Operating System (CS301)

Practical Exam

U19CS012

Q1.) A system has four processes and five resources that can be allocated. The current allocation and maximum needs are as follows:

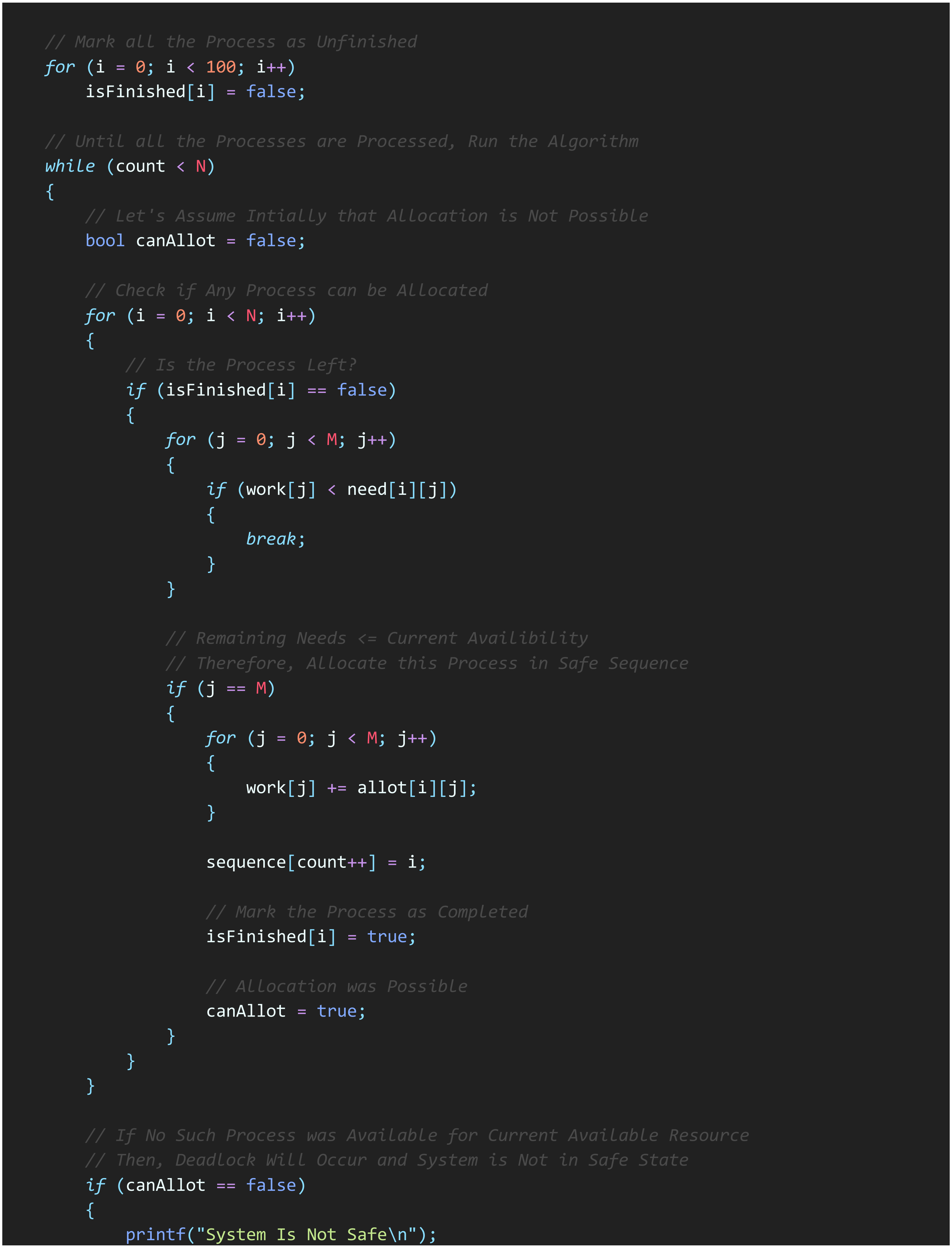
Process Id	Allocated	Maximum	Available
A	1 0 2 1 1	1 1 2 1 3	0 0 2 1 2
B	2 0 1 1 0	2 2 2 1 0	
C	1 1 0 1 0	2 1 3 1 0	
D	1 1 1 1 0	1 1 2 2 1	

Bankers Algorithm

- ✓ Banker's Algorithm is a **deadlock avoidance** algorithm.
- ✓ It is also used for **deadlock detection**.
- ✓ This algorithm tells that if any system can go into a deadlock or not by analyzing the currently allocated resources and the resources required by it in the future.

Code

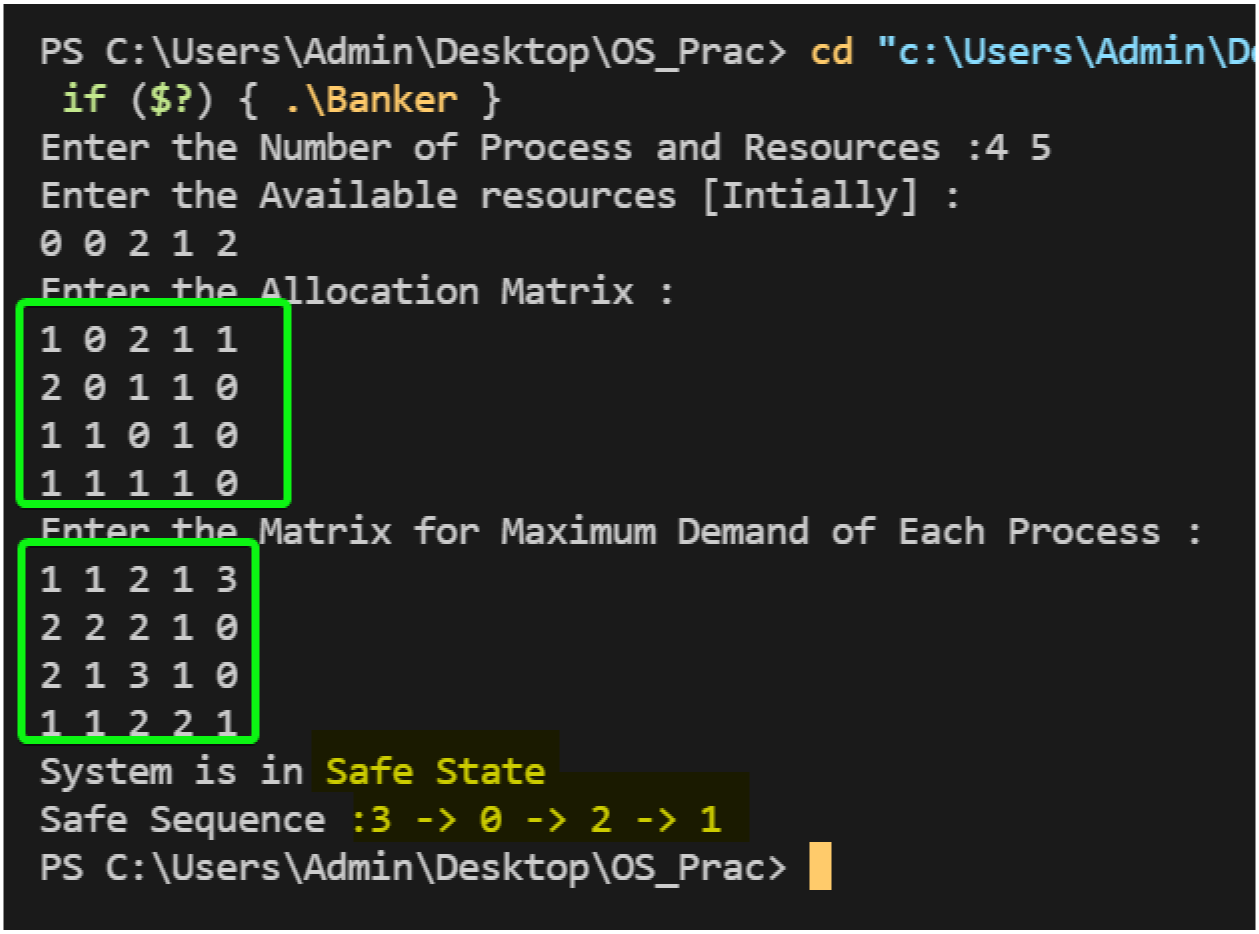
```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

int need[100][100], allot[100][100], max[100][100], available[100];
bool isFinished[100];
int sequence[100];

void isSafe(int N, int M)
{
    int i, j, work[100], count = 0;

    // Intialize the Available Resoures
    for (i = 0; i < M; i++)
        work[i] = available[i];
```



```
// Mark all the Process as Unfinished
for (i = 0; i < 100; i++)
    isFinished[i] = false;

// Until all the Processes are Processed, Run the Algorithm
while (count < N)
{
    // Let's Assume Intially that Allocation is Not Possible
    bool canAllot = false;

    // Check if Any Process can be Allocated
    for (i = 0; i < N; i++)
    {
        // Is the Process Left?
        if (isFinished[i] == false)
        {
            for (j = 0; j < M; j++)
            {
                if (work[j] < need[i][j])
                {
                    break;
                }
            }

            // Remaining Needs <= Current Availibility
            // Therefore, Allocate this Process in Safe Sequence
            if (j == M)
            {
                for (j = 0; j < M; j++)
                {
                    work[j] += allot[i][j];
                }

                sequence[count++] = i;

                // Mark the Process as Completed
                isFinished[i] = true;

                // Allocation was Possible
                canAllot = true;
            }
        }
    }

    // If No Such Process was Available for Current Available Resource
    // Then, Deadlock Will Occur and System is Not in Safe State
    if (canAllot == false)
    {
        printf("System Is Not Safe\n");
```

```c
            return;
        }
    }

    // If Control, Reaches Here, All Process have been able to Allocate and Safe Sequence Exist

    printf("System is in Safe State\n");

    printf("Safe Sequence :");
    for (i = 0; i < N; i++)
    {
        if (i == N - 1)
            printf("%d", sequence[i]);
        else
            printf("%d -> ", sequence[i]);
    }
    printf("\n");
}

int main()
{
    int i, j, N, M;
    printf("Enter the Number of Process and Resources :");
    scanf("%d %d", &N, &M);

    printf("Enter the Available resources [Intially] :\n");

    for (i = 0; i < M; i++)
        scanf("%d", &available[i]);

    printf("Enter the Allocation Matrix :\n");

    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
            scanf("%d", &allot[i][j]);

    printf("Enter the Matrix for Maximum Demand of Each Process :\n");

    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
            scanf("%d", &max[i][j]);

    // Calculation of need matrix [Remaining Need]
    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
            need[i][j] = max[i][j] - allot[i][j];

    isSafe(N, M);
}
```

Output



```
PS C:\Users\Admin\Desktop\OS_Prac> cd "c:\Users\Admin\De
 if ($?) { .\Banker }
Enter the Number of Process and Resources :4 5
Enter the Available resources [Intially] :
0 0 2 1 2
Enter the Allocation Matrix :
1 0 2 1 1
2 0 1 1 0
1 1 0 1 0
1 1 1 1 0
Enter the Matrix for Maximum Demand of Each Process :
1 1 2 1 3
2 2 2 1 0
2 1 3 1 0
1 1 2 2 1
System is in Safe State
Safe Sequence :3 -> 0 -> 2 -> 1
PS C:\Users\Admin\Desktop\OS_Prac>
```

Thus, we have Successfully Understood and **Implemented** Bankers Algorithm.

SUBMITTED BY:
U19CS012
BHAGYA VINOD RANA