Distributed Chat Application

If you weren't here the day we went to lab to try out the distributed chat app, first go download that lab from Moodle and check it out! You will modify the chatter.c program from the lab, and may find the stringManipExamples.c useful as well.

This is not really an exam! You may ask me any/all questions you have, and may use the internet or other resources THAT ARE NOT YOUR CLASSMATES for help on coding in C, using sockets, or whatever else you may wat to know. You should not, however, discuss this at all with others in the class. There are many many ways to implement each part of this program and you should come up with your own unique methods. It's very easy once you hear "a way" of doing something to think of it after that as "the only way", so it's best just to avoid even abstract general discussion about this with your classmates.

Deliverables: Your modified chatter.c program (or a zip folder with all files needed to compile/run your program, if you add others).

Task: Synchronization through token passing

Your goal is to use message sequence numbers and a token-passing system to ensure that all chat participants will receive all messages in the same order, even when there are incoming messages from multiple other chatters. To do this you will need to make 3 main changes to the current chatter program; attach sequence info to each message before you send it, buffer incoming messages and use the attached sequence info to display them in the correct order no matter what order they arrive in, and implement a token-passing method to globally synchronize the sequence numbers of all messages.

1. Sending with a sequence number

The first thing you need to do is to prepend a sequence number to each message before sending it. Look at stringManipExamples.c for examples of converting ints to chars, prepending chars to strings, and removing initial chars from strings.
In the initial code, each chatter queues up messages typed in its send buffer, and spins off a thread for each message to actually send it (with a delay) to all other chatters. You simply need to add a sequence number to each message string before it is copied into the send buffer. At first each chatter can arbitrarily choose an initial sequence number, and then attach consecutive numbers to each message sent, in order. Later you will use the token to set the initial sequence number for each group of messages, and use consecutive sequence numbers for the rest of the messages in that group.

2. Buffered receiving with reordering based on sequence number

Next you need to change how messages are received to actually use that sequence number and display messages in the order they were sent, not received. The first step will be to add a receive buffer so that you can hold multiple received messages at once without displaying them yet to the chatter screen. This can be very similar to the send buffer that already exists, and can also have a length of 10. You should also keep track of the current sequence number of the most recent message you displayed. Then each time a message is received, you check whether it's sequence number is the one after your current sequence number. If it is not, just place (or leave) that message in the receive buffer. If it is the next num, go ahead and display that message, update your cur seq num, and check your receive buffer to see if any messages saved in it can now be displayed in their correct order. Notes:

- you will have to make sure and set the current sequence number when you receive your very first message (it's easy to miss this and end up not displaying any messages!)
- until you implement token-passing, this will only work with 2 total chatters, where you receive the messages from the other chatter, and still just go ahead and print your own messages immediately as the user enters them.

- after you implement token-passing, you will need to wait until it's your turn to send messages, and then "send them to yourself" as well as everyone else, with the correct sequence numbers. You don't actually have an open socket to yourself, so you need to handle this somehow as a special case, but with the same rules as incoming messages from others (if it's not the next sequence number, stick it in the receive buffer).

3. Token-passing system for global message ordering.
To achieve global message ordering, you will add a special "token" message that will be passed around all participating computers, such that only one "holds" the token at any given time. This requires setting up all chatters in some sort of ring formation, so that each is linked to a node to receive the token from and a node to send the token to. This linked ring should handle new nodes joining without losing the token or stalling in its passing from node to node. You do NOT need to handle the case where individual nodes leave the ring. Currently joining nodes already scan all ports in a range and connect to any existing chatters. You can add code to this to insert yourself into the token ring in many ways, as long as you create an unbroken ring, and don't lose the token while inserting yourself. Make sure you create the token at some point, likely when the 2nd chatter joins the network (i.e. a joining chatter sees that there is only one other existing chatter).

The token itself is just a special message that all nodes recognize. For example, it could be as simple as sending the message “Here is the token!”. Of course, in the context of a chat program that would mean none of the chat users could ever type that same message or everything would go wrong, so you should choose a better message. You will also want to prepend a sequence number to the token message, just like any other message.

Each chatter now waits to send the messages queued up in its send buffer until it receives the token, then holds the token while sending all messages in the queue to all other chatters. The sequence number in the token when received should be used to set the sequence number of each sent message so that there is a global ordering across all messages from all users. After all messages in its queue are sent, the current chatter updates the token sequence number to reflect how many messages it sent, and finally sends the token itself along to the next node in the ring.

Note that token passing is not the same as forced ordering of turns. The difference is that if a chatter receives the token but has no message waiting to send, it simply passes the token on, as opposed to waiting until it does want to send something. This means that if all chatters are sitting around not talking, the token should travel around the ring like a hot potato, with each chatter passing it on as soon as it is received.