DF Pracs StudyDoc

Expt8 Code

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);
        }
    }
```

```
}
```

Commands:

javac *DistributedLoadBalancer.java*

java *DistributedLoadBalancer*

Expt7 Code

ChandyMisraHaassWithDeadlockManagement.java

```
import java.util.*;

class Process {
   int id;
   boolean isWaiting;  // Track if process is waiting for a resource
   List<Process> waitingFor;  // List of processes this process is waiting on
   Set<Integer> visitedProbes; // Set to track visited probes to detect cycles

   public Process(int id) {
      this.id = id;
      this.isWaiting = false;
      this.waitingFor = new ArrayList<>();
      this.visitedProbes = new HashSet<>();
   }

   // Add dependency: process is waiting for another process
   public void addDependency(Process p) {
      this.waitingFor.add(p);
      this.isWaiting = true;
   }

   // Abort the process (deadlock management)
```

```
    public void abort() {
        System.out.println("Aborting Process " + this.id + " to resolve deadlock.");
        this.isWaiting = false;
        this.waitingFor.clear();
    }
}

class DeadlockDetector {
    private Map<Integer, Process> processes; // Store all processes

    public DeadlockDetector() {
        this.processes = new HashMap<>();
    }

    // Add process to system
    public void addProcess(int id) {
        processes.put(id, new Process(id));
    }

    // Create a dependency: Pi waits for Pj
    public void addDependency(int from, int to) {
        if (processes.containsKey(from) && processes.containsKey(to)) {
            processes.get(from).addDependency(processes.get(to));
        }
    }

    // Start deadlock detection
    public boolean detectDeadlock() {
        System.out.println("Starting deadlock detection...");
        for (Process process : processes.values()) {
            if (process.isWaiting) {
```

```
            Set<Integer> visited = new HashSet<>();
            if (isCyclic(process, visited)) {
                return true;
            }
        }
    }
    return false;
}

// Helper method to simulate probe message propagation and check for cycles
private boolean isCyclic(Process process, Set<Integer> visited) {
    if (visited.contains(process.id)) {
        // Cycle detected (deadlock)
        System.out.println("Deadlock detected! Process " + process.id + " is in a cycle.");
        // Deadlock management: Abort the process involved in the cycle
        process.abort();
        return true;
    }

    // Mark the current process as visited
    visited.add(process.id);
    for (Process dependent : process.waitingFor) {
        // Simulate sending a probe to the dependent process
        System.out.println("Process " + process.id + " sends a probe to Process " + dependent.id);
        if (isCyclic(dependent, new HashSet<>(visited))) {
            return true;
        }
    }
    return false;
}
```

```
    // Getter method for processes
    public Map<Integer, Process> getProcesses() {
        return processes;
    }
}

public class ChandyMisraHaassWithDeadlockManagement {
    public static void main(String[] args) {
        // Step 1: Initialize the deadlock detector
        DeadlockDetector detector = new DeadlockDetector();

        // Step 2: Add processes to the system
        detector.addProcess(1);
        detector.addProcess(2);
        detector.addProcess(3);
        detector.addProcess(4);

        // Step 3: Establish dependencies (resource waits)
        detector.addDependency(1, 2); // Process 1 waits for Process 2
        detector.addDependency(2, 3); // Process 2 waits for Process 3
        detector.addDependency(3, 4); // Process 3 waits for Process 4
        detector.addDependency(4, 1); // Process 4 waits for Process 1 (creating a cycle)

        // Step 4: Detect deadlock
        if (detector.detectDeadlock()) {
            System.out.println("Deadlock detected and resolved.");
        } else {
            System.out.println("No deadlock detected.");
        }

        // Step 5: Verify if deadlock was resolved
```

```
        System.out.println("\nAfter deadlock management:");
        System.out.println("Processes in the system:");
        for (Process process : detector.getProcesses().values()) {
            System.out.println("Process " + process.id + " - Waiting: " + process.isWaiting);
        }
    }
}
```

Commands:

javac ChandyMisraHaassWithDeadlockManagement.java

java ChandyMisraHaassWithDeadlockManagement

Expt6 Code

MaekawaLamport.java

NOTE: Make Sure to check which algo u want for mutex (choose one), get rid of fns with one of the names.

```
import java.util.Arrays;

class Process {
    int processId;
    boolean requested;
    boolean held;
    int timestamp;
    int vote;

    public Process(int id) {
        this.processId = id;
        this.requested = false;
        this.held = false;
        this.timestamp = 0;
```

```
        this.vote = 0;
    }
}

public class MaekawaLamport {
    private static final int NUM_PROCESSES = 5;
    private static Process[] processes = new Process[NUM_PROCESSES];
    private static int[][] intersectionSets = new int[NUM_PROCESSES][NUM_PROCESSES];
    private static int logicalClock = 0;

    public static void initializeProcesses() {
        for (int i = 0; i < NUM_PROCESSES; i++) {
            processes[i] = new Process(i);
        }

        int[][] exampleSets = {
            {0, 1, 2, -1, -1},
            {0, 1, 3, -1, -1},
            {0, 2, 4, -1, -1},
            {1, 3, 4, -1, -1},
            {2, 3, 4, -1, -1}
        };

        for (int i = 0; i < NUM_PROCESSES; i++) {
            intersectionSets[i] = Arrays.copyOf(exampleSets[i], NUM_PROCESSES);
        }
    }

    public static void requestCriticalSection(int processId) {
        processes[processId].requested = true;
        processes[processId].timestamp++;
```

```
for (int i = 0; i < NUM_PROCESSES; i++) {
    if (intersectionSets[processId][i] == -1) break;
    int targetProcessId = intersectionSets[processId][i];
    System.out.println("Process " + processId + " sending request to Process " + targetProcessId);

    if (processes[targetProcessId].vote == 0 && !processes[targetProcessId].requested) {
        processes[targetProcessId].vote = 1;
        System.out.println("Process " + targetProcessId + " granting vote to Process " + processId);
    } else {
        System.out.println("Process " + targetProcessId + " denying vote to Process " + processId);
    }
}

int votesReceived = 0;
int setSize = 0;
for (int i = 0; i < NUM_PROCESSES; i++) {
    if (intersectionSets[processId][i] == -1) break;
    int targetProcessId = intersectionSets[processId][i];
    setSize++;
    if (processes[targetProcessId].vote == 1) {
        votesReceived++;
    }
}

if (votesReceived == setSize) {
    processes[processId].held = true;
    System.out.println("Process " + processId + " entering critical section");
    processes[processId].held = false;
    processes[processId].requested = false;
    System.out.println("Process " + processId + " exiting critical section");
```

```
            for (int i = 0; i < NUM_PROCESSES; i++) {
                if (intersectionSets[processId][i] == -1) break;
                int targetProcessId = intersectionSets[processId][i];
                processes[targetProcessId].vote = 0;
            }
        } else {
            System.out.println("Process " + processId + " waiting for votes");
        }
    }

    public static int max(int a, int b) {
        return Math.max(a, b);
    }

    public static void lamportEvent(int processId) {
        logicalClock++;
        System.out.println("Process " + processId + ": Event occurred at time " + logicalClock);
    }

    public static void lamportSendMessage(int senderId, int receiverId) {
        logicalClock++;
        System.out.println("Process " + senderId + " sending message to Process " + receiverId + " at
time " + logicalClock);
        lamportReceiveMessage(receiverId, logicalClock);
    }

    public static void lamportReceiveMessage(int receiverId, int messageTimestamp) {
        logicalClock = max(logicalClock, messageTimestamp) + 1;
        System.out.println("Process " + receiverId + " received message at time " + logicalClock);
    }
```

```
    public static void main(String[] args) {
        System.out.println("Maekawa's Algorithm:");
        initializeProcesses();
        requestCriticalSection(0);
        requestCriticalSection(1);

        System.out.println("\nLamport's Algorithm:");
        lamportEvent(0);
        lamportSendMessage(0, 1);
        lamportEvent(1);
        lamportReceiveMessage(0, logicalClock);
        lamportEvent(1);
        lamportSendMessage(1, 2);
        lamportReceiveMessage(2, logicalClock);
    }
}
```

Commands:

javac MaekawaLamport.java

java MaekawaLamport

Expt5 Code

BullyAlgorithmExample.java

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

// Class representing a node in the distributed system
class Node {
    private int nodeId;
```

```
private boolean isCoordinator;

public Node(int nodeId) {
    this.nodeId = nodeId;
    this.isCoordinator = false;
}

public int getNodeId() {
    return nodeId;
}

public boolean isCoordinator() {
    return isCoordinator;
}

public void setCoordinator(boolean coordinator) {
    isCoordinator = coordinator;
}

// Method to initiate an election
public void initiateElection(List<Node> nodes) {
    System.out.println("Node " + nodeId + " initiates election.");

    for (Node node : nodes) {
        if (node.getNodeId() > this.nodeId) {
            // Send election message to higher priority nodes
            node.receiveElectionMessage(this);
        }
    }

    // Assume election process completes after initiating
```

```
        becomeCoordinator();
    }

    // Method to receive election message from another node
    public void receiveElectionMessage(Node sender) {
        System.out.println("Node " + nodeId + " receives election message from Node " +
sender.getNodeId());

        // Respond if current node has higher priority
        if (this.nodeId > sender.getNodeId()) {
            System.out.println("Node " + nodeId + " responds to Node " + sender.getNodeId());
            sender.receiveResponse(this);
        }
    }

    // Method to receive response and acknowledge as coordinator
    public void receiveResponse(Node sender) {
        System.out.println("Node " + nodeId + " receives response from Node " + sender.getNodeId());
    }

    // Method to become the coordinator
    public void becomeCoordinator() {
        System.out.println("Node " + nodeId + " becomes the coordinator.");
        this.isCoordinator = true;
    }
}

public class BullyAlgorithmExample {

    public static void main(String[] args) {
        // Create nodes
```

```
        Node node1 = new Node(1);
        Node node2 = new Node(2);
        Node node3 = new Node(3);
        Node node4 = new Node(4);
        Node node5 = new Node(5);

        // List of nodes in the distributed system
        List<Node> nodes = new ArrayList<>();
        nodes.add(node1);
        nodes.add(node2);
        nodes.add(node3);
        nodes.add(node4);
        nodes.add(node5);

        // Simulate failure of current coordinator (Node 3)
        node3.setCoordinator(false);

        // Assume Node 3 detects coordinator failure and initiates election
        node3.initiateElection(nodes);
    }
}
```

Commands:

javac BullyAlgorithmExample.java

java BullyAlgorithmExample

Expt4 Code

Note: Again there are 2 approaches here, may choose what suits best

Approach 1

TimeServer.java

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

```
import java.net.*;

public class TimeServer {
   public static void main(String[] args) {
      try (ServerSocket serverSocket = new ServerSocket(8080)) {
         System.out.println("Time server started. Listening on port 8080...");
         while (true) {
            // Accept client connections
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected: " + clientSocket);
            long currentTime = System.currentTimeMillis(); // Gets the time
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
            out.println(currentTime);
            out.close();
            clientSocket.close();
         }
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

TimeClient.java

```
import java.io.*;
import java.net.*;
import java.text.SimpleDateFormat;
import java.util.Date;

public class TimeClient {
   public static void main(String[] args) {
      try {
```

```
            long clientLocalTime = System.currentTimeMillis();
            System.out.println("Client's Local Time: " + formatDate(clientLocalTime));
            Socket socket = new Socket("localhost", 8080);
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            String currentTimeStr = in.readLine();
            long serverTime = Long.parseLong(currentTimeStr);
            System.out.println("Server Time: " + formatDate(serverTime));
            long roundTripTime = System.currentTimeMillis() - serverTime;
            System.out.println("Round-trip Time: " + roundTripTime + " milliseconds");

            // Adjusting local clock by half of the round-trip time
            long adjustedTime = System.currentTimeMillis() + (roundTripTime / 2);
            System.out.println("Adjusted Local Time: " + formatDate(adjustedTime));

            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //To get time in human-readable format
    private static String formatDate(long time) {
        SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-YYYY HH:mm:ss.SSS");
        return sdf.format(new Date(time));
    }
}
```

CristiansAlgorithmMain.java

```
public class CristiansAlgorithmMain {
    public static void main(String[] args) {
        // Creating time server in a separate thread
```

```
        Thread serverThread = new Thread(() -> TimeServer.main(null));
        serverThread.start();

        //Adding wait
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Start the time client
        TimeClient.main(null);
    }
}
```

Commands:

javac TimeServer.java

javac TimeClient.java

javac CristiansAlgorithmMain.java

java CristiansAlgorithmMain

java TimeClient.java

Approach 2

BerkeleyAlgorithm.java

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.text.SimpleDateFormat;

public class BerkeleyAlgorithm {
```

```
    private static final int NUM_CLIENTS = 5; // Define number of clients

    // ---- Time Server Class ----
    static class TimeServer {
        public static void main(String[] args) {
            try {
                ServerSocket serverSocket = new ServerSocket(8080);
                System.out.println("Time server started. Listening on port 8080...");

                List<Long> clientAdjustments = new ArrayList<>();
                List<Socket> clientSockets = new ArrayList<>();
                int completedClients = 0;

                while (completedClients < NUM_CLIENTS) {
                    Socket clientSocket = serverSocket.accept();
                    clientSockets.add(clientSocket);

                    long currentTime = System.currentTimeMillis();
                    PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
                    out.println(currentTime); // Send current time to client

                    BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
                    long adjustment = Long.parseLong(in.readLine()); // Receive time difference from client
                    clientAdjustments.add(adjustment);

                    System.out.println("Time server: Received clock adjustment from client.");
                    completedClients++;
                }

                // Calculate average time adjustment
```

```
            long totalAdjustment = 0;
            for (long adjustment : clientAdjustments) {
                totalAdjustment += adjustment;
            }
            long averageAdjustment = totalAdjustment / NUM_CLIENTS;
            long serverTime = System.currentTimeMillis() + averageAdjustment;
            SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss.SSS");
            String adjustedTime = sdf.format(new Date(serverTime));

            System.out.println("Time server: Average clock adjustment calculated.");
            System.out.println("Server adjusted time: " + adjustedTime + "\n");

            // Send adjusted time to all clients
            for (Socket clientSocket : clientSockets) {
                PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
                out.println(averageAdjustment);
                out.close();
                clientSocket.close();
            }

            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

// ---- Time Client Class ----
static class TimeClient {
    public static void main(String[] args) {
        try {
```

```
            Socket socket = new Socket("localhost", 8080);
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

            // Receive server time
            long serverTime = Long.parseLong(in.readLine());
            long localTime = System.currentTimeMillis();
            long timeDifference = localTime - serverTime; // Calculate offset

            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            out.println(timeDifference); // Send offset to server

            // Receive adjustment from server
            long adjustment = Long.parseLong(in.readLine());
            long adjustedTime = localTime - adjustment;

            SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss.SSS");
            String adjustedTimeStr = sdf.format(new Date(adjustedTime));

            System.out.println("Client: Adjusted local time -> " + adjustedTimeStr);

            in.close();
            out.close();
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

// ---- Main Method to Start Server and Clients ----
```

```
    public static void main(String[] args) {
        Thread serverThread = new Thread(() -> TimeServer.main(null));
        serverThread.start();

        try {
            Thread.sleep(1000); // Wait for server to start
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        for (int i = 0; i < NUM_CLIENTS; i++) {
            new Thread(() -> TimeClient.main(null)).start();
            try {
                Thread.sleep(1000); // Stagger client connections
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        try {
            Thread.sleep(10000); // Wait for all clients to finish
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("All clients have completed their work. Program terminated.");
        System.exit(0);
    }
}
```

Commands:

Javac BerkeleyAlgorithm.java

Java BerkeleyAlgorithm

Expt3 Code

ChatClientInterface.java

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

public interface ChatClientInterface extends Remote {
   void receiveMessage(String message) throws RemoteException;
}
```

ChatClient.java

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.Scanner;

public class ChatClient extends UnicastRemoteObject implements ChatClientInterface {
   private static ChatInterface chat;

   protected ChatClient() throws RemoteException {
      super();
   }

   @Override
   public void receiveMessage(String message) throws RemoteException {
      System.out.println("New message: " + message);
   }

   public static void main(String[] args) {
      try {
         chat = (ChatInterface) Naming.lookup("rmi://localhost/ChatServer");
```

```
            ChatClient client = new ChatClient();

            // Register client with the server
            chat.registerClient(client);

            Scanner scanner = new Scanner(System.in);
            System.out.println("Enter messages (type 'exit' to quit):");

            while (true) {
                System.out.print("> ");
                String message = scanner.nextLine();
                if (message.equalsIgnoreCase("exit")) break;

                chat.sendMessage(message);
            }

            scanner.close();
        } catch (Exception e) {
            System.out.println("Client failed: " + e);
        }
    }
}
```

ChatInterface.java

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

public interface ChatInterface extends Remote {
    void sendMessage(String message) throws RemoteException;
    List<String> getMessages() throws RemoteException;
```

```
    void registerClient(ChatClientInterface client) throws RemoteException;
}
```

ChatServer.java

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
import java.util.List;

public class ChatServer extends UnicastRemoteObject implements ChatInterface {
    private List<String> messages;
    private List<ChatClientInterface> clients;

    protected ChatServer() throws RemoteException {
        super();
        messages = new ArrayList<>();
        clients = new ArrayList<>();
    }

    @Override
    public void sendMessage(String message) throws RemoteException {
        messages.add(message);
        System.out.println("Received: " + message);

        // Notify all clients
        for (ChatClientInterface client : clients) {
            client.receiveMessage(message);
        }
    }

    @Override
```

```
    public List<String> getMessages() throws RemoteException {
        return messages;
    }

    @Override
    public void registerClient(ChatClientInterface client) throws RemoteException {
        clients.add(client);
    }

    public static void main(String[] args) {
        try {
            ChatServer server = new ChatServer();
            Naming.rebind("rmi://localhost/ChatServer", server);
            System.out.println("Chat Server is running...");
        } catch (Exception e) {
            System.out.println("Server failed: " + e);
        }
    }
}
```

Commands:

compile all files (javac *.java)

Start-Process rmiregistry (or for linux os “rmiregistry &”)

java ChatServer

java ChatClient (per client)

Expt2 Code

1. 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;
}
```

2. 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...");
}
}
```

3. 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();
}
}
}
```

Commands:

compile all files as usual (javac *.java)

java RMI_Server.java

java RMI_Client.java

Expt1 Code

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();
    }
}
```

Commands:
javac *.java

Java producer

Java consumer