Laboratory 4

Title of the Laboratory Exercise: Producer Consumer problem

1. Introduction and Purpose of Experiment

In computing, the producer-consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue.

- The producer's job is to generate data, put it into the buffer, and start again.
- At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.

Aim and Objectives

Aim

- To develop programs using monitors in Java

2. Experimental Procedure

i. Analyse the problem statement

ii. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code

iii. Implement the algorithm in Java language

iv. Compile the Java program

v. Test the implemented program

vi. Document the Results

vii. Analyse and discuss the outcomes of your experiment

3. Question

Create a multithreaded program in Java

The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

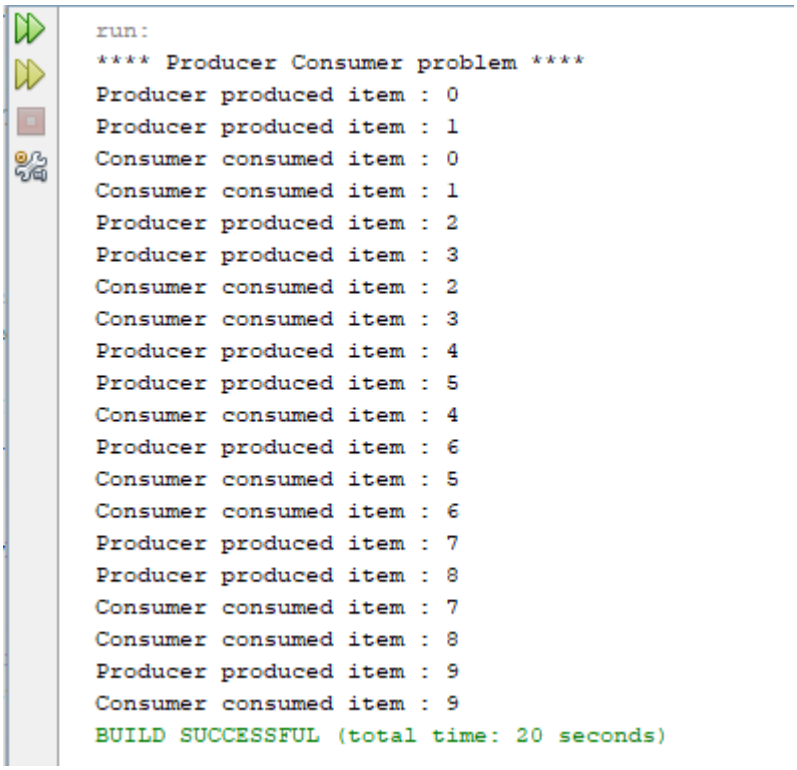
4. Computations/Algorithms/Pseudocode

```
// Create producer thread
      Thread t1 = new Thread(new Runnable() {
         @Override
         public void run()
         {try {
               pc.produce();}
            catch (InterruptedException e) {
               e.printStackTrace();}  }});
      // Create consumer thread
      Thread t2 = new Thread(new Runnable() {
         @Override
         public void run()
         {try {
               pc.consume();}
            catch (InterruptedException e) {
               e.printStackTrace(); }}
public static class PC {
      LinkedList<Integer> list = new LinkedList<>();
      intialize capacity = 2;
      public void produce() throws InterruptedException{
         intisialize value = 0,i=0;
         while (i<10) {
            synchronized (this){
               while (list.size() == capacity)
                  wait();
               Display("Producer produced item : "+ value);
               list.add(value++);
               notify();
               Thread.sleep(1000); }
            i++;}}
      consume() throws InterruptedException{
         initialize i=0;
         while (i<10) {
            synchronized (this){
               while (list.size() == 0)
                  wait();
               initialize val = list.removeFirst();
               display("Consumer consumed item : "+ val);
               notify(); }
            i++; }  } }}
```

5. Source Code

```java
//Program done by Deepak R 18ETCS002041
package lab4;
import java.util.LinkedList;
public class Producer_consumer {
    public static void main(String[] args) throws InterruptedException {
        System.out.println("**** Producer Consumer problem ****");
        final PC pc = new PC();
        // Create producer thread
        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run()
            {
                try {
                    pc.produce();
                }
                catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        // Create consumer thread
        Thread t2 = new Thread(new Runnable() {
            @Override
            public void run()
            {
                try {
                    pc.consume();
                }
                catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        // Start both threads
        t1.start();
        t2.start();
        // t1 finishes before t2
        t1.join();
        t2.join();
    }
    // This class has a list, producer (adds items to list
    // and consumber (removes items).
    public static class PC {
        // Create a list shared by producer and consumer
        // Size of list is 2.
        LinkedList<Integer> list = new LinkedList<>();
        int capacity = 2;
        // Function called by producer thread
        public void produce() throws InterruptedException{
            int value = 0,i=0;
            while (i<10) {
                synchronized (this){
                    // producer thread waits while list
                    // is full
                    while (list.size() == capacity)
                        wait();
                    System.out.println("Producer produced item : "+ value);
                    // to insert the jobs in the list
                    list.add(value++);
                    // notifies the consumer thread that
                    // now it can start consuming
                    notify();
                    // makes the working of program easier
                    // to  understand
                    Thread.sleep(1000);
                }
                i++;
            }
        }
        // Function called by consumer thread
        public void consume() throws InterruptedException{
            int i=0;
            while (i<10) {
                synchronized (this){
                    // consumer thread waits while list
                    // is empty
                    while (list.size() == 0)
                        wait();
                    // to retrive the ifrst job in the list
                    int val = list.removeFirst();
                    System.out.println("Consumer consumed item : "+ val);
                    // Wake up producer thread
                    notify();
                    // and sleep
                    Thread.sleep(1000);
                }
                i++;
            }
        }
    }
```

6. Output



```
run:
**** Producer Consumer problem ****
Producer produced item : 0
Producer produced item : 1
Consumer consumed item : 0
Consumer consumed item : 1
Producer produced item : 2
Producer produced item : 3
Consumer consumed item : 2
Consumer consumed item : 3
Producer produced item : 4
Producer produced item : 5
Consumer consumed item : 4
Producer produced item : 6
Consumer consumed item : 5
Consumer consumed item : 6
Producer produced item : 7
Producer produced item : 8
Consumer consumed item : 7
Consumer consumed item : 8
Producer produced item : 9
Consumer consumed item : 9
BUILD SUCCESSFUL (total time: 20 seconds)
```

The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

An inadequate solution could result in a deadlock where both processes are waiting to be awakened.

7. Analysis and Discussions

Limitations of Experiments

We have checked for only small data so Result may vary if done for large values

Limitations of Results

No

Learning happened

We learnt to solve Producer Consumer Problem using Monitors

Recommendations

No