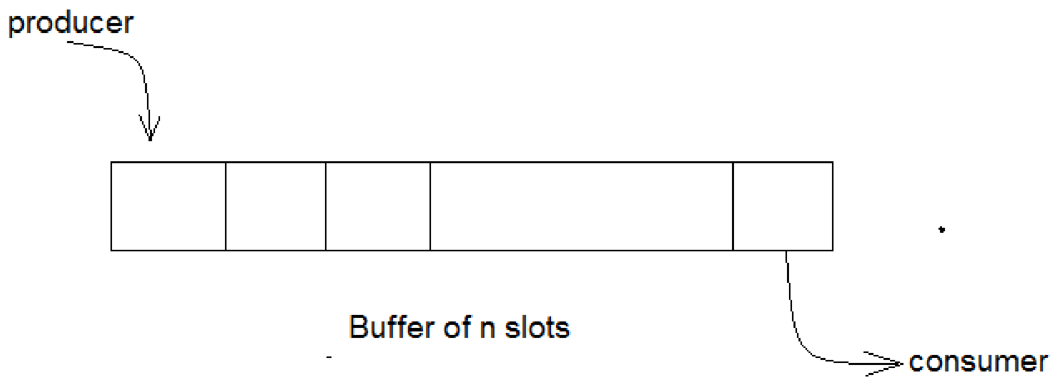
Bounded Buffer Problem (Producer-Consumer problem)

Bounded buffer problem, which is also called **producer consumer problem**, is one of the classic problems of synchronization.

Let's start by understanding the problem here, before moving on to the solution and program code.

What is the Problem Statement?

There is a buffer of `n` slots and each slot is capable of storing one unit of data. There are two processes running, namely, **producer** and **consumer**, which are operating on the buffer.



Bounded Buffer Problem

A producer tries to insert data into an empty slot of the buffer.

A consumer tries to remove data from a filled slot in the buffer. As you might have guessed by now, those two processes won't produce the expected output if they are being executed concurrently.

There needs to be a way to make the producer and consumer work in an independent manner.

Here's a Solution

One solution of this problem is to use semaphores. The semaphores which will be used here are:

- `m`, a **binary semaphore** which is used to acquire and release the lock.
- `empty`, a **counting semaphore** whose initial value is the number of slots in the buffer, since, initially all slots are empty.
- `full`, a **counting semaphore** whose initial value is `0`.

At any instant, the current value of empty represents the number of empty slots in the buffer and full represents the number of occupied slots in the buffer.

The Producer Operation

The pseudocode of the producer function looks like this:

```
do
{
    // wait until empty > 0 and then decrement 'empty'
    wait(empty);
    // acquire lock
    wait(mutex);

    /* perform the insert operation in a slot */

    // release lock
    signal(mutex);
    // increment 'full'
    signal(full);
}
while(TRUE)
```

- Looking at the above code for a producer, we can see that a producer first waits until there is at least one empty slot.
- Then it decrements the **empty** semaphore because, there will now be one less empty slot, since the producer is going to insert data in one of those slots.
- Then, it acquires lock on the buffer, so that the consumer cannot access the buffer until producer completes its operation.
- After performing the insert operation, the lock is released and the value of **full** is incremented because the producer has just filled a slot in the buffer.

The Consumer Operation

The pseudocode for the consumer function looks like this:

```
do
{
    // wait until full > 0 and then decrement 'full'
    wait(full);
    // acquire the lock
    wait(mutex);

    /* perform the remove operation in a slot */

    // release the lock
    signal(mutex);
    // increment 'empty'
    signal(empty);
}
while(TRUE);
```

- The consumer waits until there is at least one full slot in the buffer.
- Then it decrements the **full** semaphore because the number of occupied slots will be decreased by one, after the consumer completes its operation.
- After that, the consumer acquires lock on the buffer.
- Following that, the consumer completes the removal operation so that the data from one of the full slots is removed.
- Then, the consumer releases the lock.
- Finally, the **empty** semaphore is incremented by 1, because the consumer has just removed data from an occupied slot, thus making it empty.