



**Program:**

#include <stdio.h>

#include <stdbool.h>

#define P 10

#define R 3

// Create banker's structure

struct Process {

int pNo;

int allocation[R];

int maxNeed[R];

int remainingNeed[R];

};

// Function to print matrices

void printMatrices(struct Process p[], int available[], int n) {

printf("\nProcess\tAllocation\tMax Need\tRemaining Need\n");

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

printf("P%d\t", p[i].pNo);

for(int j = 0; j < R; j++) printf("%d ", p[i].allocation[j]);

printf("\t\t");

for(int j = 0; j < R; j++) printf("%d ", p[i].maxNeed[j]);

printf("\t\t");

for(int j = 0; j < R; j++) printf("%d ", p[i].remainingNeed[j]);

printf("\n");

}

printf("\nAvailable Resources: ");

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

printf("\n");

}

// RN = MN - ALLOC

void calcRemainingNeed(struct Process p[], int n) {

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

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

p[i].remainingNeed[j] = p[i].maxNeed[j] - p[i].allocation[j];

}

}

}

void calcAvailableResource(struct Process p[], int available[], int n) {

int temp[R];

for(int i = 0; i < R; i++) temp[i] = available[i];

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

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

temp[j] -= p[i].allocation[j];

}

}

printf("\nInitial Available Resources: ");

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

printf("\nAvailable after allocation: ");

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

available[i] = temp[i];

printf("%d ", available[i]);

}

printf("\n");

}

bool isSafeState(int safeSeq[], struct Process p[], int available[], int n) {

int count = 0;

bool finish[n];

int work[R];

for (int i = 0; i < n; i++) finish[i] = false;

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

while (count < n) {

bool found = false;

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

if (!finish[i]) {

bool canAllocate = true;

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

if (p[i].remainingNeed[j] > work[j]) {

canAllocate = false;

break;

}

}

if (canAllocate) {

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

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

}

safeSeq[count++] = p[i].pNo;

finish[i] = true;

found = true;

printf("\nExecuting P%d, Work becomes: ", p[i].pNo);

for(int k = 0; k < R; k++) printf("%d ", work[k]);

}

}

}

if (!found) {

return false;

}

}

return true;

}

int main() {

int n = 5;

int available[] = {10, 5, 7};

struct Process process[] = {

{1, {0, 1, 0}, {7, 5, 3}, {0, 0, 0}},

{2, {2, 0, 0}, {3, 2, 2}, {0, 0, 0}},

{3, {3, 0, 2}, {9, 0, 2}, {0, 0, 0}},

{4, {2, 1, 1}, {4, 2, 2}, {0, 0, 0}},

{5, {0, 0, 2}, {5, 3, 3}, {0, 0, 0}},

};

calcRemainingNeed(process, n);

calcAvailableResource(process, available, n);

printMatrices(process, available, n);

int safeSeq[n];

printf("\nSafety Check Sequence:\n");

if (isSafeState(safeSeq, process, available, n)) {

printf("\n\nSystem is in a safe state.\nSafe Sequence: ");

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

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

if(i < n-1) printf("-> ");

}

printf("\n");

} else {

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

}

return 0;

}

**Output:**

