UdpSobs Manual

General function: UdpSobs implements a simple online bidding system using the UDP protocol. It can run on either Server mode or Client mode. The users of this system include sellers and customers. Sellers can post item entries on the Server and all customers can view the item list. If a customer is interested in one of the items, he can directly contact the seller to request more information or to place a bid. Sellers and customers all act as clients, and the server keeps track of all information such as client information, item list and off-line messages. Clients should also be able to send messages to each other, so that a customer can contact a seller directly.

Functionalities:

1. Get start

Start a Server: *$ sobs -s <server-port>*

This command is used to start a server, which listens on port <server-port>. The port number is an integer value in the range 1024-65535.

Start a Client: *$ sobs -c <server-ip> <server-port>*

-c indicates to run as a client.

<server-ip> and <server-port> are the server information. <server-port> should be the same number as is passed in when starting the server.

Test:

Server	Client

$ sobs -s 60000
 	
Socket 60000 has been created.
	$ sobs -c 255.257.0.0 60000
	
	Invalid IPAddress.
	
	$ sobs -c localhost 90000
	
	Port number is out of range(1024 ~ 65535)

	$ sobs -c localhost 60000 sobs>

2. Sign in & sign out

A client should send a command to sign in before it can use the system.

sobs> register <user-name>

<user-name> is the ID that represents a particular seller or customer. When a client signs in, it should send all the client information to the server, including <user-name>, the IP address and port number of the client program. The server should maintain a list of client information with <user-name> as the entry ID.

When the local table has been successfully updated, the client should display the message: sobs> [Client table updated.]

If successfully log in, your client program should display the message:

sobs> [Welcome <user-name>, you have successfully signed in.]

To sign out, the client should support the command:

sobs> deregister

After successfully sign out, the screen will displays:

sobs> [You have successfully signed out. Bye!]

You cannot send any packet except sign in command to the server if you have signed out.

After receiving the client information table broadcasted, clients need to update their corresponding local table.

And when the local table has been successfully updated, the client should display the message: *sobs> [Client table updated.]*

Test:

Client
sobs>register user1

Welcome user1, you have successfully signed in.

Client table updated.

sobs>deregister

You have successfully signed out. Bye!.

sobs>deregister

You have not registered.

sobs>other command

You have not registered.

3. Sell an item

The client has to provide the server the name of the item for sale, how many transactions to allow, what price (in cents) to start the bidding at, and for what price (in cents) to sell the item immediately.

sobs> sell <item-name> <transaction-limit> <starting-bid> <buy-now> <description>

This command requires three arguments described as follows:

<item-name> is the name of the item for sale. It should be a string with no whitespace.

<transaction-limit> is the number of times the item may be bid on before the server should sell the item to the highest bidder. It should be a positive integer value. For example if it is set to 2, then the first bid raises the current bid by the specified amount. When a user completes the second bid on the item the server will sell that item to him/her.

<starting-bid> is the initial bidding price in an integer number of cents that the item should be initially listed at.

<buy-now> is the price at which another client could immediately buy the item. It's an integer number of cents.

<description> is the detailed description of the item. It is the last argument and could contain long sentences with whitespaces.

After sending the message to the server, the client can expect a response from the server as described below in the Server section. The client should display the response:

sobs> <item-name> added with number <item-num>

Test:

Successful sale

Client
sobs>sell item1 1 2 14 50 "the first item to be sold" item1 added with number 1

Error argument

Client	
sobs>sell item 0 5 8 desc Error:arguments	Transaction limit <=0
sobs>sell item 2 -1 8 desc Error:arguments	Bid <=0
sobs>sell item 2 10 0 desc Error:arguments	Buy-now price <=0
sobs>sell item 2 10 9 desc Error:arguments	Buy-now price <= bid

4. Get Items For Sale

Clients can send requests to the server for updated information of all on-sale items:

sobs> info [item-num]

This command has only one optional argument, which is the number of the item that the client wants information about.

If there is no [item-num], information of all items on the server will be returned. A customer can see the whole list so as to choose interesting items among them.

If the client sent the info command with no extra argument the server will reply with a listing of all items currently listed for sale. The response message should look like as follows:

<item-num> <item-name> <seller-name> <current-bid> <buy-now> <description>

<item-num> <item-name> <seller-name> <current-bid> <buy-now> <description>

...

<seller-name> which is the <user-name> of the client who is selling it, and the current bid (in cents), the buy-now price (in cents), and lastly the description of the item.

If NO items yet listed it will respond as follows:

Error: empty

If the client sent the info [item-num] command with an item's number and the item EXISTS in the server's table, then the server should respond with one line:

<item-num> <item-name> <seller-name> <current-bid> <buy-now> <description>

If the client sent the info [item-num] command with an item's number and the item DOES NOT EXIST in the server's table, and then the server should respond with one line:

Error: <item-num> not found

Test:

user1
sobs>info 2 item1 user1 2 5 first item 3 item2 user1 18 30 "test item 2 sobs>info 1 Error: 1 not found sobs>info 2 2 item1 user1 2 5 first item sobs>info 3 3 item2 user1 18 30 "test item 2"

5. Place a Bid

Clients can list items to be sold with the server. The client needs to indicate to the server which item it would like to bid on as well as how much it would like to bid. Clients will specify their bids as an integer number of cents that they would like to pay ABOVE the current bid price. The command should look as follows:

sobs> bid <item-num> <amount>

An example bid command could read as $ bid 5 100 and this would mean that the client would like to bid 100 cents over the current bid for the item number 5. So if the item's current bid price was 500(five dollars), its new bid price after this message should be 600 (six dollars).

If the client is already the highest bidder on the item, he/she is not allowed to bid more than once in a row. The server should respond to a client who is doing this with:

Error: duplicate bid

If the client has bid on an item that he/she has put up for sale the server should not count the bid and reply with the message:

Error: owner

If the item exists and the client has correctly specified the bid with a higher price, the server should add the bid to the current bid price and also store the username of the client with that item as the current highest bidder. It will then respond to the client with one line:

<item-num> <item-name> <new-current-bid>

It the item the client specified does not exist the server should respond with the line:

Error: <item-num> not found

If the bid price that the client is trying to place is negative, the server should not change the current bid and should reply:

Error: negative bid

If the server finds that there is anything wrong with the command arguments, it should not change anything in its table and respond to the client with:

Error: arguments

After a bid is successfully placed, if the number has reached the transaction limit, the server should regard it as the last bid for that item, and the customer that places the last bid should win. The server then sends messages to both the seller and the customer who wins:

To the seller:

sold <item-num> <item-name> <purchase-price>

To the customer who wins:

purchased <item-num> <item-name> <purchase-price>

If the server times out when sending messages to clients, it needs to store the message in a list of off-line messages and send to the client next time it signs in.

Test:

Successful bid

user1	user2
sobs>info 4 4 item2 user1 5 30 "test item 2" sobs>bid 4 1 Error: owner sobs> sold 4 item2 6	 sobs>bid 4 1 purchased 4 item2 6

Duplicate bid & item not found & negative bid

user1	user2
Sobs>sell item1 2 4 5 test item	

item1 added with number 1 sobs>	
	sobs>info 1 item1 user1 4 5 test item sobs>bid 1 3 1 item1 7 sobs>bid 1 4 Error: duplicate bid sobs>bid 2 5 Error: 2 not found
sobs>sell item2 4 7 9 desc item2 added with number 2	
	sobs>bid 2 -2 Error: negative bid

6. Direct Buy & Offline Buy

As the seller has provided the buy-now price, a customer can send a direct-buy command to a seller to request an item. The command should be sent directly to the seller's client program, without notifying the server. The seller's client will then send a direct-sell command to the server and the server notifies the customer that it has successfully bought the item.

A customer can directly send the buy command to seller, which is another client:

sobs> buy <seller-name> <item-num>

When a seller's client program receives the direct-buy commands, it should display:

sobs> [<customer-name> wants to buy your <item-num>.]

Then the customer's client program receives the message:

sobs> [<seller-name> has received your request.]

The seller's client program can then automatically send a command to the server indicating that the item will be directly sold to that customer:

sobs> direct <item-num> <customer-name>

After receiving the ACK, the client displays:

sobs> [<server-response>]

If the <seller-name> is not found in the client information list or the direct-buy command times out 5 times, the customer should assume that the seller is currently offline and send another command directly to the server instead. It will receive:

sobs> [<seller-name> is currently offline. Request forwarded to the server.]

Server:

After receiving the direct-sell command, the server should delete the item from the item list and send the success message to the corresponding customer:

purchased <item-num> <item-name> <purchase-price>

ACK should be sent back to the seller:

sold <item-num> <item-name> <purchase-price>

If the item is not found the server item list, the server should respond to the seller with an error:

Error: <item-num> not found

If the server fails to contact the customer after 5 tries, it deletes the client information from the client list and stores the message in the off-line message list.

Whenever a client signs in, the server should search the off-line message list to see if there are off-line messages for that client. Then, it should send all the off-line messages to that client.

Test

User1	User2

	sobs>info 1 item1 user1 7 5 test item 2 item2 user1 7 9 desc sobs>buy user1 1 user1 has received your request. sobs>purchased 1 item1 5
sobs>user2 want to buy your 1 sobs> sold 1 item1 5 sobs>deregister You have successfully signed out. Bye!.	
	sobs> Client table updated. buy user1 2 user1 currently is offline. Request forwarded to the server.
sobs>register user1 Welcome user1, you have successfully signed in. Client table updated.	

sold 2 item2 9 sobs>	

Data structure:

Class UdpSobs	It's the class including main() function. It chooses the mode by check the argument '-s' or '-c'. The main() is also responsible for listening on client's keyboard input and server's data input.
Class Update	It's a thread used to listen on client's data input. It and main() are multithread.
Class Client	It has method such as register(), deregister(), Sell(), GetInfo(), Bid(), Ack(), DirectBuy(), DirectSell(). It includes most actions of a client can do.
Class Server	It has method such as register(), OfflineMessageSend(), deregister(), ClientSell(), GetInfo(), Bid(), DirectBuySeller(), DirectBuyBuyer() and BroadcastClientTable(). It includes most actions of a server can do.
Class ClientTableC	To store client online information for a client. Its attributes are Name, IP, Port.
Class ClientTableS	To store client online information for a server. Its attributes are Name, IP, Port.
Class ItemTable	To store on-sold item information. Its attributes are ItemName, SellerName, StartBid, CurrBid, TransLimit, BuyNow and Desc.
Class OfflineMessage	To store offline message in the server. Its attributes are UserName and Message.

Main Algorithms:

The program chooses the mode by check the argument '-s' or '-c'. In Server mode, single thread is used to listen to date input and do the operation. However, in Client mode, two threads is used. They are responsible for keyboard input and data input separately.

The program uses information flag to distinguish different kinds of requests or ACKs.

All flags are listed as below:

Message Flag: 1.register 2.deregister 3.client table update 4.sell an item 5.get items' information 6.place a bid 7.sold item 8.purchase item 9.purchase ack 10.sold ack 11.direct buy from buyer 12.direct sell(ack) 13.direct buy from seller 14.directbuy sold ack 15.direct buy ack to buyer

0.error arguments -1.item-num should be positive. -2.Error: itemnum not found -3.Error: owner -4.Error: duplicate bid -5.Error: negative bid

Any positive integer represents one normal request or ACKs and any non-positive integer represents an error messages.

After receives different flags, the server or the client will call corresponding function to operate the arguments following the flag.