**PROGRAM 5**

**Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.**

**ALGORITHM**

1. Input:

* Read the number of processes (processCount) and resources (resourceCount).
* Read the total available units for each resource type (available[]).
* Read the maximum resource needs of each process (max[][]).
* Read the resources currently allocated to each process (allocation[][]).

1. Calculate the Need Matrix:

* For each process i and resource j, calculate the need[i][j] as: need[i][j]=max[i][j]−allocation[i][j]need[i][j] = max[i][j] - allocation[i][j]need[i][j]=max[i][j]−allocation[i][j]

1. Initialize Work and Finish Arrays:

* Set the work[] array to be the same as available[], representing the currently available resources.
* Initialize the finish[] array for each process to 'f' (false), indicating no process is finished.

1. Find a Safe Sequence:

* For each process (up to processCount iterations), repeat the following steps to find a sequence in which processes can finish safely:
  + Check each process i:
    - If finish[i] is 'f' (false), check if all resource needs of the process can be met (i.e., need[i][j] <= work[j] for each resource j).
    - If all resources needed by process i are available:
      * Mark finish[i] as 't' (true), indicating the process can safely complete.
      * Add the allocated resources of process i back to work[] for other processes to use.
      * Record the process in the safeSequence[].
      * Break the loop to restart with the updated resources.

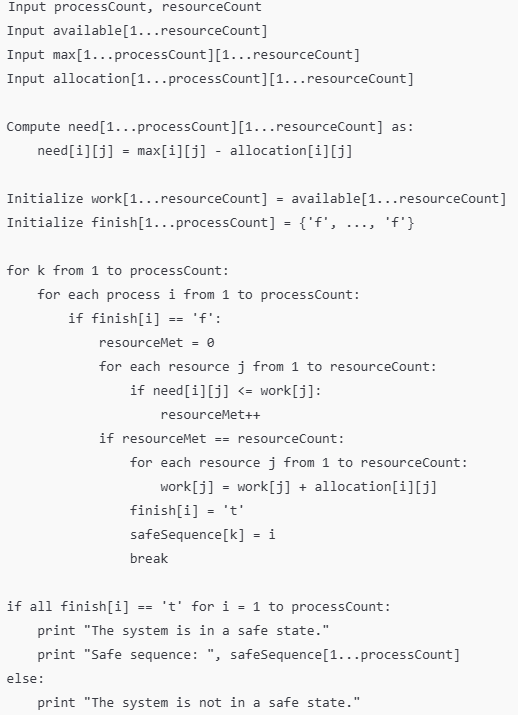
1. Check for Safe State:

* If all processes are marked as finished (all finish[i] are 't'), then the system is in a safe state, and safeSequence[] contains the safe sequence.
* If not all processes are finished, the system is not in a safe state.

1. Output:

* If the system is in a safe state, output the safe sequence (safeSequence[]).
* If the system is not in a safe state, indicate that the system is unsafe.

**PSEUDOCODE:**

****

**SOURCE CODE:**

#include <stdio.h>

void main() {

int work[10], available[10], allocation[10][10], need[10][10];

int max[10][10], processCount, resourceCount;

int i, j, k, safeSequence[10], finishCount = 0;

char finish[10] = {'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f'};

printf("\nEnter the number of processes: ");

scanf("%d", &processCount);

printf("\nEnter the number of resources: ");

scanf("%d", &resourceCount);

printf("\nEnter the total number of resources for each type:\n");

for (i = 1; i <= resourceCount; i++) {

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

}

printf("\nEnter the maximum resources required by each process:\n");

for (i = 1; i <= processCount; i++) {

for (j = 1; j <= resourceCount; j++) {

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

}

}

printf("\nEnter the allocated resources for each process:\n");

for (i = 1; i <= processCount; i++) {

for (j = 1; j <= resourceCount; j++) {

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

}

}

// Calculate the need matrix

for (i = 1; i <= processCount; i++) {

for (j = 1; j <= resourceCount; j++) {

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

}

}

// Initialize work array with available resources

for (i = 1; i <= resourceCount; i++) {

work[i] = available[i];

}

// Check for a safe sequence

for (k = 1; k <= processCount; k++) {

for (i = 1; i <= processCount; i++) {

int resourceMet = 0;

if (finish[i] == 'f') {

for (j = 1; j <= resourceCount; j++) {

if (need[i][j] <= work[j]) {

resourceMet++;

}

}

if (resourceMet == resourceCount) { // All resources for process met

for (j = 1; j <= resourceCount; j++) {

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

}

finish[i] = 't';

safeSequence[k] = i;

finishCount++;

break;

}

}

}

}

if (finishCount == processCount) {

printf("\nThe system is in a safe state.\nSafe sequence: ");

for (i = 1; i <= processCount; i++) {

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

}

} else {

printf("\nThe system is not in a safe state.");

}

}

**Expected Output:**

Enter the number of processes: 3

Enter the number of resources: 3

Enter the total number of resources for each type:

10 5 7

Enter the maximum resources required by each process:

7 5 3

3 2 2

9 0 2

Enter the allocated resources for each process:

0 1 0

2 0 0

3 0 2

The system is in a safe state. Safe sequence: 2 1 3