CL Assignment 6

Priyanshu Tirkey (21CS60R11)

October 2021

1 Problem Statement

Come up with a way to simulate the working of multiple lifts in a building (with the lift and person being processes).

```
struct FloorInfo{
    int waitingToGoUp;          /* #people waiting to go up */
    int waitingToGoDown;        /* #people waiting to go down */

    semaphore upArrow;          /* people going up wait on this */
    semaphore downArrow;        /* people going down wait on this */

    /* Added for cases 1)when two lifts going in same direction come at the floor
       or 2)People from two different lifts update the  waitingToGoXX variable */
    */
    semaphore arithmetic;     /* lock for making updates to
                         waitingToGoUp/waitingToGoDown atomic  */
};


struct LiftInfo{
    int id;                     /* unique identifier for a lift */
    int position;               /* lift's current position */
    int direction; // {+1/-1}   /* add this varible allows changing floor */
    int peopleInLift;           /* #people in lift */
    int stops[NFLOOR];          /* #people(in lift) for each stop */
    semaphore stopsem[NFLOOR];  /* #people in lift wait on one of these */

    /* Added for logging purposes */
    int step_cnt;               /* Keeps track of the time units for the lift*/

};
```

2 Three ways of using semaphore needed in order to solve the problem

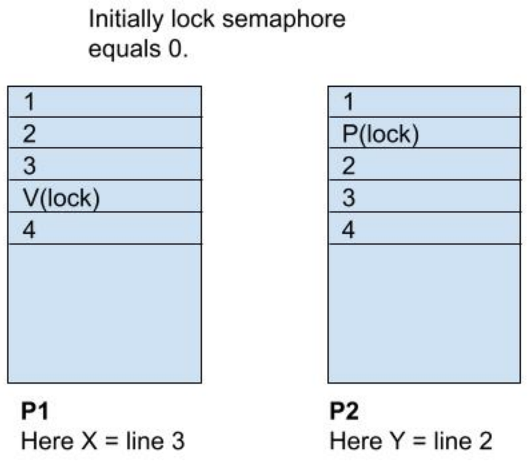
2.1 Synchronization Problem 1

Chained Actions

Say we have two processes P1 and P2 running concurrently.
We want to execute a line(say Y) in P2 only after P1 has reached a point when a line (say X) has been executed.

Solution

We can do it following way.



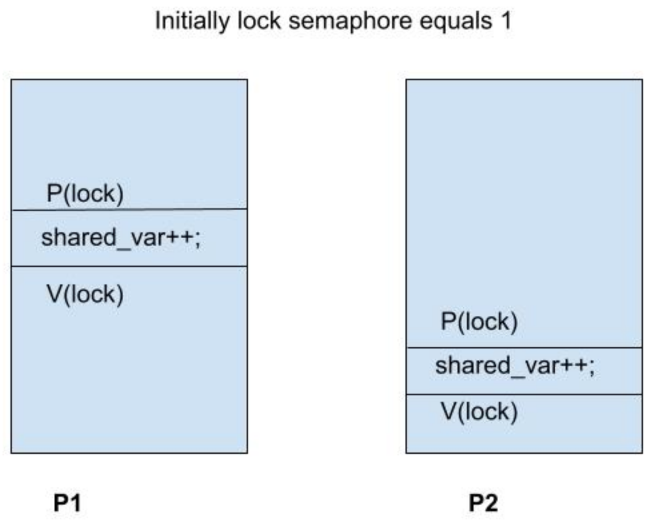
The above way of using semaphore will be useful to when we want a person process to wait for lift door to open.

2.2 Synchronization Problem 2

Racing Processes

Given multiple processes incrementing a shared varible, how can the variable be incremented so that always the most recent value is updated.

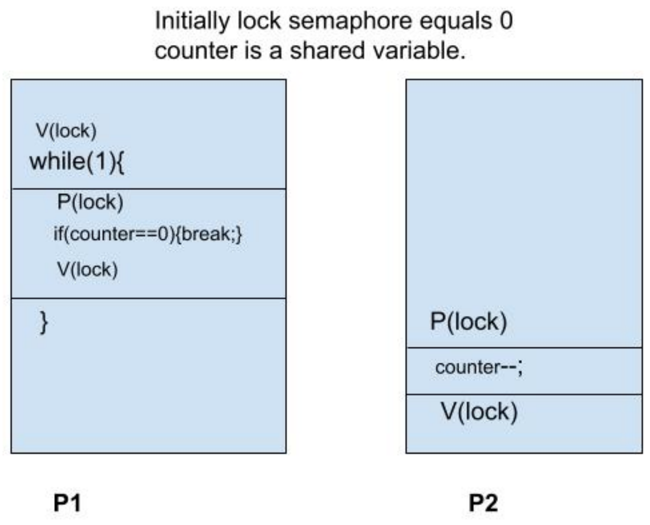
Solution



2.3 Synchronization Problem 3

[A combination of Synchronization Problem 1 and 2]
Detection Problem
Example:
Say lights turn on when at least one person is detected else it remains off.
i.e. does one action in presense and another in absence.

(Finding a solution to this problem will allow coding a behavior where the lift moves to next floor in absence of any person going in/out of the lift for a particular floor.)
Solution



P1 first releases the lock only then P2 can execute after acquiring lock. P1 while reading the variable acquires a lock so other processes cannot modify shared variable. Once the counter becomes 0 the looping stops and other lines after loop is executed. **NOTE** lock is not released.

In our given problem lift can move to next floor again acquire lock and repeat the loop. The person processes for a floor decrements the counter.

3 Lock States at start of the program

$$F.upArrow \leftarrow 0, \quad \forall F \in \text{floors}$$

$$F.downArrow \leftarrow 0, \quad \forall F \in \text{floors}$$

$$L.stopsem[F] \leftarrow 0, \quad \forall F \in \text{ floors and } \forall L \in \text{lifts}$$

$$L.arithmetic[F] \leftarrow 1, \quad \forall F \in \text{ floors and } \forall L \in \text{lifts}$$

4 Action sequences

Consider the case where we have a **person P**, and a **lift L** has arrived at **floor X** and the person want to go to **floor Y** and the direction of the lift is in the direction where the person wants to go.

In [X], X denotes a floor no.
In X i.e. without '[ ]' X denotes the floor object.

(Say the direction is UP)

4.1 Lift Action

Events at lift process with some code for added clarity.
For the lift action code will be slightly different from what is written below(refer to pseudo-code below and synchronization problem 3 to understand the implementation)

- L reaches [X]
- L releases the lock. (L.stopsem[X] semaphore)

```
while(All_Have_Not_Got_Down){}
//i.e loop while F.stops[X] not 0.
```

- L acquires the lock. (L.stopsem[X] semaphore)
- L releases the lock. (X.upArrow semaphore)

```
while(All_Have_Not_Got_Up){}
//i.e loop while X.waitingToGoUp not 0.
//during reading X.arithmetic semaphore must be acquired by L
```

- L acquires the lock (X.upArrow semaphore)
- Lift process sleep for some time to simulate the delay.
- L moves up.

4.2 Person Getting into lift

Events at person process
'!' denotes that its a event occuring at another process(in our case lift process).
'[...]' signifies CPU cycles where P was not executing.

! L reaches [X], stops there and releases the lock. (X.upArrow semaphore)

[...]

- P eventually acquires the lock. (X.upArrow semaphore)
- P acquires arithmetic lock. (X.arithmetic semaphore)
- P increments L.stops[Y]. (Y denotes the floor when he/she wants to go).
- P decrements X.waitingToGoUp.
- P releases arithmetic lock. (X.arithmetic semaphore)
- P releases the lock.(X.upArrow semaphore)

[...]

! L acquires the lock and move (X.upArrow semaphore)

4.3 Person Getting out of lift

! L reaches [X], stops there and releases the lock. (L.stopsem[X] semaphore)

[...]

- P eventually acquires the lock. (L.stopsem[X] semaphore)
- P decrements L.stops[X].
- P releases the lock.(L.stopsem[X] semaphore)

  (After this the process sleep and later wakes up and schedules jounery to a new floor.)

  [...]

! L acquires the lock (L.stops[X] semaphore)

4.4 Person scheduling new Trip(after sleep)

(Schedules to go down from floor Y)

- P acquires arithmetic lock. (Y.arithmetic semaphore)
- P increments Y.waitingToGoDown.
- P releases arithmetic lock. (Y.arithmetic semaphore)

5 Simulation

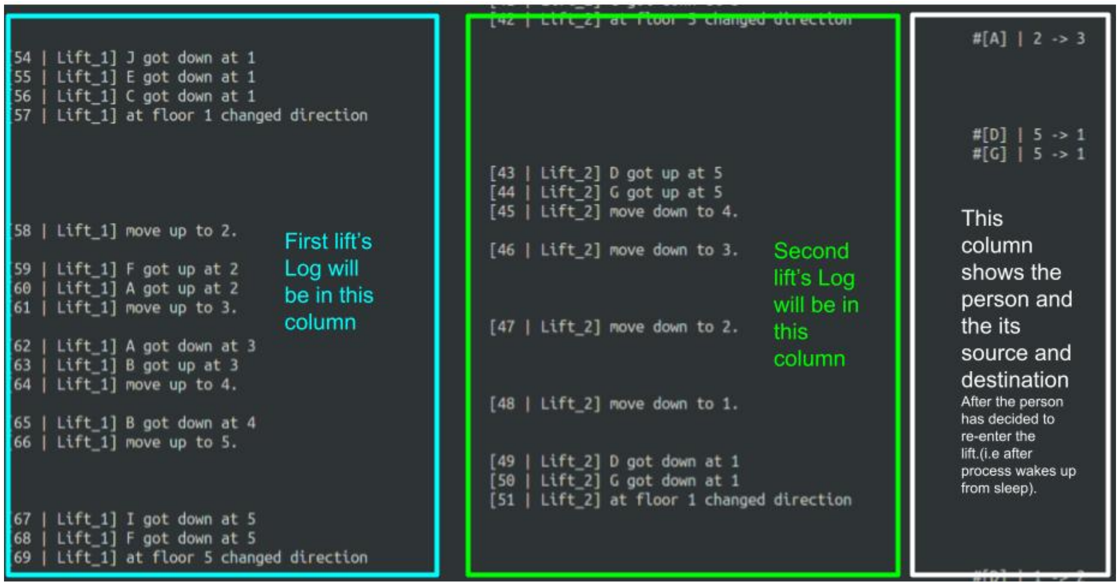
Taken the following:
No. of floors is 5.
No. of lifts is 2.
Each floor has 2 people:
1st Floor has persons A,B.
2nd Floor has persons C,D.
...
5th Floor has persons I,J.

No. of simulatation step is 50.

i.e atleast one of the lifts moves from floor to floor atleast 50 times.
(direction change is also counted as a step)
Once a person sees he/she is on lift that has moved more than 50 times that person doesnt schedule a new trip and quits.
Once all the persons has terminated lift processes start terminating.
Once all the lifts have terminated main process terminates.

```
#define NFLOOR 5
#define MAXPERSON 2 //at per floor
// NFLOOR * MAXPERSON should always <= 26 as PERSON ARE NAMED A,B,C,...
#define NLIFT 2
#define TOT_STEPS 50
```

Output1 (Shows lift movement logs)



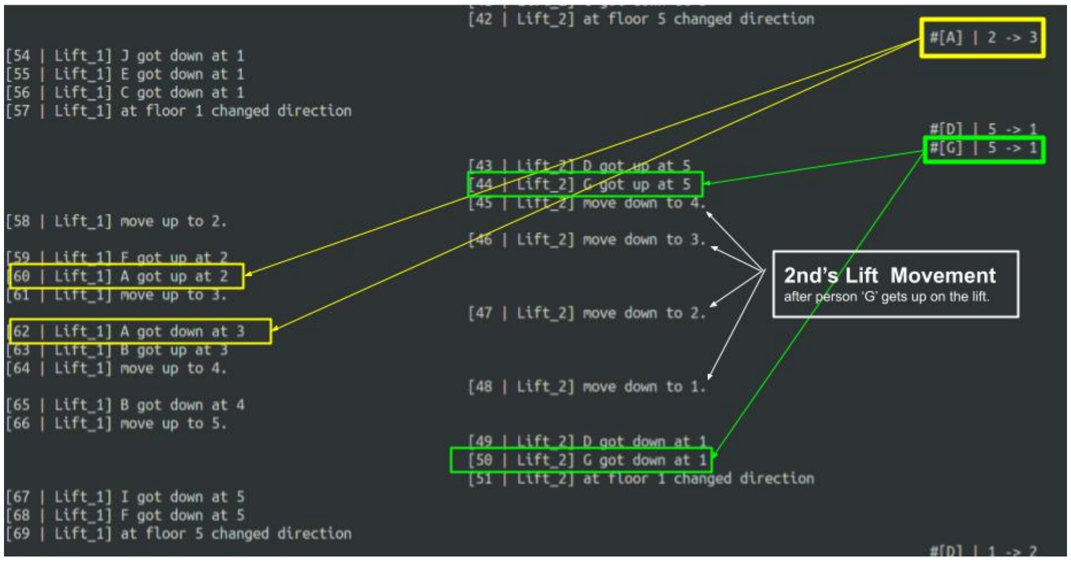
```
[54 | Lift_1] J got down at 1
[55 | Lift_1] E got down at 1
[56 | Lift_1] C got down at 1
[57 | Lift_1] at floor 1 changed direction

[58 | Lift_1] move up to 2.

[59 | Lift_1] F got up at 2
[60 | Lift_1] A got up at 2
[61 | Lift_1] move up to 3.

[62 | Lift_1] A got down at 3
[63 | Lift_1] B got up at 3
[64 | Lift_1] move up to 4.

[65 | Lift_1] B got down at 4
[66 | Lift_1] move up to 5.

[67 | Lift_1] I got down at 5
[68 | Lift_1] F got down at 5
[69 | Lift_1] at floor 5 changed direction
```

```
[42 | Lift_2] at floor 5 changed direction

[43 | Lift_2] D got up at 5
[44 | Lift_2] G got up at 5
[45 | Lift_2] move down to 4.

[46 | Lift_2] move down to 3.

[47 | Lift_2] move down to 2.

[48 | Lift_2] move down to 1.

[49 | Lift_2] D got down at 1
[50 | Lift_2] G got down at 1
[51 | Lift_2] at floor 1 changed direction
```

```
#[A] | 2 -> 3

#[D] | 5 -> 1
#[G] | 5 -> 1
```

First lift's Log will be in this column
Second lift's Log will be in this column
This column shows the person and the its source and destination
After the person has decided to re-enter the lift.(i.e after process wakes up from sleep).

Output2 (Shows person movement logs)



```
                                                         [42 | Lift_2] at floor 5 changed direction
                                                                                                        #[A] | 2 -> 3
[54 | Lift_1] J got down at 1
[55 | Lift_1] E got down at 1
[56 | Lift_1] C got down at 1
[57 | Lift_1] at floor 1 changed direction
                                                                                                        #[D] | 5 -> 1
                                                                                                        #[G] | 5 -> 1
                                                         [43 | Lift_2] D got up at 5
                                                         [44 | Lift_2] G got up at 5
                                                         [45 | Lift_2] move down to 4.
[58 | Lift_1] move up to 2.
                                                         [46 | Lift_2] move down to 3.
[59 | Lift_1] F got up at 2
[60 | Lift_1] A got up at 2
[61 | Lift_1] move up to 3.
                                                         [47 | Lift_2] move down to 2.
[62 | Lift_1] A got down at 3
[63 | Lift_1] B got up at 3
[64 | Lift_1] move up to 4.
                                                         [48 | Lift_2] move down to 1.
[65 | Lift_1] B got down at 4
[66 | Lift_1] move up to 5.
                                                         [49 | Lift_2] D got down at 1
                                                         [50 | Lift_2] G got down at 1
                                                         [51 | Lift_2] at floor 1 changed direction
[67 | Lift_1] I got down at 5
[68 | Lift_1] F got down at 5
[69 | Lift_1] at floor 5 changed direction
                                                                                                        #[D] | 1 -> 2
```