Programming Assignment 1: Distributed RPC-based ATM

Due 2/28/2014

Problem Description

A distributed banking system serves as the backbone of modern financial transactions. In this project, we consider a highly simplified system that consists of a server and some Automated Teller Machines (ATM). The server manages all users' account information. A customer can invoke the following operations at an ATM.

1. void deposit(int acnt, int amt): this operation increases the balance of user account acnt by amt, and returns nothing;
2. void withdraw(int acnt, int amt): this operation decreases the balance of user account acnt by amt, and returns nothing;
3. void transfer(int acnt1, int acnt2, int amt): this operation transfers money with amount of amt from banking account acnt1 to account acnt2;
4. int inquiry(int acnt): this operation returns the balance of user account acnt

For simplicity, in this assignment you do not need to consider the synchronization problem.

Programming Requirement

You are required to write this client-server (banking system) program which communicates via RPC. You are recommended to develop this system using either the Java programming language with JAVA/RMI (recommended), or the C programming language (older, but still feasible). You are free to use other RPC-like languages instead of JAVA or C, as long as your language also has support for RPC. (At the end of the assignment there is a list of JAVA or C RPC resources.)

Specifically, your program should consist of two parts: one for the client and another for the server. The client (as the ATM) will initiate RPC by sending request message to the bank server to execute a specified procedure (e.g. deposit) with parameters. Recall the sequence of events during a RPC:

- Client procedure calls client stub in normal way
- Client stub builds message, calls local OS
- Client's OS sends message to remote OS
- Remote OS gives message to server stub
- Server stub unpacks parameters, calls server
- Server does work, returns result to the stub
- Server stub packs it in message, calls local OS
- Server's OS sends message to client's OS
- Client's OS gives message to client stub
- Client stub unpacks result, returns to client

Design Requirement

Initially, the Bank server should have the following data in its database. You need to store these data into a separate file, and let the server to read and update this file as the user performs actions. You should not hard-code these values into your program.

Account	Balance
1000	1000
1001	2000
1002	3000
1003	4000
1004	5000
1005	6000
1006	7000
1007	8000
1008	9000
1009	10000
1010	0

The bank server has only one input parameter "server_port", which specifies the port of communication. Please choose a random port for your program to avoid conflicts with other existing programs.

Implementation Notes

You are required to use the Hydra lab for your experiments. Please choose port numbers such that your program does not reuse a port that is being used by other groups on the same machine.

Your server program only takes one parameter as the port number. Your client program, however, will take the server:port combination as the first parameter, the operation type (deposit, etc) as the second parameter, the account as the third parameter, and the amount as the fourth parameter. The second to fourth parameters can also be replaced by a single parameter as the input file for batching processing, where the file contains the second, third, and fourth parameters in lines. Each line is one operation.

Demo
We now show how the system is expected to work.

First, start the server on hydra2
hydra2> java bankserver 6666
Next, test the server on a different machine hydra3
hydra3> java bankclient hydra2.eecs.utk.edu:6666 inquiry 1000
The current balance of user 1000 is $1000
hydra3>java bankclient hydra2.eecs.utk.edu:6666 deposit 1000 200
Successfully deposit $200 to account 1000!

Hydra3>java bankclient hydra2.eecs.utk.edu:6666 withdraw 1001 50
Successfully withdraw $50 from account 100!
the C programming language or Hydra3>java bankclient hydra2.eecs.utk.edu:6666 withdraw 1012 50
No such user or account, 1012!

As demonstrated your program should be able to generate correct messages when errors occur. For our testing purpose, your client program should also support taking a script of commands and then process them one by one. For example, the format for the client when taking files as input is like this:

Hydra3>java bankclient hydra2.eecs.utk.edu:6666 commands

Here, commands is a file located in the same directory. The bankclient will then execute the commands line by line, and generate the output on the screen

Submission

Your program should be submitted electronically via the Blackboard system. You should submit the following files.

- All **source codes**.
- **Make files** that compile your programs.
- A **Readme** file, which briefly describes all submitted files. In the Readme file, you should also provide information about compiling environment, compiling steps, execution instructions etc. You are encouraged to provide as much detailed information as you think that might help TA run your code.
- A **Sample file**, which describes the tests you have run on your program.
- A **Design file**, which describes the overall program design, a verbal
- description of "how it works" including the basics of what the system is doing underneath, and design trade-offs considered and made. Also describe possible improvements and extensions to your programs (and sketch how they might be made).

Grading

Grading will be based on the following elements:

Have you submitted all required files? (10%)
Can we compile the code successfully? (10%)
Does your system pass our testing scenarios? We will test thoroughly various deposit, inquiry, withdraw and balance transfer cases. We will also test strange scenarios such as when the user account does not exist (60%)
Thoroughness of your own test cases (10%)
Design Document (10%)

START EARLY. THIS IS HARDER THAN IT LOOKS!

Online resources:

JAVA RMI Programming:
http://docs.oracle.com/javase/tutorial/rmi/
http://soft.vub.ac.be/~tvcutsem/distsys/javarmi.pdf
http://www.eg.bucknell.edu/~cs379/DistributedSystems/rmi_tut.html

C RPC Programming:

http://www.cs.cf.ac.uk/Dave/C/node33.html
http://www.linuxjournal.com/article/2204?page=0,0
http://docs.freebsd.org/44doc/psd/23.rpc/paper.pdf