**Contents**

**USCSP301 – USCS303: Operating System (OS) Practical – 06……………………………………2**

**Practical – 06: Banker’s Algorithm…………………………………………………………………2**

[**Practical Date: 21th August 2021 2**](#_Toc80552253)

[**Practical Aim: Implement Banker’s algorithm using Java. 2**](#_Toc80552254)

[** Banker’s Algorithm: 2**](#_Toc80552255)

[** Banker’s Algorithm – how it works? 2**](#_Toc80552256)

[** Data Structures required in Banker’s Algorithm 2**](#_Toc80552257)

[** Algorithm (Safety and Resource-Allocation) 3**](#_Toc80552258)

[** Solved Example 4**](#_Toc80552259)

[** Question-01 4**](#_Toc80552260)

[** Question 5**](#_Toc80552261)

[** Implementation 7**](#_Toc80552262)

[** Input of Question – 01 11**](#_Toc80552263)

[** Output of Question – 01 11**](#_Toc80552264)

[** Input Of Question – 02 12**](#_Toc80552265)

[** Output Of Question – 02 12**](#_Toc80552266)

[** Input Of Question – 03 13**](#_Toc80552267)

[** Output Of Question – 03 13**](#_Toc80552268)

[** Sample Output Of Question – 01 14**](#_Toc80552269)

[** Sample Output Of Question – 02 14**](#_Toc80552270)

**USCSP301 – USCS303: Operating System (OS) Practical – 06**

**Practical – 06: Banker’s Algorithm**

# Practical Date: 21th August 2021

# Practical Aim: Implement Banker’s algorithm using Java.

# Banker’s Algorithm:

* The **resource-allocation-graph algorithm** is not applicable to a resource allocation system with multiple instances of each resource type.
* The **deadlock-avoidance algorithm** that we describe next is applicable to such a system but is less efficient than the resource-algorithm graph scheme.
* This algorithm is commonly known as the **banker’s algorithm.**
* Banker’s algorithm is a **deadlock avoidance algorithm.**
* It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not.
* The name was chosen because the algorithm could be used in a banking system to ensure that the bank never allocated its available cash in such a way that it could no longer satisfy the needs of all its customers.

# Banker’s Algorithm – how it works?

* Consider there are n account holders in a bank and the sum of the money in all of their accounts is S.
* Every time a loan has to be granted by the bank, it subtracts the **loan amount** from the **total money** the bank has.
* Then it checks if that difference is greater than S.
* It is done because, only then, the bank would have enough money even if all the n account holders draw all their money at once.
* When a new thread enters the system, it must declare the maximum number of instances of each resource type it may need.
* This number may not exceed the total number of resources in the system.
* When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state.
* If it will, the resources are allocated; otherwise, the thread must wait until some other thread releases enough resources.

# Data Structures required in Banker’s Algorithm

* Several data structures must be maintained to implement the banker’s algorithm.
* These data structures encode the state of the resource-allocation system.
* We need the following data structures, where n is the number of threads in the system and m is the number of resource types:
* **Available:** A vector of length m indicates the number of available resources of each type. If Available[j] equals k, then k instances of resource type Rj are available.
* **Max:** An n x m matrix defines the maximum demand of each thread. If **Max[i][j]** equals k, then thread Ti may request at most k instances of resource type Rj.
* **Allocation:** An n x m matrix defines the number of resources of each type currently allocated to each thread. If **Allocation[i][j]** equals k, then thread Ti is currently allocated k instances of resource type Rj.
* **Need:** An n x m matrix indicates the remaining resource need of each thread. If **Need[i][j]** equals k, then thread Ti may need k more instances of resource type Rj to complete its task.

**Need[i][j] = Max[i][j] – Allocation[i][j]**

# Algorithm (Safety and Resource-Allocation)

* **Safety Algorithm:**
* **Step 1:** Let **Work** and **Finish** be vectors of length m and n, respectively. Initialize **Work=Available** and **Finish[i]-false** for i=0,1,…,n-1.
* **Step 2:** Find an index I such that both
* **Step 2.1: Finish[i]==false**
* **Step 2.2: Needi<** Work

If no such i exists, go to **Step 4**

* **Step 3: Work = Work + Allocationi**

**Finish[i] = true**

Go to **Step 2.**

* **Step 4:** If **Finish[i]** **== true** for all i, then the system is in a safe state
* **Resource-Request Algorithm**
* Let **Requesti** be the request vector for thread Ti.
* If **Requesti[j] == k**, then thread Ti wants k instances of resource type Rj.
* When a request for resources is made by thread Ti, the following actions are taken:
* **Step 1:** If **Requesti < Needi**, go to **Step 2.** Otherwise, raise an error condition, since the thread has exceeded its maximum claim.
* **Step 2:** If **Requesti** < Available, go to **Step 3.** Otherwise Ti must wait since the resources are not available.
* **Step 3:** Have the system pretend to have allocated the requested resources to thread Ti by modifying the state as follows:

**Available = Available – Requesti**

**Allocationi = Allocationi + Requesti**

**Needi = Needi – Requesti**

If the resulting resource-allocation state is safe, The transaction is completed, and thread Ti is allocated its fresources. However, it the

new state is unsafe, then Ti must wait for **Requesti** and the old

resource-allocation state is restored.

# Solved Example

# Question-01

* Write a Java program that implement the banker’s algorithm
* Consider a system with five threads T0 through T4 and three resource types A, B, C. Resource type A has ten instances, resource type B has five instances, resource type C has seven instances. Suppose that the following snapshot represents the current state of the system:

|  |  |  |  |
| --- | --- | --- | --- |
| **Threads** | **Allocation** | **Max** | **Available** |
|  | **A B C** | **A B C** | **A B C** |
| T0 | 0 1 0 | 7 5 3 | 3 3 2 |
| T1 | 2 0 0 | 3 2 2 |  |
| T3 | 3 0 2 | 9 0 2 |  |
| T4 | 2 1 1 | 2 2 2 |  |
| T5 | 0 0 2 | 4 3 3 |  |

**Solution:**

Need Matrix = Max – Allocation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threads** | **Allocation** | **Max** | **Available** | **Need** |
|  | **A B C** | **A B C** | **A B C** | **A B C** |
| T0 | 0 1 0 | 7 5 3 | 3 3 2 | 7 4 3 |
| T1 | 2 0 0 | 3 2 2 |  | 1 2 2 |
| T2 | 3 0 2 | 9 0 2 |  | 6 0 0 |
| T3 | 2 1 1 | 2 2 2 |  | 0 1 1 |
| T4 | 0 0 2 | 4 3 3 |  | 4 3 1 |

We claim that the system is currently in a **safe state**.

Indeed, the sequence <**T1, T3, T4, T0, T2**>satisfies the safety criteria.

# Question

**Question – 02**

* Consider the following system:

|  |  |  |  |
| --- | --- | --- | --- |
| **Processes** | **Allocation** | **Max** | **Available** |
|  | **A B C** | **A B C** | **A B C** |
| P0 | 1 1 2 | 4 3 3 | 2 1 0 |
| P1 | 2 1 2 | 3 2 2 |  |
| P2 | 4 0 1 | 9 0 2 |  |
| P3 | 0 2 0 | 7 5 3 |  |
| P4 | 1 1 2 | 1 1 2 |  |

Calculate the content of the need matrix?

Check if the system is in a safe state?

**Solution:**

Need Matrix = Max – Allocation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Processes** | **Allocation** | **Max** | **Available** | **Need** |
|  | **A B C** | **A B C** | **A B C** | **A B C** |
| P0 | 1 1 2 | 4 3 3 | 2 1 0 | 3 2 1 |
| P1 | 2 1 2 | 3 2 2 |  | 1 1 0 |
| P2 | 4 0 1 | 9 0 2 |  | 5 0 1 |
| P3 | 0 2 0 | 7 5 3 |  | 7 3 3 |
| P4 | 1 1 2 | 1 1 2 |  | 0 0 0 |

We claim that the system is currently in a **safe state.**

Indeed, the sequence **< P1, P4, P0, P2, P3>** satisfies the safety criteria.

**Question – 03**

* Consider the following example containing five processes and 4 types of resources:
* Calculate the Need Matrix and the sequence of safety allocation?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Given Matrices** | | | | | | | |  | | | | |
|  | **Allocation Matrix**  **(N0 of the resources By a process)** | | | | **Max Matrix**  **Max resources that may be used by a process** | | | | **Available Matrix**  **Not Allocated Resources** | | | | |
|  | **A** | **B** | **C** | **D** | **A** | **B** | **C** | **D** | **A** | **B** | **C** | **D** |
| P0 | 0 | 1 | 1 | 0 | 0 | 2 | 1 | 0 | 1 | 5 | 2 | 0 |
| P1 | 1 | 2 | 3 | 1 | 1 | 6 | 5 | 2 |  |  |  |  |
| P2 | 1 | 3 | 6 | 5 | 2 | 3 | 6 | 6 |  |  |  |  |
| P3 | 0 | 6 | 3 | 2 | 0 | 6 | 5 | 2 |  |  |  |  |
| P4 | 0 | 0 | 1 | 4 | 0 | 6 | 5 | 6 |  |  |  |  |

**Solution:**

Need Matrix = Max – Allocation

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Given Matrices** | | | | | | | |  | | | |  | | |  | |
|  | **Allocation Matrix**  **(N0 of the resources By a process)** | | | | **Max Matrix**  **Max resources that may be used by a process** | | | | **Available Matrix**  **Not Allocated Resources** | | | | **Need** | | |  | |
|  | **A** | **B** | **C** | **D** | **A** | **B** | **C** | **D** | **A** | **B** | **C** | **D** | **A** | **B** | **C** | **D** |
| P0 | 0 | 1 | 1 | 0 | 0 | 2 | 1 | 0 | 1 | 5 | 2 | 0 | 0 | 1 | 0 | 0 |
| P1 | 1 | 2 | 3 | 1 | 1 | 6 | 5 | 2 |  |  |  |  | 0 | 4 | 2 | 1 |
| P2 | 1 | 3 | 6 | 5 | 2 | 3 | 6 | 6 |  |  |  |  | 1 | 0 | 0 | 1 |
| P3 | 0 | 6 | 3 | 2 | 0 | 6 | 5 | 2 |  |  |  |  | 0 | 0 | 2 | 0 |
| P4 | 0 | 0 | 1 | 4 | 0 | 6 | 5 | 6 |  |  |  |  | 0 | 6 | 4 | 2 |

We claim that the system is currently in a **safe state.**

Indeed, the sequence **< P0, P3, P4, P1, P2>** satisfies the safety criteria.

**Implement Banker’s Algorithm in Java**

# Implementation

**Filename: P6\_BankersAlgo\_YP.java**

**//**Name: Yash Anand Parab

**//**Batch: B1

//PRN: 2020016400922513

//Date: 21th August 2021

//Prac-06: Banker’s Algorithm

import java.util.Scanner;

public class P6\_BankersAlgo\_YP{

private int need[][],allocate[][],max[][],avail[][],np,nr;

private void input(){

Scanner sc=new Scanner(System.in);

System.out.print("Enter no.of processes: ");

np=sc.nextInt(); //no. of processes

System.out.print("Enter no. of processes: ");

nr=sc.nextInt();//no.of rescources

need=new int[np][nr];//initializing arrays

max=new int[np][nr];

allocate=new int[np][nr];

avail=new int[1][nr];

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

System.out.print("Enter allocaton matrix for process P"+i+":");

for(int j=0;j<nr;j++)

allocate[i][j]=sc.nextInt();//allocation matrix

}

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

System.out.print("Enter maximum matrix for process P"+i+":");

for(int j=0;j<nr;j++)

max[i][j]=sc.nextInt();//max matrix

}

System.out.print("Enter available matrix for process P0:");

for(int j=0;j<nr;j++)

avail[0][j]=sc.nextInt(); //available matrix

sc.close();

}//input() ends

private int[][] calc\_need(){

for(int i=0;i<np;i++)

for(int j=0;j<nr;j++)//calculating need matrix

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

return need;

}//calc\_need()ends

private boolean check(int i){

//checking if all resources for ith process can be allocated

for(int j=0;j<nr;j++)

if(avail[0][j]<need[i][j])

return false;

return true;

} //check() ends

public void isSafe(){

input();

calc\_need();

boolean done[]=new boolean[np];

int j=0;

//printing Need Matrix

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

for(int a=0;a<np;a++){

for(int b=0;b<nr;b++){

System.out.print(need[a][b]+"\t");

}

System.out.println();

}

System.out.println("Allocated process:");

while(j<np){// until all process allocated

boolean allocated=false;

for(int i=0;i<np;i++)

if(!done[i] && check(i)){//trying to allocate

for(int k=0;k<nr;k++)

avail[0][k]=avail[0][k]-need[i][k]+max[i][k];

System.out.print("P"+i+">");

allocated=done[i]=true;

j++;

}//if block

if(!allocated)

break; //if no allocation

}//while ends

if(j==np)//if all processes are allocated

System.out.println("\nSafely allocated");

else

System.out.println("All/Remaining process can\'t be allocated safely");

}//isSafe()ends

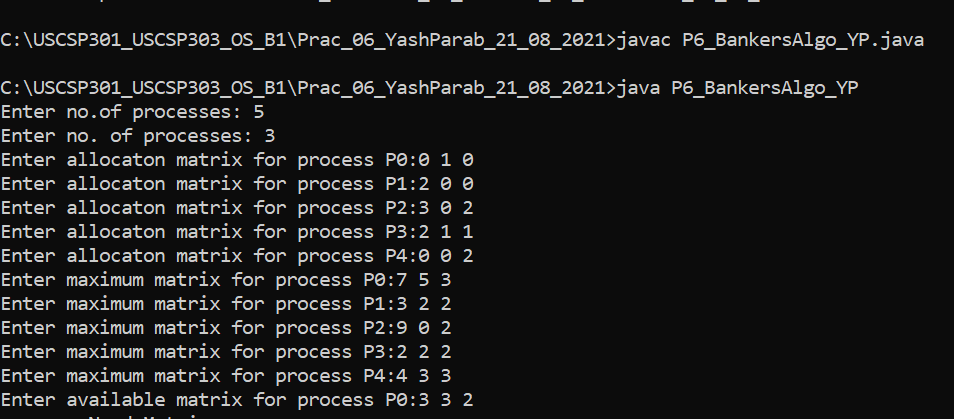
public static void main(String[]args){

new P6\_BankersAlgo\_YP().isSafe();

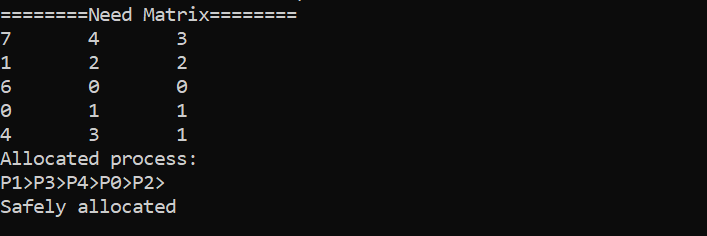
}

}//class ends

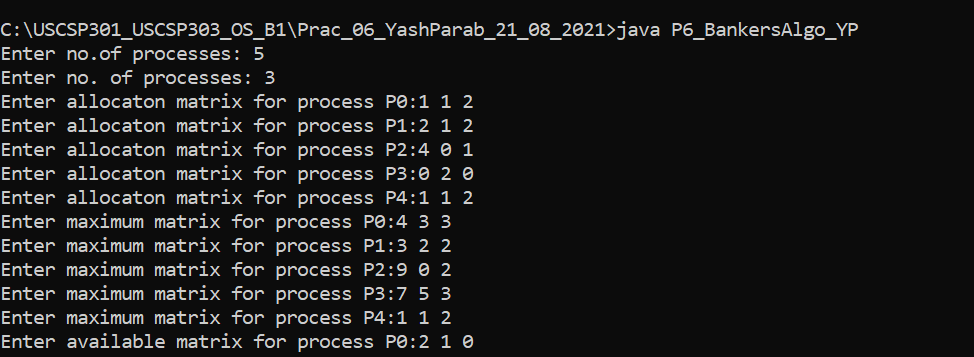
# Input of Question – 01



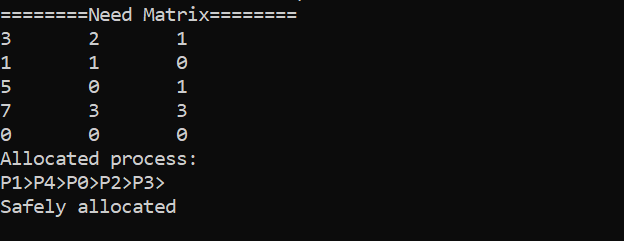
# Output of Question – 01



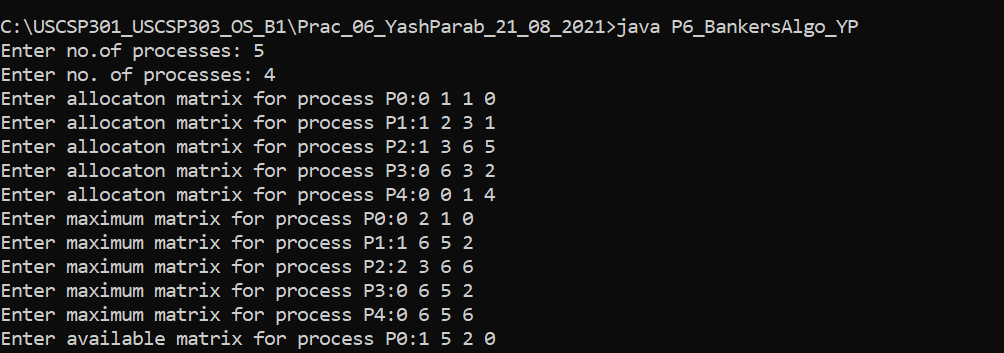
# Input Of Question – 02



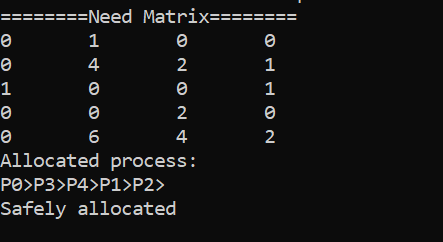
# Output Of Question – 02



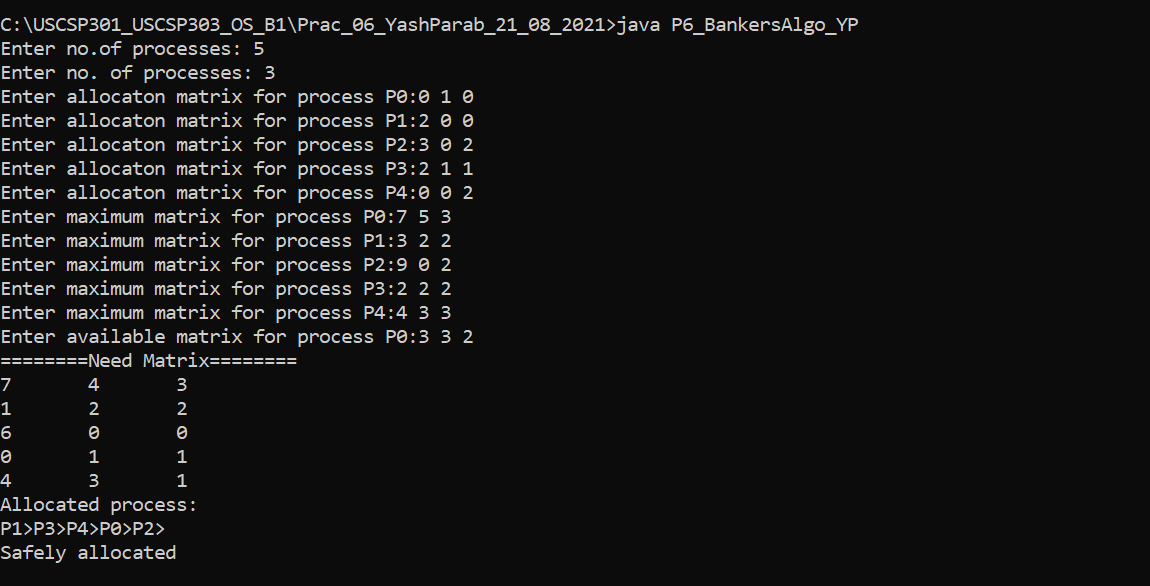
# Input Of Question – 03



# Output Of Question – 03



# Sample Output Of Question – 01



# Sample Output Of Question – 02

