Remote Procedure Call (RPC) File Transfer System

Tran Luong Hoang Anh

15/12/2024

Introduction

File transfer systems are essential for sharing data between distributed systems. In this project, a file transfer application was developed using Remote Procedure Call (RPC) mechanisms. This system allows users to upload and download files seamlessly between a client and a server. By employing RPC, the complexity of the file transfer process is abstracted, enabling developers to focus on higher-level functionalities. The system uses Python for implementation, specifically leveraging the `socket` library for communication and the `pickle` module for serializing data, ensuring a robust and extensible design.

File transfer systems are a cornerstone of modern distributed systems, providing the backbone for sharing data in a reliable and efficient manner. The RPC-based approach brings several advantages, such as reducing development complexity and enhancing the user experience through seamless communication between the client and the server. This report delves into the design, implementation, and outcomes of this RPC-based file transfer system.

Overview

The file transfer system supports two main operations:

- **File Upload**: Allows the client to send a file to the server.
- **File Download**: Enables the client to retrieve a file stored on the server.

The design emphasizes simplicity and scalability, with provisions for future improvements such as encryption and authentication mechanisms. The modular approach also ensures that the system can be easily integrated into larger distributed environments.

Choice of RPC

RPC was chosen for its ability to simplify client-server communication. By abstracting the procedural call mechanism, RPC allows the client to invoke methods on the server as if they were local. This approach reduces the complexity of implementing distributed systems. The Python `socket` library facilitates the underlying TCP-based communication, while `pickle` enables efficient serialization of method calls and responses. This combination ensures a lightweight and efficient system suitable for real-time file transfer tasks.

Additionally, RPC enhances maintainability by decoupling the client and server functionalities. Developers can focus on specific modules without delving into the intricate details of network protocols. This abstraction aligns well with the project's goals of simplicity, reliability, and extensibility.

System Architecture

Communication Protocol

The client and server communicate using a structured RPC protocol:

- **RPC Call**: The client sends serialized data containing the method name (`upload` or `download`) and the required parameters (e.g., file name, file data).

- **Response**: The server processes the request and returns a serialized response containing the status (`success` or `error`) and relevant details (e.g., message, file data).

This protocol ensures clear, structured communication, making the system easy to debug and extend. It also provides the flexibility to add new methods or features without significant overhauls.

Flow Chart

```
Client                  Server
  |                        |
  |  Request (method + params)  --->   |
  |                        |
  |                        |
  |         Process Request            |
  |                        |
  |  <--- Response (status + data)     |
  |                        |
```

Implementation Details

Server Code

The server is responsible for handling incoming RPC requests and performing the required actions:

- **File Upload**:
  - Extracts the file name and binary data from the client's request.
  - Saves the file to a designated directory on the server.
  - Sends a confirmation message back to the client.
- **File Download**:
  - Verifies if the requested file exists in the server directory.
  - Sends the file content to the client if available.
  - Returns an error message if the file is not found.

Files are stored in the server's directory:

`C:\Users\HG\Downloads\hoctap\Distributed-System\DS2024\labwork2\file-transfered\server_files`

Client Code

The client initiates file transfer requests and processes server responses:

- **File Upload**:
  - Reads the file from the local directory.
  - Sends the file name and content to the server using the `upload` RPC method.
  - Displays the server's response to the user.
- **File Download**:
  - Sends a request to the server specifying the file to download.
  - Saves the file to a designated directory upon receiving the content.
  - Displays the result of the operation.

Downloaded files are saved in the client's directory:

`C:\Users\HG\Downloads\hoctap\Distributed-System\DS2024\labwork2\file-transfered\client_files`

Benefits and Challenges

Benefits

- **Ease of Use**: The RPC-based abstraction makes the system intuitive for both developers and users.
- **Efficiency**: The streamlined communication protocol ensures minimal latency during file transfers.
- **Modularity**: The separation of client and server functionalities facilitates independent development and maintenance.

Challenges

- **Error Handling**: Robust error handling is crucial to manage issues such as network interruptions or invalid file requests.
- **Security**: The current system lacks encryption and authentication mechanisms, which are essential for secure file transfers in real-world applications.

Conclusion

This project demonstrates the effectiveness of using RPC to build a file transfer system. By abstracting the complexities of client-server communication, the system provides a straightforward and reliable mechanism for transferring files. The modular design ensures ease of maintenance and scalability. Future enhancements could include implementing security measures such as encryption and authentication to make the system suitable for sensitive data transfers.

In summary, this project showcases the potential of RPC-based solutions in simplifying distributed system development. The lessons learned from this implementation can be applied to build more advanced systems, ensuring a balance between performance, scalability, and security.