2235-CSE-5306-001Distributed Systems

Report: Project 1

Harshita Chegondi - 1002115738
Satya Pradyumna Teja Nunna - 1002028665

"I have neither given nor received unauthorized assistance on this work. I will not post the project description and the solution online."

Sign: Harshita Chegondi
Satya Pradyumna Teja Nunna

Date: 09/21/2023

Part 1 (node.py)

ServerNode Class

- The `ServerNode` class is a RPyC service representing the server-side functionality. It manages file uploads, downloads, deletions, and renames.
- The constructor (`__init__`) sets up the server's configuration, such as the host, port, and the root directory for storing files.
- The `exposed_upload`, `exposed_download`, `exposed_delete`, and `exposed_rename` methods are RPyC-exposed methods that clients can call remotely to perform file operations. They handle file I/O and return appropriate responses.

ClientNode Class

- The `ClientNode` class represents the client-side functionality. It connects to the server and provides methods for uploading, downloading, deleting, and renaming files.
- The `upload`, `download`, `delete`, and `rename` methods interact with the server by calling the corresponding remote methods exposed by the server.

Main Block

- The main block handles command-line arguments to specify whether the script should run as a server or client. It also allows users to specify file-related actions and filenames.
- For the server mode, it starts a threaded RPyC server using `ThreadedServer`.
- For the client mode, it initializes a `ClientNode` and performs file actions based on the provided arguments.

Overall Structure

- The code follows a modular and object-oriented structure, separating server and client logic into their respective classes.
- It provides user-friendly command-line argument parsing for easy interaction.
- The code appropriately handles exceptions and prints error messages when file operations fail.

Part 2 (node.py)

ServerNode Class

- The `ServerNode` class is an RPyC service responsible for handling file-related operations, similar to the previous version of the code. It exposes methods for uploading, downloading, deleting, and renaming files.

- The server also prints messages to the console indicating when clients connect and disconnect.

ClientNode Class

- The `ClientNode` class represents the client-side functionality. It connects to the server, interacts with it to perform file operations, and then closes the connection.
- It provides methods for uploading, downloading, deleting, and renaming files.

FSEventHandler Class

- The `FSEventHandler` class extends `FileSystemEventHandler` from the `watchdog` library. It is responsible for monitoring file system events in a specified local folder.
- It overrides methods such as `on_created`, `on_modified`, `on_deleted`, and `on_moved` to detect changes in the local file system.
- When a change occurs, it spawns a separate thread to interact with the server, performing the corresponding file operation (upload, delete, rename).

Main Block

- The main block parses command-line arguments to determine whether to run the code as a server or client.
- For the server mode, it starts a threaded RPyC server.
- For the client mode, it initializes an `Observer` from the `watchdog` library to watch for file system changes. It creates a `FSEventHandler` instance and attaches it to the observer. Then, it enters a loop, waiting for changes and synchronizing them with the server.

Part 3

`server.py`

1. The `ComputationService` class is defined, which extends `rpyc.Service`. This class represents the server-side computation service.
2. The `on_connect` and `on_disconnect` methods are overridden to handle client connection and disconnection events, printing messages to indicate when clients connect and disconnect.
3. Two methods, `exposed_add` and `exposed_sort`, are exposed to remote clients. These methods allow clients to perform addition and sorting operations, respectively.
4. The actual computation logic for addition and sorting is implemented in the `add` and `sort` methods.
5. In the `__main__` block, a `ThreadPoolServer` is created to serve the `ComputationService`. The server listens on a specified port (PORT), and the service is started.

`client.py`

1. The `sync_rpc` and `async_rpc` functions are defined to demonstrate synchronous and asynchronous RPC calls, respectively.
2. In the `sync_rpc` function, a connection is established to the server using `rpyc.connect`, and two synchronous RPCs are performed: `add(3, 4)` and `sort([4, 2, 7, 1, 9])`. The results are printed, and the connection is closed.
3. In the `async_rpc` function, a connection is established to the server, and two asynchronous RPCs are performed using `rpyc.async_`. An acknowledgment is received before fetching and printing the results.
4. In the `__main__` block, both `sync_rpc` and `async_rpc` functions are called sequentially to demonstrate the difference between synchronous and asynchronous RPCs.

What we have learnt:

- We have learnt how a client can communicate with a server using XMLRPC.
- File transfer between server to client
- Synchronous and Asynchronous process
- Multi-Threading connection between client and server

Issues encountered while implementing the project:

- Implementing connections between client and server using XMLRPC was felt a bit challenging as we hardly had experience on it.
- Going through RPC and some other new libraries has taken some time to understand them.