Indian Institute of Information Technology Vadodara

CS266: Operating Systems Lab

Lab 8

Roll No. 201951105 **Name: Nishant Andoriya**

Problem

Implement the producer–consumer problem using semaphore. The processes producer and consumer, which share a common, fixed-size buffer used as a queue/stack. There can be multiple producers and consumers in the system.

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. You have to ensure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer. For ensuring this,

- The producer is to either go to sleep for random time 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. **Input:-**
- Size of the Buffer
- Number of producers (p) and customers (c), $c \leq p$

Output:- if buffer size is 3, number of producers =2 and consumers=1, then output may look like this • Producer 1 produced item : 0

- Consumer 1 consumed item : 0
- Producer 2 produced item : 1
- Producer 1 produced item : 2
- Consumer 1 consumed item : 1
- Consumer 1 consumed item : 2

Ans:-

Code:-

```
import java.util.*;

public class os_lab8 {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the Buffer Size of Queue : ");
        int Qsize = sc.nextInt();
```

```
        System.out.println("Enter the Number of Producers :");
        int Pnum = sc.nextInt();
        System.out.println("Enter the Number of Consumers :");
        int Cnum = sc.nextInt();
        System.out.println("\n\n Output :- \n");
        sc.close();
        // Creation of queue (buffer)
        // that will be shared between consumer and producer
        Vector<Integer> QueueShared = new Vector<Integer>();
        // thread creation
        Thread producerThread = new Thread(new Producer(QueueShared, Qsize, Pnum)
, "Producer");
        Thread consumerThread = new Thread(new Consumer(QueueShared, Qsize, Cnum)
, "Consumer");
        // thread start
        producerThread.start();
        consumerThread.start();
    }
}

class Producer implements Runnable {
    private final Vector<Integer> QueueShared;
    private final int Qsize;
    private final int Pnum;

    public Producer(Vector<Integer> QueueShared, int Qsize, int Pnum) {
        this.QueueShared = QueueShared;
        this.Qsize = Qsize;
        this.Pnum = Pnum;
    }

    @Override
    public void run() {
        int X = 1;
        while (X <= Pnum) {
            for (int i = 0; i < Qsize; i++) {
                System.out.println("Producer " + X + " produced item : " + i);
                try {
                    produce(i);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            X++;
        }
```

```
    }

    private void produce(int i) throws InterruptedException {
        // Wait if the queue is found FULL
        while (QueueShared.size() == Qsize) {
            synchronized (QueueShared) {
                QueueShared.wait();
            }
        }
        // Producing the element and notify consumers
        synchronized (QueueShared) {
            QueueShared.add(i);
            QueueShared.notifyAll();
        }
        ;
    }
}

class Consumer implements Runnable {
    private final Vector<Integer> QueueShared;
    private final int Qsize;
    private final int Cnum;

    public Consumer(Vector<Integer> QueueShared, int Qsize, int Cnum) {
        this.QueueShared = QueueShared;
        this.Qsize = Qsize;
        this.Cnum = Cnum;
    }

    @Override
    public void run() {
        int Y = 1;
        while (Y <= Cnum) {
            for (int i = 0; i < Qsize; i++) {
                System.out.println("Consumer " + Y + " consumed item : " + i);
                try {
                    consume();
                    Thread.sleep(50);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            Y++;
        }
    }
```

```
    private int consume() throws InterruptedException {
        // Wait if the queue is foumd EMPTY
        while (QueueShared.isEmpty()) {
            synchronized (QueueShared) {
                QueueShared.wait();
            }
        }
        // else consume the element and notify the waiting producer
        synchronized (QueueShared) {
            QueueShared.notifyAll();
            return (Integer) QueueShared.remove(0);
        }
    }
}
```

Command through which the program is executed:

- javac os_lab8.java – compiling the code
- java os_lab8 – executing the code

Output:-

```
Enter the Buffer Size of Queue :
4
Enter the Number of Producers :
5
Enter the Number of Consumers :
3


 Output :-

Consumer 1 consumed item : 0
Producer 1 produced item : 0
Producer 1 produced item : 1
Producer 1 produced item : 2
Producer 1 produced item : 3
Producer 2 produced item : 0
Producer 2 produced item : 1
Consumer 1 consumed item : 1
Producer 2 produced item : 2
Consumer 1 consumed item : 2
Producer 2 produced item : 3
Consumer 1 consumed item : 3
Producer 3 produced item : 0
Consumer 2 consumed item : 0
Producer 3 produced item : 1
Consumer 2 consumed item : 1
Producer 3 produced item : 2
Consumer 2 consumed item : 2
Producer 3 produced item : 3
Consumer 2 consumed item : 3
Producer 4 produced item : 0
Consumer 3 consumed item : 0
Producer 4 produced item : 1
Consumer 3 consumed item : 1
Producer 4 produced item : 2
Consumer 3 consumed item : 2
Producer 4 produced item : 3
Consumer 3 consumed item : 3
Producer 5 produced item : 0
```