Assignment 2: Transport Layer and Micro Servers

due: **Friday, October 23, 2020 (11:59pm)**

Transport Layer and Micro Servers (40 Marks)

Learning Objectives. The purpose of this assignment is to learn about client-server network applications, TCP and UDP protocols, and data representation. In particular, you use a combination of TCP and UDP data transfer services to implement client-server programs (in Java) that provide different services to users.

Assignment Description

Background. Both TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) are transport layer protocols used for sending bits of data (known as packets) over the Internet. They both build on top of the Internet protocol (IP). TCP is a connection-oriented protocol and UDP is a connection-less protocol. TCP establishes a connection between a sender and receiver before data can be sent. UDP does not establish a connection before sending data. TCP is a reliable data transfer protocol. TCP and UDP are not the only protocols that work on top of IP. However, they are the most widely used.

Assignment. Your primary task is to implement a client/server application. The Server program offers several different services to the Client. Let's call this server program as Master Server.

Client and Master Server. In this assignment, you are supposed to implement a Client and a Master Server programs as described here:

- **Client**. There is only one Client program in this assignment. The client makes a **TCP** connection with the Master Server. The Client's interaction with the Master Server will involve connecting to the server (by getting Master Server's IP and Port number via terminal), entering a sentence of one or more words to be used as the source data, and then entering a loop for interaction with the server (you shortly will see the list of interaction). Within the loop, the client can specify what data transformations are desired on the original sentence source data, and in what order. These requests may involve one or more data transformations, to be performed in the order specified.

- **Master Server**. The master server operates on sentence-like messages entered by the user, and uses **TCP** as its transport-layer protocol, for reliable data transfer with the client. The master server then communicates with six micro-services via **UDP** to perform the composed data transformations on each word, prior to returning the final result data back to the client via TCP. Additional client commands can be sent to apply new transformations to the same original sentence source data. When the client is finished with all data transformation requests, the session with the master server ends.

Micro Servers. The micro-services can operate on a word or a sentence at a time (your choice). These data transformation services will be offered via UDP-based communication, which is simple and fast, but unreliable. There will be several different micro-services running, each on a different port. The master server needs to know which services are running where (i.e., IP address and port), and send the data to the correct place for each data transformation request.
The specific data transformation micro-services that you need to implement are:

1. **Echo**: This micro server returns exactly what was received.
   For example, the message "This is a test" would become "This is a test".

2. **Reverse**: This micro server reverses the message and returns the result back.
   For example, the message "dog" would become "god".

3. **Upper**: This transformation converts all lower-case alphabetic symbols (i.e., a-z) in a message into upper case (i.e., A-Z). Anything that is already upper case remains unchanged, and anything that is not a letter of the alphabet remains unchanged.
   For example, the message "This year is 2020" would become "THIS YEAR IS 2020".

4. **Lower**: This transformation converts all upper-case alphabetic symbols (i.e., A-Z) in a message into lower case (i.e., a-z). Anything that is not a letter of the alphabet remains unchanged.
   For example, the message "This year is 2020" would become "this year is 2020".

5. **Caesar**: This transformation applies a simple Caesar cipher to all alphabetic symbols (i.e., a-zA-Z) in a message. Recall that a Caesar cipher adds a fixed offset to each letter (with wraparound). Please use a fixed offset

of 2, and preserve the case of each letter. Anything that is not a letter of the alphabet remains unchanged.
For example, the message "I like dogs!" would become "fqiu! nkmg K" if operating on a sentence at a time, or "K nkmg fqiu!" if operating on a word at a time.

6. **Yours**: Design your own simple data transformation that is different from those above, somewhat interesting, reasonably easy to implement and explain, and applicable to one or more of the data bytes in a typical message.

Please note (and obey) the numbering on the data transformation services above, since they will be used for the client commands sent to the master server. For example, if the source data "I love cats!" is transformed using the command "2154", then the resulting data would be "!uvce gxqn k" if operating on a sentence at a time, or "k gxqn !uvce" if operating on a word at a time. It is up to your master server to coordinate this, and either is acceptable. Your master server should also do something reasonable if no response is received from one of the UDP-based micro-services within a certain time limit.

Testing. First, run all six micro servers. Then run your Master server. The Master server should have the IP address and Port number of the micro servers. The Client program should receive the Master server IP address and port number through the command line. Then the Client program should make a TCP connection to the Master server. After receiving a Hello message from the Master server, the Client should enter a Message. After confirming the message by the Master server, the Client program should select a combination of operations on the message. Finally, the Client program should be able to close the connection.

Upload in D2L and Grading

What to Upload in D2L You will compress all the Java files (Master.java, Client.java, Echo.java, Reverse.java, Upper.java, Lower.java, Caesar.java, and Yours.java) into a single zip file (*A2.zip*) and upload it with a one-page user manual (PDF) in D2L / Assignment 2 folder.

Grading. The grading scheme for the assignment is as follows:

- **14 marks** for the design and implementation of the main TCP-based client-server solution to this problem. Your implementation should include proper use of TCP and UDP socket programming in Java, and reasonably commented code.
- **12 marks** for a proper implementation of each of the UDP-based data transformation services (6 of them, each worth 2 marks).

- **4 marks** for a clear and concise user manual (at most 1 page) that describes how to compile, configure, and use your Web proxy. Make sure to clarify where and how the testing was done (e.g., local host, CPSC Machine), what works, and what does not. Be honest!
- **10 marks** for a suitable demonstration of your server to your TA in your tutorial section. A successful demo will include marks for the test cases above, as well as clear answers to questions asked during your code walk-through.

- **Bonus** (Optional): Up to 2 bonus marks are available if your master server allows the dynamic creation, registration, and termination of micro-services while the master server is running, rather than having them all started in advance.

Helps Available

- Two tutorial sessions will be dedicated to providing helps on this assignment.
- The instructor will perform an assignment demo in one of the upcoming lectures.
- The instructor is also available during office hours (8am - 10pm on Mondays) or by appointments.

Tips.

- This is a rather challenging assignment, so please get started early. You will likely need at least a week to get it fully working.
- For the micro-services, start with the echo server, which is the simplest one. Once it is working, you can make the small changes required to create the other micro-services. Test them individually, perhaps on a word at a time.
- When debugging the master server, use just one micro-service initially. Then try it with two, and make sure to keep track of which is which. And so on.
- It might be helpful to have the master server put some fixed delays (e.g., 1 second) in between the different stages of data transformation when you are doing composed micro-services.
- Once you are testing on a real network interface, you may find Wireshark particularly useful to look at the network packets being sent and received by your application (i.e., ports, size, data content).

- Work with very small amounts of data (e.g., 1-5 words) to start with, so that you can put verbose debugging in to test the functionality of your application. Once these are working right, you can scale up to larger datasets (e.g., 10-100 words) to tackle any new challenges that might arise.

Good Luck!