Lab 4: Elevator Simulation Using PThreads

IUPUI CSCI 503 – Fall 2013
Assigned: October 21, 2013
Due time: 11:59PM, November 4, 2013

1. Introduction

You will write a pthread program to simulate how multiple elevators move a lot of people up and down in a tall building.

The input to your simulator should be as follows (in the same order):

1. Number of elevators (must be > 0).
2. Number of floors (must be > 0).
3. People arrival time (p seconds per person): After every p seconds, a new person arrives at some floor and waits for an elevator. Any elevator should satisfy that person if multiple elevators arrive at his or her floor simultaneously. Time p > 0.
4. Elevator speed (e seconds per floor): An elevator takes e seconds to move from one floor to the next floor. Time e > 0.
5. Total time to simulate (s seconds): The total time you want to run the simulation. After s seconds, your program should stop and prints out required statistics information. Time s > 0.
6. Random function seed: A random number generator is used to generate a person's from-floor and to-floor. The seed can change the generated random number sequence.

In the simulator, you will need a single thread to generate persons. Every elevator has its own thread to move up and down to serve people waiting on different floors.

2. How the People-generation Thread Works

It is a while loop. For every p seconds, the thread creates a new person and adds it to a global double linked list. The person has a data structure (defined later) and its content has to be filled appropriately.

3. How the Elevator Thread Works

Each elevator is a while loop. It checks the global double linked list: if the list is empty, blocks itself on the elevator's conditional variable until a new person arrives; if not empty, removes the first person from the global list, then elevator moves to the person's from-floor, then moves to the person's destination floor, and finally drops the person. Obviously it is not efficient because this elevator only serves one person at a time!

4. Improvement

A possible solution to the problem is that you can use the disk SCAN (or LOOK) scheduling algorithm you just learned in class. First make sure you understand how the SCAN algorithm works. When an elevator is moving up, it only picks up people who are on their way up. Similarly, when an elevator is moving down, it only picks up people who are on their way down. On each floor it stops, an elevator can pick up as many people as possible (no capacity limit). The elevator should not pick up any person who is going down if the elevator is moving up, and vice versa. Note you might need to search the list to find who can be picked up.

5. Essential Data Structures

```
struct person {
  int id;                    /* 0, 1, 2, 3, … */
  int from_floor, to_floor; /* i.e., from where to where */
  double arrival_time;      /* The time at which the person arrives */
  …                          /* You can add more fields */
};
struct elevator {
    int id;                   /* 0, 1, 2, 3, … */
   int current_floor;         /* Current location of the elevator */
   pthread_mutex_t  lock;
   pthread_cond_v     cv;    /* Used to block the elevator if there is no request */
   struct person   *people;/* All the people inside the elevator */
   …                          /* You can add more fields */
};
struct gv { /* All the global information related to the elevator simulation */
   int num_elevators;
   int num_floors;
   int arrival_time;
   int elevator_speed;
   int simulation_time;
   int random_seed;

   int num_people_started;  /* statistics */

    int num_people_finished; /* statistics */

   pthread_mutex_t *lock;

    …

};
```

These data structures are not complete. You may need to define additional data structures.

6. Simulation Print-out and Statistics Output (required)

- In the people-generation thread:
    - For every new person, print "[time] Person id arrives on floor A, waiting to go to floor B".
- In each elevator thread:
    - Print "[time] Elevator id starts moving from P to Q …"
    - Print "[time] Elevator id arrives at floor Q."
    - Print "[time] Elevator id picks up Person id1, Person id2, and so on."
    - Print "[time] Elevator id drops Person id1, Person id2, and so on"
    - Note: In the basic version, an elevator picks up and drops only one person. In the improved version, it picks up and drops multiple persons.

    When the simulation starts, [time] is equal to 0. So you need to deduct current_time by beginning_time to get [time].

- When the program exits, print out:
    - "Simulation result: x people have started, y people have finished during z seconds."

7. Score Distribution

- Coding style and error checking: **10%** (as described in the "Labs Grading Policy")
- The basic version of simulation: **70%**
- The improved version: **20%**

Again, if there is a "Segmentation Fault" when testing your code, then TA will deduct 20 points from your score. Note: this lab might cause more segmentation faults than previous labs if you are not careful enough.

The total score is 100 points.

8. Deliverable

There should be two subdirectories: one for the basic version, one for the improved version.

In each subdirectory, there should be source code in C, a Makefile, and a README. Make a tar ball for the above two subdirectories and send it to the TA. The tar ball name may be “Lab4_your_name.tar”.