Abdirahman Mohamed

Design Document: The Multi-Threaded Bridge Traffic Control System

Intro

The Multi-Threaded Bridge Traffic management System is intended to mimic the flow of vehicles (cars and vans) on a bridge while following traffic management standards. Individual cars are represented as threads in the system, allowing for parallel execution and realistic modeling of traffic conditions.

Thread Process and Data Structures

The system uses several data structures to regulate the state of the bridge and the vehicles. SouthboundQueue and NorthboundQueue are the two queues that are used to keep waiting automobiles in both directions. Using conditional variables, threads may wait for certain conditions like queue availability or bridge weight capacity. In order to maintain thread safety and avoid race situations, conditional variables are used to synchronize access to shared data structures and prevent data conflicts. Threads can wait effectively until a set of conditions is satisfied thanks to conditional variables.

Threads and Behavior

When a car arrives at the bridge, it is represented as a thread and follows a specified sequence. Arrive, Cross, and Leave are the three major steps of the automobile threads.

Code Structure and Procedures:

Arrive:

- During the Arrival stage, a vehicle determines whether it can safely enter the line depending on the current traffic control policy and the weight capacity of the bridge. The vehicle thread waits until the requirements are satisfied if the line is full or the bridge cannot accept extra weight. When the vehicle is permitted to advance, it adds itself to the proper line and increases the size of the queue.

Cross:

- The vehicle is allowed to cross the bridge during the Cross stage. The vehicle's weight is added to the overall weight on the bridge during this phase, replicating the load. The thread sleeps for a set amount of time to reflect the time it takes for a car to cross the bridge.

Leave:

- After safely crossing the bridge, the vehicle departs in the Leave stage, and its weight is deducted from the overall bridge load. The thread then checks to see if there are any other cars in the same direction. If this is the case, it signals the corresponding conditional variable (northboundCond or southboundCond) to enable the following car to proceed. If there are no more cars in the thread's direction, it changes the associated traffic flag (isNorthboundTraffic or isSouthboundTraffic) to false and signals the bridgeCond if the opposite lane is similarly empty.

Traffic Control Policy:

- Within the TrafficControlPolicy() method, the system implements a traffic control policy. This regulation specifies when cars from both directions may cross in both lanes at the same time. It guarantees that the weight capacity of the bridge is not exceeded and that cars from both directions have equal access. When both lanes are in use, the system checks for empty queues on a regular basis. If one queue becomes empty, the system reverts to single-lane operation, favoring traffic in the remaining queue.

Pseudocode Implementation Using CARMEN Pseudocode:

Function Arrive(vehicleId, vehicleType, vehicleDirection):

- If there is active southbound traffic or if there are more than 1200 vehicles on the bridge (adding 200 for cars and 300 for vans), the vehicle traveling north must wait in the northbound line. If the southboundQueueSize is empty, it then advances the northboundQueueSize and adds the automobile to the northboundQueue. If the northbound traffic is active or the overall weight on the bridge exceeds 1200 pounds (adding 200 for cars and 300 for vans), the vehicle traveling south must wait in the southbound line. If the northboundQueueSize is empty, it then adds the automobile to the southboundQueue and moves the southboundQueueSize forward.

Function Cross(vehicleId, vehicleType, vehicleDirection):

- Reduce the northboundQueueSize if the automobile is traveling north; otherwise, decrease the southboundQueueSize and raise the totalWeightOnBridge by 200 pounds for cars and 300 pounds for vans. Print "Vehicle #vehicleId> is now crossing the bridge," and then halt the execution for a predetermined amount of time (X seconds) to mimic the time it will take for the vehicle to cross the bridge.

Function Leave(vehicleId, vehicleType, vehicleDirection):

- Print "Vehicle #[vehicleId] exited the bridge" once the vehicle has left the bridge, deducting the appropriate weight (200 for cars, 300 for vans) from the totalWeightOnBridge calculation. Check to see whether there are any vehicles in the northbound line if the car is moving north; if so, indicate the northbound condition; otherwise, set isNorthboundTraffic to false. In the same way, if a car is traveling south, see whether there are any cars in the southbound line. If there are, signal the southbound condition; otherwise, set isSouthboundTraffic to false and signal the bridge condition if isNorthboundTraffic is false.

<u>Function TrafficControlPolicy():</u>

- Set the isBothLanesUsed flag to true and the isNorthboundTraffic and isSouthboundTraffic flags to true if both lanes are presently not in use and either the northbound or southbound queue size is larger than 0. Set the isBothLanesUsed flag to false if the northbound queue size is greater than zero, otherwise set the flag to false and signal both northbound and southbound conditions by setting the isSouthboundTraffic flag to true. If both lanes are in use and either the northbound or southbound queue size is zero, set the isBothLanesUsed flag to false.

<u>Function PrintTrafficFlow():</u>

- Print the traffic flow and a list of the cars with their license plates and directions if there are any in the northbound line. If no vehicles are in the northbound queue, however, print "No vehicles in the Northbound queue." If there are any cars in the southbound line, publish a list of them along with their IDs and directions; else, print "No vehicles in the Southbound queue."

<u>Function getRandomDirection(probabilityNorthbound):</u>

- Create a random integer between 0 and 1, and if it is smaller than the chance for traveling north, return that direction; otherwise, travel south. Similarly, using the method getRandomVehicleType, generate a random number (random) between 0 and 1, and if it is less than 0.5, return CAR; otherwise, return VAN.

<u>Function VehicleRoutine(args):</u>

- GetRandomDirection and getRandomVehicleType methods are used to determine the vehicle's direction while taking delay time and probability into account. After that, it crosses the bridge after requesting permission to do so, changes the queue state, releases resources, and finally returns null.

<u>Function ReadScheduleFromUser(numberOfGroups, vehiclesPerGroup, probabilityNorthbound, delayTimes):</u>

- The user is prompted to input the number of groups they wish to replicate in this section of the application. Based on the value supplied for "numberOfGroups," the software dynamically allocates memory to hold arrays for vehicle counts, the chance that each group will move north, and delay lengths in seconds. The application then prompts the user to enter details for each group, such as the number of automobiles, the possibility of northbound traffic, and the delay duration, before reading and storing this information appropriately for simulation needs.

Function PrintQueueStatus():
- Print current queue status

Main function:
- The application uses the "ReadScheduleFromUser" method to get user input for the anticipated arrival times of vehicle groups and employs a "random number generator with seeds" to generate repeatable random numbers. The "VehicleRoutine" function is used to generate threads for each vehicle in each group. When the threads have finished, "PrintTrafficFlow" is used to show the traffic flow and waiting automobiles before dynamic memory is released.

Conclusion

I provide a precise and accurate simulation of the traffic flow on a bridge provided by the Multi-Threaded Bridge Traffic Control System through my independent project. By utilizing multi-threading and synchronization methods, the system achieves efficient vehicle crossing, conforms with traffic control requirements, and ensures the security of concurrent data access. The goal of the design is to provide a dependable, scalable system that can be altered to cater for more complex situations and traffic management requirements.