**Report**

The project consisted from 3 parts as it was described in the specifications of the project. These are my approaches to the 3 parts (by part):

1) First, I have directly thought about the data structure to store the tags and registered users, to know who is online, and which tags are registered. I have decided to implement linked list of linked list, because it’s not efficient to store everything the array due to large number of ports (getdtablesize), and it is better just to remove node, when the user or tag is removed. So, I have linked list of users with pointer to the linked list to the corresponding tags that belongs to the user.

When I register a tag, due to late usage of GUI client on moodle, I need to write “#tag” instead of “tag” in the GUI client. However, on my client that works fine. When I registerall, I look for the existing tags in every user’s tags, and add it. Also, even if I don’t have any tags, I’ll simply change Boolean value responsible for showing is the user registered for all messages or not. When I deregister tag, I simply remove the node from the linked list of tags (of the user). When sending msg to the client I look for the users that need to receive that message, and put their fd in the write function call. Moreover, I extract word MSG and tag from the original message. If user types some garbage, the server responds as “Oops!...”

2) In the second part, it was very convenient to have rc4.c file provided via the link in project specifications. I have just added it to the Makefile, so that it had a header, and I simply could use functions from there.

I decided to enter the key every time I enter the client, so you will be asked to enter it only once. The key is used to form stream, and then I form a stream of length the same as my encrypted message. Further, I just as it was said need to XOR the stream and my word typed. The server just receives the message as it is, extracts a tag if it exists, and redirects that message to the users that need to receive it (according to the protocol). Even though this part was easy, I have special case when encryption does not work: the key to encrypt/decrypt = “Key”, the message to be encrypted “How ya doin”, and similar starting from “How”. When the client receives the message, he then decrypts the message with the same key (or different, then it will have wrong decryption algorithm), so that I repeat the same process.

3) This part the toughest one, because the images may be of large size, and for sending such messages we need to have threads in server to handle requests.

In my server, when the client sends an image, I start a new thread for reading the image, so that server could handle other requests too. For the time the thread is running in background, I remove that client’s socket from fd\_set until the thread is finished, so that server would ignore the client for that time.

In reading thread, I read image in chunks of 2048 bytes, because they are coming like so from client. Then, I add each chunk in a loop to the buffer where I store whole image. After I have read all image, I start new thread for sending to clients (one thread per receiving client). In writing thread, I just simply forward whole buffer to the client.

Everything works properly here, but I can’t send images with tags. Some of older versions (without threads) have been receiving messages with tags. Now, it is difficult to extract the tag. For this part, there are some bugs, so for convenience please let me come to the live grading.

You may want to visit my github repo to see the work flow from the beginning by this link: <https://github.com/ardulat/Operating-Systems>

P.S. this project took me 2 weeks of sleepless nights, and 37 commits in github in total to finish it (last 3 rows in the right):

