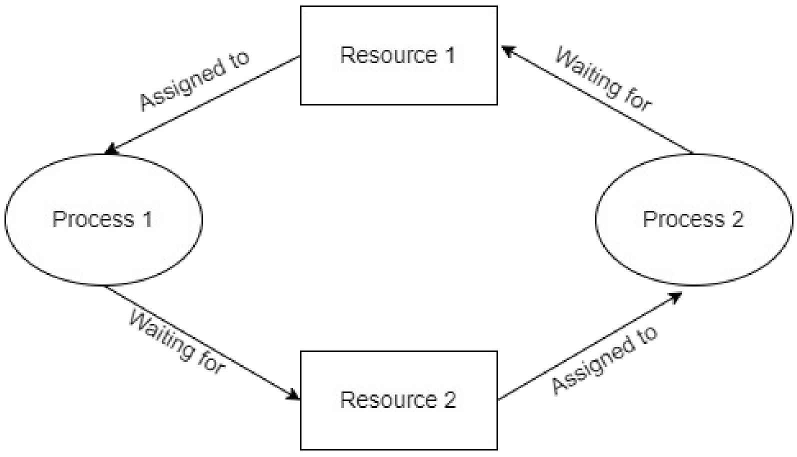
Distributed Computing

Experiment – 7

Program to demonstrate Deadlock Management in Distributed Systems	
Learning Objective:	To explore mutual exclusion algorithms and deadlock handling in the distributed system.
Learning Outcome:	Ability to demonstrate Deadlock Management in Distributed Systems
Course Outcome:	**CSL801.4**
Program Outcome:	(**PO**1) **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. (**PO**2) **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences. **(PO**3) **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations. **(PO5) Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
Bloom's Taxonomy Level:	Analysis, Create
Theory:	**Deadlock Management in Distributed Systems in details.**  Resource 1 Assigned to Waiting for Process 1 Process 2 Waiting for Assigned to Resource 2

A deadlock is a situation that arises in concurrent systems, including distributed systems, where two or more processes or threads are waiting indefinitely for one another to release resources that they need to proceed. This leads to a complete standstill, where none of the processes can make progress, resulting in a system that is essentially hung or frozen.

Types of Deadlocks

1. **Resource Deadlock**: This type of deadlock occurs when two or more processes are waiting for resources that are held by other processes, creating a circular wait condition.
2. **Communication Deadlock**: In distributed systems, processes often communicate with each other by sending and receiving messages. A communication deadlock can occur when processes are waiting for messages that will never arrive due to a cyclic dependency in the communication pattern.
3. **Deadlock by No Indefinite Delay**: This type of deadlock occurs when a process or thread is waiting for a resource that will never become available due to a programming error or a system failure.

Deadlock Management Procedure

The deadlock management procedure typically involves the following steps:

1. **Deadlock Detection**: The first step is to detect the presence of a deadlock in the system. This can be achieved through various algorithms, such as resource allocation graphs or wait-for graphs.
2. **Deadlock Resolution**: Once a deadlock is detected, the system must resolve the deadlock by breaking the circular wait condition. This can be done through several strategies, such as process termination, resource preemption, or rollback.
3. **Deadlock Prevention**: To prevent future deadlocks, the system can employ various techniques, such as careful resource allocation, avoidance of circular wait conditions, or enforcing ordering among resource acquisition.
4. **Deadlock Avoidance**: This involves dynamically monitoring resource allocation and denying requests that could potentially lead to a deadlock situation.

Types of Deadlock Management

1. **Deadlock Prevention**: This approach aims to prevent deadlocks from occurring in the first place by enforcing certain rules or protocols for resource allocation. Examples include the "no preemption" policy, the "hold and wait" policy, and the "circular wait" policy.
2. **Deadlock Avoidance**: In this approach, the system dynamically examines

	resource allocation requests and grants or denies them based on an analysis of whether granting the request could lead to a deadlock. The Banker's Algorithm is a well-known example of a deadlock avoidance strategy. 3. **Deadlock Detection and Resolution**: This approach allows deadlocks to occur but detects them as soon as possible and resolves them through techniques like process termination, resource preemption, or rollback. 4. **Deadlock Ignorance**: In some systems, deadlocks are considered so rare or their impact so minimal that the system simply ignores them. This approach is typically used in systems where deadlocks are not a significant concern or where the cost of prevention or detection is deemed too high.
Algorithm :	1. *Let Work and Finish be vectors of length m and n respectively. Initialize Work= Available. For i=0, 1, …., n-1, if Requesti = 0, then Finish[i] = true; otherwise, Finish[i]= false.* 2. *Find an index i such that both* *a) Finish[i] == false* *b) Requesti <= Work* *If no such i exists go to step 4.* 3. *Work= Work+ Allocationi* *Finish[i]= true* *Go to Step 2.* *4. If Finish[i]== false for some i, 0<=i<n, then the system is in a deadlocked state. Moreover, if Finish[i]==false the process Pi is deadlocked.*

Outcome :	**Program Code -**
	
	1. Example for Deadlock -
	
	`import java.util.concurrent.locks.Lock;`
	
	`import java.util.concurrent.locks.ReentrantLock;`
	
	`public class DeadlockExample {`
	
	`    private Lock lock1 = new ReentrantLock();`

```
    private Lock lock2 = new ReentrantLock();

    public static void main(String[] args) {
        DeadlockExample deadlock = new DeadlockExample();
        new Thread(deadlock::operation1, "T1").start();
        new Thread(deadlock::operation2, "T2").start();
    }

    public void operation1() {
        lock1.lock();
        System.out.println(Thread.currentThread().getName() + ": lock1 acquired, waiting to acquire lock2.");
        sleep(50);

        lock2.lock();
        System.out.println(Thread.currentThread().getName() + ": lock2 acquired");

        System.out.println(Thread.currentThread().getName() + ": executing first operation.");

        lock2.unlock();
        lock1.unlock();
    }

    public void operation2() {
        lock2.lock();

        System.out.println(Thread.currentThread().getName() + ": lock2 acquired, waiting to acquire lock1.");

        sleep(50);



        lock1.lock();

        System.out.println(Thread.currentThread().getName() + ": lock1 acquired");



        System.out.println(Thread.currentThread().getName() + ": executing second operation.");



        lock1.unlock();

        lock2.unlock();

    }



    // helper methods

    private void sleep(int milliseconds) {

        try {

            Thread.sleep(milliseconds);

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

    }

}
```

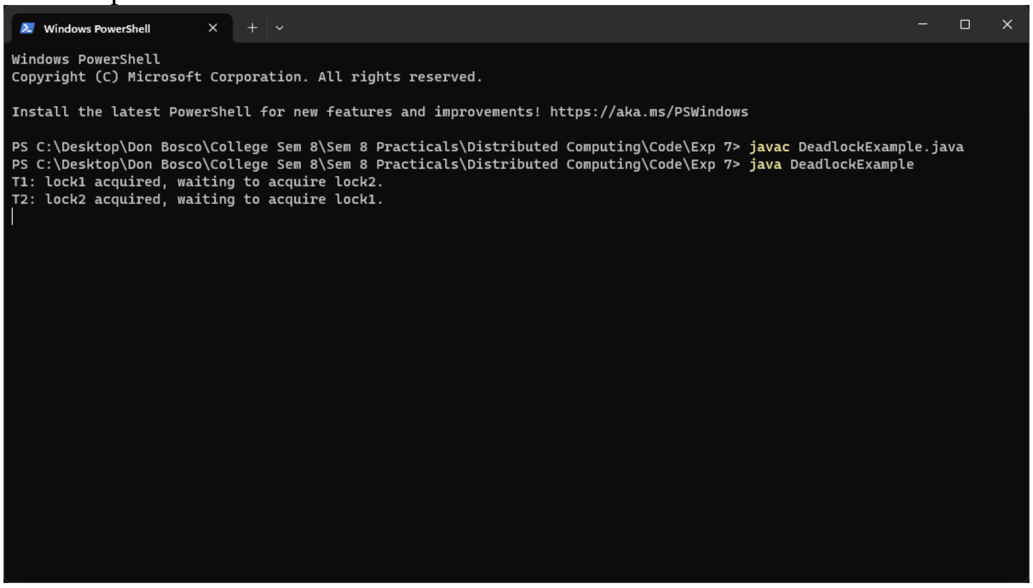
2. Example for Deadlock Resolution -

```
import java.util.Random;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;


public class LiveLockExample {


    private Lock lock1 = new ReentrantLock();

    private Lock lock2 = new ReentrantLock();


    public static void main(String[] args) {

        LiveLockExample livelock = new LiveLockExample();

        new Thread(livelock::operation1, "T1").start();

        new Thread(livelock::operation2, "T2").start();

    }


    public void operation1() {

        while (true) {

            tryLock(lock1, 10000); // Increase timeout to 10 seconds

            System.out.println(Thread.currentThread().getName() + ": lock1 acquired, trying to acquire lock2.");

            sleep(50);


            if (tryLock(lock2, 10000)) { // Increase timeout to 10 seconds
```

```
                System.out.println(Thread.currentThread().getName() + ": lock2 acquired.");

                break;

            } else {

                System.out.println(Thread.currentThread().getName() + ": cannot acquire lock2, releasing lock1.");

                lock1.unlock();

                randomDelay();

            }

        }

        System.out.println(Thread.currentThread().getName() + ": executing first operation.");

        lock2.unlock();

        lock1.unlock();

    }



    public void operation2() {

        while (true) {

            tryLock(lock2, 10000); // Increase timeout to 10 seconds

            System.out.println(Thread.currentThread().getName() + ": lock2 acquired, trying to acquire lock1.");

            sleep(50);



            if (tryLock(lock1, 10000)) { // Increase timeout to 10 seconds

                System.out.println(Thread.currentThread().getName() + ": lock1 acquired.");

                break;
            } else {
                System.out.println(Thread.currentThread().getName() + ": cannot acquire lock1, releasing lock2.");
                lock2.unlock();
                randomDelay();
            }
        }
        System.out.println(Thread.currentThread().getName() + ": executing second operation.");
        lock1.unlock();
        lock2.unlock();
    }

    // helper methods
    private boolean tryLock(Lock lock, long timeout) {
        try {
            return lock.tryLock(timeout, java.util.concurrent.TimeUnit.MILLISECONDS);
        } catch (InterruptedException e) {
            return false;
        }
    }

    private void sleep(int milliseconds) {
        try {
            Thread.sleep(milliseconds);
        } catch (InterruptedException e) {

            e.printStackTrace();

        }

    }



    private void randomDelay() {

        try {

            Thread.sleep(new Random().nextInt(100)); // Introduce a random delay
to break symmetry

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

    }

}
```

Output -

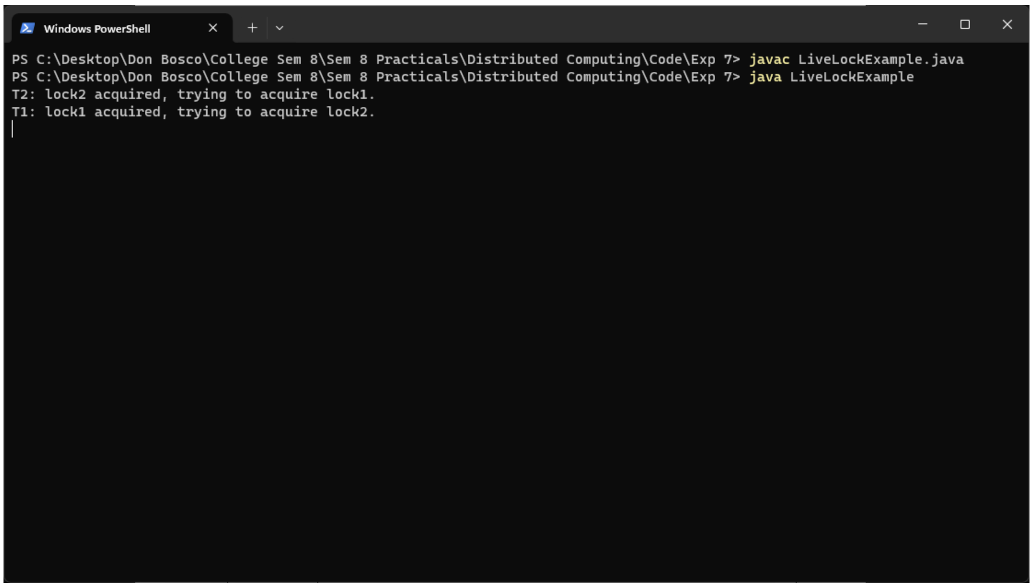
1. Example for Deadlock -



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Code\Exp 7> javac DeadlockExample.java
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Code\Exp 7> java DeadlockExample
T1: lock1 acquired, waiting to acquire lock2.
T2: lock2 acquired, waiting to acquire lock1.
```

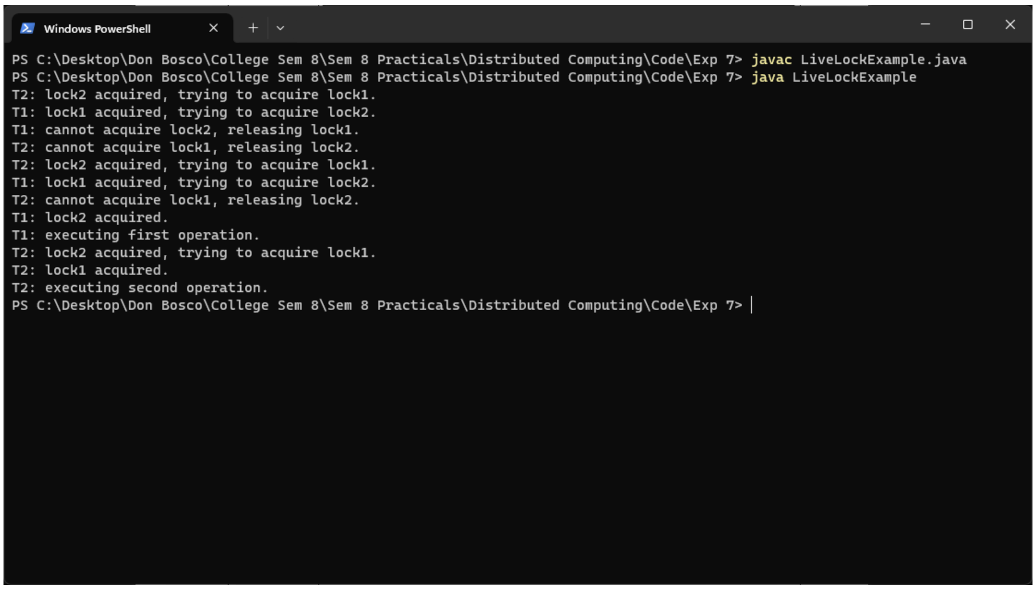
2. Example for Deadlock Resolution -

For first 10 seconds, the 2 Operations are under deadlock -



```
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Code\Exp 7> javac LiveLockExample.java
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Code\Exp 7> java LiveLockExample
T2: lock2 acquired, trying to acquire lock1.
T1: lock1 acquired, trying to acquire lock2.
```

After 10 seconds, timeout releases the deadlock -



```
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Code\Exp 7> javac LiveLockExample.java
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Code\Exp 7> java LiveLockExample
T2: lock2 acquired, trying to acquire lock1.
T1: lock1 acquired, trying to acquire lock2.
T1: cannot acquire lock2, releasing lock1.
T2: cannot acquire lock1, releasing lock2.
T2: lock2 acquired, trying to acquire lock1.
T1: lock1 acquired, trying to acquire lock2.
T2: cannot acquire lock1, releasing lock2.
T1: lock2 acquired.
T1: executing first operation.
T2: lock2 acquired, trying to acquire lock1.
T2: lock1 acquired.
T2: executing second operation.
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Code\Exp 7>
```

Conclusion :	Thus, we have studied and implemented deadlock management in distributed computing using Java to ensure system stability and prevent resource contention issues. We have implemented the methodology to detect, prevent, and resolve deadlocks, thereby enhancing the reliability and performance of distributed systems.

References:	• https://www.baeldung.com/java-deadlock-livelock • https://www.geeksforgeeks.org/introduction-of-deadlock-in-operating-system/ • https://www.geeksforgeeks.org/deadlock-handling-strategies-in-distributed-system/ • https://en.wikipedia.org/wiki/Deadlock

Rubrics for Assessment

Timely Submission	Late	Submitted just after deadline	On time Submission
	0 points	*1 points*	*2 points*
Understanding	Student is confused about the concept	Students has justifiably understood the concept	Students is very clear about the concepts
	0 points	*2 points*	*3 points*
Performance	Students has not performed the Experiment	Student has performed with help	Student has independently performed the experiment
	0 points	*2 points*	*3 points*
Development	Student struggles to write code	Student can write code with the requirement stated	Student can write exceptional code with his own ideas
	0 points	*2 points*	*3 points*
Overall Discipline in Lab	No Discipline maintained	Follows Rules	Sincerely performs the practical is always on time
	0 points	*1 points*	*4 points*