Important Notes:

- This lab has automated evaluation.
- Do not include unnecessary printf statements. Only print the strings mentioned in question.
- Any strings which are required to be sent by the client or server must be exactly as specified.
- You'll receive a script containing sample test cases, and your submission will be evaluated against hidden test cases to determine the final score. So, please refrain from hard coding solutions.
- The maximum size of strings sent and received is 1024 bytes
- Store the server.c code file, the client.c code file, and log.txt file in a directory named “impl” and zip this folder to create a file named <id_no>_lab4.zip (The zip file should only contain the “impl” folder with the two c files and nothing else)

  NOTE: Test client and server using the client_test.py, server_test.py files respectively in the “eval” folder. Both these python scripts take ip of the server as the first command line arguments and the server port as the second command line argument.

In this lab, you will be enhancing the capabilities of your polling program and implementing multi-client chat.

- You must take the number of clients via command line argument when initializing the server instead of using only 3 clients. Server will wait for all clients (number of clients given by the above method) to connect and then only start polling. The number of clients initialized should be the same as the number given to the server. Every client can work in either one of two modes (specified by <client mode> in the command line arguments): “passive\n” or “active\n”. Clients that are passive can only receive messages and cannot send messages. Clients that are active can both send and receive messages. **(1 mark)**
-
  - To start the server, use the following arguments: ./server.out <ip address> <port number> <number of clients>
  - To start the clients, use the following ./client.out <server’s ip> <server’s port> <client name> <client mode>
  - Ensure you start the correct number of clients
- After the server asks the client for a username, the server should also ask the client its mode. The server sends each client the “MODE\n” keyword right after it receives the name from the client, as a response to which the client should send their mode(given as command line argument) to the server. As specified previously, clients that are passive can only receive messages and cannot send messages. Clients that are active can both send and receive messages. To ensure this, make sure **that the server will never send “POLL\n” to clients in passive mode.** Therefore these clients should not be added to the polling pool. **(1 mark)**
  - Example of the exchange is as follows:
    - Server sends “NAME\n”

        - Client responds with username in same way as take-home (you may assume that the client has a unique name and name isn't the same same as any of the Keywords used)
        - Server sends “MODE\n”
        - Client responds “active\n” (if it is active), else “passive\n”.
- Implement the keyword “MSGC:<destination_client_name>|<message>\n”. **(4 marks)**
    - This keyword means the client wants to send a message to another client through the server.
    - Here destination_client_name is the username of the client that should receive the message. For example, “MSGC:alice|hello\n” (without quotes) means that the current client (e.g., bob) wants to send “hello\n” message to alice.
    - When the server receives this message, it appends the global message sequence number in HEXADECIMAL(this number indicates the number of messages sent through the server using MSGC, for example: for the first message sent using MSGC append 01, for the second append 02, and so on) of the destination client using “|” as the delimiter and replaces the destination client’s name with the sender’s name. It then sends the message to the destination client in the format “MSGC:<source_client_name>|<message>|<global_message_number>\n” where source client is the username of the client that is sending the message. Example: “MSGC:bob|hello|01\n” (without quotes).
    - As soon as the client receives the message from the server the destination client displays the received message as “<sender>:<global_message_number>:<message>\n” on its terminal (here, alice will display “bob:01:hello\n” (without quotes) on her terminal.
    - You may use two scanf() to gather the destination_client_name and the message.

- After the server has sent the message to the destination, it continues with its polling sequence.
- When all ACTIVE clients have exited using the “EXIT\n” command make the terminal close all the sockets associated with the PASSIVE clients. Once this is done the server can exit.
- “LIST\n” modified implementation **(1 mark)**
    - This command is used by a client to request the server for the names of all the clients that are currently connected to the server.
    - When the server receives this command it sends a string containing the names of all the clients that are currently connected to it in the format “<name1>:<name2>:<name3>\n”
    - First the active clients must be sent and then the passive clients in the order they joined.
    - For example: if the clients join in the order: bob(active), eve(passive), clint(active), bertha(passive) then the message sent by the server to client will be “bob:clint:eve:bertha\n”
    - The client on receiving this list must print the list of clients on its terminal. eg., if the server sends client1:bob:eve, then the client prints
      1. client
      2. bob
      3. eve

raw format = “1. client\n2. bob\n3. eve\n”
    - After it has sent the list of names to the client it moves on to the next client as dictated by the polling mechanism.
    - NOTE: The list of names must not include the clients that have already been terminated with the use of the “EXIT\n” command
- LOGGING at the server: **(3 marks)**
    - Create a file log.txt to store the server logs.
    - After a client has joined the server should log “CLIENT: <client_name>; MODE: <mode>” in log.txt
    - When the server receives “NOOP\n”, “EXIT\n”, “LIST\n” from a client it should log “<sender_name>: <CMD>” where sender_name is the name of the sending client and <CMD> is the command sent to the server in log.txt
    - Additionally, for “LIST\n” also log the response (list of names of the clients connected to the server separated by “:”) that is sent to the client in log.txt
    - For “MESG:” log the message in the format “<sender_name>: <message>\n” in log.txt
    - For “MSGC:” log the message from the client in the format “<sender_name>-><receiver_name>:<global_message_number>:<message>\n” in log.txt
    - If an invalid destination client name is provided for MSGC then log “INVALID RECIPIENT: <destination_client_name>\n”
    - For an invalid command log “<destination_client_name>: INVALID CMD\n” in log.txt
    - When the server exits log "SERVER TERMINATED: EXITING.....\n" in log.txt

NOTE: Clients can send a message only when they are polled.