Experiment 3: Network Protocol Programming

1 Introduction

In this lab you will be creating a client to talk to a simple communications server via a custom defined protocol. The server responds to queries with a set of standardized commands. More on the query format will be explained in Section 2. You will utilize Java (or another language with prior approval from the instructor) in order to create an application to query the server and run a set of benchmark tests on the server. You will use your created application twice to communicate between two users. Your application will need to meet a set of specified performance restraints in order to complete the task.

2 Protocol Description

The protocol you are asked to communicate via requires that you send commands in a binary formatted message. There are 4 fields to the message, the first 3 are simply Java integers (4 bytes each) and the 4th field is a variable message payload to be sent to the server. The fields of a message are described as

Message Type	Sub-Message Type	Size	Message Data
Int (4 bytes)	Int (4 bytes)	Int (4 bytes)	Byte array of variable size defined by the "Size" field.

The "Size" field of the message defines, as an integer, how many bytes are in the "Message Data" field. The size of the data is needed since when reading data off a socket you need to know how much to read! Accepted values are between 0 and 262,144 inclusively. This effectively allows a message of 256 KB as a maximum data payload. The "Message Type" field defines the original operation to the server sent from a client. The "Sub-Message Type" field defines the response from the server and is dependent on the value of the "Message Type" field. For example, consider an echo command, which has a Message Type of 2. When the server responds, it returns the same value in the Message Type field and a corresponding message in the Sub-Message Type field of 0, meaning the echo succeeded. This is of course a simple example, where the command only has one possible return value. In more complex cases, the Sub-Message Type field can take more than 1 value, corresponding to the result of the requested operation on the server.

A server for you to execute queries against will be provided with a hostname and port number you can run your tests against. The system we are designing is a basic chat functionality. A user of the system can post messages to other users' "inboxes" which the user can read at a later date. The following sections define the possible commands the server will respond to

Command Name	**Message Type Value**	**Message syntax**	**Description**
Exit	0	N/A	This command requests that the current connection to the server be terminated and the current user logged out. This command has no corresponding message data. If data is sent to the server, it is ignored.

Badly Formatted Message	1	Information message	This is returned if an invalid integer value is set for the Message Type field, meaning that an unsupported command was requested.
Echo	2	Text to echo back	This is a simply echo request, meaning that the server simply returns the same text which was sent to it.
Login	3	<username>,<password>	This command is a login request for a user in the system where the username and password are separated by a comma. The username cannot contain a comma, however all text following the comma are considered part of the password
Logoff	4	N/A	Logoff the currently logged in user
Create User	5	<username>,<password>	Create a user using the supplied username and password. The same restrictions on username and password formatting apply as in the Login command
Delete User	6	N/A	Delete the currently logged in user and log them out
Create Store	7	N/A	Create the currently logged in user's backing data store for messages. Without this data store, the user cannot receive messages but can still send them. This should be the first operation run after user creation and initial login, and should only be run once for a user.
Send message to user	8	<dest_username>, <message>	Send a message to the specified username as signified by <dest_username>. Any text following the "," is considered part of the message and will be sent to that user.
Query messages	9	N/A	Query any available messages for the currently logged in user.

Many of the listed commands will include message arguments in the data payloads upon returning a result from a request. These messages are useful when debugging your designed system to get more information on what went wrong.

These operations have a set of specified possible responses to them which are outlined in the following section

2.1 Client Operation Responses

1 **Exit Request**: The exit request will have no response, and will simply disconnect the connection immediately.

2 **Echo Request**: The echo message will only respond with a response of

2	0	Size of message data	Message data

where the message data is the intended message text to "echo" back from the server to the client. This is primarily available for debugging purposes.

3 **Delete User**: This command is the equivalent to a closing of the currently logged in user's account on the server. This command has three possible responses signified in the Sub-Message Type field:

a. **User Deletion success** = 0
b. **Not Logged In** = 1
c. **General Error**
    i. A General Error should never occur since a user must be logged-in in order to delete themselves. However it is here to complete the syntax of the system

4 **Create User**: The user creation command has four possible responses in the Sub-Message Type:

a. **User Creation Success** = 0
b. **User Already Exists** = 1
    i. This will be returned if the username already exists in the system
c. **User Already Logged In** = 2
    i. In order to create a user on the system, no one can be logged in. It is assumed you want to create an account and do not have one already.
d. **Badly Formatted Create User Request** = 3
    i. This will occur if you do not have 2 fields (username and password), separated by a "," when you as the server to create a user.

5 **Create Store:** The operation for creating a user store has three possible return values

a. **Store Created Successfully** = 0
b. **Store already exists** = 1
c. **Not Logged In** = 2
    i. This will be returned if you try to create a store before logging in first.

6 **Login**: The login operation has four possible responses:

a. **Login Ok** = 0
b. **User already logged in** = 1
c. **Bad credentials** = 2
    i. This will be returned if the supplied username and password combination was not able to be found in the database. It *is* however a correctly formatted message
d. **Badly formatted message** = 3
    i. Returned if expected fields are missing (such as no username/password supplied)

7 **Logoff**: The logoff command has three possible responses

a. **Logoff OK** = 0
b. **Not Logged In** = 1
    i. You tried to log out without first logging in.
c. **Session Expired** = 2
    i. This system employs a session management system which will automatically log a user off after a period of inactivity (60 seconds). After 60 seconds, this

message is sent to the user signifying that they need to log in again. *Any* command sent to the server will extend the period to 60 seconds from the last sent command.

8 **Send Message to User**: The send message operation has five possible responses:
   a. **Message sent** = 0
   b. **Failed to write to the user's data store** = 1
      i. Returned if the destined user hasn't created a data store, but does exist in the system
   c. **User doesn't exist** = 2
   d. **Not Logged In** = 3
      i. Returned if you tried to send a message without logging in
   e. **Badly formatted** = 4
      i. Returned if you are missing fields in the data argument for this message type.

9 **Query Messages**: The query message operation has three possible responses
   a. **No Messages** = 0
      i. There were no messages outstanding the user's message store.
   b. **Messages** = 1
      i. This will return a group of individual responses, each containing a single message found in the user's message store. The messages will be of the format

Message Type	Sub-Message Type	Message Size	Message Text
9	1	<size>	<message data>
9	1	<size>	<message data>

      ii. This represents a set of two (2) messages returned by the server for the currently logged in user.
      iii. The message text will have the format of <user_from>,<message_time>,<message text>

      As can be seen, the various fields are comma (,) separated, like the message arguments in the server commands. An example set of received messages might be

9	1	<size>	Admin,2013-12-10 10:00:00,Hello there
9	1	<size>	Admin,2013-12-10 10:01:00,How are you?
9	1	<size>	FakeUser,2013-12-10 12:00:00,I'm a fake user!
9	1	<size>	FakeUser,2013-12-11 1:00:00,I can't sleep…

10 **All other commands will return a message of**

1	0	<message size>	<Some text>

signifying that the received message was not an operation the system allows.

3 Design Exercises

You are asked to create a client application to interface with the chat server described in Section 2. In order to create an interface, you will need to support all the commands and responses available via the protocol in the API. You should make sure to make note of all design decisions made when planning out the client application. The application should be able to

1. Automatically poll for new messages for the logged in user (at a maximum rate of 1 poll/second)
2. Define some format for sending messages to users (possibly look at other protocols for chatting with users to see how this might work, e.g. using the "@<username>" symbol or something custom defined) Make sure to outline what you chose and why.
3. Automate all user creation/deletion procedures
4. Parse all operations to/from the server
5. Provide a simple UI, which allows full functionality of the system and converts all network commands

This design process is not a simple procedure and there are many aspects to keep in mind when designing a network application. Functionality is the majority of the focus of this experiment. If your interface is based on the command line, that is fine. There is no need for a full GUI, simply providing an interface to the commands is enough. Your application should however handle dealing with authentication to the system, making sure that the user is correctly logged in for commands to be run, etc. The user interface's ease of use isn't as important as all the system functionality existing and all of the commands supported (not supporting the ECHO command is fine, that's only there for debugging purposes).

Therefore you are asked to keep an active list of the design challenges you see in this problem and how you plan to solve/solved them. Also keep in mind any requirements you may not be able to meet and why. You will be asked more detail about these in the demonstration. You should outline all of this information in the report to be submitted. You should also explain how you solved each of the design constraints listed above.

You should also create test cases (pen-and-paper or JUnit are both acceptable) to show that you've implemented the requested features. These test cases will be reviewed in the demo and included in the report. You should also use Wireshark to capture the information going across the connection and verify it is correct.

Also as a side-note, when creating a user on this messaging server, passwords are transmitted and stored in plain-text. Therefore you should not use a password which might be used elsewhere as there is no security here. Using an easy-to-guess password is fine in this context.

4. Server details

The demonstration server is hosted at dsp2014.ece.mcgill.ca and the port is 5000. If load becomes too high on this port, another server may be started on another port at the instructor's discretion to compensate. Information on this can be found on the discussion boards on myCourses if it becomes necessary.

5. Important Dates and Evaluation

5.1 Demo

The demo will take place in class on March 10 and 11 and will count for 10% of your final grade. During the demo, you will also be asked to explain your design decisions and process for the tasks outlined in Section 3.

5.2 Report

The report is due at 23:59 on March 16 and it will count for 13% of your final grade. In the report you should include your solution for all the design problems as well as your methodology in solving each of them.