VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

OPERATING SYSTEMS

LAB – ASSIGNMENT: 3

17BCE0581

SATYAM SINGH CHAUHAN

Under the Guidance

Of

Prof. GOPINATH M.P

Date: 10 October 2019

Task: Deadlock – Banker's Algorithm.

AIM

To develop a code for **Banker's Algorithm** to **detect the deadlock** if any otherwise show the **safe sequence**.

ALGORITHM

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

2) Find an i such that both
      a. Finish[i] = false
      b. Needi <= Work

   if no such i exists goto step (4)

3) Work = Work + Allocation[i]
      a. Finish[i] = true
      b. goto step (2)

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

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.

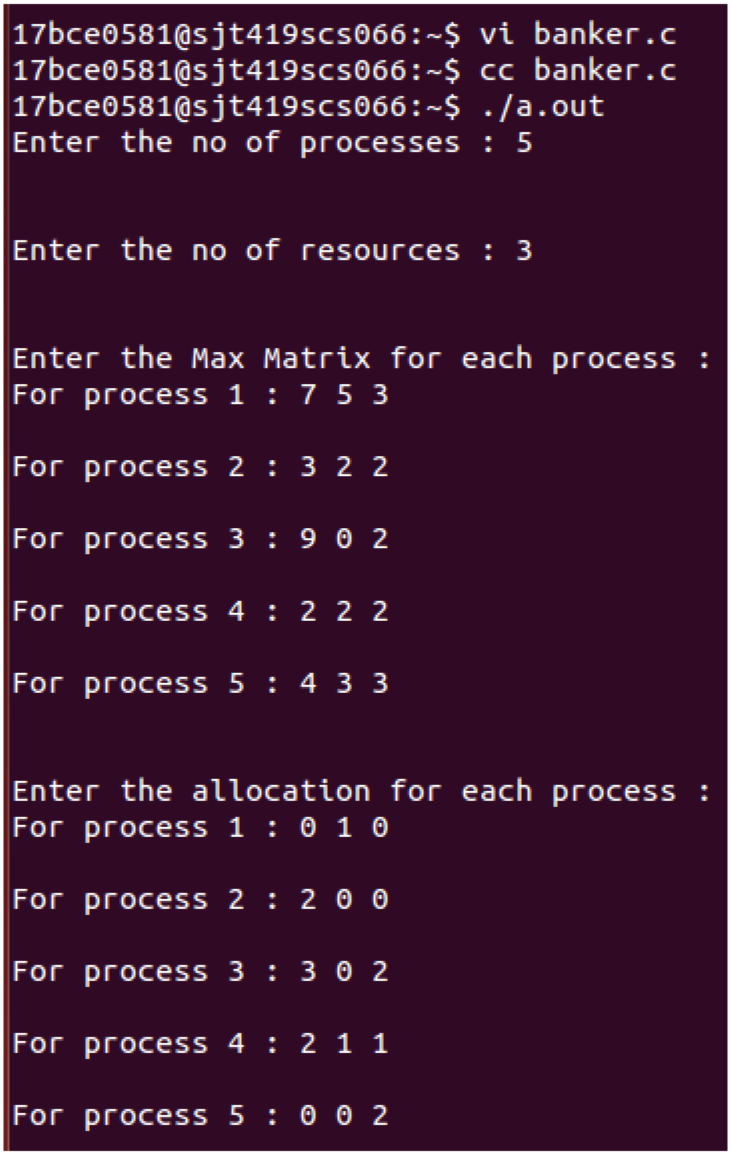
- Need [i, j] = Max [i, j] – Allocation [i, j]

CODE

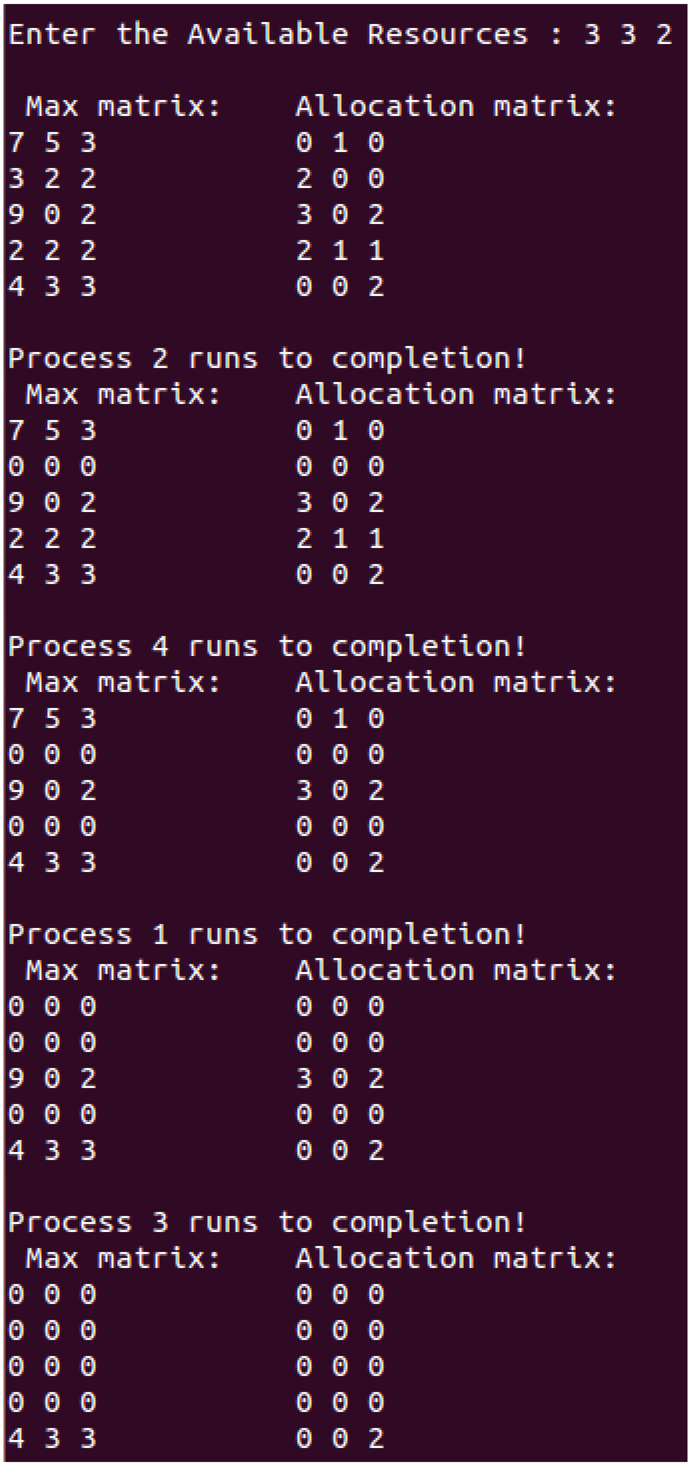
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10],
    safeSequence[10];
    int p, r, i, j, process, count;
    count = 0;
    printf("Enter the no of processes : ");
    scanf("%d", &p);
    for(i = 0; i< p; i++)
        completed[i] = 0;
    printf("\n\nEnter the no of resources : ");
    scanf("%d", &r);
    printf("\n\nEnter the Max Matrix for each process : ");
    for(i = 0; i < p; i++)
    {
        printf("\nFor process %d : ", i + 1);
        for(j = 0; j < r; j++)
        scanf("%d", &Max[i][j]);
    }
    printf("\n\nEnter the allocation for each process : ");
    for(i = 0; i < p; i++)
    {
        printf("\nFor process %d : ",i + 1);
        for(j = 0; j < r; j++)
        scanf("%d", &alloc[i][j]);
    }
    printf("\n\nEnter the Available Resources : ");
    for(i = 0; i < r; i++)
        scanf("%d", &avail[i]);
    for(i = 0; i < p; i++)
        for(j = 0; j < r; j++)
            need[i][j] = Max[i][j] - alloc[i][j];
    do
    {
        printf("\n Max matrix:\tAllocation matrix:\n");
        for(i = 0; i < p; i++)
        {
            for( j = 0; j < r; j++)
```

```
                printf("%d ", Max[i][j]);
            printf("\t\t");
            for( j = 0; j < r; j++)
                printf("%d ", alloc[i][j]);
            printf("\n");
        }
        process = -1;
        for(i = 0; i < p; i++)
        {
            if(completed[i] == 0)//if not completed
            {
                process = i ;
                for(j = 0; j < r; j++)
                {
                    if(avail[j] < need[i][j])
                    {
                        process = -1;
                        break;
                    }
                }
            }
            if(process != -1)
                break;
        }
        if(process != -1)
        {
            printf("\nProcess %d runs to completion!", process + 1);
            safeSequence[count] = process + 1;
            count++;
            for(j = 0; j < r; j++)
            {
                avail[j] += alloc[process][j];
                alloc[process][j] = 0;
                Max[process][j] = 0;
                completed[process] = 1;
            }
        }
    }
    while(count != p && process != -1);
    if(count == p)
    {
        printf("\nThe system is in a safe state!!\n");
        printf("Safe Sequence : < ");
        for( i = 0; i < p; i++)
            printf("%d ", safeSequence[i]);
        printf(">\n");
    }
    else
    printf("\nThe system is in an unsafe state!!");
}
```

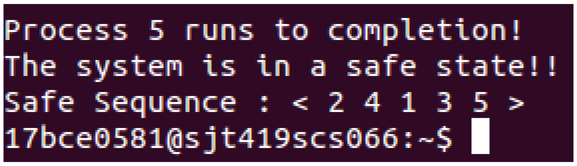
OUTPUT



```
17bce0581@sjt419scs066:~$ vi banker.c
17bce0581@sjt419scs066:~$ cc banker.c
17bce0581@sjt419scs066:~$ ./a.out
Enter the no of processes : 5


Enter the no of resources : 3


Enter the Max Matrix for each process :
For process 1 : 7 5 3

For process 2 : 3 2 2

For process 3 : 9 0 2

For process 4 : 2 2 2

For process 5 : 4 3 3


Enter the allocation for each process :
For process 1 : 0 1 0

For process 2 : 2 0 0

For process 3 : 3 0 2

For process 4 : 2 1 1

For process 5 : 0 0 2
```



```
Enter the Available Resources : 3 3 2

 Max matrix:    Allocation matrix:
7 5 3           0 1 0
3 2 2           2 0 0
9 0 2           3 0 2
2 2 2           2 1 1
4 3 3           0 0 2

Process 2 runs to completion!
 Max matrix:    Allocation matrix:
7 5 3           0 1 0
0 0 0           0 0 0
9 0 2           3 0 2
2 2 2           2 1 1
4 3 3           0 0 2

Process 4 runs to completion!
 Max matrix:    Allocation matrix:
7 5 3           0 1 0
0 0 0           0 0 0
9 0 2           3 0 2
0 0 0           0 0 0
4 3 3           0 0 2

Process 1 runs to completion!
 Max matrix:    Allocation matrix:
0 0 0           0 0 0
0 0 0           0 0 0
9 0 2           3 0 2
0 0 0           0 0 0
4 3 3           0 0 2

Process 3 runs to completion!
 Max matrix:    Allocation matrix:
0 0 0           0 0 0
0 0 0           0 0 0
0 0 0           0 0 0
0 0 0           0 0 0
4 3 3           0 0 2
```



```
Process 5 runs to completion!
The system is in a safe state!!
Safe Sequence : < 2 4 1 3 5 >
17bce0581@sjt419scs066:~$ 
```