**Ashesh Kumar**

**501254**

**I.T. – 5**

**Q. Implement Producer-Consumer Algorithm.**

**ProducerConsumerSolution.java**  
import java.util.Vector;

import java.util.logging.Level;

import java.util.logging.Logger;

public class ProducerConsumerSolution {

public static void main(String args[]) {

Vector sharedQueue = new Vector();

int size = 4;

Thread prodThread = new Thread(new Producer(sharedQueue, size), "Producer");

Thread consThread = new Thread(new Consumer(sharedQueue, size), "Consumer");

prodThread.start();

consThread.start();

}

}

**Producer.java**

import java.util.Vector;

import java.util.logging.Level;

import java.util.logging.Logger;

class Producer implements Runnable {

private final Vector sharedQueue;

private final int SIZE;

public Producer(Vector sharedQueue, int size) {

this.sharedQueue = sharedQueue;

this.SIZE = size;

}

@Override

public void run() {

for (int i = 0; i < 7; i++) {

System.out.println("Produced: " + i);

try {

produce(i);

} catch (InterruptedException ex) {

Logger.getLogger(Producer.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

private void produce(int i) throws InterruptedException {

//wait if queue is full

while (sharedQueue.size() == SIZE) {

synchronized (sharedQueue) {

System.out.println("Queue is full " + Thread.currentThread().getName()

+ " is waiting , size: " + sharedQueue.size());

sharedQueue.wait();

}

}

//producing element and notify consumers

synchronized (sharedQueue) {

sharedQueue.add(i);

sharedQueue.notifyAll();

}

}

}

**Consumer.java**

import java.util.Vector;

import java.util.logging.Level;

import java.util.logging.Logger;

class Consumer implements Runnable {

private final Vector sharedQueue;

private final int SIZE;

public Consumer(Vector sharedQueue, int size) {

this.sharedQueue = sharedQueue;

this.SIZE = size;

}

@Override

public void run() {

while (true) {

try {

System.out.println("Consumed: " + consume());

Thread.sleep(50);

} catch (InterruptedException ex) {

Logger.getLogger(Consumer.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

private int consume() throws InterruptedException {

//wait if queue is empty

while (sharedQueue.isEmpty()) {

synchronized (sharedQueue) {

System.out.println("Queue is empty " + Thread.currentThread().getName()

+ " is waiting , size: " + sharedQueue.size());

sharedQueue.wait();

}

}

//Otherwise consume element and notify waiting producer

synchronized (sharedQueue) {

sharedQueue.notifyAll();

return (Integer) sharedQueue.remove(0);

}

}

}

**C:\Users\Ashesh\Documents\Prog\OS>java ProducerConsumerSolution**

**Produced: 0**

**Queue is empty Consumer is waiting , size: 0**

**Produced: 1**

**Consumed: 0**

**Produced: 2**

**Produced: 3**

**Produced: 4**

**Produced: 5**

**Queue is full Producer is waiting , size: 4**

**Consumed: 1**

**Produced: 6**

**Queue is full Producer is waiting , size: 4**

**Consumed: 2**

**Consumed: 3**

**Consumed: 4**

**Consumed: 5**

**Consumed: 6**

**Queue is empty Consumer is waiting , size: 0**