Operating Systems Project

CS 5523 Project Report: Math Client/Server using RPC

Group Members:

- **Nagasruthi Uppalapati**
- **Sai Chandana Erram**

1. Project Title

Distributed Math Service Using Python XML-RPC with Interactive Client

2. Objective

The objective of this project is to design and implement a math service using Remote Procedure Call (RPC), enabling a client to perform mathematical operations remotely on a server, with support for concurrent client access and synchronized shared state.

3. Project Overview

The project consists of:

- **A multi-threaded server that provides 4 remote mathematical operations.**
- **A client with an interactive menu, allowing a user to choose operations and input values.**
- **Support for tracking usage counters of each operation on the server.**
- **Thread-safe access to counters using Python’s threading.Lock.**

- Ability to run multiple clients concurrently.

4. System Architecture

- Server: XML-RPC server on localhost:8000, handling multiple clients concurrently.
- Clients:
  - client.py: sends 1000 random RPC calls (automated).
  - interactive_client.py: user can manually choose operations and input parameters.
- Server logs the number of times each function is called.
- Counters are shared state and are protected using a lock to prevent race conditions.

5. Interactive Client (User-Focused Feature)

The interactive_client.py enhances usability by providing a command-line menu interface:

Features:

- Prompts the user with operation choices:
  1. Add
  2. Subtract
  3. Find Min
  4. Find Max
  5. View operation counters
  6. Exit
- Takes input from the user (integers/floats) for each operation.
- Displays the result of each RPC call.
- Uses proxy.getCounters() to fetch updated usage stats from the server.

This version allows for live testing of functionality and is ideal for presentations, debugging, or demonstrations.

6. Synchronization & Concurrency

- The server handles multiple clients using ThreadingMixIn and SimpleXMLRPCServer.
- Shared counters dictionary is updated using a threading.Lock to avoid race conditions.
- The run_clients.py script launches multiple clients concurrently to validate synchronization.

7. Testing and Results

- Ran interactive_client.py to test each function manually.
- Sample Result

```
--- Math RPC Client Menu ---
1. magicAdd(a, b)
2. magicSubtract(a, b)
3. magicFindMin(a, b, c)
4. magicFindMax(a, b, c)
5. Show operation counters
0. Exit

Choose an option (0-5): 1
Enter first number: 10
Enter second number: 20
Result: 10.0 + 20.0 = 30.0

Choose an option (0-5): 3
Enter first integer: 8
Enter second integer: 2
Enter third integer: 5
Minimum of (8, 2, 5) is: 2

Choose an option (0-5): 5
Operation Counters: {'add': 1, 'subtract': 0, 'min': 1, 'max': 0}
```

- Verified that the counters updated correctly in real-time with each action.

8. Contributions

Nagasruthi Uppalapati

- Designed and implemented the RPC server.
- Created the standard automated client (client.py) and handled counter synchronization.
- Wrote the run_clients.py script for concurrent client testing.
- Collected screenshots and output data for testing.

Sai Chandana Erram

- Developed the interactive version of the client (interactive_client.py).
- Focused on user input handling, error-checking, and formatting results.
- Assisted in testing and debugging RPC connections.
- Prepared the report and documented the project architecture.

9. Conclusion

This project successfully demonstrated:

- Implementation of distributed systems using Python XML-RPC.
- Handling of shared resources with synchronization.
- Concurrent client support and multithreading on the server.
- An intuitive, interactive client interface for testing and usability.

The interactive client provided an effective way to visualize and validate remote operations and shared state updates in real-time.