Synchronization

Dr. Eraldo Ribeiro

Department of Computer Sciences
Florida Institute of Technology
Melbourne, FL 32901, USA

CSE 4001 - Operating Systems Concepts
The producer-consumer problem

Producer

```java
event = waitForEvent()
buffer.add(event)
```

Consumer

```java
event = buffer.get()
ev.text.process()
```

Access to the buffer has to be exclusive, but `waitForEvent()` and `event.process()` can run concurrently.
Producer-consumer hint

**Initialization:**

```python
mutex = Semaphore(1)
items = Semaphore(0)
local event
```

- What is the function of `mutex`?
- What does `items` indicate when its value is positive?
- What does `items` indicate when its value is negative?
Producer-consumer hint

Initialization:

```
mutex = Semaphore(1)
items = Semaphore(0)
local event
```

- What is the function of `mutex`?
- What does `items` indicate when its value is positive?
- What does `items` indicate when its value is negative?
Producer-consumer hint

Initialization:

\[\text{mutex} = \text{Semaphore}(1)\]
\[\text{items} = \text{Semaphore}(0)\]
\[\text{local event}\]

- What is the function of \text{mutex}?
- What does \text{items} indicate when its value is positive?
- What does \text{items} indicate when its value is negative?
Producer-consumer solution

Producer

```java
event = waitForEvent()
mutex.wait()
    buffer.add(event)
items.signal()
mutex.signal()
```

Consumer

```java
items.wait()
mutex.wait()
    event = buffer.get()
mutex.signal()
event.process()
```
Performance issue: Imagine that there is at least one consumer in queue when a producer signals items. If the scheduler allows the consumer to run, what happens next?

**Producer**

```java
event = waitForEvent()
muxex.wait()
    buffer.add(event)
    items.signal()
mutex.signal()
```

**Consumer**

```java
items.wait()
muxex.wait()
    event = buffer.get()
mutex.signal()
event.process()
```
**Improve producer**: don’t bother unblocking a consumer until we know it can proceed (except in the rare case that another producer beats it to the mutex).

**Producer**

```java
event = waitForEvent()
mutex.wait()
    buffer.add(event)
items.signal()
mutex.signal()
```

**Consumer**

```java
items.wait()
mutex.wait()
    event = buffer.get()
mutex.signal()
event.process()
```

How should we change the Producer?
**Producer-consumer solution**

**Improve producer**: don’t bother unblocking a consumer until we know it can proceed (except in the rare case that another producer beats it to the mutex).

```
Producer

event = waitForEvent()
mutex.wait()
    buffer.add(event)
items.signal()
mutex.signal()

Consumer

items.wait()
mutex.wait()
    event = buffer.get()
mutex.signal()
event.process()
```

How should we change the Producer?
Producer-consumer solution

can you see any other issue with the solution below?

Producer

```java
event = waitForEvent()
mutex.wait()
    buffer.add(event)
mutex.signal()
items.signal()
```

Consumer

```java
items.wait()
mutex.wait()
    event = buffer.get()
mutex.signal()
event.process()
```

A possible fix by checking the buffer inside the mutex in Consumer.
Producer-consumer solution

can you see any other issue with the solution below?

Producer

event = waitForEvent()
mutex.wait()
    buffer.add(event)
mutex.signal()
items.signal()

Consumer

items.wait()
mutex.wait()
    event = buffer.get()
mutex.signal()
    event.process()

A possible fix by checking the buffer inside the mutex in Consumer.
Producer-consumer solution

A possible fix by checking the buffer inside the mutex in Consumer.

Producer

event = waitForEvent()
mutex.wait()
buffer.add(event)
mutex.signal()
items.signal()

Consumer

mutex.wait()
items.wait()
event = buffer.get()
mutex.signal()
event.process()

This is a bad idea. Why?

Imagine that the buffer is empty. A consumer arrives,...

Any time you wait for a semaphore while holding a mutex, there is a danger of deadlock. This is a common mistake. Check for this problem when using semaphores for synchronization.
Producer-consumer solution

A possible fix by checking the buffer inside the mutex in Consumer.

Producer

```java
event = waitForEvent()
mutex.wait()
buffer.add(event)
mutex.signal()
items.signal()
```

Consumer

```java
mutex.wait()
items.wait()
event = buffer.get()
mutex.signal()
event.process()
```

This is a bad idea. Why?

Imagine that the buffer is empty. A consumer arrives,...

Any time you wait for a semaphore while holding a mutex, there is a danger of deadlock. This is a common mistake. Check for this problem when using semaphores for synchronization.
Producer-consumer solution

A possible fix by checking the buffer inside the mutex in Consumer.

**Producer**

```java
event = waitForEvent()
mutex.wait()
buffer.add(event)
mutex.signal()
items.signal()
```

**Consumer**

```java
mutex.wait()
    items.wait()
event = buffer.get()
mutex.signal()
event.process()
```

This is a bad idea. Why?

Imagine that the buffer is empty. A consumer arrives,...

Any time you wait for a semaphore while holding a mutex, there is a danger of deadlock. This is a common mistake. Check for this problem when using semaphores for synchronization.
Producer-consumer limited buffer

Add a second semaphore to keep track of the number of available spaces in the buffer.

```python
mutex = Semaphore(1)
items = Semaphore(0)
spaces = Semaphore(buffer.size())
```

When a consumer removes an item it should signal spaces. When a producer arrives it should decrement spaces, at which point it might block until the next consumer signals.
Producer-consumer limited buffer

Add a second semaphore to keep track of the number of available spaces in the buffer.

```python
mutex = Semaphore(1)
items = Semaphore(0)
spaces = Semaphore(buffer.size())
```

When a consumer removes an item it should signal spaces. When a producer arrives it should decrement spaces, at which point it might block until the next consumer signals.
Producer-consumer limited buffer

```python
mutex = Semaphore(1)
items = Semaphore(0)
spaces = Semaphore(buffer.size())
```

### Producer

```python
event = waitForEvent()
mutex.wait()
buffer.add(event)
mutex.signal()
items.signal()
```

### Consumer

```python
items.wait()
mutex.wait()
    event = buffer.get()
mutex.signal()
    event.process()
```

What should we change?...
End of Today's Lecture