NAME: G. GANESH

REG NO: 192373008

EXERICSE-17

Illustrate the deadlock avoidance concept by simulating Banker's algorithm with C.

Aim:

To simulate the Banker's Algorithm in C for deadlock avoidance in a system managing multiple resources.

Algorithm:

1. **InitializeDataStructures:**
   Define arrays for available resources, maximum demand, allocation, and need matrices for all processes.
2. **Input Data:**
   - Read the number of processes and resource types.
   - Input available resources, maximum demand for each process, and current allocation.
   - Compute the need matrix as Need[i][j] = Max[i][j] - Allocation[i][j].
3. **Safety Algorithm:**
   - Initialize a work array with available resources and a finish array with all entries as false.
   - Find a process whose need can be satisfied with available resources (work).
   - If found, allocate resources to that process temporarily and mark it as finished.
   - Repeat until all processes are finished or no such process is found.
4. **Check System State:**
   - If all processes finish, the system is in a safe state.
   - Otherwise, it is in an unsafe state, indicating potential deadlock.
5. **OutputResult:**
   Display the safe sequence if the system is safe; otherwise, report an unsafe state.

Procedure:

1. Input the number of processes and resources.
2. Input data for available, maximum demand, and allocation matrices.
3. Compute the need matrix.
4. Implement the safety algorithm to check if a safe sequence exists.
5. Output the safe sequence or indicate an unsafe state.

Code:

```
#include <stdio.h>

#include <stdbool.h>

#define MAX_PROCESSES 10

#define MAX_RESOURCES 10

void calculate_need(int need[MAX_PROCESSES][MAX_RESOURCES], int max[MAX_PROCESSES][MAX_RESOURCES], int alloc[MAX_PROCESSES][MAX_RESOURCES], int processes, int resources) {
    for (int i = 0; i < processes; i++) {
        for (int j = 0; j < resources; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }
}

bool is_safe(int processes, int resources, int avail[MAX_RESOURCES], int max[MAX_PROCESSES][MAX_RESOURCES], int alloc[MAX_PROCESSES][MAX_RESOURCES]) {
    int need[MAX_PROCESSES][MAX_RESOURCES];
    calculate_need(need, max, alloc, processes, resources);
    bool finish[MAX_PROCESSES] = {0};
    int safe_sequence[MAX_PROCESSES];
    int work[MAX_RESOURCES];
    for (int i = 0; i < resources; i++) {
        work[i] = avail[i];
    }
    int count = 0;
    while (count < processes) {
        bool found = false;
        for (int p = 0; p < processes; p++) {
            if (!finish[p]) {
                bool can_allocate = true;
                for (int r = 0; r < resources; r++) {
```

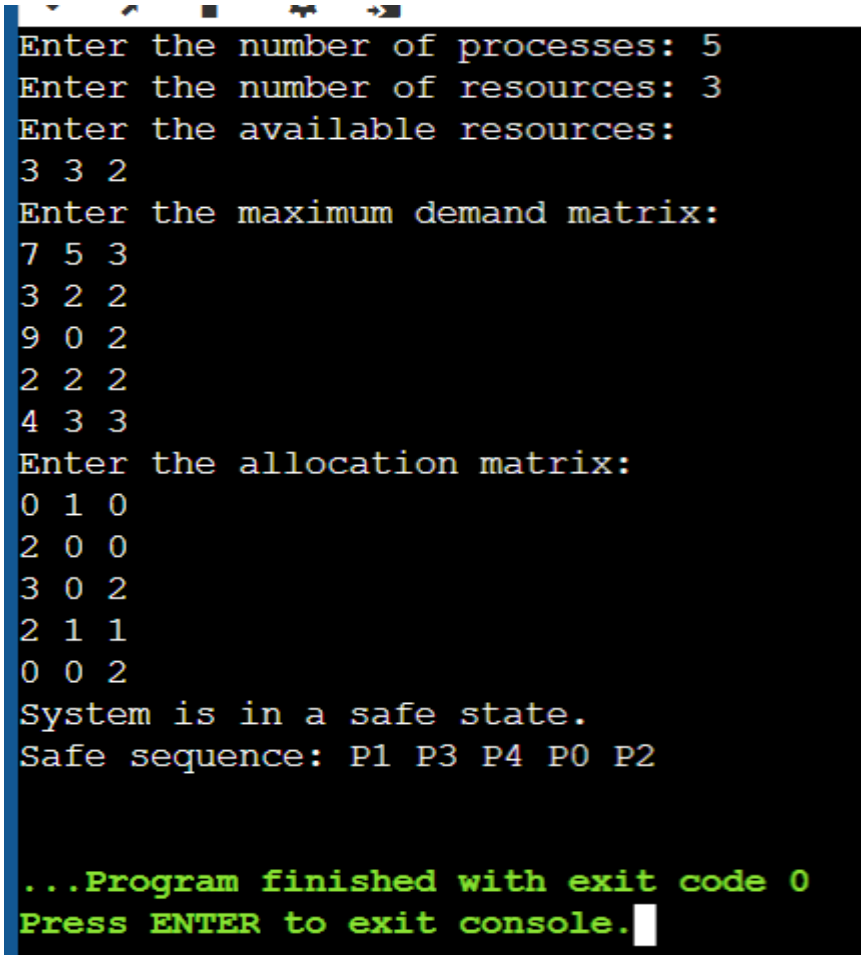
```
                if (need[p][r] > work[r]) {
                    can_allocate = false;
                    break;
                }
            }
            if (can_allocate) {
                for (int r = 0; r < resources; r++) {
                    work[r] += alloc[p][r];
                }
                safe_sequence[count++] = p;
                finish[p] = true;
                found = true;
            }
        }
        if (!found) {
            return false;
        }
    }
    printf("System is in a safe state.\nSafe sequence: ");
    for (int i = 0; i < processes; i++) {
        printf("P%d ", safe_sequence[i]);
    }
    printf("\n");
    return true;
}
int main() {
    int processes, resources;
    int avail[MAX_RESOURCES];
    int max[MAX_PROCESSES][MAX_RESOURCES];
```

```
    int alloc[MAX_PROCESSES][MAX_RESOURCES];
    printf("Enter the number of processes: ");
    scanf("%d", &processes);
    printf("Enter the number of resources: ");
    scanf("%d", &resources);
    printf("Enter the available resources: ");
    for (int i = 0; i < resources; i++) {
        scanf("%d", &avail[i]);
    }
    printf("Enter the maximum demand matrix:\n");
    for (int i = 0; i < processes; i++) {
        for (int j = 0; j < resources; j++) {
            scanf("%d", &max[i][j]);
        }
    }
    printf("Enter the allocation matrix:\n");
    for (int i = 0; i < processes; i++) {
        for (int j = 0; j < resources; j++) {
            scanf("%d", &alloc[i][j]);
        }
    }
    if (!is_safe(processes, resources, avail, max, alloc)) {
        printf("System is in an unsafe state. Deadlock may occur.\n");
    }
    return 0;
}
```

Result:

The program simulates the Banker's Algorithm to determine if the system is in a safe state and outputs a safe sequence if possible.

Output:


Enter the number of processes: 5
Enter the number of resources: 3
Enter the available resources:
3 3 2
Enter the maximum demand matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
System is in a safe state.
Safe sequence: P1 P3 P4 P0 P2
...Program finished with exit code 0
Press ENTER to exit console.