Contents

Implementing FIFO

First, we create a class named FIFO. This class has 4 attributes named pageFault, seats, capacity and pointer. As the name suggests, pageFault is showing how many page faults we've had until now. Seats is a list of seats in the restaurant. capacity is showing the total number of seats that our restaurant has and pointer is showing which customer should be thrown out next (its actually shown the first customer that came in among the ones that are already in.)

Add Function

We first check if the customer is already inside the restaurant via the "contains" function, if the customer was already inside, we had a page hit and can return. Otherwise, if the restaurant was filled, we remove the guy who is sitting on the seat that our pointer is pointing to. Then, we give that seat to the new customer and update our pointer to pointer+1. But if the restaurant already had empty seats, we just give the first empty seat we have to the new customer.

Implementing LRU

We build another class for LRU too, but this time, it extends the features of FIFO. We also define a stack which will help us in determining the LRU customer. How? Each time a customer orders we put it on top of the stack, this way the LRU customer is always in the bottom of stack.

Changing the add Function

In this algorithm, we don't necessarily throw out the first guy that came in, we throw out the first customer in our stack. As I said, first we must check if the customer is already inside the restaurant or not. If yes, we remove it from the stack and add it to the top. If no, we remove the first one in the stack and find theirs seat number using the findLRU function, then we throw out that guy and give his seat to the new customer and put him on top of the stack.

FindLRU Function

It will restore the first customer in the stack and find his seat number and returns it.

Implementing 2nd Chance

Second chance algorithm also needs an array of frequencies thus we'll make a SecondChance class which extends FIFO and make an inner class called Pair which has a customer ID and a Boolean which determines if this customer has a second chance or not.

This class will use the parameter pointer, but this time it determines the seat number where we should start checking for finding the victim the next time in contrast to FIFO.

To add someone, we first check if he is already in or not. If yes, we will give a second chance to him by setting his chance to true. If he is new and we had empty seats, we'll give the empty seat. But when we don't have an empty seat already, we throw someone out via the findVictiom function and give their seat to the new customer.

FindVictim Function

We start from the seat where our pointer is pointing to, we check the frequencies, if it is zero, we've found our victim and set our pointer to the next seat. If the frequency is 1, it will survive this time but loses its chance, thus the frequency will be set to zero and jump from this seat. We repeat this procedure until we've found some seat without a second chance.

Note that first we iterate from pointer to the end and all the seats in this interval had second chance we start iterating from the first seat to the pointer, and again if all the seats in this interval had second chance too, we return pointer and update pointer to pointer+1.

ServerHandler Thread

We defined a class named ServerHandler which is in charge of receiving messages from the server. We can make an object of this class by creating a socket variable and then this object will use a DataInpurStream to get messages via that same socket from the sever. This thread and the AlgoHandler thread are very similar to producer and consumer and we already know how to handle that problem, thus we make a PC class which has a list and two functions:

1. Produce: If the list has an empty spot, we put the item we wanted to produce in that spot, otherwise we must wait for someone to consumes one of our products.
2. Consume: If the list is empty, we should wait for someone to produce something, otherwise we take the first item that had been produced.

Our server handler thread will call produce each time it gets a message from server and if the message was 0, it will interrupt the AlgoHandler thread forcing it to stop.

AlgoHandler Thread

In the beginning, it will wait for the first produced product which is the total number of seats that the server has sent.

Then it will make objects of type FIFO, LRU and 2nd Chance based on ‘n’ and after that it will enter an infinite loop of consuming and adding the consumed product(which is basically the customer IDs sent by server) to each object.

```
Customer ID: 13
        ----FIFO----
    Seats: 13
    First: 13
    PAGE FAULT: 1

        ----LRU----
    Seats: 13
    Stack: 13
    PAGE FAULT: 1

        ----2nd Chance----
    Seats:  13
    Chance: N
    Pointer: 0
    PAGE FAULT: 1

Customer ID: 24
        ----FIFO----
    Seats: 13 24
    First: 13
    PAGE FAULT: 2

        ----LRU----
    Seats: 13 24
    Stack: 13 24
    PAGE FAULT: 2

        ----2nd Chance----
    Seats:  13 24
    Chance: N  N
    Pointer: 0
    PAGE FAULT: 2
```

```
Customer ID: 24
        ----FIFO----
    Seats: 13 24 26
    First: 13
    PAGE FAULT: 3

        ----LRU----
    Seats: 13 24 26
    Stack: 26 13 24
    PAGE FAULT: 3

        ----2nd Chance----
    Seats:  13 24 26
    Chance: Y  Y  N
    Pointer: 0
    PAGE FAULT: 3

Customer ID: 91
        ----FIFO----
    Seats: 91 24 26
    First: 24
    PAGE FAULT: 4

        ----LRU----
    Seats: 13 24 91
    Stack: 13 24 91
    PAGE FAULT: 4

        ----2nd Chance----
    Seats:  13 24 91
    Chance: N  N  N
    Pointer: 0
    PAGE FAULT: 4
```

```
Customer ID: 26
        ----FIFO----
    Seats: 13 24 26
    First: 13
    PAGE FAULT: 3

        ----LRU----
    Seats: 13 24 26
    Stack: 13 24 26
    PAGE FAULT: 3

        ----2nd Chance----
    Seats:  13 24 26
    Chance: N  N  N
    Pointer: 0
    PAGE FAULT: 3

Customer ID: 13
        ----FIFO----
    Seats: 13 24 26
    First: 13
    PAGE FAULT: 3

        ----LRU----
    Seats: 13 24 26
    Stack: 24 26 13
    PAGE FAULT: 3

        ----2nd Chance----
    Seats:  13 24 26
    Chance: Y  N  N
    Pointer: 0
    PAGE FAULT: 3
```

```
Customer ID: 91
        ----FIFO----
    Seats: 91 24 26
    First: 24
    PAGE FAULT: 4

        ----LRU----
    Seats: 13 24 91
    Stack: 13 24 91
    PAGE FAULT: 4

        ----2nd Chance----
    Seats:  13 24 91
    Chance: N  N  Y
    Pointer: 0
    PAGE FAULT: 4

Customer ID: 39
        ----FIFO----
    Seats: 91 39 26
    First: 26
    PAGE FAULT: 5

        ----LRU----
    Seats: 39 24 91
    Stack: 24 91 39
    PAGE FAULT: 5

        ----2nd Chance----
    Seats:  39 24 91
    Chance: N  N  Y
    Pointer: 1
    PAGE FAULT: 5
```