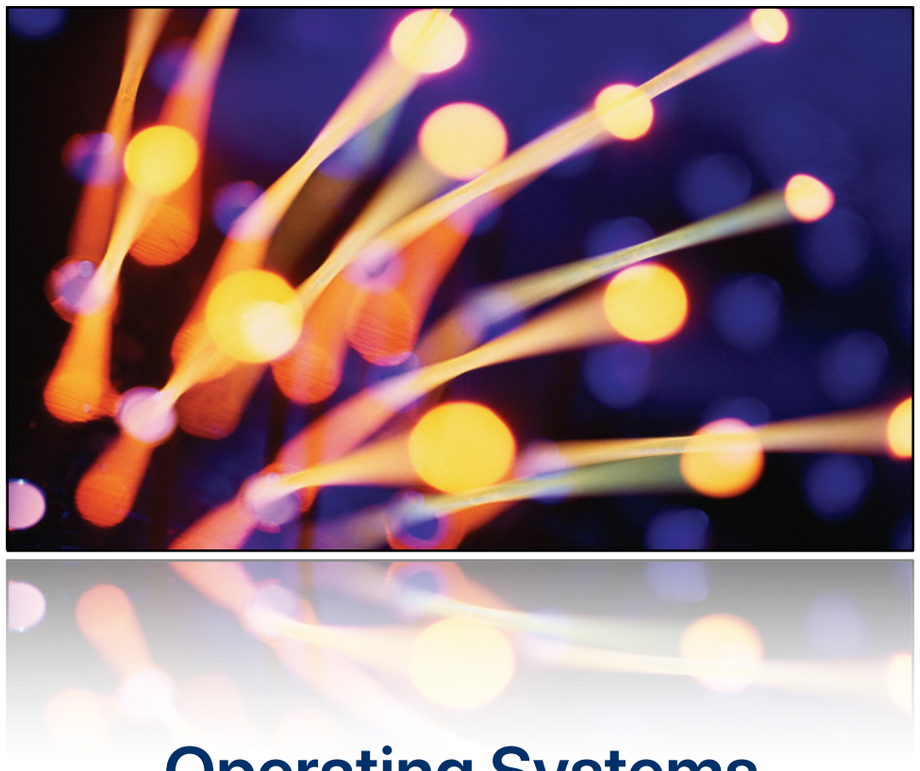
The Sleeping Barber



Operating Systems

Baochun Li
University of Toronto

The Sleeping Barber Problem

The Sleeping Barber Problem

The sleeping (sleepy?) barber —

While there are customers sitting in a waiting chair, move one customer to the barber chair, and start the haircut

When done, move to the next customer

If there are no customers, go to sleep

The customers —

If the barber is asleep, wake him up for a haircut

If someone is getting a haircut, wait for the barber to become free by sitting in a waiting chair

If all **N** waiting chairs are occupied, leave the barber shop

Basic ideas towards a solution

Model the barber and customers as threads

Model the number of waiting customers as a semaphore (0 or more)

```
semaphore customers = 0 // number of customers waiting
for service
```

Since there is no way to read the current value of this semaphore, we also need an integer variable, say **occupied_chairs**, to keep track of the number of waiting customers

To protect access to **occupied_chairs**, we need a mutual exclusion lock — `lock access_lock = UNLOCKED`

Model the state of the barber as a semaphore

```
semaphore barber = 0 // Is the barber ready to start?
```

Solving the Sleeping Barber Problem

```
semaphore barber = 0 // Is the barber ready to start?
semaphore customers = 0 // number of waiting customers
lock access_lock = UNLOCKED
int occupied_chairs = 0
barber()
   while true do
   customers.down() // wait for (or get) a customer
   acquire(access_lock)
   occupied_chairs = occupied_chairs – 1
   release(access_lock)
   barber.up() // the barber is now ready to start
   cut_hair()
```

Solving the Sleeping Barber Problem

```
customer()
   acquire(access_lock)
   if (occupied_chairs < N) then
     occupied_chairs = occupied_chairs + 1
     release(access_lock)
     customers.up()  // one more customer has taken a chair
     barber.down()  // waiting for barber to get ready
     get_haircut()
   else
     release(access_lock) // leave the barber shop
```

Subtleties in Lab 3, Task 2

In Task 2 of Lab 3, you are asked to print the state of the customers —

Enter, **S**it in a waiting chair, **B**egin haircut, **F**inish haircut, **L**eave

For a successful haircut, from the desired output, the customer and the barber go through 7 states sequentially —

Enter, **S**it in a waiting chair, **Start**, **B**egin haircut, **F**inish haircut, **End**, **L**eave

Correctly producing this sequence requires —

The barber prints **Start** before the customer prints **B**

The customer prints **F** before the barber prints **End**

The barber prints **End** before the customer prints **L**

But How? — use semaphores for ordering

```
semaphore barber_done = 0 // Is the barber done?
barber()
  while true do
    customers.down() // wait for (or get) a customer
    acquire(access_lock)
    occupied_chairs = occupied_chairs - 1
    release(access_lock)
    barber.up() // the barber is now ready to start
    cut_hair()
    barber_done.up() // the barber is now done with the haircut
customer()
   acquire(access_lock)
   if (occupied_chairs < N) then
      occupied_chairs = occupied_chairs + 1
      release(access_lock)
      customers.up()  // one more customer has taken a chair
      barber.down()  // waiting for barber to get ready
      get_haircut()
      barber_done.down() // waiting for barber to be done
   else
      release(access_lock) // leave the barber shop
```

What we've covered so far

Three Easy Pieces: Chapter 31.3 (Semaphores For Ordering)

Lab 3: SleepingBarberProblem.pdf