Design Implementation and performance explanation

CREATED BY: Anuj Rastogi

anujrast@buffalo.edu

Design for HELP: As soon as someone types the help command it displays the list of available commands. At the backend it calls a custom function Help() which enlists all the commands included as a part of application.

Design for Display: As soon as we call the display command the program gets the host name and gets the getaddrinfo() to display the IP. The port is fetched from the user input.

Design for LIST: Both the server and the clients maintain the list of their connection. As soon as a LIST command is typed on the server it displays the connections with the server. On the client whenever this command is typed we will get the list of connections the client already has and the connection LIST from the server. Both server and client maintain a struct and based on the Connections established we update that array of struct and the corresponding socket descriptor. The select API monitors the socket descriptors and hence, according to the socket activity it sends the struct from the server to the client. To convert the array of struct to the string a custom function is used namely ListGenerator().

Design for CONNECT: The connect command first passes the requested IP to the server and this is checked if this is listed in the server struct containing the list of details. If the affirmation is achieved then this CONNECTION with the peer is established. Also, the program contains the flag checking if the client is registered at all with the server or not and displays the appropriate error message correspondingly. There is a use of a custom function GetIP() to convert the name format to IP. The connected peer is added to the client struct.

Design for REGISTER: As soon as the REGISTER command is done the client registers with the server. On registration the REGISTER flag is updated to 1 enabling CONNECT and LIST command.

Design for GET: On GET the file name is sent to the connected peer. First a validation takes place assuring if the peer is connected. If so it is checked if the peer to transfer the file is SERVER. In that case the request is rejected. If all validations are true then the file name is sent to the peer and it is checked

If the peer has the file name. If so the file is read using the getc() function and an array of char is created which is then sent to the requester and pasted after opening the file using fputs().

DESIGN for PUT: ON put the file name is checked on the peer side and if no file is found then no action is taken, If it is an existing file then the file is loaded in the stream using fputc() and converted to string. This string is then written at the other side.

PERFORMANCE IMPROVEMENTS:

1. The program contains array which could have been declared in dynamic way to prevent pre occupation of memory.
2. Along with the above, as of now the program is only capable of transmitting just a few Kbytes. The constraint is because we are transmitting the data after storing in a string. This can be significantly improved by using a chunk wise approach.

IMPROVEMENTS:

The chunk wise approach will transmit chunks at a time. This will not require one transmission but a packet wise transmission. The latter is much more effective and can help transmit huge bytes of data.