[21AIE202 Operating Systems (2022 Odd)](https://amritauniv.sharepoint.com/sites/21AIE202OperatingSystems2022Odd)

**Lab Exercise** 8

Getting it all together   
Duration : 2 hours

All the related files can be accessed from our Sharepoint course page.

You are give a program (write-reader.c) that implements a writer-reader problem. In this problem, there is a shared buffer that can be read by multiple readers and written to by a single writer. The program uses synchronization techniques to ensure that there are no race conditions or deadlocks.

The program have the following constraints:

* The buffer have a maximum size of 10.
* The writer writes to the buffer every second.
* The readers read from the buffer every 2 seconds.
* If the buffer is full, the writer waits until a reader has read from the buffer.
* If the buffer is empty, the reader waits until the writer has written to the buffer.

The program output the state of the buffer after each write or read operation.

Run this program and see the output. Read, understand and analyze the code. Try to understand the following:

1. Why are full, empty semaphores used ?

2. Why is empty semaphore initialized to BUFFER\_SIZE ?

3. What happens if both the reader and writer sleeps only for 1s each (not 1s and 2s) ?

4. What happens if reader sleeps for 1s and writer sleeps for 2s ?

5. Do you think your program output is correct ? Do you find any inconsistency regarding what is expected to be printed, and what is actually printed as the contents of the buffer ?

**Question 1 :**

Rewrite this code so that you are printing the contents of the buffer correctly.

**Question 2 :**

Rewrite this code so that you are using only mutex, and not semaphores.

**Upload q1 and q2 (if completed)**

**By doing this lab you are expected to have the following skillset.**

**Learning objectives:**

1. Debugging semaphores and mutex programs
2. Understanding circular buffers (accessing and printing)
3. Understanding producer-consumer problem
4. Understanding race conditions and synchronization
5. Run test cases to simulate synchronization / race conditions / mutex / semaphores