**Traffic Control**

* **Explicit Lane and Crossing Queues**



|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |



|  |  |  |  |
| --- | --- | --- | --- |
|  |  | CarY(t=2) | ~~CarX (t=1)~~ |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | CarZ | CarX |



|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | ~~CarZ(t=3)~~ |



|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

Cars at the front of their line (queue) hold the front of line lock. The lock is a binary semaphore.

When a car has this lock, it is allowed to proceed to the cross queue.

The cross queue is the FIFO ordering that mimics the real world of cars at a stop sign intersection. Once a car is at the front of the car queue, it can begin calling sem\_wait() on the corresponding semaphores needed for the direction of travel intended.

This queue structure exists to prevent other cars that are at the front of their lines from calling sem\_wait(), as sem\_wait() does not guarantee FIFO order when more than 1 object is calling the semaphore.

* The queue is the gate keeping system to ensure FIFO is still enacted.
* This logic applies to the lane queues as well. Only the front car can begin being entered into the cross queue.
* **Semaphore Implementation**

There are 24 semaphores used in the program

16 – Intersection related

* A left turn has 4 lane crossovers
* A straight path has 4 lane crossovers

Locks are shared by paths around the intersection which condense the number of semaphores into 14 total for the whole intersection.

8 – Beginning and Ending Related

* 4 Head of line locks
* 4 End locks
* **Algorithm**

Thread is created -> takeCar()

Enqueue(lane\_queue)

//arrive intersection

If car is at the front of the queue

Enqueue(cross\_queue)

Else

Wait

While(car != cross\_queue.top()) //spin wait, but check for special condition

If the car infront of current car has the same path

If there are no other cars at the intersection

Break from loop //the car does not need to wait to acquire semaphores, and can follow suit

Else

Wait

Locks[] <- RetrieveCorrespondingLocks()

For I In locks:

Sem\_wait()

//cross intersection

Spin(t) //t = designated time (1 for right, 2 for straight, 3 for left)

dequeue(cross\_queue) //dequeue early before locks released so the next up car in cross\_queue that is spin waiting can check for semaphore ASAP

//exit intersection

For I in locks

Sem\_post()

* Output

Graphical user interface, text

Description automatically generated

The output matches the required output, except for cars 3 and 4 crossing at time 6.41 seconds.

The output should be switched, however the time for them to execute is the same, so the deviation of which is printed first is due to the CPU scheduling the threads, because one thread must execute its instructions before the other, even if they are scheduled at the same time.