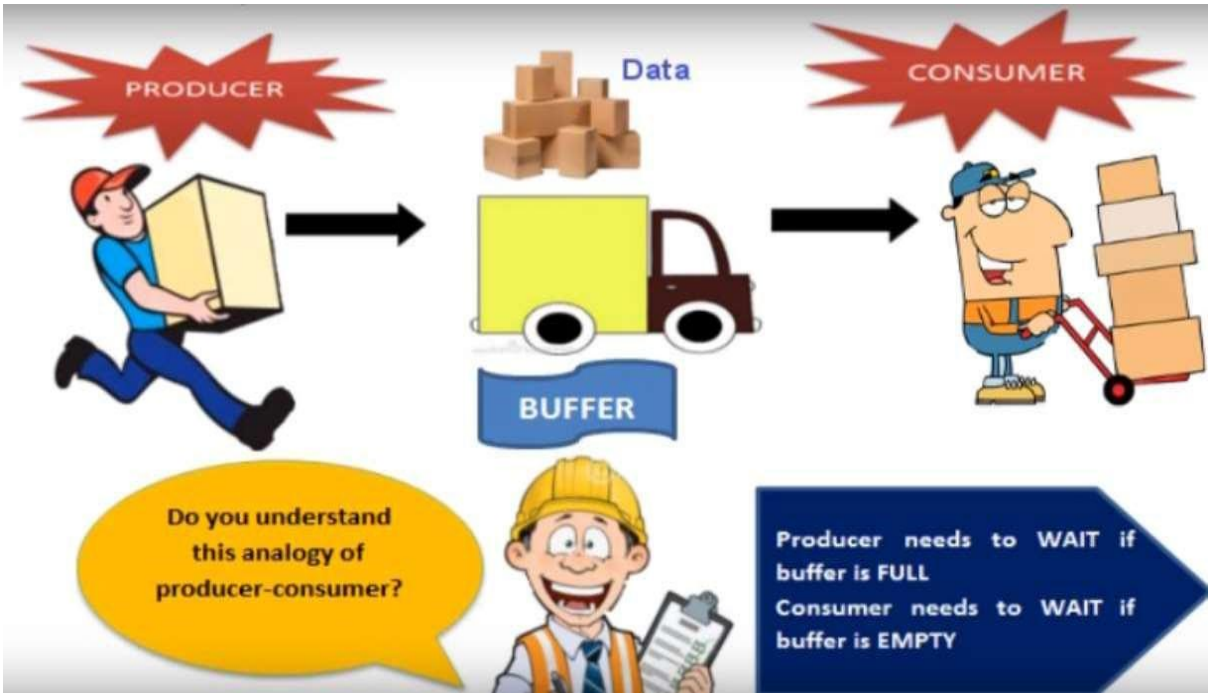
Problem In Operating System

Producer Consumer Problem

The producer–consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. In the problem, two processes share a fixed-size buffer. One process produces information and puts it in the buffer, while the other process consumes information from the buffer. These processes do not take turns accessing the buffer, they both work concurrently.
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.
Producer :- The producer's job is to generate a piece of data, put it into the buffer and start again.
Consumer :- The consumer is consuming the data (i.e., removing it from the buffer) one piece at a time.

Real Life Example:



Solution for the Producer - Consumer

- Producer wants to put but buffer- full Go to sleep and wake up when consumer takes one or more
- Consumer wants to take but buffer-empty go to sleep and wake up when producer puts one or more

The producer-consumer problem using semaphores

```
#define N 100                                   /* number of slots in the buffer */
typedef int semaphore;                          /* semaphores are a special kind of int */
semaphore mutex = 1;                            /* controls access to critical region */
semaphore empty = N;                            /* counts empty buffer slots */
semaphore full = 0;                             /* counts full buffer slots */

void producer(void)
{
     int item;

     while (TRUE) {                             /* TRUE is the constant 1 */
          item = produce_item( );                /* generate something to put in buffer */
          down(&empty);                          /* decrement empty count */
          down(&mutex);                          /* enter critical region */
          insert_item(item);                     /* put new item in buffer */
          up(&mutex);                            /* leave critical region */
          up(&full);                             /* increment count of full slots */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                             /* infinite loop */
          down(&full);                           /* decrement full count */
          down(&mutex);                          /* enter critical region */
          item = remove_item( );                 /* take item from buffer */
          up(&mutex);                            /* leave critical region */
          up(&empty);                            /* increment count of empty slots */
          consume_item(item);                    /* do something with the item */
     }
}
```

/*The most common real-life example of the Producer-Consumer algorithm is a process, called print spooling. Print spooling refers to putting jobs(in this case documents that are about to be printed)into a special location(buffer) in either computer memory, or hard disk, so that a printer could access this document whenever the printer is ready. There are a couple of advantages of spooling.
First of all the printer can access data from the buffer at any rate that is suitable for the printer.
Secondly the work of computer is not interrupted while printing, thus a user can perform other
tasks.

*/