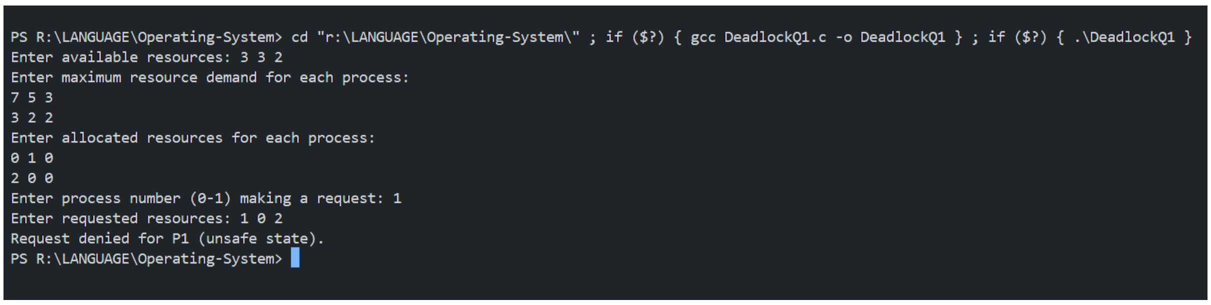
OPERATING SYSTEM EXPERIMENT

Q1 . CODE

```
#include <stdio.h>

#include <stdbool.h>


#define P 2

#define R 3


int available[R];

int maximum[P][R];

int allocation[P][R];

int need[P][R];


bool isSafeState()

{

    bool finish[P] = {false};

    int work[R];

    for (int i = 0; i < R; i++)

        work[i] = available[i];


    int count = 0;

    while (count < P)

    {

        bool found = false;

        for (int i = 0; i < P; i++)

        {
            if (!finish[i])
            {
                bool canAllocate = true;
                for (int j = 0; j < R; j++)
                {
                    if (need[i][j] > work[j])
                    {
                        canAllocate = false;
                        break;
                    }
                }
                if (canAllocate)
                {
                    for (int j = 0; j < R; j++)
                        work[j] += allocation[i][j];

                    finish[i] = true;
                    found = true;
                    count++;
                }
            }
        }
        if (!found)
            return false;
    }
    return true;
}
```

```
bool requestResources(int process, int request[R])
{
    for (int j = 0; j < R; j++)
    {
        if (request[j] > need[process][j] || request[j] > available[j])
            return false;
    }

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

    if (isSafeState())
    {
        printf("Request granted for P%d.\n", process);
        return true;
    }
    else
    {

        for (int j = 0; j < R; j++)
        {
            available[j] += request[j];
            allocation[process][j] -= request[j];
            need[process][j] += request[j];
        }
        printf("Request denied for P%d (unsafe state).\n", process);
        return false;
    }
}

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

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

    printf("Enter allocated resources for each process:\n");
    for (int i = 0; i < P; i++)
    {
        for (int j = 0; j < R; j++)
        {
            scanf("%d", &allocation[i][j]);
            need[i][j] = maximum[i][j] - allocation[i][j];
        }
    }
    int process;
    printf("Enter process number (0-%d) making a request: ", P - 1);
    scanf("%d", &process);

    int request[R];
    printf("Enter requested resources: ");
    for (int i = 0; i < R; i++)
        scanf("%d", &request[i]);

    requestResources(process, request);

    return 0;
}
```

OUTPUT:-



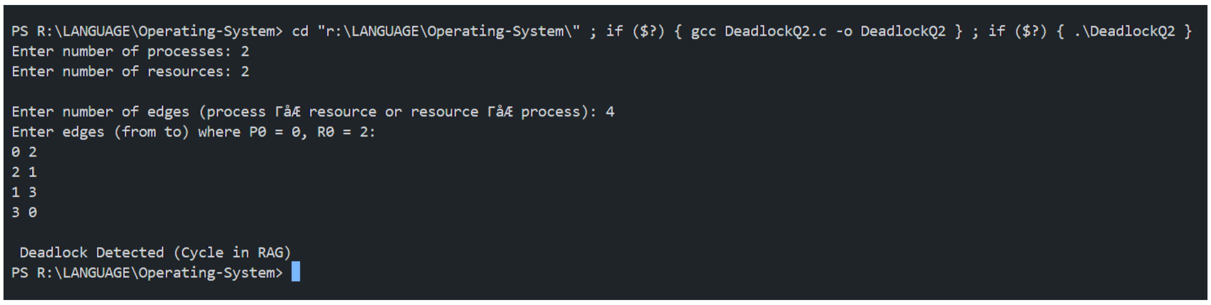
```
PS R:\LANGUAGE\Operating-System> cd "r:\LANGUAGE\Operating-System\" ; if ($?) { gcc DeadlockQ1.c -o DeadlockQ1 } ; if ($?) { .\DeadlockQ1 }
Enter available resources: 3 3 2
Enter maximum resource demand for each process:
7 5 3
3 2 2
Enter allocated resources for each process:
0 1 0
2 0 0
Enter process number (0-1) making a request: 1
Enter requested resources: 1 0 2
Request denied for P1 (unsafe state).
PS R:\LANGUAGE\Operating-System>
```

Q2. CODE

```
#include <stdio.h>

#include <stdbool.h>


#define MAX 10


int rag[MAX][MAX];

int visited[MAX], recStack[MAX];

int nodes;


bool detectCycle(int v) {

   visited[v] = 1;

   recStack[v] = 1;


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

      if (rag[v][i]) {

         if (!visited[i] && detectCycle(i))

            return true;

         else if (recStack[i])

            return true;

      }

   }


   recStack[v] = 0;

   return false;

}
```

```
bool isDeadlocked() {
    for (int i = 0; i < nodes; i++) {
        visited[i] = recStack[i] = 0;
    }

    for (int i = 0; i < nodes; i++) {
        if (!visited[i] && detectCycle(i))
            return true;
    }
    return false;
}

int main() {
    int processes, resources;

    printf("Enter number of processes: ");
    scanf("%d", &processes);
    printf("Enter number of resources: ");
    scanf("%d", &resources);

    nodes = processes + resources;

    printf("\nEnter number of edges (process → resource or resource → process): ");
    int edges;
    scanf("%d", &edges);

    printf("Enter edges (from to) where P0 = 0, R0 = %d:\n", processes);
    for (int i = 0; i < edges; i++) {
        int from, to;
        scanf("%d %d", &from, &to);
        rag[from][to] = 1;
    }

    if (isDeadlocked())
        printf("\n Deadlock Detected (Cycle in RAG)\n");
    else
        printf("\n No Deadlock (No Cycle in RAG)\n");

    return 0;
}
```

OUTPUT:-



```
PS R:\LANGUAGE\Operating-System> cd "r:\LANGUAGE\Operating-System\" ; if ($?) { gcc DeadlockQ2.c -o DeadlockQ2 } ; if ($?) { .\DeadlockQ2 }
Enter number of processes: 2
Enter number of resources: 2

Enter number of edges (process ΓåÆ resource or resource ΓåÆ process): 4
Enter edges (from to) where P0 = 0, R0 = 2:
0 2
2 1
1 3
3 0

 Deadlock Detected (Cycle in RAG)
PS R:\LANGUAGE\Operating-System>
```