Laboratory 1 - Parallel and Distributed Programming

Hardware specifications:

- 8 cores
- 16GB RAM

Domain:

Product – contains the name, sold quantity, quantity, unit price and a lock which is mutually exclusive.

Bills – contains a list of products, the total money from the sales, and its number.

Plan:

Each domain entity will have a mutex which will allow whether or not any write operations can be performed on the entity. It will serve when modifying its quantity when processing a bill.

The Transactions class will have two locks, one meant to prevent writing but allow multiple threads to read:

Eg: a bill is processed and the same product it contains is searched in the list of all products to modify to substract form its quantity the sold quantity from the bill; but the products which aren't found in the product list do not need to be locked and can be accessed by other threads;

Or require a write operation to update the total money obtained, which is much more rare, once per Bill entity.

When a certain number of bills have been processed, a thread meant to check the status of the inventory (total money from processed bills and whether the sold quantity of the product equals that of the product from the processed bills).

Threads:

ProcessThread – calls the processBill function for each bill from the sublist it was given as a parameter;

CheckIfInventoryIsValidThread – calls the checkEverything function whenever a certain nr of Bills have been processed and checks if the total money from the processed bills equal that of the Transactions variable which stores the expected total money sum.

Problems:

-heavy read operations due to the methods requiring continuous checking of the Bills status, and subsequently the recorded Products

-less frequent write operations, that influence the whole domain which in turn, have two cases:

-modifying the value of a Product entry

-modifying the value of Transactions class variables

-The greater the number of threads, the more locks would be set. This slows down the program considerably (differences ranging from 1.9 seconds to 3.5 seconds when using 100 threads compared to 10 threads)

Test Cases:

1. *The following tests were performed on 5000 products, 1000 bills and 100 threads, and have a 10 ms delay for each Checking thread*

Using no read lock nor write lock for the processing a bill function:

High chance of deadlock. 1.5965295 seconds, in order

Using no write lock in the processing bill function:

1.5796824 seconds, in order

Using no read lock in the processing bill function:

1.5766186 seconds, in order

Using read lock and write lock for the processing bill function:

1.5883721, in order

2. *The following tests were performed on 5000 products, 1000 bills and 100 threads, and have a NO delay for each Checking thread*

Using no read lock nor write lock for the processing a bill function:

High chances of causing a deadlock (if check function has no lock), 0.3730351 otherwise

Using no write lock in the processing bill function:

High chances of causing a deadlock (if check function has no lock), 0.2545668 otherwise

Using no read lock in the processing bill function:

High chance of causing deadlock, 0.3194871, not performed in order

3. * The following tests were performed on 5000 products, 1000 bills and 100 threads, and domain entities have no locks*

Using read lock and write lock for the processing bill function:

0.2912444, not performed in order

General observations:

-check function requires a lock which stops all access into the function, otherwise there are concurrency issues

-for any small Bill records (less than 100), the execution time with strict locking is under 0.08 seconds (including printing)

-When running the program while the laptop is set to save battery, the performance of the threads is worse (around a 0.3 difference in seconds at max when testing with 5000 products, 1000 bills and 100 threads)