Harsha Vardhinii.T

230701109

Ex.No.9: Deadlock Avoidance

Aim:

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

Code:

```
#include <stdio.h>
#include <stdbool.h>
int main() {
   int n, r;
   // Step 1: Get input for processes and resources
   printf("Enter number of processes: ");
   scanf("%d", &n);
   printf("Enter number of resource types: ");
   scanf("%d", &r);
   int Allocation[n][r], Max[n][r], Need[n][r], Available[r];
   // Allocation Matrix Input
   printf("Enter Allocation Matrix:\n");
   for (int i = 0; i < n; i++)
      for (int j = 0; j < r; j++)
         scanf("%d", &Allocation[i][j]);
   // Max Matrix Input
   printf("Enter Max Matrix:\n");
   for (int i = 0; i < n; i++)
      for (int j = 0; j < r; j++)
         scanf("%d", &Max[i][j]);
   // Available Resources
   printf("Enter Available Resources:\n");
   for (int j = 0; j < r; j++)
      scanf("%d", &Available[j]);
```

```
// Step 2: Calculate Need = Max - Allocation
for (int i = 0; i < n; i++)
   for (int j = 0; j < r; j++)
      Need[i][j] = Max[i][j] - Allocation[i][j];
// Step 3: Initialize Work = Available and Finish[i] = false
int Work[r];
for (int j = 0; j < r; j++)
   Work[j] = Available[j];
bool Finish[n];
for (int i = 0; i < n; i++)
   Finish[i] = false;
int SafeSequence[n], count = 0;
// Step 4: Find a process that can be completed
while (count < n) {
   bool found = false;
   for (int i = 0; i < n; i++) {
      if (!Finish[i]) {
         bool canRun = true;
         for (int j = 0; j < r; j++) {
            if (Need[i][j] > Work[j]) {
               canRun = false;
               break;
            }
         }
         if (canRun) {
            // Step 5: Work = Work + Allocation[i]
            for (int j = 0; j < r; j++)
               Work[j] += Allocation[i][j];
            // Mark process as finished
            Finish[i] = true;
```

```
                SafeSequence[count++] = i;
                found = true;
            }
        }
    }
    if (!found) {
        // Step 6: No such process found
        printf("\nNo safe sequence exists. System is in unsafe state.\n");
        return 0;
    }
  }
  // Step 7: All processes are finished
  printf("\nSystem is in a SAFE state.\nSafe sequence is: ");
  for (int i = 0; i < n; i++)
    printf("P%d ", SafeSequence[i]);
  printf("\n");
  return 0;
}
```

Input:

Enter number of processes: 5

Enter number of resource types: 3

Enter Allocation Matrix:

0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

Enter Max Matrix:

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

Enter Available Resources:

3 3 2

Output:

System is in a SAFE state.

Safe sequence is: P1 P3 P4 P0 P2