Project
Stock Auctions with Socket programming

Stock auction
A stock auction is an auction through which different traders can purchase particular securities (items) by bidding on them. Since the realistic stock auction process is quite complicated, you are required to implement a simplified version of this auction similar to regular open auctions (IPL, Ebay, etc.). The process is explained briefly below.

- Each item at the auction has a base price.
- Once the auction has started, the bidding process begins when a client makes a bid higher than this base price.
- From that point onwards, each client can make a bid on an item by specifying a bid higher than the previous bid. (similar to an open auction)
- Ultimately, the highest bidder wins the item.
- For simplicity, we assume that no two clients would bid on the same item within a time frame of 500 ms.
- Auctions for different items occur in parallel (not in a one-by-one order of items)
- The bidding ends at the end of the day. However, if someone (trader) bids within the last minute (60 sec), the bidding time for that particular item is increased by a minute.
    - Ie: If the highest bid of AAPL is $50 and someone bids $55 at 40 seconds before the end of the day, bidding ends 60 seconds after on the next day if no more bids.

Description
The stock auction has 3 stakeholders, and each stakeholder has specific objectives which are listed below.

- Auctioneer (Server) - The server functions as the auctioneer and must allow traders to make bids on specific items.
- Traders (Client / Subscriber) - The traders should be able to make bids through the auction server. They should also be able to get regular updates on changes to the highest bid of the interested items, and other financial information such as profits.
- Companies (Publisher) - Companies must be able to inform the traders regarding monthly profits.

For this project you will implement both the client-server model and the publisher-subscriber model. The server must perform 3 main tasks.

1. The client-model replicates the functionality of a stock auction. Clients must be able to bid for different items at a stock auction through a server.
2. Companies must be able to publish profit information to the server and traders should be able to get these updates by subscribing to particular topics.
3. Stakeholders should be able to get notification when bids are updated for an item.

The specific details of each component are given below.

Client - Server

- The server contains information about different items:
    - eg : AAPL, AMZN, FB, MSFT, GOOGL, TSLA and YHOO.
- Each item has the following information which is given as a CSV:
    - Symbol, Base price, Security Number and Profit.
        - eg: AAPL, 10, 12345, 100

- Server will be listening to incoming connections on port 2021. It should be able to handle more than one connection at a time.
- The server first requests an ID from the client. Once this ID has been established it cannot be changed.
- Then the server provides the following functions:
    - **Display the current highest bid of stocks**: Once the ID is given, the client is expected to provide the Symbol of the item he/she is willing to bid on. If the provided Symbol is found the server should reply back with the current highest bid or -1 to indicate that the Symbol is invalid.
        - Query Format : [SYM]
            - eg: AAPL
        - Output Format : (Bid) / -1
            - eg: 10
    - **Accept bids**: The server must accept new bids if the bid is higher than the current bid of the item. If the bid is successful, the new bid becomes the highest bid and the server must reply back with the new bid. If the symbol is invalid the server replies -1 and if the bid is lower than or equal to the current bid or if the bid time has expired, the server replies -2. (Assume that no two people bid within an interval of 500ms.)
        - Query Format : [SYM] (Bid)
            - eg: AAPL 15
        - Output Format : (Bid) / -1/-2
            - eg: 15
    - ~~**Track bid change** : The server should be able to track all the changes done to the stock item; how the offers varied with time and who made the offers. (Hint: Update a SYM.txt file every time a valid bid is made. This would also be helpful in the next section)~~
    - **Close bid** : Assume that the bidding ends after a fixed time period *t* after starting the server. Note that only the bidding ends at time *t*, but the server must be active so that traders can view the final bid of each item. The parameter *t* should be set at the start as a command line argument.
        - eg: For a 1 hour bidding period, you should be able to run the server using the following command - **java main 60**
    - **Note** :
        - The Security Number and Profit information are not required for this section.

        - CSV should not be updated. It should only be read at the beginning.

(**10 marks**)

Publisher - Subscriber

This is an extension to the client server model but can be implemented separately (with or without code re-use). Publisher-subscriber model is based on topics, where the publishers publish to certain topics and subscribers could subscribe to said topics to get updated information.

- The item information remains the same as in the previous case.
- Server will be listening to incoming connections on port 2022. It should be able to handle more than one connection at a time.
- The server first requests an ID from the client. Once this ID has been established it cannot be changed.
- The server provides the 2 functionalities in this case

1. **Updating profit information** : The publishers (companies) must be able to publish the monthly profits to inform all the stakeholders regarding the financial status of each item. The subscribers on the other hand must be able to subscribe and get immediate updates on these financial information as soon as they're published. The query formats for both the subscribers and the publishers are given below.
    - **Publish updated profits** : The publisher must be able to publish monthly profits by providing a valid Symbol and Security number. If the information is valid, the server must reply 0, and if invalid, the server will reply -1
        - Query Format : [SYM] [SEC] (Profit)
            - eg: AAPL 12345 150
        - Output Format : 0/-1
    - **Subscribe to get updates on profits**: The subscriber must be able to subscribe by providing the item symbols and then receive updates continuously. Subscription is done by providing valid space separated symbols. (SYM1, SYM2, SYM3 are the symbols of different items.) If the subscription is successful, the user would get a space separated output of 0 if the symbol is valid or -1 otherwise for each symbol. Once subscribed, the subscriber gets the symbol and profit at each update.
        - Query Format : PRFT [SYM1] [SYM2] [SYM3]
            - eg: PRFT AAPL AMZN
        - Output Format : 0/-1 0/-1 0/-1
            - eg: 0 0
        - Subscriber output : [SYM] PRFT (Profit)
            - eg: AAPL PRFT 150

(**5 marks**)

2. **Updates on bids** : Interested stakeholders can track bids of an item and get updates as the bid of the particular item increases.
    - **Subscribe to get updates on bid changes**: The subscriber can subscribe to a particular item and once subscribed they must be notified whenever the bid is updated. The query and the outputs are similar to the previous case.
        - Query Format : BID [SYM1] [SYM2] [SYM3]
            - eg: BID AAPL AXXN
        - Output Format : 0/-1 0/-1 0/-1
            - eg: 0 -1
        - Subscriber output : [SYM] BID (Bid)
            - eg: AAPL BID 15

(**10 marks**)

Getting started:
This involves a fair bit of coding. So first consider each of the operations needed and how that can be implemented before you start coding. You should be able to re-use most of the code from the server-client to publisher-subscriber model. Divide the workload among the team mates.

Hint:
- Note that the profit data and the bids are independent of each other.
- Once the server is established, you should be able to connect to the server using a common communication tool such as nc or telnet.
- The publisher-subscriber model for bid update could be slightly different (since the publisher is not defined explicitly). However, this can be implemented by creating a virtual subscriber on the server side which subscribes to the particle topic after each valid bid. You are free to use any other techniques as well.

Report:
In addition to the implementation (code), submit a report containing the following information.

- Design the protocol in a **diagram of the client-server model** highlighting the bid timing mechanism through the diagram or otherwise.
(**2 marks**)
- Design the protocol in a **diagram of the publisher-subscriber model** showing the information exchange between the publisher-subscriber and the auctioning server.
(**3 marks**)
- Describe a mechanism to ensure that the client **ID can be kept unique** for each user.
(**2 marks**)
- Clearly **justify why a publisher-subscriber model is suitable** for bid and profit updates (as opposed to client-server).
(**2 marks**)
- Clearly state **functional and non-functional requirements** of the publisher-subscriber model.
(**2 marks**)
- Explain how you use transport layer protocol to ensure these requirements.
(**2 marks**)
- State the additional functionality you add in the application layer to fulfil the requirements not guaranteed in the transport layer.
(**2 marks**)

Submission:
Submit your code and the report as a single zip file to LMS (moodle) before the deadline. You should also add a README file which would explain how to use the application.
Submission format : [Course code]_G[Group no.].zip
eg : CCS201_G01.zip

Submission deadline:
Task1 : Basic client server implementation
After Mid Semester Week

Task2 : Publisher subscriber implementation
Task3 : Report
By the end semester week