Jonathan Benson
CS303
Project 1 Final Report - Multi-Elevator System Simulator
6 October 2019

How to Build Executable

1. In a terminal, navigate to the project folder
2. Make a build directory in the project folder (we have it named “build” in the .gitignore file)
3. cd into the build directory
4. Type the command cmake -G “<BUILD SYSTEM GENERATOR>“ ..
    a. The BUILD SYSTEM GENERATOR is specific to your operating system / compiler. Checkout this link for more information:
        i. https://cmake.org/cmake/help/latest/manual/cmake-generators.7.html
5. Execute the command “make”
6. Now there should be an executable entitled “MESS” inside the build directory.

Overview of the Multi-Elevator System

Assumptions

- **The only changing size of input would be the number of elevators and clients in the system.**

Each elevator consists of 5 priority queues

1) downward inbound
    - clients waiting to be picked up below the elevator
    - sorted descending by their current floor
2) upward inbound
    - clients waiting to be picked up above the elevator
    - sorted ascending by their current floor
3) downward outbound
    - clients waiting to be dropped off below the elevator
    - sorted descending by their next floor
4) upward outbound
    - clients waiting to be dropped off above the elevator
    - sorted ascending by their next floor
5) record
    - a record of processed clients
    - sorted descending by their last time punch (arriving at their next floor)

Each client has 3 time punches

1) time when pressing the button to get on the elevator
2) time when boarding the elevator at their starting floor
3) time when dropped off of the elevator at their next floor

Simulation Flow:

When a client is presses the elevator button, their first time punch is recorded and they are pushed to an elevator's inbound queue depending on whether they are above or below the elevator. When they are picked up, their second time punch is recorded, and are pushed to an elevator's outbound queue depending on whether their next floor is above or below the elevator. When the elevator drops them off at their next floor, their third time punch is recorded and they are pushed to the elevator's record of processed clients.

Algorithm for the Multi-Elevator System Simulation
---

* This algorithm was used to simulate a multi-elevator system
* This algorithm does not include the time-keeping system used in the simulation to track the time
    - The time is kept track a master time clock (seconds) that ticks every iteration of the outermost loop
    - Each elevator has an initialized time value that it takes to traverse one floor and one for how long it takes to stop at one floor for a client
    - Each elevator also has stopwatches for both time values that keep track of its state and direction

Efficiency

$O(n^2)$

- For the multi-elevator system the Big O of the simulation would be $O(n^2)$. This is due to the constant iteration through every elevator in the system and every client contained in those elevators' queues.

- ➔ given an array of elevators with matching speeds and randomized starting floors
- ➔ given an priority queue of incoming clients with randomized starting and next floors, and sorted by a randomized first time punch // time of elevator-button-press
- ➔ while the incoming clients queue is not empty and all the elevators are not empty
    - ◆ for each incoming client who's first time punch is equal to the current time
        - get the closest elevator to the client from the array of elevators // algorithm below
        - if the elevator is idle
            - if the elevator has no direction // it is empty
                - ◆ if the elevator's current floor matches the incoming client's current floor
                    - initialize the elevator's state to idle
                    - send them to the elevator's outbound queue
                    - change the elevators direction
                - ◆ else
                    - send the client to the elevator's inbound queue
                    - set the elevator in motion in the direction of the client's current floor
                - ◆ pop the client from the queue of incoming clients
            - else if the elevator does have a direction
                - ◆ if the elevator's current floor matches the client's current floor
                    - send the client outbound
                - ◆ else
                    - send the client inbound
                - ◆ pop the client from the queue of incoming clients
        - else if the elevator is in motion
            - if the client's current floor matches the floor of the elevator // the elevator just passed the floor the client is on
                - ◆ send the client inbound queue of the opposite direction of the elevator
            - else
                - ◆ send the client inbound normally
            - pop the client from the queue of incoming clients
    - ◆ for each elevator in the array of elevators
        - if the elevator has no direction // it is empty
            - if the direction of the elevator is down
                - ◆ for each client in the elevator's downward inbound queue who's current floor matches the floor of the elevator
                    - send them outbound // pick them up
                - ◆ for each client in the elevator's downward outbound queue who's next floor matches the floor of the elevator
                    - send them to the elevator's record of processed clients // drop them off
            - else
                - ◆ for each client in the elevator's upward inbound queue who's current floor matches the floor of the elevator
                    - send them outbound // pick them up
                - ◆ for each client in the elevator's upward outbound queue who's next floor matches the floor of the elevator
                    - send them to the elevator's record of processed clients // drop them off
        - if the elevator is empty // the elevator should not move when it is empty
            - set its state to idle
            - set its direction to NA
        - if the elevator's downward inbound and outbound queues are empty
            - change its direction to UP
        - else if its upward inbound and outbound queues are empty
            - change its direction to DOWN

Closest Elevator Algorithm
---
* This algorithm was used to calculate the closest elevator to a client upon sending them inbound
* This algorithm consists of two functions:
    1) Iterates through all the elevators, returns the elevator with the closest "distance" computed from function (2) of the elevator to the client
    2) Distance function

Efficiency

O(n)

- The algorithm only iterates through each elevator in the system.

---

1) Closest elevator function

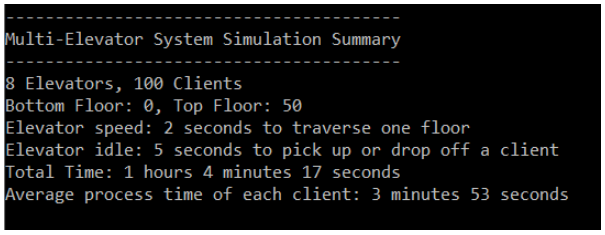
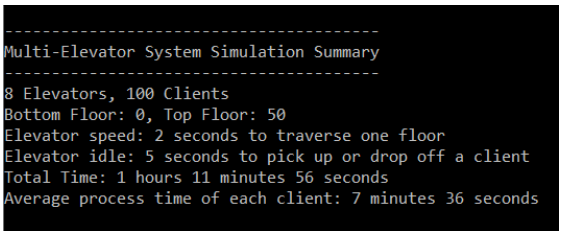
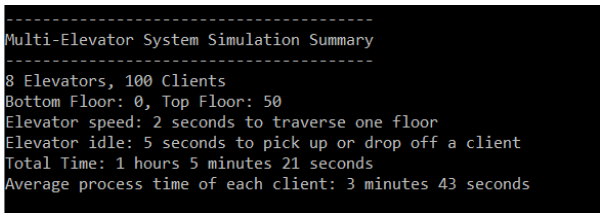
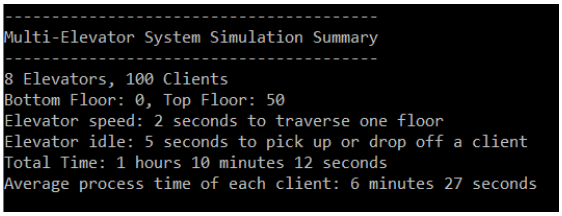
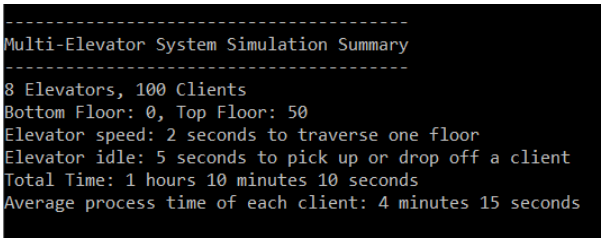
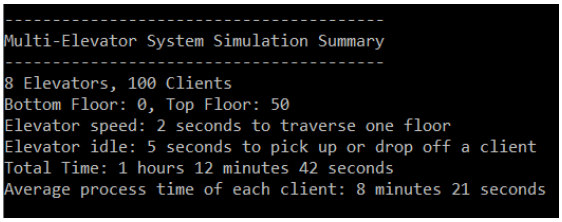
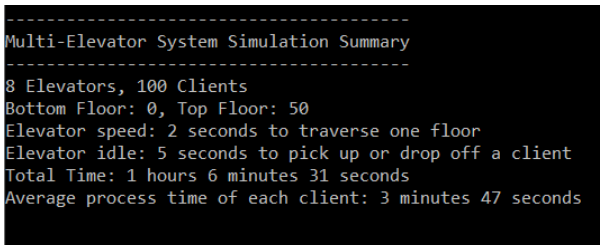
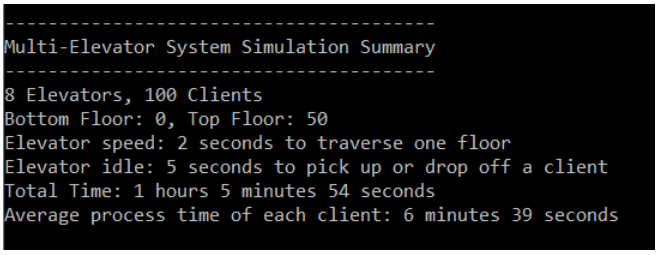
- ➔ given an array of elevators and a client
- ➔ ce = first elevator in the array of elevators
- ➔ for each e in the array of elevators
    - ◆ if the distance of the e to the client is less than the distance of ce to the client
        - ce = e
- ➔ return e

2) Distance function

- ➔ given an elevator and a client
- ➔ if the elevator is empty
    - ◆ return the absolute value of the elevator's current floor minus the client's current floor
- ➔ else if the elevator is idle and the elevator's current floor matches the client's current floor
    - ◆ return 0
- ➔ else
    - ◆ if the direction of the elevator is down
        - if the elevator's downward queues are empty
            - return abs(elevator's current floor - client's current floor)
        - if the client's current floor is greater than or equal to the elevator's current floor
            - if the downward inbound queue of the elevator is empty // clients waiting to be picked up below elevator
                - ◆ return abs(elevator's current floor - client's floor at front of elevator's downward inbound queue) + abs(elevator's current floor - client's current floor)
            - else
                - ◆ return abs(elevator's current floor - client's floor at front of elevator's downward outbound queue) + abs(elevator's current floor - client's current floor)
        - else if the client's current floor is less than the elevator's current floor
            - return abs(elevator's current floor - client's current floor)
    - ◆ else if the direction of the elevator is up
        - if the elevator's upward queues are empty
            - return abs(elevator's current floor - client's current floor)
        - if the client's current floor is greater than or equal to the elevator's current floor
            - if the upward inbound queue of the elevator is empty // clients waiting to be picked up above elevator
                - ◆ return abs(elevator's current floor - client's floor at front of elevator's upward inbound queue) + abs(elevator's current floor - client's current floor)
            - else
                - ◆ return abs(elevator's current floor - client's floor at front of elevator's upward outbound queue) + abs(elevator's current floor - client's current floor)
        - else if the client's current floor is less than the elevator's current floor
            - return abs(elevator's current floor - client's current floor)

Testing the Efficiency of the Closest Elevator Algorithm

To test the efficiency of the closest elevator algorithm, we ran 5 simulations choosing the closest elevator and another 5 simulations choosing a random elevator for each incoming client.

Closest Elevator Algorithm	Randomized
	
	
	
	

From the results above, the average percent reduction in the average process time of each client found was approximately 45% implementing the Closest Elevator Algorithm over a Randomized Selection.

$$\% \, Reduction = \frac{\left| CEA_{avg\ process\ time} - RS_{avg\ process\ time} \right|}{RS_{avg\ process\ time}}$$

UML Class Diagram of the Multi-Elevator System