Producer-consumer Problem

***Contribution***

Pranay Kalva

**Abstract**

The producer/consumer problem is a classical synchronization exercise. The problem defines a specified number of producers which create goods and consumers which consume the goods. A bounded buffer of size n mediates among producers and consumers and adapts to the different execution speeds of all the involved processes respective threads. Besides mutual exclusion issues, semaphores are required to block consumers until consumable goods are inside the buffer and to block producers until there is free space available inside the buffer to deposit additional goods. The producer/consumer problem is solved using semaphores. Code is written in Java.

**Problem description**

* Classical problem of Producer-Consumer arises when two or more producers produce and two or more consumers consume from a bounded buffer.
* Bounded buffer of size ‘N’
* Access entry at 0 through N-1, then “wrap around” to 0 again.
* Producer process writes data to buffer (should not write if buffer is full).
* Consumer process reads data from buffer (should not read if there is no data in buffer).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 |  |  |  |  |  |  | N-2 | N-1 |

IN OUT

* Only one thread should modify the buffer at any time.
* Semaphores are used to handle synchronization.

Implementation Details

We used Java as a platform to solve the problem. The first step in the program accepts user inputs in the form of arguments. The arguments accepted are

* Number of Producers
* Number of Consumers
* Time allocated for the main thread after which it will terminate.

The number of Producers and consumers should be greater than 0. Time allocated should be greater than 1 sec (1000 milliseconds)

We created two thread pools one for producer another for consumer using ExecutorService Class so that multiple threads of producer and consumer can be created. Once the thread pools are created they will call the Producer and Consumer methods i.e., the producer and consumer threads are created and started.

The Producer and Consumer threads in turn call insert data and remove data methods respectively.

**In Insert Data Method:**

Thread sleeps for random amount of time they value ranges from 0 to 999 milliseconds. After the sleep it acquires a permit of semaphore full and adds a random number to the list, then releases a permit of semaphore empty. The same process is placed in an infinite loop and executes until an interrupt is given. We used a Re-Eterent lock in the method so that duplicates won’t be added to the list. This problem arises as we have multiple threads of producers. Semaphore full used in this problem is of size 5 i.e. Semaphore with permit 5. Whenever an acquire is encountered i.e. full.acquire() one permit is reduced, i.e. permit would be reduced to 4. When permits will reach to 0 the semaphore acts as a lock and waits until a release is given i.e. full.release(). Here as we are using a buffer(list) of size 5, after 5 acquires the buffer will be full and hence it will wait for a release.

**In Remove Data Method:**

Thread sleeps for random amount of time they value ranges from 0 to 1999 milliseconds. After the sleep it acquires a permit of semaphore empty and adds a random number to the list, then releases a permit of semaphore full. The same process is placed in an infinite loop and executes until an interrupt is given. We used a Re-Eterent lock in the method so that the value can be removed from the consumer and displayed properly. This problem arises as we have multiple threads of consumers.

Semaphore empty used in this problem is of size 0 i.e. Semaphore with permits. Whenever the buffer is empty the semaphore empty will have 0 permits and so waits for the producer to produce. Once the producer produces, then the semaphore empty gets a permit and the thread start running. So when the list is empty the thread will wait for the producer, which solves the problem.

Adding a number to the list is in a synchronized block with lock ‘lock’ and executes synchronously with the synchronized block with lock ‘lock’ in remove data method, i.e. when one block executes the other block waits for the lock to be released.

Demos and tests

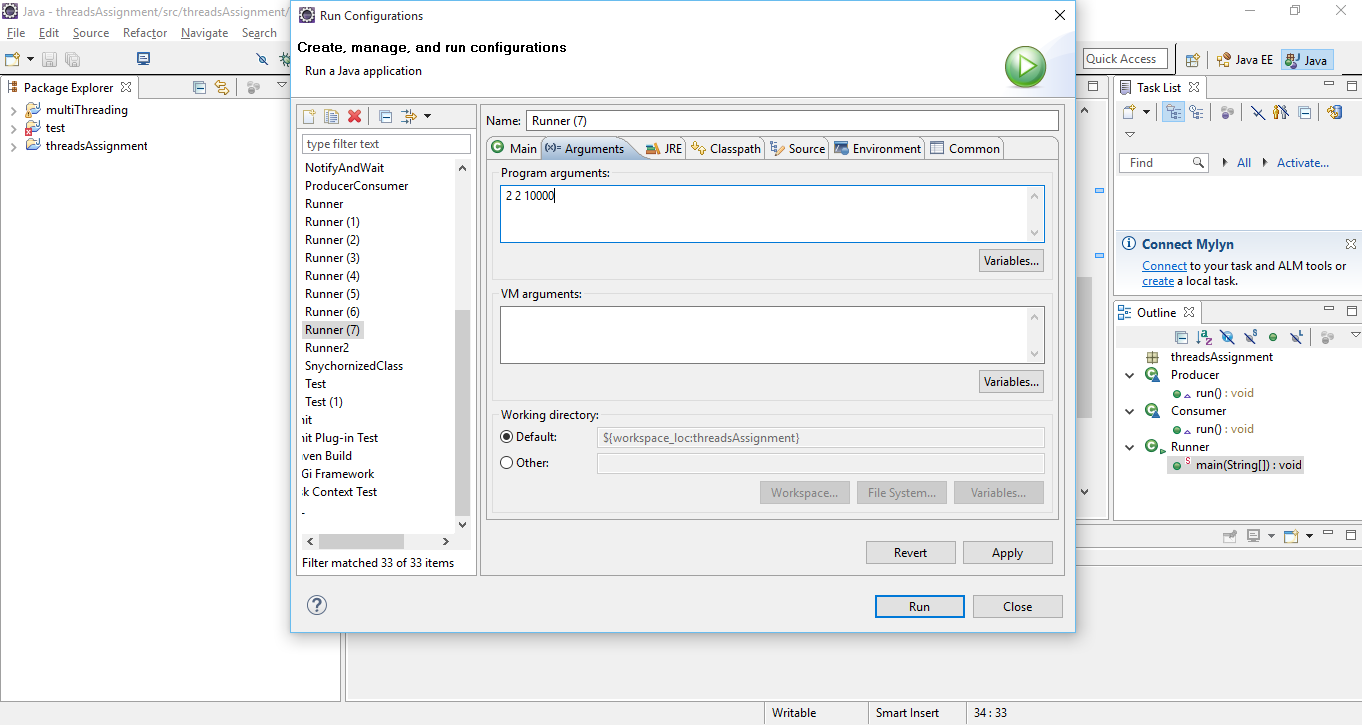
In the program arguments we pass number of producer threads, number of consumer threads and time for main function to execute before termination respectively.

**Test 1:**

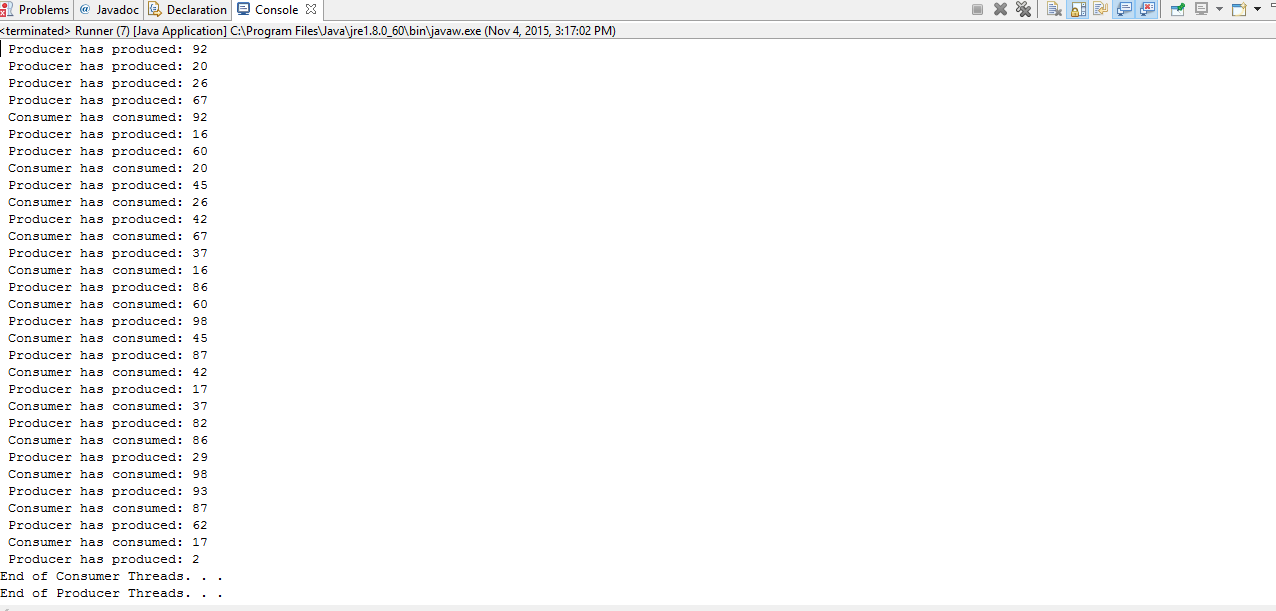
Number of producer threads: 2

Number of consumer threads: 2

Time of execution: 10000ms



Buffer size is 5.

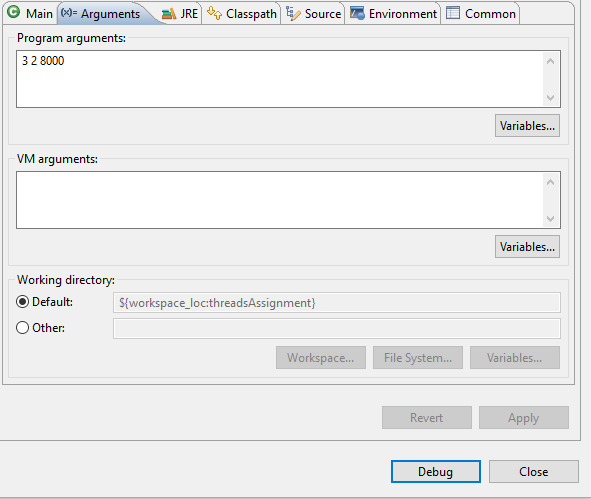
At most 5 data can be produced and placed on buffer if consumer doesn’t consume any of the data.

**Test 2:**

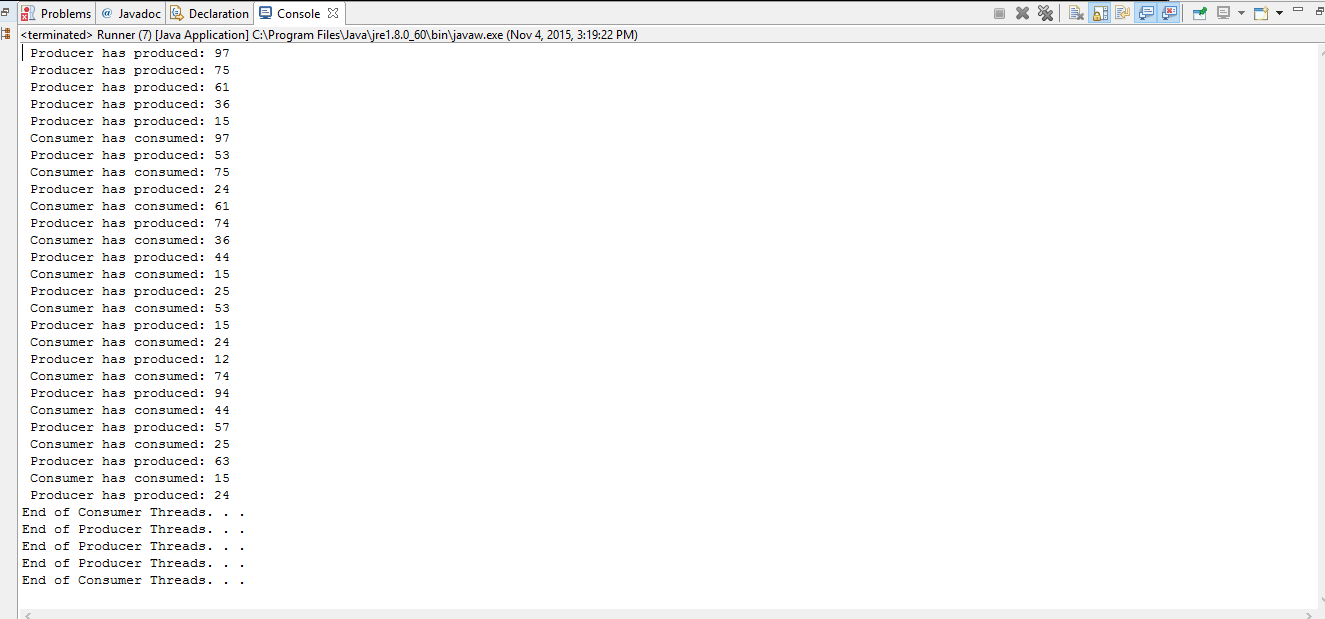
Number of producer threads: 3

Number of consumer threads: 2

Time of execution: 8000ms



As the buffer size is 5 producer waits after producing 5 data until consumer starts consuming the data.

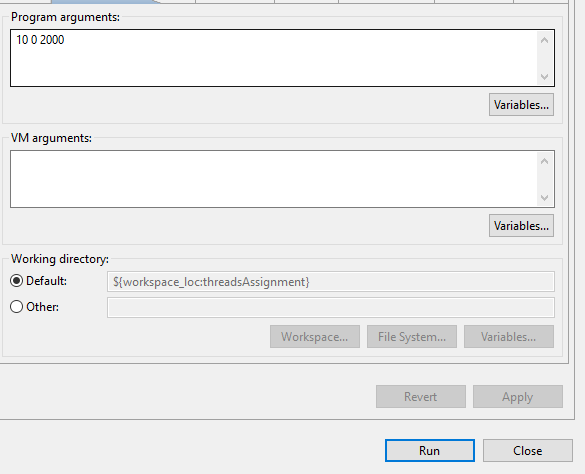


**Test 3:**

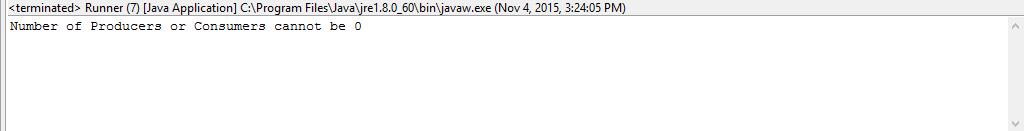
Number of producer threads: 10

Number of consumer threads: 0

Time of execution: 2000ms



As the number of consumers is 0, the program throws an error stating “Number of Producers or Consumers cannot be 0”.

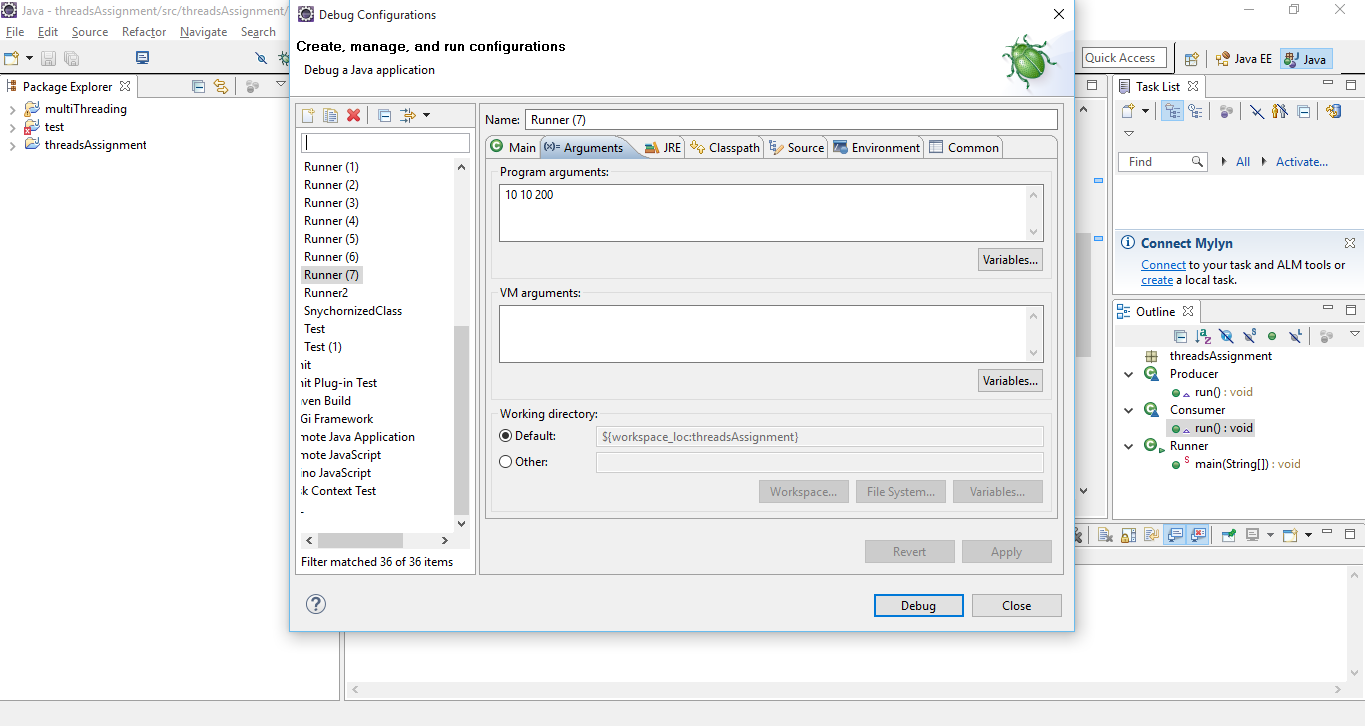


**Test 4:**

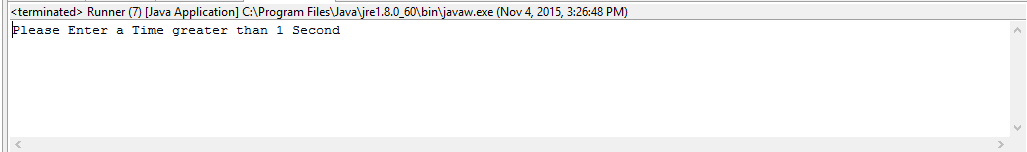
Number of producer threads: 10

Number of consumer threads: 10

Time of execution: 200ms



Execution time should be greaterthan1 sec (1000ms).



**Test 5:**

Number of producer threads: 6

Number of consumer threads: 8

Time of execution: 5000ms

