University of Washington Bothell
CSS503: System Programming
Program 2
The Sleeping Barbers Problem

1. Purpose

In this programming assignment we will extend the original 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.

2. Sleeping-Barber Problem

A barbershop consists of a waiting room with n waiting chairs and a barber room with one barber chair. If there are no customers to be served, the barber goes to sleep. If a customer enters the barbershop and all the waiting chairs are occupied, then the customer leaves the shop. If the barber is busy but waiting chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber and gets served.

3. The Extended Sleeping-Barbers Problem

A barbershop consists of a waiting room with n waiting chairs and a barber room with m barber chairs. If there are no customers to be served, all the barbers go to sleep. If a customer enters the barbershop and all chairs (including both, waiting and barber chairs) are occupied, then the customer leaves the shop.

If all the barbers are busy but chairs are available, then the customer sits in one of the free *waiting* chairs. If the barbers are asleep, the customer wakes up one of the barbers.

4. Files

A few files have been provided on canvas to help you get started with this program.
Namely:

- driver.cpp : This code is provided as suggested implementation to get you started, it is by no means working code ☺.
- Shop_org.cpp / Shop_org.h : Shop simulation implementation file example.

5. Main Program: driver.cpp

The driver.cpp is a driver program that tests your sleeping-barbers problem. It creates the shop, the barbers and the clients. It performs the following actions:

1. Receives the following command line arguments

argv[1]	`nBarbers`	The number of barbers working in your barbershop
argv[2]	`nChairs`	The number of chairs available for customers to wait in
argv[3]	`nCustomers`	The number of customers who need a haircut service
argv[4]	`serviceTime`	Each barber's service time (in µ seconds).

2. Instantiates a shop which is an object from the Shop class that you will implement.
3. Spawns the `nBarbers` number of barber threads. Each individual thread gets passed a pointer to the shop object (shared), the unique identifier (i.e. 0 ~ `nBarbers` – 1), and `serviceTime`.
4. Loops spawning nCustomers, waiting a random interval in µ seconds, (i.e., **`usleep(rand() % 1000)`**) between each new customer being spawned. All customer threads are passed in a pointer to the shop object and the identifier (i.e., 1 ~ `nCustomers`).
5. Waits until all the customer threads are service and terminated.
6. Terminates all the barber threads.

5. Barber Thread

As noted, the driver.cpp creates barber threads in their main() function. Each barber thread is implemented as follows. You will need to update the driver.cpp code.

```
// the barber thread function
void *barber( void *arg )
{
  // extract parameters
  ThreadParam &param = *(ThreadParam *)arg;
  Shop &shop = *(param.shop);
  int id = param.id;
  int serviceTime = param.serviceTime;
  delete &param;

  // keep working until being terminated by the main
  while( true )
  {
    shop.helloCustomer( id ); // pick up a new customer
    usleep( serviceTime );
    shop.byeCustomer( id ); // release the customer
  }
}
```

`ThreadParam` is an object which is used to pass more than one argument to the thread function. Below is a ***suggested*** sample implementation of `ThreadParam`. You don't need to implement it exactly this way -- as long as your code produces desired results you can use your own approach.

```
// a set of parameters to be passed to each thread
class ThreadParam
{
public:
  ThreadParam( Shop *shop, int id, int serviceTime ) :
   shop( shop ), id( id ), serviceTime( serviceTime ) { };
  Shop *shop;       // a pointer to the Shop object
  int id;          // a thread identifier
  int serviceTime;   // service time (usec) for barber; 0 for customer
};
```

6. Customer Thread

Customer threads are also created in the main( ) function of driver.cpp. Each customer thread is implemented as follows. You will need to update the driver.cpp code.

```
void *customer( void *arg )
{
  ThreadParam &param = *(ThreadParam *)arg;
  Shop &shop = *(param.shop);
  int id = param.id;
  delete &param;

  // if assigned to barber i then wait for service to finish
  // -1 means did not get barber
  int barber = -1;
  if ((barber = shop.visitShop(id)) != -1)
  {
    shop.leaveShop( id, barber ); // wait until my service is finished
  }
}
```

7. Shop Class

The Shop class needs to be designed to work with multiple barbers. The class `Shop_org` provided class file is designed for one barber, so some things will have to change to support multiple barbers.

The class updated for multiple barbers will look as follows:

```
#ifndef _SHOP_H_
#define _SHOP_H_
#include <pthread.h>
#include <queue>
using namespace std;

#define DEFAULT_CHAIRS 3   // the default number of chairs for waiting = 3
#define DEFAULT_BARBERS 1  // the default number of barbers = 1

class Shop
{
 public:
  Shop( int nBarbers, int nChairs );
  Shop( );

  int visitShop( int id ); // return barber ID or -1 (not served)
  void leaveShop( int customerId, int barberId );
  void helloCustomer( int id );
  void byeCustomer( int id );
  int nDropsOff; // the number of customers dropped off

private:
 string int2string( int i );
 void print( int person, string message );
};
```

```
#endif
```

Note that this is only a "starter" provided for your convenience. Among other things, you will need to add some private variables such as `pthread_mutex_t` and `pthread_cond_t` to implement this Shop class as a monitor. Also, since there can be multiple barbers working on multiple customers at the same time, you may consider using arrays or vectors of condition variables, e.g., to ensure that a barber correctly signals the right customer when the haircut is finished.

You may use any of the methods you want from shop_org. You will find the int2string() and print() private methods quite useful and you should use them in your shop implementation.

If you call `print(5, "was served")`, it will print out `"customer[5] was served"`. If you call `print(-2, "cuts a customer's hair.")`, it will print out `"barber[2] cuts a customer's haircut."` In other words, the print method distinguishes a barber from customers with the negative of the barber's id.

You must update the following six methods to work with multiple barbers: the two `Shop`( ) constructors, `visitShop`( ), `leaveShop`( ), `helloCustomer`( ), and `byeCustomer`( ). Each of their new specifications are summarized below:

(1) `Shop(int nBarbers, int nChairs)`

Initializes a Shop object with nBarbers and nChairs.

(2) `Shop()`

Initializes a Shop object with 1 barber and 3 chairs.

(3) `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 all chairs are full
        {
                Print "id leaves the shop because of no available waiting chairs".
                Increment nDropsOff.
                Leave the critical section. Return −1.
        }
        if all barbers are busy
        {
                Take a waiting chair (Push the customer in a waiting queue).
                Print "id takes a waiting chair. # waiting seats available = …".
                Wait for a barber to wake me up.
                Pop me out from the queue
        }
        Get my barber whose id is barberId.
        Print "id moves to a service chair[barberId], # waiting seats available = …".
        Have barberId start my haircut.
Leave the critical section. Return barberId.
```

(4) `void leaveShop(int customerId, int barberId)`

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

```
Enter the critical section.
    Print "customerId  wait for barber[barberId] to be done with hair-cut."
    While barberId  is cutting my hair,
        Wait.
    Pay barber
    Print "customerId  says good-by to barber[]" // finish transaction
Leave the critical section.
```

(5) `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 " –id sleeps because of no customers."
        Wait until a customer wakes me up.
    }
    Print "–id starts a hair-cut service for customer[customer id that woke me up]."
Leave the critical section.
```

(6) `byeCustomer( int id )`

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

```
Enter the critical section.
    Print "–id says he's done with a hair-cut service for customer[my customer id]."
    Wakes up my customer.
    Wait for my customer to pay before I take a new one
    Print ""–id calls in another customer."
    Wakes up another customer who is waiting on a waiting chair.
Leave the critical section.
```

8. Detailed Statement of Work

Step 1: Rename shop_org files (.cpp/.h) to shop (.cpp/.h). Also copy over driver.cpp.

Step 2: Update your Shop.h and Shop.cpp in accordance with the specifications of the Shop class as noted above.

Step 3: Compile with `g++ -std=c++11 driver.cpp Shop.cpp -o sleepingBarbers -lpthread`

Step 4: Run your program with the following two scenarios:

```
./sleepingBarbers 1 1 10 1000
./sleepingBarbers 3 1 10 1000
```

Compare your results with sample output files found on canvas:

```
1barber_1chair_10customer_1000stime
3barber_1chair_10customer_1000stime
```

Since the program runs with usleep( ) that may have some clock skews, your results may not be the same as these two files, but you can still check if your program runs correctly.

Step 5: Run your program with

```
./sleepingBarbers 1 chair 200 1000
```

where *chairs* should be 1 ~ 60.

Approximately how many waiting chairs would be necessary for all 200 customers to be served by 1 barber? *Note that depending on the number of cores and clock skews, it may very well require over 60 chairs to accomplish the task.*

Step 6: Run your program with

```
./sleepingBarbers barbers 0 200 1000
```

Where *barbers* should be 1 ~ 3.

Approximately how many barbers would be necessary for all 200 customers to be served without waiting?

9. What to Turn in

When you submit to canvas, put both Documentation and Discussions in the same document and utilize either word or pdf format. The limit for each of these is two pages, so, you can have 4 pages in total.

	Criteria	**Grade**
Code	**Source code** that adheres good modularization, coding style, and an appropriate amount of comments.	3
	Execution output / Correctness: verification of the correctness of the implementation and observance of steps 5 and 6.	20
Report	**Documentation** of your Shop.cpp implementation including explanations and illustration in 1 -2 pages	3
	Discussions (1-2 pages) • Limitations and possible extension of your program • Discussions on your answers to Step5 and Step6	4
	Total	30

10. Clarifications / FAQ / Diagrams

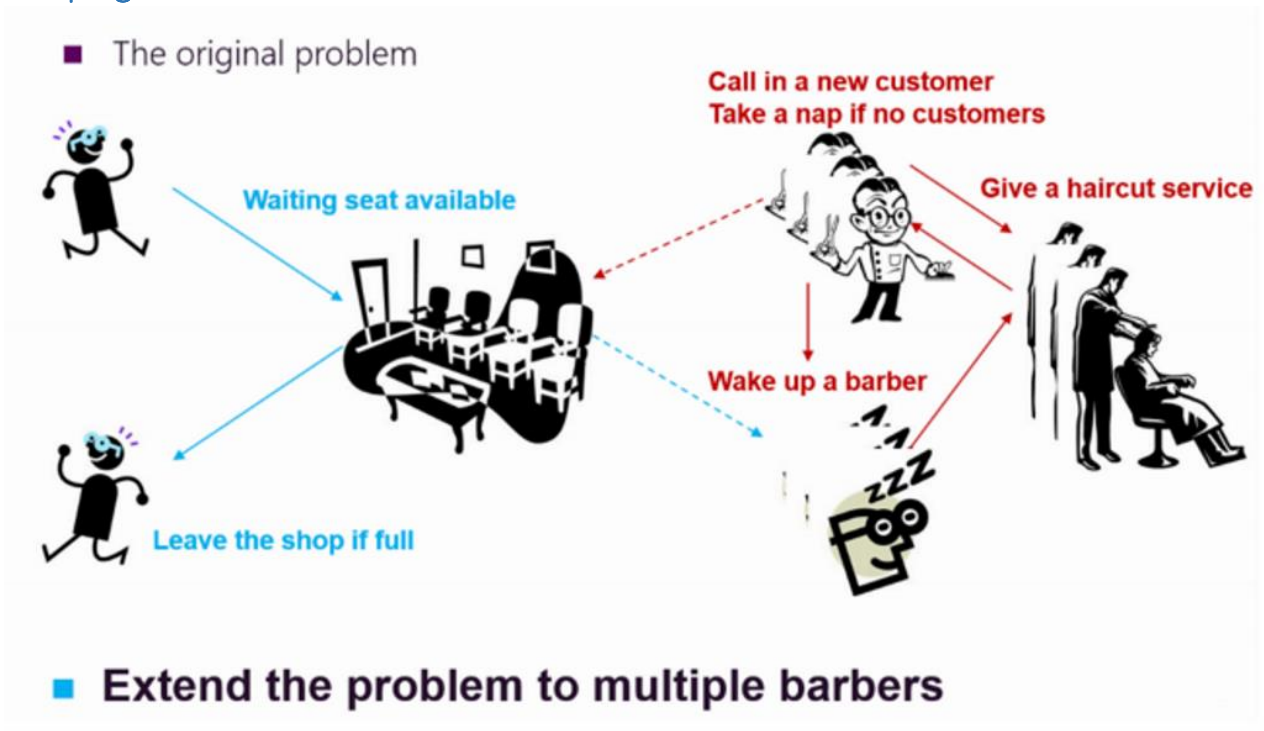
Feel free to fix any memory leaks, incomplete destructors, or other issues you may find in this program. Please report them back to me and comment clearly in your code. This will help improve the quality of the assignments in the future.

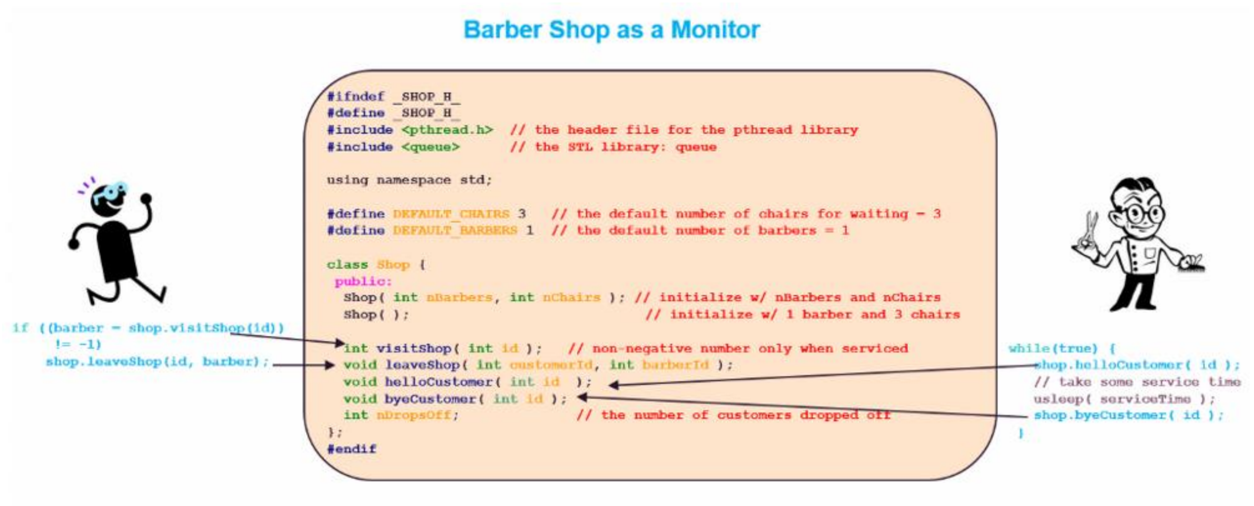
Make sure you have the following includes:

```
sstream
iostream
Stdlib
Shop.h
```

***Suggestion**: if you are having issues synchronizing your barbers and customers, try to get it working first with 1 barber, 1 waiting chair and 1 customer. Visit the Program 2 FAQs!*

Sleeping Barbers Problem

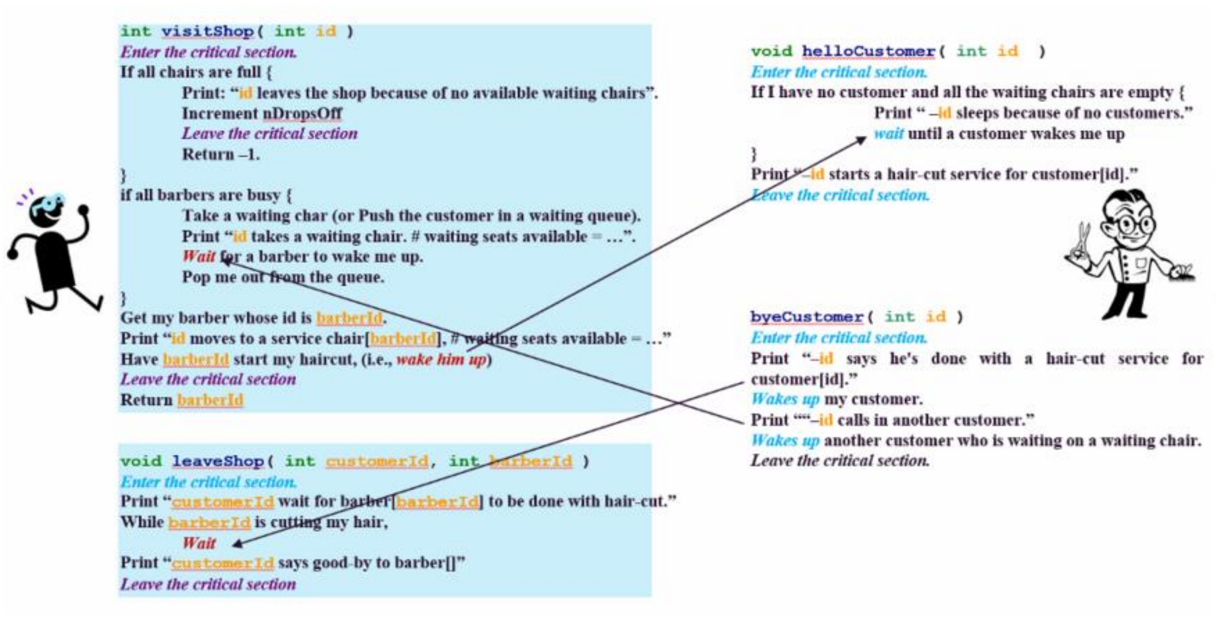




```
#ifndef _SHOP_H_
#define _SHOP_H_
#include <pthread.h>  // the header file for the pthread library
#include <queue>      // the STL library: queue

using namespace std;

#define DEFAULT_CHAIRS 3   // the default number of chairs for waiting = 3
#define DEFAULT_BARBERS 1  // the default number of barbers = 1

class Shop {
 public:
  Shop( int nBarbers, int nChairs ); // initialize w/ nBarbers and nChairs
  Shop( );                           // initialize w/ 1 barber and 3 chairs

  int visitShop( int id );   // non-negative number only when serviced
  void leaveShop( int customerId, int barberId );
  void helloCustomer( int id  );
  void byeCustomer( int id );
  int nDropsOff;             // the number of customers dropped off
};
#endif
```

```
if ((barber = shop.visitShop(id))
    != -1)
  shop.leaveShop(id, barber);
```

```
while(true) {
  shop.helloCustomer( id );
  // take some service time
  usleep( serviceTime );
  shop.byeCustomer( id );
}
```



```
int visitShop( int id )
Enter the critical section.
If all chairs are full {
        Print: "id leaves the shop because of no available waiting chairs".
        Increment nDropsOff
        Leave the critical section
        Return –1.
}
if all barbers are busy {
        Take a waiting char (or Push the customer in a waiting queue).
        Print "id takes a waiting chair. # waiting seats available = …".
        Wait for a barber to wake me up.
        Pop me out from the queue.
}
Get my barber whose id is barberId.
Print "id moves to a service chair[barberId], # waiting seats available = …"
Have barberId start my haircut, (i.e., wake him up)
Leave the critical section
Return barberId
```

```
void leaveShop( int customerId, int barberId )
Enter the critical section.
Print "customerId wait for barber[barberId] to be done with hair-cut."
While barberId is cutting my hair,
        Wait
Print "customerId says good-by to barber[]"
Leave the critical section
```

```
void helloCustomer( int id  )
Enter the critical section.
If I have no customer and all the waiting chairs are empty {
                Print " –id sleeps because of no customers."
                wait until a customer wakes me up
}
Print "–id starts a hair-cut service for customer[id]."
Leave the critical section.
```

```
byeCustomer( int id )
Enter the critical section.
Print "–id says he's done with a hair-cut service for customer[id]."
Wakes up my customer.
Print ""–id calls in another customer."
Wakes up another customer who is waiting on a waiting chair.
Leave the critical section.
```

Output Example

```
~/programming/prog2$ ./sleepingBarbers 1 1 10 1000
barber  [0]: sleeps because of no customers.
customer[1]: moves to a service chair[0], # waiting seats available = 1
customer[1]: wait for barber[0] to be done with hair-cut
barber  [0]: starts a hair-cut service for customer[1]
customer[2]: takes a waiting chair. # waiting seats available = 0
barber  [0]: says he's done with a hair-cut service for customer[1]
customer[1]: says good-bye to barber[0]
barber  [0]: calls in another customer
customer[2]: moves to a service chair[0], # waiting seats available = 1
customer[2]: wait for barber[0] to be done with hair-cut
barber  [0]: starts a hair-cut service for customer[2]
customer[3]: takes a waiting chair. # waiting seats available = 0
barber  [0]: says he's done with a hair-cut service for customer[2]
customer[2]: says good-bye to barber[0]
barber  [0]: calls in another customer
customer[3]: moves to a service chair[0], # waiting seats available = 1
customer[3]: wait for barber[0] to be done with hair-cut
barber  [0]: starts a hair-cut service for customer[3]
customer[4]: takes a waiting chair. # waiting seats available = 0
customer[5]: leaves the shop because of no available waiting chairs.
barber  [0]: says he's done with a hair-cut service for customer[3]
customer[3]: says good-bye to barber[0]
barber  [0]: calls in another customer
customer[4]: moves to a service chair[0], # waiting seats available = 1
customer[4]: wait for barber[0] to be done with hair-cut
barber  [0]: starts a hair-cut service for customer[4]
customer[6]: takes a waiting chair. # waiting seats available = 0
customer[7]: leaves the shop because of no available waiting chairs.
barber  [0]: says he's done with a hair-cut service for customer[4]
customer[4]: says good-bye to barber[0]
barber  [0]: calls in another customer
customer[6]: moves to a service chair[0], # waiting seats available = 1
customer[6]: wait for barber[0] to be done with hair-cut
barber  [0]: starts a hair-cut service for customer[6]
customer[8]: takes a waiting chair. # waiting seats available = 0
customer[9]: leaves the shop because of no available waiting chairs.
customer[10]: leaves the shop because of no available waiting chairs.
barber  [0]: says he's done with a hair-cut service for customer[6]
customer[6]: says good-bye to barber[0]
barber  [0]: calls in another customer
customer[8]: moves to a service chair[0], # waiting seats available = 1
customer[8]: wait for barber[0] to be done with hair-cut
barber  [0]: starts a hair-cut service for customer[8]
barber  [0]: says he's done with a hair-cut service for customer[8]
customer[8]: says good-bye to barber[0]
barber  [0]: calls in another customer
barber  [0]: sleeps because of no customers.
# customers who didn't receive a service = 4
```

```
~/programming/prog2$ ./sleepingBarbers 3 1 10 1000
barber  [0]: sleeps because of no customers.
barber  [1]: sleeps because of no customers.
barber  [2]: sleeps because of no customers.
customer[1]: moves to a service chair[0], # waiting seats available = 1
customer[1]: wait for barber[0] to be done with hair-cut
barber  [0]: starts a hair-cut service for customer[1]
customer[2]: moves to a service chair[1], # waiting seats available = 1
customer[2]: wait for barber[1] to be done with hair-cut
barber  [1]: starts a hair-cut service for customer[2]
barber  [0]: says he's done with a hair-cut service for customer[1]
customer[1]: says good-bye to barber[0]
barber  [0]: calls in another customer
barber  [0]: sleeps because of no customers.
customer[3]: moves to a service chair[0], # waiting seats available = 1
customer[3]: wait for barber[0] to be done with hair-cut
barber  [0]: starts a hair-cut service for customer[3]
barber  [1]: says he's done with a hair-cut service for customer[2]
customer[2]: says good-bye to barber[1]
barber  [1]: calls in another customer
barber  [1]: sleeps because of no customers.
customer[4]: moves to a service chair[1], # waiting seats available = 1
customer[4]: wait for barber[1] to be done with hair-cut
barber  [1]: starts a hair-cut service for customer[4]
barber  [0]: says he's done with a hair-cut service for customer[3]
customer[3]: says good-bye to barber[0]
barber  [0]: calls in another customer
barber  [0]: sleeps because of no customers.
customer[5]: moves to a service chair[0], # waiting seats available = 1
customer[5]: wait for barber[0] to be done with hair-cut
barber  [0]: starts a hair-cut service for customer[5]
barber  [1]: says he's done with a hair-cut service for customer[4]
customer[4]: says good-bye to barber[1]
barber  [1]: calls in another customer
barber  [1]: sleeps because of no customers.
customer[6]: moves to a service chair[1], # waiting seats available = 1
customer[6]: wait for barber[1] to be done with hair-cut
barber  [1]: starts a hair-cut service for customer[6]
customer[7]: moves to a service chair[2], # waiting seats available = 1
customer[7]: wait for barber[2] to be done with hair-cut
barber  [2]: starts a hair-cut service for customer[7]
barber  [0]: says he's done with a hair-cut service for customer[5]
customer[5]: says good-bye to barber[0]
barber  [0]: calls in another customer
barber  [0]: sleeps because of no customers.
customer[8]: moves to a service chair[0], # waiting seats available = 1
customer[8]: wait for barber[0] to be done with hair-cut
barber  [0]: starts a hair-cut service for customer[8]
barber  [1]: says he's done with a hair-cut service for customer[6]
customer[6]: says good-bye to barber[1]
barber  [1]: calls in another customer
barber  [1]: sleeps because of no customers.
barber  [2]: says he's done with a hair-cut service for customer[7]
customer[7]: says good-bye to barber[2]
barber  [2]: calls in another customer
barber  [2]: sleeps because of no customers.
```

```
customer[9]: moves to a service chair[1], # waiting seats available = 1
customer[9]: wait for barber[1] to be done with hair-cut
barber [1]: starts a hair-cut service for customer[9]
barber [0]: says he's done with a hair-cut service for customer[8]
customer[8]: says good-bye to barber[0]
barber [0]: calls in another customer
barber [0]: sleeps because of no customers.
customer[10]: moves to a service chair[0], # waiting seats available = 1
customer[10]: wait for barber[0] to be done with hair-cut
barber [0]: starts a hair-cut service for customer[10]
barber [1]: says he's done with a hair-cut service for customer[9]
customer[9]: says good-bye to barber[1]
barber [1]: calls in another customer
barber [1]: sleeps because of no customers.
barber [0]: says he's done with a hair-cut service for customer[10]
customer[10]: says good-bye to barber[0]
barber [0]: calls in another customer
barber [0]: sleeps because of no customers.
# customers who didn't receive a service = 0
```