Middle East Technical University   Department of Computer Engineering

HW1'in textini okur musun? Anlatım + şekil + format olarak onu örnek al.

CENG 334

Introduction to Operating Systems

Spring 2019-2020

Homework 2 - Elevator

Due date: 06/04/2020, Monday, 23:59

1 Introduction

The objective of In this assignment, ~~you will be~~ is to gain~~ing~~ hands-on experience on solving **synchronization problems** through simulating an elevator in a multi-threaded program.

For this task, you will use threads. Additionally, you will use a preimplemented C++ Monitor class which uses condition variables and a mutex to synchronize the threads that represents the elevator and people.

Daha sonraya

→ İkinci paragrafta Elevator nasıl çalışır anlat. (HW1'e bak örnek olarak)

2 The Elevator

The elevator

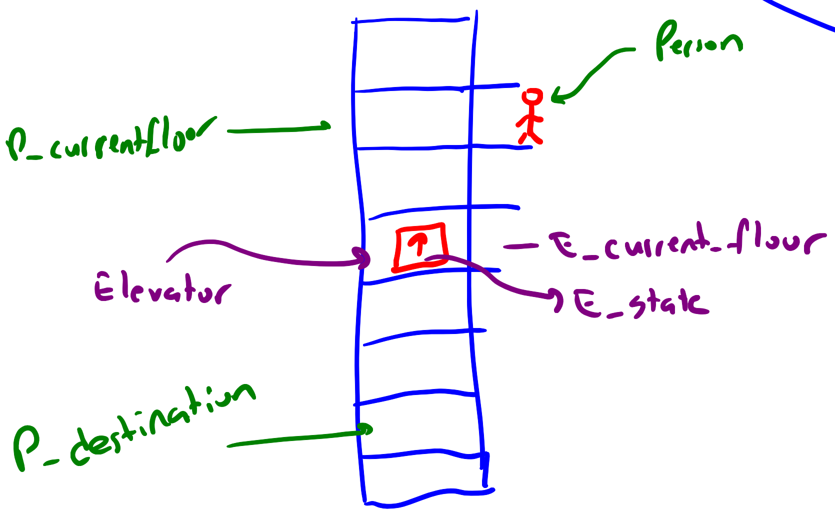
The elevator will always move upwards or downwards. However, the elevator will be idle at the start of the program or if it is waiting for a request to come from a person unless all the people have been served. The elevator always starts from floor 0 and eventually a person requests to enter the elevator and it starts to move upwards. The overall movement of the elevator is given in **Algorithm 1.**

The elevator will have 3 states idle, Moving_up, Moving_down.

− ↑ ↓

capacity — weight

↳ # of persons



```
while (Not all people have been served) do
    Assume that elevator is stationary at floor $x$ ;
    while No one is calling the elevator do
        Wait for requests to come ;
    end
    A request comes from a person thread at floor $y$ ;
    if $x < y$ then
        ~~Start~~ Moving up~~ward~~s after some delay ;
    end
    if $x > y$ then
        ~~Start~~ Moving down~~wards~~ after some delay ;
    else
        ~~Start~~ Moving ~~along~~ the direction of destination of person ;
    end
    if Elevator reaches to the floor of some person that wants to enter then
        if Elevator moving in the same direction that person wants to go and there is room for the
          person then
            Person enters the elevator ;
        end
    end
    if Elevator reaches to a destination that a person inside the elevator wants to go then
        Person leaves the elevator ;
    end
end
```

Algorithm 1: Elevator Movement

A person can enter the elevator if there is enough ~~space~~ capacity(in terms of both total weight and number of people) in the elevator and the elevator moves in the direction of the person assuming that the person is ahead of the elevator in terms of location. For example, assume that elevator is empty, at floor 2, and moving upwards. A person X calls the elevator from floor 1 and wants to go upwards and another person Y calls the elevator from floor 3 and also wants to go upwards. In this case, the elevator will simply ignore the call from person X and add Y's floor to its destinations(if floor 3 is not already a destination). **Note that a person's direction is determined by from where he/she calls the elevator and where does he/she wants to leave.**

If Elevator is not in idle state then it moves up/down one floor with time *TRAVEL_ TIME*. If the floor that elevator reaches is a destination for anyone in the elevator or someone calls the elevator from that destination then Elevator stops at that floor and waits for *IN_ OUT_ TIME*, otherwise elevator keeps moving towards its destination without waiting.

3 Person Threads

?!

BU SAÇMA! Person can call the elevator anytime!

Person threads will call the elevator if the elevator is not moving in the opposite direction. People will have priority levels, weight, initial/final floors, and IDs. They will always dictate the movement of the elevator such that whenever its empty it will wait for a request to come. There are ~~2~~ two types of people that use the elevator: 1) high priority people(with value 2), 2) low priority people(with value 1). When elevator reaches to a floor that is a destination(either some thread is asking to enter or asking to leave or even both) then if there is any person wanting to leave the elevator they leave before anyone can enter. At a floor, high priority people will always have the priority to check whether they can enter the elevator. Low priority people can only enter the elevator if all of the high priority people on that particular floor whether entered the elevator or decided that they can not enter due to capacity constraints.

4 Implementation Specifications

- Each person should be implemented as a separate thread.
- There should be a thread that controls the movement of the elevator and it should run up until all people have been served
- If elevator moves up/down by 1 floor each time you need to ~~write~~ print: "Elevator now moved to floor x" terminated with newline character.
- If elevator reaches to a destination floor after letting people leave/enter you need to print out the status of the destination queue. E.g:
  - If it is not empty ~~write~~ print: "Printing destinations: 8 7 3 " terminated with newline,
  - If it is empty ~~write:~~ "Printing destinations: " terminated with newline
- If a person calls the elevator(remember the conditions to be able to call it) write: "xPerson PID y Called the elevator" where x is either hp or lp depending on priority, and y is the ID number of that person. Again terminated with newline. Afterwards print out the new status of the destination queue with aforementioned way.
- If a person enters the elevator write: "xPerson PID y Entered the elevator" terminated with newline.
- If a person leaves the elevator write: "Person y has left the elevator at floor $current\_floor\_number$"

5 Input Specifications

You will read input from txt files. First line of input will consist of num_floors (N_D), num_people(N_P), weight_capacity(W_C), person_capacity(P_C), $TRAVEL\_TIME(TT)$, $IDLE\_TIME(IT)$ and IN_OUT_TIME (IOT), all time values are in terms of microseconds. Rest of the input will consists of N_P many lines each of which with the form weight_person(W_P), initial_floor(I_F), destination_floor(D_F) and priority(P). A sample input is as follows:

- 10 3 150 2 10000 5000 15000 -> First line, $(N_D, N_P, W_C, P_C, TT, IT, IOT)$
- 50 2 5 2 -> First person, (W_P, I_F, D_F, P)
- 60 1 5 1 -> Second person, (W_P, I_F, D_F, P)
- 50 2 5 1 -> Third person, (W_P, I_F, D_F, P)

Please note that all lines end with a white space followed by a newline character.

6 Homework Specifications

- Your codes must be written in C++, however you can use C's functionalities after making sure that you correctly inherited from Monitor class.
- Your programs will be compiled with g++ and expected to run on inek machines. Make sure that you also give -lpthread flag when compiling your program.
- This homework is mainly on Monitors, do not use Semaphores, Mutexes etc., all the functionality provided to you with Monitor should be enough to complete the homework.

- Although it is possible to have correct results with different outputs due to the concurrent nature of threads, please follow the output instructions carefully, the grading will mostly be black box and make sure you follow these rules.
- There will be penalties for cases like busy waits, deadlocks etc. The punishment of a deadlock will likely be getting 0 from the corresponding input.
- Do not delete anything from the code that is provided to you and in fact, you may need to add some small piece of code into "monitor.h".
- **Using code that is not your own is strictly forbidden and constitutes as cheating. This includes code from your friends, previous homework, or the internet. We have a zero tolerance policy on cheating.**
- Follow the course page on COW for any updates and clarifications. Please ask your questions on COW instead of e-mailing if the question does not contain code or solution.
- **Maybe one more item for grading here...**

7 Submission

Submission will be done via ODTUClass. You will submit a tar file called “hw2.tar.gz” that contain all your source code together with your makefile. You do not need to submit the libraries we have provided. Even if you do, they will be replaced with the originals, so do not make any changes on them. Your tar file should not contain any folders. Your makefile should be able to create a single executable named Elevator and it should be able to run using the following command sequence.

```
> tar -xf hw2.tar.gz
> make all
> ./Elevator inp.txt
```

You can assume that all the executables and library files are present for the required compilation and execution. **If there is a mistake in any of the 3 steps mentioned above, you will lose 10 points.**

8 Sample Execution

\begin{verbatim}

Given input:

Courier

```
10 3 150 2 10000 5000 15000
50 2 5 2
60 1 5 1
50 2 5 1
```

\end{verbatim}

The corresponding output(**not a unique correct output**) is:

```
hpPerson PID 0 Called the elevator
Printing destinations: 2
lpPerson PID 1 Called the elevator
Printing destinations: 1 2
lpPerson PID 2 Called the elevator
Printing destinations: 1 2
Elevator now moved to floor 1
lpPerson PID 1 Entered the elevator
Printing destinations: 2 5
Elevator now moved to floor 2
```

Courier

```
hpPerson PID 0 Entered the elevator
Printing destinations: 5
Elevator now moved to floor 3
Elevator now moved to floor 4
Elevator now moved to floor 5
Person 0 has left the elevator at floor 5
lpPerson PID 2 Called the elevator
Printing destinations: 2
Person 1 has left the elevator at floor 5
Elevator now moved to floor 4
Elevator now moved to floor 3
Elevator now moved to floor 2
lpPerson PID 2 Called the elevator
Printing destinations:
lpPerson PID 2 Entered the elevator
Printing destinations: 5
Elevator now moved to floor 3
Elevator now moved to floor 4
Elevator now moved to floor 5
Person 2 has left the elevator at floor 5
```

Carin