Experiment 05

Banker's Algorithm

Aim: To write a program in C to perform **Banker's Algorithm** for deadlock prevention.

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.

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.

When working with a banker's algorithm, it requests to know about three things:

- ✓ How much each process can request for each resource in the system. It is denoted by the [MAX] request.
- ✓ How much each process is currently holding each resource in a system. It is denoted by the [ALLOCATED] resource.
- ✓ It represents the number of each resource currently available in the system. It is denoted by the [AVAILABLE] resource.

Example: Consider a system that contains five processes P1, P2, P3, P4, P5 and the three resource types A, B and C. Following are the resources types: A has 10, B has 5 and the resource type C has 7 instances.

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2
P2	2	0	0	3	2	2			
P3	3	0	2	9	0	2			
P4	2	1	1	2	2	2			
P5	0	0	2	4	3	3			

we execute the banker's algorithm to find the safe state and the safe sequence like P2, P4, P5, P1 and P3.

Program:

```
#include<stdio.h>
void input_array(int a[], int res){

    for(int j=0;j<res;j++)
        scanf("%d",&a[j]);

}
int main(){
    printf("\t\t\tBanker's Algorithm");
    int pro,res;
    printf("\nEnter no. of resources: ");
    scanf("%d",&res);
    printf("Enter no. of processes: ");
    scanf("%d",&pro);
    int max[pro][res], need[pro][res], available[res], allocated[pro][res],finish[pro]={0};
    for(int i=0;i<pro;i++){
        printf("\nProcess %d\n",i+1);
        printf("Max: ");
        input_array(max[i],res);
        printf("Allocated: ");
        input_array(allocated[i],res);
    }
    printf("\nAvailable resources: ");
    for(int i=0;i<res;i++){
        scanf("%d",&available[i]);
    }
    for(int i=0;i<pro;i++)
        for(int j=0;j<res;j++)
            need[i][j]=max[i][j]-allocated[i][j];
    int completed = 0,i=0,can=0,j,dead=0;
    printf("\nSafe sequence: ");
    while(completed!=pro){
        if(finish[i]){
            i=(i+1)%pro;
            continue;
        }
        for(j=0;j<res;j++){
            if(available[j]<need[i][j]){
                i=(i+1)%pro;
                dead++;
                break;
            }
        }
        if(dead==pro-completed){
            printf("\nt\t\tDeadlock!!!");
            break;
        }
        if(j==res)
```

```
        {   dead=0;
            finish[i]=1;
            printf("P%d->",i+1);
            completed++;
            for(int k=0;k<res;k++)
                available[k]+=allocated[i][k];
        }
    }
    return 0;
}
```

Output:

```
                Banker's Algorithm
Enter no. of resources: 3
Enter no. of processes: 5

Process 1
Max: 7      5      3
Allocated: 0      1      0

Process 2
Max: 3      2      2
Allocated: 2      0      0

Process 3
Max:  9      0      2
Allocated: 3      0      2

Process 4
Max: 2      2      2
Allocated: 2      1      1

Process 5
Max: 4      3      3
Allocated: 0      0      2

Available resources: 3      3      2

Safe sequence: P2->P4->P5->P1->P3->
```