REPORT

(COURSE TITLE : OPERATING SYSTEM)

SUBMITTED BY :

NAME : SHUBHAM KUMAR

REG NO : 11809368

ROLL NO : 55

SEC : K18NZ

SUBMITTED TO :

MISS SHAINA GUPTA

EMAIL ADDRESS : shubham139kumar@gmail.com

GITHUB RESPIRATORY : https://github.com/shubhamkr139/Operating-system-project

Banker's Algorithm

The **Banker algorithm**, sometimes referred to as the **detection algorithm**, is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources, and then makes an "s-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue.

The algorithm was developed in the design process for the THE operating system and originally described (in Dutch) in EWD108. When a new process enters a system, it must declare the maximum number of instances of each resource type that it may ever claim; clearly, that number may not exceed the total number of resources in the system. Also, when a process gets all its requested resources it must return them in a finite amount of time.

Banker's algorithm is named so because it is used in banking system to check whether loan can be sanctioned to a person or not. Suppose there are n number of account holders in a bank and the total sum of their money is S. If a person applies for a loan then the bank first subtracts the loan amount from the total money that bank has and if the remaining amount is greater than S then only the loan is sanctioned. It is done because if all the account holders comes to withdraw their money then the bank can easily do it.

In other words, the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers. The bank would try to be in safe state always.

Implementation of Banker's Algorithm

Let **'n'** be the number of processes in the system and **'m'** be the number of resources types

.**Available :**

- It is a 1-d array of size **'m'** indicating the number of available resources of each type.
- Available[j] = k means there are **'k'** instances of resource type $\mathbf{R_j}$

Max :

- It is a 2-d array of size '**n*m'** that defines the maximum demand of each process in a system.
- Max[i, j] = k means process $\mathbf{P_i}$ may request at most **'k'** instances of resource type $\mathbf{R_j}$.

Allocation :

- It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to each process.
- Allocation[i, j] = k means process $\mathbf{P_i}$ is currently allocated **'k'** instances of resource type $\mathbf{R_j}$

Need :

- It is a 2-d array of size **'n*m'** that indicates the remaining resource need of each process.
- Need [i, j] = k means process $\mathbf{P_i}$ currently need **'k'** instances of resource type $\mathbf{R_j}$

for its execution.

- Need [i, j] = Max [i, j] – Allocation [i, j]

$Allocation_i$ specifies the resources currently allocated to process P_i and $Need_i$ specifies the additional resources that process P_i may still request to complete its task.

Banker's(Sefety) Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1) Let Work and Finish be vectors 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

- The time complexity of the Banker's algorithm as a function of the number n of processes and m of resources is o(n*n*m).

Resource-Request Algorithm

Let $Request_i$ be the request array for process P_i. $Request_i[j] = k$ means process P_i wants k instances of resource type R_j. When a request for resources is made by process P_i, the following actions are taken:

1) If Requesti <= Needi

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

2) If Requesti <= 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 – Requesti

Allocationi = Allocationi + Requesti

Needi = Needi– Requesti

Problem :

Reena's operating system uses an algorithm for deadlock avoidance to manage the allocation of resources say three namely A, B, and C to three processes P0, P1, and P2. Consider the following scenario as reference .user must enter the current state of system as given in this example :

Suppose P0 has 0,0,1 instances , P1 is having 3,2,0 instances and P2 occupies 2,1,1 instances of A,B,C resource respectively.

Also the maximum number of instances required for P0 is 8,4,3 and for p1 is 6,2,0 and finally for P2 there are 3,3,3 instances of resources A,B,C respectively. There are 3 instances of resource A, 2 instances of resource B and 2 instances of resource C available. Write a program to check whether Reena's operating system is in a safe state or not in the following independent requests for additional resources in the

current state:

1. Request1: P0 requests 0 instances of A and 0 instances of B and 2 instances of C. 2. Request2: P1 requests for 2 instances of A, 0 instances of B and 0 instances of C.

All the request must be given by user as input.

Code

```
 #include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
 void print(int x[][10],int n,int m)
{
int i,j;
for(i=0;i<n;i++)
 {
            printf("\n");
           for(j=0;j<m;j++)
            {
              printf("%d\t",x[i][j]);
             }
 }
}

// Resource Request
void res_request(int A[10][10],int Y[10][10],int AV[10][10],int pid,int m)
 {
 int reqmat[1][10];
 int i;
 printf("\n Enter additional request : \n");
```

```
for(i=0;i<m;i++)
 {
    printf(" Request for the resource %d : ",i+1);
    scanf("%d",&reqmat[0][i]);
 }

 for(i=0;i<m;i++)
 if(reqmat[0][i] > Y[pid][i])
  {
    printf("\n Error encountered.\n");
    exit(0);
  }
 for(i=0;i<m;i++)
 if(reqmat[0][i] > AV[0][i])
  {
   printf("\n Resources unavailable.\n");
   exit(0);
  }
 for(i=0;i<m;i++)
   {
     AV[0][i]-=reqmat[0][i];
     A[pid][i]+=reqmat[0][i];
     Y[pid][i]-=reqmat[0][i];
```

```
   }
 }
//Algorithm for sefety
int safety(int A[][10],int Y[][10],int AV[1][10],int n,int m,int a[])
{
 int i,j,k,x=0;
 int S[10],R[1][10];
 int pflag=0,flag=0;
 for(i=0;i<n;i++)
 S[i]=0;
 for(i=0;i<m;i++)
 R[0][i]=AV[0][i];
 for(k=0;k<n;k++)
 {
  for(i=0;i<n;i++)
   {
     if(S[i] == 0)
      {
       flag=0;
       for(j=0;j<m;j++)
        {
         if(Y[i][j] > R[0][j])
         flag=1;
```

```
        }
 if(flag == 0 && S[i] == 0)
  {
       for(j=0;j<m;j++)
       R[0][j]+=A[i][j];
       S[i]=1;
       pflag++;
       a[x++]=i;
  }
       }
    }
 if(pflag == n)
 return 1;
 }
 return 0;
}

void accept(int A[][10],int Y[][10],int M[10][10],int R[1][10],int *n,int *m)
{
 int i,j;
 printf("\n Enter total number of processes : ");
 scanf("%d",n);
 printf("\n Enter total number of resources : ");
```

```
 scanf("%d",m);     for(i=0;i<*n;i++)
  {
   printf("\n Process %d\n",i+1);
   for(j=0;j<*m;j++)
   {
   printf(" Allocation for resource %d : ",j+1);
   scanf("%d",&A[i][j]);
   printf(" Maximum for resource %d : ",j+1);
   scanf("%d",&M[i][j]);
   }
  }
printf("\n Available resources : \n");
for(i=0;i<*m;i++)
{
  printf(" Resource %d : ",i+1);
  scanf("%d",&R[0][i]);
}

for(i=0;i<*n;i++)
for(j=0;j<*m;j++)
Y[i][j]=M[i][j]-A[i][j];

printf("\n Allocation Matrix");
```

```
print(A,*n,*m);
printf("\n Maximum Requirement of Matrix are :");
print(M,*n,*m);
printf("\n Need Matrix");
print(Y,*n,*m);

}

int banker(int A[][10],int Y[][10],int R[1][10],int n,int m)
{
  int j,i,a[10];
  j=safety(A,Y,R,n,m,a);
  if(j != 0 )
   {
    printf("\n\n");
    for(i=0;i<n;i++)
    printf(" P%d  ",a[i]);
    printf("\n Safety sequence has been detected.\n");
    return 1;
   }
  else
   {
     printf("\n Deadlock occured.\n");
```

```
    return 0;
   }
}



int main()
{
 int ret;
 int A[10][10];
 int M[10][10];
 int Y[10][10];
 int R[1][10];
 int n,m,pid,ch;
 printf("\nAVOIDANCE DEADLOCK BY USING BANKER'S ALGORITHM\n");
 accept(A,Y,M,R,&n,&m);
 ret=banker(A,Y,R,n,m);
 if(ret !=0 )
  {
   printf("\n Do you want to make additional request ? (1=Yes|0=No)");
   scanf("%d",&ch);
   if(ch == 1)
    {
     printf("\n Enter process number : ");
```

```
    scanf("%d",&pid);
    res_request(A,Y,R,pid-1,m);
    ret=banker(A,Y,R,n,m);
    if(ret == 0 )
    exit(0);
          }
  }
else
exit(0);
return 0;
}
```

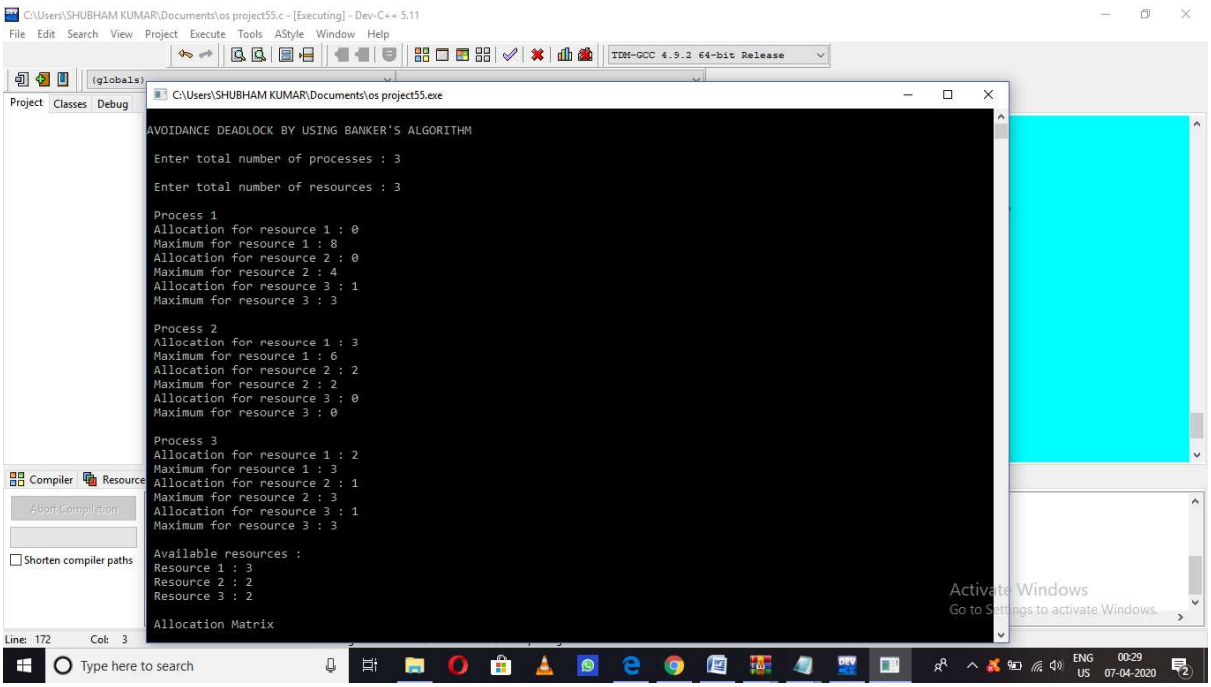
Process	Allocation			Need Max			Available			Remaining Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	0	1	8	4	3	3	2	2	8	4	2
P1	3	2	0	6	2	0				3	0	1
P2	2	1	1	3	3	3				1	2	2

P1----→P0----→P2

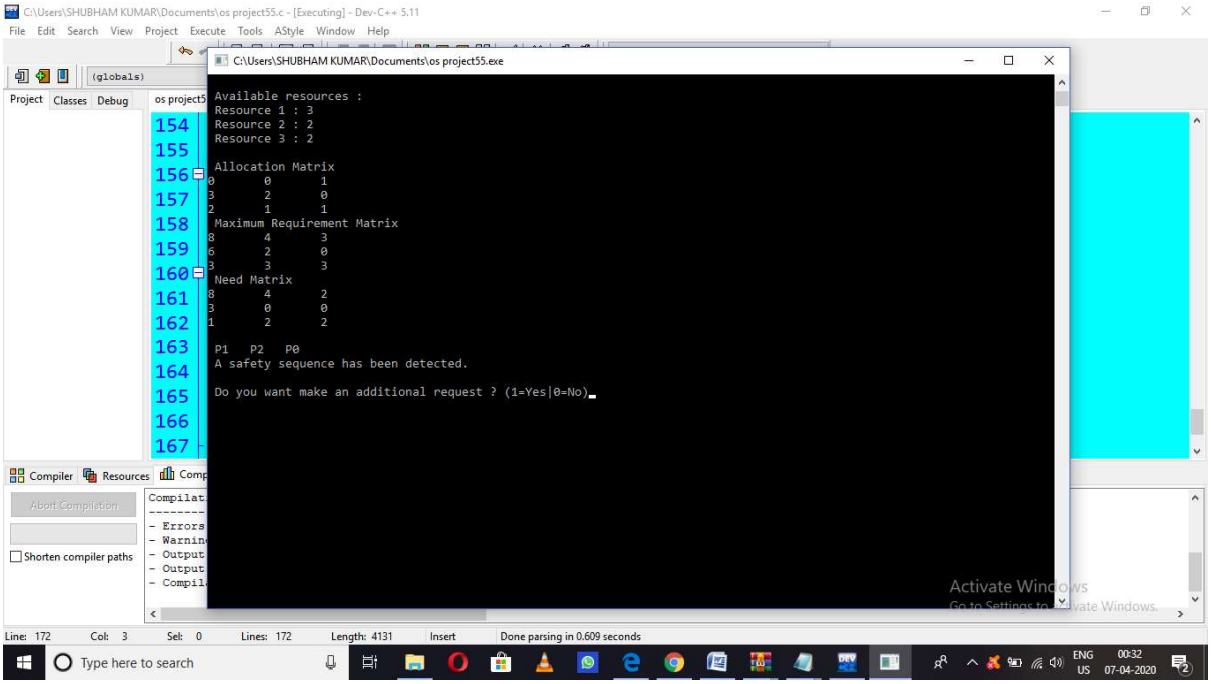
Thus , The process in safe state . So there is no deadlock

Output of Program:

Part 1 : Ask user to enter values of process and resource.



Part 2 : Assign values into matrix ,find need matrix and using safety algorithm it will make sequence of the process and print weather process is in safe state or not.



Part 3: After finishing all process it will ask for additional resource and ask user to input them according to that it will do sequencing of process and print the output.