CIS427 Fall 2022 **Programming Assignment 1**

Online Crypto Currency Trading

Due 10:00pm Sunday, October 2, 2022

1. Introduction

This programming assignment is designed to introduce you to the socket interface and client-server applications. Minimum knowledge in database is needed including insert, update and delete is needed. If no previous background in databases, please talk to the instructor. This assignment weights 10 % of your final grade.

For this assignment, you will design and implement an online crypto currency trading application using network sockets. You will write both the client and server portions of this application. The client and server processes will communicate using **TCP** sockets and will implement the protocol discussed below.

2. The Assignment

You will write two programs, a server and a client. The server creates a socket in the Internet domain bound to port SERVER_PORT (a constant you should define in both programs, you may use last 4 digits of your UM-ID to get a unique port number). The server receives requests through this socket, acts on those requests, and returns the results to the requester. The client will also create a socket in the Internet domain, send requests to the SERVER_PORT of a computer specified on the **command-line**, and receive responses through this socket from a server.

For this assignment, you just need to allow one active client connect to the server. In the next assignment, you should allow multiple clients connect the server at the same time with multiple users.

Your client and server should operate as follows. Your server begins execution by opening a SqLite database file (which is uploaded on Canvas along with the schema) and that contains two tables(initially empty): One table that holds crypto records and the other table holds user records including the balance in USD. A good introduction and sample code to work with SQLite could be found here: https://www.tutorialspoint.com/sqlite/sqlite_c_cpp.htm
Each User record in the users table should have the following fields:

- ID int NOT NULL AUTO_INCREMENT,
- email varchar(255) NOT NULL,
- first_name varchar(255),
- last_name varchar(255),
- user_name varchar(255) NOT NULL,
- password varchar(255),
- usd_balance DOUBLE NOT NULL,

the ID field is the primary key. Your code should check if there are any users, if there is no users, you need to create a user record. The following code can be used to create the Users table:

```
create table if not exists Users
           (
            ID int NOT NULL AUTO_INCREMENT,
            first_name varchar(255),
            last_name varchar(255),
            user_name varchar(255) NOT NULL,
            password varchar(255),
            usd_balance DOUBLE NOT NULL,
            PRIMARY KEY (ID)
            );
```

Each Crypto record in the Cryptos table should have the following fields:

- ID int NOT NULL AUTO_INCREMENT,
- crypto_name varchar(10) NOT NULL,
- crypto_balance DOUBLE,
- user_id varchar(255),

The following code can be used to create the Cryptos table:

```
create table if not exists Cryptos
           (
            ID int NOT NULL AUTO_INCREMENT,
            crypto_name varchar(10) NOT NULL,
            crypto_balance DOUBLE,
            user_id int,
            PRIMARY KEY (ID),
            FOREIGN KEY (user_id) REFERENCES Users ID)
            );
```

Only your server should create, read, write, update, or delete records into/from the database. Once the server has started, it should check if there is at least one user, if there are no users the server should create one user manually or using code with, say, $100 balance. Then the server should wait for the connection requests from a client.

Your client operates by sending one of the following commands: BUY, SELL, BALANCE, LIST, SHUDOWN, QUIT commands to the server. You should create a client that is able to send any of the commands above and allows a user to specify which of the commands the client should send to the server.

The details of the protocol depend on the command the client sends to the server.

BUY

Buy an amount of crypto. A client sends the ASCII string “BUY” followed by a space, followed by a Crypto_Name, followed by a space, followed by a Crypto_amount, followed by a space, followed by the price per crypto, followed by a User_ID, and followed by the newline character (i.e., '\n').

When the server receives a BUY command from a client, the server should handle the operation by calculating the crypto price and deducting the crypto price from the user balance, a new record in the cryptos table is created or updated if one does exist. If the operation is successful return a message to the client with “200 OK”, the new usd_balance and new crypto_balance; otherwise, an appropriate message should be displayed, e.g.: Not enough balance, or user 1 doesn’t exist, etc.

A client-server interaction with the BUY command thus looks like:

```
c: BUY WLVRN 3.4 1.35 1
s: Received: BUY WLVRN 3.4 1.35 1
c: 200 OK
  BOUGHT: New balance: 3.4 WLVRN.  USD balance $95.41
```

Where 3.4 is the amount of crypto to buy, $1.35 price per one crypto, 1 is the user id.

SELL

SELL an amount of crypto. A client sends the ASCII string “BUY” followed by a space, followed by a Crypto_Name, followed by a space, followed by crypto price, followed by a space, followed by a Crypto_amount, followed by a space, followed by a User_ID, and followed by the newline character (i.e., '\n').

When the server receives a SELL command from a client, the server should handle the operation by calculating the crypto price and deposit the crypto price to the user balance. If the operation is successful return a message to the client with “200 OK”, the new usd_balance and new crypto_balance.

A client-server interaction with the SELL command thus looks like:

```
c: SELL WLVRN 2 1.45 1
s: Received: SELL WLVRN 2 1.45 1
c: 200 OK
  SOLD: New balance: 1.4 WLVRN. USD $98.31
```

Where crypto name is WLVRN, amount to be sold 2, price per crypto $1.45, and 1 is the user id.

LIST

List all records in the Cryptos table.

A client-server interaction with the LIST command looks like:

```
c: LIST
s: Received: LIST
c: 200 OK
   The list of records in the Crypto database for user 1:
   1    WLVRN 3.4 1
   2    BTC 5 1
   3    Dogecoin 200 1
```

BALANCE

Display the USD balance for user 1

A client-server interaction with the BALANCE command looks like:

```
c: BALANCE
s: Received: BALANCE
c: 200 OK
Balance for user John Doe: $98.31
```

SHUTDOWN

The SHUTDOWN command, which is sent from the client to the server, is a single line message that allows a user to shutdown the server. A user that wants to shutdown the server should send the ASCII string "SHUTDOWN" followed by the newline character (i.e., '\n').

Upon receiving the SHUTDOWN command, the server should return the string "200 OK" (terminated with a newline), close all open sockets and database connection, and then terminate.

A client-server interaction with the SHUTDOWN command looks like:

```
c: SHUTDOWN
s: Received: SHUTDOWN
c: 200 OK
```

QUIT

Only terminate the client. The client exits when it receives the confirmation message from the server.

A client-server interaction with the QUIT command looks like:

```
c: QUIT
```

c: 200 OK

Note, “400 invalid command” or “403 message format error” should be returned to the client if a server receives an invalid command or the command in the wrong format along with an appropriate message, e.g.: Not enough WLVRN crypto balance, not enough USD or user 3 doesn’t exist, etc.

3. Format

You should form a team of up to 3 students and then jointly work on your project. While each project should be an integrated effort, you should identify in your README.docx file what part of the project each member is responsible for.

Note, please form your group as soon as possible. Every student needs to sign up for one of the programming groups on canvas no later than 9/20.

4. Programming Environment

You can use either C/C++, Java or Python3 to implement the assignments. The assignments will be tested on the UMD Login servers (**login.umd.umich.edu**). For easy grading, please don’t use any GUI interface.

5. Requirements

The following items are required for full credit (100%):

- Implement all five commands: BUY, SELL, LIST, BALANCE, QUIT, SHUTDOWN
- After BUY and SELL the Cryptos table and Users table should be updated
- Both the server and client should be able to run on any UMD login servers.
- Make sure that you do sufficient error handling such that a user can't crash your server. For instance, what will you do if a user provides invalid input?
- The server IP address should be a command line parameter for the client program.
- The server should print out all messages received from clients on the screen.
- When the previous client exits, the server should allow the next client to connect.
- Your source codes must be well commented
- Submission should not include any object files, compiled files, or project files.
- Include a **README.docx** file in your submission with all compile and run instructions in details
- Include a **Makefile** in your submission.
- Submission size must be less than 2 MB, otherwise, 10 points will be deducted from the project grade.
- Submit a video showing that your project is running as expected if you are using Visual Studio for development

In your README file, the following information should be included: the commands that you have implemented, the instructions about how to compile and run your program, any known problems or bugs, and **the output at the client side of a sample run of all commands you**

implemented, this can be replaces with a recorded video of running the server and client in all cases.

6. Grading (100 points)

- Correctness and Robustness (90 points)
  - At least 10 points deducted for any bugs that cause the system crash.
  - At least 5 points deducted for any other bugs.
  - Only the server should handle the database, failing to do so will result in 35 points deduction
  - 35 points deduction when not using a database
- Comments and style (5 points)
- README and Makefile (5 points)

7. Submission Instruction

(1) Copy all your files (only source codes, data file, README, and Makefile, no object file and executable file) in a directory. The directory should be named like lastname_firstnameinitial_p1. For example, if you name is John Smith, your directory name should **smith_j_p1**. **Submission should not include any object files or project configuration files.**

(2) Generate a tar file of the directory using the following command. Enter the parent directory of your current directory and type

tar cvf lastname_firstnameinitial_p1.tar lastname_firstnameinitial_p1

For example

tar cvf smith_j_p1.tar smith_j_p1

Note, only the tar file will be accepted. You are fully responsible for generating the right tar file.

(3) Submit the tar file to the canvas website "P1" assignment folder.

8. Hints

- Start early. These programs will not be very long, but they may be difficult to write, and they will certainly be difficult to debug.
- **SSH**

  Mac OS and Linux come with a ssh client you can run from a Terminal. You do not need to install anything. Microsoft Windows users may download the SSH client software, from the following site: https://www.bitvise.com/download-area

- **Accessing the UM-Dearborn Home Drive**

  https://umdearborn.teamdynamix.com/TDClient/2019/Portal/KB/ArticleDet?ID=42879