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

#include <stdio.h>

#include <stdbool.h>

#define P 5 // Number of processes

#define R 3 // Number of resources

int allocation[P][R] = { {0, 1, 0}, {2, 0, 0}, {3, 0, 2}, {2, 1, 1}, {0, 0, 2} };

int max[P][R] = { {0, 1, 0}, {2, 0, 2}, {3, 0, 2}, {2, 1, 1}, {0, 0, 2} };

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

int need[P][R];

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];

while (1) {

bool found = false;

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

if (!finish[p]) {

bool canAllocate = true;

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

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

canAllocate = false;

break;

}

if (canAllocate) {

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

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

finish[p] = true;

found = true;

}

}

}

if (!found) break;

}

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

if (!finish[i]) 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;

}