SIT315 M2.T3D: Traffic Control Simulator

Codey Funston | S222250824

Solution Design

Logic

For the sequential version of this task I used my parallel version which I did first as a logical guide because the producer/consumer design fits better with multiple threads conceptually. I created three functions, one for producing, consuming, and running the pattern. The produce and consume functions simply check container sizes before collecting or placing data, and stop running when all the data has been worked (detailed with "silly" records below).

I decided to go use *pthreads* for the parallel side of this task as I prefer how conditions are created and handled compared to with OpenMP. The program design is fairly simple with two worker functions for producer and consumer threads. Producer threads access the data file with mutual exclusion, taking the next line in the file/next data entry. They then check to see if it contains data or if it is the EOF. If it is the EOF, indicated by an empty string returned from another function, then the producer thread will send out "silly" records with negative values for traffic data so that consumer threads know to exit.

The program is blocking to threads for access to the queue data structure. Producers can only access it when the *buff_has_space* condition is broadcast by consumers, and consumers can only access it when the *buff_has_tasks* condition is broadcast by producers.

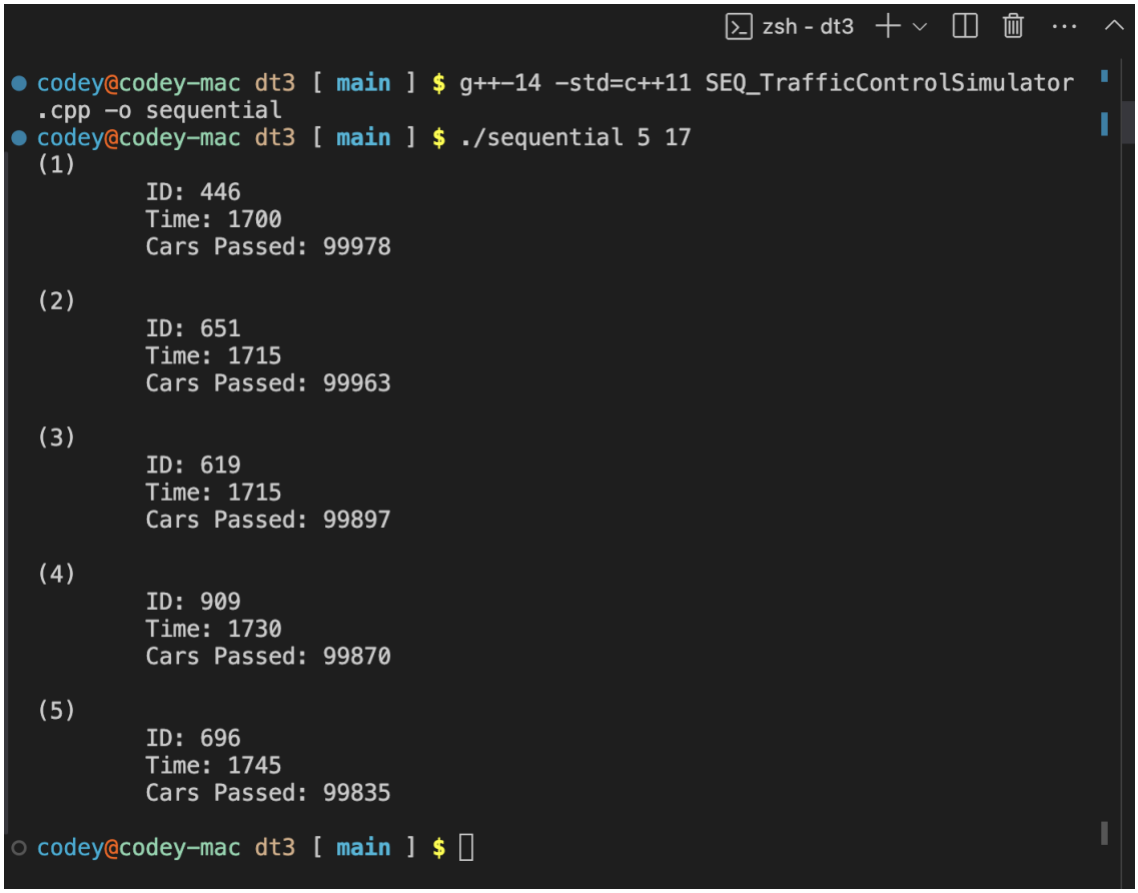
Consumer threads take tasks from the queue if there are any and place them into the task vector. Exiting when a silly record is received.

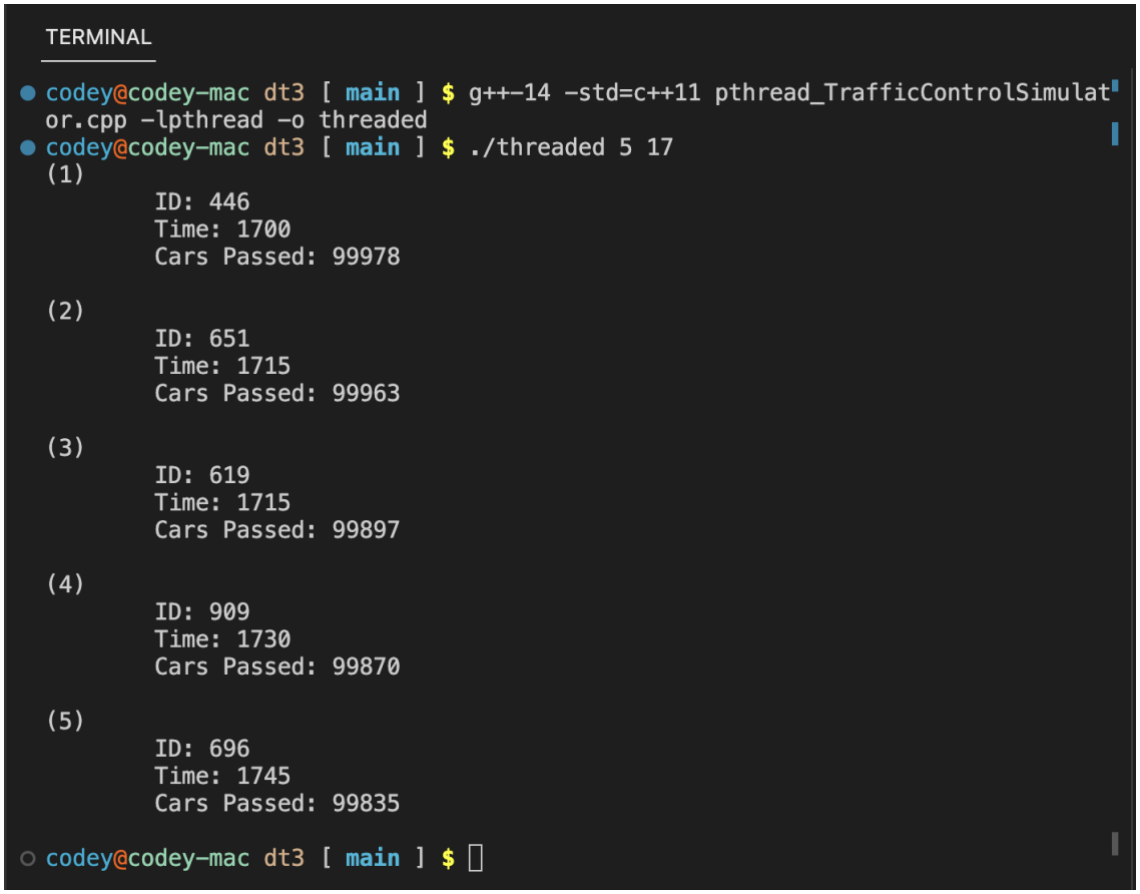
Data Structures

Data for each traffic light is stored as a struct with a field for each column: *time*, *id*, and *cars*. The buffer is represented as a queue since tasks are added and accepted in a first in first out (FIFO) way. The records are a vector as the positioning does not matter and sorted is done in a special way as highlighted in line 208-213, where I do an *nth_element* sort when information on the data is requested.

Below is the output of both the sequential (top) and parallel (bottom) program with the same executable arguments passed (N = 5, hour = 17/ 5 pm).

*Note: The sequential file is pasted at the bottom of the parallel file for OnTrack submission.


zsh - dt3
codey@codey-mac dt3 [main] $ g++-14 -std=c++11 SEQ_TrafficControlSimulator.cpp -o sequential
codey@codey-mac dt3 [main] $ ./sequential 5 17
(1)
ID: 446
Time: 1700
Cars Passed: 99978
(2)
ID: 651
Time: 1715
Cars Passed: 99963
(3)
ID: 619
Time: 1715
Cars Passed: 99897
(4)
ID: 909
Time: 1730
Cars Passed: 99870
(5)
ID: 696
Time: 1745
Cars Passed: 99835
codey@codey-mac dt3 [main] $


TERMINAL
codey@codey-mac dt3 [main] $ g++-14 -std=c++11 pthread_TrafficControlSimulator.cpp -lpthread -o threaded
codey@codey-mac dt3 [main] $ ./threaded 5 17
(1)
ID: 446
Time: 1700
Cars Passed: 99978
(2)
ID: 651
Time: 1715
Cars Passed: 99963
(3)
ID: 619
Time: 1715
Cars Passed: 99897
(4)
ID: 909
Time: 1730
Cars Passed: 99870
(5)
ID: 696
Time: 1745
Cars Passed: 99835
codey@codey-mac dt3 [main] $