Design Decisions & Assumptions made when developing my program

Structs

I created 4 struct datatypes to represent parts of the system, this was because it allows different data types to be combined into one encapsulated variable/type and also makes the upgradability of code easier if the scope of the project was to be expanded.

Vehicle (runOneSimulation.c)

The 'Vehicle' struct stores a pointer to the 'Lights' struct that the vehicle is at, this decision was made as when a vehicle passes the lights, a number of parameters belonging to he lights bust be updated and having this pointer allows for more generic functions to be written, adding readability, upgradability and maintenance. This struct also stores the iteration that the vehicle was generated, this is useful as once the vehicle passes the lights, a simple calculation can be made to find its waiting time as opposed to incrementing a counter for the vehicle at each loop.

Lights (runOneSimulation.c)

The Lights struct stores various variables to allow lights to be controlled, updated and their statistics to be stored in a contained variable. It stores the lightPeriod of a light, a timer integer which starts off as the same value as the lightPeriod and is decremented at each iteration when the light is green to act as a timer.

The statistics stored are the average waiting time, maximum waiting time, number of vehicles and the clearance time of the light. A status integer is also included to state whether the light is Red or Green

Node (runOneSimulation.c)

The Node struct stores a pointer to a vehicle that it represents as well as a pointer to the next node in the linked list. Due to recurring use of the Node struct, I implemented it using a typedef to help with the readability of the code. The use of a separate Vehicle struct keeps the linked list I used to hold vehicles at a light abstracted and ensures that it is its own system which can be used independently of changes to Vehicles in future systems.

Res (runOneSimulation.h)

The Res struct is used to create a neat and informative datatype for the return parameters of each run of the runOneSimulation function, and was therefore declared and implemented within an ifndef block in runOneSimulation.h. This allows it to be included in runSimulations.c where the averages of each return struct can be easily calculated. I Implemented this as a typedef named 'ReturnData' for ease of readability of the code

An assumption that was made is that in every simulation, the right light is initially green, one light must become green first, hence this decision. Furthermore I made the assumption that vehicles cannot turn back. In a future iteration of this project, types of vehicles could be added with varying probabilities of turning back before reaching a light.

Storing Vehicles at Lights

It became apparent that arrays would not be a suitable data structure to use. This is because arrays are not as expandable in C as they are in other languages and require a lot of overhead when adding elements to them (a new array with a greater length must be created with all the old contents copied over). I therefore decided to implement a linked list using my own Node structs, with one linked list per light. This meant, from the creation of a head node, as many or as few nodes could be added or removed with a simple traversing code and adjustment of a pointer. I decided to design my program with expandability in mind, and therefore created a separate Vehicle struct which can be changed to store more information about vehicles if necessary. I created functions to get the last element, which was useful for the function adding an element to the linked list. There was also a function to remove the first element of the linked list (excluding the head) which was useful when allowing a vehicle to pass through the lights.

Random Function

Deciding whether vehicles are to be added to a light is entirely dependent on a percentage. The built in rand() function could be used to generate a random number, and by dividing this by RAND_MAX and multiplying the result by 100, a random number between 1 and 100 could be generated. This however is not an accurate method due to its inner workings. I therefore created a function to use the GSL library which as included at the top of the file. This seeded the random generator using the time of day and also used the uniform distribution to gain a more 'random', random number.

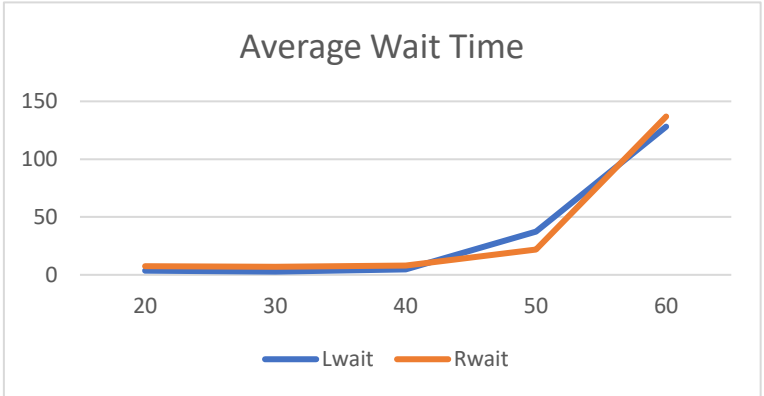
Memory Management

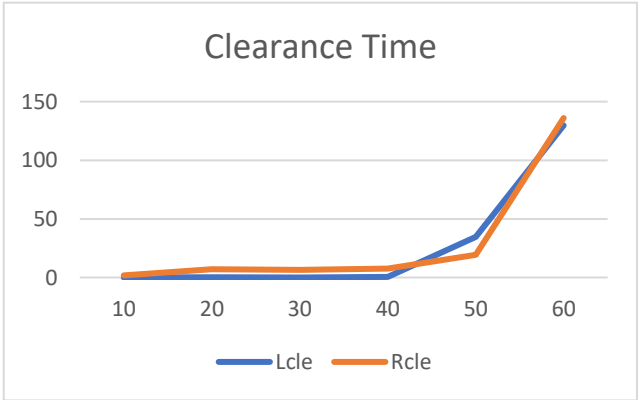
When managing memory in this program, I used Malloc for Node and Vehicle structs as they exist outside of the block they were created in, due to the fact that I used functions to generate variables. Every use of malloc in my code was directly followed by a check to ensure that the memory was has been allocated correctly. Furthermore, all the variables declared using malloc were freed before the end of the code. The head of the linked list was freed at the end of the code, and for each vehicle and node with them, they were freed when the vehicle passed the light and when the node was removed from the linked list, as they would no longer be needed. I used valgrind to test whether there were any memory leaks and am glad to say that there are none, below a the printout of a run with the input 50 5 50 5

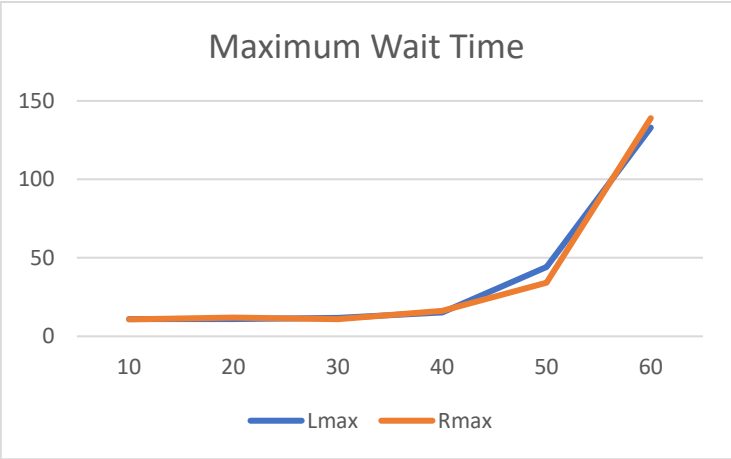
```
==8113== HEAP SUMMARY:
==8113==     in use at exit: 0 bytes in 0 blocks
==8113==   total heap usage: 252,415 allocs, 252,415 frees, 419,715,701 bytes allocated
==8113==
==8113== All heap blocks were freed -- no leaks are possible
==8113==
==8113== For counts of detected and suppressed errors, rerun with: -v
==8113== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

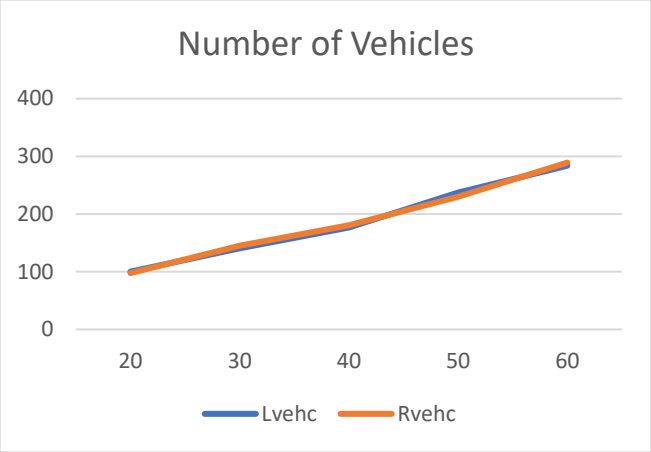
Experiment Performed using my simulation program

Experiment 1 – Effect of increasing the Arrival Rates with constant Light Period



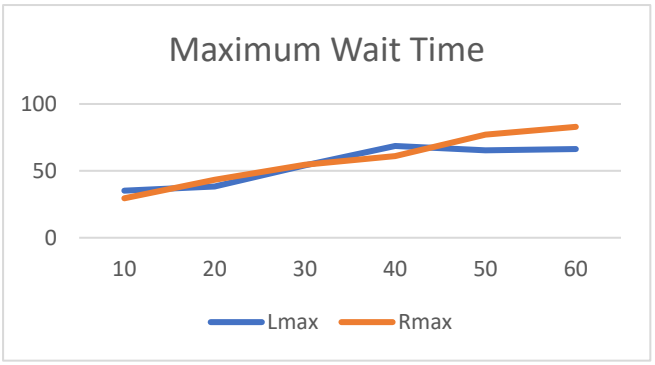


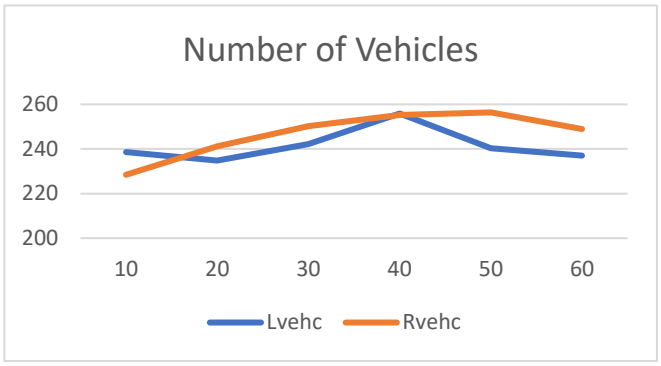


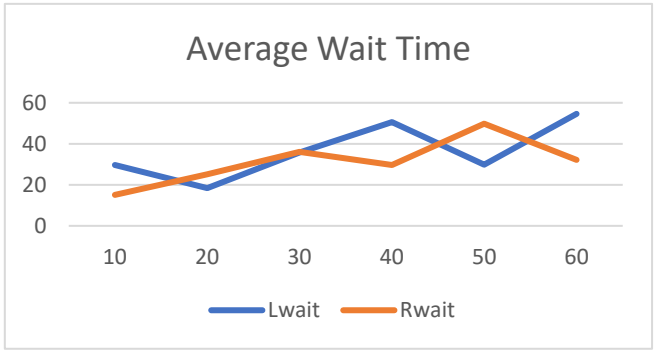


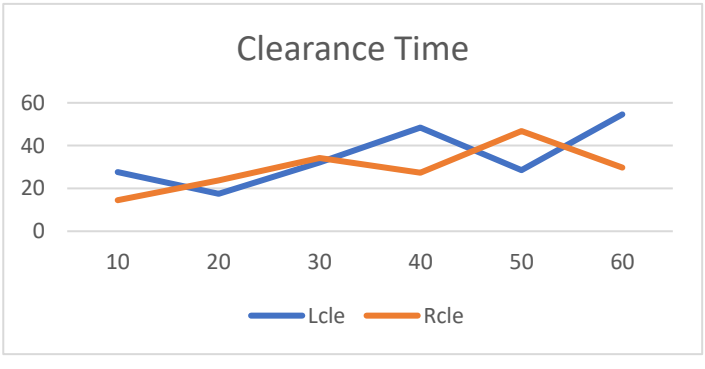
This test was run over 6 tests, the light period for both lights was kept at 10 for each of the tests, however the arrival rate was increased by 10 each test as can be seen on the x-axis of all 4 graphs above. With the increase of the arrival rate, it can be seen that there is a constant rate of increase with the number of vehicles that arrive at each light, this is expected as with a higher probability of vehicles arriving, there would be more cars arriving. What is interesting to see is an exponential increase in the Average and Maximum wait time, as well as the clearance time. This is also expected and would be due to the fact that the light periods were remained the same, therefore cars were unable to move through the lights as quickly.

Experiment 2 – Effect of increasing the Light Periods with constant Arrival Rates









This test was run over 6 tests, the arrival rate for both lights was kept at 50 for each of the tests, however the light period was increased by 10 each test as can be seen on the x-axis of all 4 graphs above. With the increase of the light period, it can be seen that there is a constant rate of increase of the maximum wait time at each light, this shows that for lower wait times, a lower light period may be preferable. Another observation is there is no correlation when it comes to the number of vehicles arriving at each light, This is expected as the number of vehicles is not affected by the light period in any way. On the contrary, the

average wait time and clearance time fluctuate with an increase and also follow a similar pattern, most likely due to the change in the number of vehicles and that random instance.

ReadMe – to run this code using the university VM

In this Readme Section, information regarding the running of my program will be explained.

Viewing Files

The server contains the files ‘runOneSimulation.c’, ‘runOneSimulation.h’, ‘runSimulations.c’, ‘runSimulations.h’ and ‘compileSim’

These are held in the ‘/home/links/na510/ecm2433ca1’ directory of the virtual machine and are accessed as follows:

1) Login to the na510 user system on the virtual machine (I used ugcs1 and an ssh connection)
2) Navigate to the ecm2433ca1 folder using the command ‘cd ecm2433ca1’

```
[na510@emps-ugcs1 ~]$ ls
ecm2433ca1  project  test
[na510@emps-ugcs1 ~]$ cd ecm2433ca1
[na510@emps-ugcs1 ecm2433ca1]$ ls
compileSim  runOneSimulation.c  runOneSimulation.h  runSimulations.c  runSimulations.h
```

Figure 1 - Reaching the directory with my code (Please note the correspnding .o files may be present when viewing files, these may be ignored as they are replaced by the compileSim script upon compilation)

Running Code

The code can be run using the command line and providing the inputs upon running of the compiled executable

1) Once you have accessed the’ecm2433ca1’ directory, the code can be compiled using the command ‘bash compileSim’
2) This will produce .o files for each of the c files in the folder, as well as an executable named ‘runSimulations’ shown below

```
[na510@emps-ugcs1 ecm2433ca1]$ ls
compileSim  runOneSimulation.c  runOneSimulation.h  runSimulations.c  runSimulations.h
[na510@emps-ugcs1 ecm2433ca1]$ bash compileSim
[na510@emps-ugcs1 ecm2433ca1]$ ls
compileSim  runOneSimulation.c  runOneSimulation.h  runOneSimulation.o  runSimulations
 runSimulations.c  runSimulations.h  runSimulations.o
[na510@emps-ugcs1 ecm2433ca1]$
```

Figure 2 - '.o' and 'runSimulations' files created when using the 'compileSim' script

3) The executable can now be run and parameters passed in using the format below WHERE the arrival rates are an integer percentage between 0 and 100, and the light period is an integer number of iterations a light stays green for.

“./runSimulations arrivalRateLHS lightPeriodLHS arrivalRateLHS lightPeriodLHS ”

Example Output for runs using this code

Once the folder containing the code has been sourced and the code has been compiled using the compileSim script, code can be run using the command shown above, an example run of the code using the inputs as shown in the code printout on the right.

```
[na510@emps-ugcs1 ecm2433ca1]$ ./runSimulations 75 5 80 4
Parameter Values:
    from left:
        traffic arrival rate: 75
        traffic light period 5
    from right:
        traffic arrival rate: 80
        traffic light period 4
Results (averaged over 100 runs):
    from left:
        number of vehicles: 308.519806
        average waiting time: 180.132996
        maximum waiting time: 182.362701
        clearance time: 178.306976
    from right:
        number of vehicles: 330.720154
        average waiting time: 405.616333
        maximum waiting time: 407.036743
        clearance time: 404.898560
[na510@emps-ugcs1 ecm2433ca1]$
```