Name: Aditya Matale

Class: AIDS-B

Roll No: 25

Q] What is Banker's Algorithm?

A] The Banker's Algorithm is a deadlock avoidance technique used in operating systems. It ensures safe resource allocation by granting resources to processes only if it can guarantee a safe state. The algorithm checks if requested resources are available and if their allocation maintains system safety. It prevents potential deadlocks by carefully managing resource allocation and is used to allocate resources to processes in a way that avoids the possibility of deadlock. If a resource request would lead to an unsafe state, it is denied.

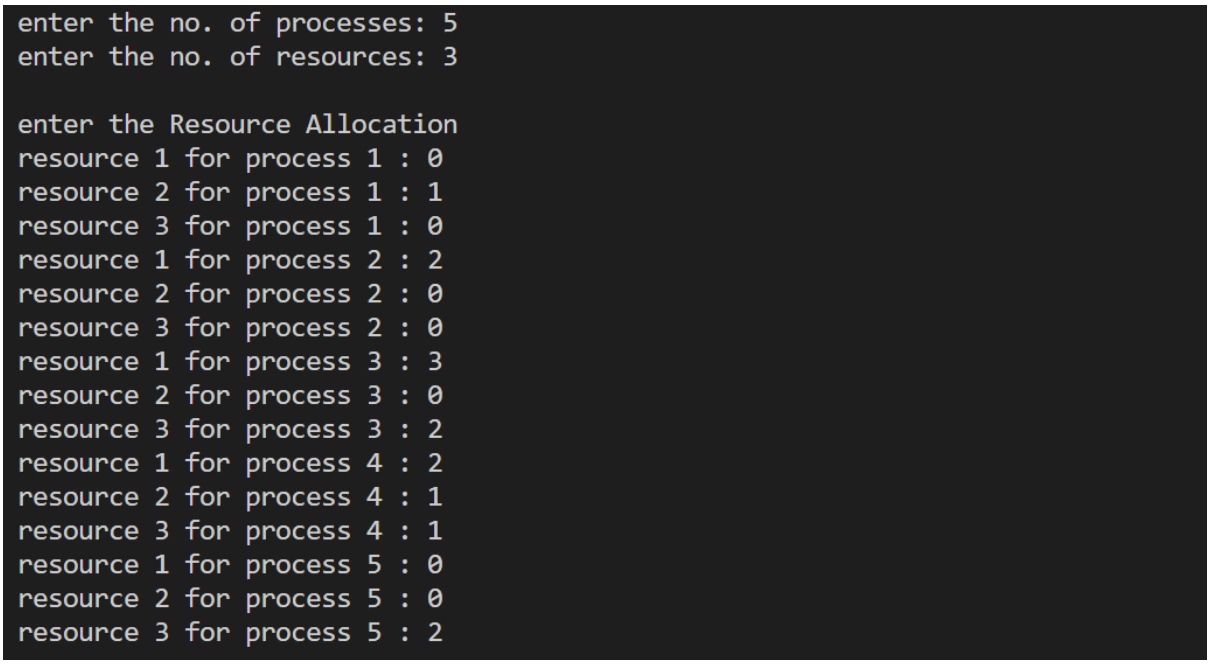
CODE:

```c
// Banker's Algorithm
#include <stdio.h>
int main()
{
    int n, m, i, j, k;
    printf("enter the no. of processes: ");
    scanf("%d",&n);
    printf("enter the no. of resources: ");
    scanf("%d",&m);

    int alloc[n][m];

    printf("\nenter the Resource Allocation\n");
    for(int i=0;i<n;i++){
        for (int j = 0; j < m; j++)
        {
            printf("resource %d for process %d : ",j+1,i+1);
            scanf("%d",&alloc[i][j]);
        }
    }

    int max[n][m];

    printf("\nenter the Max Need\n");
    for(int i=0;i<n;i++){
        for (int j = 0; j < m; j++)
        {
            printf("max need of resource %d for process %d : ",j+1,i+1);
            scanf("%d",&max[i][j]);
        }
    }
```

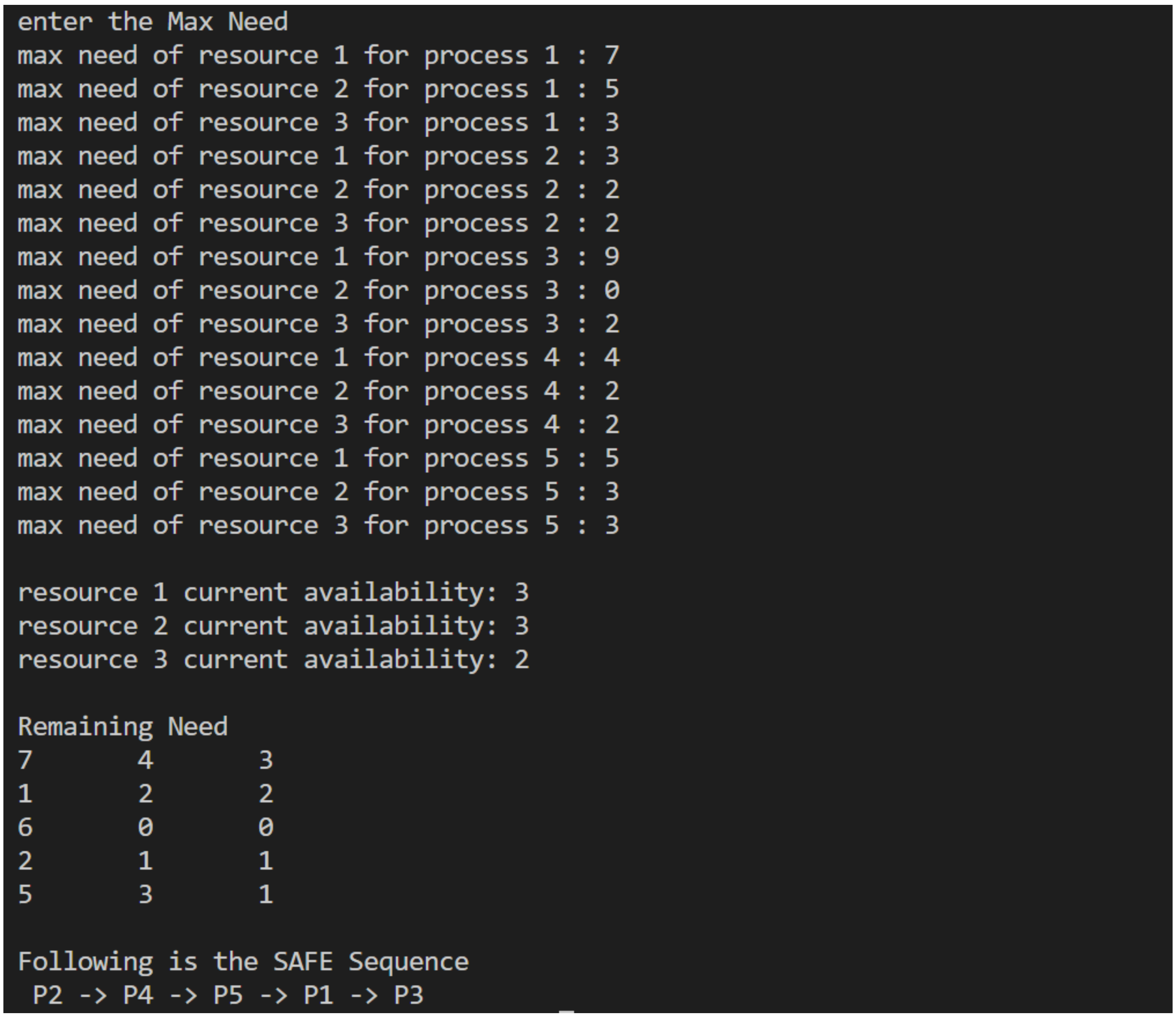
```c
int avail[m];
printf("\n");
for (int i=0;i<m;i++){
    printf("resource %d current availability: ",i+1);
    scanf("%d",&avail[i]);
}

int f[n], ans[n], ind = 0;
for (k = 0; k < n; k++) {
    f[k] = 0;
}
int need[n][m];
printf("\nRemaining Need\n");
for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++){
        need[i][j] = max[i][j] - alloc[i][j];
        printf("%d\t",need[i][j]);
    }
    printf("\n");
}
int y = 0;
for (k = 0; k < 5; k++) {
    for (i = 0; i < n; i++) {
        if (f[i] == 0) {

            int flag = 0;
            for (j = 0; j < m; j++) {
                if (need[i][j] > avail[j]){
                    flag = 1;
                    break;
                }
            }

            if (flag == 0) {
                ans[ind++] = i;
                for (y = 0; y < m; y++)
                    avail[y] += alloc[i][y];
                f[i] = 1;
            }
        }
    }
}

int flag = 1;

for(int i=0;i<n;i++)
{
if(f[i]==0)
{
    flag=0;
    printf("The following system is not safe");
    break;
}
}
```

```c
    if(flag==1)
    {
    printf("\nFollowing is the SAFE Sequence\n");
    for (i = 0; i < n - 1; i++)
        printf(" P%d ->", ans[i]+1);
    printf(" P%d", ans[n - 1]+1);
    }


    return (0);
}
```

OUTPUT:



```
enter the no. of processes: 5
enter the no. of resources: 3

enter the Resource Allocation
resource 1 for process 1 : 0
resource 2 for process 1 : 1
resource 3 for process 1 : 0
resource 1 for process 2 : 2
resource 2 for process 2 : 0
resource 3 for process 2 : 0
resource 1 for process 3 : 3
resource 2 for process 3 : 0
resource 3 for process 3 : 2
resource 1 for process 4 : 2
resource 2 for process 4 : 1
resource 3 for process 4 : 1
resource 1 for process 5 : 0
resource 2 for process 5 : 0
resource 3 for process 5 : 2
```



```
enter the Max Need
max need of resource 1 for process 1 : 7
max need of resource 2 for process 1 : 5
max need of resource 3 for process 1 : 3
max need of resource 1 for process 2 : 3
max need of resource 2 for process 2 : 2
max need of resource 3 for process 2 : 2
max need of resource 1 for process 3 : 9
max need of resource 2 for process 3 : 0
max need of resource 3 for process 3 : 2
max need of resource 1 for process 4 : 4
max need of resource 2 for process 4 : 2
max need of resource 3 for process 4 : 2
max need of resource 1 for process 5 : 5
max need of resource 2 for process 5 : 3
max need of resource 3 for process 5 : 3

resource 1 current availability: 3
resource 2 current availability: 3
resource 3 current availability: 2

Remaining Need
7       4       3
1       2       2
6       0       0
2       1       1
5       3       1

Following is the SAFE Sequence
 P2 -> P4 -> P5 -> P1 -> P3
```