CS447 : Networks and Data Communications

Programming Assignment #1 (P1)

Total Points: 100

Assigned Date : Thursday, September 12, 2024
Due Date : Thursday, September 26, 2024 @ 01:59:59 p.m.

Overview

Your first programming assignment is to **implement a basic client/server application** using the socket interface. The learning objectives are as follows:

a. to get your feet wet on socket programming basics;
b. to understand the ordering of the socket interface primitives;
c. to get you exposed to Linux system calls (if you haven't already);
d. to gain a basic understanding of network protocols; and
e. to set yourself up for the rest of the course.

Back Story

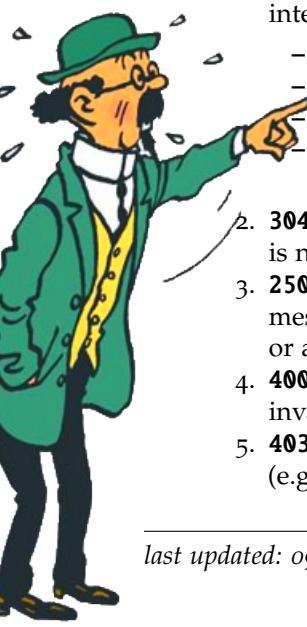
Captain Haddock, ever the avid reader, often finds himself misplacing books amidst the chaos of his sea voyages. The ship's library, though modest, is a treasure trove of knowledge and escape for the crew, but tracking borrowed volumes has become a headache. *Professor Calculus*, ever the ingenious inventor, has stepped up to the challenge. Leveraging his newfound expertise from CS447 at SIUE, Calculus proposes a solution: an online library book management system accessible to the entire crew. Not only will this system keep meticulous records, but it will also allow multiple users to search, check out, and return books simultaneously, even amidst the unpredictable swells of the high seas. In this first iteration, Calculus envisions focusing on essential features like searching, checking availability, and managing borrowed books, etc, categorized into 3 modes as follows:

1. **`SEARCH`**
   - Given search term (title or author), return list of matching books
   - Given book title, return detailed information (author, availability)
2. **`BOOK MANAGEMENT`**
   - Given book title, check out the book (if available)
   - Given book title, return the book (if checked out)
   - List all books currently checked out
3. **`RECOMMENDATIONS`**
   - Given genre, provide book recommendations
   - Given book title and rating, record the rating

Technical Requirements

- `server` should be capable of accepting requests from TCP clients.
- `server` should support **concurrency** (more than one `client` should be capable of using the library book manager at the same time).
- Your protocol interaction should adhere to the following specifications.
- **Client Commands:**
  1. **`HELO <server-hostname>`** – This is the **first** command issued by the `client` to the `server`. It initiates the connection and identifies the client. The correct reply code is `200`.
  2. **`HELP`** – This command can be issued anytime after the **`HELO`** command. It requests a list of available commands supported by the server. If this command is issued inside a mode, then the mode specific list of available commands will be listed. The correct server reply code is `200`.
  3. **`SEARCH`** – This command puts the client in search mode, allowing them to search for books. The correct server reply code is `210`.
     (a) **`FIND <search_term>`** – The **`FIND`** command searches for books matching the given term. The correct server reply code is `250` with a list of matching books or `304` if no books are found.
     (b) **`DETAILS <book_id>`** – The **`DETAILS`** command requests detailed information about a specific book. The correct server reply code is `250` with the book details or `404` if the book is not found.
  4. **`MANAGE`** – This command puts the client in book management mode, allowing them to check out and return books. The correct server reply code is `220`.
     (a) **`CHECKOUT <book_id>`** – The **`CHECKOUT`** command attempts to check out a book. The correct server reply code is `250` if successful, `403` if the book is unavailable, or `404` if the book is not found.
     (b) **`RETURN <book_id>`** – The `RETURN` command returns a checked-out book. The correct server reply code is `250` if successful or `404` if the book was not checked out by the user.
     (c) **`LIST`** – The `LIST` command lists all books currently available for check out by the user. The correct server reply code is `250` with the list of books or `304` if no books are available.
  5. **`RECOMMEND`** – This command puts the client in recommendation mode, allowing them to get book suggestions and rate books they've read. The correct server reply code is `230`.
     (a) **`GET <genre>`** – The `GET` command requests book recommendations for a specific genre. The correct server reply code is `250` with a list of recommendations or `304` if no recommendations are found.
     (b) **`RATE <book_id> <rating>`** – The `RATE` command rates a book on a scale of 1 to 5. The correct server reply code is `250` if successful or `400` if the rating is invalid.
  6. **`BYE`** – This command closes the connection and requests a graceful exit. This command can be issued anytime during the interaction. The correct server reply code is `200`.

- **Server Reply Codes:**
  1. **`200/210/220/230`** Command Success. The command success reply code is issued only when the interaction happens according to the correct specification. Examples:
     - `200` – The correct response code for **`HELO`** and **`HELP`** commands.
     - `210 Switched to Search Mode` – If the **`SEARCH`** command is issued at the correct point of interaction.
     - `220 Switched to Manage Mode` – If the **`MANAGE`** command is issued at the correct point of interaction.
     - `230 Switched to Recommend Mode` – If the `RECOMMEND` command is issued at the correct point of interaction.
  2. **`304 NO CONTENT`** – This reply code indicates that the server successfully processed the request, but there is no content to send back (e.g., no search results or no checked-out books).
  3. **`250 <data>`** – This reply code is issued in response to a correct command syntax received in the previous message from the client. `data` represents the requested information, such as search results, book details, or a success message.
  4. **`400 BAD REQUEST`** – This reply code indicates that the server could not understand the request due to invalid syntax or missing parameters.
  5. **`403 FORBIDDEN`** – This reply code indicates that the server understood the request, but refuses to fulfill it (e.g., trying to check out an unavailable book).



6. **`404 NOT FOUND`** – This reply code indicates that the server did not find the requested resource (e.g., a book with a specific ID).
7. **`500 INTERNAL SERVER ERROR`** – This reply code indicates that the server encountered an unexpected condition that prevented it from fulfilling the request.

Functional Requirements

1. Starter Code:
   - Utilize the provided starter code from Moodle. Using any other base code will result in an invalid submission.
   - IP addresses and port numbers should be configured via `server.conf` and `client.conf` files, not hardcoded.
   - Run the server with `./server server.conf` and clients with `./client client.conf`.
2. Helper Code:
   - An additional helper function (`p1_helper.cpp`) along with a book database (`books.db`) is also provided to you through moodle. The helper function is intended to be used to maintain an in-memory database of books at the server side.
   - The Book database (`books.db`) is a text file that you can expand with more entries to enhance your testing. Feel free to add more books, explore different genres, or even include books that are already checked out to test various scenarios. Just remember to maintain the same format!
3. Network Setup:
   - The `client-server` connection must be TCP-based.
   - The `server` should be capable of handling multiple simultaneous client connections (hence **concurrency**). You have the flexibility to achieve this using either multi-threading techniques or the `fork()` system call.
4. Client-Server Interaction:
   - Clients must exit gracefully using the **`BYE`** command. The server can be terminated forcefully.
   - The protocol interaction should adhere to the specifications detailed in the previous section (client commands, server reply codes, etc.).
   - Here's a sample (non-comprehensive) interaction. Assume the client's IP address is 192.168.0.11 and running TCP and the server's running on the `tgamage-0` zone-server instance. Note: The server must acknowledge client IP and connection type.

```
          Client       Server
HELO tgamage-0   →
                 ←   200 HELO 192.168.0.11 (TCP)
 CHECKOUT Dune   →
                 ←   503 Bad sequence of commands.  Must enter a mode first.
         HELP    →
                 ←   200 Available commands:  HELO, SEARCH, MANAGE, RECOMMEND, BYE
       SEARCH    →
                 ←   210 Ready for search!
    FIND Dune    →
                 ←   250 Dune
       MANAGE    →
                 ←   220 Ready for book management!
 CHECKOUT Dune   →
                 ←   250 Book checked out successfully
          BYE    →
```

5. Testing & Deployment:
   - Test your implementation with at least 2 simultaneous client connections, preferably on separate `zone-server` containers.
   - Ensure your code compiles and runs on a Linux machine. Provide a `README` with clear compilation instructions and any additional software dependencies.
6. At the end of your implementation, you should be able to:
   - Compile and run your code in a Linux machine.
   - Run your server program first.
   - Run one or more clients to connect to the server.
   - Perform library book manager functionality while meeting the technical requirements mentioned above.
   - Exit the client(s) gracefully.

Instructions

- **Start early & backup often:** This assignment requires careful planning and implementation. Don't procrastinate, and remember to save your progress frequently.
- Maintain clean, readable code. Adhere to Google's C++ Style Guide or another established standard. Use one of Google's coding standard found here `https://google.github.io/styleguide/`, if you don't already follow one.
- Your code must compile and run flawlessly on a standard Linux environment. Test it thoroughly using command-line tools.
- Implement your solution in C++. Submit a `.tgz` file containing only your source code, a `README` with compilation instructions, and your report.
- Handling parallelism is not trivial. Start with a single-client version, then gradually scale up.
- Focus on core C++ socket and I/O functionalities. Avoid external libraries unless explicitly permitted.
- DEADLINE: **Thursday, September 26, 2024 @ 01:59:59 p.m.** through Moodle. Email submissions are not honored.

Deliverables

A complete solution comprises of:

1. Report (**`pdf`**):
   - Introduction: Your objectives for the assignment and what you hope to learn.
   - Design: Explain your overall system design, key choices you made, and the protocol's reply codes.
   - Sample Run: Include screenshots showcasing a typical interaction with your system. These can help illustrate functionality and potentially earn partial credit if certain features aren't fully implemented.
   - Summary and Issues: Summarize your achievements and explicitly list any unimplemented or buggy functionality.
2. Compressed Tarball (**`siue-id-p1.tgz`**):
   - Source Code Directory: Include all your C++ source code and configuration files (`server.conf`, `client.conf`).
   - `README`: Provide clear instructions on how to compile and run your code.
   - `Makefile`: A Makefile is required, especially if your project involves multiple executables or compilation flags.

   Use the following command to create the compressed tarball:
   `tar -zcvf siue-id-p1.tgz source/`.

   (Replace `siue-id` with your actual SIUE ID and `source/` with the name of your source code directory)

Your code should be a testament to your abilities, not a copy of someone else's work. Collaborate, don't copy! Learning from peers is great, but copying code (from classmates or online) is strictly prohibited. Cite any external resources you use for ideas, and then implement those ideas in your own way.

This assignment is a challenge designed to push your boundaries and foster growth. Embrace the learning process and don't hesitate to seek help when needed. Remember, the goal is to master these concepts, not just complete the assignment. Plagiarism, whether from classmates, online sources, or AI tools, stifles learning and compromises academic integrity. The instructor actively uses MOSS http://theory.stanford.edu/~aiken/moss/ to check for software similarity. Plagiarism has severe consequences including and will result in a failing grade. No exceptions.

Some Useful Resources

- Linux Man pages – found in all Linux distributions
- Beej's Guide to Network Programming – A pretty thorough free online tutorial on basic network programming for C/C++ https://beej.us/guide/bgnet/.
- Linux Socket Programming In C++ – https://tldp.org/LDP/LG/issue74/tougher.html
- The Linux HOWTO Page on Socket Programming – https://www.linuxhowtos.org/C_C++/socket.htm
- Learn Makefiles With the tastiest examples – https://makefiletutorial.com/