EXPERIMENT 7

Experiment 7

Aim: To implement bankers algorithm for deadlock avoidance.

Theory:

The Banker's Algorithm ~~developed by~~ is a deadlock avoidance algorithm used in operating systems. It ensures that a system can allocate resources to each process in a safe manner, avoiding deadlock before it happens. It is named so because it is analogous to how a banker might allocate available cash to customers such that the bank never runs out of resources.

- **Concepts:**

i) **Safe state:** A system is in a safe state if there exists a sequence of all processes such that each process can be allocated its maximum resources even if all other processes use their maximum simultaneously.

ii) **Unsafe state:** An unsafe state does not imply deadlock but it could lead to one.

iii) **Deadlock:** A condition where a set of processes are blocked because each process is holding a resource.

and waiting for another held by others.

- Working:

i) Each process declares it maximum resource needs at the beginning.

ii) The system maintains the following data structure:

→ Available : Resources currently available.
→ Max : Maximum demand of each process.
→ Allocation : Resources currently allocated.
→ Need : Remaining resources needed.

iii) When a process requests resources, check:
→ Request [resources] <= Need [resources].
→ Request [resources] <= Available [resources].

iv) if both conditions are true, allocate resources & update the matrices accordingly.

v) The system checks if it is in safe mode.

vi) if system is safe, request is granted otherwise denied.

- Conclusion:

The Banker's Algorithm is an effective method for avoiding deadlocks by carefully checking resource requests against system safety. It allows process to run without conflict by ~~or~~ ensuring a safe sequence of execution. While it adds complexity & needs predefined maximum demands, it provides a strong framework for safe resource allocation.

Code:

```
// total resources=3
// total processes=n
import java.util.*;
class Main {
public static void main(String[] args) {
int n;

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number of processes:");
n=sc.nextInt();

int all[][]=new int[n][3];
int totalAll[]=new int[3];
int total[]=new int [3];
int avl[]=new int [3];
int claim[][]=new int[n][3];
int rem[][]=new int [n][3];


System.out.println("Enter total resources:");

for (int i=0; i<3; i++) {
System.out.println("Resource "+(i+1));
total[i]=sc.nextInt();
}

System.out.println("Total Resouces:");
System.out.println("1|2|3");
for (int i=0; i<3; i++) {
System.out.print(total[i]+"|");
}

System.out.println();

System.out.println("Allocate resources for "+n+" processes :");

for (int i=0; i<n; i++) {
System.out.print("Process "+(i+1)+": ");
for (int j=0; j<3; j++) {
all[i][j]=sc.nextInt();
totalAll[j]+=all[i][j];
}
System.out.println();
}
```

```
System.out.println("Allocated resources:");
for (int i=0; i<n; i++) {
System.out.println("Process "+(i+1)+": ");
for (int j=0; j<3; j++) {
System.out.print(all[i][j]+" ");
}
System.out.println();
}
System.out.println("Available Resources:");

for (int i=0; i<3; i++) {
avl[i]=total[i]-totalAll[i];
System.out.print(avl[i]+" ");
}
System.out.println();
System.out.println("Claim resources for "+n+" processes :");

for (int i=0; i<n; i++) {
System.out.print("Process "+(i+1)+": ");
for (int j=0; j<3; j++) {
claim[i][j]=sc.nextInt();
}
System.out.println();
}

System.out.println("Claimed resources:");
for (int i=0; i<n; i++) {
System.out.print("Process "+(i+1)+": ");
for (int j=0; j<3; j++) {
System.out.print(claim[i][j]+" ");
}
System.out.println();
}


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

for (int j=0; j<3; j++) {
rem[i][j]=claim[i][j]-all[i][j];
}
}


System.out.println("Remaining resources:");
for (int i=0; i<n; i++) {
System.out.print("Process "+(i+1)+": ");
for (int j=0; j<3; j++) {
System.out.print(rem[i][j]+" ");
```

```
}
System.out.println();
}

ArrayList<Integer>arr=new ArrayList<>();
int count=0;
boolean isDone[]= new boolean[n];
boolean allDone=false;
while(count<n && allDone==false) {

allDone=true;
for (int i=0; i<n; i++) {

if(isDone[i]==false) {
allDone=false;
break;
}
}



for (int i=0; i<n; i++) {
 System.out.println("Available Resources after checking Process "+(i+1)+": ");
     for (int x=0; x<3; x++) {
System.out.print(avl[x]+" ");
}
System.out.println();
if(rem[i][0]<=avl[0] && rem[i][1]<=avl[1] && rem[i][2]<=avl[2]) {
rem[i][0]=0;
rem[i][1]=0;
rem[i][2]=0;
if(isDone[i]==false)
{
arr.add(i);
avl[0]+=all[i][0];
avl[1]+=all[i][1];
avl[2]+=all[i][2];
isDone[i]=true;
count=0;
}
} else {
count++;
}
}


for (int x=0; x<n; x++) {
System.out.print(isDone[x]+" ");
```

```
}
System.out.println();

}
boolean safe=true;
for (int i=0; i<n; i++) {

if(isDone[i]==false) {
safe=false;
break;
}
}


if(safe) {
System.out.println("Safe. No Deadlock will occur.");
for (Integer i:arr) {
System.out.print("P"+(i+1)+" ");
}
} else {
System.out.println("Unsafe. Deadlock will occur");
}


}
}
```

Output:

Enter the number of processes:

5

Enter total resources:

Resource 1

10 5 7

Resource 2

Resource 3

Total Resouces:

1|2|3

10|5|7|

Allocate resources for 5 processes :

Process 1: 0 1 0

Process 2: 2 0 0

Process 3: 3 0 2

Process 4: 2 1 1

Process 5: 0 0 2

Allocated resources:

Process 1:

0 1 0

Process 2:

2 0 0

Process 3:

3 0 2

Process 4:

2 1 1

Process 5:

0 0 2

Available Resources:

3 3 2

Claim resources for 5 processes :

Process 1: 7 5 3

Process 2: 3 2 2

Process 3: 9 0 2

Process 4: 4 2 2

Process 5: 5 3 3

Claimed resources:

Process 1: 7 5 3

Process 2: 3 2 2

Process 3: 9 0 2

Process 4: 4 2 2

Process 5: 5 3 3

Remaining resources:

Process 1: 7 4 3

Process 2: 1 2 2

Process 3: 6 0 0

Process 4: 2 1 1

Process 5: 5 3 1

Available Resources after checking Process 1:

3 3 2

Available Resources after checking Process 2:

3 3 2

Available Resources after checking Process 3:

5 3 2

Available Resources after checking Process 4:

5 3 2

Available Resources after checking Process 5:

7 4 3

false true false true true

Available Resources after checking Process 1:

7 4 5

Available Resources after checking Process 2:

7 5 5

Available Resources after checking Process 3:

7 5 5

Available Resources after checking Process 4:

10 5 7

Available Resources after checking Process 5:

10 5 7

true true true true true

Available Resources after checking Process 1:

10 5 7

Available Resources after checking Process 2:

10 5 7

Available Resources after checking Process 3:

10 5 7

Available Resources after checking Process 4:

10 5 7

Available Resources after checking Process 5:

10 5 7

true true true true true

Safe. No Deadlock will occur.

P2 P4 P5 P1 P3