**5**

**Lab 7: Banker’s Algorithm**

**Objectives:**

The primary objective of this C program is to implement the Banker's Algorithm, a resource allocation and deadlock avoidance algorithm used in operating systems. It aims to determine whether a given set of processes and resources is in a safe state, meaning that all processes can complete their execution without leading to a deadlock. The program accepts input for the number of processes, the number of resources, available resources, maximum demand of resources for each process, and the resources currently allocated to each process. It then calculates the need for each process and determines whether a safe sequence exists, where all processes can safely run to completion. If such a sequence exists, the program outputs it; otherwise, it indicates that the system is in an unsafe state.

1. **Write a program to implement a Banker’s algorithm.**

**Code:**

#include <stdio.h>

int main() {

int num\_processes, num\_resources;

printf("Enter number of processes: ");

scanf("%d", &num\_processes);

printf("Enter number of resources: ");

scanf("%d", &num\_resources);

int available[num\_resources];

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

for (int i = 0; i < num\_resources; i++) {

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

}

int max[num\_processes][num\_resources];

int allocation[num\_processes][num\_resources];

int need[num\_processes][num\_resources];

printf("Enter the maximum resources for each process:\n");

for (int i = 0; i < num\_processes; i++) {

printf("Process %d:\n", i);

for (int j = 0; j < num\_resources; j++) {

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

}

}

printf("Enter the allocated resources for each process:\n");

for (int i = 0; i < num\_processes; i++) {

printf("Process %d:\n", i);

for (int j = 0; j < num\_resources; j++) {

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

need[i][j] = max[i][j] - allocation[i][j];

}

}

int work[num\_resources];

for (int i = 0; i < num\_resources; i++) {

work[i] = available[i];

}

int finish[num\_processes];

for (int i = 0; i < num\_processes; i++) {

finish[i] = 0;

}

int safe\_sequence[num\_processes];

int num\_safe = 0;

while (num\_safe < num\_processes) {

int found = 0;

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) {

found = 1;

for (int k = 0; k < num\_resources; k++) {

work[k] += allocation[i][k];

}

safe\_sequence[num\_safe++] = i;

finish[i] = 1;

}

}

}

if (!found) {

printf("Unsafe state!\n");

return 1;

}

}

printf("Safe sequence:");

for (int i = 0; i < num\_processes; i++) {

printf(" %d", safe\_sequence[i]);

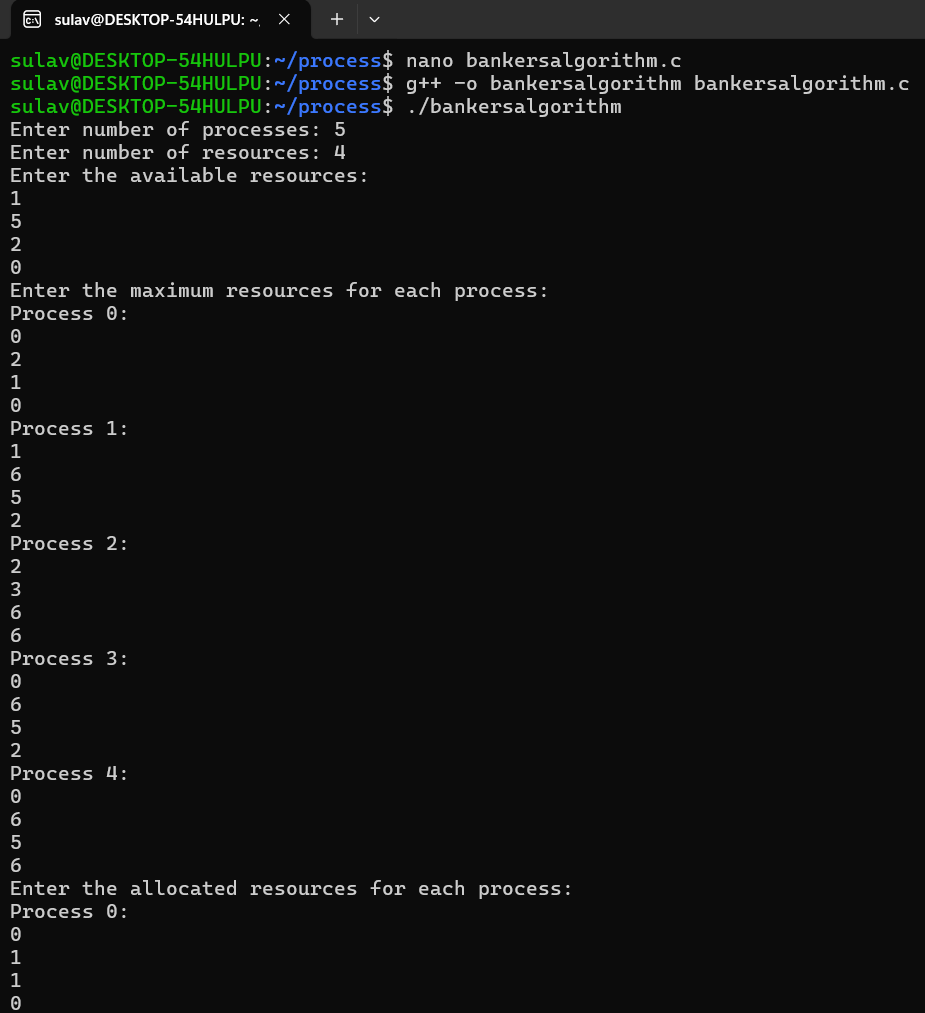
}

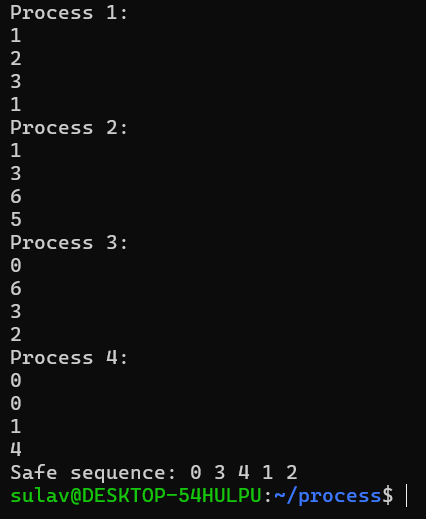
printf("\n");

return 0;

}

**Output:**



****

**Conclusion:**

This implementation of the Banker's Algorithm successfully identifies whether the system is in a safe or unsafe state based on the current resource allocation. The program efficiently calculates the necessary resources for each process and checks if the system can fulfill these needs without causing a deadlock. By ensuring that a safe sequence exists, it helps to prevent potential deadlocks, thus maintaining system stability. In cases where no safe sequence is found, the program accurately identifies the unsafe state, allowing for corrective actions to be taken. Overall, this program serves as a critical tool in managing resource allocation and ensuring system safety in concurrent processing environments.