```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int canalloc = 0;
void findsafesequence(int n, int m, int alloc[n][m], int max[n][m], int avail[m], int need[n][m]) {
    int y = 0;
    int f[n], ans[n], ind = 0;
    // Initialize finish array to 0
    for (int k = 0; k < n; k++) {
        f[k] = 0;
    }
for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            if (f[i] == 0) { // Process i has not finished
                int flag = 0;
                for (int j = 0; j < m; j++) {
                    if (need[i][j] > avail[j]) {
                        flag = 1; // If resources can't be allocated to process i
                        break;
                    }
                }
                if (flag == 0) { // If process i can be allocated resources
                    ans[ind++] = i; // Add process i to safe sequence
                    for (int y = 0; y < m; y++) {
                        avail[y] += alloc[i][y]; // Release resources after process finishes
                    }
                    f[i] = 1; // Mark process as finished
                }
            }
        }
    }
 int flag = 1;
    for (int i = 0; i < n; i++) {
        if (f[i] == 0) { // If some process could not finish
            canalloc = -1;
            flag = 0;
            printf("The system is not in a safe state.\n");
            break;
        }
    }
if (flag == 1) {
        canalloc = 1;
        printf("The system is in a safe state.\n");
        printf("Safe Sequence: ");
        for (int i = 0; i < n - 1; i++) {
            printf("P%d -> ", ans[i]);
        }
        printf("P%d\n", ans[n - 1]);

        printf("\nFinal Available matrix\n");
        for (int i = 0; i < m; i++) {
            printf("%d ", avail[i]);
        }
        printf("\n");
    }
}
int main() {
    int n, m;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter the number of resources: ");
    scanf("%d", &m);
    int alloc[n][m], max[n][m], avail[m];
    printf("Enter the Allocation Matrix (%dx%d): \n", n, m);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            scanf("%d", &alloc[i][j]);
        }
    }
    printf("Enter the Maximum Matrix (%dx%d): \n", n, m);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            scanf("%d", &max[i][j]);
        }
    }
    printf("Enter the Available Resources for %d resources: \n", m);
    for (int i = 0; i < m; i++) {
        scanf("%d", &avail[i]);
    }
    int need[n][m];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }
    printf("The need matrix is:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            printf("%d\t", need[i][j]);
        }
        printf("\n");
    }
     int ch;
    printf("1. Resource request\n");
    printf("2. Safe sequence\n");
    printf("Enter your choice: ");
    scanf("%d", &ch);
     switch (ch) {
        case 1: {
            printf("Enter the process ID for which the request is made: ");
            int pid;
            scanf("%d", &pid);
            printf("Enter the resource request for process %d: ", pid);
            int req[m];
            for (int i = 0; i < m; i++) {
                scanf("%d", &req[i]);
            }
            // Check if request can be granted
            for (int i = 0; i < m; i++) {
                if (req[i] > need[pid][i]) {
                    printf("Error: Process has exceeded its maximum claim.\n");
                    return 0;
                }
                if (req[i] > avail[i]) {
                    printf("Resources are not available. Process must wait.\n");
                    return 0;
                }
            }
           // Temporarily allocate the resources and check if safe
            for (int i = 0; i < m; i++) {
                avail[i] -= req[i];
                alloc[pid][i] += req[i];
                need[pid][i] -= req[i];
            }
            findsafesequence(n, m, alloc, max, avail, need);
            if (canalloc == -1) {
                printf("The resource request cannot be granted immediately.\n");
             } else {
                printf("The resource request can be granted immediately.\n");
            }
            break;
        }
        case 2:
            findsafesequence(n, m, alloc, max, avail, need);
            break;
        case 3:break;
        default:
            printf("Invalid choice\n");
            break;
       }
     return 0;
}
```