Programming Project 04

Threads & Synchronization

CS 441/541 – Fall 2017

Project Available		Oct. 12
Component	**Points**	**Due Date (at 11:59 pm)**
Notification of Group Participation		Oct. 14
Project 3.1	30	Oct. 19
Project 3.2	60	Nov. 2
Oral Presentation	10	(Week of Nov. 6)
Project Total	100	

For this project you have the option to work in **groups of only two (2) people**. You must sign up for a group on D2L by the date indicated above if you intend to work in a group for this project. Those that work in groups are required to complete a group evaluation on D2L. Not completing the group evaluation for each part of the project will decrease your individual score by 3 points per subproject.

Packaging & handing in your projects

You will turn in a single compressed archive of your completed project directory to the D2L Dropbox **for each portion of the project**. The compressed archive should be downloaded from the BitBucket website, which is a `.zip` archive. If, for some reason, you cannot download the source from the BitBucket website then turn in a manually compressed archive of the project as either a bzip/gzip'ed tar file or zip file. Only one group member needs to turn in the project for all group members to receive credit.

Overview

In this project you will be working though synchronization problems to gain experience with threaded programming using the Pthreads library and using synchronization primitives for concurrency control. In the first part, you will be reasoning about and implementing what are pseudo code *solutions*. In the second part, you will be solving a synchronization problem both on paper and in code. For each part you will write up why you believe your program solves the problem while preventing deadlock and starvation.

Program Specifications

Your project must be written in C (not C++) using the Pthreads library and the provided semaphore portability library. No credit will be given for projects written in any programming language other than C or a threading library other than Pthreads. Your programs must be able to be compiled using a Makefile by typing the command `make` without any additional arguments in the project directory.

All of your programs must incorporate the following design points.

- Your program **must never busy wait** for an event to occur. If you need to cause a thread to wait then it should do so using semaphores. **Do not assume** that the semaphore wait provides a proper queue for the waiting threads (if you need such a concept).
- Make sure to seed the random number generator before using it. Seed the random number generator before starting threads with the following command: `srandom(time(NULL))`
- You will need to put bounds on the random number generator by using the modulus operator on the output: `i = random()%LIMIT;`
- To sleep for less than 1 full second you must use the usleep() command (instead of sleep()) to sleep for some number of microseconds.
- When printing output to stdout you may notice that the output from multiple threads interleave themselves. To *fix* this problem you might consider creating a binary semaphore to protect calls to the `printf` function so only one thread is printing to the console at a time.
- You must print out the parsed command line parameters before starting execution. If an optional parameter is not supplied, then you must display the default value for that parameter.
- If the user provides an incorrect set of command line arguments to your program you must immediately display an error message and a message describing the correct use of your program. After displaying those messages your program will immediately exit.

Oral Presentation - Project 4.2

Following the conclusion of Project 4.2, your **entire group** must setup a time to meet with the professor outside of class to discuss your solution to the problem. This oral presentation will ask at least the following from your group:

- Present your solution to the problem without the code (whiteboard will be available, you can bring notes and other non-source code materials to aid the presentation).
- Discuss the methodology that you used to test your software.
- Demonstrate the code working on the course server. The code you turned in to the dropbox will be loaded onto the server for you before the demo.
- Source code debriefing - Discuss the code, any key data structures, and any problem areas.
- Discuss the individual contributions of each member of the group.

Project 3.1 – The Bounded-Buffer Problem – (30 Points)

In this part of the project you will be implementing a solution to the bounded-buffer problem. The programming project is described as Programming Project 6.40 in the text with some modifications as described below.

- Producer and consumer threads should **not** sleep for an unbounded random amount of time. Instead they should sleep for a random amount of **microseconds** between 0 and 1 second.
- The buffer should be of a **dynamically allocated** size, *not* a fixed size as indicated by the text. The user may supply a command line option specifying the buffer size which can be any integer value greater than zero (0). This should be an *optional* fourth argument. If it is not provided then the buffer size should default to ten (10) items.
- You will need to print the buffer upon initialization, and after a element has been inserted or removed. You will also need to add a marker indicating where the next inserted ([^] below), and next removed ([v] below) markers are positioned.
- Producers will produce a random integer value between 0 and 9 to place in the buffer.
- You will need to track the number of elements produced and consumed. These totals will be displayed just before the program terminates (after the time to live expires). When a thread updates the buffer it will print its thread identifier, the total corresponding to their task, and the value either produced or consumed.

Examples

Example specifying buffer size of 4 elements:

```
shell$ ./bounded-buffer 2 2 1 4
Buffer Size               :   4
Time To Live (seconds)    :   2
Number of Producer threads:   2
Number of Consumer threads:   1
-------------------------------
Initial Buffer:                          [  -1^v  -1  -1  -1]
Producer  1: Total    1, Value   6       [   6v  -1^  -1  -1]
Consumer  0: Total    1, Value   6       [  -1  -1^v  -1  -1]
Producer  0: Total    2, Value   4       [  -1   4v  -1^  -1]
Consumer  0: Total    2, Value   4       [  -1  -1  -1^v  -1]
Producer  1: Total    3, Value   9       [  -1  -1   9v  -1^]
Consumer  0: Total    3, Value   9       [  -1  -1  -1  -1^v]
Producer  0: Total    4, Value   5       [  -1^  -1  -1   5v]
Consumer  0: Total    4, Value   5       [  -1^v  -1  -1  -1]
Producer  0: Total    5, Value   1       [   1v  -1^  -1  -1]
Producer  1: Total    6, Value   6       [   1v   6  -1^  -1]
Consumer  0: Total    5, Value   1       [  -1   6v  -1^  -1]
Producer  0: Total    7, Value   4       [  -1   6v   4  -1^]
-----------+---------
Produced   |    7
Consumed   |    5
-----------+---------
```

Example using the default buffer size of 10 elements.

```
shell$ ./bounded-buffer 3 3 2
Buffer Size                :  10
Time To Live (seconds)     :   3
Number of Producer threads:   3
Number of Consumer threads:   2
------------------------------
Initial Buffer:                          [  -1^v  -1  -1  -1  -1  -1  -1  -1  -1  -1]
Producer  1: Total    1, Value   9       [   9v  -1^  -1  -1  -1  -1  -1  -1  -1  -1]
Consumer  1: Total    1, Value   9       [  -1  -1^v  -1  -1  -1  -1  -1  -1  -1  -1]
Producer  2: Total    2, Value   6       [  -1   6v  -1^  -1  -1  -1  -1  -1  -1  -1]
Consumer  1: Total    2, Value   6       [  -1  -1  -1^v  -1  -1  -1  -1  -1  -1  -1]
Producer  0: Total    3, Value   6       [  -1  -1   6v  -1^  -1  -1  -1  -1  -1  -1]
Consumer  0: Total    3, Value   6       [  -1  -1  -1  -1^v  -1  -1  -1  -1  -1  -1]
Producer  2: Total    4, Value   2       [  -1  -1  -1   2v  -1^  -1  -1  -1  -1  -1]
Consumer  0: Total    4, Value   2       [  -1  -1  -1  -1  -1^v  -1  -1  -1  -1  -1]
Producer  1: Total    5, Value   3       [  -1  -1  -1  -1   3v  -1^  -1  -1  -1  -1]
Consumer  0: Total    5, Value   3       [  -1  -1  -1  -1  -1  -1^v  -1  -1  -1  -1]
Producer  0: Total    6, Value   7       [  -1  -1  -1  -1  -1   7v  -1^  -1  -1  -1]
Producer  1: Total    7, Value   9       [  -1  -1  -1  -1  -1   7v   9  -1^  -1  -1]
Consumer  1: Total    6, Value   7       [  -1  -1  -1  -1  -1  -1   9v  -1^  -1  -1]
Producer  2: Total    8, Value   6       [  -1  -1  -1  -1  -1  -1   9v   6  -1^  -1]
Producer  0: Total    9, Value   1       [  -1  -1  -1  -1  -1  -1   9v   6   1  -1^]
Consumer  1: Total    7, Value   9       [  -1  -1  -1  -1  -1  -1  -1   6v   1  -1^]
Consumer  1: Total    8, Value   6       [  -1  -1  -1  -1  -1  -1  -1  -1   1v  -1^]
Producer  1: Total   10, Value   1       [  -1^  -1  -1  -1  -1  -1  -1  -1   1v   1]
Consumer  0: Total    9, Value   1       [  -1^  -1  -1  -1  -1  -1  -1  -1  -1   1v]
Consumer  0: Total   10, Value   1       [  -1^v  -1  -1  -1  -1  -1  -1  -1  -1  -1]
Producer  1: Total   11, Value   8       [   8v  -1^  -1  -1  -1  -1  -1  -1  -1  -1]
Producer  0: Total   12, Value   4       [   8v   4  -1^  -1  -1  -1  -1  -1  -1  -1]
Producer  0: Total   13, Value   3       [   8v   4   3  -1^  -1  -1  -1  -1  -1  -1]
Consumer  0: Total   11, Value   8       [  -1   4v   3  -1^  -1  -1  -1  -1  -1  -1]
Consumer  1: Total   12, Value   4       [  -1  -1   3v  -1^  -1  -1  -1  -1  -1  -1]
Producer  2: Total   14, Value   5       [  -1  -1   3v   5  -1^  -1  -1  -1  -1  -1]
Consumer  1: Total   13, Value   3       [  -1  -1  -1   5v  -1^  -1  -1  -1  -1  -1]
Producer  1: Total   15, Value   5       [  -1  -1  -1   5v   5  -1^  -1  -1  -1  -1]
Consumer  0: Total   14, Value   5       [  -1  -1  -1  -1   5v  -1^  -1  -1  -1  -1]
Consumer  1: Total   15, Value   5       [  -1  -1  -1  -1  -1  -1^v  -1  -1  -1  -1]
Producer  0: Total   16, Value   3       [  -1  -1  -1  -1  -1   3v  -1^  -1  -1  -1]
Consumer  0: Total   16, Value   3       [  -1  -1  -1  -1  -1  -1  -1^v  -1  -1  -1]
Producer  2: Total   17, Value   4       [  -1  -1  -1  -1  -1  -1   4v  -1^  -1  -1]
Producer  0: Total   18, Value   4       [  -1  -1  -1  -1  -1  -1   4v   4  -1^  -1]
-----------+---------
Produced   |   18
Consumed   |   16
-----------+---------
```

Testing

Testing is an important part of the software development life cycle. You must describe in your documentation how you tested your project to ensure it met the requirements of the problem.

Note, that for synchronization problem just running the program one, twice, . . . , 100, . . . , 10,000 times **will not suffice** to highlight all of the possible ways that the threads can be interwoven during execution. You will want to think about how to force more contention in the system to become more confident in your solution. Testing and debugging tend to be one of the greatest challenges to writing concurrent code because of this reality.

Your documentation must discuss the testing methodology that you used used to validate that your solution was valid in design and in the implementation of that design.

Documentation

Edit the `README.md` file in this subdirectory to contain at least the following pieces of information (each identified in the documentation as separate sections):

- Author(s), date
- Brief summary of the software
- How to build the software (e.g., `make` with output)
- How to use the software (summarize all command line options)
- Discussion of how the software was tested
- Known bugs and problem areas

Deliverables:

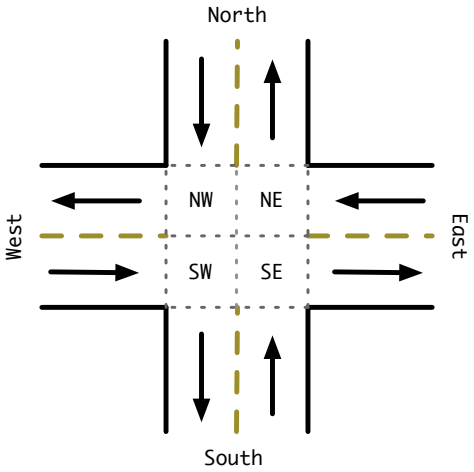
Your project submission is required be written in a **consistent style** according to the **Style Requirements** handout. Be sure to carefully review the entire specification in detail for the list of requirements.

You will turn in a single compressed archive of the entire project directory to the D2L Dropbox **for each portion of the project**.

If you are working in a group, do not forget to fill out the group evaluation quiz on D2L for this part of the project by the deadline for this part of the project.

Project 3.2 – Traffic Management Problem – (60 Points)

Traffic through an intersection in the town of Smallville, WI has increased over the past few years. Until now the intersection has been a four-way stop but now the gridlock has forced the residents of Smallville to admit that they need a more efficient way for traffic to pass through this section of town. Your job is to design and implement a solution using **only semaphores** as your synchronization primitive.



The intersection could be modeled as shown above, dividing it into quarters and identifying each quarter with which lane enters the intersection through that portion. (Just to clarify: Smallville is in the US, so we're driving on the right side of the road.) Turns are represented by a progression through one, two, or three portions of the intersection (for simplicity assume that U-turns do not occur in the intersection). So if a car approaches from the North, depending on where it is going, it proceeds through the intersection as follows:

- **West**: NW
- **South**: NW – SW
- **East**: NW – SW – SE

The following are the requirements for your solution:

- No two cars can be in the same portion of an intersection at the same time.
  (In Smallville they call this an accident).
- Residents of Smallville do not pass each other going the same way.
  If two cars both approach from the same direction and head in the same direction, the first car to reach the intersection must be the first to reach the destination. For example, consider a Car A is traveling from North to East and Car B is behind them traveling from North to West. Car A must go through the intersection before Car B is allowed to enter the intersection.
- Your solution **must improve traffic flow** without allowing traffic from any direction to starve traffic from any other direction. So locking an entire intersection for a single car to pass is **not an acceptable solution**. For example, a car approaching from the North and another approaching from the South, both going straight, should be allowed to cross the intersection at the same time (e.g., concurrently). Further, a car approaching from the South turning left, and another approaching from the West turning right should be allowed to cross the intersection at the same time (e.g., concurrently).

- Folks in Smallville love driving around their town, so they pass through this intersection many times in a day. After a car leaves the simulation it waits for a random amount of time before entering the simulation again with a new approaching direction and destination direction.
- Your program will take two *required* arguments.
  1. Time to run in seconds. **Required argument**.
     This is the length of time that your application will run before terminating itself.
     The user must specify a positive integer greater than 0.
  2. Number of cars in the simulation. **Required argument**.
     The user must specify a positive integer greater than 0.
- The time to cross one portion of the intersection is defined in `support.h` as `TIME_TO_CROSS` in microseconds.
- Each car should print a message as it changes state (use the `print_state` function). Below are the state transitions:
  - **Waiting in line** : When a car starts waiting to enter the intersection
  - **Next in line** : When a car is the first in line from their approach direction.
  - **Go Straight**, **Turn Left**, **Turn Right** : When the car enters the intersection.
  - **Leave** : When the car leaves the intersection
- You will also need to track the total thread execution time and the per thread minimum, maximum, and average time all in milliseconds. Before exiting, your program should print these four time values. You may want to review the man page for `getimeofday` and the provided support library.

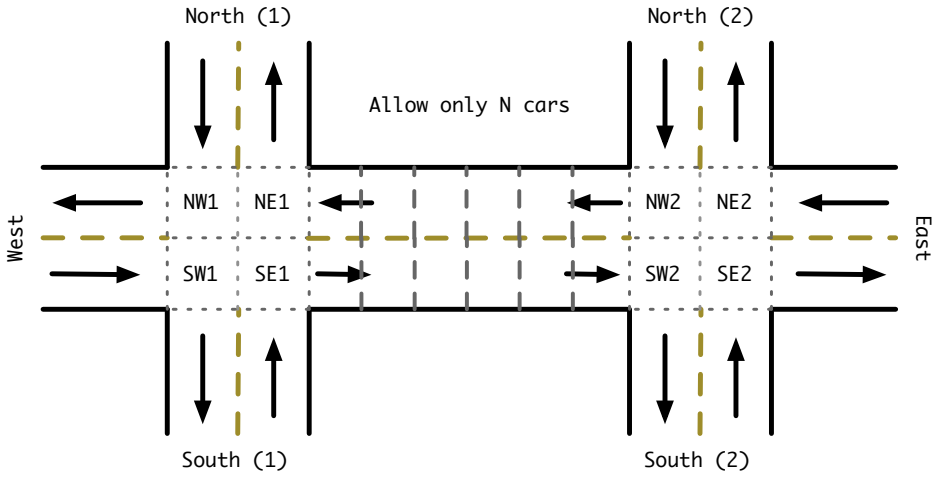
The template provides some code that you should use to get started. Be sure to review **all** of the code provided before getting started.

Write all of your code and documentation for this part of the project in the `part2` directory in the repository.

Before You Begin Coding

Before you begin coding, in a ***special section*** within your documentation you must answer the following questions considering just a single intersection:

1. Consider first, a single intersection:
   (a) Assume that the residents of Smallville are exceptional and follow the old (and widely ignored) convention that whoever arrives at the intersection first proceeds first. Using the language of synchronization primitives describe the way this intersection is controlled. In what ways is this method suboptimal?
   (b) Now, assume that the residents of Smallville are like most people and do not follow the convention described above. In what one instance can this four-way-stop intersection produce a deadlock? (It will be helpful to think of this in terms of the model we are using instead of trying to visualize an actual intersection).
2. Consider next, a two intersection situation (seen below):



   (a) Assume that the bridge between the two intersections had space for at most N cars in any one direction. In what instances can this produce deadlock? Provide at least 4 specific, distinct examples that use a path crossing the bridge. Include in those examples what you would consider the worst case scenario.

After You Finish Coding

Once you have completed your solution, in a ***special section*** within your documentation you must answer the following questions, **separately**:

1. Describe your solution to the problem (in words, not in code).
2. Describe how your solution meets the goals of this part of the project, namely (each in a separate section):
   - How does your solution avoid deadlock?
   - How does your solution prevent accidents?
   - How does your solution improve traffic flow?
   - How does your solution preserve the order of cars approaching the intersection?
   - How is your solution "fair"?
   - How does your solution prevent "starvation"?

Example 1:

```
shell$ ./stoplight 1 2
--------------------
Time to live  :   1
Number of Cars:   2
--------------------
------------------------------
 Car ID | FROM to DEST  | Loc. |  Time (msec) | State
--------+---------------+------+--------------+-------------------------------
      0 |   N. to S.    |  1-- |        0.287 | Waiting in line
      0 |   N. to S.    |  1-- |        0.348 | Next in line
      0 |   N. to S.    |  1-- |        0.359 | Go Straight
      0 |   N. to S.    |  1-- |       41.069 | Leave
      1 |   N. to E.    |  1-- |        0.060 | Waiting in line
      1 |   N. to E.    |  1-- |        0.100 | Next in line
      1 |   N. to E.    |  1-- |        0.115 | Turn Left
      1 |   N. to E.    |  1-- |       60.667 | Leave
      1 |   N. to S.    |  1-- |        0.006 | Waiting in line
      1 |   N. to S.    |  1-- |        0.034 | Next in line
      1 |   N. to S.    |  1-- |        0.047 | Go Straight
      0 |   E. to N.    |  1-- |        0.007 | Waiting in line
      0 |   E. to N.    |  1-- |        0.035 | Next in line
      0 |   E. to N.    |  1-- |        0.047 | Turn Right
      1 |   N. to S.    |  1-- |       40.652 | Leave
      0 |   E. to N.    |  1-- |       20.655 | Leave
      0 |   E. to S.    |  1-- |        0.007 | Waiting in line
      0 |   E. to S.    |  1-- |        0.033 | Next in line
      0 |   E. to S.    |  1-- |        0.045 | Turn Left
      0 |   E. to S.    |  1-- |       60.585 | Leave
      0 |   E. to N.    |  1-- |        0.008 | Waiting in line
      0 |   E. to N.    |  1-- |        0.053 | Next in line
      0 |   E. to N.    |  1-- |        0.071 | Turn Right
      1 |   W. to E.    |  1-- |        0.009 | Waiting in line
      1 |   W. to E.    |  1-- |        0.050 | Next in line
      1 |   W. to E.    |  1-- |        0.062 | Go Straight
      0 |   E. to N.    |  1-- |       20.234 | Leave
      1 |   W. to E.    |  1-- |       40.827 | Leave
--------+---------------+------+--------------+-------------------------------
Min.  Time :      20.230 msec
Avg.  Time :      40.664 msec
Max.  Time :      60.661 msec
Total Time :     284.648 msec
--------+---------------+------+--------------+-------------------------------
```

Example 2: Your output should include all of the output, even though the output has been trimmed in the trace below for brevity

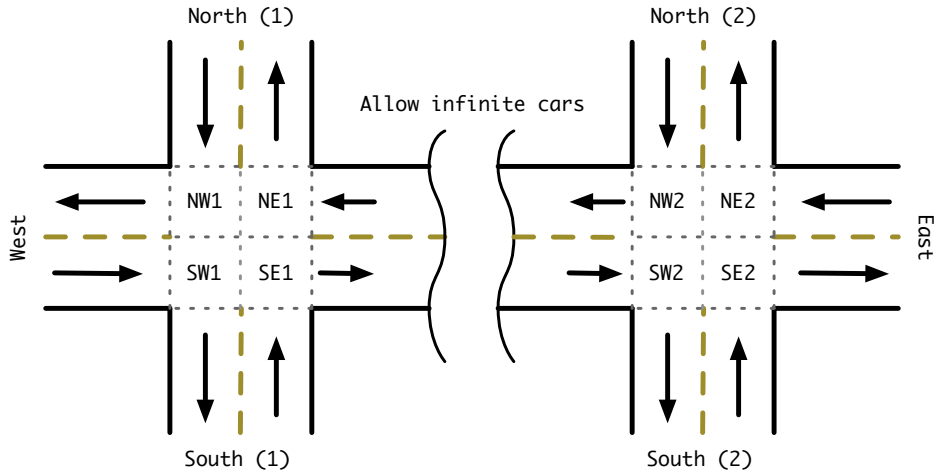
```
shell$ ./stoplight 10 50
--------------------
Time to live  :  10
Number of Cars:  50
--------------------
------------------------------
 Car ID | FROM to DEST  | Loc. |  Time (msec) | State
--------+---------------+------+--------------+--------------------------------
     18 |   E. to S.    |  1-- |        0.219 | Waiting in line
     18 |   E. to S.    |  1-- |        0.355 | Next in line
     18 |   E. to S.    |  1-- |        0.362 | Turn Left
     45 |   N. to S.    |  1-- |        0.047 | Waiting in line
     45 |   N. to S.    |  1-- |        0.059 | Next in line
     19 |   N. to E.    |  1-- |        0.014 | Waiting in line
     19 |   N. to E.    |  1-- |        0.021 | Next in line
     36 |   W. to N.    |  1-- |        0.016 | Waiting in line
     36 |   W. to N.    |  1-- |        0.027 | Next in line
     48 |   W. to S.    |  1-- |        0.014 | Waiting in line
      1 |   S. to E.    |  1-- |        0.056 | Waiting in line
      1 |   S. to E.    |  1-- |        0.077 | Next in line
      1 |   S. to E.    |  1-- |        0.083 | Turn Right
     21 |   S. to W.    |  1-- |        0.241 | Waiting in line
     21 |   S. to W.    |  1-- |        0.258 | Next in line
      1 |   S. to E.    |  1-- |       20.644 | Leave
     30 |   W. to E.    |  1-- |        0.392 | Waiting in line

// Lines removed for brevity

     49 |   N. to S.    |  1-- |     2112.497 | Leave
     41 |   N. to E.    |  1-- |     2022.409 | Turn Left
     41 |   N. to E.    |  1-- |     2082.975 | Leave
     17 |   N. to E.    |  1-- |     2050.840 | Turn Left
     17 |   N. to E.    |  1-- |     2111.270 | Leave
     34 |   E. to W.    |  1-- |     1922.097 | Go Straight
     34 |   E. to W.    |  1-- |     1962.758 | Leave
     37 |   E. to S.    |  1-- |     1897.201 | Turn Left
     37 |   E. to S.    |  1-- |     1957.420 | Leave
     47 |   N. to E.    |  1-- |     1951.363 | Turn Left
     47 |   N. to E.    |  1-- |     2011.917 | Leave
--------+---------------+------+--------------+--------------------------------
Min.  Time :      20.209 msec
Avg.  Time :    2003.516 msec
Max.  Time :    3361.641 msec
Total Time :  468822.675 msec
--------+---------------+------+--------------+--------------------------------
```

Project 2.2 – Bonus: Unbounded Traffic Problem – (+10 Points)

For our situation we have two intersections connected via a bridge that we need to manage, so the simulation looks like the image below. For simplicity, we will assume that an infinite number of cars can be on the bridge at any one time. Intersection #1 on the left and #2 on the right.



So if a car approaches from the North in intersection 1 (N^1), depending on where it is going, it proceeds through the intersections as follows:

- **West**: NW1
- **South**[1]: NW1 – SW1
- **South**[2]: NW1 – SW1 – SE1 – Bridge – SW2
- **East**: NW1 – SW1 – SE1 – Bridge – SW2 – SE2
- **North**[2]: NW1 – SW1 – SE1 – Bridge – SW2 – SE2 – NE2

Write all of your code and documentation for this part of the project in the `bonus` directory in the repository. Make sure you use all of the state and location changes described in the support library.

After You Finish Coding

Once you have completed your bonus solution, in a ***special section*** within your documentation you must answer the following questions, **separately**:

1. Describe your solution to the problem (in words, not in code).
2. Describe how your solution meets the goals of this part of the project, namely (each in a separate section):
   - How does your solution avoid deadlock?
   - How does your solution prevent accidents?
   - How does your solution improve traffic flow?
   - How does your solution preserve the order of cars approaching the intersection?
   - How is your solution "fair"?
   - How does your solution prevent "starvation"?

Bonus Example 1:

```
shell$ ./stoplight 1 2
--------------------
Time to live  :   1
Number of Cars:   2
--------------------
-------------------------------
 Car ID | FROM to DEST  | Loc. |  Time (msec) | State
--------+---------------+------+--------------+--------------------------------
      0 |   E. to S1    |  --2 |        0.127 | [I.2] Waiting in line
      0 |   E. to S1    |  --2 |        0.415 | [I.2] Next in line
      0 |   E. to S1    |  --2 |        0.862 | [I.2] Go Straight
      0 |   E. to S1    |  -B- |       41.868 | [Br.] Waiting in line
      0 |   E. to S1    |  1-- |       62.769 | [I.1] Turn Left
      0 |   E. to S1    |  1-- |      123.498 | [I.1] Leave
      1 |   S1 to W.    |  1-- |        0.280 | [I.1] Waiting in line
      1 |   S1 to W.    |  1-- |        0.553 | [I.1] Next in line
      1 |   S1 to W.    |  1-- |        1.020 | [I.1] Turn Left
      1 |   S1 to W.    |  1-- |       61.993 | [I.1] Leave
      0 |   E. to W.    |  --2 |        0.010 | [I.2] Waiting in line
      0 |   E. to W.    |  --2 |        0.282 | [I.2] Next in line
      0 |   E. to W.    |  --2 |        0.662 | [I.2] Go Straight
      0 |   E. to W.    |  -B- |       41.892 | [Br.] Waiting in line
      0 |   E. to W.    |  1-- |       62.960 | [I.1] Go Straight
      0 |   E. to W.    |  1-- |      103.694 | [I.1] Leave
      1 |   E. to N1    |  --2 |        0.009 | [I.2] Waiting in line
      1 |   E. to N1    |  --2 |        0.280 | [I.2] Next in line
      1 |   E. to N1    |  --2 |        0.672 | [I.2] Go Straight
      0 |   W. to N1    |  1-- |        0.007 | [I.1] Waiting in line
      0 |   W. to N1    |  1-- |        0.243 | [I.1] Next in line
      0 |   W. to N1    |  1-- |        0.749 | [I.1] Turn Left
      1 |   E. to N1    |  -B- |       41.491 | [Br.] Waiting in line
      1 |   E. to N1    |  1-- |       72.350 | [I.1] Turn Right
      0 |   W. to N1    |  1-- |       62.124 | [I.1] Leave
      1 |   E. to N1    |  1-- |       93.299 | [I.1] Leave
      1 |   S1 to W.    |  1-- |        0.008 | [I.1] Waiting in line
      1 |   S1 to W.    |  1-- |        0.242 | [I.1] Next in line
      1 |   S1 to W.    |  1-- |        0.969 | [I.1] Turn Left
      1 |   S1 to W.    |  1-- |       62.336 | [I.1] Leave
      0 |   N1 to S1    |  1-- |        0.008 | [I.1] Waiting in line
      0 |   N1 to S1    |  1-- |        0.240 | [I.1] Next in line
      0 |   N1 to S1    |  1-- |        0.614 | [I.1] Go Straight
      0 |   N1 to S1    |  1-- |       41.588 | [I.1] Leave
--------+---------------+------+--------------+--------------------------------
Min.  Time :      41.582 msec
Avg.  Time :      78.355 msec
Max.  Time :     123.492 msec
Total Time :     548.487 msec
--------+---------------+------+--------------+--------------------------------
```