Design

First of all, main thread take all information about simulation from input file. Create customers and sellers. After that, starts simulation.

Customer threads briefly do that, for given simulation days, if day did not end yet he needs to pass five stages to reach seller.

1. He should not reach his operation limit.
2. Generate a random operation. If the operation is cancel operation but the customer does not have a reservation. So, generate random operation until it is not cancel.
3. Customer should be availability for sale. (he should wait his sale if there is any)
4. Day should not be ended.
5. He should lock any seller.

Between stage 3 and stage 4 for customers who consume his all reservation right, if random operation is reservation operation change it to buy operation.

Customers pass their id to seller via seller_work[seller_id] = thread_id, and operation, product_id, product number via job transaction 2d array.

job_transaction[customer_id][0] = operation
job_transaction[customer_id][1] = product id
job_transaction[customer_id][2] = product number

Seller threads briefly do that, for given simulation days, if day did not end yet he waits for a work OR work has come and day end main thread wait seller to finish his final work. Also, seller thread writes his transaction to log file.

Main thread, first takes informations from input file do some initializations and fill necessary ones. Manages simulation for given day.

How main thread manages it?

- Send a signal for simulation begin.
- Sleeps (for a reasonable second)
- After wake up send a signal to customers to reach their day end line.
- After customers reach it send another signal to sellers to finish their last work and reach their day end line.
- Print transactions which sent by customers and also print transactions which received by sellers.
- Print for each product how many bought, how many reserved and how many reserve canceled.
- Re_initialize necessary variables.
- Go next day and begin day.

What we used?

- barrier_customer_count and its mutex_lock. This variable condition variable for main thread can understand whether all customers reach their point or not.
- barrier_seller_count and its mutex_lock. Do same for sellers.
- seller_transaction and customer_transaction keeps total number of transaction sent by customer, received by seller respectively. Also, each variable has their mutex_lock.
- seller_mutex array for each seller.
- product_mutex array for each product.
- used_product keeps for each product how many bought, reserved, and canceled. Also each has their mutex_lock.
- customer_information keeps each customer’s operation limit and reservation limit.

- current_information keeps current informations.
- customer_availability   0 (if customer does not work with any seller) 1 otherwise.
- day_over_customer condition variable to customer understands whether day end or not.
- day_over_seller seller version of above variable.
- reservations keeps all reservations in a day.
- customer_reservations keeps number of reservation done for each customer.