COMP3331 ASSIGNMENT REPORT
DIGITAL FORUM PROTOCOL(DFP)
Z5284381(SHUBHAM JOHAR)

PROGRAM DESIGN-

For the purpose of this assignment I used python3.7, the code was divided into two major directories/folders and had four files each, each of them corresponding to similar feature/functionality in both the ends. The files in each directories were as follows-

Client

- client.py – Main file used for instantiating the interaction between the client and server uses <Server IP> <Server Port> as command line arguments. The file just does a basic error checking for the command line arguments and then creates a Client class followed by beginning the interaction with the server using the begin method of the class.
- client_manger.py – This is basically the file with the implementation of Client class which starts off by beginning the interaction with the server and then goes off to authentication->forum_interaction->command_exec_funcs all of which are embedded as functions within the class.
- client_process_commands.py – It is a file where all the responses to the given commands are handled. It has different functions for each command which handles variety of responses from the server and tells the client of the response on the terminal.
- DFP_CONSTS.py - This is a special file which will be present at both the client and server directory. It consists of all the constant string values and other related constants required for the interaction between the server and client. This is where the real mechanism and constants for the Application layer protocol lies, it is named DFP_CONSTS because the protocol controlling the interaction is named DFP from my side as I thought what is better than just taking the acronym for Digital forum protocol as the formal name for the protocol.

Server - The server on the other hand also has four files under its directory which are named as server.py, server_manager.py, server_process_commands.py and DFP_CONSTS.py. Each of the files at the server end does the similar jobs just as their client counterparts, so the explanations defined above also satisfies for the server just switch over the roles for clients with server for each of the files and that is what each file at the server end does and the entire flow of commands is maintained/ managed by the constants defined in DFP_CONSTS.py

Please Note: When you are running the files try to run it in two separate directories, the submission for assign.tar that I have given has two sub directories of client and server which contains all the files that are needed for both the end systems to work, please make sure that while running the program DFP_CONSTS.py is available under both the directories.(my submission has it available in both the directories) otherwise one of the executables will not work properly.

APPLICATION LAYER MESSAGE FORMAT

For the purpose of communication between the server and client, I have majorly used string constants, all of which are defined in the file DFP_CONSTS.py they use the mixture of concepts related to HTTP and TCP FIN and SYN interaction, with the major advantage that there is no need for string parsing the way it is required with HTTP. Here basically all the constants work as the current state managers at both the ends of interaction. For certain command functionalities(RDT and LST) where just the constants were not enough for the interaction I used json for serialisation and easier flow of data for the interaction, in general for the part of the interaction the server or client first send a

constant indicating the beginning of a certain activity and then clients accepts the response and then again start executing and sends back the response to the server. Some of the constants defined in DFP_CONSTS.py are as follows-

FORUM_START = 'DFP 2024 INTERACTION WITH FORUM BEGINS'
INVALID_COMMAND = 'DFP 1024 THE COMMAND ENTERED IS NOT VALID'
COMMAND_FOUND = 'DFP 2020 VALID COMMAND'
PROCESS_COMMAND = 'DFP 132 processing the current command'

The flow of execution happens when server sends the client FORUM_START, the client sends back the server with a command, and it keeps sending the command until the command is found at the server end, as soon as it is found the server responds with command found and then the processing of command starts when the client sends the server PROCESS_COMMAND where the specific function for the command is called and again the similar type of interaction happens between the client and server.

BRIEF DESCRIPTION OF HOW SYSTEM WORKS

The system interaction begins off by firstly starting the server by writing-
Server.py <Server_Port> <admin_password> which creates the server object and starts waiting for the connections it is always listening on the address 127.0.0.1:(Server_Port),
Then it accepts the connections as they come which primarily comes off by running the script client.py <Server_IP> <Server_Port> that connects the client with the server. The server uses a multithreaded approach and spawns off a new thread for every new connection it gets. Now, the interesting part of this implementation is that it creates an extra socket when the server class is initialised i.e it creates one general socket which handles the basic request response interaction with each of the clients that are connected but for it also creates an extra socket where it is constantly communicating with the client to tell that it is alive(Heartbeat packet implementation) this extra socket was basically added for making the implementation flexible and robust with the SHT command which required all of the clients to be notified when the server got closed, and for the similar feature we created an extra socket per each client which was constantly sending 'are you alive' message to the other server socket which is always listening on the <Server IP > <Server Port + 1>. So, the begin() method of each class is called to initialise these sockets and then create a separate threads for managing the request response by moving to the manage client function and a separate thread for managing the alive responses in the alive_handling thread. When it reaches the manage client thread it begins off by authenticating and then follows this set of methods- authentication->forum_interaction->command_exec_funcs which gets the command process the commands through their independent function and also maintains the state through the SERVER_STATE dictionary which has all the states of Threads and files uploaded in their specefic list of dictionaries of threads and files on the other hand with the clients, the steps of execution of the functions is quite similar its just there is not any extra state management, the server also maintains a list of active users (i.e the users which are currently communicating with the server on the major thread) and also a list of wating users which is the list of sockets to whom server is sending the 'I am alive messages'. All the specific commands also handles the error handling using the MESSAGE FORMAT described above and sends specific messages if something fails and then client responds according to it.

DESIGN TRADE-OFFS MADE AND CONSIDERED

The major design trade off considered for this program was to use multithreading for all the individual client sockets and also for the sockets received for the parallel I am alive thread. In addition to using the multithreading I also used mutex_locks wherever required for managing the thread safety and managing safe access to the shared data structures and to prevent race conditions while writing to the files and maintaining the state at the server. I majorly used object oriented approach at both the client and server end since each client and server had separate states to manage so using the object oriented

approach seemed like a great stylistic choice for it. Most of the state management was done within the server class where everything was appended to either a list created as a part of the object and the state management for the Threads and files was done in the SERVER_STATE dictionary which was stored in the server_process_commands since there was no need of storing it within the Server object and it was only required during the forum interaction period. For the message sharing I used a combination of json and Constant strings. For every command execution function I firstly go on by doing all the error and exception handling and then execution of function with the SERVER_STATE updation and then sending the client with the Success message.

POSSIBLE IMPROVEMENTS AND EXTENSIONS TO YOUR PROGRAM AND INDICATE HOW YOU COULD REALISE THEM

For the major part of the project I was pretty happy with the outcome that I got but I would definitely like to improve certain things with my project one of them being using string constants for application level protocol although using it is pretty easy and makes the code very robust but the strings are not very scalable i.e if I wanted to extend the functionality of one of my interactions I would have to most probably create a new string for that, which after some time could lead a lot of overhead, better approach would be to use something like json or pickle for every piece of communication and just generate a interaction message dynamically as we go within a predefined framework of interaction that makes it much more flexible. Other than that I also feel that creating a single thread for each client and then creating two threads and two sockets per client on the clients side is a bit of overkill, I would definitely like to improve it by using thread pools and assigning only a handful of threads the management of clients or maybe also using something like event driven programming and select for the efficient message sending between the clients for making the SHT work. I would also like add a functionality for removing a file that has been uploaded since to remove a file from the thread the only way right now is to remove the thread itself which is something that will not exist in the realistic setting.

If your program does not work under any particular circumstances, please report this here- I am really glad and happy to tell that thankfully after a lot of efforts and hard work my program worked for the most part of what was expected out of us, it works perfectly fine for a single client under all circumstances and handles all the errors gracefully for every command, and for the multithreaded version as well the program works for all the functions taking into consideration that all the interactions are atomic, the program also works with locking the common resources but just under the case when one user starts the uploading of a file which is quite huge(around 512mb) and in the middle of that if another user tries to write a message to the same thread where file is being uploaded then the adding message is blocked until the file is uploaded since we lock the resources during the execution of command and consider the atomicity of commands but as soon as the file is uploaded just instantly after that or just at that particular instance of time the message gets added to the file and the SERVER_STATE and when we use RDT to see the messages and files the order is maintained properly, so just wanted to clarify this that since the resources are locked if multiple things are executed at the same time it takes into consideration that until the first operation on the common resource's lock is released the other operation is locked but instantly after that it is executed so just wanted to clarify this behaviour of the implementation. Also if any user writes SHT with the correct admin password and someone other client is in the middle of downloading the file then the all the clients are shut down instantly irrespective of whatever they are doing and a shutdown message is send to all the clients making it feasible and flexible according to the spec.

ACKNOWLEDGEMENT FOR THE SECTIONS OF BORROWED CODE-

- Used this video as a reference for the UPD command- https://www.youtube.com/watch?v=LJTaPaFGmM4
- Used this link for a bit of reference on the EDT command- https://www.reddit.com/r/Python/comments/464cim/replace_a_line_in_a_txtfile/

- Used this as a starting point for the RMV command- https://stackoverflow.com/questions/2012670/deleting-files-which-start-with-a-name-python
- Use this link as a reference for a certain section in the DLT command- https://www.geeksforgeeks.org/python-concatenate-two-lists-element-wise/
- Used the multithreaded server program on webcms3 as as a starting point for my server.