OPERATING SYSTEMS

Lab-8:Bankers Algorithm Safety Sequence

Name – Vinay Santosh Menon
Registration Number - 20BAI1103

TASK:
To write a C code for Banker’s algorithm safety sequence.

A process in OS can request a resource, can use the resource, and can also release it. There comes a situation of deadlock in OS in which a set of processes is blocked because it is holding a resource and also requires some other resources at the same time that are being acquired by other processes. So, to avoid such a situation of deadlock, we have the Bankers algorithm in Operating System.

Below is the implantation of Bankers Algorithm in C. We first get the avl, max and alloc matrix, and then using banker's algorithm we use banker's algorithm to allocate resources.

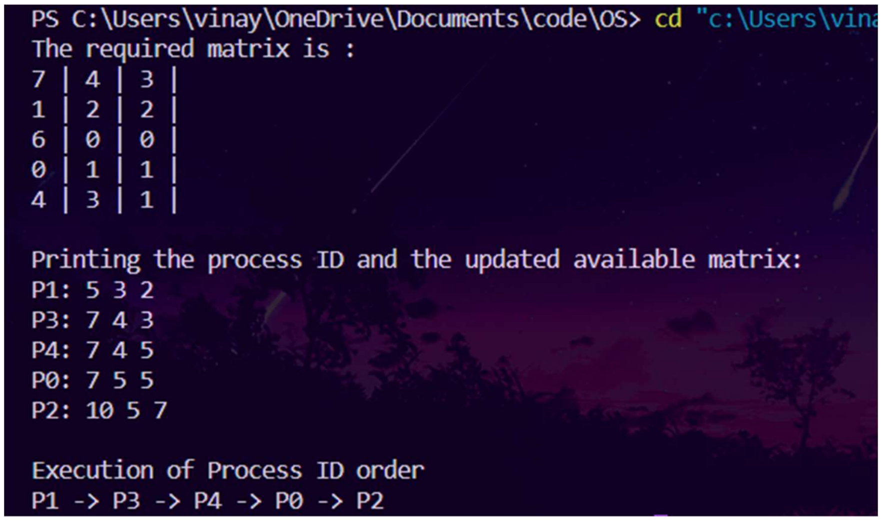
Code:

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n = 5, m = 3;
    int alloc[5][3] = {{0,1,0},{2,0,0},{3,0,2},{2,1,1},{0,0,2}};
    int max[5][3] = {{7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3}};
    int req[5][3];
    int avl[3] = {3,3,2};
    int check[5] = {0,0,0,0,0};
    int proc[5];
    int flag = 1, ind = -1;
    int count = 0;
    int arr_count = 0;
    for(int i = 0; i<n; i++)
    {
        for(int j = 0; j<m; j++)
        {
            req[i][j] = max[i][j]-alloc[i][j];
        }
    }
    printf("The required matrix is :\n");
    for(int i = 0; i<n; i++)
    {
        for(int j = 0; j<m; j++)
        {
            printf("%d | ",req[i][j]);
        }
        printf("\n");
    }
    printf("\n");
    printf("Printing the process ID and the updated available matrix:\n");
    while(count!=4)
    {
        for(int i = 0; i<n; i++)
        {
            for(int j = 0; j<m; j++)
            {
                if(req[i][j]>avl[j])
                {
                    flag = 0;
                    break;
                }
            }
            if(flag==1)
            {
                if(check[i]==0)
                {
                    proc[arr_count++] = i;
                    printf("P%d: ",i);
                    for(int j = 0; j<m; j++)
                    {
                        avl[j] += alloc[i][j];
                        printf("%d ",avl[j]);
                    }
                    printf("\n");
                    check[i]=1;
                }
            }
            else
            flag = 1;
        }
        count++;
    }
    printf("\n");
    printf("Execution of Process ID order\n");
    for(int i = 0; i<arr_count; i++)
    {
        if(i==arr_count-1)
        {
            printf("P%d",proc[i]);
        }
        else
        printf("P%d -> ",proc[i]);
    }
```

```
}
```

Output:



```
PS C:\Users\vinay\OneDrive\Documents\code\OS> cd "c:\Users\vina
The required matrix is :
7 | 4 | 3 |
1 | 2 | 2 |
6 | 0 | 0 |
0 | 1 | 1 |
4 | 3 | 1 |

Printing the process ID and the updated available matrix:
P1: 5 3 2
P3: 7 4 3
P4: 7 4 5
P0: 7 5 5
P2: 10 5 7

Execution of Process ID order
P1 -> P3 -> P4 -> P0 -> P2
```

In the above C code, we have implemented the Bankers algorithm in OS and at the start, we have initialized allocation matrix (indicating the already allocated resources to each process of different kinds), max matrix (indicating the maximum number of resources that can be allocated to each process) and available array (indicating the currently present available resources in the system). With the help of the max matrix and allocation matrix, we will calculate need matrix (indicating the required resources to be allocated of different types by each process). (Need)i = (MAX)i - (Allocation)i$(Need)i=(MAX)i-(Allocation)i$

After that, we need to allocate resources to each process such that needed resources <= available resources for each process. If in case, available resources are less than the needed resources (say for P1), the system will move on to the next process (say P2) for allocation and will return back to P1 only after checking all the remaining processes in a loop. If available resources are allocated to a process, the then available array will be updated and allocated resources of that process will be added to the available array. In the end, we just need to check that if all the processes are allocated the desired number of resources, then the system is in a safe state and we will print the safe state sequence (indicating which process has been allocated the resources first).

There are some important data structure terms that are used in the above C++ code to implement Bankers' algorithm in OS and they are as follows: 1. Available: Example: Available[i] = k indicates that there are 'k' instances of resource type 'Rj' available within the system. 2. Max: Example: Max[i][j] = k indicates that process 'Pi' can request for maximum k instances of resource type 'Rj' within the system. 3. Allocation: Example: Allocation[i][j] = k indicates that currently process ‘Pi’ is allocated ‘k’ instances of resource type 'Rj' within the system. 4. Need: Example: Need [i, j] = k indicates that ‘Pi’ process presently requires ‘k’ instances of resource type ‘Rj’ for its execution. 5. Finish Example: Finish[i] = true indicates that process Pi has been allocated all the required number of resources by the system.

While implementing or working with Banker’s algorithm in OS, it is quite important to keep a track of three things:

1. How many resources of each type, a process can request for and it is denoted by the [MAX] array.
2. How many resources of each type, a process is currently holding or allocated and it is denoted by [ALLOCATION] array.
3. How many resources of each type are currently available within the system and it is denoted by the [AVAILABLE] array.