Project Architecture

1. Architecture Design

The architecture of the File/Music Synchronization Application is designed to facilitate efficient communication between multiple clients and a central server. The application consists of two main components: the **Client** and the **Server**.

- **Client**: Each client application will allow users to perform four primary actions:
    1. **LIST**: Retrieve the list of files stored on the server.
    2. **DIFF**: Compare the client's files with the server's and identify any files that are missing or different.
    3. **PULL**: Request files that are missing from the client, as identified in the DIFF step, and download them from the server.
    4. **LEAVE**: Terminate the client's session with the server.
- **Server**: The server application will manage multiple clients simultaneously, handling requests for file lists, diffs, and file transfers. It will be responsible for:
    - Maintaining a directory of files available for clients.
    - Computing file hashes to determine differences in files between the server and clients.
    - Sending the requested files to the clients.

2. Components and Data Flow

- **Network Communication**:
    - The client will communicate with the server over TCP using sockets. Each client will establish a connection to the server, allowing for direct message exchanges.
    - Commands will be sent as string messages, which will be parsed by the server to determine the appropriate action.
- **Multithreading**:

  - The server will utilize **Pthreads** to manage multiple client connections concurrently. Each client connection will be handled in its own thread, enabling simultaneous processing of requests without blocking other clients.

3. Justification for Using Pthreads

The choice to use **Pthreads** for this application architecture is based on the following considerations:

- **Concurrency**: Pthreads allows the server to handle multiple clients at the same time by creating a new thread for each client connection. This design enhances the responsiveness of the server and provides a smoother user experience, as clients can interact with the server without delays due to other clients' requests.
- **Simplicity of Implementation**: Implementing Pthreads is straightforward in C, especially for this application, where each client's requests are independent. The server can easily manage state and data for each client in its own thread without complex state management that would be necessary with select().
- **Performance**: With Pthreads, the server can take advantage of multi-core processors, distributing client requests across multiple threads. This leads to improved performance, especially when handling file transfers, which can be I/O intensive.
- **Ease of Error Handling**: Pthreads allow for simpler error handling and debugging since each client's state is managed within its own thread context. If one thread fails, it does not necessarily impact the others, allowing for better stability in handling client requests.

4. File Synchronization Logic

The application will implement the following logic for file synchronization:

- **LIST**: When a client sends a LIST command, the server will respond with the names of files stored in its directory.
- **DIFF**: Upon receiving a DIFF command, the server will compute the hashes of its files and compare them with the hashes received from the client. It will respond with a list of files that are either missing or have different content.
- **PULL**: For the PULL command, the client will request the specific files identified in the DIFF step. The server will read the requested files from its directory and send them to the client.

- **LEAVE**: When the LEAVE command is sent, the server will clean up the resources associated with that client thread and close the connection.

5. Conclusion

The architecture of the File/Music Synchronization Application is structured to ensure efficient and reliable communication between multiple clients and a central server. By leveraging Pthreads for concurrency, the application aims to provide a responsive user experience while maintaining the integrity of file synchronization operations. This design allows for effective handling of client requests and lays the groundwork for future enhancements, such as error recovery and user authentication.