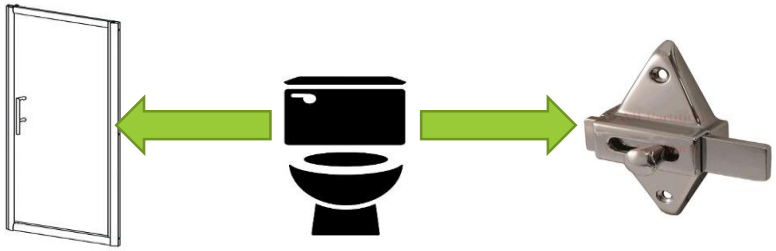
ENGI 7894/9869

Semaphores

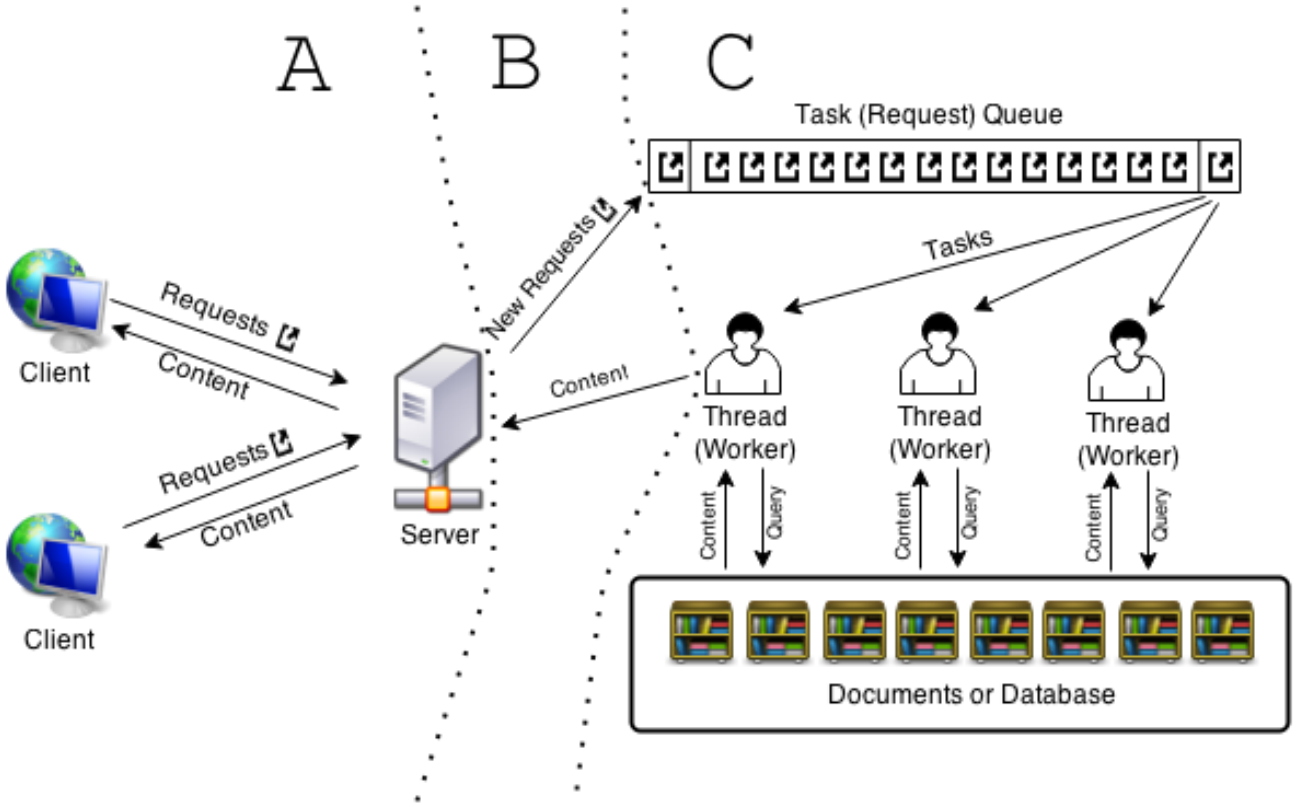
Presented by
Md Monjur Ul Hasan

Question or Comments From Previous Class?

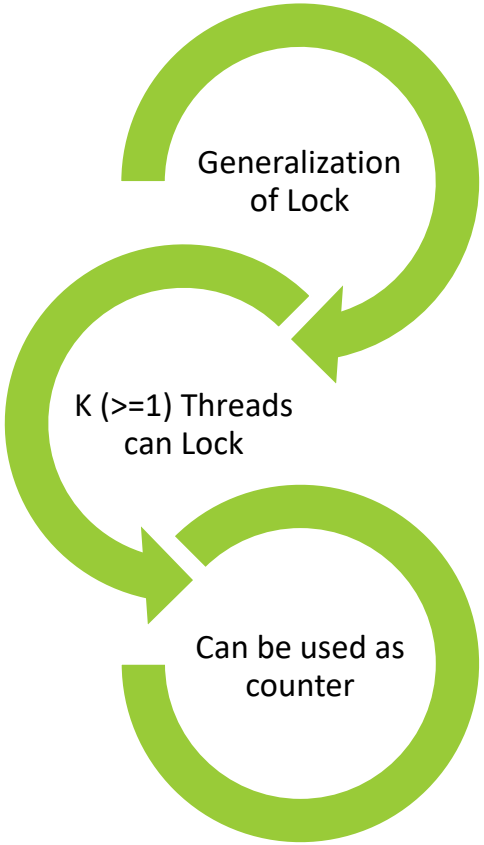
Analogy



Client Server Applications



Semaphores: Definition



- Semaphores with capacity 1 is called *binary semaphore*
- Another name of generic semaphore is counting semaphores

Semaphores: Operations

P Operation

- Similar to Lock()
- Passing (*passering*) operation
- Decrement *count* when ***greater*** than *0*
- *Otherwise blocks*

V Operation

- Similar to Unlock()
- Release (*vrijgave*) operation
- Increment count

Semaphores: Advantages

Lower Errors

- Impose Deliberate Constraints

Clean Solution

- Organized
- Easy to demonstrate the correctness

Efficient

- Implement efficiently on systems

Semaphores: Implementation Example

```
public class Semaphore {
        int state;
        iLock lock;
        Condition condition;
        public Semaphore(int c) {
                state = c;
                lock = new Lock();
        }
        public void P() {
                lock.lock();
                try {
                        while (state == 0) {} ;
                        state--;
                }
                finally {
                       lock.unlock();
                }
        }
       public void V() {
                lock.lock();
                try {
                        state++;
                }
                finally {
                          lock.unlock();
                }
        }
}
```

Types of Semaphore

Binary Semaphore

- Capacity 1
- Symone of Locks
- `mutex = Semaphore(1);`

Counting Semaphore

- Initialize with capacity 0
- `items = Semaphore(0);`

Split Binary Semaphore

- A set of binary semaphore
- Only one of the semaphore can set

Question or Comments?

Producer Consumer Problem

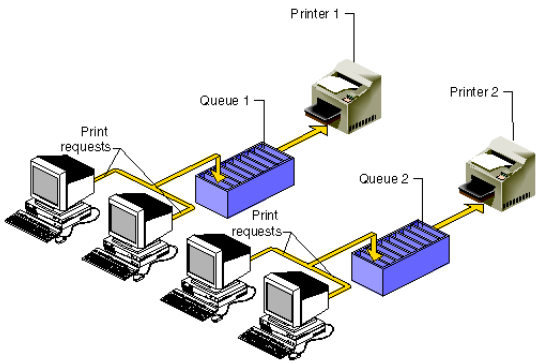
- Two Processes: Producer(s) and Consumer(s)
- Producer:
  - Generate a piece of data
  - Put it into buffer
  - Start Over
- Consumer:
  - Consuming the data, one piece at a time
  - Remove it from the buffer
  - Start Over
- Number of Producer can be 1 or more
- Number of Consumer can be 1 or more

One Producer – One Consumer

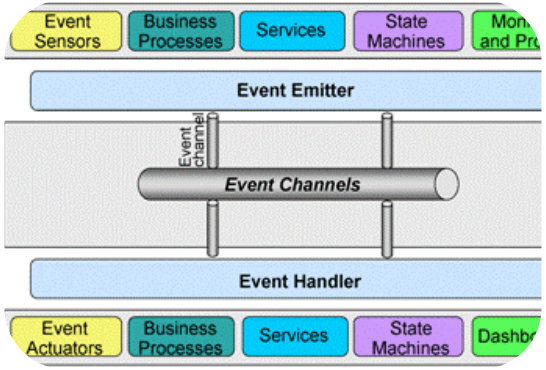
Eating faster then producing
Eater needs to wait

Producing faster
Producer needs to wait

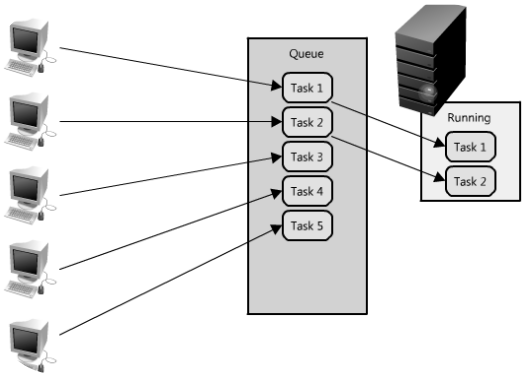
Producer-Consumer Problem: Examples



Printing Queue



Event Handling



Web Server

Producer-Consumer Problem

Buffer

- Bounded/Unbounded
- Queue, Circular queue, Double Ended queue

Producer Process

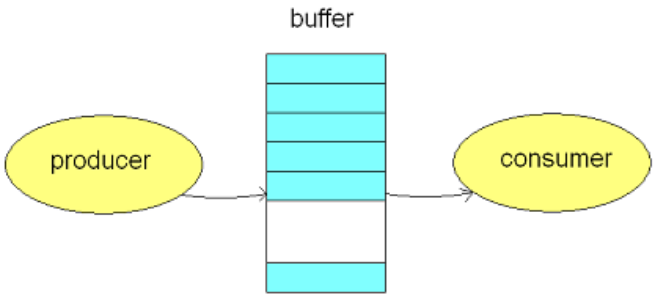
- Write task in the queue

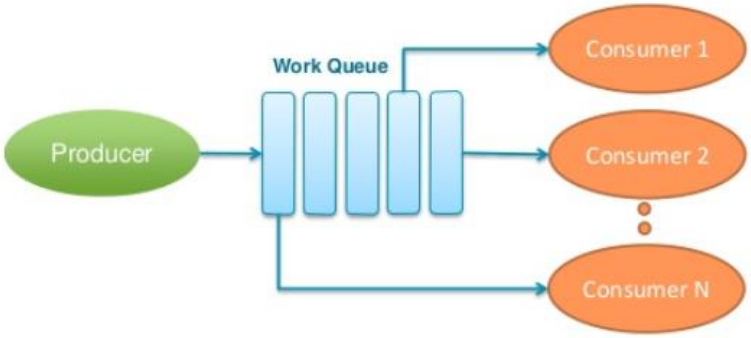
Consumer Process

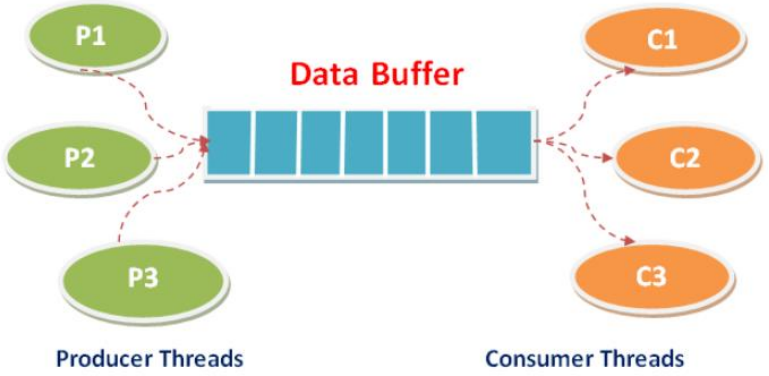
- Read task from the queue
- Remove the task from the queue

Produce when buffer is full

Consume when buffer is empty







Unbounded Buffer

- Consumer need to check
  - If buffer is NOT currently used by producer
  - If buffer has item to consume
- Producer are not limited by buffer size
  - Producer needs to wait if buffer is currently accessed by consumer
  - Producer DO NOT need to check (and wait on) for a buffer full situation

Producer – Consumer Solution: Unbounded Buffer

```
mutex = Semaphore(1);
items = Semaphore(0);
```

Producer

```
event = waitForEvent();
mutex.P();
        buffer.add(event);
        items.V();
mutex.V()
```

CS Protection

CS

Consumer

```
items.P();
mutex.P();
    event = buffer.get();
mutex.V();
event.process();
```

CS Protection

A Better Approach

```
event = waitForEvent();
mutex.P();
        buffer.add(event);
mutex.V();
items.V();
```

Ensure:

- Protect the CS
- Consumer not reading the empty buffer

Producer – Consumer Solution: Bounded Buffer

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

Producer

```
spaces.P();
event = waitForEvent();
mutex.P();
        buffer.add(event);
mutex.V()
items.V();
```

Consumer

```
items.P();
mutex.P();
        event = buffer.get();
mutex.V();
spaces.V();
event.process();
```

Ensure:

- Protect the CS
- Consumer not reading the empty buffer
- Producer wait when buffer full

Question or Comments?

Producer – Consumer Solution: Bounded Buffer

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

Producer

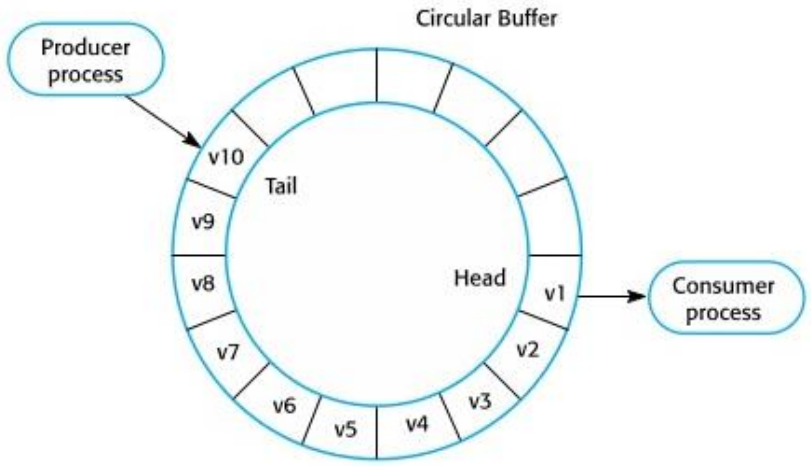
```
spaces.P();
event = waitForEvent();
mutex.P();
        buffer.add(event);
mutex.V()
items.V();
```

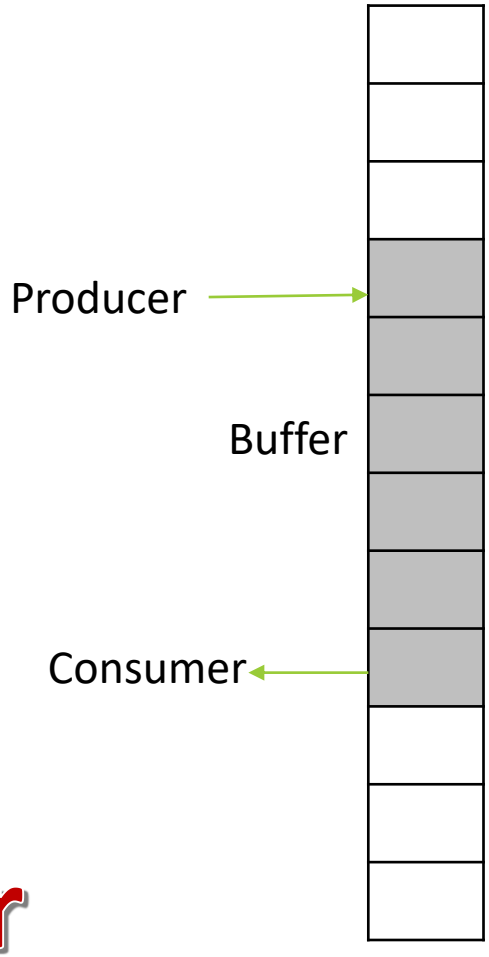
Consumer

```
items.P();
mutex.P();
        event = buffer.get();
mutex.V();
spaces.V();
event.process();
```

Is it Coarse Grained or Fine Grained Solution?

Producer – Consumer Solution: Bounded Buffer





No need to wait for all write or read operations on Buffer

Fine Grain Solution of Producer – Consumer Problem

```
int front=0;
int rear=0;
int count=0;
mutex = Semaphore(1);
items = Semaphore(0);
space = Semaphore(1);
```

Ensure:

- Protect the CS
- Consumer not reading the empty buffer
- Producer wait when buffer is full

Producer

```
event.WaitForEvent();
space.P();
buffer[rear] = data;
rear = (rear+1) %n;
mutex.P();
count++;
if (count == 1)
    items.V();
if (count < n)
    space.V();
mutex.V();
```

Consumer

```
items.P();
data=buffer[front];
front = (front+1) %n;
mutex.P();
count--;
if (count == n-1)
    space.V();
if (count > 0)
    items.V();
mutex.V();
event.ProcessEvent();
```

Why Within Condition?

Topics

- Semaphore
  - Definition
  - Operations
  - Advantages
  - Implementation Example
  - Types of Semaphores
- Producer-Consumer Problem

Thank you for your attention

Any Questions?

Reference:
Little Book of Semaphores, by Allen B. Downey
http://greenteapress.com/wp/semaphores/