Operating Systems

Experiment – Banker's Algorithm

Name: Kartik Jolapara **SAP ID: 60004200107**

Div.: B1 **Branch: Computer Engineering**

Aim -

To apply bankers algorithm and check whether the system is in deadlock or not.

Theory –

It is a banker algorithm used to avoid deadlock and allocate resources safely to each process in the computer system. The 'S-State' examines all possible tests or activities before deciding whether the allocation should be allowed to each process. It also helps the operating system to successfully share the resources between all the processes.

The banker's algorithm is named because it checks whether a person should be sanctioned a loan amount or not to help the bank system safely simulate allocation resources. In this section, we will learn the Banker's Algorithm in detail. Also, we will solve problems based on the Banker's Algorithm. To understand the Banker's Algorithm first we will see a real word example of it. Suppose the number of account holders in a particular bank is 'n', and the total money in a bank is 'T'.

If an account holder applies for a loan; first, the bank subtracts the loan amount from full cash and then estimates the cash difference is greater than T to approve the loan amount. These steps are taken because if another person applies for a loan or withdraws some amount from the bank, it helps the bank manage and operate all things without any restriction in the functionality of the banking system. Similarly, it works in an operating system.

When a new process is created in a computer system, the process must provide all types of information to the operating system like upcoming processes, requests for their resources, counting them, and delays. Based on these criteria, the operating system decides which process sequence should be executed or waited so that no deadlock occurs in a system. Therefore, it is also known as deadlock avoidance algorithm or deadlock detection in the operating system.

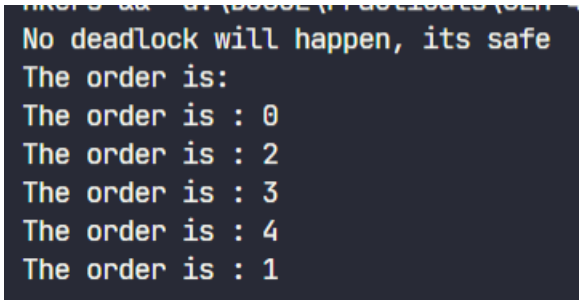
Code –

```
#include <stdio.h>
#include <stdbool.h>
#define process 5
#define resource 4

void isSafe(int processes[], int available[], int
max_req[process][resource], int allocated[process][resource])
{
    int remaining_need[process][resource];
    int order[process], index = 0;
    for (int i = 0; i < process; i++)
    {
        for (int j = 0; j < resource; j++)
        {
            remaining_need[i][j] = max_req[i][j] - allocated[i][j];
        }
    }
    bool check[process] = {false, false, false, false, false};
    for (int i = 0; i < process; i++)
    {
        for (int j = 0; j < process; j++)
        {
            int count = 0;
            if (check[j] == false)
            {
                for (int k = 0; k < resource; k++)
                {
                    if (available[k] >= remaining_need[j][k])
                    {
                        count++;
                    }
                    else
                    {
                        break;
                    }
                }
                if (count == resource)
                {
```

```
                    check[j] = true;
                    for (int l = 0; l < resource; l++)
                    {
                        available[l] += allocated[j][l];
                    }
                    order[index] = j;
                    index++;
                }
            }
            else
            {
                continue;
            }
        }
    }
    if (index == process)
    {
        printf("No deadlock will happen, its safe\nThe order is: \n");
        for (int p = 0; p < process; p++)
        {
            printf("The order is : %d \n", order[p]);
        }
    }
    else
    {
        printf("deadlock will happen, its unsafe\n");
    }
}
int main()
{
    int processes[process] = {0, 1, 2, 3, 4};
    int available[process] = {1, 5, 2, 0};
    int allocated[process][resource] = {{0, 0, 1, 2},
                                        {1, 0, 0, 0},
                                        {1, 3, 5, 4},
                                        {0, 6, 3, 2},
                                        {0, 0, 1, 4}};
    int max_req[process][resource] = {{0, 0, 1, 2},
                                      {1, 7, 5, 0},
                                      {2, 3, 5, 6},
                                      {0, 6, 5, 2},
                                      {0, 6, 5, 6}};
    isSafe(processes, available, max_req, allocated);
}
```

Output –



Conclusion -

Banker's Algorithm has been successfully implemented. Banker's Algo helps the operating system manage and process control request for each type of resource in the computer system.