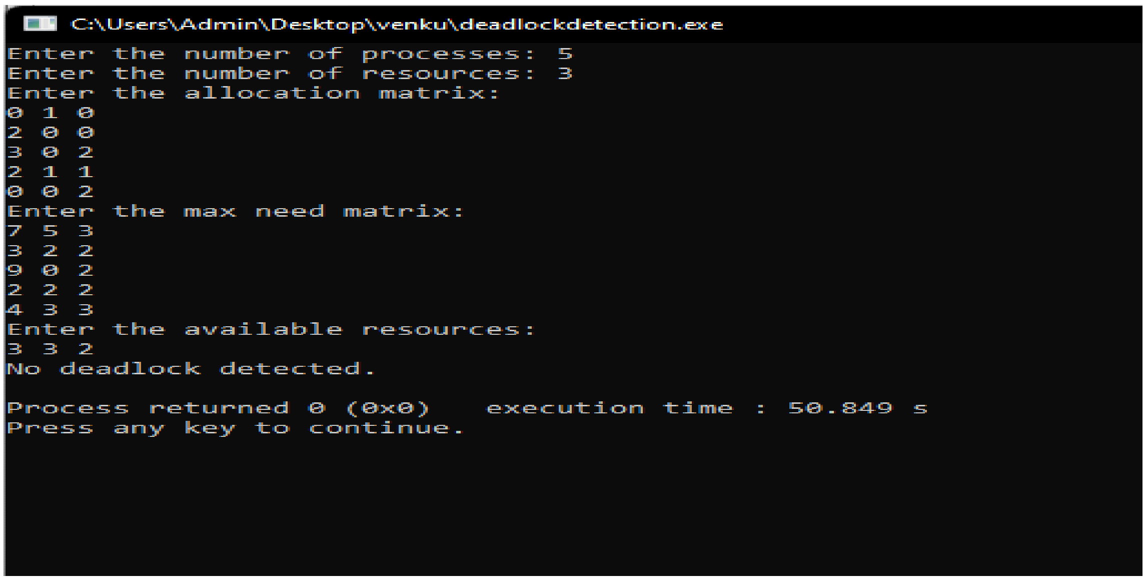
LAB 7

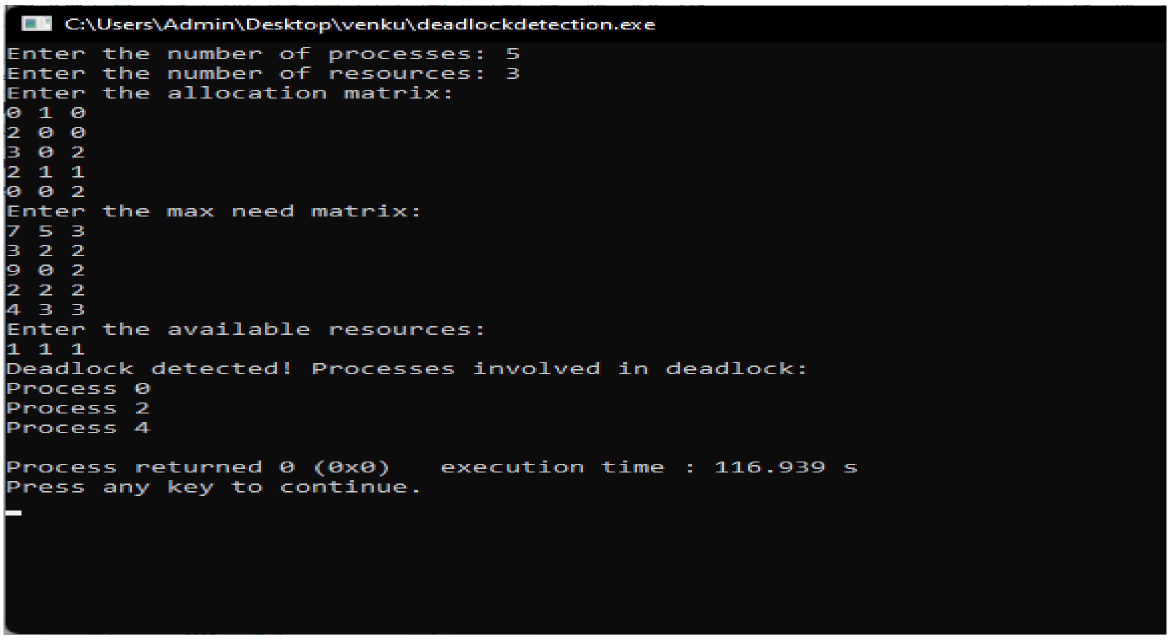
Q1. Write a C program to simulate deadlock detection.

```
#include <stdio.h>


#define MAX_PROCESSES 10

#define MAX_RESOURCES 10


int processes, resources;

int allocation[MAX_PROCESSES][MAX_RESOURCES];

int max_need[MAX_PROCESSES][MAX_RESOURCES];

int available[MAX_RESOURCES];

int marked[MAX_PROCESSES];

int finished[MAX_PROCESSES];


void initialize() {

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

    scanf("%d", &processes);

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

    scanf("%d", &resources);


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

    for (int i = 0; i < processes; i++) {

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

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

        }

    }


    printf("Enter the max need matrix:\n");

    for (int i = 0; i < processes; i++) {

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

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

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

void detectDeadlock() {
    for (int i = 0; i < processes; i++) {
        marked[i] = 0;
        finished[i] = 0;
    }

    int marked_count = 0;
    while (marked_count < processes) {
        int found = 0;
        for (int i = 0; i < processes; i++) {
            if (!finished[i] && !marked[i]) {
                int can_allocate = 1;
                for (int j = 0; j < resources; j++) {
                    if (max_need[i][j] - allocation[i][j] > available[j]) {
                        can_allocate = 0;
                        break;
                    }
                }
                if (can_allocate) {
                    marked[i] = 1;
                    marked_count++;
                    found = 1;
                    for (int j = 0; j < resources; j++) {
                        available[j] += allocation[i][j];
                    }
                    break;
                }
            }
        }
        if (!found) {
            printf("Deadlock detected! Processes involved in deadlock:\n");
            for (int i = 0; i < processes; i++) {
                if (!finished[i] && !marked[i]) {
                    printf("Process %d\n", i);
                }
            }
            return;
        }
    }

    printf("No deadlock detected.\n");
}

int main() {
    initialize();
    detectDeadlock();
    return 0;
}
```

OUTPUT:



```
C:\Users\Admin\Desktop\venku\deadlockdetection.exe
Enter the number of processes: 5
Enter the number of resources: 3
Enter the allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the max need matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the available resources:
3 3 2
No deadlock detected.

Process returned 0 (0x0)   execution time : 50.849 s
Press any key to continue.
```



```
C:\Users\Admin\Desktop\venku\deadlockdetection.exe
Enter the number of processes: 5
Enter the number of resources: 3
Enter the allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the max need matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the available resources:
1 1 1
Deadlock detected! Processes involved in deadlock:
Process 0
Process 2
Process 4

Process returned 0 (0x0)   execution time : 116.939 s
Press any key to continue.
```