University of Pittsburgh

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 **this Friday**
  - Lab 1 is due on Tuesday 2/7 at 11:59 pm
  - Project 1 is due on Friday 2/17 at 11:59 pm
    - Discussed in this week's recitations

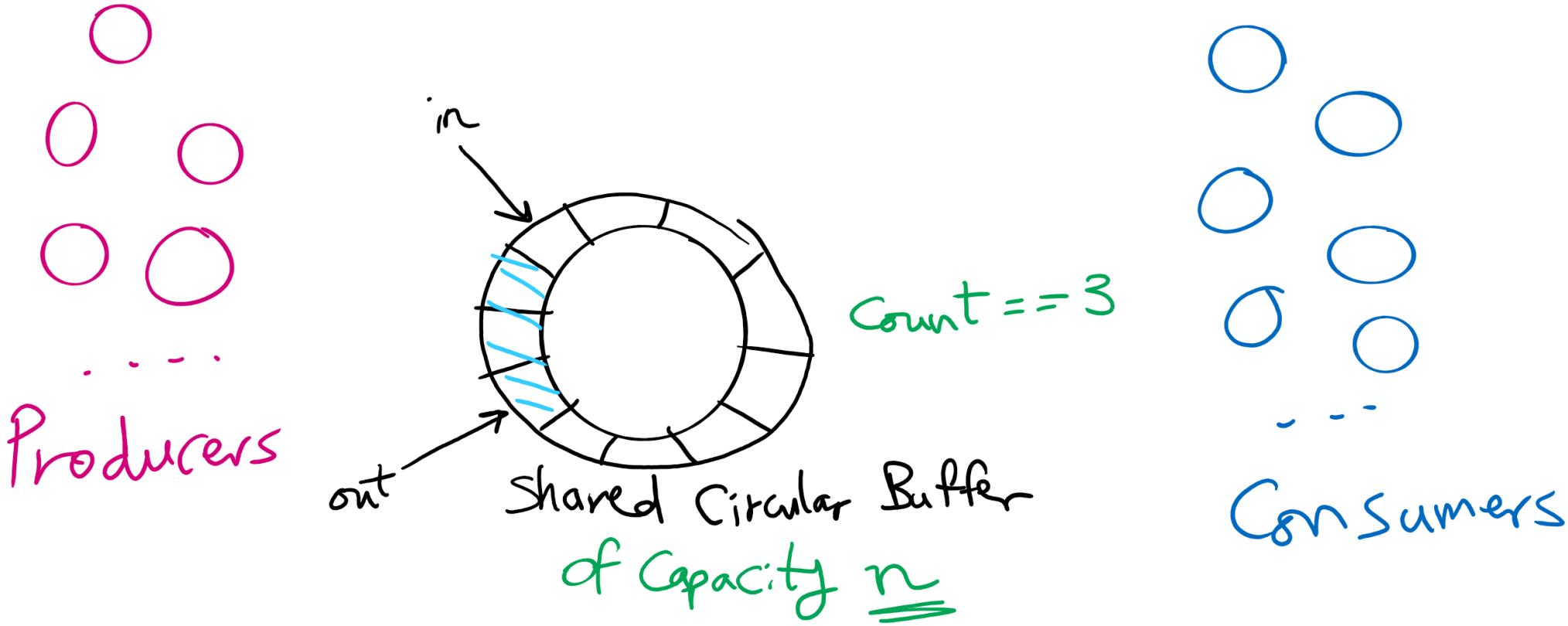
Previous lecture …

- Bounded buffer problem
  - semaphore-based solution

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

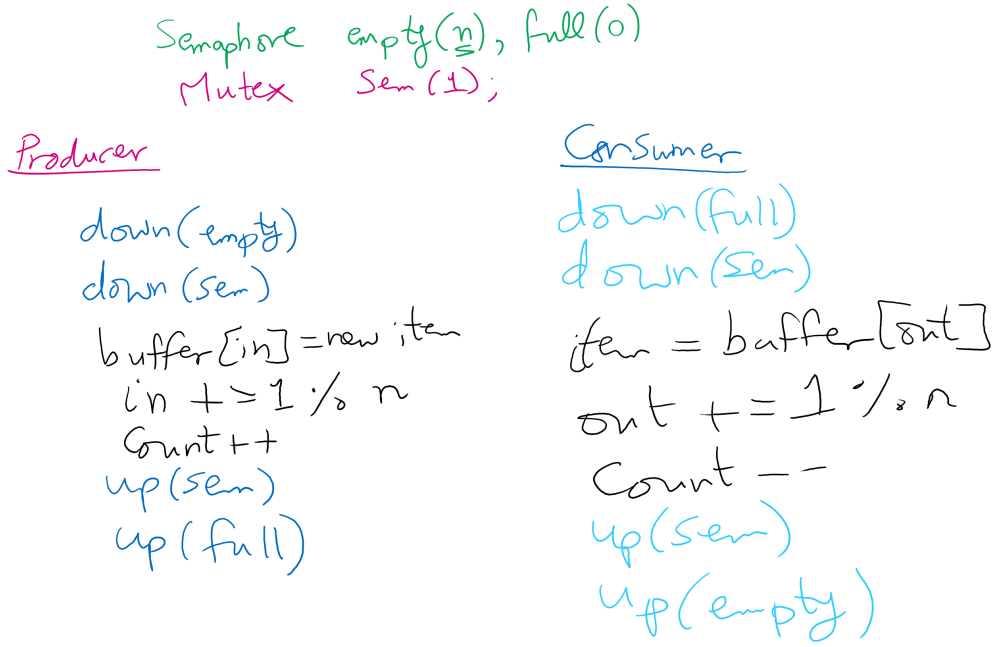
```
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
Count++
up(sem)
up(full)
```

Consumer

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

Some thoughts

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



Condition Variables

- Yet another construct for process/thread synchronization
  - (Add to Spinlock and Semaphore)
- Has 3 operations
  - wait()
  - signal()
  - broadcast()
- Not foreign to us at all
  - Every object variable in Java is a Condition Variable

Condition Variables Operations

- These 3 operations must 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

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)
```

Readers & Writers Problem

- Many processes that may read and/or write
- Only one writer allowed at any time
- Many readers allowed, but not while a process is writing
- Real-world Applications
  - Database queries
  - We have this problem in Project 1

Semaphore-based Solution

Shared variables

```
int nreaders;
Semaphore mutex(1), writing(1);
```

Reader process

```
…
mutex.down();
nreaders += 1;
if (nreaders == 1) // wait if
  writing.down();  // 1st reader
mutex.up();
// Read some stuff
mutex.down();
nreaders -= 1;
if (nreaders == 0)      // signal if
  writing.up();         // last reader
mutex.up();
…
```

Writer process

```
…
writing.down();
// Write some stuff
writing.up();
…
```

Solution Tracing

- enterRead

Reader process

```
…
mutex.down();
nreaders += 1;
if (nreaders == 1) // wait if
  writing.down();  // 1st reader
mutex.up();
// Read some stuff
mutex.down();
nreaders -= 1;
if (nreaders == 0)          // signal if
  writing.up();   // last reader
mutex.up();
…
```

Solution Tracing

- read

Reader process

```
…
mutex.down();
nreaders += 1;
if (nreaders == 1) // wait if
  writing.down();  // 1st reader
mutex.up();
// Read some stuff
mutex.down();
nreaders -= 1;
if (nreaders == 0)          // signal if
  writing.up();   // last reader
mutex.up();
…
```

Solution Tracing

- doneRead

Reader process

```
…
mutex.down();
nreaders += 1;
if (nreaders == 1) // wait if
  writing.down();  // 1st reader
mutex.up();
// Read some stuff
mutex.down();
nreaders -= 1;
if (nreaders == 0)          // signal if
  writing.up();   // last reader
mutex.up();
…
```

Writer Events

- enterWrite

```
Writer process
…
writing.down();
// Write some stuff
writing.up();
…
```

Writer Events

- write

```
Writer process
…
writing.down();
// Write some stuff
writing.up();
…
```

Writer Events

- doneWrite

```
Writer process
…
writing.down();
// Write some stuff
writing.up();
…
```

Sequence 1

- W0 enterWrite
- W0 write
- R0 enterRead
- R1 enterRead
- R2 enterRead
- W0 doneWrite
- R2 read
- W1 enterWrite
- R2 doneRead
- W1 write

Reader process

```
…
mutex.down();
nreaders += 1;
if (nreaders == 1) // wait if
  writing.down();  // 1st reader
mutex.up();
// Read some stuff
mutex.down();
nreaders -= 1;
if (nreaders == 0)      // signal if
  writing.up();         // last reader
mutex.up();
…
```

Writer process

```
…
writing.down();
// Write some stuff
writing.up();
…
```

Sequence 2

- R0 enterRead
- R0 read
- R1 enterRead
- R1 read
- W0 enterWrite
- R2 enterRead
- R2 read
- R2 doneRead
- R1 doneRead
- R0 doneRead
- W0 write
- W0 doneWrite

Reader process

```
…
mutex.down();
nreaders += 1;
if (nreaders == 1) // wait if
  writing.down();  // 1st reader
mutex.up();
// Read some stuff
mutex.down();
nreaders -= 1;
if (nreaders == 0)       // signal if
  writing.up();          // last reader
mutex.up();
…
```

Writer process

```
…
writing.down();
// Write some stuff
writing.up();
…
```

Solution using Mutex and Condition Variables

- https://cs1550-2214.github.io/cs1550-code-handouts/ProcessSynchronization/Slides/