Chandibai Himathmal Mansukhani College

Batch:b2 Name: AbhishekNikam

USCS 3P01: USCS303-Operating System(OS) date-20/08/2020

Practical:06

CONTENTS

USCS3P01: USCS303-Operating System (OS)

Aim: Blanker's Algorithm **Contents:**

For the banker algorithm to operate each process has to a Priority specify its maximum requirement of resources.

Process:

One can also determine whether a process request for allocation of resources be safely granted immediately.

Prior Knowledge:

Date structure used in banker algorithm .

Safety algorithm and resource request algorithm.

Banker's Algorithm:

1) The resource-allocation -graph algorithm is not applicable toa resource allocation system with instences of each resources type.
2) The deadlock -Avoidance algorithm that we describe next is applicable to such a system but is less efficient than the resources -allocation graph scheme.
3) This algorithm is commonly know as the banker's algorithm.
4) Banker's algorithm is a deadlock avoidance algorithm.
5) It is the name so because this algorithm is used in banking system to determine whether a loan can be granted or not
6) The name was choose because the algorithm could be used in banking system to ensure that the bank never are located its available cash in such a way that it would no longer satisfy the needs of its customer.

Banker's Algorithm -how it works:

1) Consider there are an account holders in a bank and the sum of the money in all of their account is S.

2) Every time a loan has to be granted by the bank it subtract the loan amount from the total money the bank has.

3) Then it's check if that different is greater than S.

4) It is done because only then the bank would have enough money if all the an account holder draw all their money At once.

5) When a new thread enter the system it must declare the maximum number of instance of each resource type that it may need

6) This number may not exceed the total number of a source in the system .

7) when a user request a set of resource the system must determine whether the allocation of these resources will leave the system in a safe state.

8) If it will , the resources are allocation ; otherwise the thread must wait until some other thread release enough resources.

Date Structures (Banker's Algorithm):

Available : A vector of length m indicate the number of available resources of each type. If Available[j] equals k, then k instance of resource type Rj are available.

Max: An n × m matrix defines the maximum demand of each thread . if Max[i][j] equals k, then thread Ti may request at most k instance of resources type Rj.

Allocation : An n × m matrix defines the number of resources of each type currently allocated to each thread. If Allocation[i][j] equals k, then thread k, then thread Ti is currently allocation k instance of resources type rj.

Need: An n × m matrix indicate the remaining resources need of each thread . if Need[i][j] equals k, then thread Ti may need k more instance of resources type Rj complete its task.

Need[i][j]=Max[i][j]-Allocation[i][j]

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: Finish[i]==false

Step 3: $Need_i \leq Work$

If no such I exists go to step 4.

Step 3 : Work = Work + Allocation;

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 :

1) Let Request be the request vector for thread Ti.
2) If $Request_i$ [j]==k , then thread Ti wants k instance of resources type Rj.
3) When a request for resources is made by thread Ti , the following actions are taken:

Step 1: If $Request_i \leq Need_i$ go to Step 2. Otherwise , raise an error condition , since the thread has exceeded its maximum claim.

Step 2 : if $Request_i \leq Available_i$ go to Step 3, Otherwise , Ti must wait, since the resources are not available.

Step 3: Heather system printed to had allocated the requested resource to third T_i by modify the state has follows;

$Available = Available – request_i$

$Allocation_i = Allocation_i + Request_i$

$Need_i = Need_i – Request_i$

if the resulting resource allocation state it safe the transaction is completed and the thread T_i is allocated its resources .

however if the new state is unsafe then T_i must wait a request and the old resource allocation state is restored.

Example 1: Consider a system with five Threads T_0 through T_4 and three resource type A ,B and C. resource type A has ten instance ,resource type B has file systems and resource type C has seven instance. suppose that the following is snapshot represent the current state of the system.

Threads	Allocation			Max			Avilable		
	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			
T2	3	0	2	9	0	2			
T3	2	1	1	2	2	2			
T4	0	0	2	4	3	3			

Need Matrix = Max-Allocation

Threads	Allocation			Max			Avilable			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 of current in safe state in the sequence <T1 ,T3 ,T4, T0 ,T2> satisfy the supplies criteria.

Example 2: Consider the following System.

Threads	Allocation			Max			Avilable		
	A	B	C	A	B	C	A	B	C
P0	1	1	1	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			

Solve:

Need Matrix = Max-Allocation

Threads	Allocation			Max			Avilable			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	1	1	1	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 of current in safe state in the sequence <P1 ,P4 ,P0, P2 ,P3> satisfy the supplies criteria.

Example 3: Consider the following example containing five processes and 4 types of resources :

	Allocation Matrix	Max Matrix	Available Matrix
	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	

Solve:

Need Matrix = Max-Allocation

	Allocation Matrix	Max Matrix	Available Matrix	Need Matrix
	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 of current in safe state in the sequence P0>P3>P4>P1>P2> satisfy the supplies criteria.

Implementation:

```
//Name: Abhishek Nikam
//Batch:B2
//PRN: 2020016400805951
//Date:20/8/2021
//Prac-06:Banker's Algorithm



import java.util.Scanner;


public class P6_BankersAlgo_PD{   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_PD().isSafe();

    }

}//class ends
```

Input question 1:

```
C:\Users\SD CONSULTANTS\OneDrive\Desktop\USCSP301_OS_B2\prac-06-pd>javac P6_BankersAlgo_PD.java

C:\Users\SD CONSULTANTS\OneDrive\Desktop\USCSP301_OS_B2\prac-06-pd>java P6_BankersAlgo_PD
Enter no.of processes: 5
Enter no. of processes: 3
Enter allocaton matrix for process P0:0 1 0
Enter allocaton matrix for process P1:2 0 0
Enter allocaton matrix for process P2:3 0 2
Enter allocaton matrix for process P3:2 1 1
Enter allocaton matrix for process P4:0 0 2
Enter maximum matrix for process P0:7 5 3
Enter maximum matrix for process P1:3 2 2
Enter maximum matrix for process P2:9 0 2
Enter maximum matrix for process P3:2 2 2
Enter maximum matrix for process P4:4 3 3
Enter available matrix for process P0:3 3 2
```

Input question 2:

```
C:\Users\SD CONSULTANTS\OneDrive\Desktop\USCSP301_OS_B2\prac-06-pd>java P6_BankersAlgo_PD
Enter no.of processes: 5
Enter no. of processes: 3
Enter allocaton matrix for process P0:1 1 2
Enter allocaton matrix for process P1:2 1 2
Enter allocaton matrix for process P2:4 0 1
Enter allocaton matrix for process P3:0 2 0
Enter allocaton matrix for process P4:1 1 2
Enter maximum matrix for process P0:4 3 3
Enter maximum matrix for process P1:3 2 2
Enter maximum matrix for process P2:9 0 2
Enter maximum matrix for process P3:7 5 3
Enter maximum matrix for process P4:1 1 2
Enter available matrix for process P0:2 1 0
```

Input question 3:

```
C:\Users\SD CONSULTANTS\OneDrive\Desktop\USCSP301_OS_B2\prac-06-pd>javac P6_BankersAlgo_PD.java

C:\Users\SD CONSULTANTS\OneDrive\Desktop\USCSP301_OS_B2\prac-06-pd>java P6_BankersAlgo_PD
Enter no.of processes: 5
Enter no. of processes: 4
Enter allocaton matrix for process P0:0 1 1 0
Enter allocaton matrix for process P1:1 2 3 1
Enter allocaton matrix for process P2:1 3 6 5
Enter allocaton matrix for process P3:0 6 3 2
Enter allocaton matrix for process P4:0 0 1 4
Enter maximum matrix for process P0:0 2 1 0
Enter maximum matrix for process P1:1 6 5 2
Enter maximum matrix for process P2:2 3 6 6
Enter maximum matrix for process P3:0 6 5 2
Enter maximum matrix for process P4:0 6 5 6
Enter available matrix for process P0:1 5 2 0
```

Output question 1:

```
========Need Matrix========
7       4       3
1       2       2
6       0       0
0       1       1
4       3       1
Allocated process:
P1>P3>P4>P0>P2>
Safely allocated

C:\Users\SD CONSULTANTS\OneDrive\Desktop\USCSP301_OS_B2\prac-06-pd>javac P6_BankersAlgo_PD.java
```

Output question 2:

```
========Need Matrix========
3       2       1
1       1       0
5       0       1
7       3       3
0       0       0
Allocated process:
P1>P4>P0>P2>P3>
Safely allocated

C:\Users\SD CONSULTANTS\OneDrive\Desktop\USCSP301_OS_B2\prac-06-pd>javac P6_BankersAlgo_PD.java
```

Output question 3:

```
========Need Matrix========
0       1       0       0
0       4       2       1
1       0       0       1
0       0       2       0
0       6       4       2
Allocated process:
P0>P3>P4>P1>P2>
Safely allocated
```

Sample Output question 1:

```
C:\Windows\System32\cmd.exe
  location: class P6_BankersAlgo_PD
1 error

C:\Users\SD CONSULTANTS\OneDrive\Desktop\USCSP301_OS_B2\prac-06-pd>javac P6_BankersAlgo_PD.java

C:\Users\SD CONSULTANTS\OneDrive\Desktop\USCSP301_OS_B2\prac-06-pd>java P6_BankersAlgo_PD
Enter no.of processes: 5
Enter no. of processes: 3
Enter allocaton matrix for process P0:0 1 0
Enter allocaton matrix for process P1:2 0 0
Enter allocaton matrix for process P2:3 0 2
Enter allocaton matrix for process P3:2 1 1
Enter allocaton matrix for process P4:0 0 2
Enter maximum matrix for process P0:7 5 3
Enter maximum matrix for process P1:3 2 2
Enter maximum matrix for process P2:9 0 2
Enter maximum matrix for process P3:2 2 2
Enter maximum matrix for process P4:4 3 3
Enter available matrix for process P0:3 3 2
========Need Matrix========
7       4       3
1       2       2
6       0       0
0       1       1
4       3       1
Allocated process:
P1>P3>P4>P0>P2>
Safely allocated

C:\Users\SD CONSULTANTS\OneDrive\Desktop\USCSP301_OS_B2\prac-06-pd>javac P6_BankersAlgo_PD.java
```

Sample Output question 2:

```
Select C:\Windows\System32\cmd.exe

C:\Users\SD CONSULTANTS\OneDrive\Desktop\USCSP301_OS_B2\prac-06-pd>java P6_BankersAlgo_PD
Enter no.of processes: 5
Enter no. of processes: 3
Enter allocaton matrix for process P0:1 1 2
Enter allocaton matrix for process P1:2 1 2
Enter allocaton matrix for process P2:4 0 1
Enter allocaton matrix for process P3:0 2 0
Enter allocaton matrix for process P4:1 1 2
Enter maximum matrix for process P0:4 3 3
Enter maximum matrix for process P1:3 2 2
Enter maximum matrix for process P2:9 0 2
Enter maximum matrix for process P3:7 5 3
Enter maximum matrix for process P4:1 1 2
Enter available matrix for process P0:2 1 0
========Need Matrix========
3       2       1
1       1       0
5       0       1
7       3       3
0       0       0
Allocated process:
P1>P4>P0>P2>P3>
Safely allocated

C:\Users\SD CONSULTANTS\OneDrive\Desktop\USCSP301_OS_B2\prac-06-pd>javac P6_BankersAlgo_PD.java
```

Sample Output question 3: