Assignment No. 5

Name: **Sahil Jagadale** Division: TE-11 Batch: L-11 Roll No.: **33327**

Title: Bankers Algorithm

AIM: Implement C program for Deadlock Avoidance: Banker’s Algorithm

OBJECTIVE: Understand deadlock avoidance and implement it in C

THEORY :

Banker's algorithm is a deadlock avoidance algorithm. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not. Consider there are n account holders in a bank and the sum of the money in all of their accounts is S. Every time a loan has to be granted by the bank, it subtracts the loan amount from the total money the bank has. Then it checks if that difference is greater than S. It is done because, only then, the bank would have enough money even if all the n account holders draw all their money at once.

A) Safety Algorithm

A safety algorithm is an algorithm used to find whether or not a system is in its safe state. The algorithm is as follows:

1. Let Work and Finish be vectors of length m and n, respectively. Initially,

Work = Available

Finish[i] =false for i = 0, 1, ... , n - 1.

This means, initially, no process has finished and the number of available resources is represented by the Available array.

2. Find an index i such that both

Finish[i] ==false

Needi <= Work

If there is no such i present, then proceed to step 4.

It means, we need to find an unfinished process whose needs can be satisfied by the available resources. If no such process exists, just go to step 4.

3. Perform the following:

Work = Work + Allocationi

Finish[i] = true

Go to step 2.

When an unfinished process is found, then the resources are allocated and the process is marked finished. And then, the loop is repeated to check the same for all other processes.

4. If Finish[i] == true for all i, then the system is in a safe state.

That means if all processes are finished, then the system is in safe state.

This algorithm may require an order of mxn2 operations in order to determine whether a state is safe or not.

B. Resource Request Algorithm

Now the next algorithm is a resource-request algorithm, and it is mainly used to determine whether requests can be safely granted or not.

Let Requesti be the request vector for the process Pi. If Requesti[j]==k, then process Pi wants k instance of Resource type Rj.When a request for resources is made by the process Pi, the following are the actions that will be taken:

1. If Requesti <= Needi, then go to step 2; else raise an error condition, since the process has exceeded its maximum claim.

2.If Requesti <= Availablei then go to step 3; else Pi must have to wait as resources are not available.

3.Now we will assume that resources are assigned to process Pi and thus perform the following steps:

Available= Available-Requesti ;

Allocationi=Allocationi +Requesti;

Needi =Needi - Requesti;

If the resulting resource allocation state comes out to be safe, then the transaction is completed and, process Pi is allocated its resources. But in this case, if the new state is unsafe, then Pi waits for Requesti, and the old resource-allocation state is restored.

Code:

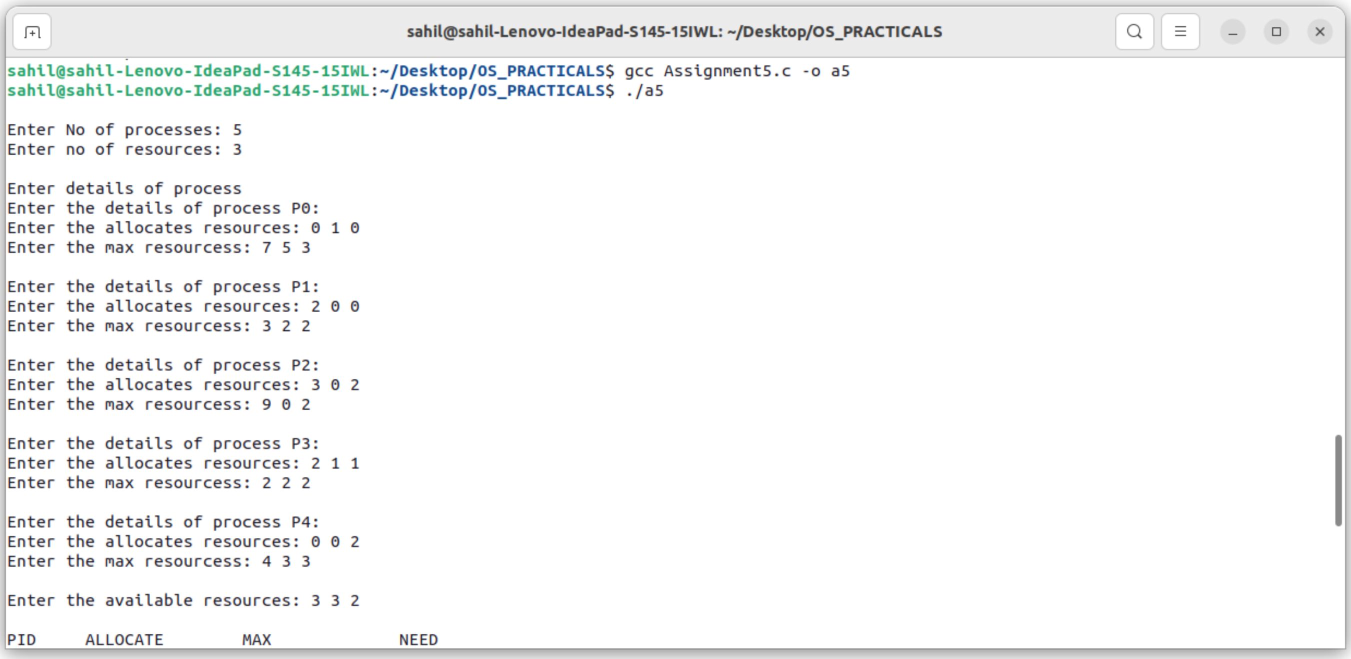
```
#include <stdio.h>

struct process
{
    int max[10], allocate[10], need[10];
} p[10];

int n, m;
void input(int[]);
void display(struct process p[]);
```

```c
int isSafestate(int[], int[],struct process p[]);
int safetyalgorithm(int[], int[],struct process p[]);

int main()
{
    int i;
    printf("\nEnter No of processes: ");
    scanf("%d", &n);
    printf("Enter no of resources: ");
    scanf("%d", &m);

    int available[m];
    int safesequence[n];

    printf("\nEnter details of process");
    input(available);
    display(p);
    printf("\n\n");

    int flag = 1;
    while (flag)
    {
    printf("Enter your choice: \n");
        printf("1) Find System Safety\t2) Get Safe Sequence\t3) Resource Request\t4) Exit\n");
        int ch; scanf("%d",&ch);

        switch (ch)
        {
            case 1:
                if (isSafestate(available, safesequence,p))
                {
                    printf("\nSYSTEM IS IN SAFE STATE...\n\n");
                }
                else
                    printf("\nSYSTEM IS IN UNSAFE STATE!!!\n\n");
                break;
            case 2:
                printf("\n");
                if (isSafestate(available, safesequence,p))
                {
                    printf("Safe Sequence is :: \t");
                    for (i = 0; i < n; i++)
                        printf("P%d -> ", safesequence[i]);
                }
                printf("\n\n");
                break;
            case 3:
                printf("Enter the Process no. for request :: ");
                int proc; scanf("%d",&proc);

                //copying the processes
                struct process pc[10];

                for(int i = 0; i<n; i++){
                    for(int j = 0; j<m; j++){
```

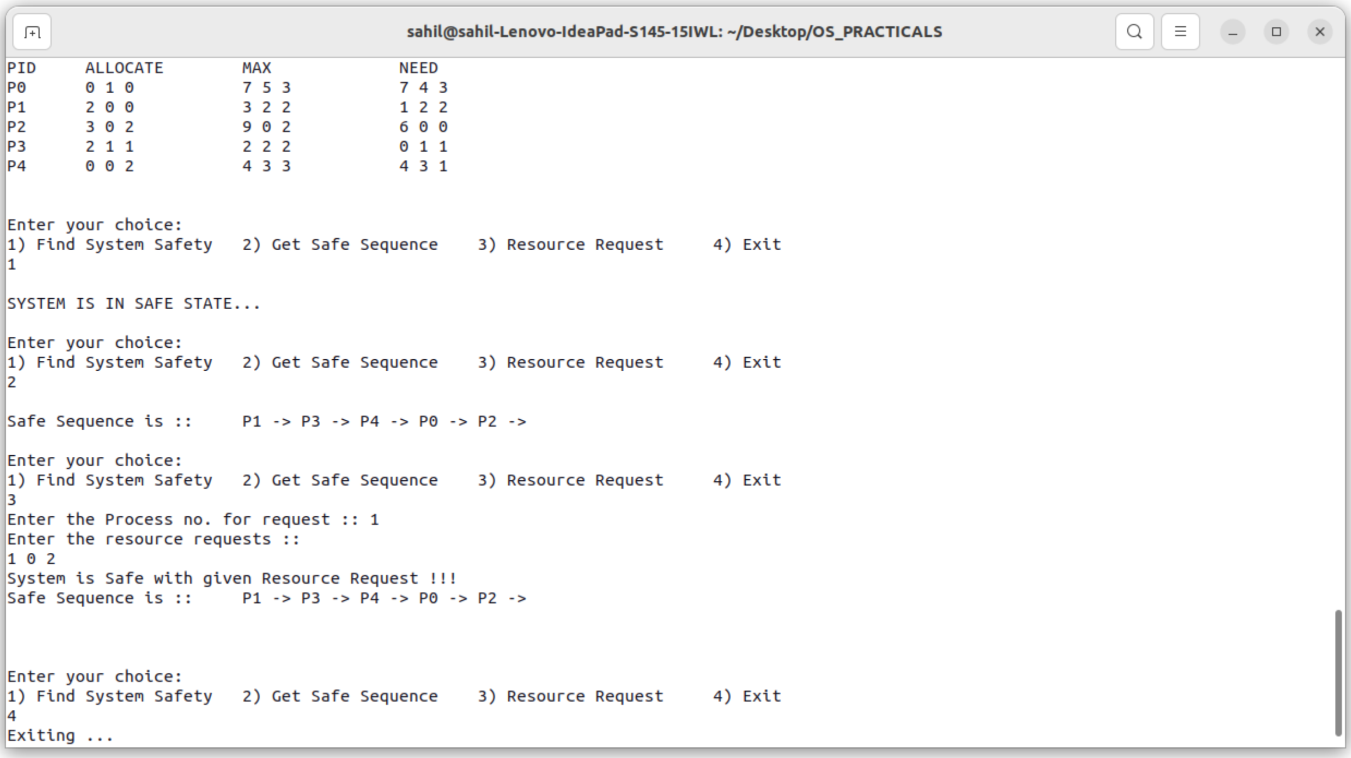
```
                pc[i].max[j] = p[i].max[j];
                pc[i].allocate[j] = p[i].allocate[j];
                pc[i].need[j] = p[i].need[j];
            }
        }

        printf("Enter the resource requests :: \n");

        for(int i = 0; i<m; i++){
            int tp; scanf("%d",&tp);
            pc[proc].allocate[i] += tp;
            pc[proc].need[i] = pc[proc].max[i] - pc[proc].allocate[i];
        }

        if (isSafestate(available, safesequence,pc))
        {
            printf("System is Safe with given Resource Request !!!");
            printf("\n");
            if (isSafestate(available, safesequence,p))
            {
                printf("Safe Sequence is :: \t");
                for (i = 0; i < n; i++)
                    printf("P%d -> ", safesequence[i]);
            }
            printf("\n\n");
        }
        else
            printf("System is Not Safe with given Resource Request !!!");


        printf("\n\n");
        break;
    case 4:
        printf("Exiting ...\n\n");
        flag = 0;
        break;
    default:
        printf("Invalid Input !!!\n\n");
        break;
  }
 }

 return 0;
}

void input(int available[m])
{

 int i, j;
 for (i = 0; i < n; i++)
 {

    printf("\nEnter the details of process P%d: ", i);
    printf("\nEnter the allocates resources: ");
    for (j = 0; j < m; j++)
```

```
        {
            scanf("%d", &p[i].allocate[j]);
        }
        printf("Enter the max resourcess: ");
        for (j = 0; j < m; j++)
        {
            scanf("%d", &p[i].max[j]);
            p[i].need[j] = p[i].max[j] - p[i].allocate[j];
        }
    }
    printf("\nEnter the available resources: ");
    for (j = 0; j < m; j++)
    {
        scanf("%d", &available[j]);
    }
}

void display(struct process p[])
{
    int i, j;
    printf("\nPID\tALLOCATE\tMAX\t\tNEED\n");
    for (i = 0; i < n; i++)
    {
        printf("P%d\t", i);
        for (j = 0; j < m; j++)
            printf("%d ", p[i].allocate[j]);
        printf("\t\t");
        for (j = 0; j < m; j++)
            printf("%d ", p[i].max[j]);
        printf("\t\t");
        for (j = 0; j < m; j++)
            printf("%d ", p[i].need[j]);
        printf("\n");
    }
}

int isSafestate(int available[m], int safesequence[n],struct process p[])
{
    if (safetyalgorithm(available, safesequence,p) == 1)
        return 1;
    return 0;
}

int safetyalgorithm(int available[m], int safesequence[n],struct process p[])
{
    int i, j;

    int work[m], finish[n];
    for (j = 0; j < m; j++)
        work[j] = available[j];

    for (i = 0; i < n; i++)
        finish[i] = 0;

    int proceed = 1, k = 0;
    while (proceed)
    {
        proceed = 0;
        for (i = 0; i < n; i++)
        {
            int flag = 1;
            if (finish[i] == 0)
            {
                for (j = 0; j < m; j++)
                {
                    if (p[i].need[j] <= work[j])
                    {
                        continue;
                    }
                    else
                    {
                        flag = 0;
                        break;
                    }
                }
                if (flag == 0)
                    continue;

                for (j = 0; j < m; j++)
                {
                    work[j] += p[i].allocate[j];
                }
                finish[i] = 1;
                safesequence[k++] = i;
                proceed = 1;
            }
        }
    }

    for (i = 0; i < n && finish[i] == 1; i++)
        continue;
    if (i == n)
        return 1;
    return 0;
}
```

Output:


sahil@sahil-Lenovo-IdeaPad-S145-15IWL: ~/Desktop/OS_PRACTICALS

```
sahil@sahil-Lenovo-IdeaPad-S145-15IWL:~/Desktop/OS_PRACTICALS$ gcc Assignment5.c -o a5
sahil@sahil-Lenovo-IdeaPad-S145-15IWL:~/Desktop/OS_PRACTICALS$ ./a5

Enter No of processes: 5
Enter no of resources: 3

Enter details of process
Enter the details of process P0:
Enter the allocates resources: 0 1 0
Enter the max resourcess: 7 5 3

Enter the details of process P1:
Enter the allocates resources: 2 0 0
Enter the max resourcess: 3 2 2

Enter the details of process P2:
Enter the allocates resources: 3 0 2
Enter the max resourcess: 9 0 2

Enter the details of process P3:
Enter the allocates resources: 2 1 1
Enter the max resourcess: 2 2 2

Enter the details of process P4:
Enter the allocates resources: 0 0 2
Enter the max resourcess: 4 3 3

Enter the available resources: 3 3 2

PID     ALLOCATE        MAX             NEED
```


sahil@sahil-Lenovo-IdeaPad-S145-15IWL: ~/Desktop/OS_PRACTICALS

```
PID     ALLOCATE        MAX             NEED
P0      0 1 0           7 5 3           7 4 3
P1      2 0 0           3 2 2           1 2 2
P2      3 0 2           9 0 2           6 0 0
P3      2 1 1           2 2 2           0 1 1
P4      0 0 2           4 3 3           4 3 1


Enter your choice:
1) Find System Safety   2) Get Safe Sequence    3) Resource Request     4) Exit
1

SYSTEM IS IN SAFE STATE...

Enter your choice:
1) Find System Safety   2) Get Safe Sequence    3) Resource Request     4) Exit
2

Safe Sequence is ::     P1 -> P3 -> P4 -> P0 -> P2 ->

Enter your choice:
1) Find System Safety   2) Get Safe Sequence    3) Resource Request     4) Exit
3
Enter the Process no. for request :: 1
Enter the resource requests ::
1 0 2
System is Safe with given Resource Request !!!
Safe Sequence is ::     P1 -> P3 -> P4 -> P0 -> P2 ->



Enter your choice:
1) Find System Safety   2) Get Safe Sequence    3) Resource Request     4) Exit
4
Exiting ...
```