Assignment 5

```
#include <stdio.h>
#include <stdbool.h>

#define MAX 10

// Function prototypes
void calculateNeed(int need[MAX][MAX], int max[MAX][MAX], int allot[MAX][MAX], int p, int r);
bool isSafe(int processes[], int avail[], int max[][MAX], int allot[][MAX], int need[][MAX], int n, int m);
bool requestResources(int processes[], int avail[], int max[][MAX], int allot[][MAX], int pid, int request[], int n, int m);

int main() {
    int processes[MAX], n, m;
    int max[MAX][MAX], allot[MAX][MAX], avail[MAX], need[MAX][MAX];

    int choice;
    int pid, request[MAX];

    while (1) {
        printf("\n--- Banker's Algorithm ---\n");
        printf("1. Initialize System\n");
        printf("2. Request Resources\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
```

```
scanf("%d", &choice);


switch (choice) {
    case 1:
        // Input number of processes and resources
        printf("Enter number of processes: ");
        scanf("%d", &n);
        printf("Enter number of resources: ");
        scanf("%d", &m);


        // Input Allocation Matrix
        printf("Enter Allocation Matrix:\n");
        for (int i = 0; i < n; i++) {
            processes[i] = i; // Initialize process ID
            for (int j = 0; j < m; j++) {
                scanf("%d", &allot[i][j]);
            }
        }


        // Input Max Matrix
        printf("Enter Max Matrix:\n");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                scanf("%d", &max[i][j]);
            }
        }
        // Input Available Resources
        printf("Enter Available Resources:\n");
        for (int j = 0; j < m; j++) {
            scanf("%d", &avail[j]);
        }

        // Calculate Need Matrix
        calculateNeed(need, max, allot, n, m);

        // Check for safe state
        if (isSafe(processes, avail, max, allot, need, n, m)) {
            printf("System is in a safe state.\n");
        } else {
            printf("System is not in a safe state.\n");
        }
        break;

    case 2:
        // Request resources
        printf("Enter Process ID to request resources (0 to %d): ", n-1);
        scanf("%d", &pid);
        if (pid < 0 || pid >= n) {
            printf("Invalid Process ID.\n");
            break;
        }
```

```
                printf("Enter resource request for Process %d:\n", pid);
                for (int j = 0; j < m; j++) {
                    scanf("%d", &request[j]);
                }

                if (requestResources(processes, avail, max, allot, pid, request, n, m)) {
                    printf("Resources allocated successfully.\n");
                } else {
                    printf("Resources could not be allocated. Process must wait.\n");
                }
                break;

            case 3:
                printf("Exiting...\n");
                return 0;

            default:
                printf("Invalid choice, please try again.\n");
        }
    }

    return 0;
}

// Function to calculate Need matrix
void calculateNeed(int need[MAX][MAX], int max[MAX][MAX], int allot[MAX][MAX], int p, int r) {
    for (int i = 0; i < p; i++) {
        for (int j = 0; j < r; j++) {
            need[i][j] = max[i][j] - allot[i][j];
        }
    }
}

// Function to check if the system is in a safe state
bool isSafe(int processes[], int avail[], int max[][MAX], int allot[][MAX], int need[][MAX], int n, int m) {
    int finish[MAX] = {0};
    int safeSeq[MAX];
    int work[MAX];
    for (int i = 0; i < m; i++) {
        work[i] = avail[i];
    }

    int count = 0;
    while (count < n) {
        bool found = false;
        for (int p = 0; p < n; p++) {
            if (!finish[p]) {
                int j;
                for (j = 0; j < m; j++) {
                    if (need[p][j] > work[j]) {
                        break;
                    }
                }
                if (j == m) { // If all needs can be satisfied
                    for (int k = 0; k < m; k++) {
                        work[k] += allot[p][k];
                    }
                    safeSeq[count++] = p;
                    finish[p] = 1;
                    found = true;
                }
            }
        }
        if (!found) {
            break; // If no process could be found
        }
    }

    if (count == n) {
        printf("Safe sequence is: ");
        for (int i = 0; i < n; i++) {
            printf("%d ", safeSeq[i]);
        }
        printf("\n");
        return true;
    } else {
        return false;
    }
```

```
}


// Function to request resources
bool requestResources(int processes[], int avail[], int max[][MAX], int allot[][MAX], int pid, int
request[], int n, int m) {
    for (int j = 0; j < m; j++) {
        if (request[j] > max[pid][j]) {
            printf("Error: Process has exceeded its maximum claim.\n");
            return false;
        }
    }

    for (int j = 0; j < m; j++) {
        if (request[j] > avail[j]) {
            printf("Resources are not available, process must wait.\n");
            return false;
        }
    }

    // Pretend to allocate resources
    for (int j = 0; j < m; j++) {
        avail[j] -= request[j];
        allot[pid][j] += request[j];
        max[pid][j] -= request[j];
    }
    // Check if this state is safe

    int need[MAX][MAX];

    calculateNeed(need, max, allot, n, m);

    if (isSafe(processes, avail, max, allot, need, n, m)) {

        return true; // Resources allocated successfully

    } else {

        // Rollback

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

            avail[j] += request[j];

            allot[pid][j] -= request[j];

            max[pid][j] += request[j];

        }

        printf("Request denied, system is not in a safe state.\n");

        return false; // Request cannot be granted

    }

}
```

Example Output:

```
--- Banker's Algorithm ---

1. Initialize System

2. Request Resources

3. Exit

Enter your choice: 1

Enter number of processes: 3

Enter number of resources: 3

Enter Allocation Matrix:
```

```
1 2 2
1 0 3
0 2 1
Enter Max Matrix:
3 2 2
1 1 4
2 2 2
Enter Available Resources:
2 3 2
System is in a safe state.
Safe sequence is: 1 0 2

Enter your choice: 2
Enter Process ID to request resources (0 to 2): 1
Enter resource request for Process 1:
0 1 1
Resources allocated successfully.
Safe sequence is: 1 0 2

Enter your choice: 3
Exiting...
```