**Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.**

#include <stdio.h>

#define MAX\_PROCESSES 10

#define MAX\_RESOURCES 10

int available[MAX\_RESOURCES];

int max[MAX\_PROCESSES][MAX\_RESOURCES];

int allocation[MAX\_PROCESSES][MAX\_RESOURCES];

int need[MAX\_PROCESSES][MAX\_RESOURCES];

int num\_processes, num\_resources;

void initialize() {

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

scanf("%d", &num\_processes);

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

scanf("%d", &num\_resources);

printf("Enter the available resources for each type: ");

for (int j = 0; j < num\_resources; j++) {

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

}

printf("Enter the maximum demand of each process: \n");

for (int i = 0; i < num\_processes; i++) {

printf("Process %d: ", i);

for (int j = 0; j < num\_resources; j++) {

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

}

}

printf("Enter the current allocation for each process: \n");

for (int i = 0; i < num\_processes; i++) {

printf("Process %d: ", i);

for (int j = 0; j < num\_resources; j++) {

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

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

}

}

}

int request\_resources(int process\_id, int request[]) {

for (int j = 0; j < num\_resources; j++) {

if (request[j] > need[process\_id][j] || request[j] > available[j]) {

return 0; // Request exceeds maximum allowed or available resources

}

}

// Try to allocate resources temporarily

for (int j = 0; j < num\_resources; j++) {

available[j] -= request[j];

allocation[process\_id][j] += request[j];

need[process\_id][j] -= request[j];

}

// Check for safety

if (is\_safe()) {

return 1; // Resources allocated successfully

} else {

// Rollback changes if not safe

for (int j = 0; j < num\_resources; j++) {

available[j] += request[j];

allocation[process\_id][j] -= request[j];

need[process\_id][j] += request[j];

}

return 0; // Resources cannot be allocated safely

}

}

int release\_resources(int process\_id, int release[]) {

for (int j = 0; j < num\_resources; j++) {

if (release[j] > allocation[process\_id][j]) {

return 0; // Trying to release more resources than allocated

}

}

// Release resources

for (int j = 0; j < num\_resources; j++) {

available[j] += release[j];

allocation[process\_id][j] -= release[j];

need[process\_id][j] += release[j];

}

return 1; // Resources released successfully

}

int is\_safe() {

int work[MAX\_RESOURCES];

int finish[MAX\_PROCESSES] = {0};

// Initialize work array

for (int j = 0; j < num\_resources; j++) {

work[j] = available[j];

}

// Find an index i such that both finish[i] is false and need[i] is less than or equal to work

int i, j;

for (i = 0; i < num\_processes; i++) {

if (!finish[i]) {

for (j = 0; j < num\_resources; j++) {

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

break;

}

}

if (j == num\_resources) {

break; // Found a process that can complete

}

}

}

if (i == num\_processes) {

return 1; // All processes can complete safely

} else {

return 0; // Deadlock detected

}

}

int main() {

initialize();

int process\_id;

printf("Enter the process id making the request: ");

scanf("%d", &process\_id);

int request[MAX\_RESOURCES];

printf("Enter the request for resources: ");

for (int j = 0; j < num\_resources; j++) {

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

}

if (request\_resources(process\_id, request)) {

printf("Resources allocated successfully.\n");

} else {

printf("Request denied. Resources cannot be allocated safely.\n");

}

return 0;

}

**OUTPUT**

Enter the number of processes: 3

Enter the number of resources: 4

Enter the available resources for each type: 3 2 2 1

Enter the maximum demand of each process:

Process 0: 7 5 3 2

Process 1: 3 2 2 2

Process 2: 9 0 2 2

Enter the current allocation for each process:

Process 0: 0 1 0 0

Process 1: 2 0 0 1

Process 2: 3 0 2 2

Enter the process id making the request: 1

Enter the request for resources: 0 1 0 1

Resources allocated successfully.