## Exp 1: Inter- Process communication using Java

**Producer.java**

```
import java.io.IOException;
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

public class Producer {
   public static void main(String args[]) throws IOException, InterruptedException {
      RandomAccessFile rd = new RandomAccessFile("D:/Code/Exp 1/mapped.txt", "rw");
      FileChannel fc = rd.getChannel();
      MappedByteBuffer mem = fc.map(FileChannel.MapMode.READ_WRITE, 0, 1000);

      try {
         Thread.sleep(10000);
      } catch (InterruptedException e) {
         e.printStackTrace();
      }

      for (int i = 1; i <= 10; i++) {
         mem.put((byte) i);
         System.out.println("Process 1: " + (byte) i);
         Thread.sleep(1); // time to allow CPU cache refreshed
      }

      // Close resources
      fc.close();
      rd.close();
   }
}
```

**Consumer.java**

```
import java.io.IOException;
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

/**
 * Consumer process reading data from the memory-mapped file
 */
public class Consumer {
    public static void main(String args[]) throws IOException, InterruptedException {
        RandomAccessFile rd = new RandomAccessFile("D:/Code/Exp 1/mapped.txt", "r");
        FileChannel fc = rd.getChannel();
        MappedByteBuffer mem = fc.map(FileChannel.MapMode.READ_ONLY, 0, 1000);

        // Assuming that the producer has already written the data
        for (int i = 0; i < 9; i++) {
            byte value = mem.get();
            System.out.println("Process 2: " + value);
        }

        // Close resources
        fc.close();
        rd.close();
    }
}
```

## Exp 2 Program to demonstrate Client/Server application Using RMI

**RMI_Client.java**

```
import java.rmi.Naming;
import java.util.Scanner;

public class RMI_Client {
   static Scanner input = null;
   public static void main(String[] args) throws Exception {
      RMI_Chat_Interface chatapi = (RMI_Chat_Interface)

      Naming.lookup("rmi://localhost:6000/chat");
      input = new Scanner(System.in);
      System.out.println("Connected to server...");

      System.out.println("Type a message for sending to server...");
      String message = input.nextLine();
      while (!message.equals("Bye")) {
         chatapi.sendToServer(message);
         message = input.nextLine();
      }
   }
}
```

**RMI_Server.java**

```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class RMI_Server extends UnicastRemoteObject implements RMI_Chat_Interface {
    public RMI_Server() throws RemoteException {
        super();
    }

    @Override
    public void sendToServer(String message) throws RemoteException {
        System.out.println("Client says: " + message);
    }
    public static void main(String[] args) throws Exception {
        Registry rmiregistry = LocateRegistry.createRegistry(6000);
        rmiregistry.bind("chat", new RMI_Server());
        System.out.println("Chat server is running...");
    }
}
```

RMI_Chat_Interface.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RMI_Chat_Interface extends Remote {
    public void sendToServer(String message) throws RemoteException;
}
```

## Exp 3 Group Communication using Java

**Server.java**

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

public class Server {
   private static List<PrintWriter> writers = new ArrayList<>();

   public static void main(String[] args) throws Exception {
      ServerSocket listener = new ServerSocket(9001);
      System.out.println("The server is running at port 9001.");
      while (true) {
         new Handler(listener.accept()).start();
      }
   }

   private static class Handler extends Thread {
      private Socket socket;

      public Handler(Socket socket) {
         this.socket = socket;
      }

      public void run() {
         try {
```

```
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            writers.add(out);

            while (true) {
                String input = in.readLine();
                if (input == null) {
                    return;
                }
                for (PrintWriter writer : writers) {
                    writer.println(input);
                }
            }
        } catch (Exception e) {
            System.err.println(e);
        }
    }
  }
}
```

**master.java**

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class master {
   public static void main(String[] args) throws Exception {
```

```
        Scanner sc = new Scanner(System.in);
        Socket socket = new Socket("localhost", 9001);
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        System.out.print("Enter your name: ");
        String name = sc.nextLine();

        out.println(name + " (Master)");

        Thread readerThread = new Thread(() -> {
            try {
                while (true) {
                    String line = in.readLine();
                    if (line != null && !line.isEmpty()) {
                        System.out.println(line);
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
        readerThread.start();

        while (true) {
            System.out.print("Enter a message: ");
            String message = sc.nextLine();
            out.println(name + ": " + message);
        }
    }
}
```

**Slave1.java**

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class slave1 {
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        Socket socket = new Socket("localhost", 9001);
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        System.out.print("Enter your name: ");
        String name = sc.nextLine();

        out.println(name + " (Slave)");

        Thread readerThread = new Thread(() -> {
            try {
                while (true) {
                    String line = in.readLine();
                    if (line != null && !line.isEmpty()) {
                        System.out.println(line);
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
```

```
        readerThread.start();

        while (true) {
            System.out.print("Enter a message: ");
            String message = sc.nextLine();
            out.println(name + ": " + message);
        }
    }
}
```

**Slave2,java**

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class slave2 {
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        Socket socket = new Socket("localhost", 9001);
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        System.out.print("Enter your name: ");
        String name = sc.nextLine();

        out.println(name + " (Slave)");

        Thread readerThread = new Thread(() -> {
            try {
```

```
            while (true) {
                String line = in.readLine();
                if (line != null && !line.isEmpty()) {
                    System.out.println(line);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    });
    readerThread.start();

    while (true) {
        System.out.print("Enter a message: ");
        String message = sc.nextLine();
        out.println(name + ": " + message);
    }
  }
}
```

## EXP 4 : Clock Synchronization

**ClockServer.java**

```
import java.io.*;
import java.net.*;

public class ClockServer {
    public static void main(String[] args) {
        final int port = 9090;

        try {
            ServerSocket serverSocket = new ServerSocket(port);
            System.out.println("Server started. Waiting for clients...");

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Client connected: " + clientSocket.getInetAddress());

                // Handle client in a separate thread
                Thread clientThread = new Thread(new ClientHandler(clientSocket));
                clientThread.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

class ClientHandler implements Runnable {
    private final Socket clientSocket;
```

```
    public ClientHandler(Socket clientSocket) {
        this.clientSocket = clientSocket;
    }

    @Override
    public void run() {
        try {
            // Get input and output streams
            InputStream inFromClient = clientSocket.getInputStream();
            OutputStream outToClient = clientSocket.getOutputStream();

            // Receive client's time
            DataInputStream in = new DataInputStream(inFromClient);
            long clientTime = in.readLong();
            System.out.println("Received client time: " + clientTime);

            // Get server's current time
            long serverTime = System.currentTimeMillis();
            System.out.println("Server time: " + serverTime);

            // Send server's time to client
            DataOutputStream out = new DataOutputStream(outToClient);
            out.writeLong(serverTime);
            // Close the connection
            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

**ClockClient.java**

```
import java.io.*;
import java.net.*;

public class ClockClient {
    public static void main(String[] args) {
        final String serverName = "localhost";
        final int port = 9090;

        try {
            Socket socket = new Socket(serverName, port);
            System.out.println("Connected to server.");

            // Get input and output streams
            OutputStream outToServer = socket.getOutputStream();
            InputStream inFromServer = socket.getInputStream();

            // Get client's current time
            long clientTime = System.currentTimeMillis();
            System.out.println("Client time: " + clientTime);

            // Send client's time to server
            DataOutputStream out = new DataOutputStream(outToServer);
            out.writeLong(clientTime);

            // Receive server's time
            DataInputStream in = new DataInputStream(inFromServer);
            long serverTime = in.readLong();
            System.out.println("Server time: " + serverTime);
```

```
            // Calculate time difference
            long timeDifference = serverTime - clientTime;
            System.out.println("Time difference: " + timeDifference);

            // Adjust client's clock
            long adjustedTime = System.currentTimeMillis() + timeDifference;
            System.out.println("Adjusted time: " + adjustedTime);

            // Close the connection
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

## EXP 5: Election Algorithm.

**BullyAlgo.java**

```
import java.io.*;


class BullyAlgo {
   int coord, ch, crash;
   int prc[];
   public void election(int n) throws IOException {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      System.out.println("\nThe Coordinator Has Crashed!");
      int flag = 1;
      while (flag == 1) {
         crash = 0;
         for (int i1 = 0; i1 < n; i1++)
            if (prc[i1] == 0)
               crash++;
         if (crash == n) {
            System.out.println("\n*** All Processes Are Crashed ***");
            break;
         } else {
            System.out.println("\nEnter The Initiator");
            int init = Integer.parseInt(br.readLine());
            if ((init < 1) || (init > n) || (prc[init - 1] == 0)) {
               System.out.println("\nInvalid Initiator");
               continue;
            }
            for (int i1 = init - 1; i1 < n; i1++)
               System.out.println("Process " + (i1 + 1) + " Called For Election");
            System.out.println("");
            for (int i1 = init - 1; i1 < n; i1++) {
```

```
            if (prc[i1] == 0) {
                System.out.println("Process " + (i1 + 1) + " Is Dead");
            } else
                System.out.println("Process " + (i1 + 1) + " Is In");
        }
        for (int i1 = n - 1; i1 >= 0; i1--)
            if (prc[i1] == 1) {
                coord = (i1 + 1);
                System.out.println("\n*** New Coordinator Is " + (coord) + " ***");
                flag = 0;
                break;
            }
      }
   }
}
public void Bully() throws IOException {
   BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
   System.out.println("Enter The Number Of Processes: ");
   int n = Integer.parseInt(br.readLine());
   prc = new int[n];
   crash = 0;
   for (int i = 0; i < n; i++)
      prc[i] = 1;
   coord = n;
   do {
      System.out.println("\n\t1. Crash A Process");
      System.out.println("\t2. Recover A Process");
      System.out.println("\t3. Display New Coordinator");
      System.out.println("\t4. Exit");
      ch = Integer.parseInt(br.readLine());
```

```
switch (ch) {
    case 1:
        System.out.println("\nEnter A Process To Crash");
        int cp = Integer.parseInt(br.readLine());
        if ((cp > n) || (cp < 1)) {
            System.out.println("Invalid Process! Enter A Valid Process");
        } else if ((prc[cp - 1] == 1) && (coord != cp)) {
            prc[cp - 1] = 0;
            System.out.println("\nProcess " + cp + " Has Been Crashed");
        } else if ((prc[cp - 1] == 1) && (coord == cp)) {
            prc[cp - 1] = 0;
            election(n);
        } else
            System.out.println("\nProcess " + cp + " Is Already Crashed");
        break;
    case 2:
        System.out.println("\nCrashed Processes Are: \n");
        for (int i = 0; i < n; i++) {
            if (prc[i] == 0)
                System.out.println(i + 1);
            crash++;
        }
        System.out.println("Enter The Process You Want To Recover");
        int rp = Integer.parseInt(br.readLine());
        if ((rp < 1) || (rp > n))
            System.out.println("\nInvalid Process. Enter A Valid ID");
        else if ((prc[rp - 1] == 0) && (rp > coord)) {
            prc[rp - 1] = 1;
            System.out.println("\nProcess " + rp + " Has Recovered");
            coord = rp;
```

```
                System.out.println("\nProcess " + rp + " Is The New Coordinator");
                crash--;
            } else if (crash == n) {
                prc[rp - 1] = 1;
                coord = rp;
                System.out.println("\nProcess " + rp + " Is The New Coordinator");
                crash--;
            } else if ((prc[rp - 1] == 0) && (rp < coord)) {
                prc[rp - 1] = 1;
                System.out.println("\nProcess " + rp + " Has Recovered");
            } else
                System.out.println("\nProcess " + rp + " Is Not A Crashed Process");
            break;
        case 3:
            System.out.println("\nCurrent Coordinator Is " + coord);
            break;
        case 4:
            System.exit(0);
            break;
        default:
            System.out.println("\nInvalid Entry!");
            break;
      }
    } while (ch != 4);
  }
  public static void main(String args[]) throws IOException {
    BullyAlgo ob = new BullyAlgo();
    ob.Bully();
  }
}
```

## EXP 6 : Mutual Exclusion Algorithm

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


class MasterSlaveMutualExclusion {
    private Semaphore mutex;
    private boolean isMaster;

    public MasterSlaveMutualExclusion(boolean isMaster) {
        this.mutex = new Semaphore(1);
        this.isMaster = isMaster;
    }

    public void enterCriticalSection() {
        try {
            if (isMaster) {
                mutex.acquire();
            } else {
                // Slave nodes request permission from the inster
                requestPermissionFromMaster();
            }
            System.out.println(Thread.currentThread().getName() + " is entering the critical
section.");
            Thread.sleep(1000); // Simulating some work inside the critical section
            System.out.println(Thread.currentThread().getName() + " is leaving the critical
section.");
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            if (isMaster) {
                mutex.release();
            }
```

```
        }
    }

    private void requestPermissionFromMaster() throws InterruptedException {
        // Here, we can implement communication with the master node to request permission
        // For simplicity, we'll just wait until the master is available
        while (!mutex.tryAcquire()) {
            Thread.sleep(100); // Retry after some time
        }
        mutex.release(); // Immediately release to maintain mutual exclusion
    }
}

class Node implements Runnable {
    private MasterSlaveMutualExclusion mutex;

    public Node(MasterSlaveMutualExclusion mutex) {
        this.mutex = mutex;
    }

    @Override
    public void run() {
        mutex.enterCriticalSection();
    }
}

public class Main {
    public static void main(String[] args) {
        MasterSlaveMutualExclusion mutex = new MasterSlaveMutualExclusion(true); // Assume the first node is the master
```

```
        // Create slave nodes
        Thread[] slaves = new Thread[5];
        for (int i = 0; i < slaves.length; i++) {
            slaves[i] = new Thread(new Node(new MasterSlaveMutualExclusion(false)), "Slave-"
+ i);
        }

        // Start slave threads
        for (Thread slave : slaves) {
            slave.start();
        }

        // Simulate master work
        try {
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Master releases the resource
        mutex.enterCriticalSection();
    }
}
```

```
PS D:\Code\Exp 6> javac Main.java
PS D:\Code\Exp 6> java Main
Slave-2 is entering the critical section.
Slave-3 is entering the critical section.
Slave-0 is entering the critical section.
Slave-4 is entering the critical section.
Slave-1 is entering the critical section.
Slave-1 is leaving the critical section.
Slave-0 is leaving the critical section.
Slave-4 is leaving the critical section.
Slave-3 is leaving the critical section.
Slave-2 is leaving the critical section.
main is entering the critical section.
main is leaving the critical section.
```

## EXP 7: Deadlock Management in Distributed Systems

**DeadlockExample.java**

```
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();
        }
    }
}
```

**LiveLockExample.java**

```
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();
        }
    }
}
```

## EXP 8: Load Balancing Algorithm in Java

**DistributedLoadBalancer.java**

```
import java.util.ArrayList;
import java.util.List;

// Represents a node in the distributed system
class Node {
    private int id;
    private int workload;

    public Node(int id) {
        this.id = id;
        this.workload = 0;
    }

    public int getId() {
        return id;
    }

    public int getWorkload() {
        return workload;
    }

    public void assignTask(int taskWorkload) {
        this.workload += taskWorkload;
    }
    @Override
    public String toString() {
        return "Node " + id + " (Workload: " + workload + ")";
    }
```

```
}

// Represents a task to be executed in the distributed system
class Task {
    private int workload;

    public Task(int workload) {
        this.workload = workload;
    }

    public int getWorkload() {
        return workload;
    }
}

// Load balancing algorithm for distributing tasks among nodes
class LoadBalancer {
    private List<Node> nodes;

    public LoadBalancer(List<Node> nodes) {
        this.nodes = nodes;
    }
    // Assigns a task to the least loaded node
    public void assignTask(Task task) {
        Node leastLoadedNode = nodes.get(0);
        for (Node node : nodes) {
            if (node.getWorkload() < leastLoadedNode.getWorkload()) {
                leastLoadedNode = node;
            }
        }
```

```
        leastLoadedNode.assignTask(task.getWorkload());
        System.out.println("\n");
        System.out.println("Assigned Task with Workload " + task.getWorkload() + " to " +
leastLoadedNode);
        System.out.println("\n");
    }
}

public class DistributedLoadBalancer {
    public static void main(String[] args) {
        // Create nodes
        List<Node> nodes = new ArrayList<>();
        for (int i = 1; i <= 3; i++) {
            nodes.add(new Node(i));
        }

        // Initialize load balancer
        LoadBalancer loadBalancer = new LoadBalancer(nodes);
        // Create tasks
        List<Task> tasks = new ArrayList<>();
        tasks.add(new Task(30));
        tasks.add(new Task(40));
        tasks.add(new Task(55));
        tasks.add(new Task(75));

        // Assign tasks to nodes using load balancer
        for (Task task : tasks) {
            loadBalancer.assignTask(task);
        }
    }
}
```