Lydia Calderon-Aceituno
LM4
DSC Fall 2022

Question 1: Specify the following questions:

- **All your assumptions in the simulation, including how did you determine the customer arrival time, shopping time, the items they buy, etc.**
  We want in total to have 1000 customers. To seem more realistic, I wanted to have their arrival time reflect the time of day. I split the 10 hours into four windows of time and split the 1000 customers into different batches to arrive in each window. Within each window, the time was evenly split between customers. For windows in "rush hours", more customers would arrive and thus there was less time between customers (lines 217-240 in my LM4.cpp handle this).

  Shopping time and item selection were basically instantaneous, I just had each customer add a random number of groceries from 10 possible options to their shopping cart and then attempt to checkout.

- **The structure of your code. For multi-thread simulations, how did you synchronize threads? How to protect critical sections? For event-driven simulations, what events you used? How does your simulation proceed?**
  My simulation was event-driven and I used `arriveAtStore, arriveAtCheckout,` and `depart` events. My simulation begins with one customer arriving in the store—a single `arriveAtStore` event is pushed back to the FEL. From there, the simulation jumps into a while loop driven by the current time of the simulation. The while loop ends once the simulation time reaches 10 hours. Each iteration, the imminent event is found and removed from the FEL. The current time is set to the imminent event's timestamp. A special method is then called based on the imminent event type. After the imminent event method has completed, the status of the store (running total of revenue, average wait time for each customer) is displayed. The final stats are displayed at the end of the simulation (after the while loop has completed).

  SPECIAL EVENT METHODS:

  Arrival event
  - A customer arrives
  - The number and type of groceries they shop for is randomly determined, and all of these items are added to their personal shopping cart
  - The revenue based on these items is calculated and added to the store's running total
  - The thread sleeps for a while to simulate the time required for shopping
  - The amount of time required for this customer to checkout is calculated based on the number of groceries to be purchased
  - This customer is ready to checkout! A `arriveAtCheckout` event is pushed to the FEL
  - Based on the time of day, the next customer's `arriveAtStore` event is pushed to the FEL a certain timestep away in the future

Arrive at checkout event

- If a cashier is free, the customer is added to their line and a departure event is scheduled based on how much time it will take for them to checkout
- If no cashier is free, the customer is added to the shortest cashier line

Departure event

- Based on the cashier line that this customer is in, remove them from their line
- Check if there is anybody waiting behind them
- If there is someone waiting, schedule their departure event based on how long it will take for them to checkout

- **What statistics you collected? How did you collect them?**
  I collected running total of store revenue by adding up the cost of each item bought as customers added them to their shopping cart.

  I also collected the average time spent waiting in line by adding up each individual customer's time waiting in line and dividing it by the number of customers that came to the store.

Question 2: List two commercial software simulators using multi-thread programming and two using event driven simulation.

AnyLogic and Arena, both commercial dynamics modeling softwares, are discrete event simulation softwares.

Excel and Chrome are both examples of software simulators that use multi-threading.

Question 3: Least at least two advantages and two disadvantages of your simulation method (multi thread programming or event-driven simulation) compared to the other one. Explain each of your points

Event-driven simulation is advantageous in that it does not require protection of critical sections since nothing is really happening concurrently. This greatly simplifies the code and prevents issues such as deadlock and race conditions.

Another advantage of event-driven simulation is that is provides a structure to the progression of events (unlike multithreading where the programmer has no control at all over when exactly events take place). In event-driven simulation, the programmer is given a lot more control to decide when events occur.