Computer Networks Final Project

Protocol Specification

Client Side

Format:

- Client sends: `request`
  - Server responds: `response`

- Login: `LOGIN\n<username>\n<password>`
  - Success: `LOGIN SUCCEEDED`
  - Failure: `FAIL`
- Register: `SIGNUP\n<username>\n<password>`
  - Success: `SIGNUP SUCCEEDED`
  - Failure: `FAIL`
- Get list of users: `GETUSERLIST`
  - Response: `<userA>\n<userB>\n....<userN>\n`
- Get mailbox: `GETMAILBOX`
  - Success: `<From <user>\n<message>`
  - Failure: `NO RECORD`
- Send a file:
  - Part 1: specify the file size and file name:
    `SENDFILE_START\n<target_user>\n<filename>\n<filesize>`
    - Does not wait for response.
  - Part 2: send()/recv() for $\lceil filesize/bufsize \rceil$ times:
    - Client: `SENDFILE_TRANSFER\n`
    - Server: `OK`
    - Client: `<ith_message_segment>`
- Get list of (user's) files: `GETFILELIST\n`
  - Response: `<fileA>\n<fileB>\n....<fileN>\n`
- Get a specific file: `GETFILE\n<filename>`
  - Part 1: server responds with the file size: `<filesize>`
  - Part 2: send()/recv() for $\lceil filesize/bufsize \rceil$ times: `<ith_message_segment>`
- Send a message (limit: 500 characters per message):
  - Client: `SENDMSG_START\n<target_user>\n<msg_size>`

Server side

- Send to target_user: `<src_user>\n<msg_size>\n`
  - if online => send directly and save record in two `.record` files in the dirs of two users
  - if offline => save record and also save in an `.unsend` file

User & Operator Guide

Client

In general, the instructions are shown on the screen. Simply type the number of the desired command. Note that if you do not type in a valid number, it is possible that the program will malfunction.

To send a file to another user, the client must first place the file in their own directory, named `<username>_fire_dir/`.

Server

The debug messages are shown on the screen. In general, you simply need to make sure the server stays alive.

Instructions on how to run server & clients

- Server: type the following commands:
  - First time: type `make clean` to initialize everything.
  - Afterwards:

```
make server
./client <IP> <port>
```

- Client: type the following commands:

```
make client
./client <port>
```

The client must know the **IP** of the server beforehand.

For simplicity, you may modify the `PORT` and `SERV_IP` variables in makefile, and then do the following:

- Server: `make rs`
- Client: `make rc`

System & Program Design

This system is implemented using a client-server architecture.
The server is assumed to always be on and its IP and port number is known. It recieves requests from clients, handles the request, and responds accordingly.

The client first connects to the server, and attempts to login or register. If the login is successful, the server creates a thread to handle requests from that specific client.

In order to obtain files and messages, the client will send a corresponding command to the server, who will then reply with the desired data. This includes historical and current messages and files.

To send a message to another online user, a client will first send the message to the server, who will then immediately notify the target user and deliver the message. The server also stores this record for later retrieval by the client. If the target user is not online, then the server keeps this message and sends it to the user when the user logs in.

File sending is explained in the user guide above.