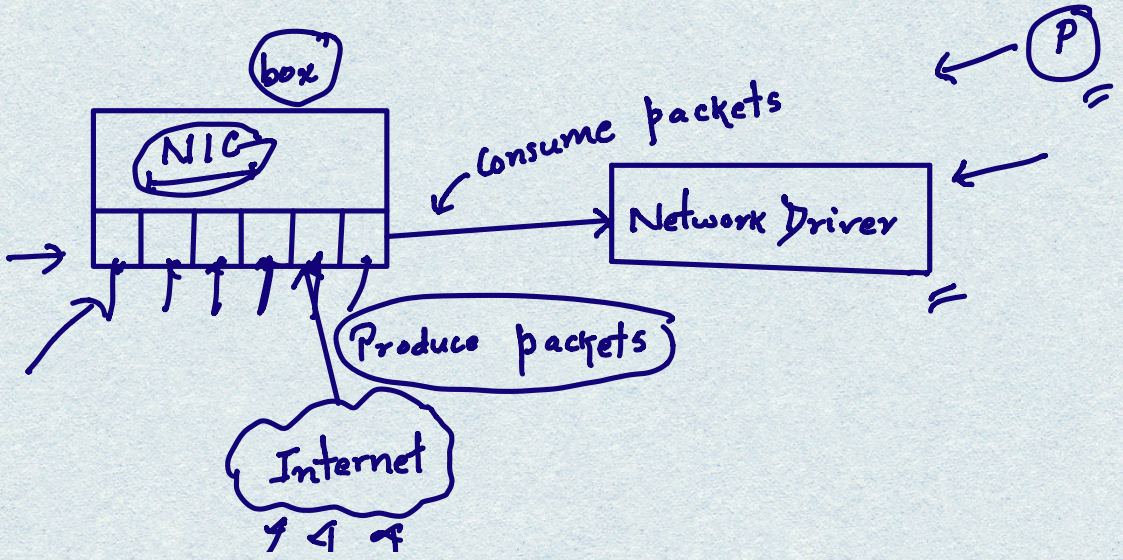
Producer - Consumer Problem -

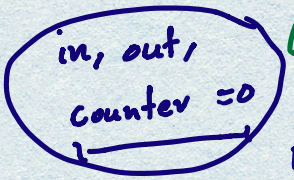
- Producer and Consumer share a common, fixed size buffer (Queue).
- Producer job is to generate data, put it in the buffer and start again.
- Consumer is consuming data, one piece at a time.
- **Problem**: The problem is to make sure that the producer doesn't add data when it's full and consumer won't try to remove data from empty buffer.
- This is also known as bounded-buffer problem.

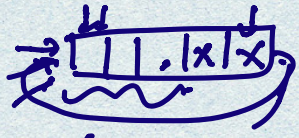
Real Life Example



- If the NIC buffer is full and the kernel tries to insert packets, it wastes CPU.
- Similarly if the network driver tries to consume packets when the buffer is empty, it also wastes CPU.

Producer - Consumer pseudo Code





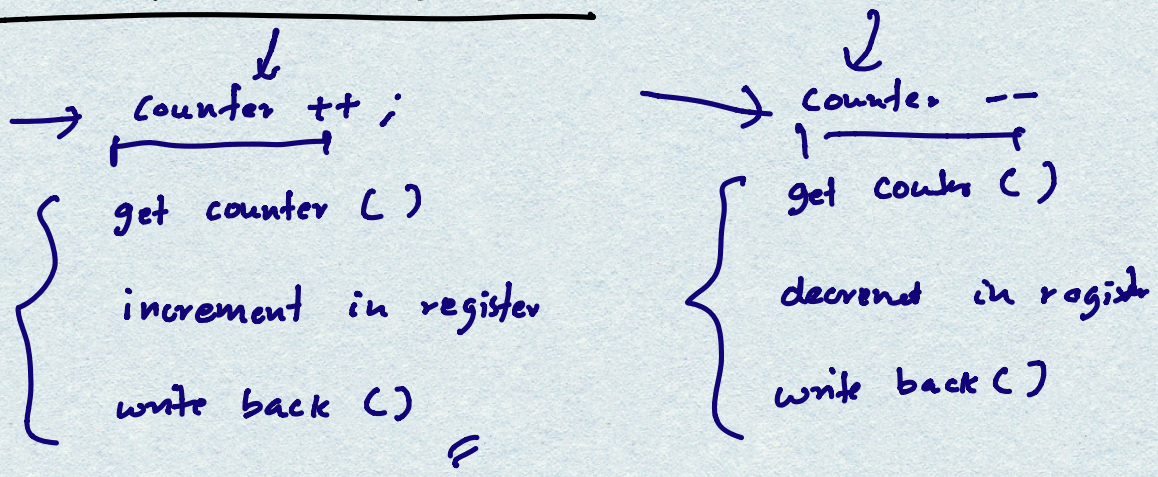
Producer

```
while (True) {
    // produce an item
    while (counter == buffersize) {
        // do nothing
    }
    buffer[in] = item
    in = (in + 1) % buffersize
    Counter++;
}
```

Consumer

```
while (True) {
    while (counter == 0) {
        // do nothing
    }
    item = buffer[out]
    out = (out + 1) % buffer size
    Counter--
}
```

Issues with this solution



Correct Solution using mutex and Semaphore

- when buffer is full, producer sleeps.
- when next Consumer removes the data, it wakes up producer.
- when buffer is empty, Consumer sleeps
- when producer puts next data, it wakes up the Consumer.
- The access to shared Counter Concurrently should not happen. [in, out]

* use mutex to protect critical section and Semaphore for signalling }

* use spinlock to protect critical section and Semaphore for signalling. }

Code :-

```
Mutex mutex;
Semaphore empty = N   // how many slots are empty
Semaphore full  = 0   // how many slots are full
```

Producer

```
while (True) {
    // produce an item
    wait(empty);
    wait(mutex);
    buffer[in] = item
    in = (in+1) % buffersize
    signal(mutex);
    signal(full);
}
```

Consumer

```
while (True) {
    wait(full);
    wait(mutex);
    item = buffer[out]
    out = (out+1) % buffersize
    signal(mutex);
    signal(empty);
    // consume item
}
```

empty = 0

if while (full < 0) { sleep }

full + -1

full = 0

+1