

# SOME CLASSIC CONCURRENCY PROBLEMS

## THE PRODUCER CONSUMER PROBLEM

Joseph Kehoe<sup>1</sup>

<sup>1</sup>Department of Computing and Networking  
SETU, Carlow Campus

Concurrency, 2023

# PRODUCERS AND CONSUMERS

- A common pattern is for the division of labor amongst threads
  - E.g. Some threads consume while others produce
- Producers create items of some kind and add them to a data structure (buffer)
- Consumers remove the items and process them
  - E.g. Event driven programs
- Consumers sometimes known as event handlers

# THE PRODUCER CONSUMER PATTERN

- While an item is being add to or removed from buffer the buffer is in an inconsistent state
- Therefore we must guarantee exclusive access to the buffer
- If a consumer thread arrives when the buffer is empty it must wait until a producer adds a new item

- Producer

```
Event= createEvent()  
Buffer.add(event)
```

- Consumer

```
Event = Buffer.get()  
Event.process()
```

Leader

```
Mutex=semaphore(1) //control access to buffer
```

```
Items=semaphore(0) //blocks when buffer is empty
```

# HINT TWO

Local variable event (for adding to or taking from buffer)

- Event is local to thread
- Each thread has their own version of event!
- Each thread may have their own run-time stack so all local variables are thread specific
  - If threads are objects then we can add attributes to the objects
  - If threads have unique ID's then we can use ID as an index into an array or hash table

## Producer

```
Event= createEvent()  
Mutex.wait()  
Buffer.add(event)  
Items.signal()  
Mutex.signal()
```

## Consumer

```
Items.wait()  
Mutex.wait()  
Event = buffer.get()  
Mutex.signal()  
Event.process()
```

# IMPROVED SOLUTION

Signaling inside the mutex can be inefficient (Why?)

```
Event= createEvent()  
Mutex.wait()  
Buffer.add(event)  
Mutex.signal()  
Items.signal()
```

# INCORRECT SOLUTION

- Items can be inaccurate given certain interleavings
- We can try correct this...

But what is wrong with this Consumer (below)?

```
Mutex.wait()  
Items.wait()  
Event = buffer.get()  
Mutex.signal()  
Event.process()
```



- If buffer is finite it can fill up
- In that case producers should wait until the buffer has freed up some space before adding to buffer
- We cannot check the value of items as we are not allowed to do this!
- Hint
  - Add another semaphore initialised to the buffer size!
  - `Spaces=semaphore(buffer.size)`

# FINITE BUFFER SOLUTION

## Producer

```
Event= createEvent()  
Spaces.wait()  
Mutex.wait()  
Buffer.add(event)  
Mutex.signal()  
Items.signal()
```

## Consumer

```
Items.wait()  
Mutex.wait()  
Event = buffer.get()  
Mutex.signal()  
Spaces.signal()  
Event.process()
```