Maharashtra State Board of Technical Education

Government polytechnic Solapur

DIPLOMA IN INFORMATION TECHNOLOGY

(IF)2023-2024

Academic Year 2023-2024

A

Micro Project on

Implement Banker's Algorithm for Deadlock Handling using C

Group Members

Roll No	Name of the Group Members	Enrolment No
15	Dharashivkar Aditya Mahesh	2100150261
23	Sawalgi Shriyash Balasaheb	2100150275
36	Makude Kaustubh Ishwar	2100150298
41	Maske Abhishek Sunil	2100150303

Under the Guidance of:

Prof. Nikunj Nomulwar

(Third Year)

Government polytechnic Solapur

Certificate

Certified that this Microproject Report

Roll No	Name of the Group Members	Enrolment No
15	Dharashivkar Aditya Mahesh	2100150261
23	Sawalgi Shriyash Balasaheb	2100150275
36	Makude Kaustubh Ishwar	2100150298
41	Maske Abhishek Sunil	2100150303

Implement Banker's Algorithm for Deadlock Handling using C

In this work.

The Students of Semester Fifth Operating System (OSY). Diploma in Information technology 2023-2024 Partial fulfilment for the Award of Diploma in information technology branch by MSBTE

Sign of Subject Teacher **Sign of principal**

Prof. Nikunj Nomulwar

PART-A MICROPROJECT REPORT

1.0 Title of Microproject: -

Implement Banker’s Algorithm for Deadlock Handling using C

2.0 Brief Introduction:

The Banker's Algorithm is a foundational technique for deadlock avoidance and handling in computer systems. Its historical development, key concepts, variations, practical applications, and challenges have been extensively studied and refined over the years. Understanding these aspects is essential for effectively applying the algorithm in modern computing environments and for further research in the field of deadlock handling.

3.0 Aim of the micro-project:

Implement Banker’s Algorithm for Deadlock Handling using

4.0 Intended course outcomes:

- Decide suitable software for project
- Choose correct languages for development
- Lean how to implement Banker’s algorithm

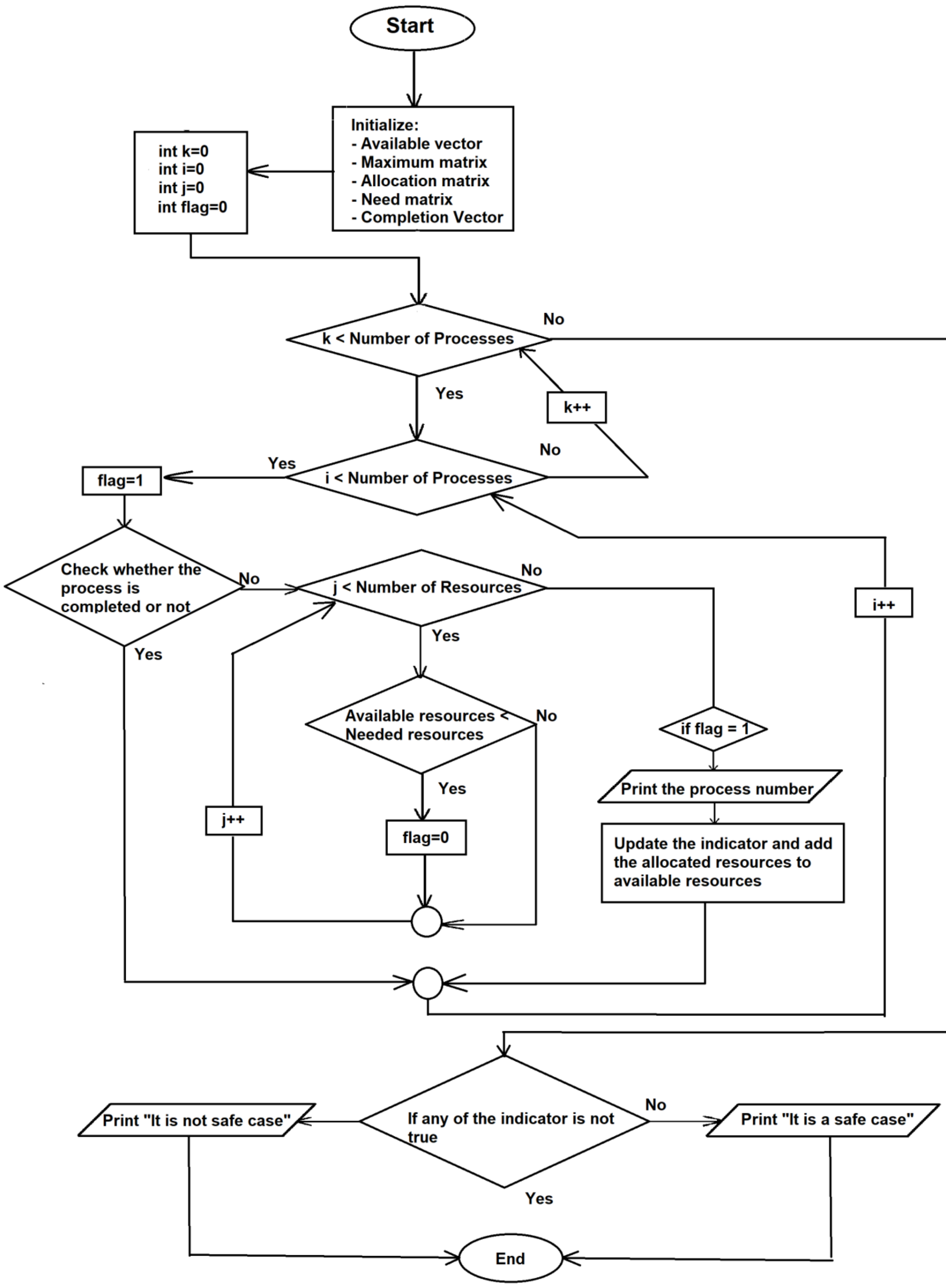
5.0 Literature review:

The Banker's Algorithm is a classic deadlock avoidance and handling technique used in operating systems and resource allocation systems. It ensures that processes request and release resources in a safe manner, preventing deadlock situations. In this literature review, we will explore the historical development, underlying principles, and practical applications of the Banker's Algorithm

The Banker's Algorithm was first introduced by Edger W. Dijkstra in his seminal paper titled "Cooperating Sequential Processes" in 1965. Dijkstra formulated the algorithm as a solution to the problem of preventing and managing deadlocks in computer systems. Since its inception, the Banker's Algorithm has undergone various refinements and adaptations.

In conclusion, the Banker's Algorithm is a foundational technique for deadlock avoidance and handling in computer systems. Its historical development, key concepts, variations, practical applications, and challenges have been extensively studied and refined over the years. Understanding these aspects is essential for effectively applying the algorithm in modern computing environments and for further research in the field of deadlock handling.

• Flow Chart for Banker's Algorithm



Steps for Banker’s Algorithm

1. Start
2. Initialise available, maximum, allocated, needed, and completed processes vectors and matrices.
3. Iterate the following process from 4 to 7 for number of times equals to number of processes.
4. Iterate the following process from 5 to 7 for number of times equals to number of processes.
5. Check whether the process is executed or not. If executed then go to next iteration.
6. Else check whether available resources are less than required number of resources.
7. If yes then no change in indicator if no then complete the process and deallocate the resources from the current process.
8. Check all the indicators are changed or not. If yes then print “It’s a safe case” else print “It’s not a safe case”.
9. Stop

6.0 Proposed Methodology:

1) Discussion about given topic.
2) Selection of group leader and distribution of responsibility.
3) Collection of information using different resources.
4) Analysis of information as per format given.
5) Represent of information and required format.
6) Preparation of project report.
7) Complications of and submission of given assign task

7.0 Resources required:

Sr. No	Name Of resources	Quantity	Remarks
1	Books	Operating Systems: Internals and Design Principles	
2	PC/ laptop	hp computer Processor- Intel(R) Core (TM) i5-8365U CPU @ 1.60GHz 1.90 GHz Installed Memory- (RAM)16:00GB System type - 64-byte operating system.	

8.0 Action plan:

Sr. No	Details of activity	Number of students
1	Discussion	15 Dharashivkar Aditya Mahesh 23 Sawalgi Shriyash Balasaheb 36 Makude Kaustubh Ishwar 41 Maske Abhishek Sunil
2	Collection of Information	15 Dharashivkar Aditya Mahesh 23 Sawalgi Shriyash Balasaheb 36 Makude Kaustubh Ishwar 41 Maske Abhishek Sunil
3	Analysis of Information	15 Dharashivkar Aditya Mahesh 23 Sawalgi Shriyash Balasaheb 36 Makude Kaustubh Ishwar 41 Maske Abhishek Sunil
4	Coding	15 Dharashivkar Aditya Mahesh 23 Sawalgi Shriyash Balasaheb 36 Makude Kaustubh Ishwar 41 Maske Abhishek Sunil
5	Preparation of Report	15 Dharashivkar Aditya Mahesh 23 Sawalgi Shriyash Balasaheb 36 Makude Kaustubh Ishwar 41 Maske Abhishek Sunil

PART-B MICROPROJECT REPORT

1.0 Title of Microproject:

Implement Banker’s Algorithm for Deadlock Handling using C

2.0 Aim of the micro-project:

To Implement Banker’s Algorithm for Deadlock Handling using C

3.0 Course Outcomes:

1. Install operating system and configure it.
2. Use operating system tools to perform various functions.
3. Execute process commands for performing process management operations.
4. Apply scheduling algorithms to calculate turnaround time and average waiting time.
5. Calculate efficiency of different memory management techniques.
6. Apply file management techniques.

4.0 Literature review:

Introduction:

The Banker's Algorithm is a classic deadlock avoidance and handling technique used in operating systems and resource allocation systems. It ensures that processes request and release resources in a safe manner, preventing deadlock situations. In this literature review, we will explore the historical development, underlying principles, and practical applications of the Banker's Algorithm.

Historical Development:

The Banker's Algorithm was first introduced by Edsger W. Dijkstra in his seminal paper titled "Cooperating Sequential Processes" in 1965. Dijkstra formulated the algorithm as a solution to the problem of preventing and managing deadlocks in computer systems. Since its inception, the Banker's Algorithm has undergone various refinements and adaptations.

Key Concepts:

- Resources and Processes: The Banker's Algorithm operates on the idea of resources and processes. Resources can be categorized into different types, and each process can request and release a certain number of resources of each type.
- Allocation and Maximum Matrices: These matrices are used to represent the current allocation of resources to processes and the maximum resources each process may request, respectively.
- Need Matrix: The need matrix represents the remaining resources that a process needs to complete its execution. It is calculated as the difference between the maximum and allocation matrices.
- Safety and Deadlock: The algorithm determines system safety by checking if there is a safe sequence of resource allocation. A safe sequence indicates that deadlock will not occur. If no safe sequence exists, the system is in a deadlock state.

Algorithm Variations and Extensions:

Over the years, researchers have proposed several variations and extensions to the Banker's Algorithm to enhance its applicability and efficiency. Some of these include:

- Resource Pre-emption: Introducing the ability to pre-empt resources from one process and allocate them to another to resolve resource contention more flexibly.
- Dynamic Resource Allocation: Adapting the Banker's Algorithm for scenarios where the resource availability dynamically changes during execution.
- Priority-Based Allocation: Incorporating priority levels for processes to determine resource allocation, ensuring that high-priority processes get preference.

Practical Applications:

The Banker's Algorithm has found applications in various domains, including:

- Operating Systems: It is a fundamental component of many operating systems, including Unix, Linux, and Windows, for managing resource allocation and preventing deadlocks.
- Database Management Systems: In database systems, the algorithm is used to manage concurrent access to resources, such as tables and records, ensuring data consistency and preventing deadlocks.

- Real-time Systems: It plays a crucial role in ensuring the timely execution of critical tasks in real-time systems, where resource allocation must be carefully managed to meet strict timing constraints.

Challenges and Limitations:

Despite its usefulness, the Banker's Algorithm has some limitations, such as the assumption of fixed resource requirements and the need for complete information about maximum resource demands. Researchers have explored alternative approaches to address these limitations.

Conclusion:

In conclusion, the Banker's Algorithm is a foundational technique for deadlock avoidance and handling in computer systems. Its historical development, key concepts, variations, practical applications, and challenges have been extensively studied and refined over the years. Understanding these aspects is essential for effectively applying the algorithm in modern computing environments and for further research in the field of deadlock handling.

5.0 Actual methodology:

1) Discussion about given topic.
2) Selection of group leader and distribution of responsibility.
3) Collection of information using different resources.
4) Analysis of information as per format given.
5) Represent of information and required format.
6) Preparation of project report.

Code for Banker’s Algorithm Using C

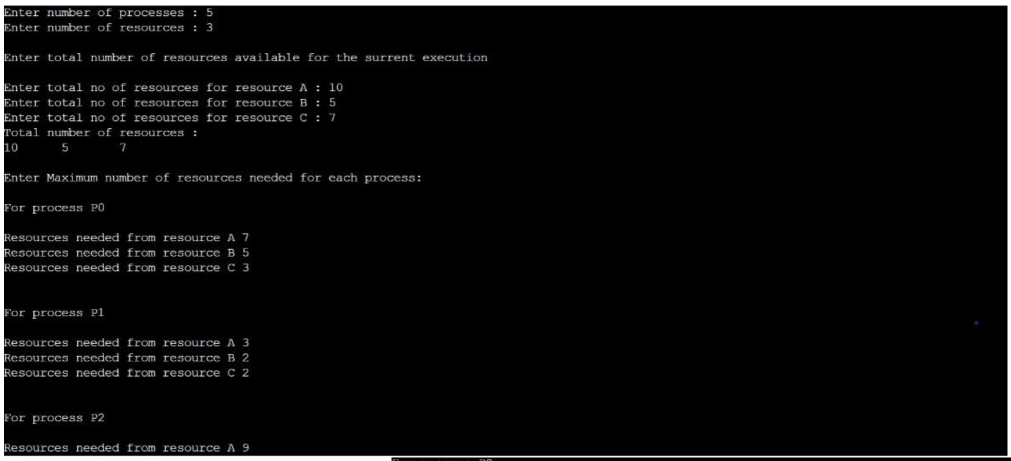
//C Program to Implement the Banker’s Algorithm
//Time Complexity = O(n^3)

```
#include<stdio.h>
int main()
{
    int n;
    int m;
    int i,j,k;
    int alloc[100][100];
    int max[100][100];
    int avail[100];
    int total[100];
    int totalAlloc[100];
    int flag;
    int ind[100];
    printf("Enter number of processes : ");
    scanf("%d",&n);
    printf("Enter number of resources : ");
    scanf("%d",&m);
    printf("\nEnter total number of resources available for the surrent execution\n\n");
    for(i=0;i<m;i++)
    {
        printf("Enter total no of resources for resource %c : ",(i+65));
        scanf("%d",(total+i));
    }
    printf("Total number of resources : \n");
    for(i=0;i<m;i++)
    {
        printf("%d\t",total[i]);
    }
```

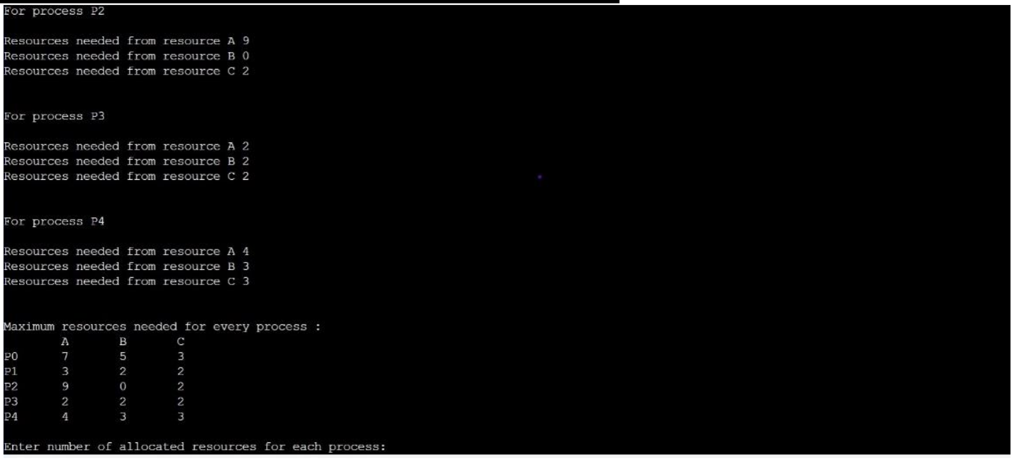
```
printf("\n\nEnter Maximum number of resources needed for each process: ");
for(i=0;i<n;i++)
{
    ind[i]=0;
    printf("\n\nFor process P%d\n\n",i);
    for(j=0;j<m;j++)
    {
        requirementError:
        printf("Resources needed from resource %c ",(j+65));
        scanf("%d",&max[i][j]);
        if(max[i][j]>total[j])
        {
            printf("\nRequired number of resources are greater than total number of resources\n");
            goto requirementError;
        }
    }
}
printf("\n\nMaximum resources needed for every process : \n");
printf("\tA\tB\tC");
for(i=0;i<n;i++)
{
    printf("\nP%d",i);
    for(j=0;j<m;j++)
    {
        printf("\t%d",max[i][j]);
    }
}
printf("\n\nEnter number of allocated resources for each process: ");
reEnterAllocated:
for(i=0;i<n;i++)
{
    printf("\n\nFor process P%d\n\n",i);
    allocationError:
    for(j=0;j<m;j++)
    {
            printf("Resources allocated from resource %c : ",(j+65));
            scanf("%d",&alloc[i][j]);
            if(alloc[i][j]>max[i][j])
            {
                printf("\nNumber of allocated resources are more than required resources\n");
                goto allocationError;
            }
        }
    }
    printf("\n\nResources allocated for every process : \n");
    printf("\tA\tB\tC");
    for(i=0;i<n;i++)
    {
        printf("\nP%d",i);
        for(j=0;j<m;j++)
        {
            printf("\t%d",alloc[i][j]);
        }
    }
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        {
            totalAlloc[j]= totalAlloc[j]+alloc[i][j];
        }
    }
    printf("\n\nTotal number of allocated resources : ");
    for(i=0;i<m;i++)
    {
        printf("\t%d",totalAlloc[i]);
    }
    for(i=0;i<m;i++)
    {
        avail[i]=total[i]-totalAlloc[i];
        if(avail[i]<0)
        {
            printf("Total number of allocated resources is greater than total number of resources");
            goto reEnterAllocated;
        }
    }
    //Algorythm for Bankers Algorythm
    printf("\n\nThe processes will be executed according to following order\n\n");
    for(k=0;k<n;k++)
    {
        for(i=0;i<n;i++)
        {
            flag=1;
            if(ind[i]==0)
            {
                for(j=0;j<m;j++)
                {
                    if(avail[j]<(max[i][j]-alloc[i][j]))
                       {
                        flag=0;
                    }
                }
                if(flag==1)
                {
                    printf("P%d --> ",i);
                    ind[i]=1;
                    for(j=0;j<m;j++)
                    {
                        avail[j]= avail[j]+alloc[i][j];
                        alloc[i][j]=0;
                    }
                }
            }
        }
    }
    for(i=0;i<n;i++)
```

```
    {
        if(ind[i]==0)
        {
            printf("It is not a safe case");
            return 0;
        }
    }
    printf("Execution ends\nIt is a safe case");
    return 0;
}
```

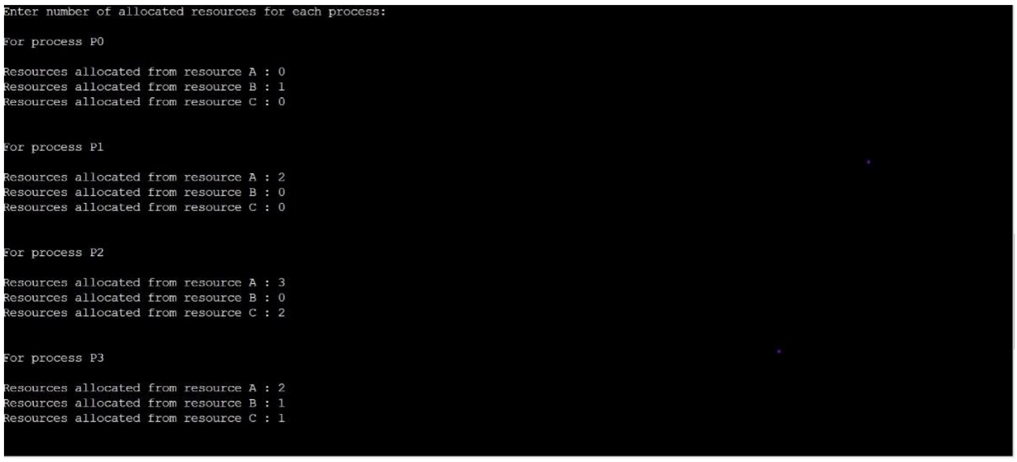
Output



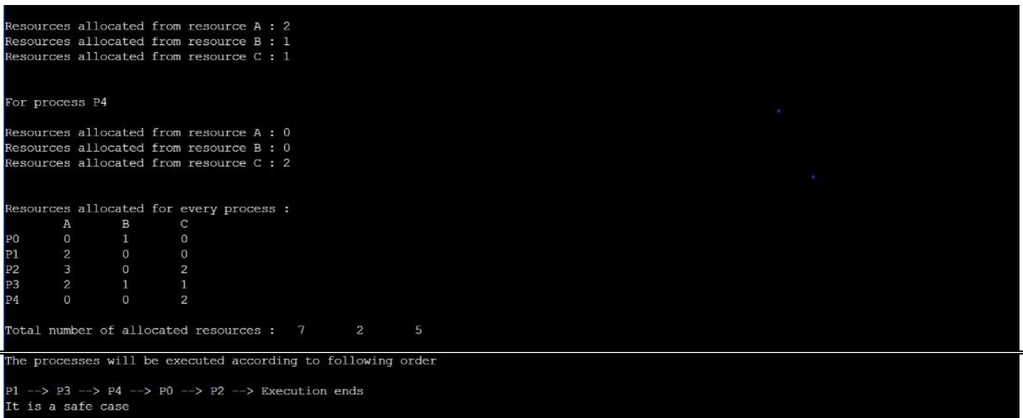
```
Enter number of processes : 5
Enter number of resources : 3

Enter total number of resources available for the surrent execution

Enter total no of resources for resource A : 10
Enter total no of resources for resource B : 5
Enter total no of resources for resource C : 7
Total number of resources :
10      5       7

Enter Maximum number of resources needed for each process:

For process P0

Resources needed from resource A 7
Resources needed from resource B 5
Resources needed from resource C 3


For process P1

Resources needed from resource A 3
Resources needed from resource B 2
Resources needed from resource C 2


For process P2

Resources needed from resource A 9
```



```
For process P2

Resources needed from resource A 9
Resources needed from resource B 0
Resources needed from resource C 2


For process P3

Resources needed from resource A 2
Resources needed from resource B 2
Resources needed from resource C 2


For process P4

Resources needed from resource A 4
Resources needed from resource B 3
Resources needed from resource C 3


Maximum resources needed for every process :
        A       B       C
P0      7       5       3
P1      3       2       2
P2      9       0       2
P3      2       2       2
P4      4       3       3

Enter number of allocated resources for each process:
```



```
Enter number of allocated resources for each process:

For process P0

Resources allocated from resource A : 0
Resources allocated from resource B : 1
Resources allocated from resource C : 0


For process P1

Resources allocated from resource A : 2
Resources allocated from resource B : 0
Resources allocated from resource C : 0


For process P2

Resources allocated from resource A : 3
Resources allocated from resource B : 0
Resources allocated from resource C : 2


For process P3

Resources allocated from resource A : 2
Resources allocated from resource B : 1
Resources allocated from resource C : 1
```



```
Resources allocated from resource A : 2
Resources allocated from resource B : 1
Resources allocated from resource C : 1


For process P4

Resources allocated from resource A : 0
Resources allocated from resource B : 0
Resources allocated from resource C : 2


Resources allocated for every process :
        A       B       C
P0      0       1       0
P1      2       0       0
P2      3       0       2
P3      2       1       1
P4      0       0       2

Total number of allocated resources :   7       2       5

The processes will be executed according to following order

P1 --> P3 --> P4 --> P0 --> P2 --> Execution ends
It is a safe case
```

7.0 Skill developed:

Leadership:

If we have learnt anything this project is that great leadership is an Essential skill to be a good project manager our leadership hole means We lead a manage teem setting in vision and motivating the learn.

8.0 Area of feature:

Using this Project, we can predict that whether the given processes form a safe case or not if yes then what will be order of their execution if not then where which process won't be executed.

Resource Reference:

Sr.no	Title of Book	Author	Published
1	Operating System Concepts Silberschatz	Galvin John Wiley and Sons	Ninth Edition, 2015, ISBN: 978-51-265-5427-0
2	Operating Systems: Internals and Design Principles	Achyut S.	GodboleTata McGraw Hili Education, 2015, ISBN: 978007059113J
3	Unix Concept and Programming Das,	Sumitabha McGraw	Hill education, 2015, ISBN: 978-0070635463

References:

- **http://www.howstuffworks.com/operating-system1.htm**
- **https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/Storage.htm**
- **http://www.computerhope.com/jargon/o/os.htm**
- **http://www.en.wikipedia.org/wiki/Operating**