Contents

USCSP301-USCS303: Operating System(OS) Practical-07

Practical-07: Synchronization

Practical Date: 25th Aug, 2021

Practical Aim: Bounded Buffer Problem, Readers – Writers Problem, Sleeping Barber Problem

Practical – 07: Synchronization (Bounded-Buffer, Reader-Writer's, Sleeping Barber Problem)

- Content:
  - To minimize the amount of waiting time for threads that share resources and operate at the same average speeds, we implement a bounded buffer.
  - To avoid Starvation and deadlock situation, we use the concept of synchronization or locks.
- Process:
  - One can also determine whether a process's request for allocation of resources be safely granted immediately.
- Prior Knowledge:
  - Concept of Synchronization, Lock, Semaphore.

Bounded Buffer Problem

- The **producer-consumer problem**, also known as **Bounded Buffer Problem**, illustrates the need for synchronization in system where many processes share a resource.
- In the problem, two processes **share a fixed-size buffer.**
- One process produce information and puts it in the buffer, while the other process consumes information from the buffer.
- These processes do not take turns accessing the buffer, they both work concurrently. Herein lies the problem.

- What happens if the producer tries to put an item into a full buffer?
- What happens if the consumer tries to take an item from an empty buffer?
    - **In order to synchronize these processes, we will block the producer when the buffer is full, and we will block the consumer when the buffer is empty.**

So the two processes, Producer and Consumer, should work as follows:

i. The Producer must first create a new widget.
ii. Then, it checks to see if the buffer is full. If it is, the producer will put itself to sleep until the consumer wakes it up. A "wakeup" will come if the consumer finds the buffer empty.
iii. Next, the producer puts the new widget in the buffer. If the prosucer goes to sleep in step (ii), it will not wake up until the buffer is empty, so the buffer will never overflow..
iv. Then, the producer checks to see if the buffer is empty. If it is, the producer assumes that the consumer is sleeping, and so it will wake the consumer. Keep in mind that between any of these steps, an interrupt might occur, allowing the consumer to run.

Producer:

1. **Make new widget** $\longrightarrow$ W
2. **If buffer is full, go to sleep**   W W W = **Producer**
3. **Put widget in buffer**   ↓ W
4. **If buffer was empty, wake consumer**

Wake up!

Producer ↙ **Consumer**

So the two processes, Producer and Consumer, should work as follows:

1. The consumer checks to see if the buffer is empty. If so, the consumer will put itself to sleep until the producer wakes it up. A “wakeup” will occur if the producer finds the buffer empty after it puts an item into the buffer.
2. Then, the consumer will remove a widget from the buffer. The consumer will never try to remove a widget from an empty buffer because it will not wake up until the buffer is full.
3. If the buffer was full before it removed the widget, the consumer will wake the producer.
4. Finally , the consumer will consume the widget. As was the case with the producer, an interrupt could occur between any of these steps, allowing the producer to run.

Consumer:

1. **If buffer is empty, go to sleep =Consumer**
2. **Take widget from bffer**

   W W $\longrightarrow$ W

3. **If buffer was full, wake producer**

   Wake up!

   Consumer Producer

4. **Consume the widget < W**

Question-01:

Write a java program for Bounded Buffer Problem using synchronization.

Source Code 1:

```
//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 25th  Aug, 2021

//PRAC-07: SNYCHRONIZATION


public interface P7_Q1_Buffer_SS

{

  public void set(int value) throws InterruptedException;

  public int get() throws InterruptedException;

}
```

Source Code 2:

```
//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 25th  Aug, 2021

//PRAC-07: SNYCHRONIZATION


public class P7_Q1_CircularBuffer_SS implements

P7_Q1_Buffer_SS

{
  private final int[] buffer = {-1, -1, -1}; //shared buffer

  private int occupiedCells = 0; //count number of buffers used

  private int writeIndex = 0; //index of next element to write tp

  private int readIndex = 0; //index of next element to read

public synchronized void set(int value) throws InterruptedException

{

 while(occupiedCells == buffer.length)

 {

   System.out.println("Buffer is full. Producer waits.");

   wait();

  }

   buffer[writeIndex] = value;

   writeIndex = (writeIndex + 1) % buffer.length;

   ++occupiedCells;

   displayState("Producer write" +value);

   notifyAll();

}//set() ends

public synchronized int get() throws InterruptedException

 {

  while(occupiedCells == 0)

  {

     System.out.println("Buffer is empty. Consumer waits.");

     wait();

  }
```

```
    int readValue = buffer[readIndex];

    readIndex = (readIndex + 1) % buffer.length;

    --occupiedCells;

    displayState("Consumer reads" + readValue);

    notifyAll();

    return readValue;

}//get() ends

public void displayState(String operation)

{

  System.out.printf("%s%s%d)\n%S", operation,"(buffer cells occupied:", occupiedCells, "buffer
cells:");

  for(int value: buffer)

  System.out.printf(" %2d ", value);

  System.out.print("\n   ");

  for(int i=0; i<buffer.length; i++)

  System.out.print(" ---- ");

  System.out.print("\n  ");

  for(int i=0; i<buffer.length; i++)

 {

  if(i==writeIndex && i==readIndex)

   System.out.print("  WR");

  else if(i==writeIndex)

    System.out.print("  W");

 else if(i==readIndex)
```

```
    System.out.print("  R");
 else
   System.out.print("  ");
}
  System.out.println("\n");
}//displayState() ends
}//CircularBuffer class end
```

Source Code 3:

```
//NAME: SHRADDHA SAWANT
//BATCH: B1
//PRN: 2020016400773862
//DATE: 25th  Aug, 2021
//PRAC-07: SNYCHRONIZATION

import java.util.Random;
public class P7_Q1_Producer_SS implements Runnable
{
  private final static Random generator = new Random();
  private final P7_Q1_Buffer_SS sharedLocation;
  public P7_Q1_Producer_SS(P7_Q1_Buffer_SS shared)
  {
   sharedLocation = shared;
   }
public void run()
{
  for(int count = 1; count<= 10; count++)
  {
   try{
         Thread.sleep(generator.nextInt(3000));
          sharedLocation.set(count);
    }catch(InterruptedException e){
       e.printStackTrace();
    }
}
System.out.println("Producer done producing. Terminating Producer");
}//run() ends
}//Producer class ends
```

Source Code 4:

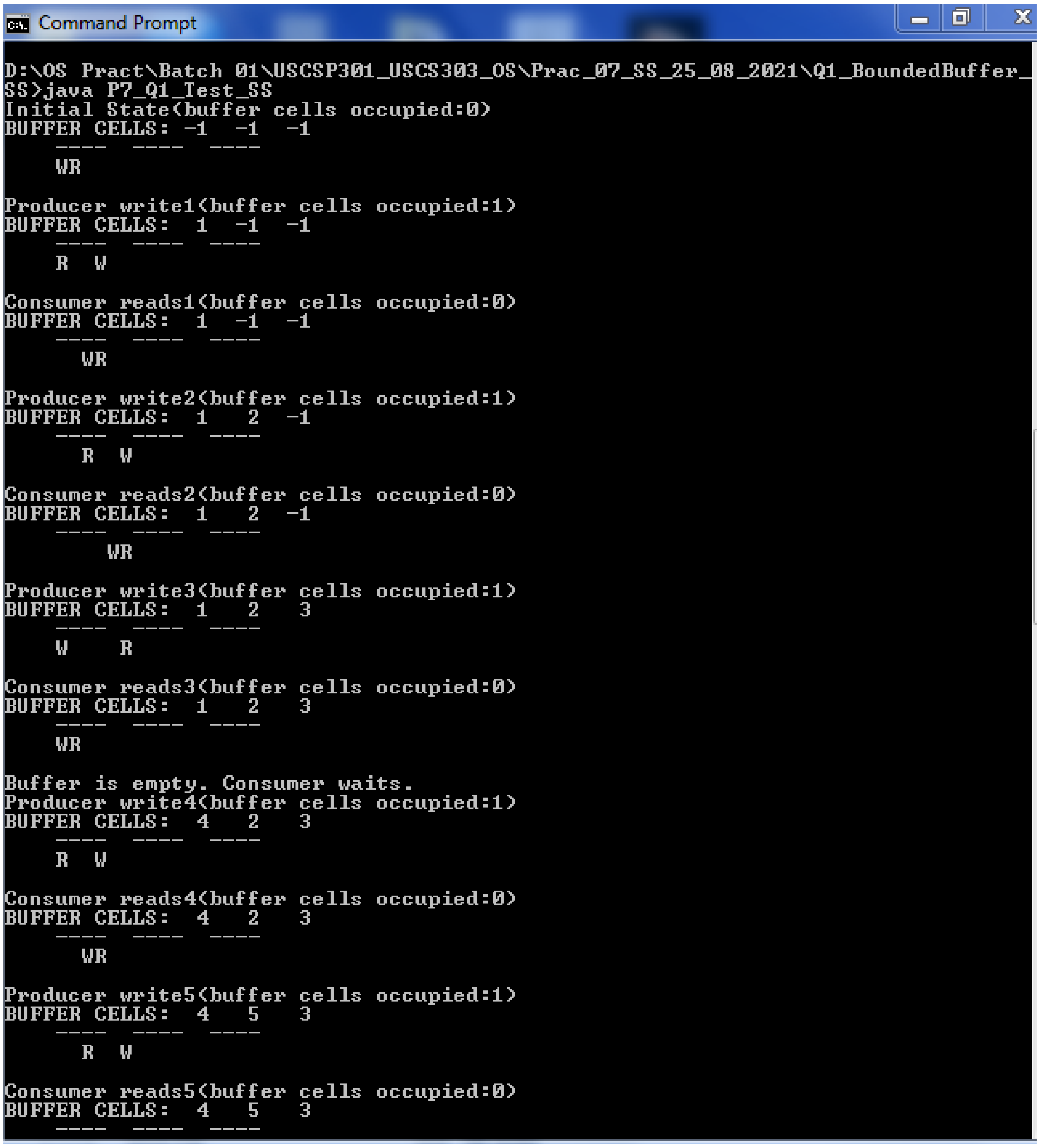
```
//NAME: SHRADDHA SAWANT
//BATCH: B1
//PRN: 2020016400773862
//DATE: 25th  Aug, 2021
//PRAC-07: SNYCHRONIZATION


import java.util.Random;
public class P7_Q1_Consumer_SS implements Runnable
{
  private final static Random generator = new Random();
```

```
 private final P7_Q1_Buffer_SS sharedLocation;

 public P7_Q1_Consumer_SS(P7_Q1_Buffer_SS shared)

 {

   sharedLocation = shared;

 }

public void run()

{

 int sum = 0;

 for(int count =1; count <=10; count++)

 {

  try{

       Thread.sleep(generator.nextInt(3000));

        sum += sharedLocation.get();

   }catch(InterruptedException e){

    e.printStackTrace();

 }

}

System.out.printf("\n%s %d\n%s\n", "Consumer read values totalling", sum, "Terminating
Consumer");

}//run() ends

}// Consumer class ends
```

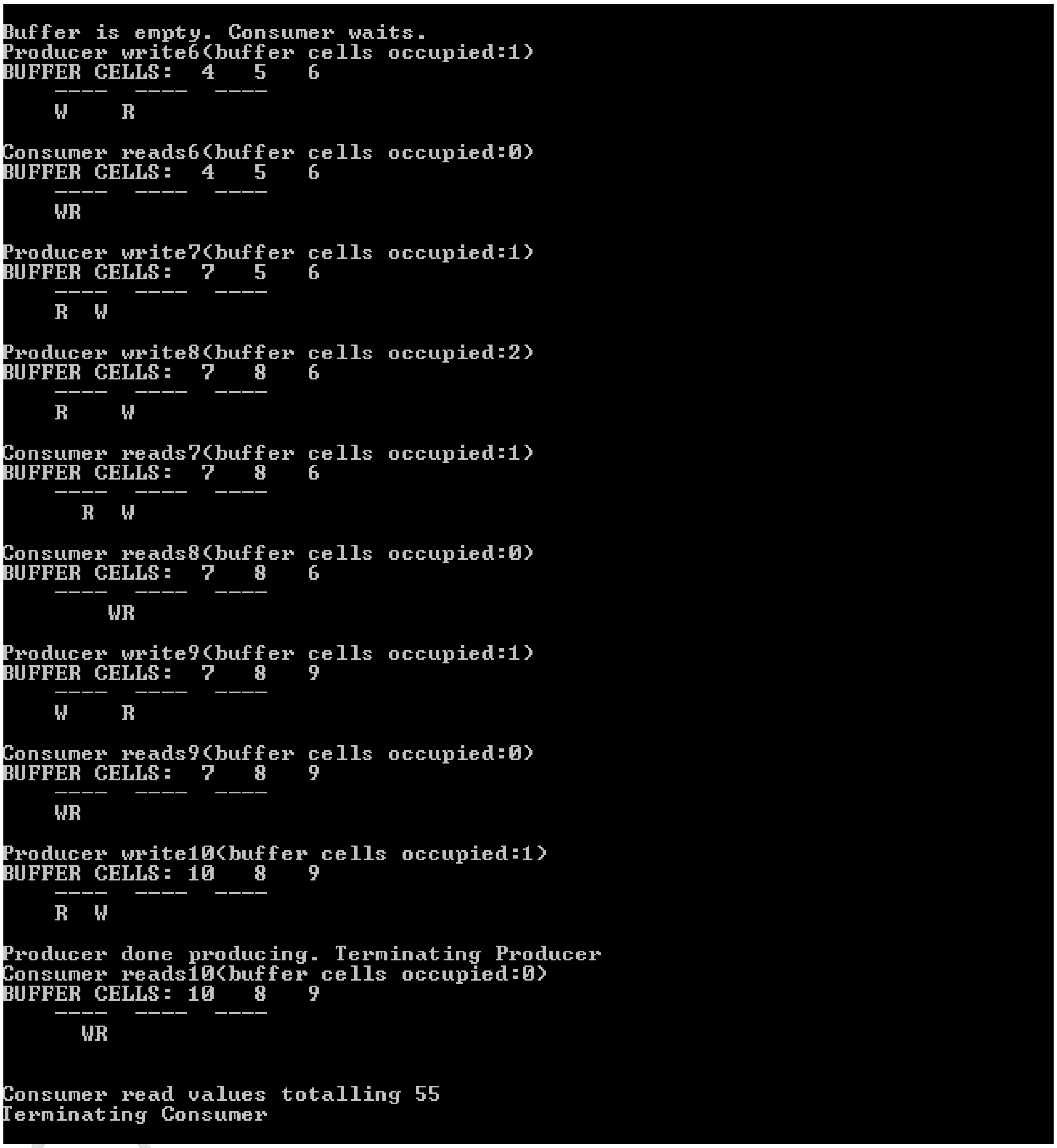
Source Code 5:

```
//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 25th  Aug, 2021

//PRAC-07: SNYCHRONIZATION


import java.util.concurrent.*;

public class P7_Q1_Test_SS

{

  public static void main(String args[])

 {

  ExecutorService application = Executors.newCachedThreadPool();

  P7_Q1_CircularBuffer_SS sharedLocation = new P7_Q1_CircularBuffer_SS();

  sharedLocation.displayState("Initial State");

  application.execute(new P7_Q1_Producer_SS(sharedLocation));

  application.execute(new P7_Q1_Consumer_SS(sharedLocation));

  application.shutdown();

}

}
```

Output:



```
D:\OS Pract\Batch 01\USCSP301_USCS303_OS\Prac_07_SS_25_08_2021\Q1_BoundedBuffer_
SS>java P7_Q1_Test_SS
Initial State(buffer cells occupied:0)
BUFFER CELLS: -1  -1  -1
    ----  ----  ----
    WR

Producer write1(buffer cells occupied:1)
BUFFER CELLS:  1  -1  -1
    ----  ----  ----
    R  W

Consumer reads1(buffer cells occupied:0)
BUFFER CELLS:  1  -1  -1
    ----  ----  ----
      WR

Producer write2(buffer cells occupied:1)
BUFFER CELLS:  1   2  -1
    ----  ----  ----
      R  W

Consumer reads2(buffer cells occupied:0)
BUFFER CELLS:  1   2  -1
    ----  ----  ----
        WR

Producer write3(buffer cells occupied:1)
BUFFER CELLS:  1   2   3
    ----  ----  ----
    W    R

Consumer reads3(buffer cells occupied:0)
BUFFER CELLS:  1   2   3
    ----  ----  ----
    WR

Buffer is empty. Consumer waits.
Producer write4(buffer cells occupied:1)
BUFFER CELLS:  4   2   3
    ----  ----  ----
    R  W

Consumer reads4(buffer cells occupied:0)
BUFFER CELLS:  4   2   3
    ----  ----  ----
      WR

Producer write5(buffer cells occupied:1)
BUFFER CELLS:  4   5   3
    ----  ----  ----
      R  W

Consumer reads5(buffer cells occupied:0)
BUFFER CELLS:  4   5   3
    ----  ----  ----
```



```
Buffer is empty. Consumer waits.
Producer write6(buffer cells occupied:1)
BUFFER CELLS:  4   5   6
    ----  ----  ----
    W    R

Consumer reads6(buffer cells occupied:0)
BUFFER CELLS:  4   5   6
    ----  ----  ----
    WR

Producer write7(buffer cells occupied:1)
BUFFER CELLS:  7   5   6
    ----  ----  ----
    R  W

Producer write8(buffer cells occupied:2)
BUFFER CELLS:  7   8   6
    ----  ----  ----
    R    W

Consumer reads7(buffer cells occupied:1)
BUFFER CELLS:  7   8   6
    ----  ----  ----
      R  W

Consumer reads8(buffer cells occupied:0)
BUFFER CELLS:  7   8   6
    ----  ----  ----
        WR

Producer write9(buffer cells occupied:1)
BUFFER CELLS:  7   8   9
    ----  ----  ----
    W    R

Consumer reads9(buffer cells occupied:0)
BUFFER CELLS:  7   8   9
    ----  ----  ----
    WR

Producer write10(buffer cells occupied:1)
BUFFER CELLS: 10   8   9
    ----  ----  ----
    R  W

Producer done producing. Terminating Producer
Consumer reads10(buffer cells occupied:0)
BUFFER CELLS: 10   8   9
    ----  ----  ----
      WR


Consumer read values totalling 55
Terminating Consumer
```

Reader – Writers Problem

- In computer science, the reader-writers problems are examples of a common computing problem in concurrency.
- Here many threads (small processes which share data) try to access the same shared resource at one time.
- Some threads may read and some may write, with the constaraint that no process may access the shared resource for either reading or writing while another process is n the act of writing to it.
- (In particular, we want to prevent more than one thread modify the shared resource simultaneously and allowed for two or more readers to access the shared resource at the same time).
- A reader-writer lock is a data structure that solves one or more of the readers-writers problems.

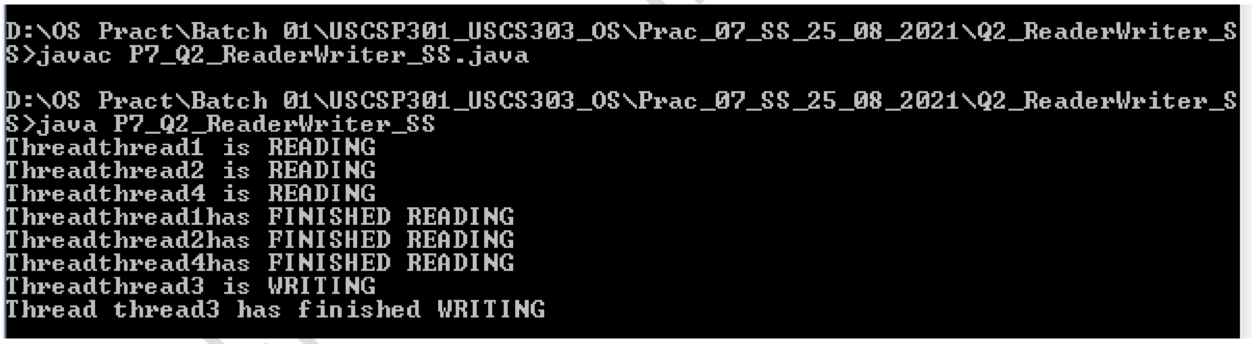
Question-02:

Write a java program for Readers Writers Problem using semaphore.

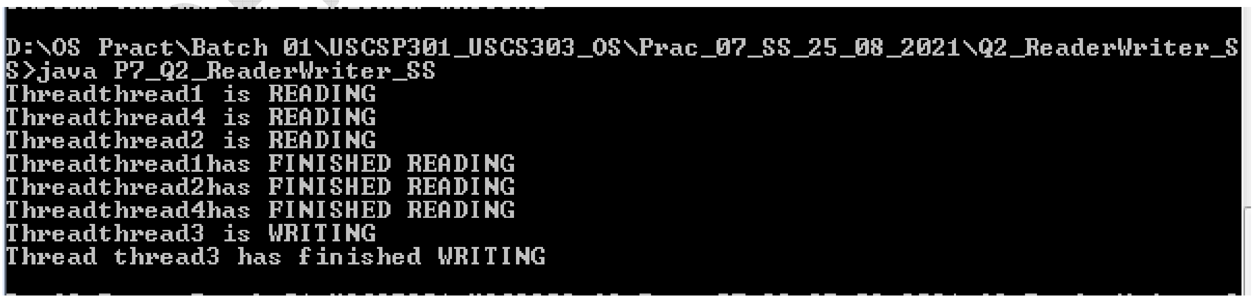
Source code 01:

```
//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 25th  Aug, 2021

//PRAC-07: SNYCHRONIZATION


import java.util.concurrent.Semaphore;

class P7_Q2_ReaderWriter_SS

{

  static Semaphore readLock = new Semaphore(1, true);

  static Semaphore writeLock = new Semaphore(1, true);

  static int readCount = 0;
static class Read implements Runnable{
 @Override
 public void run(){
  try{
  //Acquire Section
  readLock.acquire();
  readCount++;
  if(readCount==1){
   writeLock.acquire();
 }
  readLock.release();
  //Reading section
  System.out.println("Thread" +Thread.currentThread().getName()+" is READING");
  Thread.sleep(1500);
  System.out.println("Thread"+Thread.currentThread().getName()+ "has FINISHED READING");
//Releasing section
  readLock.acquire();
  readCount--;
  if(readCount==0){
   writeLock.release();
 }
 readLock.release();
}//try ends
catch(InterruptedException e){
 System.out.println(e.getMessage());
 }
}//run()ends
}//static class Read ends
static class Write implements Runnable{
 @Override
 public void run(){
 try{
  writeLock.acquire();
  System.out.println("Thread" +Thread.currentThread().getName()+" is WRITING");
  Thread.sleep(2500);
   System.out.println("Thread "+Thread.currentThread().getName() +" has finished WRITING");
  writeLock.release();
}catch(InterruptedException e){
  System.out.println(e.getMessage());
 }
}//run() ends
}//class Write ends
public static void main(String[] args)throws Exception{
 Read read = new Read();
 Write write = new Write();
 Thread t1 = new Thread(read);
 t1.setName("thread1");
 Thread t2 = new Thread(read);
 t2.setName("thread2");

 Thread t3 = new Thread(write);

 t3.setName("thread3");

 Thread t4 = new Thread(read);

 t4.setName("thread4");

 t1.start();

 t3.start();

 t2.start();

 t4.start();

 }//main ends

}//class P7_Q2_ReaderWriter_SS ends
```

Output:



```
D:\OS Pract\Batch 01\USCSP301_USCS303_OS\Prac_07_SS_25_08_2021\Q2_ReaderWriter_S
S>javac P7_Q2_ReaderWriter_SS.java

D:\OS Pract\Batch 01\USCSP301_USCS303_OS\Prac_07_SS_25_08_2021\Q2_ReaderWriter_S
S>java P7_Q2_ReaderWriter_SS
Threadthread1 is READING
Threadthread2 is READING
Threadthread4 is READING
Threadthread1has FINISHED READING
Threadthread2has FINISHED READING
Threadthread4has FINISHED READING
Threadthread3 is WRITING
Thread thread3 has finished WRITING
```



```
D:\OS Pract\Batch 01\USCSP301_USCS303_OS\Prac_07_SS_25_08_2021\Q2_ReaderWriter_S
S>java P7_Q2_ReaderWriter_SS
Threadthread1 is READING
Threadthread4 is READING
Threadthread2 is READING
Threadthread1has FINISHED READING
Threadthread2has FINISHED READING
Threadthread4has FINISHED READING
Threadthread3 is WRITING
Thread thread3 has finished WRITING
```

Sleeping-Barber Problem

- A barber shop consists of awaiting room with n chairs and a barber room with one barber chair.
- If there are no costumers to be served, the barber goes to sleep.
- If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop.
- If the barber is busy but chairs are available, then the customer sits in one of the chairs. If the barber is asleep, the customer wakes up the barber.

Question-03:

Write a program to coordinate the barber and the customers using java synchronization.

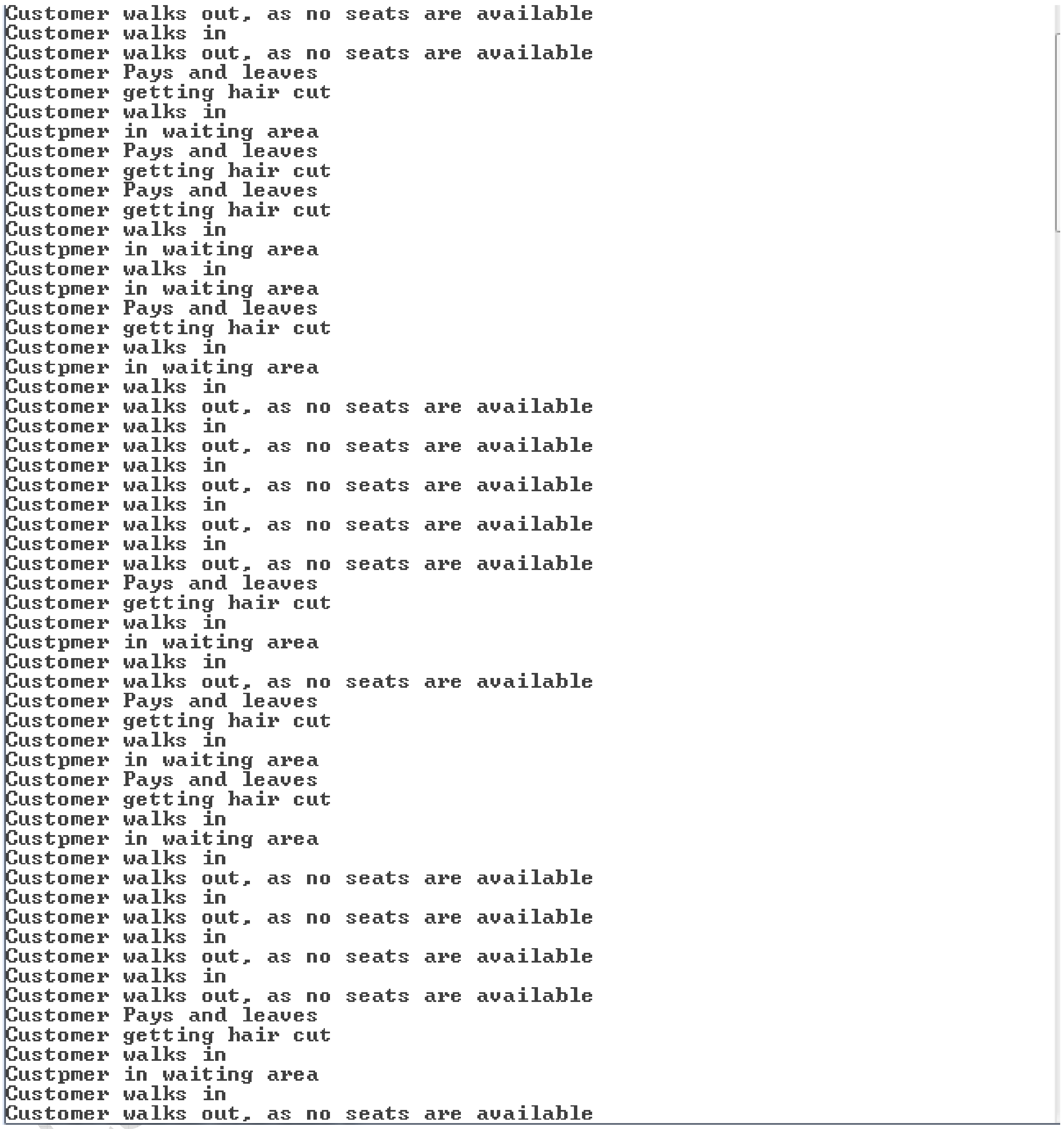
Source Code 1:

```
//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 25th  Aug, 2021

//PRAC-07: SNYCHRONIZATION


import java.util.concurrent.*;

import java.util.concurrent.atomic.AtomicInteger;

import java.util.Random;


public class P7_Q3_Barber_SS implements Runnable

{

 private AtomicInteger spaces;

 private Semaphore bavailable;

 private Semaphore cavailable;

 private Random ran = new Random();
 public P7_Q3_Barber_SS (AtomicInteger spaces, Semaphore bavailable, Semaphore
cavailable){
 this.spaces=spaces;
 this.bavailable=bavailable;
 this.cavailable=cavailable;
}
@Override
public void run(){
 while(true){
  try{
        cavailable.acquire();
        //Space freed up in waiting area
        System.out.println("Customer getting hair cut");
    Thread.sleep(ThreadLocalRandom.current().nextInt(1000, 10000+1000));
    //Sleep to imitate length of time to cut hair
    System.out.println("Customer Pays and leaves");
    bavailable.release();
  }catch(InterruptedException e){}
 }//while ends
}//run() ends
}//class ends
```

Source Code 2:

```
//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 25th  Aug, 2021

//PRAC-07: SNYCHRONIZATION


import java.util.concurrent.atomic.AtomicInteger;

import java.util.concurrent.*;


class P7_Q3_BarberShop_SS{

 public static void main(String[] args)

 {

  AtomicInteger spaces = new AtomicInteger(15);

  final Semaphore barbers = new Semaphore(3,true);

  final Semaphore customers = new Semaphore(0, true);

  ExecutorService openUp = Executors.newFixedThreadPool(3);

  P7_Q3_Barber_SS[] employees = new P7_Q3_Barber_SS[3];

  System.out.println("Opening up shop");

  for(int i =0; i<3; i++){

   employees[i] = new P7_Q3_Barber_SS(spaces, barbers, customers);

   openUp.execute(employees[i]);

 }

 while(true)
{
 try{
      Thread.sleep(ThreadLocalRandom.current().nextInt(100, 1000+100)); //sleep until next
person gets in
 }
  catch(InterruptedException e) {}
  System.out.println("Customer walks in");
  if(spaces.get()>= 0){
   new Thread(new P7_Q3_Customer_SS(spaces, barbers, customers)).start();
}
else{
      System.out.println("Customer walks out, as no seats are available");
 }
}//while ends
}//main ends
}//P7_Q3_BarberShop_SS class ends
```

Output:



```
Customer walks out, as no seats are available
Customer walks in
Customer walks out, as no seats are available
Customer Pays and leaves
Customer getting hair cut
Customer walks in
Custpmer in waiting area
Customer Pays and leaves
Customer getting hair cut
Customer Pays and leaves
Customer getting hair cut
Customer walks in
Custpmer in waiting area
Customer walks in
Custpmer in waiting area
Customer Pays and leaves
Customer getting hair cut
Customer walks in
Custpmer in waiting area
Customer walks in
Customer walks out, as no seats are available
Customer walks in
Customer walks out, as no seats are available
Customer walks in
Customer walks out, as no seats are available
Customer walks in
Customer walks out, as no seats are available
Customer walks in
Customer walks out, as no seats are available
Customer Pays and leaves
Customer getting hair cut
Customer walks in
Custpmer in waiting area
Customer walks in
Customer walks out, as no seats are available
Customer Pays and leaves
Customer getting hair cut
Customer walks in
Custpmer in waiting area
Customer Pays and leaves
Customer getting hair cut
Customer walks in
Custpmer in waiting area
Customer walks in
Customer walks out, as no seats are available
Customer walks in
Customer walks out, as no seats are available
Customer walks in
Customer walks out, as no seats are available
Customer walks in
Customer walks out, as no seats are available
Customer Pays and leaves
Customer getting hair cut
Customer walks in
Custpmer in waiting area
Customer walks in
Customer walks out, as no seats are available
```