Practical Work 2: RPC File Transfer

Nguyen The Xuan

December 5, 2025

1 Introduction

The objective of this practical work is to upgrade a file transfer system from a standard TCP socket implementation to a Remote Procedure Call (RPC) architecture. We utilized Python's built-in `xmlrpc` library to create a client-server model where the file upload logic is abstracted as a remote function call.

2 RPC Service Design

In our design, the server acts as a function provider, exposing a specific method named `upload_file`. The client acts as the caller. Unlike raw sockets where we manually handle the stream of bytes, RPC allows us to treat the network interaction as a local function call.

The core design decisions were:

- **Protocol:** XML-RPC over HTTP (via Python's `xmlrpc`).
- **Data Encapsulation:** Binary file data is wrapped in an `xmlrpc.client.Binary` object to ensure safe transport of non-text data across the XML protocol.
- **Method Signature:** The server exposes `upload_file(filename, data)`, which returns a status string.

3 System Organization

The system is organized into two main components:

1. **The Server (`rpc_server.py`):** Initializes the RPC server, registers the introspection functions, and defines the `save_file_rpc` method. It listens on port 8000.
2. **The Client (`rpc_client.py`):** Reads a local file, wraps it in the XML-RPC binary format, and invokes the proxy method.

4 Implementation

Below are the critical code snippets demonstrating the RPC implementation.

4.1 Server-Side: Exposing the Method

The server uses `SimpleXMLRPCServer` to register the function.

```python
def save_file_rpc(filename, file_data):
    file_path = os.path.join(UPLOAD_DIR, os.path.basename(filename))
    with open(file_path, 'wb') as handle:
        handle.write(file_data.data)
    return f"ACK: {filename} uploaded successfully."

# Registering the function
server.register_function(save_file_rpc, 'upload_file')
```

Listing 1: Server Registration Logic

4.2 Client-Side: The Remote Call

The client connects via a `ServerProxy` and wraps the file content.

```python
# Connect to server
proxy = xmlrpc.client.ServerProxy('http://localhost:8000')

# Read and wrap binary data
with open(path_to_file, 'rb') as handle:
    binary_data = handle.read()
rpc_binary = xmlrpc.client.Binary(binary_data)

# Remote Procedure Call
response = proxy.upload_file(filename, rpc_binary)
```

Listing 2: Client Remote Call

5 Experimental Results

We tested the system by transferring various files (text and images) from the client to the server.

1. **Setup:** The server was started on `localhost` port 8000.
2. **Execution:** The client script was executed with a target file as an argument.
3. **Observation:**
    - The client output showed a successful connection and received an "ACK" message from the server.
    - The server console logged the incoming request.
    - The transferred file appeared correctly in the `server_files` directory with no data corruption.