**EX 17: 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 using a C program.

**Algorithm:**

1. **Input Resources and Processes**:
   * Number of processes and resources.
   * Max matrix: Maximum resources a process may request.
   * Allocated matrix: Resources already allocated to each process.
   * Available array: Resources currently available.
2. **Calculate Need Matrix**:
   * Need[i][j]=Max[i][j]−Allocated[i][j]\text{Need[i][j]} = \text{Max[i][j]} - \text{Allocated[i][j]}Need[i][j]=Max[i][j]−Allocated[i][j].
3. **Check Safe State**:
   * Use a safety algorithm to determine if the system is in a safe state.
4. **Resource Request**:
   * Process requests resources.
   * Temporarily allocate the resources and check if the system is in a safe state.
   * Grant the request only if the system remains safe.

**Program:**

#include <stdio.h>

#include <stdbool.h>

#define P 5

#define R 3

int allocation[P][R] = { {0, 1, 0},

{2, 0, 0},

{3, 0, 2},

{2, 1, 1},

{0, 0, 2} };

int max[P][R] = { {7, 5, 3},

{3, 2, 2},

{9, 0, 2},

{2, 2, 2},

{4, 3, 3} };

int need[P][R];

int available[R] = {3, 3, 2};

void calculateNeed() {

for (int i = 0; i < P; i++)

for (int j = 0; j < R; j++)

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

}

bool isSafe() {

int work[R];

bool finish[P] = {0};

for (int i = 0; i < R; i++)

work[i] = available[i];

int count = 0;

while (count < P) {

bool found = false;

for (int p = 0; p < P; p++) {

if (!finish[p]) {

int j;

for (j = 0; j < R; j++)

if (need[p][j] > work[j])

break;

if (j == R) {

for (int k = 0; k < R; k++)

work[k] += allocation[p][k];

finish[p] = 1;

found = true;

count++;

}

}

}

if (!found)

return false;

}

return true;

}

int main() {

calculateNeed();

if (isSafe())

printf("System is in a safe state.\n");

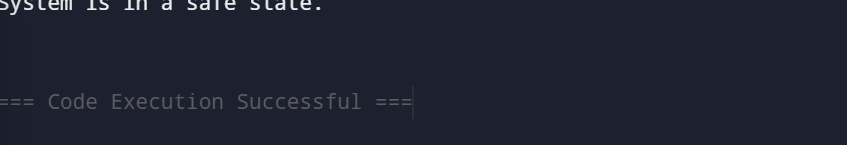
else

printf("System is not in a safe state.\n");

return 0;

}

**OUTPUT:**

****