**Weicheng Shi, Yue Tu**
**Prof. Brian Rogers**
**ECE 568 ERSS**
**04/03/2019**

# ERSS Homework 4 Report

Our Project is primarily divided into four parts. Server, XmlParser, Database Handler, and Responder.

1. Server provides structures for listening to all incoming connections, receiving xml data and responding to client.
2. XmlParser takes TiXmlDocument type element and parse it to create and transaction objects corresponding to create xml and transaction xml respectively.
3. Database Handler takes create or transaction objects as input, and generate relative sql insert, select and other database manipulation commands. Database Handler also generate TiXmlDocument type response.
4. Responder takes response, and send it to clients using socket.

When We started project, we looked into multiple open source libraries for parsing xml files and extracting useful elements. In the end, we decided to use tinyxml as a tool for desired operation, as tinyxml provides significant amount of methods followed with detailed documentation that is easy to understand.

## Functionality Testing:

1. Objective: Check if input xml files can be parsed into corresponding elements.

Result: parsed element acquired from parsing method has string type, but object generated takes int and double type. The wrong conversion leads to unexpected resulting value. Also, there is an endless while loop in the parsing method, which freezes out program.

Solution: (a). Implement desired type conversion logic.

(b). We realized that in one of the while loop which check the null status of curr, there is no curr=curr->NextSiblingElement() to break the loop, and this modification has the problem resolved.

2. Objective: Check if database can be updated using object generated by parsing method.

   Result: We received segmentation fault when our program tries to process generated object to create SQL command.

   Solution: (a). We added some test print methods to check if the input entering SQL generating method exited, and the result printed was empty. We realized that our program assumes object created from parsing method is not null, and it tries to access pqxx::result[0][0] which does not exist.

   (b). Then We looked into the Create.h and Transaction.h files Where we define all elements to be passed to create SQL commands, and we realized that there were no constructors for subclass such as order, query and cancel. So, we initialized the constructor with default value, and this resolved the problem we encountered.

3. Problem Encountered: When we divided rooted SQL command into two, and tried to commit changes to database, we received error message: what() started. transaction<read committed> while non-transaction still active. We encountered similar problems when we tried to use Work(Connection *C).

   Explanation: We realized that one non transaction has to end before we call another similar command, and we cannot execute two SQL commands without any commit in between.

   Solution: We created SQL execution method which takes SQL string. It has its own work or non-transaction defined in the beginning of the method, and commit in the end. As a result, non-transaction and work will be destructed every time it leaves scope of SQL execution method, and there will not be any alive non-transaction when we access the database.

4. Problem Encountered: We first defined Database Handler after spawning new thread. Although we did not encounter message, but it provides unexpected output. We create a new database every time server spawns a new thread that sends request to server for create/transaction behavior.

   Explanation: In the Database Handler, there is drop table, and create table. Since this handler method is inside server which spawns threads, if a new thread is generated,

the current table will be dropped, and a new table will be created. This is not dangerous behavior for database, as no data can be stored.

Solution: We move Database Handler out of spawning threads section. As a result, we will only drop and create a new.

5. Problem Encountered: When we tried to add non-existent symbol to user's position, we received segmentation fault.

   Explanation: When we tried to pass create symbols xml to server as client, if there existed no such symbol, we were supposed to create a new symbol as users' positions. However, we forgot to create add symbol method. We directly tried to access res[0][0] from result table, but there was no returned table.

   Solution: We create a add symbol logic and place it in the desired position.

6. Problem Encountered: When create xml inputs value greater than 1000000, the value is changed to 2e+06. This value was hard to evaluate when we tried to transit between int and string.

   Solution: We assumed value of fund to be less than 10^6 for ease of use.

## More Test Cases:

1. Procedure: Follow the steps from hw4 instructions, create sell and buy orders same as graph to the left

| Buy | Sell |
|---|---|
| | 5: 200 @ $140 |
| | 2: 100 @ $130 |
| | 4: 500 @ $128 |
| 1: 100 @ $125 | |
| 6: 400 @ $125 | |

| Executed | |
|---|---|
| 3: 200 @ $127 | 7: 200 @ $127 |
| 1: 200 @ $125 | 7: 200 @ $125 |

```
<?xml version=" 1.0 " encoding=" UTF - 8 "?>
<results>
    <opened sym="switch" amount="-400" limit="124" id="7" />
    <status id="1">
        <open shares="100" />
        <executed shares="200" price="125" time="APR-03-2019 09:45PM" />
    </status>
    <status id="2">
        <open shares="-100" />
    </status>
    <status id="3">
        <executed shares="200" price="127" time="APR-03-2019 09:45PM" />
    </status>
    <status id="4">
        <open shares="-500" />
    </status>
    <status id="5">
        <open shares="-200" />
    </status>
    <status id="6">
        <open shares="400" />
    </status>
    <status id="7">
        <executed shares="-200" price="127" time="APR-03-2019 09:45PM" />
        <executed shares="-200" price="125" time="APR-03-2019 09:45PM" />
    </status>
</results>
```

Expected Result: There will be two executed orders for user 7 each sells 200 stocks. There will also be 200 buy orders from player 1 and 200 buy orders from player 3.

Actual Result: We ran 7 querys, there were only three users with executed orders: user 1, user 3 and user 7 with 200 buy orders, 200 buy orders and 400 sell orders respectively.

2. Transmit the following xml documents to server.

C1.xml:

```
<create>
 <account id="1" balance="500"/>
 <account id="2" balance="500"/>
 <symbol sym = "switch">
   <account id="2">50</account>
 </symbol>
</create>
```

t1.xml

```
<transactions id="1">
 <order sym="switch" amount="-50" limit="10"/> #Account 1 does not have enough "switch", should return error
 <order sym="switch" amount="60" limit="10"/> # Account 1 does not have enough money, return error
 <order sym="switch" amount="20" limit="10"/> #Post the order, trans_id should be 1, now balance in account 1 is 500 - 20 * 10 = 300
 <order sym="switch" amount="50" limit="10"/> #Account 1 does not have enough money, return error
 <cancel id="3" /> #Trans_id 3 does not exist
 <query id="1" /> #One open order: amount 20, limit 10
</transactions>
```

t2.xml

```
<transactions id="2">
  <order sym="switch" amount="-10" limit="11.0"/> #Post the order, trans id is 2, now account 2 has 40 "switch" left in account
  <order sym="switch" amount="-40" limit="9.0"/> #Post the order, trans id is 3; now account 2 has no "switch" left in account; should executed this order with an amount of 20, price of 10; account 2 now has 500 + 20 * 10 = 700 balance
```

<query id="1" /> #Executed order: shares 20, price 10

<query id="2" /> #Open order: amount -10, limit 11.

<query id="3" />2.open order: amount -10, limit 9.  3.executed order: shares -20, price 10.

</transactions>

t3.xml

<transactions id="1">

<order sym="switch" amount="20" limit="12.0"/> #Post the order, trans id is 4; now account 1 has 300 - 20 * 12 = 60 balance left; should first match with trans_id 3: shares 10, price 9; then match with trans_id 2: shares 10, price 11. So money returned to account 1 is 10 *(12 - 11) + 10 * (12 - 9) = 40, so account 1 noe has 60 + 40 = 100 balance; account 2 has 700 + 10 * 11 + 10 * 9 = 900 balance

<query id="1" /> #No change

<query id="2" /> #Executed: shares -10, price 11

<query id="3" /> #1.open: amount: -10, limit 9.  2.Executed: shares -10, price 9

<query id="4" /> #1.executed: shares 10, price 11.  2.Executed: shares 10, price 9

</transactions>

t4.xml

<transactions id="2">

<order sym="switch" amount="-20" limit="12.0"/> #Error: not enough items in account

<cancnel id="3" /> #Error. No open order for trans_id 3

<order sym="switch" amount="-10" limit="12.0"/> #Post the order: trans id is 5.

<order sym="bitcoin" amount="-10" limit="12.0"/> #Error: symbol "bitcoin" does not exist

<query id="1" /> #...

<query id="2" /> #...

<query id="3" /> #...

<query id="4" /> #...

<query id="5" /> #...

</transactions>

Expected Result:

Account 1: 100

Account 2: 900

Actual Result:

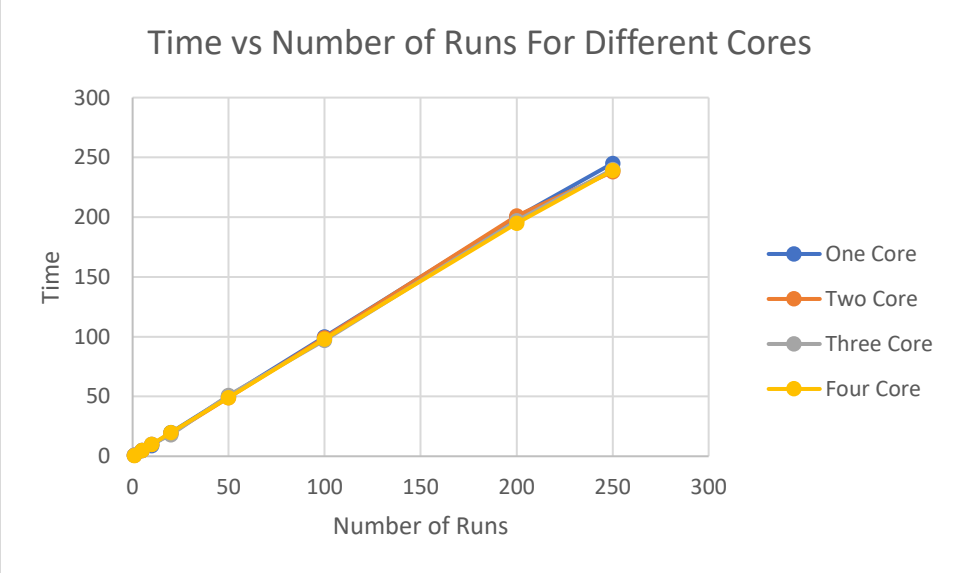
account_id: 1 balance: 100

account_id: 2 balance: 900

Actual Result matches with Expected Results.

## Scalability Test:

We ran testing from one core to four cores, from one transaction to 800 transactions. Multiple sets of data were recorded and we took average of these data, and generated the following graph that compares four cores performance.



**Analysis:**

From the graph, we can tell that number of cores will not influence the speed of program running significantly. In our test, time of programing is proportional to number of runs, because we implement our database operations in a serialized approach.

The max number of transactions that our server can handle is around 300. If we exceed 300 calls, we will receive connection error. After Investigation, we realized listen only allows wait size of 128, which means if our database operation is slow, the maximum wait size of listen will be exceeded. This situation will cause the further connection to be denied.

## Other Problem Encountered:

We also spent hours trying to create docker container for this project. However, we always encountered some unsolvable issues. So we decided submit our project without docker.