K NARENDRA-192321162

1. Illustrate the deadlock avoidance concept by simulating Banker’s algorithm with C.

# AIM:

To illustrate the concept of deadlock avoidance by simulating the Banker's Algorithm in C, ensuring system safety by allocating resources only when a safe sequence exists.

# ALGORITHM:

* 1. **Input Data:**
     + Read the number of processes (n) and resources (m).
     + Input the Allocation matrix, Max matrix, and Available resources.

# Calculate Need Matrix:

* + - Compute the Need matrix using the formula: Need[i][j] = Max[i][j] - Allocation[i][j].

# Initialize Variables:

* + - Set Work = Available resources.
    - Set Finish array to false for all processes.
    - Initialize an empty Safe Sequence array.

# Find a Process to Allocate:

* + - Search for an unfinished process i such that: Need[i][j] <= Work[j] for all j.

# Allocate Resources if Safe:

* + - If such a process is found:
      * Add the allocated resources of i to Work: Work[j] += Allocation[i][j] for all j.
      * Mark i as finished (Finish[i] = true).
      * Add i to the Safe Sequence array.

# Repeat Allocation Check:

* + - Continue steps 4 and 5 until either all processes are finished or no suitable process is found.

# Check System State:

* + - If all processes are marked finished, the system is in a **safe state**, and the safe sequence is printed.
    - If not, the system is in an **unsafe state**, and no safe sequence exists.

# PROCEDURE:

1. **Start:**

Initialize variables to store the Allocation matrix, Max matrix, Available resources, and the Need matrix.

# Input Data:

* + Enter the number of processes (n) and resources (m).
  + Input the Allocation matrix, Max matrix, and Available resources.

# Calculate Need Matrix:

Compute Need[i][j] for each process and resource using the formula: Need[i][j] = Max[i][j] - Allocation[i][j].

# Initialize Safety Check:

* + Set Work = Available.
  + Mark all processes in the Finish array as false.

# Allocate Resources:

* + Find an unfinished process i such that Need[i][j] <= Work[j] for all resources j.
  + If found:
    - Add Allocation[i][j] to Work[j] for all j.
    - Mark Finish[i] = true.
    - Add the process to the safe sequence.

# Repeat Allocation:

Repeat Step 5 until all processes are marked finished or no suitable process is found.

# Check System Safety:

* + If all processes are marked Finish = true, the system is in a safe state. Print the safe sequence.
  + Otherwise, declare the system to be in an unsafe state.

# Stop:

End the procedure.

CODE:

#include <stdio.h>

#include <stdbool.h>

int main() { int n, m;

printf("Enter number of processes and resources: "); scanf("%d %d", &n, &m);

int allocation[n][m], max[n][m], available[m], need[n][m], work[m], finish[n];

printf("Enter Allocation Matrix: \n"); for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) {

scanf("%d", &allocation[i][j]);

}

}

printf("Enter Max Matrix: \n"); for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) { scanf("%d", &max[i][j]);

}

}

printf("Enter Available Resources: \n");

for (int j = 0; j < m; j++) { scanf("%d", &available[j]);

}

for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) {

need[i][j] = max[i][j] - allocation[i][j];

}

}

for (int i = 0; i < m; i++) { work[i] = available[i];

}

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

}

int safeSequence[n], index = 0; bool found;

do {

found = false;

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

bool canAllocate = true; for (int j = 0; j < m; j++) {

if (need[i][j] > work[j]) {

canAllocate = false; break;

}

}

if (canAllocate) {

for (int j = 0; j < m; j++) {

work[j] += allocation[i][j];

}

safeSequence[index++] = i; finish[i] = 1;

found = true;

}

}

}

} while (found);

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

printf("The system is in an unsafe state.\n"); return 0;

}

}

printf("The system is in a safe state. Safe sequence is: "); for (int i = 0; i < n; i++) {

printf("P%d ", safeSequence[i]);

}

return 0;

}

OUTPUT:

