DEADLOCK AVOIDANCE USING BANKER'S ALGORITHM

Aditya Vardhan Sharma, Akhilesh Sharma, Khushi, Lakshay Magotra

2021a1r031@mietjammu.in
2021a1r035@mietjammu.in
2021a1r039@mietjammu.in
2021a1r054@mietjammu.in

MODEL INSTITUTE OF ENGINEERING AND TECHNOLOGY, CSE DEPARTMENT, KOT BHALWAL, JAMMU, JAMMU & KASHMIR

ABSTRACT

Deadlock is a situation in which two or more processes are blocked and unable to proceed because they are waiting for a resource held by another process. Deadlock avoidance is a technique used to prevent deadlocks from occurring in a system by detecting and resolving potential deadlock situations before they occur. One well-known technique for deadlock avoidance is the Banker's Algorithm, which uses a resource-allocation graph to determine whether a system is in a safe state or not. If a system is in a safe state, it is possible to allocate resources to processes in a way that avoids deadlocks. If a system is not in a safe state, the Banker's Algorithm can be used to identify processes that must be delayed in order to avoid deadlocks. This report discusses the principles and implementation of the Banker's Algorithm for deadlock avoidance, its code and both the safety and request algorithm for deadlock avoidance.

OBJECTIVE

Implementation of most representative Banker's Algorithm in C Language for deadlock avoidance.

Idea is to introduce the application of an example of deadlock avoidance banker's algorithm. In the method of avoiding deadlock, the limited condition is weak, and it is possible to obtain satisfactory system performance. In this method, the system state is divided into safe state and unsafe state. As long as the system is always in safe state, deadlock can be avoided. We can do this by using bankers algorithm.The Banker's Algorithm is a resource allocation and deadlock avoidance algorithm that was developed by Edsger Dijkstra in the 1960s. It is used to ensure that a set of processes requesting resources from a common pool do not end up in a deadlock, where each process is waiting for a resource that is held by another process.

The Banker's Algorithm works by maintaining a set of available resources and a set of maximum resources needed by each process. It then allocates resources to processes in a way that ensures that no process exceeds its maximum resource needs and that no deadlocks occur.

REQUIREMENT ANALYSIS

The requirements of this project are:

1. Linux Operating System
2. GCC
3. Nano Editor
4. Bankers Algorithm

ALGORITHM DESIGN

Safety Algorithm:

1) Let Work and Finish be Arrays of length ‘m’ and ‘n’ respectively.

Initialize: Work = Available

Finish[i] = false; for i=1, 2, 3, 4….n

2) Find an i such that both

a) Finish[i] = false

b) Needi <= Work

if no such i exists goto step (4)

3) Work = Work + Allocation[i]

Finish[i] = true

goto step (2)

4) if Finish [i] = true for all i

then the system is in a safe state

Request Algorithm:

1) If Request i <= Need i

Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2) If Request i <= Available

Goto step (3); otherwise, Pi must wait, since the resources are not available.

3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as

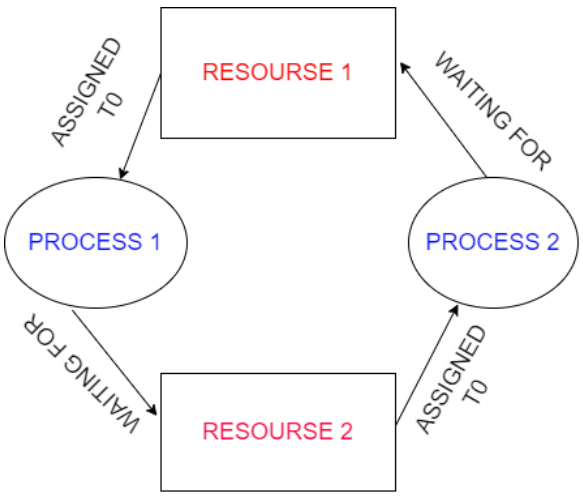
follows:

Available = Available – Request i

Allocation i = Allocation i +Request i

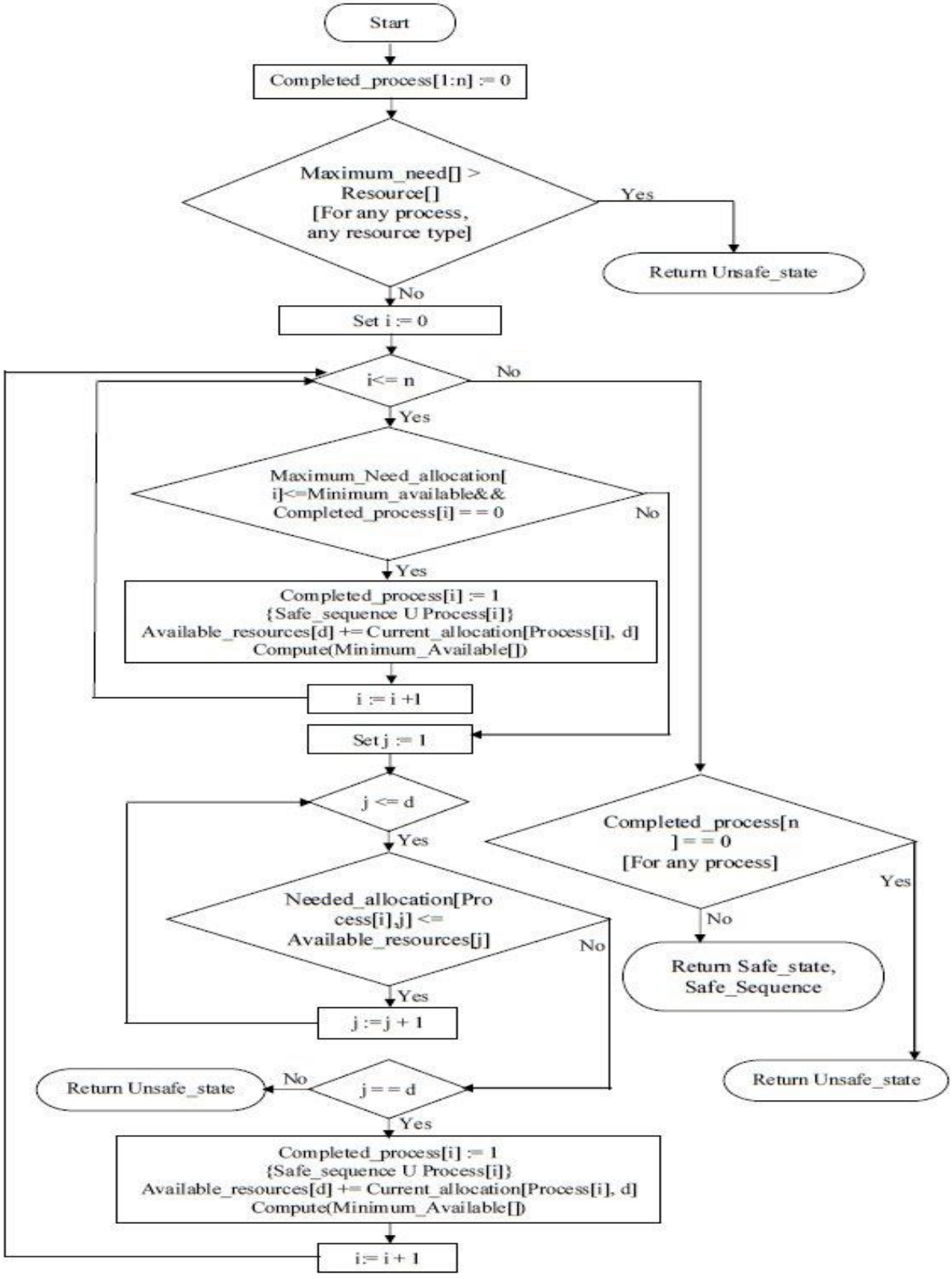
Need i = Need i– Request i

INPUT/OUTPUT DIGRAM



METHODOLOGY

FLOWCHART



C PROGRAM

```
#include<stdio.h>

int max[100][100];

int alloc[100][100];

int need[100][100];

int avail[100];

int n,r;

void input();

void show();

void cal();

int main()

{

 int i,j;

 printf("********** Banker's Algo
************\n");

 input();

 show();

 cal();

 getch();

 return 0;

}void input()

{


int i,j;

 printf("Enter the no of Processes\t");

 scanf("%d",&n);

 printf("Enter the no of resources
instances\t");

 scanf("%d",&r);

 printf("Enter the Max Matrix\n");

 for(i=0;i<n;i++)

 {

  for(j=0;j<r;j++)

  {

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

  }

 }

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

 for(i=0;i<n;i++)

 {

  for(j=0;j<r;j++)

  {

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

  }

 }

 printf("Enter the available Resources\n");

 for(j=0;j<r;j++)

 {

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

 }

}

 void show()

 {
```

```
 int i,j;
 printf("Process\t Allocation\t Max\t
Available\t");
 for(i=0;i<n;i++)
 {
  printf("\nP%d\t   ",i+1);
  for(j=0;j<r;j++)
  {
   printf("%d ",alloc[i][j]);
  }
  printf("\t");
  for(j=0;j<r;j++)
  {
   printf("%d ",max[i][j]);
  }
  printf("\t");
  if(i==0)
  {
   for(j=0;j<r;j++)
   printf("%d ",avail[j]);
  }
 }
}
void cal()
{
 int
finish[100],temp,need[100][100],flag=1,k,c1
=0;
int safe[100];
int i,j;
for(i=0;i<n;i++)
{
 finish[i]=0;
}
//find need matrix
for(i=0;i<n;i++)
{
 for(j=0;j<r;j++)
 {
  need[i][j]=max[i][j]-alloc[i][j];
 }
}
printf("\n");
while(flag)
{
 flag=0;
   for(i=0;i<n;i++)
 {
  int c=0;
  for(j=0;j<r;j++)
  {
   if((finish[i]==0)&&(need[i][j]<=avail[j]))
   {
       c++;
```

```
   if(c==r)
   {
    for(k=0;k<r;k++)
    {
     avail[k]+=alloc[i][j];
     finish[i]=1;
     flag=1;
        }
     printf("P%d->",i);
     if(finish[i]==1)
    {
      i=n;
     }
        }
  }
 }
  }
}
for(i=1;i<n;i++)
{
if(finish[i]==1)
{
 c1++;
 }
 else
 {
  printf("P%d->",i);
 }
}
if(c1==n)
{
 printf("\n The system is in safe state");
}
else
{  printf("\n Process are in dead lock");
printf("\n System is in unsafe state");
}
}
```

OUTPUT

Safe state:

```
********** Banker's Algo ************
Enter the no of Processes        3
Enter the no of resources instances      2
Enter the Max Matrix
4
8
3
8
2
8
Enter the Allocation Matrix
1
9
3
8
6
8
Enter the available Resources
6
9
Process  Allocation       Max      Available
P1          1 9           4 8              6 9
P2          3 8           3 8
P3          6 8           2 8
P1->P2->P3->
 The system is in safe stateaditya@Aditya:~$
```

Unsafe State:

```
********** Banker's Algo ************
Enter the no of Processes        3
Enter the no of resources instances      2
Enter the Max Matrix
5
6
2
55
8
3
Enter the Allocation Matrix
9
2
5
3
9
3
Enter the available Resources
6
9
Process  Allocation       Max      Available
P1          9 2           5 6              6 9
P2          5 3           2 55
P3          9 3           8 3
P2->P3->P1->
 Process are in dead lock
 System is in unsafe stateaditya@Aditya:~$
```

CONCLUSION

In conclusion, the Banker's Algorithm is a valuable tool for avoiding deadlocks in a system that allocates resources to multiple processes. By keeping track of the available resources and the maximum resources needed by each process, the algorithm ensures that no process exceeds its resource needs and that no deadlocks occur. The Banker's Algorithm is widely used in operating systems and other software that manage resources in a multi-tasking environment, and has proven to be an effective method for ensuring the efficient and deadlock-free allocation of resources. While the algorithm is not foolproof and can still lead to deadlocks in certain situations, it is a valuable tool for avoiding deadlocks in many cases.

Git Hub Repository

Aditya Vardhan Sharma
https://github.com/adityavardhansharma?tab=repositories

Akhilesh Sharma
https://github.com/akhilex?tab=repositories

Khushi
https://github.com/khushirajput10

REFERENCES

1. www.geekforgeeks.com
2. http://suraj1693.blogspot.com/2013/11/program-for-bankers-algorithm-for.html
3. https://chat.openai.com
4. www.programiz.com
5. www.javapoint.com
6. www.researchgate.com