Introduction to Operating Systems
CS 1550

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Some slides are from **Silberschatz, Galvin and Gagne ©2013)**

Announcements

- Upcoming deadlines
  - Homework 3 is due next Monday 2/7
  - Lab 1 is due this Friday 2/4
  - Project 1 due on 2/18

Previous lecture …

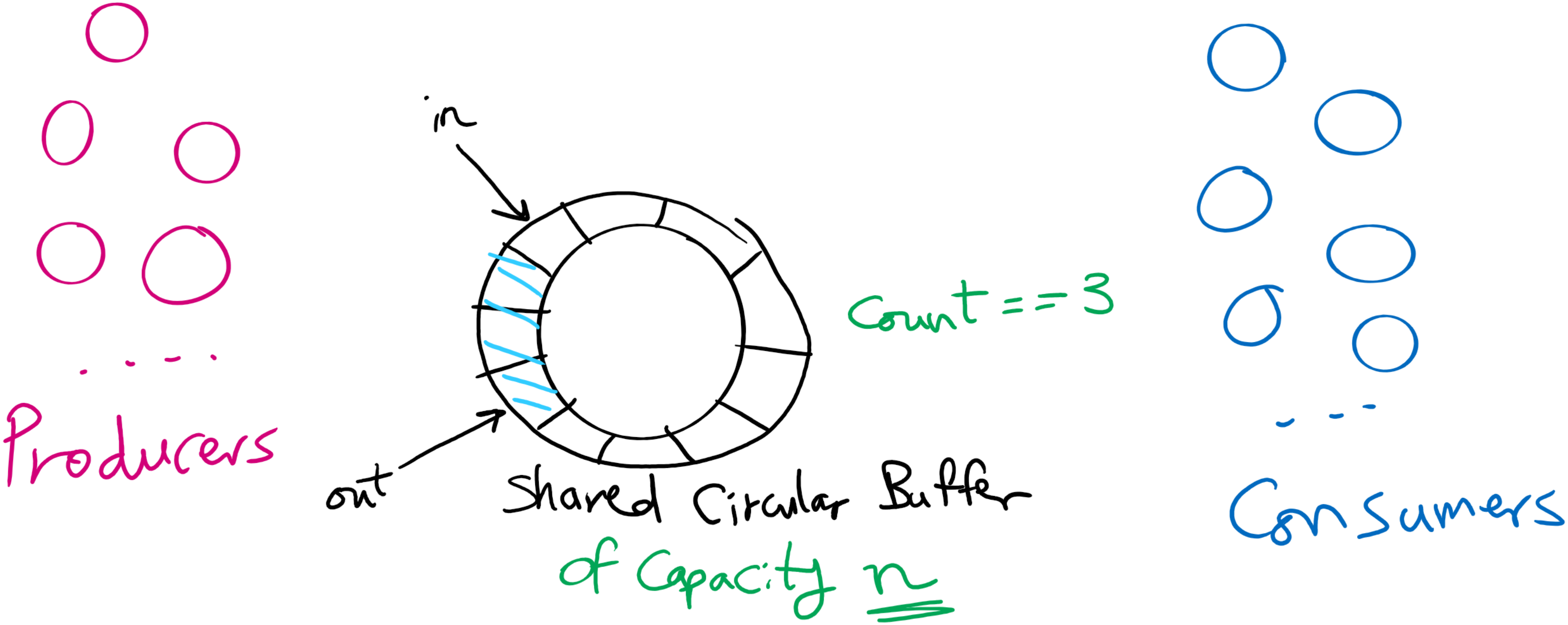
- Bounded buffer problem
    - semaphore-based solution

Muddiest Points

Problem of the Day

It is easy to make mistakes when using semaphores

Produces Consumers Problem



Solving Producers Consumers using Semaphores

```
Semaphore  empty(n), full(0)
  Mutex    Sem(1);
```

Producer

```
down(empty)
down(sem)
 buffer[in]=new item
  in +=1 % n
  Count++
 up(sem)
 up(full)
```

Consumer

```
down(full)
down(Sem)
item = buffer[out]
out +=1 % n
 Count --
 up(sem)
 up(empty)
```

Is this sequence feasible?

```
n == 3
for (i=0; i<3; i++){
    Pi arrives
    Pi enters
    Pi leaves
}
P3 arrives
C0 arrives
C0 enters
C0 leaves
P3 enters
P3 leaves
```

Some thoughts

- If we have one producer and one consumer
  - do we need count?
  - do we need the mutex?
- For multiple producers and consumers
  - what benefit do we get if we have one mutex for producers and one for consumers?

Let’s make a “small” change

```
Semaphore empty(n), full(0)
Mutex     Sem(1);
```

Producer

```
down(empty)
down(sem)
  buffer[in] = new item
  in += 1 % n
  Count++
up(sem)
up(full)
```

Consumer

```
down(full)
down(Sem)
item = buffer[out]
out += 1 % n
Count --
up(Sem)
up(empty)
```

Let’s make a “small” change

Semaphore empty(n), full(0);
Mutex sem(1);

Producer

```
down(sem)
down(empty)
buffer[in] = new item
in = (in + 1) % n
count++
up(empty)
up(sem)
```

Consumer

```
down(full)
down(sem)
Item = buffer[out]
out = (out + 1) % n
count--
up(sem)
up(full)
```

Is this sequence feasible?

```
n == 3
for (i=0; i<3; i++){
   Pi arrives
   Pi enters
   Pi leaves
}
P3 arrives
C0 arrives
C0 enters
C0 leaves
P3 enters
P3 leaves
```

Solution

- Condition Variable
  - Yet another construct (Add to Spinlock and Semaphore)
  - Has 3 operations
    - These 3 operations have to be called while holding a mutex lock
    - wait()
      - unlock mutex
      - block process
      - when awake, relock mutex
      - when successful, return
    - signal()
      - wakeup one waiting process in the condition variable’s queue if any
    - broadcast()
      - wakeup all waiting processes in the condition variable’s queue if any
  - Not foreign to us at all
    - Every object variable in Java is a Condition Variable

Solving Bounded Buffer Using Condition Variables

```
Mutex sem;
ConditionVariable cv;
```

Producer

```
down (sem)
while (count==n)
        cv.wait ()
buffer[in]=new item
in =(in+1)%n
count ++
cv.broadcast()
up(sem )
```

Consumer

```
down(sem)
while(count==0)
    cv.wait()
item=buffer[out]
out=(out+1)%n
count --
cv.broadcast()
up(Sem)
```