CSCI 450: Data Communication and Network Programming

TCP Programming Project

Due October 18th, 11 PM

You are required to submit a document in the PDF format at

http://hucs.dynu.net/lij/courses/submit_hw.html

The document should include

- A The link to your Github repo. Make sure that it is public.
- 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 case results. For each test case, the expected output and the actual output should be provided. The test cases should include but are not limited to the following:
  - The file given in the command to run client does not exist
  - The file given in the command to run client is empty
  - The format given in the command to run client is out of range
  - The file from client has only type 0 units and the "to format" is 0.
  - The file from client has only type 0 units and the "to format" is 1.
  - The file from client has only type 0 units and the "to format" is 2.
  - The file from client has only type 0 units and the "to format" is 3.
  - The file from client has only type 1 units and the "to format" is 0.
  - The file from client has only type 1 units and the "to format" is 1.
  - The file from client has only type 1 units and the "to format" is 2.
  - The file from client has only type 1 units and the "to format" is 3.
  - The file from client has both type 0 and type 1 units and the "to format" is 0.
  - The file from client has both type 0 and type 1 units and the "to format" is 1.
  - The file from client has both type 0 and type 1 units and the "to format" is 2.
  - The file from client has both type 0 and type 1 units and the "to format" is 3.
  - The file from client has only type 0 units but have errors
  - The file from client has only type 1 units but have errors
  - The file from client has both type 0 and type 1 units and have errors in type 0 units only
  - The file from client has both type 0 and type 1 units and have errors in type 1 units only
  - Two clients send files to the server one by one
  - Ten clients send files to the server one by one
- The usage of your client and server program respectively.

- The instructions to compile your client and server program respectively.
- Any code base borrowed.
- Any significant references used (other than the text book and slides).
- Any known problems of your program, which will help you earn partial credits.

Note:

1. Your application protocol is expected to be different in some way from all other classmates' protocol. Interviews will be conducted to check plagiarism 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, or points may be deducted.

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, then quits. The server keeps running and waits for another client to connect.

The client should be invoked by the following command:

<client> <server IP> <server port> <file path> <to format> <to name>

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.

The server should be invoked by the following command:

<server> <port>

Where `<server>` is the name of the server executable file name, `<port>` is the port the server listens to.

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`

Server command: `myserver 5678`

In the above example, the client will send the units in the file `ex_file1` from the path `dir1/dir2`. 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. After receiving either the error message or the confirmation message, the client will end. The server will keep running.

Credit allocation:

Documentation: (40% total)

- Protocol: 20%
- Test cases: 20% (no credit if the application does not work)

Programs (including comment) and others: 60%

Note:

- If your protocol does not match the implementation, no credits will be given.
- If your program cannot compile or crash immediately when it runs, you will receive no credits for the programs and test cases (total 80%).
- Segmentation faults, freezing, infinite loop or unsatisfactory commit history will result in significant loss of credits of those parts.