The shared variables between the producers and consumers are the Boolean “done” and the Atomic Integers ”totalProduced” and” totalConsumed”. “TotalProduced” is shared between the producers, and “totalConsumed” is shared between the consumers. The consumers are waiting for the Boolean “done” to be asserted true before it starts to consume from the circular Boolean. There is a circular buffer called “buffer”, this is an array of items, which represents the sales.

There are three semaphores in the circular buffer (***empty***, ***full*** and ***mutex***) . The ***empty*** semaphore lets the producers know there is still room left for the circular buffer. The ***full*** semaphore lets the consumers know that the circular buffer needs to have sales read from it. The ***Mutex*** semaphore is what allows it to lock out access to the buffer for every other consumer/producer except for the consumer/producer that is currently accessing it. A triply nested loop statement was used to iterate through the two buffers (3,10), the three producers (2, 5,10), and the three consumers (2,5,10). There were issues with running the program in the console, so, for accuracies sake, a report of the results is also generated in the file “Sample output file.txt”. If the program is run multiple times without ensuring its completion it has the potential to corrupt one run with insertions from the previous run.

For Buffer = 3; Producer = 2; Consumer =2

Execution time was 11950 ms

For Buffer = 3; Producer = 2; Consumer =5

Execution time was 4619 ms

For Buffer = 3; Producer = 2; Consumer =10

Execution time was 2363 ms

For Buffer = 3; Producer = 5; Consumer =2

Execution time was 11485 ms

For Buffer = 3; Producer = 5; Consumer =5

Execution time was 4706 ms

For Buffer = 3; Producer = 5; Consumer =10

Execution time was 2358 ms

For Buffer = 3; Producer = 10; Consumer =2

Execution time was 11858ms

For Buffer = 3; Producer = 10; Consumer =5

Execution time was 4654ms

For Buffer = 3; Producer = 10; Consumer =10

Execution time was 2371ms

For Buffer = 10; Producer = 2 ; Consumer =2

Execution time was 11775ms

For Buffer = 10; Producer = 2; Consumer =5

Execution time was 4799ms

For Buffer = 10; Producer = 2; Consumer =10

Execution time was 2371ms

For Buffer = 10; Producer = 5 ; Consumer =2

Execution time was 11751ms

For Buffer = 10; Producer = 5; Consumer =5

Execution time was 4784ms

For Buffer = 10; Producer = 5; Consumer =10

Execution time was 2359ms

For Buffer = 10; Producer = 10 ; Consumer =2

Execution time was 11683ms

For Buffer = 10; Producer = 10; Consumer =5

Execution time was 4632ms

For Buffer = 10; Producer = 10; Consumer =10

Execution time was 2344ms

It appears that there is an inverse relationship between the number of consumers and execution time. When there is an increase in consumers the execution time consistently decreased drastically. This is shown in every set, for example:

For Buffer = 10; Producer = 10 ; Consumer =2

Execution time was 11683ms

For Buffer = 10; Producer = 10; Consumer =5

Execution time was 4632ms

For Buffer = 10; Producer = 10; Consumer =10

Execution time was 2344ms

This shows that when the consumer value is at 2, the execution time is roughly 11700ms, while when the consumer is at 5 the execution time is 4632ms. The execution time between 2 consumers and 5 consumers is roughly 2/5 (0.4) times the previous execution time. The same scalar relationship is applicable with the execution time for 5 consumers and the execution time for 10 consumers, it's multiplier is 5/10 (0.5), roughly twice as fast as execution time of 5. The days, months, register# and sales amount are all randomly generated so there is little to no relevant data present.

The producer-consumer program demonstrates the required concepts via the use of semaphores with synchronized access to a shared circular buffer. With the increase of the number of consumers used in the program, the faster the program will run. This is confirmed by observing the results of the programs execution and comparing the execution times from each run.