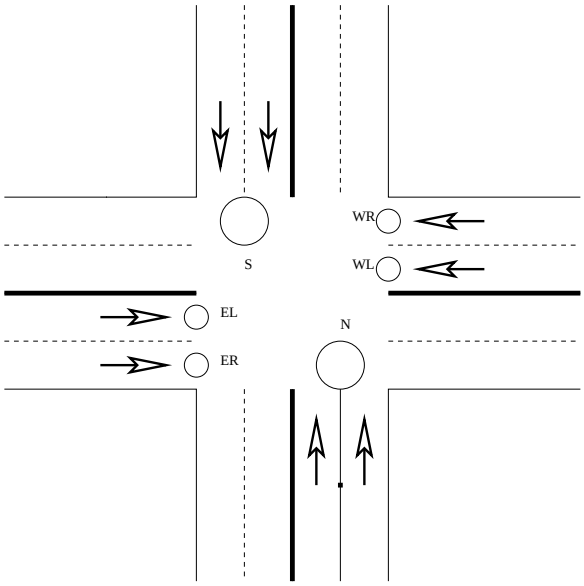
Traffic Signal Controller
Concurrent Programming Project 1

17th September 2002



The above diagram is an illustration of a particular street intersection that I used to traverse daily. Each circle stands for a set of traffic signals that all behave identically at all times. For example, both northbound traffic lanes get the same signal, but the left and right lanes of eastbound traffic get different signals. Northbound and southbound traffic get different signals.

Compatibility of movement: the following green (go) signals are compatible: (ER and WR), (ER and EL), (WR and WL) and (EL and WL). If any other signals are green simultaneously, crashes will result. This is because, in fact, north and southbound traffic are permitted to turn left, while traffic facing the EL and WL signals is required to turn left.

For each of the signals there is a sensor that indicates the presence or passing of a vehicle. The sensor sends a pulse when a vehicle arrives at it.

The behavior of the signals is dictated by what is detected by the sensors. If there is no traffic, or only ER andWR traffic, the ER and WR signals stay green. When traffic arrives for another direction the sensor indicates this to the controller. The controller stops any incompatible movements by first signalling yellow for 6 seconds and then changing to red. When all conflicting movements are stopped, the controller changes the signal to green. Once green, a signal changes to yellow (and then red)

- no sooner than 6 seconds after turning green
- no later than 12 seconds after turning green if there is conflicting traffic
- 2 seconds after the last signal from its sensor provided this is within the 6..12 second period
- In addition to these rules there is a nearby traintrack that has to be accomodated (see below).

If sensors indicate that there are conflicting requests they are to be served in round-robin order, beginning with ER and WR, N, S, and finally EL and WL. Except for the ER and WR directions, lights should only turn green when sensors indicate traffic is present except as these rules are modified to handle the train.

The Train

Not shown on the diagram but nevertheless present, there are train tracks to the south of the intersection, parallelling the east-west roadway. A train arriving sensor and a train departing sensor are on the tracks. When an arriving train is detected the signals must allow northbound traffic to clear the tracks for a 10 second green period, regardless of the information from the N sensor and regardless of the current state (of course any current incompatible signals must first be changed to red after a 6-second yellow). After that, only ER and WR traffic can proceed until the train departs. You may assume that each train departs before the next train arrives. After the train leaves, the signal should pick up where it left off, either restarting the interrupted phase if it has traffic, or continuing with the next phase in round robin order that does have traffic.

Undergraduates

Undergraduates may solve a simpler problem that manages ER and WR together as one signal, and also EL and WL together as one signal. Thus sensors and signal names EWR and EWL replace ER, WR, EL, WL in the input and output.

Graduate students

Graduate students must solve the whole problem. The additional difficulty comes from the fact that now multiple signals are compatible with one another. For example, ER is compatible with WR, but ER is also compatible with EL. If ER and WR are both green and there is EL traffic but no WL traffic we need to set WR to red and EL to green without affecting ER.

Consider also the case that ER and WR are both green and there is northbound traffic waiting. The WR sensor indicates moving traffic so the WR signal will stay green until it reaches its green maximum limit according to the rules above. If there is no ER traffic the rules above say that the ER signal should go red, but this doesn't make sense. As long as WR is green in this case ER might as well be, too. So in some cases the ER signal has to respond to either the state of the WR signal or the WR sensor. Similar observations can be made about other compatible pairs.

Designing a solution

I encourage you to initially think about your design independently of the details of how we represent the signal lights and sensors in the program. What states have to be made mutually exclusive? How might you ensure mutual exclusion using the synchronization mechanisms that we've talked about?

Suggestion: start with a simpler intersection that has only a single signal controlling north-south traffic and a single signal controlling east-west traffic. If you come up with good design principles you should be able to extend your solution to the more complex intersection in a straightforward way.

Decide what threads you are going to have and what conditions you need to signal between threads. Decide whether you are going to use monitors or semaphores. How many monitors or semaphores do you need? That is, how do the monitors and semaphores correspond to the principle elements of the problem, the signals and sensors. How will your program know when to take action based on the time specifications in the problem description?

Implementing signals and sensors

Your program will interact with the signal lights and sensors through its standard input and output.

Sensor input

Sensor input comes in the form of lines read from standard input. A line consists of the identity of the sensor (N, S, ER, EL, WR, WL, TA, TD) followed by a

newline character (i.e. you can run your program in a terminal session and type sensor input to it).

Signal and sensor output

Output from your program will be a series of lines, each line containing a timestamp and either a command to a single signal bulb or an echo of a sensor input. Timestamps are integers representing the number of milliseconds since the start of the run. A light command is the letter L followed by a signal name, a color indicator, and 1 to turn on, 0 to turn off. The signals are named N, S, ER, EL, WR, WL.

Sensor inputs must be echoed to the output as soon as they arrive. The output format allows a program that I will write to check whether the controller is operating properly. The output might also be used to drive a graphical display of the state of the intersection.

If you wish to interleave debugging or other information in output feel free. Just make the first token a timestamp and the second token, which must be present, different from L and from S.

Example output

```
0.00 L N R 1
0.00 L S R 1
0.00 L EL R 1
0.00 L WL R 1
0.00 L ER G 1
0.00 L WR G 1
2.25 S N
5.00 L ER G 0
5.00 L ER Y 1
5.00 L WR G 0
5.00 L WR Y 1
...
```

Assignment details

You may write your program in Java, C, or C++. If using C or C++ you will use the pthreads package. If using Java you'll start from the built-in synchronization facilities. If you want to program this for Windows please see me. We'll have to negotiate how to make the program testable.

Graduate students are responsible for the complete problem, separate controls for ER, EL, WR, and WL. Undergraduates may control ER and WR together and EL and WL together, or receive extra credit for additions. (Don't attempt

the split ER/WR, split EL/WL until you have success with the basic intersection *and* you have archived a copy of it. You will lose credit if you turn in something that doesn't work on the basic problem.)

Preliminary design document due Sept. 26: this should lay out your general approach to the problem, including what you've determined to be the basic synchronization requirements, areas that you see as problematic, what threads you are going to use, etc. (10%)

Preliminary code due Oct 4 (under my door): this should illustrate that you have mastered the input and output requirements, the synchronization primitives, and the compiling and running steps for your choice of concurrent language. Working code for a 2-way intersection with sensors would represent good progress. (10%)

Final turn-in: Oct 15th in class for printed material including code and a revised design document describing in detail what you built. Tell me what is the most important thing for me to look at in your design and code. Tell me what changed between the preliminary design and what you ended up with. Tell me how well it works. Does it meet the specification? I will also provide an on-line submission mechanism for your source code and executables. Your executable will be tested with programs that supply sensor input and check the output to make sure the signals work correctly.

Suggested plan

I suggest building this project incrementally: build working code for a series of simpler problems leading up to the final program. One possible series of increasingly complicated intersections would be:

1. A two-way intersection using only timed lights – no sensors.
2. A two-way intersection that stays in say EW mode until traffic is sensed in the NS direction, stays in NS until traffic is sensed in EW. Try to get the minimum and maximum times working at this stage.
3. Split N and S.
4. Split EW into EWR and EWL corresponding to through traffic and left turns on EW road. You may use EWR and EWL as the names for the sensors and signals if you wish.
5. Add the train; this will meet the requirements for undergraduates.
6. Split EWT into ER and WR, and EWL into EL and WL.

Up until the last split I think you can manage with just two threads: one to manage the signals, and the other to read the input. For the last split, you will

probably need to go to one thread per signal – and doing the whole problem that way leading up to the last split will probably give you a better starting point.

You will find it helpful to describe the intersection in data structures that are interpreted by the code for the signals. This will vastly reduce the amount of code you have to write. You do not want to write, and I do not want to read, even 150 lines of code for each different signal in this problem! As a point of comparision, my java code corresponding roughly to a combination of steps 4 and 6 above contains 30 lines to build the datastructure describing the intersection, 175 lines to describe the behavior of the signals including producing the output, and 20 lines to read the sensor input. I estimate that adding the train will cost less than 20 lines of code. Finishing step 6 will probably add another 20-30 lines, but also will require some significant additions to the data structure.

Do not be fooled by the small size of the code for this project: there is a great deal going on in this problem, You will have to spend considerable time thinking about the problem and about your plan for solution in order to even get started. You cannot put this off until the week before it is due and expect to succeed. The preliminary due dates are intended to both get you moving on the project, but also to give me a chance to provide feedback to keep you on the right track.