project name :- Network File Sharing Server And Client

**Team members :- J . Ameesha**

**B. Supriya**

**M. Haritha**

**High-Level Design**

**Components**

**Server:**

**Responsibilities:**

Listen for incoming client connections.

Receive file requests from clients.

Send requested files to clients.

Handle multiple clients (optional for scalability).

**Client:**

**Responsibilities:**

Connect to the server.

Send file requests to the server.

Receive and save files from the server.

**Communication Protocol**

**Protocol :-**

**Request:** Client sends a request message containing the file name.

**Response:** Server responds by sending the file in chunks.

**Error Handling:** If the file is not found, the server sends an error message.

Workflow

**Client Requests File:**

The client sends a file request message (file name) to the server.

Server Handles Request:

The server receives the request, opens the requested file, and sends it to the client in chunks.

**Client Receives File:**

The client receives the file data in chunks and writes it to a file on disk.

Low-Level Design

Server :-

**Socket Creation:**

Create a socket using socket.socket().

**Bind to Port:**

Bind the socket to a specific port and IP address using socket.bind().

**Listen for Connections:**

Set the socket to listen for incoming connections using socket.listen().

**Accept Connection:**

Accept a client connection using socket.accept().

**Receive File Request:**

Receive the file name request from the client using socket.recv().

**Open and Send File:**

Open the requested file and send it in chunks using file.read() and socket.sendall().

**Error Handling:**

If the file is not found, send an error message to the client.

**Close Connections:**

Close the file and client connection using file.close() and socket.close().

**Client :-**

**1. Socket Creation:**

Create a socket using socket.socket().

**2. Connect to Server:**

Connect the socket to the server using socket.connect().

**3. Send File Request:**

Send the file name request to the server using socket.sendall().

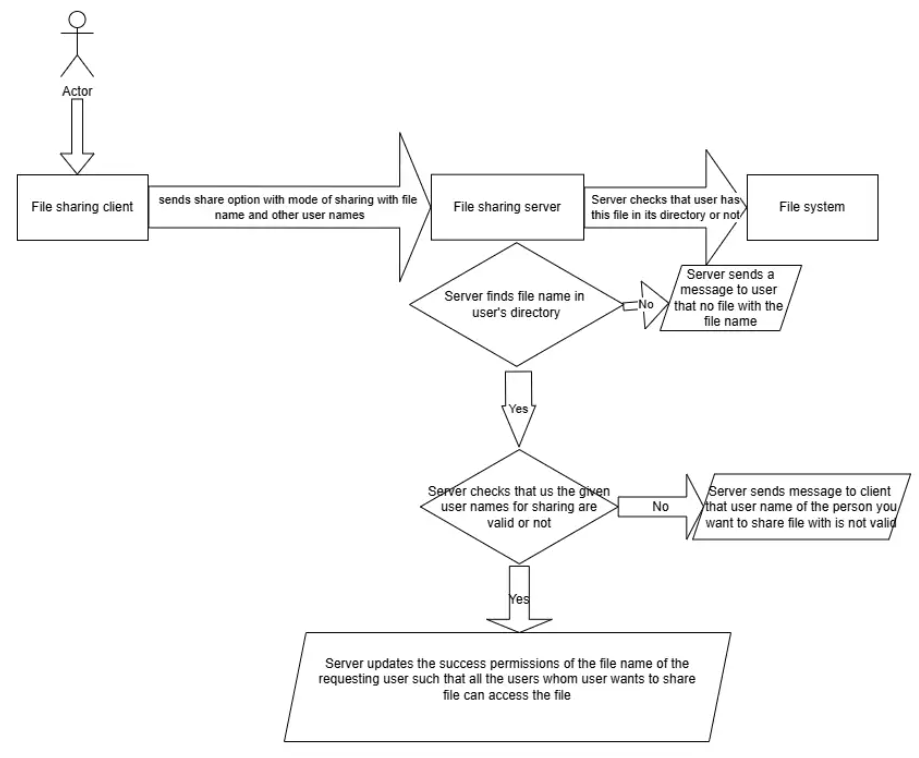
**4. Receive and Save File:**

Receive the file data in chunks and write it to a file on disk using file.write() and socket.recv().

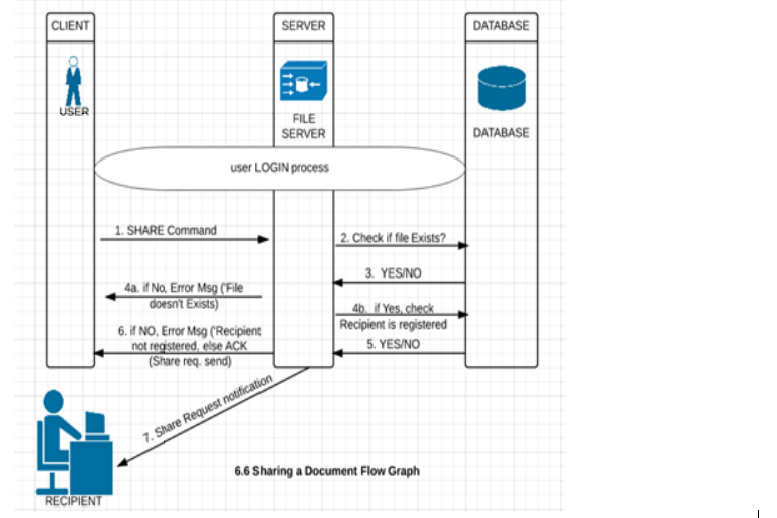
**5. Close Connection:**

Close the socket connection using socket.close().

**FLOWCHART :-**

****

**Sharing a Document Flow Graph :-**

****

**PSUEDOCODES:**

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**

**LOGGER CODES:**

CLASS **CODE FOR LOGGER.H :-**

**#ifndef LOGGER\_H**

**#define LOGGER\_H**

**#include <iostream>**

**#include <fstream>**

**#include <string>**

**#include <mutex>**

**class Logger {**

**private:**

**std::ofstream log\_file;**

**std::mutex log\_mutex;**

**public:**

**Logger(const std::string &file\_name) {**

**log\_file.open(file\_name, std::ios::app);**

**if (!log\_file.is\_open()) {**

**std::cerr << "Failed to open log file: " << file\_name << std::endl;**

**}**

**}**

**~Logger() {**

**if (log\_file.is\_open()) {**

**log\_file.close();**

**}**

**}**

**void log(const std::string &message) {**

**std::lock\_guard<std::mutex> guard(log\_mutex);**

**std::cout << message << std::endl;**

**if (log\_file.is\_open()) {**

**log\_file << message << std::endl;**

**}**

**}**

**};**

**#endif**

**SERVER CODE WITH LOGGER.H :-**

**#include <iostream>**

**#include <fstream>**

**#include <cstring>**

**#include <sys/socket.h>**

**#include <netinet/in.h>**

**#include <unistd.h>**

**#include <thread>**

**#include "logger.h" // Include the logger**

**Logger logger("server\_log.txt"); // Create a logger instance for the server**

**void handle\_client(int client\_socket) {**

**char buffer[1024];**

**// Log client connection**

**logger.log("Client connected");**

**// Receive file info**

**ssize\_t bytes\_received = recv(client\_socket, buffer, sizeof(buffer) - 1, 0);**

**if (bytes\_received <= 0) {**

**logger.log("Error receiving file info");**

**close(client\_socket);**

**return;**

**}**

**buffer[bytes\_received] = '\0';**

**std::string file\_info(buffer);**

**size\_t comma\_pos = file\_info.find(',');**

**std::string filename = file\_info.substr(0, comma\_pos);**

**size\_t filesize = std::stoul(file\_info.substr(comma\_pos + 1));**

**// Open file to write received data**

**std::ofstream output\_file(filename, std::ios::binary);**

**if (!output\_file) {**

**logger.log("Error opening file");**

**close(client\_socket);**

**return;**

**}**

**// Receive file data**

**while (filesize > 0) {**

**ssize\_t bytes\_received = recv(client\_socket, buffer, sizeof(buffer), 0);**

**if (bytes\_received <= 0) {**

**logger.log("Error receiving file data");**

**break;**

**}**

**output\_file.write(buffer, bytes\_received);**

**filesize -= bytes\_received;**

**}**

**output\_file.close();**

**logger.log("Received file: " + filename);**

**close(client\_socket);**

**}**

**int main() {**

**int server\_fd = socket(AF\_INET, SOCK\_STREAM, 0);**

**if (server\_fd < 0) {**

**logger.log("Socket creation error");**

**return 1;**

**}**

**sockaddr\_in server\_addr {};**

**server\_addr.sin\_family = AF\_INET;**

**server\_addr.sin\_addr.s\_addr = INADDR\_ANY;**

**server\_addr.sin\_port = htons(65432);**

**if (bind(server\_fd, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0) {**

**logger.log("Bind failed");**

**close(server\_fd);**

**return 1;**

**}**

**if (listen(server\_fd, 5) < 0) {**

**logger.log("Listen failed");**

**close(server\_fd);**

**return 1;**

**}**

**logger.log("Server listening on port 65432");**

**while (true) {**

**sockaddr\_in client\_addr {};**

**socklen\_t client\_addr\_len = sizeof(client\_addr);**

**int client\_socket = accept(server\_fd, (struct sockaddr\*)&client\_addr, &client\_addr\_len);**

**if (client\_socket < 0) {**

**logger.log("Accept failed");**

**continue;**

**}**

**std::thread(handle\_client, client\_socket).detach();**

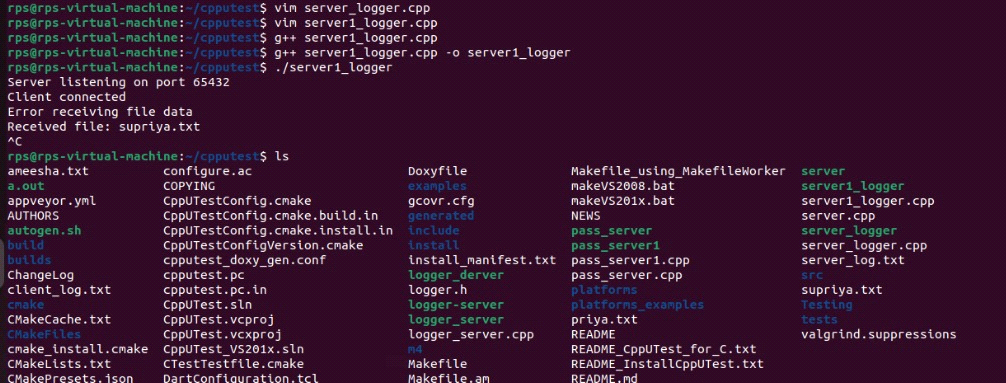
**}**

**close(server\_fd);**

**return 0;**

**}**

**OUTPUT FOR SERVER CODE WITH LOGGER.H :-**

****

**CLIENT CODE WITH LOGGER.H :-**

**#include <iostream>**

**#include <fstream>**

**#include <cstring>**

**#include <sys/socket.h>**

**#include <netinet/in.h>**

**#include <unistd.h>**

**#include "logger.h" // Include the logger**

**Logger logger("client\_log.txt"); // Create a logger instance for the client**

**void send\_file(const std::string& filename, const std::string& server\_ip, uint16\_t server\_port) {**

**int client\_socket = socket(AF\_INET, SOCK\_STREAM, 0);**

**if (client\_socket < 0) {**

**logger.log("Socket creation error");**

**return;**

**}**

**sockaddr\_in server\_addr {};**

**server\_addr.sin\_family = AF\_INET;**

**server\_addr.sin\_port = htons(server\_port);**

**if (inet\_pton(AF\_INET, server\_ip.c\_str(), &server\_addr.sin\_addr) <= 0) {**

**logger.log("Invalid address/ Address not supported");**

**close(client\_socket);**

**return;**

**}**

**if (connect(client\_socket, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0) {**

**logger.log("Connection failed");**

**close(client\_socket);**

**return;**

**}**

**std::ifstream input\_file(filename, std::ios::binary | std::ios::ate);**

**if (!input\_file) {**

**logger.log("Error opening file");**

**close(client\_socket);**

**return;**

**}**

**std::streamsize filesize = input\_file.tellg();**

**input\_file.seekg(0, std::ios::beg);**

**std::string file\_info = filename + "," + std::to\_string(filesize);**

**send(client\_socket, file\_info.c\_str(), file\_info.size(), 0);**

**char buffer[1024];**

**while (input\_file.read(buffer, sizeof(buffer)) || input\_file.gcount() > 0) {**

**send(client\_socket, buffer, input\_file.gcount(), 0);**

**}**

**input\_file.close();**

**logger.log("File sent successfully");**

**close(client\_socket);**

**}**

**int main() {**

**std::string filename;**

**std::cout << "Enter the filename to send: ";**

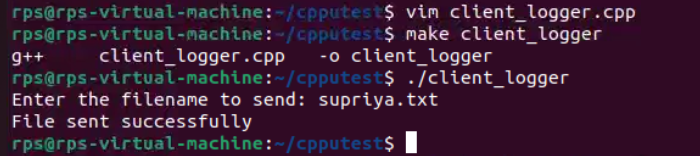
**std::cin >> filename;**

**send\_file(filename, "127.0.0.1", 65432);**

**return 0;**

**}**

**OUTPUT FOR CLIENT WITH LOGGER.H :-**

****