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).

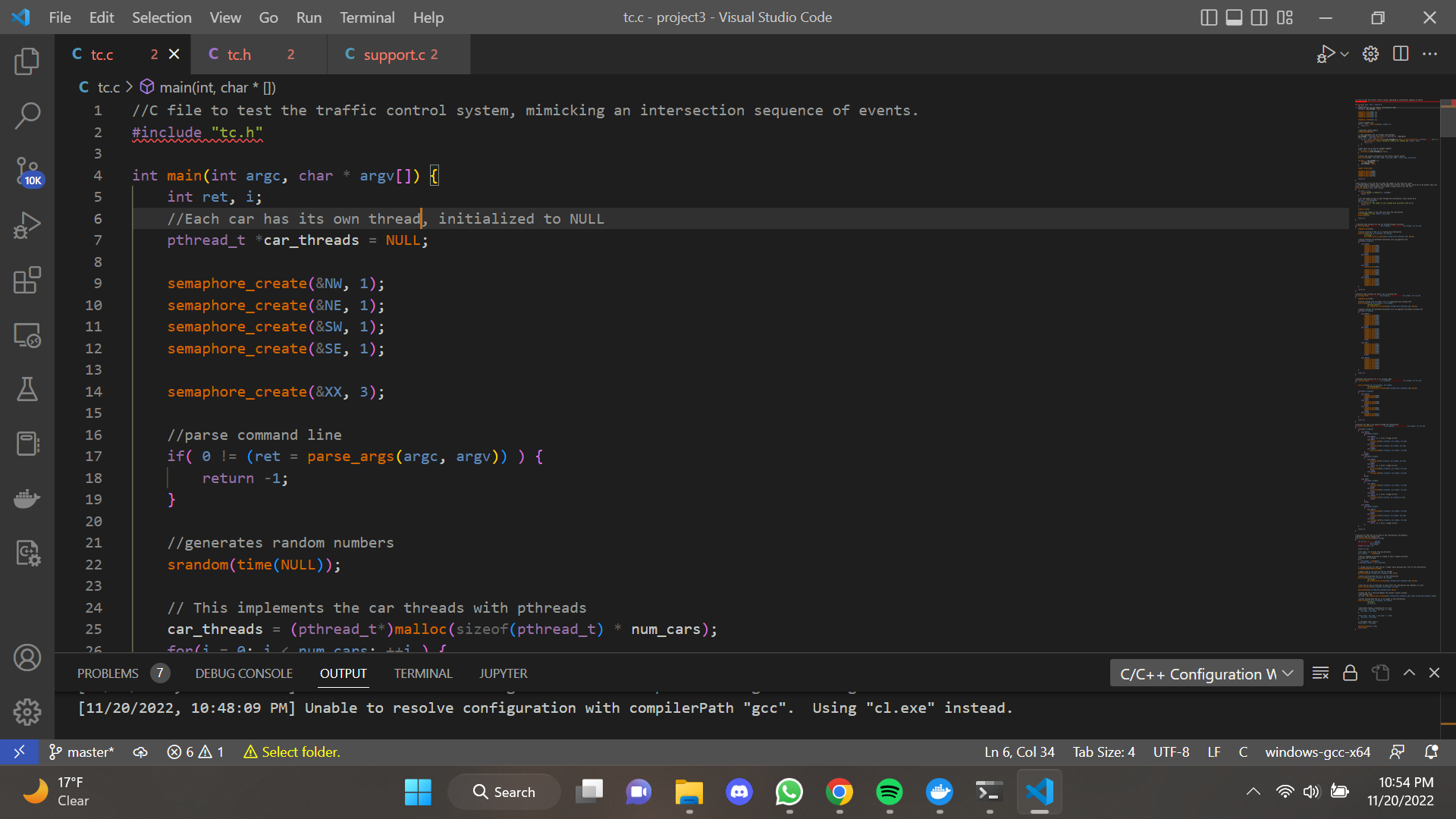
From any direction, the car has the ability to three different tasks, drive through the intersection(int drive\_through), which in the output is denoted as “crossing”, turn left (int turning\_left) from their position, denoted as “Turning Left(<)” in the output, or turn right (int turning\_right) from 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).

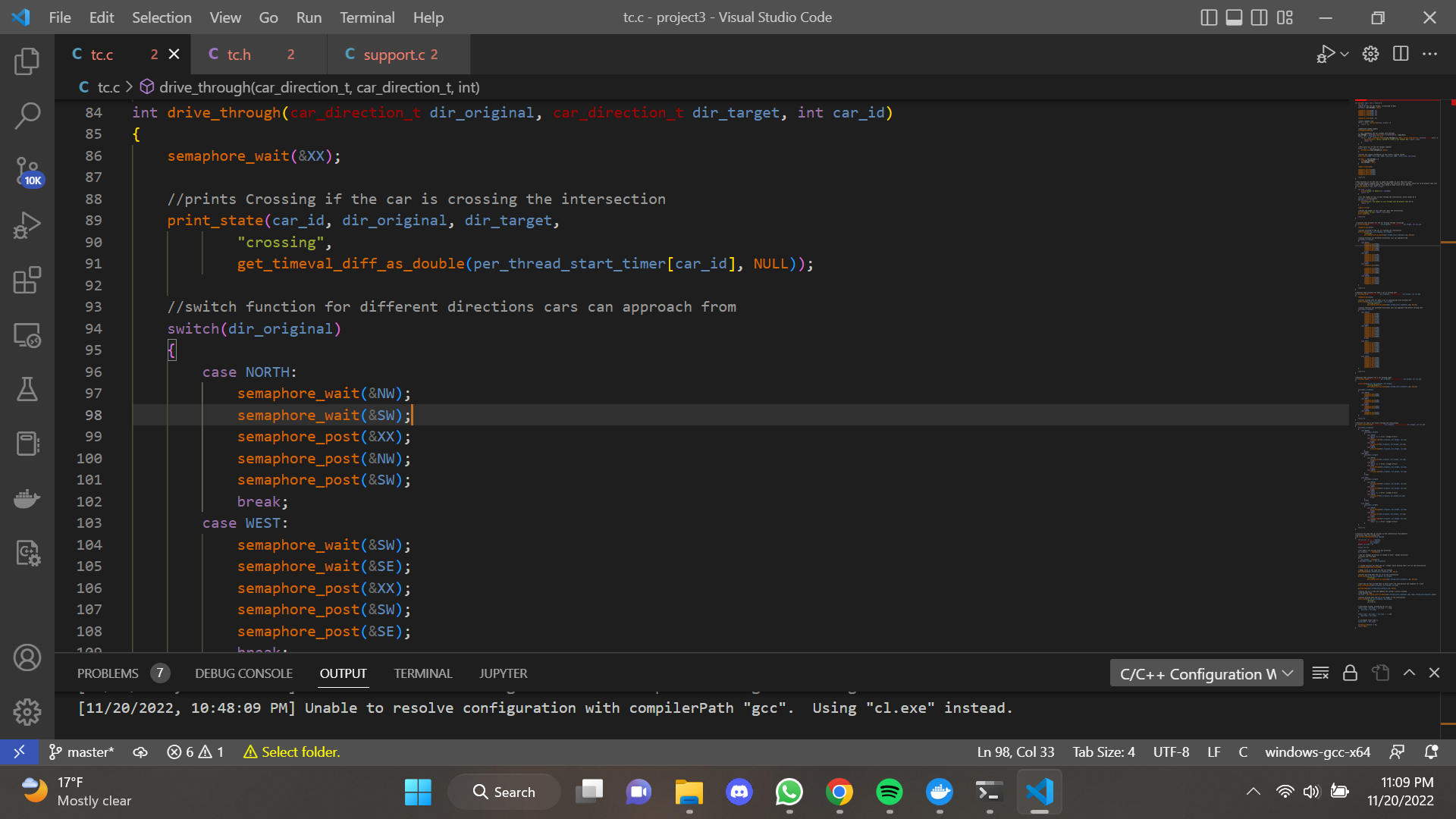
The layout of this project is a little different in terms of format, and the time at which each car arrives and exits. I set it to have each car randomly pick an action to take at the intersection to save time on having to create each car's individual actions. I did this so that when testing I can pick 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.

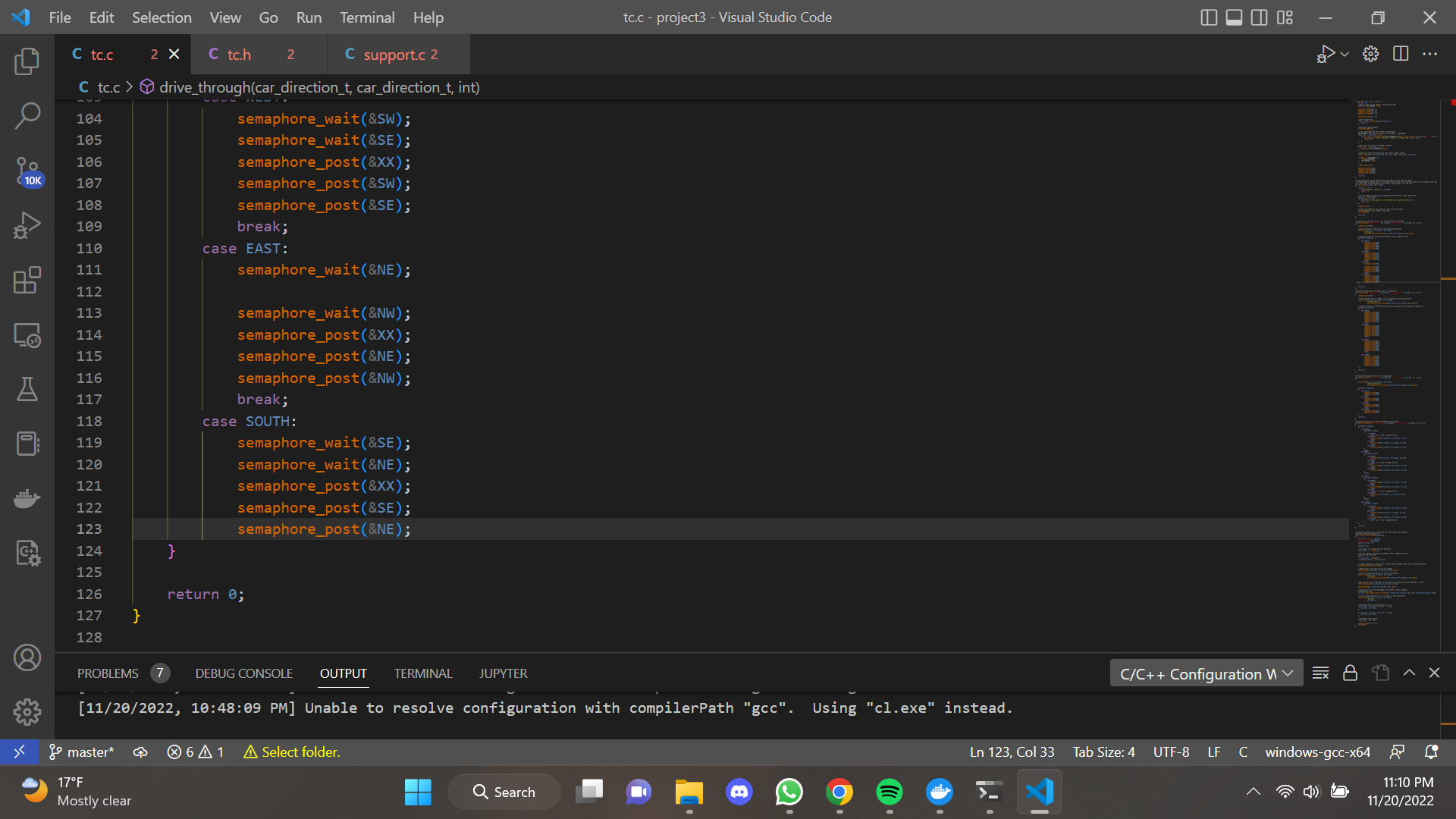
Screenshots

#1

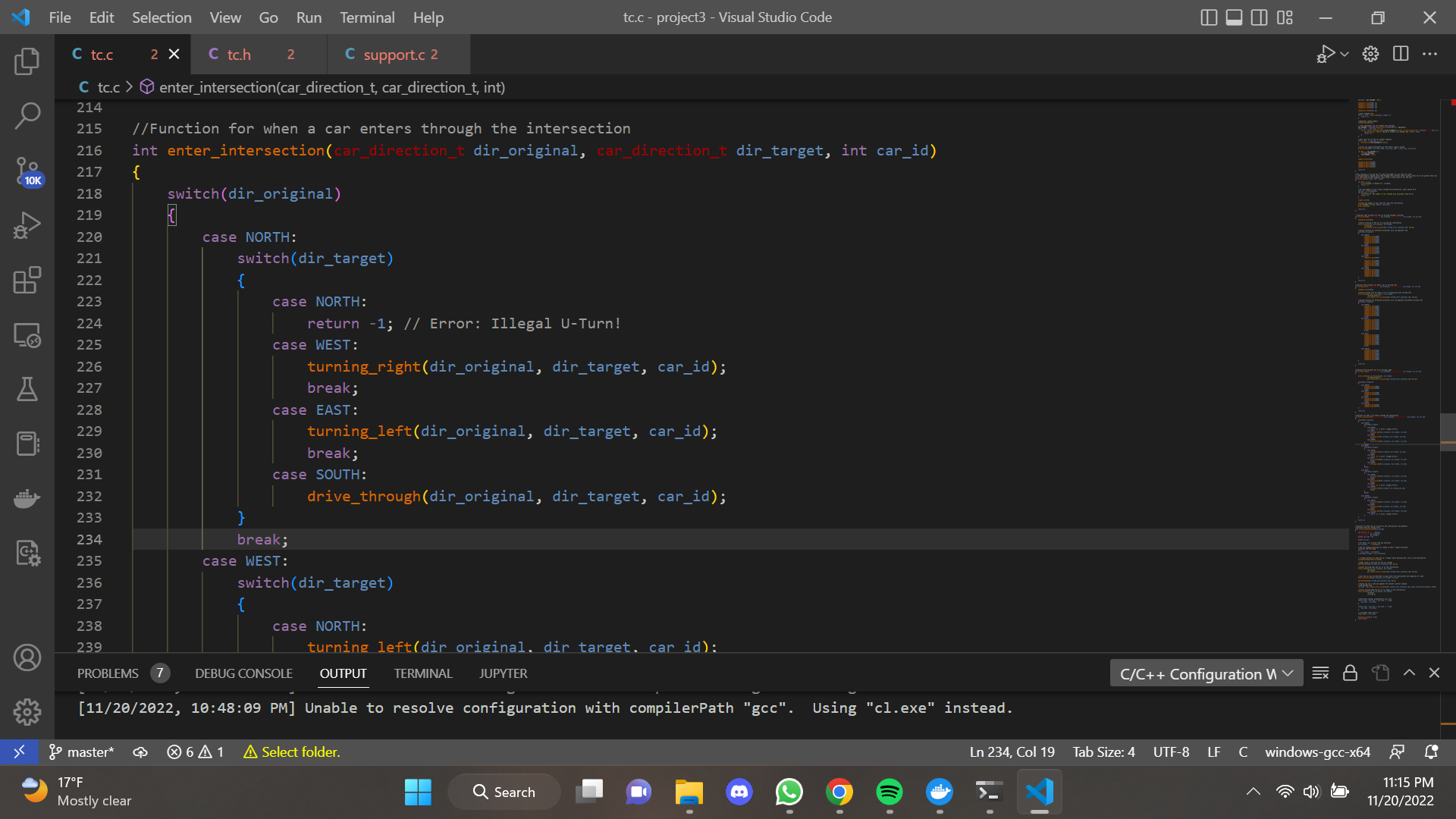


#2

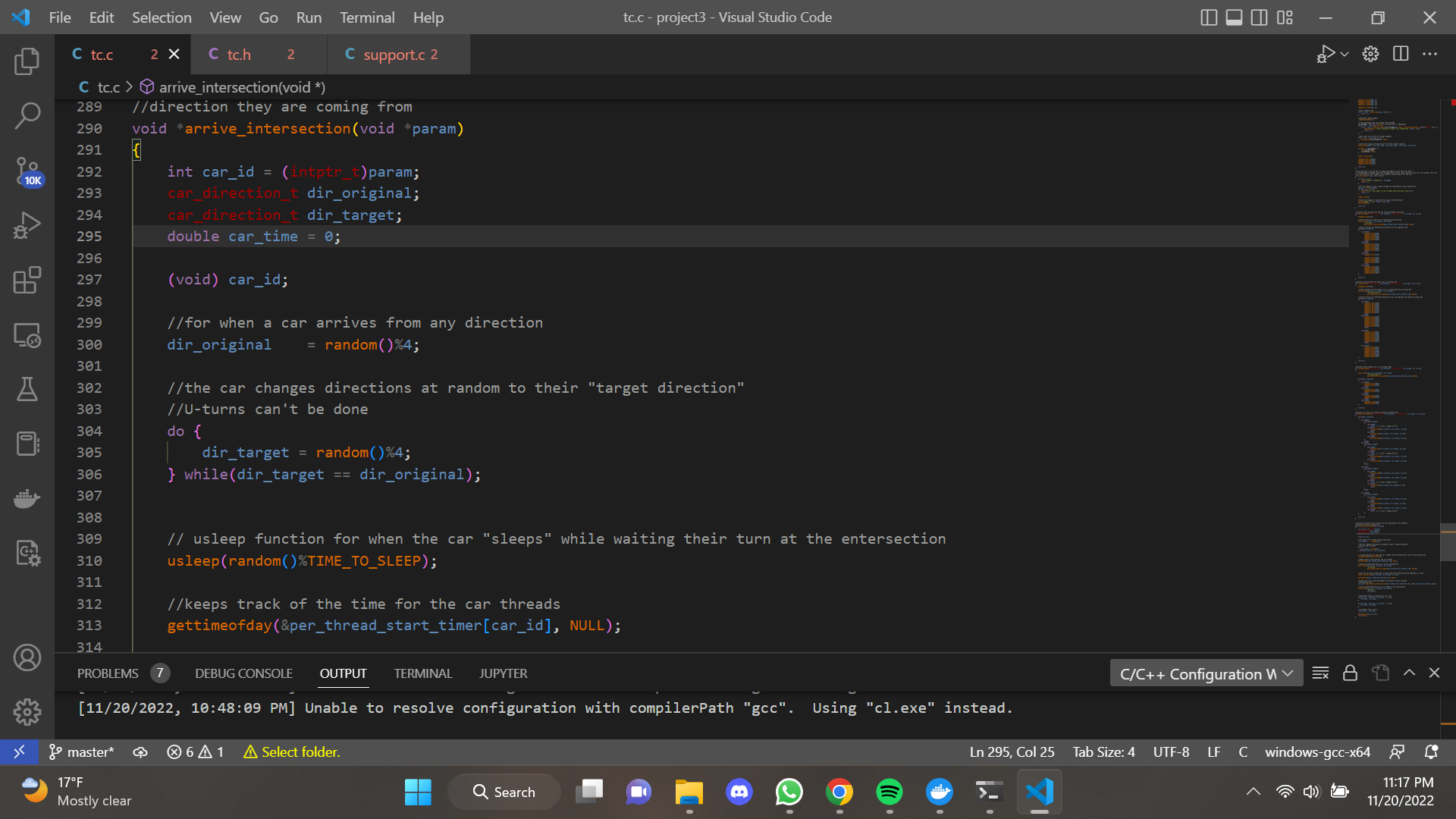


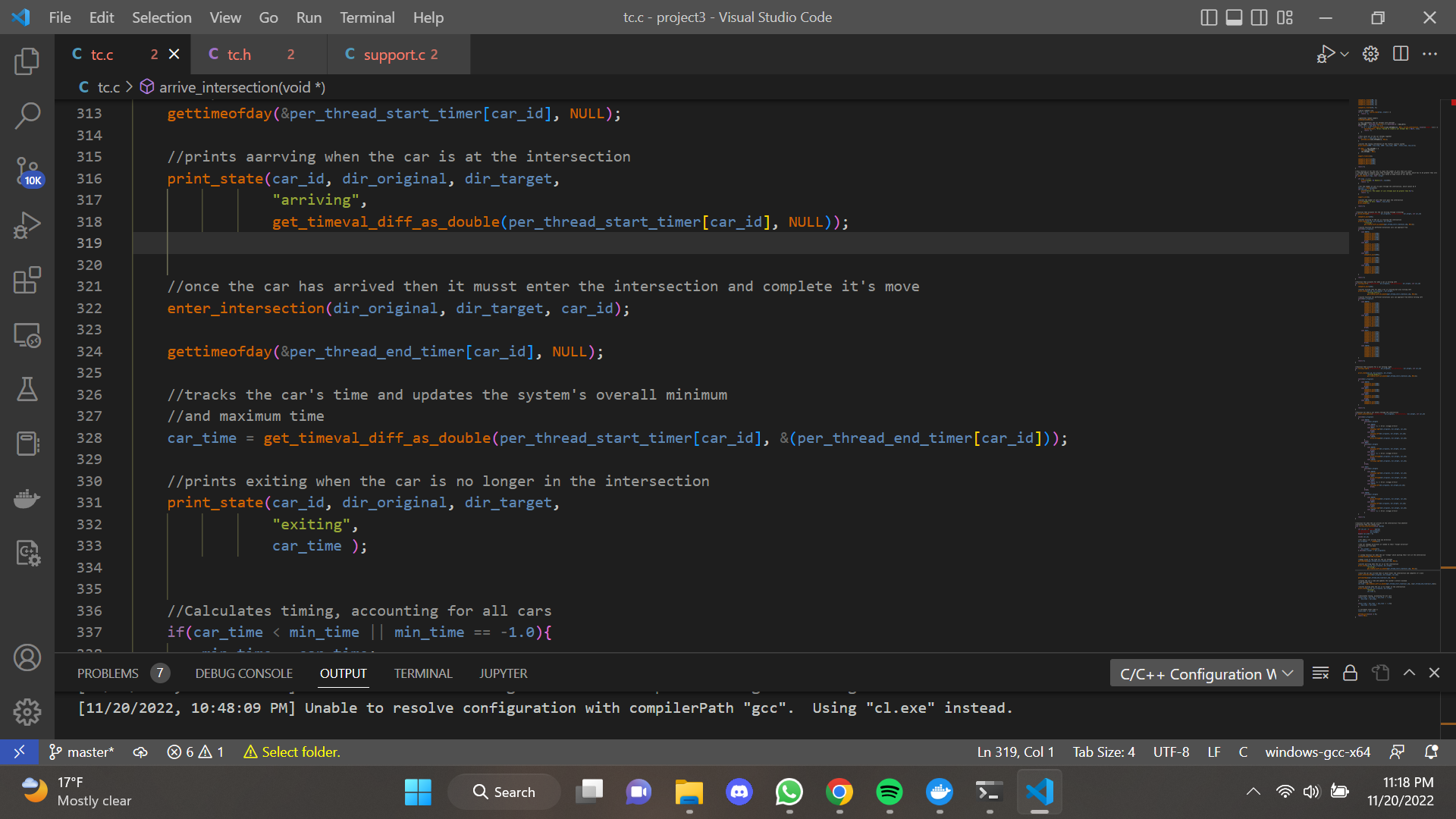


#3



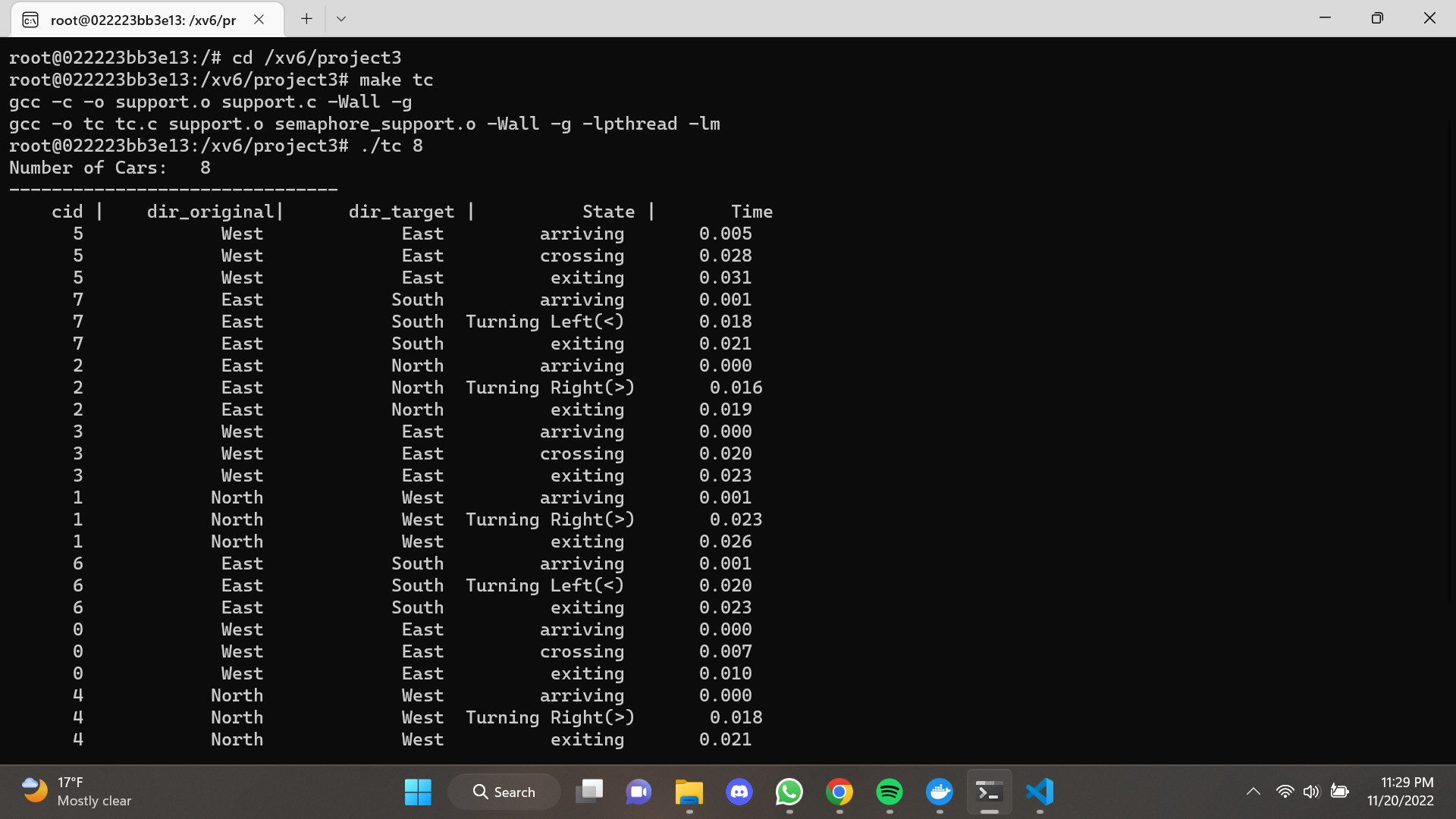
#4



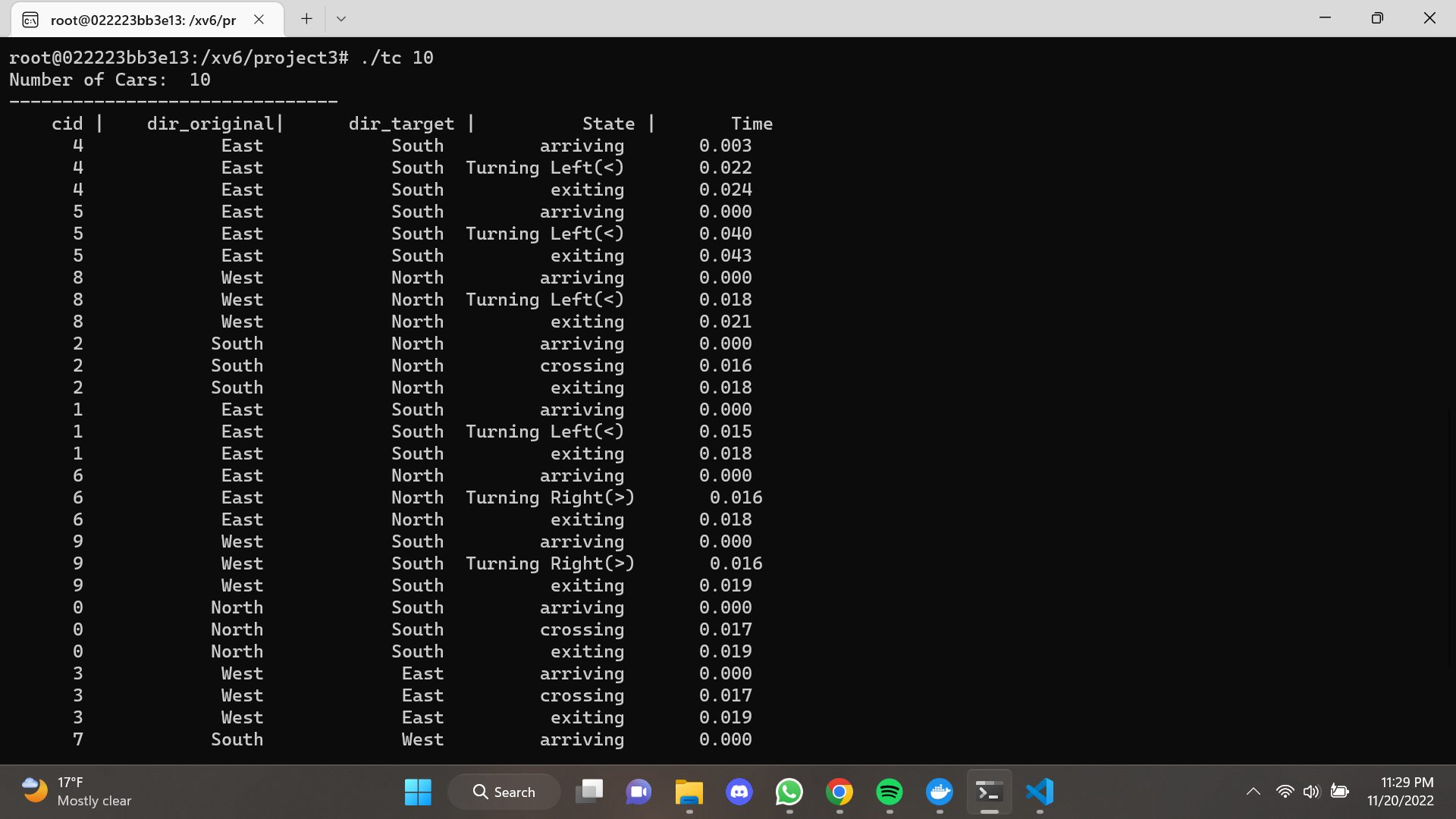


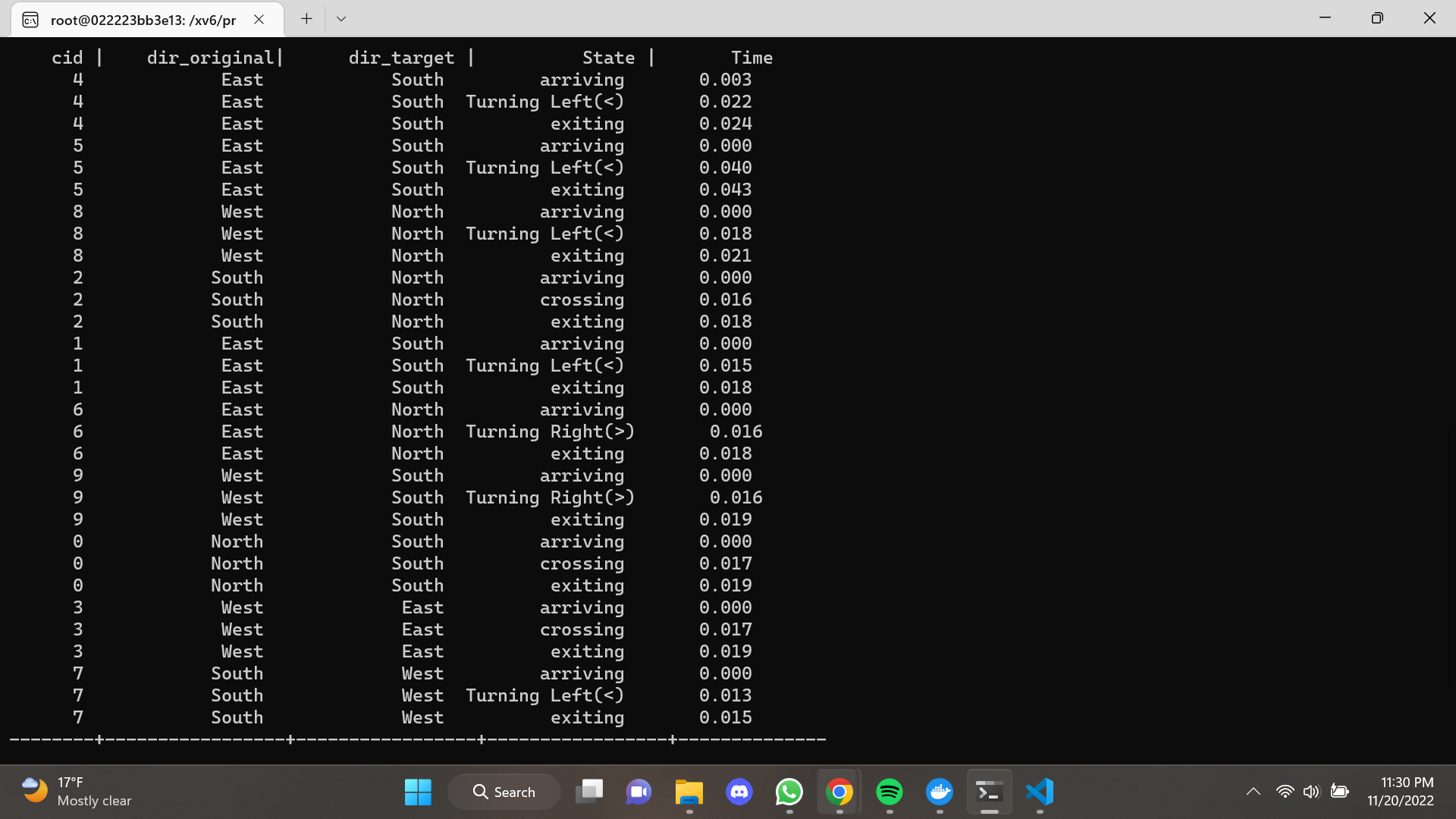
Outputs

With 8 cars



With 10 cars





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

tc.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

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

//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);