Matthew Kim  
0424375  
Program 3: Barber Shop

## Documentation

### ~Shop()

The following member variables are freed:

* customer\_in\_chair[]
* in\_service[]
* money\_paid[]
* cond\_customer\_served[]
* cond\_barber\_paid[]
* cond\_barber\_sleeping[]

The pthread\_cond’s stored in the cond\_customer\_served[], cond\_barber\_paid[], and cond\_barber\_sleeping[] arrays are destroyed prior to freeing the member variables.

### void init()

Dynamically allocates the following:

* customer\_in\_chair[]
* in\_service[]
* money\_paid[]
* cond\_customer\_served[]
* cond\_barber\_paid[]
* cond\_barber\_sleeping[].

Elements in customer\_in\_chair[] are initialized to 0. Elements in in\_service[] are initialized to false. Elements in money\_paid[] are initialized to false. The pthread\_cond\_t stored in cond\_customer\_served[], cond\_barber\_paid[], and cond\_barber\_sleeping[] arrays are initialized.

The member variables cust\_drops, wait\_count, and in\_chair\_count are set to 0. mMutex and cond\_customes\_waiting are initialized.

### string int2string(int i)

Passes the parameter int to a stringstream and returns the output of the stringstream.

### void print(bool isCustomer, int person, string message)

Prints “customer” or “barber” based on the isCustomer value. Prints the int value of the person and the string message to console.

### int get\_cust\_drops()

Returns the value of cust\_drops.

### int visitShop(int custId)

Processes a customer who visits the barber shop. It returns -1 if the barber shop is full and the customer had to leave the shop. Otherwise, it returns the barberId of the barber servicing the customer.

The visitShop process diagram is shown in Figure 1. It first checks if the barbershop if full. If the barber shop is full, then the customer exits and the method returns -1. If the barbershop is not full, then it checks if the customer needs to wait for a barber. If the customer needs to wait, then it waits for the signal from any barber to the cond\_customer\_waiting variable. Next, it selects an available barber and signals the barber to wake in case the barber is sleeping.

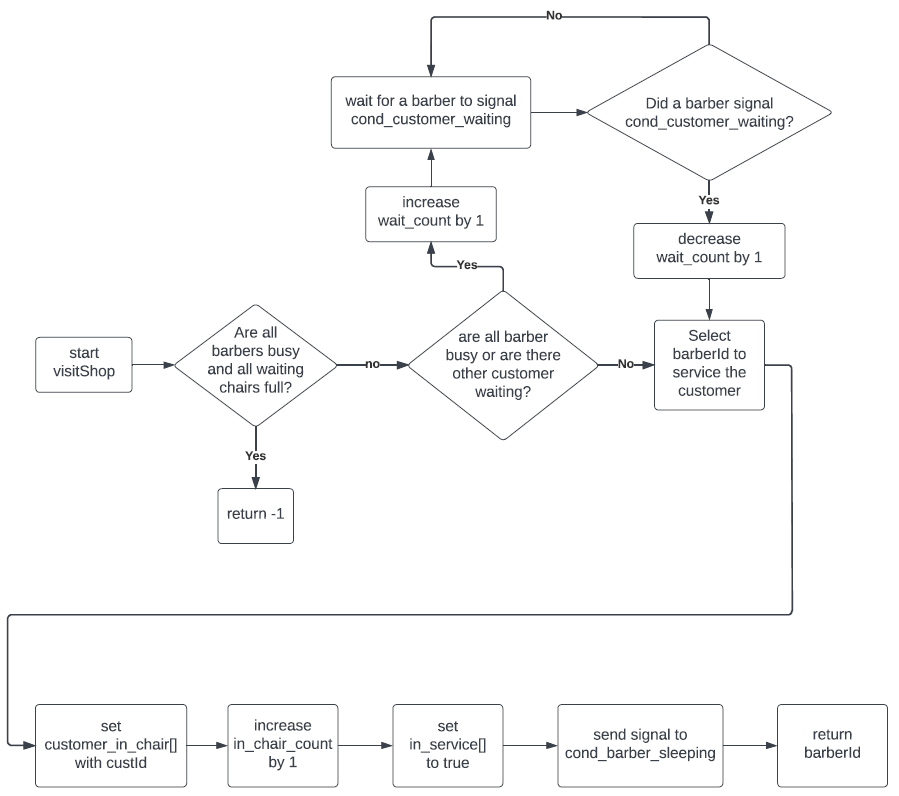


Figure 1: visitShop Process Diagram

### void leaveShop(int custId, int barberId)

Processes a customer who is serviced by a barber. The leaveShop process diagram is shown in Figure 2. It checks if the barber is currently providing service to the customer. If the barber is providing service, then the customer waits for a signal from the barber on cond\_customer\_served[]. Once the service ends, it sets money\_paid[] to true and sends a signal that the barber has been paid to cond\_barber\_paid[].

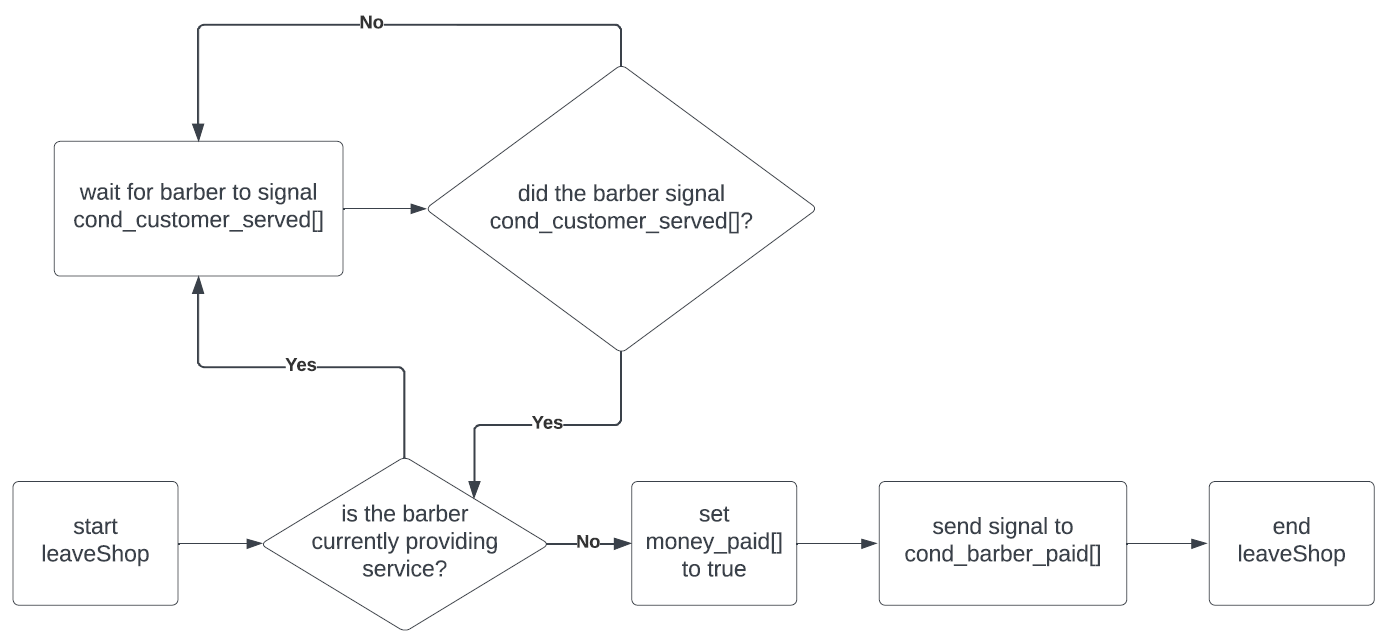


Figure 2: leaveShop Process Diagram

### void helloCustomer(int barberId)

Processes a barber who is available to service the next customer. The helloCustomer process diagram is shown in Figure 3. It checks if there are customers waiting to be serviced. If no customers are waiting, then the barber goes to sleep and waits for a signal from a customer on the cond\_barber\_sleeping[] variable. Next, it checks if a customer is seated at the barber’s chair. If no customer is seated, then the barber goes to sleep and waits for a signal from a customer on the cond\_barber\_sleeping[] variable.

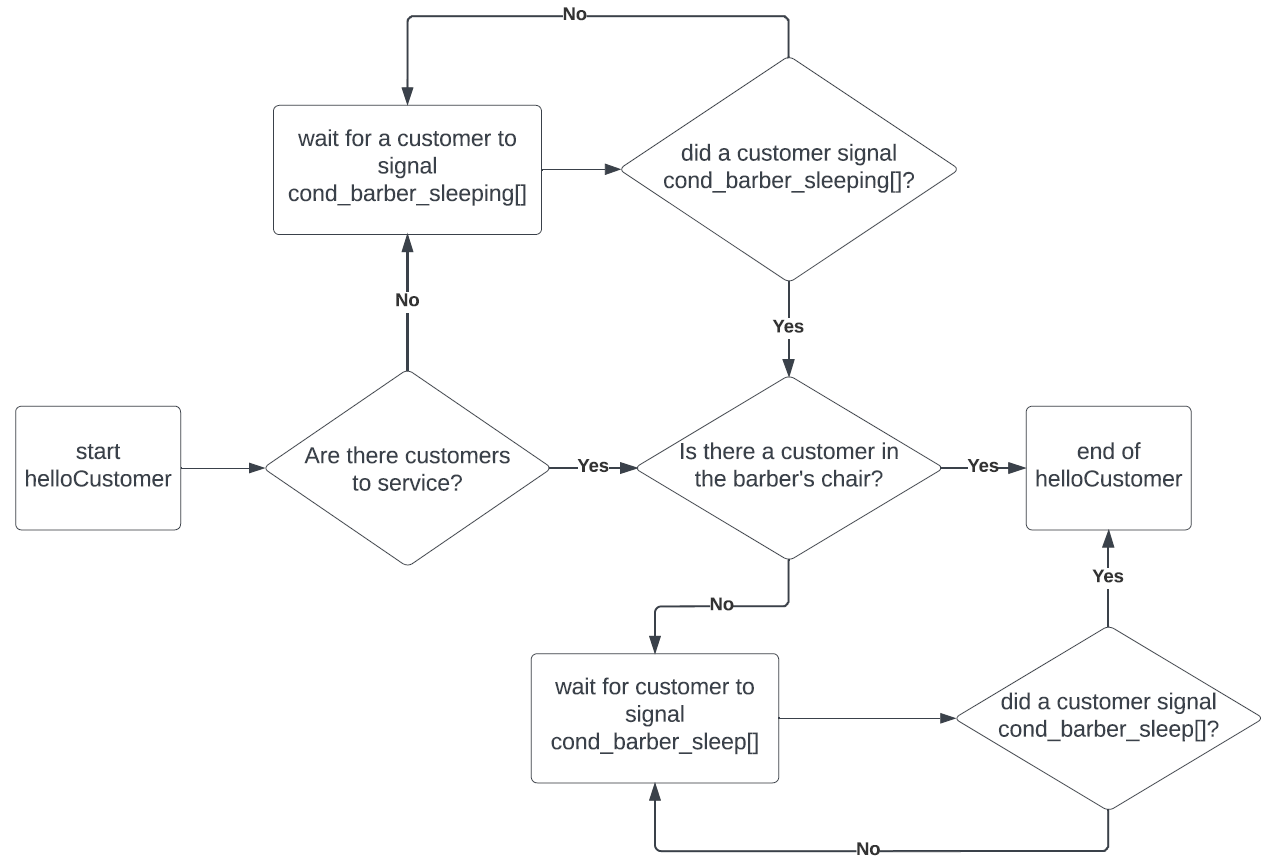


Figure 3: helloCustomer Process Diagram

### byeCustomer(int barberId)

Processes a barber who finishes servicing a customer. The byeCustomer process diagram is shown in Figure 4. It sets both in\_service[] and money\_paid[] to false to indicate that service is finished and money has not been paid. It sends a signal to the customer on the cond\_customer\_served[] communcating that the service is over. Then it checks if the customer paid. If the customer did not pay, then it waits for the customer to signal on the cond\_barber\_paid[]. Next, it sets the customer\_in\_chair to 0, decrease in\_chair\_count by 1, and sends a signal to waiting customers on the cond\_customer\_waiting variable.

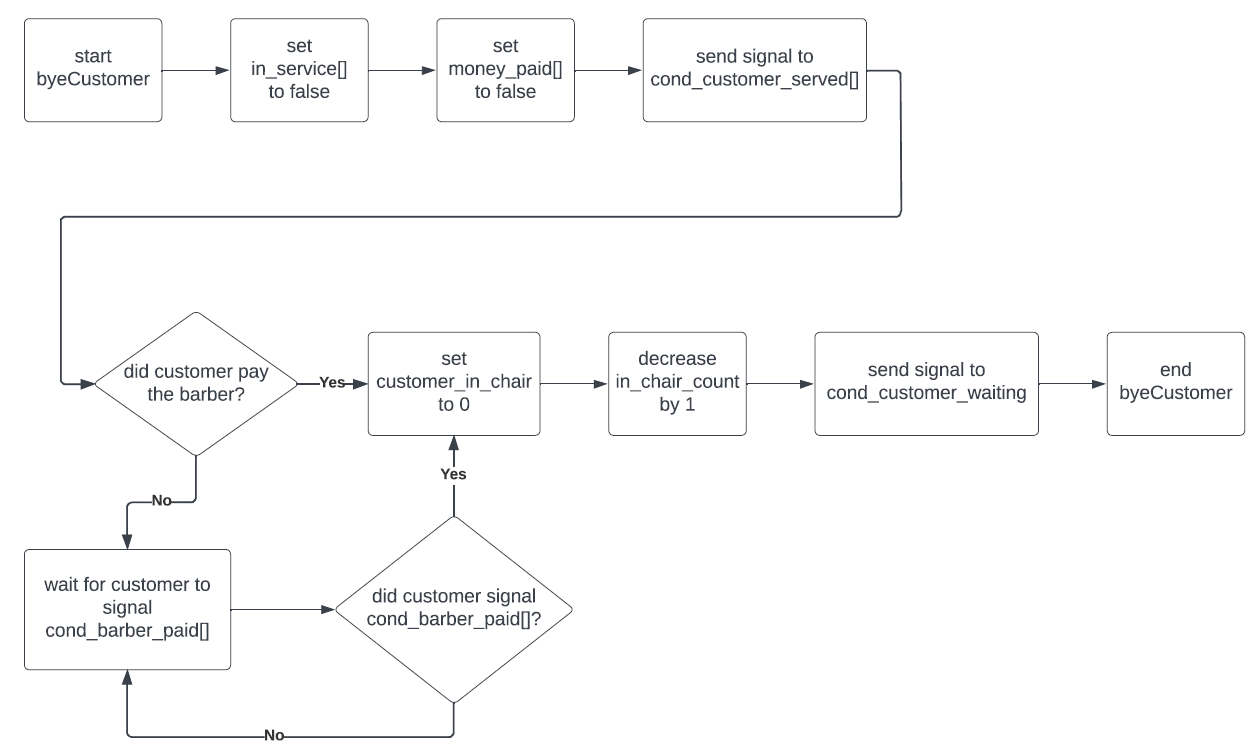


Figure 4: byeCustomer Process Diagram

## Discussion

### Limitations and Possible Extensions

The program does not process the waiting customers with a FIFO approach. Instead, it relies on the scheduler to choose the waiting customer thread that will be serviced. Thus, it is theoretically possible for a customer to wait for a long time. There are two possible extensions to address this issue. One, the waiting customers could be queued with a linked list so that a FIFO algorithm approach could be implemented. Second, the program could implement a time limit on the waiting customer so that they leave the barber shop after a certain amount of elapsed time. Both extensions would build up the barber shop model to more accurately reflect reality.

### Discussion on your answers to Step5

Discussions regarding the necessary numbers of barbers or chairs must factor in the turnover rates for each barber and the input rates of new customers. The service times for each barber is set to 1000 μsec. Thus, the average turnover rate for the barbershop will be 1000/n with n being the number of barbers. The input rate of new customers into the barbershop is not a fixed rate. The program waits a random interval between 0 and 1000 μsec before creating a new customer thread. While any particular customer may enter between 0 and 1000 μsec , on average, the barbershop will expect a new customer every 500 μsec.

For the first case, where the barbershop has one barber with 200 customers, a little over 100 chairs are necessary for all 200 customers to be served by one barber. At an average input rate of one customer every 500 μsec, it will take 100,000 μsec for 200 customers will arrive at the shop. Since the sole barber can service each customer in approximately 1000 μsec, 100 customers will have been serviced after 100,000 μsec. Thus, about 100 waiting chairs will be needed to hold the remaining customers in queue. There is some overhead time required to allow customers to get seated at the barber’s chair, pay for the service, and exit the shop. Thus, the barbershop will need slightly more than 100 chairs.

The program was run with iterations on the number of chairs. The smallest number of chairs that consistently provided service to all 200 customers was 105 chairs. This number agrees with the expected quantity of a little over 100 chairs. It is important to note that, in the worst case, where all 200 customers arrive at the beginning, 200 chairs will be required in order to service all customers.

### Discussion on your answers to Step6

In the second case, where the barbershop has zero chairs and 200 customers, approximately 3 barbers are necessary for all 200 customers to be served. Since there are no waiting chairs, the customer turnover rate of the barbershop must be faster than the input rate of new customers. If there are three barbers, they will be able to service 3 customers every 1000 μsec, resulting in a rate of one customer per 333 μsec. This is lower than the average customer creation rate of one customer every 500 μsec.

The program was run with iterations on the number of barbers. The smallest number of barbers that consistently provided service to all 200 customers was 3 barbers. This number agrees with the expected quantity of 3 barbers. It should be noted that, in the worst case, where all 200 customers arrive at the same time, 200 barbers are necessary in order to service all customers.