safesequence.c

#include <stdio.h>

#include <stdbool.h>

#define P 5 // Processes

#define R 3 // Resources

int main() {

// Maximum resources needed by each process

int max[P][R] = {

{7, 5, 3},

{3, 2, 2},

{9, 0, 2},

{2, 2, 2},

{4, 3, 3}

};

// Resources allocated to each process

int alloc[P][R] = {

{0, 1, 0},

{2, 1, 1},

{3, 2, 2},

{2, 1, 1},

{0, 0, 2}

};

// Available resources

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

// Need matrix: Max - Allocated

int need[P][R];

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

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

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

// Work vector and Finish array

int work[R];

bool finish[P] = {0}; // All processes initially unfinished

for (int i = 0; i < R; i++) work[i] = avail[i];

// Safe sequence array

int safeSeq[P], idx = 0;

// Banker's algorithm to find safe sequence

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

bool found = false;

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

if (!finish[i]) {

bool canFinish = true;

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

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

canFinish = false;

break;

}

if (canFinish) {

// Update work and mark process as finished

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

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

finish[i] = true;

safeSeq[idx++] = i;

found = true;

break;

}

}

}

if (!found) { printf("System is in an unsafe state.\n"); return 0; }

}

// If all processes finished, print the safe sequence

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

for (int i = 0; i < P; i++) printf("P%d ", safeSeq[i]);

printf("\n");

return 0;

}

System is in a safe state.

Safe sequence: P1 P3 P0 P2 P4