LOVELY PROFESSIONAL UNIVERSITY

Academic Task-3

School of Computer Science and Engineering Faculty of Technology And Sciences

Course Title: Operating System

Term: Max. Marks:30 **Date of Submission: 22 April,2019**

By

Aman Kumar Soni

Registration No: 11715307

Roll No: 16

Group: 1

Section: K17KH

GitHub Link: github.com/computerwala

Submitted to: Amandeep Kaur

School of Computer Science and Engineering

Lovely Professional University
Phagwara,Punjab

Student Name: Aman Kumar Soni

Email Address: shivsoni377@gmail.com

GitHub Repository : https://github.com/computerwala/Bankers-Algorithm

Banker's algorithm:

The Banker's algorithm is a resource allocation & deadlock avoidance algorithm developed by Edsger Dijkstra that tests for safety by simulating the allocation of pre-determined maximum possible amounts of all resources, and then makes a "safe-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 operating system and originally described (in Dutch) in EWD108. The name is by analogy with the way that bankers account for liquidity constraints.

Algorithm The Banker's algorithm is run by the operating system whenever a process requests resources. The algorithm prevents deadlock by denying or postponing the request if it determines that accepting the request could put the system in an unsafe state (one where deadlock could occur). When a new process enters a system, it must declare the maximum number of instances of each resource type that may not exceed the total number of resources in the system.

Let us assume that there are `n` processes and `m` resource types. Some data structures that are used to implement the banker's algorithm

Safe and Unsafe States

A state is considered safe if it is possible for all processes to finish executing (terminate). Since the system cannot know when a process will terminate, or how many resources it will have requested by then, the system assumes that all processes will eventually attempt to acquire their stated maximum resources and terminate soon afterward. This is a reasonable assumption in most cases since the system is not particularly concerned with how long each process runs (at least not from a deadlock avoidance perspective). Also, if a process terminates without acquiring its maximum resources, it only makes it easier on the system. Given that assumption, the algorithm determines if a state is safe by trying to find a hypothetical set of requests by the processes that would allow each to acquire its maximum resources and then terminate (returning its resources to the system). Any state where no such set exists is an unsafe state.

Banker's Algorithm:

Define quantities:

- AVAILABLE : array [1 .. m] of integer; -- it specifies for each resource how many copies of it are available
- ALLOCATION: array [1..n, 1..m] of integer; -- ALLOCATION[i,j] specifies the number of copies of resource j that are allocated to process i.
- MAXIM: array [1..n, 1..m] of integer; -- MAXIM[i,j] specifies the maximum number of copies of resource j that process i will use.
- NEED; array [1..n, 1..m] of integer; -- NEED[i,j] specifies the number of copies of resource j that process i still requires. It is equal to MAXIM[i,j]-ALLOCATION[i,j]

and the following notation

- A < B, where A and B are m-ary vectors, is true iff for all i, A[i] < B[i]
- If A is a rectangular matrix, Ai is its ith row.

```
procedure BANKER(REQUEST_I: array[1..m] of integer;
                i : 1..n) is
{
    if REQUEST_I > NEED i then
       ERROR;    -- The user is asking more than the agreed maximum
    repeat
        while (REQUEST_i > AVAILABLE)
           yield; -- Resources are not available at this time
        ALLOCATION_i = ALLOCATION_i + REQUEST_i;
        AVAILABLE = AVAILABLE - REQUEST_I;
        if SAFE_STATE then
           RETURN;  -- The request is approved
        ALLOCATION_i = ALLOCATION_I - REQUEST_i;
        AVAILABLE = AVAILABLE + REQUEST_i;
        YIELD;  -- The request cannot safely be satisfied at this time
    forever;
}

BOOLEAN function SAFESTATE is -- Determines if current state is safe
{ NOCHANGE : boolean;
  WORK : array[1..m] of INTEGER = AVAILABLE;
```

```
    FINISH : array[1..n] of boolean = [false, ..,false];
    I : integer;

    repeat
      NOCHANGE = TRUE;
      for I = 1 to N do
        if ((not FINISH[i]) and
           NEEDi <= WORK) then {
           WORK = WORK + ALLOCATION_i;
              FINISH[i] = true;
           NOCHANGE = false;
        }
    until NOCHANGE;
    return (FINISH == (true, .., true));
  }
```

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).

Banker's (Safety) Algorithm:

find a safe sequence,

i.e. is the system in a safe state?

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

   Initialize Work = Available, and Finish[i]=false for i=0,...,n-1

2. Find a process i such that both

   • Finish[i] = = false, and

   • Needi ≤ Work

   If no such i exists, go to step 4.

3. Work = Work + Alloci

   Finish[i] = true

   Go to step 2.

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

Given Problem: (Code)

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.

Proposed Solution:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.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 algorithm
void res_request(int A[10][10],int N[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 resource %d : ",i+1);
```

```
        scanf("%d",&reqmat[0][i]);
    }

    for(i=0;i<m;i++)
        if(reqmat[0][i] > N[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];
        N[pid][i]-=reqmat[0][i];
    }
}

//Safety algorithm
int safety(int A[][10],int N[][10],int AV[1][10],int n,int m,int a[]){

    int i,j,k,x=0;
    int F[10],W[1][10];
    int pflag=0,flag=0;
    for(i=0;i<n;i++)
        F[i]=0;
    for(i=0;i<m;i++)
        W[0][i]=AV[0][i];

    for(k=0;k<n;k++){
        for(i=0;i<n;i++){
            if(F[i] == 0){
                flag=0;
                for(j=0;j<m;j++){
                    if(N[i][j] > W[0][j])
                        flag=1;
                }
                if(flag == 0 && F[i] == 0){
                    for(j=0;j<m;j++)
                        W[0][j]+=A[i][j];
                    F[i]=1;
                    pflag++;
```

```
                a[x++]=i;
            }
        }
    }
    if(pflag == n)
        return 1;
  }
  return 0;
}


//Banker's Algorithm
void accept(int A[][10],int N[][10],int M[10][10],int W[1][10],int *n,int *m){
  int i,j;
  printf("\n Enter total no. of processes : ");
  scanf("%d",n);
  printf("\n Enter total no. 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",&W[0][i]);
  }

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

  printf("\n Allocation Matrix");
  print(A,*n,*m);
  printf("\n Maximum Requirement Matrix");
  print(M,*n,*m);
  printf("\n Need Matrix");
  print(N,*n,*m);

}
```

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


int main(){
   int ret;
   int A[10][10];
   int M[10][10];
   int N[10][10];
   int W[1][10];
   int n,m,pid,ch;
   printf("\n DEADLOCK AVOIDANCE USING BANKER'S ALGORITHM\n");
   accept(A,N,M,W,&n,&m);
   ret=banker(A,N,W,n,m);
   if(ret !=0 ){
      printf("\n Do you want make an additional request ? (1=Yes|0=No)");
      scanf("%d",&ch);
      if(ch == 1){
         printf("\n Enter process no. : ");
         scanf("%d",&pid);
         res_request(A,N,W,pid-1,m);
         ret=banker(A,N,W,n,m);
         if(ret == 0 )
            exit(0);
      }
   }else
      exit(0);
   return 0;
}
```

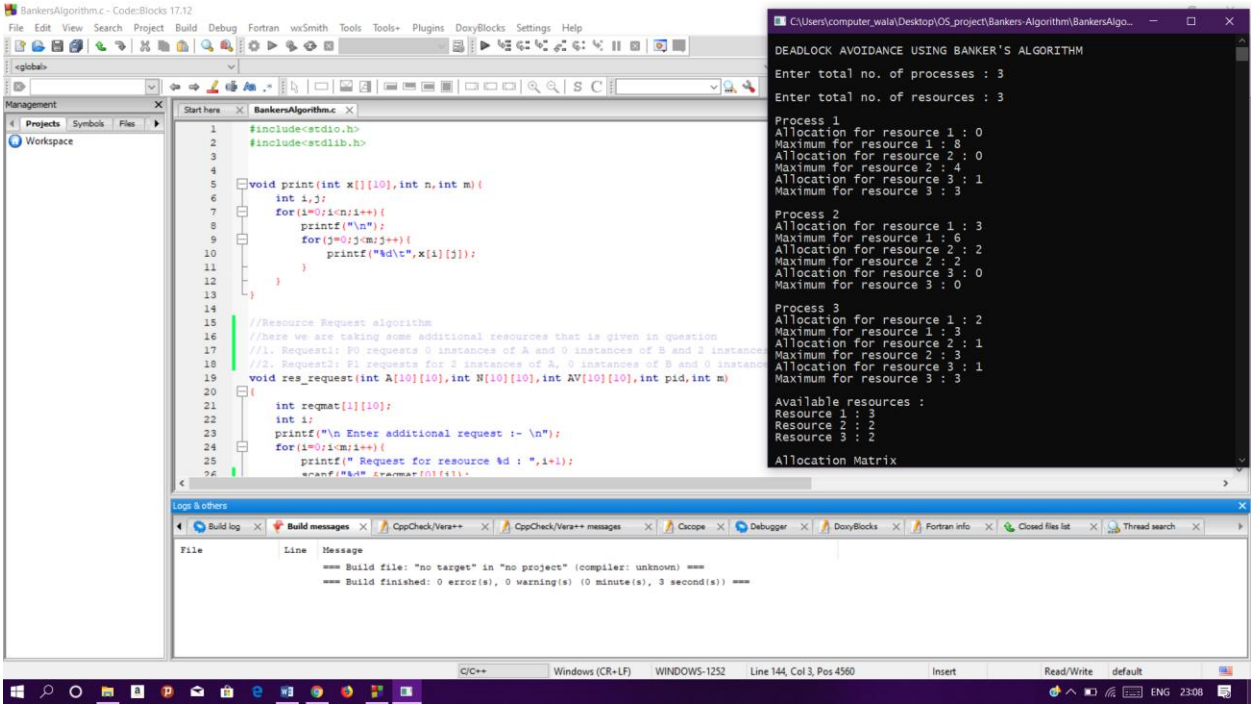
Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	0	1	8	4	3	3	2	2
P1	3	2	0	6	2	0			
P2	2	1	1	3	3	3			

Need		
A	B	C
8	4	2
3	0	1
1	2	2

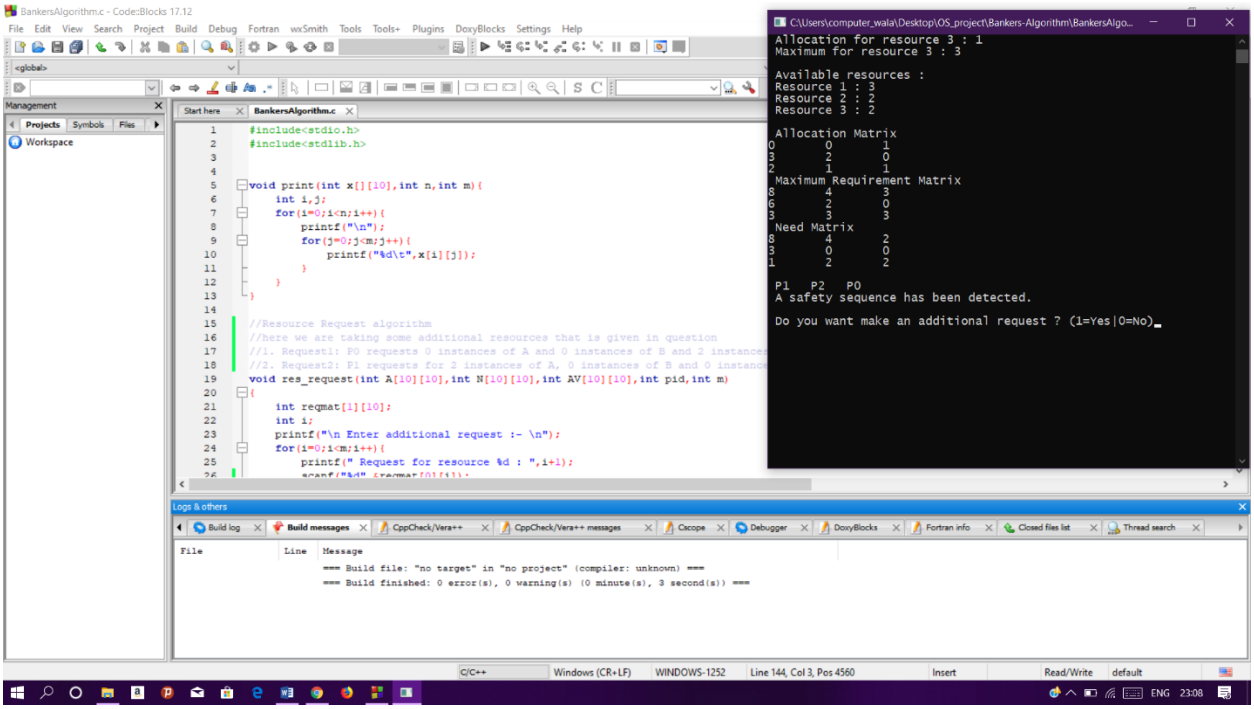
P1---------→P2---------→P0

The process will be in safe state

Output of Program: Part 1 ask user to enter values of process and resource



Part 2 : It will 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