CSS 503
Program 2: The Sleeping-Barbers Problem

Professor: Munehiro Fukuda
Due date: see the syllabus

1. Purpose

In this programming assignment, we will extend the original so-called sleeping-barber problem to a multiple sleeping barbers problem where many customers visit a barbershop and receive a haircut service from any one available among barbers in the shop.

2. Sleeping-Barber Problem

The original problem description from our textbook is:

A barbershop consists of a waiting room with n 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 chairs are occupied, then the customer leaves the shop. If the barber is busy but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber.

3. Our Extended Sleeping-Barbers Problem

We will change the original description as follows:

A barbershop consists of a waiting room with n 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 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 chairs. If the barbers are asleep, the customer wakes up one of the barbers.

4. Main Program: driver.cpp

The driver.cpp (found under ~css503/programming/prog2/) is a driver program that tests your sleeping-barbers problem:

(1) Receives parameters such as:

argv[1]	nBarbers	The number of barbers working in your barbershop
argv[2]	nChairs	The number of chairs available for customers to wait on
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 shop, an object from the Shop class that you will implement.
(3) Spawns the nBarbers of barber threads as passing to a pointer to the shop object, the identifier (i.e. 0 ~ nBarbers – 1), and serviceTime.
(4) With a Random interval in μ seconds, (i.e., `usleep( rand( ) % 1000 )`), spawns one after another customer thread as passing 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

Barber threads are created in main( ). Each barber thread calls the following function (no need to modify):

```
// 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 );     // spend a service time
    shop.byeCustomer( id );    // release the customer
  }
}
```

6. Customer Thread

Customer threads are created in main( ). Each customer thread calls the following function (no need to modify).

```
// the customer thread function
void *customer( void *arg ) {

  // extract parameters
  ThreadParam &param = *(ThreadParam *)arg;
  Shop &shop = *(param.shop);
  int id = param.id;
  delete &param;

  int barber = -1;
  if ( ( barber = shop.visitShop( id ) ) != -1 ) // am I assigned to barber i or no barber (-1)?
    shop.leaveShop( id, barber );                // wait until my service is finished
}
```

7. Shop Class

This is the class that you have to design. The template is as follows:

```
#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 a Shop object with nBarbers and nChairs
  Shop( );                            // initialize a Shop object with 1 barber and 3 chairs

  int visitShop( int id );   // return a non-negative number only when a customer got a service
  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 template. You must add some private variables such as pthread_mutex_t and pthread_cond_t, etc. to implement this Shop class as a monitor. You may use two private helper methods int2string( ) and print( ) whose implementations are given below. (If you want to use them, please copy and paste the following code into your Shop.cpp.)

```
string Shop::int2string( int i ) {
  stringstream out;
  out << i;
  return out.str( );
}

void Shop::print( int person, string message ) {
  cout << ( ( person > 0 ) ? "customer[" : "barber  [" )
       << abs( person ) << "]: " << message << endl;
}
```

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 hair cut." In other words, the print method distinguishes a barber from customers with the negative of the barber's id.

In this programming assignment, you must implement the following six methods: the two Shop( ) constructors, visitShop( ), leaveShop( ), helloCustomer( ), and byeCustomer( ). Their specifications are summarized below:

(1) `Shop( int nBarbers, int nChairs )`
Initializes a Shop object with nBarabers 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.

```
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.
Leave the critical section.
Return barberId.
```

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

Is called by a customer thread.

```
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.
```

(5) `void helloCustomer( int id  )`

Is called by a barber thread.

```
Enter the critical section.
If I have no customer and all the waiting chairs are empty {
    Print “ –id sleeps because of no customers.”
    (Note that we display barber’s id from 0 to –(nBarbers – 1) in order to distinguish it from a
    customer id (from 1 to nCustomers).)
    wait until a customer wakes me up.
}
Print “–id starts a hair-cut service for customer[the customer thread id that woke me up].”
Leave the critical section.
```

(6) `byeCustomer( int id )`

Is called by a barber thread.

```
Enter the critical section.
Print “–id says he's done with a hair-cut service for customer[my customer thread 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.
```

8. Statement of Work

Follow through the six steps described below:

Step 1: Copy ~css503/prog2/driver.cpp to your directory; copy and paste the Shop.h template into your own Shop.h; and copy and paste int2string( ) and print( ) functions into your own Shop.cpp.

Step 2: Complete your Shop.h and Shop.cpp in accordance with the specifications of the Shop class.

Step 3: Compile with “g++ 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 the following two files under ~css503/prog2/

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 *chars* should be 1 ~ 60. Approximately how many waiting chairs would be necessary for all 200 customers to be served by 1 barber?

Step 6: Run your program with
./sleepingBarbers *barbars* 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

This programming assignment is due at the beginning of class on the due date. Please turn in the following materials in a hard copy. No email submission is accepted.

Criteria	**Grade**
Documentation of your Shop.cpp implementation including explanations and illustration in one or two pages. (No more than two, otherwise – 2pts)	20pts
Source code that adheres good modularization, coding style, and an appropriate amount of commends. • 25pts: well-organized and correct code • 23pts: messy yet working code or code with minor errors receives • 20pts: code with major bugs or incomplete code receives	25pts
Execution output that verifies the correctness of your implementation and observes the execution changes in Step 5 and Step 6. • 25pts: Sample outputs with ./sleepingBarbers 1 1 10 1000 and ./sleepingBarbers 3 1 10 1000 that verify the correctness of your Shop.h and Shop.cpp as well as your answer to Step 5 and Step 6 in Section 8. Statement of Work. • 20pts: Sample outputs with ./sleepingBarbers 1 1 10 1000 and ./sleepingBarbers 3 1 10 1000 that verify the correctness of your Shop.h and Shop.cpp but no answers to Step 5 and Step 6 in Section 8. Statement of Work. • 15pts: Sample outputs with ./sleepingBarbers 1 1 10 1000 and ./sleepingBarbers 3 1 10 1000 that however show some incorrectness of your Shop.h and Shop.cpp but your answer to Step 5 and Step 6 in Section 8. Statement of Work. • 10pts: Sample outputs with ./sleepingBarbers 1 1 10 1000 and ./sleepingBarbers 3 1 10 1000 that however show some incorrectness of your Shop.h and Shop.cpp and no answers to Step 5 and Step 6 in Section 8. Statement of Work. • 5pts: No results.	25pts
Discussions in one or two pages. (No more than two, otherwise – 2pts) • Limitation and possible extension of your program (+15pts) • Discussions on your answers to Step5 and Step6 (+10pts)	25pts
Lab Session 2 If you have not yet turned in a hard copy of your source code and output or missed this session, please turn in together with program 2.	5pts
Total Note that program 2 takes 11% of your final grade.	100pts