Order Management System

Author: Vipul Raj Jha
College: IIIT Delhi

Overview

This project implements an Order Management System that sends orders to an exchange. The system is designed to receive orders from various upstream systems and process them according to specified requirements, including time constraints, throttling, and order modification or cancellation.

Features

- Time Window Management: Orders are only sent to the exchange within a configurable time window (e.g., 10 AM to 1 PM IST). Outside this window, orders are rejected.
- Throttling Mechanism: Limits the number of orders sent to the exchange to a maximum number per second (configurable). Excess orders are queued and sent when permissible.
- Order Queue Management:
  - Modify Requests: If a modify request is received for an order in the queue, the price and quantity are updated.
  - Cancel Requests: If a cancel request is received for an order in the queue, it is removed from the queue.
- Response Handling: Matches responses from the exchange with sent orders and records the response type, order ID, and round-trip latency in persistent storage.
- Concurrency: Utilizes multithreading for handling incoming orders and communication with the exchange.

File Structure

- `OrderManagement.cpp`: The main C++ source file containing the implementation of the `OrderManagement` class and associated functionality.
- `order_responses.txt`: Output file where order responses with latency information are stored.
- `README.md`: This readme file providing an overview and instructions.

Dependencies

- Standard C++ libraries (`<iostream>`, `<thread>`, `<mutex>`, `<deque>`, etc.).
- C++11 or later compiler support (for threading and chrono libraries).

Compilation Instructions

To compile the `OrderManagement.cpp` file, you need a C++ compiler that supports C++11 or later standards.

Using g++ (GCC Compiler)

Open a terminal or command prompt and navigate to the directory containing the `OrderManagement.cpp` file.

BASH

```
g++ -std=c++11 -o OrderManagement OrderManagement.cpp
```

This command compiles the code and outputs an executable named `OrderManagement`.

Using Visual Studio (Windows)

1. Open Visual Studio.
2. Create a new Console Application project.
3. Add the `OrderManagement.cpp` file to the project.
4. Set the C++ language standard to C++11 or later in project settings.
5. Build the project.

Execution Instructions

After compiling, run the executable. The system is designed to simulate receiving and processing orders. Since the code does not include the implementation for receiving real-time data or sending actual requests to an exchange, you may need to simulate inputs.

Simulating Inputs

You can modify the `main` function (not provided in the original code) to create instances of `OrderRequest` and `OrderResponse` and call `onData` methods.

Example:

C++

Collapse

```cpp
int main()
{
    OrderManagement om;

    OrderRequest order1 = {1, 100.5f, 10, 'B', 1001, RequestType::New};
    om.onData(std::move(order1));

    // Simulate a response after some time
    std::this_thread::sleep_for(std::chrono::milliseconds(500));

     OrderResponse response1 = {1001, ResponseType::Accept};
     om.onData(std::move(response1));

     // Add more orders and responses as needed
     return 0;

}
```

Assumptions and Design Decisions

- Data Types:
    - Order IDs and quantities are represented using 32-bit unsigned integers (`uint32_t`).
    - Symbol IDs are 32-bit integers (`int32_t`).
    - Prices are represented using 32-bit floats (`float`), assuming sufficient precision.
- Time Management:
    - System uses `std::chrono` for time-related functions.
    - The allowed time window is specified in seconds since midnight for simplicity.
- Threading Model:
    - A dedicated sender thread handles communication with the exchange.
    - Mutexes and condition variables manage synchronization between threads.
- Order Processing:
    - Orders received outside the allowed time window are rejected.
    - The system enforces a per-second throttle limit on orders sent to the exchange.
    - Modify and Cancel requests are handled according to whether the order is in the queue.
- Persistent Storage:
    - Order responses with latency information are written to `order_responses.txt`.
    - Simple file I/O is used for persistent storage.
- Logging:
    - Console output (`std::cout`) is used for logging activities and tracing execution flow.
- Simplifications:
    - Actual sending and receiving of orders to/from an exchange are simulated with placeholder methods.
    - No external libraries or third-party dependencies are used.
    - Error handling and exception management are minimal.
- System Limitations:
    - The implementation assumes a reliable system without the need for fault tolerance mechanisms.
    - The code assumes system clock accuracy and does not handle clock drift or time synchronization issues.

Code Structure

Classes and Structs

- OrderRequest:

    - Represents an order request with necessary details such as symbol ID, price, quantity, side, order ID, and request type.
    - Includes a timestamp indicating when the order was received.
- OrderResponse:
    - Represents a response from the exchange containing the order ID and response type.
    - Includes a timestamp indicating when the response was received.
- OrderManagement:
    - Main class handling order processing, including receiving orders (`onData`), sending orders (`send`), handling responses, and managing the order queue.
    - Contains private helper methods for time management and sender thread functionality.

Key Methods

- onData(OrderRequest && request):
    - Processes incoming order requests.
    - Handles new orders, modify requests, and cancel requests according to the specified logic.
- onData(OrderResponse && response):
    - Processes responses from the exchange.
    - Calculates and records the latency for each order.
- send(const OrderRequest& request):
    - Simulates sending an order to the exchange.
- sendLogon() and sendLogout():
    - Simulate sending logon and logout messages to the exchange.
- senderThreadFunction():
    - Dedicated thread function that manages sending orders to the exchange according to the time window and throttle limit.

Testing

To thoroughly test the system:

1. Time Window Testing:
    - Attempt to send orders before and after the allowed time window to ensure they are rejected.
2. Throttling Testing:
    - Send more orders than the throttle limit within the same second to confirm that excess orders are queued.

3. Order Modification and Cancellation:
    - Send modify and cancel requests for orders in the queue and verify that they are processed correctly.
4. Latency Recording:
    - Check the `order_responses.txt` file to ensure that responses are recorded with correct latency calculations.
5. Concurrent Execution:
    - Simulate receiving multiple orders from different threads to test thread safety and synchronization.

Extensions and Improvements

- Error Handling:
    - Implement comprehensive error checking and exception handling.
- Configuration Files:
    - Allow configuration parameters (e.g., time window, throttle limit) to be set via external files or command-line arguments.
- Logging Framework:
    - Integrate a logging library for more robust logging capabilities.
- Database Integration:
    - Store order responses and latency information in a database instead of a flat file.
- Unit Tests:
    - Develop unit tests for each component using a testing framework like Google Test.

Contact

For any questions or further information, please contact:

Vipul Raj Jha
Email:vipul21435@iiitd.ac.in
College: IIIT Delhi