Zach Dingels

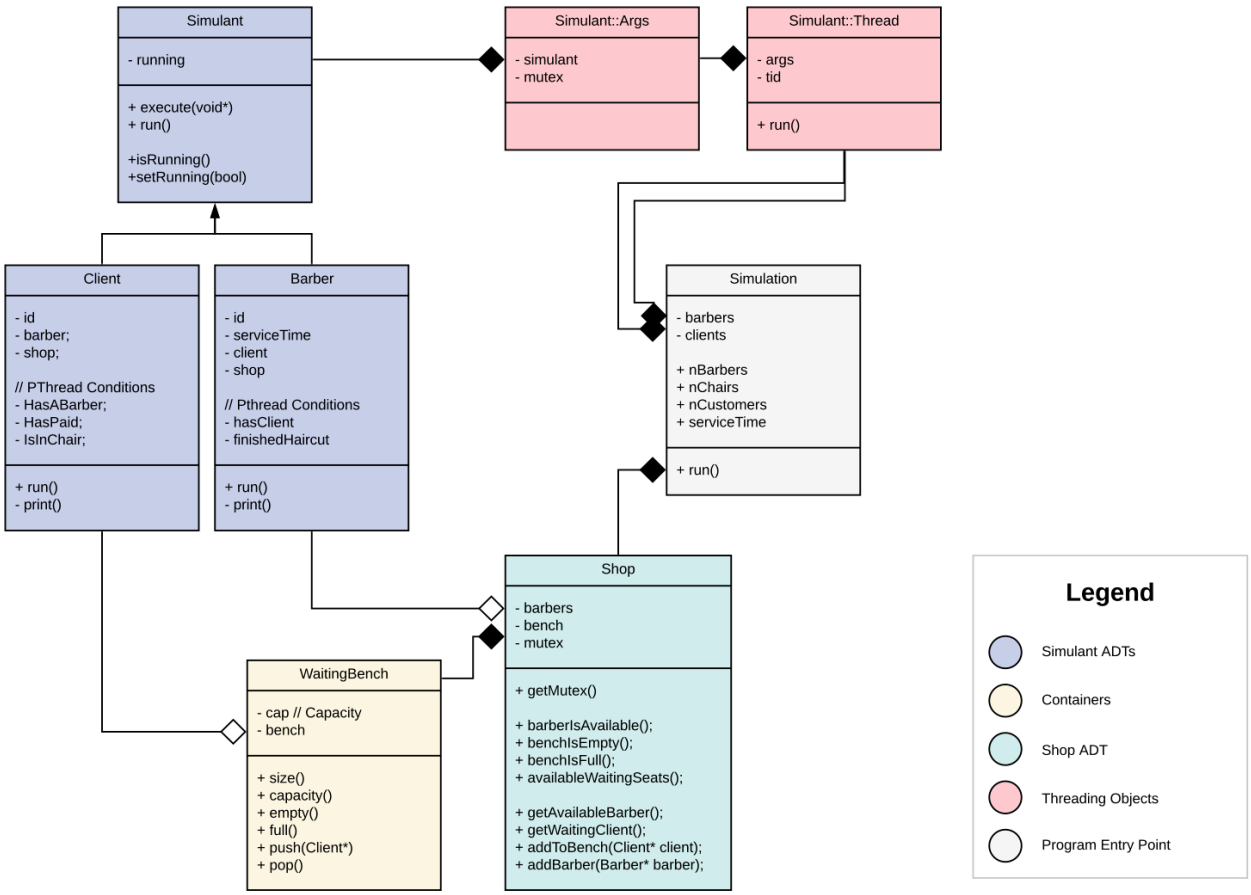
Prof. Dimpsey

May 12th, 2020

CSS 503 - Program 2: nBarbers

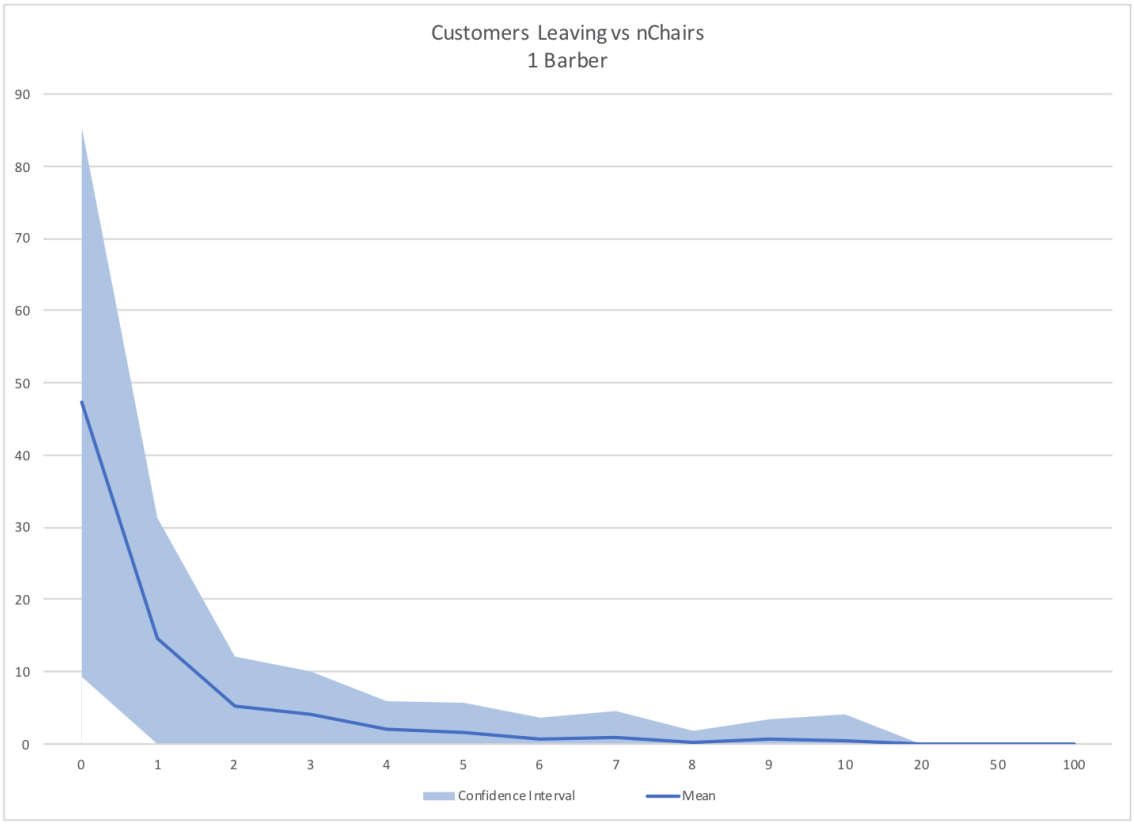
Design

The program was designed as a simulation. Logic for the barber and client were both abstracted into Client and Barber classes. In the context of a simulation, these are both simulants and were thus sub classed from a Simulant base class. Information about the context that both the barber and clients shared was encapsulated in a Shop class. A major component of the shop was the waiting bench, which was given its own class. Logic and data for running simulants on threads was put into two structures: Simulant::Args holds data needed for a Simulant to be passed to a pthread and Simulant::Thread couples args with a unique thread so they can be easily managed. Finally, a Simulation class was created to represent the problem as a whole and coordinate different parts of the program. The following class diagram shows these relationships:



Performance Evaluation

The following graph shows how many customers left the shop given the number of waiting chairs a shop had. Each value of nChairs was ran 30 times to estimate 95% confidence intervals.



With more than 1 barber servicing customers there was no significant number of people waiting. When two barbers were working and nChairs was 0 a few customers left the shop. No customers left with 3 barbers.

Approximately how many waiting chairs would be necessary for all 200 customers to be served by 1 barber?

About 20 chairs are necessary.*

Approximately how many barbers would be necessary for all 200 customers to be served without waiting?

At least 3 Barbers would be necessary.*

*This was all ran on my local machine, results may vary.

Limitations & Extensions

The program is obviously limited by the number of threads that can be run at a given time. On my machine I ran `cat /proc/sys/kernel/threads-max` which told me I can run 7592 threads at once.

There was also a lag between a thread being created and when it is actually running. This limits the accuracy of the model. Sometimes, a customer is scheduled when the barber is busy and should leave the shop. Because of the lag the barber is able to finish serving it's current customer and be ready by the time the scheduled customer is actually running.

It would be interesting to extend the program by customizing individual parts. For example, what if barbers service times were different? Maybe the rate at which customers are created followed a more complex pattern. The program could also be expanded to have multiple shops.

Finally, the program could be extended by adding more detailed logging information. For example, calculating statistics on the average number of people waiting would be useful. Recording the time that events happen would be very useful. This would answer questions like how long does the average customer wait for a haircut, how long is there in between customers entering the shop, etc...