Lab 1 - Design Document

Overview

A concurrent Go program simulating a single-lane bridge with bidirectional traffic, implementing conditional variables and locks to prevent deadlock and ensure safe passage.

System Constants

Constant	Value
Maximum Bridge Weight	750 units
Maximum Vehicles on Bridge	6
Truck Weight	200 units
Car Weight	100 units
Crossing Time	2 seconds

Architecture

Core Data Structures

Vehicle Information (v_info)

Represents a single vehicle with:

- id: Unique identifier
- isTruck: Vehicle type (truck vs car)
- isNorth: Direction (northbound vs southbound)

Queue System

Two separate queues for each direction:

- North Queue: Holds northbound waiting vehicles
- South Queue: Holds southbound waiting vehicles

Implementation uses linked list (qNode) with:

- Mutex lock for thread-safe operations
- Condition variable for signaling

Bridge (Bridge)

Centralized structure managing:

- bridge[6]: Array of vehicles currently crossing
- weight: Current total weight on bridge
- crossers: Current vehicle count
- isNorth: Current allowed direction
- dirCount: Vehicles passed in current direction
- Mutex and condition variable for synchronization

Concurrency Model

Vehicle Lifecycle

Each vehicle runs as a goroutine with three phases:

1. **Arrive**: Lock queue > Add to queue > Print arrival > Broadcast > Unlock
2. **Cross**: Wait in queue > Acquire bridge access > Change direction if needed > Enter Bridge > Cross for 2 seconds
3. **Depart**: Remove from bridge > Update state > Broadcast > Unlock

Synchronization Mechanisms

Queue Synchronization

- Vehicles wait until they reach queue head
- Condition variable wakes next vehicle when predecessor crosses

Bridge Synchronization

Vehicles wait until:

- Bridge direction matches vehicle direction (or bridge empty)
- Bridge has space (< 6 vehicles)
- Bridge can support weight (total + vehicle <= 750)

Direction Change Protocol

Triggered when:

- Bridge empty and opposite direction has waiters
- Current direction passes 6+ vehicles and opposite direction has waiters (fairness)

Process:

1. Broadcast to wake queue waiters
2. Wait until bridge empty
3. Flip direction flag
4. Reset direction counter

Fairness Algorithm

Rule: After 6 consecutive vehicles in one direction, if the opposite queue has waiters, force direction change.

This prevents starvation where one direction monopolizes the bridge indefinitely.

Key Functions

Queue Operations

- queue_add(): Append vehicle to tail
- queue_rem(): Remove vehicle from head
- queue_print(): Display waiting vehicles

Bridge Operations

- bridge_add(): Place vehicle on bridge, update weight/count
- bridge_rem(): Remove vehicle, update state
- bridge_changeDir(): Coordinate direction flip
- bridge_getIndex(): Find empty slot in bridge array

Vehicle Behavior

- arrive(): Queue entry and notification
- cross(): Wait for conditions, enter bridge, traverse
- depart(): Exit bridge and signal waiters

Input Configuration

Program accepts:

1. Number of delay periods

2. Duration of each delay (seconds)
3. Vehicle count per group (between delays)
4. Probability of vehicle being a car (vs truck)

Total vehicles must equal 30.

Safety Properties

1. **Mutual Exclusion**: Only one direction active at a time
2. **Weight Safety**: Total weight never exceeds 750
3. **Capacity Safety**: Never more than 6 vehicles on bridge
4. **Deadlock Freedom**: Broadcasts ensure waiting vehicles eventually proceed

Potential Issues

1. **Delay Calculation**: time.Duration(delays[i]*time.Now().Second()) multiplies delay by current second value (incorrect - should be delays[i] * time.Second)
2. **Race Condition Risk**: bridge_changeDir() unlocks queue CV lock while holding bridge lock, potentially causing lock ordering issues