Student Name: Janmenjoy Singh
Student ID: 11903400 K18CJ
Email Address: mail.lionel11@gmail.com
GitHub Link:

QUESTION: 11

Reena's operating system uses an algorithm for deadlock avoidance to manage the allocation of resources say three namely A, B, and C to three processes P0, P1, and P2. Consider the following scenario as reference .user must enter the current state of system as given in this example : Suppose P0 has 0,0,1 instances , P1 is having 3,2,0 instances and P2 occupies 2,1,1 instances of A,B,C resource respectively. Also the maximum number of instances required for P0 is 8,4,3 and for p1 is 6,2,0 and finally for P2 there are 3,3,3 instances of resources A,B,C respectively. There are 3 instances of resource A, 2 instances of resource B and 2 instances of resource C available. Write a program to check whether Reena's operating system is in a safe state or not in the following independent requests for additional resources in the current state: 1. Request1: P0 requests 0 instances of A and 0 instances of B and 2 instances of C. 2. Request2: P1 requests for 2 instances of A, 0 instances of B and 0 instances of C. All the request must be given by user as input.

Ans:

The financier's calculation is an asset assignment and gridlock evasion calculation that tests for wellbeing by recreating the designation for foreordained greatest potential measures all things considered, at that point makes a "s-state" check to test for potential exercises, before choosing whether distribution ought to be permitted to proceed.

What is Banker's Algorithm?

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Banker's algorithm is named so because it is used in banking system to check whether loan can be sanctioned to a person or not. Suppose there are n number of account holders in a bank and the total sum of their money is S. If a person applies for a loan then the bank first subtracts the loan amount from the total money that bank has and if the remaining amount is greater than S then only the loan is sanctioned. It is done because if all the account holders comes to withdraw their money then the bank can easily do it.
In other words, the bank would never allocate its money in such a way that it can no longer

satisfy the needs of all its customers. The bank would try to be in safe state always

Following **Data structures** are used to implement the Banker's Algorithm:
Let **'n'** be the number of processes in the system and **'m'** be the number of resources types.

Available :

- It is a 1-d array of size **'m'** indicating the number of available resources of each type.
- Available[j] = k means there are **'k'** instances of resource type $\mathbf{R_j}$

Max :

- It is a 2-d array of size 'n*m' that defines the maximum demand of each process in a system.
- Max[i, j] = k means process $\mathbf{P_i}$ may request at most **'k'** instances of resource type $\mathbf{R_j}$.

Allocation :

- It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to each process.
- Allocation[i, j] = k means process $\mathbf{P_i}$ is currently allocated **'k'** instances of resource type $\mathbf{R_j}$

Need :

- It is a 2-d array of size **'n*m'** that indicates the remaining resource need of each process.
- Need [i, j] = k means process $\mathbf{P_i}$ currently need **'k'** instances of resource type $\mathbf{R_j}$

for its execution.

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

.

Algorithm:

1) Let Work and Finish be vectors of length 'm' and 'n' respectively.
Initialize: Work = Available
Finish[i] = false; for i=1, 2, 3, 4….n

2) Find an i such that both
a) Finish[i] = false
b) $Need_i$ <= Work
if no such i exists goto step (4)
3) Work = Work + Allocation[i]
Finish[i] = true
goto step (2)

4) if Finish [i] = true for all i
then the system is in a safe state

Resource-Request Algorithm

1) If $Request_i$ <= $Need_i$
Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.
2) If $Request_i$ <= Available

Goto step (3); otherwise, P_i must wait, since the resources are not available.
3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:
Available = Available – Requesti
$Allocation_i = Allocation_i + Request_i$
$Need_i = Need_i – Request_i$

Example:

Considering a system with five processes P_0 through P_4 and three resources of type A, B, C Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose time t_0 following snapshot of the system has been taken:

Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Question1. What will be the content of the Need matrix?

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

So, the content of Need Matrix is:

Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Code:

General Terms:

- **Arrival Time**: Time at which the process arrives in the ready queue.
- **Completion Time**: Time at which process completes its execution.
- **Burst Time**: Time required by a process for CPU execution.
- **Turn Around Time**: Time Difference between completion time and arrival time.
- **Waiting Time**: Time Difference between turnaround time and burst time.

Code:

```
#include<stdio.h>
#include<conio.h>
int main()
{
	int Process=5,Resource=3;
	printf("Enter number of process and resources accorting to choice:  ");
	scanf("%d%d",&Process,&Resource);
	const int P=Process,R=Resource;
   int avl[15];
   printf("Enter the number of available resources respectively:     ");
	    for(int i=0;i<R;i++)
	{
		scanf("%d",&avl[i]);
	}
   int max[10][10];
   for(int i=0;i<P;i++)
	{
		printf("Enter max number of resources required for process P%d:    ",i+1);
		for(int j=0;j<R;j++)
		{
			scanf("%d",&max[i][j]);
		}
	}
   int allt[10][10];
   for(int i=0;i<P;i++)
	{
		printf("Enter all the  resources allocated by P%d:      ",i+1);
		for(int j=0;j<R;j++)
		{
			scanf("%d",&allt[i][j]);
		}
	}
```

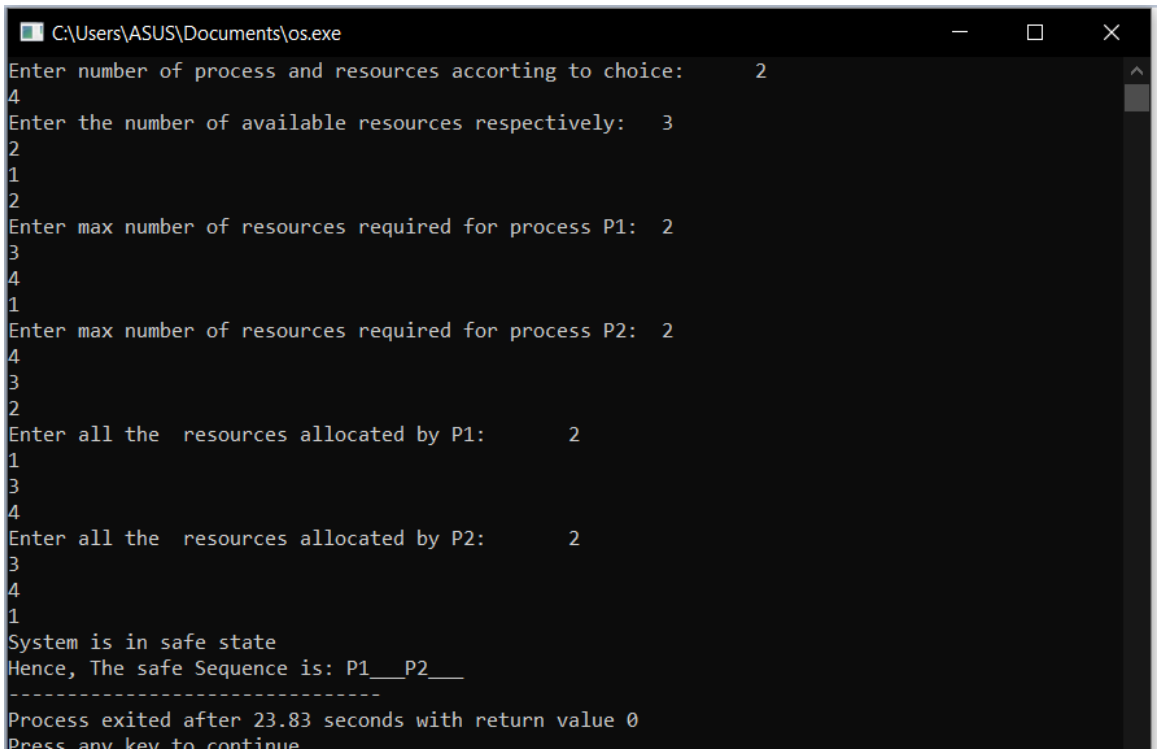
```
    int need[P][R];
     for (int i = 0 ; i < P ; i++)
        for (int j = 0 ; j < R ; j++)
           need[i][j] = max[i][j] - allt[i][j];
     bool finish[P] = {0};
     int safeSeq[P];
     int work[R];
     for (int i = 0; i < R ; i++)
        work[i] = avl[i];
     int count = 0;
         while (count < P)
     {
        bool found = false;
        for (int p = 0; p < P; p++)
        {
           if (finish[p] == 0)
           {
              int j;
              for (j = 0; j < R; j++)
                 if (need[p][j] > work[j])
                    break;
                                    if (j == R)
              {

                 for (int k = 0 ; k < R ; k++)
                    work[k] += allt[p][k];
                  safeSeq[count++] = p;
                 finish[p] = 1;

                 found = true;
              }
           }
        }
        if (found == false)
        {
           printf("System is not in safe state");
        }
     }
     printf("System is in safe state \n");
     printf("Hence, The safe Sequence is: ");
     for (int i = 0; i < P ; i++)
        printf("P%d___",safeSeq[i]+1);

  }
```

Output:


C:\Users\ASUS\Documents\os.exe
Enter number of process and resources accorting to choice: 2
4
Enter the number of available resources respectively: 3
2
1
2
Enter max number of resources required for process P1: 2
3
4
1
Enter max number of resources required for process P2: 2
4
3
2
Enter all the resources allocated by P1: 2
1
3
4
Enter all the resources allocated by P2: 2
3
4
1
System is in safe state
Hence, The safe Sequence is: P1___P2___

Process exited after 23.83 seconds with return value 0
Press any key to continue