

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

## Contents

USCSP301_USCS303_Operating System (OS) Practical-07 .....	2
Practical 07: Synchronization .....	2
Practical Date: 25 August 2021 .....	2
Practical Aim: Bounded-Buffer Problem, Readers-Writers, Sleeping Barber Problem .....	2
Synchronization .....	2
Bounded Buffer Problem .....	2
Reader's Writers Problem .....	10
Sleeping Barber Problem .....	14

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

## USCSP301\_USCS303\_Operating System (OS) Practical-07

### Practical 07: Synchronization

**Practical Date: 25 August 2021**

**Practical Aim: Bounded-Buffer Problem, Readers-Writers, Sleeping Barber Problem**

#### Synchronization

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.

One can also determine whether a process's request for allocation of resources be safely granted immediately.

#### Bounded Buffer Problem

**The producer-consumer problem, also known as Bounded Buffer Problem, illustrates the need for synchronization in systems where many processes share a resource.**

In the problem, two processes **share a fixed-size buffer**.

One process produces 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. Here lies the problem.

**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.**

#### QUESTION 1:

Write a java program for Bounded Buffer Problem using synchronization.

//Name:Ritika Sahu

//Batch : B1

//PRN:2020016400783543

//Date: 25 August, 2021.

NAME: RITIKA SAHU

BATCH: B1

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

//Practical 7: Synchronization

```
public interface P7_Q1_Buffer_RS
{
    public void set(int value) throws InterruptedException;

    public int get() throws InterruptedException;
}
```

//Name:Ritika Sahu

//Batch : B1

//PRN:2020016400783543

//Date: 25 August, 2021.

//Practical 7: Synchronization

```
public class P7_Q1_CircularBuffer_RS implements P7_Q1_Buffer_RS
{
    private final int[] buffer = {-1,-1,-1}; // sharedBuffer

    private int occupiedCells = 0; // count number of buffers used
    private int writeIndex = 0; // index of next element to write to
    public synchronized void set(int value) throws InterruptedException
    {
        while(occupiedCells == buffer.length)
        {
```

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

```
        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)
{
```

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

```
System.out.println("%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
```

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

```
} // CircularBuffer class ends
```

```
//Name:Ritika Sahu
```

```
//Batch : B1
```

```
//PRN:2020016400783543
```

```
//Date: 25 August, 2021.
```

```
//Practical 7: Synchronization
```

```
import java.util.Random;
```

```
public class P7_Q1_Producer_RS
```

```
{
```

```
    private final static Random generator = new Random();
```

```
    private final P7_Q1_Buffer_RS sharedLocation;
```

```
    public P7_Q1_Producer_RS(P7_Q1_Buffer_RS shared)
```

```
    {
```

```
        sharedLocation = shared;
```

```
    }
```

```
    public void run()
```

```
    {
```

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

```
        {
```

```
            try {
```

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

```
                sharedLocation.set(count);
```

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

```
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
  
    System.out.println("Producer done producing. Terminating Producer");  
  
} // run() ends  
  
} // Producer class ends
```

//Name: Ritika Sahu

//Batch : B1

//PRN: 2020016400783543

//Date: 25 August, 2021.

//Practical 7: Synchronization

```
import java.util.Random;  
  
public class P7_Q1_Consumer_RS implements Runnable  
{  
    private final static Random generator = new Random();  
    private final P7_Q1_Buffer_RS sharedLocation;  
    public P7_Q1_Consumer_RS(P7_Q1_Buffer_RS shared)  
    {  
        sharedLocation = shared;  
    }  
    public void run()  
    {
```

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

```
int sum =0;

for(int count = 1;count <=0;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 totaling",
sum, "Terminating Consumer");

} // run() ends

} // Consumer class ends

//Name:Ritika Sahu

//Batch : B1

//PRN:2020016400783543

//Date: 25 August, 2021.

//Practical 7: Synchronization

import java.util.concurrent.*;

public class P7_Q1_Test_RS

{

    public static void main(String[] args)

    {
```



# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

```
ExecutorService application = executors.newCachedThreadPool();

P7_Q1_CircularBuffer_RS sharedLocation = new P7_Q1_CircularBuffer_RS();

sharedLocation.displayState("Initial State");

application.execute(new P7_Q1_Producer_RS(sharedLocation));

application.execute(new P7_Q1_Consumer_RS(sharedLocation));

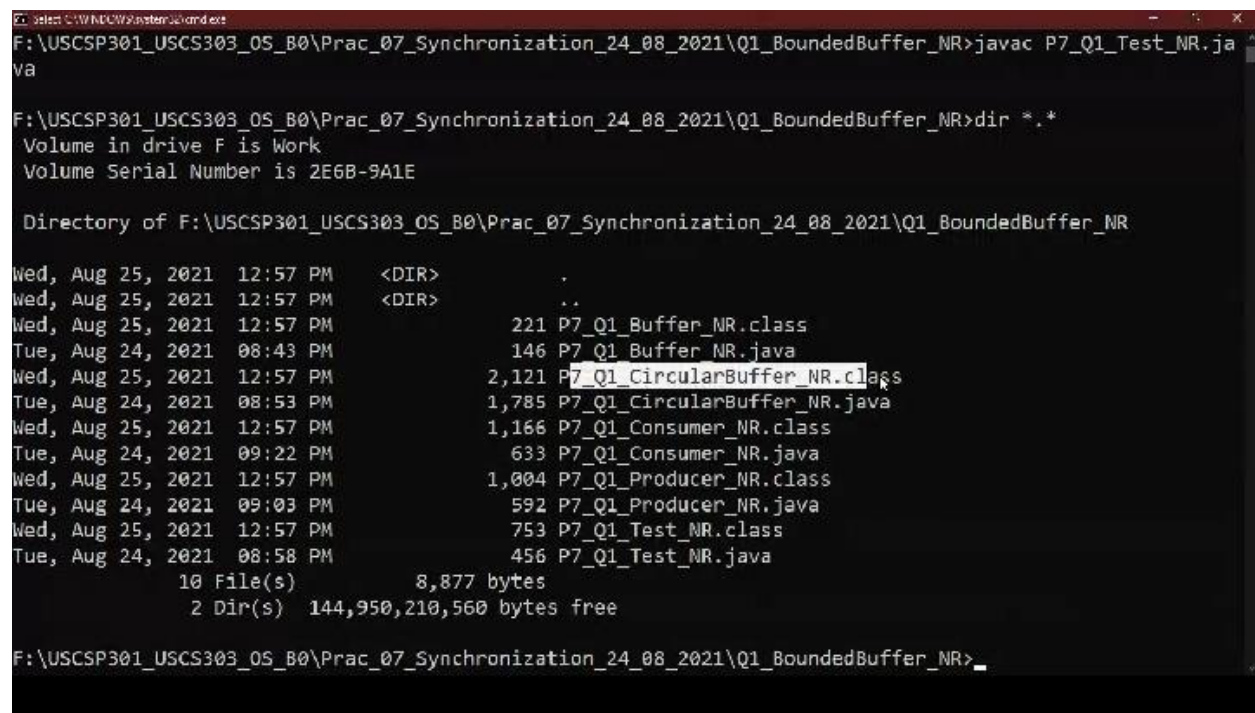
application.shutdown();

}

}

S
```

## OUTPUT:



```
C:\Users\ADMIN\Documents>cd F:\USCSP301_USCS303_OS_B0\Prac_07_Synchronization_24_08_2021\Q1_BoundedBuffer_NR
F:\USCSP301_USCS303_OS_B0\Prac_07_Synchronization_24_08_2021\Q1_BoundedBuffer_NR>javac P7_Q1_Test_NR.java
F:\USCSP301_USCS303_OS_B0\Prac_07_Synchronization_24_08_2021\Q1_BoundedBuffer_NR>dir *.*
Volume in drive F is Work
Volume Serial Number is 2E6B-9A1E

Directory of F:\USCSP301_USCS303_OS_B0\Prac_07_Synchronization_24_08_2021\Q1_BoundedBuffer_NR

Wed, Aug 25, 2021 12:57 PM <DIR>      .
Wed, Aug 25, 2021 12:57 PM <DIR>      ..
Wed, Aug 25, 2021 12:57 PM          221 P7_Q1_Buffer_NR.class
Tue, Aug 24, 2021 08:43 PM          146 P7_Q1_Buffer_NR.java
Wed, Aug 25, 2021 12:57 PM      2,121 P7_Q1_CircularBuffer_NR.class
Tue, Aug 24, 2021 08:53 PM      1,785 P7_Q1_CircularBuffer_NR.java
Wed, Aug 25, 2021 12:57 PM      1,166 P7_Q1_Consumer_NR.class
Tue, Aug 24, 2021 09:22 PM          633 P7_Q1_Consumer_NR.java
Wed, Aug 25, 2021 12:57 PM      1,004 P7_Q1_Producer_NR.class
Tue, Aug 24, 2021 09:03 PM          592 P7_Q1_Producer_NR.java
Wed, Aug 25, 2021 12:57 PM          753 P7_Q1_Test_NR.class
Tue, Aug 24, 2021 08:58 PM          456 P7_Q1_Test_NR.java
                10 File(s)          8,877 bytes
                2 Dir(s) 144,950,210,560 bytes free

F:\USCSP301_USCS303_OS_B0\Prac_07_Synchronization_24_08_2021\Q1_BoundedBuffer_NR>
```

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

```
WR
Buffer is empty. Consumer waits.
Producer write 4 (buffer cells occupied: 1)
buffer cells:   4       2       3
               ----
               R       W

Consumer reads 4 (buffer cells occupied: 0)
buffer cells:   4       2       3
               ----
               WR

Buffer is empty. Consumer waits.
Producer write 5 (buffer cells occupied: 1)
buffer cells:   4       5       3
               ----
               R       W

Consumer reads 5 (buffer cells occupied: 0)
buffer cells:   4       5       3
               ----
               WR
```

```
buffer cells:   7       5       6
               ----
               R       W

Consumer reads 7 (buffer cells occupied: 0)
buffer cells:   7       5       6
               ----
               WR

Producer write 8 (buffer cells occupied: 1)
buffer cells:   7       8       6
               ----
               R       W

Consumer reads 8 (buffer cells occupied: 0)
buffer cells:   7       8       6
               ----
               WR

Producer write 9 (buffer cells occupied: 1)
buffer cells:   7       8       9
               ----
               W       R
```

## Reader's Writers Problem

- (1) In computer science, the **readers-writers problems** are example of a common computing problem in concurrency.
- (2) Here many threads (small processes which share data) try to access the same thread resource at one time.

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

- (3) Some threads may read and some may write, with the constraint that no process may access the shared resource for either reading or writing while another process is in the act of writing to it.
- (4) (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).
- (5) A readers-writer lock is a data structure that solves one or more of the readers-writers problems.

## QUESTION 2:

Write a java program for Readers Writers Problem using semaphore.

//Name:Ritika Sahu

//Batch : B1

//PRN:2020016400783543

//Date: 25 August, 2021.

//Practical 7: Synchronization

```
import java.util.concurrent.Semaphore;
```

```
class P7_Q2_ReaderWriter_RS
```

```
{
```

```
    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() {
```

NAME: RITIKA SAHU

BATCH: B1

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

```
try {  
    // Acquire Section  
    readLock.acquire();  
    readCount++;  
    if (readCount == 1) {  
        writeLock.acquire();  
    }  
    readLock.release();  
  
    // Reading section  
    System.out.print("Thread "+Thread.currentThread().getName() + " is READING");  
    Thread.sleep(1500);  
    System.out.print("Thread "+Thread.currentThread().getName() + " has FINISHED  
READING");  
    //Releasing section  
    readLock.acquire();  
    readCount --;  
    if(readCount == 0) {  
        writeLock.release();  
    } // try ends  
    catch(InterruptedException e) {  
        System.out.println(e.getMessage());  
    }  
} // run() ends  
}
```

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

```
// static class Read ends

static class Write implements Runnable {

    @Override

    public void run() {

        try {

            writeLock.acquire();

            System.out.print("Thread "+Thread.currentThread().getName() + " is WRITING");

            Thread.sleep(2500);

            System.out.print("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(read);
```

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

```
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_RS ends
```

## OUTPUT:

```
Thread thread4 is READING  
Thread thread2 is READING  
Thread thread1 is READING  
Thread thread4 has FINISHED READING  
Thread thread1 has FINISHED READING  
Thread thread2 has FINISHED READING  
Thread thread3 is WRITING  
Thread thread3 has finished WRITING
```

```
Thread thread1 is READING  
Thread thread4 is READING  
Thread thread2 is READING  
Thread thread2 has FINISHED READING  
Thread thread4 has FINISHED READING  
Thread thread1 has FINISHED READING  
Thread thread3 is WRITING  
Thread thread3 has finished WRITING
```

NOTE: The output keeps on changing based on the run due to random calls made to Threads.

## Sleeping Barber Problem

- (1) A barber shop consist of awaiting room with n chairs and a barber room with one barber chain.

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

- (2) If there are no customers to be served, the barber goes to sleep.
- (3) If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop.
- (4) If the barber is busy but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes u the barber.

//Name: Ritika Sahu

//Batch : B1

//PRN:2020016400783543

//Date: 25 August, 2021.

//Practical 7: Synchronization

```
import java.util.concurrent.*;
```

```
import java.util.concurrent.atomic.AtomicInteger;
```

```
import java.util.Random;
```

```
public class P7_Q3_Barber_RS implements Runnable
```

```
{
```

```
    private AtomicInteger spaces;
```

```
    private Semaphore bavailable;
```

```
    private Semaphore cavailable;
```

```
    private Random ran = new Random();
```

```
    public class P7_Q3_Barber_RS (AtomicInteger spaces, Semaphore bavailable, Semaphore  
cavailable) {
```

```
        this.spaces = spaces;
```

```
        this.bavailable = bavailable;
```

```
        this.cavailable = cavailable;
```

```
    }
```

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

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

//Name:Ritika Sahu

//Batch : B1

//PRN:2020016400783543

//Date: 25 August, 2021.

//Practical 7: Synchronization



# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

```
import java.util.concurrent.*;

import java.util.concurrent.atomic.AtomicInteger;

import java.util.Random;

public class P7_Q3_Customer_RS implements Runnable
{
    private AtomicInteger spaces;

    private Semaphore bavailable;

    private Semaphore cavailable;

    private Random ran = new Random();

    public class P7_Q3_Customer_RS (AtomicInteger spaces, Semaphore bavailable, Semaphore
cavailable) {

        this.spaces = spaces;

        this.bavailable = bavailable;

        this.cavailable = cavailable;

    }

    @Override

    public void run() {

        try {

            cavailable.release();

            if(bavailable.hasQueuedThreads()) {

                spaces.decrementAndGet();

                System.out.println("Customer in waiting area");

                bavailable.acquire();

            }

        }

    }

}
```

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

```
        spaces.incrementAndGet();
    }
    else
    {
        bavailable.acquire();
    }
} catch (InterruptedException e) {}
} // run() ends
} // P7_Q3_Customer_RS class
```

//Name:Ritika Sahu

//Batch : B1

//PRN:2020016400783543

//Date: 25 August, 2021.

//Practical 7: Synchronization

```
import java.util.concurrent.atomic.AtomicInteger;
```

```
import java.util.concurrent.*;
```

```
public class P7_Q3_BarberShop_RS {
    public static void main(String[] args)
    {
        AtomicInteger spaces = new AtomicInteger(15);
        final Semaphore barbers = new Semaphore (3, true);
```

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

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

ExecutorService openUp = exceution.newFixedThreadPool(3);

P7_Q3_Barber_RS[] employees = new P7_Q3_Barber_RS[3];

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

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

    employee[i] = new P7_Q3_Barber_RS(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_RS(spaces, barbers, customers)).start();

    }

    else {

        System.out.println("Customer walks out, as no seats are available");

    }

} // while ends

} // main ends
```

# SMT. CHANDIBAI HIMATHMAL MASUKHANI COLLEGE

```
} // P7_Q3_BarberShop_RS class ends
```

## OUTPUT:

```
Opening up shop
Customer walks in
Customer getting hair cut
Customer walks in
Customer getting hair cut
Customer walks in
Customer getting hair cut
Customer walks in
Customer walks in
Customer in waiting area
Customer walks in
Customer in waiting area
Customer walks in
Customer in waiting area
```

```
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 Pays and leaves
Customer getting hair cut
Customer walks in
Customer in waiting area
Customer walks in
Customer 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 Pays and leaves
Customer getting hair cut
Customer walks in
Customer 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
```