Design Document - Part 1

The Part 1 code contains three different folders: “Client”, “Server”, and “Util” which contain the client, server, and ThreadPool code respectively.

CLIENT :

The file “client.py” contains the client code.
In “client.py”, there is a class named ”Client”. It contains a function StartClient() which is called in order to start the client process. The arguments of StartClient() are server hostname and port which is used to connect the client to the server using a socket. These arguments are input by the user using command line arguments.

The client has a fixed number of requests that it sends sequentially. This number of requests is hard-coded into the code in the variable “max_client_requests”. The client sequentially sends a request to the server in a for-loop, which runs till the max number of requests. Inside the for-loop, the client creates and establishes a new socket connection with the server. The server hostname and port are supplied in the StartClient() function as function arguments as described above. The stock catalog in Part 1 currently contains only two stock names, "GameStart" and "FishCo".

Inside the for-loop, the client randomly selects a stock name and sends a request in bytes. The format of the client request string is: “method_name,stock_name”. The string is then encoded into bytes based on UTF-8. Part 1 only contains the method Lookup so the method name is always “Lookup” in the code. The encoded bytes are then sent to the server using send() function.

After receiving a request from the server using the recv() function, it decodes the server response from bytes to float. If the decoded price returned is “-2”, it means that the method name is invalid, that is it is not “Lookup”. If the stock name is not present in the catalog, then the response/error received is “-1”. If the error response “0” is received, it means that the trading is suspended. Error code “-1” is never returned in Part 1 as trading is not happening. If the client request is valid, the decoded server response is the stock price of type float.

The latency is calculated by using the difference between the time just before establishing the connection with the server -
“client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)”
and the time after closing the connection -
“client_socket.close()”.
The average of all <max_client_requests> latencies is calculated for each client.

SERVER :

The file “server.py” contains the server code.
In “server.py”, there is a class named ”Server”. The constructor of class “Server” takes “max_threads” as an argument. The “max_threads” or the thread pool size variable is hard-coded in the code and can be changed by changing the argument passed in the Server constructor. The Server class has a member variable “server_socket” which is used for socket connection to other clients. It contains “server_thread_pool” which is an instance of the handwritten ThreadPool “MyThreadPool”. The ThreadPool size is given by the argument “max_threads”. It contains a member variable “catalog” for storing the stock catalog. It is a dictionary with the stock name as the key and a list of stock price and trading volume. “Is_trading_suspended” is a boolean member variable that is set to False. It is set to True if the trading is suspended. This case never happens in Part 1 as trading is not happening.

The Server class contains a function ”Lookup”. It takes company_name for denoting the stock name, client_socket for denoting the client socket connection with the server, and address for denoting the client, as arguments. Inside the function, the stock name is checked in the catalog. If the stock name is not found, then “-1” is returned after encoding into bytes based on UTF-8. If the trading is suspended, then “0” is returned after encoding into bytes based on UTF-8. Otherwise, if the stock name is present, the stock price is returned after encoding into bytes based on UTF-8. The server response is sent to the client using send() function.

The Server class contains a function ”StartServer()” which is called in order to start the server process. The arguments of StartServer() are the server hostname and port which are used to create and bind the server socket to the server IP and port. These arguments are input by the user using command line arguments. The server socket binds to the hostname and port present in the arguments of StartServer(). The server then starts listening on this socket connection.

In an infinite loop, the server accepts the connections from the clients. The server receives a message from the client which is in the form of “method_name,stock_name” in bytes. The client request is decoded into the method name and stock name using decode(). Part 1 only contains the method Lookup so any other method name is considered invalid. In case of an invalid method name, the error “-2” is returned after encoding into bytes. If the method is “Lookup”, the client request is inserted into the request queue by calling the function "IssueRequest" of MyThreadPool.

THREADPOOL :

“MyThread” is a class for the threads that are used in the handwritten ThreadPool class "MyThreadPool". The “MyThread” has the "MyThreadPool" object as a member variable to denote the ThreadPool it belongs to. The function run() is executed by the thread. It executes the next client request returned by the GetNextClientRequest() function of the “MyThread” class. It thus calls the Lookup function of the Server class.

The class “MyThreadPool” is a ThreadPool used for creating and starting a fixed number of threads, which is supplied in the constructor arguments. It contains the request queue. The request queue is a list that stores all the client requests. It contains a list of threads that are created and started in the constructor. It also contains a condition lock which should be acquired in order to insert or remove items from the request queue.

The MyThreadPool class also contains GetNextClientRequest() function to get the next client request from the request queue. The client request is defined by the class “Request” which comprises the function(“Lookup”) and its arguments as member variables. For getting the next request from the request queue, the condition lock has to be acquired so that concurrent access to the request queue is synchronized and doesn’t lead to errors. After acquiring the lock, there is a while loop that checks if the request queue is empty, and if it is empty, it invokes the wait() call in order to release the lock and waits for the queue to get some items i.e. it waits for a notify() call from one of the threads which produce/insert items into the request queue. After receiving a notify() from any thread, it again acquires the lock(wait() does the acquiring of the lock) and then pops the item from the front or the 0th index. After popping the item, it calls notify() function in order to notify the producer threads that it has consumed one item from the request queue. Finally, it relinquishes the condition lock on the request queue.

The MyThreadPool class also contains IssueRequest() function to insert a client request into the request queue. The client request is defined by the class “Request” which comprises the function(“Lookup”) and its arguments as member variables. The client request is passed in the arguments of the IssueRequest() function. In the request queue on the server side, the elements are inserted at the last position and popped from the first position in a First-In-First-Out manner. For inserting a request into the request queue, the condition lock has to be acquired so that concurrent access to the request queue is synchronized and doesn’t lead to errors. After acquiring the lock, it appends the request to the request queue and issues a notify() call in order to notify the other threads that it has inserted an item into the request queue. Finally, it relinquishes the condition lock on the request queue.