Firstly, both programs conduct extensive argument checking. Apart from just checking the input for appropriate data types, they also perform exception handling on given inputs relating to ports and sockets, giving helpful error messages if there is an exception.

The server program, after receiving valid parameters will open a socket (srv_sock) and attempt to bind it to the inputted port number. If this is unsuccessful, the program will leave a helpful error message and then terminate, otherwise the server prints that it is running and it proceeds to listen for connection requests. This is set to hold a backlog of 5 connections to be able to handle multiple requests if they come up.

The client program will perform more extensive argument checking as the user needs to give a valid command and a filename if appropriate. The client then opens a socket (cli_sock) and requests to connect to the server through the address and port given, and it will give an error message and terminating if this is unsuccessful.

If successful, the client program will notify the user that it has connected successfully. Then the program will encode a string consisting of a request flag and the filename and send it to the server. The client then either sends data or prepares to receive data, depending on the request. All the data sent and received for these requests is handled 4096 bytes at a time to reduce strain on the connection and be quicker to process. A large limit like this allows for large filenames to be transferred as well.

When the server receives the connection request from the client, it will accept and print a message notifying the user that a connection has been made. The server then proceeds to receive the request message from the client and decodes it to obtain the request flag and filename. The server then responds as appropriate to the request.

The process for sending and receiving files is identical between the client and server. In the case of sending, the target file is opened in a read-only binary mode and data is sent 4096 bytes at a time until there is no more data to be read. When receiving, a file is created using exclusive creation in binary mode and data is received 4096 bytes at a time. This data is written directly to the file until no data is being received anymore. Because these programs stop reading or writing when there is no data left to read or write, files with no contents are accounted for. Files with no content can be created as the creation of the file takes place before the program starts to check whether it has received data.

The process of requesting the top directory to be listed is similar. After receiving the request, the server creates a string consisting of the directory data separated by commas, it encodes it, and proceeds to sent it to the client. The client receives these 4096 bytes at a time and stores them in an internal variable until it stops receiving data. This data is then processed and printed.

Exception handling is done throughout the request handling process to ensure robustness. The client will report on the success of its request, close its socket, and terminate. The server will also produce a report and then continue to run so it can process further requests.

The data sent between the two programs first consists of a request flag and filename sent to the server and then data sent to or from the server, depending on the request. This simple implementation exploits the properties of the ordered bytestreams used in TCP connections. They allow for important flags to be sent before the data, so requests can be handled in a simple way.