RPC File Transfer System

Nguyen Viet Khoa

December 5, 2025

1 Design of the RPC Service

The RPC service is designed using Python's built-in `xmlrpc` library, which allows the client to call remote procedures on the server over HTTP. The server exposes a single method `get_file(filename)`, which reads the requested file and returns its binary content wrapped in `xmlrpc.client.Binary` for safe transmission. The client invokes this method and saves the received data locally.

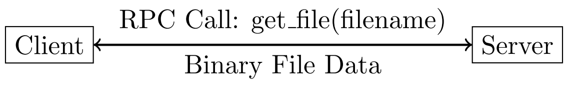

Figure 1: RPC Service Design

2 Organization of the System

The system is organized in a new directory called `RPC`, copied from the original TCP implementation. It contains:

- `server.py`: Manages the RPC server setup and file serving.
- `client.py`: Handles RPC client calls and file saving.
- Files to transfer (e.g., `example.txt`) placed in the server's working directory.

No additional subdirectories are needed, as the RPC framework handles communication without raw sockets.

3 Implementation of the File Transfer

The file transfer is implemented via an RPC method on the server that reads the file and returns it as binary data. The client calls this method and writes the data to a new file. Here is a code snippet from the server:

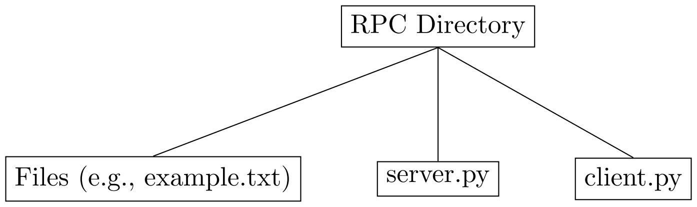

Figure 2: System Organization

Listing 1: Server Code Snippet

```
def get_file(self, filename):
    try:
        with open(filename, 'rb') as f:
            return xmlrpc.client.Binary(f.read())
    except FileNotFoundError:
        return xmlrpc.client.Binary(b'File␣not␣found')
    except Exception as e:
        return xmlrpc.client.Binary(str(e).encode())
```

And from the client:

Listing 2: Client Code Snippet

```
data = proxy.get_file(filename)
if data.data.startswith(b'File␣not␣found') or data.data.
    startswith(b'Error'):
    print(data.data.decode())
else:
    received_filename = f"received_{filename}"
    with open(received_filename, 'wb') as f:
        f.write(data.data)
    print(f"File␣received␣and␣saved␣as␣{received_filename}")
```

4 Who Does What

- **Server**: Listens for RPC requests, reads the requested file from its local filesystem, wraps the content in a binary object, and returns it. Handles errors such as file not found.
- **Client**: Connects to the server via RPC proxy, sends the filename as a parameter, receives the binary data, and saves it to a local file. Checks for error messages in the response.