

# Project 3 (P3): Traffic Control

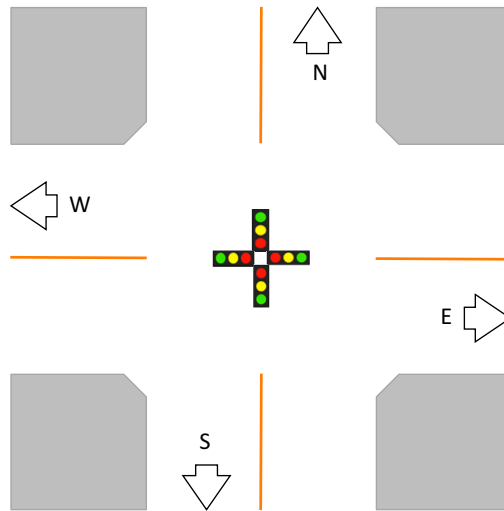
Instructor: Dr. Shengquan Wang

Due Time: 10PM, 11/28/2018

In this project, you will use `mutex` plus `condition variables` or `semaphore` together with `pthread` to implement a `traffic-control` system with traffic lights at an intersection.

## 1 Traffic Lights

We consider a typical intersection with traffic lights as shown in the following picture. It has four directions (E, W, S, and N). The road on each direction has only one lane.



The intersection has a traffic control system with three different colors (Green, Yellow, and Red). Here is the pattern for the traffic lights:

The light at each direction follows this pattern: Green for  $T_G$  seconds, then Yellow for  $T_Y$  seconds, and Red for  $T_R$  seconds, then repeating Green, Yellow, and Red, ... We assume that the lights on E and W are synchronized and the lights on S and N are synchronized. Moreover, when the lights on E and W directions are in Green or Yellow, the lights on S and N directions should be in Red; when the lights on S and N directions are in Green or Yellow, the lights on E and W directions should be in Red. Therefore, we have the following requirement in terms of the time duration:  $T_G + T_Y = T_R$ . In this system, all traffic lights will be modeled together by a **single thread**, which should define the colors in all directions.

In the following, we describe the traffic control policy for cars. In this system, each car is represented by one thread, which executes the subroutine `Car` when it arrives at the intersection:

```
Car(directions dir) {
```

```

    ArriveIntersection(dir);
    CrossIntersection(dir);
    ExitIntersection(dir);
}

```

Data type `directions` is defined as

```

typedef struct _directions {
    char dir_original;
    char dir_target;
} directions;

```

where `dir_original` and `dir_target` show the original and targeted directions, respectively.

**ArriveIntersection** When a car arrives at the intersection, if it sees a car (with the same original direction) stopping in front, it should wait until the front car starts crossing (it doesn't need to wait until the front car finishes crossing). This is called **head-of-line blocking**. Otherwise, it can either drive through, or turn left, or turn right (U-turn is not allowed). But it should first check the traffic light in front.

- **Green Light:** If it attempts to **drive through**, make sure that no car from the opposite direction is turning left, or any car is turning right to the same lane; if it attempts to **turn left**, make sure that no car is driving through the intersection in the opposite direction, or any car is turning right to the same lane; if it attempts to **turn right**, make sure that no car is driving to the same lane.
- **Yellow Light:** If it attempts to turn right, make sure that no car is driving to the same lane, otherwise it should stop in front of the intersection.
- **Red Light:** If it attempts to drive through or turn left, it should wait for the Green light and then follows the rule for Green Light; if it attempts to turn right, make sure that no car is driving to the same lane.

We assume there is no collision for two turning-left cars from the opposite directions. You could use `usleep` for the arrival time.<sup>1</sup>

**CrossIntersection** We assume that it takes fixed time periods ( $\Delta_L$ ,  $\Delta_S$ , and  $\Delta_R$  for turning left, going straight, and turning right, respectively) to cross the intersection and print out a debug message. You could use the `Spin` function <sup>2</sup> to simulate the crossing.

**ExitIntersection** It is called to indicate that the caller has finished crossing the intersection and should take steps to let additional cars cross the intersection.

## 2 Testing

For the traffic lights, fix all the time periods described above as follows: ( $T_G = 18s$ ,  $T_Y = 2s$ ,  $T_R = 20s$ ). and ( $\Delta_L = 3s$ ,  $\Delta_S = 2s$ ,  $\Delta_R = 1s$ ). We assume that the lights on S and N turn in Green and the lights on E and W turn in Red at time 0 in the beginning of the testing. The simulation will run 30 seconds.

In the main thread, you need to create one thread for each car and one thread to manage traffic lights. The arrival pattern in the simulator is described as follows:

---

<sup>1</sup><http://linux.die.net/man/3/usleep>

<sup>2</sup>Download this: <http://pages.cs.wisc.edu/~remzi/OSTEP/Code/code.intro.tgz>. You can find the `Spin` function in `common.h`.

cid	arrival_time	dir_original	dir_target
0	1.1	N	N
1	2.0	N	N
2	3.3	N	W
3	3.5	S	S
4	4.2	S	E
5	4.4	N	N
6	5.7	E	N
7	5.9	W	N

Associated with every car is a unique id (`cid`) as the car number, the arrival time (`arrival_time`) for the given car, and the original `dir_original` and targeted `dir_target` directions. For example, the first line shows Car 0 arriving at Time 1.1 which originally goes northward and continue to go northward. Make sure that your simulation outputs information that clearly shows that your solution works. The message should indicate car number and both original and targeted directions. In particular, messages should be printed at the following times:

- Whenever a car arrives at an intersection;
- Whenever a car is crossing the intersection;
- Whenever a car exits from the intersection.

This is the right output for the arrival pattern given above:

```

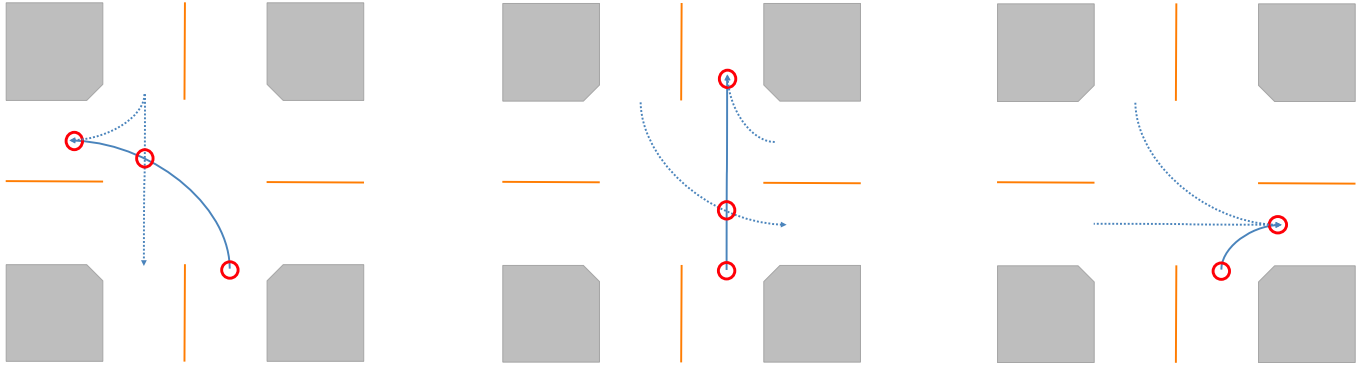
Time 1.1: Car 0 (->N ->N) arriving
Time 1.1: Car 0 (->N ->N) crossing
Time 2.0: Car 1 (->N ->N) arriving
Time 2.0: Car 1 (->N ->N) crossing
Time 3.1: Car 0 (->N ->N) exiting
Time 3.3: Car 2 (->N ->W) arriving
Time 3.3: Car 2 (->N ->W) crossing
Time 3.5: Car 3 (->S ->S) arriving
Time 4.0: Car 1 (->N ->N) exiting
Time 4.2: Car 4 (->S ->E) arriving
Time 4.4: Car 5 (->N ->N) arriving
Time 4.4: Car 5 (->N ->N) crossing
Time 5.7: Car 6 (->E ->N) arriving
Time 5.9: Car 7 (->W ->N) arriving
Time 6.3: Car 2 (->N ->W) exiting
Time 6.3: Car 3 (->S ->S) crossing
Time 6.4: Car 5 (->N ->N) exiting
Time 6.4: Car 4 (->S ->E) crossing
Time 6.4: Car 7 (->W ->N) crossing
Time 7.4: Car 7 (->W ->N) exiting
Time 8.3: Car 3 (->S ->S) exiting
Time 9.4: Car 4 (->S ->E) exiting
Time 20.0: Car 6 (->E ->N) crossing
Time 23.0: Car 6 (->E ->N) exiting

```

### 3 Coding and Hints

You are recommended to use threading programming in C/C++. The programming should be tested in the Ubuntu environment in your P1 and P2, not in the xv6. Your executable file should be named as `tc`. Use the two sample codes: `condvar.c` and `mythread_new.c` as the basic framework.

Make sure to identify the key resources each car needs to acquire, which you need to model with mutex or semaphore. The following figures show the different scenarios: turning left, crossing straight, and turning right. The first two are for green light; the last one is for all possible color light. For each



red circle, you need to create a mutex or semaphore.

For the head-of-line lock, when a car starts crossing, it should release the head-of-line lock so that the other cars behind can start crossing ASAP. For the other mutexes or semaphores, the car should release them after finishing crossing. If a flow of cars with the same original direction go towards the same targeted direction back-to-back, they should be allowed to go back-to-back if all conditions are met. In this case, you could follow the example of readers' operations in the readers-writers problem to synchronize them.

### 4 Submission

You are going to report the following things:

- (a) Describe in details how you implemented it in the report.
- (b) Write a detailed report with the screenshots of the testing results. Each screenshot should include your username and the current time, which show that you did it by yourself. **If your output is different from the expected one, provide a reason for the cause.**
- (c) Specify the contribution made by each member if you work as a group.

The report should be written in a “.docx”, “.doc”, or “.pdf” format. All source codes should be well formatted with good comments. Submit the report and the source code to Project P3 on Canvas. Any compression file format such as .zip is not permitted.