CSCI 450: Data Communication and Network Programming

Programming Project 2

Due April 19th, 7PM

You are required to submit the following files (zipped into a single file)

a. A document (in MS Word) that describes
   - The application protocol, including
     - Type of messages
     - The syntax and semantics of each message type, including
       - The format of the messages
       - The meaning of each field in the format
       - The value range of each field
     - Rules of sending messages (i.e. which message should be sent by which end (client or server) under what circumstance)
   - A table of test cases that you used to test your program. For each of the test cases, provide
     - The rationale (Why is the test case needed, or what aspect of the program is tested?)
     - The content (the input, the expected output and the actual output)
     - The errors observed for the test case, if any.
   - The usage of your client and server program respectively.
   - The instructions to compile your client and server program respectively.
   - Any known problems of your program, which will help you earn partial credits.
   - Any significant references used (other than the text book and slides).

b. The link to your Github repo.

Note:

1. Your application protocol is expected to be different in some way from all other classmates' protocol. Interviews will be conducted if your protocol is the same as others'.
2. Your effort is expected to spread evenly over the project period, as shown by the commit history.

In this project, you will develop two network programs, one for client and one for server. The client program sends a file containing units as defined in the practice project to the server. The server translates the units to the format specified by the client, and saves the units to a file of the name also specified by the client. Before translating the units, the server checks if received units have a correct format. If not, the server sends back an error message immediately without saving the file. If yes, the server sends back a confirmation message after saving the file. The client prints the result, "Format error" or "Success", after getting the message from the server. All the network communication uses UDP with possible segment loss introduced. Therefore, you must implement the stop-and-wait protocol at the *client* side in order to have data reliably transferred.

The client should be invoked by the following command:

```
<client> <server IP> <server port> <file path> <to format> <to name>
<loss probability> <random seed>
```

Where `<client>` is the name of the client executable file name, `<server IP>` is the IP address of the server, `<server port>` is the TCP port of the server, and `<file path>` is the path of the file to be sent to the server. (The file path indicates the location of the file in the system on which the server runs. It includes the file name, and possibly the hierarchy of directories.) There is no size limit of the file. `<to format>` indicates how the server should translate the received units. 0 means no translation, 1 means to only translate type 0 units to type 1 with type 1 units unchanged, 2 means to only translate type 1 units to type 0 with type 0 units unchanged, and 3 means to translate type 0 to 1 and type 1 to 0. `<to name>` is the name of the file the server should save the units to. `<loss probability>` is between 0 and 1 and is the probability of segment loss. `<random seed>` is an integer to control random number generation. Setting the same random seed and loss probability will provide the same segment losses across multiple executions.

The server should be invoked by the following command:

```
          <server> <port> <loss probability> <random seed>
```

Where `<server>` is the name of the server executable file name, `<port>` is the port the server listens to. `<loss probability>` and `<random seed>` server the same purposes as explained for the client.

In both programs, you must call the `lossy_sendto` function in the provided `send-lib.c` to send segments. The random seed from the command line arguments should be passed to this function.

The parameters provided by the command line arguments must NOT be hard-coded.

Example:

Client command:

```
   myclient 127.0.0.1 5678 dir1/dir2/ex_file1 3 target 9
```

Server command:

```
   myserver 5678 3412
```

The client will send the units in the file `ex_file1` from the path `dir1/dir2` on the server. The server will check the received units. If any unit has wrong format, the server will simply send back an error message and close the connection. If everything is right, the server will translate type 0 units to type 1 and type 1 units to type 0, then save them to a file named `target` in the directory which the server application is in, send a confirmation message and close the connection.

Credit allocation:

Documentation: (30% total)

- Protocol: 15%
- Test cases: 15% (no credit if the application does not work)

Programs (including comment): 70%

No credits will be given to the programs and test cases (total 85%) if

1) The programs cannot compile, or
2) The programs crash immediately when it runs, or
3) The programs send segments without calling the `lossy_sendto` function.

Segmentation faults, freezing or infinite loop will result in significant loss of credits of those parts.