The code contains three different folders: Client, Server, and Util which contains the client code, server code, and the ThreadPool code.

The client has a fixed number of requests that it sends. This number 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 establishes a new socket connection with the server. The server hostname and port are supplied in the StartClient() function as function arguments. The hostname and port are supplied using command line arguments. The catalog 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 in the format : (method_name,stock_name). Part 1 only contains the method Lookup so the method name is always “Lookup” in the code. After receiving a request from the server, 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. This never occurs in Part 1 though as trading is not happening.

The latency is calculated by using the difference between the time just before establishing the connection with the server and the time after closing the connection. The average of all latencies is calculated for each client.

The server has a socket that binds to a hostname and port which is supplied once while starting the server in the arguments of StartServer(). The hostname and port are supplied using command line arguments. While initializing, the server creates an object of the ThreadPool with a thread pool size of a fixed value. The thread pool size is hard-coded in the code and is supplied in the constructor arguments of the server.
In an infinite loop, the server accepts the connections from the server. The server receives a message from the client which is in the form of (method_name,stock_name). 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.
If the method is “Lookup”, the client request is inserted into the request queue.

The ThreadPool is 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 server contains a catalog which is a map from the stock name to a list of the stock price and trade volume. If the stock name is not present in the catalog, then the response/error sent to the client is “-1”. If the trading is suspended, the error response “0” is sent to the client. This never occurs in Part 1 though as trading is not happening. Otherwise, in the case of a valid stock name, the stock price is returned from the server to the client.

In the request queue (which is a list) 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.

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/insert2 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.