APPLICATION PROTOCOL

Architecture

Client-server architecture is used for the network application. Server is given an IP address which does not change throughout the session. They need to establish connection before they can communicate. Client requests services from the server and server responds to it.

- **Types of Messages**
  Request: After a connection is established between the client and the server, client sends a request to the server.

  Response: To answer to request of client, the server either sends client data or do the data translation job.

- **Syntax and Semantics of message type**
  - **Request**
    Format of the message
    [file size] [data] [file format] [output filename size] [output filename]
    [file size]: size of long
    [data]: 1000 bytes of ASCII characters
    [file format]: size of int
    [output filename size]: size of int
    [output filename]: (filename size) bytes of ASCII characters

    Meaning of each field in the format

    [file size]: size of the file
    [data]: data of the file stored in buffer
    [file format]: format number that specifies the different types of translation
    [output filename size]: size of the output file name
    [output filename]: name of the output file where the translated data are written

    The value range of each field

    Size of primitive data types may differ in different platform
    [file size]: 0 to 4294967295
    [data]: file size
    [file format]: -32768 to 32767 or -2147483648 to 2147483647
    [output filename size]: -32768 to 32767 or -2147483648 or 2147483647
    [output filename]: output filename size

  - **Response**
    Format of the message
    [file size] [error message]
    [file size]: size of long
    [error message]: size of int

Meaning of each field in the format
[file size]: size of the file received by the client
[error message]: integer that tells whether the format of the units in the file was ok or not

The value range of each field
[file size]: 0 to 4294967295
[error message]: 0 or -1

- **Rules of sending messages**
  - Client always sends the request to the server
  - Server responds to the request of the server
  - There is always one response for one request.
  - Messages must follow the format and syntax mentioned above.
  - If not followed, the program will throw an error.

TEST CASES

Input	Expected Output	Actual Output	Rationale	Content	Errors
practice_project_test_file_1	Success	Success	Normal functionality	Data from test file 1 from practice project	No error
practice_project_test_file_2	Success	Success	Normal functionality	Data from test file 2 from practice project	No error
largeFile	Format error	Format error	Sending and receiving file larger than buffer size.	Data in largeFile.txt	No error
wrongType	Format error	Format error	Testing for the incorrect type (incorrect format)	Data in wrongType.txt	No error

USAGE OF CLIENT AND SERVER PROGRAM

- **Usage of Client**
  The following command invokes the client:
  <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 of the server
  - <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 possible the hierarchy of directories.)
  - <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, 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

  If there are not enough or more than enough arguments, format is not in the specified range (0 to 3), then the program is terminated. After creating the socket, setting the remote IP address, connection to the remote echo server, the client sends the file size to the client after calculating the file size. Followed by that, client sends data in the file to the server. Thereafter, client sends file format to the server. Client sends size of the output file name and also sends the output file name.

  Finally, the client receives the error message from the server. 0 is a confirmation message saying that the data was translated and saved in the destination file by the server. -1 means that there was an error translating the file. Finally, the client closes the TCP connect and returns EXIT_SUCCESS.

- **Usage of Server**
  The following command invokes the server:
  <server> <port>
  If enough arguments (2) are not passed, then the server program is terminated. After the necessary setup is made (creating listening socket, filling relevant data members in socket address structure, binding socket address to listening socket), the program enters into an infinite loop to listen to client requests. The client will send the units in the file from the path 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 type1 and type 1 units to type 0, then save them to the specified file in the directory which the server application is in and send a confirmation message and close the connection.

INSTRUCTIONS TO COMPILE CLIENT

The client can be complied by the following command in the command line:

gcc <client.c> <helper.c> <helper.h> -o <filename>

where,

- <client.c> is the client c file
- <helper.c> is the file that contains the implementation of helper functions
- <helper.h> is the file that contains the definitions of helper functions
- <filename> is the compiled executable file name

INSTRUCTIONS TO COMPLIE SERVER

The server can be compiled by the following command in the command line:

gcc <server.c> <helper.c> <helper.h> -o <filename>

where,

- <server.c> is the server c file
- <helper.c> is the file that contains the implementation of helper functions
- <helper.h> is the file that contains the definitions of helper functions
- <filename> is the compiled executable file name

KNOWN PROBLEMS

As of now, there aren't any known problems.

SINGNIFICANT REFERENCES

For the helper functions, significant references were made from the following link:

www.paulgriffiths.net/program/c/sockets.php

GITHUB LINK

You can go to the following link to access the network program.

https://github.com/henchhing-limbu/Socket_Programming