Name : Brian Kheng

Student ID : A0290248B (e1324741)

Usage of channels and goroutines

There will be one daemon goroutine when the engine is initialized. Every client input will pass through the daemon goroutine using the daemon channel. The daemon goroutine will then initialize a goroutine and buffered channel for each instrument, then assign the input to the corresponding instrument goroutine via instrument channel. Because of the instrument buffered channel, the daemon goroutine doesn't need to wait the instrument goroutine to be able to receive the input before the daemon goroutine can continue with the next input request, this allows us to achieve an instrument-level of concurrency.

Concurrent execution of orders from multiple clients

Each order request from multiple clients will concurrently pass through the daemon goroutine and will be assigned to the corresponding instrument goroutine and allow us to achieve the instrument-level of concurrency.

Go patterns

1. Fan-out, fan-in pattern
   Each of the client request will first fan-in to a single daemon goroutine, which then fan-out to each of the corresponding instrument goroutine.
2. Semaphore
   To ensure each client request is processed sequentially, we need to use channel as a mutex before we can process the client request.

Data structures

1. Hash Map
   Mapping between instrument_name with instrument_channel and instrument_id with instrument_channel.
2. Priority Queue
   Orderbook uses this data structure and sets the priority based on the price and timestamp.

Testing Methodology

I used my hand-written test cases that cover 40+ clients, race condition, data races, and using the grader provided to check whether my program was able to pass the given test cases.