Deadlock detection and Avoidance

#include <stdio.h>

#include <stdbool.h>

#define MAX 10

int processes, resources;

int allocation[MAX][MAX], max[MAX][MAX], need[MAX][MAX];

int request[MAX][MAX];

int available[MAX];

void calculateNeed() {

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

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

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

}

void displayMatrices() {

printf("\nProcess\tAllocation\tMaximum\t\tNeed\n");

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

printf("P%d\t", i);

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

printf("%d ", allocation[i][j]);

printf("\t\t");

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

printf("%d ", max[i][j]);

printf("\t\t");

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

printf("%d ", need[i][j]);

printf("\n");

}

}

bool check(int i, int work[]) {

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

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

return false;

return true;

}

bool isSafe() {

int work[MAX], safeSeq[MAX], count = 0;

bool done[MAX] = {false};

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

work[i] = available[i];

while (count < processes) {

bool found = false;

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

if (!done[p] && check(p, work)) {

for (int k = 0; k < resources; k++)

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

safeSeq[count++] = p;

done[p] = true;

found = true;

}

}

if (!found)

break;

}

if (count < processes) {

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

return false;

}

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

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

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

printf("\n");

return true;

}

void deadlockDetection() {

printf("Enter the allocation matrix:\n");

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

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

scanf("%d", &allocation[i][j]);

printf("Enter the request matrix (currently needed resources):\n");

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

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

scanf("%d", &request[i][j]);

printf("Enter the available resources:\n");

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

scanf("%d", &available[i]);

printf("\nProcess\tAllocation\tRequest (Need)\n");

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

printf("P%d\t", i);

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

printf("%d ", allocation[i][j]);

printf("\t\t");

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

printf("%d ", request[i][j]);

printf("\n");

}

bool finish[MAX] = {false};

int work[MAX];

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

work[i] = available[i];

int count = 0;

while (count < processes) {

bool found = false;

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

if (!finish[i]) {

bool canProceed = true;

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

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

canProceed = false;

break;

}

}

if (canProceed) {

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

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

finish[i] = true;

found = true;

count++;

}

}

}

if (!found)

break;

}

bool deadlock = false;

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

if (!finish[i]) {

deadlock = true;

break;

}

}

if (deadlock)

printf("\nSystem is in deadlock.\n");

else

printf("\nSystem is in safe state.\n");

}

int main() {

int choice;

printf("Enter the number of processes: ");

scanf("%d", &processes);

printf("Enter the number of resources: ");

scanf("%d", &resources);

printf("\nChoose option:\n1. Deadlock Avoidance (Banker's Algorithm)\n2. Deadlock Detection\nChoice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the allocation matrix:\n");

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

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

scanf("%d", &allocation[i][j]);

printf("Enter the maximum resource matrix:\n");

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

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

scanf("%d", &max[i][j]);

printf("Enter the available resources:\n");

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

scanf("%d", &available[i]);

calculateNeed();

displayMatrices();

isSafe();

break;

case 2:

deadlockDetection();

break;

default:

printf("Invalid choice.\n");

}

return 0;

}



