CS 362 - Lab 4 – Process Synchronization and Threads

Due: December 5

For this lab assignment, you are required to implement the Traffic Intersection Problem in two languages, i.e., C++ and Java. The objective is to illustrate the different levels of control when one language (Java) does not allow you to micro-manage your synchronization primitives while the other (C++) gives you much more control.

Assume that there is an intersection with a 4-way stop sign. The usual rules of the road apply: the first to come to the stop sign gets to cross the intersection and there can only be one car in the intersection at any one time. If cars are lined up at each incoming road, they take turns. There apparently is no rule on who goes first if four cars happen to arrive at the intersection sign from four different directions, so you can define your own rule. Be careful to state your assumptions on your synchronization primitives, as not all implementations have the same behavior.

Your solution has to take into account that cars are not served round robin. Queues should be implemented for each street. There are several ways to organize the code. For example, we execute functions on arrival and departure, with a time delay function called **`drive()`** representing the time it takes to cross the intersection. The program has to make sure that a car doesn't wait if the intersection is empty.

The pseudo code (minus critical semaphore variables) for arrival and departure should be implemented as follows:

```
street[4];
Q[4] ;           //each queue for each street
empty           // intersection is empty (Boolean)

// car arrives at street[i]
arrival(int i) {
   if (empty) {
       empty =false;
       drive(i);
   else {
       Q[i]++;
       // wait for someone to signal that it is street[i]'s
           turn to go
       drive[i];
     }
} //arrival(i)
// departure of car from intersection
departure (int i) {
   // one car departs
```

```
    // pick the next intersection for next car to go
    // if no cars are in queues, empty = true
}
```

Make sure that each car is represented as a thread. When a car thread is created, randomly assign it to any of the four streets. Since it is well-known that the C++ and Java's random number generator are not random enough, use the following code for random number generation:

```
/* Returns a random number from 0 to 1 /

double rndom() {
    const long A = 48271;
    const long M = 2147483647;
    const long Q = M/A;
    const long R = M%A;

      static long state = 1;
      long t = A * (state % Q) - R * (state / Q);

      if (t > 0)
            state = t;
      else
            state = t + M;
      return ((double) state/M);
}
```

Your program should be menu-driven and allow for the user to enter the number of cars to be run for the demo.

You may use the Java on the Windows platform to do the Java portion of this. However, if you use the one on the Linux platform, you can easily use one menu for both versions instead of switching platforms during the demo.

Project Submission

Project is only considered completely submitted, i.e., to get a grade, if **ALL** items mentioned below are accomplished.

1. W drive:
   A directory consisting of the source code, makefile, and README files tar'ed or zipped and put under the cs362/lab4 directory on the W drive. Name your directory with both your email names, e.g., wagnerpj_tanjs_lab1.tar.

2. Email as attachment (tar'ed or zipped project) and percentage workloads.

3. Make an appointment to demo your project before the last day of class. No credit will be given if deadline is not met.