Order Management System (OMS) Documentation

VIGHNESH M S
232104@iiitt.ac.in
8088012208

Assumptions

1. **Order Time Window**:
   - Orders can only be processed within a defined time window (e.g., 10:00 AM to 1:00 PM). Orders outside this window are rejected.
2. **Order Processing Rate**:
   - The system can process a maximum of `max_orders_per_second` orders per second.
   - Orders are processed in a **First-In-First-Out (FIFO)** manner.
3. **Order Modification**:
   - If an order with the same `order_id` already exists in the queue, it is modified instead of being added as a new order.
   - Modifications apply only to price (`m_price`) and quantity (`m_qty`).
4. **Order Cancellation**:
   - If a request is received with `m_price = 0` and `m_qty = 0`, the corresponding order is removed from the queue.
5. **Concurrency Handling**:
   - Orders are queued using a thread-safe `deque`.
   - A lock is used (`order_lock`) to prevent race conditions while modifying or accessing the queue.
6. **Order Response Handling**:
   - Responses received from the exchange contain an order ID and a response type (Accept or Reject).
   - Latency between sending an order and receiving a response is measured.

Design Decisions and Architecture

1. Class Structure

- **Order Management System (OMS)** follows an event-driven architecture where orders are received, queued, processed, and responded to asynchronously.
- Core classes include:
    - `OrderManagement`: Manages order queuing, processing, and response handling.
    - `OrderRequest`: Represents an order with attributes like price, quantity, side (buy/sell), and order ID.
    - `OrderResponse`: Stores exchange responses.
    - `RequestType`: Defines different types of order actions (New, Modify, Cancel).
    - `ResponseType`: Represents exchange responses (Accept or Reject).

2. Multithreading for Order Processing

- A separate processing thread runs in the background to dequeue and send orders at a controlled rate.
- The `stop_event` flag ensures graceful shutdown of the processing thread when required.

3. Concurrency Control

- `order_lock` is used to synchronize access to `order_queue`, preventing race conditions in a multithreaded environment.

4. Handling Order Responses

- Responses are logged along with their latency.
- The system ensures only known orders (previously sent) receive responses.

5. Unit Testing and Validation

- The implementation includes test cases to validate:
    - Order queuing (`test_order_queuing`)
    - Order modification (`test_order_modification`)
    - Order cancellation (`test_order_cancellation`)
    - Order rejection outside allowed time window (`test_order_rejection_outside_time_window`)
    - Response handling (`test_order_response_processing`)
- The `unittest` framework is used to automate test execution.
