Braden Wolf

CSS 430 A

Prof. Dimpsey

Program 4

Documentation

Shop class Description:

Acts as structure for barber threads and customer threads to interact through

a monitor.

Assumptions:

-- valid params sent to constructor

Implementation

-- uses one mutex for all critical sections

-- uses one condition for all customers waiting: cond_customers_waiting_

-- uses a vector to store conditions for barbers sleeping, one per barber: cond_barber_sleeping_

-- uses a vector to store conditions for customer served, one per barber: cond_customer_served_

-- uses a vector to store conditions for barber paid, one per barber: cond_barber_paid_

Explanation:

Barber threads are generated before customer threads. Each barber goes in a loop of helloCustomer function, sleeping for the service time, and then byeCustomer function. Each customer will call the visitShop function and if there are available seats it will call the leaveShop function.

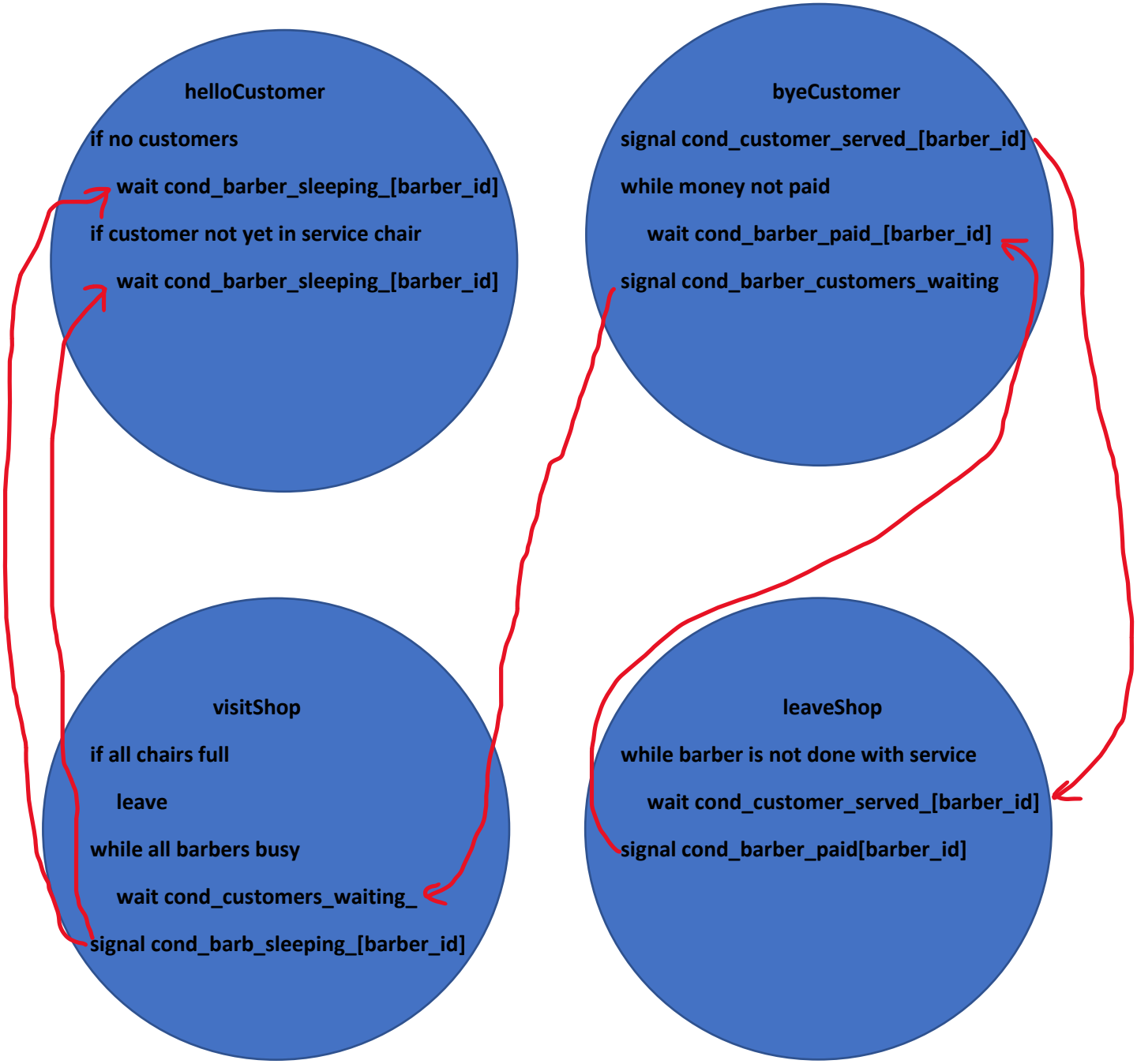
In the helloCustomer function, the barber will grab the mutex, if there are no customers in waiting chairs or in its service chair then it will wait with that specific barber's cond_barber_sleeping_ signal. After that, the barber will check if the customer is in its service chair, if not it will wait with that specific barber's cond_barber_sleeping_ signal. After that, the barber will release the mutex.

In the byeCustomer function, the barber will grab the mutex, then signal that specific barber's cond_customer_served signal. After that, if the customer has not paid, then the barber will wait with that specific barber's cond_barber_paid signal. After that, the barber will signal the cond_customers_waiting signal. After that, the barber will release the mutex.

In the visitShop function, the customer will grab the mutex, then if all the service and waiting chairs are full it will release the mutex and return from the function. If all the chairs are not full, then the customer checks if all the barbers are busy, if they are the customer will wait with the cond_customer_waiting signal. After that, the customer will find a barber that has an available chair and signal that specific barber's cond_barber_sleeping signal. After that, the customer will release the mutex.

In the leaveShop function, the customer will grab the mutex, then while the barber is in service it will wait with its specific barber's cond_customer_served signal**.** After that, the customer will signal its specific barber's cond_barber_paid signal. After that, the customer will release the mutex.

Illustration:



Discussion

Step 4:

For both run cases in the example output, for each first case of a barber's interaction with a customer,
"customer[id]: wait for barber[id] to be done with hair-cut"
was printed before:
"barber [id]: starts a hair-cut service for customer[id]."

Then after that first case,
"barber [id]: starts a hair-cut service for customer[id]"
was printed before:
"customer[id]: wait for barber[id] to be done with hair-cut."

However, on my output,
"customer[id]: wait for barber[id] to be done with hair-cut"
always came before:
"barber [id]: starts a hair-cut service for customer[id]."

When I ran ./sleepingBarbers 1 1 10 1000 on my code, I almost never got 0 customers who didn't receive a service. The example output had 0 customers who didn't receive a service, so there were more lines of output because there are 5 lines of output for every customer barber interaction when the customer gets served but there is only one line of output when the customer does not get served.

When I ran ./sleepingBarbers 3 1 10 1000 on my code, so my output had the same number of lines as the example output. My output and example output both had 1 waiting chair available each time the number of waiting chairs got displayed, so both programs never used any waiting chairs.

Step 5: ./sleepingBarbers 1 chair 200 1000

At 96 chairs I started getting 0 customers who didn't receive a service.

At 103 chairs I got 10/10 times that 0 customers didn't receive a service.

Step 6: ./sleepingBarbers barbers 0 200 1000

At 5 barbers I got 6/10 times that 0 customers didn't receive a service.

Limitations:

If last barber number of customers who didn't receive a service will not print to console.

If you go over about 32700 threads the program will segmentation fault.

Extensions:

If more threads are desired to use, a distributed system could be implemented to allow more storage for data.

Checking for validity of passed in parameters to the shop class in the constructor could be added.

Checking for only customer threads using visitShop and leaveShop.

Checking for only barber threads using helloCustomer and byeCustomer.