**OS MINI PROJECT**  C-13

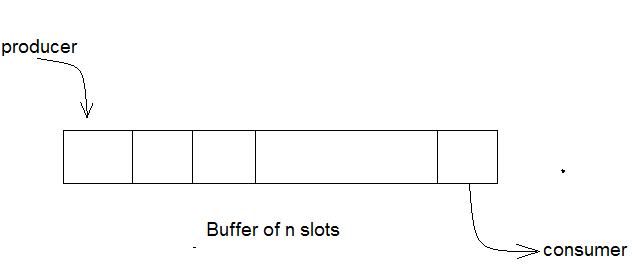
Title of Project: **Producer Consumer Problem using Semaphores**.

Name**: Jenil Jain-55 , Vedant Jasani-59**

Theory:

**What is Producer Consumer Problem?**

In [computing](https://en.wikipedia.org/wiki/Computing), the **producer–consumer problem**[[1]](https://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem#cite_note-ostep1-1)[[2]](https://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem#cite_note-ostep2-2) (also known as the **bounded-buffer problem**) is a classic example of a multi-[process](https://en.wikipedia.org/wiki/Process_(computing)) [synchronization](https://en.wikipedia.org/wiki/Synchronization_(computer_science)) problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size [buffer](https://en.wikipedia.org/wiki/Buffer_(computer_science)) used as a [queue](https://en.wikipedia.org/wiki/Queue_(data_structure)). The producer's job is to generate data, put it into the buffer, and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.



**Solution for the problem**:

The solution for the producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer. The solution can be reached by means of [inter-process communication](https://en.wikipedia.org/wiki/Inter-process_communication), typically using [semaphores](https://en.wikipedia.org/wiki/Semaphore_(programming)). An inadequate solution could result in a [deadlock](https://en.wikipedia.org/wiki/Deadlock) where both processes are waiting to be awakened. The problem can also be generalized to have multiple producers and consumers.

**Solution using Semaphores:**

**Semaphore** is an integer variable whose value can be accessed and changed by two operations wait and signal. In our program we have made use of binary semaphores **semProd** and  **semCon** which handle the sequencing of get() and put() calls.

* Before put() can produce an item, it must acquire a permit from semProd. After it has produced the item, it releases semCon.
* Before get() can consume an item, it must acquire a permit from semCon. After it consumes the item, it releases semProd.
* This “give and take” mechanism ensures that each call to put() must be followed by a call to get( ).
* Also notice that semCon is initialized with no available permits. This ensures that put() executes first. The ability to set the initial synchronization state is one of the more powerful aspects of a semaphore.

**CODE:**

import java.util.concurrent.Semaphore;

class Q

{

int item;

static Semaphore semCon = new Semaphore(0);

static Semaphore semProd = new Semaphore(1);

void get()

{

try{

semCon.acquire();

}

catch(InterruptedException e) {

System.out.println("InterruptedException caught");

}

System.out.println("Consumer consumed item : " + item);

semProd.release();

}

void put(int item)

{

try {

semProd.acquire();

} catch(InterruptedException e) {

System.out.println("InterruptedException caught");

}

this.item = item;

System.out.println("Producer produced item : " + item);

semCon.release();

}

}

class Producer implements Runnable

{

Q q;

Producer(Q q) {

this.q = q;

new Thread(this, "Producer").start();

}

public void run() {

for(int i=0; i<10; i++)

try{

q.put(i);

Thread.sleep(1000);

}

catch(InterruptedException e){

}

}

class Consumer implements Runnable

{

Q q;

Consumer(Q q){

this.q = q;

new Thread(this, "Consumer").start();

}

public void run()

{

for(int i=0; i<10; i++)

try{

q.get();

Thread.sleep(2200);

}

catch(InterruptedException e){}

}

}

class Sem

{

public static void main(String args[])

{

Q q = new Q();

new Consumer(q);

new Producer(q);

}

}

**OUTPUT:**

