Topic:

Traffic Intersection

Subject:

Operating System

Submitted by:

Anns Shahzad (2019-CS-601)

Muhammad Hassan Ijaz (2019-CS-617)

Submitted to:

Mr. Nadeem Iqbal

Department of Computer Science

University of Engineering & Technology, Lahore (New Campus)

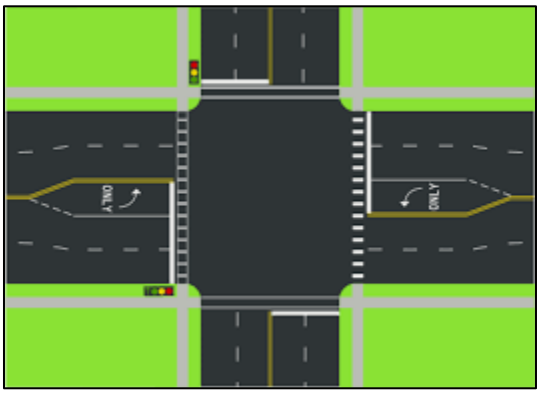
Table of Content:

Traffic Intersection System:

Introduction:

Overview:

Intersection is **an area shared by two or more roads**. This area is designated for the vehicles to turn to different directions to reach their desired destinations. Its main function is to guide vehicles to their respective directions. Traffic intersections are complex locations on any highway.



Background:

The intersection traffic signal control problem (ITSCP) has become even more important as traffic congestion has been more intractable. The ITSCP seeks an efficient schedule for traffic signal settings at intersections with the goal of maximizing traffic flow while considering various factors such as real-time strategies, signal timing constraints, rapid developments in traffic systems, and practical implementation. Since the factors constituting the ITSCP exhibit stochastically complicated interactions, it is essential to identify these factors to propose solution methods that can address this complexity and still be practically implemented.

Problem:

The main problem is deadlock when there is no traffic intersection signal. Due to deadlock no one can move from their position and the traffic flow is minimize. As we know when deadlock occur 4 things must be occurred,

⇒ Mutual Exclusion
⇒ Hold and Wait
⇒ No preemption
⇒ Circular wait.

Suppose the car in SE1 and NE2 are making right turns. The rest of the quadrants are filled with cars attempting to go straight on their second move. Deadlock will occur in this situation. The other problems are accidents, Traffic flow and starvation and preserve car order. We have covered these problem in this project.

Solution:

Cars are first initialized. Cars are added to a queue. Car to be more specific to our solution. Each car constantly checks to see if they are the first in the car. This makes a pool of cars that are waiting to be the next in line. This can be seen by our while (1) loop. Normally this would be busy waiting. However, we used mutex to solve this issue. Only one car is allowed to check to see if it is the first in the queue at any given time, while the rest must wait for a mutex to be signaled.

Once a car knows that it can slot into the waiting in line position, (another check that we make alongside being the first in the queue), then the car can move into position to be the next one to go.

Once that is done, we solve the deadlock issue. Each car that is next in line constantly checks to see if it can lock up one of the intersections. If it cannot then it must back out of all the other intersections that is locked up previously. Essentially, we are saying that if a car cannot lock all its intersections based on its turn, then do not go and repeat those checks once again.

Once the car successfully obtains all its resources, we let it pass with the time to cross and release the lock for each resource (mutex) it has based on its turn. Finally, we set the state to leaving, update some numbers and then have this on repeat.

- **Avoiding Deadlock:**

  Remove the circular wait in the queue as we are forcing each car to obtain all resources before going through the intersection.

- **Prevent Accident:**

The car must move all the quadrants before it starts to go. So, it must leave through the intersection when there is no other car in the intersection. If there is a car in the intersection, then it will try again later.

- **Improve traffic flow:**

Multiple cars can be within the intersection assuming they do not collide when trying to lock all their resources. At worst case, only one car will be in the intersection under a rare circumstance. At best case there can be up to 4 fours in the intersection at a given time.

- **Preserve car order:**

We used a Queue to accomplish this. There was a big decision in deciding if we wanted to implement 4 queues for each possibly direction of arrival or having one large queue that all the cars waited in. We went with one large queue because of fairness. If you have 4 different queues and you have the situation where say N and S queues have a bunch of straight turns. Then you will run into an issue where the W and E directions will be starved of going through the intersection. Whereas if you only have one large queue you might not be as efficient, as it can be as you might have open slots for the Next in Line while a car isn't at the first of the queue, BUT you guarantee that no direction will ever be starved from letting cars into the intersection because of a queue.

- **Fairness:**

Fairness was ensured using a queue and our **isOpen** array. The queue preserves order, which means that arrival order is preserved there. There is an assumption that semaphores have some aspect of fairness in their implementation. While we don't know exactly how semaphores decide who goes next, the material in class and the documentation online suggests a light level of fairness, even if it is small. Randomness over a large data set also helps keep fairness, mostly regarding starvation, which is covered below.

- **Starvation:**

Starvation is prevented through a queue and our **isOpen** array. This array keeps track of whether a car is ready to go from a certain direction or not. This array allows us to allow every direction to go at some point, even if it isn't right away. The queue only feeds into this array when there is space available. Through randomness over a large enough data set, we can be certain that there will be a point when every car can go.

How to use the Program:

This software is run via the command line and takes in two required arguments:

⇒ **Argument # 01:** Time to live in seconds.
⇒ **Argument # 02:** Number of cars.

Short Description:

This program runs a simulation of an intersection with emphasis on the use of threads and semaphores. The program will take in user input to determine how long to run and how many cars to simulate, and then create a unique thread for each car.

As these cars run through various stages of an intersection to either turn right, go straight, or turn left, the status of each car is printed out to monitor correct activity.

At the end of the simulation the minimum, maximum, average, and total time spent at the intersection is printed out.

Related Study:

For understanding the project, we search from the internet and read some articles on related this topic to get to know how this process work how we stimulate this problem. We understand a system scheduling algorithm. We understand the traffic light through images-based model.

These are the main article we used in our project:

- https://en.wikipedia.org/wiki/Traffic_light
- https://wonderopolis.org/wonder/how-does-a-traffic-light-work
- https://practical.engineering/blog/2019/5/11/how-do-traffic-signals-work
- https://www.hindawi.com/journals/jat/2018/3785957/
- https://www.hindawi.com/journals/jat/2020/3828395/
- https://www.youtube.com/watch?v=8slD16fj2YU
- https://www.youtube.com/watch?v=yITr127KZtQ
- https://etrr.springeropen.com/articles/10.1186/s12544-020-00440-8
- http://par.cse.nsysu.edu.tw/~cbyang/person/publish/c07traffic.pdf
- https://github.com/Nomiizz/Traffic-Intersection-Problem/blob/master/traffic.c

Methodology:

After gathering related material and searching the project it is simply divided into 3 different section:

⇒ Semaphore

⇒ Threads

⇒ Queue

- **Semaphore:**

  First, we create semaphore for signals the car. According to signals we move from the lanes as regard the traffic assemble.

semaphore.c

```c
int semaphore_create(semaphore_t *sem, unsigned int value)
{
    /* Initialize the variables */
    sem->sem  = NULL;
    sem->name = NULL;

if (USE_NAMED_SEMAPHORES == 0)
 {
    if(sem->name != NULL ) {
        printf("Error: semaphore_create(): Failed to allocate memory for the name\n");
        return -1;
    }
    printf("Creating a semaphore named: %s\n", sem->name);

    sem->sem = sem_open(sem->name, O_CREAT|O_EXCL, S_ALL, value);
    if(sem->sem == SEM_FAILED)
        {
        printf("Error: sem_open failed!");
        printf("Failed on Semaphore named %s\n", sem->name);
        return -2;
    }
    return 0;
}
else
{
     printf("Creating a semaphore named: UNNAMED (value %d)\n", value);
    sem->sem = (sem_t*)malloc(sizeof(sem_t) * 1);
    if(sem->sem == NULL) {
        printf("Error: semaphore_create(): Failed to allocate memory for the semaphore\n");
        return -1;
    }
```



```c
int semaphore_destroy(semaphore_t *sem)
{
    int rtn;

if (USE_NAMED_SEMAPHORES == 0)
{
    rtn = sem_close(sem->sem);
    if( rtn != 0 )
        {
        printf("Error: semaphore_destroy(): sem_close failed with %d (skip sem_unlink)\n", rtn);
        }
    else
        {
        rtn = sem_unlink(sem->name);
        }
}
else
{
    rtn = sem_destroy(sem->sem);

    if(sem->sem != NULL)
        {
        free(sem->sem);
        sem->sem = NULL;
        }
}
    if(sem->name != NULL)
        {
        free(sem->name);
        sem->name = NULL;
        }
    sem->sem  = NULL;
```


- **Threads:**

After creating the semaphore, we create threads. Threads are used to create the car id as the car have a unique ID in the intersection which is given by threads and the car can move according to direction which is given to this id. Otherwise, the car will stay in line.

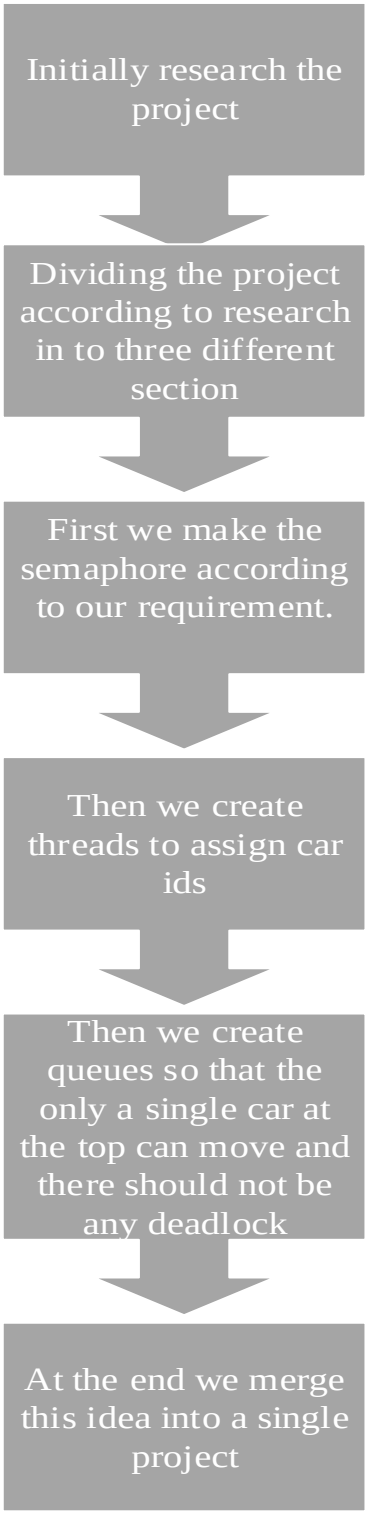
```
if( create_semaphores() != 0 )
    {
    printf("Error: Creating semaphores failed.");
    return -1;
    }
/*
 * Create Car Thread(s)
 *
 */
pthread_t carThreads[num_cars];
for(i = 0; i < num_cars; ++i)
          {
     if(pthread_create(&carThreads[i],NULL,start_car,num_cars)
     {
        printf("Error: Cannot Create thread\n");
        exit(-1);
    }
}
```

- **Queues:**

After creating semaphore and threads we create queues so that queues are used to arrange the lanes. As when one car is at the peek of the mutex locked and this car is move according to his id and when this car dequeuer another car will be at the peek. And at the end of the lane cars enqueue and the cars at the peek will be dequeuer.

```
semaphore_wait(&queueMutex);
    car_t endCarPtr;

    queuePeek(&car_q, &endCarPtr);

semaphore_post(&queueMutex);

semaphore_wait(&conditionSignal);//t1

    semaphore_wait(&isOpenMutex);
    if(this_car.car_id == endCarPtr.car_id && isOpen[this_car.appr_dir] == 1)
    {

        semaphore_wait(&queueMutex);

            dequeue(&car_q, &endCarPtr);

        semaphore_post(&queueMutex);
        isOpen[this_car.appr_dir] = 0;
        this_car.state = STATE_APPROACH_I1;
        print_state(this_car,NULL);

        semaphore_post(&isOpenMutex);
        semaphore_post(&conditionSignal);
        break;
```

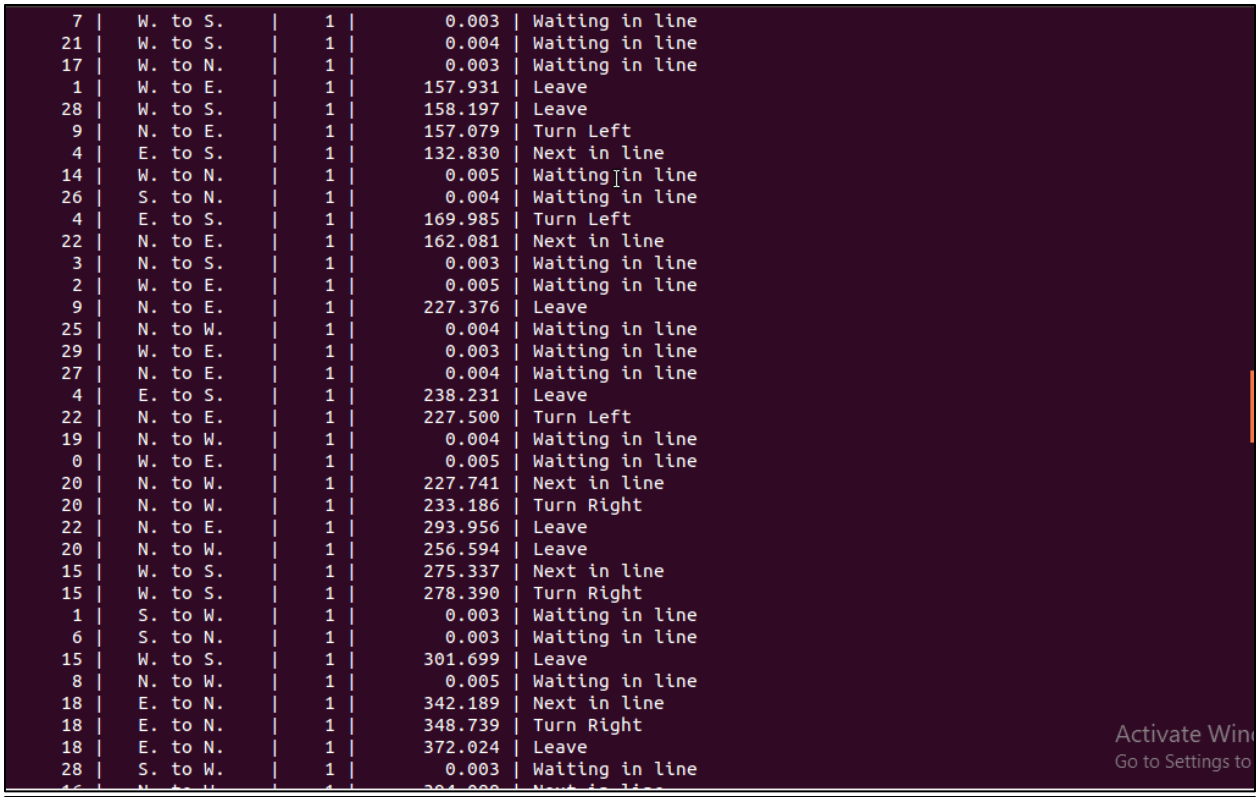
Block Diagram:



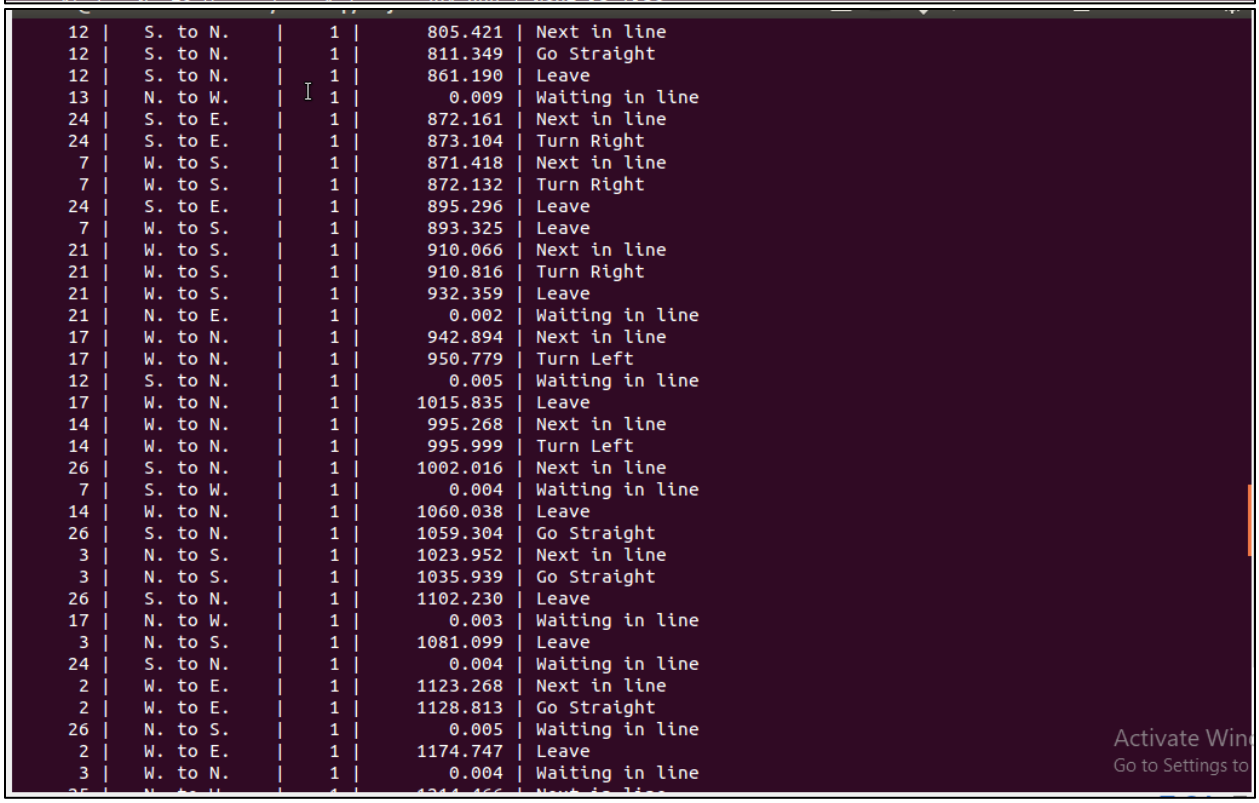
Output:

```
hassan@hassan-VirtualBox:~/Desktop/Project$ gedit semaphore.c
hassan@hassan-VirtualBox:~/Desktop/Project$ gedit support.h
hassan@hassan-VirtualBox:~/Desktop/Project$ gedit support.c
hassan@hassan-VirtualBox:~/Desktop/Project$ gedit stoplight.h
hassan@hassan-VirtualBox:~/Desktop/Project$ gedit stoplight.c
hassan@hassan-VirtualBox:~/Desktop/Project$ gcc -pthread stoplight.c semaphore.c support.c -o output
hassan@hassan-VirtualBox:~/Desktop/Project$ ./output 5 30
Creating a semaphore named: UNNAMED (value 1)
Creating a semaphore named: UNNAMED (value 1)
Creating a semaphore named: UNNAMED (value 1)
Creating a semaphore named: UNNAMED (value 1)
Creating a semaphore named: UNNAMED (value 1)
Creating a semaphore named: UNNAMED (value 1)
Creating a semaphore named: UNNAMED (value 1)
Creating a semaphore named: UNNAMED (value 1)
Creating a semaphore named: UNNAMED (value 1)
Creating a semaphore named: UNNAMED (value 1)
------------------------------
 Car ID | FROM to DEST   | Loc. |  Time (msec) | State
--------+---------------+------+--------------+-------------------------------
     12 |   W. to N.    |    1 |        0.014 | Waiting in line
      5 |   W. to E.    |    1 |        0.003 | Waiting in line
     12 |   W. to N.    |    1 |        1.021 | Next in line
      0 |   W. to S.    |    1 |        0.002 | Waiting in line
     12 |   W. to N.    |    1 |        1.611 | Turn Left
      5 |   W. to E.    |    1 |        1.702 | Next in line
      8 |   S. to N.    |    1 |        0.003 | Waiting in line
      5 |   W. to E.    |    1 |       41.953 | Go Straight
      0 |   W. to S.    |    1 |       41.401 | Next in line
      8 |   S. to N.    |    1 |       35.118 | Next in line
      6 |   S. to E.    |    1 |        0.002 | Waiting in line
     12 |   W. to N.    |    1 |       63.974 | Leave
      0 |   W. to S.    |    1 |       70.265 | Turn Right
      5 |   W. to E.    |    1 |       84.047 | Leave
      8 |   S. to N.    |    1 |       64.757 | Go Straight
```

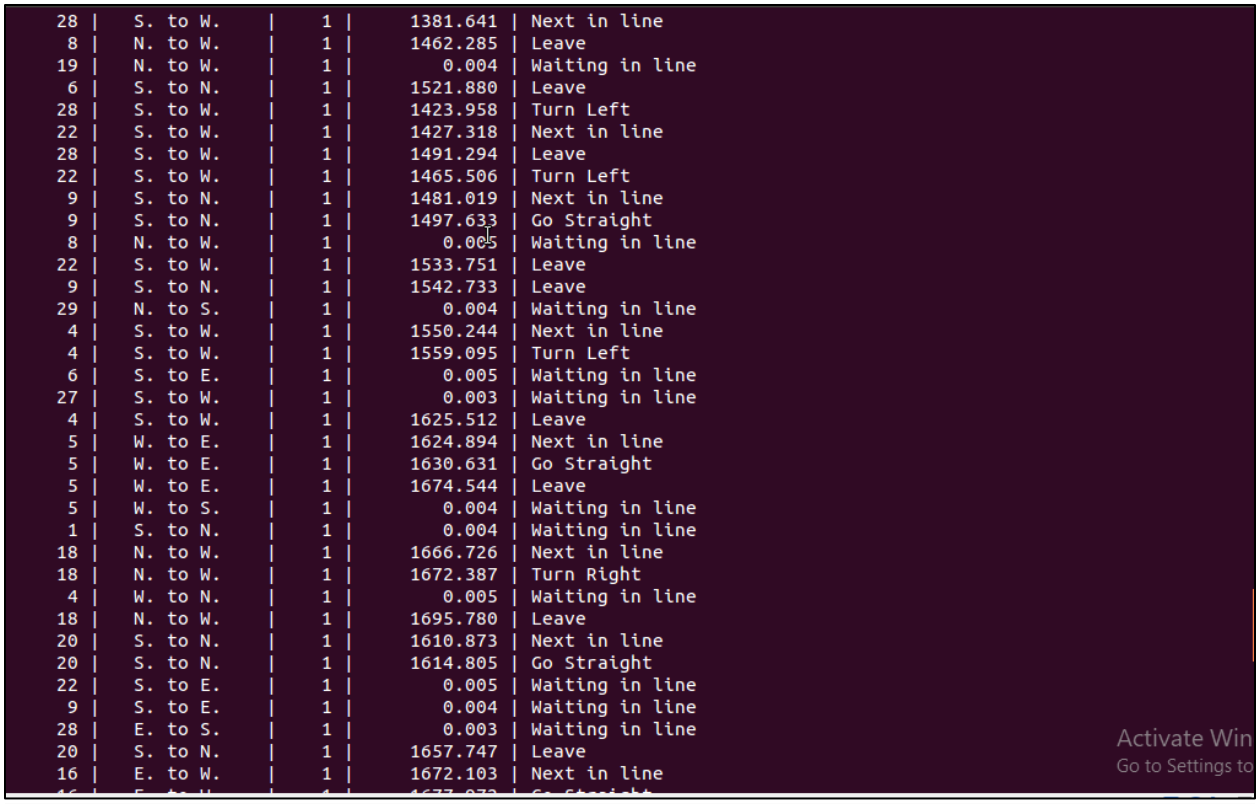
```
      5 |   W. to E.    |    1 |       84.047 | Leave
      8 |   S. to N.    |    1 |       64.757 | Go Straight
      6 |   S. to E.    |    1 |       30.048 | Next in line
      0 |   W. to S.    |    1 |       93.687 | Leave
      5 |   W. to E.    |    1 |        0.003 | Waiting in line
      5 |   W. to E.    |    1 |        3.742 | Next in line
      6 |   S. to E.    |    1 |       50.478 | Turn Right
     25 |   S. to W.    |    1 |        0.004 | Waiting in line
     25 |   S. to W.    |    1 |        1.079 | Next in line
      1 |   W. to E.    |    1 |        0.002 | Waiting in line
     28 |   W. to S.    |    1 |        0.003 | Waiting in line
      9 |   N. to E.    |    1 |        0.002 | Waiting in line
      8 |   S. to N.    |    1 |      112.513 | Leave
      6 |   S. to E.    |    1 |       78.186 | Leave
      5 |   W. to E.    |    1 |       32.683 | Go Straight
      1 |   W. to E.    |    1 |       35.630 | Next in line
      4 |   E. to S.    |    1 |        0.002 | Waiting in line
     22 |   N. to E.    |    1 |        0.005 | Waiting in line
      5 |   W. to E.    |    1 |       89.360 | Leave
     25 |   S. to W.    |    1 |       83.993 | Turn Left
     20 |   N. to W.    |    1 |        0.004 | Waiting in line
     15 |   W. to S.    |    1 |        0.004 | Waiting in line
     18 |   E. to N.    |    1 |        0.003 | Waiting in line
      1 |   W. to E.    |    1 |      104.993 | Go Straight
     16 |   N. to W.    |    1 |        0.008 | Waiting in line
     28 |   W. to S.    |    1 |      107.425 | Next in line
     23 |   E. to S.    |    1 |        0.005 | Waiting in line
     13 |   N. to W.    |    1 |        0.004 | Waiting in line
     10 |   W. to E.    |    1 |        0.003 | Waiting in line
     28 |   W. to S.    |    1 |      132.074 | Turn Right
     11 |   S. to N.    |    1 |        0.003 | Waiting in line
     12 |   S. to N.    |    1 |        0.017 | Waiting in line
      9 |   N. to E.    |    1 |      138.600 | Next in line
     25 |   S. to W.    |    1 |      151.321 | Leave
     24 |   S. to E.    |    1 |        0.003 | Waiting in line
```



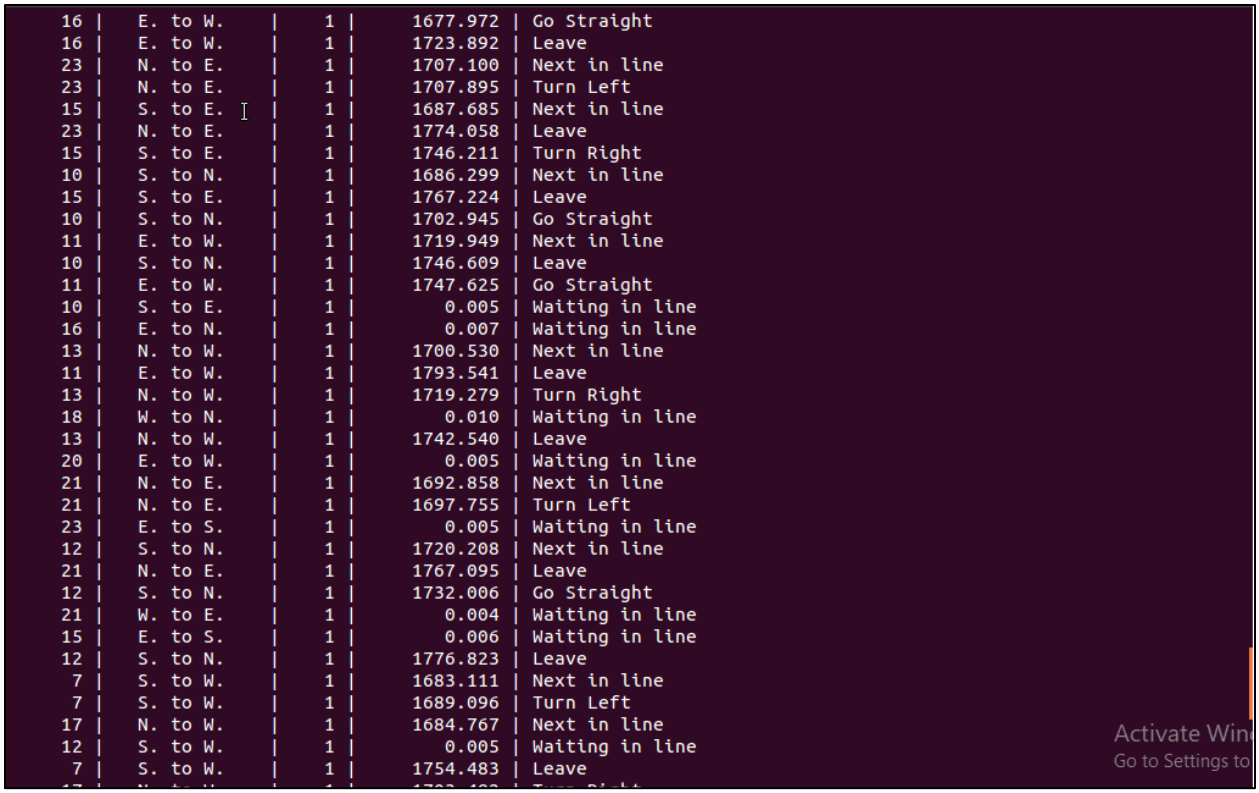
```
  7 |   W. to S.    |    1 |        0.003 | Waiting in line
 21 |   W. to S.    |    1 |        0.004 | Waiting in line
 17 |   W. to N.    |    1 |        0.003 | Waiting in line
  1 |   W. to E.    |    1 |      157.931 | Leave
 28 |   W. to S.    |    1 |      158.197 | Leave
  9 |   N. to E.    |    1 |      157.079 | Turn Left
  4 |   E. to S.    |    1 |      132.830 | Next in line
 14 |   W. to N.    |    1 |        0.005 | Waiting in line
 26 |   S. to N.    |    1 |        0.004 | Waiting in line
  4 |   E. to S.    |    1 |      169.985 | Turn Left
 22 |   N. to E.    |    1 |      162.081 | Next in line
  3 |   N. to S.    |    1 |        0.003 | Waiting in line
  2 |   W. to E.    |    1 |        0.005 | Waiting in line
  9 |   N. to E.    |    1 |      227.376 | Leave
 25 |   N. to W.    |    1 |        0.004 | Waiting in line
 29 |   W. to E.    |    1 |        0.003 | Waiting in line
 27 |   N. to E.    |    1 |        0.004 | Waiting in line
  4 |   E. to S.    |    1 |      238.231 | Leave
 22 |   N. to E.    |    1 |      227.500 | Turn Left
 19 |   N. to W.    |    1 |        0.004 | Waiting in line
  0 |   W. to E.    |    1 |        0.005 | Waiting in line
 20 |   N. to W.    |    1 |      227.741 | Next in line
 20 |   N. to W.    |    1 |      233.186 | Turn Right
 22 |   N. to E.    |    1 |      293.956 | Leave
 20 |   N. to W.    |    1 |      256.594 | Leave
 15 |   W. to S.    |    1 |      275.337 | Next in line
 15 |   W. to S.    |    1 |      278.390 | Turn Right
  1 |   S. to W.    |    1 |        0.003 | Waiting in line
  6 |   S. to N.    |    1 |        0.003 | Waiting in line
 15 |   W. to S.    |    1 |      301.699 | Leave
  8 |   N. to W.    |    1 |        0.005 | Waiting in line
 18 |   E. to N.    |    1 |      342.189 | Next in line
 18 |   E. to N.    |    1 |      348.739 | Turn Right
 18 |   E. to N.    |    1 |      372.024 | Leave
 28 |   S. to W.    |    1 |        0.003 | Waiting in line
```

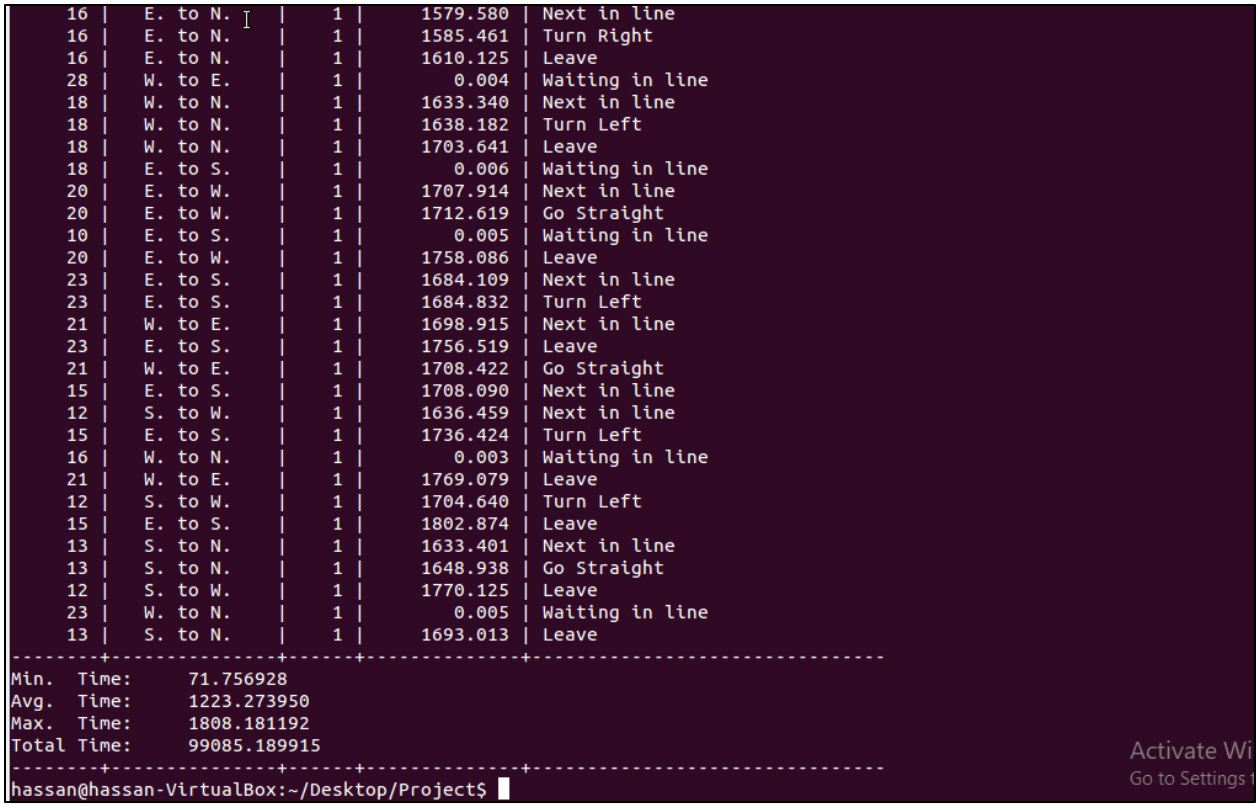
```
 12 |   S. to N.    |    1 |      805.421 | Next in line
 12 |   S. to N.    |    1 |      811.349 | Go Straight
 12 |   S. to N.    |    1 |      861.190 | Leave
 13 |   N. to W.    |    1 |        0.009 | Waiting in line
 24 |   S. to E.    |    1 |      872.161 | Next in line
 24 |   S. to E.    |    1 |      873.104 | Turn Right
  7 |   W. to S.    |    1 |      871.418 | Next in line
  7 |   W. to S.    |    1 |      872.132 | Turn Right
 24 |   S. to E.    |    1 |      895.296 | Leave
  7 |   W. to S.    |    1 |      893.325 | Leave
 21 |   W. to S.    |    1 |      910.066 | Next in line
 21 |   W. to S.    |    1 |      910.816 | Turn Right
 21 |   W. to S.    |    1 |      932.359 | Leave
 21 |   N. to E.    |    1 |        0.002 | Waiting in line
 17 |   W. to N.    |    1 |      942.894 | Next in line
 17 |   W. to N.    |    1 |      950.779 | Turn Left
 12 |   S. to N.    |    1 |        0.005 | Waiting in line
 17 |   W. to N.    |    1 |     1015.835 | Leave
 14 |   W. to N.    |    1 |      995.268 | Next in line
 14 |   W. to N.    |    1 |      995.999 | Turn Left
 26 |   S. to N.    |    1 |     1002.016 | Next in line
  7 |   S. to W.    |    1 |        0.004 | Waiting in line
 14 |   W. to N.    |    1 |     1060.038 | Leave
 26 |   S. to N.    |    1 |     1059.304 | Go Straight
  3 |   N. to S.    |    1 |     1023.952 | Next in line
  3 |   N. to S.    |    1 |     1035.939 | Go Straight
 26 |   S. to N.    |    1 |     1102.230 | Leave
 17 |   N. to W.    |    1 |        0.003 | Waiting in line
  3 |   N. to S.    |    1 |     1081.099 | Leave
 24 |   S. to N.    |    1 |        0.004 | Waiting in line
  2 |   W. to E.    |    1 |     1123.268 | Next in line
  2 |   W. to E.    |    1 |     1128.813 | Go Straight
 26 |   N. to S.    |    1 |        0.005 | Waiting in line
  2 |   W. to E.    |    1 |     1174.747 | Leave
  3 |   W. to N.    |    1 |        0.004 | Waiting in line
```



28	S. to W.	1	1381.641	Next in line
8	N. to W.	1	1462.285	Leave
19	N. to W.	1	0.004	Waiting in line
6	S. to N.	1	1521.880	Leave
28	S. to W.	1	1423.958	Turn Left
22	S. to W.	1	1427.318	Next in line
28	S. to W.	1	1491.294	Leave
22	S. to W.	1	1465.506	Turn Left
9	S. to N.	1	1481.019	Next in line
9	S. to N.	1	1497.633	Go Straight
8	N. to W.	1	0.005	Waiting in line
22	S. to W.	1	1533.751	Leave
9	S. to N.	1	1542.733	Leave
29	N. to S.	1	0.004	Waiting in line
4	S. to W.	1	1550.244	Next in line
4	S. to W.	1	1559.095	Turn Left
6	S. to E.	1	0.005	Waiting in line
27	S. to W.	1	0.003	Waiting in line
4	S. to W.	1	1625.512	Leave
5	W. to E.	1	1624.894	Next in line
5	W. to E.	1	1630.631	Go Straight
5	W. to E.	1	1674.544	Leave
5	W. to S.	1	0.004	Waiting in line
1	S. to N.	1	0.004	Waiting in line
18	N. to W.	1	1666.726	Next in line
18	N. to W.	1	1672.387	Turn Right
4	W. to N.	1	0.005	Waiting in line
18	N. to W.	1	1695.780	Leave
20	S. to N.	1	1610.873	Next in line
20	S. to N.	1	1614.805	Go Straight
22	S. to E.	1	0.005	Waiting in line
9	S. to E.	1	0.004	Waiting in line
28	E. to S.	1	0.003	Waiting in line
20	S. to N.	1	1657.747	Leave
16	E. to W.	1	1672.103	Next in line



16	E. to W.	1	1677.972	Go Straight
16	E. to W.	1	1723.892	Leave
23	N. to E.	1	1707.100	Next in line
23	N. to E.	1	1707.895	Turn Left
15	S. to E.	1	1687.685	Next in line
23	N. to E.	1	1774.058	Leave
15	S. to E.	1	1746.211	Turn Right
10	S. to N.	1	1686.299	Next in line
15	S. to E.	1	1767.224	Leave
10	S. to N.	1	1702.945	Go Straight
11	E. to W.	1	1719.949	Next in line
10	S. to N.	1	1746.609	Leave
11	E. to W.	1	1747.625	Go Straight
10	S. to E.	1	0.005	Waiting in line
16	E. to N.	1	0.007	Waiting in line
13	N. to W.	1	1700.530	Next in line
11	E. to W.	1	1793.541	Leave
13	N. to W.	1	1719.279	Turn Right
18	W. to N.	1	0.010	Waiting in line
13	N. to W.	1	1742.540	Leave
20	E. to W.	1	0.005	Waiting in line
21	N. to E.	1	1692.858	Next in line
21	N. to E.	1	1697.755	Turn Left
23	E. to S.	1	0.005	Waiting in line
12	S. to N.	1	1720.208	Next in line
21	N. to E.	1	1767.095	Leave
12	S. to N.	1	1732.006	Go Straight
21	W. to E.	1	0.004	Waiting in line
15	E. to S.	1	0.006	Waiting in line
12	S. to N.	1	1776.823	Leave
7	S. to W.	1	1683.111	Next in line
7	S. to W.	1	1689.096	Turn Left
17	N. to W.	1	1684.767	Next in line
12	S. to W.	1	0.005	Waiting in line
7	S. to W.	1	1754.483	Leave



Future Work:

According to this project there is a single queue come from one side and travel towards different section only one car at a single direction not multiple car at a single direction at the same time. So, this work can be done at the future. Multiple queues can be added so that many cars can travel from one side to another. Car come from different location so that must be handle in this situation. These are most of the work that can be done in future.