Lab 1: Process Coordination using Sempahores & Mutexes

Sanjana Shashidhar

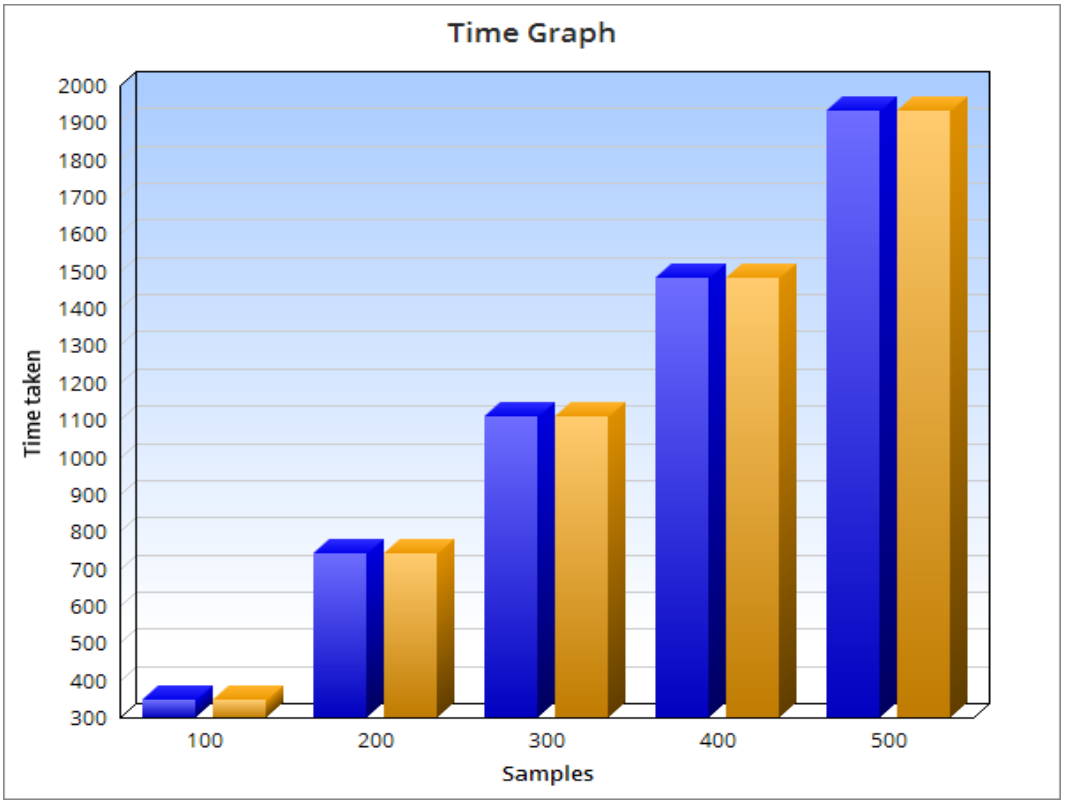
UFID: 48954285

A. Analysis of the problem

1. Producer might input values into the buffer faster than the consumer can read it and output from the buffer. Hence producer must not input any values if the buffer is full.
2. Similarly, consumer might try to read data from the buffer even if the buffer Is empty and the producer may not be ready to input any values into the buffer yet. So the consumer must also not attempt to output any values from the buffer if the producer has not inserted any values into the buffer. Hence we will need a synchronization mechanism like using semaphores.
3. Since the buffer is shared, there is a possibility of race condition occurring. The data input by the producer might be changed before the consumer can read the value and output it. Hence there will be a requirement of a locking mechanism where the process that is currently using the buffer must lock the shared resource and release it once the task is completed. This can be done using critical section concept where a process will not execute the instructions that is currently held under critical section by another process.
4. Another problem that might occur is busy waiting. If we ensure that the producer waits for the consumer to complete reading the value before it can insert the next value, then the producer will have to constantly check if the consumer has completed the task. This will result in unnecessary usage of the processor by the producer, even if it is not executing any task. Hence we will have to find a mechanism to avoid busy waiting.

B. Time Graph for Questions (c) and (d).

The producer in my code produces up to 2000 values. The resulting graph is below:



The results for both Mutex and Semaphore versions are below:

Sample	Mutex Version(Time)	Semaphore Version(Time)
100	349	346
200	739	740
300	1109	1111
400	1480	1482
500	1930	1932

C. Conclusion

The Mutex and Semaphore executions both have almost similar results. Since the I/O being performed here between the producer and consumer is huge, the time being saved by using semaphores over Mutex is not much, almost negligible.

From the implementations we can observe that semaphores help avoid busy waiting and race conditions. They help the processes cooperate with each other and use the given buffer efficiently.

References:

1. http://www.xappsoftware.com/wordpress/2012/09/27/a-simple-implementation-of-a-circular-queue-in-c-language/comment-page-1/ - To see how a circular buffer can be implemented easily.
2. http://www.chartgo.com/ - To create the time graph.