Spring 2021

Modified Hilzer's Barbershop problem

CSE325: Operating System
Section 02

Presenting to Yasir Rayhan, Lecturer, EWU

Md Shamsur Rahman Talukdar
2019-1-60-264

Md. Nahid Hasan
2019-1-60-207

Problem

Design, implement and test a solution for the IPC problem specified below.

Our barbershop has three chairs, three barbers, and a waiting area that can accommodate four customers on a sofa and that has standing room for additional customers. Fire codes limit the total number of customers in the shop to 20.A customer will not enter the shop if it is filled to capacity with other customers. Once inside, the customer takes a seat on the sofa or stands if the sofa is filled. When a barber is free, one of the customers is served (whoever gets to the chair first) and, if there are any standing customers, one of them takes a seat on the sofa whoever gets the chance first. When a customer’s haircut is finished, any barber can accept payment, but because there is only one cash register, payment is accepted for one customer at a time. The barbers divide their time among cutting hair, accepting payment, and sleeping in their chair waiting for a customer.

The problem introduce starvation and deadlock issue

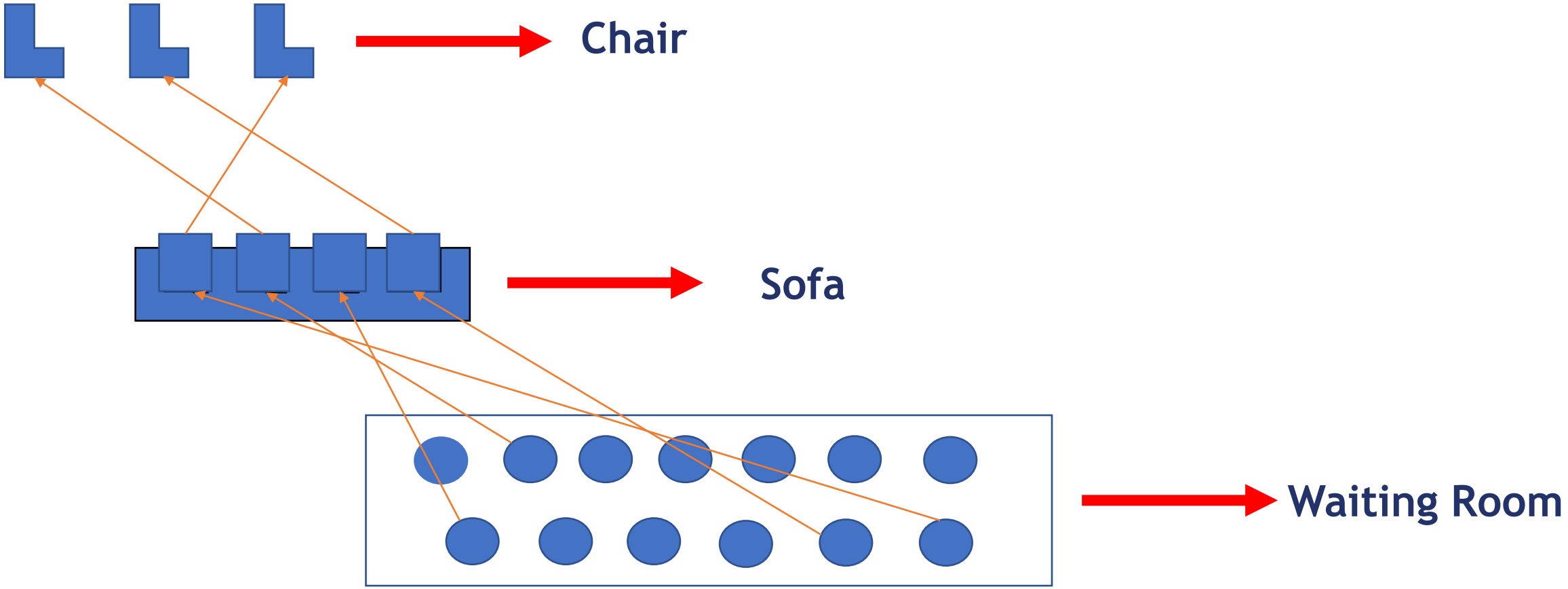
Starvation

- As there is no order for the customer in sofa and if a customer want to get hair cut first, it may cause starvation to other customers who already waits for long time.
- Also same for the customer who is waiting to get the seat in the sofa as there is no order.

Deadlock

- The customer may ends up waiting on the barber
- The barber may ends up waiting on the customer

Starvation



Solution

Semaphore Initialization

For Counting Availability of Sofa

For Managing Critical Section of Buffer Sofa

For Counting Availability of Chair

For Managing Critical Section of Buffer Chair

For Managing Payment

For Counting Total Number of Customer inside Shop

```
void init_semaphore()
{
    sem_init(&sofaEmpty, 0, 4);
    sem_init(&sofaFull, 0, 0);
    pthread_mutex_init(&lockSofa, 0);

    sem_init(&barberChairEmpty, 0, 3);
    sem_init(&barberChairFull, 0, 0);
    pthread_mutex_init(&lockBarberChair, 0);

    sem_init(&cashRegister, 0, 1);

    sem_init(&customerLimit, 0, 20);
}
```

Customer Enter the Shop and Takes a Seat in Sofa

Customer waiting for the seat

Completes seats in sofa

```
void *Customer(void *arg)
{
    int customerID = *(int *)arg;
    printf("I am customer %d\n", customerID);

    sem_wait(&sofaEmpty);
    pthread_mutex_lock(&lockSofa);
    sleep(1);

    sofaBuffer.push(customerID);
    printf("Customer %d sit in sofa\n", customerID);

    pthread_mutex_unlock(&lockSofa);
    sem_post(&sofaFull);

    delete arg;
}
```

Barber gets ready when a customer is available in sofa, one of the customers from the sofa is served

Pop a customer from Buffer sofa if any available

The customer has popped so increase empty

Add customer to barber chair

The customer has pushed to chair buffer so increase empty

```
void *Barber(void *arg)
{
    int barberID = *(int *)arg;
    printf("I am Barber %d\n", barberID);
    while (true)
    {
        sem_wait(&sofaFull);
        pthread_mutex_lock(&lockSofa);
        sleep(1);
        int cus = sofaBuffer.front();
        sofaBuffer.pop();
        printf("Customer %d left the sofa for haircut\n", cus);
        pthread_mutex_unlock(&lockSofa);
        sem_post(&sofaEmpty);

        sem_wait(&barberChairEmpty);
        pthread_mutex_lock(&lockBarberChair);
        sleep(1);
        barberBuffer.push(cus);
        printf("Customer %d sit in barber's chair \n", cus);
        pthread_mutex_unlock(&lockBarberChair);
        sem_post(&barberChairFull);
```

Any Barber Can Accept Payment,
After Payment a Customer can Leave

Barber is processing the payment

After payment, resetting the cash-register

Customer left

One slot available in waiting room for entering new customer

```
        sem_wait(&barberChairFull);
        pthread_mutex_lock(&lockBarberChair);
        int i = barberBuffer.front();
        sleep(2);
        sem_wait(&cashRegister);
        printf("Customer %d has done the payment \n", i);
        sem_post(&cashRegister);

        barberBuffer.pop();

        pthread_mutex_unlock(&lockBarberChair);
        sem_post(&barberChairEmpty);

        printf("Customer %d left the shop \n", i);
        sem_post(&customerLimit);
    }
    delete arg;
}
```

Any Barber Can Accept Payment,
After Payment a Customer can Leave

New customer has entered in shop

```
while (1)
{
    sem_wait(&customerLimit);
    pthread_t customerThread;
    int *id = (int *)malloc(sizeof(int));
    *id = i;
    if (pthread_create(&customerThread, NULL, &Customer, id))
    {

        perror("Could not create a thread\n");
        return 0;
    }
    i++;
}
```