Assignment C-2

TITLE: Implementation of Banker's algorithm.

PROBLEM STATEMENT: Write a Java program to implement Banker's Algorithm

OBJECTIVE:

- To study the algorithm for finding out whether a system is in a safe state.
- To study the resource-request algorithm for deadlock avoidance.
- To study and implement the Banker's algorithm to avoid deadlock.

S/W and H/W

Requirements:

- 64-bit open source Linux (Fedora 20)
- Eclipse IDE, JAVA
- I3 and I5 machines

OUTCOME:

We will be able to

- Implement deadlock avoidance algorithm
- Compute resource allocation sequences which lead to safe state
- Demonstrate the limitations of deadlock avoidance algorithms

Theory:

Deadlock: A set of processes is in a deadlock state when every process in the set is waiting for an event that can only cause by another process in the set. Examples of such processes are resources acquisition and release.

As shown in the diagram process P1 is holding the resource R2 and requesting resource R1.

Process P2 is holding the resource R1 and requesting resource R2. So no process can proceed further, indicating the deadlock.

Four basic conditions for deadlock to happen:

1**.** mutual exclusion: at least one resource must be held in a non-sharable mode.

2. hold and wait: there must be a process holding one resource and waiting for another.

3**.** no pre-emption: resources cannot be pre-empted.

4. circular wait: there must exist a set of processes [p1, p2, …, pn] such that p1 is waiting for p2, p2 for p3, and so on and pn waits for p1….

The approaches used to handle the deadlock are:

Deadlock Avoidance

Deadlock Prevention

Deadlock Detection and Recovery

Banker's algorithm is a deadlock avoidance algorithm. The name was chosen since this algorithm can be used in a banking system to ensure that the bank never allocates its available cash in such a way that it can no longer satisfy further requests for cash.

When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system. When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. The resources are allocated; otherwise, the process must wait until some other process releases enough resources.

Algorithm:

1. If request[i] > need[i] then error (asked for too much)

2. If request[i] > available[i] then wait (can't supply it now)

3. Resources are available to satisfy the request:

Let's *assume* that we satisfy the request. Then we would have:

available = available - request[i]

allocation[i] = allocation [i] + request[i]

need[i] = need [i] - request [i]

Now, check if this would leave us in a safe state; if yes, grant the request, if no, then leave the state as is and cause process to wait.

Steps To Do/algorithm:

- Input need the Claim matrix (C) and Allocation matrix (A) and Resource Vector (R)
- Calculate (C-A) and Available Vector V.
- Test for safety condition of the system.
- Decide on whether the resources have to be allocated or not.

Let Request[i] be the request vector for process P[i]. If Request[i,j] = k, then process P[i] wants k instances of resource type R[j]. When a request for resources is made by process P[i], the following action are taken:

If Request[i] <= Need[i], go to step 2. Otherwise, raise an error condition, since the process has exceeded it's maximum claim.

If Request[i] <= Available, go to step 3. Otherwise, P[i] must wait, since the resources are not available.

Have the system pretend to have allocated the required resources to process P[i] by modifying the state as follows:

Available: = Available - Request[i];

Allocation[i]:= Allocation[i] + Request[i];

Need[i]:= Need[i]- Request[i];

Conclusion: Thus, we successfully studied and implemented Banker's algorithm.