Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Information Technology Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** IV **Course:** IT206 Operating Systems

Name: Amal Thundiyil

UID: 2020400066

Class: IT (Batch-D)

Experiment No.: 8

Title: Deadlock

Aim: Write a multithreaded program for preventing race conditions and deadlock avoidance for the banker's algorithm as follows. Panashikar supplies sweets, snacks and Indian food traces. Customer thread will request the particular product. The shop will release and grant the product only if it leaves the system in safe state. A request that leaves the federation in an unsafe state will be denied. Take the Allocation and Available from the user. Print the Allocation, Max and Available. Find and print the Need. Find the safe sequence if any and print it and tell whether federation is in safe state or not. Take a request from a user thread and tell whether this request will be granted immediately or not.

Code:

```
import java.util.concurrent.locks.ReentrantLock;
import static java.lang.System.out;
import java.io.FileInputStream;
import java.util.*;

public class Main {
    static Scanner fs;
    static int[][] allocation, max, need;
    static List<Integer> safeSequence = new ArrayList<>();
    static HashSet<Integer> safeSet = new HashSet<>();
    static boolean safe = true;
    static int[] available, work, request;
    static int requestingThread, threads, products;
```

```
static volatile ReentrantLock rl;

static class ProcessExecution implements Runnable {
    int t_no;

    public ProcessExecution(int t_no) {
        this.t_no = t_no;
    }

    @Override
    public void run() {
        rl.lock();
        try {
            out.println("Acquired the lock: " + t_no);
            try {
                Thread.sleep(1000);
            } catch (Exception e) {
            }
            out.println("Current Available: ");
            for (int i = 0; i < allocation[t_no].length; i++) {
                work[i] += allocation[t_no][i];
            }
            out.println(Arrays.toString(work));
        } finally {
            rl.unlock();
        }
```

```
    }
}

public static int findThreadWithMinNeed() {
    for (int i = 0; i < need.length; i++) {
        if (safeSet.contains(i)) {
            continue;
        }
        boolean min = true;
        for (int j = 0; j < need[i].length; j++) {
            if (need[i][j] > work[j]) {
                min = false;
                break;
            }
        }
        if (min == true) {
            safeSet.add(i);
            safeSequence.add(i);
            return i;
        }
    }
    return -1;
}

public static void calcSafeSequence() {
    work = Arrays.copyOf(available, available.length);
```

```
    for (int i = 0; i < need.length; i++) {
        int threadNo = findThreadWithMinNeed();
        if (threadNo == -1) {
            System.out.println("No safe sequence found!");
            safe = false;
            return;
        }
        for (int j = 0; j < allocation[i].length; j++) {
            work[j] += allocation[threadNo][j];
        }
    }
}

public static void calcNeed() {
    need = new int[allocation.length][allocation[0].length];
    for (int i = 0; i < allocation.length; i++) {
        for (int j = 0; j < allocation[i].length; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
}

public static boolean grantRequest() {
    if (!safe) {
        return false;
    }
```

```
    for (int i = 0; i < available.length; i++) {
        if (request[i] > available[i]
                || request[i] > need[requestingThread][i]) {
            return false;
        }
    }
    return true;
}

public static void display() {
    System.out.println("");
    System.out.println("Available: " + Arrays.toString(available));

    System.out.println("Allocation\tMax\t\tNeed");
    for (int i = 0; i < allocation.length; i++) {
        for (int j = 0; j < allocation[i].length; j++) {
            System.out.print(allocation[i][j] + " ");
        }
        System.out.print("\t\t");
        for (int j = 0; j < max[i].length; j++) {
            System.out.print(max[i][j] + " ");
        }
        System.out.print("\t\t");
        for (int j = 0; j < need[i].length; j++) {
            System.out.print(need[i][j] + " ");
        }
```

```
        System.out.println("");
    }
    if (safe) {
        System.out.println("Safe Sequence: " + safeSequence.toString());
    }
}

public static void banker() throws Exception {
    safeSequence.clear();
    safeSet.clear();
    calcNeed();
    calcSafeSequence();
    display();

    Thread th[] = new Thread[threads];
    ProcessExecution processes[] = new ProcessExecution[threads];
    work = Arrays.copyOf(available, products);
    if (!safe) {
        return;
    }
    for (int i = 0; i < threads; i++) {
        processes[i] = new ProcessExecution(safeSequence.get(i));
        th[i] = new Thread(processes[i]);
        th[i].setPriority(i + 1);
        th[i].start();
        th[i].join();
```

```
    }
}

public static void main(String args[]) throws Exception {
    System.setIn(new FileInputStream("./input.txt"));
    fs = new Scanner(System.in);
    rl = new ReentrantLock();

    threads = fs.nextInt();
    products = fs.nextInt();

    allocation = new int[threads][products];
    for (int i = 0; i < threads; i++) {
        for (int j = 0; j < products; j++) {
            allocation[i][j] = fs.nextInt();
        }
    }
    max = new int[threads][products];
    for (int i = 0; i < threads; i++) {
        for (int j = 0; j < products; j++) {
            max[i][j] = fs.nextInt();
        }
    }
    available = new int[products];
    for (int i = 0; i < products; i++) {
        available[i] = fs.nextInt();
```

```
        }
        banker();
        System.out.println("");
        request = new int[products];
        for (int i = 0; i < products; i++) {
            request[i] = fs.nextInt();
        }
        requestingThread = fs.nextInt();
        System.out.println("Request of thread " + requestingThread + " is " +
Arrays.toString(request));
        boolean grant = grantRequest();
        if (!grant) {
            System.out.println("Request cannot be granted.");
        } else {
            System.out.println("Request granted ...");
            for (int i = 0; i < products; i++) {
                allocation[requestingThread][i] += request[i];
            }
            for (int i = 0; i < products; i++) {
                available[i] -= request[i];
            }
            banker();
        }
        fs.close();
    }
}
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Information Technology Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** IV **Course:** IT206 Operating Systems

Output:

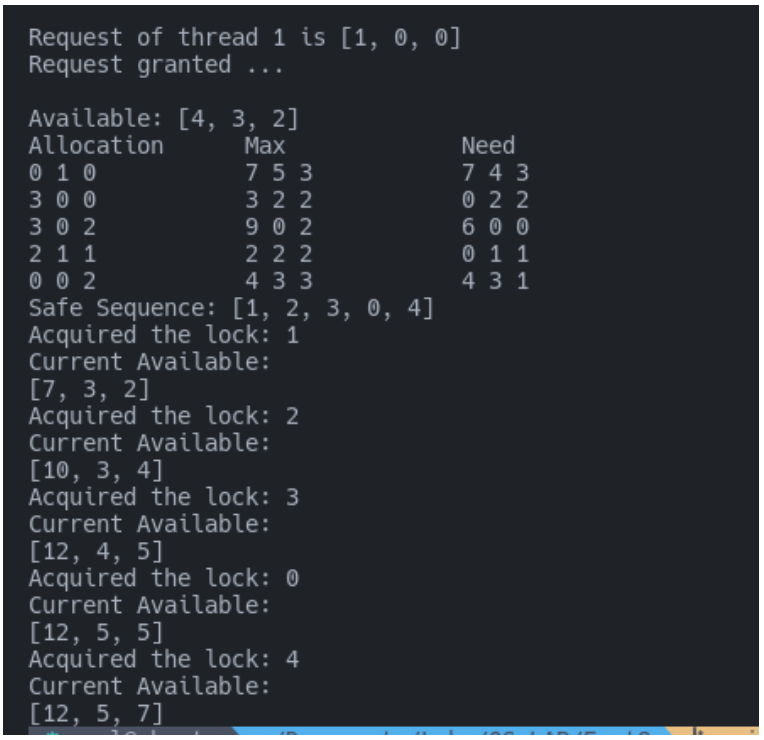
```
amal@ubuntu ~/Documents/Labs/OS_LAB/Expt8 main ± javac Main.java
amal@ubuntu ~/Documents/Labs/OS_LAB/Expt8 main ± java Main

Available: [5, 3, 2]
Allocation      Max             Need
0 1 0           7 5 3           7 4 3
2 0 0           3 2 2           1 2 2
3 0 2           9 0 2           6 0 0
2 1 1           2 2 2           0 1 1
0 0 2           4 3 3           4 3 1
Safe Sequence: [1, 2, 3, 0, 4]
Acquired the lock: 1
Current Available:
[7, 3, 2]
Acquired the lock: 2
Current Available:
[10, 3, 4]
Acquired the lock: 3
Current Available:
[12, 4, 5]
Acquired the lock: 0
Current Available:
[12, 5, 5]
Acquired the lock: 4
Current Available:
[12, 5, 7]
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Information Technology Department

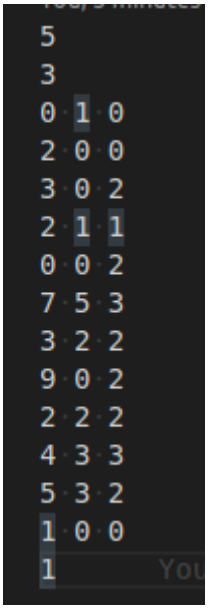
Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** IV **Course:** IT206 Operating Systems



```
Request of thread 1 is [1, 0, 0]
Request granted ...

Available: [4, 3, 2]
Allocation      Max             Need
0 1 0           7 5 3           7 4 3
3 0 0           3 2 2           0 2 2
3 0 2           9 0 2           6 0 0
2 1 1           2 2 2           0 1 1
0 0 2           4 3 3           4 3 1
Safe Sequence: [1, 2, 3, 0, 4]
Acquired the lock: 1
Current Available:
[7, 3, 2]
Acquired the lock: 2
Current Available:
[10, 3, 4]
Acquired the lock: 3
Current Available:
[12, 4, 5]
Acquired the lock: 0
Current Available:
[12, 5, 5]
Acquired the lock: 4
Current Available:
[12, 5, 7]
```

Input (from file - input.txt)



```
5
3
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
5 3 2
1 0 0
1
```