OPERATING SYSTEMS 2
PROGRAMMING ASSIGNMENT 3

Solution To Producer-Consumer Problem

Implementation :

Producer and consumer threads are created at the start of the program
also shared quantites are also initialised at the start of the program
The implementation of both the programs follow the same structure
In one file mutex locks are used where as in other file semaphores are used

avg_time_p* , *avg_time_c - variables to calculate average waiting
time for producer and consumer threads respectively.

At the start of thread (be it consumer or producer), start time is saved in
variable. After finishing current iteration, end time is noted and the difference
is added to corresponding time variable, which is used to calculate average
waiting over the iterations, over all producers.

During the iteration the producer thread busy waits if the buffer is already
full and consumer thread also busy waits if buffer is empty

usleep function is called over to signify that the thread is performing some
complex operation at that time.

Design Of Program :

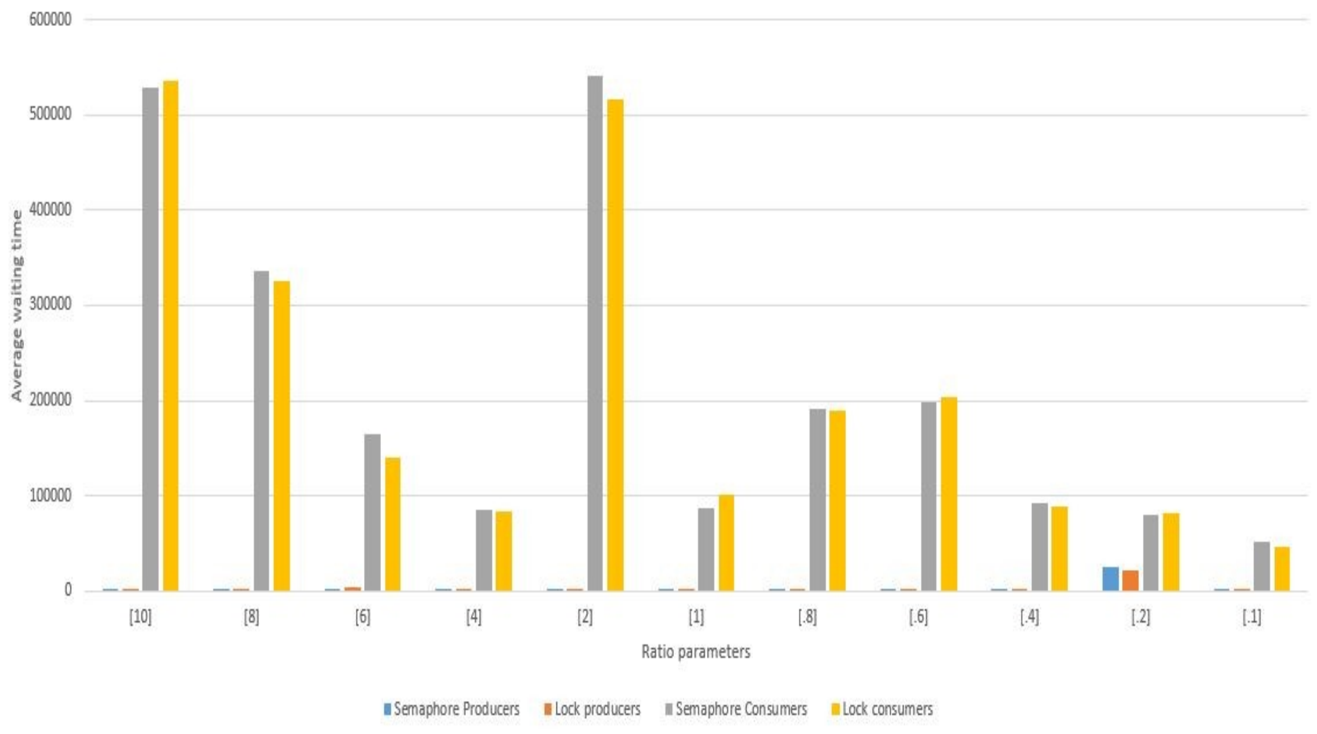
The program has been designed to solve the problem of race
condition in producer-consumer situation, with proper synchronization using
semaphores and locks.

<thread> - for thread creation
<semaphore.h> **-** for creating semaphores
<random> **-** To create random number generating engines.
<unistd.h> **-** To calculate current system time.
<fstream> - Creating file streams.
<mutex> - for creating locks

* The producer threads produce some item to put up in the shared buffer which are then consumed by the consumer threads. Most of the variables are shared thus can create race conditions.

* The produce should wait whenever the buffer is full, whereas the consumers should wait whenever the buffer is empty.

Graph:



From the graphs it is evident that the average values of waiting time is less when we use locks as compared to using semaphores.

One plausible reason may be that semaphore does more complex operations for maintaining atomicity (atomic operations have to be done while – changing the values of semaphore along with busy wait), whereas, for locks, simple atomic instructions, like compare_and_swap does the work and change the values at hardware level in registers.