

Producer and consumer with a Circular Buffer



```
/*
Code revised from Book published by
(C) Copyright 1992-2006 by Deitel & Associates, Inc. and
Pearson Education, Inc. All Rights Reserved.

*/
using System;
using System.Threading;

public class Producer
{
    private CircularBuffer sharedLocation;
    private Random randomSleepTime;

    public Producer( CircularBuffer shared, Random random )
    {
        sharedLocation = shared;
        randomSleepTime = random;
    }

    public void Produce()
    {
        for ( int count = 1; count <= 10; count++ ) { Thread.Sleep(
            randomSleepTime.Next( 1, 3001 ) ); sharedLocation.Buffer =
            count; } Console.WriteLine( "{0} done producing. Terminating
            {0}.", Thread.CurrentThread.Name ); } } public class Consumer
    { private CircularBuffer sharedLocation; private Random
        randomSleepTime; public Consumer( CircularBuffer shared,
        Random random ) { sharedLocation = shared; randomSleepTime =
        random; } public void Consume() { int sum = 0; for ( int count
        = 1; count <= 10; count++ ) { Thread.Sleep(
        randomSleepTime.Next( 1, 3001 ) ); sum +=
```

```

sharedLocation.Buffer; } Console.WriteLine("{0} read values
totaling: {1}. Terminating {0}.", Thread.CurrentThread.Name,
sum ); } } public class CircularBuffer { private int[] buffers
= { -1, -1, -1 }; private int occupiedBufferCount = 0; private
int readLocation = 0; private int writeLocation = 0; public
int Buffer { get { lock ( this ) { if ( occupiedBufferCount ==
0 ) { Console.Write( " All buffers empty. {0}
waits.",Thread.CurrentThread.Name ); Monitor.Wait( this ); }
int readValue = buffers[ readLocation ]; Console.Write( " {0}
reads {1} ",Thread.CurrentThread.Name, buffers[ readLocation ]
); --occupiedBufferCount; readLocation = ( readLocation + 1 )
% buffers.Length; Console.Write( CreateStateOutput() );
Monitor.Pulse( this ); return readValue; } } set { lock ( this
) { if ( occupiedBufferCount == buffers.Length ) {
Console.Write( " All buffers full. {0}
waits.",Thread.CurrentThread.Name ); Monitor.Wait( this ); }
buffers[ writeLocation ] = value; Console.Write( " {0} writes
{1} ",Thread.CurrentThread.Name, buffers[ writeLocation ] );
++occupiedBufferCount; writeLocation = ( writeLocation + 1 ) %
buffers.Length; Console.Write( CreateStateOutput() );
Monitor.Pulse( this ); } } } public string CreateStateOutput()
{ string output = "(buffers occupied: " + occupiedBufferCount
+ ") buffers: "; for ( int i = 0; i < buffers.Length; i++ )
output += " " + string.Format( "{0,2}", buffers[ i ] ) + " ";
output += " "; output += " "; for ( int i = 0; i <
buffers.Length; i++ ) output += "---- "; output += " "; output
+= " "; for ( int i = 0; i < buffers.Length; i++ ) { if ( i ==
writeLocation && writeLocation == readLocation ) output += "
WR "; else if ( i == writeLocation ) output += " W "; else if
( i == readLocation ) output += " R "; else output += " "; }
output += " "; return output; } static void Main( string[]
args ) { CircularBuffer shared = new CircularBuffer(); Random
random = new Random(); Console.Write(
shared.CreateStateOutput() ); Producer producer = new
Producer( shared, random ); Consumer consumer = new Consumer(
shared, random ); Thread producerThread = new Thread( new
ThreadStart( producer.Produce ) ); producerThread.Name =

```

```
"Producer"; Thread consumerThread = new Thread( new  
ThreadStart( consumer.Consume ) ); consumerThread.Name =  
"Consumer"; producerThread.Start(); consumerThread.Start(); }  
} [/csharp]
```