Experiment No. 06

Semester	B.E. Semester VIII – Computer Engineering
Subject	Distributed Computing Lab
Subject Professor In-charge	Dr. Umesh Kulkarni
Assisting Professor	Prof. Prakash Parmar
Academic Year	2024-25

Student Name	Deep Salunkhe
Roll Number	21102A0014

Title: Distributed Resource Management and Deadlock Handling

---

1. Introduction

In distributed computing, managing shared resources efficiently is crucial to ensure optimal system performance and prevent issues like race conditions and deadlocks. This simulation models a distributed resource management system where multiple processes compete for a limited number of shared resources. The primary goal is to ensure that resources are allocated safely and efficiently while preventing conflicts among processes.

---

2. Objectives

The purpose of this simulation is to:

- Implement a distributed resource allocation mechanism using multithreading.
- Ensure thread-safe access to shared resources using synchronization techniques.
- Simulate the behavior of multiple processes requesting, utilizing, and releasing resources.

- Analyze the efficiency and fairness of resource allocation in a multi-process environment.

---

3. Concept of Distributed Resource Management

3.1 Resource Allocation

In a distributed system, resources (e.g., CPU time, memory, files) must be allocated among multiple processes efficiently. The key challenges include:

- **Avoiding race conditions:** Ensuring that multiple processes do not access a shared resource simultaneously in an unsafe manner.
- **Preventing deadlocks:** Avoiding circular dependencies where processes hold resources and wait indefinitely for others to be released.
- **Ensuring fairness:** Allocating resources in a way that no process is starved of resources for long periods.

3.2 Concurrency and Synchronization

Concurrency issues arise when multiple threads (or processes) try to access shared data simultaneously. In this simulation, synchronization mechanisms such as **locks (ReentrantLock)** are used to ensure that only one process can access a resource at a time, preventing conflicts.

---

4. Explanation of the Simulation

4.1 Initialization of Resources

- The system has a **fixed number of resources (5)** that multiple processes (10) compete for.
- Each resource is protected using a **ReentrantLock**, ensuring only one process can acquire it at a time.
- A boolean array tracks which resources are currently allocated.

4.2 Process Behavior

- Each process runs in a separate thread and continuously tries to request a resource.

- When a resource is available, the process locks it, performs work, and then releases it.
- The process sleeps for a random period to simulate real-world execution delays and varying execution times.

4.3 Resource Request and Release Mechanism

- When a process requests a resource, it checks if the resource is free. If available, it locks the resource and marks it as allocated.
- After performing work with the resource, the process releases it by unlocking and marking it as free.
- This ensures that resources are not held indefinitely, allowing fair allocation to all processes.

4.4 Termination of Simulation

- The simulation runs for a fixed period (10 seconds).
- After this period, all processes are interrupted and joined, ensuring a clean shutdown.
- The final message confirms that the simulation has completed successfully.

---

5. Key Takeaways

- **Synchronization is essential:** The use of **ReentrantLock** prevents race conditions, ensuring safe access to shared resources.
- **Randomized execution simulates real-world behavior:** By introducing random sleep times, the simulation models unpredictability in process execution times.
- **Efficient resource allocation avoids starvation:** The approach ensures that every process gets a fair chance to use resources, preventing long waiting times.
- **Proper thread management ensures stability:** Interrupting and joining threads ensures a smooth termination without leaving processes in an inconsistent state.

Code:

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.concurrent.locks.ReentrantLock;

public class DistributedResourceManager {
    // Total number of resources
    private static final int TOTAL_RESOURCES = 5;

    // Number of processes
    private static final int TOTAL_PROCESSES = 10;

    // Resource locks to prevent race conditions
    private static final List<ReentrantLock> resourceLocks = new ArrayList<>();

    // Tracking resource allocation
    private static final boolean[] resourceAllocated = new boolean[TOTAL_RESOURCES];

    public static class Process implements Runnable {
        private final int processId;
        private final Random random;

        public Process(int processId) {
            this.processId = processId;
            this.random = new Random();
        }

        @Override
        public void run() {
            try {
                // Simulate work and resource requests
                while (!Thread.currentThread().isInterrupted()) {
                    int resourceId = requestResource();
                    if (resourceId != -1) {
                        // Simulate work with the resource
                        performWork(resourceId);

                        // Release the resource
                        releaseResource(resourceId);

                        // Random delay to simulate different process behaviors
                        Thread.sleep(random.nextInt(1000) + 500);
                    }
                }
            } catch (InterruptedException e) {
```

```
                System.out.println("Process " + processId + " interrupted.");
            }
        }

        private synchronized int requestResource() {
            for (int i = 0; i < TOTAL_RESOURCES; i++) {
                // Use a lock to ensure thread-safe resource allocation
                resourceLocks.get(i).lock();
                try {
                    if (!resourceAllocated[i]) {
                        resourceAllocated[i] = true;
                        System.out.println("Process " + processId + " acquired resource
" + i);
                        return i;
                    }
                } finally {
                    resourceLocks.get(i).unlock();
                }
            }
            return -1; // No available resources
        }

        private void performWork(int resourceId) throws InterruptedException {
            // Simulate some work being done with the resource
            System.out.println("Process " + processId + " working with resource " +
resourceId);
            Thread.sleep(random.nextInt(500) + 200);
        }

        private synchronized void releaseResource(int resourceId) {
            resourceLocks.get(resourceId).lock();
            try {
                resourceAllocated[resourceId] = false;
                System.out.println("Process " + processId + " released resource " +
resourceId);
            } finally {
                resourceLocks.get(resourceId).unlock();
            }
        }
    }

    public static void main(String[] args) {
        // Initialize resource locks
        for (int i = 0; i < TOTAL_RESOURCES; i++) {
            resourceLocks.add(new ReentrantLock());
        }

        // Create and start processes
```

```
        List<Thread> processes = new ArrayList<>();
        for (int i = 0; i < TOTAL_PROCESSES; i++) {
            Thread processThread = new Thread(new Process(i));
            processThread.start();
            processes.add(processThread);
        }

        // Let the simulation run for a while
        try {
            Thread.sleep(10000); // Run for 10 seconds
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Interrupt all processes
        processes.forEach(Thread::interrupt);

        // Wait for all processes to terminate
        processes.forEach(thread -> {
            try {
                thread.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

        System.out.println("Distributed resource simulation completed.");
    }
}
```

Output:

```
PS E:\GIt\Sem-8> cd "e:\GIt\Sem-8\DC\Lab6\" ; if ($?) { javac
DistributedResourceManager.java } ; if ($?) { java DistributedResourceManager }
Process 0 acquired resource 0
Process 6 acquired resource 1
Process 6 working with resource 1
Process 0 working with resource 0
Process 4 acquired resource 2
Process 4 working with resource 2
Process 3 acquired resource 3
Process 3 working with resource 3
Process 2 acquired resource 4
Process 2 working with resource 4
Process 4 released resource 2
Process 1 acquired resource 2
Process 1 working with resource 2
```

```
Process 0 released resource 0
Process 7 acquired resource 0
Process 7 working with resource 0
Process 6 released resource 1
Process 8 acquired resource 1
Process 8 working with resource 1
Process 1 released resource 2
Process 5 acquired resource 2
Process 5 working with resource 2
Process 3 released resource 3
Process 9 acquired resource 3
Process 9 working with resource 3
Process 2 released resource 4
Process 7 released resource 0
Process 8 released resource 1
Process 5 released resource 2
Process 6 acquired resource 0
Process 6 working with resource 0
Process 9 released resource 3
Process 0 acquired resource 1
Process 0 working with resource 1
Process 1 acquired resource 2
Process 1 working with resource 2
Process 4 acquired resource 3
Process 4 working with resource 3
Process 6 released resource 0
Process 2 acquired resource 0
Process 2 working with resource 0
Process 0 released resource 1
Process 4 released resource 3
Process 7 acquired resource 1
Process 7 working with resource 1
Process 3 acquired resource 3
Process 3 working with resource 3
Process 2 released resource 0
Process 9 acquired resource 0
Process 9 working with resource 0
Process 1 released resource 2
Process 8 acquired resource 2
Process 8 working with resource 2
Process 3 released resource 3
Process 5 acquired resource 3
Process 5 working with resource 3
Process 0 acquired resource 4
Process 0 working with resource 4
Process 7 released resource 1
Process 6 acquired resource 1
Process 6 working with resource 1
```

```
Process 5 released resource 3
Process 4 acquired resource 3
Process 4 working with resource 3
Process 9 released resource 0
Process 8 released resource 2
Process 0 released resource 4
Process 6 released resource 1
Process 4 released resource 3
Process 5 acquired resource 0
Process 5 working with resource 0
Process 2 acquired resource 1
Process 2 working with resource 1
Process 1 acquired resource 2
Process 1 working with resource 2
Process 0 acquired resource 3
Process 0 working with resource 3
Process 3 acquired resource 4
Process 3 working with resource 4
Process 2 released resource 1
Process 6 acquired resource 1
Process 6 working with resource 1
Process 3 released resource 4
Process 7 acquired resource 4
Process 7 working with resource 4
Process 5 released resource 0
Process 9 acquired resource 0
Process 9 working with resource 0
Process 6 released resource 1
Process 4 acquired resource 1
Process 4 working with resource 1
Process 1 released resource 2
Process 8 acquired resource 2
Process 8 working with resource 2
Process 0 released resource 3
Process 7 released resource 4
Process 4 released resource 1
Process 2 acquired resource 1
Process 2 working with resource 1
Process 6 acquired resource 3
Process 6 working with resource 3
Process 5 acquired resource 4
Process 5 working with resource 4
Process 8 released resource 2
Process 9 released resource 0
Process 0 acquired resource 0
Process 0 working with resource 0
Process 5 released resource 4
Process 6 released resource 3
```

```
Process 3 acquired resource 2
Process 3 working with resource 2
Process 7 acquired resource 3
Process 7 working with resource 3
Process 2 released resource 1
Process 1 acquired resource 1
Process 1 working with resource 1
Process 0 released resource 0
Process 7 released resource 3
Process 8 acquired resource 0
Process 8 working with resource 0
Process 4 acquired resource 3
Process 4 working with resource 3
Process 1 released resource 1
Process 3 released resource 2
Process 5 acquired resource 1
Process 5 working with resource 1
Process 9 acquired resource 2
Process 9 working with resource 2
Process 2 acquired resource 4
Process 2 working with resource 4
Process 8 released resource 0
Process 9 released resource 2
Process 4 released resource 3
Process 2 released resource 4
Process 7 acquired resource 0
Process 7 working with resource 0
Process 6 acquired resource 2
Process 6 working with resource 2
Process 0 acquired resource 3
Process 0 working with resource 3
Process 5 released resource 1
Process 9 acquired resource 1
Process 9 working with resource 1
Process 6 released resource 2
Process 1 acquired resource 2
Process 1 working with resource 2
Process 3 acquired resource 4
Process 3 working with resource 4
Process 7 released resource 0
Process 0 released resource 3
Process 8 acquired resource 0
Process 8 working with resource 0
Process 3 released resource 4
Process 9 released resource 1
Process 1 released resource 2
Process 4 acquired resource 1
Process 4 working with resource 1
```

```
Process 2 acquired resource 2
Process 2 working with resource 2
Process 8 released resource 0
Process 0 acquired resource 0
Process 0 working with resource 0
Process 5 acquired resource 3
Process 5 working with resource 3
Process 3 acquired resource 4
Process 3 working with resource 4
Process 4 released resource 1
Process 2 released resource 2
Process 1 acquired resource 1
Process 1 working with resource 1
Process 0 released resource 0
Process 7 acquired resource 0
Process 7 working with resource 0
Process 6 acquired resource 2
Process 6 working with resource 2
Process 5 released resource 3
Process 3 released resource 4
Process 9 acquired resource 3
Process 9 working with resource 3
Process 1 released resource 1
Process 2 acquired resource 1
Process 2 working with resource 1
Process 6 released resource 2
Process 7 released resource 0
Process 0 acquired resource 0
Process 0 working with resource 0
Process 2 released resource 1
Process 9 released resource 3
Process 5 acquired resource 1
Process 5 working with resource 1
Process 4 acquired resource 2
Process 4 working with resource 2
Process 8 acquired resource 3
Process 8 working with resource 3
Process 0 released resource 0
Process 4 released resource 2
Process 5 released resource 1
Process 6 acquired resource 0
Process 6 working with resource 0
Process 1 acquired resource 1
Process 1 working with resource 1
Process 3 acquired resource 2
Process 3 working with resource 2
Process 6 released resource 0
Process 2 acquired resource 0
```

```
Process 2 working with resource 0
Process 3 released resource 2
Process 8 released resource 3
Process 1 released resource 1
Process 6 interrupted.
Process 1 interrupted.
Process 9 interrupted.
Process 2 interrupted.
Process 5 interrupted.
Process 8 interrupted.
Process 3 interrupted.
Process 0 interrupted.
Process 7 interrupted.
Process 4 interrupted.
Distributed resource simulation completed.
```

Conclusion

This simulation successfully demonstrates the principles of distributed resource management. By implementing a locking mechanism and tracking resource allocation, the system ensures safe, fair, and efficient sharing of limited resources. This experiment provides a foundational understanding of how operating systems and distributed systems handle resource contention in multi-threaded environments.