CSC503
Jesse Leu
2020.05.10

Program 2: Sleeping Barbers

Documentation:

To make sure the order of output followed below, I had to use pthread_cond_signal and pthread_cond_wait to make sure the thread interact with each other correctly.

The workflow of serving a customer should follow the steps:

(barber sleeps because of no customers.)
1. customer moves to the service chair
2.customer wait for barber to be done with hair cut
3. barber starts a hair-cut service for customer
4.barber says he's done with a hair-cut service for customer
5.customer says good-bye to the barber
6.barber calls in another customer
(barber sleeps because of no customers.)

The workflow of customer waiting is:

1.customer takes a waiting chair.
2.barber calls in another customer
3.customer moves to the service chair

Condition variable:

- Barber:

1. cond_barber_sleeping
2. cond_barber_paid

- Customer:

1. cond_customers_waiting
2. cond_customers_sit

Shop.cpp pseudo code

```
int Shop_org::visitShop(int customerId)
{
   Enter critical session;
```

```
    Create a customer;
    if( no waiting seat)
    {

        nDropsOff++;
        print(leave);
        leave critical session;
        return UNSET;
    }

    if(no available barber)
    {
        Put the customer into waiting chair
        print(takes a waiting chair)
        Wait until customer get signal // cond_customers_waiting

    }
    If there is any sleeping barber
    {
         pair the first barber in the Sleeping barber queue with customer
    }
    print(moves to the service chair)
    Change customer serving to true

    Signal the barber  // cond_barber_sleeping
    Leave the critical session
    return barberId;
}
```

void Shop_org::leaveShop(int customerId, int barberId)

```
{
    Enter critical session
    print(wait for barber to be done with hair-cut )
    Wait while the barber is still working// cond_customers_sit


    print(say good bye)

    pay the barber
    signal the barber // cond_barbers_paid
    leave the critical session
}
```

void Shop_org::helloCustomer(int barberId)

```
{
        enter the critical session
```

```
    if no customer{
        print(sleeping);
        put barber into sleeping queue
wait for the customer pairing // cond_barber_sleeping
    }

    Wait for customer to sit down // cond_barber_sleeping

    Print(start hair cut)

    Leave critical session
}
```

```
void Shop_org::byeCustomer(int barberId)
{
    enter critical session
    print( barber say he is done)
    reset the customer's status;
signal the customer it is done // cond_customers_sit
    wait customer to pay
    reset the barber
    print(calls in another customer");
    if there is any waiting customer
    {
        pair the first customer in the waiting chair with the barber
        signal the customer // cond_customers_waiting
    }

     Leave critical session
}
```

Discussions

Step 5
For sleepingBarbers 1 *chair* 200 1000, I need around 95 chairs to serve all customers without anyone leaving.
For sleepingBarbers 1 *10* 200 1000, I have 92 customers leave without service.
For sleepingBarbers 1 *20* 200 1000, I have around 80 customers leave without service.
For sleepingBarbers 1 *30* 200 1000, I have around 70 customers leave without service.
For sleepingBarbers 1 *40* 200 1000, I have around 62 customers leave without service.
For sleepingBarbers 1 *50* 200 1000 I have around 50 customers leave without service.
For sleepingBarbers 1 *60* 200 1000, I have around 40 customers leave without service.

CSC503
Jesse Leu
2020.05.10
If I change the service time to 100->*sleepingBarbers 1 chair 200 100*, I need 2 barbers to serve all customers without anyone leaving.

Step 6

For *sleepingBarbers 1 0 200 1000*, I have around 120 customers leave without service.
For *sleepingBarbers 2 0 200 1000*, I have around 60 customers leave without service.

For *sleepingBarbers 3 0 200 1000*, I have around 20 customers leave without service.

For *sleepingBarbers barbers 0 200 1000*, I need around 4 barbers to serve all customer without anyone leaving.

If I change the service time to 100 -> *sleepingBarbers barbers 0 200 100*, I need 2 barbers to serve all customers without anyone leaving.

Limitation and possible extension:

possible extension:

- we can add more interaction between barbers and customers, and we can also add other parties into the program such as cell phone ringing, which will interrupt the barbers' working.
- Some activities can be moved into the barbers and customers class, and shop_org class's job can focus on pairing customers and barbers.

Limitation:

- The interacting of threads is hard to debug, which might limit the feature extension difficult.