Homework 6 - Concurrent Elevator Controller

A M story highrise has N elevators in a single “elevator bank”, each able to serve every floor of the building. Contrary from most elevator designs, there are no buttons to choose a destination floor inside the elevator. Instead, the elevator lobby at each floor has M individual “destination floor” buttons, and passengers choose their destination by pushing the appropriate button. Once a button is pushed, a display near the buttons tells the passenger which elevator door to wait by. In this assignment, elevators serve only one passenger at a time.

Your assignment is to create an elevator controller that ensures that all passengers receive service, minimizing the amount of time it takes to serve all of the passengers, and how much CPU time is used to run the controller.

The assignment skeleton code is available here

The code

The homework 6 template is a multithreaded program which simulates passengers and elevator travel. There is one thread per elevator, and one thread per passenger. There is a fixed number of passengers, that embark on a number of trips each. These threads interact with each other through a central elevator controller, which is your responsibility to implement.

The central controller responds to the following two function calls:

```
/* called whenever a passenger pushes a button in the elevator lobby.
   call enter / exit to move passengers into / out of elevator return
   only after the passenger has been delivered to requested floor */

void passenger_request(int passenger, int from_floor, int to_floor, void (*enter)(int, int),
 void(*exit)(int, int));

/* called whenever the elevator needs to know what to do next.
   call move_direction with direction -1 to descend, 1 to ascend.
   must call door_open before letting passengers in, and door_close
   before moving the elevator */

void elevator_ready(int elevator, int at_floor, void(*move_direction)(int, int), void(*door_o
pen)(int), void(*door_close)(int));
```

Performance Expectations

The template code works correctly, but is terribly slow. In part, this is because passengers often “miss” the elevator (don’t get in in time), or miss their floor (stay on the elevator). In part, it is because the elevator stops at every floor, and we only use one elevator.

For a full score, have a fully working solution that uses all available elevators, and finishes each test case in less than 10 seconds with out any busy waiting (aka uses less than 1% CPU).

There is also a **leaderboard** for this assignment. The leaderboard is *only* for style points, which are not included in your final grade. The leaderboard score is computed as 10000 - (execution time for the leaderboard test case in milliseconds).

The autograder runs a slightly modified version of the Makefile given in the handout - all of the test case variables are the same, but it does not print to the screen, which will likely cause execution time and CPU usage to go down by a *very small* amount compared to running on systems1. You should be able to write this code on any modern linux machine. Please see the autograder for specific point values. The entire assignment is graded out of 80 points.

Notes

- Every passenger pushes a button.
- Passengers can't change their minds, and will wait until they are told to enter the elevator, using the `enter()` callback function
- Passengers will stay in the elevator until they are told to leave, using the `exit()` callback function.
- Elevators can hold at most one passenger.
- The program finishes after all passengers have been served, and all functions have returned.

Hints

- Start by making sure your passengers have time to get on and off the elevator. You can do this with condition variables (pthread_cond_t), or barriers (pthread_barrier_t). A barrier-based solution seems easier. Use one barrier to make the passenger wait for the door to open, and another to make the elevator wait for the passenger to enter.
- The template solution can a single set of global variables. You'll probably want to have one set per elevator. Define an elevator struct that holds all your necessary state per elevator, and make an array of such structs.
- Handling multiple elevators should be the last item on your TODO list. An easy way to do it is to randomly decide, for each passenger, which elevator they should use, independent of everything else. Then you can treat each elevator+passengers group separately.
- You could use an extra set of mutexes around the whole `passenger_request()` function to make sure only one user's request is handled by each elevator at one time.

Logging

To add more logging output, try using the log() function instead of printf. `log(loglevel, format_string, parameters)` works just like `fprintf(stderr,...)` except it only prints if the configured log level is higher than the loglevel argument provided. To set the log level to, for example, 8 use the following flag to gcc `-DLOGLEVEL=8`

Good documents on threaded programming

- http://www.ibm.com/developerworks/linux/library/l-posix1.html?S_TACT=105AGX03&S_CMP=EDU (http://www.ibm.com/developerworks/linux/library/l-posix1.html?S_TACT=105AGX03&S_CMP=EDU)
- http://www.ibm.com/developerworks/linux/library/l-posix2/?S_TACT=105AGX03&S_CMP=EDU (http://www.ibm.com/developerworks/linux/library/l-posix2/?S_TACT=105AGX03&S_CMP=EDU)

- http://www.ibm.com/developerworks/linux/library/l-posix3/index.html?S_TACT=105AGX03&S_CMP=EDU (http://www.ibm.com/developerworks/linux/library/l-posix3/index.html?S_TACT=105AGX03&S_CMP=EDU)
- https://computing.llnl.gov/tutorials/pthreads/#ConVarSignal (https://computing.llnl.gov/tutorials/pthreads/#ConVarSignal)
- http://pages.cs.wisc.edu/~travitch/pthreads_primer.html (http://pages.cs.wisc.edu/~travitch/pthreads_primer.html)

Due date: 11:59 PM Wednesday, November 28th.

If you have any questions about the lab requirements or specification, please post on Piazza.

Chris Kanich (https://www.cs.uic.edu/~ckanich/) · UIC Computer Science (https://www.cs.uic.edu/)