Internet Protocols, Sections 002

2022 Fall Socket Programming Assignment - ver 1.0

In this project you will develop a two-client interrogation network using TCP sockets. The primary learning objectives are:

- to illustrate the client-server paradigm
- to gain experiences with the socket programming interface

1 Requirements

Background

Auction theory has been proven very useful in many types of economic markets. The basic scenario of an auction is to sell a piece of good or service to one of a group of potential Buyers with the maximum price. There are two types of auctions, open-bid auction and sealed-bid auction, which are commonly used in different scenarios:

- **Open-bid auction**: The bids are open to all participants. The auction commonly goes in a sequential manner, with either Buyers or the auctioneer continuously announcing new prices until a highest acceptable price is identified. In an **English auction** (aka open-bid ascending price auction), a lowest price is announced by the auctioneer, after which participants compete with each other by raising the price until a highest bid receives no further competition and wins the auction. In a **Dutch auction** (aka open-bid descending price auction), the auctioneer announces a high asking price, and continuously lower the price until a first accepting bid is received, which wins the auction. Note that in an open-bid auction, the final winning price is known to all participants.
- **Sealed-bid auction**: Instead of having multiple bidding rounds, a sealed-bid auction happens in only one round. Each participant secretly submits a bid in a sealed envelope to the auctioneer, who then finds the highest bid and the winner. In a **sealed-bid first-price auction**, participant with the highest bid wins, and pays exactly the price that he/she bids (the highest bid). In a **sealed-bid second-price auction** (aka **Vickrey auction**), participant with highest bid also wins, but only pays the price of the second

highest bid. Note that in a sealed-bid auction, the bids of each participant (including the winning bid) can be kept secret from all other participants, thus respecting participants' privacy.

The sealed-bid auction is very useful when people have secret values about the good and do not want to reveal their secret values to other participants. A typical application scenario is actually in the reverse auction scenario, where a government / company is choosing among multiple subcontracting companies to complete a specific task with the lowest price, a process called tendering for procurement.

Our Scenario

In this project, you will implement socket-based programs that will simulate the sealed-bid auctions. There will be 2 types of processes running to create this simulation: an **"Auctioneer" server** for hosting the auctions, and a **"Seller/Buyer" client** both for submitting auction requests to the auctioneer, and for bidding for items in the auction. A client can by either a Seller, or a Buyer, but not both.

An auction starts with a Seller submitting an auction request to the Auctioneer, which specifies the type of auction (first-price or second-price), the lowest price acceptable to the Seller, the intended number of bids, and the good to be sold. The Auctioneer opens up the auction after receiving the request (if no other auction is on-going), and starts accepting bids from Buyers. A Buyer submits a bid to the Auctioneer by first connecting to the Auctioneer server, and then submitting its bid. Once a bid is received from every connected, the Auctioneer processes all bids, finds the highest bid, and notifies the Seller and the Buyers status of the auction.

"Auctioneer" Server

The server is continuously listening on a welcoming TCP port for clients to connect to it. When the server detects that a new client is trying to connect to it, it creates a new TCP socket to talk to that specific client and does the following:

1. The server runs in two states: "Waiting for Seller" (status 0), and "Waiting for Buyer" (status 1). The server starts in status 0.
2. If the server is running in status 0, and there has not been any client connected to it, the next connected client assumes the Seller role. The server will inform the client to submit an auction request, and wait for it to submit its auction request. For any other in-coming connection before the server receives the auction request, the server replies "Server busy!" to the new client and closes the connection.
    a. To realize this, you can create a new thread to handle the input from the Seller client, while the original thread is still accepting new in-coming clients, sending out a busy message after each connected, and closing the connection.
    b. Display a message when the new thread is spawned.
3. Upon receiving a message from the Seller, the server decodes the message into four fields: <type_of_auction> (1 for first-price, and 2 for second-price), <lowest_price>

(positive integer), <number_of_bids> (positive integer < 10), and <item_name> (char array of size <= 255).
    a. If the decoding fails, the server informs the Seller “Invalid auction request!”, and waits for the Seller to send another request.
    b. If the decoding succeeds, the server informs the Seller “Auction request received: [message_received]”. The server then changes to status 1.
4. In status 1, the server will accept in-coming connections from up to <number_of_bids> clients as Buyers. When a client is connected:
    a. If the current number of Buyers is smaller than <number_of_bids>, the server informs the client that the server is currently waiting for other Buyers, and waits for other connections.
    b. If the current number of Buyers is equal to <number_of_bids>, the server tells every connected client “Bidding start!”, and starts the bidding process.
    c. If the current number of Buyers exceeds <number_of_bids>, the server tells the new client “Bidding on-going!”, and closes the connection.
5. The server starts the bidding process by creating a new thread for handling the bidding inputs from Buyers, while the original thread is dedicated to rejecting newly coming clients as in Step 4 during this process. Display a message when the new thread is spawned.
6. During bidding, the server will continuously receive data from any Buyer who has not submitted a bid yet. Upon receiving a message from a Buyer:
    a. If the message contains a single positive integer as the bid, the server records the bid with the sending Buyer, tells the Buyer “Bid received. Please wait…”, and does not accept any further message from this Buyer.
    b. If the message contains any other data, the server informs the Buyer “Invalid bid. Please submit a positive integer!”, and attempts to receive another round of message from this Buyer.
7. Upon receiving all bids, the server will run through all Buyers and their bids, and identify the Buyer with the highest bid ***b***. It then decides the result of this auction as follows:
    a. If the highest bid is no less than <lowest_price>, the auction succeeds. The server does the following:
        i. If the <type_of_auction> = 1 (first-price), the server tells the Seller that the item is sold for ***b*** dollars, then tells the highest-bid Buyer that it won in this auction and now has a payment due of ***b*** dollars, and finally tells each of the other Buyers that unfortunately it did not win in this auction. The server then closes the connection to all clients. (We omit the actual payment and good delivery in this project, and defer that to our next project.)
        ii. If the <type_of_auction> = 2 (second-price), the server finds the second highest bid ***b1*** among all bids including the **Seller’s <lowest_price>**. The server then tells the Seller that the item is sold for ***b1*** dollars, then tells the highest-bid Buyer that it won in this auction and now has a payment due of ***b1*** dollars, and finally tells each of the other Buyers that

unfortunately it did not win in this auction. The server then closes the connection to all clients.

b. If the highest bid is less than <lowest_price>, the server tells the Seller that unfortunately its item was not sold in the auction, and tells all other clients that unfortunately they did not win in the auction. The server then closes all connections.

8. After the auction is done and everyone is informed, the server cleans all the information about this auction including the list of Buyers, and then reset its status to **0**, to wait for any other Seller's connection and auction request.

"Seller/Buyer" Client

When the client starts, it tries to contact the server on a specified port.

Seller Mode

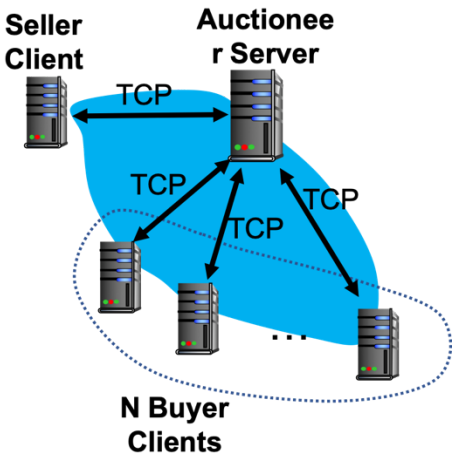
1. If the client receives message from the server to submit an auction request after connection, it holds the role of a Seller.
2. To proceed, the client sends an auction request, in the form of "<type_of_auction> <lowest_price> <number_of_bids> <item_name>", to the server, and waits for server reply.
3. If the server indicates an invalid request, the client must resend a valid request.
4. If the server indicates "Auction request received: [message_sent]", the client waits until the auction finishes to obtain the auction result.

Buyer Mode

1. If the client receives message from the server to wait for other Buyer or start bidding, it holds the role of a Buyer.
2. When the bidding starts, the Buyer sends its bid (a positive integer value) to the server.
3. If the server indicates an invalid bid, the client must resend a valid bid.
4. If the server indicates "Bid received", the client waits for the final result of the auction.

2 Example Output

The figure below represents a visual of the communication between the Seller/Buyer clients and the Auctioneer Server.



2.1 Starting the Auctioneer Server

The server is started by specifying the TCP port of the welcoming socket. In the example below, the server is running on port 12345. Display a message as in Step 1 in the figure.

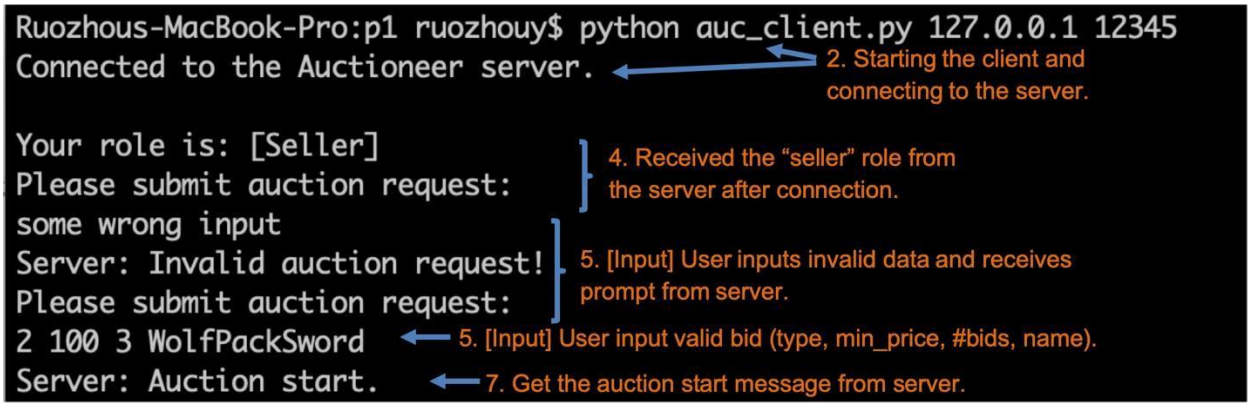
```
Ruozhous-MacBook-Pro:p1 ruozhouy$ python auc_server.py 12345
Auctioneer is ready for hosting auctions!
```

1. Starting the service and waiting for client.

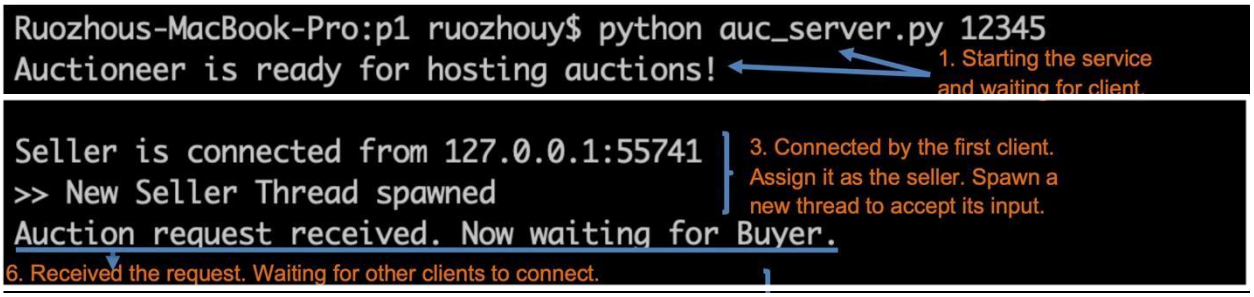
2.2 Starting the client (Seller)

When starting the client, specify the IP address of the Auction Server and the port number where the server is running. In the example below, the server is at IP address 127.0.0.1 running on TCP port 12345. If successfully connected, a message is displayed with relevant information, as in Steps 2 (client) and 3 (server), respectively. Step 4 is displayed when the client receives the Seller role prompt from the server. When a request is entered from the Seller side (Step 5), if it is invalid, the server should return an invalid prompt; if it is valid, the server should send a feedback to the Seller (Step 6) and set itself to wait for Buyers. The Seller displays the a message indicating the auction has started (Step 7). Here the request “2 10 3 WolfPackSword” is used as an example (which means “use auction type 2, starting at $10 minimum, allow 3 bids, for the item name ‘WolfPackSword’ to be sold.”)

On the Seller side:



On the server side:

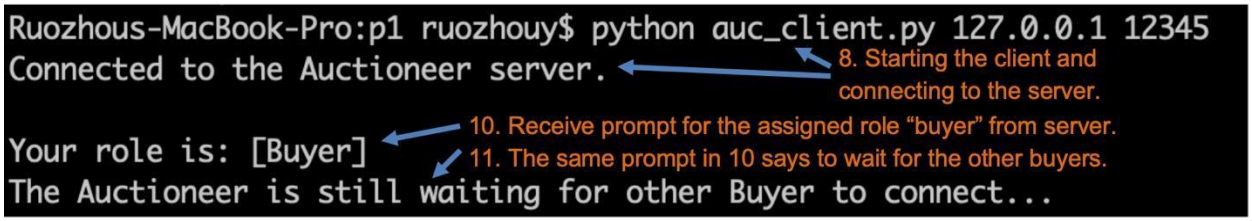


If another client is trying to connect to the server after the Seller is connected but before the request is received, the server will give a busy prompt, and then close the connection:

2.3 Starting the client (Buyer)

After the server receives a request from the Seller, Buyer clients can start to connect (Step 8). When connected, it will be told the assigned role and a waiting / bid start prompt sent automatically by the server after its connection (Steps 10—11).

Buyer connected and waiting for other Buyers:

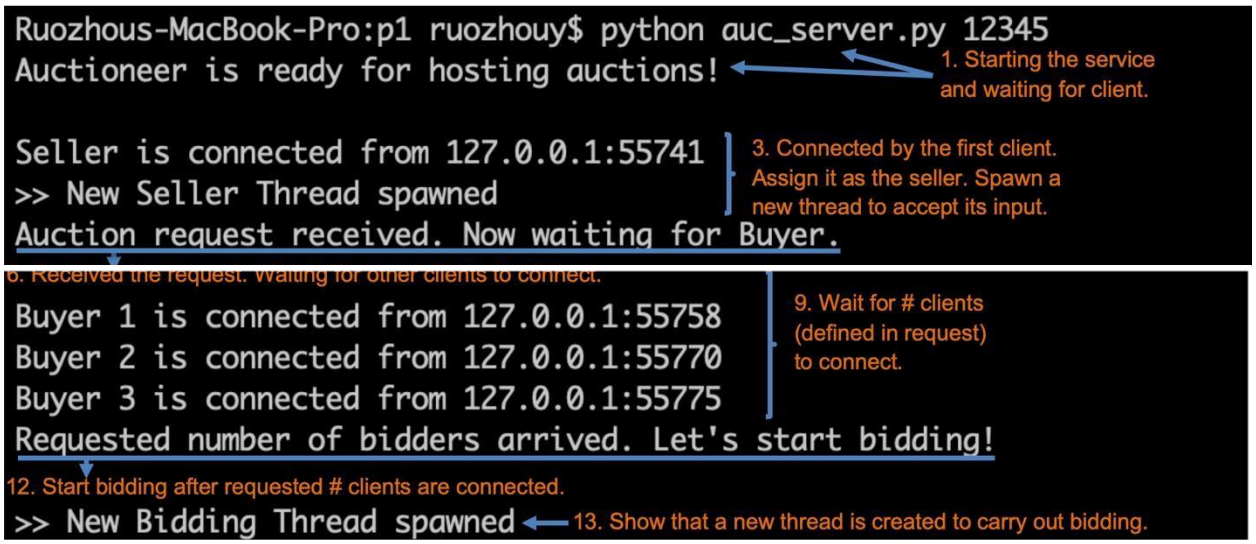


Buyer connected and bidding started immediately afterwards (no waiting prompt and the bidding start prompt immediately follows as in the next subsection):

```
Ruozhous-MacBook-Pro:p1 ruozhouy$ python auc_client.py 127.0.0.1 12345
Connected to the Auctioneer server.

Your role is: [Buyer]
```

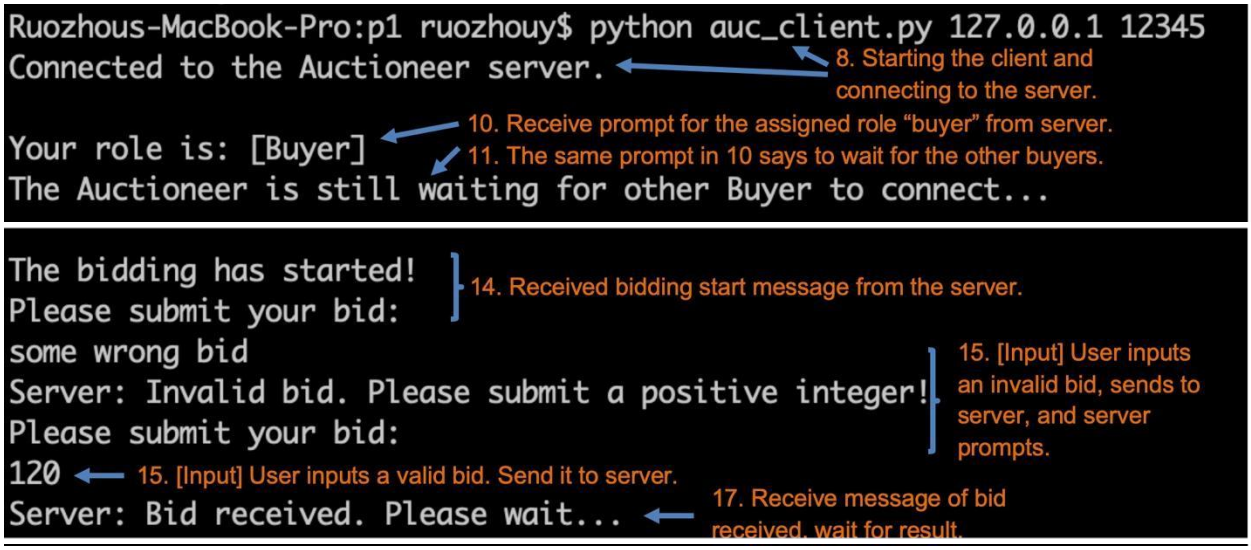
On the server side, a message should be posted whenever a client is connected (Step 9), in the meantime all clients should be informed of their roles, and whether the server is still waiting for other Buyers. After the requested number of Buyers are connected, the server will display that the bidding starts (Step 12), send a prompt to all the Buyers to start bidding, and spawn a new thread for handling the bidding (display a message as in Step 13).



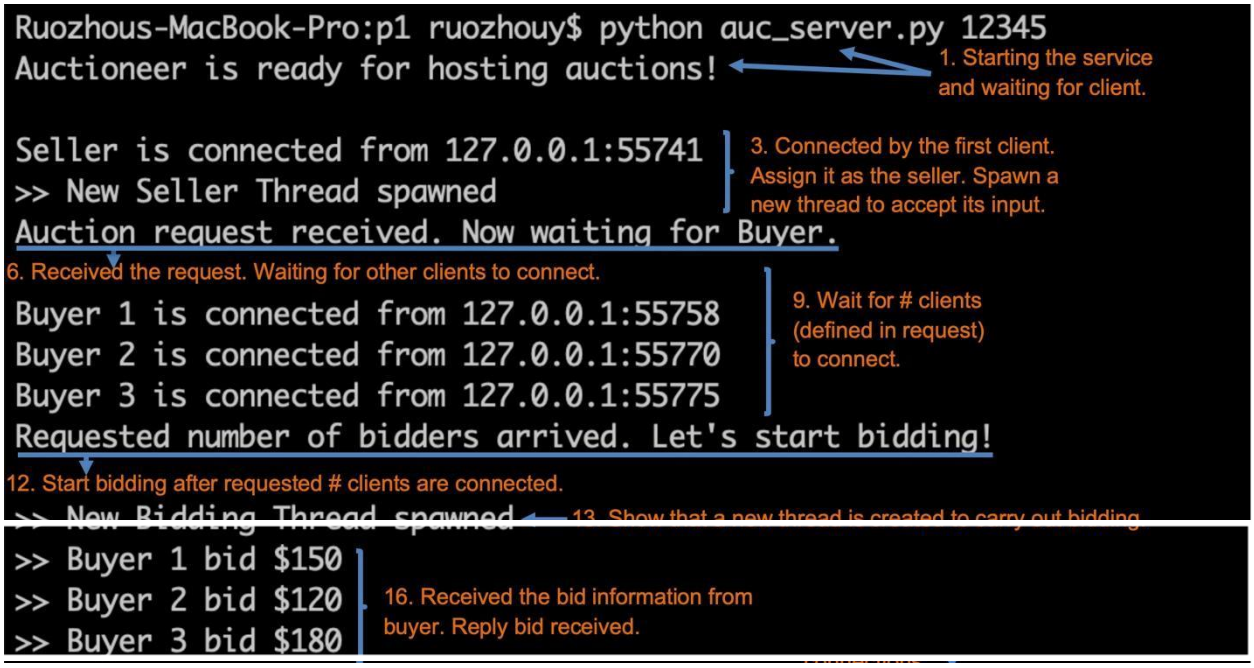
2.4 Bidding start

Once the bidding start, each Buyer will receive a message indicating they can start bidding (Step 14). They will each enter a bid (Step 15), get a prompt from the server indicating the bid is received (Step 17), and wait for the final result.

On the Buyer side:



The Server will need to display each bid received (Step 16).



If another client is attempting to connect to the server during bidding, the server should accept this connection, tell the client that it is busy, and close the connection. The client should display a message as follows:

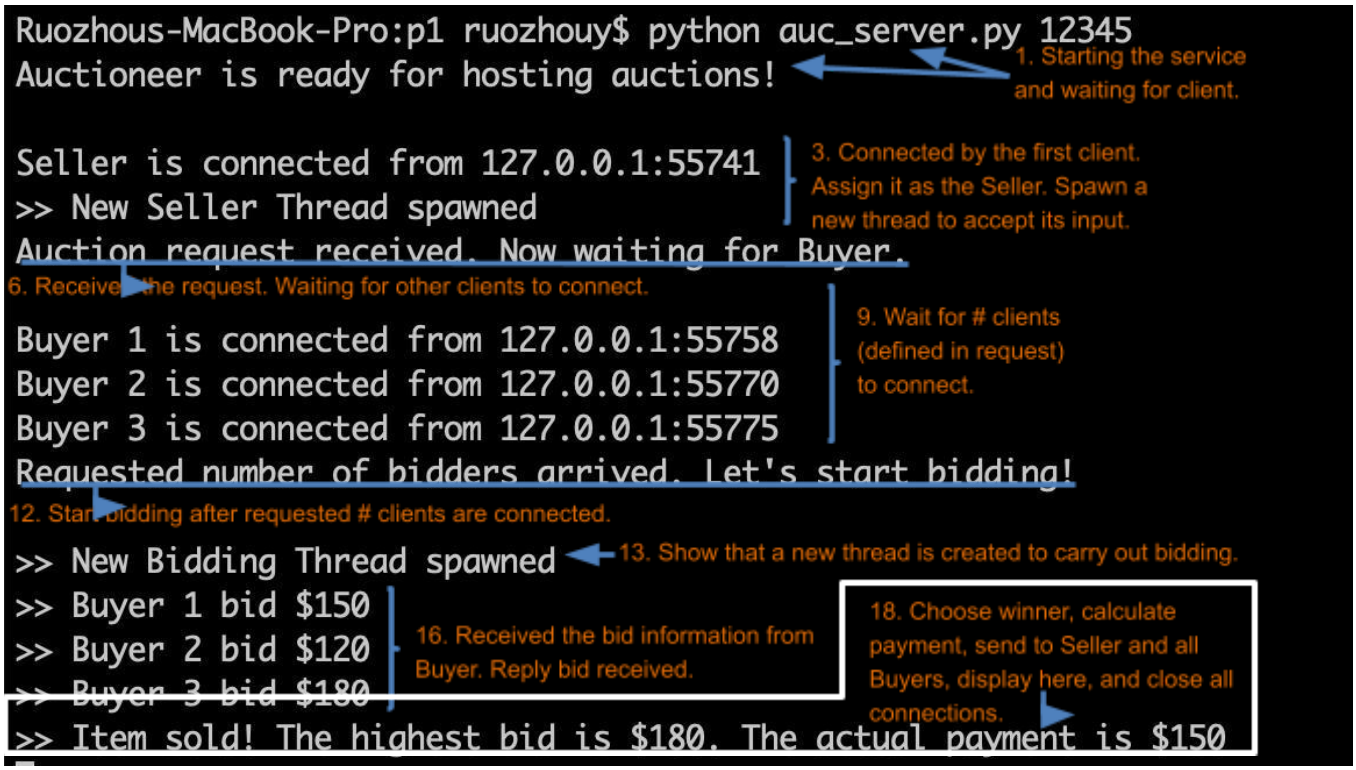
```
Ruozhous-MacBook-Pro:p1 ruozhouy$ python auc_client.py 127.0.0.1 12345
Connected to the Auctioneer server.

Server is busy. Try to connect again later.
```

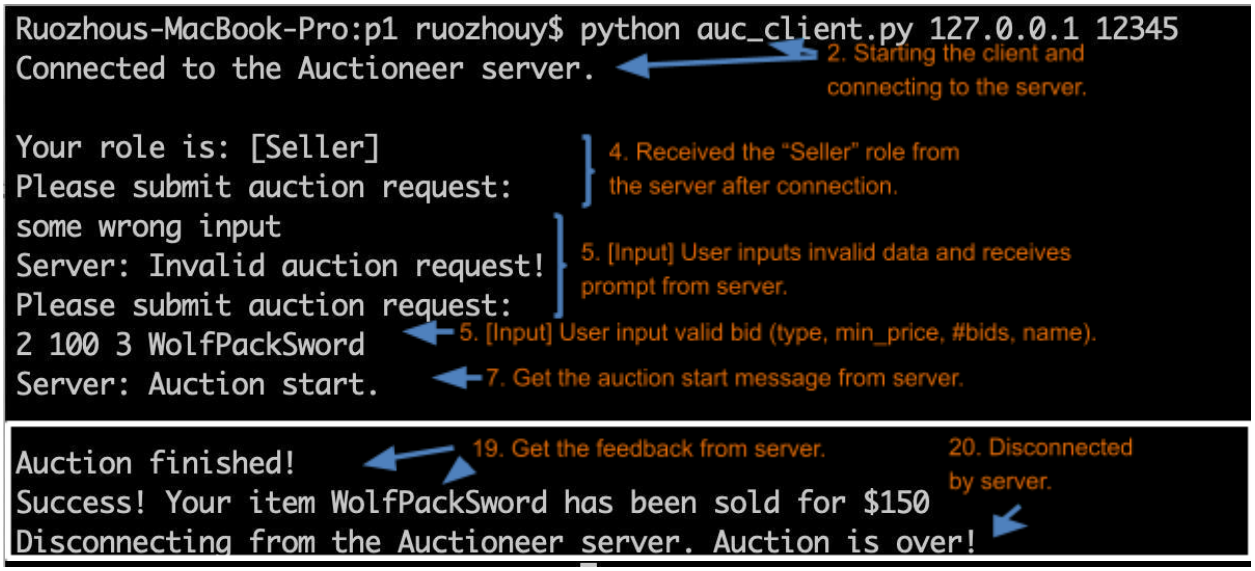
2.5 Get the final auction result

After all bids are received, the server should determine which Buyer wins this auction, and determine the price for the Buyer based on the policy defined in the request by the Seller client (Step 18). The server then needs to send the result via different messages to the Seller, the winning Buyer, and the losing Buyers (Step 19). The server finally closes the connection to all clients (clients display messages as in Step 20), and starts waiting for the next Seller to connect (for another round of auction). The following screenshots are essentially wrap-ups of the entire process of an auction, with highlights on outputs for the current step.

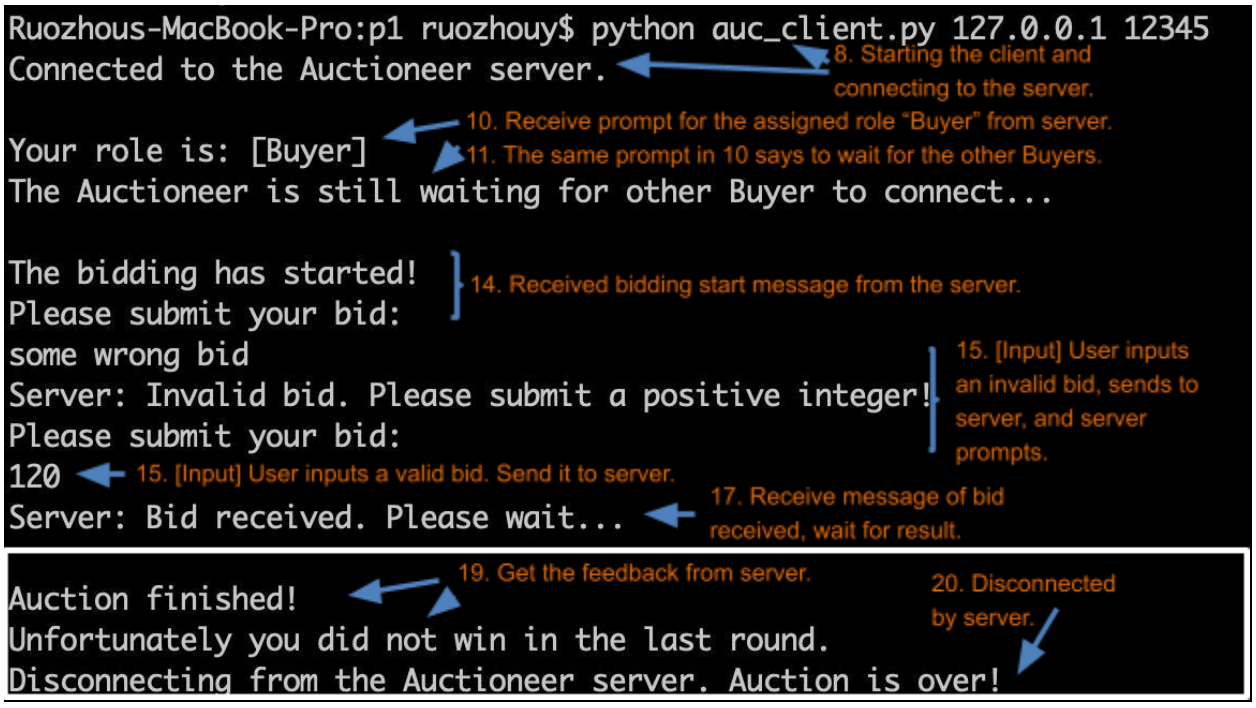
On the server side:



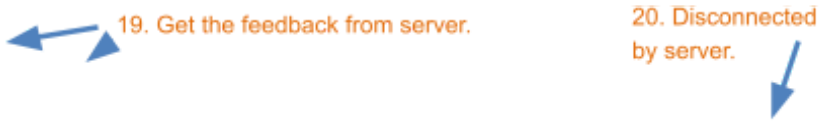
On the Seller side:



On the Buyer side:



Or if this Buyer won:



```
Auction finished!
You won this item WolfPackSword! Your payment due is $150
Disconnecting from the Auctioneer server. Auction is over!
```

3 Implementation Notes

- Write your code in Python 3. The textbook presents sockets programming in Python in Sec 2.7.
  A tutorial on Python is available at http://docs.python.org/tutorial/
  A guide for beginners to learn Python can be found at
  https://developers.google.com/edu/python/
  For network sockets, check out: 4 Beej's Guide to Network Programming
  Python documentation: https://www.python.org/doc/
- Some useful socket functions: **.send(), .recv(), .close(), .accept(), .bind(), .listen()**
- For starting new threads, you might want to use **thread.start_new_thread()**
- Include proper error checking, when the server and a client communicate
- During development, you can use your own system to write and debug your program. You will of course need to install and setup Python 3.
- You can run the server and the clients locally. The server would be running locally on your system on a specified port. Therefore, the clients will try to connect to 127.0.0.1 on that port.
- Test your code using VCL systems, see Section 4.

4 Grading done on VCL systems

To grade your assignment, we will use Virtual Computing Lab (VCL) machines. So once you are done testing locally your code, then you should try it out on VCL. Go to and make a reservation for the VCL image called . This image is for Linux CentOS 7 and already has Python 3.6.8 installed.

When you are testing your code on the VCL systems, you have to run your Auctioneer Server on ports in the range **[3000 to 5000].** Other ports are blocked via firewall rules so these are the only ports that will work. Your code should handle an arbitrary number N of clients connecting, and you should test it for at least N=3 on VCL. So you can request and reserve **up to five VCL Virtual Machines (VMs):** one VM for Auctioneer Server, one VM for Seller Client, and up to four VMs for Buyer Clients. (see first figure in Section 2).

Instruction of file transfer
You can use scp (for MacOS or Linux) or winscp (for Windows) to transfer files to the VCL machines.

- scp: if you want to transfer your local file to remote machine, the following command can be used:
  scp your_local_path_and_filename username@remote_ip_address:remote_path_stores_this_file
  If you want to transfer a directory to the remote machine, add the '-r' option after the 'scp' command.
- winscp: Once you create a new session with SFTP protocol and login in that machine, you can directly transfer local files to the remote machine by drag and drop.

5 Submission Details

The project is due Thur, Oct 13, 2022 at 14:55 p.m. Anything beyond the deadline receives a 0% score.

5.1 Team Work

You will be able (but not required) to work in teams of two. If you choose to work with a partner, there won't be any excuses in regard to partner not doing her/his share of the work.

Everyone in the team needs to submit the final version of the project. Please make sure that each submission includes the names of all the members of the team.

5.2 Plagiarism

The work submitted has to be yours. If you do end up using some reference code you found online, make sure that you cite the website. Make sure you comment all important lines so that it is clear you have a good understanding of the code.

5.3 What to submit

1. Well-commented **source files** (in python): **auc_server.py** and **auc_client.py**. Put your name(s) at the top of each source file.

2. A **README pdf** file. In the README file, please provide the following information:
   - Your name(s)
   - Your unityid(s)
   - **Screenshots** of you testing your code, i.e. running an auction server, connecting via clients, etc. Add discussion on what the different screenshots mean.
   - **A detailed description on how to compile & run your code.**
   - (Optional) Your entire answer to the extra credit question as below.
   - (Optional) Any other thing you want to tell us.

6 Extra Credit Question

This is an advanced question and is optional to answer. If correct, extra credit will be granted to this project, following the rubric as details in the next section. However, you must both correctly answer the question with explanation of your reasoning, and provide your detailed steps (including screenshots and annotations for each step) for validating your answer using the auction program you developed. In other words, **the grading for this extra credit is binary**.

Extra Credit Question

As discussed at the beginning, a sealed-bid auction is commonly used when Buyers each has a (potentially different) private valuation v_i of the exhibited item, which should not be known to anyone else other than the trusted Auctioneer. As an example, imagine you are bidding for a piece of ancient antique. You may think the piece is extremely important to you, because with this piece you can get a full collection of antiques that worth a lot of money; others may thus not have such a high valuation of the item like you since they do not have that collection. But you do not want to reveal this information to the other competitors as in an open-bid auction, as they may intentionally raise the price to make you pay more for this piece.

Following this intuition, a Buyer's utility in an auction is determined by two factors: 1) whether it wins the item, and 2) the price it has to pay for the item if it wins. Let us assume that each Buyer defines its utility as follows:

1. If it loses in the auction, its utility is $u = 0$.
2. If it wins the auction and has to pay a price of p, its utility is defined as $u = (v_i - p)$.

The goal of each Buyer is to maximize its own utility in this auction, regardless of what the others do.

Now, say you act as the Seller. Instead of selling the item for the highest price, **you are more interested in knowing the true valuation of this item from each Buyer** while still selling it for a reasonable price. The auctioneer has assured you that you will be told the bidding prices of all Buyers, but you have no control over what price each Buyer will bid. In this case, by analyzing the potential bidding strategy of each Buyer, will you choose a **first-price auction**, or a **second-price auction**, to achieve your goal?

Expected Answer for Extra Credit

To earn the extra credit for this project, you must correctly submit all of the information below:

1. Your answer: first-price, or second-price.
2. Your reasoning: why does your answer fulfill the goal of the Seller?
   Hint: To answer this, you can perform utility analysis of the Buyers, i.e., what is the utility that a Buyer will get, when either **bidding its true valuation** or **bidding another price higher or lower than its true valuation**, in both the winning case and the losing case, under both auction types. Find the auction type where a Buyer will receive the maximum utility by choosing exactly its true valuation as its bidding price.
   Note: If you use any online source (like Wikipedia) for your analysis or reasoning, you must properly cite it when writing your answer. Otherwise, you will run into academic

integrity issues.

3. Execution steps (with screenshots and annotations) of your programs that validate your reasoning.
   Hint: Essentially you want to show the utility a Buyer obtains by submitting its true valuation vs. submitting any other value, under the auction type you selected. You may use a minimal two-Buyer example for answering this question. You can assume that the Buyers always have different private valuations of the item, and that they each acts independently to maximize its own utility (i.e., they won't collude).

7 Grading Rubric

This is how your project will be graded:

Points	Description
10	**README Format:** It is well formatted with all required details
10	**Output:** Terminal output similar to the skeleton
4	Client 1 connected as Seller
4	Seller socket created
4	Seller receives role prompt from Auctioneer
4	Another client tries to connect during Seller process and shows waiting for Seller
4	Seller sends incorrect auction request and receives feedback
4	Seller sends correct auction request to Auctioneer
4	Seller receives auction start prompt
4	Clients 2 to (N+1) connected as Buyers 1 to N
4	Buyers 1 to N sockets created
4	Buyers 1 to (N-1) receive role and waiting prompt from Auctioneer
4	Buyers 1 to N receive bidding start prompt from Auctioneer
4	Another client tries to connect during bidding and shows bidding in-progress
4	A Buyer sends incorrect bids and receives feedback
4	Auctioneer (internally) displays all bids and auction result (sold/unsold, price)
4	Seller receives final auction result (sold/unsold, price)
4	Buyers 1 to N receive final auction result (win/lose, payment due)
4	All sockets (Seller & Buyers) closed
4	Another client tries to connect after current auction is done
4	Another round with a different auction type works as above
4	**Source code:** Well commented
Possible Extra Credit or Deductions	
+10	Extra credit question fully, correctly answered, including answer, reasoning, and program validation.
-2	Name(s), date, etc. not included at the top of each .py file
-5	Incorrectly Named Files (**auc_server.py and auc_client.py**)

-5	Additional, irrelevant output displayed to the Console Window
-10	Code results in an error
-100	Submitted after 14:55 PM on 10/13/2022