Banker

#include <stdio.h>

#include <stdbool.h>

// Define number of processes and resources

#define P 5 // Number of processes

#define R 3 // Number of resource types

int main() {

// Allocation Matrix: currently allocated resources

int alloc[P][R] = {

{0, 1, 0}, // P0

{2, 0, 0}, // P1

{3, 0, 2}, // P2

{2, 1, 1}, // P3

{0, 0, 2} // P4

};

// Maximum Matrix: max demand of each process

int max[P][R] = {

{7, 5, 3}, // P0

{3, 2, 2}, // P1

{9, 0, 2}, // P2

{2, 2, 2}, // P3

{4, 3, 3} // P4

};

// Available Resources

int avail[R] = {3, 3, 2}; // Initially available resources

int need[P][R]; // Need matrix

bool finish[P] = {false}; // Tracks if process is finished

int safeSeq[P]; // Stores the safe sequence

int count = 0; // Count of finished processes

// Step 1: Calculate the Need matrix = Max - Alloc

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

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

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

}

}

// Step 2: Apply Banker's Algorithm

while (count < P) {

bool found = false;

// Try to find a process whose need can be satisfied

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

if (!finish[i]) { // Process not finished

bool canAllocate = true;

// Check if all needed resources are available

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

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

canAllocate = false;

break;

}

}

// If resources can be allocated to process i

if (canAllocate) {

// Simulate process execution → release its allocated resources

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

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

}

// Add this process to safe sequence

safeSeq[count++] = i;

finish[i] = true; // Mark process as finished

found = true; // A process was able to finish

}

}

}

// If no process could be allocated in this cycle → unsafe state

if (!found) {

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

return 0;

}

}

// Step 3: If all processes are finished, print the safe sequence

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

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

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

}

printf("\n");

return 0;

}

START

1. Input number of processes (P) and number of resource types (R)

2. Input:

a. Allocation matrix: alloc[P][R]

b. Maximum matrix: max[P][R]

c. Available resources: avail[R]

3. Calculate the Need matrix:

for i = 0 to P-1:

for j = 0 to R-1:

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

4. Initialize:

a. finish[i] = false for all processes

b. safeSeq = empty list

c. count = 0

5. While count < P:

a. Set found = false

b. For each process i from 0 to P-1:

i. If finish[i] == false:

- Check if need[i][j] ≤ avail[j] for all j from 0 to R-1

- If true:

• For each resource j:

→ avail[j] = avail[j] + alloc[i][j]

• Mark finish[i] = true

• Add process i to safeSeq

• Increment count

• Set found = true

c. If found == false:

- Print "System is not in a safe state"

- Exit

6. If all processes are finished (count == P):

- Print "System is in a safe state"

- Print "Safe Sequence = safeSeq"

END