**PARALLEL POKÉMON CARD TRADING CLIENT/SERVER**

In this networking project, you will be required to create an online Pokémon card 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. 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**
* **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 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 Pokémon cards table for current user, John:*

ID Card Name Type Rarity Count OwnerID

1 Pikachu Electric Common 2 John

5 Charizard Fire Rare 6 John

*Scenario 2, if a* ***root user*** *is currently logged in*

Client sends: LIST

Server sends: 200 OK

*The list of records in the cards database:*

ID Card Name Type Rarity Count OwnerID

1 Pikachu Electric Common 2 John

2 Charizard Fire Rare 1 Jane

5 Charizard Fire Rare 6 John

**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 Pokémon card name in the list. Display a complete card record for the logged in user that matched full or partial name of the card’s name. 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 card’s name and returns the matched record for that logged in user. When there is a match, it returns the “200 OK” message and all matched record(s). If there is no match, it returns the “404 Your search did not match any records”. Partial or full card name are both accepted.

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

Client sends: LOOKUP Fire

Server sends: 200 OK

Found 1 match

ID Card Name Type Rarity Count Owner

2 Charizard Fire Rare 1 John

Client sends: LOOKUP random\_card

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 reply: deposit successfully. New User 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 a 401 status code, and an error message accordingly.

*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*

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: instructions on how to compile and run the two programs, 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.

*Submission Instruction*

1. Copy all your files (only source codes, data file, README, and document - no object file and executable files) in a directory.

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 use cases. You can upload the video recording to either canvas directly 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:*

• Introduction:

o …. Platform, programming language, etc.

• Running Instructions:

o ….

• Each Student’s role:

o ….

• Bugs in the code:

o …..

• A table showing the test cases with details for the bugs mentioned above.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test #** | **Description of test** | **Input Value** | **Expected Output** | **Actual Output** | **Test Pass/Fail** |
| 1 | LOGIN: test correct user/password | LOGIN type\_in\_username  And  password | 200 OK |  |  |
| 2 | LOGIN: test invalidinput | LOGIN root  password | 401 wrong username/password |  |  |
| 3 | LOGOUT |  |  |  |  |
| 4 | WHO |  |  |  |  |
| 5 |  |  |  |  |  |
| 6 |  |  |  |  |  |

**SCREENSHOTS:**

Test #1 Screenshot (Server and client 1):

…..

Test #2 Screenshot (Server, with at least two clients: client 1 and client 2):

…..