Chase Howard
2021-05-10

Program 2: Sleeping Barbers Documentation

Purpose

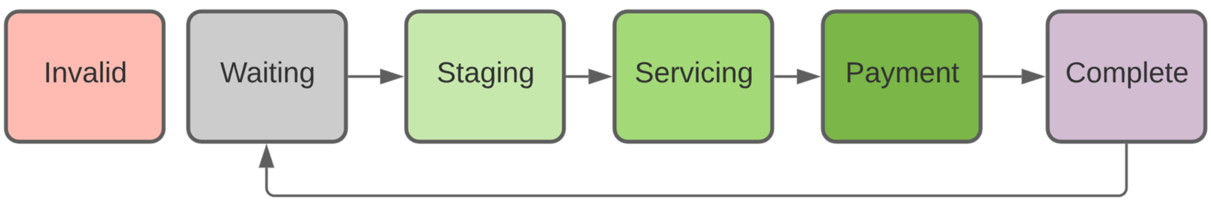
This program is to be used to demonstrate the use of a monitor class (Shop) and condition variables to control access to shared data structures to have a program free of race conditions with well-defined behavior.

Program Description

This program consists of a Store with two internal structures, Barbers and Customers. Each barber can only service a single customer at a time at a Service Chair. Customers enter the store at a randomized time and if no Service Chairs are available, they will wait at a Waiting Chair.

Synchronization

A single mutex shared between Customers and Barbers protects against modification of shared data. Conditional variables are used to signal across Customers and Barbers at various stages of the life cycle of a process as defined by states below:

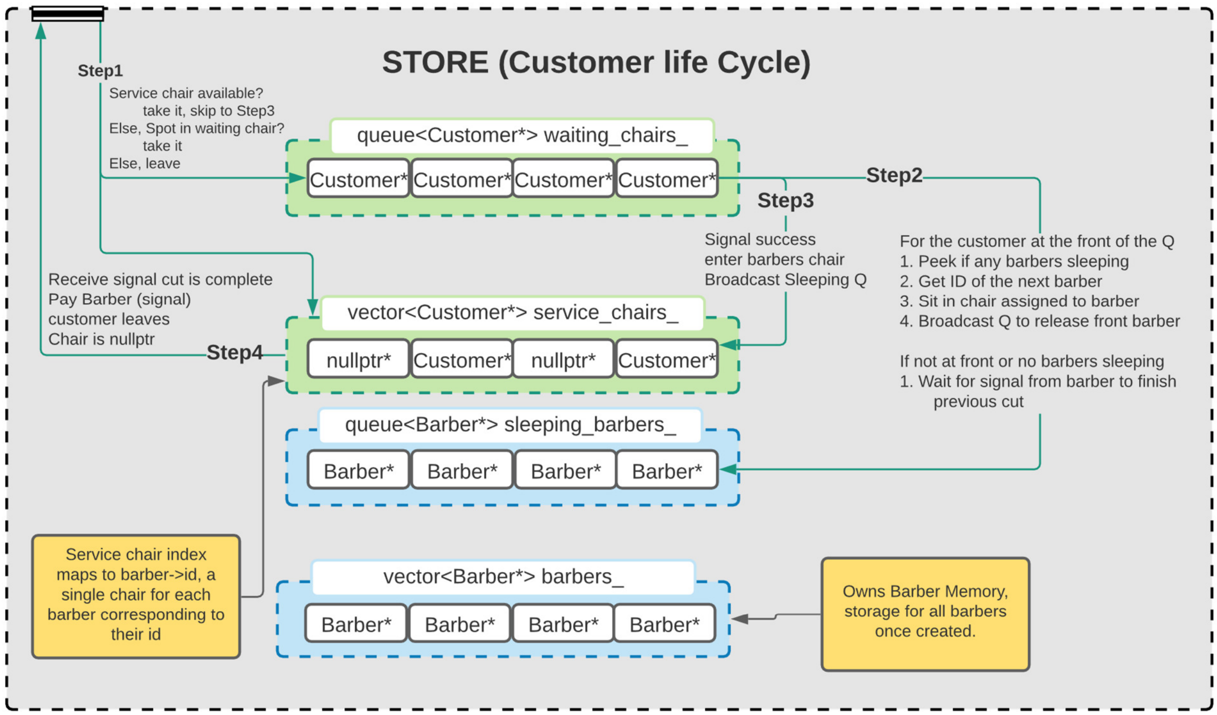


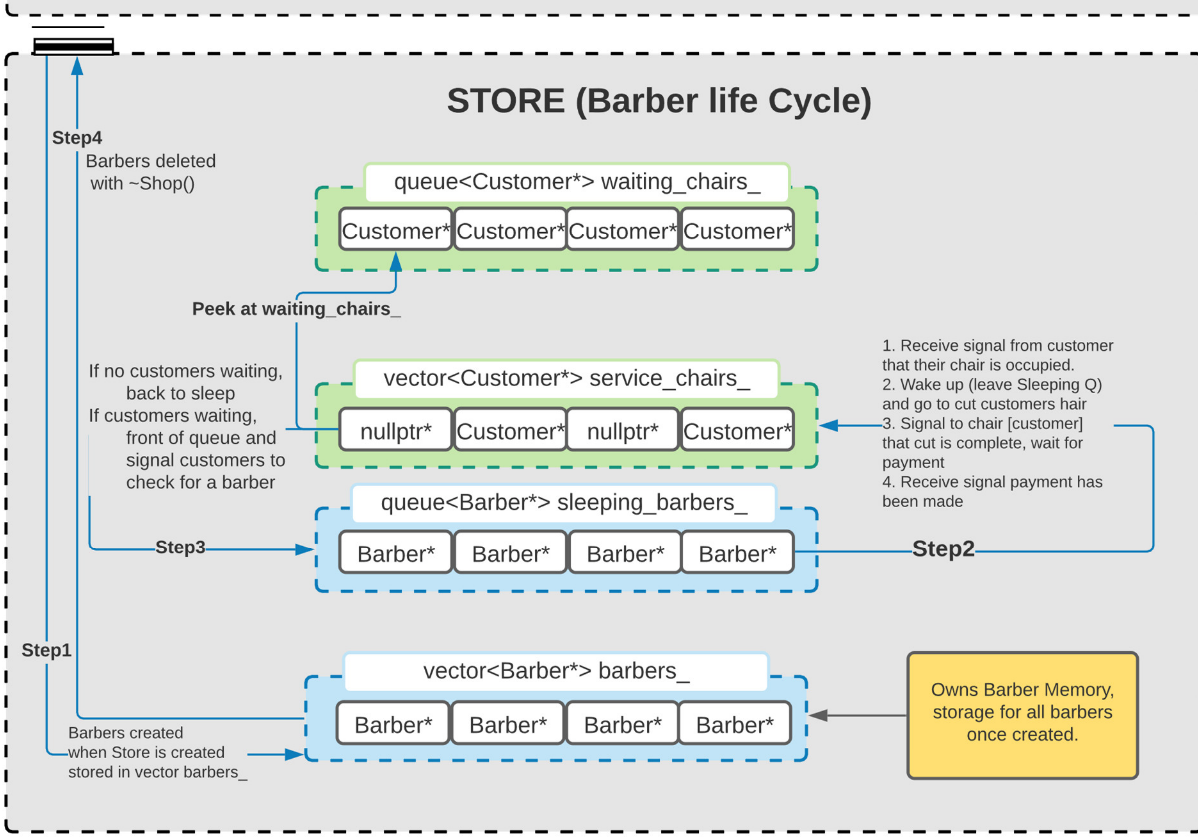
The conditional variables listed below labeled with [B] indicate that the signals is Broadcast, in this case the broadcast signals go to all those in the waiting / sleeping queues. Signals labeled with [S] indicate that a unique signal is sent to the barber or customer of interest. In this case it is the Service chair that is being signaled; however, both the customer and barber are aware they are occupying the service chair at the time of the signal.

Table 1: State Transitions and Signals [B: Broadcast, S: Signal]

State	**Customer**	**Transition Signal**	**Barber**
Invalid	Init state – describes drop_cust		Init State
Waiting	Sitting in waiting_chair_	[*B*]cond_wake_barber_	Sleeping in sleeping_barbers_
Staging	Waiting for barber in Service Chair	[*B*]cond_next_cust_	Starting Haircut
Servicing	Waiting for haircut to finish	[*S*]cond_cust_served_	Finishing Haircut
Payment	Payment requested by barber		Requests Payment
Complete	Say goodbye, Leave shop	[*S*]cond_barber_paid_	Payment Made–next customer

Life Cycle Diagrams





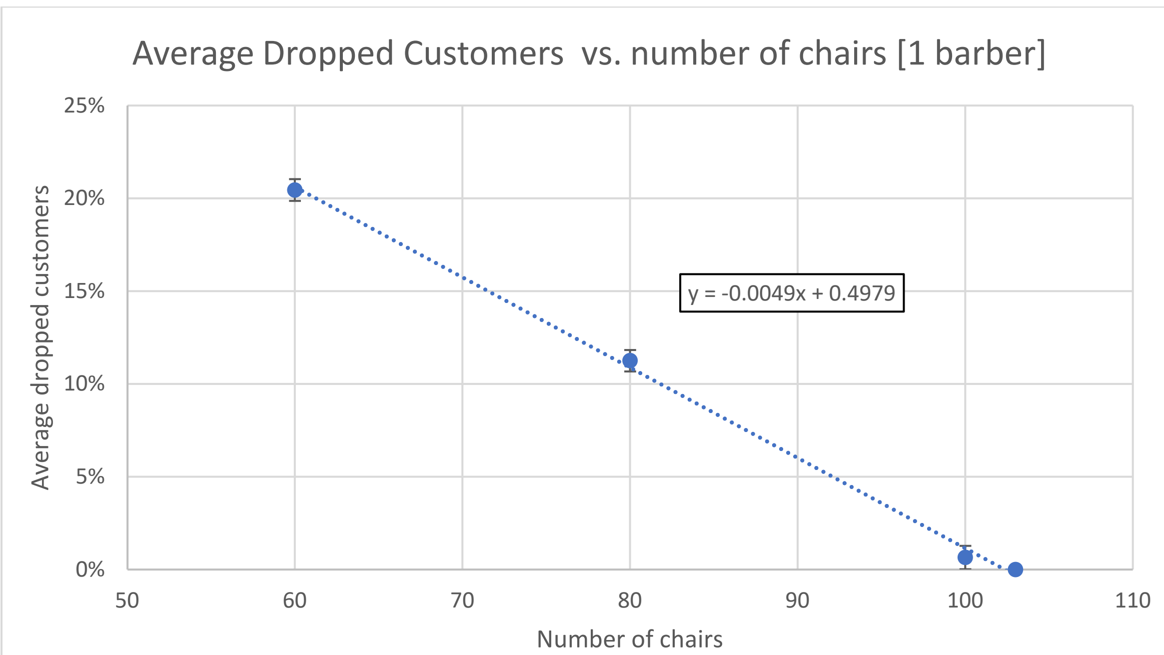
Discussion Questions

Step 5: Approximately how many waiting chairs would be necessary for all 200 customers to be served by 1 barber?

Answer: With the current implementation of my program and the hardware used for testing, it would take approximately 105 seats to ensure a single barber could successfully serve 200 customers

Table 2: Step 5 Summary Table [50 samples per seat#]

#seats	**60**	**80**	**100**	**103**
Min	20%	10%	0%	0%
Max	22%	13%	2%	0%
Avg	20%	11%	1%	0%
Stddev	0.01	0.01	0.01	0.00
Percent zeros	0%	0%	35%	100%

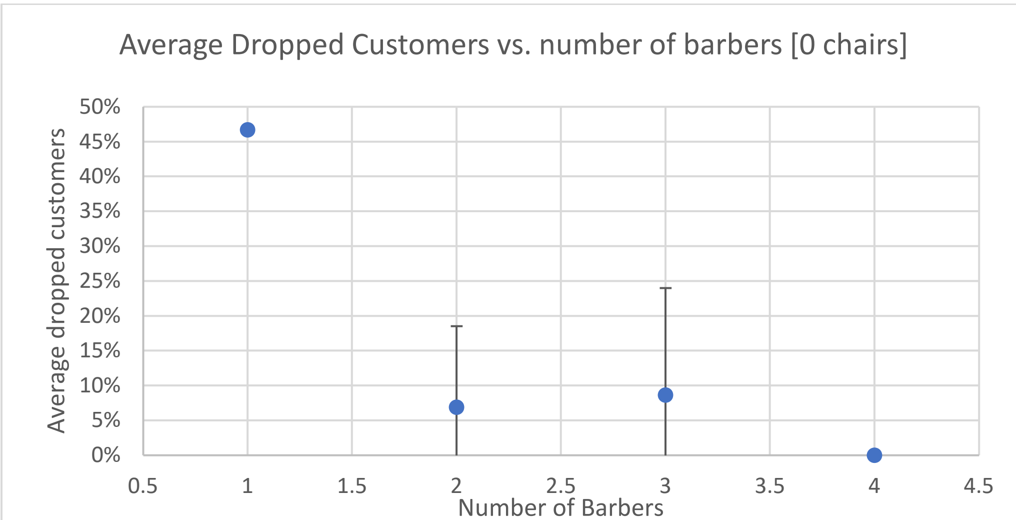


Step 6: Approximately how many barbers would be necessary for all 200 customers to be served without waiting?

Answer: With the current implementation and hardware setup, one would need 4 barbers to ensure that no customer needed to wait before getting a haircut.

Table 3: Step 6 Summary Table [50 samples per barber#]

#barbers	1	2	3	4
min	46%	1%	0%	0%
max	48%	43%	38%	0%
avg	47%	7%	9%	0%
stddev	0.00	0.12	0.15	0.00
Percent zeros	0%	0%	64%	100%



Program Limitations and Improvements

- *Limitation*: Currently, the shop operates with a single mutex (couldn't get two working)
    - *Improvement*: There is a possibility there could be two mutexes for each the customers and barbers. There is a single structure in which they both commonly access which is the service_chairs_. This may prove an opportunity to improve speed of operation of the program by segregating the two different mutexes.
- *Limitation*: Barber ID's are assigned sequentially from 1 – nBarbers
    - *Improvement*: Non sequential (more complex) id's could be stored within a hashmap, the service chairs could also be stored within a hashmap with a common hashing function so that there is still a 1-1 relationship.
- *Limitation*: Lacking OOP approach
    - Possibility of inherited structure between customers and barbers to allow for further modularity and scalability in the future if you were to add different types of customers and data items such as cost of a haircut, child, adult, haircut type, preferred barber etc. Possibly a factor method for construction.