**Task No. 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. The objective is to demonstrate a basic understanding of concurrent programming by developing a program that creates multiple threads, each performing random deposit and withdrawal operations on different accounts. The evaluation will focus on the correctness of account operations, the implementation of proper synchronization mechanisms, and the ability to showcase concurrent operations through the program's output.

**Solution:**

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

new\_balance = current\_balance + amount

time.sleep(0.1)

self.balance = new\_balance

print(f"Deposited ${amount}. New balance: ${new\_balance}")

def withdraw(self, amount):

with self.lock:

current\_balance = self.balance

if amount > current\_balance:

print("Insufficient funds!")

else:

new\_balance = current\_balance - amount

time.sleep(0.1)

self.balance = new\_balance

print(f"Withdrew ${amount}. New balance: ${new\_balance}")

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(10, 100)

if operation == "deposit":

account.deposit(amount)

else:

account.withdraw(amount)

time.sleep(0.05)

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

num\_accounts = 3

num\_operations\_per\_account = 5

accounts = [BankAccount() for \_ in range(num\_accounts)]

threads = []

for account in accounts:

thread = threading.Thread(target=perform\_operations, args=(account, num\_operations\_per\_account))

threads.append(thread)

thread.start()

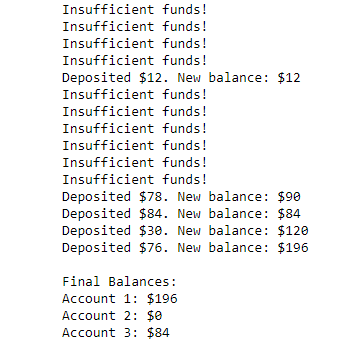
for thread in threads:

thread.join()

print("\nFinal Balances:")

for i, account in enumerate(accounts):

print(f"Account {i + 1}: ${account.get\_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 queue

import time

class MessageQueue:

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

self.messages = queue.Queue()

self.lock = threading.Lock()

def send\_message(self, message):

with self.lock:

self.messages.put(message)

print(f"Sent message: {message}")

time.sleep(0.1) # Simulating some processing time

def receive\_message(self):

with self.lock:

if not self.messages.empty():

message = self.messages.get()

print(f"Received message: {message}")

time.sleep(0.1) # Simulating some processing time

return message

else:

print("No messages in the queue.")

return None

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

for i in range(num\_messages):

queue.send\_message(f"Message {i + 1}")

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

for \_ in range(num\_messages):

message = queue.receive\_message()

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

message\_queue = MessageQueue()

num\_messages = 5

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

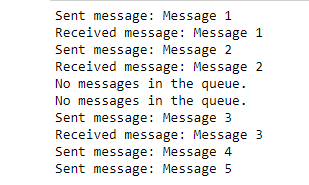
receiver\_thread = threading.Thread(target=receive\_messages, args=(message\_queue, num\_messages))

sender\_thread.start()

receiver\_thread.start()

sender\_thread.join()

receiver\_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:**

**Server Code**

import socket

import threading

import tkinter as tk

from tkinter import scrolledtext

import queue

from datetime import datetime

class Server:

def \_\_init\_\_(self, host, port):

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

self.server\_socket.bind((host, port))

self.server\_socket.listen(5)

self.clients = []

self.gui = ServerGUI(self)

def start(self):

server\_thread = threading.Thread(target=self.run\_server)

server\_thread.start()

self.gui.run()

def run\_server(self):

while True:

client\_socket, addr = self.server\_socket.accept()

client\_thread = threading.Thread(target=self.handle\_client, args=(client\_socket, addr))

client\_thread.start()

self.clients.append((client\_socket, addr))

def handle\_client(self, client\_socket, addr):

try:

while True:

data = client\_socket.recv(1024)

if not data:

break

message = f"{addr}: {data.decode('utf-8')}"

self.gui.queue.put(f"{message}")

self.broadcast(message, client\_socket)

except Exception as e:

print(f"Error: {e}")

finally:

client\_socket.close()

self.clients.remove((client\_socket, addr))

def broadcast(self, message, sender\_socket):

for client\_socket, \_ in self.clients:

if client\_socket != sender\_socket:

try:

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

except Exception as e:

print(f"Error broadcasting to client: {e}")

class ServerGUI:

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

self.server = server

self.window = tk.Tk()

self.window.title("Server")

self.chat\_log = scrolledtext.ScrolledText(self.window, state='disabled')

self.chat\_log.grid(row=0, column=0, padx=10, pady=10, columnspan=10)

self.message\_entry = tk.Entry(self.window, width=60)

self.message\_entry.grid(row=1, column=0, padx=0, pady=10)

self.send\_button = tk.Button(self.window, text="Broadcast", command=self.send\_message)

self.send\_button.grid(row=1, column=1, padx=10, pady=10)

self.queue = queue.Queue()

self.window.after(100, self.check\_queue) # Start checking the queue

def send\_message(self):

message = self.message\_entry.get()

if message:

self.server.broadcast(f"Server: {message}", None)

self.update\_chat\_log(f"Server: {message}")

self.message\_entry.delete(0, 'end')

def update\_chat\_log(self, message):

self.window.after(0, self.\_update\_chat\_log, message)

def \_update\_chat\_log(self, message):

timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

self.chat\_log.configure(state='normal')

self.chat\_log.insert('end', message + ' ' + timestamp + '\n')

self.chat\_log.configure(state='disabled')

self.chat\_log.yview('end')

def check\_queue(self):

try:

while True:

message = self.queue.get\_nowait()

self.\_update\_chat\_log(message)

except queue.Empty:

pass

self.window.after(100, self.check\_queue)

def run(self):

self.window.mainloop()

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

server = Server('localhost', 5556)

server.start()

**Client Code**

import socket

import threading

import tkinter as tk

from tkinter import scrolledtext

from datetime import datetime

class Client:

def \_\_init\_\_(self, host, port, name):

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

self.client\_socket.connect((host, port))

self.gui = ClientGUI(self, name)

def start(self):

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

receive\_thread.start()

self.gui.run()

def receive\_messages(self):

while True:

try:

data = self.client\_socket.recv(1024)

if not data:

break

message = data.decode('utf-8')

self.gui.update\_chat\_log(message)

except Exception as e:

print(f"Error receiving message: {e}")

break

def send\_msg(self, message):

try:

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

except Exception as e:

print(f"Error sending message: {e}")

class ClientGUI:

def \_\_init\_\_(self, client, name):

self.name = name

self.client = client

self.window = tk.Tk()

self.window.title(name)

self.chat\_log = scrolledtext.ScrolledText(self.window, state='disabled')

self.chat\_log.grid(row=0, column=0, padx=10, pady=10, columnspan=10)

self.message\_entry = tk.Entry(self.window, width=60)

self.message\_entry.grid(row=1, column=0, padx=0, pady=10)

self.send\_button = tk.Button(self.window, text="Send >", command=self.send\_message)

self.send\_button.grid(row=1, column=1, padx=10, pady=10)

def send\_message(self):

name = self.name

message = self.message\_entry.get()

if message:

self.client.send\_msg(message)

self.update\_chat\_log(f"{name}: {message}")

self.message\_entry.delete(0, 'end')

def update\_chat\_log(self, message):

self.window.after(200, self.\_update\_chat\_log, message)

def \_update\_chat\_log(self, message):

timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

self.chat\_log.configure(state='normal')

self.chat\_log.insert('end', message + ' ' + timestamp + '\n')

self.chat\_log.configure(state='disabled')

self.chat\_log.yview('end')

def run(self):

self.window.mainloop()

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

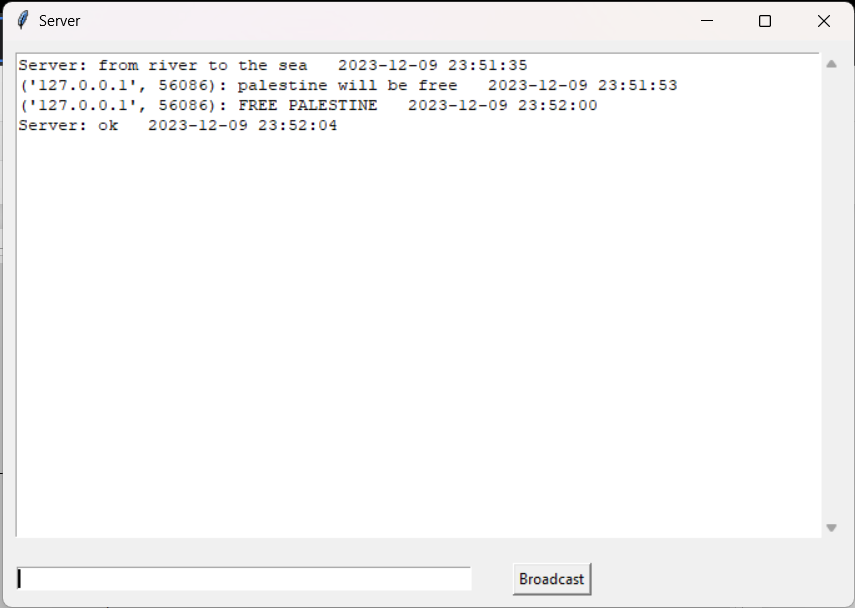
client = Client('localhost', 5556, 'Client')

client.start()

client2 = Client('localhost', 5556, 'Client')

client2.start()

**Output:**

****

**A screenshot of a computer

Description automatically generated**