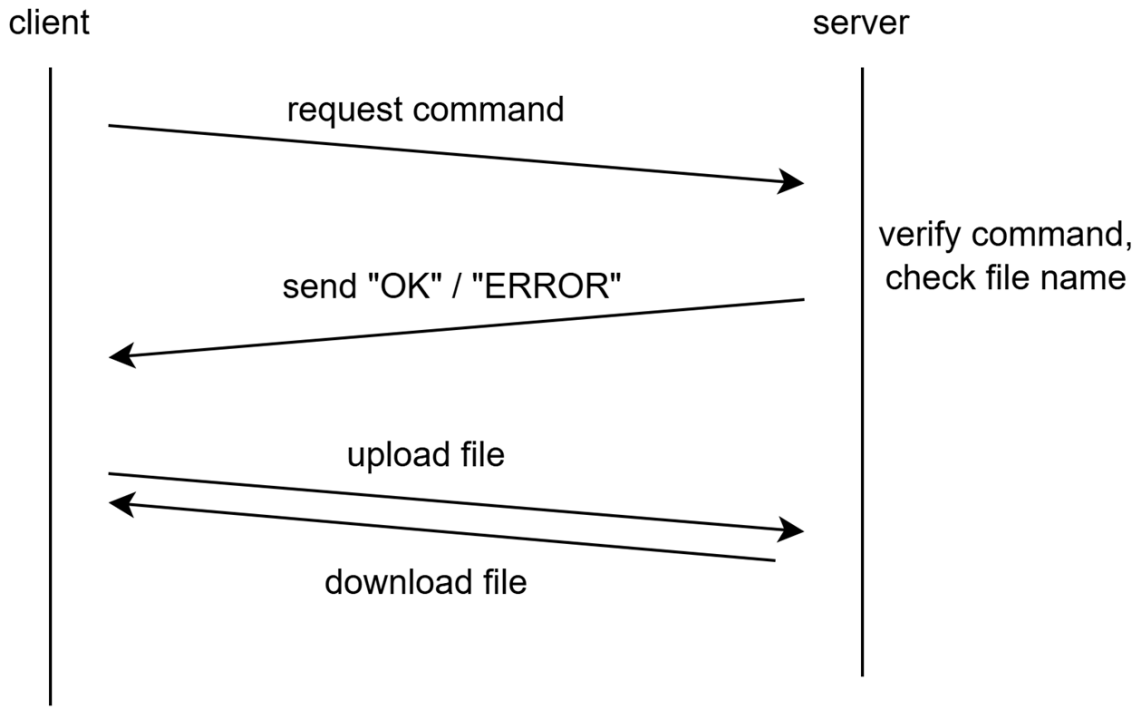
1. Protocol Design

Command phase: The client sends a command (ls, get, put, delete, rename) to server.

Verification phase: The receiver checks if the command is valid. For uploads/downloads, it checks file existence and sends a header response (OK or ERROR).

Data phase: If OK, response or data are sent to client. File is sent in 1024-byte chunks.



2. System Organization

Server side: The server is an iterative process. It binds to a specific IP and port (127.0.0.1:65432) and listens for incoming TCP connections. It handles one client connection at a time in a continuous loop until that client sends a quit command or disconnects.

Client side: The client is a command-line interface that connects to the server. It interprets user inputs, sends them over the socket.

3. File Transfer Implementation

The file transfer mechanism is implemented using chunked data transfer.

Phase	PUT (Upload)	GET (Download)
1. Command	Client sends: put <remote-file> <filesize>	Client sends: get <remote-file>
2. Verification	Server checks if file name exists to avoid overwriting it.	Server checks if the file exists on the server .
3. Data/Response	Server sends OK. Client then sends the binary file data.	Server sends OK <filesize>. Server then sends the binary file data.

Client uploads file to server (put)

```
with open(local_path, 'rb') as f:
    while True:
        bytes_read = f.read(BUFFER_SIZE)
        if not bytes_read: break
        client_socket.sendall(bytes_read)
print("Upload complete.")
```

The file is opened in binary mode. The while true loop reads BUFFER_SIZE (1024 bytes) at a time. client_socket.sendall(bytes_read) is used to guarantee that the entire chunk is transmitted before the loop continues, preventing data loss.

Client downloads file from server (get)

```
if os.path.exists(filename):
    filesize = os.path.getsize(filename)
    send_message(conn, f"OK {filesize}")
    with open(filename, 'rb') as f:
        while True:
            bytes_read = f.read(BUFFER_SIZE)
            if not bytes_read: break
            conn.sendall(bytes_read)

    print(f"Sent: {filename}")
```

First, os.path.exists(filename) verifies if the target file exists on the server disk. The server then sends a OK {filesize}. Finally, the file is opened in binary mode and the loop reads and sends 1024-byte chunks until the transfer is complete.