University of Arkansas

Department of Computer Science and Computer Engineering

CSCE 3613 – Operating Systems — Spring 2019
Due: March 29 (Friday) 3:05PM — Homework #5

The objective of this assignment is to provide some hands-on experience on **multithreaded programming** and **thread synchronization**. You will design and implement a multithreaded program to control southbound and northbound traffic on an imaginary narrow bridge.

Traffic Patterns: Traffic will consist of cars and trucks. A vehicle (car or truck) may be traveling south or north: the direction of each vehicle will be determined according to a probability distribution at the time of its arrival. Each car/truck will take 2 seconds to cross the bridge.

Bridge Restrictions: The following restrictions are to be enforced.

- **R1.** Since the bridge has only one lane, the traffic may cross the bridge in one direction at a time.
- **R2.** Due to construction constraints, the bridge can accommodate at most 3 cars at a time.
- **R3.** While a truck is crossing the bridge, no other vehicle (car or truck) should be on the bridge.

Violating the restriction **R1** will cause collisions, while violating **R2** or **R3** will result in the collapse of bridge. **Your program should prevent collisions, deadlocks and the collapse of the bridge**.

Traffic Control Policies: Traffic flow will be controlled according to the following rules.

- **P1.** Trucks will have absolute priority over cars. That is, no car should be allowed to start to cross the bridge if there is any waiting truck at either end of the bridge. In case there are waiting trucks in both directions, then the traffic direction must be switched after each truck, to provide a fair waiting time to trucks in each direction.
- **P2.** If there are no waiting trucks, then the cars (if any) can start to cross the bridge. If there are waiting cars at both ends of the bridge, then you are free to choose the direction of the first car to cross the bridge (initially, or following a group of trucks). However, once a car C starts crossing the bridge in one direction, the cars heading in the same direction as C will have priority over any car that may be waiting/arriving at the other end of the bridge. Clearly, policy **P1** will be effective again as soon as a southbound or northbound truck arrives. But observe that before allowing any truck, your program must wait until *all the cars that are already on the bridge have left* (otherwise the bridge will collapse).
- **P3.** Subject to restrictions **R1**-**R3** and policies **P1**-**P2**, no vehicle should incur a delay if there is no opposite traffic crossing the bridge when it arrives. If there is sufficient traffic, the bridge capacity must be fully used (in other words, having cars cross the bridge one by one is not acceptable).
- **P4.** Any waiting vehicle can be selected to cross the bridge as long as restrictions **R1**-**R3** and policies **P1**-**P2** are followed.

Representing Vehicles as Threads: You will represent each vehicle by one thread, which executes the procedure `Vehicle-Routine` upon arrival at the bridge:

```
Vehicle-Routine(parameter-list)
{
Arrive(...)
Cross(...)
Leave(...)
```

}

The *parameter-list* in `Vehicle-Routine` above should contain at least *vehicle-id* (an integer uniquely identifying the vehicle), *vehicle-type* (car or truck) and *direction* (southbound or northbound). You are free to use additional parameters and to determine the parameter lists of procedures `Arrive`, `Cross` and `Leave`.

The `Arrive` procedure must check all the traffic/bridge restrictions and it must not return until it is safe for the vehicle to cross the bridge. The `Cross` procedure will just delay for 2 seconds for each vehicle. Finally, the `Leave` procedure must take additional steps to let the traffic control mechanism update its internal state and to let additional vehicles to cross the bridge, if any. **These procedures must print all the information necessary to follow the arrival and departure patterns as well as the list of waiting queues and crossing vehicles**. The format shown in the following example should be adopted:

Car #5 (northbound) arrived.
Car #7 (northbound) arrived.
Truck #3 (southbound) arrived.
.....
Car # 5 is now crossing the bridge.
Vehicles on the bridge (northbound): [Car #3, Car #4, Car #5]
Waiting vehicles (northbound): [Car #7, Car #8, Truck #2]
Waiting vehicles (southbound): [Car #10, Car #12]
.....
Car #5 exited the bridge.
Vehicles on the bridge (northbound): [Car #3, Car #4]
Waiting vehicles (northbound): [Car #7, Car #8, Truck #2]
Waiting vehicles (southbound): [Car #10, Car #12]
.....

Once a vehicle arrives, the type, the id, the direction of the vehicle have to be printed. Once the status of a vehicle changes (i.e., start crossing and exit the bridge), the information of the vehicles on the bridge and the waiting vehicles have to be reported as above.

Programming Language, Thread Libraries: For this assignment, you need to use C as the programming language. You can use any multithreaded programming library; however, you are encouraged to use Pthreads, since you are likely to find a large number of references for Pthreads. Pthreads are also available on the Turing.

Synchronization Primitives: You will write the procedures `Arrive`, `Cross` and `Leave` using *mutex locks* and *condition variables*. **Your solution must not employ busy waiting**. For full credit, you should only use `pthread_cond_signal()` for signalling any threads blocked on a condition variable. In other words, *avoid* using `pthread_cond_broadcast()` function. Because of thread scheduling mechanism of the library you are using, you may occasionally observe that vehicles do not leave the bridge in the same order they entered: a vehicle may overtake another vehicle traveling in the same direction. You do not have to fix this problem.

Running Your Program with Specific Schedules: In this assignment, you have to run your program for six vehicle arrival schedules given below:

1) 10 : DELAY (10) : 10
   car/truck probability: [1.0, 0.0]

2) 10 : DELAY (10) : 10
   car/truck probability: [0.0, 1.0]
3) 20
   car/truck probability: [0.65, 0.35]
4) 10 : DELAY(25) : 10 : DELAY(25) : 10
   car/truck probability: [0.5, 0.5]
5) 10 : DELAY(3) : 10 : DELAY (10): 10
   car/truck probability: [0.65, 0.35]
6) 20 : DELAY(15) : 10
   car/truck probability: [0.75, 0.25]

Here the numbers indicate the number of a group of vehicles arriving simultaneously at the bridge, while the numbers in parentheses indicate the delay before the next arrival(s). For example, under schedule (4) 10 vehicles arrive simultaneously at the bridge at the start of the experiment, 10 more vehicles arrive simultaneously 25 seconds after the arrival of the first ten vehicles and so on. Under schedule (4), 30 vehicles arrive at the bridge during the course of the experiment.

When multiple vehicles arrive simultaneously, no vehicle in a group is allowed to cross the bridge until all vehicles in the same group arrive.

When multiple vehicles arrive simultaneously, the direction and type of each vehicle will be determined randomly. **During all experiments, assume that the probabilities of having a southbound or northbound vehicle are equal (i.e. both are 0.50). The type of each vehicle (that is, whether it is a car or truck) will be determined according to the probabilities given at the end of each schedule in pair [car=X, truck=Y]**. For example, the schedule (3) is to be executed with pair (car=0.65, truck=0.35), meaning that the probability that an arriving vehicle is a car is 0.65, while the probability that an arriving vehicle is a truck is 0.35. For each arriving vehicle, first determine its direction and then its type according to these distributions. Observe that schedules (1) and (2) represent car-only and truck-only traffic patterns, respectively. If you are using C, you can use the `rand()` to generate a random number. The random number generator can be initialized by calling `srand()` with the desired seed.

Input and output of the program: The six specific schedules are required to given as command-line arguments to the program, one schedule each time. For example, if you run Schedule #1, type in `./bridge_crossing 1`. `bridge_crossing` is the executable. All the output of the program are printed to the screen.

Submission: Please create a tar file using the name hw5_<your last name>.tar including the following items.

1) The source code of your programs.
   Your code should include meaningful comments and variable names.
   Name your executable as `bridge_crossing`.
2) Makefile, and/or any other information necessary to compile and run your program if you are using additional facilities/libraries.
   Please note that I will use Turing to compile and verify your code.
3) A brief report consisting of the following contents:
   - A write-up describing the data structures you use.
   - The pseudocode for your solution (please note that source code augmented with comments is not pseudocode).
   - The output of runs for the schedules given above.
4) A README file specifies the names of the above files.

Resources: You are strongly encouraged to refer to parallel programming links available at http://hthreads.csce.uark.edu/wiki/Howtos.

Suggestions: First, you will have to study the basics of multithreaded programming in C (see Resources above). It is strongly suggested that you solve the problem with pen and paper before starting to code. An incremental implementation where you first address the simpler cases of car-only and truck-only traffic may be helpful. Then you can focus on the case of mixed traffic. **It is important not to postpone the assignment to the very last days unless you have prior experience in multithreaded programming**.

Grading: The relative weights of grading components will be approximately as follows:

- Write-up and pseudo-code: 15 points
  The output of six schedules are necessary for grading
- Correct Design and Implementation: 85 points
  - Bridge Restrictions: 25 points
  - Control Policy for Truck-Only Traffic: 10 points
  - Control Policy for Car-Only Traffic: 10 points
  - Control Policy for Mixed Traffic: 40 points
- **Total Points: 100**

Correct implementation of bridge restrictions is a necessary condition for traffic control policy implementation.

Late Penalty: No late submission will be accepted unless you consult with the instructor in advance.