

بسم الله الرحمن الرحيم

Producer-Consumer Problem

Documentation

➤Pseudocode

Classes:

- 1.1 Main Function.
- 1.2 DataQueue Class.
- 1.3 Producer Class.
- 1.4 Consumer Class.
- 1.5 Message Class.

1.1 Main function

- 1. Start
- 2. Create new DataQueue object and pass queue maxSize.
- 3. Create Producer and Consumer threads as many as desired and pass the DataQueue object as parameter.
- 4. Start Producer and Consumer threads.
- 5. Start queue.
- 6. End.

1.2 DataQueue

class DataQueue

Attributes

```
private int maxSize
private boolean queueStartFlag := false
private Queue <Message> queue
private Semaphore producer_criticalSecLock
private Semaphore consumer_criticalSecLock
private int producerCount := 0
private boolean producerStartFlag := false
private Semaphore producerId_Lock
private Queue <Object> producer_queue
private int consumerCount :=0
private Boolean consumerStartFlag :=false
private Semaphore ConsumerId_Lock
private Queue <Object> consumer_queue
```

Constructors

➤ DataQueue(int size)

```
    mazSize := size
```

```
end
```

Methods

➤ function start()

```
If Not queueStartFlag And ( consumer_queue Not empty Or  
(producer_queue Not empty And maxSize <= 10000)) then
```

```
    print "queue started"
```

```
    queueStartFlag := true
```

```
    wake Next Producer
```

```
    wake Next Consumer
```

```
end
```

```
return queueStartFlag
```

```
end
```

➤ function isProducerStarted()

```
    return producerStartFlag
```

```
end
```

➤ function isConsumerStarted()

```
    return ConsumerStartFlag
```

```
end
```

➤ function getProcucer_Id()

```
    integer id := 0
```

```
    acquire Semaphore producerId_Lock
```

```
    id := producerCount
```

```
    producerCount := producerCount + 1
```

```
    release producer Id section Lock
```

```
    return id
```

```
end
```

```
➤ function getProducerLock()  
    return producer_criticalSecLock  
end
```

```
➤ function getConsumerLock()end  
    return consumer_criticalSecLock  
end
```

```
➤ function getConsumer_Id()  
    integer id := 0  
    acquire consumer Id Lock to Create new id  
    id := consumerCount  
    consumerCount := consumerCount + 1  
    release consumer Id section Lock  
    return id  
end
```

```
➤ function getProducerCount()  
    integer count := 0  
    acquire to get the producer id Lock  
    count := producerCount;  
    release the producer id section Lock  
    return count  
end
```

➤ function getConsumerCount()

integer count := 0

acquire to get the consumer id Lock

count := consumerCount;

release the consumer id section Lock

return count

end

➤ function waitOnProducerQ() throws InterruptedException

Create new object nextProducer of type Object class

synchronized (producer_queue) {

add nextProducer to producer_queue

}

synchronized(nextProd){

wait on the nextProducer object

}

end

➤ function waitOnProducerQ() throws InterruptedException

Create new object nextConsumer of type Object class

synchronized (consumer_queue) {

add nextConsumer to consumer_queue

}

synchronized(nextProd){

wait on the nextConsumer object

}

end

```

➤ function wakeNextProducer() end
    synchronized (producer_queue) {
        if producer_queue Not empty then
            poll head of the producer_queue
            assign the polled value to new object of type Object
            synchronized(nextProducer){
                wake all producers waiting on this object
                if producerStartFlag Not true then
                    print "producer queue is active"
                    producerStartFlag := true;
                end}
            end}
        end}
    end
end

➤ function wakeNextConsumer()
    synchronized (consumer_queue) {
        if consuemr_queue Not empty then
            poll head of the consumer_queue
            assign the polled value to new object of type Object
            synchronized(nextConsumer){
                wake all consumers waiting on nextConsumer
                if consumerStartFlag Not true then
                    print "producer queue is active"
                    consumerStartFlag := true;
                end}
            end}
        end}
    end
end

```

```
➤ function isEmpty()  
    synchronized (queue) {  
        if queue size = zero then return true  
        else return false  
    end  
}  
end
```

```
➤ function isFull()  
    synchronized (queue) {  
        if queue size = maxSize then return true  
        else return false  
    end  
}  
end
```

```
➤ function      add(Message message)  
    synchronized (queue) {  
        add message to queue  
        if added then return true  
        else return false  
    end  
}  
end
```

```

➤ function    remove()

    synchronized (queue) {

        remove the head element of queue

        if removed then return true

        else return false

    end

}

end

➤ function size()

    synchronized(queue){ return queue size }

end

end of class

```

1.3 Producer class

Import Semaphore class from java .util.concurrent

class Producer

Attributes

```

private DataQueue  dataQueue

private boolean  runFlag

private Boolean produced := false

private Semaphore criticalSecLock

private int id

private int  producerCount

```

Constructors

```
➤ Public Producer(DataQueue queue)
    dataQueue := queue
    criticalSecLock := dataQueue.getProducerLock()
    id := dataQueue.getProd_id
    runFlag := true
```

```
end
```

```
*****
```

Methods

```
➤ function run ()
```

```
    produce ()
```

```
end
```

```
➤ function produce()
```

```
    wait on producers queue until notify
```

```
    create message, object of Message type
```

```
    while runFlag = true
```

```
        produced := false
```

```
        message := generateMessage()
```

```
        if no one inside critical section Or producers count < 3 then
```

```
            acquire Lock to enter critical section
```

```
        else
```

```
            wait on producers queue until notify
```

```
            acquire lock to enter critical section
```

```
        end
```

```

while Not produced
    if dataQueue is not full then
        add message to the queue
    else if consumer queue Not started then
        wake next consumer
    else
        produced := false
    end
end
end
release crititcal section lock
if producer count >2 then
    wake next producer
    wait on producers queue until notify
else if producers count = 2 then
    wake new producer
    get producers count

else
    get producers count
end
end
end
end

```

```

➤ function generateMessage()
    create new object of Message type
    return the object
end

```

```

➤ function stop()
    runFlag = false
    wake next producer
end

```

1.4 Consumer class

Import Semaphore class from java .util.concurrent

class Consumer

Attributes

```

private DataQueue  dataQueue
private boolean  runFlag
private Semaphore criticalSecLock
private int id
private int consumersCount

```

Constructors

```

➤ Public Consumer(DataQueue queue)
    dataQueue := queue
    criticalSecLock := dataQueue.getConsumerLock()
    id := dataQueue.getConsumer_id()
    runFlag := true
end

```

Methods

➤ function run()

 consume()

end

➤ function consume()

 wait on consumers queue

 create new message object of Message type

 while runFlag = true

 message := null

 if no one inside critical section Or consumers count < 3 then

 acquire Lock to enter critical section

 else

 wait on consumers queue until notify

 acquire lock to enter critical section

 end

 while message = null

 if dataQueue is not empty then

 remove message from the queue

 else if producers queue Not started then

 wake next producer

 end

 end

 release crititcal section lock

```

        if consumers count > 2 then
            wake next consumer
            wait on consumers queue until notify
        else if consumers count = 2 then
            wake new consumers
            get consumers count
        else
            get consumers count
        end
    end
end
end

```

```

➤ function stop()
    runFlag = false
    wake next consumer
end

```

end of class

1.5 Message class

class Message

Attributes

private int id

private double data

➤ public Constructor (double userData)

data := userData

id := generate_id()

end

➤ Function getValue ()

return data

end

➤ function int getId()

return id

end

➤ function generate_id()

generate new message id

return new id

end

end of class

➤ Deadlock

Examples to explain where Deadlock can occur


Ex 1

- In this example, we'll create two threads, producer1 and consumer1.
 - Thread producer1 calls produce, and consumer1 calls consume.
 - To complete their operations, thread producer needs to acquire first_mutex first and then second_mutex, whereas thread consume needs to acquire second_mutex first and then first_mutex.
 - So, basically, both threads are trying to acquire the locks in the opposite order.
 - Deadlock is possible if producer1 acquires first_mutex and consumer1 acquires second_mutex.
 - Producer1 then waits for second_mutex and Consumer waits for first_mutex.
 - Obviously no one gets the lock to enter their critical section and they remain stuck forever.
- ❖ Solution of this situation is to either acquire locks in same order or to use only one lock and that should be enough same as the code mentioned below :

```

cSecLock.acquire();
// Enter Critical Section
/*****
while(!produced)
    if(!dataQueue.isFull() ){
        produced = dataQueue.add(message);
        ++producer[id];
        System.out.println(id+"# produced "+message.getValue());
    }else if (!dataQueue.isConsStarted() )
        dataQueue.wakeNextConsumer();
*****/

```



Ex 2

- In this example, we'll create N threads, producer1 to producerN.
- There is a queue for threads to wait in until the thread inside critical finishes and releases the lock.
- So, if thread under processing leaves critical section without notifying the head of the queue there will be a deadlock as all threads waiting in the queue will not be notified and they wait for ever with no progress.
- ❖ **Solution is to wake the head of the queue after leaving the critical section.**




Starvation

Example to explain where Starvation can take place

- In this example, we'll create N threads, producer1 to producerN.
 - each thread has priority value assigned to it.
 - all threads call produce method.
 - To complete their operations, threads need to acquire produce_lock.
 - So, if there is a very low priority thread it won't be able to acquire the lock and complete its task if there are many other threads with high priority.
- ❖ Possible solution for this situation is to create a queue for threads to wait in until the thread inside critical finishes and releases the lock then wakes the first thread in the queue.
- ❖ As we can see in this code

```
cSecLock.acquire();  
// Enter Critical Section  
/*****  
  
while(!produced)  
    if(!dataQueue.isFull() ){  
        produced = dataQueue.add(message);  
        ++producer[id];  
        System.out.println(id+"# produced "+message.getValue());  
    }else if (!dataQueue.isConsStarted() )  
        dataQueue.wakeNextConsumer();  
  
/*****  
// Leave Critical Section  
cSecLock.release();  
dataQueue.wakeNextProducer();
```



➤ Explanation for real world application

The application is an application that provides services to people with daily nutritional needs and for each person a certain number is available to him, the number should not exceed in the event that the goods are not available or insufficient, and in the event that the number is available, the person is allowed to take what he wants, and every time the person takes it, the worker asks the user Does he want, even if he wants to do business, he checks the number of goods and the number of users, and in the event that there is no one and the goods are available, the person is allowed to

take, and if it is not available, the user cannot take and the user leaves.