Pseudocode for server code:

1. Include necessary libraries and header files

- Input/Output

- File handling

- Socket programming

- Threading

- Logger

2. Initialize a Logger instance for logging server activities

3. Define a function `handle\_client` with a parameter:

- `client\_socket` (socket descriptor for the connected client)

Inside `handle\_client` function:

a. Define a buffer for data transfer

b. Log the client connection

c. Receive file information from the client

- If receiving fails, log an error message and close the client socket

d. Extract filename and file size from the received information

e. Open a file for writing received data

- If opening the file fails, log an error message and close the client socket

f. Receive the file data in chunks

- Write each chunk to the file

- Update the remaining file size

- If receiving fails, log an error message

g. Close the file

h. Log the successful receipt of the file

i. Close the client socket

4. Define the `main` function

a. Create a socket for server communication

- If socket creation fails, log an error and exit

b. Set up the server address structure

- Set the address family to IPv4

- Set the port number

- Bind the socket to the server address

- If binding fails, log an error and exit

c. Start listening for incoming connections

- If listening fails, log an error and exit

d. Log that the server is listening on the specified port

e. Enter an infinite loop to accept incoming connections

- Accept a client connection

- If accepting fails, log an error and continue to the next connection

f. For each accepted client connection:

- Create a new thread to handle the client

- Detach the thread to handle the client independently

g. Close the server socket (unreachable in the current loop)

5. End of program

Pseudocode for client:

1. Include necessary libraries and header files

- Input/Output

- File handling

- Socket programming

- Logger

2. Initialize a Logger instance for logging client activities

3. Define a function `send\_file` with parameters:

- `filename` (the file to be sent)

- `server\_ip` (the IP address of the server)

- `server\_port` (the port on which the server is listening)

Inside `send\_file` function:

a. Create a socket for communication

- If socket creation fails, log an error message and exit

b. Set up the server address structure

- Set the address family to IPv4

- Set the port number

- Convert the IP address from string to network format

- If address conversion fails, log an error message and close the socket

c. Connect to the server using the socket

- If connection fails, log an error message and close the socket

d. Open the file to be sent

- If file opening fails, log an error message and close the socket

e. Get the file size

- Prepare a file info string containing filename and file size

- Send the file info string to the server

f. Send the file content in chunks

- Read the file content into a buffer

- Send the buffer contents to the server

- Continue until the entire file is sent

g. Close the file

h. Log a success message

i. Close the socket

4. Define the `main` function

a. Prompt the user for the filename to send

b. Call `send\_file` function with the user-provided filename, server IP, and port

5. End of program