Kelly Kauffman

CSS 503

5.6.2023

Program 2 Write-Up

# Documentation

The shop.cpp file starts with defining init(). Init() includes the initializers for pthread mutex and condition variables. There are four pthread condition variables with three of these four, defined as an array. The size of these arrays is determined by the number of barbers. These condition variable arrays help match up the specific barber with the specific customer they are serving based on the barber’s id. To also help keep barber/customer interactions separate from each other, the in\_service\_, money\_paid\_ and customer\_in\_chair\_ variables are all defined as arrays with the barber id being the index. Every array is dynamically allocated in the shop.h file.

Graphical user interface, text

Description automatically generated Figure 1 – init() in shop.ccp

The next important piece of the shop.cpp file is the visitShop() function that each customer thread calls. To support multiple barbers, I added a check to see if all the barber chairs are full. This check helps the customer know if they can grab a barber chair right away or need to try to find an available waiting chair.

Graphical user interface, text, application

Description automatically generated Figure 2 – check for open barber chairs

With the addition of multiple barbers, I also added another for loop after the waiting chair logic to match up the customer with the first available barber. The barber becomes linked with the customer through the barber’s id.

Graphical user interface, text

Description automatically generated Figure 3 – Assign customer to barber

The assigned barber id was returned in visitShop(), so that it could be passed to the next customer function, leaveShop(), in the thread customer() function defined in the driver file. This is to continue to link the customer and barber through the haircut and payment process. The barber and customer signal to and wait for each other through condition variables (served and paid) marked by the barber’s id.

Graphical user interface, text, application

Description automatically generated Figure 4 – leaveShop() customer function

The barber’s two functions are helloCustomer() and byeCustomer(). Both functions work through the steps of the haircut through the barber’s perspective. Condition variables (sleeping, served, and paid) are determined by the barber’s id to keep the barber linked with a specific customer thread. The helloCustomer() function has added functionality for the barber to check his chair and the waiting chairs for any customers. If none, then the barber goes to sleep.

Graphical user interface, text, application

Description automatically generated Text

Description automatically generated

Figures 5 and 6 – the barber functions helloCustomer() and byeCustomer()

The last note for shop.cpp is that all customer and barber functions are bookended with a mutex\_lock and mutex\_unlock to protect the critical sections and allow for thread synchronization. Since the barber id also heavily supports thread synchronization, it is protected even outside the critical section. For example, the barber’s id is passed directly to leaveShop(), helloCustomer(), and byeCustomer() and not used as a global variable in the Shop class(). There is a barber\_ variable created in visitShop() to capture the available barber’s id, but since it is created within visitShop(), it will only be available to that specific customer thread.

# Discussions

## Limitations of the Program

One of the limitations of this program is the matching of waiting chairs that recently become available with new customers that arrive. There can be a situation where a barber thread receives payment from a customer, but before they can signal to a waiting customer a new customer arrives at the shop, sees there are no waiting chairs, and leaves the shop immediately. A possible solution to this would be to have a customer in this situation pause for short time to see if a chair opens and if not, then leave.

Another limitation I see with this program is the random time of sleep before each customer thread is created. It is possible for the randomness to skew toward shorter time intervals between customer thread spawns, which results in more customers needing to leave the shop right away, especially for runs with large customer counts. This is not good for the barber(s)’s business.

The last limitation I see with this program is the variation of the number of CPU cores between different machines and clock skews cause more inconsistency. These factors cause the program to run differently between different machines, which makes it hard to compare results between different users of the program.

## Extensions of the Program

A possible extension of this program is to include a break schedule for the barbers. For instance, a barber needs to take a break (or sleep) after helping a specified number of customers. To implement this, each barber thread would need to keep track of how many customers they have helped. This could be done with a counter defined in the ThreadParam class. After helping each customer, individual barber threads could check the counter. If the counter is at a specified number, then the barber would “sleep” (i.e. pthread\_cond\_wait with the cond\_barber\_sleeping condition). The specified number of consecutive customers could be defined as a command line argument.

Another extension of this program could allow for customers to specify the type of service they would like, for example, a regular haircut, a shave, or a haircut and shave together. To implement this extension, the different services could be defined by integer values in an array or as an emum defined in the ThreadParam class. The services could be chosen at random and passed to the barber thread via the visitShop() function. The Shop class could have a serviceType variable so the barber can access the type of service. Each service would take a different amount of time. The times could be defined in an array in the Shop class. The barber and customer thread would have to sleep for that specified amount of time, before moving to the payment step.

## Step #5 Discussion

For my program it takes approximately 85 chairs to serve all 200 customers with just one barber. This is shown in step5\_output.txt. Testing the top of the provided range in the program 2 instructions, 60 chairs, resulted in 31 customers that did not get served. Even with 80 chairs, 8 customers still did not get served. The 60 and 80 chair outputs are not documented in a text file. Because of the variation in CPU cores, clock skews and random time intervals between spawning customer threads, it may be possible to have 84 chairs and still get all the customers through the barber shop. When I ran this 84 chair scenario, only 3 customers were left without service. This output is shown in step5\_output2.txt.

## Step #6 Discussion

For my program it would take approximately 4 barbers to serve all 200 customers when there are no waiting chairs available. When I ran my program with 4 barbers only 2 customers were not served. I think with the variation in CPU cores, clock skews, and random time intervals between spawning customer threads, it is possible for 4 barbers to serve all 200 customers. With 5 barbers it is guaranteed all customers will be served. The file, step6\_output2.txt, shows 0 customers left.