Lab 5-1

1. Reference Code:

- **Lab 4_ex**
- **The Bounded-Buffer Problem in the Lecture slides.**

2. Overall Constraints:

For mutual exclusion locks (**Data Synchronization**): ***spinlock*** or ***mutex***
For synchronization of activities between threads (**Process Synchronization**): ***semaphore*** or ***condition variable***

3. Lab Assignment: Submission date on the moodle (**Mar. 6, Fri. 1 pm**),
Online Grading during lecture sessions or Offline grading using the submitted files.

- **Specific Constraints**:
  1) **n** buffers, each can hold one item (A global array: ***char arr[n]***)
  2) **Mutex *mtx*** initialized to the value 1
  3) A **condition variable *full*** initialization
  4) A **condition variable *empty*** initialization

- In the main thread: create two threads (one is **producer thread** the other is **consumer thread**).
  - But, explicitly **consumer thread MUST** be created firstly (e.g., give 1 second sleep between the creation of the two threads).

- The **consumer** thread prints the buffer values and the number of buffer size after the operation (e.g., [AA] [2], [A] [1] or [] [0] etc. **NOTE**: from 0 to buffer size-1), and the **producer** thread prints also the number of buffer size after the operation (e.g., A 1, AA 2 or AAA 3 etc. **NOTE:** from 1 to buffer size).
  - But, the bounded-buffer problem **MUST** be synchronized (Process Sync) with the protection of the critical section problem (Data Sync) in the shared array (i.e., the ***arr*** global array) by operating the buffer array and printing.

  - **Constraint1**: **MUST** use **a mutex** and **two condition variables**.
  - **Constraint2**: Semaphores **MUST** be implemented **outside the critical section**.

  - **Assumption**:
    For convenience, can use **n** = 10, (but **MUST** be scalable up to any **n** buffers).
    For convenience, create and test only 20 operations, (but **MUST** be scalable up to any N operations).

- You **MUST** implement the synchronization techniques **inside the created threads** (not the main thread) except initialization.

- Lastly, the expected **Output** could be like below. (**NOTE**: the results depend on the remainder section or speed. Please test the speed variations of the two threads using the **usleep() API** or **for loop**.

Expected Output 1: Producing speed is faster than consuming speed.

jurn@ZBookG4:~/work/cs332$ **./lab5_1**
con thread
pro thread
A 1
[] [0]
A 1
AA 2
AAA 3
AAAA 4
AAAAA 5
AAAAAA 6
[AAAAA] [5]
AAAAAA 6
AAAAAAA 7
AAAAAAAA 8
AAAAAAAAA 9
AAAAAAAAAA 10
[AAAAAAAAA] [9]
AAAAAAAAAA 10
[AAAAAAAAA] [9]
AAAAAAAAAA 10
[AAAAAAAAA] [9]
AAAAAAAAAA 10

Expected Output 1: Consuming speed is faster than producing speed. (Always, the consumer will wait the first produced item)

jurn@ZBookG4:~/work/cs332$ **./lab5_2**
con thread
pro thread
A 1
[] [0]
A 1
[] [0]

A 1
[] [0]
A 1
[] [0]
A 1
[] [0]
A 1
[] [0]
A 1
[] [0]
A 1
[] [0]
A 1
[] [0]
A 1
[] [0]