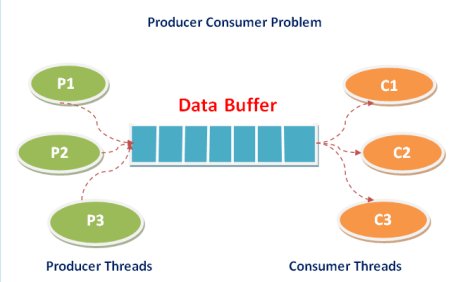
Producer Consumer Problem

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.



Using the provided [ProducerConsumer.c](https://lustd-my.sharepoint.com/personal/kamal_beydoun_ul_edu_lb/Documents/Parallel%20Computing/Slides%20and%20Exercices/Exercices%20labs%20and%20solutions/Exercices%20Examples%20PThreads%20OpenMP%20MPI/PThreads%20Exercices/Producer_Consumer.c) file, write a parallel code to implement the above problem using PThreads library:

* 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 only one producer and only one consumer. In this case, the producer **fills in one or more chunk(s)** of the buffer, then the consumer **reads one or more chunks of the buffer**.
* Case 4: **we have many consumers and many producers**. Let us assume that each producer **fills in one or more chunk** of the buffer and each consumer **reads one or more chunk of the buffer**. The buffer should be **fully produced** before consuming and should be **fully consumed** before producing it again.
* Case 5: **we have many consumers and many producers**. Let us assume that each producer **fills in only a chunk** of the buffer and each consumer **reads only a chunk of the buffer**. The buffer should be **fully produced** before consuming and should be **fully consumed** before producing it again.

Implementing Multithreaded Operations

# Objective:

In this exercise and using the provided [FactorialMinMax.c](https://lustd-my.sharepoint.com/personal/kamal_beydoun_ul_edu_lb/Documents/Parallel%20Computing/Slides%20and%20Exercices/Exercices%20labs%20and%20solutions/Exercices%20Examples%20PThreads%20OpenMP%20MPI/PThreads%20Exercices/FactorialMinMax.c) file, you will enhance your understanding of POSIX Threads (PThreads) by implementing functions that calculate the factorial of a number, and find the minimum and maximum values in an array using multithreading. The results will be returned to the main function through thread exits.

# Requirements:

1. **Shared Data:**
   * An integer variable **X** and an integer array **arr** of size **N** (defined by **#define N <size>**), both populated by user input.
2. **Functions to Implement:**
   * **factorial()**: Calculates the factorial of **X** and exits with the result using **pthread\_exit**.
   * **min()**: Finds the minimum value in **arr** and **exits** with the result.
   * **max()**: Finds the maximum value in **arr** and **exits** with the result.
   * **minMax()**: Manages threads for **min** and **max** functions, stores their results in an array, and **exits** with this array.
3. **Main Function:**
   * Retrieves the factorial of **X**, and both the minimum and maximum values from the array by joining threads initiated for these computations.

# Instructions:

* You will need to implement the thread creation and management within these functions using **PThread library functions**.
* Ensure proper synchronization (if necessary) to avoid race conditions and data inconsistency.