Assignment - 6

OPERATING SYSTEM

Topic: Deadlock Avoidance (Banker's Algorithm)

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.

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

#define MAX_PROCESSES 5
#define MAX_RESOURCES 3

int available[MAX_RESOURCES];
int maximum[MAX_PROCESSES][MAX_RESOURCES];
int allocation[MAX_PROCESSES][MAX_RESOURCES];
int need[MAX_PROCESSES][MAX_RESOURCES];


void calculateNeed(int need[MAX_PROCESSES][MAX_RESOURCES],
                   int maximum[MAX_PROCESSES][MAX_RESOURCES],
                   int allocation[MAX_PROCESSES][MAX_RESOURCES],
                   int n, int r) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < r; j++) {
            need[i][j] = maximum[i][j] - allocation[i][j];
        }
    }
}


bool isSafe(int n, int r) {
    int work[MAX_RESOURCES];
    bool finish[MAX_PROCESSES] = {0};
    int safeSequence[MAX_PROCESSES];
    int count = 0;
```

```
    for (int i = 0; i < r; i++) {
        work[i] = available[i];
    }

    while (count < n) {
        bool found = false;
        for (int p = 0; p < n; p++) {
            if (!finish[p]) {
                int j;
                for (j = 0; j < r; j++) {
                    if (need[p][j] > work[j]) {
                        break;
                    }
                }
                if (j == r) {
                    for (int k = 0; k < r; k++) {
                        work[k] += allocation[p][k];
                    }
                    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 < n; i++) {
        printf("%d ", safeSequence[i]);
    }
    printf("\n");
    return true;
}


bool requestResources(int process, int request[], int n, int r) {
    for (int i = 0; i < r; i++) {
        if (request[i] > need[process][i]) {
            printf("Error: process has exceeded its maximum claim.\n");
            return false;
        }
    }

    for (int i = 0; i < r; i++) {
        if (request[i] > available[i]) {
            printf("Resources are not available. Process must wait.\n");
```

```
            return false;
        }
    }

    for (int i = 0; i < r; i++) {
        available[i] -= request[i];
        allocation[process][i] += request[i];
        need[process][i] -= request[i];
    }

    if (isSafe(n, r)) {
        printf("Request can be granted.\n");
        return true;
    } else {
        printf("Request cannot be granted to avoid unsafe state.\n");

        for (int i = 0; i < r; i++) {
            available[i] += request[i];
            allocation[process][i] -= request[i];
            need[process][i] += request[i];
        }
        return false;
    }
}

int main() {
    int n = 5;
    int r = 3;

    printf("Enter the available instances for each resource:\n");
    for (int i = 0; i < r; i++) {
        scanf("%d", &available[i]);
    }

    printf("Enter the maximum demand of each process for each resource:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < r; j++) {
            scanf("%d", &maximum[i][j]);
        }
    }

    printf("Enter the allocated resources for each process:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < r; j++) {
            scanf("%d", &allocation[i][j]);
        }
    }

    calculateNeed(need, maximum, allocation, n, r);
```

```
    isSafe(n, r);

    int process;
    int request[MAX_RESOURCES];
    printf("Enter the process number making the request: ");
    scanf("%d", &process);

    printf("Enter the request for resources: ");
    for (int i = 0; i < r; i++) {
        scanf("%d", &request[i]);
    }

    requestResources(process, request, n, r);

    return 0;
}
```

Output

```
rudra@DESKTOP-40986GA:~$ ./bankers_algorithm
Enter the available instances for each resource:
3 3 2
Enter the maximum demand of each process for each resource:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the allocated resources for each process:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
System is in a safe state.
Safe sequence is: 1 3 4 0 2
Enter the process number making the request: 1
Enter the request for resources: 1 0 2
System is in a safe state.
Safe sequence is: 1 3 4 0 2
Request can be granted.
rudra@DESKTOP-40986GA:~$
```