Ex. No.: 9

DEADLOCK AVOIDANCE

Aim:

To find out a safe sequence using Banker's algorithm for deadlock avoidance.

Algorithm:

1. Initialize work=available and finish[i]=false for all values of i 2. Find an i such that both:
finish[i]=false and $Need_i$<=
work 3. If no such i exists go to step
6
4. Compute work=work+allocationi
5. Assign finish[i] to true and go to step 2
6. If finish[i]==true for all i, then print safe sequence
7. Else print there is no safe sequence

Program Code:

```
#include <stdio.h>
#include <stdbool.h>

#define MAX 10
#define RESOURCE_TYPES 3

int processes, resources;
int allocation[MAX][RESOURCE_TYPES];
int maximum[MAX][RESOURCE_TYPES];
int available[RESOURCE_TYPES];
int need[MAX][RESOURCE_TYPES];
int finish[MAX] = {0};
int safeSequence[MAX];

void findNeedMatrix() {
   for(int i = 0; i < processes; i++) {
      for(int j = 0; j < resources; j++) {
         need[i][j] = maximum[i][j] - allocation[i][j];
      }
   }
}

int isSafeState() {
   int work[RESOURCE_TYPES];
```

```
    for(int i = 0; i < resources; i++) {
        work[i] = available[i];
    }

    int count = 0;
    while(count < processes) {
        bool foundProcess = false;
        for(int i = 0; i < processes; i++) {
            if(finish[i] == 0) {
                int j;
                for(j = 0; j < resources; j++) {
                    if(need[i][j] > work[j]) {
                        break;
                    }
                }
                if(j == resources) {
                    for(int k = 0; k < resources; k++) {
                        work[k] += allocation[i][k];
                    }
                    safeSequence[count++] = i;
                    finish[i] = 1;
                    foundProcess = true;
                    break;
                }
            }
        }
        if(!foundProcess) {
            return 0;  // No safe sequence found
        }
    }
    return 1;  // Safe sequence found
}

void printSafeSequence() {
    printf("\nSafe Sequence is: ");
    for(int i = 0; i < processes; i++) {
        printf("P%d ", safeSequence[i]);
    }
    printf("\n");
}

int main() {
    printf("Enter the number of processes: ");
    scanf("%d", &processes);

    printf("Enter the number of resource types: ");
    scanf("%d", &resources);
```

```
    printf("Enter the Allocation Matrix:\n");
    for(int i = 0; i < processes; i++) {
        for(int j = 0; j < resources; j++) {
            printf("Allocation[%d][%d]: ", i, j);
            scanf("%d", &allocation[i][j]);
        }
    }

    printf("Enter the Maximum Matrix:\n");
    for(int i = 0; i < processes; i++) {
        for(int j = 0; j < resources; j++) {
            printf("Maximum[%d][%d]: ", i, j);
            scanf("%d", &maximum[i][j]);
        }
    }

    printf("Enter the Available Resources:\n");
    for(int i = 0; i < resources; i++) {
        printf("Available[%d]: ", i);
        scanf("%d", &available[i]);
    }

    findNeedMatrix();

    if(isSafeState()) {
        printSafeSequence();
    } else {
        printf("\nThe system is not in a safe state.\n");
    }

    return 0;
}
```

Sample Output:

```
The SAFE Sequence is
P1 -> P3 -> P4 -> P0 -> P2
```

Output:

```
Enter the number of processes: 5
Enter the number of resource types: 3
Enter the Allocation Matrix:
Allocation[0][0]: 0
```

Allocation[0][1]: 1
Allocation[0][2]: 0
Allocation[1][0]: 2
Allocation[1][1]: 0
Allocation[1][2]: 0
Allocation[2][0]: 3
Allocation[2][1]: 0
Allocation[2][2]: 3
Allocation[3][0]: 2
Allocation[3][1]: 1
Allocation[3][2]: 1
Allocation[4][0]: 0
Allocation[4][1]: 0
Allocation[4][2]: 2

Enter the Maximum Matrix:
Maximum[0][0]: 7
Maximum[0][1]: 5
Maximum[0][2]: 3
Maximum[1][0]: 3
Maximum[1][1]: 2
Maximum[1][2]: 2
Maximum[2][0]: 9
Maximum[2][1]: 0
Maximum[2][2]: 2
Maximum[3][0]: 4
Maximum[3][1]: 2
Maximum[3][2]: 2
Maximum[4][0]: 5
Maximum[4][1]: 3
Maximum[4][2]: 3

Enter the Available Resources:
Available[0]: 3
Available[1]: 3
Available[2]: 2

Safe Sequence is: P1 P3 P4 P0 P2

Result:

The safe sequence for the given processes has been successfully found using Banker's algorithm for deadlock avoidance and the output has been verified successfully.