**Shop Class Implementation Document**

**Overview**

Shop class extends the sleeping-barber problem to a multiple sleeping barbers’ problem where many customers visit a barbershop and receive a haircut service from any one of the available barbers in the shop.

It simulates this scenario: A customer comes into a barber shop, which has a waiting room with m (default 3) chairs and has n (default 1) barber. If the barbers are all busy and the chairs are all occupied, customer leaves, or the customer sits in one free chair to wait for haircut. After finishing haircut, the customer pays and leave. The free barber will service the next customer if there is, or he will sleep.

**Private**

1. Struct
2. Barber struct is to define a barber, the struct as follows:

struct Barber {

int id; // barber’s id

pthread\_cond\_t barberCond; // define barber condition when waiting for customer

pthread\_cond\_t paid\_cond; // define paid condition when waiting for customer paying

int myCustomer = -1; // no customer by default

};

1. Customer struct is to define a barber, the struct as follows:

struct Customer {

int id; // customer id

pthread\_cond\_t customerCond; // define customer condition when waiting for a barber

customerState state = WAIT; // waiting state by default

int myBarber = -1; // no barber by default

};

1. Data

int nBarbers; // the barbers number of the shop

int nChairs; // the waiting chairs number of the shop

enum customerState {WAIT, CHAIR, LEAVING}; // customer three state

Barber \*barbers; // array of barber objects

map<int, Customer> customers; // container for customer objects

queue<int> waitingCustomers; // waitingCustomers queue

queue<int> sleepingBarbers; // sleepingBarbers queue

pthread\_mutex\_t mutex1; // public mutex

1. Private function

Barber\* getBarber(int barberId); // let others get private barber of Shop class

**Public**

1. **Shop(int num\_barbers, int num\_chairs)**

Initializes a Shop object with nBarbers and nChairs.

1. **Shop()**

Initializes a Shop object with default number of barbers and chairs: 1 barber and 3 chairs.

1. ~**Shop()**

Free space occupied by Shop object.

1. **int visitShop(int id)**

Is called by a customer thread to visit the shop. The flow should be as follows:

Enter the critical section.

If initializing chair number is 0

{

If no barber free:

customer leaves;

Print “customer[i]: leaves the shop because of 0 chairs and no available barbers.”

Increment nDropsOff.

return -1.

}else{

If the initializing chair number is not 0, but no chair is free:

customer leaves;

Print “customer[i]: leaves the shop because of no available waiting chairs.”

Increment nDropsOff.

return -1.

}

if all barbers are busy

{

Take a waiting chair (Push the customer in a waiting queue).

Print “customer[i]: takes a waiting chair. # waiting seats available = i”.  // waiting seats available =nChairs-waitingCustomers.size()

Wait for a barber to wake me up, but if I can’t be assigned the barber, sleep again.

If I was assigned a barker, I wakeup;

}

Get my barber whose id is barber\_id.

Print “customer[i]: moves to a service chair[i], # waiting seats available = i”

// waiting seats available =nChairs-waitingCustomers.size()

Change customer state into “CHAIR”

Send “CHAIR” signal to the chair’s owner barber

Leave the critical section.

Return barber\_id

1. **void leaveShop(int customer\_id, int barber\_id)**

Is called by a customer thread to visit the shop. The flow should be as follows:

Enter the critical section.

Print “**customer[i]: wait for barber[i] to be done with hair-cut.**.”

While **barber\_id**is cutting my hair,

Wait.

Print “**customer[i]: pay to barber[i]**”

Send payment signal to my barber;

Print “**customer[i]: says good-bye to barber[i]”** // finish transaction

Leave the critical section.

1. **void helloCustomer(int id)**

Is called by a barber thread, and the flow is as follows:

Enter the critical section.

If I have no customer and all the waiting chairs are empty

{

Print “**barber [i]: sleeps because of no customers**”

Wait until a customer sitting in my service chair to wake me up.

}

Wait until my service chair status occupied by a customer in order to synchronization with customer thread

Print “**barber [i]: starts a hair-cut service for customer[i**]”

Leave the critical section.

1. **byeCustomer(int id)**

Is called by a barber thread, and the flow isas follows:

Enter the critical section.

Print “**barber [i]: says he's done with a hair-cut service for customer[i]**.”

Wakes up my customer who are taking my service.

Printf " **barber [i]: waiting to be paid by customer[i]”;**

Wait for my customer to pay condition to wake up before I take a new one

Printf " **barber [i]: paid by customer[i]** ”;

Print “**barber [i]: calls in another customer.** ”

Wakes up another customer who is waiting on a waiting chair.

Leave the critical section.