\_balance + amount

            print(f"Deposited {amount}. New balance: {self.balance}")

    def withdraw(self, amount):

        with self.lock:

            current\_balance = self.balance

            if current\_balance >= amount:

                time.sleep(0.1)

                self.balance = current\_balance - amount

                print(f"Withdrew {amount}. New balance: {self.balance}")

            else:

                print("Insufficient funds. Withdrawal canceled.")

    def get\_balance(self):

        with self.lock:

            return self.balance

def perform\_operations(account, num\_operations):

    for \_ in range(num\_operations):

        operation = random.choice(['deposit', 'withdraw'])

        amount = random.randint(50, 200)

        if operation == 'deposit':

            account.deposit(amount)

        else:

            account.withdraw(amount)

        time.sleep(0.2)

account1 = BankAccount()

account2 = BankAccount()

thread1 = threading.Thread(target=perform\_operations, args=(account1, 5))

thread2 = threading.Thread(target=perform\_operations, args=(account2, 5))

thread1.start()

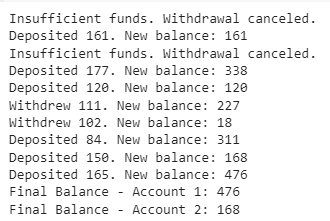
thread2.start()

thread1.join()

thread2.join()

print("Final Balance - Account 1:", account1.get\_balance())

print("Final Balance - Account 2:", account2.get\_balance())



**TASK#2: Ensuring Thread Safety in a Messaging System**

import threading

import time

class MessageQueue:

    def \_\_init\_\_(self):

        self.messages = []

        self.lock = threading.Lock()

    def send\_message(self, message):

        with self.lock:

            self.messages.append(message)

            print(f"Sent: {message}")

    def receive\_message(self):

        with self.lock:

            if self.messages:

                message = self.messages.pop(0)

                print(f"Received: {message}")

                return message

            else:

                print("No messages to receive.")

                return None

def send\_messages(queue, num\_messages):

    for i in range(num\_messages):

        time.sleep(0.1)

        queue.send\_message(f"Message {i}")

def receive\_messages(queue, num\_messages):

    for \_ in range(num\_messages):

        time.sleep(0.1)

        queue.receive\_message()

message\_queue = MessageQueue()

sender\_thread = threading.Thread(target=send\_messages, args=(message\_queue, 5))

receiver\_thread = threading.Thread(target=receive\_messages, args=(message\_queue, 5))

sender\_thread.start()

receiver\_thread.start()

sender\_thread.join()

receiver\_thread.join()



**TASK#3: GUI-Based Bidirectional Chat System with Socket Programming**

import socket

import threading

HOST = '127.0.0.1'

PORT = 5555

ADDR = (HOST, PORT)

clients = []

lock = threading.Lock()

def handle\_client(client, addr):

with lock:

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

clients.append(client)

connected = True

while connected:

try:

message = client.recv(1024).decode('utf-8')

if message:

print(f"[{addr}] {message}")

broadcast(message, addr)

except:

with lock:

print(f"[CONNECTION LOST] {addr} disconnected.")

clients.remove(client)

client.close()

connected = False

def broadcast(message, sender\_addr):

with lock:

for client in clients:

if client != sender\_addr:

try:

client.send(message.encode('utf-8'))

except:

continue

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.bind(ADDR)

server.listen()

print(f"[LISTENING] Server is listening on {HOST}:{PORT}")

while True:

client, addr = server.accept()

client\_handler = threading.Thread(target=handle\_client, args=(client, addr))

client\_handler.start()

import socket

import threading

from tkinter import \*

HOST = '127.0.0.1'

PORT = 5555

ADDR = (HOST, PORT)

def receive():

while True:

try:

message = client\_socket.recv(1024).decode('utf-8')

message\_listbox.insert(END, message)

except:

break

def send(event=None):

message = my\_message.get()

my\_message.set("")

client\_socket.send(message.encode('utf-8'))

if message.lower() == "/exit":

client\_socket.close()

root.quit()

root = Tk()

send\_button = Button(root, text="Send", command=send)

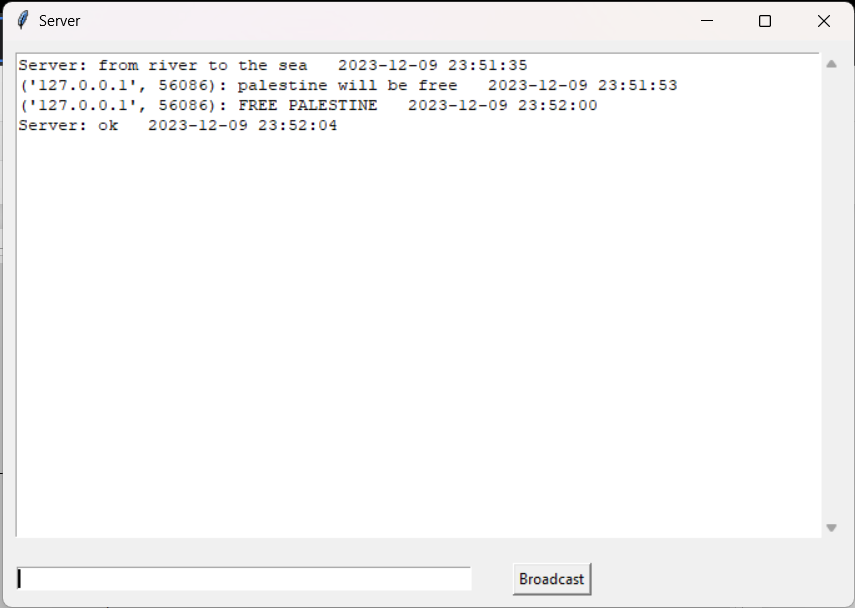
send\_button.pack(padx=10, pady=10)

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect(ADDR)

receive\_thread = threading.Thread(target=receive)

receive\_thread.start()

**A screenshot of a computer

Description automatically generated**root.mainloop()