Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Report on

"Traffic Light Queue Simulation"

Data Structures and Algorithms (COMP202) - Assignment #1

Submitted by:

Avipsa Parajuli

37

CS-60 (II/I)

Submitted to:

Prof. Rupak Ghimire

Department of Computer Science and Engineering

Submission Date: 02/17/2025

Acknowledgement

I would like to express my sincerest gratitude to our professor **Mr. Rupak Ghimire** for his invaluable guidance, support, and encouragement throughout the semester which was of very keen help for this assignment. I also thank the Department of Computer Science and Engineering for providing us with the necessary resources and learning opportunities.

Special appreciation goes to our professor who has supported us in technical and conceptual discussions. Without his contributions, this project would not have been possible.

Abstract

"Traffic Light Queue Simulation" implements a multi-threaded, queue-based simulation for a four-way traffic junction. Using C programming, the system models vehicle queues, dynamic traffic lights, and priority-based lane control. The project uses file-based communication **(*vehicles.data*)** for handling traffic data efficiently and implements multi-threading to optimize performance. Additionally, SDL2 graphics have been integrated to visualize traffic movement, providing a real-time representation of vehicle processing. This report outlines the problem, design, implementation, results, and future improvements of the system.

Keywords: Traffic Simulation, Queues, Multi-threading, SDL2, File Communication.

List of Figures

Abbreviations

FIFO	First In First Out
SDL2	Simple Direct-media Layer
IPC	Inter-Process Communication
GCC	GNU Compiler Collection

Table of Contents

Chapter 1 Introduction

1.1 Background

Traffic congestion is a significant challenge in urban areas. The Traffic Light Queue Simulation project aims to model a four-way junction where vehicles arrive, wait, and pass through based on real-world traffic light rules. The project is implemented using queue data structures, multi-threading, and file-based communication to efficiently manage traffic flow. The main goal is to provide an accurate representation of traffic management principles while integrating SDL2 for graphical visualization.

1.2 Objectives

The main objectives of this project are:

- Implement a queue-based traffic simulation for a four-way junction.
- Utilize multi-threading to optimize traffic processing.
- Enable priority-based lane control for managing heavy traffic.
- Use file-based communication ***(vehicles.data*)** for data handling.
- Integrate SDL2 graphics for real-time visualization of vehicle flow.

1.3 Motivation and Significance

This project is designed to improve understanding of queue-based traffic management while reinforcing skills in C programming, data structures, and multi-threading. By incorporating SDL2, the simulation provides a visual representation, making it an engaging learning experience.

1.4 Basic Characteristics of the Project

- **Technology Stack:** C (with SDL2 for graphics)
- **Core Features:** Queue-based vehicle management, file-based communication, SDL2 visualization

Chapter 2 Related/Existing Works

Various traffic simulation models have been developed, ranging from simple queue-based simulations to AI-powered traffic management systems. This project focuses on a fundamental queue-based approach with additional multi-threading and SDL2 rendering.

Chapter 3 Design and Implementations

3.1 Data Structures Used

Queue (FIFO)

- Used to store vehicles waiting in each lane.
- Implemented using linked lists.

Priority Queue (Traffic Light System)

- Used to control which lane gets the green light.
- If a priority lane has more than 10 vehicles, it gets priority until the count drops below 5.

3.2 Functions Implemented

Queue Operations (queue.c)

- ***enqueue(Queue* q, int vehicle_id)***: Adds a vehicle to the queue.
- ***dequeue(Queue* q):*** Removes a vehicle from the queue.
- ***isQueueEmpty(Queue* q):*** Checks if a queue is empty.
- ***displayQueue(Queue* q):*** Displays the vehicles in a queue.

Traffic Generator (traffic_generator.c)

- ***generateTraffic(int numVehicles):*** Generates vehicles randomly and stores them in **vehicles.data** instead of separate lane files.

Traffic Light Control (traffic_light.c)

- ***initializeLights(TrafficLight lights[], int size):*** Sets all lights to RED initially.
- ***updateLights(TrafficLight lights[], Queue* laneA, Queue* laneB, Queue* laneC, Queue* laneD):*** Updates traffic light states based on queue conditions.

Traffic Simulation (simulator.c)

- ***loadTrafficFromFile(Queue* laneA, Queue* laneB, Queue* laneC, Queue* laneD):*** Reads vehicle data from *vehicles.data* and fills queues.
- ***runSimulation(Queue* laneA, Queue* laneB, Queue* laneC, Queue* laneD, TrafficLight lights[]):*** Controls vehicle movement and traffic lights.
- ***initSDL():*** Initializes SDL2 for rendering.
- ***processTraffic():*** Simulates vehicle movement and renders basic graphics using SDL2.

3.3 Algorithm for Traffic Processing

1. **Generate Vehicles**: *traffic_generator.c* writes vehicles to **vehicles.data**.
2. **Load Vehicles**: *simulator.c* reads from **vehicles.data** and fills queues.
3. **Traffic Light Control**:
    - Normal condition: Lanes take turns in a round-robin fashion.
    - Priority condition: If a lane has >10 vehicles, it gets priority until the count drops below 5.
4. **Vehicle Processing**:
    - Vehicles are dequeued from a lane when the light is GREEN.
    - The process repeats until all vehicles are processed.
5. **Graphics Rendering**:
    - SDL2 is used to visualize vehicle movement as rectangles representing cars.

3.4 Database Management

- Vehicles are stored in ***vehicles.data*** instead of separate lane files.
- Traffic data is read and processed in real-time by *simulator.c.*

3.5 Flowchart

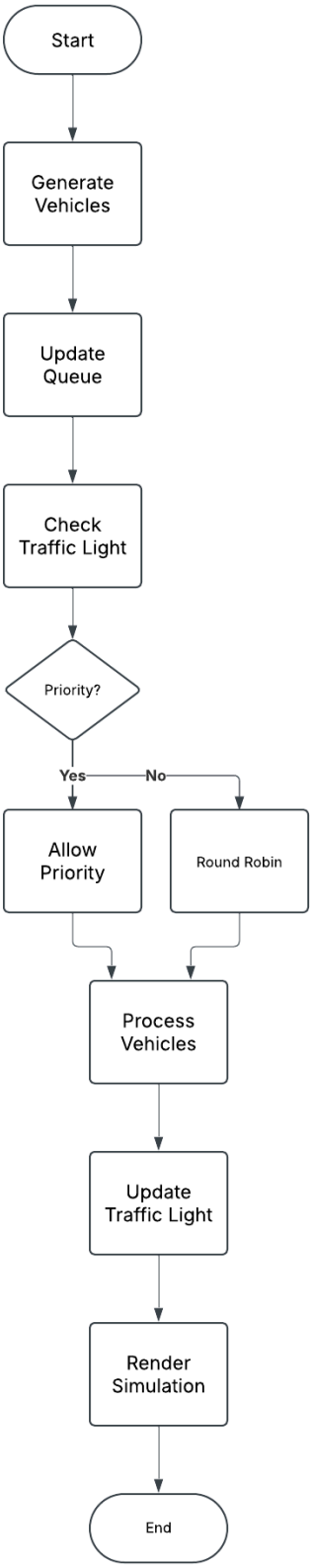

Figure 3.5.1: Flowchart

Chapter 4 Compilation and Execution Instructions

Step 1: Install Dependencies

Windows (Using MSYS2)

pacman -S mingw-w64-x86_64-SDL2 mingw-w64-x86_64-SDL2_ttf

Linux (Ubuntu/Debian)

sudo apt update

sudo apt install libsdl2-dev libsdl2-ttf-dev

Mac (Homebrew)

brew install sdl2 sdl2_ttf

Step 2: Compile the Program

gcc src/simulator.c src/queue.c -o simulator.exe -I include -I C:/msys64/mingw64/include -L C:/msys64/mingw64/lib -lmingw32 -lSDL2main -lSDL2 -lSDL2_ttf -lpthread

Step 3: Run the Programs

1. **Generate Traffic Data**

./traffic_gen

This will create *vehicles.data.*

2. **Run the Traffic Simulator**

./simulator.exe

A window should open with a basic SDL2 visualization of traffic movement.

Chapter 5 Discussion and Achievements

The project successfully simulates a queue-based traffic management system using C programming and SDL2 graphics. The implementation achieved the following key milestones:

- **Efficient Queue Management:** The system correctly processes vehicle queues, ensuring a FIFO order for each lane.
- **Multi-threading Implementation:** Using ***pthread***, traffic data is loaded and processed in parallel, improving performance.
- **File-Based Communication:** Instead of managing multiple lane files, the system uses a single ***vehicles.data*** file for efficient data handling.
- **Dynamic Traffic Light Control:** Traffic lights change dynamically based on queue length and priority-based scheduling.
- **SDL2 Visualization:** A basic graphical representation of traffic movement is displayed, improving simulation clarity.

This project serves as an effective educational tool for understanding queue data structures, process synchronization, and multi-threading concepts.

5.1 Time Complexity Analysis

Operation	**Time Complexity**
Enqueue (adding vehicles)	O(1)
Dequeue (removing vehicles)	O(1)
Checking Traffic Light Priority	O(1)
Updating Traffic Lights	O(1)
File Read/Write Operations	O(N)

Chapter 6 Conclusion and Recommendation

This project successfully implements a multi-threaded traffic simulation using C and SDL2. Future improvements include:

- Integrating real-time IPC-based communication.
- Enhancing visualization with 3D rendering.
- Implementing adaptive AI-based traffic control.

Chapter 7 Project Planning and Scheduling

A good amount of dedication, consistency, and teamwork was required for the completion of this project. To achieve this goal the project was divided into different components so that adequate time could be provided to each component. The time allocation of different tasks is shown below in the chart:

Task	Started	Ended
Initial Project Setup & Repo Creation	Jan 24, 2025	Jan 24, 2025
Implemented Queue Structure	Jan 27, 2025	Jan 31, 2025
Integrated Queue into Simulator & Traffic Generator	Jan 30, 2025	Feb 1, 2025
Implemented Traffic Light System	Feb 2, 2025	Feb 4, 2025
Integrated Traffic Light Rules into Simulator	Feb 5, 2025	Feb 7, 2025
Implemented File-Based Communication	Feb 8, 2025	Feb 9, 2025
Fixed Compilation Errors & Adjusted Includes	Feb 10, 2025	Feb 11, 2025
Refactored and Organized Code (Renamed & Moved Files)	Feb 11, 2025	Feb 12, 2025
Edited MSYS Configuration for SDL2 Support	Feb 12, 2025	Feb 13, 2025
Implemented Multi-Threading	Feb 13, 2025	Feb 14, 2025
Integrated SDL2 Graphics	Feb 14, 2025	Feb 16, 2025
Updated *README.md* with Correct Compilation & Execution Steps	Feb 16, 2025	Feb 16, 2025
Final Report Writing & Submission Preparation	Feb 12, 2025	Feb 17, 2025

Figure 7.1: Timeline Chart

References

- GitHub Repository - avxxsa/dsa-queue-simulator
- SDL2 Documentation - SDL2/FrontPage - SDL2 Wiki
- Professor's Assignment PDF & Skeleton Codes