Operating Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder

Module: CO-562
Date: 2022-09-27

OS 2022 Class Problem Sheet #3

Problem 3.1: *unisex bathroom problem* (0 points)

Allen B. Downey describes the "Unisex Bathroom Problem" in his book "The Little Book of Semaphores". He states the synchronization problem as follows:

> I wrote this problem when a friend of mine left her position teaching physics at Colby College and took a job at Xerox.
>
> She was working in a cubicle in the basement of a concrete monolith, and the nearest women's bathroom was two floors up. She proposed to the Uberboss that they convert the men's bathroom on her floor to a unisex bathroom, sort of like on Ally McBeal.
>
> The Uberboss agreed, provided that the following synchronization constraints can be maintained:
>
> - There cannot be men and women in the bathroom at the same time.
> - There should never be more than three employees squandering company time in the bathroom.
>
> Of course the solution should avoid deadlock. For now, though, don't worry about starvation. You may assume that the bathroom is equipped with all the semaphores you need.

Develop a solution in pseudocode using semaphores.

Solution:

The problem has two wait conditions: (1) If the room is full, then employees have to wait. (2) If the room is occupied by the wrong gender, then employees have to wait. The first wait condition can be easily solved since it matches the multiplex problem. The second wait condition is more difficult to handle. One option is to think about an empty bathroom having a door open so whoever arrives first can walk in. Once the first person walks in, the door closes in such a way that only employees of the same gender can pass.

```
semaphore_t empty = 1;      /* open if the bathroom is empty */

typedef struct gender {
    int count;              /* count of people of this gender */
    semaphore_t mutex;      /* protecting the shared count */
    semaphore_t multiplex;  /* limit of # of people of this gender */
} gender_t;

shared gender_t female = { count = 0, mutex = 1, multiplex = 3 };
shared gender_t male   = { count = 0, mutex = 1, multiplex = 3 };

void women() {
    go(&female);
}

void men() {
    go(&male);
}

void go(gender_t *g) {
    down(&g->mutex);
```

```
    g->count++;
    if (g->count == 1) {
        down(&empty)
    }
    up(&g->mutex);
    down(&g->multiplex);
    // enter and use bathroom
    up(&g->multiplex);
    down(&g->mutex);
    g->count--;
    if (g->count == 0) {
        up(&empty)l
    }
    up(g->mutex);
}
```

Let us walk through an example scenario to understand how this solution works. We start with an empty bathroom.

- When the first female f_1 arrives, she goes into the female critical section, increments the female counter to 1, closes the door, leaves the critical section, proceeds into the multiplex section and into the bathroom.
- When the second female f_2 arrives, she goes into the female critical section, increments the female counter to 2, leaves the critical section, proceeds into the multiplex section and into the bathroom.
- When the third female f_3 arrives, she goes into the female critical section, increments the female counter to 3, leaves the critical section, proceeds into the multiplex section and into the bathroom. The bathroom has now reached its limit.
- When the fourth female f_4 arrives, she goes into the female critical section, increments the female counter to 4, leaves the critical section, proceeds to the multiplex section and waits on the multiplex semaphore.
- When the first male m_1 arrives, he goes into the male critical section, increments the male counter to 1, tries to close the door but gets blocked because the room is not empty anymore. Note that m_1 is getting blocked inside the male critical section.
- When the second male m_2 arrives, he gets blocked immediately since m_1 is blocked inside the male critical section.

  At this point, we have f_1, f_2, and f_3 in the bathroom, f_4 is waiting on the female multiplex semaphore, m_1 is waiting for the bathroom to become empty, and m_2 is waiting to get into the male critical section. (Additional arriving females would queue up at the female multiplex semaphore and additional arriving males would queue up at the initial male critical section.)
- When f_2 leaves the bathroom, she lets the next female enter the bathroom (if any), decrements the number of females to 3 and leaves. The female f_4 waiting at the multiplex semaphore now walks into the bathroom.
- When f_3 leaves the bathroom, she lets the next female enter the bathroom (if any), decrements the number of females to 2 and leaves.
- When f_0 leaves the bathroom, she lets the next female enter the bathroom (if any), decrements the number of females to 1 and leaves.
- When f_4 leaves the bathroom, she lets the next female enter the bathroom (if any), decrements the number of females to 0 and since she is the last female to leave, she opens the door again before she leaves.
- The male m_0 can now close the door, leave the male critical section, proceed to the multiplex section and into the bathroom.
- The male m_1 now enters the male critical section, increments the male counter to 2, leaves the critical section, proceeds to the multiplex section and into the bathroom.

- When m_1 leaves the bathroom, he lets the next male enter the bathroom (if any), decrements the number of males to 1 and leaves.
- When m_2 leaves the bathroom, he lets the next male enter the bathroom (if any), decrements the number of males to 0 and since he is the last male to leave, he opens the door again before he leaves.

The above solution can suffer from starvation. This can be fixed by introducing another gate, a turnstile, that protects the first person waiting outside.

```
semaphore_t turnstile = 1;

void go(gender_t *g) {
    down(&turnstile);
    down(&g->mutex);
    g->count++;
    if (g->count == 1) {
        down(&empty)
    }
    up(&g->mutex);
    up(&turnstile);
    down(&g->multiplex);
    // enter and use bathroom
    up(&g->multiplex);
    down(&g->mutex);
    g->count--;
    if (g->count == 0) {
        up(&empty)l
    }
    up(g->mutex);
}
```