Hilzer's Barbershop with SCOOP

Riccardo Tommasini and Marco Balduini

June 16, 2016

1 Problem Description

The Hilzer's barbershop[1] consists of a shop with a waiting room of a given capacity (waiting-room-size); a sofa with a give capacity (sofa-size); a variable number of chairs and a corresponding number of barbers; a cashdesk that the barbers use to accept payments.

A customer enters the barbershop if it has room, i.e. the total amount of customers inside is lower than the limit imposed by fire laws.

Once inside, the customer tries to take a sit on the sofa, if the sofa is at its dimension limit, he waits in the standing room.

When a barber is free, the CUSTOMER that has been on the sofa the longest is served (FIFO) and the customer who was waiting standing the longest sits (FIFO).

After the hair cut the customer can pay a free barber (anyone, not only the one who cut his hair). Since there is only one cash desk the payment happens one customer at time.

[1] Little Green Book of Semaphores, 5.3, http://www.greenteapress.com/semaphores/downey08semaphores.pdf

Notably, the barbers divide their time among cutting hair, accepting payment, and sleeping in their chair waiting for a customer.

2 Analysis

On the problem specification presented, we can elicit the following requirements: (R1) the shop has to have a size; (R2) the waiting room has to have a size; (R3) the sofa has to have a size; (R5) the waiting room is a FIFO queue; (R5) the sofa is a FIFO queue; (R6) the cashdesk is a queue with no specific ordering constraints and (R6) only one customer can pay at time.

To ensure synchronization FIFO ordering in the standing waiting room of the SHOP and SOFA processing we implemented a Ticketing mechanism that works as follows.

Each customer has a ticket that specifies if he is allowed to enter a certain FIFO queue.

When a customer enters the shop, a ticket is assigned to him.

The SOFA is aware about which are the customer's tickets allowed to sit on it, so it can control if a customer can sit or has to wait in the room.

Once a customer successfully sits, a new ticket is assigned to him, so that the FIFO ordering to sit on the SOFA is independent from the FIFO ordering to sit on barber chair.

The BARBER_LIST is aware about which are the customer's tickets allowed to sit on barbers' chairs, so it can control if a customer can sit or has to wait on the SOFA.

Once a customer successfully sits on a barber's chair, he gets an haircut and then has to pay.

No FIFO ordering is guaranteed for the CASHDESK queue, as specified in the problem.

Since we are dealing with concurrent programming, we have to face the following issues to properly design the problem:

I1 Customers are a living object in our system.

I2 Customers creation must be random as well as their interaction with the system

I3 Barbers and Customers have directly interact so that the latter can cut the former's hair and/or pay.

I4 The way that FIFO queues are implemented characterizes the final execution model.

3 Design

Our problem design is Customer-centric, i.e. the customer is the only living actor in the system of separate objects and contains a direct reference to all the other involved classes.

We modeled the following classes:

- BARBERSHOP: it keeps track of how many CUSTOMERs are in the shop and how many are in the waiting room. It assigns tickets according to the mechanism explained in Section 3.3 and it makes CUSTOMERs sit on the sofa ensuring **FIFO** ordering.
- SOFA: it keeps track of how many CUSTOMERs are waiting sat. It assigns tickets to customers according to the mechanism explained in Section3.3 and it makes CUSTOMERs sit on barber's chair ensuring their **FIFO** ordering.
- CUSTOMER: it is the agent (guides the execution) and its life-cycle is explained in the Section 3.1.
- BARBER: it cuts customer hairs and accepts payments.
- BARBER_LIST: it is a proxy for a **QUEUE** of *separate* BARBER that allows them to be connected to the CUSTOMERs in a synchronized way.
- CASHDESK: the access to it is required to a CUSTOMER in order to pay a BARBER.

Figure 1 shows an UML representation of the classes.

All the classes are declared separate in our main class. The CUSTOMER has separate references to BARBERSHOP, SOFA, BARBER_LIST and CASHDESK.

All the CUSTOMERs wait for the BARBERSHOP to open by the means of a SCOOP waiting condition. Once the shop is open, CUSTOMERs can enter, ticket is assigned to those that successfully enter and they start its life-cycle; those who could not enter will sleep and attempt again after a little while.

3.1 Customer Life-Cycle

The CUSTOMER continuously cycles over the following actions, until it has reached a zero needs of haircuts:

(1) it waits for the shop to open; (2) it attempts to enter the shop, if there is room inside it enters and get a ticket for the sofa; (3) it attempts to sit on the sofa, if not possible, i.e. his ticket is not allowed, it waits in the standing room (waiting condition); (4) once sofa has free sit and its ticket is allowed, it sit on the sofa and get a new ticket to sit on a barber's chair. It will wait on the sofa until (5) a free barber's chair is available and its ticket is also allowed, so it gets a free barber; (6) it will get an haircut and the it will free the barber (7) it will wait until the cash desk is free to pay, i.e. it attempts to get a free barber for paying, then it will free the barber and (7) it leaves the shop.

3.2 Solving the issues

SI1 The CUSTOMER is a "living" class in our system.

SI2 The SHOP opens and acts like a barrier for the CUSTOMERs to start "living".

SI3 The CUSTOMER has a reference to a BARBER_LIST that allows to access the shared resources, BARBERs, in a synchronized fashion.

SI4 Our solution ensures FIFO ordering to sit on the sofa and to get an haircut by using a ticketing mechanism.

3.3 Ticketing Mechanism

The ticketing mechanism ensures the FIFO ordering on top of SCOOP waiting conditions as follows:

Tickets are assigned to customers before they try to access a resource; they monotonically increase within the resource, so they keep tracks of CUSTOMERs arrival at a given resource. A CUSTOMER, during its life-cycle, receives two tickets one as soon as it enters and one as soon as it sits on the sofa.

The access to such a shared resource is regulated by a require clause that checks if a particular customer's ticket is in the list of the valid ones for that resource. This clause is a waiting condition since the customers are separate objects and it will eventually evaluate to true. When a customer actually accesses the resource, its ticket is updated so a new one is now valid of access.

To allow multiple CUSTOMERs to access a resource at the same time, for instance because we have more than one BARBERS, the number of allowed tickets must be greater than one. We implemented it as a LIST, whose size corresponds to the number of concurrent access allowed to that resource.

4 Conclusion

The results and the code are available in an online repository [2]. The hardest part of the work was to understand how to handle the waiting conditions and tame with the FIFO ordering.

Moreover, we initially thought as the problem as a producer-consumer one, where the barbers were consuming the queue of customer waiting in the sofa.

We still think that such a solution is possible, but we failed in our attempt to realize it. The blocking issue we found regards how to keep the customer waiting while the barbers are completing their life-cycles.

[2] https://github.com/riccardotommasini/hilzerbs

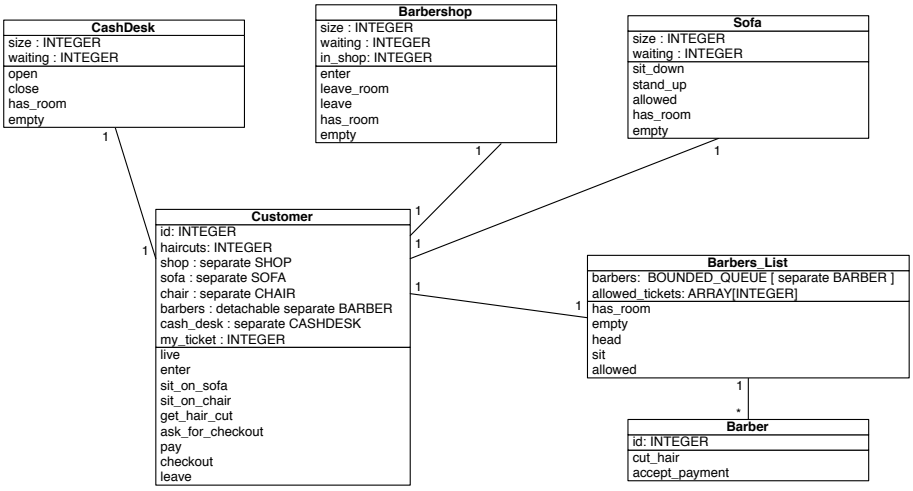

Figure 1: UML schema of proposed solution