I. Documentation

Environment and Overview

This assignment involved solving a well-known synchronization problem called The Sleeping Barbers Problem. My implementation of the solution was done in Linux and uses the pthreads library to create multiple threads that run in concurrent execution. Some of the threads are “barbers” who are attempting to serve as many “customer” threads as possible given a limited number of available resources: service chairs to give a customer thread a haircut and waiting chairs that allow customer threads to wait for an open service chair (an available barber).

The user provides a command-line execution to the run the program, as follows:

```
$ ./sleepingBarbers 1 2 10 1000
```

Here, the first number (1) is the number of barber threads, second (2) is the number of waiting chairs in the shop, third (10) is the number of customers and fourth (1000) is the time in microseconds for a barber to finish a haircut once the customer is in a service chair. If no waiting chairs are available, the customer leaves the shop and we increment a counter to indicate a customer that was not served.

Technique to Solve Sleeping Barbers

With all these concurrent threads come obvious thread synchronization issues with customer threads competing for service chairs and waiting seats. The method used to solve these issues is a **monitor class,** called Shop. The primary means of synchronization are achieved by two means:

1. **pthreads_mutex_t:** A mutex lock provided by the pthreads library
2. **pthreads_cond_t:** A condition variable that can be waited to halt execution until another thread signals, indicating execution can continue.

Each barber is assigned a sequential ID from 0 to N, and each customer receives and ID also, from 1 to N. This allows the output to console to show which barber and which customer are at what stage.

Program Structure

The Shop class, which includes Shop.h and Shop.cpp, has a single mutex, **shopLock**, and two condition variable arrays, **barberIsReady** and **signalNext**. barberIsReady is a condition variable array waited on by customers who take a waiting chair when all barbers are busy, and is signaled to by a barber when their current haircut is complete. barberIsReady uses an index of the number of seats available when the customer took a waiting seat. signalNext is a condition variable array used by both customer and barber, which uses an index of barberID (0 to N).

The other data members are:

- int numberChairs: A semaphore that indicates the number of available waiting chairs.
- int maxChairs: The maximum number of chairs available in the shop, may be 0.
- int * customerAssignedTo: This is an array of size equal to the number of barbers. It uses an index of the barber ID to assign a value of the customer ID, linking the two for signal passing.
- queue<int> barberQueue: Contains barbers waiting on a customer to arrive.
- queue<int> customerQueue: Contains customers waiting for a barber to be available.

Every event perpetrated by a barber or customer is sent to the console via cout to show which customer or barber is doing what action.

The class has the following synchronization methods:

- **visitShop():** This is the method first called by the customer thread in the Shop object monitor. If there are no waiting chairs, or if no waiting chairs exist and the barbers are all busy, the customer leaves (thread terminates). Otherwise, the customer takes a waiting chair before being signaled by a barber to start a haircut, or wakes a sleeping barber.
- **leaveShop():** Once the customer thread's haircut begins, they wait for barber to finish the haircut, before saying goodbye to barber. The customer thread then terminates.
- **helloCustomer():** This is the initial method where a barber thread enters the monitor. If there are no customers, the barber sleeps and waits for a signalNext condition from a new customer. Otherwise, the barber signals the next customer waiting and begins a haircut service.
- **goodbyeCustomer:** The barber finishes a haircut service for the customer, and calls in another customer before re-entering the helloCustomer() method.

There are also a few other methods to create and destroy the Shop object:

- **Shop():** Default constructor sets barbers to 1 and number chairs to 3.
- **Shop(int nBarbers, int nChairs):** Sets barbers to nBarbers and waiting chairs to nChairs.
- **~Shop():** Destructor deletes arrays customerAssignedTo, barberIsReady and signalNext.

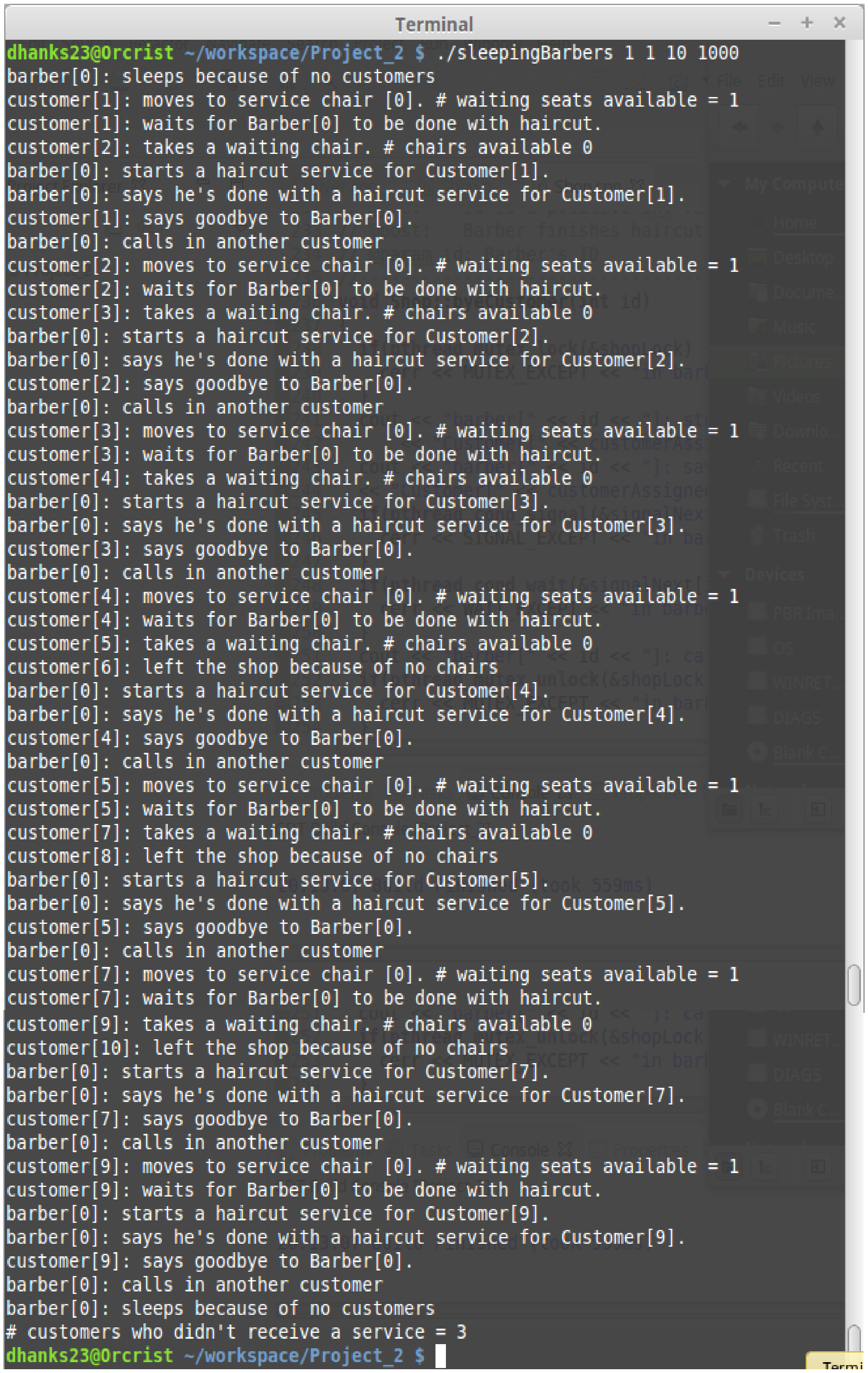
Error checking was added to the program in driver.cpp to catch and exception from a bad allocation of a new thread, and in the Shop class. Inside the Shop methods, as individual threads are operating, a message is sent to cerr if a cond_wait, cond_signal, mutex_lock or mutex_unlock fails for any reason. The message contains the nature of the error and the thread ID that generated it. cerr outputs are used in place of throwing exceptions, so a thread is not terminated resulting in deadlock or a segmentation fault.

III. Execution Output

All output is for a **machine with 4 Cores** (Slower than 2-Core)

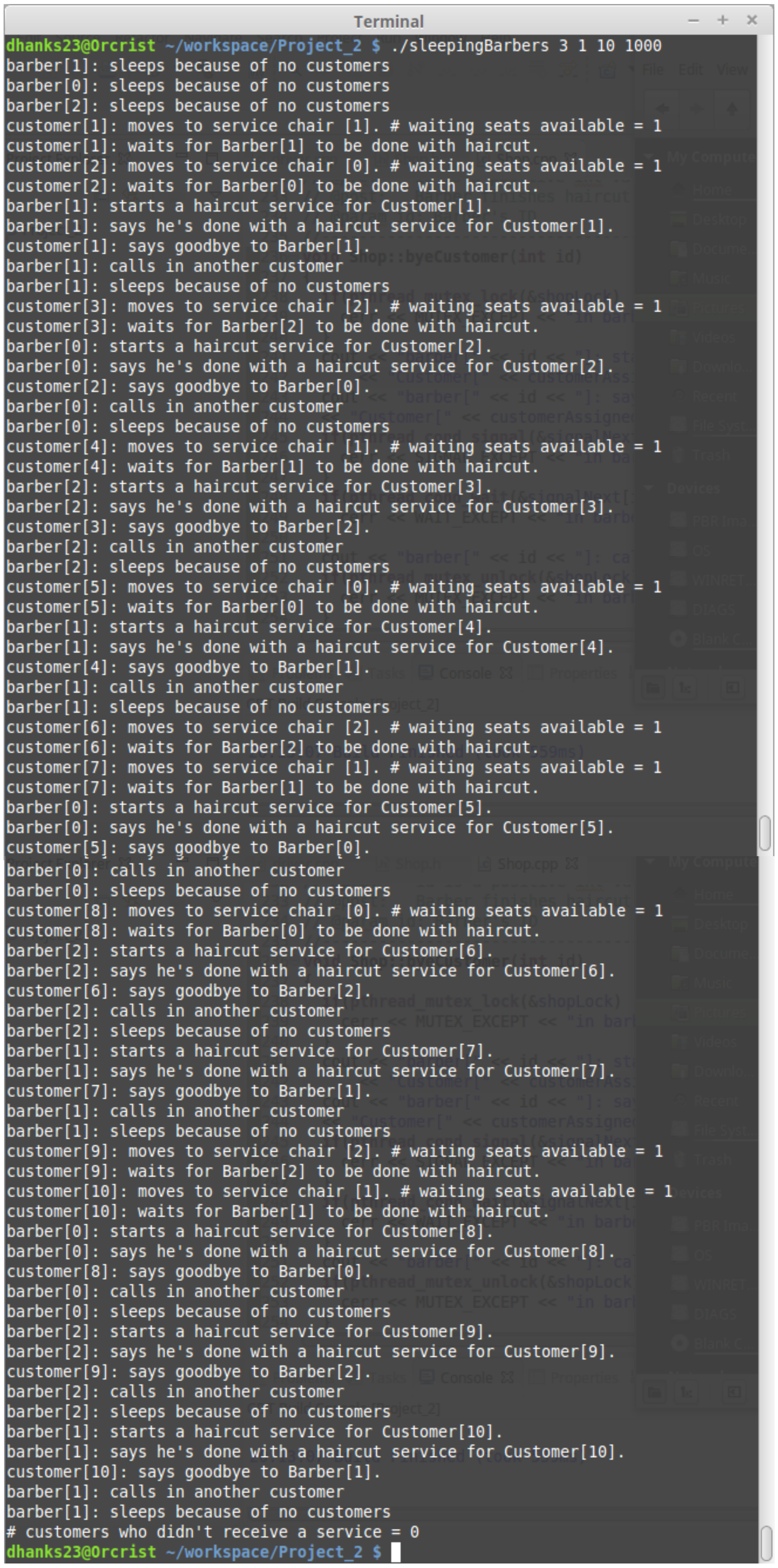
Output for 1 Barber, 1 Chair and 10 Customers with 1,000 ms service time.

3 customers dropped.



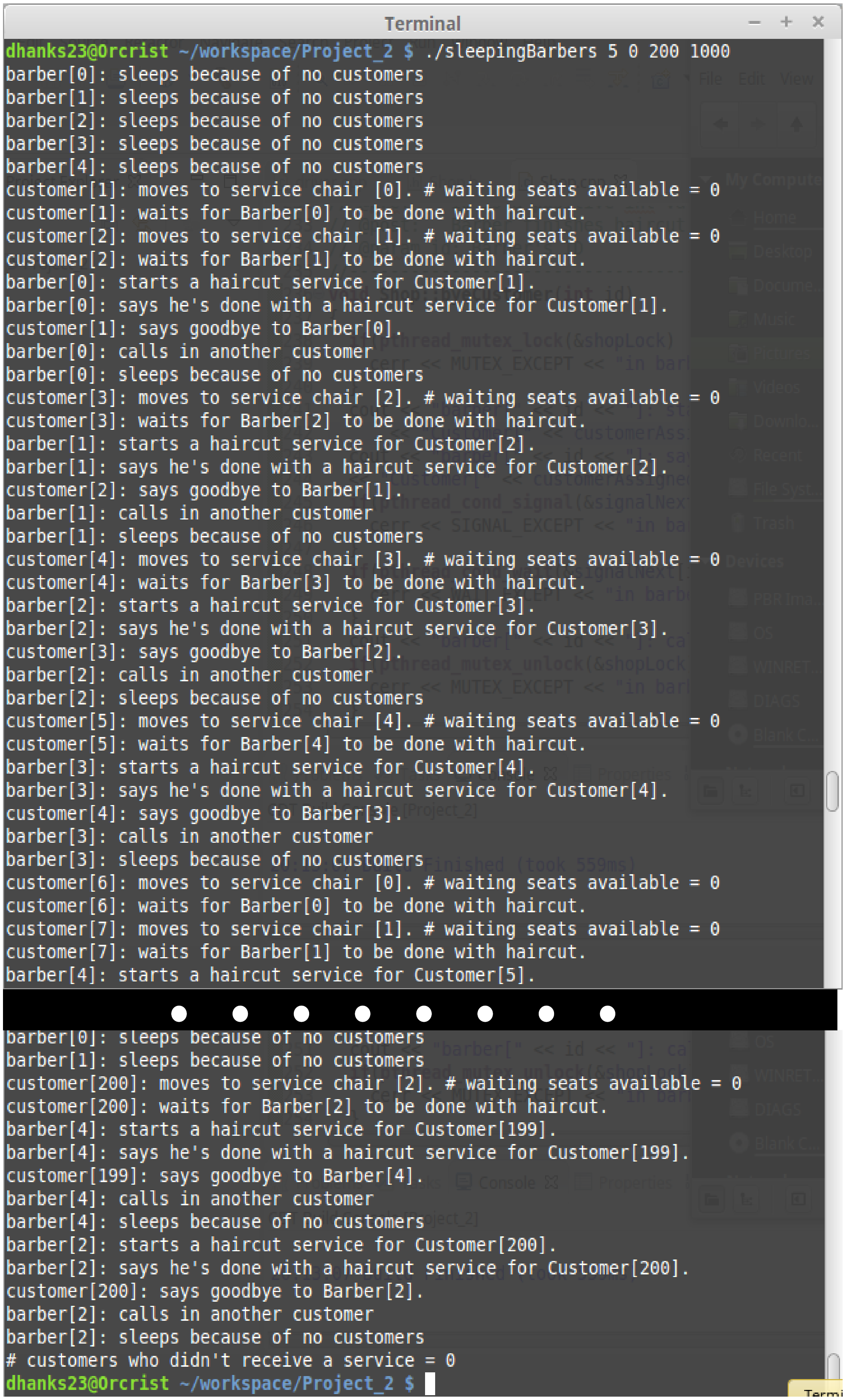
```
dhanks23@Orcrist ~/workspace/Project_2 $ ./sleepingBarbers 1 1 10 1000
barber[0]: sleeps because of no customers
customer[1]: moves to service chair [0]. # waiting seats available = 1
customer[1]: waits for Barber[0] to be done with haircut.
customer[2]: takes a waiting chair. # chairs available 0
barber[0]: starts a haircut service for Customer[1].
barber[0]: says he's done with a haircut service for Customer[1].
customer[1]: says goodbye to Barber[0].
barber[0]: calls in another customer
customer[2]: moves to service chair [0]. # waiting seats available = 1
customer[2]: waits for Barber[0] to be done with haircut.
customer[3]: takes a waiting chair. # chairs available 0
barber[0]: starts a haircut service for Customer[2].
barber[0]: says he's done with a haircut service for Customer[2].
customer[2]: says goodbye to Barber[0].
barber[0]: calls in another customer
customer[3]: moves to service chair [0]. # waiting seats available = 1
customer[3]: waits for Barber[0] to be done with haircut.
customer[4]: takes a waiting chair. # chairs available 0
barber[0]: starts a haircut service for Customer[3].
barber[0]: says he's done with a haircut service for Customer[3].
customer[3]: says goodbye to Barber[0].
barber[0]: calls in another customer
customer[4]: moves to service chair [0]. # waiting seats available = 1
customer[4]: waits for Barber[0] to be done with haircut.
customer[5]: takes a waiting chair. # chairs available 0
customer[6]: left the shop because of no chairs
barber[0]: starts a haircut service for Customer[4].
barber[0]: says he's done with a haircut service for Customer[4].
customer[4]: says goodbye to Barber[0].
barber[0]: calls in another customer
customer[5]: moves to service chair [0]. # waiting seats available = 1
customer[5]: waits for Barber[0] to be done with haircut.
customer[7]: takes a waiting chair. # chairs available 0
customer[8]: left the shop because of no chairs
barber[0]: starts a haircut service for Customer[5].
barber[0]: says he's done with a haircut service for Customer[5].
customer[5]: says goodbye to Barber[0].
barber[0]: calls in another customer
customer[7]: moves to service chair [0]. # waiting seats available = 1
customer[7]: waits for Barber[0] to be done with haircut.
customer[9]: takes a waiting chair. # chairs available 0
customer[10]: left the shop because of no chairs
barber[0]: starts a haircut service for Customer[7].
barber[0]: says he's done with a haircut service for Customer[7].
customer[7]: says goodbye to Barber[0].
barber[0]: calls in another customer
customer[9]: moves to service chair [0]. # waiting seats available = 1
customer[9]: waits for Barber[0] to be done with haircut.
barber[0]: starts a haircut service for Customer[9].
barber[0]: says he's done with a haircut service for Customer[9].
customer[9]: says goodbye to Barber[0].
barber[0]: calls in another customer
barber[0]: sleeps because of no customers
# customers who didn't receive a service = 3
dhanks23@Orcrist ~/workspace/Project_2 $
```

Output for 3 Barbers, 1 Chair, 10 Customers and 1,000 ms service time. **0 customers dropped**



```
dhanks23@Orcrist ~/workspace/Project_2 $ ./sleepingBarbers 3 1 10 1000
barber[1]: sleeps because of no customers
barber[0]: sleeps because of no customers
barber[2]: sleeps because of no customers
customer[1]: moves to service chair [1]. # waiting seats available = 1
customer[1]: waits for Barber[1] to be done with haircut.
customer[2]: moves to service chair [0]. # waiting seats available = 1
customer[2]: waits for Barber[0] to be done with haircut.
barber[1]: starts a haircut service for Customer[1].
barber[1]: says he's done with a haircut service for Customer[1].
customer[1]: says goodbye to Barber[1].
barber[1]: calls in another customer
barber[1]: sleeps because of no customers
customer[3]: moves to service chair [2]. # waiting seats available = 1
customer[3]: waits for Barber[2] to be done with haircut.
barber[0]: starts a haircut service for Customer[2].
barber[0]: says he's done with a haircut service for Customer[2].
customer[2]: says goodbye to Barber[0].
barber[0]: calls in another customer
barber[0]: sleeps because of no customers
customer[4]: moves to service chair [1]. # waiting seats available = 1
customer[4]: waits for Barber[1] to be done with haircut.
barber[2]: starts a haircut service for Customer[3].
barber[2]: says he's done with a haircut service for Customer[3].
customer[3]: says goodbye to Barber[2].
barber[2]: calls in another customer
barber[2]: sleeps because of no customers
customer[5]: moves to service chair [0]. # waiting seats available = 1
customer[5]: waits for Barber[0] to be done with haircut.
barber[1]: starts a haircut service for Customer[4].
barber[1]: says he's done with a haircut service for Customer[4].
customer[4]: says goodbye to Barber[1].
barber[1]: calls in another customer
barber[1]: sleeps because of no customers
customer[6]: moves to service chair [2]. # waiting seats available = 1
customer[6]: waits for Barber[2] to be done with haircut.
customer[7]: moves to service chair [1]. # waiting seats available = 1
customer[7]: waits for Barber[1] to be done with haircut.
barber[0]: starts a haircut service for Customer[5].
barber[0]: says he's done with a haircut service for Customer[5].
customer[5]: says goodbye to Barber[0].
barber[0]: calls in another customer
barber[0]: sleeps because of no customers
customer[8]: moves to service chair [0]. # waiting seats available = 1
customer[8]: waits for Barber[0] to be done with haircut.
barber[2]: starts a haircut service for Customer[6].
barber[2]: says he's done with a haircut service for Customer[6].
customer[6]: says goodbye to Barber[2].
barber[2]: calls in another customer
barber[2]: sleeps because of no customers
barber[1]: starts a haircut service for Customer[7].
barber[1]: says he's done with a haircut service for Customer[7].
customer[7]: says goodbye to Barber[1].
barber[1]: calls in another customer
barber[1]: sleeps because of no customers
customer[9]: moves to service chair [2]. # waiting seats available = 1
customer[9]: waits for Barber[2] to be done with haircut.
customer[10]: moves to service chair [1]. # waiting seats available = 1
customer[10]: waits for Barber[1] to be done with haircut.
barber[0]: starts a haircut service for Customer[8].
barber[0]: says he's done with a haircut service for Customer[8].
customer[8]: says goodbye to Barber[0].
barber[0]: calls in another customer
barber[0]: sleeps because of no customers
barber[2]: starts a haircut service for Customer[9].
barber[2]: says he's done with a haircut service for Customer[9].
customer[9]: says goodbye to Barber[2].
barber[2]: calls in another customer
barber[2]: sleeps because of no customers
barber[1]: starts a haircut service for Customer[10].
barber[1]: says he's done with a haircut service for Customer[10].
customer[10]: says goodbye to Barber[1].
barber[1]: calls in another customer
barber[1]: sleeps because of no customers
# customers who didn't receive a service = 0
dhanks23@Orcrist ~/workspace/Project_2 $
```

Output for 5 Barbers, 0 waiting chairs, 200 customers and 1,000 ms wait time . **0 customers dropped**

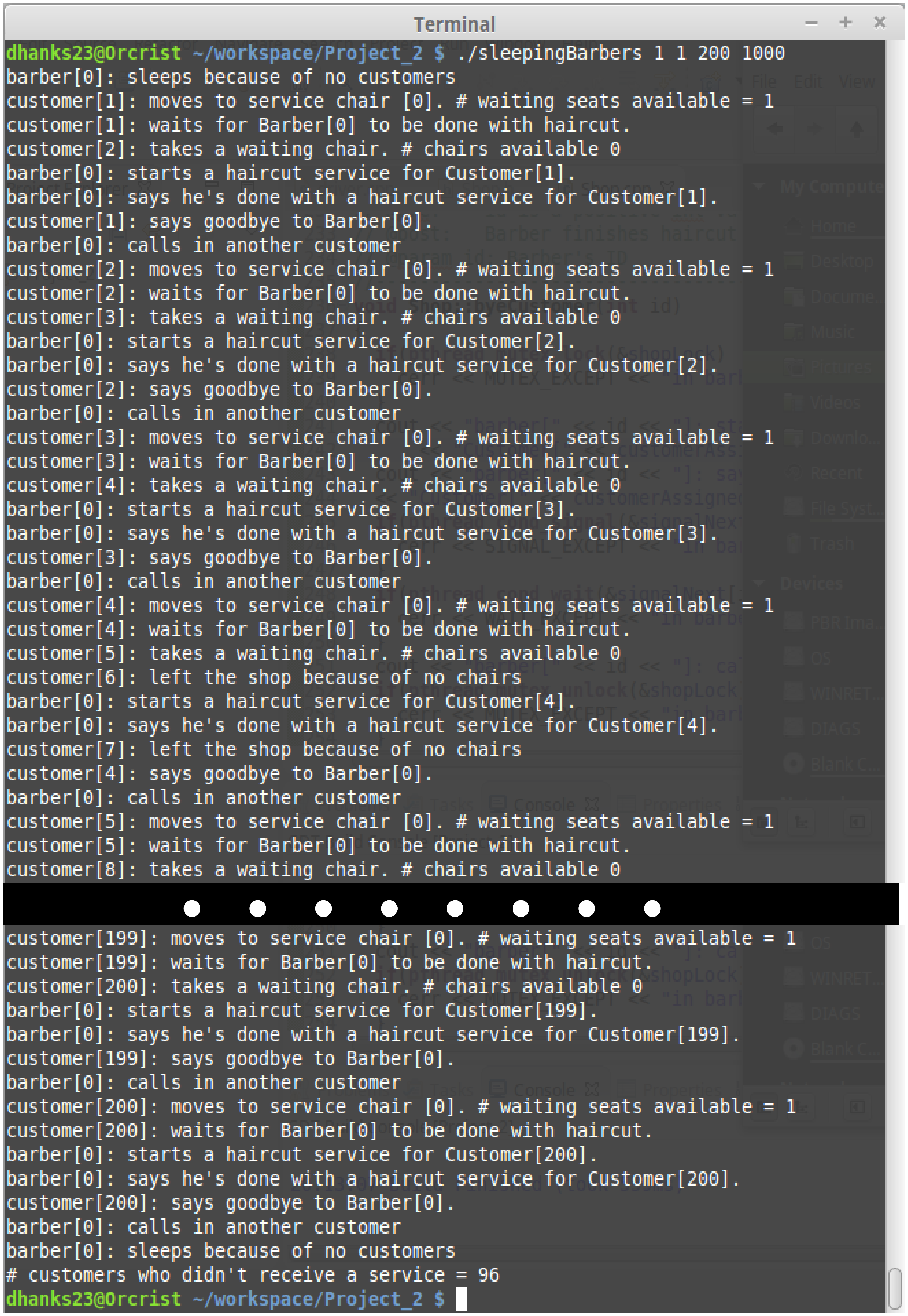

Terminal

```
dhanks23@Orcrist ~/workspace/Project_2 $ ./sleepingBarbers 5 0 200 1000
barber[0]: sleeps because of no customers
barber[1]: sleeps because of no customers
barber[2]: sleeps because of no customers
barber[3]: sleeps because of no customers
barber[4]: sleeps because of no customers
customer[1]: moves to service chair [0]. # waiting seats available = 0
customer[1]: waits for Barber[0] to be done with haircut.
customer[2]: moves to service chair [1]. # waiting seats available = 0
customer[2]: waits for Barber[1] to be done with haircut.
barber[0]: starts a haircut service for Customer[1].
barber[0]: says he's done with a haircut service for Customer[1].
customer[1]: says goodbye to Barber[0].
barber[0]: calls in another customer
barber[0]: sleeps because of no customers
customer[3]: moves to service chair [2]. # waiting seats available = 0
customer[3]: waits for Barber[2] to be done with haircut.
barber[1]: starts a haircut service for Customer[2].
barber[1]: says he's done with a haircut service for Customer[2].
customer[2]: says goodbye to Barber[1].
barber[1]: calls in another customer
barber[1]: sleeps because of no customers
customer[4]: moves to service chair [3]. # waiting seats available = 0
customer[4]: waits for Barber[3] to be done with haircut.
barber[2]: starts a haircut service for Customer[3].
barber[2]: says he's done with a haircut service for Customer[3].
customer[3]: says goodbye to Barber[2].
barber[2]: calls in another customer
barber[2]: sleeps because of no customers
customer[5]: moves to service chair [4]. # waiting seats available = 0
customer[5]: waits for Barber[4] to be done with haircut.
barber[3]: starts a haircut service for Customer[4].
barber[3]: says he's done with a haircut service for Customer[4].
customer[4]: says goodbye to Barber[3].
barber[3]: calls in another customer
barber[3]: sleeps because of no customers
customer[6]: moves to service chair [0]. # waiting seats available = 0
customer[6]: waits for Barber[0] to be done with haircut.
customer[7]: moves to service chair [1]. # waiting seats available = 0
customer[7]: waits for Barber[1] to be done with haircut.
barber[4]: starts a haircut service for Customer[5].
```

● ● ● ● ● ● ● ●

```
barber[0]: sleeps because of no customers
barber[1]: sleeps because of no customers
customer[200]: moves to service chair [2]. # waiting seats available = 0
customer[200]: waits for Barber[2] to be done with haircut.
barber[4]: starts a haircut service for Customer[199].
barber[4]: says he's done with a haircut service for Customer[199].
customer[199]: says goodbye to Barber[4].
barber[4]: calls in another customer
barber[4]: sleeps because of no customers
barber[2]: starts a haircut service for Customer[200].
barber[2]: says he's done with a haircut service for Customer[200].
customer[200]: says goodbye to Barber[2].
barber[2]: calls in another customer
barber[2]: sleeps because of no customers
# customers who didn't receive a service = 0
dhanks23@Orcrist ~/workspace/Project_2 $
```

Output for 1 Barber, 1 waiting chair, 200 customers and 1,000 ms wait time. **96 customers dropped.**

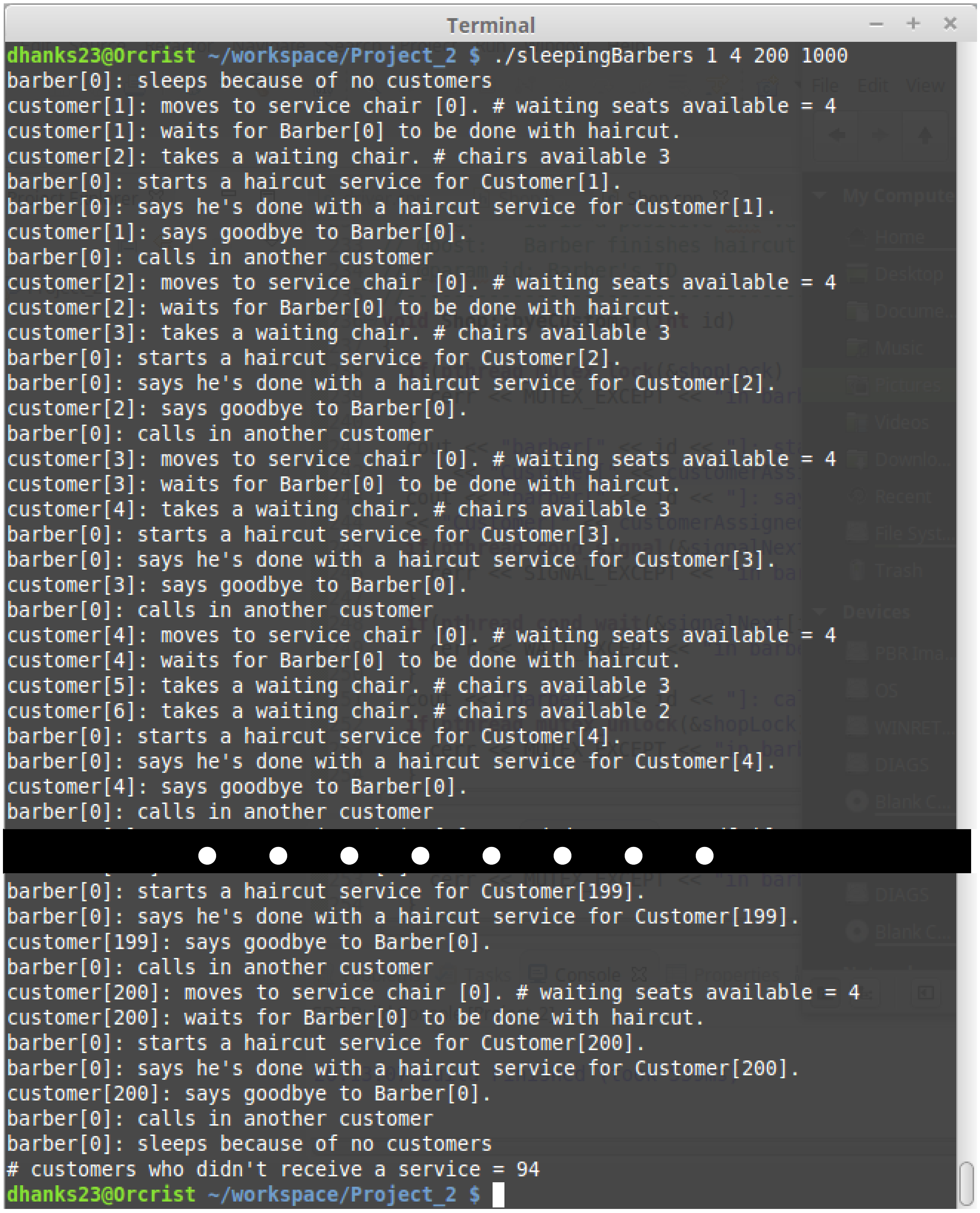

Terminal

```
dhanks23@Orcrist ~/workspace/Project_2 $ ./sleepingBarbers 1 1 200 1000
barber[0]: sleeps because of no customers
customer[1]: moves to service chair [0]. # waiting seats available = 1
customer[1]: waits for Barber[0] to be done with haircut.
customer[2]: takes a waiting chair. # chairs available 0
barber[0]: starts a haircut service for Customer[1].
barber[0]: says he's done with a haircut service for Customer[1].
customer[1]: says goodbye to Barber[0].
barber[0]: calls in another customer
customer[2]: moves to service chair [0]. # waiting seats available = 1
customer[2]: waits for Barber[0] to be done with haircut.
customer[3]: takes a waiting chair. # chairs available 0
barber[0]: starts a haircut service for Customer[2].
barber[0]: says he's done with a haircut service for Customer[2].
customer[2]: says goodbye to Barber[0].
barber[0]: calls in another customer
customer[3]: moves to service chair [0]. # waiting seats available = 1
customer[3]: waits for Barber[0] to be done with haircut.
customer[4]: takes a waiting chair. # chairs available 0
barber[0]: starts a haircut service for Customer[3].
barber[0]: says he's done with a haircut service for Customer[3].
customer[3]: says goodbye to Barber[0].
barber[0]: calls in another customer
customer[4]: moves to service chair [0]. # waiting seats available = 1
customer[4]: waits for Barber[0] to be done with haircut.
customer[5]: takes a waiting chair. # chairs available 0
customer[6]: left the shop because of no chairs
barber[0]: starts a haircut service for Customer[4].
barber[0]: says he's done with a haircut service for Customer[4].
customer[7]: left the shop because of no chairs
customer[4]: says goodbye to Barber[0].
barber[0]: calls in another customer
customer[5]: moves to service chair [0]. # waiting seats available = 1
customer[5]: waits for Barber[0] to be done with haircut.
customer[8]: takes a waiting chair. # chairs available 0
```

● ● ● ● ● ● ● ●

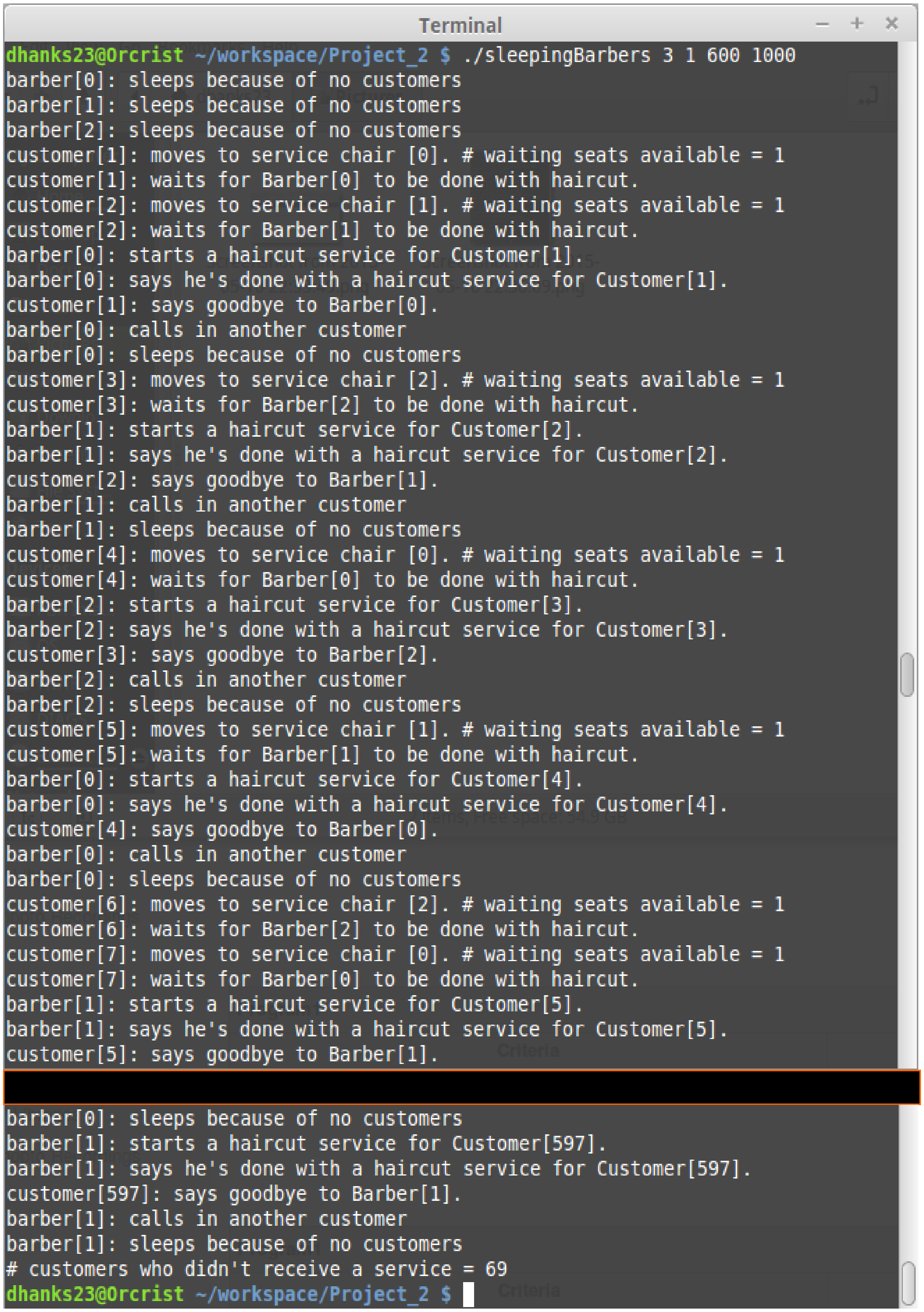
```
customer[199]: moves to service chair [0]. # waiting seats available = 1
customer[199]: waits for Barber[0] to be done with haircut.
customer[200]: takes a waiting chair. # chairs available 0
barber[0]: starts a haircut service for Customer[199].
barber[0]: says he's done with a haircut service for Customer[199].
customer[199]: says goodbye to Barber[0].
barber[0]: calls in another customer
customer[200]: moves to service chair [0]. # waiting seats available = 1
customer[200]: waits for Barber[0] to be done with haircut.
barber[0]: starts a haircut service for Customer[200].
barber[0]: says he's done with a haircut service for Customer[200].
customer[200]: says goodbye to Barber[0].
barber[0]: calls in another customer
barber[0]: sleeps because of no customers
# customers who didn't receive a service = 96
dhanks23@Orcrist ~/workspace/Project_2 $
```

Output for 1 barber, 4 waiting chairs, 200 customers and 1,000 ms wait time. **94 customers dropped.**



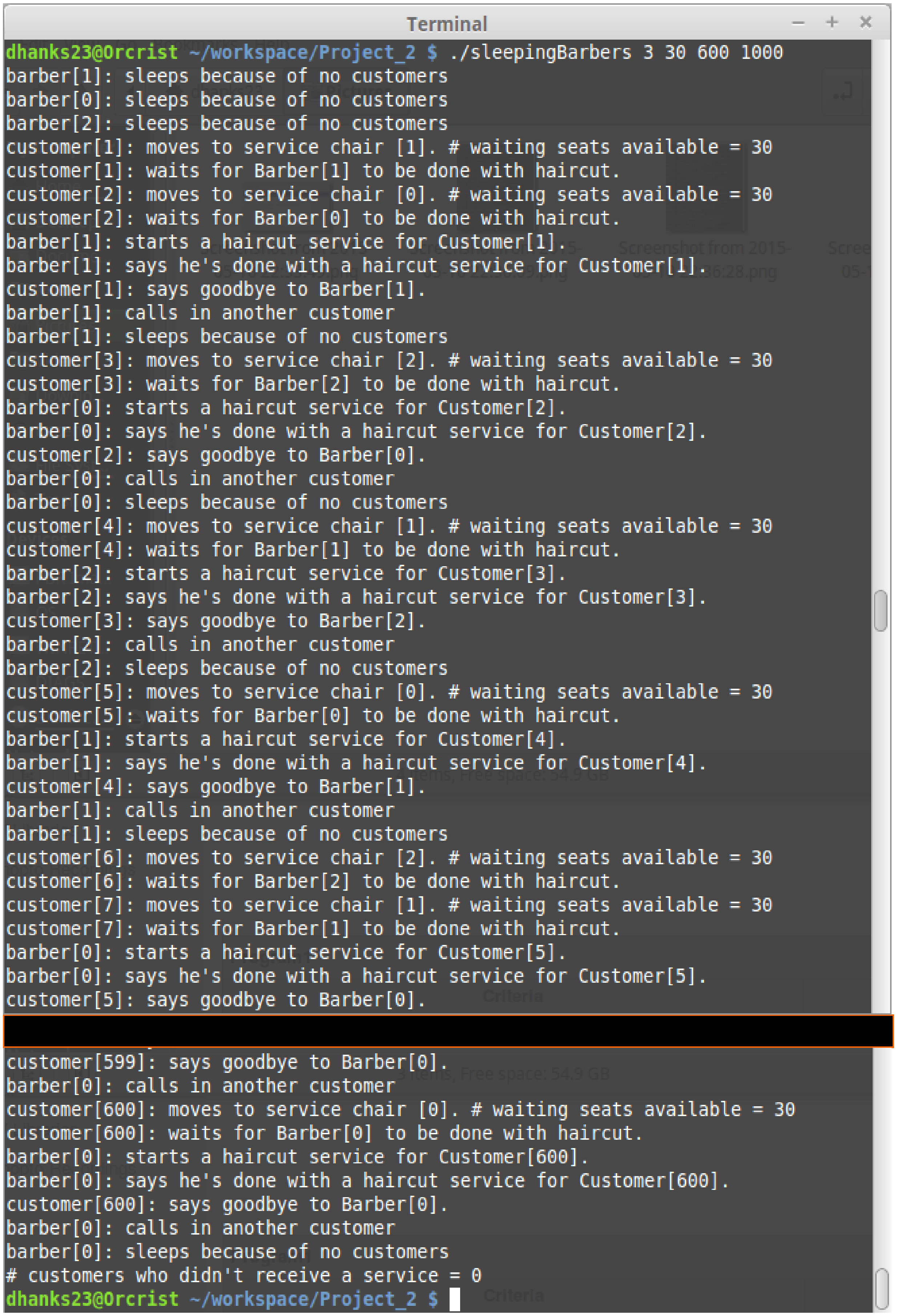
```
dhanks23@Orcrist ~/workspace/Project_2 $ ./sleepingBarbers 1 4 200 1000
barber[0]: sleeps because of no customers
customer[1]: moves to service chair [0]. # waiting seats available = 4
customer[1]: waits for Barber[0] to be done with haircut.
customer[2]: takes a waiting chair. # chairs available 3
barber[0]: starts a haircut service for Customer[1].
barber[0]: says he's done with a haircut service for Customer[1].
customer[1]: says goodbye to Barber[0].
barber[0]: calls in another customer
customer[2]: moves to service chair [0]. # waiting seats available = 4
customer[2]: waits for Barber[0] to be done with haircut.
customer[3]: takes a waiting chair. # chairs available 3
barber[0]: starts a haircut service for Customer[2].
barber[0]: says he's done with a haircut service for Customer[2].
customer[2]: says goodbye to Barber[0].
barber[0]: calls in another customer
customer[3]: moves to service chair [0]. # waiting seats available = 4
customer[3]: waits for Barber[0] to be done with haircut.
customer[4]: takes a waiting chair. # chairs available 3
barber[0]: starts a haircut service for Customer[3].
barber[0]: says he's done with a haircut service for Customer[3].
customer[3]: says goodbye to Barber[0].
barber[0]: calls in another customer
customer[4]: moves to service chair [0]. # waiting seats available = 4
customer[4]: waits for Barber[0] to be done with haircut.
customer[5]: takes a waiting chair. # chairs available 3
customer[6]: takes a waiting chair. # chairs available 2
barber[0]: starts a haircut service for Customer[4].
barber[0]: says he's done with a haircut service for Customer[4].
customer[4]: says goodbye to Barber[0].
barber[0]: calls in another customer
            ●     ●     ●     ●     ●     ●     ●     ●
barber[0]: starts a haircut service for Customer[199].
barber[0]: says he's done with a haircut service for Customer[199].
customer[199]: says goodbye to Barber[0].
barber[0]: calls in another customer
customer[200]: moves to service chair [0]. # waiting seats available = 4
customer[200]: waits for Barber[0] to be done with haircut.
barber[0]: starts a haircut service for Customer[200].
barber[0]: says he's done with a haircut service for Customer[200].
customer[200]: says goodbye to Barber[0].
barber[0]: calls in another customer
barber[0]: sleeps because of no customers
# customers who didn't receive a service = 94
dhanks23@Orcrist ~/workspace/Project_2 $
```

Output for 3 barbers, 1 waiting chair, 600 customers and 1,000 ms wait time. **69 Customers dropped.**



```
dhanks23@Orcrist ~/workspace/Project_2 $ ./sleepingBarbers 3 1 600 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 service chair [0]. # waiting seats available = 1
customer[1]: waits for Barber[0] to be done with haircut.
customer[2]: moves to service chair [1]. # waiting seats available = 1
customer[2]: waits for Barber[1] to be done with haircut.
barber[0]: starts a haircut service for Customer[1].
barber[0]: says he's done with a haircut service for Customer[1].
customer[1]: says goodbye to Barber[0].
barber[0]: calls in another customer
barber[0]: sleeps because of no customers
customer[3]: moves to service chair [2]. # waiting seats available = 1
customer[3]: waits for Barber[2] to be done with haircut.
barber[1]: starts a haircut service for Customer[2].
barber[1]: says he's done with a haircut service for Customer[2].
customer[2]: says goodbye to Barber[1].
barber[1]: calls in another customer
barber[1]: sleeps because of no customers
customer[4]: moves to service chair [0]. # waiting seats available = 1
customer[4]: waits for Barber[0] to be done with haircut.
barber[2]: starts a haircut service for Customer[3].
barber[2]: says he's done with a haircut service for Customer[3].
customer[3]: says goodbye to Barber[2].
barber[2]: calls in another customer
barber[2]: sleeps because of no customers
customer[5]: moves to service chair [1]. # waiting seats available = 1
customer[5]: waits for Barber[1] to be done with haircut.
barber[0]: starts a haircut service for Customer[4].
barber[0]: says he's done with a haircut service for Customer[4].
customer[4]: says goodbye to Barber[0].
barber[0]: calls in another customer
barber[0]: sleeps because of no customers
customer[6]: moves to service chair [2]. # waiting seats available = 1
customer[6]: waits for Barber[2] to be done with haircut.
customer[7]: moves to service chair [0]. # waiting seats available = 1
customer[7]: waits for Barber[0] to be done with haircut.
barber[1]: starts a haircut service for Customer[5].
barber[1]: says he's done with a haircut service for Customer[5].
customer[5]: says goodbye to Barber[1].
[redacted]
barber[0]: sleeps because of no customers
barber[1]: starts a haircut service for Customer[597].
barber[1]: says he's done with a haircut service for Customer[597].
customer[597]: says goodbye to Barber[1].
barber[1]: calls in another customer
barber[1]: sleeps because of no customers
# customers who didn't receive a service = 69
dhanks23@Orcrist ~/workspace/Project_2 $
```

Output for 3 barbers, 30 waiting chairs, 600 customers and 1,000 ms time. **0 Customers Dropped**



```
dhanks23@Orcrist ~/workspace/Project_2 $ ./sleepingBarbers 3 30 600 1000
barber[1]: sleeps because of no customers
barber[0]: sleeps because of no customers
barber[2]: sleeps because of no customers
customer[1]: moves to service chair [1]. # waiting seats available = 30
customer[1]: waits for Barber[1] to be done with haircut.
customer[2]: moves to service chair [0]. # waiting seats available = 30
customer[2]: waits for Barber[0] to be done with haircut.
barber[1]: starts a haircut service for Customer[1].
barber[1]: says he's done with a haircut service for Customer[1].
customer[1]: says goodbye to Barber[1].
barber[1]: calls in another customer
barber[1]: sleeps because of no customers
customer[3]: moves to service chair [2]. # waiting seats available = 30
customer[3]: waits for Barber[2] to be done with haircut.
barber[0]: starts a haircut service for Customer[2].
barber[0]: says he's done with a haircut service for Customer[2].
customer[2]: says goodbye to Barber[0].
barber[0]: calls in another customer
barber[0]: sleeps because of no customers
customer[4]: moves to service chair [1]. # waiting seats available = 30
customer[4]: waits for Barber[1] to be done with haircut.
barber[2]: starts a haircut service for Customer[3].
barber[2]: says he's done with a haircut service for Customer[3].
customer[3]: says goodbye to Barber[2].
barber[2]: calls in another customer
barber[2]: sleeps because of no customers
customer[5]: moves to service chair [0]. # waiting seats available = 30
customer[5]: waits for Barber[0] to be done with haircut.
barber[1]: starts a haircut service for Customer[4].
barber[1]: says he's done with a haircut service for Customer[4].
customer[4]: says goodbye to Barber[1].
barber[1]: calls in another customer
barber[1]: sleeps because of no customers
customer[6]: moves to service chair [2]. # waiting seats available = 30
customer[6]: waits for Barber[2] to be done with haircut.
customer[7]: moves to service chair [1]. # waiting seats available = 30
customer[7]: waits for Barber[1] to be done with haircut.
barber[0]: starts a haircut service for Customer[5].
barber[0]: says he's done with a haircut service for Customer[5].
customer[5]: says goodbye to Barber[0].
[redacted]
customer[599]: says goodbye to Barber[0].
barber[0]: calls in another customer
customer[600]: moves to service chair [0]. # waiting seats available = 30
customer[600]: waits for Barber[0] to be done with haircut.
barber[0]: starts a haircut service for Customer[600].
barber[0]: says he's done with a haircut service for Customer[600].
customer[600]: says goodbye to Barber[0].
barber[0]: calls in another customer
barber[0]: sleeps because of no customers
# customers who didn't receive a service = 0
dhanks23@Orcrist ~/workspace/Project_2 $
```

IV. Discussion

Limitations and Potential Improvements

The program has a few noticeable limitations:

- The number of customers passed in at execution is not passed to the Shop class constructor. If the number of customers could be passed to a Shop class object, a condition variable array could be made of size == number of customers, rather than number of barbers. This would simplify synchronization because no wait or signal of the condition variable at index[customerID] could be the same for two different customers.
- Professor Fukuda, who wrote the original driver.cpp, possibly does not delete all dynamically allocated memory, creating potential memory leaks.
- A single barber thread can get lost and not execute often if a large number of customer threads are created, especially since the default pthreads scheduling priority is equal for both customer and barber.
- The results of the program's execution, mainly the number of customers who did not receive service, can vary wildly as different OS processes operate in the background at different times. In fact, especially with smaller numbers of customers leaving, it's not uncommon for the results to vary by more than 100% in back-to-back executions of the same command.
- As the number of cores increases, and computing power is higher, the program actually experiences degradation in performance. Explained further below in Number of Cores.

Potential improvements:

- The number of customers could be passed to the Shop class constructor, which as mentioned above, would simplify synchronization inside the monitor.
- A destructor for the ThreadParam class could eliminate potential memory leaks in driver.cpp.
- The pthreads library allows you to adjust your **scheduling policy,** which affects the priority of threads currently running using the library. pthreads typically operates on a "round-robin" distribution of scheduling, but this could be adjusted to give the barber threads higher priority than customer threads, giving the program higher performance because a small number of barbers wouldn't be overwhelmed by a massive number of customer threads entering the shop.
- Using a sleep() command for customers that fixes the time between customer thread creation and arrival in the Shop monitor that's based on the number of cores (ie: sleep(numberCores * 1000), would negate the effect on performance experienced by more vs. fewer CPU cores.

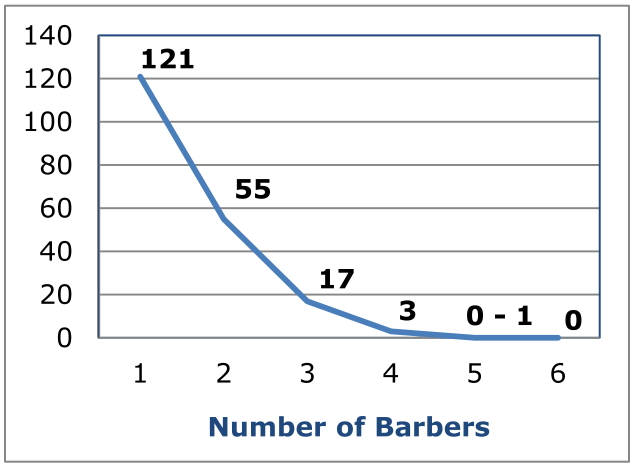
Number of Cores

As with the previous assignment, the number of cores has a direct impact on the program's performance. Unlike the first assignment, as the number of cores increases, the performance decreases and the number of customers leaving the shop due to lack of waiting chairs becomes higher. All the barber threads are created and enter the monitor one-by-one, then all the customer threads are created and call their first Shop class method. The issue with more cores is that more customer threads can run in parallel and call their first Shop class method at once, meaning customer threads are entering the barber shop at a faster rate, but the barber service time (default 1,000 microseconds) remains unchanged. So, with more customers entering more quickly, the waiting chairs are filled faster and customers begin leaving before a barber thread becomes available again.

Discussions on Step 5 – 0 Waiting Seats

As mentioned in the Number of Cores paragraph above, running this program on 4 cores instead of the original 2 core results illustrated by Professor Fukuda, results in slightly worse performance. At 4 cores, my program can reach 0 customers lost with 5 barbers working and 0 chairs, which is a compromise between the 3 barbers required at 2 cores, and the 6 barbers required by an 8-core machine, as illustrated by Professor Parsons in her sample output.

As you can see from the chart, the reduction in customer losses decreases with each barber added until it reaches 0. These values represent the average number of customers dropped over several executions, which helps eliminate outliers and unusual variation generated by other OS processes running.



This illustrates exponentially better performance for each barber added, even though the number of customer threads that can be serviced concurrently increases linearly.

Discussions on Step 6 – Number of Chairs

Theoretically, as you add more chairs, the number of customers that leave without a haircut decreases. If no barbers are available, the customer threads issue a wait condition and are blocked until the signal from barber and associated mutex are both available. Since these customer threads are blocked, ideally they consume little to no CPU cycles (except to check for signal), which frees the CPU up to work on barber threads and the customer threads currently getting a haircut. So, the higher number of customer threads that can wait in a chair, and are blocked until a signal is issued to them, which avoids performance degradation, the fewer customers will leave the shop without a haircut service.

However, in my program, as shown in the execution output, moving from 1 waiting chair to 4 chairs for a single barber produced little improvement in results. Adding even more chairs actually reduces performance, increasing the number of customers dropped. The only explanation which I can offer is that the customer threads are consuming too many CPU cycles, creating a starvation scenario for the barber thread, which renders him even less effective than if the Shop had fewer total chairs.

So, despite a single barber thread struggling to execute with more chairs, you can see from the last two pages of execution output that with 3 barbers and 3 x the customers, more chairs does in fact improve results. **For 3 barbers and 600 customers, it took only 30 chairs to reduce customers lost to 0,** so my program meets performance standards for N chairs with more than 1 barber.