Project Title: "Dining Philosophers Problem Simulation"

Team Members:

1. Enita Mujkanović

Executive Summary: This project aims to develop a console-based simulation of the classic Dining Philosophers Problem, a well-known synchronization challenge in operating systems. The program will represent philosophers as threads and forks as shared resources managed by semaphores. The solution will ensure that philosophers alternate between thinking and eating while preventing deadlock and starvation. The project demonstrates key concepts in process synchronization, threading, and resource management in a simple and educational manner.

Key Features:

1. **Philosopher Simulation**
   - Represents 3 philosophers using threads.
   - Philosophers alternate between "Thinking" and "Eating" states.
2. **Fork Management**
   - Forks are managed using semaphores to control access.
3. **Deadlock Prevention**
   - Implements techniques like limiting concurrent philosophers or asymmetric fork-picking strategies to avoid deadlock.
4. **Console Output**
   - Displays real-time updates of philosopher states and fork usage.

Technologies Used:

- **Programming Language**: Python.
- **Libraries**: Threading module (for threads) and Semaphore (for synchronization).

Proposed Implementation:

1. **Philosopher Class**:
   - Encapsulates philosopher behavior (thinking, picking forks, eating, releasing forks).
   - Each philosopher operates in an infinite loop simulating their lifecycle.
2. **Forks as Resources**:
   - Forks are represented as semaphores, ensuring exclusive access.
   - Philosopher threads attempt to acquire two forks (left and right) before eating.
3. **Deadlock Prevention**:

    - Limits the number of philosophers picking forks simultaneously to .
    - Alternatively, uses an asymmetric strategy for fork acquisition (odd philosophers pick left first, even pick right first).

4. **Console Interface**:
    - Logs state changes (e.g., "Philosopher 1 is eating," "Philosopher 2 is thinking").

Benefits:

- Provides hands-on experience with process synchronization.
- Demonstrates concepts like semaphores, threads, and deadlock prevention.
- Offers a visual and interactive way to learn operating system fundamentals.

Timeline:

- Week 1: Research and design the solution.
- Week 2: Develop core functionality (threads, semaphores, philosopher lifecycle).
- Week 3: Implement deadlock prevention strategies and console output.
- Week 4: Test and finalize the program.

Expected Outcome: A working console application that accurately simulates the Dining Philosophers Problem, demonstrating proper synchronization and deadlock-free execution.