COSI 131a
Programming Assignment 4

(Release: 10/29/24 Due: 11/10/24)

Overview

After you have implemented the simple tunnel admission controller for vehicles in Programming Assignment 3 (PA3), your next task is to implement an improved tunnel controller that will provide admission to the tunnels based on priority.

This assignment requires you to build on your synchronized code of PA3 and (1) add Java condition variables to avoid busy-waiting, unless it is advantageous, and (2) add a priority scheduler.

The Priority Scheduler

You have successfully programmed BasicTunnel in PA3, which enforces admission criteria and prevents race conditions. However, the admission policy in PA3 has some issues:

- While no tunnel is available to a Vehicle, that Vehicle busy-waits (loops over all tunnels repeatedly inside of its run() method). Busy waiting could be advantageous when the wait is short but is wasteful when the wait is long. An admission policy that works for both short and long wait times is needed.

- There is no way of prioritizing important vehicles (say, an ambulance) over less-important vehicles (say, an ice cream truck). A policy that takes in account priorities is needed.

The Priority Definition

In this assignment, vehicle priority is defined as follows:

- Vehicle priority is a number between 0 and 4 (inclusive)
- A HIGHER number means a higher priority

Your Task

You will implement a Priority Scheduler that addresses the admission policy limitations explained above and enforces an improved admission policy that supports priorities and

eliminates wasteful busy waiting.
Like in PA3, you will be provided with a code base that you will modify to implement your solution. There are rules specified below that you must follow when modifying the code base. These rules allow you and us to test your solution

Priority Scheduler

You are provided with a class called PriorityScheduler (in PriorityScheduler.java) that implements Scheduler and you must add functionality that controls access to a collection of Tunnels in order to implement the improved admission policy. The Tunnels "behind" the PriorityScheduler will be BasicTunnels. BasicTunnel should be the same class you implemented in PA3. PriorityScheduler will carry out the following tasks:

Keep references to BasicTunnels as private member variables inherited from Scheduler inside PriorityScheduler.

Enter tunnel: When a Vehicle v calls enter on a PriorityScheduler Instance to enter a tunnel, the following general steps should be executed:

- If v.getPriority() is less than the highest priority from among all the other vehicles waiting to enter a tunnel (i.e., there is another vehicle with higher priority), then the vehicle must wait.
- Otherwise the vehicle v successfully enters one of the tunnels. Note that you should call the tryToEnter method on BasicTunnel instances when attempting to enter a tunnel (do not call tryToEnterInner directly!).

Exit tunnel: When a Vehicle v exits the tunel by calling exit on a PriorityScheduler instance, the scheduler must call exitTunnel(v) on the appropriate tunnel from the collection of tunnels managed by the scheduler . . You may not modify Vehicle or any of the other classes provided to you, or BasicTunnel, so you must solve this problem within PriorityScheduler). Again, please keep in mind that you should call the exitTunnel method (not the exitTunnelInner). After a vehicle has successfully exited a tunnel, the waiting vehicles should be signaled to retry to enter a tunnel. The vehicles with highest priority should be allowed to enter.

Use condition variables to avoid busy waiting if the vehicle cannot find a tunnel to enter. Make sure the use of the condition variables is safe.

You are **not allowed to use the synchronized keyword** (except in the BasicTunnel.java from PA3)

Testing your code:

A correct solution must pass the tests PrioritySchedulerTest.Car_Enter, PrioritySchedulerTest.Sled_Enter and PrioritySchedulerTest.Priority included with this assignment. **Please make sure to run the tests several times** as a race condition problem can appear in only some of the runs. We provide the NUM_RUNS variable inside each test class that specifies how many times to run each test. It is set to 1 by default but feel free to change it to how many times you want to run each test (we suggest 10).

Just like in PA3, your only point of entry in the code we provide is through the JUnit tests, which will set up the environment in which your client threads will run. Your tasks for this assignment do not include writing a main method. Rather, you must rely on your understanding of condition variable synchronization, and the tests we provide, to know if you have completed the assignment correctly.

Submission

Please **rename your project (not just your submission)** to FirstnameLastnamePA4 export your project together with JavaDocs as described on LATTE and submit as a zip file. You should not submit source files individually.

Important note about disallowed Java Tools

- You may not use the thread priority methods provided by Java (e.g., you may not use Thread.setPriority).
- Just like in PA3, you may not use any synchronized data structure included in the Java API (that is any data structure class that has synchronized methods – for example LinkedBlockingQueue). You will implement your own synchronization protocol by **using condition variables**.
- Solve the assignment using Java 8 or later, and Junit 5

Assignment Policies

- NO CODE EXCHANGE between students, past or present.
- Any sign of collaboration will result in a 0
- Violations will be penalized appropriately according to the syllabus ● Using ChatGPT is a bad idea. Trust us. It won’t understand our loggers, it won’t understand our test cases, it writes buggy code, just don’t do it.
- Check the course syllabus for late submission policy