16.35 Assignment 2

Austin Floyd

19 April 2019

1 Analyzing Critical Regions

1. Why is waiting on a condition variable better than a busy-wait loop?

   Busy waiting, or continuously checking a condition, is computationally expensive when compared to condition variables.

2. Why is no synchronization or protection required for the control_vehicle and update_state method calls?

   These are not shared resources. Each vehicle has its own state.

3. What is the critical region in this program?

   The critical region (where shared resources are accessed) of this program is the counter that tracks the number of vehicles that have been updated. This counter can be accessed by multiple vehicles simultaneously if this is not accounted for.

4. Why do we need to protect the incrementing of the vehicles_updated variable?

   For example, Vehicle A could access the counter when it is equal to 0 in order to add 1. Before 1 is added, however, Vehicle B could also access the counter while it is still 0. Both vehicles add 1, but because they both accessed the counter when it was equal to 0, the counter is only updated to 1. This "race condition" leads to an undercount.

5. Is a mutex appropriate for incrementing the vehicles_updated variable?

   Yes, this makes the act of accessing the counter and adding to it atomic (no other vehicles can access the counter once another has accessed it, until it completes writing).

6. Is there a possibility of deadlock or synchronization failure in this design?

   No, the use of atomic mutexes and waiting and signaling ensures that shared resources are never accessed by multiple vehicles at any given time.

2 Getting Your Ducks in a Row

1. The above code could lead to deadlock. Please explain how deadlock could occur. Provide a specific example of how deadlock could occur in a scenario with two vehicles and describe what behavior a user would observe.

   This situation (where a car follows the next vehicle) leads to deadlock because the movement of the ith vehicle is dependent upon the first vehicle, which itself cannot move until the ith vehicle does so. In a scenario with two vehicles, Vehicle 1 waits for Vehicle 2 to move in order to follow it. Vehicle 2 waits for Vehicle 1 to move in order to follow it. Thus, deadlock exists.

2. Describe how one could modify the get_follower_control implementation to compute the control while simultaneously holding exclusive access to both the current vehicle's position and the leader position's and still prevent deadlock.

   One way to remove deadlock is by attacking the circular wait condition. This can be done by requiring the vehicles to access the ith vehicle's position in ascending order. In the case of two vehicles, Vehicle 2 cannot access Vehicle 1's position until because 2 is greater than 1. Because of this, it must wait for Vehicle 1 to update its position first.

3. If we removed the requirement that get follower control locks on vehicle.position, could deadlock arise? Is removing the lock a reasonable thing to do given the current threading behavior of one thread per vehicle?

   Yes, by removing these locks, there is no longer a risk for deadlock. However, because other threads are accessing vehicle.position, it is not a good idea to remove the locks as concurrency problems may arise.

4. Assuming you have implemented a solution that removes the possibility of deadlock, please analyze the performance of this program if all your vehicles used a follow controller and they all followed the same vehicle. What if they all followed different vehicles?

   The performance would improve all of the vehicles following the same vehicle, and the opposite would occur for different vehicles.

3 Identifying Critical Regions

1. What critical regions of the code does your new controller depend on? Do you need to introduce additional synchronization to protect the regions?

   Had I been able to successfully implement this controller, the critical regions of the code would have been the assignment of the control policy, updating the vehicle count, and adding the time increment.

2. Does your new controller introduce the possibility of deadlock? If so, what

strategy did you implement to avoid it?

The new controller would not have introduced the possibility of deadlock. This would have been avoided by preventing circular wait.

3. What sort of performance would you expect to see if all your vehicles used your new controller?

   The performance of this controller would decrease due to the need to access the positions of other vehicles.

4 Completion Time

q1: 10 hr (I was eventually able to get this section working for the final assignment, though it was incomplete for the predeliverable. Is any credit given for this?)
q2: 0.75 hr
q3: 7 hr
q4: 0.75 hr
q5: 0.33 hr
q6: 0.33 hr
total: 19.16 hr