OPERATING SYSTEMS LAB - PRACTICAL 8

Name - Sakshi Soni
Roll No - 13

AIM -

Write C programs to demonstrate the Banker's Algorithm and recovery processes.

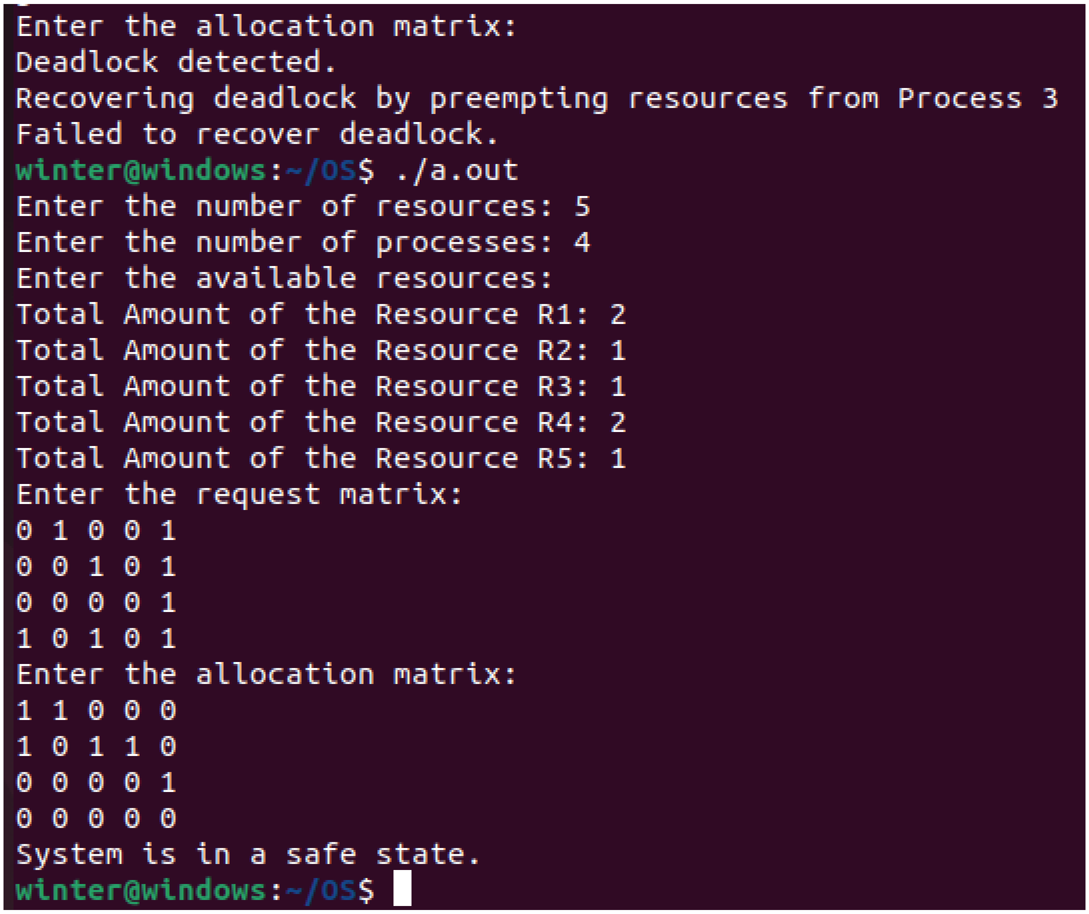
PROGRAM AND OUTPUT -

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

#define MAX_RESOURCES 10
#define MAX_PROCESSES 10

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

int num_resources;
int num_processes;

bool is_safe_state()
{
    int work[MAX_RESOURCES];
    bool finish[num_processes];

    // Initialize work and finish arrays
```

```
for (int i = 0; i < num_resources; i++)
    work[i] = available[i];
for (int i = 0; i < num_processes; i++)
    finish[i] = false;

// Find an unfinished process whose needs can be satisfied
int count = 0;
while (count < num_processes)
{
    bool found = false;
    for (int i = 0; i < num_processes; i++)
    {
        if (!finish[i])
        {
            int j;
            for (j = 0; j < num_resources; j++)
            {
                if (need[i][j] > work[j])
                    break;
            }
            if (j == num_resources)
            {
                // Process i can be finished
                for (int k = 0; k < num_resources; k++)
                    work[k] += allocation[i][k];
                finish[i] = true;
                found = true;
                count++;
            }
        }
    }
    if (!found)
        break;
}
```

```
    // Check if all processes are finished
    for (int i = 0; i < num_processes; i++)
    {
        if (!finish[i])
            return false;
    }

    return true;
}

bool request_resources(int process_id, int request[])
{
    // Check if the requested resources exceed the process's maximum needs
    for (int i = 0; i < num_resources; i++)
    {
        if (request[i] > need[process_id][i])
            return false;
    }

    // Check if the requested resources are available
    for (int i = 0; i < num_resources; i++)
    {
        if (request[i] > available[i])
            return false;
    }

    // Simulate allocation of resources
    for (int i = 0; i < num_resources; i++)
    {
        available[i] -= request[i];
        allocation[process_id][i] += request[i];
        need[process_id][i] -= request[i];
    }
    // Check if the system is still in a safe state
    if (is_safe_state())
        return true;
    else
    {
        // Undo the allocation of resources
        for (int i = 0; i < num_resources; i++)
        {
            available[i] += request[i];
            allocation[process_id][i] -= request[i];
            need[process_id][i] += request[i];
        }
        return false;
    }
}

void release_resources(int process_id, int release[])
{
    // Release the allocated resources
    for (int i = 0; i < num_resources; i++)
    {
        available[i] += release[i];
        allocation[process_id][i] -= release[i];
        need[process_id][i] += release[i];
    }
}

void recover_deadlock()
{
    // Find a process that is not finished and has unmet needs
    int process_id = -1;
    for (int i = 0; i < num_processes; i++)
    {
        if (!finished[i])
        {
            bool unmet_needs = false;
            for (int j = 0; j < num_resources; j++)
            {
                if (need[i][j] > available[j])
                {
                    unmet_needs = true;
                    break;
                }
            }
            if (unmet_needs)
            {
                process_id = i;
                break;
            }
        }
    }

    // If a process is found, preempt its allocated resources
    if (process_id != -1)
    {
        printf("Recovering deadlock by preempting resources from Process
%d\n", process_id);

        // Release the allocated resources of the process
        int release[MAX_RESOURCES];
        for (int i = 0; i < num_resources; i++)
        {
            release[i] = allocation[process_id][i];
            allocation[process_id][i] = 0;
            need[process_id][i] += release[i];
            available[i] += release[i];
        }

        // Check if the system is now in a safe state
        if (is_safe_state())
```

```
            printf("Deadlock recovered. System is in a safe state.\n");
        else
            printf("Failed to recover deadlock.\n");
    }
    else
    {
        printf("No process found with unmet needs. Deadlock cannot be
recovered.\n");
    }
}

int main()
{
    // Input the number of resources and processes
    printf("Enter the number of resources: ");
    scanf("%d", &num_resources);
    printf("Enter the number of processes: ");
    scanf("%d", &num_processes);

    // Input the available resources
    printf("Enter the available resources:\n");
    for (int i = 0; i < num_resources; i++)
    {
        printf("Total Amount of the Resource R%d: ", i + 1);
        scanf("%d", &available[i]);
    }

    // Input the request matrix
    printf("Enter the request matrix:\n");
    for (int i = 0; i < num_processes; i++)
    {
        for (int j = 0; j < num_resources; j++)
        {
            scanf("%d", &maximum[i][j]);
            need[i][j] = maximum[i][j];
        }
    }

    // Input the allocation matrix
    printf("Enter the allocation matrix:\n");
    for (int i = 0; i < num_processes; i++)
    {
        for (int j = 0; j < num_resources; j++)
        {
            scanf("%d", &allocation[i][j]);
            need[i][j] -= allocation[i][j];
        }
    }

    // Check if the system is in a safe state
    if (is_safe_state())
    {
        printf("System is in a safe state.\n");
    }
    else
    {
        printf("Deadlock detected.\n");
        recover_deadlock();
    }

    return 0;
}
```



```
Enter the allocation matrix:
Deadlock detected.
Recovering deadlock by preempting resources from Process 3
Failed to recover deadlock.
winter@windows:~/OS$ ./a.out
Enter the number of resources: 5
Enter the number of processes: 4
Enter the available resources:
Total Amount of the Resource R1: 2
Total Amount of the Resource R2: 1
Total Amount of the Resource R3: 1
Total Amount of the Resource R4: 2
Total Amount of the Resource R5: 1
Enter the request matrix:
0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
1 0 1 0 1
Enter the allocation matrix:
1 1 0 0 0
1 0 1 1 0
0 0 0 0 1
0 0 0 0 0
System is in a safe state.
winter@windows:~/OS$
```

RESULT -

C program to demonstrate the Banker’s Algorithm and recovery processes has been implemented.