Material Application: Producer Consumer Problem

44-550: Operating Systems

What is it?

- Involves two or more processes.
- Has at least one *producer* that creates data items and puts them in a buffer (shared memory, etc, etc)
- Has at least one *consumer* that removes data items from the buffer and does things with them

Why now?

- Encompasses multiple topics from this course
  - Process synchronization
  - Deadlock
  - Inter-process communication

Problem Definition

- All processes need access to the shared buffer
- Producers cannot put items in the buffer when it is full
- Consumer can't remove items from the buffer when it is empty
- Insertion and Removal of data items to/from the buffer are mutually exclusive operations

One solution

```
def producer():
    while true:
        item = createItem()
        if buffer is full:
            sleep until awoken
        insert item into buffer
        increment item count
        if itemCount == 1:
            wake(consumer)
```

```
def consumer():
    while true:
        if buffer is empty:
            sleep until awoken
        remove item from buffer
        decrement item count
        if buffer was full:
            wake(producer)
```

Why doesn't this work?

- Race condition that can lead to deadlock
    - Consumer checks if the buffer is empty, and it is
    - Producer creates an item, puts it into buffer, and increases itemCount
    - Producer tries to wake consumer
    - Consumer falls alseep
- We need semaphores and mutexes to make sure that all *critical sections* are protected
    - sem_t fullCount = 0
    - sem_t emptyCount = BUFFER_SIZE

Solution: one producer, one consumer

```
def producer():
    while true:
        item = createItem()
        sem_wait(&emptyCount) # decrement
        insert item into buffer
        sem_post(&fullCount) # increment
```

```
def consumer():
    while true:
        sem_wait(&fullCount) # decrement
        remove item from buffer
        sem_post(&emptyCount) # increment
```

Fails when multiple producers: two producers decrement emptyCount, and both producers decide to write to the same place in the buffer. Must add a mutex to protect insertion and removal

Solution: N producers, N consumers

```
def producer():
    while true:
        item = createItem()
        sem_wait(&emptyCount)
        lock(mutex)
        insert item into buffer
        unlock(mutex)
        sem_post(&fullCount)
```

```
def consumer():
    while true:
        sem_wait(&fullCount)
        lock(mutex)
        remove item from buffer
        unlock(mutex)
        sem_post(&emptyCount)
```

Beyond Synchronization

- Producer Consumer style problems are common
- Interestingly, the problem becomes easier for distributed style systems when you have a central manager (or message queue)
    - Since some shared memory is required, building additional functionality into the central queue can help make using P/C easier
    - Technologies such as RabbitMQ, ZeroMQ, and others have this pattern built in for programmers to use.