**Sachin Chaudhary**

**230701402**

**EX NO. 9:**

**Banker’s Algorithm**

Aim:

To find out a safe sequence using the Banker’s algorithm for deadlock avoidance.

Algorithm:

Initialize work = available and finish[i] = false for all values of i.

Find an i such that both:

finish[i] = false

Need[i] <= work (where Need[i] is the remaining resource requirement for process i)

If no such i exists, go to step 6.

Compute work = work + allocation[i] (i.e., add the allocated resources of process i to work).

Assign finish[i] to true and go to step 2.

If finish[i] == true for all i, then print the safe sequence.

Else, print "there is no safe sequence."

Program Code:

#include <stdio.h>

#include <stdbool.h>

#define P 5 // Number of processes

#define R 3 // Number of resources

void calculateSafeSequence(int available[], int allocation[][R], int maximum[][R]) {

int need[P][R];

bool finish[P] = {false};

int work[R];

int safeSequence[P];

int count = 0;

// Calculate Need matrix (Need[i][j] = Maximum[i][j] - Allocation[i][j])

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

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

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

}

}

// Initialize work[] as available[]

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

work[i] = available[i];

}

// Find the safe sequence

while (count < P) {

bool found = false;

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

if (!finish[i]) {

int j;

// Check if all needed resources are available for process i

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

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

break;

}

}

if (j == R) { // All needed resources are available

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

work[k] += allocation[i][k]; // Add allocated resources to work[]

}

finish[i] = true;

safeSequence[count++] = i;

found = true;

break;

}

}

}

if (!found) {

printf("There is no safe sequence\n");

return;

}

}

// Print the safe sequence

printf("The SAFE Sequence is\n");

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

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

}

printf("\n");

}

int main() {

int available[] = {3, 3, 2}; // Available resources

int allocation[P][R] = {

{0, 1, 0},

{2, 0, 0},

{3, 0, 2},

{2, 1, 1},

{0, 0, 2}

};

int maximum[P][R] = {

{7, 5, 3},

{3, 2, 2},

{9, 0, 2},

{2, 2, 2},

{4, 3, 3}

};

calculateSafeSequence(available, allocation, maximum);

return 0;

}

Sample Output:

The SAFE Sequence is

P1 -> P3 -> P4 -> P0 -> P2 ->

Explanation:

The program uses the Banker's algorithm to calculate a safe sequence of processes based on the available resources, the allocation matrix, and the maximum resource requirements of each process.

It checks the condition Need[i] <= work, and if all resources are available for a process, it adds the allocated resources back to the available pool and updates the safe sequence.

If at any point no process can proceed (because its needs can't be met with the available resources), the program reports that no safe sequence exists.

Result:

The program successfully calculates and displays the safe sequence of processes using Banker's algorithm for deadlock avoidance.