DINING PHILOSOPHER's & READER-WRITER's SIMULATION

A MINI-PROJECT REPORT

Submitted by

PRIYANGA M 231901037

DHARSHNA U 231901008

In partial fulfillment of the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING(CYBER SECURITY)

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI

MAY 2025

BONAFIDE CERTIFICATE

Certified that this project "**DINING PHILOSOPHER's & READER-WRITER's SIMULATION**" is the bonafide work of "**PRIYANGA M**, **DHARSHNA U**" who carried out the project work under my supervision.

This mini project report is submitted for the viva voce examination to be held on _____

SIGNATURE
MRS V JANANEE
ASSISTANT PROFESSOR
Dept. of Computer Science and Engg,
Rajalakshmi Engineering College
Chennai

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

This project simulates two core OS concepts: **Deadlock Avoidance in Railway Track Allocation** and the **Reader-Writer Synchronization Problem**, using Python multithreading, semaphores, and a Tkinter GUI.

The **Railway Track Allocation** module models train scheduling with shared track access, using semaphores and Banker's Algorithm to avoid deadlock. Users can input train data, assign priorities, and view live animations with performance graphs.

The **Reader-Writer Simulation** supports both reader- and writer-priority modes, illustrating synchronization, fairness, and race condition prevention through animated threads and live stats.

Together, these modules provide an interactive platform to explore key OS concepts like multithreading, synchronization, and resource management—ideal for academic learning and demonstrations.

ACKNOWLEDGEMENT

We express our sincere thanks to our Head of the Department **Mr. Benedict JN**, for encouragement and being ever supporting force during our project work.

We also extend our sincere and hearty thanks to our internal guide **Mrs.V.JANANEE**, for her valuable guidance and motivation during the completion of this project.

Our sincere thanks to our family members, friends and other staff members of computer science engineering.

DHARSHNA U 231901008

PRIYANGA M 231901037

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Efficient synchronization and safe resource allocation are fundamental in modern operating systems to ensure performance and stability. This project presents an interactive simulation combining two classical concurrency problems—the **Reader-Writer Problem** and the **Railway Track Allocation Problem**—developed using **Python multithreading** and a **Tkinter-based GUI**.

The **Reader-Writer Simulation** demonstrates how multiple threads access a shared resource, where readers can operate concurrently, but writers require **exclusive access**. It supports both **Reader-Priority** and **Writer-Priority** modes and uses **locks and condition variables** to coordinate access. Real-time visual feedback and a **live performance graph** provide insights into scheduling efficiency.

The **Railway Track Allocation Simulation**, inspired by the **Dining Philosophers Problem**, models trains as threads competing for **two adjacent tracks**. Access is managed through **mutex locks and semaphores**, with the **Banker's Algorithm** ensuring **deadlock-free execution**. Users can set train names and priorities, and observe animated train movements alongside real-time efficiency tracking.

Together, these simulations offer a practical, interactive platform to explore key concepts in **concurrent programming**, **deadlock avoidance**, and **resource management**, making the project ideal for academic learning and demonstrations.

1.1. Key Features

- **Thread Synchronization**: Utilizes locks, semaphores, and condition variables to coordinate safe access to shared resources.

- **Dual Priority Modes**: Enables both reader-priority and writer-priority configurations for evaluating synchronization strategies.

- **Deadlock Prevention**: Integrates the Banker's Algorithm to ensure resource allocations remain in a safe state, preventing deadlocks.

- **Customizable Simulation**: Users can define the number of readers, writers, trains, and assign dynamic priorities.

- **Live Performance Graphs**: Real-time visualization of efficiency metrics, including waiting and movement times using Matplotlib.

- **Real-Time Visualization**: Interactive GUI displays thread states and transitions with animated indicators for active and waiting processes.

- **Efficiency Reporting**: Summarizes total and average performance metrics to assess system throughput and responsiveness.

- **Educational Value**: Bridges theoretical concepts with practical simulation, ideal for learning and teaching core OS-level topics.

CHAPTER 2

SCOPE OF THE PROJECT

This project delivers a comprehensive suite of simulations—**Railway Track Allocation** and the **Reader-Writer Problem**—each designed to illustrate core operating system principles such as synchronization, scheduling, deadlock handling, and performance analysis in concurrent environments. These simulations bridge theoretical foundations with practical application, offering an interactive and visually engaging platform for education, experimentation, and research.

Concurrent Systems Modeling

- Simulates multiple threads (trains or readers/writers) competing for shared resources, highlighting synchronization challenges in real-world concurrent systems.

- In the Railway Simulation, trains request adjacent tracks for movement, representing critical section entry.

- In the Reader-Writer Simulation, multiple readers and writers interact with a shared resource under varying access constraints.

Deadlock Prevention and Scheduling Strategy

- The Railway Simulation integrates the Banker's Algorithm to preemptively avoid deadlocks, ensuring resource allocation only proceeds in a safe sequence.

- The Reader-Writer Simulation allows toggling between Reader-Priority and Writer-Priority scheduling, helping users analyze fairness, starvation, and responsiveness under

different policies.

Interactive GUI and Visualization

- Built using Tkinter, both simulations feature intuitive graphical interfaces that dynamically reflect thread states (e.g., waiting, active, moving).

- Tracks or shared resources are visually represented, aiding comprehension of concurrent activity and synchronization behavior.

Live Performance Monitoring

- Real-time graphs powered by Matplotlib track metrics like waiting time, active time, and throughput.

- Efficiency reports offer valuable insights into system responsiveness and help users assess operational effectiveness.

Custom Configuration and Flexibility

- Users can customize:

  - Number and names of trains/readers/writers

  - Train priorities and scheduling modes

  - Execution behavior and simulation speed

- This adaptability supports diverse experimental setups for in-depth learning and scenario testing.

Academic and Research Relevance

- Ideal for students, educators, and researchers studying:
  - Mutual exclusion and critical section handling
  - Resource allocation and deadlock avoidance
  - Scheduling fairness and thread prioritization
  - Starvation scenarios in multi-threaded environments
- Both simulations serve as foundational tools for extending into advanced topics like real-time systems, priority inversion, and dynamic resource management.

CHAPTER 3

ARCHITECTURE

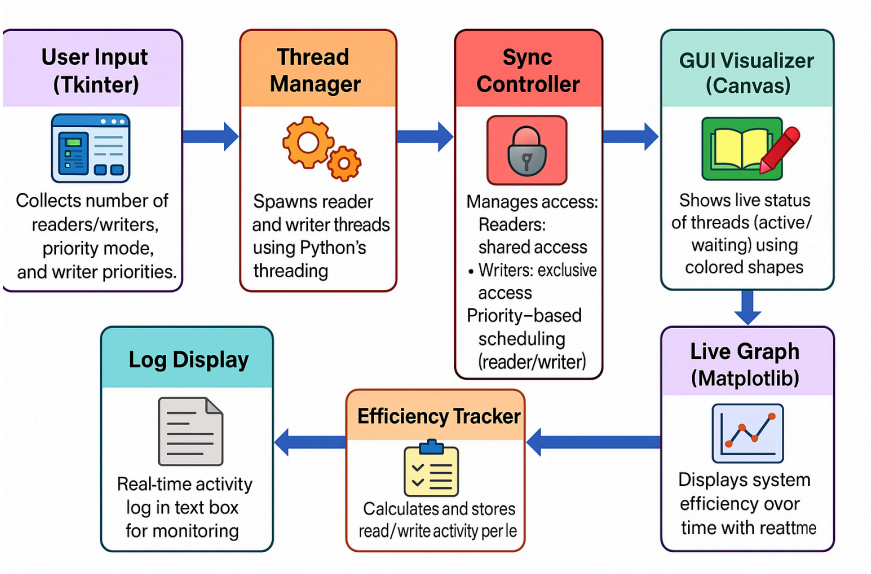

Figure 1. Reader-Writer's System Architecture

- **User Input (Tkinter)**

Collects number of readers/writers, priority mode, and writer priorities.

- **Thread Manager**

Spawns reader and writer threads using Python's threading.

- **Sync Controller (Condition + Lock)**

Manages access:

  - Readers: shared access

  - Writers: exclusive access

- Priority-based scheduling (reader/writer)

- **GUI Visualizer (Canvas)**

Shows live status of threads (active/waiting) using colored shapes.

- **Log Display**

Real-time activity log in text box for monitoring.

- **Efficiency Tracker**

Calculates and stores read/write activity per cycle.

- **Live Graph (Matplotlib)**

Displays system efficiency over time with real-time updates.

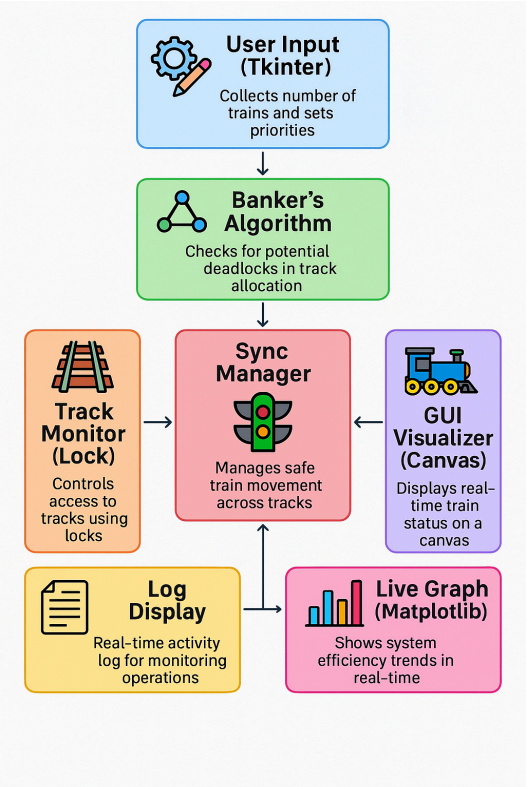

Figure 2. Train track allocation System Architecture

The system consists of several key modules:

- **User Interaction:** Uses a Tkinter GUI to get train details and display logs and reports.

- **Configuration:** Collects the number of trains, names, and priorities.

- **Deadlock Detection:** Implements the Banker's Algorithm to ensure safe resource allocation.

- **Resource Management:** Uses semaphores to manage access to track segments.

- **Thread Management:** Runs each train on a separate thread to simulate parallel movement.

- **Movement and Logging:** Animates train movement on a canvas while recording waiting and moving times.

- **Reporting:** Provides efficiency reports and live graphs to visualize performance.

CHAPTER 4

FLOWCHART

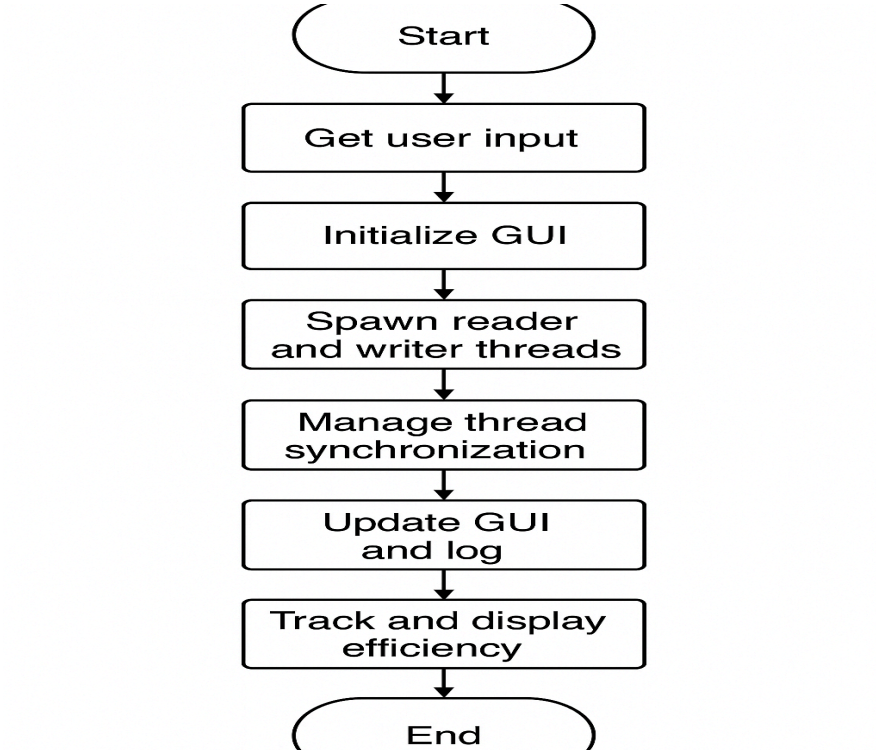

Figure 3: Flowchart of Train Simulation Using Deadlock Avoidance

- **Start** – Launch app and GUI.
- **User Input** – Enter readers, writers, mode, priorities.
- **Init Threads** – Create and start reader/writer threads.
- **Loop Execution** – Threads try to access the resource.
- **Priority Check** – Sync access based on selected mode.
- **Resource Access** – Readers (shared) / Writers (exclusive).
- **GUI & Graph Update** – Animate status, track efficiency.
- **Repeat / End** – Loop continues until closed.

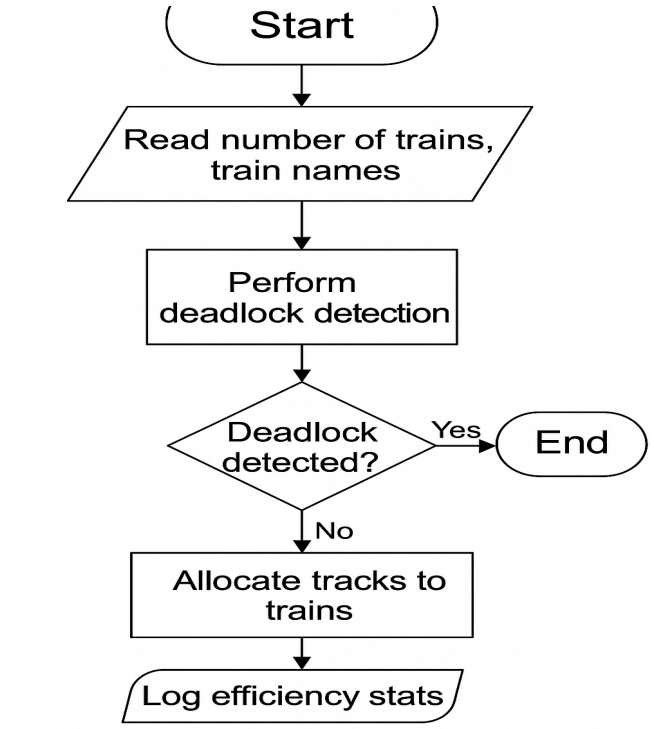

Figure 4: Flowchart of Reader-Writer Simulation

- **Start** – Launch the simulation.
- **Input Trains** – User enters number and names of trains.
- **Deadlock Check** – Banker's Algorithm ensures safe track allocation. If not safe, exit.
- **Initialize Tracks** – Create semaphores and locks for track control.
- **Set Priorities** – User assigns train priorities (optional).
- **Start Threads** – Each train runs in a separate thread.
- **Train Behavior Loop**
  - Wait for tracks
  - Acquire tracks

- Move train
- Release tracks

- **Log Activity** – Record actions in a live log window.
- **Track Efficiency** – Monitor waiting and moving times.
- **Show Report/Graph** – Display performance report or graph on user request.
- **End** – Simulation ends when GUI is closed.

CHAPTER 5
CODE IMPLEMENTATION

5.1. PROGRAM

5.1.1. READER-WRITER's SIMULATION

```
def __init__(self, canvas, x, y, name, priority):
    # Initializes a writer or reader with position, name, and priority on the canvas

def get_color(self):
    # Returns active/inactive color for visual status on canvas

def set_active(self, active):
    # Sets the writer's active state (used to toggle writing mode)

def render(self):
    # Updates writer/reader visuals (icon, emoji, label, caption) on canvas

def update_state(self, writer_active):
    # Updates reader state depending on writer's activity (reading/waiting)

def __init__(self, root, num_readers, num_writers, mode):
    # Initializes the simulation environment, GUI components, and variables

def log(self, message):
    # Logs events to the GUI log box

def export_log(self):
    # Saves the simulation log as a .txt file

def reset_simulation(self):
    # Resets simulation state and requests fresh user input for readers/writers/mode
```

```
def setup_simulation(self):
    # Sets up initial readers and writers based on input and begins the simulation

def setup_writers(self, num_writers):
    # Creates writer objects and places them on the canvas

def setup_readers(self, num_readers):
    # Creates reader objects and places them on the canvas

def resolve_initial_deadlock(self):
    # Resolves any potential deadlock at the simulation start by activating a writer

def update_simulation(self):
    # Main loop: periodically checks and updates reader/writer states

def activate_writer(self):
    # Activates the highest-priority writer if no readers are active

def deactivate_writer(self):
    # Deactivates current writer and triggers reader activation

def activate_readers(self):
    # Activates all readers in order of priority

def deactivate_readers(self):
    # Stops all readers and triggers writer activation

def check_deadlock(self):
    # Checks for deadlock (no readers/writers active) and triggers resolution

def resolve_deadlock(self):
    # Resolves deadlock by activating a writer
```

5.1.2. DINING PHILOSOPHERs SIMULATION

```
def bankers_algorithm():
    # Allocates tracks to trains using Banker's Algorithm to prevent deadlock

def log_message(message):
    # Appends a message to the on-screen simulation log

def set_priorities():
    # Opens a pop-up window to collect priority input for each train

def draw_track(y_pos):
    # Draws a horizontal line on canvas to represent a train track

def draw_train(x, y, train_id, color):
    # Places a train icon and label at the given position on the canvas

def move_train(train_id, start_x, end_x, y_pos, color):
    # Animates train movement along the track and calculates move time

def train_routine(train_id):
    # Manages a single train's flow: request track, move, release track

def show_efficiency_report():
    # Displays average wait time, move time, and performance stats

def show_efficiency_graph():
    # Visualizes wait time vs move time using a bar graph
```

5.2. SCREENSHOTS

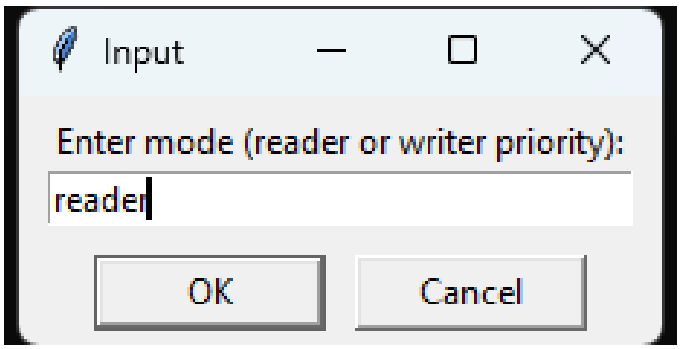

Figure 5. Choosing priority for reader or writer

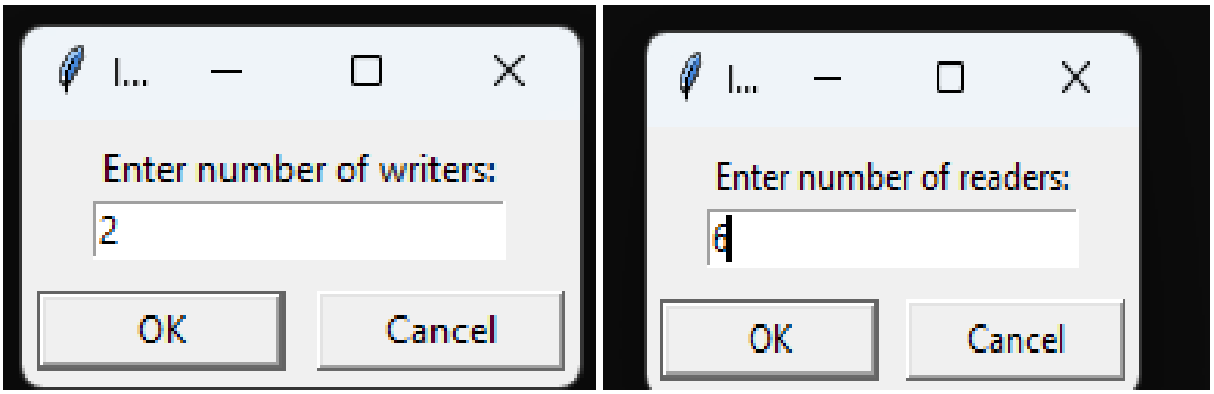

Figure 6. Choosing number of readers and writers

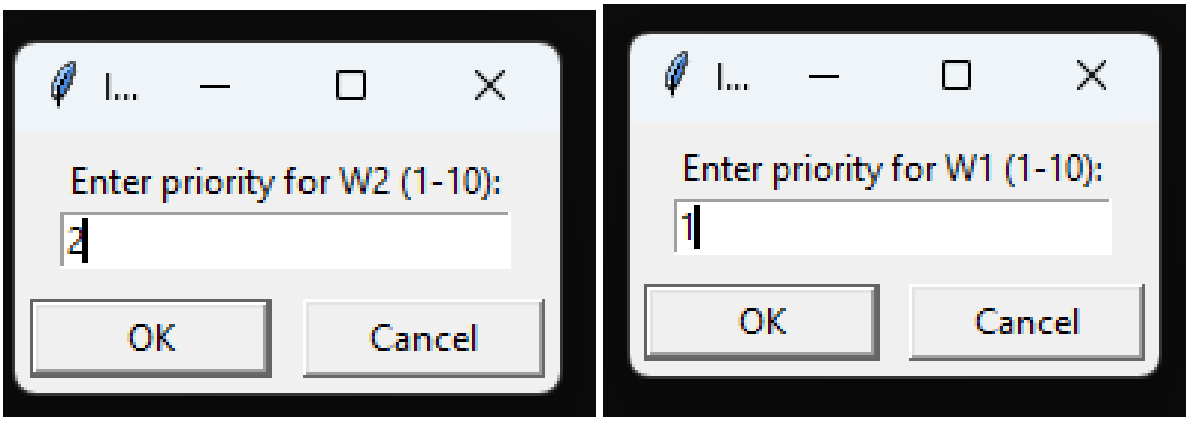

Figure 7. Setting priority for writer

Figure 8. Simulation of Reader-Writer

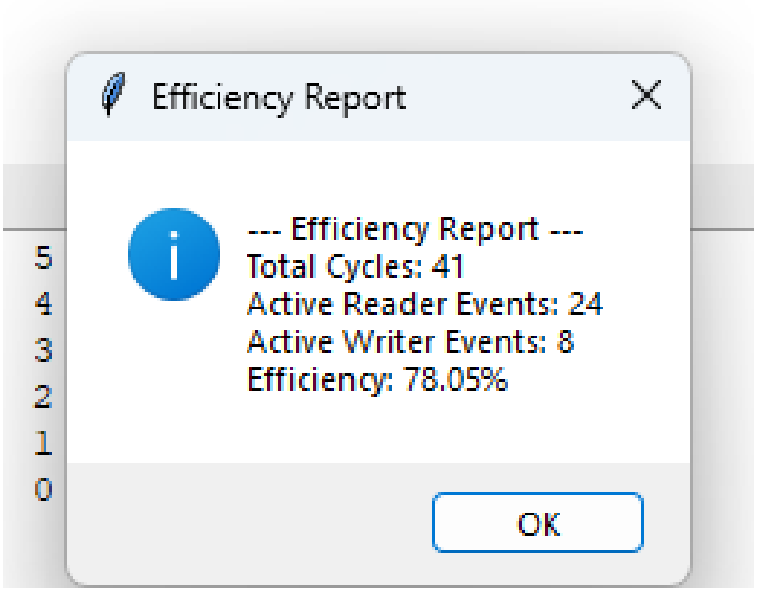

Figure 9. Efficiency report after every cycle

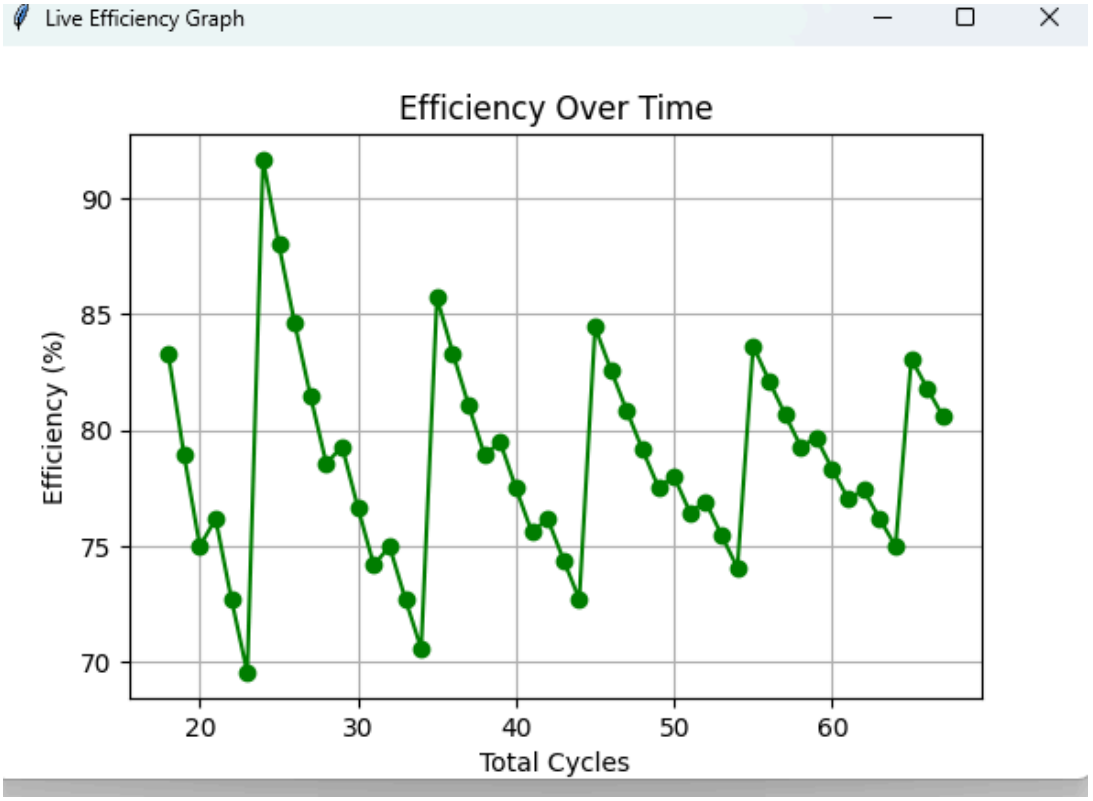

Figure 10. Life efficiency graph

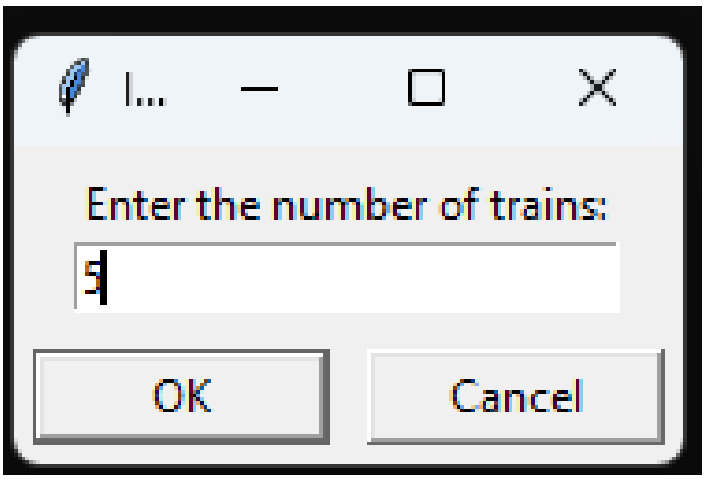

Figure 11. Choosing number of trains

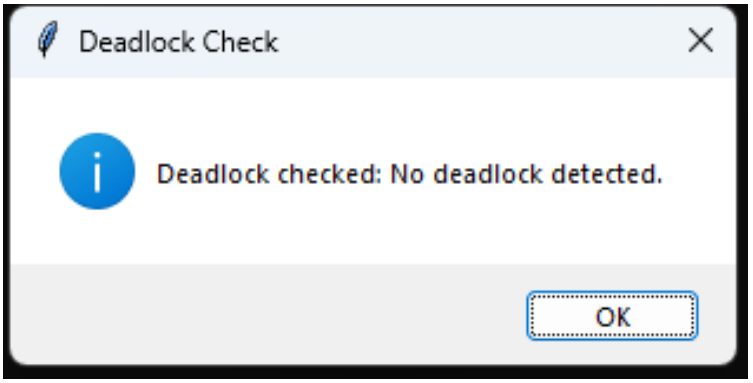

Figure 12. Ensuring no deadlock

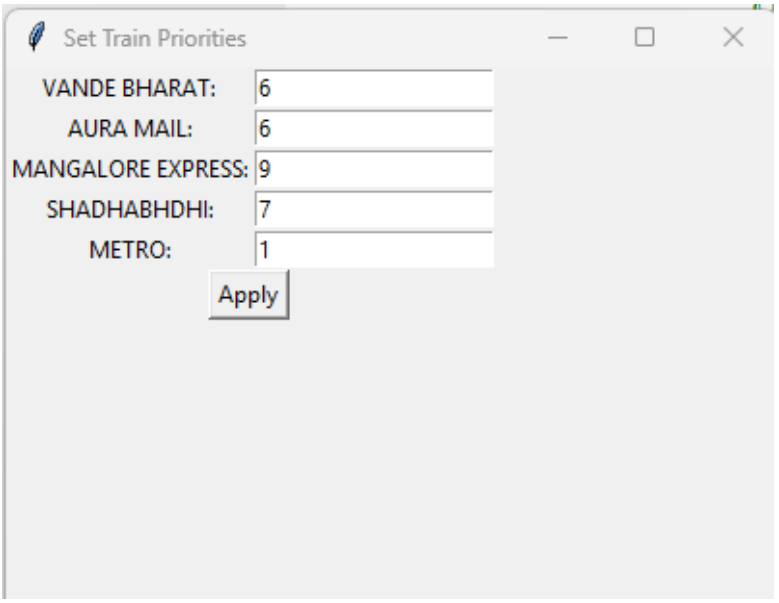

Figure 13. Setting priorities for trains

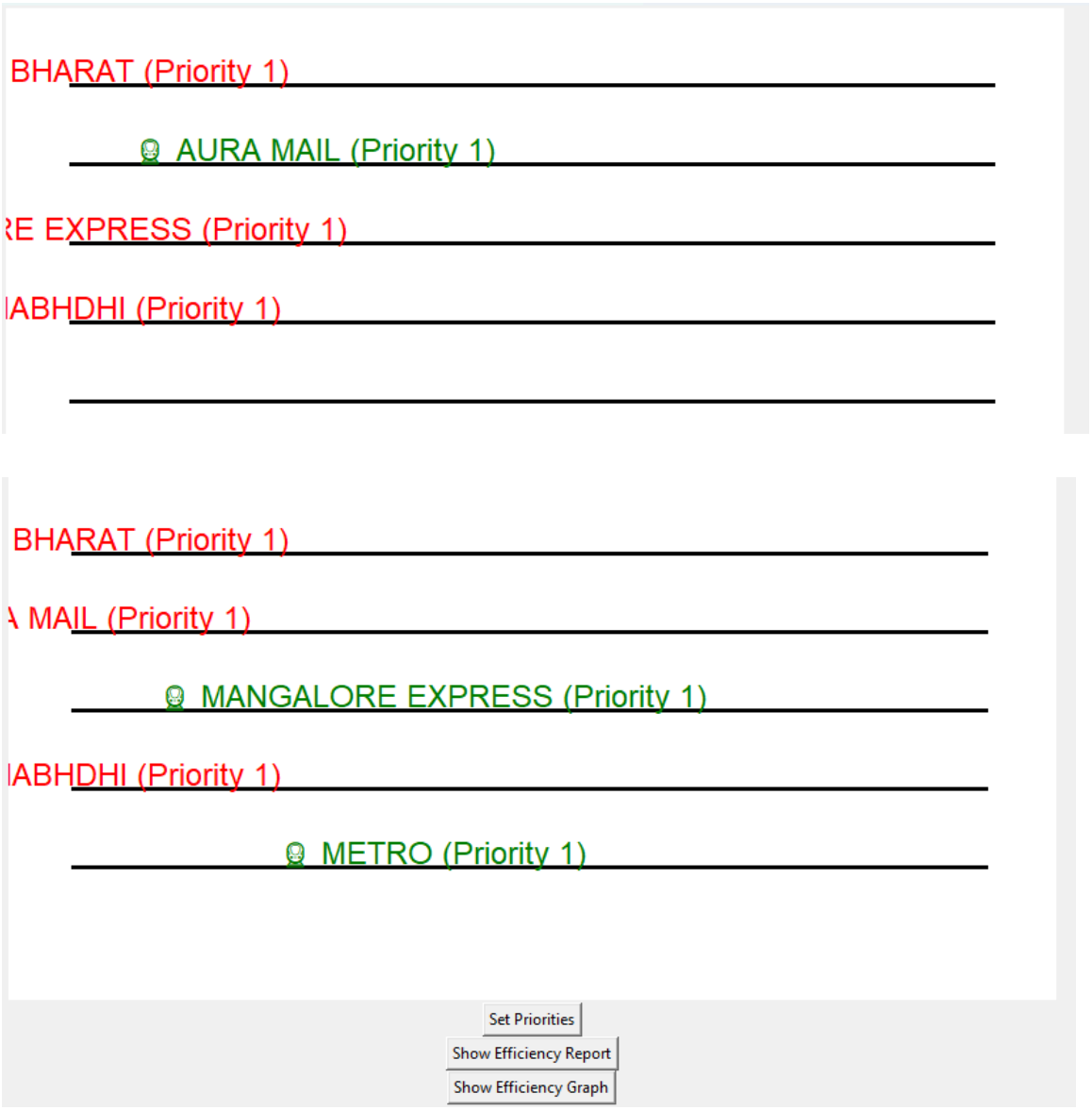

Figure 14. Simulation of train track allocation

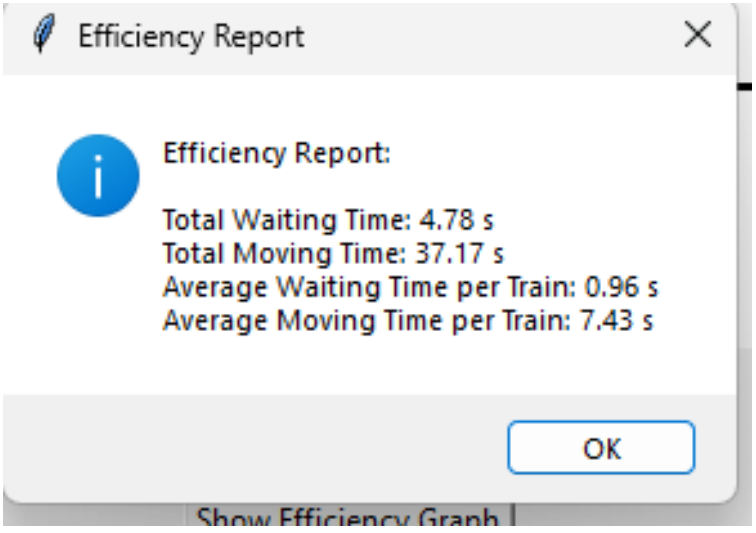

Figure 15. Efficiency report after each transmission

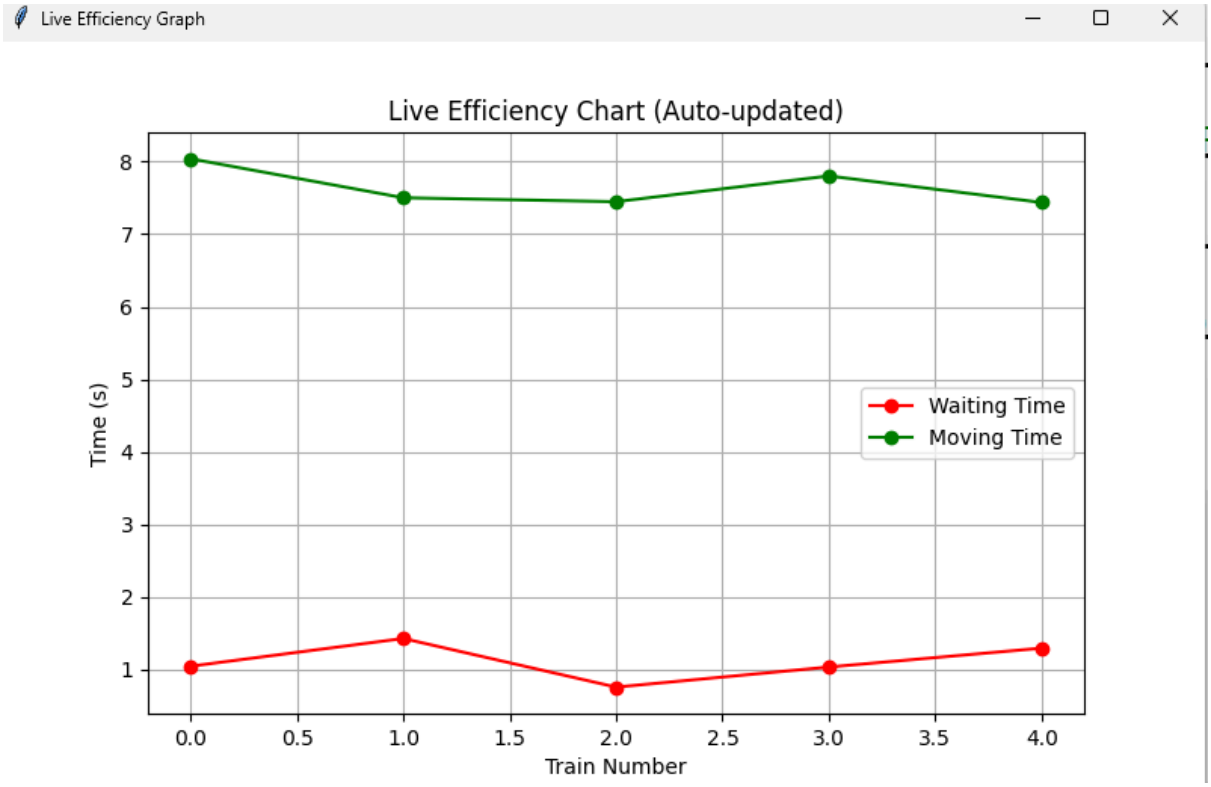

Figure 16. Live efficiency graph

CHAPTER 6

CONCLUSION

The *Dining Philosopher's & Reader-Writer's Simulation* project effectively demonstrates the practical application of core operating system concepts such as synchronization, resource allocation, and deadlock avoidance using Python's multithreading and semaphore capabilities. By simulating two foundational problems—the Railway Track Allocation (modeled on Dining Philosophers) and the Reader-Writer problem—the project provides a strong educational platform for understanding the intricacies of concurrent programming.

The Reader-Writer module explores synchronization by allowing multiple readers or a single writer to access shared resources, emphasizing fairness and priority-based scheduling. The Railway Track Allocation module simulates deadlock scenarios and their prevention through the Banker's Algorithm, reinforcing real-world applicability in shared-resource environments.

Both modules feature an intuitive Tkinter-based GUI, real-time visual feedback, and performance graphs that offer an immersive learning experience. The live log display, efficiency tracking, and user-defined configurations enhance interaction, making it not just a demonstration but a versatile educational toolkit.

Ultimately, this project serves as a bridge between theory and practice—transforming textbook OS concepts into engaging simulations that foster better comprehension, experimentation, and innovation in the field of computer science.

Advantages

- **Educational Value**

  - Makes complex OS concepts like deadlocks, synchronization, and thread scheduling easy to understand via interactive simulations.

- **Real-Time Visualization**

Live GUI representation of thread activities helps in better conceptual clarity and learning.

- **Banker's Algorithm Implementation**

  Effectively prevents deadlocks in resource sharing, making the simulation realistic and practical.

- **Customizable Interface**

  Users can input names, numbers, and priorities for threads/trains, tailoring the simulation to different scenarios.

- **Priority Modes Support**

  Enables evaluation of Reader-Priority and Writer-Priority scheduling strategies, facilitating fairness analysis.

- **Multithreaded Architecture**

  Demonstrates efficient use of Python's threading for simulating real-world concurrent systems.

- **Performance Tracking**

  Graphs and logs provide insight into wait times and execution efficiency, supporting system optimization studies.

- **Reusable Framework**

  The modular code design allows for easy extension to include other synchronization problems or scheduling techniques.

- **Practical Relevance**

  Applies to real-world systems like railway networks and database management where resource contention is common.

- **Platform Independent**

  Developed in Python with Tkinter, ensuring cross-platform compatibility and ease of deployment.

CHAPTER 7

FUTURE WORK

- **AI-based Scheduling Algorithms**

  Integrate machine learning to predict optimal track or resource allocation based on usage patterns.

- **Integration with Real-Time Data**

  Sync with real-world railway or server data to simulate actual traffic and resource usage.

- **Advanced Deadlock Handling**

  Implement dynamic deadlock detection and recovery mechanisms in addition to Banker's Algorithm.

- **IoT Integration**

  Use IoT sensors for live data input, enabling automation in track management or access control.

- **Scalability Improvements**

  Expand simulations to accommodate large-scale systems like metro networks or distributed databases.

- **Mobile Application Development**

  Design a mobile interface for remote monitoring and control of simulations.

- **Blockchain-Based Logging**

Use blockchain for immutable and secure logging of thread activities and resource access histories.

- **Role-Based Access Control**

  Add user roles for different control levels (admin, viewer, editor) in managing the simulation.

- **Real-Time Alert System**

  Integrate notifications for deadlocks, delays, or potential collisions during simulation.

- **Cloud Deployment**

  Host the simulation on cloud platforms to allow broader access for students and educators.

CHAPTER 8

REFERENCES

[1] William Stallings, “Operating Systems – Internals and Design Principles”, 9thEdition, Pearson, 2018.

[2] Pavel Y., Alex I., Mark E., David A., “Windows Internal Part I - System Architecture, Processes, Memory Management and More”, 7th Edition, Microsoft Press, 2017.

[3] Andrew S. Tanenbaum and Herbert Bos, “Modern Operating Systems”, 4th Edition, Pearson, 2016.