ATA CEYHUN ALACA | 220315003

Parallel Programming Term Project

1) Introduction

The project implements the Resource Hierarchy solution for the Dining Philosophers problem through utilizing threads (representing philosophers) and locks (representing forks) to prevent potential deadlocks and ensure mutual exclusion. Each philosopher is required to acquire two forks at the same time to begin eating. The Resource Hierarchy solution introduces an ordering to the resources (forks) and requires the philosophers to acquire the resources in a specific order to prevent circular waiting and deadlock. In the project, the solution is implemented implicitly in the "eat" method of the "Philosopher" class by acquiring forks in minimum index first, then maximum index order to avoid circular waiting and ensure that each philosopher can eventually acquire necessary resources to eat without leading to a deadlock. The code creates a multi-threaded simulation of the Dining Philosophers problem, illustrating how threads (philosophers) contend for resources (forks) to demonstrate synchronization and resource allocation challenges in concurrent systems. Finally, the Pygame visualization and the text-based representation complement each other to provide a comprehensive view of the problem at hand.

2) Project Implementation

Fork Class:

Represents a fork on the table.

Attributes:

"Index": Indicates the fork's unique identifier.

"lock": Utilizes Python's "threading.Lock" to manage access between philosophers.

"picked_up": Tracks if the fork is in use.

"owner": Stores the philosopher's index currently holding the fork.

Methods:

"__enter__", "__exit__": Implements context management to acquire and release the fork.

"__call__": When a philosopher attempts to pick up the fork, it assigns ownership if available.

"__str__": Returns a string representation of the fork.

"__lt__": Compares forks based on their index.

ATA CEYHUN ALACA | 220315003

Philosopher Class:

Represents a philosopher.

Attributes:

“index": Unique identifier for the philosopher.

“left_fork”, “right_fork”: References to the philosopher's adjacent forks.

“spaghetti”: Indicates the amount of spaghetti the philosopher has.

“eating”, “state”: Tracks if the philosopher is eating and their current state.

“x”, “y”: Coordinates for positioning the philosopher in the visualization.

Methods:

“run”: Main logic for the philosopher to think and eat.

“think”, “eat”: Simulates the philosopher's actions.

“__str__”: Returns a string representation of the philosopher.

“draw_philosophers” Function:

Visualizes the philosophers on a Pygame window based on their state (thinking or eating).

“animated_table_pygame” Function:

Initializes and runs the Pygame visualization of the dining philosophers problem. Utilizes Pygame to create a graphical representation of the philosophers and their states as the code executes.

“table” Function:

Displays a textual representation of the philosophers and their states in the terminal.

“main” Function:

Sets up the simulation with a certain number of philosophers (n) and spaghetti portions (m). Initializes philosophers and forks, starts their threads, and initiates the visual and textual representations.