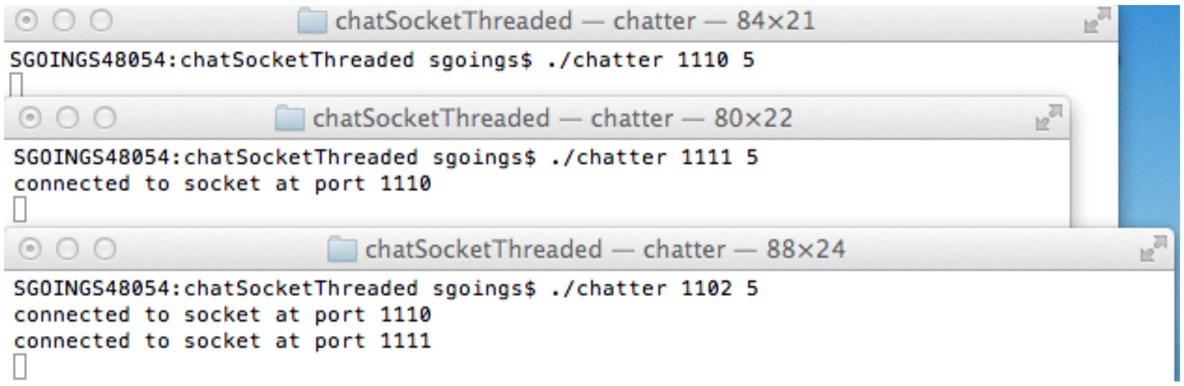
Distributed Chat Application

I have provided you with a program that handles unsynchronized distributed chat in the file chatter.c. Each time you run an instance of this program, it will first search for any existing chatters on ports 1100-1199 and join any that it finds. Then it will set up its own socket on the port included in the command line arguments and continually check for new connection requests on that port, so as to join all future chatters. All running chatters will accept messages typed in their respective terminal window, display these instantly in their own window, and send those messages to all other chatters, who will display incoming messages as soon as they are received (more or less). However, a random delay is added to the time to send each individual message to its recipient, so each chatter may display messages in a different order. The random delay is from 0 to the max allowed delay; the max delay is 2 seconds by default but can be changed for each chatter instance by adding another command line argument when running which is the # of seconds to set the max delay to.

Test the given program thoroughly until you understand how it works, and look through the as well. Here are some things to try to get you started:

1. open 3 terminal windows and start 3 instances of chatter on 3 different ports with a max delay of 5 seconds. For example my 3 terminal windows look like the following after this:

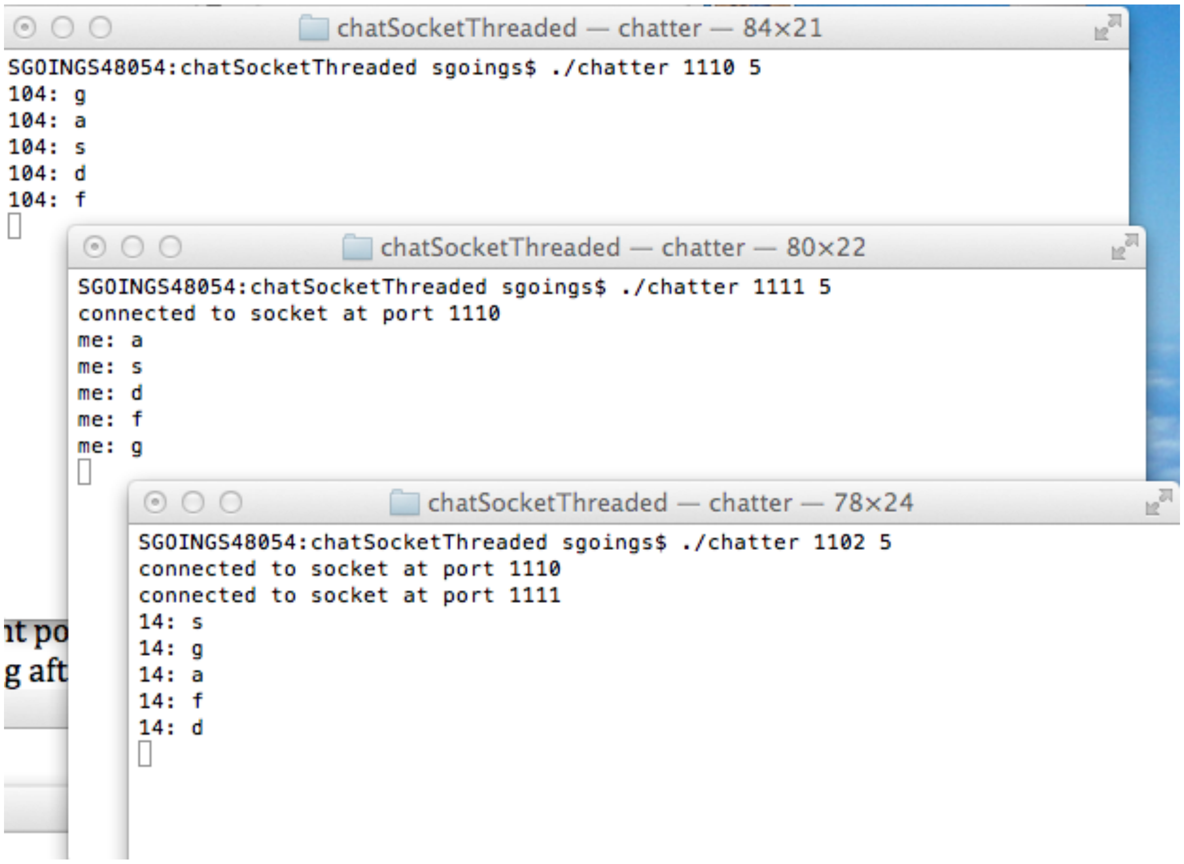

chatSocketThreaded — chatter — 84×21
chatSocketThreaded — chatter — 80×22
chatSocketThreaded — chatter — 88×24

```
SGOINGS48054:chatSocketThreaded sgoings$ ./chatter 1110 5
```

```
SGOINGS48054:chatSocketThreaded sgoings$ ./chatter 1111 5
connected to socket at port 1110
```

```
SGOINGS48054:chatSocketThreaded sgoings$ ./chatter 1102 5
connected to socket at port 1110
connected to socket at port 1111
```

2. Enter several short messages in quick succession in one of the chat windows and check the results in the other 2. Here are the results of my entering in the middle window the messages 'a', 's', 'd', 'f', 'g', each followed by hitting the <return> key as quickly as I could (note I had to wait a few seconds for the other 2 windows to receive all 5 messages):


chatSocketThreaded — chatter — 84×21
chatSocketThreaded — chatter — 80×22
chatSocketThreaded — chatter — 78×24

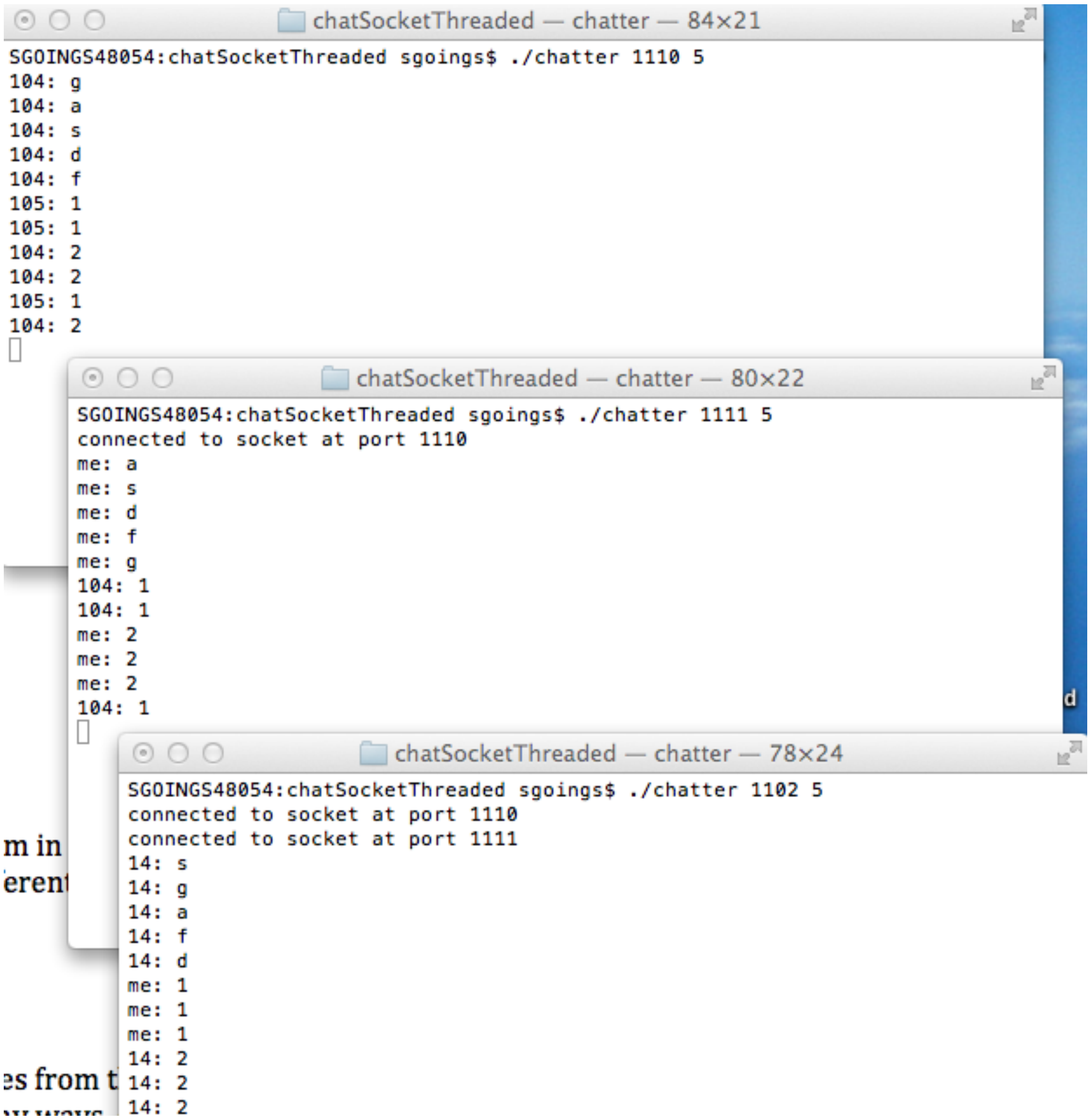
```
SGOINGS48054:chatSocketThreaded sgoings$ ./chatter 1110 5
104: g
104: a
104: s
104: d
104: f
```

```
SGOINGS48054:chatSocketThreaded sgoings$ ./chatter 1111 5
connected to socket at port 1110
me: a
me: s
me: d
me: f
me: g
```

```
SGOINGS48054:chatSocketThreaded sgoings$ ./chatter 1102 5
connected to socket at port 1110
connected to socket at port 1111
14: s
14: g
14: a
14: f
14: d
```

Note that the window I entered the messages in (middle) displays them in the same order as they were entered, but the other 2 chatters display them in completely different orders.

3. Enter a few messages quickly in one window and then immediately switch windows and enter a few other messages in the new window. Below I entered ‘1’, ‘1’, ‘1’ in the 3rd window down, then ‘2’, ‘2’, ‘2’ in the 2nd window down.


chatSocketThreaded — chatter — 84×21
```
SGOINGS48054:chatSocketThreaded sgoings$ ./chatter 1110 5
104: g
104: a
104: s
104: d
104: f
105: 1
105: 1
104: 2
104: 2
105: 1
104: 2
```
chatSocketThreaded — chatter — 80×22
```
SGOINGS48054:chatSocketThreaded sgoings$ ./chatter 1111 5
connected to socket at port 1110
me: a
me: s
me: d
me: f
me: g
104: 1
104: 1
me: 2
me: 2
me: 2
104: 1
```
chatSocketThreaded — chatter — 78×24
```
SGOINGS48054:chatSocketThreaded sgoings$ ./chatter 1102 5
connected to socket at port 1110
connected to socket at port 1111
14: s
14: g
14: a
14: f
14: d
me: 1
me: 1
me: 1
14: 2
14: 2
14: 2
```

Note that the 3rd window instantly displays the 1’s as I enter them, then the 2’s when it receives them. The middle window happened to receive 2 of the 1’s before I switched to it and started typing 2’s, but displayed the 2’s instantly as they were entered, and so the final 1 only showed up after when it eventually arrived. The 1st window could have displayed the 6 messages in literally any order provided I typed them all within a 5 second window (as 5 seconds was the max possible delay in this example), and you can see it shows them in a different order from either of the other 2 chatters.

Try whatever other tests you think of as you look through the code and question how it works before you start making your own modifications.

Your Task: Preserve message order from each individual sender

The application as provided does not even preserve ordering among messages from the same sender, so your first task is to ensure at least this basic ordering. You can do this in many ways, one that should be straightforward is to add a sequence number to each message and use this on the receiving end to re-order incoming messages when necessary. Note that you will probably want to create a receive buffer similar to the given send buffer to store messages that have been received but not output to the user yet (including those the user themselves type).