**16.a) BANKERS ALGORITHM For Deadlock Avoidance**

**AIM:**

To stimulate banker’s algorithm for dead lock avoidance.

**DESCRIPTION**:

Deadlock is a state in which a process is waiting for the resource that is already used by another process and that process is waiting for another resource. There are some techniques used to avoid deadlocks. There are two states involved in it. They are:

1. Safe state

2. Unsafe state

**ALGORITHM:**

**Step 1:**start

**Step 2:** declare int Max[10][10], need[10][10], alloc[10][10],avail[10],completed[10],safeSequence[10] and int p, r, i, j, process, count

**Step 3:** initialize count = 0

**Step 4:** read the no of processes (p)

**Step 5:** for i<-0 repeat upto p

completed[i] = 0

**Step 6:** read the no of resources (r)

**Step 7:** read Max Matrix for each process

For i<-0 repeat upto p

begin

read For process %d : ", i + 1

For i<-0 repeat upto p

Write Max[i][j]

End for

**Step 8:** read “Enter the allocation for each process : "

For i<-0 repeat upto p

begin

read For process %d :,i + 1

For j<-0 repeat upto r

Write alloc[i][j]

End for

**Step 9:** read “Enter the Available Resources : "

For j<-0 repeat upto r

write avail[i][j]

For i<-0 repeat upto p

For j<-0 repeat upto r

need[i][j] = Max[i][j] - alloc[i][j];

**Step 10:** do

begin

read Max matrix and Allocation matrix

For i<-0 repeat upto p

begin

For j<-0 repeat upto r

Read Max[i][j]);

For j<-0 repeat upto r

Read alloc[i][j]

End for

process = -1;

For i<-0 repeat upto p

begin

if(completed[i] == 0)

begin

process = 1

For j<-0 repeat upto r

begin

if(avail[j] < need[i][j])

begin

process = -1

break

end if

end for

end if

if(process != -1)

break;

if(process != -1)

begin

write “Process %d runs to completion!", process + 1

safeSequence[count] = process + 1

count++

For j<-0 repeat upto r

begin

avail[j] += alloc[process][j]

alloc[process][j] = 0

Max[process][j] = 0

completed[process] = 1

end for

end if

end do

while(count != p && process != -1);

**Step 11:** if(count == p)

Begin

Read “The system is in a safe state!!”

Read "Safe Sequence : < “

For i<-0 repeat upto p

Read“safeSequence[i]”

Read ">”

else

read “The system is in an unsafe state!!"

end if

**Step 12:**stop

**SOURCE CODE:**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];

int p, r, i, j, process, count;

count = 0;

printf("Enter the no of processes : ");

scanf("%d", &p);

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

completed[i] = 0;

printf("\n\nEnter the no of resources : ");

scanf("%d", &r);

printf("\n\nEnter the Max Matrix for each process : ");

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

{

printf("\nFor process %d : ", i + 1);

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

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

}

printf("\n\nEnter the allocation for each process : ");

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

{

printf("\nFor process %d : ",i + 1);

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

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

}

printf("\n\nEnter the Available Resources : ");

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

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

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

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

need[i][j] = Max[i][j] - alloc[i][j];

do

{

printf("\n Max matrix:\tAllocation matrix:\n");

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

{

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

printf("%d ", Max[i][j]);

printf("\t\t");

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

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

printf("\n");

}

process = -1;

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

{

if(completed[i] == 0)//if not completed

{

process = i ;

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

{

if(avail[j] < need[i][j])

{

process = -1;

break;

}

}

}

if(process != -1)

break;

}

if(process != -1)

{

printf("\nProcess %d runs to completion!", process + 1);

safeSequence[count] = process + 1;

count++;

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

{

avail[j] += alloc[process][j];

alloc[process][j] = 0;

Max[process][j] = 0;

completed[process] = 1;

}

}

}

while(count != p && process != -1);

if(count == p)

{

printf("\nThe system is in a safe state!!\n");

printf("Safe Sequence : < ");

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

printf("%d ", safeSequence[i]);

printf(">\n");

}

else

printf("\nThe system is in an unsafe state!!");

}

**OUTPUT:**

Enter number of process:5

Enter number of resources:3

Enter Maximum matrix for each process:

for process 1:7 5 3

for process 2:3 2 2

for process 3:9 0 2

for process 4:2 2 2

for process 5:4 3 3

Enter allocation matrix:

for process 1:0 1 0

for process 2:2 0 0

for process 3:3 0 2

for process 4:2 1 1

for process 5:0 0 2

enter available resources: 3 3 2

Allocated matrix Max

0 1 0| 7 5 3

2 0 0| 3 2 2

3 0 2| 9 0 2

2 1 1| 2 2 2

0 0 2| 4 3 3

Process 2 runs to completion

Allocated matrix Max

0 1 0| 7 5 3

0 0 0| 0 0 0

3 0 2| 9 0 2

2 1 1| 2 2 2

0 0 2| 4 3 3

Process 4 runs to completion

Allocated matrix Max

0 1 0| 7 5 3

0 0 0| 0 0 0

3 0 2| 9 0 2

0 0 0| 0 0 0

0 0 2| 4 3 3

Process 1 runs to completion

Allocated matrix Max

0 0 0| 0 0 0

0 0 0| 0 0 0

3 0 2| 9 0 2

0 0 0| 0 0 0

0 0 2| 4 3 3

Process 3 runs to completion

Allocated matrix Max

0 0 0| 0 0 0

0 0 0| 0 0 0

0 0 0| 0 0 0

0 0 0| 0 0 0

0 0 2| 4 3 3

Process 5 runs to completion

The system is in a safe state!!

Safe sequence : < 2 4 1 3 5>

**b). BANKERS ALGORITHM For Deadlock Detection**

**AIM:**

To stimulate deadlock detection algorithm

**DESCRIPTION:**

Deadlock detection is the process of actually determining that a deadlock exists and identifying the processes and resources involved in the deadlock.

The basic idea is to check allocation against resource availability for all possible allocation sequences to determine if the system is in deadlocked state. Deadlock detection algorithm is one of this strategy.

**ALGORITHM:**

**Step 1:** Start

**Step 2:** declare mark[20] as static int and i,j,np,nr as integer globally

**Step 3:** declare int alloc[10][10],request[10][10],avail[10],r[10],w[10]

**Step 4:** read no of process(np)

**Step 5:**read no of resources(nr)

**Step 6:** for I<-0 repeat upto nr

Read”Total Amount of the Resource R%d: ",i+1

**Step 7:**read “Enter the request matrix:”

**Step 8:**for I<-0 repeat upto np

Begin

for j<-0 repeat upto nr

read request[i][j]

**Step 9:** read the allocation matrix

for I<-0 repeat upto np

Begin

**Step 10:**for j<-0 repeat upto nr

read alloc[i][j]

for j<-0 repeat upto nr

avail[j]=r[j]

**Step 11:** for I<-0 repeat upto np

Int count=0

for j<-0 repeat upto nr

**Step 12:**if(alloc[i][j]==0)

count++

else

break

if(count==nr)

mark[i]=1

**Step 13:**for j<-0 repeat upto nr

w[j]=avail[j];

for i<-0 repeat upto np

begin

Int canbeprocessed=0

if(mark[i]!=1)

begin

for j<-0 repeat upto nr

begin

if(request[i][j]<=w[j])

canbeprocessed=1

else

begin

canbeprocessed=0

break

end else

end for

if(canbeprocessed)

begin

mark[i]=1;

for j<-0 repeat upto nr

w[j]+=alloc[i][j];

end if

end if

end for

**Step 14:**int deadlock=0

for i<-0 repeat upto np

if(mark[i]!=1)

deadlock=1

if(deadlock)

write “Deadlock detected”

else

write “No Deadlock possible”

**Step 15:** stop

**SOURCE CODE:**

#include<stdio.h>

static int mark[20];

int i,j,np,nr;

int main()

{

int alloc[10][10],request[10][10],avail[10],r[10],w[10];

printf("\nEnter the no of process: ");

scanf("%d",&np);

printf("\nEnter the no of resources: ");

scanf("%d",&nr);

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

{

printf("\nTotal Amount of the Resource R%d: ",i+1);

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

}

printf("\nEnter the request matrix:");

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

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

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

printf("\nEnter the allocation matrix:");

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

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

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

/\*Available Resource calculation\*/

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

{

avail[j]=r[j];

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

{

avail[j]-=alloc[i][j];

}

}

//marking processes with zero allocation

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

{

int count=0;

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

{

if(alloc[i][j]==0)

count++;

else

break;

}

if(count==nr)

mark[i]=1;

}

// initialize W with avail

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

w[j]=avail[j];

//mark processes with request less than or equal to W

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

{

Int canbeprocessed=0;

if(mark[i]!=1)

{

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

{

if(request[i][j]<=w[j])

canbeprocessed=1;

else

{

canbeprocessed=0;

break;

}

}

if(canbeprocessed)

{

mark[i]=1;

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

w[j]+=alloc[i][j];

}

}

}

//checking for unmarked processes

int deadlock=0;

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

if(mark[i]!=1)

deadlock=1;

if(deadlock)

printf("\n Deadlock detected");

else

printf("\n No Deadlock possible");

}

**OUTPUT:**

Enter no of process 5

Enter no of resources 3

Total amount of resource R1: 7

Total amount of resource R2: 2

Total amount of resource R3: 6

Enter the request matrix: 0 0 0

2 0 2

0 0 0

1 0 0

0 0 2

Enter the allocation matrix: 0 1 0

2 0 0

3 0 3

2 1 1

0 0 2

No deadlock detected

**c.) BANKERS ALGORITHM For Deadlock Prevention**

**AIM:**

To stimulate banker’s algorithm for dead lock prevention.

**DESCRIPTION:**

Deadlock prevention algorithms ensure that atleast one of the necessary conditions such as 1. Mutual exclusion

2. Hold and wait

3. No preemption

4. Circular wait does not hold true

However most prevention algorithms have poor resource utilization and hence result in reduced throughputs.

**ALGORITHM:**

**Step 1:** Start

**Step 2:** declare allocated[15][15],max[15][15],need[15][15],avail[15],tres[15],work[15],flag[15] as int and declare int pno,rno,i,j,prc,count,t,total

**Step 3:** initialize count=0

**Step 4:** read number of process (pno)

**Step 5:**read number of resources(rno)

For i<-1 repeat upto pno

Begin

flag[i]<-0;

end

**Step 6:** read total number of each resources

For i<-1 repeat upto rno

Write tres[i]

**Step 7:** read Max resources for each process

For i<-1 repeat upto pno

Begin

Read “for process %d:",i

For j<-1 repeat upto rno

Write max[i][j]

End for

**Step 8:**read “enter allocated resources for each process:"

For i<-1repeat upto pno

begin

read for process %d:",i

for j<-1repeat upto rno

write allocated[i][j]);

end

**Step 9:**read “available resources:\n"

For j-<1 repeat upto rno

begin

avail[j]<-0;

total<-0;

for i<-1 repeat upto pno

begin

total+=allocated[i][j]

end for

avail[j]<-tres[j]-total

work[j]<-avail[j]

read work[j]

end for

**Step 10:**do

Begin

For i<-1 repeat upto pno

begin

for j<-1 repeat upto rno

begin

need[i][j]<-max[i][j]-allocated[i][j]

end for

end for

read “ Allocated matrix Max need”

for i<-1 repeat upto pno

begin

for j<-1 repeat upto rno

begin

read allocated[i][j]

end for

read "|"

for j<-1 repeat upto rno

begin

read max[i][j]

read "|"

for j<-1 repeat upto rno

begin

read need[i][j]

end for

end for

prc<-0;

for i<-1 repeat upto rno

begin

if(flag[i]==0)

begin

prc=i;

for j<-1 repeat upto rno

begin

if(work[j]< need[i][j])

begin

prc=0

break

end if

end for

end if

if(prc!=0)

break

end for

if(prc!=0)

begin

read“Process %d completed",i

count++

read” Available matrix:"

for j<-1 repeat upto rno

begin

work[j]+=allocated[prc][j]

allocated[prc][j]=0

max[prc][j]=0

flag[prc]=1

read work[j])

end for

end if

end if

while(count!=pno&&prc!=0)

**Step 11:** if(count==pno)

read “The system is in a safe state!!"

else

read “The system is in an unsafe state!!"

**Step 12:** stop

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

void main()

{

int allocated[15][15],max[15][15],need[15][15],avail[15],tres[15],work[15],flag[15];

int pno,rno,i,j,prc,count,t,total;

count=0;

printf("\n Enter number of process:");

scanf("%d",&pno);

printf("\n Enter number of resources:");

scanf("%d",&rno);

for(i=1;i<=pno;i++)

{

flag[i]=0;

}

printf("\n Enter total numbers of each resources:");

for(i=1;i<=rno;i++)

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

printf("\n Enter Max resources for each process:");

for(i=1;i<= pno;i++)

{

printf("\n for process %d:",i);

for(j=1;j<=rno;j++)

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

}

printf("\n Enter allocated resources for each process:");

for(i=1;i<=pno;i++)

{

printf("\n for process %d:",i);

for(j=1;j<= rno;j++)

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

}

printf("\n available resources:\n");

for(j=1;j<=rno;j++)

{

avail[j]=0;

total=0;

for(i=1;i<=pno;i++)

{

total+=allocated[i][j];

}

avail[j]=tres[j]-total;

work[j]=avail[j];

printf(" %d \t",work[j]);

}

do

{

for(i=1;i<=pno;i++)

{

for(j=1;j<=rno;j++)

{

need[i][j]=max[i][j]-allocated[i][j];

}

}

printf("\n Allocated matrix Max need");

for(i=1;i<=pno;i++)

{

printf("\n");

for(j=1;j<=rno;j++)

{

printf("%4d",allocated[i][j]);

}

printf("|");

for(j=1;j<=rno;j++)

{

printf("%4d",max[i][j]);

}

printf("|");

for(j=1;j<=rno;j++)

{

printf("%4d",need[i][j]);

}

}

prc=0;

for(i=1;i<= pno;i++)

{

if(flag[i]==0)

{

prc=i;

for(j=1;j<= rno;j++)

{

if(work[j]< need[i][j])

{

prc=0;

break;

}

}

}

if(prc!=0)

break;

}

if(prc!=0)

{

printf("\n Process %d completed",i);

count++;

printf("\n Available matrix:");

for(j=1;j<= rno;j++)

{

work[j]+=allocated[prc][j];

allocated[prc][j]=0;

max[prc][j]=0;

flag[prc]=1;

printf(" %d",work[j]);

}

}

}while(count!=pno&&prc!=0);

if(count==pno)

printf("\nThe system is in a safe state!!");

else

printf("\nThe system is in an unsafe state!!");

}

**OUTPUT:**

Enter number of process:5

Enter number of resources:3

Enter total numbers of each resources:10 5 7

Enter Max resources for each process:

for process 1:7 5 3

for process 2:3 2 2

for process 3:9 0 2

for process 4:2 2 2

for process 5:4 3 3

Enter allocated resources for each process:

for process 1:0 1 0

for process 2:3 0 2

for process 3:3 0 2

for process 4:2 1 1

for process 5:0 0 2

available resources:

2 3 0

Allocated matrix Max need

0 1 0| 7 5 3| 7 4 3

3 0 2| 3 2 2| 0 2 0

3 0 2| 9 0 2| 6 0 0

2 1 1| 2 2 2| 0 1 1

0 0 2| 4 3 3| 4 3 1

Process 2 completed

Available matrix: 5 3 2

Allocated matrix Max need

0 1 0| 7 5 3| 7 4 3

0 0 0| 0 0 0| 0 0 0

3 0 2| 9 0 2| 6 0 0

2 1 1| 2 2 2| 0 1 1

0 0 2| 4 3 3| 4 3 1

Process 4 completed

Available matrix: 7 4 3

Allocated matrix Max need

0 1 0| 7 5 3| 7 4 3

0 0 0| 0 0 0| 0 0 0

3 0 2| 9 0 2| 6 0 0

0 0 0| 0 0 0| 0 0 0

0 0 2| 4 3 3| 4 3 1

Process 1 completed

Available matrix: 7 5 3

Allocated matrix Max need

0 0 0| 0 0 0| 0 0 0

0 0 0| 0 0 0| 0 0 0

3 0 2| 9 0 2| 6 0 0

0 0 0| 0 0 0| 0 0 0

0 0 2| 4 3 3| 4 3 1

Process 3 completed

Available matrix: 10 5 5

Allocated matrix Max need

0 0 0| 0 0 0| 0 0 0

0 0 0| 0 0 0| 0 0 0

0 0 0| 0 0 0| 0 0 0

0 0 0| 0 0 0| 0 0 0

0 0 2| 4 3 3| 4 3 1

Process 5 completed

Available matrix: 10 5 7

The system is in a safe state!!