Network Applications Programming - Homework 3
(On-line chatting service)

Motivation:

You have learned how to develop iterative & concurrent servers in class. This homework asks you to develop an on-line chatting service (e.g., LINE) by using a concurrent server with one thread.

Homework Content:

In this homework, you need to develop one server and at least **THREE** clients. Clients cannot talk to each other directly. Instead, the server has to relay messages for them. In the on-line chatting service, you have to implement the following items:

- **User login and notification:** When a client wants to send messages to another client, it has to login to the server first. The server should maintain the mapping between the user name and its socket address. When a client, say, Bill logins, all other active clients will see that Bill is on the line. Of course, when Bill logouts, these clients are also notified.
- **Unicast communication:** Two clients can send messages with each other. The server is responsible for relaying messages between them.
- **Multicast communication:** A client can send messages to more than two clients.
- **Off-line message:** An active client can send off-line messages to an inactive client. When the latter becomes active (i.e., login to the server again), these messages will send to that client.

Below are some examples. Suppose that there are three clients, say, Alice, Bill, and Caesar. Alice and Bill have connected to the server. Then, Caesar will try to connect to the server:

$ **connect** 192.168.1.1 1234 Caesar

Here, the server's IP address is 192.168.1.1 and its port is 1234 in this example. The last parameter indicates the username of the connected client. Suppose that the IP address of this client is 192.168.2.3. Then, the server has to map 192.168.2.3 to Caesar. On the other hand, Alice and Bill will see the following information:

<User Caesar is on-line, IP address: 192.168.2.3.>

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

$ **chat** Caesar "Hello, Caesar."

Of course, she can talk to both Bill and Caesar:

$ **chat** Bill Caesar "Hello, guys."

Thus, your client program has to know whether this is a unicast or multicast communication from the parameters. When Alice talks to a guy that does not exist in the system, the server has to notify her:

$ **chat** HandsomeGuy “Hello~”
<User HandsomeGuy does not exist.>
When Caesar wants to leave the system, he can use the following command:
$ **bye**
In this case, all other on-line clients have to be notified:
<User Caesar is off-line.>
In this case, if Alice wants to talks to Caesar, the following case will happen:
$ **chat** Caesar “Lend me some money.”
<User Caesar is off-line. The message will be passed when he comes back.>
Later when Caesar logins again (using the same IP address), he will get the following information:
$ **connect** 192.168.1.1 1234 Caesar
<User Alice has sent you a message “Lend me some money.” at 03:00PM 2021/4/28.>
You can use some functions such as time() and ctime() to obtain the time information.

In this homework, you have three choices to implement the server:

- Apparent concurrency (i.e., using select()): 100%
- Multiple threads: 120%
- Multiple processes: 150%

Requirements:

- You need to use UNIX socket programming in this homework.
- You have to 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 have to 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 **05/19**. You will get **no point** if you do NOT demonstrate your program (even if you submit your codes). Discussion among your classmates is encouraged. However, plagiarists will get **ZERO point**. Below are the points you can get in this homework:

Items	Points
Socket connection (i.e., connect and bye commands)	15%
Off-line messages	15%
User name & IP mapping	10%
Chat command (unicast & multicast)	40%-90%

(1) select(): 40% (2) Multiple threads: 60% (3) Multiple processes: 90%	
User interface (for example, welcome message or help command)	10%
Code comments & README file	10%

Total: 100%, 120%, 150%