DISTRIBUTED AUCTION SERVICE

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. **In two different versions**, you will implement the system using first TCP, and then using UDP as the transport layer protocol.

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 BUYER-PORT constant. Similarly, the seller process will connect to another port of the server (given by 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, Amy, Bob, Dave, Pam, 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. **To get full credit, the incoming messages should not be delayed while waiting for input from the user at the client side; they must be displayed as soon as they are received.**

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.

Programming Language/Environment: For this assignment, you can use C, 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.ite.gmu.edu – it will be tested and graded on these systems. Berkeley Sockets and Java Sockets are also available on IT&E lab machines. **Do not use socket multicast/broadcast**. The use of middleware solutions (e.g. RPC or RMI) is not allowed in this assignment.

Run-time Details: 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.

Submission: Submissions will have both soft and hard copy components. First, you have to send a single e-mail to *aydin@cs.gmu.edu* with cc: to CS 571 TA *cliu6@gmu.edu* by November 3, 4:30 PM, with the following components in the attachment.

a. The source codes of your programs (the auction server, seller and buyer programs for both UDP and TCP –that makes a total of six programs). Please compress your files before submission; the name of the compressed file should be identical to your last name (e.g. smith.zip). *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).
b. a write-up describing/explaining high-level design of your programs (which should also include comments on how you achieved the *concurrent server design*).
c. A README file containing clear instructions on how to configure (e.g. change the server port number), compile and run your programs.

Please do not send multiple e-mails corresponding to different protocols or clients/servers; all the components above should be included as attachment(s) of a single e-mail. The subject of your submission e-mail should read Submission: CS 571 Assignment 2. In the body of your e-mail, include your full name, GMU student id, and the name of the GMU server your programs compile and run on (i.e. mason.gmu.edu or zeus.ite.gmu.edu). You should also include a statement summarizing the known problems in your implementation (if any).

You will receive an acknowledgement when your soft copy submission is received by us. The items (a) and (b) above (namely, the source code and the write-up) should be also received by the instructor as hard copy by November 3, 4:30 PM.

Resources: You are encouraged to refer to Network programming links available at `http://www.cs.gmu.edu/~aydin/cs571/resources.html` as well as Unix man pages when applicable. Please direct your programming and system questions to CS 571 TA Changwei Liu (*cliu6@gmu.edu*), using the keyword "CS 571 Assignment 2" on the subject line. The CS 571 TA prepared also a web page with sample socket programs that compile and run on GMU systems; this can page can be accessed at `http://mason.gmu.edu/~cliu6/cs571/cs571.html`. However, before contacting TA, make every effort to check thoroughly the web tutorials and resources provided at the address above. CS 571 TA holds regular office hours on Monday 5:00-6:30 PM and Wednesday 5:00-6:30 PM in Engineering Building Rm. 4456. Conceptual questions about the project can be directed to the instructor (*aydin@cs.gmu.edu*).

Suggestions: 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. The TCP and UDP components will have equal weight in grading.

Late and Incomplete Submission Penalties: Both the soft and hard copy components, with material for both TCP and UDP, should be submitted in a timely manner to avoid the late penalty (i.e. on or before 4:30 PM on November 3). Late penalty for Assignment 2 will be 15% for each day. Submissions that are late by five days or more will not be accepted. Please plan in advance.