McGill | School of Continuing Studies
Instructor Services and Educational Technologies

CCCS 431 (764) Networking Fundamentals - Programming Assignment

McGill University - School of Continuing Studies

Due: August 8, 2021

5% of final Grade

Socket Programming - Network Application Layer

The purpose of this assignment is to implement a client-server application using sockets. The assignment consists of two console applications: the client application, and the server application. You may use java or any programming language of your choice for the implementation.

Overview

The system consists of a client and a server to implement a Simple Date-Time Info Protocol (hereby known as the SDTP), as described in the following.

The Simple Date-Time Information Protocol (SDTP)

The client may connect to an available server on the network. The SDTP server is listening on port 431. The SDTP protocol is implemented using TCP and is outlined in the following. It illustrates a sample "SDTP session" (started by HELLO, ended by BYE).

```
SERVER: HELLO CCCS 431 SDTP Server written by … READY
CLIENT: HELLO
SERVER: ALOHA <<the remote ip address>>
      CLIENT: HELP
      SERVER: Server responds with a list of commands
              and additional information;
              The information is ended with a single dot in
              the very last line
              .
      CLIENT: DOW
      SERVER: Server responds with the day of week
      CLIENT: TIME
      SERVER: Server responds with the time of the day
      CLIENT: DATE
      SERVER: Server responds with the current date
      CLIENT: DATETIME [<<FORMAT>>]
      SERVER: Server responds with the current date and time
CLIENT: BYE
SERVER: BYE (followed by closing the connection)
```

The client connects to the server on port 431 and waits to receive the HELLO message from the server. Every command is a single line command (terminated by a newline character). Every message is also a single line response (terminated with newline) except for the HELP command, which responds to the client with multiline data terminated with a single dot in the very last line.

The detailed explanations of the commands are given in the following:

The SDTP Commands:

HELLO: required for starting the STDP session. Upon receiving, it sends a personalized welcome message to the client by indicating the client's IP address. The command is only valid if the client is not already in an STDP session. If the client has already started a session (the client has already sent a previous HELLO command), this should be considered as an error in protocol and the server may send the following error message to the client.

```
ERROR Already in session.
```

HELP: displays the help information about the protocol. It may be received any time whether and STDP session is established or not.
DOW: returns the day of week (i.e. WED for Wednesday). This command is valid only if STDP is in session (HELLO is already received).
TIME: returns the current time. This command is valid only if STDP is in session.
DATE: returns the current date. This command is valid only if STDP is in session.
DATETIME: returns the current date and time. The command has an optional format argument (if not specified, a default format may be used). This command is also valid if STDP is in session.
BYE: ends the session. Valid only if STDP is in session. The server sends the closing message and ends the remote TCP session.

STEP 1 – Basic Server

In this step, you create a basic TCP listener that upon receiving an incoming connection, it sends the following message to the client following by a new line character:

```
HELLO CCCS 431 SDTP Server written by … READY
```

The server application continues listening on port 431 unless the user wishes to close the application.

Create such a listener and test your program using telnet.

STEP 2 – Simple Client

In this step, you create a TCP client application that connects to server (the server name or ip address to be entered by the user in the command line).

The client application simply prints out what it received from the server.

Upon connecting to the server send a "`HELLO`" command (followed by a newline character) to the server. At this stage, the server does not respond to the client, however the data is properly received.

Close the connection on the client side and terminate the program normally.

STEP 3 – The SDTP Server (STDP Partial Protocol Implementation)

In this step, you implement the full server protocol. The application server is a java console application that is listening on port 431 implementing the server side of the SDTP protocol as specified in the previous section.

It implements the TCP listener. You may start the server by running this application on the console. Upon running, it displays the following message (on the console), by which the user may decide to end the program by hitting the ENTER key.

```
CCCS 431 SDTP Server written by … running
Hit enter to STOP.
```

Server Implementation Notes:

The server side may receive simultaneous connections. Each client is served by a different socket, running in a different thread.

Note that every client command is given in a single line. Therefore, the server always reads the input stream until it reaches the new line character. It, then, processes the input command and continues to receive the next command until the client wishes to exit.

During the communication if the protocol is not followed by the client (i.e. the client closes the connection or send an invalid command text, etc.) the server immediately closes the connection and writes an error message on the screen.

You may test the server implementation using a telnet program by manually connecting to the server application.

Commands to implement:

- HELLO: to start a STDP session by sending a personlaized message to the client.
  Hint: Use `getRemoteSocketAddress()` to obtain the client's IP address.
- DOW: upon receiving the string DOW followed by a new line, it sends the current Day of Week to the client. The text is followed by a new line character.
- TIME: returns the current time followed by a new line character.
- DATE: returns the current date followed by a new line character.
- BYE: sends the closing message to the client (followed by a new line) and closes the TCP session.

You may modify the server application you created in step 1.

Use telnet to test your server application. Test it with two simultaneous client connections.

STEP 3.2 – The Full STDP Server (Bonus)

In this step, you may extend the implementation by implementing the HELP and DATETIME commands. Notes:

- The HELP command sends back a multi-line response data (terminated by a single dot in a new line followed by a new line character)
- The DATETIME command has an optional format argument. If not specified a default argument may be used.

STEP 4 – The Client Application

The client application is a java console application that lets the user to do the following. Upon running, it displays a menu and receives user commands in a "command loop" until the user wishes to exit the program:

- Show Help/About: (displays a friendly help / about information)
- Connect*: connects to an existing SDTP server (receives the IP address or the name of the server computer from the user, connects to the server, and initiates an STDP session by sending the HELLO command, displays the welcoming message form the server, and waits for more command from the client to send to the server)
- Server HELP (Bonus): performs a HELP command on the STDP server and displays the result. Note that the server returns a multi-line output.
- Get DOW: Get the day of the week (the client side of the TCP protocol, see below)
- Get time: Get the time (the client side of the TCP protocol, see below)
- Get Date: Get the date (the client side of the TCP protocol, see below)
- Get Date and Time: Get the date and time (the client side of the TCP protocol, see below)
- Exit*: safely terminates the application (sends a BYE command to the connected server and closes the active socket, if connected; see below)

Notes:

1. The Exit command "properly" closes the current connection and connects to the new server. Proper closure of the connection means sending the BYE command to the server (see the TCP protocol in the previous section). If the client is not connected, it simply exits the program.
2. The Connect command also "properly" closes the current connection (if applicable) and then proceeds with a new connection.

Client Implementation Notes:

The client side starts the communication by connecting to the server. The server must be running. In case of any network errors, the client simply reports the error and terminates the program.

The client displays a menu to the user, and upon receiving the menu option, it creates the command, sends is to the server, and displays the result. All results are provided in a single line (with the exception of the HELP command) and therefore the client reads the input steam until it reaches the end of the line. In case of the HELP response, the client continues reading the input stream until it reaches a sequence of a new line, a dot character, followed by another a new line character.

In case of any errors in the protocol the client simply closes the connection, displays the error message on the screen, and terminates the program.

Useful links:

Java Sockets: https://www.baeldung.com/a-guide-to-java-sockets

Java Date Class, as used in: https://www.w3schools.com/java/java_date.asp

Evaluation Criteria

The following scheme is used for marking this assignment:

Step 1	25 marks
Step 2	25 marks
Step 3 (full protocol and error handling)	25 marks
Step 3.2	10 marks
Step 4	25 marks
Total	**100+10** marks

Submission

Submit the code and executables through myCourses. Include a README file and provide any special instructions on how to run the code, if necessary.

This is a group assignment (a group of three). Please submit one per group. Clearly indicate your group information (full name name + McGill Id).