Design Document for Stoplight Intersection Simulation

Michael Barry G01236739, Fangzhou Cheng G01150645

Overview

This document outlines the design and functionality of the stoplight intersection simulation implemented in stoplight.c. The simulation models vehicles approaching an intersection from three directions, making either left or right turns. Vehicles are categorized into ambulances, cars, and trucks, each with different priorities. The system ensures safe passage through the intersection using synchronization primitives such as locks and condition variables.

System Components

Synchronization Primitives

- Locks: Ensure mutual exclusion for critical sections. Implemented with functions to create, acquire, release, and destroy locks. Special functions like lock_try_acquire_alert are introduced for non-blocking lock acquisition attempts. This allows the vehicle scheduler to discover which parts of the intersection are currently unoccupied.

Data Structures

- Vehicle: Represents a vehicle approaching the intersection. Contains attributes such as vehicle number, type, direction of entrance, and turn direction.
- Queue: A FIFO data structure for managing vehicles. Supports operations like enqueue, dequeue, and initialization with a dummy node for simplicity.
- Multilevel Queue (MLQ): Holds separate queues for ambulances, cars, and trucks, each with its own lock. This structure facilitates priority handling of different vehicle types. The waiting zone and the vehicle scheduler both use the MLQ.

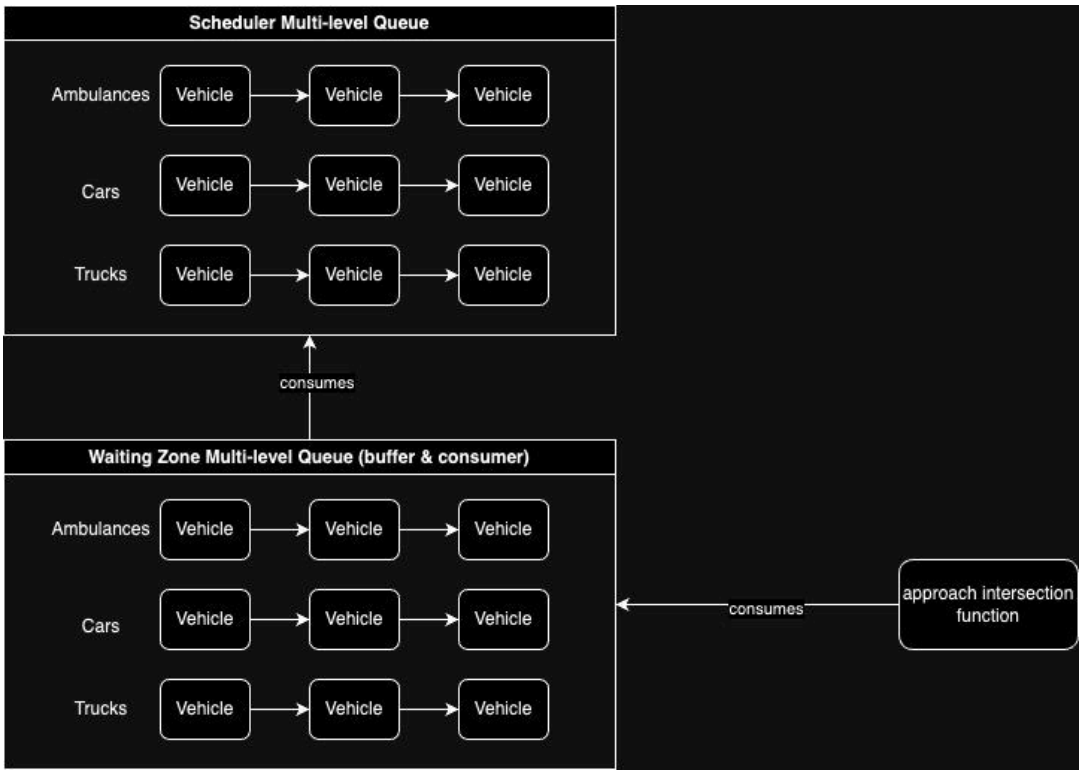
Main Functions

- Vehicle Management: Functions to create, format messages for, and free vehicle instances.
- Queue Operations: Functions to manage vehicle queues, including enqueue, dequeue, and free operations.
- Multilevel Queue Management: Initialization and destruction of MLQs, along with functions to move vehicles from the waiting zone to the scheduler's queues (Waiting_zone_consume).

- Intersection Management: Core functions to simulate vehicles approaching the intersection, making turns, and safely passing through. This includes turnright, turnleft, and approachintersection.

Design Choices

- Priority Handling: Ambulances are given the highest priority, followed by cars and trucks. This is implemented through the use of a multilevel queue, ensuring that ambulances can proceed through the intersection with minimal delay.
- Concurrency Control: Locks and condition variables are used extensively to synchronize access to shared resources and ensure the safe operation of vehicles through the intersection. Hand over hand locking is used when consuming and producing the waiting zone queues in order to allow for the vehicle scheduler and the approaches vehicles to access the shared data structure simultaneously.
- Efficiency Considerations: The vehicle scheduling system uses non-blocking lock acquisition attempts (lock_try_acquire_alert) to avoid unnecessary blocking, improving the throughput of vehicles through the intersection.
- Three locks (lockA, lockC, lockT) are used to synchronize access to the respective ambulance, car, and truck queues. These locks ensure that vehicles are queued in a thread-safe manner.
- Intersection Segment Locks: Separate locks for each intersection segment (isegAB_lock, isegBC_lock, isegCA_lock) prevent multiple vehicles from entering the same segment concurrently.



- Scheduler Lock: This lock is utilized by the CTS to maintain a consistent view of the intersection's state and to update the intersection occupancy status.
- Global Exit Counter Lock (numExitedVLock): It manages the count of exited vehicles to determine when the simulation is complete.

Challenges and Solutions

- Deadlock Prevention: Careful design of lock acquisition and release order, combined with priority rules, helps prevent deadlocks.
- Dynamic Vehicle Management: The use of dynamic memory allocation for vehicles and synchronization primitives allows the simulation to handle a variable number of vehicles efficiently.

Conclusion

The stoplight intersection simulation demonstrates a practical application of synchronization primitives to manage concurrent processes safely and efficiently. Through careful design and the use of appropriate data structures, the simulation effectively models the complex dynamics of traffic flow through an intersection.