System Programming and Operating Systems Lab

ASSIGNMENT 12

Name: Shrirang Mhalgi
Roll No: 322008
Batch: B1

1 Date of Completion:

26/03/2019

2 Aim:

Write a Java program to impliment Banker's Algorithm.

3 Objectives:

To implement Banker's Algorithm by using Java.
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.

4 Theory:

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that test for a safety by simulating the allocation for predetermined maximum on "S-State" check to text for a possible should be allowed to continue
The Banker's Algorithm derives its name from the fact that this algorithm could be used in a banking system to ensure that the bank does not run out of resources, because the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers[citation needed]. By using the Banker's algorithm, the bank ensures that when customers request money the bank never leaves a safe state. If the customer's request does not cause the bank to leave a safe state, the cash will be allocated, otherwise the customer must wait until some other customer deposits enough.
Data structures for the banker's algorithm,
let,
n= Number of processes.
m= Number of resource types.

Available:

Vector of length m, available [ ]=k, there are k instances of resources type Rj correctly available.

Max:

nxm matrix if max is [i,j]=k, then process pi will request of most k instances of resources type RJ.

Allocation:

nxm matrix if allo [i,j]=k then the currently allocates(i.e holding) is k. Instance of Rj.

Need:

nxm matrix if need [i,j]=k then it may need k more instances of Rj to complete its task.

Need[i,j] = Max[i,j] - Allocation[i,j]. n=m-a.

5 Safety Algorithm:

1. Let work and finish be vectors of length mond n resp.
initialize
work = Available
finish [i] == false for i=1,2,3,….. n.

2. Find an i such that both
finish [i] == false.
Need = work.
if no such i exists go to step4.

3. work = workAllocation
finish [i] = true
go to step2.

4. If finish [i] = true for all i, then the sim is in safe state.

6 Resource Request Algorithm for process pi:

request i = request vector for pi
request [i] = k means process pi wants k instance of resource type pj.

1. If request i = Need i goto step2 - otherwise error.

2. If request i = Available go to step3 - otherwise pi burst wait.

3. ”Allocate” requested resources to pi as follows,

   Available = Available_Request;

   Alloct = Alloci + Request i;

   Need = Need i - Request;

   if safe = the resources are allocated to pi.

   if unsafe = restre the old resource allocation state and block pi.

C Program:

```
import java.util.Scanner;
public class Bankers{
private int need[][],allocate[][],max[][],avail[][],np,nr;

private void input() {
Scanner sc=new Scanner(System.in);
System.out.print("Enter no. of processes and resources : ");
np=sc.nextInt();  //no. of process
nr=sc.nextInt();  //no. of resources
need=new int[np][nr];  //initializing arrays
max=new int[np][nr];
allocate=new int[np][nr];
avail=new int[1][nr];

System.out.println("Enter allocation matrix -->");
for(int i=0;i<np;i++)
for(int j=0;j<nr;j++)
allocate[i][j]=sc.nextInt();  //allocation matrix

System.out.println("Enter max matrix -->");
for(int i=0;i<np;i++)
for(int j=0;j<nr;j++)
max[i][j]=sc.nextInt();  //max matrix

System.out.println("Enter available matrix -->");
for(int j=0;j<nr;j++)
avail[0][j]=sc.nextInt();  //available matrix

sc.close();
}

private int[][] calcneed(){
for(int i=0;i<np;i++)
for(int j=0;j<nr;j++)  //calculating need matrix
need[i][j]=max[i][j]-allocate[i][j];

return need;
}

private boolean check(int i){
//checking if all resources for ith process can be allocated
for(int j=0;j<nr;j++)
if(avail[0][j]<need[i][j])
return false;

return true;
```

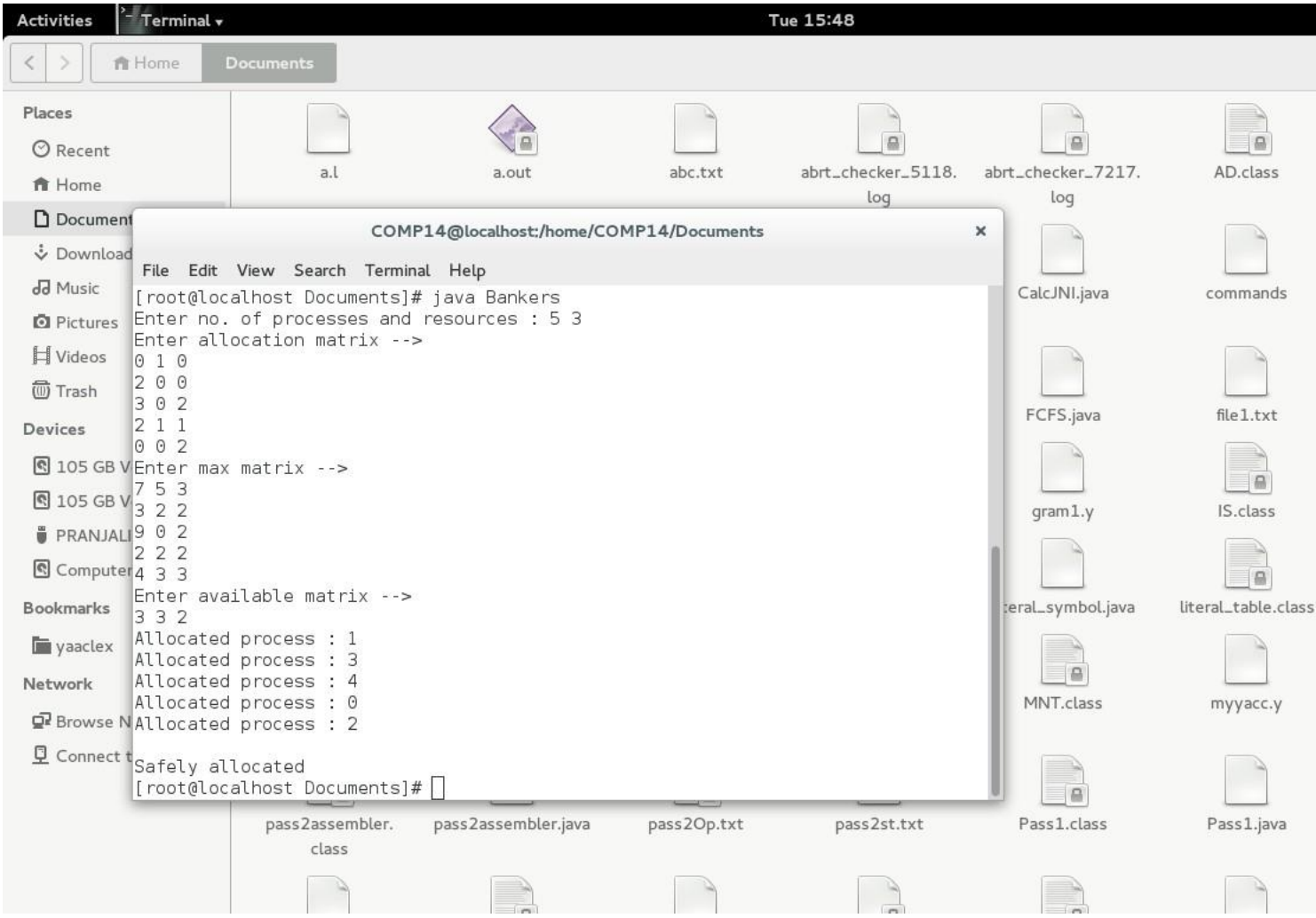
```
}

public void isSafe(){
input();
calcneed();
boolean done[]=new boolean[np];
int j=0;

while(j<np){  //until all process allocated
boolean allocated=false;
for(int i=0;i<np;i++)
if(!done[i] && check(i)){  //trying to allocate
for(intk=0;k<nr;k++)
avail[0][k]=avail[0][k]-need[i][k]+max[i][k];
System.out.println("Allocated process : "+i);
allocated=done[i]=true;
j++;
}
if(!allocated) break;  //if no allocation
}
if(j==np)  //if all processes are allocated
System.out.println("\nSafely allocated");
else
System.out.println("All proceess cant be allocated safely");
}

public static void main(String[] args) {
new Bankers().isSafe();
}
}
```

7 output:



8 Conclusion:

Hence, we implemented the Banker's Algorithm for deadlock Avoidance.