Assignment 1: Client-Server Program

The goal of this assignment is to implement a simple non-blocking server and a simple interactive client used for a networks exercise. (Course: Communication Networks)

About the Project

- Small non-blocking TCP server and a simple interactive client used for a networks exercise.
- Server implements a minimal line-based protocol with a two-step login (username then password) and a few example commands: balanced parentheses checking, LCM, and Caesar cipher.

Repository Files

- `ex1_server.py` — Non-blocking TCP server using `select`. Loads users from a simple text file and manages per-client state machine:
  - States: `await_username`, `await_password`, `logged_in`.
  - Login is strictly two messages (one line each): `User: <username>` then `Password: <password>`.
  - After login the server accepts single-line commands (see Protocol).
- `ex1_client.py` — Simple interactive TCP client. Connects to server, prints server lines, reads user input and sends each line.
- `README.md` — This file.

Quick Usage

1. Start server:

```
python ex1_server.py users_file [port]
```

- users_file: Users file. Format: each non-empty line is `username<TAB>password`.
- default port: 1337

2. Start client:

```
python ex1_client.py [host] [port]
```

- default host: localhost, default port: 1337

Protocol

1. Server connects -> sends: `Welcome! Please log in.`
2. Client -> `User: <username>` (single line)
3. Server waits for password
4. Client -> `Password: <password>` (single line)
5. On success: `Hi <username>, good to see you.`; on failure: `Failed to login.` and server returns to `await_username`.

After login, accepted single-line commands: `- parentheses: <text>` — checks that text contains only '(' and ')' and whether balanced. Replies `the parentheses are balanced: yes|no` or `error: invalid input`. `- caesar: <text> <shift>` — expects integer `shift`; `<text>` must contain only English letters (A–Z, a–z) and whitespace; returns `the ciphertext is: <text>` or `error: invalid input`. `- quit` — server closes connection. - unknown commands -> `error: unknown command` then disconnect.

Design Decisions

- Uses IPv4 TCP (stream) sockets for client connections in order to avoid disconnects and server errors due to packages arriving not in order (or not at all).
- select-based non-blocking server: simple, dependency-free way to serve multiple clients in a small exercise.
- Explicit two-step login state machine: avoids parsing multiple messages combined together and matches the exercise requirement to receive username and password in separate client messages.
- Minimal, strict parsing: usernames/passwords are tokenized simply (no whitespace allowed) as instructed by the course staff.
- `safe_send` and `disconnect_client` helper functions centralize socket error handling and avoid crashes from broken pipes.

Notes

- If the server reports "not enough values to unpack", check `users_file_` separators — it must be a TAB between username and password. Use VS Code Replace (regex) or the small Python/PowerShell snippets to convert spaces to tabs.
- The client expects user input one line at a time. For automated tests, use stdin redirection: `python ex1_client.py < scripted_input.txt`.
- To make tabs visible in VS Code enable "Render Whitespace" or change indentation settings (bottom-right) to "Insert Tabs".

Collaborators

This project was written by Oz Cabiri and Nachman Rogosinsky.