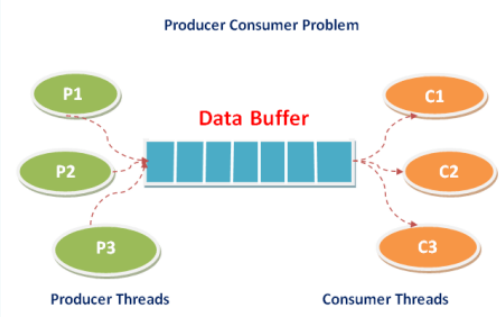
What is Producer Consumer Problem?

The Producer-Consumer issue is a Classic Synchronization Problem. It is also known as Bounded Buffer Problem. This problem focuses primarily on two different tasks: Producer and Consumer. Both of them share a fixed size and a common buffer.

- The producer creates data and puts it into the buffer and restarts it.
- The consumer consumes the data. In other words, the consumer removes the data from the buffer that the producer has created.



Write a parallel code to implement the above problem:

- Case 1: we have only one producer and only one consumer. In this case, the producer fills **one or more** elements then the consumer reads **one or more** elements.
- Case 2: we have only one producer and only one consumer. In this case, the producer fills **all** elements then the consumer reads all elements before reproducing the data.
- Case 3: we have many consumers and many producers. Let us assume that each producer **writes in a chunk** of the buffer and each consumer reads a chunk of the buffer. The buffer should be fully produced before consuming and should be fully consumed before producing it again.

Array

Define an array using c language with its appropriate functions for filling (random integer number between 1 and 100) and displaying its values.

Implement in serial:

- A function ***searchMax()*** in order to return the maximum of the values in the array.
- A function ***searchElement ()*** in order to search an element in the array. It returns 1 if it exists, 0 otherwise.
- A function ***nbOccurencesElement ()*** in order to count the number of occurrences of an element in the array.
- A function ***multEven()*** that calculates the multiplication of all even numbers in the array.
- A function ***compareArrays()*** that compares two arrays. It returns 1 if they are identical, 0 otherwise.
- A function ***average ()*** that calculates the average of the values in the array.

Parallel your code and give the time of processing in serial and parallel code (vary the number of the threads and eventually the schedule mode in the for loop).

LinkedList

You will have the file LinkedList.c

Implement in serial:

- A function ***searchMax()*** in order to return the maximum of the values in the list.
- A function ***searchElement ()*** in order to search an element in the list. It returns 1 if it exists, 0 otherwise.
- A function ***nbOccurencesElement ()*** in order to count the number of occurrences of an element in the list.
- A function ***multEven()*** that calculates the multiplication of all even numbers in the list.
- A function ***compareList()*** that compares two lists. It returns 1 if they are identical, 0 otherwise.
- A function ***average ()*** that calculates the average of the values in the list.

Parallel your code and give the time of processing in serial and parallel code (vary the number of the threads and eventually the schedule mode in the for loop).

Hint: Try to divide the list into fragments of N nodes based on the number of threads and distribute every fragment work (via loop) to a thread. Each thread can send their result to the main via join.