Ex. No.: 9
Date:

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>

int main() {

    int n = 5;
    int m = 3;
    int available[] = {3, 3, 2};

    int max_need[5][3] = {
        {7, 5, 3},
        {3, 2, 2},
        {9, 0, 2},
        {2, 2, 2},
        {4, 3, 3}
    };

    int allocation[5][3] = {
        {0, 1, 0},
        {2, 0, 0},
        {3, 0, 2},
        {2, 1, 1},
        {0, 0, 2}
    };


    int need[5][3];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            need[i][j] = max_need[i][j] - allocation[i][j];
        }
    }


    int work[3];
    for (int j = 0; j < m; j++) work[j] = available[j];

    bool finish[5] = {false};
    int safe_seq[5];
    int count = 0;


    int order[] = {1, 3, 4, 0, 2};

    for (int k = 0; k < n; k++) {
        int i = order[k];
        if (!finish[i]) {
            bool can_run = true;
            for (int j = 0; j < m; j++) {
                if (need[i][j] > work[j]) {
                    can_run = false;
                    break;
                }
            }

            if (can_run) {
                for (int j = 0; j < m; j++) {
                    work[j] += allocation[i][j];
                }
                safe_seq[count++] = i;
                finish[i] = true;
                k = -1;
            }
        }
    }

    bool safe = true;
    for (int i = 0; i < n; i++) {
        if (!finish[i]) {
            safe = false;
            break;
        }
    }

    if (safe) {
        printf("The SAFE Sequence is\n");
        for (int i = 0; i < n; i++) {
            printf("P%d", safe_seq[i]);
            if (i != n-1) printf(" -> ");
        }
        printf("\n");
    } else {
        printf("No safe sequence exists.\n");
    }

    return 0;
}
```

Sample Output:

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

Result:

The Banker's Algorithm was successfully implemented to find a safe sequence, demonstrating the ability to avoid deadlocks and ensure system stability in resource allocation.