Report

1. Execution output:

1barber_1chair_10customer_1000stime	3barber_1chair_10customer_1000stime
[zhengsq@csslab9 ver2]$ ./a.out 1 1 10 1000 barber [0]: sleeps because of no customers. customer[1]: moves to the service chair[0]. # waiting seats available = 1 customer[1]: wait for barber[0] to be done with hair-cut barber [0]: starts a hair-cut service for customer[1] customer[2]: takes a waiting chair. # waiting seats available = 0 barber [0]: says he's done with a hair-cut service for a customer[1] customer[1]: says good-bye to the barber[0] barber [0]: calls in another customer customer[2]: moves to the service chair[0]. # waiting seats available = 0 customer[2]: wait for barber[0] to be done with hair-cut barber [0]: starts a hair-cut service for customer[2] customer[3]: takes a waiting chair. # waiting seats available = 0 barber [0]: says he's done with a hair-cut service for a customer[2] customer[2]: says good-bye to the barber[0] barber [0]: calls in another customer customer[3]: moves to the service chair[0]. # waiting seats available = 0 customer[3]: wait for barber[0] to be done with hair-cut barber [0]: starts a hair-cut service for customer[3] customer[4]: takes a waiting chair. # waiting seats available = 0 barber [0]: says he's done with a hair-cut service for a customer[3] customer[3]: says good-bye to the barber[0] barber [0]: calls in another customer customer[4]: moves to the service chair[0]. # waiting seats available = 0 customer[4]: wait for barber[0] to be done with hair-cut barber [0]: starts a hair-cut service for customer[4] customer[5]: takes a waiting chair. # waiting seats available = 0 barber [0]: says he's done with a hair-cut service for a customer[4]	[zhengsq@csslab9 ver2]$ ./a.out 3 1 10 1000 barber [0]: sleeps because of no customers. barber [1]: sleeps because of no customers. barber [2]: sleeps because of no customers. customer[1]: moves to the service chair[0]. # waiting seats available = 1 customer[1]: wait for barber[0] to be done with hair-cut barber [0]: starts a hair-cut service for customer[1] customer[2]: moves to the service chair[1]. # waiting seats available = 1 customer[2]: wait for barber[1] to be done with hair-cut barber [1]: starts a hair-cut service for customer[2] barber [0]: says he's done with a hair-cut service for a customer[1] customer[1]: says good-bye to the barber[0] barber [0]: calls in another customer barber [0]: sleeps because of no customers. customer[3]: moves to the service chair[2]. # waiting seats available = 1 customer[3]: wait for barber[2] to be done with hair-cut barber [2]: starts a hair-cut service for customer[3] barber [1]: says he's done with a hair-cut service for a customer[2] customer[2]: says good-bye to the barber[1] barber [1]: calls in another customer barber [1]: sleeps because of no customers. customer[4]: moves to the service chair[0]. # waiting seats available = 1 customer[4]: wait for barber[0] to be done with hair-cut barber [0]: starts a hair-cut service for customer[4] barber [2]: says he's done with a hair-cut service for a customer[3] customer[3]: says good-bye to the barber[2] barber [2]: calls in another customer barber [2]: sleeps because of no customers. customer[5]: moves to the service chair[1]. # waiting seats available = 1 customer[5]: wait for barber[1] to be done with hair-cut barber [1]: starts a hair-cut service for customer[5] barber [0]: says he's done with a hair-cut service for a customer[4] customer[4]: says good-bye to the barber[0] barber [0]: calls in another customer barber [0]: sleeps because of no customers.

customer[6]: leaves the shop because of no available waiting chairs.
customer[4]: says good-bye to the barber[0]
barber [0]: calls in another customer
customer[5]: moves to the service chair[0]. # waiting seats available = 0
customer[5]: wait for barber[0] to be done with hair-cut
barber [0]: starts a hair-cut service for customer[5]
customer[7]: takes a waiting chair. # waiting seats available = 0
customer[8]: leaves the shop because of no available waiting chairs.
barber [0]: says he's done with a hair-cut service for a customer[5]
customer[5]: says good-bye to the barber[0]
barber [0]: calls in another customer
customer[7]: moves to the service chair[0]. # waiting seats available = 0
customer[7]: wait for barber[0] to be done with hair-cut
barber [0]: starts a hair-cut service for customer[7]
customer[9]: takes a waiting chair. # waiting seats available = 0
barber [0]: says he's done with a hair-cut service for a customer[7]
customer[7]: says good-bye to the barber[0]
customer[10]: leaves the shop because of no available waiting chairs.
barber [0]: calls in another customer
customer[9]: moves to the service chair[0]. # waiting seats available = 0
customer[9]: wait for barber[0] to be done with hair-cut
barber [0]: starts a hair-cut service for customer[9]
barber [0]: says he's done with a hair-cut service for a customer[9]
customer[9]: says good-bye to the barber[0]
barber [0]: calls in another customer
barber [0]: sleeps because of no customers.
customers who didn't receive a service = 3

customer[6]: moves to the service chair[2]. # waiting seats available = 1
customer[6]: wait for barber[2] to be done with hair-cut
barber [2]: starts a hair-cut service for customer[6]
barber [1]: says he's done with a hair-cut service for a customer[5]
customer[5]: says good-bye to the barber[1]
barber [1]: calls in another customer
barber [1]: sleeps because of no customers.
customer[7]: moves to the service chair[0]. # waiting seats available = 1
customer[7]: wait for barber[0] to be done with hair-cut
barber [0]: starts a hair-cut service for customer[7]
barber [2]: says he's done with a hair-cut service for a customer[6]
customer[6]: says good-bye to the barber[2]
barber [2]: calls in another customer
barber [2]: sleeps because of no customers.
customer[8]: moves to the service chair[1]. # waiting seats available = 1
customer[8]: wait for barber[1] to be done with hair-cut
barber [1]: starts a hair-cut service for customer[8]
barber [0]: says he's done with a hair-cut service for a customer[7]
customer[7]: says good-bye to the barber[0]
barber [0]: calls in another customer
barber [0]: sleeps because of no customers.
customer[9]: moves to the service chair[2]. # waiting seats available = 1
customer[9]: wait for barber[2] to be done with hair-cut
barber [2]: starts a hair-cut service for customer[9]
barber [1]: says he's done with a hair-cut service for a customer[8]
customer[8]: says good-bye to the barber[1]
barber [1]: calls in another customer
barber [1]: sleeps because of no customers.
customer[10]: moves to the service chair[0]. # waiting seats available = 1
customer[10]: wait for barber[0] to be done with hair-cut
barber [0]: starts a hair-cut service for customer[10]
barber [2]: says he's done with a hair-cut service for a customer[9]
customer[9]: says good-bye to the barber[2]
barber [2]: calls in another customer
barber [2]: sleeps because of no customers.
barber [0]: says he's done with a hair-cut service for a customer[10]
customer[10]: says good-bye to the barber[0]

	barber [0]: calls in another customer barber [0]: sleeps because of no customers. # customers who didn't receive a service = 0

Compare the results above with sample files, although they are not exactly the same, I still can see my source code run correctly due to the same main flow.

2. Observance of step 5 and 6:

   Step 5: Run the program with ./sleepingBarbers 1 *chair* 200 1000

   Only if the chair goes up to at least 92 , all the customers can be served. i.e. "# customers who didn't receive a service = 0."

1barber_92chair_200customer_1000stime	1barber_91chair_200customer_1000stime
[zhengsq@csslab9 ver2]$ ./a.out 1 92 200 1000 ……. # customers who didn't receive a service = 0	[zhengsq@csslab9 ver2]$ ./a.out 1 91 200 1000 …… # customers who didn't receive a service = 4

   Step 6: Run the program with `./sleepingBarbers barbers 0 200 1000`
   Where *barbers* should be 1 ~ 3.

1barber_0chair_200customer_1000stime	2barber_0chair_200customer_1000stime	3barber_0chair_200customer_1000stime
[zhengsq@csslab9 ver2]$ ./a.out 1 0 200 1000 ……. # customers who didn't receive a service = 121	[zhengsq@csslab9 ver2]$ ./a.out 2 0 200 1000 ……. # customers who didn't receive a service = 50	[zhengsq@csslab9 ver2]$ ./a.out 3 0 200 1000 ……. # customers who didn't receive a service = 14

3. Clarification of shop.cpp implementation:

   The role that shop.cpp acts is a monitor, which is responsible to exchange information between barbers and customers.
   I utilize the algorithms below to help behave monitor's functionality:
   1). When a customer thread visitShop, after entering the critical section. If all chairs are full , he has to leave, and increment dropsoff number, and leave the critical section.
   if all barbers are busy, he takes a waiting chair--Push the customer in a waiting queue). Wait for a barber to wake him up. Then pop him out from the queue , then get his barber whose id is barberId. Then have barberId start his haircut. then leave the critical section.
   2). when a customer leaveShop, after entering the critical section. While barberId is cutting his hair, Wait. If all done, pay barber , then leave the critical section.

3). when a barber functions helloCustomer, after enteringr the critical section. If he has no customer and all the waiting chairs are empty, wait until a customer wakes him up. Then leave the critical section.

4). when a barber functions byeCustomer, after entering the critical section. Wakes up his customer. Wait for his customer to pay before he take a new one, then wakes up another customer who is waiting on a waiting chair. At last, Leave the critical section.

To support the algorithms above, I make use of several data structures and synchronization tools. As shown below:

```
bool* in_service; //bool type array which hints the barber is in service or not
bool* money_paid; //bool type array which hints the barber is paid or not

queue<int> waiting_chairs;   // includes the ids of all waiting threads of customers
deque<int> freeBarbers;    // includes the ids of all free barbers
vector<int> seats;    // service seats contain current serving customer id

pthread_mutex_t mutex; //lock for critical section
// conditions which can support different scenarios,  since there can be multiple barbers
//working on multiple customers at the same time, I use arrays of condition variables,
// e.g. cond_customer_served[id] is to ensure that a barber correctly signals the right
//customer when the haircut is finished.
pthread_cond_t  cond_customers_waiting;
pthread_cond_t* cond_customer_served;
pthread_cond_t* cond_barber_paid;
pthread_cond_t* cond_barber_sleeping;
```

4. Limitations and possible extension of the program:

The greatest limitation is that the program's performance depends on machine's configuration, in some sense, it cannot perfectly imitate this problem's scenario in the real life. In the real life, the more the barbers, the more customers can be served. However, in the simulation, when the number of barbers is greater than the number of threads which machine can hold concurrently, the number of customers who can be served doesn't have a great change, in some cases, ,maybe it would decrease.

Another limitation is the serve time is set to a constant, at this point, we set it 1000, and we got a bunch of results from this fixed serve time. What if serve time increases or decreases? Actually, the longer the serve time, the less the drop-off number is. So, the performance should also take into account this variable.

The possible extension is that we may create

5. Analysis based on observance of step 5 and 6:

[zhengsq@csslab9 ver2]$ lscpu

……

CPU(s): 2

……

Thread(s) per core: 1

Core(s) per socket: 1

Socket(s): 2

……

CPU MHz: 2199.986

…..

The information above is the configuration of the machine on which my program ran. The total number of threads which can action at the same time is 1, which means creating too many threads will not help to improve performance, instead, degrade it due to context switch overhead.

From results of experimenting on step 5, we can see, it requires 92 chairs to accomplish the task, which goes far beyond the recommended number 60. This result implies process waiting lists depend well on the number of cores and clock skews, in this case, the serve time, i.e. the clock skews, we define it as 1000, the number of core, as shown above, is just 1 core, and only support 1 thread per core. Another thing should be considered on is the CPU speed(2199.986 MHz), which determines the calculation time for each process.

Next, in step 6, there's no waiting chair at all, the number of barbers thread is the one that we need to test. Based on the practice of it, we can see, with the number of threads increases, the program performance improves as well. This result illustrates that multi thread programming is a good practice to increase programming efficiency.