Mini Project
CCS1305 – Communication Models and Protocols
Real Time Stock Market Auction using Python Web-Socket

Stock Market Auction

A stock market auction is an auction through which different people can purchase particular organization's stocks by bidding on them. Since the realistic stock market auction process is quite complicated, you are required to implement a simplified version of this kind of auction similar to regular open auctions (IPL, EBAY, etc.). The process is explained briefly below.

- Each organization's stock 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 a stock by specifying a bid higher than the previous bid. (Similar to an open auction)
- Ultimately, the highest bidder wins the stock.
- **For simplicity, we assume that no two clients would bid on the same property within a time frame of 500ms.**
- Auctions for different stocks occur in parallel (not in a one-by-one order of stock)
- **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 stock is increased by a minute.**
    - e.g.: If the highest bid of a property is $50 and someone bids $55 at 40 seconds before the end of the day, bidding time increase by 60 seconds after bidding. If no more bids, during increased time period bid will close. Else, will have a bid on increased time period, the time will increase again by 60 seconds.

Description

The stock market 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 buyers to make bids on the stocks and must allow sellers to publish the details about the stocks to be sold.
- **Buyers (Client / Subscriber)** - The buyers should be able view the details of the stocks and make bids through the auction server. They should also be able to get regular updates on changes to the highest bid of the interested stocks.
- **Sellers (Publisher)** - Sellers must be able to publish the information about the stocks to be sold.

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-server model replicates the functionality of a stock market auction. Clients must be able to bid for different stocks through a server.
2. Sellers must be able to publish the stock information to the server and buyers should be able to get these updates by subscribing to particular stocks.
3. Buyers should be able to get notifications when bids are updated for a stock.

The specific details of each component are given below.

1st Step: Client - Server

- The server contains information about different organization's stocks:
    - e.g.: AAPL, AMZN, FB, MSFT, GOOGL, TSLA and YHOO.
  - ➢ Note: Additionally, You can give meaning full names to these key words form the CSV and you can make a UI to show these stocks
- Each stock has the following information which is given as a CSV:
    - Stock Code, Base price, Stock Security, Profit
        - e.g.: AAL, 1.32, 74902, 7500
- Server will be listening to incoming connections on port 2021. It should be able to handle more than one connection at a time.
  - ➢ Note: Server should have an IP address to access the server, then port is to access the application
- The server first requests an ID from the client. Once this ID has been established it cannot be changed.
  - ➢ Note: Client when request the application from server, server should ask an ID from the client (Log ID).
- Then the server provides the following functions:
    - **Display the current highest bid of Stocks in Stock Auction**: Once the ID is given, Client can log in to the server. Server will identify the client through the ID. Once client log in to the server, server will provide the all details about auction. (Note: You are free to use any kind of design methods to implement this)
    - **Accept bids**: The server must accept new bids if the bid is higher than the current bid of the Stock. If the bid is successful, the new bid becomes the highest bid and the server must reply back with the new bid. If the Stock Code is invalid the server replies **Invalid Stock Code** and if the bid is lower than or

equal to the current bid or if the bid time has expired, the server replies **Invalid Bid**. (Assume that no two people bid within an interval of 500ms.)

  - Query Format for Application: [SYM] [Bid]
    - e.g.: AAL 10
  - Output Format for Application: Successfully Bided! [SYM] [New Bid] / Invalid Stock Code [SYM] / Invalid Bid [SYM] [Current Bid]
    - e.g.: Successfully Bided! AAL 10 / Invalid Stock Code AAL / Invalid Bid AAL 5

- **Track Bid Change**: The server should be able to track all the changes done to the stocks; 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 buyers can view the final bid of each property. The parameter *t* should be set at the start as a command line argument.
  - e.g.: For a 1-hour bidding period, you should be able to run the server using the command from the server side.
  - Furthermore, if a bid is made at the last minute the bidding period increases.
- **Special Note**:
  - CSV should not be updated. It should only be read at the beginning.
  - You can use text field for type each command. But you have to access the server using these commands.

(5 Marks)

2nd Step: 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 property 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 Information**: The publishers (Sellers) must be able to publish the information about the Stocks to be sold. The subscribers on the other hand must be able to subscribe and get immediate updates on this information as soon as they're published. The query formats for both the subscribers and the publishers are given below.
    - **Publish Information**: The publisher must be able to publish the information by providing a valid Stock Code. If the information is valid, the server must reply **Successfully Published**, and if invalid, the server will reply **Invalid**
        - Query Format: PUB [SYM] (information) [SECURITY]
            - e.g.: PUB AAL "Open Day on 3.4.2022" 74904
                - **Note: You have to use double quotes between message**
        - Output Format: Successfully Published! [SYM] (information) / Invalid Stock Code [SYM] / Invalid Security Code [SYM]
            - e.g.: Successfully Published! AAL "Open Day on 3.4.2022" / Invalid Stock Code AAL / Invalid Security Code AAL
    - **Subscribe to receive information**: The subscriber must be able to subscribe by providing the Stock Code and then receive updates continuously. Subscription is can done by for many stocks providing valid space separated Stock Codes. (SYM1, SYM2, SYM3 are the symbols of different properties.) If the subscription is successful, the user would get separated output of **Successfully Subscribed** if the symbol is valid or **Invalid** otherwise for each symbol. Once subscribed, the subscriber gets the details at each update.
        - Query Format: SUB [SYM1] [SYM2] [SYM3]
            - e.g.: SUB AAL AAME
        - Output Format: Successfully Subscribed! [SYM] / Invalid Code [SYM]
          Successfully Subscribed! [SYM] / Invalid Code [SYM]
            - e.g.: Successfully Subscribed! AAL / Invalid Code AAL
              Successfully Subscribed! AAME / Invalid Code AAME
        - Subscriber Output: [SYM] (information)
            - e.g.: AAL "Open Day on 3.4.2022"

(5 marks)

2. **Updates on bids**: Interested stakeholders can track bids of a Stocks and get updates as the bid of the particular Stocks increases.
    - **Subscribe to get updates on bid changes**: Once a client subscribes to a property as specified above, the subscriber must be notified whenever the bid is updated. The outputs are similar to the previous case.
        - Query Format: SUB [SYM1] [SYM2] [SYM3]
            - e.g.: SUB AAL AAME
        - Output Format: Successfully Subscribed! [SYM] / Invalid Code [SYM]
          Successfully Subscribed! [SYM] / Invalid Code [SYM]
            - e.g.: Successfully Subscribed! AAL / Invalid Code AAL
              Successfully Subscribed! AAME / Invalid Code AAME
        - Subscriber Output: [SYM] BID (Bid)
            - e.g.: AAL BID 10

(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 stock information and the bids are independent of each other.
- Once the server is established, you should be able to connect to the server.
- 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.
- **You need to use Python programming language to implement this project**.

Demo and Viva:

Project Implementation and group member contribution are evaluated in a demo and a viva.

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 mark)

- Design the protocol in a **diagram of the publisher-subscriber model** showing the information exchange between the publisher-subscriber and the auctioning server.

  (1 mark)

- Describe a mechanism to ensure that the client **ID can be kept unique** for each user.

  (1 mark)

- Describe the data structure you used to track the bid changes. Justify its effectiveness.

  (1 mark)

- Clearly **justify why a publisher-subscriber model is suitable** for bid and profit updates (as opposed to client-server).

  (1 mark)

- 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.

  (1 mark)

- State the additional functionality you add in the application layer to fulfil the requirements not guaranteed in the transport layer.

  (1 mark)

Submission:

Submit your code and the report as a single zip file to LMS (Moodle) before the deadline.

Submission Format: [Course code] _ [Group no.].zip

e.g.: CCS1305_01.zip

Note: Incorrect formats are penalized

Submission deadline:

Task1: Basic client server implementation

Week After Mid Semester

Task2: Publisher subscriber implementation

Task3: Report

End of the week after End-Semester last week