Order Book Programming Exercise

Produce a program which accepts orders and cancels from a udp port, maintains multiple price-time order books one per symbol, publishes acknowledgements, trades and top of book changes.

Requirements

Input

Interface directly with sockets if you wish, boost asio is the highest level of abstraction accepted. Maintain a separate thread to buffer input messages.

Test Input

See provided input file inputFile.csv for format, supporting three types of transactions: new order 'N', cancel order 'C' and flush 'F' orderbook.

Options:

- Write your own file reader and streamer client program in any language of choice (C++, python etc), stream to local udp address and port.
- Use netcat eg:

  cat inputfile.csv | netcat -u 127.0.0.1 1234

Tip: strip the input file of blank lines and comments prior to streaming

Bonus: if writing own client program, use a codec for wire protocol (eg google protobufs)

Order Book Processing

Order book is price-time:

- market orders take the opposite side immediately, unmatched portion assume cancelled (fill and kill)
- limit orders (if not matched immediately) join the book in time priority
- partial quantity matches are possible

Publish on separate thread:

- to console/stdout
- publish order or cancel acknowledgement format:

A, userId, userOrderId

- publish trades (matched orders) format:

  T, userIdBuy, userOrderIdBuy, userIdSell, userOrderIdSell, price, quantity

- publish changes in Top Of Book per side using format, use '-' for side elimination:

  B, side (B or S), price, totalQuantity

Bonus: create unit tests around the book interface, shortcut: convert input scenarios to unit tests, provide more scenarios time permitting.

Bonus: create end to end tests testing performance, measure ingest through orderack or trade publish, provide mean and standard deviation in usec

Test Outputs

Outputs are provided for odd scenarios in outputFile.csv, its expected that you generate your own expected outputs for even numbered scenarios and validate your output against it (strip comments and blank lines out).

Project

Use make at a minimum, use gcc/g++ 6.0+ and boost 1.58+.

Tar, gzip and submit. Please do not include shared libraries, object files or executables.

Provide a readme.txt file describing how to build, run.

Bonus: provide documentation, project structure, architectural aspects, threads, classes etc. Include improvements you would make if you had more time.

Bonus: containerize and provide docker image, provide instructions to build and run via docker.

Important Note: Please do not submit any code that is derived from proprietary code/or code you worked on for another company previously.

Further tip: due to time constraints, please prioritize bonuses as follows:

- ***unit tests***
- ***performance tests***
- ***documentation***
- ***containerization***
- ***binary wire protocol***