# Project 3 (P3): Traffic Control

By: Taylor D. Williams

Program Description

This project is made to be a reflection of a traffic control system. The goal of this project is to implement this system with pthread lock(s) and semaphores. What was known about this traffic control system is that it controls traffic at an intersection where cars can come from the North, South, East, or West. This was implemented in the project through semaphores and pthreads where each car, which can come from any direction, has its own thread (screenshot #1).

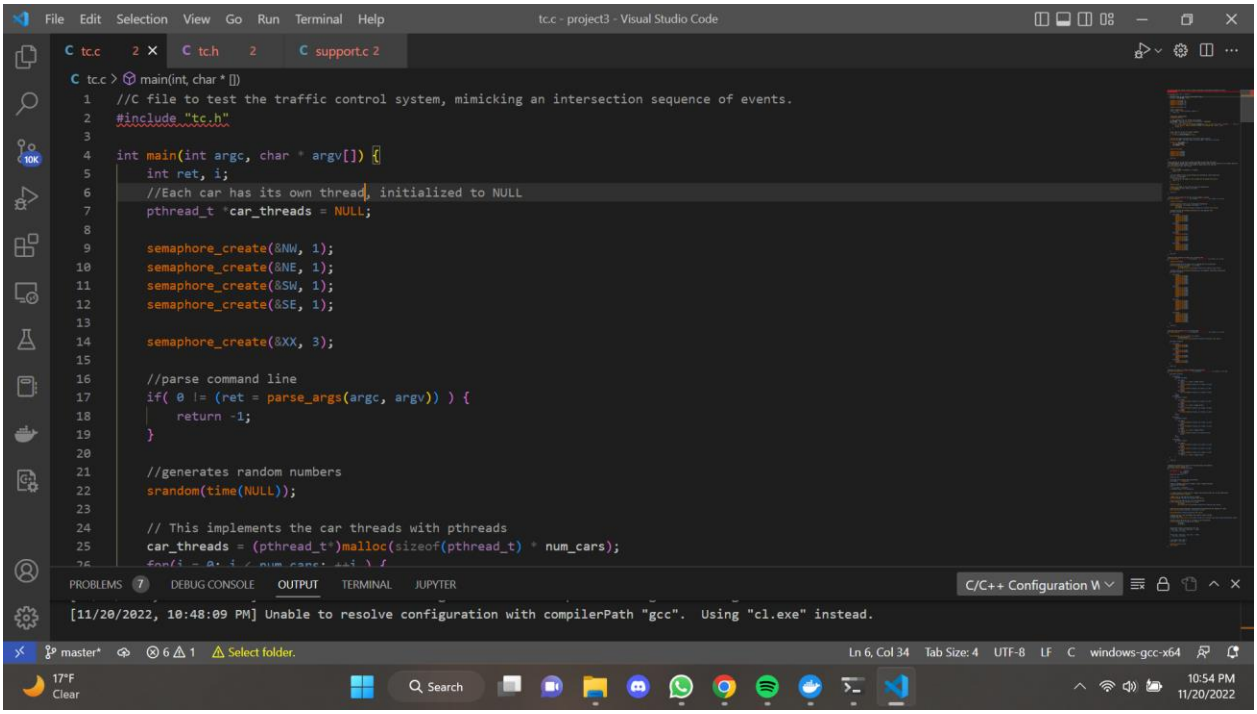
From any direction, the car has the ability to three different tasks, drive through the intersection(int drive_through), which in the output is denoted as “crossing”, turn left (int turning_left) from their position, denoted as “Turning Left(<)” in the output, or turn right (int turning_right) from their position, denoted as “Turning Right(>)” in the output. However, in no way could the cars perform any U-turns.

Each of these possible actions the cars can take were made into functions, where within have a switch function that takes each car’s original direction (dir_original) and sorts by different cases of whatever way the car decides to go (North, South, East, West), an example is shown from the drive_through function in screenshot #2. There are also different functions made to account for the car’s location, when the car has entered the intersection and is performing its action (int enter_intersection)(screenshot #3) and for when the car arrives at the intersection (void *arrive_intersection) and exits, a usleep function is present for when the car is waiting to arrive as well (screenshot #4).

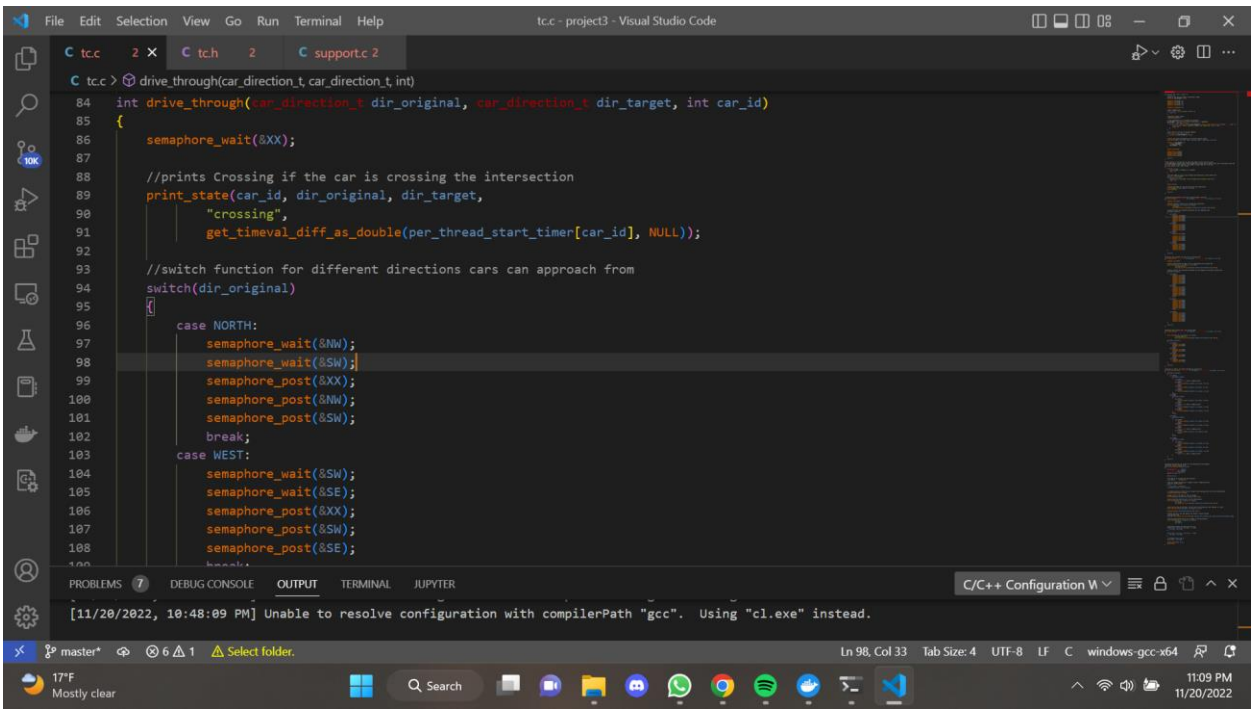
The layout of this project is a little different in terms of format, and the time at which each car arrives and exits. I set it to have each car randomly pick an action to take at the intersection to save time on having to create each car's individual actions. I did this so that when testing I can pick out however many cars I want to go through the intersection and see if they perform the actions required within a reasonable time frame (as seen in outputs screenshots). Overall, the output is not identical, but cars from different directions are able to pass through the intersections in a reasonable amount of time, going either straightforward, turning left or right, then exiting. The output displays the car ID, the car’s original direction, the car’s target direction, the actions they perform in order, and the time in double.

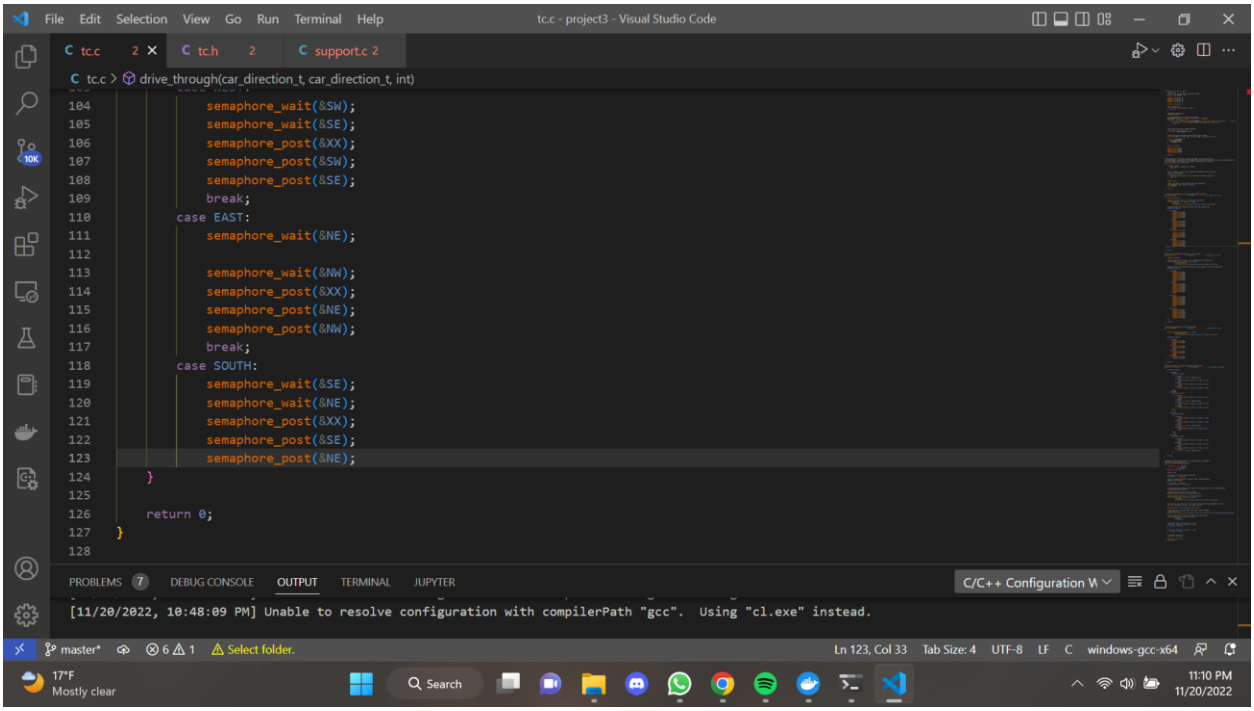
Screenshots

#1

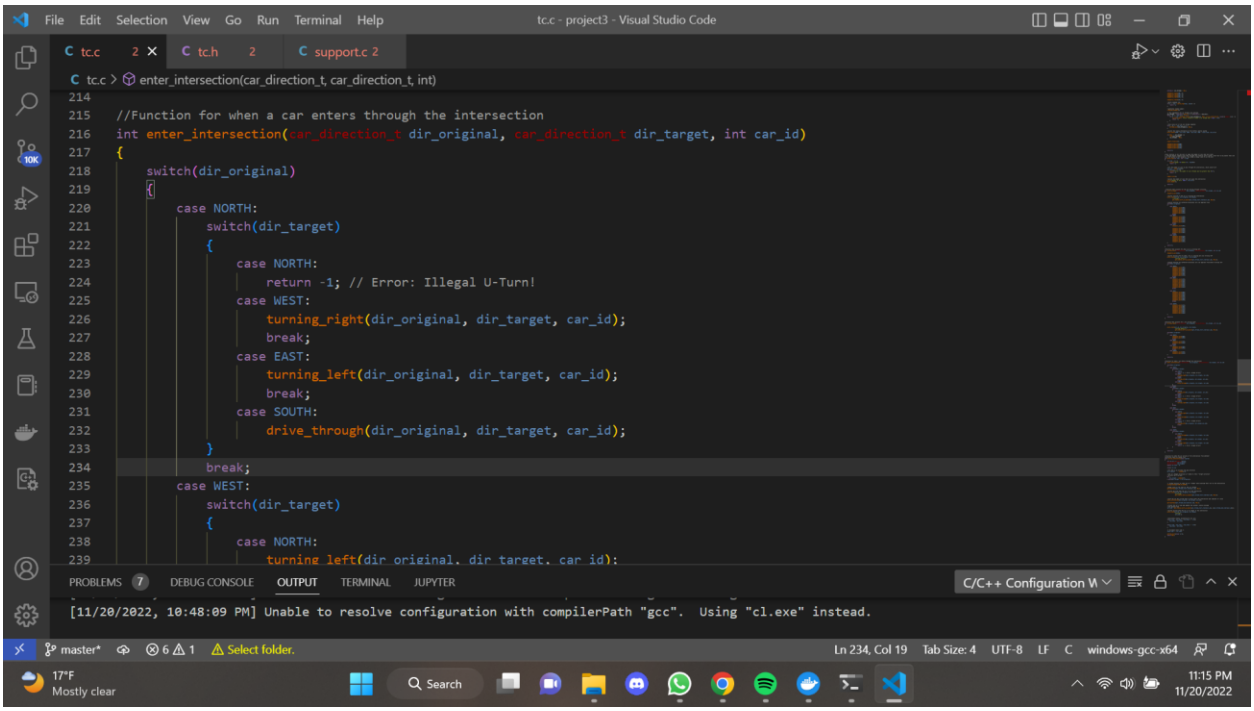



#2

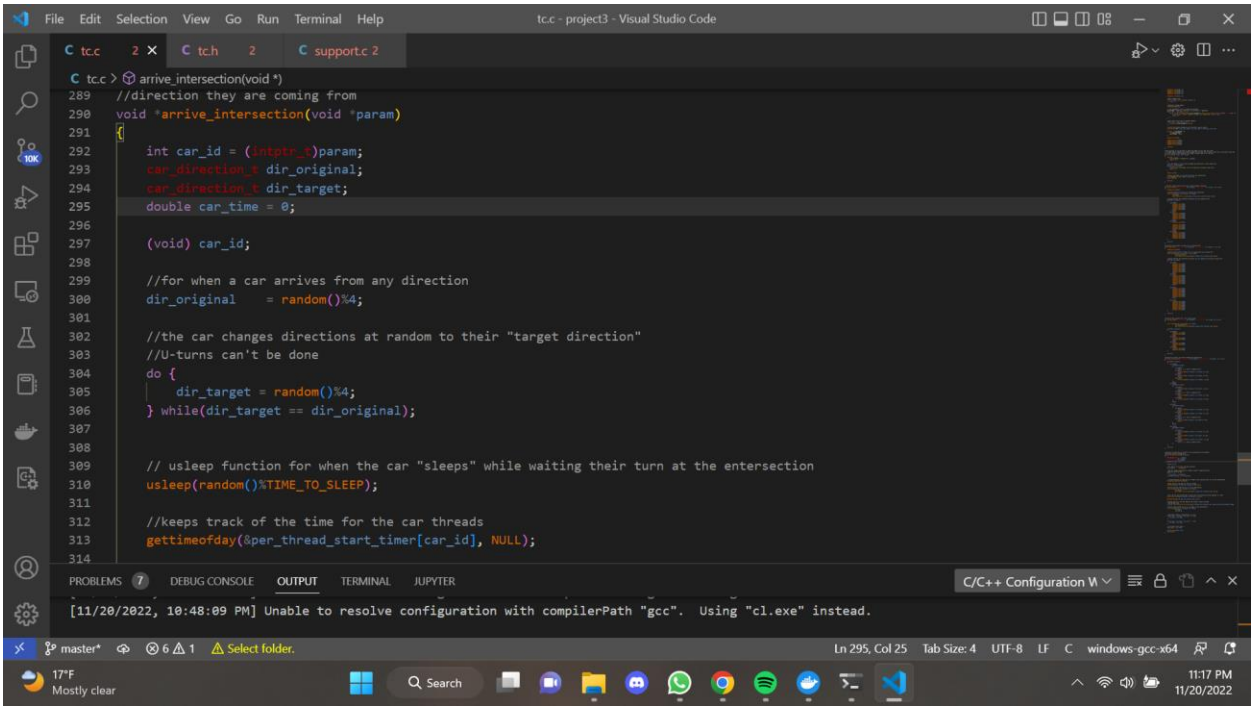


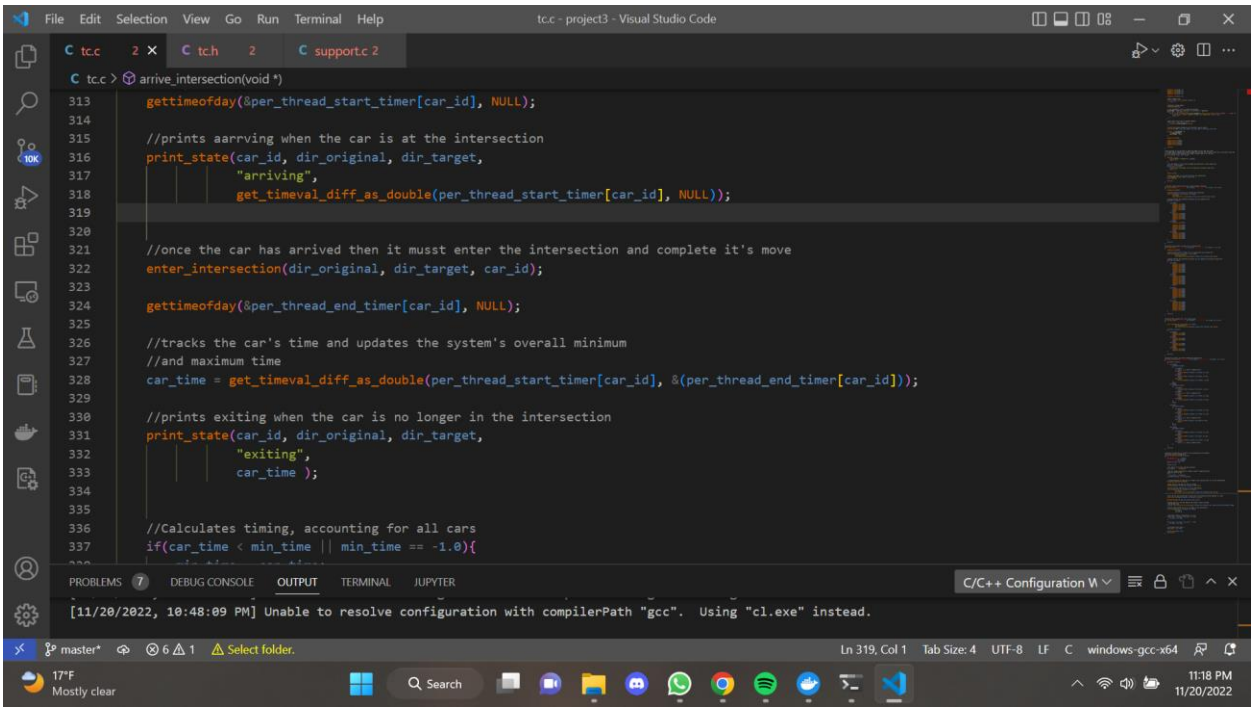

File Edit Selection View Go Run Terminal Help
tc.c - project3 - Visual Studio Code
tc.c 2
tch 2
support.c 2
tc.c > drive_through(car_direction_t, car_direction_t, int)
semaphore_wait(&SW);
semaphore_wait(&SE);
semaphore_post(&XX);
semaphore_post(&SW);
semaphore_post(&SE);
break;
case EAST:
semaphore_wait(&NE);
semaphore_wait(&NW);
semaphore_post(&XX);
semaphore_post(&NE);
semaphore_post(&NW);
break;
case SOUTH:
semaphore_wait(&SE);
semaphore_wait(&NE);
semaphore_post(&XX);
semaphore_post(&SE);
semaphore_post(&NE);
}
return 0;
}
PROBLEMS 7 DEBUG CONSOLE OUTPUT TERMINAL JUPYTER
C/C++ Configuration W
[11/20/2022, 10:48:09 PM] Unable to resolve configuration with compilerPath "gcc". Using "cl.exe" instead.
master* 6 1 Select folder.
Ln 123, Col 33 Tab Size: 4 UTF-8 LF C windows-gcc-x64
17°F Mostly clear
Search
11:10 PM 11/20/2022

#3


File Edit Selection View Go Run Terminal Help
tc.c - project3 - Visual Studio Code
tc.c 2
tch 2
support.c 2
tc.c > enter_intersection(car_direction_t, car_direction_t, int)
//Function for when a car enters through the intersection
int enter_intersection(car_direction_t dir_original, car_direction_t dir_target, int car_id)
{
switch(dir_original)
{
case NORTH:
switch(dir_target)
{
case NORTH:
return -1; // Error: Illegal U-Turn!
case WEST:
turning_right(dir_original, dir_target, car_id);
break;
case EAST:
turning_left(dir_original, dir_target, car_id);
break;
case SOUTH:
drive_through(dir_original, dir_target, car_id);
}
break;
case WEST:
switch(dir_target)
{
case NORTH:
turning_left(dir_original, dir_target, car_id);
PROBLEMS 7 DEBUG CONSOLE OUTPUT TERMINAL JUPYTER
C/C++ Configuration W
[11/20/2022, 10:48:09 PM] Unable to resolve configuration with compilerPath "gcc". Using "cl.exe" instead.
master* 6 1 Select folder.
Ln 234, Col 19 Tab Size: 4 UTF-8 LF C windows-gcc-x64
17°F Mostly clear
Search
11:15 PM 11/20/2022

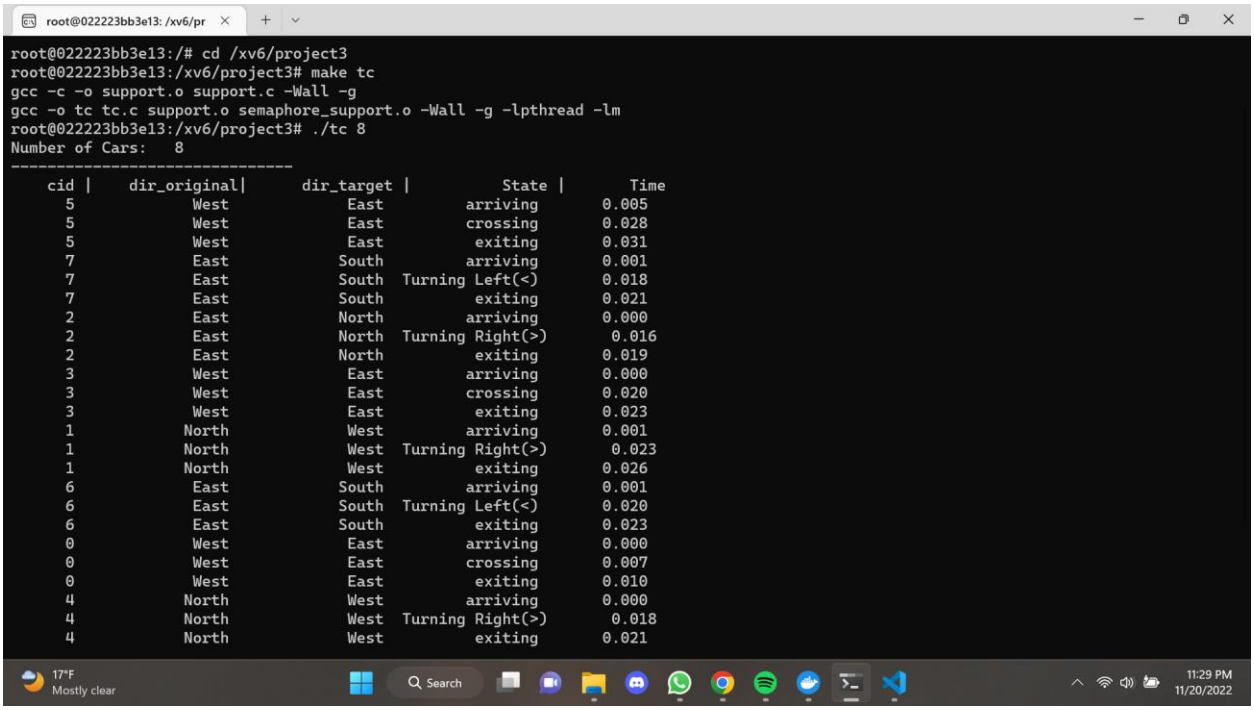
#4


File Edit Selection View Go Run Terminal Help
tc.c - project3 - Visual Studio Code
tc.c 2
tc.h 2
support.c 2
tc.c > arrive_intersection(void *)
//direction they are coming from
void *arrive_intersection(void *param)
{
int car_id = (intptr_t)param;
car_direction_t dir_original;
car_direction_t dir_target;
double car_time = 0;
(void) car_id;
//for when a car arrives from any direction
dir_original = random()%4;
//the car changes directions at random to their "target direction"
//U-turns can't be done
do {
dir_target = random()%4;
} while(dir_target == dir_original);
// usleep function for when the car "sleeps" while waiting their turn at the entersection
usleep(random()%TIME_TO_SLEEP);
//keeps track of the time for the car threads
gettimeofday(&per_thread_start_timer[car_id], NULL);
PROBLEMS 7
DEBUG CONSOLE
OUTPUT
TERMINAL
JUPYTER
C/C++ Configuration W
[11/20/2022, 10:48:09 PM] Unable to resolve configuration with compilerPath "gcc". Using "cl.exe" instead.
master*
6 1
Select folder.
Ln 295, Col 25
Tab Size: 4
UTF-8
LF
C
windows-gcc-x64
17°F
Mostly clear
Search
11:17 PM
11/20/2022


File Edit Selection View Go Run Terminal Help
tc.c - project3 - Visual Studio Code
tc.c 2
tc.h 2
support.c 2
tc.c > arrive_intersection(void *)
gettimeofday(&per_thread_start_timer[car_id], NULL);
//prints aarrving when the car is at the intersection
print_state(car_id, dir_original, dir_target,
"arriving",
get_timeval_diff_as_double(per_thread_start_timer[car_id], NULL));
//once the car has arrived then it musst enter the intersection and complete it's move
enter_intersection(dir_original, dir_target, car_id);
gettimeofday(&per_thread_end_timer[car_id], NULL);
//tracks the car's time and updates the system's overall minimum
//and maximum time
car_time = get_timeval_diff_as_double(per_thread_start_timer[car_id], &(per_thread_end_timer[car_id]));
//prints exiting when the car is no longer in the intersection
print_state(car_id, dir_original, dir_target,
"exiting",
car_time );
//Calculates timing, accounting for all cars
if(car_time < min_time || min_time == -1.0){
PROBLEMS 7
DEBUG CONSOLE
OUTPUT
TERMINAL
JUPYTER
C/C++ Configuration W
[11/20/2022, 10:48:09 PM] Unable to resolve configuration with compilerPath "gcc". Using "cl.exe" instead.
master*
6 1
Select folder.
Ln 319, Col 1
Tab Size: 4
UTF-8
LF
C
windows-gcc-x64
17°F
Mostly clear
Search
11:18 PM
11/20/2022

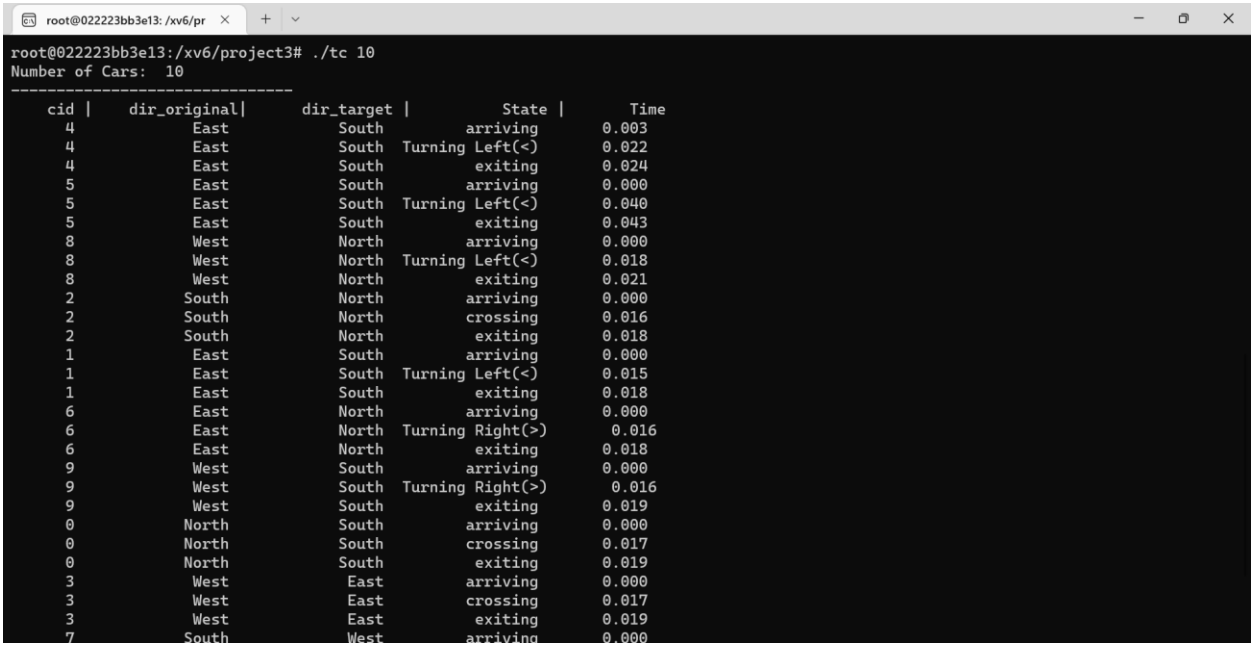
## Outputs

### With 8 cars

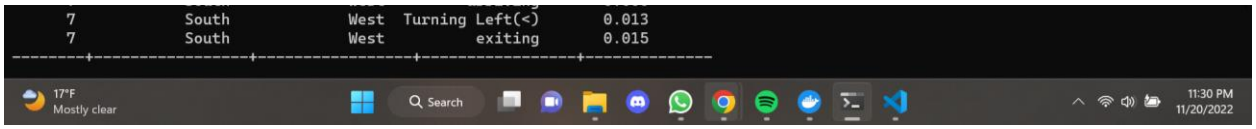


```
root@022223bb3e13:/# cd /xv6/project3
root@022223bb3e13:/xv6/project3# make tc
gcc -c -o support.o support.c -Wall -g
gcc -o tc tc.c support.o semaphore_support.o -Wall -g -lpthread -lm
root@022223bb3e13:/xv6/project3# ./tc 8
Number of Cars:   8
------------------------------
    cid |    dir_original|      dir_target |          State |       Time
      5            West            East        arriving       0.005
      5            West            East        crossing       0.028
      5            West            East         exiting       0.031
      7            East           South        arriving       0.001
      7            East           South  Turning Left(<)       0.018
      7            East           South         exiting       0.021
      2            East           North        arriving       0.000
      2            East           North  Turning Right(>)       0.016
      2            East           North         exiting       0.019
      3            West            East        arriving       0.000
      3            West            East        crossing       0.020
      3            West            East         exiting       0.023
      1           North            West        arriving       0.001
      1           North            West  Turning Right(>)       0.023
      1           North            West         exiting       0.026
      6            East           South        arriving       0.001
      6            East           South  Turning Left(<)       0.020
      6            East           South         exiting       0.023
      0            West            East        arriving       0.000
      0            West            East        crossing       0.007
      0            West            East         exiting       0.010
      4           North            West        arriving       0.000
      4           North            West  Turning Right(>)       0.018
      4           North            West         exiting       0.021
```

### With 10 cars



```
root@022223bb3e13:/xv6/project3# ./tc 10
Number of Cars:  10
------------------------------
    cid |    dir_original|      dir_target |          State |       Time
      4            East           South        arriving       0.003
      4            East           South  Turning Left(<)       0.022
      4            East           South         exiting       0.024
      5            East           South        arriving       0.000
      5            East           South  Turning Left(<)       0.040
      5            East           South         exiting       0.043
      8            West           North        arriving       0.000
      8            West           North  Turning Left(<)       0.018
      8            West           North         exiting       0.021
      2           South           North        arriving       0.000
      2           South           North        crossing       0.016
      2           South           North         exiting       0.018
      1            East           South        arriving       0.000
      1            East           South  Turning Left(<)       0.015
      1            East           South         exiting       0.018
      6            East           North        arriving       0.000
      6            East           North  Turning Right(>)       0.016
      6            East           North         exiting       0.018
      9            West           South        arriving       0.000
      9            West           South  Turning Right(>)       0.016
      9            West           South         exiting       0.019
      0           North           South        arriving       0.000
      0           North           South        crossing       0.017
      0           North           South         exiting       0.019
      3            West            East        arriving       0.000
      3            West            East        crossing       0.017
      3            West            East         exiting       0.019
      7           South            West        arriving       0.000
```




**Source Code (canvas was not allowing .c files to upload)**

tc.c

```c
//C file to test the traffic control system, mimicking an intersection
sequence of events.
#include "tc.h"

int main(int argc, char * argv[]) {
    int ret, i;
    //Each car has its own thread, initialized to NULL
    pthread_t *car_threads = NULL;

    semaphore_create(&NW, 1);
    semaphore_create(&NE, 1);
    semaphore_create(&SW, 1);
    semaphore_create(&SE, 1);

    semaphore_create(&XX, 3);

    //parse command line
    if( 0 != (ret = parse_args(argc, argv)) ) {
        return -1;
    }

    //generates random numbers
    srandom(time(NULL));

    // This implements the car threads with pthreads
    car_threads = (pthread_t*)malloc(sizeof(pthread_t) * num_cars);
    for(i = 0; i < num_cars; ++i ) {
```

```
        if( 0 != (ret = pthread_create(&(car_threads[i]), NULL,
arrive_intersection, (void*)(intptr_t)i)) ) {
            fprintf(stderr, "Error: Failed to create a car thread! Ret
= %d\n", ret);
            return -1;
        }
    }

    //This joins all of the car threads together
    for(i = 0; i < num_cars; ++i ) {
        pthread_join(car_threads[i], NULL);
    }

    //prints the timing information of the traffic control system
    print_footer(1000 * min_time, 1000 * max_time, 1000 * total_time,
num_cars);

    if( NULL != car_threads ) {
        free(car_threads);
        car_threads = NULL;
    }

    support_finalize();

    semaphore_destroy(&NW);
    semaphore_destroy(&NE);
    semaphore_destroy(&SW);
    semaphore_destroy(&SE);

    return 0;
}

//This function is for the user to input the number of cars that will
enter
// (through Ubuntu command line) when throught the traffick contrl system,
which has to be greater than zero
//for the system to work properly, this helps to keep track of all the
cars
```

```c
int parse_args(int argc, char **argv)
{
    if( argc < 2 ) {
        printf("Usage: %s NumCars\n", argv[0]);
        return -1;
    }

    //for the number of cars to pass through the intersection, which
cannot be 0
    num_cars = atoi(argv[1]);
    if( num_cars <= 0 ) {
        printf("Error: The number of cars threads must be greater than
0\n");
        return -1;
    }

    support_init();

    //prints the number of cars that will pass the intersection
    printf("Number of Cars: %3d\n", num_cars);
    print_header();

    return 0;
}


 //function that accounts for the car driving through (crossing)
int drive_through(car_direction_t dir_original, car_direction_t
dir_target, int car_id)
{
    semaphore_wait(&XX);

    //prints Crossing if the car is crossing the intersection
    print_state(car_id, dir_original, dir_target,
            "crossing",
            get_timeval_diff_as_double(per_thread_start_timer[car_id],
NULL));
```

```
    //switch function for different directions cars can approach from
    switch(dir_original)
    {
        case NORTH:
            semaphore_wait(&NW);
            semaphore_wait(&SW);
            semaphore_post(&XX);
            semaphore_post(&NW);
            semaphore_post(&SW);
            break;
        case WEST:
            semaphore_wait(&SW);
            semaphore_wait(&SE);
            semaphore_post(&XX);
            semaphore_post(&SW);
            semaphore_post(&SE);
            break;
        case EAST:
            semaphore_wait(&NE);

            semaphore_wait(&NW);
            semaphore_post(&XX);
            semaphore_post(&NE);
            semaphore_post(&NW);
            break;
        case SOUTH:
            semaphore_wait(&SE);
            semaphore_wait(&NE);
            semaphore_post(&XX);
            semaphore_post(&SE);
            semaphore_post(&NE);
    }

    return 0;
}

//Function that accounts for when a car is turning left
```

```
int turning_left(car_direction_t dir_original, car_direction_t dir_target,
int car_id)
{
    semaphore_wait(&XX);

    //prints Turning Left for when a car is crossing but also turning left
    print_state(car_id, dir_original , dir_target,
                "Turning Left(<)",
                get_timeval_diff_as_double(per_thread_start_timer[car_id],
NULL));

    //switch function for different directions cars can approach from
before turning left
    switch(dir_original)
    {
        case NORTH:
            semaphore_wait(&NW);
            semaphore_wait(&SW);
            semaphore_post(&NW);
            semaphore_wait(&SE);
            semaphore_post(&XX);
            semaphore_post(&SW);
            semaphore_post(&SE);
            break;
        case WEST:
            semaphore_wait(&SW);
            semaphore_wait(&SE);
            semaphore_post(&SW);
            semaphore_wait(&NE);
            semaphore_post(&XX);
            semaphore_post(&SE);
            semaphore_post(&NE);
            break;

        case EAST:
            semaphore_wait(&NE);
            semaphore_wait(&NW);
            semaphore_post(&NE);
```

```
            semaphore_wait(&SW);
            semaphore_post(&XX);
            semaphore_post(&NW);
            semaphore_post(&SW);
            break;

        case SOUTH:
            semaphore_wait(&SE);
            semaphore_wait(&NE);
            semaphore_post(&SE);
            semaphore_wait(&NW);
            semaphore_post(&XX);
            semaphore_post(&NE);
            semaphore_post(&NW);
    }

    return 0;
}


//Function that accounts for a car turning right
int turning_right(car_direction_t dir_original, car_direction_t
dir_target, int car_id)
{

    print_state(car_id, dir_original, dir_target,
                "Turning Right(>)",
                get_timeval_diff_as_double(per_thread_start_timer[car_id],
NULL));

    switch(dir_original)
    {
        case NORTH:
            semaphore_wait(&NW);
            semaphore_post(&NW);
            break;
        case WEST:
            semaphore_wait(&SW);
```

```
            semaphore_post(&SW);
            break;
        case EAST:
            semaphore_wait(&NE);
            semaphore_post(&NE);
            break;
        case SOUTH:
            semaphore_wait(&SE);
            semaphore_post(&SE);
    }

    return 0;
}

//Function for when a car enters through the intersection
int enter_intersection(car_direction_t dir_original, car_direction_t
dir_target, int car_id)
{
    switch(dir_original)
    {
        case NORTH:
            switch(dir_target)
            {
                case NORTH:
                    return -1; // Error: Illegal U-Turn!
                case WEST:
                    turning_right(dir_original, dir_target, car_id);
                    break;
                case EAST:
                    turning_left(dir_original, dir_target, car_id);
                    break;
                case SOUTH:
                    drive_through(dir_original, dir_target, car_id);
            }
            break;
        case WEST:
            switch(dir_target)
            {
```

```
            case NORTH:
                turning_left(dir_original, dir_target, car_id);
                break;
            case WEST:
                return -1; // Error: Illegal U-Turn!
            case EAST:
                drive_through(dir_original, dir_target, car_id);
                break;
            case SOUTH:
                turning_right(dir_original, dir_target, car_id);
        }
        break;

    case EAST:
        switch(dir_target)
        {
            case NORTH:
                turning_right(dir_original, dir_target, car_id);
                break;
            case WEST:
                drive_through(dir_original, dir_target, car_id);
                break;
            case EAST:
                return -1; // Error: Illegal U-Turn!
            case SOUTH:
                turning_left(dir_original, dir_target,car_id);
                break;
        }
        break;

    case SOUTH:
        switch(dir_target)
        {
            case NORTH:
                drive_through(dir_original, dir_target, car_id);
                break;
            case WEST:
                turning_left(dir_original, dir_target, car_id);
```

```
                    break;
                case EAST:
                    turning_right(dir_original, dir_target, car_id);
                case SOUTH:
                    return -1; // Error: Illegal U-Turn!
            }
    }

    return 0;
}


//function for when the car arrives at the intersection from whatever
//direction they are coming from
void *arrive_intersection(void *param)
{
    int car_id = (intptr_t)param;
    car_direction_t dir_original;
    car_direction_t dir_target;
    double car_time = 0;

    (void) car_id;

    //for when a car arrives from any direction
    dir_original    = random()%4;

    //the car changes directions at random to their "target direction"
    //U-turns can't be done
    do {
        dir_target = random()%4;
    } while(dir_target == dir_original);


    // usleep function for when the car "sleeps" while waiting their turn
at the entersection
    usleep(random()%TIME_TO_SLEEP);

    //keeps track of the time for the car threads
```

```
    gettimeofday(&per_thread_start_timer[car_id], NULL);

    //prints aarrving when the car is at the intersection
    print_state(car_id, dir_original, dir_target,
                "arriving",
                get_timeval_diff_as_double(per_thread_start_timer[car_id],
NULL));


    //once the car has arrived then it musst enter the intersection and
complete it's move
    enter_intersection(dir_original, dir_target, car_id);

    gettimeofday(&per_thread_end_timer[car_id], NULL);

    //tracks the car's time and updates the system's overall minimum
    //and maximum time
    car_time = get_timeval_diff_as_double(per_thread_start_timer[car_id],
&(per_thread_end_timer[car_id]));

    //prints exiting when the car is no longer in the intersection
    print_state(car_id, dir_original, dir_target,
                "exiting",
                car_time );


    //Calculates timing, accounting for all cars
    if(car_time < min_time || min_time == -1.0){
        min_time = car_time;
    }

    if(car_time > max_time || max_time == -1.0){
        max_time = car_time;
    }

    /* Increment total time */
    total_time += car_time;
```

```
    pthread_exit((void *) 0);
    return NULL;
}
```

support.c (addition to tc.c)

```
#include "support.h"

static int initialized = FALSE;

int support_init(void) {
    int ret;

    ret = semaphore_create(&support_print_lock,  1);
    initialized = TRUE;

    return ret;
}

int support_finalize(void) {
    int ret;

    ret = semaphore_destroy(&support_print_lock);
    initialized = FALSE;

    return ret;
}

void print_footer(double min_time, double max_time, double total_time, int
num_cars) {
    if( FALSE == initialized ) {
        fprintf(stderr, "Warning: You forgot to call support_init() before
calling print_footer()\n");
        support_init();
    }
```

```
    printf("--------+-----------------+-----------------+-----------------
+--------------\n");
    printf("Min.  Time :%12f msec\n", min_time);
    printf("Avg.  Time :%12f msec\n", total_time / num_cars);
    printf("Max.  Time :%12f msec\n", max_time);
    printf("Total Time :%12f msec\n", total_time);
    printf("--------+-----------------+-----------------+-----------------
+--------------\n");
}

void print_header(void) {
    if( FALSE == initialized ) {
        fprintf(stderr, "Warning: You forgot to call support_init() before
calling print_header()\n");
        support_init();
    }

    printf("-------------------------------\n");
    printf("%7s | %15s| %15s |%15s | %10s\n", "cid", "dir_original",
"dir_target", "State", "Time");
    //printf("--------+-----------------+-----------------+--------------
--+--------------\n");
}

void print_state(int car_id, car_direction_t dir_original, car_direction_t
dir_target, char * state, double timer) {
    if( FALSE == initialized ) {
        fprintf(stderr, "Warning: You forgot to call support_init() before
calling print_state()\n");
        support_init();
    }

    semaphore_wait(&support_print_lock);

    printf("%7d  %15s  %15s  %15s  %10.3f\n",
           car_id,
           (dir_original == NORTH ? "North" :
```

```
             (dir_original == WEST ? "West" :
              (dir_original== EAST ? "East" :
               (dir_original == SOUTH ? "South" : "?")))),
            (dir_target== NORTH ? "North" :
             (dir_target == WEST ? "West" :
              (dir_target== EAST ? "East" :
               (dir_target == SOUTH ? "South" : "?")))),
            state,
            timer*TIME_MSEC);

    semaphore_post(&support_print_lock);

    return;
}

double timeval_to_double(struct timeval ctime) {
    if( FALSE == initialized ) {
        fprintf(stderr, "Warning: You forgot to call support_init() before
calling timeval_to_double()\n");
        support_init();
    }

    return (ctime.tv_sec + (ctime.tv_usec/(1.0 + TIME_USEC)));
}

struct timeval get_timeval_diff_as_timeval(struct timeval start, struct
timeval end) {
    struct timeval loc_diff;

    if( FALSE == initialized ) {
        fprintf(stderr, "Warning: You forgot to call support_init() before
calling get_timeval_diff_as_timeval()\n");
        support_init();
    }


    if( end.tv_usec < start.tv_usec ) {
        loc_diff.tv_usec = (TIME_USEC - start.tv_usec) + end.tv_usec;
```

```
        end.tv_sec -= 1;
    } else {
        loc_diff.tv_usec = end.tv_usec - start.tv_usec;
    }

    loc_diff.tv_sec = end.tv_sec - start.tv_sec;

    return loc_diff;
}

double get_timeval_diff_as_double(struct timeval start, struct timeval
*given_end) {
    struct timeval loc_diff, end;

    if( FALSE == initialized ) {
        fprintf(stderr, "Warning: You forgot to call support_init() before
calling get_timeval_diff_as_double()\n");
        support_init();
    }

    if( NULL == given_end ) {
        gettimeofday(&end, NULL);
    } else {
        end.tv_sec  = given_end->tv_sec;
        end.tv_usec = given_end->tv_usec;
    }

    loc_diff = get_timeval_diff_as_timeval(start, end);

    return timeval_to_double(loc_diff);
}
```

tc.h (initilizes functions for tc.c)

```
//header to define the functions that will be used for the
```

```c
//traffic control system test file tc.c
#include "support.h"

//defines the intersections coming from North, South, East or West
#define XSECT_NW 0
#define XSECT_NE 1
#define XSECT_SE 2
#define XSECT_SW 3


//for the number of cars threads that will go through the traffic control system
//which can be determined by user in an Ubuntu command line
int num_cars = 0;

//the minimum, maximum and total time a car spent in the sytem
double min_time = -1.0;
double max_time = -1.0;
double total_time = 0;

//tracks the start and end times for each car
struct timeval per_thread_start_timer[1000];
struct timeval per_thread_end_timer[1000];

//Semaphores to lock each quadrant North, South, East and West
semaphore_t NW;
semaphore_t NE;
semaphore_t SW;
semaphore_t SE;

//limit of 3 cars in the intersection that aren't exiting
semaphore_t XX;


//parse command line
int parse_args(int argc, char **argv);
```

```
//This function is for when a car goess straight through the intersection
//from any direction. Has arguments for each Car's ID (car_id) and the
original
//direction the car arrives from (dir_original) and its target direction
(dir_target)
int drive_through(car_direction_t dir_original, car_direction_t
dir_target, int car_id);


//This function is for when a car in any direction decides to turn left
int turning_left(car_direction_t dir_original, car_direction_t dir_target,
int car_id);


//This function is for when a car in any direction decides to turn right
int turning_right(car_direction_t dir_original, car_direction_t
dir_target, int car_id);


//This function is for when a car is in the intersection, it decides if a
car
//is going to turn left, right, or drive through
int enter_intersection(car_direction_t dir_original, car_direction_t
dir_target, int car_id);


//This main thread function is for when a car arrives to the intersection
void *arrive_intersection(void *param);
```