Banker's Algorithm

Li Chengyao
5140219288

May 28, 2017

1 PROBLEM

1.1 DESCRIPTION

Write a problem that realize the resource allocation and deadlock avoidance algorithm Banker's Algorithm. You can see the detail in the end of chapter 7 of *OPERATING SYSTEM CONCEPTS WITH JAVA (Eighth Edition)*, page 344.

2 ALGORITHM

2.1 RESOURCE TYPES

Several data structures must be maintained to implement the banker's algorithm. These data structures encode the state of the resource-allocation system. We need the following data structures, where n is the number of processes in the system and m is the number of resource types:

- **Available**

  A vector of length m indicates the number of available resources of each type. If *Available[j]* equals k, then k instances of resource type R_j are available.

- **Max**

  An $n \times m$ matrix defines the maximum demand of each process. If *Max[i][j]* equals k, then process P_i may request at most k instances of resource type R_j.

- **Allocation**

  An $n \times m$ matrix defines the number of resources of each type currently allocated to each process. If *Allocation[i][j]* equals k, then process P_i is currently allocated k instances of resource type R_j.

- **Need**

  An $n \times m$ matrix indicates the remaining resource need of each process. If *Need[i][j]* equals k, then process P_i may need k more instances of resource type R_j to complete its task. Note that $Need[i][j] = Max[i][j] - Allocation[i][j]$.

2.2 Safety Algorithm

1. Let Work and Finish be vectors of length m and n, respectively. Initialize *Work = Available* and *Finish[i] = false for i = 0, 1, ..., n - 1.*

2. Find an index i such that both

   a) *Finish[i] == false*

   b) $Need_i \leq Work$

   If no such i exists, go to step 4.

3. $Work = Work + Allocation_i$
   Finish[i] = true
   Go to step 2.

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

2.3 Resource-Request Algorithm

Let $Request_i$ be the request vector for process P_i. If $Request_i == k$, 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 actions are taken:

1. If $Request_i \leq Need_i$, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.

2. If $Request_i \leq Available$, go to step 3. Otherwise, P_i must wait, since the resources are not available.

3. Have the system pretend to have allocated the requested resources to process P_i by modifying the state as follows:

$$Available = Available - Request_i$$

$$Allocation_i = Allocation_i + Request_i$$

$$Need_i = Need_i - Request_i$$

   If the resulting resource-allocation state is safe, the transaction is completed, and process P_i is allocated its resources. However, if the new state is unsafe, then P_i must wait for $Request_i$, and the old resource-allocation state is restored.

2.4 Implementation

The program requires a txt file to read in matrix *Max* and the initial *Allocation* matrix is a zero matrix. The user is required to input a formatted txt file and the *Available* vector like

$$java\ TestHarness < input\ file > 10\ 5\ 7$$

to execute the algorithm.
Then the program will ask the user to input a command iteratively. The user can use "status" to check current matrices an "exit" to quit the program. In order to request or release resources the user have to obey the format:

$$[request \mid\mid release]\ < customer\ \# > < resource\ \#1 > < \#2 > < \#3 >$$

And the result will be returned in the command line window.

3 Results

3.1 Environment

- Windows 10
- Java Development Kit 1.8.0_131
- Eclipse

3.2 Screenshots of the Result

We use command line to compile and execute the program. The result is shown in Fig. 3.1.

3.3 Thoughts

The given interface helps me a lot when finishing this project. In this way the code becomes more standardized and neat.

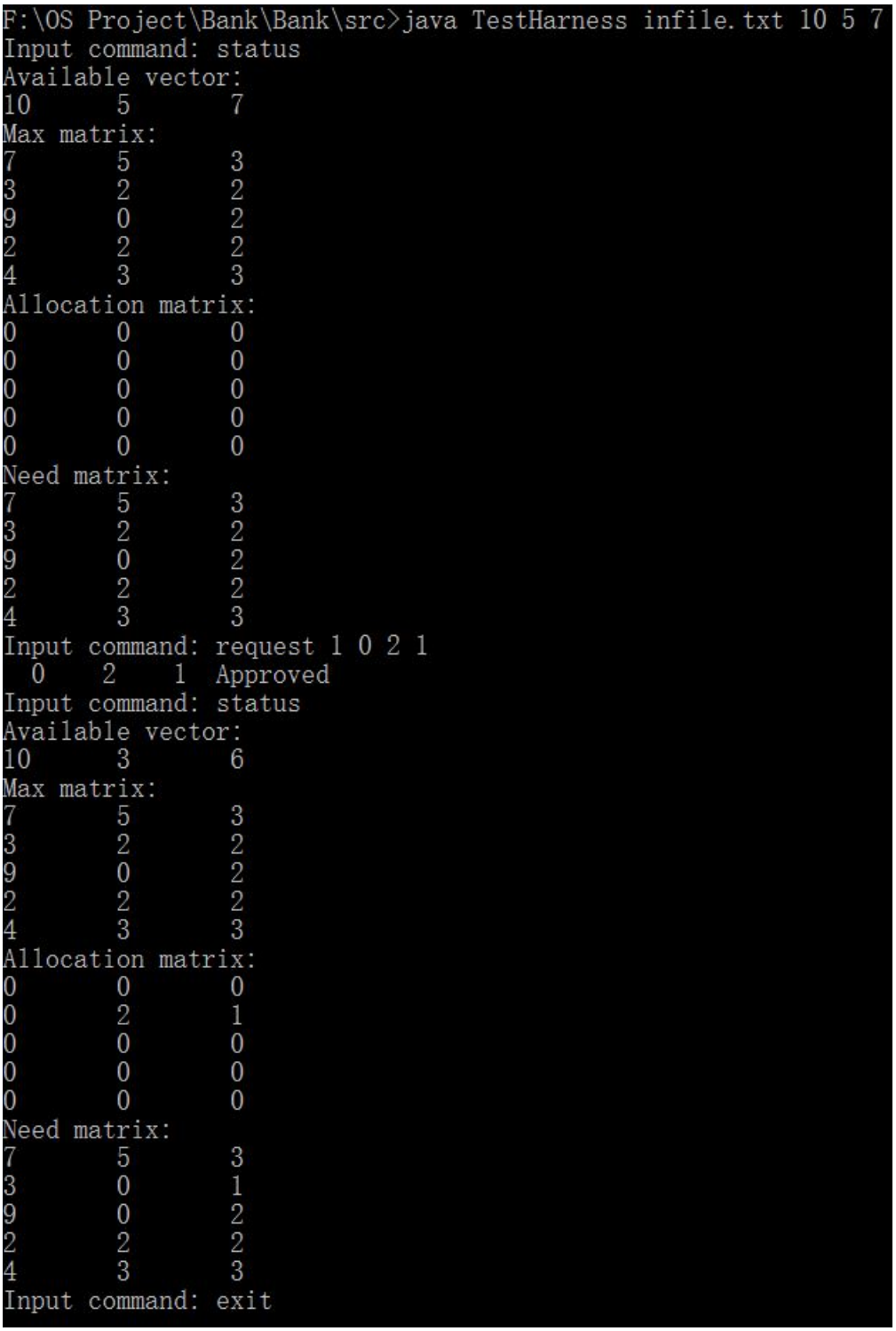

Figure 3.1: Screenshot of Banker's Algorithm