T.C.

MANİSA CELAL BAYAR UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

PARALLEL PROGRAMMING

DINING PHILOSOPHERS

MAXIMUM NUMBER OF PHILISOPHERS

190315004 Mehmet Anıl

190315064 Burak Can Altunoğlu

190315082 Burak Yılmaz

MANİSA 2023

T.C.

MANİSA CELAL BAYAR UNIVERITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CONTENTS

1. INTRODUCTION 3

2. SOLUTION APPROACH 4

2.1 Thread Management and Synchronization 4
2.2 Visual Representation using tkinter 4
2.3 Philosopher's Behavior Definition 4

3. SIMULATION INITIALIZATION AND WORKFLOW 5

4. CODE EXPLANATION: DINING PHILOSOPHERS PROBLEM 7

4.1 Importing Libraries and Modules 7
4.2 Philosopher Class Definition 7
4.3 Creating root and Building the GUI Structure 10

5. Code Operation: Dining Philosophers Problem Simulation 12

5.1 Objective 12
5.2 Implementation Overview 12
5.3 GUI Setup 12
5.4 Philosopher Class 12
5.5 Philosopher Actions 12
5.6 Concurrency Handling 13
5.7 GUI Updates 14
5.8 Key Components 14

6. Dining Philosophers Problem: Resource Allocation 15

6.1 Philosophers 15
6.2 Forks 15
6.3 Semaphores 15
6.4 Threads 15
6.5 Maximum Number of Philosophers 15

7. Conclusion 16

1. INTRODUCTION

The provided Python code addresses the classical "Dining Philosophers" problem. The problem represents a scenario where philosophers sit around a dining table and use forks to eat, but the forks must be shared among adjacent philosophers.

The code utilizes the threading module in Python to create a thread for each philosopher, managing fork sharing using threading.Lock and threading.Semaphore. Additionally, it employs the tkinter library to visually represent philosophers, fork lines, and their respective states.

The behaviors of the philosophers involve cycles of thinking, eating, and picking up/putting down forks, each encapsulated within the Philosopher class.

2. SOLUTION APPROACH

2.1 Thread Management and Synchronization

The code employs Python's threading module to handle concurrent execution for each philosopher, ensuring that multiple philosophers can execute concurrently. The use of threading.Lock for each fork and threading.Semaphore to restrict the number of philosophers simultaneously dining ensures synchronization in accessing shared resources.

2.2 Visual Representation using tkinter

The tkinter library facilitates the graphical representation of philosophers and their actions. Each philosopher is represented as an icon on the canvas, and fork lines are drawn to indicate their connections. The graphical elements are dynamically updated to reflect changes in the philosophers' states during the simulation.

2.3 Philosopher's Behavior Definition

Philosopher Class: Defines the behavior of each philosopher as a separate thread.

dine() Method: Manages the thinking, eating, and fork-handling behaviors of the philosophers. It encapsulates the logic of transitioning between different states and updating the GUI to reflect these changes.

pick_up_forks() and put_down_forks() Methods: Control the acquisition and release of forks, managing the critical section where philosophers interact with shared resources.

3. SIMULATION INITIALIZATION AND WORKFLOW

Method Overview:

The code initializes a simulation of the Dining Philosophers problem using Python's threading and tkinter libraries. It employs a class called Philosopher to represent each philosopher and their behaviors within the dining scenario.

Workflow:

Class Definition (Philosopher):

Initialization: Each philosopher is represented by an instance of the Philosopher class, initialized with necessary attributes such as their index, forks, state, GUI callbacks, and graphical representations.

Update GUI: Handles the graphical user interface to display the state of each philosopher.

Run Method: Governs the execution of the philosopher's actions (thinking, eating, picking up forks, and putting down forks) in an infinite loop.

Dining Method: Controls the sequence of actions (think, pick up forks, eat, put down forks) for each philosopher.

Pick Up Forks Method: Manages the process of picking up the forks by the philosopher, considering constraints and interactions with neighboring philosophers.

Put Down Forks Method: Handles the process of putting down the forks by the philosopher, ensuring proper release and GUI updates.

Drawing Figures (draw_figures):

Canvas Setup: Draws the graphical elements representing philosophers, fork lines, and their states on the tkinter canvas.

Simulation Start (start_simulation):

Initialization: Sets up the simulation environment by creating locks for forks, semaphores for controlling philosopher numbers, and loading philosopher images.

Creating Philosophers: Instantiates a set of philosophers, assigning each their respective attributes and GUI elements.

Starting Threads: Initiates the threads for each philosopher to commence the simulation.

GUI Components:

Root Window: Creates a tkinter window as the interface for the simulation.

Start Button: Initiates the simulation when clicked.

Canvas and Text Widgets: Display the graphical representation and status updates of the philosophers respectively.

Visual Components:

Canvas: Used to draw and display the graphical elements representing philosophers, forks, and their interactions.

Text Widget: Shows the status updates and actions of the philosophers during the simulation.

4. CODE EXPLANATION: DINING PHILOSOPHERS PROBLEM

4.1 Importing Libraries and Modules

```python
import threading
import time
import random
import tkinter as tk
import math
```

4.2 Philosopher Class Definition

This class is used to model the behavior of philosophers. Each philosopher is a thread.

__init__(self, index, left_fork, right_fork, max_philosophers_sem, gui_callback, fork_lines, philosopher_images): Constructor of the Philosopher class. Initializes necessary variables for each philosopher.

update_gui(self): Used to update the GUI. Shows the philosopher's state and number of times they've eaten.

```python
class Philosopher(threading.Thread):
    philosophers = []

    def __init__(self, index, left_fork, right_fork, max_philosophers_sem, 
gui_callback, fork_lines, philosopher_images):
        threading.Thread.__init__(self)
        self.index = index
        self.left_fork = left_fork
        self.right_fork = right_fork
        self.max_philosophers_sem = max_philosophers_sem
        self.gui_callback = gui_callback
        self.figure = None
        self.background_circle = None
        self.state = "thinking"
        self.is_left_fork_taken = False
        self.is_right_fork_taken = False
        self.fork_lines = fork_lines
        self.eat_count = 0
        self.philosopher_images = philosopher_images

    def update_gui(self):
        self.gui_callback(f"Philosopher {self.index} is {self.state}.")
        fill_color = "black"
        if self.state == "eating":
            fill_color = "red"
            self.eat_count += 1
            if self.eat_count >= 2:
                self.state = "full"

        background_oval = 
canvas.find_withtag(f"background_circle_{self.index}")[0]
        canvas.tag_raise(background_oval)
        canvas.tag_raise(self.figure)
        canvas.itemconfig(background_oval, fill=fill_color)
        status_text_id = canvas.find_withtag(f"status_text_{self.index}")
        canvas.itemconfig(status_text_id, text=self.state.capitalize())
        root.update()
        time.sleep(2)
```

run(self): Main loop of the philosopher's thread. Simulates eating and thinking until the philosopher is 'fully fed'.

```
def run(self):
    while True:
        if self.state != "full":
            self.dine()
```

dine(self): Simulates the process of thinking and eating for the philosopher.

```
def dine(self):
    think_time = random.uniform(1, 3)
    time.sleep(think_time)
    self.state = "thinking"
    self.update_gui()
    self.pick_up_forks()
    if self.state != "full":
        eat_time = random.uniform(1, 3)
        self.state = "eating"
        self.update_gui()
        time.sleep(eat_time)
        self.put_down_forks()
        time.sleep(1)
```

pick_up_forks(self): Used to pick up forks. Acquires the left and right fork for the philosopher. Uses a semaphore to prevent deadlocks.

```
def pick_up_forks(self):
    if self.state != "full":
        self.max_philosophers_sem.acquire()
        left_neighbor_index = (self.index + 1) % len(self.fork_lines)
        right_neighbor_index = (self.index - 1) % len(self.fork_lines)
        self.right_fork.acquire()
        self.state = "picking up right fork"
        self.is_right_fork_taken = True
        self.update_gui()
        line_right = canvas.create_line(self.fork_lines[self.index][0],
self.fork_lines[self.index][1],

self.fork_lines[right_neighbor_index][0],

self.fork_lines[right_neighbor_index][1], width=2, fill="red",
                                        tags=f"line_{self.index}_right")
        canvas.tag_raise(line_right, self.figure)
        canvas.itemconfig(line_right, fill="red")
        self.left_fork.acquire()
        self.state = "picking up left fork"
        self.is_left_fork_taken = True
        self.update_gui()
        line_left = canvas.create_line(self.fork_lines[self.index][0],
self.fork_lines[self.index][1],

self.fork_lines[left_neighbor_index][0],
self.fork_lines[left_neighbor_index][1], width=2, fill="red",
                                        tags=f"line_{self.index}_left")
        canvas.tag_raise(line_left, self.figure)
        canvas.itemconfig(line_left, fill="red")
        for philosopher in Philosopher.philosophers:
            canvas.tag_raise(philosopher.figure)
        left_neighbor_figure =
canvas.find_withtag(f"philosopher_{left_neighbor_index}_icon")[0]
        right_neighbor_figure =
canvas.find_withtag(f"philosopher_{right_neighbor_index}_icon")[0]
        canvas.tag_lower(line_left, left_neighbor_figure)
        canvas.tag_lower(line_right, right_neighbor_figure)
        background_circle_left =
canvas.find_withtag(f"background_circle_{left_neighbor_index}")[0]
        background_circle_right =
canvas.find_withtag(f"background_circle_{right_neighbor_index}")[0]
        canvas.tag_lower(line_left, background_circle_left)
        canvas.tag_lower(line_right, background_circle_right)
        canvas.tag_lower(line_left, background_circle_left)
        canvas.tag_lower(line_right, background_circle_right)
```

put_down_forks(self): Used to put down forks. Releases the acquired forks.

draw_figures(canvas, philosopher_images) Function

Used to draw philosophers and fork lines on the graphical interface.

```
def put_down_forks(self):
    self.right_fork.release()
    self.is_right_fork_taken = False
    line_id_right = canvas.find_withtag(f"line_{self.index}_right")
    canvas.delete(line_id_right)
    self.left_fork.release()
    self.is_left_fork_taken = False
    line_id_left = canvas.find_withtag(f"line_{self.index}_left")
    canvas.delete(line_id_left)
    self.max_philosophers_sem.release()
    if self.state == "full":
        self.state = "full"
        fill_color = "yellow"
    else:
        self.state = "putting down forks"
        fill_color = "black"
    self.update_gui()
    canvas.itemconfig(self.background_circle, fill=fill_color)
    canvas.itemconfig(self.figure,
image=self.philosopher_images[self.index])
```

gui_callback(message): Callback function to update the GUI. Displays the philosopher's state on the screen.

```
def gui_callback(message):
    text.insert(tk.END, message + "\n")
    text.see(tk.END)
    return "eating" in message
```

start_simulation(): Starts the simulation. Creates philosophers and begins the simulation.

```
def start_simulation():
    num_philosophers = 5
    max_philosophers = 2
    forks = [threading.Lock() for _ in range(num_philosophers)]
    max_philosophers_sem = threading.Semaphore(max_philosophers)
    philosopher_images = load_images()
    figures, fork_lines = draw_figures(canvas, philosopher_images)
    philosophers = [
        Philosopher(i, forks[i], forks[(i + 1) % num_philosophers],
max_philosophers_sem, gui_callback, fork_lines,
                        philosopher_images)
        for i in range(num_philosophers)
    ]
    for i, philosopher in enumerate(philosophers):
        philosopher.figure = figures[i]
        philosopher.background_circle =
canvas.find_withtag(f"background_circle_{i}")[0]
        philosopher.start()
```

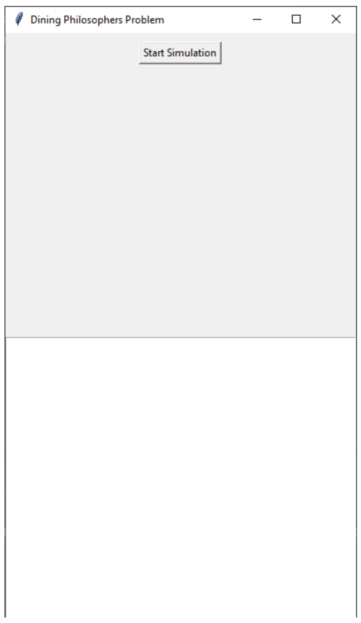
load_images(): Loads images for the philosophers.

```
def load_images():
    philosopher_images = []
    for i in range(5):
        image_path = f".\\icon_0.png"
        image = tk.PhotoImage(file=image_path)
        philosopher_images.append(image)
    return philosopher_images
```

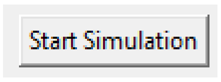
4.3 Creating root and Building the GUI Structure

This section involves creating and setting up the GUI using the tkinter library.

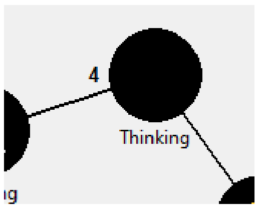
root = tk.Tk(): Creates the main window of the tkinter application.



start_button = tk.Button(root, text="Start Simulation", command=start_simulation): Creates a button to start the simulation.



canvas = tk.Canvas(root, height=300, width=300): Creates a canvas to draw philosophers and fork lines.



text = tk.Text(root, height=20, width=50): Creates a text area to display updates.

root.mainloop(): Runs the GUI and waits for user interactions.

5. Code Operation: Dining Philosophers Problem Simulation

5.1 Objective

The implemented code simulates the classic Dining Philosophers problem, demonstrating a scenario where philosophers contend for finite resources (forks) to eat while avoiding deadlocks and ensuring fair resource allocation.

5.2 Implementation Overview

The code employs Python's threading module and tkinter library for creating a visual representation of the Dining Philosophers problem. It operates as follows:

5.3 GUI Setup

The graphical user interface (GUI) is created using tkinter.

A canvas is used to draw the philosophers, fork lines, and status updates.

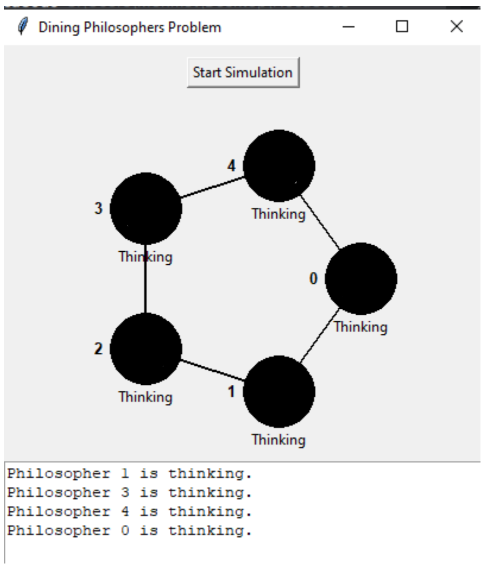
5.4 Philosopher Class

Represents individual philosophers as threads.

Each philosopher thread models the actions of thinking, eating, and managing forks.

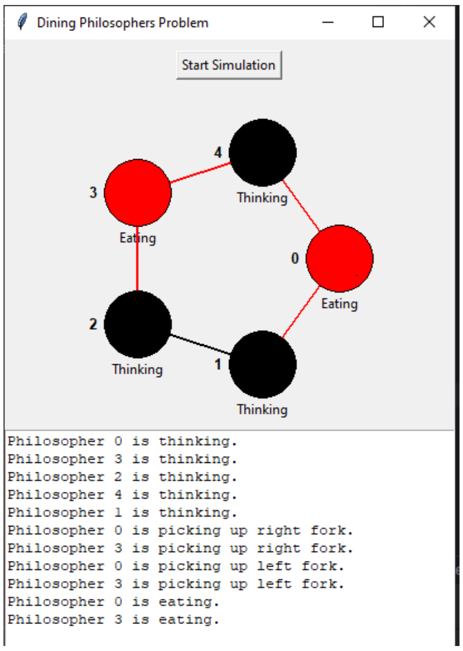
5.5 Philosopher Actions

Thinking Phase: Philosophers initially start by thinking for a random duration.



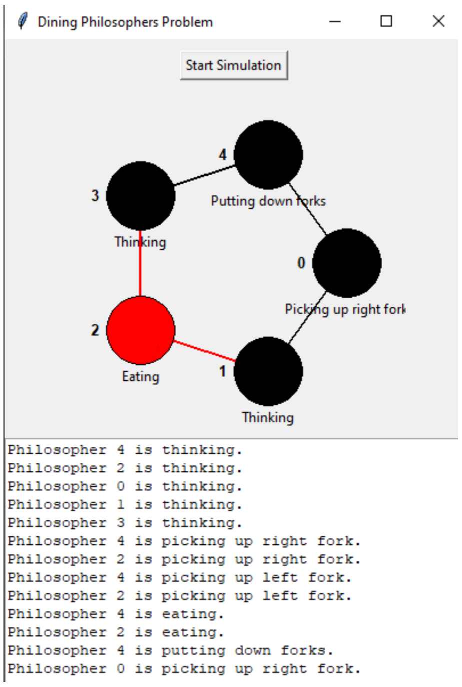
```
Philosopher 1 is thinking.
Philosopher 3 is thinking.
Philosopher 4 is thinking.
Philosopher 0 is thinking.
```

Eating Phase: Philosophers attempt to pick up two forks simultaneously to eat for a random duration.



```
Philosopher 0 is thinking.
Philosopher 3 is thinking.
Philosopher 2 is thinking.
Philosopher 4 is thinking.
Philosopher 1 is thinking.
Philosopher 0 is picking up right fork.
Philosopher 3 is picking up right fork.
Philosopher 0 is picking up left fork.
Philosopher 3 is picking up left fork.
Philosopher 0 is eating.
Philosopher 3 is eating.
```

Fork Handling: Semaphores prevent deadlock by ensuring philosophers pick up both forks when available and release them after eating.



```
Philosopher 4 is thinking.
Philosopher 2 is thinking.
Philosopher 0 is thinking.
Philosopher 1 is thinking.
Philosopher 3 is thinking.
Philosopher 4 is picking up right fork.
Philosopher 2 is picking up right fork.
Philosopher 4 is picking up left fork.
Philosopher 2 is picking up left fork.
Philosopher 4 is eating.
Philosopher 2 is eating.
Philosopher 4 is putting down forks.
Philosopher 0 is picking up right fork.
```

5.6 Concurrency Handling

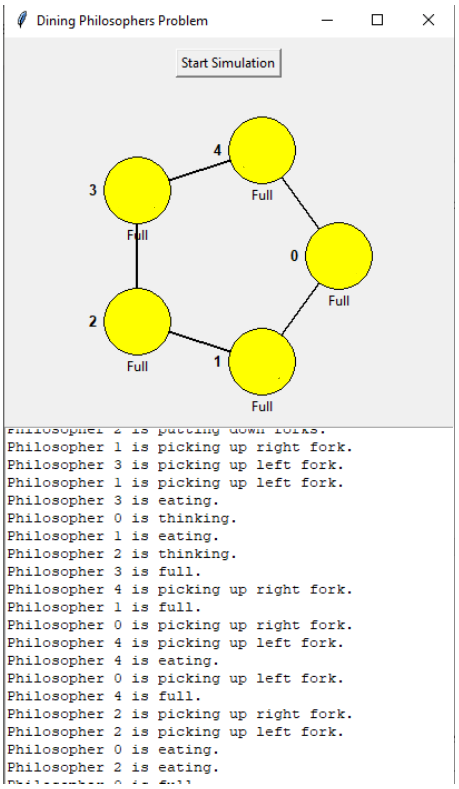
Threads manage the concurrent actions of multiple philosophers.

Semaphores regulate access to shared resources (forks) to prevent conflicts.

5.7 GUI Updates

The GUI is updated to reflect the state changes of each philosopher (thinking, eating, and status updates).

Visualization includes graphical representations of philosophers, forks, and their interactions.



```
Philosopher 2 is putting down forks.
Philosopher 1 is picking up right fork.
Philosopher 3 is picking up left fork.
Philosopher 1 is picking up left fork.
Philosopher 3 is eating.
Philosopher 0 is thinking.
Philosopher 1 is eating.
Philosopher 2 is thinking.
Philosopher 3 is full.
Philosopher 4 is picking up right fork.
Philosopher 1 is full.
Philosopher 0 is picking up right fork.
Philosopher 4 is picking up left fork.
Philosopher 4 is eating.
Philosopher 0 is picking up left fork.
Philosopher 4 is full.
Philosopher 2 is picking up right fork.
Philosopher 2 is picking up left fork.
Philosopher 0 is eating.
Philosopher 2 is eating.
Philosopher 0 is full.
```

5.8 Key Components

Philosophers: Modeled as threads, representing concurrent agents competing for finite resources.

Forks: Symbolize shared resources that philosophers require to eat.

Semaphores: Ensure controlled access to forks, preventing deadlocks by limiting concurrent access.

Threads: Model individual philosopher behaviors concurrently.

6. Dining Philosophers Problem: Resource Allocation

6.1 Philosophers

Philosophers represent concurrent agents competing for finite resources, mimicking the scenario of individuals sitting around a dining table, where each philosopher requires both left and right forks to eat.

6.2 Forks

Forks symbolize the shared resources (or forks) on the table. In the problem, each philosopher needs two forks to eat. These forks act as the limited resources that the philosophers contend for.

6.3 Semaphores

Semaphores are synchronization primitives used to control access to shared resources. In this problem, semaphores are employed to prevent deadlocks by limiting the number of philosophers that can simultaneously pick up forks. A semaphore helps ensure that a philosopher only acquires forks when both required forks are available.

6.4 Threads

Threads are used to model each philosopher's actions concurrently. Each philosopher operates as an individual thread, performing actions such as thinking, eating, picking up forks, and putting down forks. Threads simulate the concurrent nature of the philosophers' behaviors.

6.5 Maximum Number of Philosophers

The maximum number of philosophers directly influences the number of concurrent agents contending for the finite resources (forks). The Dining Philosophers problem is typically represented with a set number of philosophers, often denoted as 'N'.

7. Conclusion

In summary, the Dining Philosophers problem involves 'N' philosophers sitting around a table, each needing two forks to eat. Semaphores are utilized to coordinate access to the forks, ensuring that deadlocks are prevented. Threads model the concurrent actions of each philosopher in their pursuit of dining.