

Operating Systems - Assignment 1 (Client-Server Model and Threads)

Motivation:

In class, we have taught the client-server model and threads. This assignment asks you to use both techniques to develop an on-line chatting application (e.g., LINE).

Assignment Content:

In this assignment, you need to create a server and some clients. When a client wants to send messages to another client, it logs in to the server first. The server maintains the mapping between each username and its socket address. When a client logs in, all other active clients see that the client is on the line. They are also informed when the client logs out. In addition, two clients can send messages with each other. The server takes charge of relaying messages between them. Here is an example. Suppose that there are two clients, say, Alice and Bill. Alice has logged in to the server. Then, Bill tries to connect to the server by typing the following command:

```
$ connect socketAddr (192.168.1.1, 1234) user Bill
```

The server's IP address is 192.168.1.1 and its port is 1234. Suppose that the IP address of this client is 192.168.2.3 and its port is 5678. The server then maps 192.168.2.3 to user Bill. In addition, Alice can see the following information:

```
<User Bill is now on-line, the socket address is (IP: 192.168.2.3, Port: 5678).>
```

If Alice wants to talk to Bill, she can use the following command:

```
$ chat user Bill message "Hello, Bill."
```

When Alice talks to a guy that does not exist in the system, the server notifies her:

```
$ chat user Goblin message "Hello~"
```

```
<User Goblin does not exist in the system.>
```

When Bill wants to leave the system, he can use the following command:

```
$ leave
```

In this case, all other on-line clients need to be notified:

```
<User Bill left the system.>
```

In this case, if Alice wants to talk to Bill, the following case happens:

```
$ chat user Bill message "Are you on-line?"
```

```
<User Bill is now off-line.>
```

Whenever a client connects to the server, the server creates a thread to serve that client. The server maintains a "whiteboard" (i.e., a memory space) for communication. If a client A wants to say something to another client B, A's thread writes words on the whiteboard, the server notifies B's thread, and B's thread reads these words and sends them to B. According to the previous example, the whiteboard's content may be as follows:

Communication	Whiteboard
Alice: \$ chat <i>user</i> Bill <i>message</i> “Hello, Bill.”	[2026 Sept. 30, 15:20:45] Alice used the whiteboard and wrote the following message: <To Bill> Hello, Bill.

On the server side, your program needs to show the whiteboard’s content in real time. Just like what OS should do, Since the whiteboard is shared by all threads, the server must handle **synchronization**. If two clients are sending messages concurrently, the server must ensure that the whiteboard can be written by at most one thread at any time. You need to demonstrate this behavior to TA and show somethings on the whiteboard.

Requirements:

- You need to use UNIX socket programming and Pthread in this assignment.
- You must provide a makefile. TA will deduct your points if there is no makefile or the makefile is erroneous.
- You must submit a README file along with your program. The README file should briefly describe how you write your codes (for example, the idea of your program).
- You must demonstrate your program. TA will announce the demonstration time.

Grading Policy:

You need to submit your codes and demonstrate your program to TA. The due day of this homework is **11/4**. You will get **no point** if you do NOT demonstrate your program (even if you submit codes). Discussion among your classmates is encouraged. However, plagiarists (including using AI tools) obtains **ZERO point**. Below are the points you can get in this homework:

- Main program: 90%
- Code comments & README: 10%