**Experiment 5**

**1. Implementation of Semaphores**

**Program:**

import java.util.concurrent.locks.\*;

public class Main {

public static void main(String[] args) {

Semaphore binarySemaphore = new Semaphore(1);

Semaphore mutex = new Semaphore(1);

Semaphore workerSemaphore = new Semaphore(0);

for (int i = 0; i < 5; i++) {

Thread thread = new Thread(new Worker(binarySemaphore, mutex, workerSemaphore, i));

thread.start();

}

}

public static class Worker implements Runnable {

private final Semaphore binarySemaphore;

private final Semaphore mutex;

private final Semaphore workerSemaphore;

private final int id;

private static int previousId = 0;

public Worker(Semaphore binarySemaphore, Semaphore mutex, Semaphore workerSemaphore, int id) {

this.binarySemaphore = binarySemaphore;

this.mutex = mutex;

this.workerSemaphore = workerSemaphore;

this.id = id;

}

@Override

public void run() {

try {

mutex.acquire(); // Acquire mutex to update previousId

if (id != previousId) {

mutex.release();

workerSemaphore.acquire();

} else {

mutex.release();

}

System.out.println("Worker " + id + " is trying to acquire permit.");

binarySemaphore.acquire();

System.out.println("Worker " + id + " has acquired permit.");

Thread.sleep(2000);

System.out.println("Worker " + id + " has released permit.");

binarySemaphore.release();

workerSemaphore.release();

previousId++;

if (previousId == 5) {

previousId = 0;

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

class Semaphore {

private int permits;

private final Lock lock;

private final Condition condition;

public Semaphore(int permits) {

this.permits = permits;

this.lock = new ReentrantLock();

this.condition = lock.newCondition();

}

public void acquire() throws InterruptedException {

lock.lock();

try {

while (permits == 0) {

condition.await();

}

permits--;

} finally {

lock.unlock();

}

}

public void release() {

lock.lock();

try {

permits++;

condition.signal();

} finally {

lock.unlock();

}

}

}

**Output:**

Worker 0 is trying to acquire permit.

Worker 0 has acquired permit.

Worker 0 has released permit.

Worker 2 is trying to acquire permit.

Worker 2 has acquired permit.

Worker 2 has released permit.

Worker 3 is trying to acquire permit.

Worker 3 has acquired permit.

Worker 3 has released permit.

Worker 1 is trying to acquire permit.

Worker 1 has acquired permit.

Worker 1 has released permit.

Worker 4 is trying to acquire permit.

Worker 4 has acquired permit.

Worker 4 has released permit.

**Implementation of InterProcess Communication:**

**Using Sockets:**

**1.Server side:**

import java.io.\*;

import java.net.\*;

public class Server {

public static void main(String[] args) {

try {

ServerSocket serverSocket = new ServerSocket(5000);

System.out.println("Server started. Waiting for client...");

Socket socket = serverSocket.accept();

System.out.println("Client connected.");

BufferedReader inputFromClient = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter outputToClient = new PrintWriter(socket.getOutputStream(), true);

String messageFromClient = inputFromClient.readLine();

System.out.println("Message from client: " + messageFromClient);

outputToClient.println("Message received by server: " + messageFromClient);

inputFromClient.close();

outputToClient.close();

socket.close();

serverSocket.close();

} catch (IOException e) {

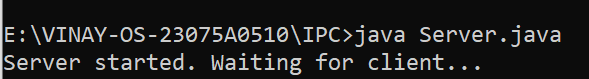
e.printStackTrace();

}

}

}

**Output:**



**2.Client Side:**

import java.io.\*;

import java.net.\*;

public class Client {

public static void main(String[] args) {

try {

Socket socket = new Socket("localhost", 5000);

System.out.println("Connected to server.");

BufferedReader inputFromServer = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter outputToServer = new PrintWriter(socket.getOutputStream(), true);

outputToServer.println("Hello from client.");

String responseFromServer = inputFromServer.readLine();

System.out.println("Response from server: " + responseFromServer);

inputFromServer.close();

outputToServer.close();

socket.close();

} catch (IOException e) {

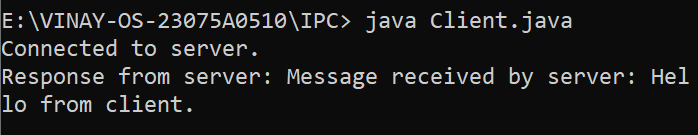
e.printStackTrace();

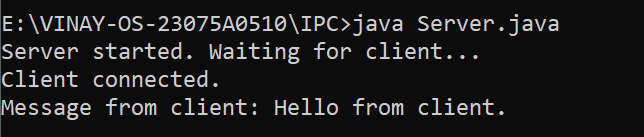
}

}

}

**Output:**





**Dining Philosophers problem:**

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

class Philosopher extends Thread {

private final Lock leftChopstick;

private final Lock rightChopstick;

private final int id;

public Philosopher(int id, Lock leftChopstick, Lock rightChopstick) {

this.id = id;

this.leftChopstick = leftChopstick;

this.rightChopstick = rightChopstick;

}

public void run() {

while (true) {

think();

eat();

}

}

private void think() {

System.out.println("Philosopher " + (id+1) + " is thinking");

try {

Thread.sleep((long) (Math.random() \* 1000));

} catch (InterruptedException e) {

e.printStackTrace();

}

}

private void eat() {

leftChopstick.lock();

rightChopstick.lock();

System.out.println("Philosopher " + (id+1) + " is eating");

try {

Thread.sleep((long) (Math.random() \* 1000));

} catch (InterruptedException e) {

e.printStackTrace();

} finally {

leftChopstick.unlock();

rightChopstick.unlock();

}

}

}

class DiningPhilosophers {

public static void main(String[] args) {

int numPhilosophers = 5;

Lock[] chopsticks = new Lock[numPhilosophers];

Philosopher[] philosophers = new Philosopher[numPhilosophers];

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

chopsticks[i] = new ReentrantLock();

}

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

philosophers[i] = new Philosopher(i, chopsticks[i], chopsticks[(i + 1) % numPhilosophers]);

philosophers[i].start();

}

}

}

**Output:**

Philosopher 4 is thinking

Philosopher 5 is thinking

Philosopher 3 is thinking

Philosopher 1 is thinking

Philosopher 2 is thinking

Philosopher 2 is eating

Philosopher 2 is thinking

Philosopher 2 is eating

Philosopher 4 is eating

Philosopher 2 is thinking

Philosopher 1 is eating

Philosopher 3 is eating

Philosopher 4 is thinking

Philosopher 3 is thinking

Philosopher 3 is eating

Philosopher 1 is thinking

Philosopher 5 is eating

Philosopher 5 is thinking

Philosopher 2 is eating

Philosopher 3 is thinking

Philosopher 4 is eating

Philosopher 4 is thinking

Philosopher 2 is thinking

Philosopher 1 is eating

Philosopher 3 is eating

Philosopher 3 is thinking

Philosopher 1 is thinking

Philosopher 5 is eating

Philosopher 5 is thinking

Philosopher 4 is eating

Philosopher 4 is thinking

Philosopher 3 is eating

Philosopher 5 is eating

Philosopher 3 is thinking

Philosopher 2 is eating

Philosopher 2 is thinking

**Bankers’s Algorithm:**

import java.util.Scanner;

public class BankersAlgorithm {

private int numProcesses;

private int numResources;

private int[][] max;

private int[][] allocation;

private int[] available;

private int[][] need;

private int[] safeSequence;

private void initializeValuesFromUser() {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of processes: ");

numProcesses = scanner.nextInt();

System.out.print("Enter the number of resources: ");

numResources = scanner.nextInt();

max = new int[numProcesses][numResources];

allocation = new int[numProcesses][numResources];

available = new int[numResources];

need = new int[numProcesses][numResources];

System.out.println("Enter the allocation matrix:");

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

for (int j = 0; j < numResources; j++) {

allocation[i][j] = scanner.nextInt();

}

}

System.out.println("Enter the maximum matrix:");

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

for (int j = 0; j < numResources; j++) {

max[i][j] = scanner.nextInt();

}

}

System.out.println("Enter the available resources:");

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

available[i] = scanner.nextInt();

}

}

private boolean isSafe() {

int[] work = new int[numResources];

boolean[] visited = new boolean[numProcesses];

safeSequence = new int[numProcesses];

int count = 0;

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

work[i] = available[i];

}

while (count < numProcesses) {

boolean flag = false;

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

if (!visited[i]) {

int j;

for (j = 0; j < numResources; j++) {

if (need[i][j] > work[j]) {

break;

}

}

if (j == numResources) {

safeSequence[count++] = i;

visited[i] = true;

flag = true;

for (j = 0; j < numResources; j++) {

work[j] += allocation[i][j];

}

}

}

}

if (!flag) {

break;

}

}

return count == numProcesses;

}

private void printUnsafeSequence() {

System.out.println("Unsafe sequence:");

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

if (safeSequence[i] == 0) {

System.out.print("P" + (safeSequence[i] + 1));

} else {

System.out.print(" -> P" + (safeSequence[i] + 1));

}

}

System.out.println();

}

private void calculateNeed() {

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

for (int j = 0; j < numResources; j++) {

need[i][j] = max[i][j] - allocation[i][j];

}

}

}

private void printNeedMatrix() {

System.out.println("Need Matrix:");

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

for (int j = 0; j < numResources; j++) {

System.out.print(need[i][j] + " ");

}

System.out.println();

}

}

public static void main(String[] args) {

BankersAlgorithm bankersAlgorithm = new BankersAlgorithm();

bankersAlgorithm.initializeValuesFromUser();

bankersAlgorithm.calculateNeed();

bankersAlgorithm.printNeedMatrix();

if (bankersAlgorithm.isSafe()) {

System.out.println("System is in safe state.");

System.out.println("Safe sequence:");

for (int i = 0; i < bankersAlgorithm.numProcesses; i++) {

if (i != 0) {

System.out.print(" -> ");

}

System.out.print("P" + (bankersAlgorithm.safeSequence[i] + 1));

}

System.out.println();

} else {

System.out.println("System is in unsafe state.");

bankersAlgorithm.printUnsafeSequence();

}

}

}

**Output:**

**java -cp /tmp/1CXbqLiVCS/BankersAlgorithm**

**Enter the number of processes: 5**

**Enter the number of resources: 3**

**Enter the allocation matrix:**

**0 1 0**

**2 0 0**

**3 0 2**

**2 1 1**

**0 0 2**

**Enter the maximum matrix:**

**7 5 3**

**3 2 2**

**9 0 2**

**2 2 2**

**4 3 3**

**Enter the available resources:**

**3 3 2**

**Need Matrix:**

**7 4 3**

**1 2 2**

**6 0 0**

**0 1 1**

**4 3 1**

**System is in safe state.**

**Safe sequence:**

**P2 -> P4 -> P5 -> P1 -> P3**

**=== Code Execution Successful ===**