L8: Sopenalia

Overview

Warriors from many places gather each year for the spectacular event — Sopenalia. On the Fields of Hammer-Sword occurs a clash between Righteous Warriors, Ghostly Vampires, Swift Messengers of the Terrible Storm, and many other forces. Groups of adventurers organize joint expeditions to help one another in the event of inevitable death.

Wizards from the Magical and Scientific Institute have devised a plan to help such groups. They designed an application that allows companions to communicate amid the chaos of battle. This app also helps promptly identify when a companion falls so that they can be taken to the paladins for resurrection immediately.

The beauty of their solution is that it works with just the `netcat` program — which all adventurers already have, as they use Linux systems.

You may add separators like `--=x=--` and color messages using functions such as `set_color` and `reset_color`, as long as it doesn't reduce program readability. The wizards like them.

Task

Write a single-threaded and single-process chat server compatible with `netcat`. Due to time constraints in this lab, not all edge cases need to be handled. We make a few assumptions:

- All messages sent to the server have a maximum length of `MAX_MESSAGE_SIZE`, do not contain control characters (e.g., `\0`), and always end with a newline character `\n`.
- To simulate non-atomic socket behavior, you can use the `ctrl+d` shortcut in `netcat`, which sends text without a newline (unlike `Enter`, which includes it).

Stages (Total: 25 points)

1. (5 points)

Implement accepting a single client in a TCP server. The server takes the port number as an argument and listens for a connection. Once connected:

- Print the file descriptor (`fd`) of the connection, followed by a separator (e.g., `'|'`).
- Receive and print bytes until a newline character (`'\n'`) is received.
- Respond with: `"Hello world!\n"`

Then, close the connection and exit. Sending a reply to a disconnected client is considered an error.

---

2. (6 points)

Implement support for multiple clients. When a client connects and if the maximum `MAX_CLIENTS` isn't exceeded:

- Add the client to the end of the client list.
- Print to the console: `"[<fd>] connected"`
- Send: `"Please enter your username\n"` to the new client.
- Send: `"User logging in...\n"` to all other clients.

Otherwise, send: `"Server is full\n"` and close the connection.

Print all client messages to the console (being mindful of newline endings). Use `epoll` (or `select`, `poll`) for implementation.

ⓘ At this stage, handling client disconnections is not required.

---

3. (5 points)

Extract and save the first line of input (up to `'\n'`) from a client as their username. This process is called "logging in".

- If this is the first logged-in user, send: `"You're the first one here!\n"`
- Otherwise, send:
  ```
  Current users:
  [user1]
  [user2]
  ...
  ```
- Notify all other logged-in users (in green text): `"User [<name>] logged in\n"`
  - ⚠ Remember to reset the text color after sending to avoid coloring the rest of the `netcat` session.
- Print to console: `"[<fd>] logged in as [<name>]"`

ⓘ One epoll event may contain partial or multiple messages — ensure buffering behaves correctly.
ⓘ Clients can connect without logging in — do not assume all are known users.

---

4. (3 points)

Implement the chat functionality:

After a user has logged in, every newline-ended string is considered a message and should be broadcast as:

`[<name>]: <message>`

to all other logged-in users.

---

5. (5 points)

Handle client disconnections:

- If the client disconnects before logging in, print: `"[<fd>] failed to log in\n"`
- Otherwise:
  - Broadcast: `"User [<name>] is gone!\n"` to all other users.
  - Print to the console: `"[<fd>] known as [<name>] has disconnected"`

Ensure the client list is contiguous (move the last client in the list to take the place of the removed one).

Example Output

Server

```
[5] connected
[5] logged in as [Anne]
[6] connected
[6] logged in as [Bart]
[Anne]:Hi, Bart!
```

```
[Bart]:Hello, Anne.
[7] connected
[7] failed to log in
[5] known as [Anne] has disconnected
```

Client 1 (**Anne**)

```
Please enter your username
Anne
[     ---------==[ S O P ]==---------     ]
You're the first one here!
[     ---------==[ S O P ]==---------     ]
User logging in... <- This is green text :)
User [Bart] logged in
Hi, Bart!
[Bart]:Hello, Anne.
User logging in... <- This is green text :)
```

(terminated using Ctrl+C)

Client 2 (**Bart**)

```
Please enter your username
Bart
[     ---------==[ S O P ]==---------     ]
Currently present:
[Anne]
[     ---------==[ S O P ]==---------     ]
[Anne]:Hi, Bart!
Hello, Anne.
User logging in... <- This is green text :)
User [Anne] is gone!
```

Client 3

Started after Anne and Bart exchanged messages. Disconnected immediately with Ctrl+C.

ⓘ The green-colored text in actual implementation appears in color. This Markdown file is in plain text and does not show colors.