CS2028C-F19

HW#3&4 Elevator Simulation

Design a program/framework to do **simulations** of the daily service of elevators in a building, using a variety of data structures such as stacks, queues, priority queues, linked-lists, binary trees, and binary search trees, and later on, graphs.

HW#3- Single Car, Single Floor (e.g., home floor) [Due 11/20]
HW#4- Multiple Floors, Elevator Controller [Due 11/27]
Bonus – Multiple Cars, Multiple Floors [Due 12/4]

Tasks:

- Develop a program to be used in elevator simulation applications, at first consider simple situations, e.g., one car serving passengers at a floor.
- Create a sample testing data set of passengers (per a single floor, first, then multiple floors) for this program.
- Model the elevator controller and the states of the elevator run/idle, then simulate elevator run.
- Generate a daily report of elevators performance: including the number of passengers served, the average wait time, the maximum wait time, the average total travel time, and the maximum total travel time.

NOTES on modeling:

Below is only a sketch of things involved in this type of problem FOR YOUR REFERENCE and future discussion in class. Feel free to add your own assumptions.

You may use the ERC building as an example:

ERC has six floors (labeled from 3 to 8).

It has two cars.

The home floor for the elevators is Floor3.

Important things to model may include passenger (P), elevator car (C), hallway up/dn buttons (HBtn), hallway display (HD), car buttons (CBtn), motor, car door, floor limit sensors, clock, timer.
These things can be represented by a struct/class, each containing all the relevant data.
Passenger: PID, timeOfRequest, originFloor, destFloor, waitTime, travelTime.
Car: CarID, tripCounter, floorServedByTrip, passengerCount.
HallCallQueue: upServiceCall, dnServiceCall
Timer and time counter: idle, doorOpen, doorClose, moveByOneFlightTime=5s, accelerateUpTime=2s, accelerateDnTime=2s, decelerateUpTime=2s, decelerateDnTime=2s,

Car movement can be represented by a state-transition diagram (state machine)

a) Each car goes through a finite number of states:

S1-Idle/Wait state, S2-Accelerate, S3-ConstantMotion, S4-MoveByOneFloorSensor, S5-Decelerate.

b) Events and conditions that trigger the transition between states:

S1->S2:

Inputs: (CBtn pressed->CBtnQ entry added) or (HBtn pressed->HBtnQ entry added)
Pre: {CBtnQ not empty} or {HBtnQ not empty}
Determine direction of travel: if destF – currF > 0
setTripDirection=Up; else =Dn
Post: close car door, motor on for the direction, timestamp TripStartTime, TimeCounter on
UpdateCBtnQ and HBtnQ, TripPlanBST
Outputs: car in motion, serve the CBtnQ and HBtnQ

S2->S3:

Inputs: {CBtn} or {HBtn}
Pre: ExitFloor-sensor
Post: TripTime = TripTime + accelerateTime
Outputs: NextFloor <- nextFloorOfBST

S3-> S4:

Inputs: {CBtn} or {HBtn}
Pre: ExitFloor-sensor -> currF
Post: TripTime = TripTime + moveByOneFlightTime
Outputs: currF < nextFloorOfBST

S3->S5:

Inputs: {CBtn not empty} or {HBtn not empty}
Pre: NextFloor = nextFloorOfBST
Post:
Outputs:

S5->S1:

Inputs: TripPlan
Pre: TripPlan not completed
Post:currF = EnterF
Outputs:

c) Time line

As we consider a time-dependent event driven program, time needs to be represented in some format. Assume the period to be studied is during the business hours of a weekday. The activities may occur any time between 7:00 AM till 7:00 PM (7:00:00 – 19:00:00).
What would be the unit for time? Seconds or minutes?
Using second as the smallest time unit, 12 hours will have 43,200 seconds. We may choose a coarser time unit, say 5-second intervals. Then the time line can be divided into 8,640 units.

On a campus setting, the people traffic for elevator could be dependent on the scheduled activities at a workplace or school. It is usually heaviest during morning and late rush hours, lunch time, and during class breaks. Based on these observations and some assumptions, we could try to develop an outline of the rough traffic pattern based on the nature of the occupants and the usage of the building.

Using a sample run in one of the elevators on campus: A round trip: F8->F6 and -- F6->F8

======== ========== P# = {#, Fat-src, up/dn, Fdest,Tarr, Tgeton, Tgetoff, Twait, Tride}

P1-arrFsrc8-phb-dn:13:09:07 == > P1 = {1, F8, dn, --, 13:09:07, --, --, --, --}

S1-Tcar-arvF8: 13:10:01 == > P1 = {1, F8, dn, --, 13:09:07, --, --, --, --}

S1-Tp1-getonCar 13:10:05 == > P1 = {1, F8, dn, --, 13:09:07, 13:10:05, --, 58, --} 10:05 – 09:07 = 58

S1-Tp1-pb6 13:10:06 == > P1 = {1, F8, dn, F6, 13:09:07, 13:10:05, --, 58, --}

S2-Taccel-dn: 13:10:10

S3-Tmov8 13:10:13

S4-Tmov8-7x: 13:10 19

S3-Tmov7 13:10:24

S4-Tmov7-6x: 13:10:27

S3-Tmov6 13:10:29

S5-Tslow-dn: 13:10:31

S1-Tcar-arvF6 13:10:35 == > P1 = {1, F8, dn, F6, 13:09:07, 13:10:05, --, 58, --}

P1 leave 13:10:36 == > P1 = {1, F8, dn, F6, 13:09:07, 13:**10:05**, 13:**10:36**, 58, **21**}

== Trip 1: {run#=1, 13:10:01-13:10:35, dir = dn, 8 to 6, travel-time = 10:35-10:10= 25, #rider=1}

==========

P2-arrFsrc6- 13:10:35 == > P2 = {2, F6, up, --, 13:10:35, --, --, --, --}

S1-Tp2-getonCar 13:10:35 == > P2 = {2, F6, up, --, 13:10:35, 13:10:35, --, 0, --}

S1-Tp2-pb8 13:10:37 == > P2 = {2, F6, up, F8, 13:10:35, 13:10:35, --, 0, --}

S2-Taccel-up 13:10:43

S3-Tmov6 13:10:46

S4-Tmov6-7x 13:10:51

S3-Tmov7 13:10:56

S4-Tmov7-8x 13:10:58

S3-Tmov8 13:10:59

S5-Tslow-up 13:11:02

S1-Tcar-arvF8 13:11:06 == > P2 = {2, F6, up, F8, 13:10:35, 13:10:35, --, 0, --}

P2 leave 13:11:07 == > P2 = {2, F6, up, F8, 13:10:35, 13:**10:35**, 13:**11:07**, 0, **25**}

== trip 2: {run#=2, 13:10:35-13:11:06, dir = up, 6 to 8, travel-time = 11;06-10:43= 23, #rider=1}

The travel time of the elevator car can be viewed as the following:.

S1- Idle time (including opening the door, door open wait time, close door) = 10 seconds, two units.

S2- Car Move-Accelerate time (departing and pickup speed before crossing the floor sensor) = 3~5 seconds

S3- Car Move – steady state/constant speed at a floor

S4- Car Move - Crossing floors (sensor/beep) = about 10 seconds per floor

S5- Car Move-Decelerate (to stop) = about 3 ~ 5 seconds

d) The context: Building Layout an occupant population

A more elaborated regular traffic pattern can be developed for the tenants of the building. But it will not be possible for the visitors. Using the ERC building as an example, it houses both offices and laboratories. Office occupants are the permanent users who are part of the daily elevator user. Lab users are irregular users of the building. Floor 4 is the main entrance. Floor6 and Floor 7 connect to an adjacent building. Therefore, there are people who transit through Floor4 to Floor6 or Floor7 to and from the adjacent building. A wild guess. Let's assume that Floor 8 has about 20 regular tenants, Floor 7 about 30 tenants and 20 lab users, Floor6 20 tenants and 10 users, Floor 5 20 tenants and 15 users, Floor 4 20 tenants and 15 users and Floor 3 10 users of the labs. Assume these regulars use elevator at least twice a day, that would be around 400 rides. Adding the transit people, we may make a rough estimate of a total of 800 passengers using the elevator daily.

The distribution of passenger arrival at each floor should be context-dependent and a time-series.
Context: usage of the rooms in the building: {person: office/lab, role, task purpose}

Data structures

To keep the current active list of all the passengers waiting at all floors, a queue data structure, called *carServRequest*, could be defined. To manage the car travel, these requests can be further separated into smaller lists, identified by the floor and by the direction of the target destination floor, either up and down. (floor passenger list, both FPLup and FPLdn).

People using elevator: {passenger#, Fat-src, direction:up/dn, Fdest,Tarr, Tgetoncar, Tgetoff, Twait, Tride}

The range for each of these parameters:

PassengerCount [1, MaxPassengerNum=800], Floor-origin:Fat-src [3-8], car-direction:dir [up, dn], Floor-destination:Fdest [3-8], TimeOfArrivalOnFloor:Tarr [1-8640], Tgetoncar [1-8640], Tgetoffcar [1-8640], Twait [1-600], Tride [1-100].

The wait time is the difference between Tgetoncar and Tarr, and the time of the travel/ride is Tgetoff – Tgeton.

e) Car Workflow

The system initialization:
Each car has a home floor, where the starting state, the S1-Idle state, begins.
Assume that the elevator begins its service at time line on 7:00:00.

Each trip consists of a directional run until the top/bottom floor on the plan list is reached.
When reaching to the end of a run, the elevator can either idle at that floor, indefinitely, or after a given dormant interval, goes back to its home floor, say, Floor4.

The trip plan is generated when in idle state (no previous calls) a call or calls(*), either a hall call or a car floor call, is entered into a queue (carServRequest). The first destination floor entered will determine the direction of the trip.
Two strategies: a) This plan can be locked up and no new request is serviced until the trip is completed or b) the plan can be dynamically updated as the car travels along the path and more calls are entered during the run.

A sample scenario: A Case study of Car Controller data structure.

Car idle at F4. Hall up call registered at F7, and car floor call F8 entered. *carServRequest*={F7, F8}
Car trip plan created: {trip#, dir = up, Fsrc=F4, Fdest=(F7, F8), Plan=(nextStop=F7, endFloor=F8)}
Trip Log:
Car moves: F4->F5, (*) new Hall up call at F6 registered in carServRequest queue== > update trip plan
{trip#, dir = up, Fsrc=F4, Fdest = (**F6**, F7, F8), Plan=(nextStop: **F6**, endFloor=F8)}
Car moves: F5->F6. F6 == nextStop.
Car stops at F6, serves a new passenger, whose car floor call (*) is registered in carServRequest queue, and added to the plan, if not already on the list.
== > update trip plan: {trip#, dir = up, Fsrc=F4, Fdest = (F7, F8), Plan=(nextStop: F7, endFloor=F8)}
Resume motion: {trip#, dir = up, Fsrc=F4, Fdest = (F7, F8), Plan=(nextStop: F7, endFloor=F8)}
Car moves: F6->F7. F7 == nextStop.
Car stops at F7, serves a new passenger, whose car floor call is added to the plan, if not already on the list.
== > update trip plan: {trip#, dir = up, Fsrc=F4, Fdest = (F8), Plan=(nextStop: F8, endFloor=F8)}
Car moves: F7->F8. F8 == nextStop.
Car stops at F8. F8== endFloor. Trip complete.
Close out trip plan. {trip#, dir=up, Fsrc, Fstopped={F6, F7, F8}, Ttraveled}
Enter the start Idle state. Check the carServRequest queue to begin a new run, or wait.