Code

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

struct proc {
int* max;
int* alloc;
int* need;
int* request;
int finish;
};

int bankers(struct proc process[], int available[], int m, int n){
 int sequence[n];
 int i,j;
 //for marking finish as 0 and 1 (initial value being 0)
for(i=0;i<n;i++)
        process[i].finish = 0;

//calculating the need of instances of resources of each process
for(i=0;i<n;i++){
        for(j=0;j<m;j++){
                process[i].need[j] = process[i].max[j]-process[i].alloc[j];
        }

 }
 //dispaying
 printf("The table formed is: \n");
 printf("Proc |\t Alloc |\t Max \n");
 printf("\t");
 for(i=0;i<m;i++)
        printf("| R%d \t",i+1);
 for(i=0;i<m;i++)
        printf("| R%d \t",i+1);
 printf("\n");
 for(i=0;i<n;i++){
        printf("P%d \t", i+1);
        for(j=0;j<m;j++){
                printf(" | %d | \t %d ",process[i].alloc[j],process[i].max[j]);
        }
        printf("\n");
 }

 printf("The need Matrix is: \n");
 printf("Need \t");
 for(i=0;i<m;i++)
        printf("| R%d \t",i+1);
 printf("\n");
 for(i=0;i<n;i++){
        printf("P%d \t", i+1);
        for(j=0;j<m;j++){
```

```
                printf(" | %d ",process[i].need[j]);
        }
        printf("\n");
}
//safety algorithm
int flag=0,count=0,k=0;
for(i=0;;i=(i+1)%n){
        for(j=0;j<m;j++){
                if((process[i].finish==0)&&(process[i].need[j]<=available[j])){
                        flag=1;
                }
                else{
                        count++;
                        if(count==n){
                                printf("The processes create a deadlock \n");
                                return 0;
                        }
                        flag=0;
                        break;
                }
        }
        if(flag==1){
                process[i].finish=1;
                sequence[k]=i+1;
                k++;
                if(k==n){
                        printf("The processes are in safe state and the sequence formed is: \n");
                        for(i=0;i<n;i++)
                                printf("P%d \t",sequence[i]);
                        return 1;
                }
                for(j=0;j<m;j++){
                        available[j]+=process[i].alloc[j];
                        count=0;
                }
        }
}
}

void request(struct proc process[], int available[], int m, int n){
        int j,num;
        printf("Enter the process id: \n");
        scanf("%d",&num);
        num=num-1;
        printf("Enter the request of each resource: \n");
        for(j=0;j<m;j++){
                printf("R%d: ",j+1);
                scanf("%d",&process[num].request[j]);
        }
        for(j=0; j<m; j++){
                if(process[num].request[j] > process[num].need[j]){
                        printf("P%d's request cannot be granted\n", num+1);
```

```
                return;
            }
        }
        for(j=0; j<m; j++){
            if(process[num].request[j] > available[j]){
                printf("P%d's request cannot be granted\n", num+1);
                return;
            }
        }
        int max[m];
        int alloc[m];
        int need[m];
        int temp[m];
        for(j=0; j<m; j++){
            temp[j] = available[j];
            max[j] = process[num].max[j];
            alloc[j] = process[num].alloc[j];
            need[j] = process[num].need[j];
        }
        for(j=0; j<m; j++){
            available[j] -= process[num].request[j];
            process[num].alloc[j] += process[num].request[j];
            process[num].need[j] -= process[num].request[j];
        }
        if(bankers(process, available, m, n))
            printf("Hence,the request can be granted immediately.\n");
        else
            printf("Hence, the request cannot be granted immediately!\n");
        for(j=0; j<m; j++){
            available[j]=temp[j];
            process[num].max[j]=max[j];
            process[num].alloc[j]=alloc[j];
            process[num].need[j]=need[j];
        }
}
void main(){
 int m,n,i,j;
 struct proc *process;
 int *available;
 printf("Enter the number of processes: \n");
 scanf("%d",&n);
 process =(struct proc*) malloc (sizeof(struct proc)*n);
 printf("Enter the total number of resources: \n");
 scanf("%d",&m);
 for(i=0;i<n;i++){
   process[i].alloc = (int*)malloc(m*sizeof(int));
        process[i].max = (int*)malloc(m*sizeof(int));
        process[i].need = (int*)malloc(m*sizeof(int));
        process[i].request = (int*)malloc(m*sizeof(int));

 }
 available = (int*)malloc(m*sizeof(int));
```

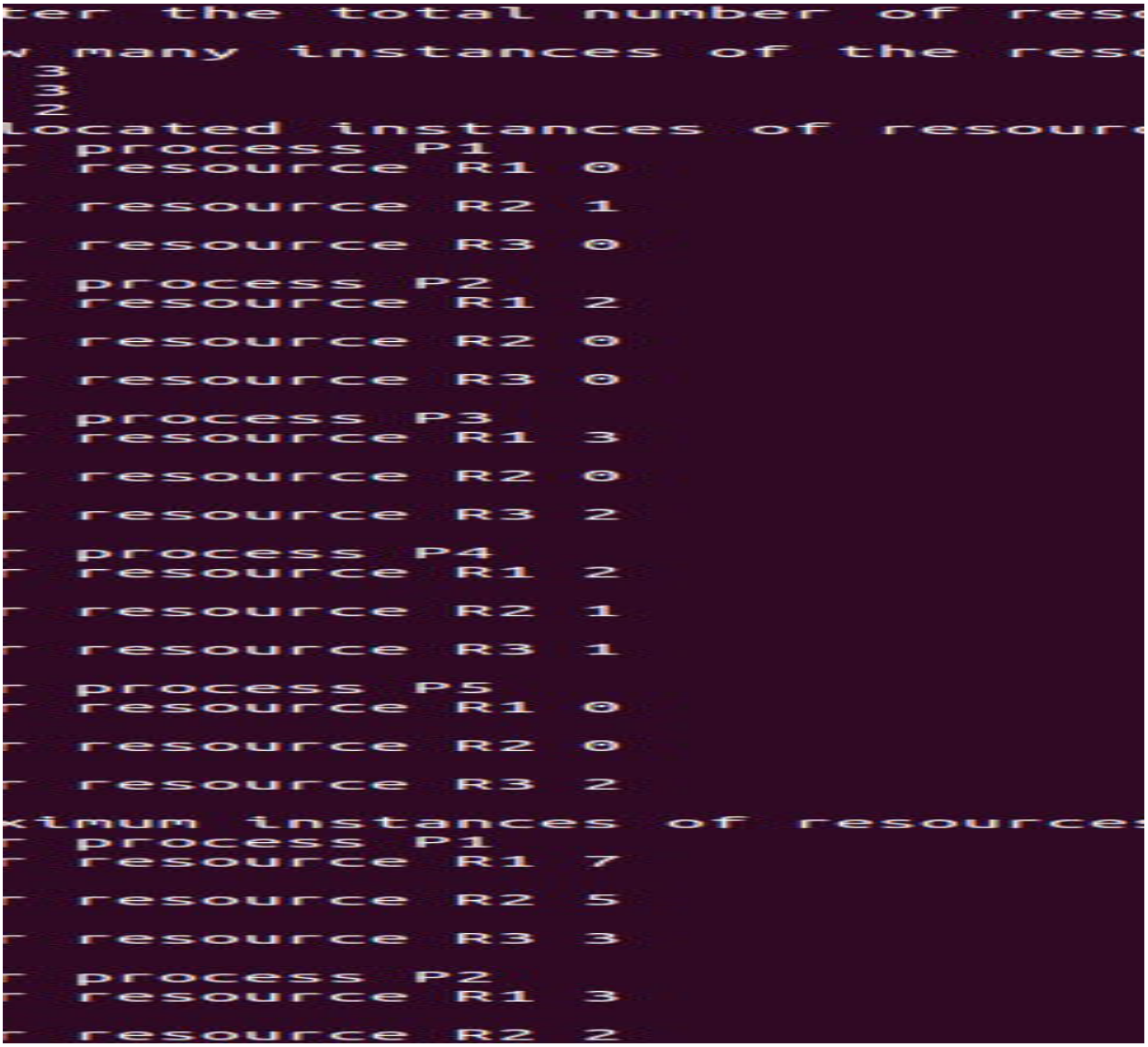
```
 printf("How many instances of the resources are available: \n");
 for(i=0;i<m;i++){
        printf("%d: ", i+1);
        scanf("%d",&available[i]);
 }

 //asking the allocation of each process
 printf("Allocated instances of resources to each process: \n");
 for(i=0;i<n;i++){
        printf("For process P%d \n", i+1);
        for(j=0;j<m;j++){
                printf("For resource R%d ",j+1);
                scanf("%d",&process[i].alloc[j]);
                printf("\n");
        }
 }

 //asking the maximum of each process
 printf("Maximum instances of resources required for each process: \n");
 for(i=0;i<n;i++){
        printf("For process P%d \n", i+1);
        for(j=0;j<m;j++){
                printf("For resource R%d ",j+1);
                scanf("%d",&process[i].max[j]);
                printf("\n");
        }
 }

int choice;
while(1) {
        printf("\n1. Safety Algorithm\n2. Resource Request Algorithm\n3. Exit\nEnter your input:
");
        scanf("%d", &choice);
        switch(choice) {
                case 1:bankers(process, available, m, n);
                        break;
                case 2:request(process, available, m, n);
                        break;
                case 3: printf("\nExit.\n");
                        exit(0);
                default:printf("\nInvalid option!\n");
                        break;
        }
}
}
```

Output


ter the total number of resc
w many instances of the resc
3
3
2
located instances of resourc
process P1
resource R1 0
resource R2 1
resource R3 0
process P2
resource R1 2
resource R2 0
resource R3 0
process P3
resource R1 3
resource R2 0
resource R3 2
process P4
resource R1 2
resource R2 1
resource R3 1
process P5
resource R1 0
resource R2 0
resource R3 2
ximum instances of resources
process P1
resource R1 7
resource R2 5
resource R3 3
process P2
resource R1 3
resource R2 2

```
The table formed is:
Proc |   Alloc |          Max
        | R1    | R2    | R3    | R1    | R2    | R3
P1       | 0 |   7   | 2 |         5   | 1 |         3
P2       | 2 |   3   | 0 |         2   | 0 |         2
P3       | 3 |   9   | 0 |         0   | 2 |         2
P4       | 2 |   2   | 1 |         2   | 1 |         2
P5       | 0 |   4   | 0 |         3   | 2 |         3
The need Matrix is:
Need    | R1    | R2    | R3
P1       | 7   | 3   | 2
P2       | 1   | 2   | 2
P3       | 6   | 0   | 0
P4       | 0   | 1   | 1
P5       | 4   | 3   | 1
The processes are in safe state and the sequence formed is:
P1      P2      P3      P4      P5      Hence,the request can be gra

1. Safety Algorithm
2. Resource Request Algorithm
3. Exit
Enter your input: 3

Exit.
```