Summary

This java program simulates a number of customers being processed through a DMV Office. They take a number, get called to wait on an agent, and perform an examination to receive a license. This sounds simple to make in a real world scenario, but it gets complicated in the no context, no assumptions world of programming. The simple action of reacting to an event is very complicated to describe to a computer which has an objective perspective.

Starting this project was probably the hardest part. It’s easy to look at the Barbershop example after it’s finished and think to understand it, but to make it from scratch is like trying to organize colors. At first it just seemed impossible to describe every interaction, and that is because it is. The best way to start is to describe one entity's action at a time. All the DMV workers' first action is to wait for a customer, and each Customer’s first action is to go to the Info desk. I made the project by thinking about this series of actions and reactions.

Further difficulties came from defining exclusive interaction between actors. In this project, 20 different customers are set to enter the DMV in some unspecified order, and each expects to be served as they come in. However, there is only one Announcer, and once the Announcer is occupied the others are still waiting at the same time. Then once the Announcer finishes the call, it has to decide who goes next, but it has to do that by the numbers they have, which it does not have direct access to. Additionally, the Announcer can only call them by their number.

With these constraints, the only way to have this interaction work is to create a communication resource between the customers and the Announcer. This has its own issues. Now access to this resource has to be controlled or there will be synchronization issues. I controlled this by introducing a semaphore to provide mutual exclusion to the communication resource, but it does limit concurrency.

After making all these checkpoints, I realized that this kind of problem can be thought of as a series of buckets on strings or something similar. The four workers all start as empty buckets, and each customer comes in as a full bucket on a parallel string. A full bucket represents a unit of action, and the bucket is allowed to move down the string as long as it’s full, but must stop at empty buckets it crosses to have them pour into each other. The entire program is just the passing back and forth of units of action until each customer bucket carries its unit of action out.