PaveAI Work Sample: Stock Exchange

Description & Example

Your task will be creating the server for a stock exchange. The main purpose of a stock exchange is connecting buyers with sellers for a particular stock.

If Bob wants to buy a stock, he will offer the maximum he is willing to pay. If Sara wants to sell a stock, she will offer the minimum she is willing to accept. If there is overlap between those two offers, a match will be made. See the example below for more details.

Note: APPL is the Apple stock

```
Bob: offer to buy 1 APPL for at most $95
Bill: offer to buy 1 APPL for at most $97
Bill: offer to buy 2 APPL for at most $95
Sam: offer to sell 2 APPL for at least $102
Sara: offer to sell 3 APPL for at least $103
Simon: offer to sell 10 APPL for at least $120
```

The current active APPL orders are now:

- BUY 2*95 (Bill)
- BUY 1*95 (Bob)
- BUY 1*97 (Bill)
- SELL 2*102 (Sam)
- SELL 3*103 (Sara)
- SELL 10*120 (Simon)

```
Sam: what is the status of my order
Server: your order is pending, you sold 0 shares

Bart: offer to buy 3 APPL for at most $104
Server: your order is complete, you bought 2 shares at $102 and 1 share at $103
```

Note: Here Bart's orders were matched with Sam's and Sara's.

```
Sam: what is the status of my order
Server: your order is complete, you sold 2 shares at $102

Sara: what is the status of my order
Server: your order is pending, you sold 1 shares at $103
```

*Note: All of Sam's shares are sold therefore the order is complete. But only part of Sara's shares have sold so it is still pending. Bart was willing to buy at most $104 so the server matches him with the best prices available (2*102 from Sam and 1*103 from Sara). If there are two people willing to buy/sell at the same price, the person who submitted their order first should be completed.*

The current active APPL orders are now:

- BUY 2*95 (Bill)
- BUY 1*95 (Bob)
- BUY 1*97 (Bill)
- SELL 1*103 (Sara)
- SELL 10*120 (Simon)

```
Betty: offer to sell 1 APPL for at least $96

Betty: what is the status of my order
Server: your order is complete, you sold 1 shares at $97

Simon: cancel my order with ID 123
Server: your order is done, you sold 0 shares
```

The current active APPL orders are now:

- BUY 2*95 (Bill)
- BUY 1*95 (Bob)
- SELL 1*103 (Sara)

Documentation

The following APIs are defined in the stock_exchange.proto file.

OrderCreate/OrderStatus/OrderCancel

Create, check the status, and cancel an order. When creating an order, the ID is generated by the server (any value can be selected). This same ID must be used to check or cancel an order.

OrderIdRequest

- order_id: (uint64) A unique identifier for the order.

Order

- user: (string) The username of the person who owns the order. e.g. “Bob”
- stock: (string) The name of the stock they want to trade. e.g. “APPL”
- buy: (boolean) If this is a buy order, this value is set to true. If it is a sell order, the value is set to false.
- quantity: (float) The number of shares the user wants to trade.
- price: (float) The price at which the user is willing to buy or sell (either the max price they are willing to buy, or the min price they are willing to sell).
- created_at: (uint64) A unix timestamp for when the order was created. This value may be in the past (orders may be received with created_at times that are not increasing, you should assume that the created_at value is always valid)

OrderStatusResponse

- order_id: (uint64) A unique identifier for the order.
- active: (bool) Whether the order is active on the markets. An order is active until either all the shares are traded or the user cancels the order.
- matches: (repeated OrderMatch) All the “matches” or transactions that have been completed for this order. See OrderMatch for details.

OrderMatch

- quantity: (float) The number of shares that was matched.
- price: (float) The price at which the trade was matched.
- created_at: (uint64) A unix timestamp for when the trade was matched.

The transactions that have been matched between two orders. In the example above, Bart's order would have the the following *matches* field:

```
{
        matches {
                quantity: 2.0
                price: 102.0
                created_at: 15000000001
        }
        matches {
                quantity: 1.0
                price: 103.0
                created_at: 15000000001
        }
}
```

And Sara's order would only have 1 match:

```
{
        matches {
                quantity: 1.0
                price: 103.0
                created_at: 15000000001
        }
}
```

UserOrders

Display the orders for one user that were created between a certain start and end time.

UserRequest

- user: (string) The username of the person who owns the order. e.g. "Bob"
- start_time: (uint64) All returned orders must have a created time of at or after this time (>=).
- end_time: (uint64) All returned orders must have a created time of at or before this time (<=).

OrderStatusResponse

- See above

MultiOrderStatusResponse

- orders: (repeated OrderStatusResponse) All the OrderStatusResponses that match this query.

StockVolume1h/StockPrice1h

Returns the volume traded or average price of a particular stock in the last hour based on the orders matched.

BONUS - Make these two operations constant time (ie O(1))

VolumeRequest

- stock: (string) The name of the stock they want information about. e.g. “APPL”

VolumeResponse

- volume: (float) The total volume of all the stocks traded in the past hour.

In the example above, the volume would be (2+1+1) = 4. Note that only matched amounts are counted here.

PriceRequest

- stock: (string) The name of the stock they want information about. e.g. “APPL”

PriceResponse

- price: (float) The average price of all the stocks traded in the past hour.

*In the example above, the average price would be (2*102 + 1*103 + 1*97)/(2+1+1) = 101.0. *

OHLC

OHLC stands for open, high, low, close. For the given stock and timeframe, it calculates certain stats; see the response object for more details.

OHLCRequest

- stock: (string) The name of the stock they want information about. e.g. “APPL”
- start_time: (uint64) The start time for the data
- end_time: (uint64) The end time for the data

OHLCResponse

- open: (float) The first price the stock was traded within the specified timeframe.
- high: (float) The highest price the stock was traded within the specified timeframe.
- low: (float) The lowest price the stock was traded within the specified - timeframe.
- close: (float) The last price the stock was traded within the specified timeframe.
- volume: (float) The total volume of all the stocks traded within the specified timeframe.

In the example above, the data would be:

```
OHLCResponse {
        open: 102.0
        high: 103.0
        low: 97.0
        close: 97.0
        volume: 4.0
}
```

Other Information

- The *sample-client/* directory provides a sample client and sample output for the example above. You probably want to create a simple client in whatever language you are using to test whether your code works as expected. It should look very similar to the python example provided.
- You are not required to write unit tests or any other forms of automated testing. However, if you choose to do so, you should definitely submit it along with your code.
- You **should not** comment every line or document every function of your code. Clear naming and code structure is mostly sufficient, however, for unintuitive portions of the

code, you can leave a brief comment.
- You do not need to authenticate or verify users.
- Assume all data received by the server will be correct (e.g. they will not request an order ID that does not exist)
- Float values only need to be accurate to 2 decimal places (floats are used for simplicity here, however they are not normally recommended for financial applications)