192311035

M.RAVI SAI VINAY

EXPERIMENT-17

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

AIM:-

To illustrate the concept of deadlock avoidance by simulating the Banker's Algorithm using C.

ALGORITHM:-

1. **Start the Program**.
2. Input the number of processes and resources.
3. Input the allocation, maximum, and available resources matrices.
4. Compute the need matrix as Need[i][j] = Max[i][j] - Allocation[i][j].
5. Implement the safety algorithm:
    - Check if a process can be executed safely by verifying the need ≤ available resources.
    - If it is safe, simulate allocation and update the available resources.
6. Check for the safe sequence of process execution. If all processes can execute safely, display the sequence.
7. Exit the program.

CODE:-

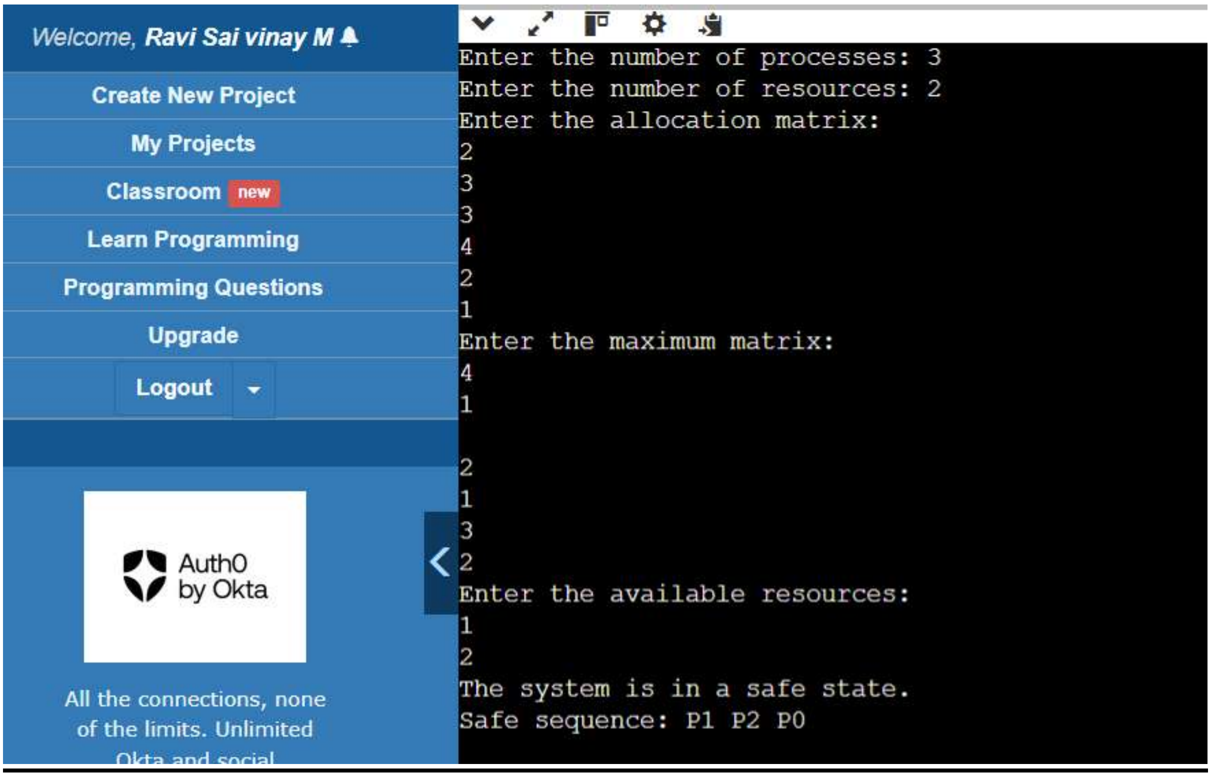
```
#include <stdio.h>

#include <stdbool.h>


int main() {

    int n, m, i, j, k;
```

```
printf("Enter the number of processes: ");

scanf("%d", &n);

printf("Enter the number of resources: ");

scanf("%d", &m);



int allocation[n][m], max[n][m], available[m], need[n][m];

int finish[n], safeSequence[n], index = 0;



printf("Enter the allocation matrix:\n");

for (i = 0; i < n; i++)

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

        scanf("%d", &allocation[i][j]);



printf("Enter the maximum matrix:\n");

for (i = 0; i < n; i++)

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

        scanf("%d", &max[i][j]);



printf("Enter the available resources:\n");

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

    scanf("%d", &available[j]);



for (i = 0; i < n; i++)

    finish[i] = 0;
```

```
for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
        need[i][j] = max[i][j] - allocation[i][j];


for (k = 0; k < n; k++) {
    for (i = 0; i < n; i++) {
        if (finish[i] == 0) {
            bool flag = true;
            for (j = 0; j < m; j++) {
                if (need[i][j] > available[j]) {
                    flag = false;
                    break;
                }
            }
            if (flag) {
                for (j = 0; j < m; j++)
                    available[j] += allocation[i][j];
                safeSequence[index++] = i;
                finish[i] = 1;
            }
        }
    }
}
```

```
    bool safe = true;
    for (i = 0; i < n; i++) {
        if (finish[i] == 0) {
            safe = false;
            break;
        }
    }

    if (safe) {
        printf("The system is in a safe state.\nSafe sequence: ");
        for (i = 0; i < n; i++)
            printf("P%d ", safeSequence[i]);
    } else {
        printf("The system is not in a safe state.\n");
    }

    return 0;
}
```

OUTPUT:-



```
Enter the number of processes: 3
Enter the number of resources: 2
Enter the allocation matrix:
2
3
3
4
2
1
Enter the maximum matrix:
4
1

2
1
3
2
Enter the available resources:
1
2
The system is in a safe state.
Safe sequence: P1 P2 P0
```

RESULT:-

The Banker’s Algorithm was successfully implemented to avoid deadlock. The program identified whether the system is in a safe state and provided the safe sequence of process execution if possible.