Assignment - 6
OPERATING SYSTEM

Topic: Deadlock Avoidance (Banker's Algorithm)

NAME: Devesh Tulshyan
MCA SEM-3
SECTION A
Roll Number: 22

Q: For deadlock avoidance, write a C program to simulate the Bankers algorithm.

DESCRIPTION:
In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, other waiting processes hold the resources a waiting process has requested, preventing it from changing state again. We refer to this situation as a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach necessitates providing the operating system with additional resources beforehand, as well as information about which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. The system considers the resources currently available, the resources allocated to each process, and the future requests and releases of each process to determine whether to satisfy the current request or delay it. Banker's algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type.

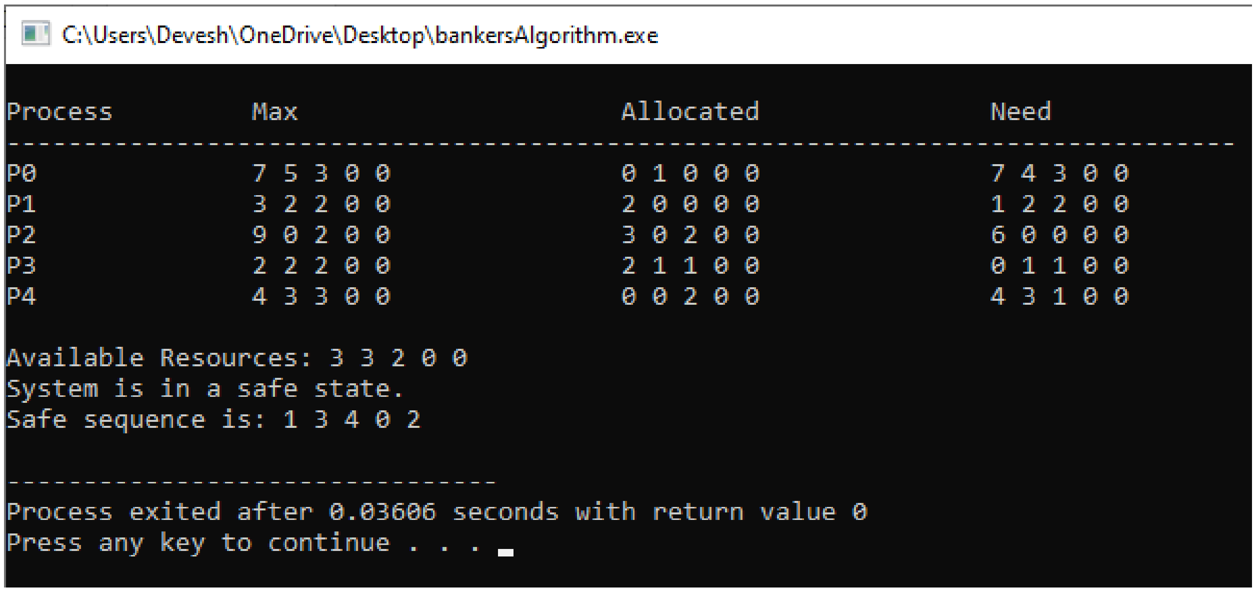
CODE:

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

#define MAX 5

void printTable(int processes, int avail[], int max[][MAX], int alloc[][MAX], int need[][MAX]) {
    printf("\nProcess\t\tMax\t\t\tAllocated\t\tNeed\n");

printf("------------------------------------------------------------------------------------
-\n");

    for (int i = 0; i < processes; i++) {
        printf("P%d\t\t", i);
        for (int j = 0; j < MAX; j++) {
            printf("%d ", max[i][j]);
        }
        printf("\t\t");
        for (int j = 0; j < MAX; j++) {
            printf("%d ", alloc[i][j]);
        }
        printf("\t\t");
        for (int j = 0; j < MAX; j++) {
            printf("%d ", need[i][j]);
        }
        printf("\n");
    }

    printf("\nAvailable Resources: ");
    for (int i = 0; i < MAX; i++) {
        printf("%d ", avail[i]);
    }
    printf("\n");
}

bool isSafeState(int processes, int avail[], int max[][MAX], int alloc[][MAX], int need[][MAX]) {
    int work[MAX];
```

```
bool finish[MAX] = {false};
int safeSequence[MAX];
int count = 0;

for (int i = 0; i < MAX; i++)
    work[i] = avail[i];

while (count < processes) {
    bool found = false;

    for (int p = 0; p < processes; p++) {
        if (!finish[p]) {
            bool canAllocate = true;
            for (int r = 0; r < MAX; r++) {
                if (need[p][r] > work[r]) {
                    canAllocate = false;
                    break;
                }
            }

            if (canAllocate) {
                for (int r = 0; r < MAX; r++)
                    work[r] += alloc[p][r];

                safeSequence[count++] = p;
                finish[p] = true;
                found = true;
            }
        }
    }

    if (!found) {
        printf("System is not in a safe state.\n");
        return false;
    }
}

printf("System is in a safe state.\nSafe sequence is: ");
```

```
    for (int i = 0; i < processes; i++)
        printf("%d ", safeSequence[i]);
    printf("\n");

    return true;
}

void bankersAlgorithm(int processes, int avail[], int max[][MAX], int alloc[][MAX]) {
    int need[MAX][MAX];

    for (int i = 0; i < processes; i++) {
        for (int j = 0; j < MAX; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }

    printTable(processes, avail, max, alloc, need);
    isSafeState(processes, avail, max, alloc, need);
}

int main() {
    int processes = 5; // Number of processes
    int resources = 3; // Number of resources

    int avail[MAX] = {3, 3, 2};
    int max[MAX][MAX] = {
        {7, 5, 3},
        {3, 2, 2},
        {9, 0, 2},
        {2, 2, 2},
        {4, 3, 3}
    };

    int alloc[MAX][MAX] = {
        {0, 1, 0},
        {2, 0, 0},
        {3, 0, 2},
        {2, 1, 1},
        {0, 0, 2}
    };

    bankersAlgorithm(processes, avail, max, alloc);
    return 0;
}
```

OUTPUT:



```
C:\Users\Devesh\OneDrive\Desktop\bankersAlgorithm.exe

Process         Max                     Allocated               Need
--------------------------------------------------------------------------------
P0              7 5 3 0 0               0 1 0 0 0               7 4 3 0 0
P1              3 2 2 0 0               2 0 0 0 0               1 2 2 0 0
P2              9 0 2 0 0               3 0 2 0 0               6 0 0 0 0
P3              2 2 2 0 0               2 1 1 0 0               0 1 1 0 0
P4              4 3 3 0 0               0 0 2 0 0               4 3 1 0 0

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

--------------------------------
Process exited after 0.03606 seconds with return value 0
Press any key to continue . . .
```