CIS427 Winter 2024 Programming Assignment 2

Parallel Online Stock Trading Client/Server

Introduction

In this networking project, you will be required to create an online stock trading application that allows for **multiple clients to connect to the server simultaneously**. The client and server components of this application will communicate with each other using TCP sockets and must adhere to the protocol specifications provided.

To ensure the efficient handling of multiple client connections, you are required to implement a multithreaded server using either Pthread, Java threads, or Python Threads. Additionally, the client component of the application should be designed to monitor both the server's message and user input concurrently using select() statement and threads.

It is important to note that no third-party libraries are allowed to manage threads and sockets in this assignment. A third-party library is a library that need to be installed or added to the project that is not part of the programming language's standard library. For example, packages that are installed using pip or Conda are not allowed if Python is the programming language used for this assignment. This project will carry a weight of 10% towards your final grade.

To efficiently manage numerous clients, the server can create a thread pool with a maximum number of concurrent connections, such as 10, and accept multiple connections using the select() API. It will then create a new thread to manage each connection.

In addition to the commands previously handled by the client in PA1, namely BUY, SELL, LIST, BALANCE, QUIT, and SHUTDOWN, the server must also manage several additional commands, such as LOGIN, LOGOUT, DEPOSIT, WHO, and LOOKUP. Slight modifications will be made to the LIST and SHUTDOWN commands in this assignment.

LOGIN

Login the user to the remote server. A client that wants to login should begin by sending the ASCII string "LOGIN" followed by a space, followed by a UserID, followed by a space, followed by a Password, and followed by the newline character (i.e., '\n').

Your server should be initialized with the UserIDs and Passwords of at least four users who will be allowed to execute the aforementioned commands on the server. However, a non-root user is allowed to execute the QUIT commands anonymously (without login).

When the server receives a LOGIN command from a client, it should check if the UserID and password are correct and match what the server stores. If login info is not correct or doesn't exist, the server should return the string "403 Wrong UserID or Password,"

otherwise the server should return the “200 OK” message. This command will result in a server creating a new thread for this client.

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

Client sends: LOGIN john john01
Server sends: 200 OK

LIST

There is a slight modification on this command from PA1. Only a **root** user can list ALL records for ALL users. A LIST command issued by user John should return only John’s records. A client server interaction looks like this:

Client sends: LIST
Server sends: 200 OK
The list of records in the Stock database for John:
1 MSFT 3.4
2 APPL 5
3 U 200

Scenario 2, if a root user is currently logged in
Client sends: LIST
Server sends: 200 OK
The list of records in the Stock database:
1 MSFT 3.4 John
2 APPL 5 John
3 U 200 John
4 SP500 100.32 Alex
5 MSFT 12.12 Alex

LOGOUT

Logout from the server. A client sends the ASCII string “LOGOUT" followed by a name followed by the newline character (i.e., '\n'). A user is not allowed to send BUY, SELL, LIST, BALANCE, and SHUTDOWN commands after logout, but it can still send the QUIT commands. This command should result in the server terminating the allocated socket and thread for this client.

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

Client sends: LOGOUT
Server sends: 200 OK

WHO

List all active users, including the UserID and the user's IP addresses. A client sends the ASCII string "WHO" followed by the newline character (i.e., '\n'). This command is only allowed for the root user.

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

Client sends: WHO
Server sends: 200 OK
The list of the active users:
John 141.215.69.202
root 127.0.0.1

LOOKUP

Look up a stock name in the list. Display the complete stock record for the logged in user. A client sends the ASCII string "LOOKUP" followed by a space, followed by a name followed by the newline character (i.e., '\n').

When the server receives a LOOKUP command from a client, it will look up the name of the stock and return the matched record for that logged in user. When there is a match, it returns the "200 OK" message and all the matched record(s). If there is no match, it returns the "404 Your search did not match any records". Partial or full stock name are both accepted.

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

Client sends: LOOKUP MSFT
Server sends: 200 OK
Found 1 match
MSFT 5

Client sends: LOOKUP NOSTOCK
Server sends: 404 Your search did not match any records.

DEPOSIT

Deposit USD to the user's account/record. A user can deposit an amount of USD into their account. A client that wants to deposit an amount of USD should begin by sending the ASCII string "DEPOSIT" followed by a space, followed by a USD amount, followed by a space, followed the newline character (i.e., '\n').

A client-server interaction with the LOOKUP command thus looks like:
Client sends: DEPOSIT 12.34
Server sends: deposit successfully. New balance $112.34

SHUTDOWN

If a client sends a shutdown command to the server, it will terminate all connected clients and shut down the server. However, only the root user has the authority to execute a server shutdown. If a non-root user attempts to send a shutdown command, the server should refuse it and send an error message accordingly.

The Assignment

Programming Environment

The programming assignments can be implemented using C/C++, Java or Python. It is advised that the progress of each team member is tracked using either github or bitbucket. To avoid one student bearing the bulk of the workload, it is suggested that you start by creating a list of tasks and distributing them equally among all members of the class. Individual grading will be employed, and a teamwork survey like the one used for PA1 will be conducted.

Requirements

The following items are required for full credit:

1. implement all commands from this and previous assignments: BUY, SELL, DEPOSIT, BALANCE, LIST, QUIT, SHUTDOWN, LOGIN, LOGOUT, WHO, LOOKUP.
2. all users share the same data source (one database or one text file). The user's information should be maintained by the server. You must have the following users (lower case) in your system:

UserID	Password
Root	Root01
Mary	Mary01
John	John01
Moe	Moe01

3. 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?
4. the client should be able to connect to the server running on any machine. Therefore, the server IP address should be a command line parameter for the client program.
5. the server should print out all messages received from clients on the screen.
6. when the previous client exits, the server should allow the next client to connect.
7. your source codes must be well commented.
8. include a README file in your submission. A document with instructions, screenshots and **what was each student's role in this assignment**.

In your README file, you must provide the following details: a list of functions that were implemented by each team member individually, instructions on how to compile and run the program, any known bugs, and sample outputs from a comprehensive test of all implemented commands. A sample README format can be found at the conclusion of this document.

Grading (100 points)

- 5x11 points for each command
- 10 points for using Select() statement
- 10 points using Threads to handle client requests
- 10 points for running the code without any bugs that cause the server to crash.
- 5 points for code comments and logging the activity on the server
- 10 points for including a README and the video recording

Submission Instruction

1. Copy all your files (only source codes, data file, README, and document - no object file and executable files) in a directory. The directory should be named like *group_number_pa2*.
2. Generate a zipped file of the directory and submit it to Canvas.
3. To fulfill the requirement, you must create a video recording of the program while it is running sample use cases in the table below. You can upload the video recording to either canvas or a hosting server, such as youtube.com, and include a link to the video recording in your README file.

A sample README file layout:

- Student Names and emails
- Introduction:
    - Including but not limited to: Platform, programming language used, etc.
- Running Instructions or using the Makefile:
    - For example, gcc -o server.c
- Each Student's role:
    - Student 1: Implemented the following features: ….
    - Student 2: Implemented the following features: …..

- Bugs in the code:
    - Including but not limited to the bugs in the code and cases that were not implemented in this code…
-