FIQ - File Based Message Queue

Design and implement FiQ; a simple file store acting as a messaging queue that holds messages in an optimal structure for sequential reads and writes. There can be simultaneous read and write operations on the FiQ which it should be able to support without any data loss at the reading end. Reader(s) must receive all data that was written by the writer(s).

Note: Focus on core implementation of file based queue and producer consumer operations. Structuring the implementation well is more important than finishing all parts of the problem. You are free to start with the implementation before starting the interview.

Design and Implement the following, solving iteratively:

1. Design a simple file backed message store along with a producer system wherein the producer thread/process writes messages to the store.
    a. The FiQ should acknowledge and persist the messages once written in the structure you choose.

    b. For simplicity, following anatomy of a message can be assumed. A set of messages can be hardcoded to be produced in a loop from producer logic to start with.
    `<key>:<message>:<processing_time>`

    ```
    Key is a unique alphanumeric string
    Message is an alphanumeric string
    Processing_time is time in seconds
    ```

    c. Producer spawns a thread/process to write message objects adhering to structure defined above. Once successfully written to file FiQ must acknowledge back to the producer (writer)

2. Design a consumer system that spawns a thread/process to read messages stored in the FiQ sequentially and assigns to a pool of concurrent workers for processing of the message.
    a. Concurrency factor or number of workers can be accepted as an argument while initializing the consumer.

    b. For simplicity, processing would simply mean a sleep of <processing_time> seconds from the message. Worker threads/processes should sleep for the mentioned time while processing the particular message.

    c. Solve for concurrent consume operation while the producer is still continuously writing to the FiQ without message loss at the consumer's end.

3. Implement a CLI interface wrapping the produce and consume operation modularly. For simplicity this could live in a common code base as modes of execution.

    a. *Producer CLI:* The CLI should accept user prompts to enqueue messages until the user decides to quit the session using SIGINT or types exit
    Messages should be parsed and forwarded to the producer interface to be written to disk. Corresponding acknowledgement should also be printed on the same CLI session.

    b. Consumer CLI: The CLI should accept a concurrency factor as input to initialize and start the consumer system. CLI should start to print the messages once they are processed by the workers on the terminal session until user decides to quit the session using SIGINT or types exit

---