EX.NO: 8 Implement BANKERS ALGORITHM for Deadlock Avoidance
Date:23.9.2024

AIM:

A program to simulate the BANKERS ALGORITHM for Deadlock Avoidance.

ALGORITHM:

Step1: Get the number of processes and number of resource instances.
Step2: Get the allocation matrix and Available matrix from the user.
Step3: Calculate need matrix.
Step4: Using banker's ALGORITHM allocate resources to processes.
Step5: Print safe sequence of processes.
Step6: Stop the program.

PROGRAM:

```
#include <stdio.h>
#define MAX 100
int max[MAX][MAX], alloc[MAX][MAX], need[MAX][MAX], avail[MAX];
int n, r;
void input() {
   printf("Enter the number of Processes: ");
   scanf("%d", &n);
   printf("Enter the number of Resource Instances: ");
   scanf("%d", &r);
   printf("Enter the Max Matrix:\n");
   for (int i = 0; i < n; i++)
      for (int j = 0; j < r; j++)
         scanf("%d", &max[i][j]);

   printf("Enter the Allocation Matrix:\n");
   for (int i = 0; i < n; i++)
      for (int j = 0; j < r; j++)
         scanf("%d", &alloc[i][j]);

   printf("Enter the Available Resources:\n");
   for (int j = 0; j < r; j++)
      scanf("%d", &avail[j]);
}
void show() {
   printf("Process\tAllocation\tMax\tAvailable\n");
   for (int i = 0; i < n; i++) {
      printf("P%d\t", i + 1);
      for (int j = 0; j < r; j++)
printf("%d ", alloc[i][j]);
      printf("\t");
```

```
        for (int j = 0; j < r; j++)
            printf("%d ", max[i][j]);
        if (i == 0) {
            printf("\t");
            for (int j = 0; j < r; j++)
                printf("%d ", avail[j]);
        }
        printf("\n");
    }
}

void cal() {
    int finish[MAX] = {0}, safeSeq[MAX], count = 0;

    for (int i = 0; i < n; i++)
        for (int j = 0; j < r; j++)
            need[i][j] = max[i][j] - alloc[i][j];

    while (count < n) {
        int found = 0;
        for (int i = 0; i < n; i++) {
            if (!finish[i]) {
                int j;
                for (j = 0; j < r; j++)
                    if (need[i][j] > avail[j]) break;

                if (j == r) {
                    for (j = 0; j < r; j++)
                        avail[j] += alloc[i][j];
                    finish[i] = 1;
                    safeSeq[count++] = i;
                    found = 1;
                    printf("P%d->", i);
                }
            }
        }
        if (!found) break;  // No more processes can be finished
    }
    printf("\n");
    if (count == n) {
        printf("The system is in a safe state.\n");
        printf("Safe sequence is: ");
        for (int i = 0; i < n; i++)
            printf("P%d ", safeSeq[i] + 1);
        printf("\n");
    } else {
```

```
            printf("Processes are in deadlock.\n");
            printf("The system is in unsafe state.\n");
        }
    }
    int main() {
        printf("********** Banker's ALGORITHM ************\n");
        input();
        show();
        cal();
        return 0;
    }
```

OUTPUT:

```
Enter the number of Processes: 5
Enter the number of Resource Instances: 3
Enter the Max Matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the Allocation Matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the Available Resources:
3 3 2

********** Banker's ALGORITHM ************
Process         Allocation      Max     Available
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
P1->P3->P4->P2->

The system is in a safe state.
Safe sequence is: P1 P3 P4 P2 P5
```

RESULT:

Thus the program for implementing deadlock avoidance ALGORITHM was implemented has been executed successfully.