SLEEPING BARBER

Adilet Kuroda

Table of Contents

Introduction

This project involves synchronizing barber and customer threads in a way that they have coordinated access to shared resources. The Shop class acts as a monitor to coordinate barbers and customers in a way that shared resources are accessed and updated accordingly, so that there is no race condition. Shared resources include chairs that customers wait on and barber chairs. The Shop class utilizes mutex, conditional signals, and waits to achieve this synchronization.

Overview

The constructors initialize the mutex, conditional variables, and other data members. The destructor ensures that all the resources are freed once or before the object goes out of scope. In addition to the constructors and destructor, there are several utility member functions, such as **int2string**, that provide supporting functionalities. Main functionaries are provided by **visitShop, leaveShop, helloCustomer and byeCustomer.**

All the barbers represented as a struct and stored in an array. Each barber has associated conditional signals and variables that allows barber receive barber specific conditions. In addition, there is a member variable **customer_in_chair** an integer array, which allows to have each barber a specific chair. This also allows to have a relationship between barber and customer so that they can interact accordingly.

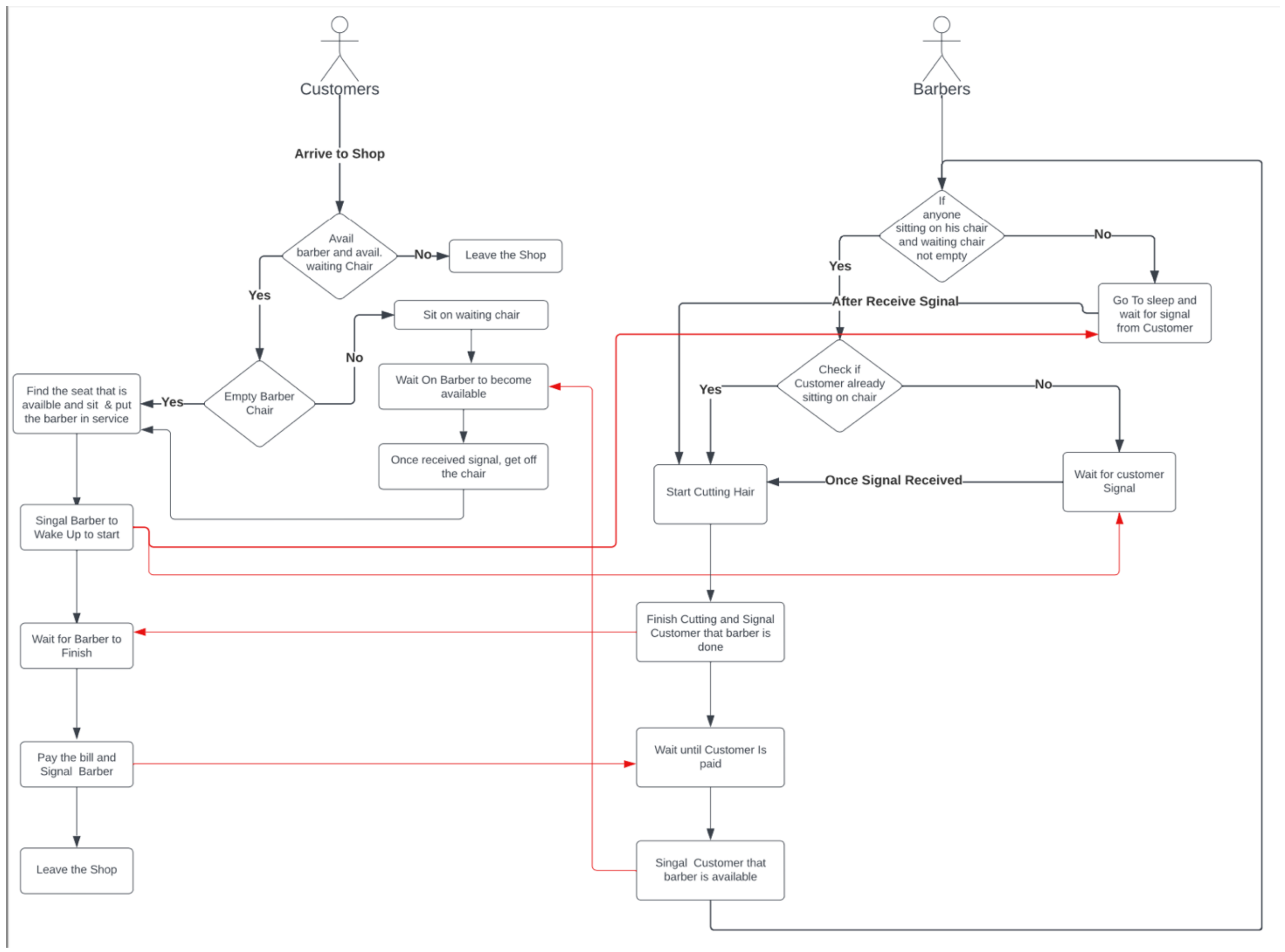
Customer: Visit Shop and Leave Shop

The **visitShop** member functions allows customers to visit to the shop and find a barber to get a haircut. If there is no waiting chair and all the barbers are busy, customer leaves the shop. If all the barbers are busy but there is still available chair than customer than customer takes a seat on queue and waits until barber becomes available. Once the barber is available, customer picks the barber and puts the barber in service by signaling the barber to wake up and start cutting hair. The function **LeaveShop** facilitates customer to wait for the barber to finish the service and pay for the bill. Once the bill is paid, customer leaves the shop, which is the end of customer thread. Please refer to **Figure 1** for details of customer interaction with barber.

Barber: Hello Customer and Bye Customer

The **helloCustomer and byeCustomer** member functions facilitate barber thread responsibilities. Unlike customer thread, barber thread continuously loops through until all the customers get service. **helloCustomer** allow barber to check if there is any person waiting or if customer took barber's chair. If not, barber goes to sleep and waits for signal from customer to wake up. If there are available customers, barber checks if anyone took the chair and waits for signal if the customer is not sitting on barber's chair. Once the wake-up signal is received, barber starts haircut. The **byeCustomer** member function handles the barber to finish the service. Barber finishes service and signals **LeaveShop** function so that customer pays the bill and waits until receives confirmation that customer is paid. Once the customer pays for the service, barber removes the customer from his/her chair and signal the possible waiting customer that barber is ready for next haircut. Refer to **Figure 1 (**red arrows represent signals) for detailed interaction between customer and barber.

Figure 1: Interaction between barber and customer



Customers
Barbers
Arrive to Shop
Avail barber and avail. waiting Chair
No
Leave the Shop
Yes
Sit on waiting chair
No
Wait On Barber to become available
Empty Barber Chair
Yes
Find the seat that is availble and sit & put the barber in service
Once received signal, get off the chair
Singal Barber to Wake Up to start
Wait for Barber to Finish
Pay the bill and Signal Barber
Leave the Shop
If anyone sitting on his chair and waiting chair not empty
Yes
No
Go To sleep and wait for signal from Customer
After Receive Sginal
Check if Customer already sitting on chair
Yes
No
Start Cutting Hair
Once Signal Received
Wait for customer Signal
Finish Cutting and Signal Customer that barber is done
Wait until Customer Is paid
Singal Customer that barber is available

Limitation and Possible Extension

I have tried to run for number of customers up to 100 thousand and once the customer count reaches about 32 thousand, the Linux machine stops working. I adjusted number of barbers and seats, but outcome did not change. I would assume that app in current state does not scale for bigger customer count due to possible inefficiency in synchronization, maybe hardware limitation or pthread library.

In addition, current implementation does not impose fairness even though waiting queue designed to be first come fist serve. In a situation when customer thread enters the shop and if there is available barber, it directly jumps to chair and starts getting haircut even though there are some customers that is in the line. This app could be extended to impose some form of stricter mechanism where customers threads served in the order they arrive.

Step 5: Running program with different number of chairs. (`1 chair 200 1000`)

Since the customer arrival time is randomized, there number of chairs it requires differs little bit every time this app runs. I have run multiple times and approximately it takes about 84 to 87 chairs to service all the customers. I plotted one of the outcomes on Figure 2. It shows relationship between number of seats and number of customers who did not receive haircut.

Step 6: Run your program with different number of barbers. (`barbers 0 200 1000`)

Figure 3 show the relationship between number of barbers to number of customers who did not receive service. Since the customer arrival time is randomized, most of the time it takes about 5 barbers so that every customer gets service. There are few times where it only took 4 barbers to achieve the goal.

In conclusion, we can minimize the number of customers who did not receive service by balancing between number of waiting chairs and barbers.

Figure 2:

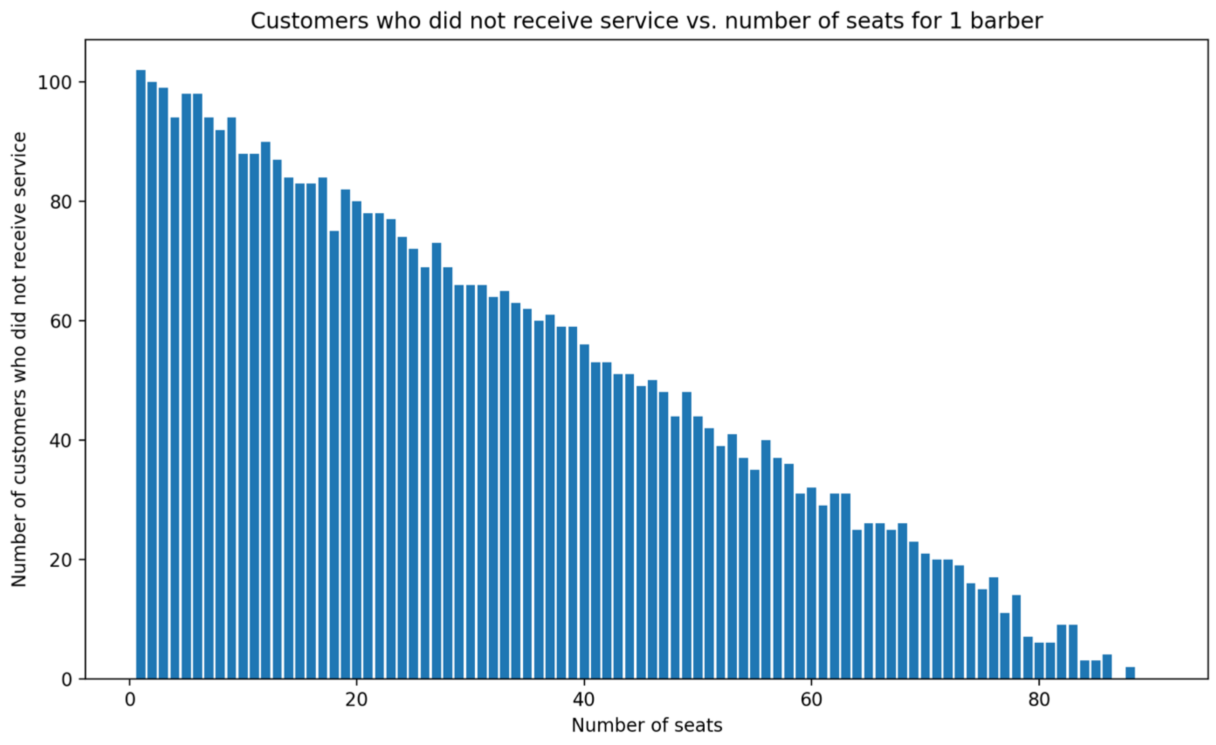
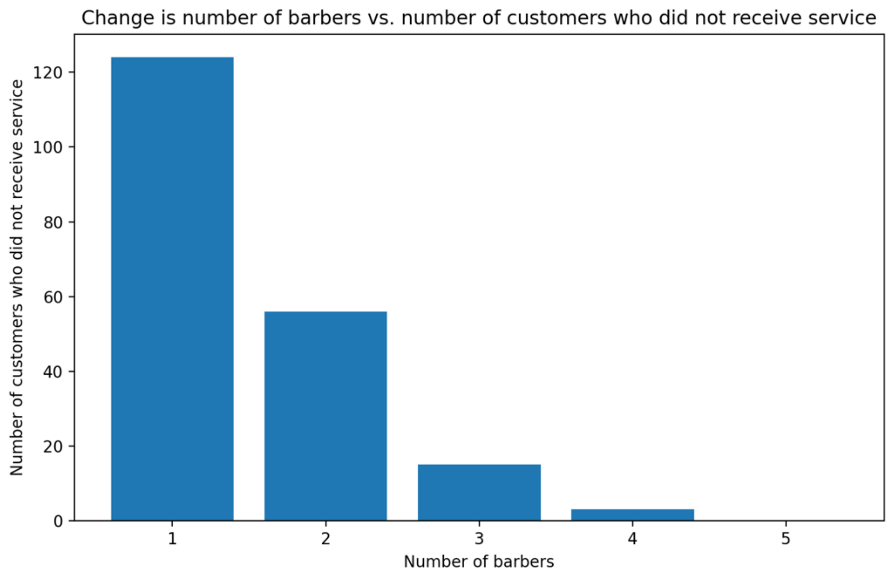


Figure 3:


Change is number of barbers vs. number of customers who did not receive service
Number of customers who did not receive service
0
20
40
60
80
100
120
1
2
3
4
5
Number of barbers