Assignment 2: Exchange matching engine in Go

In this assignment, you will implement an exchange matching engine in Go, using channels and goroutines. The functionality of the matching engine is the same as the one presented in CS3211 Assignment 1, with minor clarifications.

1 Clarifications on the functionality of the matching engine

Start by reading again the functionality of the matching engine described in Assignment 1 writeup.

1.1 Price-time priority

Your exchange should match orders using the *price-time* priority rule. This rule for matching two orders on an exchange is expressed using the following conditions – which must all be true for the matching to happen:

- The *side* of the two orders must be different (i.e. a buy order must match against a sell order or vice versa).
- The *instrument* of the two orders must be the same (i.e. an order for "GOOG" must match against another order for "GOOG").
- The *size* of the two orders must be greater than zero.
- The *price* of the buy order must be *greater or equal* to the price of the sell order.
- In case multiple orders can be matched, the order with the earliest completion time (`Timestamp completed` from the output generated by adding resting order) will be matched first. This is to ensure fairness towards the orders that have waited the longest.
- New buy orders spend the least amount of money, while new sell orders obtain the most amount of money.

Orders can be partially matched in case the size of the other order is lower. Consequently, orders can be matched multiple times.

▶ **Example 4: Price-time priority**

1.2 Cancel requests

A Cancel request for a specific order ID may come only from the client that sent that order ID. Cancel requests for a non-existing order ID may come from any client.

1.3 Processing order and arrival order

The orders are received from multiple clients at the matching engine in some order (arrival order). The order is considered `active` between the time it is received and the time when its processing ends (and it is added to the order book or executed). The output timestamps prints (`Timestamp completed`) should reflect the processing order.

Due to concurrent processing, the order in which the requests arrive at the engine and the requests are processed might not match. When two orders are active at the same time, any of them might be processed first and their partial matching can be interleaved.

There might be cases where matching sell and buy orders are active at the same time, while none of the resting orders are matching with those active orders. The expected behavior is for one (any) of the active orders to be added to the book first (it has a smaller `Timestamp completed` in the output), and the other one to match it.

There might be cases where cancel requests for an order ID and the order ID are active at the same time. In that case, any of the following actions is acceptable in handling the Cancel request:

- rejected if the Cancel is processed before the order gets added to the book (cancel has a smaller `Timestamp completed` in the output); in this case, the cancel does not affect the buy order in any way. Note that cancel orders are not added to the order book, and they are rejected if they cancel an invalid order id.
- accepted if the Cancel is processed AFTER the buy order gets added to the book.

When checking correctness for your matching engine, ensure that output lines sorted by `Timestamp completed` corresponds to a valid interleaving of the matching for the provided inputs.

2 Requirements

Unlike Assignment 1 where any synchronization primitive and the usage of lock-free programming was allowed, Assignment 2 has a strict requirement on what type of constructs you are allowed to use:

- **You must implement the assignment only using goroutines, channels, and channel peripherals like `select`.**
- **You should have at least one goroutine per client AND at least one goroutine per instrument in the order book.**
- You are not allowed to use any utilities from `sync` package in Go (mutex, condition variable, pools are not allowed)
- You are not be allowed to use any shared data structure other than channels (buffered or unbuffered)

Same as in Assignment 1, you are required to implement in Go the concurrent matching in the matching engine.

2.1 Provided skeleton

The following files are provided.

Files you will likely need to modify only:

- `engine.go`: An empty `engine` implementation that creates a new goroutine when a connection is received, and then simply read commands and dumps them to standard output. You will likely need to modify `handleConn`.

Files you should not need to modify:

- `main.go`: Main function.
- `io.h`: Shared declarations for I/O
- `client.c`: Same as assignment 1
- `Makefile`: Same as assignment 1, except all `*.go` files should be compiled when running `make engine`

You can modify the `Makefile` to exclude some Go files from compilation if needed.

2.2 Submission

Your program must be written in Go 1.17. You have to rely on the Go standard library for this task. Any `go.mod` files will be ignored.

Your program must work on the lab machines provided in the Parallel Computing Lab at COM1-B1-02, which are running Ubuntu 20.04 (details about how to connect and use those computers can be found in the assignment 1 writeup).

With your submission, include a writeup as a PDF of up to two A4 pages including the following:

- an explanation about how your usage of channels and goroutines enables the concurrency. This description is essential for our grading, as such, it should match your implementation and help us understand.
- an explanation of how you support the concurrent execution of orders coming from multiple parallel clients.
- a description of any Go patterns used in your implementation.
- a description of your data structure(s) used to keep track of the orders.
- a description of how you tested your code with multiple clients and orders. Feel free to add any scripts or test case file(s) to your submission, under folder `scripts`

You are advised to work in groups of two students for this assignment (but you are allowed to work independently as well). You may discuss the assignment with others but in the case of plagiarism, both parties will be severely penalized. Cite your references or at least mention them in your report (what you referenced, where it came from, how much you referenced, etc.).

Submit your assignment before the deadline under LumiNUS Files. Each group must submit ONE zip archive named with your student number(s) (`A0123456Z.zip` if you worked by yourself, or `A0123456Z_A0173456T.zip` if you worked with another student) containing the following files and folders.

Your zip archive must contain **only** the following:

- all source and header files that comprise your program
  - Please include files from the skeleton that are required by your program, even if you do not modify them.
- a `Makefile` whose default target produces an `engine` binary conforming to the requirements described above
  - The provided `Makefile` conforms to this.
- the 2-page writeup as a PDF named `A0123456Z_report.pdf` or `A0123456Z_A0173456T_report.pdf`
- an optional folder, named `scripts`, containing any additional scripts and testcases that you might have used for the assignment.

Do not include compiled binaries or object files in your submission archive.

It is required to use a GitHub repository for your team. Access **GitHub Classroom** at this link to create such a repository. Name your team `A0123456Z` (if you work by yourself) or `A0123456Z_A0173456T` (if you work with another student).

Submit your zip archive to the **Assignment 2 Submissions** folder before **Fri, 25 Mar, 2pm**.

All FAQs for this assignment will be answered here. If there are any questions you have that are not covered by the FAQ document, please post on the LumiNUS Forum.

Note that for submissions made as a group, only the most recent submission (from any of the students) will be graded, and both students receive that grade.

A penalty of 5% per day (out of your grade for this assignment) will be applied for late submissions.