Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

**09**

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| 1 | Basic Concurrent Account Operations  In this task, students are required to implement a simple banking system that supports concurrent account operations. They should create a BankAccount class with methods for deposit, withdrawal, and balance inquiry, ensuring the proper synchronization to handle concurrent transactions. |
| 2 | Ensuring Thread Safety in a Messaging System  In this task, students are tasked with designing and implementing a simple messaging system that ensures thread safety. The system should allow multiple threads to send and receive messages concurrently without the risk of data corruption or race conditions. Students should create a Message Queue class with methods for sending and receiving messages, and they must incorporate synchronization mechanisms such as locks or semaphores to guarantee thread safety. |
| 3 | GUI-Based Bidirectional Chat System with Socket Programming  The task involves creating a graphical user interface (GUI) for a bidirectional chat system using socket programming. Students are required to design an intuitive GUI with features like message input fields, chat logs, and message display areas. Through socket programming, they need to implement the server to handle multiple clients, manage connections, and facilitate real-time communication. The bidirectional nature of the system should allow users to send and receive messages, complete with timestamps and sender identification. Additionally, students should ensure thread safety to manage concurrent connections and implement robust error handling mechanisms. |

Submitted On:

05/12/2023

(Date: DD/MM/YY)

**Task No 1:** Basic Concurrent Account Operations

**Solution:**

import threading

import time

import random

class BankAccount:

def \_\_init\_\_(self, account\_id, initial\_balance=0):

*self*.account\_id = account\_id

*self*.balance = initial\_balance

*self*.lock = threading.Lock()

def deposit(self, amount):

with *self*.lock:

current\_balance = *self*.balance

new\_balance = current\_balance + amount

time.sleep(1)

*self*.balance = new\_balance

print(f"Deposited ${amount} to Account {*self*.account\_id}. New balance: ${new\_balance}")

def withdraw(self, amount):

with *self*.lock:

current\_balance = *self*.balance

if current\_balance >= amount:

new\_balance = current\_balance - amount

time.sleep(1)

*self*.balance = new\_balance

print(f"Withdrew ${amount} from Account {*self*.account\_id}. New balance: ${new\_balance}")

else:

print(f"Insufficient funds in Account {*self*.account\_id} to withdraw ${amount}")

def check\_balance(self):

with *self*.lock:

print(f"Balance in Account {*self*.account\_id}: ${*self*.balance}")

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

for \_ in range(num\_operations):

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

amount = random.randint(1, 100)

if operation == "deposit":

account.deposit(amount)

else:

account.withdraw(amount)

time.sleep(0.5)

if \_\_name\_\_ == "\_\_main\_\_":

accounts = [BankAccount(account\_id) for account\_id in range(1, 4)]

threads = []

for account in accounts:

thread = threading.Thread(target=perform\_operations, args=(account, 2))

threads.append(thread)

thread.start()

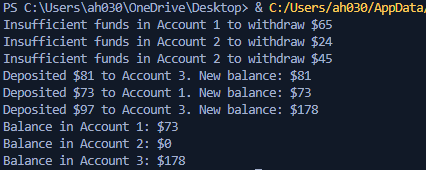
for thread in threads:

thread.join()

for account in accounts:

account.check\_balance()

**Output:**

****

**Task No 2:** Ensuring Thread Safety in a Messaging System

In this task, students are tasked with designing and implementing a simple messaging system that ensures thread safety. The system should allow multiple threads to send and receive messages concurrently without the risk of data corruption or race conditions. Students should create a Message Queue class with methods for sending and receiving messages, and they must incorporate synchronization mechanisms such as locks or semaphores to guarantee thread safety. The goal is to demonstrate the ability to handle concurrent operations on shared resources securely. Evaluation will be based on the correctness of the messaging system, the effectiveness of thread safety measures, and the demonstration of proper synchronization in a multi-threaded environment.

**Solution:**

import threading

import time

from queue import Queue

class MessageQueue:

def \_\_init\_\_(self):

*self*.messages = Queue()

*self*.lock = threading.Lock()

def send\_message(self, sender, content):

with *self*.lock:

message = f"{sender}: {content}"

*self*.messages.put(message)

print(f"Message sent by {sender}: {content} :)")

def receive\_message(self, receiver):

with *self*.lock:

if not *self*.messages.empty():

message = *self*.messages.get()

print(f"Message received by {receiver}: {message} :)")

else:

print(f"No messages for {receiver}")

def user\_function(user, message\_queue):

for \_ in range(3):

time.sleep(1)

message\_queue.send\_message(user, f"Hi, How are you doing {\_ + 1}")

for \_ in range(3):

time.sleep(1)

message\_queue.receive\_message(user)

if \_\_name\_\_ == "\_\_main\_\_":

message\_queue = MessageQueue()

user1\_thread = threading.Thread(target=user\_function, args=("User1", message\_queue))

user2\_thread = threading.Thread(target=user\_function, args=("User2", message\_queue))

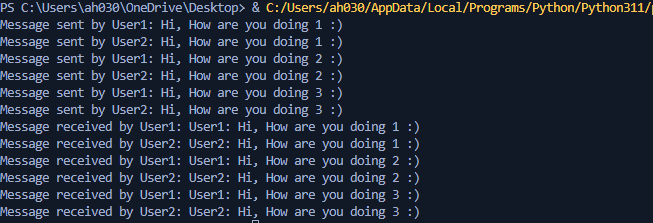
user1\_thread.start()

user2\_thread.start()

user1\_thread.join()

user2\_thread.join()

**Output:**

****

**Task No 3:** GUI-Based Bidirectional Chat System with Socket Programming

The task involves creating a graphical user interface (GUI) for a bidirectional chat system using socket programming. Students are required to design an intuitive GUI with features like message input fields, chat logs, and message display areas. Through socket programming, they need to implement the server to handle multiple clients, manage connections, and facilitate real-time communication. The bidirectional nature of the system should allow users to send and receive messages, complete with timestamps and sender identification. Additionally, students should ensure thread safety to manage concurrent connections and implement robust error handling mechanisms. The final evaluation will consider the completeness and correctness of the GUI, the effectiveness of bidirectional communication, the implementation of thread safety measures, and the overall reliability of the chat system. Additional features, such as file sharing or encryption, can be explored for extra credit.

**Solution:**

|  |  |
| --- | --- |
| import tkinter as tk  from tkinter import scrolledtext  import socket  import threading  from datetime import datetime  class ChatApp:  def \_\_init\_\_(self, master):  self.master = master  self.master.title("Chat Application")  self.message\_area = scrolledtext.ScrolledText(self.master, wrap=tk.WORD, width=50, height=20)  self.message\_area.tag\_configure("server", foreground="blue")  self.message\_area.tag\_configure("client", foreground="green")  self.message\_area.pack(padx=10, pady=10)  self.entry\_var = tk.StringVar()  self.message\_entry = tk.Entry(self.master, textvariable=self.entry\_var, width=50)  self.message\_entry.pack(padx=10, pady=10)  self.send\_button = tk.Button(self.master, text="Send", command=self.send\_message)  self.send\_button.pack(pady=10)  self.server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  self.server.bind(('localhost', 5555))  self.server.listen(5)  self.clients = []  self.accept\_thread = threading.Thread(target=self.accept\_connections)  self.accept\_thread.start()  def accept\_connections(self):  while True:  client\_socket, client\_address = self.server.accept()  self.clients.append(client\_socket)  client\_thread = threading.Thread(target=self.handle\_client, | args=(client\_socket,))  client\_thread.start()  def handle\_client(self, client\_socket):  while True:  try:  message = client\_socket.recv(1024).decode('utf-8')  if message:  timestamp = datetime.now().strftime("%H:%M:%S")  formatted\_message = f"[{timestamp}] Client: {message}"  self.message\_area.insert(tk.END, formatted\_message + "\n", "client")  except (socket.error, ConnectionResetError):  self.clients.remove(client\_socket)  break  def send\_message(self):  message = self.entry\_var.get()  if message:  timestamp = datetime.now().strftime("%H:%M:%S")  formatted\_message = f"[{timestamp}] You: {message}"  self.message\_area.insert(tk.END, formatted\_message + "\n", "server")  for client\_socket in self.clients:  try:  client\_socket.send(message.encode('utf-8'))  except socket.error:  pass  self.entry\_var.set("")  def main():  root = tk.Tk()  app = ChatApp(root)  root.mainloop()  if \_\_name\_\_ == "\_\_main\_\_":  main() |

**Output:**

A screenshot of a chat application

Description automatically generated