This assignment requires you to create a matching engine in C++ for a stock exchange simulation. The matching engine will accept buy and sell orders and match them based on their price and the time they were received. Here's a breakdown of the requirements and how you can approach the implementation:

**Accepting Orders:**

You need to create a data structure to store order information, which includes an order ID, type (buy or sell), price, and quantity.

Implement a mechanism to accept orders at any time. This can be done through a user interface or reading orders from a file.

**Matching Orders:**

Orders with the same price should be matched based on the time they were received. You can use data structures like priority queues or lists to manage pending orders.

When a new order is received, check if there are any existing orders with the same price. If yes, match them according to the time they arrived. If not, add the order to the **pending orders list.**

**Partial Fills:**

Your matching engine should support partial fills. This means that a single order can be matched with multiple orders, and the quantity traded can be less than the original order's quantity.

Ensure that you keep track of the remaining quantity of each order and update it as orders are matched partially.

**TCP/IP Connection:**

Your implementation should use the TCP/IP protocol for receiving orders and sending matched trades.

You will need to implement a server to listen for incoming order messages and a client to send trade messages.

**Efficiency:**

To optimize for speed, you should choose data structures and algorithms carefully. Priority queues and efficient sorting algorithms may be essential for matching orders quickly.

Consider multithreading to handle a large volume of orders and trades efficiently. Separate threads can manage incoming orders, matching, and trade execution.

**Deliverables:**

Provide the C++ source code for your matching engine.

Create a user manual that includes instructions for compiling and running your code, and detailed specifications for the order and trade message format over TCP/IP.

Develop a **test plan** that thoroughly tests your implementation, covering various scenarios such as different order types, prices, quantities, and high-speed order submission.

Include a **data generator** program that simulates high-speed order data for testing.

**Grading Criteria:**

Your implementation will be evaluated based on correctness, efficiency, TCP/IP connectivity, code quality, and testing.

Make sure your code is well-documented, structured, and follows best practices to make it readable and maintainable.

For such a complex project, it's important to break it down into smaller tasks and implement and test each component one by one. Additionally, testing and benchmarking will play a significant role in demonstrating the efficiency and correctness of your matching engine. Finally, communication between the matching engine and the TCP/IP server should be well-designed and thoroughly tested to ensure it can handle high-speed trading.