GEORGE MASON UNIVERSITY

Department of Computer Science

CS 471 - Operating Systems, Fall 2013 Project #3
Due: Tuesday, December 3, 11:59 PM

DISTRIBUTED AUCTION SERVICE

1 Overview

The objective of this assignment is to provide some hands-on experience on **client-server programming using sockets and TCP/IP protocol suite in a distributed environment**. We consider a simple distributed application consisting of an *Auction Server (AS)*, a seller and multiple buyer processes. The seller and buyers will connect to Auction Server to perform transactions on individual items. You will write the auction server and client (seller and buyer) programs using TCP as the transport layer protocol. Below we provide the specifications and details about the project as well as some suggestions. **Please read all the parts of this hand-out carefully before starting to work on your project**.

2 Specification

You can assume that only one seller program is active in the system. Through the seller program, the merchant will connect to AS, offer items for sale and monitor existing bids. At any time, the merchant can decide to sell a given item to the *highest bidder*; at this point the auction for that item will be closed. The buyers will connect to AS by invoking the buyer program. By doing so, they will able to get the listing of items and place bids. Multiple buyer programs can be simultanously active at a given time.

Associated with each item offered by the seller, there will be a unique *item number* (an integer) and an *item name* (a descriptive string). For each item, AS will keep track of the current highest bid and highest bidder.

Auction Server (AS): There will be only one auction server process in the system. Its IP address (AUCTIONSERVER-IP-ADDRESS), and the ports at which it will receive connection requests of the seller and the buyers (SELLER-PORT and BUYER-PORT, respectively) will be known to all the processes in the system. Upon the receipt of specific commands, AS must take the appropriate actions:

- Login <UserName>: May be issued by a seller or buyer. AS will update its records to indicate that the seller or a new buyer program is connecting. Depending on your design, it may need to record the connection data for the involved client, such as the user name, IP address and Socket/Port information.
- List: May be issued by a seller or buyer. AS will send the listings of all the items currently on auction. For each item, the item number, item name, highest bid and highest bidder should be displayed on the console of the client process (seller or buyer).

- Bid <Item Number> <Bid Amount>: May be issued by a buyer program. If this is the first bid on the item, then the buyer sending the command becomes automatically the highest bidder, and its bid becomes the highest bid. Otherwise, the buyer will become the highest bidder only if <Bid Amount> is greater than the existing highest bid on that item. In this case, AS will update its highest bid and highest bidder records, and an informative message must be sent to both the new highest bidder ("You have now the highest bid on item X") and the old one ("You have been outbid on item X").

- Add <Item Number> <Item Name>: May be issued by the seller program. A new item will be added to the auction list with the given number and name.

- Sell <Item Number>: May be issued by the seller program. The auction on the item with the given number will be closed upon the receipt of the sell command. An informative message must be sent to the current highest bidder on that item ("You have won the auction on item X".) Other bidders/buyers should not receive this message.

Server program: AS should react reasonably to unexpected message sequences. For example, unrecognized commands and commands involving unknown (or already sold) items should be rejected. After processing each client request, the server must return an informative reply which must be also displayed on the client's console (e.g. "The new item has been added to the auction list", "Your bid is not high enough", "Unrecognized command", etc.) For debugging purposes, AS should display an informative message in its own console/window as well, after processing each request.

AS cannot have any prior information about the port numbers of the clients – for example the port numbers of the potential client processes cannot be hard coded in the AS source program; nor can this information be read from a file or keyboard. On the other hand, the IP address and the port number of the server can be provided to the clients before execution. Note that all buyer processes need to initialize conection at the same (unique) port of the server, whose number is given by the BUYER-PORT constant. On the other hand, the seller process will connect to another port of the server (given by the SELLER-PORT constant).

AS should be able to deal with multiple clients concurrently – it is unrealistic to assume that a new client has to wait until the previous client has issued the Exit command. This is an important requirement of this project. In order to have a *concurrent* server design, you can consider creating new (child) processes or threads on the server side.

The following user names can be assumed to be pre-registered with AS: Seller, Alice, Bob, Dave, Pam, Susan, Tom. There is no need to use passwords after the Login command and the connection requests from users other than these seven can be rejected for simplicity (You are free to add more user names if you wish). The username *Seller* is reserved for the seller program. All commands given above should be implemented in case-insensitive manner.

Client Programs: Each client (seller or buyer) program will wait for input from the user, forward the request to AS, and display incoming messages from AS.

Each client (seller/buyer) process and AS can run in a different window: when working on mason or zeus remotely, you can open multiple *ssh* sessions and activate a client process or AS in a different connection window.

You can assume that a given user does not login through multiple clients at the same time. Similarly, you can assume the auction server is launched first and that a client program needs to be re-started once the corresponding user has exited. <Item Number> and <Bid Amount> can be represented by positive integers. <Item name> can be a string of at most 32 characters. In addition to alphanumeric characters, item names may also include whitespace character(s).

Provided that you comply with the rules above, the details of the interface between the client programs and the users are left unspecified: you can use a simple text-based interface, but it should be efficient and easy to use. Similarly, the format of socket messages between clients and the server is left to you; yet, you should be able to justify your design decisions. You should identify any potential race conditions on shared variables (if any), and take the preventive measures.

3 Extra Credits

If, like in any realistic client software, your system is able to display the incoming messages on the client side without any delay (i.e., as soon as they are received), you can get up to 25 bonus points. In other words, it is possible to get 125/100 in this project. On the other hand, if your implementation first waits for the input from the user on the client side, and only then it checks and prints any incoming messages from the server, then you will not be eligible for the bonus points.

An example to clarify this point is the following: Suppose a user (say, Bob) places a bid on a certain item. The client software on Bob's side will be naturally waiting for Bob's next command. If, another user (say, Tom) places a higher bid on the same item, the server is supposed to send a "You have been outbid" message to Bob. To receive the bonus points, that message should appear on Bob's console even if Bob doesn't enter a new command; waiting until Bob enters a new command and only then printing pending messages from the server is obviously not the best way to serve the client.

However, we recommend that you first implement and fully debug your system without this feature, and think about this extension if you have additional time. (*Hints: There are multiple ways of achieving that functionality. You can, for example, consider creating child processes or new threads).*

4 Operational Details

In this project, you can work alone or with another CS 471 student. No project team can have more than two members. Both members of a team will get the same project grade and there will not be bonus or penalty points for those working alone.

For this assignment, you can use C or Java as the programming language. Your project must be developed on (or ported to) a Unix/Linux system. Your program *must* compile and run on either mason.gmu.edu or zeus.vse.gmu.edu – it will be tested and graded on these systems. Berkeley Sockets and Java Sockets are also available on VSE lab machines. **Do not use socket multicast/broadcast**. The use of middleware solutions (e.g., RPC or RMI) is not allowed in this assignment. Also note: your socket programs must use TCP as the transport layer protocol; the use of UDP is not allowed in this project.

In this assignment, you will create a unique process to represent the auction server and each of the clients (sellers and buyers), on the **same** computer. During testing, we will invoke the server and each client program in its own, separate window. You should be careful when choosing port numbers for your server. All port numbers below 1024 are reserved and the ports used by other services/students may not be available either. Make sure to 'close' every socket you use in your programs. If you abort a program, the socket may still hang around and the next time you bind a new socket to the same port you previously used (but never closed), you may get an error. While managing sockets, you can assume that server/clients never crash(es) during execution.

GMU Honor Code will be enforced by comparing the source codes, possibly by automated mechanisms. You are not allowed to co-operate with any other person except your project partner (if you have one).

5 Running Your Programs with a Specific Command Sequence

In this assignment, you have to run your programs with the following command sequence. In order to be able to follow the interactions, please you should wait for 4-5 seconds after each initialization for socket link establishment phase. Each server, buyer and seller should be invoked in a separate window, so that their interactions and message traffic can be monitored in real-time.

1. Initialize Auction Server
2. Initialize Seller
3. Initialize Buyer 1
4. Initialize Buyer 2
5. Seller sends the command - login Seller
6. Seller sends the command - add 1 OS Book
7. Seller sends the command - add 2 Math Book
8. Seller sends the command - add 3 Bach CD
9. Seller sends the command - list
10. Buyer 1 sends the command - login mike
11. Buyer 2 sends the command - login bob
12. Buyer 1 sends the command - list
13. Buyer 1 sends the command - bid 1 4
14. Buyer 2 sends the command - list
15. Buyer 2 sends the command - bid 2 3
16. Buyer 1 sends the command - bid 2 2
17. Buyer 2 sends the command - bid 1 7
18. Initialize Buyer 3
19. Buyer 3 sends the command - login susan
20. Seller sends the command - add 4 Chess Set
21. Buyer 3 sends the command - list
22. Buyer 3 sends the command - bid 5 9
23. Buyer 3 sends the command - bid 4 9
24. Buyer 1 sends the command - bid 1 10
25. Buyer 3 sends the command - bid 1 9
26. Buyer 3 sends the command - bid 3 20
27. Seller sends the command - list
28. Seller sends the command - sell 3
29. Buyer 2 sends the command - bid 1 9
30. Buyer 1 sends the command - bid 3 30
31. Buyer 1 sends the command - list
32. Seller sends the command - sell 4
33. Seller sends the command - sell 1
34. Seller sends the command - sell 2
35. Seller sends the command - list
36. Buyer 1 sends the command - list

Your project will be tested first with the above command sequence and after that, other sequences will be tried. Consequently, you can consider the above sequence as an initial test.

6 Submission Details

Submissions will have both soft and hard copy components.

The soft copy components will be submitted through *Blackboard*. You will need to submit all the softcopy components (listed below) as a single compressed *zip* or *tar* file. Submit your compressed file through the Blackboard by Tuesday, December 3, 11:59 PM. Before submitting your compressed file through the Blackboard, you should rename it to reflect the project number and your name(s). For example, if Mike Smith and Amy White are jointly submitting a tar file, their submission file should be named as: project3-m-smith-a-white.tar. One group member can make the submission for the entire group. On Blackboard, select the Projects link, then the Project 3 folder. In addition to the assignment specification file, you will see a link to submit your compressed tar or zip file. *Please do not use Windows-based compression programs such as WinZip or WinRar* – use tools such as zip, gzip, tar or compress (on zeus or mason).

The soft copy components that need to be submitted in a single *tar* or *zip* file are as follows:

- a README file with:
  - information on whether you prefer that your program be tested on *mason* or *zeus*
  - clear instructions on how to compile and run your program
  - a statement summarizing the known problems in your implementation (if any).
- the source codes of your programs (TCP auction server, TCP buyer and seller programs –that makes three programs). Your source files should include in the first commented lines your name(s) and G number(s).
- a write-up explaining the high-level design of your programs (Also, list the main modules of your program and their functions).
- the output of four TCP clients for the message sequence above.

You should also submit the hard copy of all the components above to the instructor at the beginning of the class meeting on Wednesday, December 4, 1:30 PM. Failure to follow the submission guidelines may result in penalties. You can re-submit your programs through *Blackboard* as many times as you like (as long as each submission has all the required components); we will grade only your latest submission.

Late penalty for Project 3 will be 15% per day. Submissions that are late by five days (or more) will not be accepted.

7 Suggestions

- First of all, allow sufficient time for your project unless you have prior experience in socket programming. As the first step, you will have to study the basics of socket communication by reading Web tutorials and examining/writing simple client-server codes. Then you can incrementally develop your system, by adding multiple clients, enhancing the server, designing the user interface module for the clients, and so on.
- Choose your partner wisely. Efficient cooperation with your partner can provide a better project with less effort only if you are well-coordinated. If you have a partner go over the project specification with him/her as soon as possible and decide on the individual responsibilities.

- You are strongly encouraged to refer to Network and Socket programming links available through the CS 471 Blackboard site (*Resources* item on the left menu), as well as Unix man pages when applicable. You can direct your programming and system questions to CS 471 TA (*aroy6@gmu.edu*). However, before contacting TA, please make every effort to check thorougly the web tutorials. The CS 471 TA prepared also a web page with sample socket programs that compile and run on GMU systems; that can page can be accessed at `http://mason.gmu.edu/~aroy6/cs471f13.html`. Conceptual questions about the project should be directed to the instructor (*aydin@cs.gmu.edu*). Please add the prefix **CS 471: Project 3** to the subject line in any correspondence with the TA or the instructor.