CPE464 - Program #2 Chat Design Work

See Canvas page for due date

You are to upload a copy of your design to Canvas in a PDF file.

Your assignment will not be returned to you before the program is due. We are not that fast in getting assignments graded. Handwritten is fine as long as it is neat.

You may work on this design assignment in groups (up to 3 people). In fact, I suggest you work with a group. It is more fun and hopefully you will understand the problem better when you are done. If you work in a group, you only need to turn in one assignment for the group. All group members must be present for all work (except the final write-up). If you work in a group, you must provide a picture of your group with some flow diagrams in the picture. Attached this picture to the assignment you turn in.

After completing the design assignment, you can no longer work in a group and each student must independently write their own cclient/server programs.

Part I

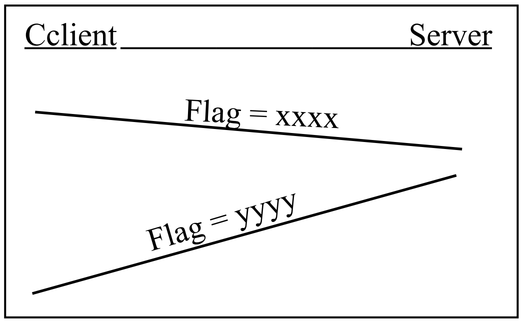
1) Draw up the packet structure – For each packet type (so one drawing per Flag number) draw up what the packet will look like including the header (e.g. length, flag, destination handles, source handle) and payload. Label the figure so that it is clear how big each field is and if the field is in network order. Include both a %M packet with one destination handle and two destination handles.
    a. Identify any fields that must be in network order

2) For each message flow (so message(s) and response(s) if a response is needed) draw up a packet flow diagram (with a description). The description should state which case it is and if there was any error handled and also what processing should be done on the client and server. Include diagrams for cases where errors occur (e.g. handle not found on server).

    There may be more than one diagram for a situation (e.g. case without a problem, case with a problem)

    The message flows you need to deal with are:
    a. Connection setup (sending handle to server)
    b. Sending a message (%M) to a single client
    c. Sending a message (%M) to multiple clients
    d. Broadcasting a message (%B)
    e. Sending messages if the entered message text is 450 characters.
    f. Listing the handles (%L) (complete flow, including flags 10,11,12,13)
    g. Ending a connection (%E)

A packet flow diagram (with descriptions) looks like (in some of your flows you may want the server in the middle between two cclients):



Description of flow: Case where xxxxx happened

Processing on client: Client prints out yyy and goes back to $ prompt

Processing on Server: Server does something with xxxx and responds to client with flag = yyyy

3) Handle Table data structure on the server.

In order to keep track of the active handles on the **server** you will need to have a data structure that somehow maps a handle to a socket number. At some point you will need to define this data structure (e.g. malloc/realloc array, tree, link list…).

a. On the server, the socket and handle can move through a number of conditions/states. Using a table like the one below, explain the conditions/states of your sockets/handles.

Socket/Handle State[1] on the Server	**How can this happen (what can cause this on the server)**	**If you looked at your socket/handle table, how do you know the socket/handle is currently in this state (what in your data structure would tell you this)**
Client socket is opened but handle not valid		
Client Socket open and client handle is valid		
Socket was opened and had a		

[1] By state I mean the condition or status of the socket+handle entry in your handle table.

handle, but then the socket was closed[2]		

b. There are a number of ways your **server** program will interact with this socket handle table (e.g. add a handle to the table), list these interfaces and describe the **inputs/action/output** (think function/method) for each interaction. Use your packet flows to make sure you cover each type interaction you need to make with your table. Think of this as describing the methods in a Java class (I know you are writing in C, but pretend you had an object called SocketHandleTable and you needed to define the interfaces (methods) for this class.)

Part II – Hierarchical drawing and listing common code

This section has two parts:

a. **Hierarchical drawing:** Turn in a function hierarchical (e.g. a tree) drawing of each of your programs (one for cclient one for server). This is a course grain overview of your implementation. Its this function calls this function in a tree format

   Many of the branches in this tree should be at least 4 nodes in depth. The goal is to identify the worker function (lower functions used to support your higher level logic) that are common between the cclient and server and also within cclient and server.

b. Common Code: List the functionality (functions) that is common between the programs

   Looking at your design including your hierarchical drawings, what functionality is common between the two programs (cclient and server)? This is functionality that will be written in a separate code file (.c and related .h) and linked into cclient and server as part of creating your executables (i.e. in your Makefile).

[2] If you implement your server's handle table data structure with an array of structures, this state will happen. If you implement it using some other data structure this state may or may not happen.