File Transfer using RPC

Lab Report

Nguyen Manh Khoi
22BI13219

December 11, 2024

1 Introduction

RPC (Remote Procedure Call) is a protocol that allows a program to execute code on another computer or server over a network.This was used to call procedures on other machines.

The goal of this lab is to use Python's ‘xmlrpc‘ library to develop a simple file transfer system that using RPC for distributed systems.

2 Objectives

The objectives of this lab:

- Implement a file transfer system using RPC.
- Successfully transfer files between a client and a server.

3 Methodology

1. The **server** exposes an RPC function, ‘sendfile()‘, which reads and sends the contents of a file.
2. The **client** creates an RPC connection to the server and requests a specific file by calling the ‘sendfile()‘ method.
3. The **server** checks whether the requested file exists. If it does, the server reads the file in binary mode and sends the data to the client. If the file does'nt exist, the server returns back an error message.
4. The **client** receives the file data from the server and saves it.

4 System Design

The system consists of two main components: the **server** and the **client**. The server listens for incoming RPC requests and provides a method to send files. The client interacts with the server via RPC, requesting and receiving files.

4.1 File Transfer Protocol

The file transfer protocol follows this sequence:

1. The client requests file from the server by calling ‘sendfile()‘ method.
2. The server checks if the file exists. If it exists, the server sends the file’s binary data to the client. If not, the server returns an error message.
3. The client saves the received data to a local file.
4. The process concludes once the file transfer is complete, and both the client and server terminate the session.

5 Implementation

The implementation of the file transfer system was achieved using Python’s ‘xmlrpc.server‘ for the server and ‘xmlrpc.client‘ for the client. The code is provided below.

5.1 Server Code

Listing 1: Server Code

```
import xmlrpc.server
import os

class FileTransferRPC:
    def sendfile(self, filename):
        """Send a file to the client."""
        if not os.path.exists(filename):
            return f"File '{filename}' not found on the server."

        with open(filename, 'rb') as f:
            file_data = f.read()

        return file_data

def run_rpc_server():
    # Create RPC server
    server = xmlrpc.server.SimpleXMLRPCServer(('0.0.0.0', 12344))
    server.register_instance(FileTransferRPC())
    print("RPC Server is running on port 12344...")
    server.serve_forever()

if __name__ == '__main__':
    run_rpc_server()
```

5.2 Client Code

Listing 2: Client Code

```
import xmlrpc.client

def client_program():
    # Server info
    host = 'http://localhost'  # Local
    port = 12344  # The same port with the server

    # Create an RPC client to interact with server
    server = xmlrpc.client.ServerProxy(f'{host}:{port}')

    # File to be received
    filename = 'received_transfer_file.txt'

    # Request file from server
    print(f"Requesting file '{filename}' from the server...")
    file_data = server.sendfile(filename)  # RPC call

    if isinstance(file_data, str) and file_data.startswith("File"):
        # If the server returns an error message
        print(file_data)
    else:
        # Otherwise, save the received file
        with open(filename, 'wb') as f:
            print("Receiving file data...")
            f.write(file_data)
        print(f"File received successfully! Saved as {filename}")

if __name__ == '__main__':
    client_program()
```

5.3 Sample Output from the Server

```
RPC Server is running on port 12344...
```

The server waits for RPC requests. When a request is received, it sends the requested file data or an error message if the file does not exist.

5.4 Sample Output from the Client

```
Requesting file 'received_transfer_file.txt' from the server...
Receiving file data...
File received successfully! Saved as received_transfer_file.txt
```

If the file exists on the server, the client receives the file data and saves it locally. If the file does not exist, an appropriate error message is displayed.

6 Discussion

This lab demonstrated simple file transfer using RPC.

There is some improvement we can make:

- The current system does not handle large files efficiently. File chunking could be introduced to avoid memory overload.
- Error handling for network interruptions could be improved to make the system more robust.
- Adding concurrent client handling to the server would improve scalability.

7 Conclusion

In this lab, we successfully implemented a file transfer system using Remote Procedure Call (RPC). Python's 'xmlrpc' library made it easy to set up the server and client and enabled efficient communication between them. The system allowed the client to request and receive files from the server.

Future improvements could include chunking for large files, better error handling, and support for concurrent client requests.