Systems Engineer Challenge

Implement a simple **chat server** to which multiple clients can connect and chat via a text-based protocol with other clients in one of multiple rooms.

In this document, `<NL>` represents the newline character (ASCII 0x0A or '\n' in C).

- Within each room, any client joined to the room may send a line of text to the room. The server prepends the client's username to the line, then sends the line to every client in the room, including the client that originally sent the text.
  - The server must be able to handle lines up to 20,000 bytes in length, including the terminating newline.
  - Messages sent in one room must not "leak" into another room.
- The server program shall take a single optional argument: the TCP port on which the server should run.
  - If the argument is invalid, print an error and exit.
  - If the argument is omitted, start the server on port 1234.
- The implementation should be reasonably performant, memory efficient, and scalable enough to handle at least 100 simultaneous clients.
- It should not be possible for clients to crash the server program or cause it to leak memory with invalid or malicious input.
- Write your program in C, C++, or another similar non-garbage-collected systems programming language.

Protocol Details

The server should listen on the port specified above for incoming TCP connections from clients on the network or Internet. The steps below describe the flow of each client connection.

1. Upon establishing a client TCP connection, the server receives the following command line from the client: `JOIN {ROOMNAME} {USERNAME}<NL>`
   - **JOIN** is the command used to join the room. It is case-insensitive.
   - **{ROOMNAME}** is the name of the room to join as an ASCII string of 1-20 characters. The string shall not contain any spaces or newlines.
   - **{USERNAME}** is the name of the user as an ASCII string of 1-20 characters. The string shall not contain any spaces or newlines.
   - For example, if a client wants to join the room cooking as the user james, the server would receive
     `JOIN cooking james<NL>`
2. When the server receives the command line, it should validate the command.
   - If the command line is not a valid JOIN command, the server should send `ERROR<NL>` to the client, then close the client connection.

3. The server joins the client to the room, creating one if it doesn’t already exist. It then sends the following line to every client in the room, including the client that has just joined:
   `{USERNAME} has joined<NL>`
4. For each line that the client sends, prepend the client’s username, a colon, and a space to the line, then send it to every client in the room, including the client that sent the original line.
   - For example, if the above client sends the line
     `hello all<NL>`
     The server should send the following line to every client in the room:
     `james: hello all<NL>`
5. Repeat step 4 until the client wishes to disconnect, at which point the client closes the TCP connection. The server should detect this and send the following line to every client in the room:
   `{USERNAME} has left<NL>`
   If the room is empty after this client leaves, delete the room.
6. At any point, if fulfilling a client’s request would use an unreasonably large amount of memory and/or CPU, the server may send `ERROR<NL>` to the client and close the connection. This should be done if e.g. a client sends an unreasonably long line.

Notes

- You can use telnet from another terminal to test your server:
  ```
  $ telnet localhost 1234
  ```

Example Sessions

Typical Session

Client A -> Server	**Server -> Client A**	**Client B -> Server**	**Server -> Client B**
`[initiates TCP conn]`			
`JOIN cooking joe<NL>`			
	`joe has joined<NL>`		
`no one here yet<NL>`			
	`joe: no one here yet<NL>`	`[initiates TCP Conn]`	
		`Join cooking bob<NL>`	
	`bob has joined<NL>`		`bob has joined<NL>`
		`hi joe<NL>`	
	`bob: hi joe <NL>`		`bob: hi joe <NL>`
`good day, bob<NL>`			
	`joe: good day, bob<NL>`		`joe: good day, bob<NL>`
`[closes TCP conn]`			
			`joe has left<NL>`
		`all alone now<NL>`	
			`bob: all alone now<NL>`
		`[closes TCP conn]`	

Error Situations

Client A -> Server	**Server -> Client A**
`[initiates TCP conn]`	
`JOIN foo bar blah<NL>`	
	`ERROR<NL>`
	`[closes TCP conn]`

Client A -> Server	**Server -> Client A**
`[initiates TCP conn]`	
`JOIN foo usernameislongerthan20chars<NL>`	
	`ERROR<NL>`
	`[closes TCP conn]`

Client A -> Server	**Server -> Client A**
`[initiates TCP conn]`	
`EAT tacos<NL>`	
	`ERROR<NL>`
	`[closes TCP conn]`