Date: 19th September 2016

ASSIGNMENT 1

JINAL MASHRUWALA
UFID = 6498-3393
EMAIL –jmashruw@ufl.edu

(II - a) Solution:

Observation:

- Buffered is a shared resource and thus needs to be guarded.
- A producer's function is to insert data in the buffer and it has to notify consumer when buffer is full.
- A consumer's job is to retrieve data from buffer and inform producer in case of buffer underflow.

Problems:

- **Overflow/Underflow**: There can also be a situation where it's possible that the producer generates data at very high speed when compared to speed at which consumer consumes data or vice-a-versa. Thus, we must ensure that a producer is allowed to insert data in buffer until it overflows and consumer is allowed to process data from buffer as long as it's not empty.
- **Deadlock:** As buffer is shared resource. Therefore, access to buffer needs to be guarded. However, this might result in deadlock. For example, let's assume consumer just noticed that buffer count is zero and so it will want to go to sleep. Let's say, before going to sleep it got interrupted and producer is resumed. Now producer generated data and stored it in buffer and will hence signal consumer because buffer count was zero prior to this insertion. But consumer never slept and so this signal is lost. When consumer resumes it goes to sleep. The producer loops until buffer overflows and then it goes to sleep as well.
- **Data Race**: Data race could occur if at least one of the operations on shared location is writing. Also, simultaneous insertion and processing of data from same buffer may interfere with each other and could result in data inconsistency.
- **Atomic Operation:** At times, it might be possible that while one process has partially completed some operation on some I/O device, a context switch might occur and the process might be interrupted. After the process resumes, it is might have to redo entire operation due to lost update of data which it previously stored.
- **Bounded Waiting:** After requesting access or before request is permitted, we must ensure an upper bound on number of times a process can enter its critical section.

(II - f) Solution:

Based on the inputs on executing code for mutex and semaphore, I have observed following outcomes and have depicted in form of bar chart as shown below:

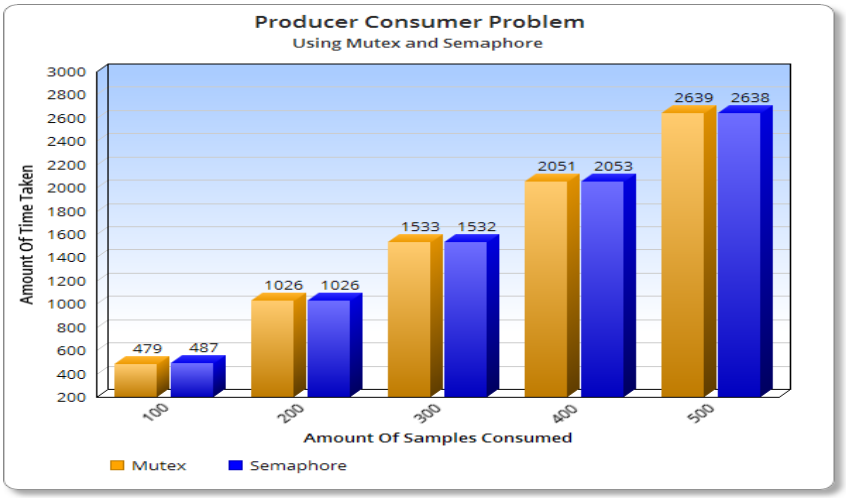

Fig 1: Producer Consumer Problem

(II - e) Solution (Conclusion):

1. Mutex can be released only by the process that acquired it, thus serializing access to shared resource. If there is one resource which you want controlled execution, mutex is more useful.
2. Semaphore allows simultaneous use of shared resource by devising mechanism to wait until one receives signal to access it, thus signaling availability of shared resource. However, one can lock multiple instance of resource at the same time.

It's completely implementation dependent and in our case I find mutex and semaphore performed equally well because both consumer and producer have equal priority and CPU time taken to execute is almost same.

References:

1. Understanding Circular Queue - Data Structures and Algorithms Made Easy by Prof. Narisimha Karumanchi
2. Definition of Producer Consumer Problem - Operating Systems Concept by Prof. Peter Baer Galvin