

200 points. Individual Work Only. Due September 18, 2007 before class.

1. Objective:

Design and implement a multi-threaded client-server program in Java using sockets for communication between the client and server programs.

2. Description:

The server program maintains a set of inventory records that will be queried by the clients and the server updates these records based on client requests. In addition to the inventory records the server also maintains a set of user records to keep track of which user has checked-out the inventory. The aim of this project is to implement a multithreaded server that will handle multiple client requests and manage the updates to inventory and user records.

Assume that the server program first reads the inventory and user records from a file and creates a collection of inventory and user objects. The inventory record format, user record format, and sample input files for inventory and user records are provided below.

Inventory Record Format:

ID	Integer
Title	String
Category	String
Format	String
Year	Integer
Count	Integer

User Record Format:

UserId	String
Password	String
Count	Integer
Item1	Integer
Item2	Integer
Item3	Integer

Sample Inventory Records:

1001 "We are Marshall" "Drama" "Widescreen" 2006 0
1002 "Blood Diamond" "Action and Adventure" "Widescreen" 2006 4
1003 "Must Love Dogs" "Comedy" "Widescreen" 2005 1
1004 "Super Size Me" "Documentary" "Widescreen" 2004 5
1005 "Along Came Polly" "Romance" "Widescreen" 2004 8
1006 "The Station Agent" "Independent" "Widescreen" 2003 6
1007 "Minority Report" "Sci-Fi" "Widescreen" 2002 3
1008 "Shrek" "Children and Family" "Full Screen" 2001 1
1009 "Shrek" "Children and Family" "Widescreen" 2001 2
1010 "Lawrence of Arabia" "Classics" "Widescreen" 1962 10

Sample User Records:

john johnsmith 3 1001 1003 1005
joe joesmith 0 0 0 0
jane janesmith 2 1001 1002 0
alice alice 1 1009 0 0
bob bob 2 1004 1007 0

After creating the collection of inventory and user objects, the server program performs the following operations:

1. Opens a socket connection on the specified port (port # is provided as a command-line argument to the server program)
2. Listen for request from clients

3. When a client program sends a connect request, checks if the *Userld* provided by the client exists in the user collection, if it exists then assigns a new port for the client to communicate, creates a new thread that will handle all the client requests and performs the appropriate action based on the request (this thread will terminate when the client program sends a disconnect request). If the *Userld* provided by the client does not exist then the server returns an error message to the client.
4. Go to step 2

The client program takes the hostname and port # on which the server process is listening as command-line arguments and performs the following operations:

1. Open a socket connection to connect with the server process (the first client request will be CONNECT with the *Userld*)
2. If the connection is successful, the server returns on new port # for the client to connect for all further communications. Close the first connection and open connection to the new port provided by the server.
3. Provide a command prompt for user for enter commands and wait for the user to enter commands
4. Process the commands entered by the user
5. Send the appropriate command/request to the server process
6. Wait for the server to perform the appropriate action and respond to the client request
7. Process server response if necessary
8. Go to step 3, if the input is "QUIT" then send the request disconnect to the server and exit.

The various commands that the client and server can process along with the action performed at the server is given below:

1. CONNECT – Establish connection with the server (client program provides the server with a *Userld* so that the server knows which user record to update), server assigns a new port for the client, creates a new thread, and returns the new port # for the client to connect (if there is any error with the *Userld* provided by the client, assigning a new port, or creating a new thread, the server should return a negative number – use a different number for each error condition)
2. LIST – Send the list of all inventory records available one record per line (similar to the input records file shown above)
3. INFO *ID* – Send the details about the specific inventory record in the format shown above
4. SEARCH *SearchString* – Send the list of inventory records that match the search string in the title field of all records **[Optional for Undergraduate students]**
5. CHECKOUT *ID* – Check if the user has already checked out three items, if not update the corresponding count fields for the inventory record and the user record and send a SUCCESS message to the client (if the inventory count is 0 or user checkout item count is three the server should send the appropriate error message to the client)
6. RETURN *ID* – Update the corresponding inventory and user record count (in case of error send an appropriate error message) and send the list of all available records
7. QUIT – client program exists after closing any open connections and the corresponding server thread is also terminated (this is the thread created by the server for this client).

Use the sample input files provided to test your program. Each user will be assigned specific hostname and port range to start the server process and establish client connections. The server process must be started before starting the client process. For initial testing and debugging purposes you can start the server and client process on the same machine after that you can start the client processes on different machines.

3. Guidelines/Hints:

First follow the Java Tutorials for sockets and threads (see Resources Section 10) and learn the Java APIs for sockets and threads. Initially, implement the client and server program without any socket code. Use Java packages to separate the client and server programs (e.g., cs631.hw1.server and cs631.hw1.client). Make sure you can parse all the input commands correctly on the client side, and then test the basic functionality on the server side. Then write the sockets code and test it first without threads. Finally, add threads and synchronization to complete the program (you have to design your program from the start with multi-threading in mind, if you try to add threads at the end you will end up modifying your code). Make sure that your program is modular and well designed, you will use this program throughout the semester.

4. Graduate Students Only (Bonus for Undergraduates):

The SEARCH protocol described above is optional for undergraduate students, but required for graduate students.

5. Working Environment:

You can develop and test the client and server programs on any machine that you have access to, but the final demonstration of the project must be done in the CIS Undergraduate Lab (CH 154). Hence you have to test the completed project on the CIS machines vulcan[1-8].cis.uab.edu (Linux workstations in the undergraduate lab) before submission. There is no need to be physically present in the lab to use these machines, you can connect to these machines remotely using an SSH Client. For more details on how to connect to these machines using SSH check out sample screen shots and a Flash demo provided at: http://www.cis.uab.edu/cs333/CIS_UNIX_howto.html.

You need to add the following lines to the end of .bash_profile file to have the correct Java environment on the vulcans:

```
export JAVA_HOME=/netbin/java
export PATH=${JAVA_HOME}/bin:${PATH}
```

After adding the above two lines, logout and login again. If you type “which java” you should see: /netbin/java/bin/java.

6. Short Answer Questions:

1. If student A has the server program running on machine P, could student B use his client program running on machine Q to connect to A’s server program if student B knows the port on which student A’s server is running?
2. If your answer to question (1) is yes, then explain how to prevent this from happening.
3. If your answer to question (1) is no, then explain why this is not possible.

7. Feedback Questions (answer to these questions has no impact on your grade):

Was this homework too difficult, or too easy?

Was the assignment fun or challenging?

Was there something that was unclear?

Was the homework too long for the given amount of time?

What did you learn from this homework?

8. Submission Instructions:

List ALL the references you used in this homework as well as test cases used to test your programs. This includes any classes that you used that you did not write and any help you received from any other sources. Use appropriate class name and include comments to indicate

various operations performed by the program. Your program must have the following header information within comments:

```
/*  
    Name:  
    BlazerId:  
    Homework #:
```

```
*/
```

Make a directory CS631/homework1 and then two separate directories one for the server (server) and one for the client program (client). Create a tar/zip file (only tar and zip are accepted, all other file formats will not be considered) of the homework1 directory using the following filename format: blazerId-cs431-hw1.tar or blazerId-cs431-hw1.zip (graduate students use 631 or 731 instead of 431). The tar/zip file must include the source code, txt/word/pdf/ps document for the typed solution to short answer questions and the feedback questions, instructions for executing the programs. Login to WebCT and go to the assignments section and upload the tar/zip file. There is no need to turn in any printed solutions for this homework, WebCT submission is sufficient.

9. Late Submissions:

Submissions must be made on the due date before the beginning of the class. Late submissions will lose 10% for every 24-hour period, up to a maximum of 50% (weekends and holidays count as one 24-hour period). Any submissions made after one-week will receive a score of 0 for this homework.

10. Resources:

1. Setting started with Java: Problem 2 in CS 201 Lab:
<http://www.cis.uab.edu/cs201/oldfiles/spring2004/labs/lab10/lab10.html>.
2. For Packages: Java Tutorials: Creating and Using Packages:
<http://java.sun.com/docs/books/tutorial/java/package/packages.html>.
3. For Sockets: Java Tutorials: All About Sockets:
<http://java.sun.com/docs/books/tutorial/networking/sockets/index.html>.
4. For Threads: Java Tutorials: Concurrency:
<http://java.sun.com/docs/books/tutorial/essential/concurrency/>.
5. General Java Help: Java Almanac: <http://java.sun.com/developer/codesamples/examplets/>.