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 Needi<= 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_PROCESSES 5 #define MAX_RESOURCES 3

// Function to find a process that can be executed bool isSafe(int processes[], int avail[],
int max[][MAX_RESOURCES], int allot[][MAX_RESOURCES], int n, int m, int safeSeq[])
{ int finish[n], work[m];

// Initialize work and finish arrays
for (int i = 0; i < m; i++) {
    work[i] = avail[i];
}

for (int i = 0; i < n; i++) {
    finish[i] = 0;
```

```
}

int count = 0; // Count of processes that are finished
while (count < n) {
    bool found = false;
    for (int p = 0; p < n; p++) {
        // If process p is not finished and needs can be satisfied with current work
        if (finish[p] == 0) {
            int canFinish = 1;
            for (int j = 0; j < m; j++) {
                if (max[p][j] - allot[p][j] > work[j]) {
                    canFinish = 0; // If process cannot finish, break
                    break;
                }
            }

            if (canFinish) {
                // Add allocation of process to work
                for (int j = 0; j < m; j++) {
                    work[j] += allot[p][j];
                }
                safeSeq[count++] = p; // Add process to safe sequence
                finish[p] = 1; // Mark process p as finished
                found = true;
                break;
            }
        }
    }

    // If no process was found that could be executed, then system is in unsafe state
    if (!found) {
        return false; // No safe sequence found
    }
}

return true; // Safe sequence found


}
```

```
int main() { // Number of processes and resources int n = 5; // Number of processes int
m = 3; // Number of resources

// Available resources
int avail[] = {3, 3, 2};

// Maximum demand of each process
int max[5][3] = {
    {7, 5, 3},
    {3, 2, 2},
    {9, 0, 2},
    {2, 2, 2},
    {4, 3, 3}
};

// Allocation of resources for each process
int allot[5][3] = {
    {0, 1, 0},
    {2, 0, 0},
    {3, 0, 2},
    {2, 1, 1},
    {0, 0, 2}
};

// Safe sequence array
int safeSeq[n];

// Check if a safe sequence exists
if (isSafe(allot, avail, max, allot, n, m, safeSeq)) {
    printf("Safe sequence: ");
    for (int i = 0; i < n; i++) {
        printf("P%d ", safeSeq[i]);
    }
} else {
    printf("No safe sequence exists. Deadlock may occur.");
}

return 0;


}
```

OUTPUT:

Safe sequence: P0 P1 P3 P4 P2

RESULT:

Program To find out a safe sequence using Banker’s algorithm for deadlock avoidance is executed successfully.