IARE

IARE
INSTITUTE OF AERONAUTICAL ENGINEERING
(An Autonomous Institute affiliated to JNTUH, Hyderabad)
Dundigal, Hyderabad - 500 043

LABORATORY WORK BOOK

Name of the Student : RACHERLA SANTHOSH

Class : IT-B Semester : 03

Course Code : ACSD10 Course Name : OS Laboratory

Name of the Course Faculty : Mr. N. Raghava Rao Faculty ID : IARE10924

Exercise Number : 09 Week Number : 09 Date : 22/11/2024

Roll Number									
2	3	9	5	1	A	1	2	G	3

S. No.	Exercise Number	EXERCISE NAME	MARKS AWARDED						
			Aim/ Preparation	Algorithm / Procedure / Performance in the Lab		Source Code / Calculations and Graphs	Program Execution / Results and Error Analysis	Viva - Voce	Total
			4	4		4	4	4	20
1	9.1	The Tale of the Library Management System	4		4	4	4	3	19
2	9.2	The Tale of the Restaurant Reservation System							
3	9.3	The Tale of the Online Shopping cart System							
4	9.4	The tale of the Collaborative Document Editing							
5	9.5	The Tale of the Bank Account Management.							
6									
7									
8									
9									
10									
11									
12									

Santhosh

Signature of the Student

Signature of the Faculty

9. Concurrency Control.

9.1 The Tale Of the Library Management System

AIM :- Write a Program for a Unilib to manage its book inventory and handle user requests.

PROGRAM :-

```
import threading
import time
import random
class Book :
 def __init__(self, title):
   self.title = title
   self.is_available = True
   self.lock = threading.Lock()
 def checkout(self):
  """ Attempt to checkout the book, return False if
     already checked out."""
  with self.lock :
   if self.is_available :
     self.is_available = False
     return True
```

```
        else:
            return False
class Library:
    def __init__(self):
        self.books = {}
        self.book_lock = threading.Lock()
    def add_book(self, book):
        """Add a book to the inventory"""
        with self.book_lock:
            self.books[book.title] = book
    def return_book(self, title):
        """Handle the return process."""
        with self.book_lock:
            if title in self.books:
                book = self.books[title]
                if book.return_book():
                    Print(f"Book '{title}' returned Successfully.")
                    return True
                else:
                    Print(f"Book '{title}' was not checked out")
                    return False
```

ROLL NUMBER :

```
        else:
            print(f"Book '{title}' not found in inventory")
            return False

def simulate_user_action(library, action, book_title):
    """Simulate a user performing an action"""
    time.sleep(random.uniform(0.5, 2))
    if action == "checkout":
        library.checkout_book(book_title)
    elif action == "return":
        library.return_book(book_title)

if __name__ == "__main__":
    library = Library()
    book1 = Book("Python Programming")
    book2 = Book("Data Structures and Algorithms")
    library.add_book(book1)
    library.add_book(book2)
    for thread in threads:
        thread.join()
    print("Simulation finished")
```

OUTPUT :- Executed.

ROLL NUMBER :

9.2 The Table Of the Restaurant Reservation System.

AIM :- Write a program to restaurant named Gourmet Haven to manage table bookings.

PROGRAM :-

```
import threading
import time
import random
class Table :
  def __init__(self, table_id):
    self.table_id = table_id
    self.is_reserved = False
    self.version = 1
    self.lock = threading.Lock()
  def reserve_optimistic(self, customer_name):
    """Attempt to reserve the table using optimistic
    locking (check version)."""
    if self.is_reserved :
      Print(f" Table {self.table_id} is already reserved.
            Reservation failed for {customer_name} ")
```

```
        return False
class ReservationSystem:
    def __init__(self, num_tables):
        self.tables = [Table(i) for i in range(1, num_tables + 1)]
        self.system_lock = threading.Lock()
    def book_table(self, customer_name, optimistic = False):
        """Attempt to book a table for a customer."""
        with self.system_lock:
            available_table = None
            for table in self.tables:
                if not table.is_reserved:
                    available_table = table
                    break
    def cancel_reservation(self, customer_name):
        """Cancel the reservation for the customer."""
        with self.system_lock:
            for table in self.tables:
                if table.is_reserved:
                    table.release()
                    print(f"Reservation canceled for {customer_name}")
```

```
        return True
    Print (f"No reservation found for {customer_name}")
    return False
if __name__ == "__main__":
    reservation_system = ReservationSystem(num_tables = 10)
    customers = ["Alice", "Bob", "Charlie", "David", "Eve",
                 "Frank", "Grace", "Hannah", "Issac",
                 "Jack"]
    threads = []
    for customer in customers:
        action = random.choice(["optimistic", "persimistic"])
        optimistic = True if action = "optimistic" else False
        thread = threading.Thread(target = simulate_user_
        action, args = (reservation_system, customer, optimistic))
        threads.append(thread)
        thread.start()
        for thread in threads:
            thread.join();
        Print ("Reservation simulation completed")
```

OUTPUT :- Executed successfully.

ROLL NUMBER :

9.3 The Tale Of the Online Shopping Cart System.

AIM :- Write a Program for ShopEase that allows customers to add items to their Shopping cart & Proceed to checkout.

PROGRAM :-

```
import threading
import time
import random
class InventoryItem:
    def __init__(self, item_id, name, quantity):
        self.item_id = item_id
        self.name = name
        self.quantity = quantity
        self.lock = threading.Lock()
class ShoppingCart:
    def __init__(self, cart_id):
        self.cart_id = cart_id
        self.items = {}
```

```
    def add_item(self, item, quantity):
        """Add an item to it to shopping cart,
        ensuring synchronization"""
        with self.lock:
            if item.item_id in self.items:
                self.items[item.item_id] += quantity
            else:
                self.items[item.item_id] = quantity
            Print(f"Added {quantity} of {item.name} to
                cart {self.cart_id}. Current quantity:
                {self.items[item.item_id]}")

class ShopEase:
    def __init__(self, inventory):
        self.inventory = {item.item_id: item for item in
                          inventory}
        self.cart_locks = {}
        self.transaction_lock = threading.lock()
    def Process_transactions(self, cart, add_items,
                             remove_items):
```

```
""" Process the entire transaction automatically """
with self.transaction_lock:
    if cart.cart_id in self.cart_locks:
        Print(f"Cart {cart.cart_id} is currently being
        modified. Transaction aborted")
        return False
    self.cart_locks[cart.cart_id] = cart.lock
    try:
        for item, quantity in add_items.items():
            if not self.inventory[item.item_id].update_
            quantity(-quantity):
                Print("Transaction failed: Not enough memory")
                return False
        cart.add_item(item, quantity)
        for item, quantity in remove_items.items():
            cart.remove_item(item, quantity)
            self.inventory[item.item_id].update_quantity
            (quantity)
```

```
Print ( f " Transaction for cart {cart. cart_id}
    completed successfully.")
return True
except Exception as e :
Print (f" Error during transaction : {e}")
return False
finally :
def self . cart_locks [cart . cart_id]
if __name__ == "__main__":
inventory = [
    InventoryItem ( 1, " Laptop", 10),
    InventoryItem ( 2, " Headphones", 20),
    InventoryItem ( 3, " Keyboard", 30),
    InventoryItem ( 4, " Mouse", 15),
    InventoryItem ( 5, " Monitor", 5),
]
Shop = ShopEase (inventory)
cart 1 = Shopping Cart (1)
cart 2 = Shopping Cart (2)
```

```
cart3 = ShoppingCart(3)
for thread in threads:
    thread.start()
for thread in threads:
    thread.join()
Print("Shopping Simulation completed")
```

OUTPUT :- The Program is Executed Successfully.

9.4 The Tale Of the Collaborative Document Editing System.

AIM :- Write a program for Edit Together to allow multiple Users to edit the Same document Similarly.

PROGRAM :-

```
import threading
import time
import random
class Document:
    def __init__(self, content):
        self.sections = content.split("\n")
```

```
        self.locks = [threading.Lock() for _ in self.sections]
        self.versions = [0] * len(self.sections)

    def get_section(self, section_id):
        """Get the content of a specific section"""
        return self.sections[section_id]

class User:
    def __init__(self, user_id, document):
        self.user_id = user_id
        self.document = document
        self.edits = {}

    def edit_section(self, section_id, new_content):
        """Simulate editing a document section"""
        current_version = self.document.versions[section_id]
        print(f"User {self.user_id} is editing section
            {section_id}, current version: {current_version}")
        self.document.lock_section(section_id)
        try:
            time.sleep(random.uniform(0.5, 1.5))
            if not self.document.update_section:
```

ROLL NUMBER :

```
        Print(f" User {self.user_id} failed to update Section)
        Return False
    self.edits [section_id] = (new_content, current_version+1)
    Print(f"User {self.user_id} Successfully edited Section")
    return True
finally:
    self.document.unlock_Section(section_id)
class EditTogether:
if __name__ == "__main__":
    document_content = """ This is first Section of document.
    This is the Second Section of the document
    This is the third Section of the document."""
    for thread in threads:
        thread.start()
    for thread in threads:
        thread.join()
    Print("Document editing Simulation completed")
```

OUTPUT :-

The Program Executed Successfully.

ROLL NUMBER :

9.5 The Tale Of the Bank Account Management System.

AIM :- Write a Program for SecureBank to manage an automated System customer accounts.

PROGRAM :-

```
import threading
import time
import random
class Account:
  def __init__(self, account_id, initial_balance=0):
    self.account_id = account_id
    self.balance = initial_balance
    self.lock = threading.lock()
    self.version = 0
  def deposit(self, amount, version):
  def withdraw(self, amount, version):
class Bank:
  def __init__(self):
    self.accounts = {}
    self.lock = threading.Lock()
```

```
if __name__ == "__main__":
    bank = Bank()
    bank.add_account("A001", 1000)
    bank.add_account("A002", 1500)
    threads = []
    threads.append(threading.Thread(target=simulate_transactions, args=(bank, "A001", "A002", 200)))
    threads.append(threading.Thread(target=simulate_transactions, args=(bank, "A002", "A001", 300)))
    for thread in threads:
        thread.start()
    for thread in threads:
        thread.join()
    Print(f"Final balance of Account A001: {bank.get_account('A001').get_balance()}")
    Print(f"Final balance of Account A002: {bank.get_account('A002').get_balance()}")
```

OUTPUT :-

The Program Executed Successfully.

21/11/24.