Operating Systems-2
Assignment 3
Report

<u>Related Terms:</u>

- Entry Section: A segment of the code which is common to all the threads and where threads request & wait to enter the critical section.

- Critical Section(CS): A segment of the code where race condition can happen. So, only one thread should enter this section at a time.

- Exit Section: A segment of the code where threads exit the critical section and another thread is allowed to enter the critical section.

- Remainder Section: The remaining segment of the code which is not critical i.e. where race condition will not happen.

- Waiting Time: It is the time that a threads waits in the entry section to get into the critical section.

<u>Design & Implementation of code:</u>

The design of the two programs is almost the same except the implementation of locks and semaphores for the mutual exclusion of the CS.

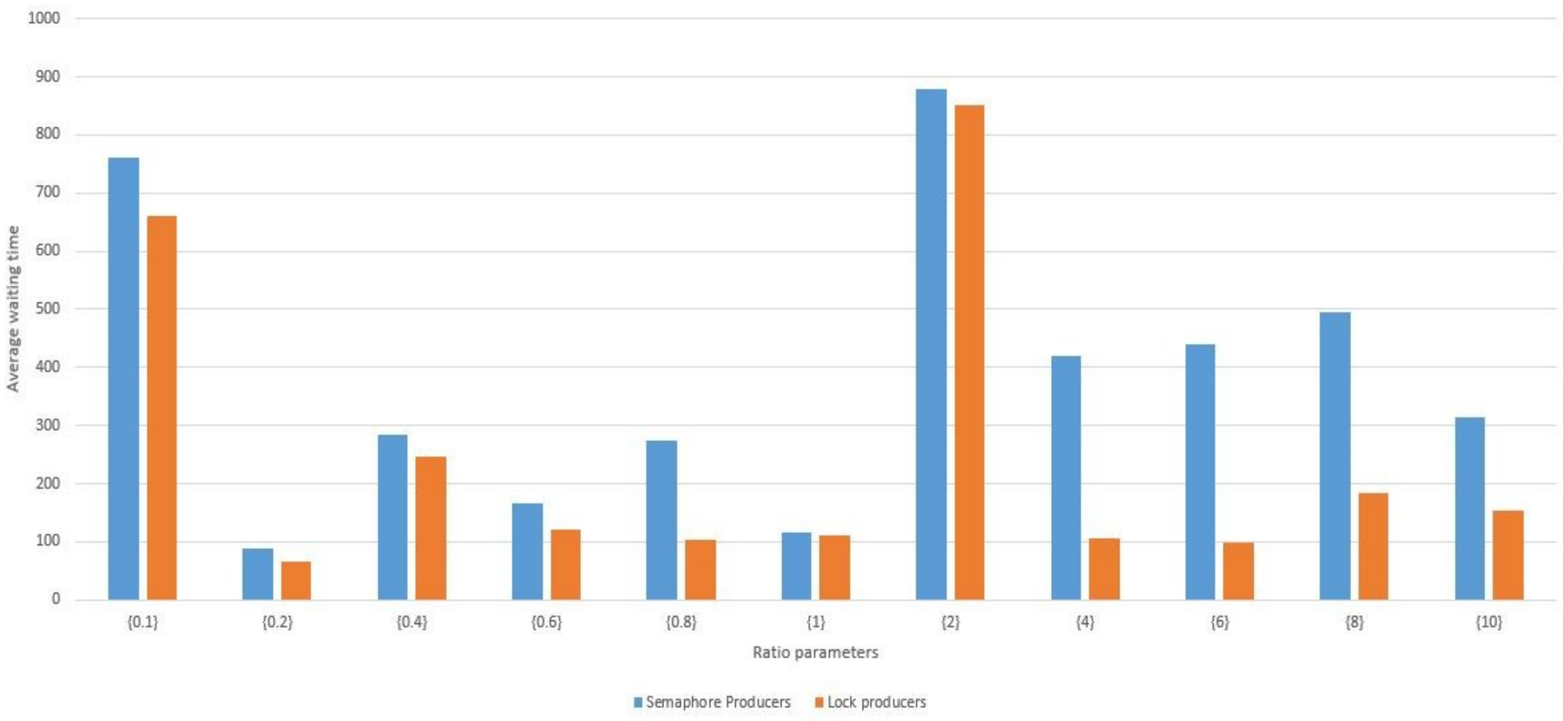
- At the start of the program, main() reads the input from “inp-params.txt” and store the values which are needed in all the threads globally.
- Then, we create two arrays of threads, one for producers and the other for consumers.
- Counter variable in the locks based implementation is an atomic variable. It stores the no. of available blocks ready for consumers. In semaphore based implementation, we don’t need it as this information is stored in the semaphore(full & empty) itself.
- To generate random numbers, we use default_random_engine.

- To implement the exponential distribution of time, we use exponential_distribution<> in library <random>.
- We pass the producer and consumer function to their respective threads and all of them run simultaneously.
- Producer threads measures the time at which the value is produced into the buffer by using functions from library <chrono> and prints that to a log file.
- Consumer threads measures the time at which the value is read from the buffer and prints them to same log file.
- To get the waiting time for each item produced and consumed, we subtract the time at which it came to produce/consume the item and the time at which the item actually got produced/consumed.
- We put the threads to sleep to simulate the actions of producing a item and consuming a item. To implement it, we used sleep() which takes parameters in seconds.
- To make semaphores and locks implementation atomic, we used functions from “<semaphore.h>” and “<atomic>”.

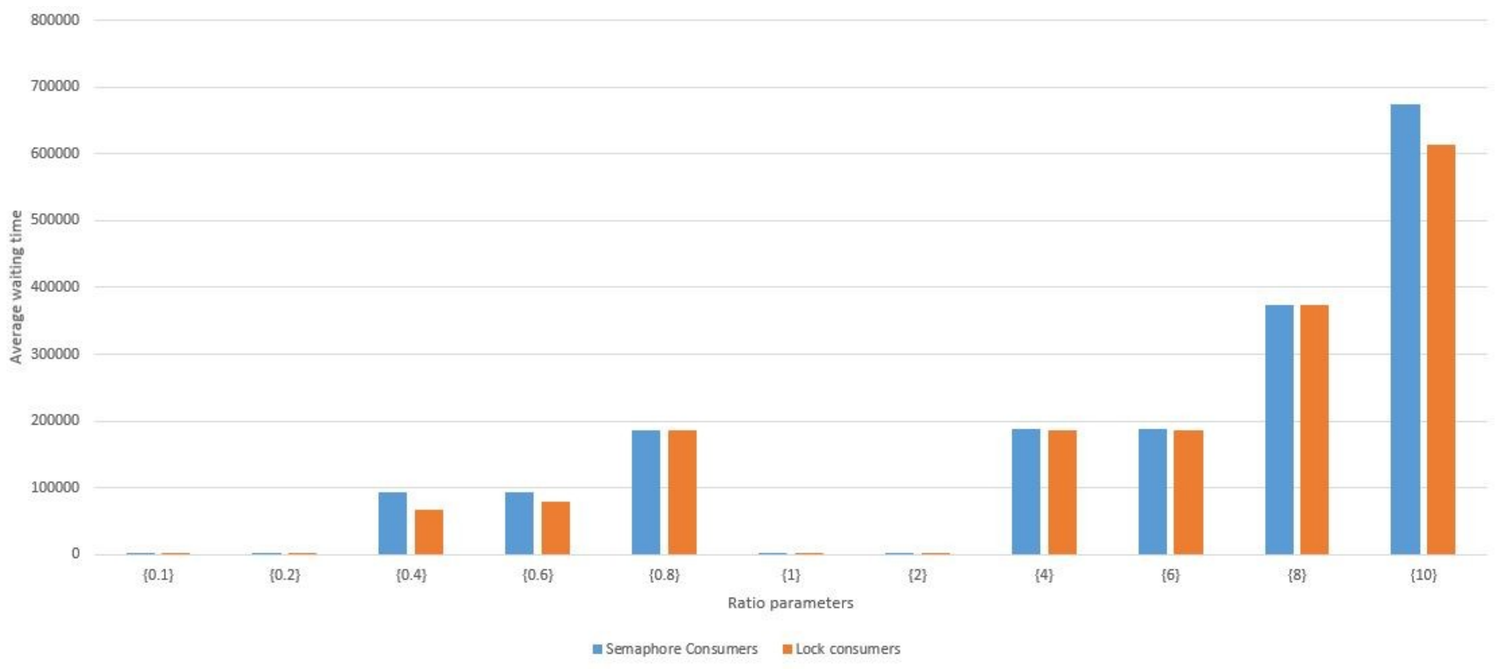
Graphs:

Graph is plotted for
(capacity, np, nc, cntp, cntc) = (100 10 15 15 10).
The values of μp and μc are varied and their ratio is taken as the x-axis.
Average Time taken by the producers and consumers is shown on the y-axis(in microseconds).

Graph for Producers:



Graph for Consumers:



We have drawn two different graphs for producers and consumers as their average time taken had a massive difference. So, producer values wouldn’t have been visible in a single graph.
Also, some values in the consumers graph were very small as compared to the other values as the data depends upon the origiional values of µp and µc & not only on the ratio.

The lock based implementation almost always takes less time as compared to the semaphore based implementation because in semaphore implementation a process goes to sleep if there is no space in buffer or it is empty.