I A R E

EDUCATION FOR LIBERATION

IARE

INSTITUTE OF AERONAUTICAL ENGINEERING

(An Autonomous Institute affiliated to JNTUH, Hyderabad)
Dundigal, Hyderabad - 500 043

LABORATORY WORK BOOK

Name of the Student : RAGHERLA SANTHOSH

Roll Number									
2	3	9	5	1	A	1	2	6	3

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 : 11 Week Number : 11 Date : 29/11/2024

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	11.1	The Tale of the Bakery and its Busy Kitchen.	4	4	4	4	3	19
2	11.2	The Tale of the Busy Coffee Shop.						
3	11.3	The Tale of the Conference Room & Reservations.						
4	11.4	The Tale of the restaurant Kitchen & its limited resources.						
5	11.5	The Tale of the garden & its watering schedule.						
6								
7								
8								
9								
10								
11								
12								

Santhosh

Signature of the Student

N-R

Signature of the Faculty

11. Process Synchronization.

11.1 The Tale Of the Bakery & Its Busy Kitchen :-

AIM :- Write a Program for Sweet Delights to manage access to the oven So that no two bakers use it at the Bakery.

PROGRAM :-

```
import threading
import time
import queue
class Bakery :
  def __init__(Self):
    Self.oven_Semaphore = threading.Semaphore(1)
    Self.queue = queue.Queue()
    Self.lock = threading.lock()
  def request_oven( Self, baker_name, item):
```

ROLL NUMBER :

```
    with Self. lock :
      Self. queue. put( ( baker_name, item))
   While True :
      with Self. lock :
        if Self. queue. queue [0][0] = = baker_name :
            break
    time. Sleep (0.1)
  Self. oven_Semaphore. acquire ()
  Self. use_oven (baker_name, item)
 def use_oven ( Self, baker_name, item) :
   Print (f" {baker_name } is banking {item }")
  time. Sleep (2)
  Print (f" { baker_name } has finished baking {item }")
  Self. release_oven ( baker_name)
def release_oven ( Self, baker_name) :
  Self. oven_Semaphore. release( )
  with Self. lock :
    Self. queue. get( )
    Print (f"{baker_name} has released the oven")
  def baker_task (bakery, baker_name, item) :
    bakery. request_oven (baker_name, item)
 def Main() :
   bakery = Bakery()
   bakers = [
     ("A", "chocolate Cake'),
     ("B", "Fruit Tart"),
     ("C", "Cheese Croissant"),
   ]
  threads = []
  for baker_name, item in bakers:
    thread = threading. Thread (target= baker_task,
       args = (bakery, baker_name, item))
  threads. append (thread)
  for thread in threads:
    thread. Start()
```

```
for thread in threads :
    thread . join()
if __name__ == "__main__" :
    main()
```

OUTPUT :-

The Program executed Successfully.

11.2 The Tale Of the Busy Coffee Shop and Its Coffee Machines :-

AIM :- Write a Program for Brewed Bliss to manage access to the coffee machines so that no two baristas use the same machine at the same time.

PROGRAM :-

```
import threading
import time
import queue
class CoffeeShop :
```

```
def __init__ (Self) :
    Self. machine1_Semaphore = threading. Semaphore (1)
    Self. machine2_Semaphore = threading. Semaphore (1)
    Self. queue = queue. Queue()
    Self. lock = threading. lock()
  def release_Machines (Self, barista_name, Machine):
    if machine == 1 :
      Self. machine1_Semaphore. release()
    elif machine == 2 :
      Self. machine2_Semaphore. release()
  with Self. lock :
    Self. queue. get()
  Print (f"{barista_name} has released" Machine {machine}")
def barista_task (coffee_Shop, barista_name,
                  machine, drink) :
  coffee_Shop. request_machine (barista_name,
    machine, drink)
def main() :
```

```
coffee_shop = CoffeeShop()
baristas = [
    ("Emma", 1, "Latte"),
    ("Liam", 2, "Cappuccino"),
    ("Olivia", 1, "Espresso"),
]
threads = []
for barista_name, machine, drink in baristas:
    thread = threading.Thread(target=barista_task,
        args=(coffee_shop, barista_name, machine,
        drink))
    threads.append(thread)
for thread in threads:
    thread.start()
for thread in threads::
    thread.join()
```

```
if_name__=="__main_":
    main()
```

OUTPUT :-

The Program is executed Successfully.

11.3 **The Tale Of The Conference Room and Its Reservations :-**

AIM :- Write a Program for Tech Innovations to manage the reservations for the Conference room.

PROGRAM :-

```
import threading
import time
import queue
class Conference Room :
  def _init_(Self):
    Self.room_Semaphore = threading.Semaphore(1)
    Self.queue = queue.Queue()
    Self.lock = threading.Lock()
    def request_room(self, employee_name, start_time,
                     end_time):
      with self.lock:
         self.queue.put((employee_name, start_time,
                         end_time))
      while True:
        with self.lock:
          if self.queue.queue[0][0] == employee_name:
             break
       time.sleep(0.1)
   self.room_semaphore.acquire()
   self.use_room(employee_name, start_time,
                 end_time)
  def main():
    conference_room = ConferenceRoom()
    reservations = [
      ("A", "10:00 AM", "11:00 AM"),
      ("B", "11:00 AM", "12:00 PM"),
      ("C", "12:00 pm", "1:00 pm"),
  ]
```

```
    threads = []
    for employee_name, start_time, end_time in
        reservations :
        thread = threading . Thread (target = employee_task,
        args = (conference_room, employee_name, start_
        time, end_time))
        threads . append (thread)
    for thread in threads :
        thread . start()
    for thread in threads :
        thread . join()
if __name__ == "__main__" :
    main()
```

OUTPUT :-

The Program is executed... Successfully.

11.4 The Tale Of the Restaurant Kitchen & Its Limited Resources :-

AIM :- Write a Program for Gourmet Haven to manage access to the stove and refrigerator so that no two cooks use the same resource at the same time.

PROGRAM :-

```
import threading
import time
import queue
class Kitchen :
 def __init__ (Self) :
   Self. Stove_Semaphore = threading. Semaphore (1)
   Self. refrigerator_Semaphore = threading. Semaphore (1)
   Self. lock = threading. Lock()
   Self. queue = queue. Queue()
def request_ resources ( Self, cook_name, tasks):
    with Self.lock :
      Self.queue.put((cook_name, tasks))
   while True :
     with Self.lock :
       if Self.queue.queue [0] [0] = = cook_name:
          break
   time.Sleep (0.1)
  Self.use_resources (cook_name, tasks)
 def use_resources (Self, cook_name, tasks) :
   Self. Stove_Semaphore. acquire()
 def Cook_task (kitchen, cook_name, tasks):
   Kitchen. request_resources(cook_name, tasks)
 def main():
   Kitchen = Kitchen()
   cooks = [
     ("J", {"Stove": "pasta", "refrigerator": "cheese"}),
     ("M", {"Stove": "Stir-fry", "refrigerator": "vegetables"}),
     ("R", {"Stove": "Soup", "refrigerator": "herbs"}),
 ]
```

```
threads = []
for cook_name, tasks in cooks :
  thread = threading. Thread (target = cook_task,
     args = (kitchen, cook_name, tasks))
  threads. append (thread)
for thread in threads :
  thread. Start()
for thread in threads :
  thread. join()
if __name__ == "__main__" :
  main()
```

OUTPUT :-

The Program is executed Successfully.

11.5 The Tale Of the Garden and Its Watering Schedule :-

AIM :- Write a Program for Green Oasis to handle access to water pump fairly & efficiently.

PROGRAM :-

```
import threading
import time
import queue
class Community Garden :
  def __init__(Self) :
    Self. water_pump_Semaphore = threading. Semaphore(1)
    Self. queue = queue. Queue()
    Self. lock = threading. Lock()
  def request_pump(Self, gardener_name,
                                  watering_time) :
    with Self.lock :
      Self. queue. put ((gardener_name, watering_time))
    while True :
        with Self.lock :
            if Self.queue.queue [0] [0] == gardener_name :
                break
        time.Sleep (0.1)
    Self.use_pump (gardener_name, watering_time)
  def release_pump (Self, gardener_name) :
    Self.water_pump_Semaphore.release( )
    with Self.lock :
        Self.queue.get( )
    Print (f" {gardener_name} has released the
              water Pump ")
def gardener_task (garden, gardener_name,
                        watering_time) :
   garden.request_pump( gardener_name, watering_
                         time)
def main() :
   garden = CommunityGarden( )
```

```
gardeners = [
  ("S", 30),
  ("J", 45),
  ("O", 20),
]
threads = []
for gardener_name, watering_time in gardeners:
  thread = threading.Thread(target = gardener_task,
    args = (gardon, gardener_name, watering_time))
  threads.append(thread)
for thread in threads:
  thread.start()
for thread in threads:
  thread.join()
if __name__ == "__main__":
  main()
```

OUTPUT :- 26/11/2024

The Program is Executed Successfully.