COMP3331 Assignment Report
Term 1, 2021
By Boqian Ma, z5260890

Content

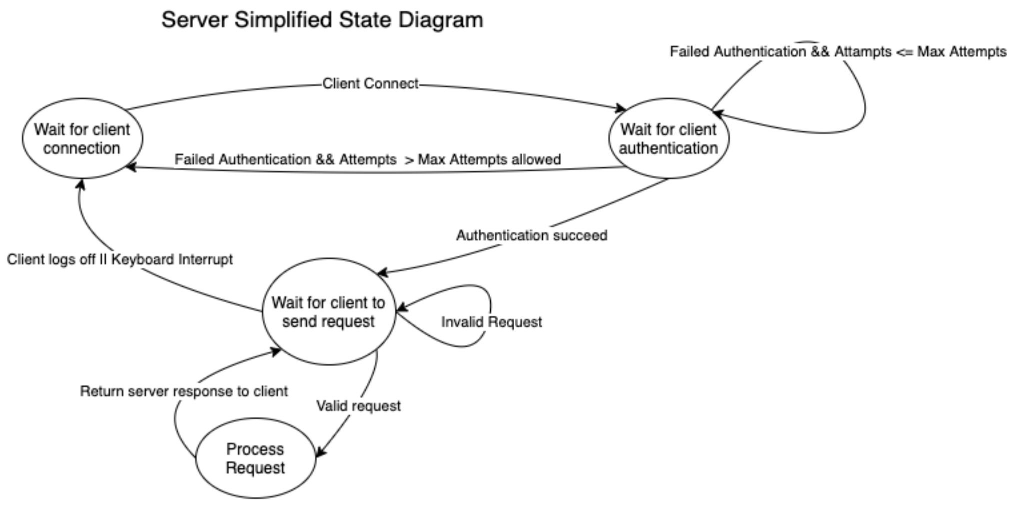
1. Program Design
2. Application Layer Message Format
3. Design Trade-offs and considerations
4. Potential Improvement
5. Possible Extensions
6. How to run

Program design

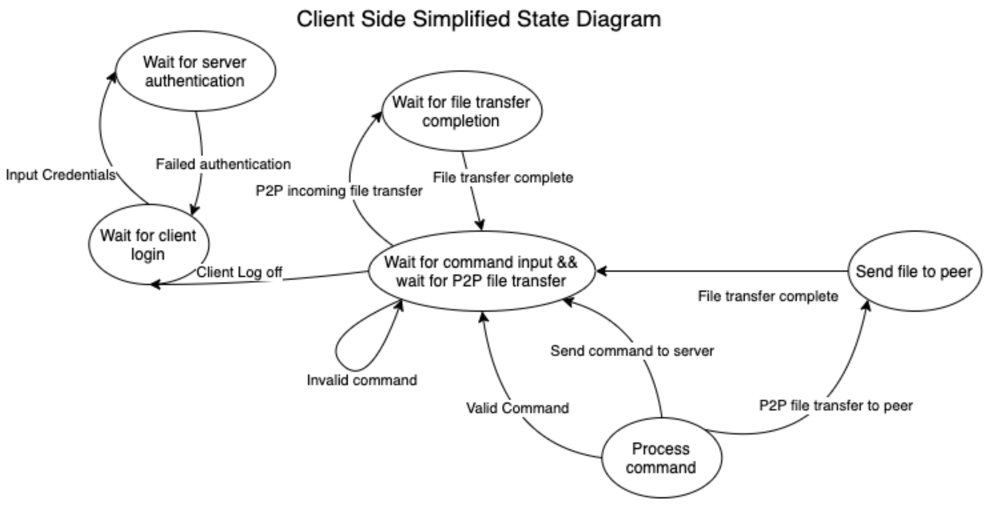
Available commands: MSG, DLT, EDT, RDM, ATU, OUT, UPD

State Diagram

The following state diagrams are simplified.



Server Simplified State Diagram
Failed Authentication && Attampts <= Max Attempts
Client Connect
Wait for client connection
Wait for client authentication
Failed Authentication && Attempts > Max Attempts allowed
Authentication succeed
Client logs off || Keyboard Interrupt
Wait for client to send request
Invalid Request
Return server response to client
Valid request
Process Request



Client Side Simplified State Diagram
Wait for server authentication
Input Credentials
Failed authentication
Wait for client login
Wait for file transfer completion
P2P incoming file transfer
File transfer complete
Client Log off
Wait for command input && wait for P2P file transfer
Send file to peer
File transfer complete
Send command to server
Invalid command
Valid Command
P2P file transfer to peer
Process command

24/04/2021

COMP3331 Assignment Report
Term 1, 2021
By Boqian Ma, z5260890

Command line output reference

General errors: `[ERROR] {message}`

Server:

`handle_client(conn, addr):`

- New client connection: `[NEW CONNECTION] {addr}`
- Client logins succeed: `[LOGIN SUCCEED] {addr}`
- Client closes connection: `[CLOSE CONNECTION]`

`handle_requests(msg):`

- Client issues command: `[INCOMING COMMAND] {current_user} {command}`
- Server response message: `[SERVER RESPONSE] {response}`

`Main:`

- Server exits: `[EXIT SERVER]`

Client:

`connect(tcp_client, UDP_ADDR):`

- Welcome user: `[WELCOME] Please Login...`
- User logs out: `Logging out…`
- Server response message: `[SERVER RESPONSE] {response}`

`udp_server_setup(UDP_ADDR):`

- Server set up complete: `[UDP SERVER LISTENING] {message}`

`upd_recv_handler():`

- File received: `[FILE RECEIVED] {message}`

`upd_send_handler():`

- File sent successfully: `[FILE SENT SUCCESSFULLY] {message}`

Application Layer Message Format

Authentication Phase

During the login phase, a client sends the following message to the server:

- `username + " " + password + " " + client_udp_address`

Server returns the following responses

- if login succeeded: `"LOGIN: SUCCESS"`
- if login failed: `"LOGIN: FAILED"`
- if login attempts reached maximum: "LOGIN: REACHED MAXIMUM ATTAMPTS " + seconds_left_until

After authentication

Client cleans up command string before sending it to server, where the command string is processed and a response is generated per requirement.

Client Logout

Client logs out from server by sending the following messages:

- if log out from command line: “OUT”
- if log out from keyboard interrupt: “OUTX”

Client will not receive a server response if it logs out by sending “OUTX”.

Design Trade-offs and considerations

Server

- Locks are used to prevent global data structures update anomaly.
- Implemented threading to allow concurrent users connection.

Client

- Most requests pre-processing (request message formatting) is done on the clients’ side to reduce server workload.
- A peer can receive from multiple peers at once through threading.

24/04/2021

- When a file is transferred, it is divided up into multiple packets indicated by “UDP_PACKET_SIZE”.
- The first UDP packet contains the file name and the second UDP packet contains the number of packets in this file. It is assumed that the size of the file name and the size of the number of packets are smaller than “UDP_PACKET_SIZE”.

Potential Improvement

Server

- Currently, the server shuts down by KeyboardInterrput. An improvement could be adding a command line keyword such as “OUT” to indicate whenever a server wants to exit.
- Currently, once the server exits, none of its clients are informed. An improvement for this is to close all connections when the server exits.
- Currently, the messages are not persistent, meaning whenever the server restarts, the messages from the previous session are lost. An improvement is to load the messages from the previous session whenever the server starts.
- Current system can only handle one chat room at a time. An improvement can be made such that multiple chat rooms can be hosted on the server at once.
- Currently, the log files only contain the current state of the system such as who is online and which messages are present right now. This can be extended such that all actions are logged and sustained instead of modified.

Client

- Currently, whenever an UPD command is made, an ATU request is made. This can be improved by caching a copy locally whenever an actual ATU request is made.
- Currently, in UDP, each user peer can only send one file to one peer at a time. This can be improved by introducing a threading mechanism to send multiple files at a time.
- Currently, in UPD, it is assumed that the first two packets will contain the filename and the number of UDP packets in this file contains. However, this is not ideal because packet loss may occur in UDP. An improvement is to use TCP to transfer essential data and use UDP to transfer the rest.

General improvements

- Currently, the system can only handle KeyboardInterrupts. This is unsafe because other interrupts or exceptions can still happen and disrupt the flow. Better error handling is required for better user experience.

Possible Extensions

1. General Security Extensions

The current design has no security features. Possible additional features include

- Encrypting credentials.txt, messagelog.txt
- implementing public and private keys for safer authentication

2. Multiple Chatrooms on a Server

The current server design can be improved by introducing multiple chatrooms together.

How to run

Start Server: `python3 server.py localhost <Server TCP port> <max login attempts>`
Start client: `python3 client.py localhost <Server TCP port> <client UDP port>`

Please refer to specs for specific command formats.