Experiment No 6
Class: SE Comp
Year: 2020-21
Performed by: Danyl Fernandes, 72

Aim: Write a program to implement a solution of Producer-Consumer problem through Semaphores.

Theory:

Producer Consumer problem:

- The producer consumer problem is a synchronization problem.
- There is a fixed size buffer and the producer produces items and enters them into the buffer.
- The consumer removes the items from the buffer and consumes them.
- A producer should not produce items into the buffer when the consumer is consuming an item from the buffer and vice versa.
- So the buffer should only be accessed by the producer or consumer at a time.
- The producer consumer problem can be resolved using semaphores
- The codes for the producer and consumer process are given as follows:

Semaphores:

- The above problems of Producer and Consumer which occurred due to context switch and producing inconsistent result can be solved with the help of semaphores
- A semaphore is an integer variable in S, that apart from initialization is accessed by only two standard atomic operations - wait and signal, whose definitions are as follows:
    - **Binary Semaphore:**
        - In Binary Semaphore, only two processes can compete to enter into its ***CRITICAL SECTION*** at any point in time, apart from this the condition of mutual exclusion is also preserved.
    - **Counting Semaphore:**
        - In counting semaphores, more than two processes can compete to enter into its ***CRITICAL SECTION*** at any point of time apart from this the condition of mutual exclusion is also preserved.

Solution for Producer code:

```
do {
//produce an item
wait(empty);
wait(mutex);

//place in buffer
signal(mutex);
signal(full);

} while(true)
```

- When the producer produces an item then the value of “empty” is reduced by 1 because one slot will be filled now.
- The value of mutex is also reduced to prevent consumers from accessing the buffer.
- Now, the producer has placed the item and thus the value of “full” is increased by 1.
- The value of mutex is also increased by 1 because the task of the producer has been completed and consumers can access the buffer.

Solution for Consumer code:

```
do {
wait(full);
wait(mutex);

// remove item from buffer
signal(mutex);
signal(empty);

// consumes item
} while(true)
```

- As the consumer is removing an item from the buffer, therefore the value of “full” is reduced by 1 and the value of mutex is also reduced so that the producer cannot access the buffer at this moment.
- Now, the consumer has consumed the item, thus increasing the value of “empty” by 1.
- The value of mutex is also increased so that producers can access the buffer now.

- As the consumer is removing an item from the buffer, therefore the value of “full” is reduced by 1 and the value of mutex is also reduced so that the producer cannot access the buffer at this moment.
- Now, the consumer has consumed the item, thus increasing the value of “empty” by 1.
- The value of mutex is also increased so that producers can access the buffer now.

Conclusion:

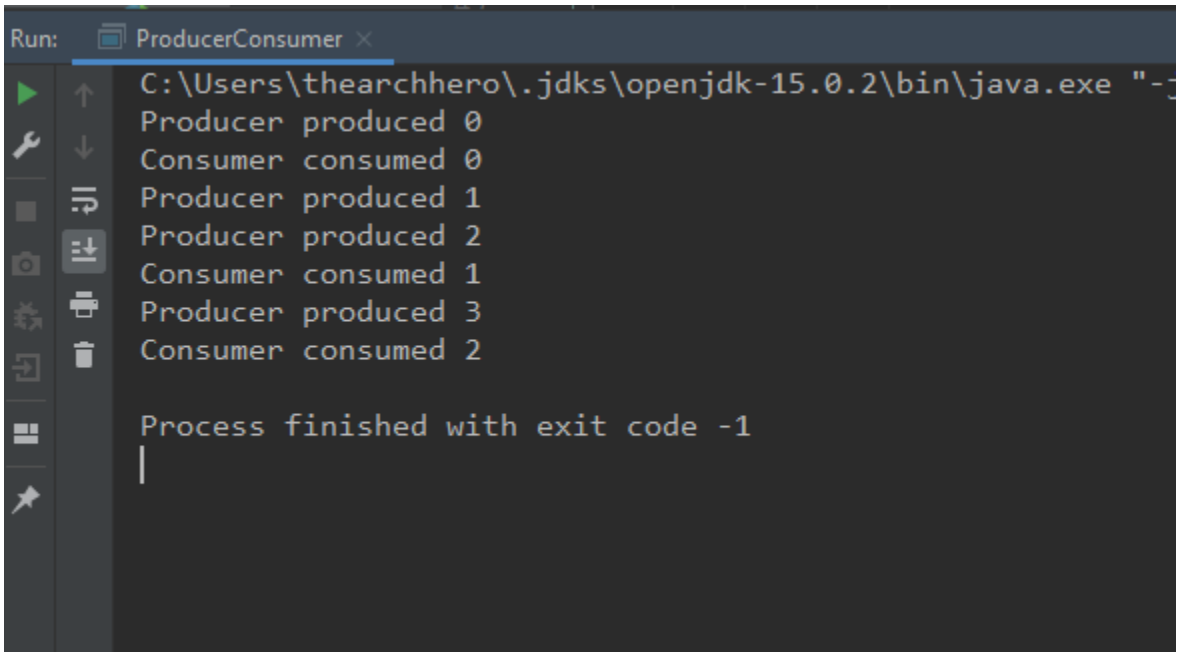
- In this experiment, we were successfully able to implement a solution of the Producer-Consumer problem through Semaphores

Implementation:

```
import java.util.LinkedList;

class ProducerConsumer {
     public static void main(String[] args)
          throws InterruptedException {

     final PC pc = new PC();

     Thread t1 = new Thread(new Runnable() {
          @Override
          public void run() {
               try {
                    pc.produce();
               } catch (InterruptedException e) {
                    e.printStackTrace();
               }
          }
     });

     Thread t2 = new Thread(new Runnable() {
          @Override
          public void run() {
               try {
                    pc.consume();
               } catch (InterruptedException e) {
                    e.printStackTrace();
               }
          }
     });

     t1.start();
     t2.start();

     t1.join();
     t2.join();
     }


     public static class PC {
     LinkedList<Integer> list = new LinkedList<>();
```

```
     int capacity = 2;

     public void produce() throws InterruptedException {
          int value = 0;
          while (true) {
               synchronized (this) {
                    while (list.size() == capacity)
                         wait();

                    System.out.println("Producer produced " +
value);

                    list.add(value++);
                    notify();

                    Thread.sleep(1000);
               }
          }
     }

     public void consume() throws InterruptedException {
          while (true) {
               synchronized (this) {
                    while (list.size() == 0) wait();

                    int val = list.removeFirst();

                    System.out.println("Consumer consumed " +
val);

                    notify();

                    Thread.sleep(1000);
               }
          }
     }
     }
}
```

Output:


Run:
ProducerConsumer
C:\Users\thearchhero\.jdks\openjdk-15.0.2\bin\java.exe "-
Producer produced 0
Consumer consumed 0
Producer produced 1
Producer produced 2
Consumer consumed 1
Producer produced 3
Consumer consumed 2
Process finished with exit code -1