**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().

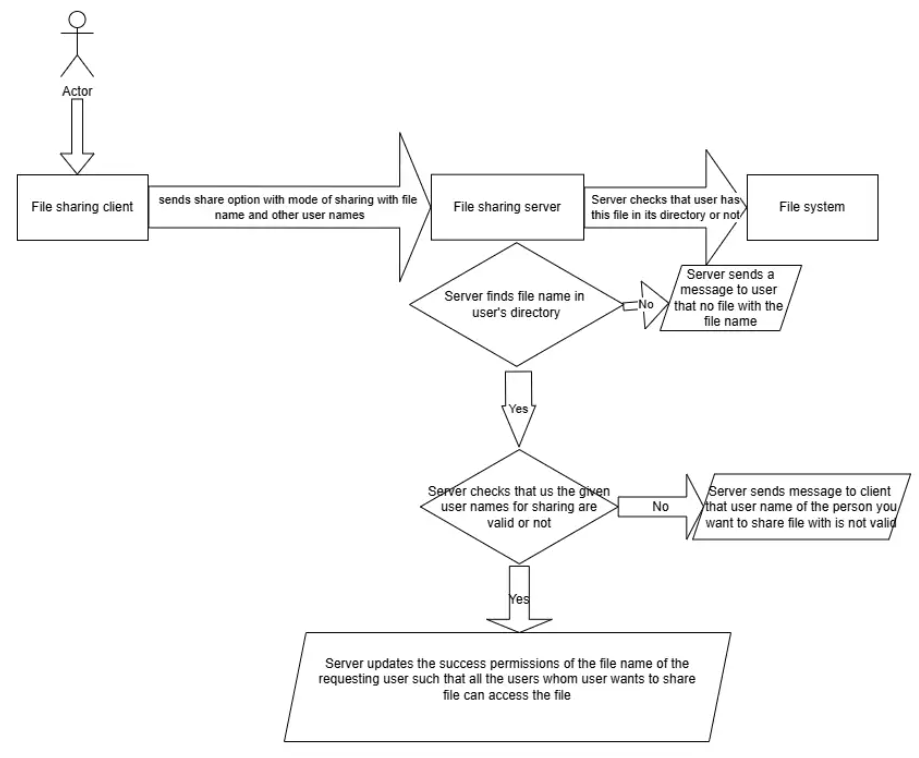
**4. Receive and Save File:**

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

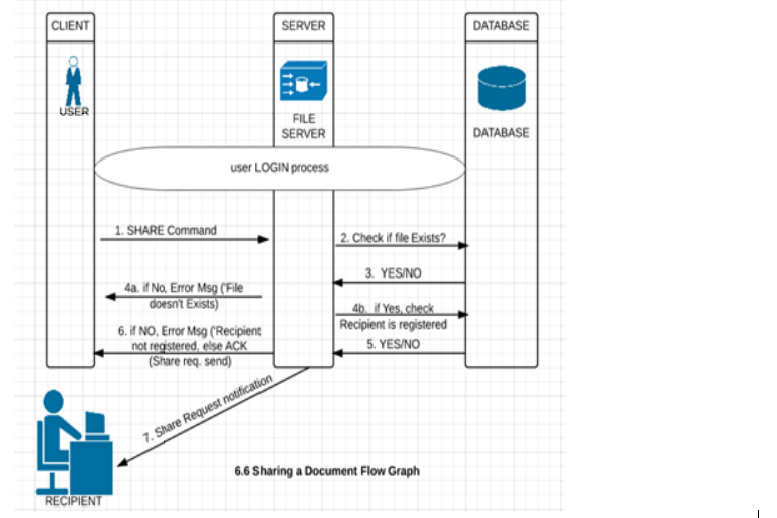
**5. Close Connection:**

Close the socket connection using socket.close().

**FLOWCHART :-**

****

**Sharing a Document Flow Graph :-**

****

**FILES SHARING FROM CLIENT TO SERVER :-**

**SERVER CODE :-**

#include <iostream>

#include <fstream>

#include <cstring>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <thread>

void handle\_client(int client\_socket) {

char buffer[1024];

// Receive file info

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

if (bytes\_received <= 0) {

std::cerr << "Error receiving file info\n";

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) {

std::cerr << "Error opening file\n";

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) {

std::cerr << "Error receiving file data\n";

break;

}

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

filesize -= bytes\_received;

}

output\_file.close();

std::cout << "Received file: " << filename << "\n";

close(client\_socket);

}

int main() {

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

if (server\_fd < 0) {

std::cerr << "Socket creation error\n";

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) {

std::cerr << "Bind failed\n";

close(server\_fd);

return 1;

}

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

std::cerr << "Listen failed\n";

close(server\_fd);

return 1;

}

std::cout << "Server listening on port 65432\n";

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) {

std::cerr << "Accept failed\n";

continue;

}

std::cout << "Client connected\n";

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

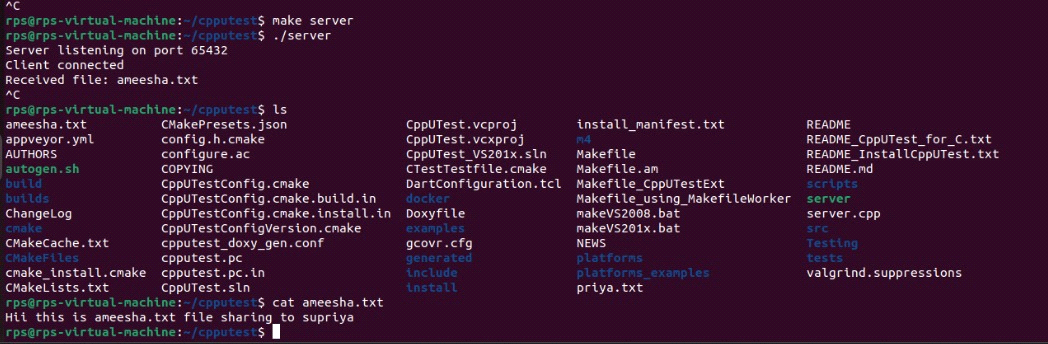
}

close(server\_fd);

return 0;

}

**OUTPUT :-**

****

**CLIENT CODE :-**

#include <iostream>

#include <fstream>

#include <cstring>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

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) {

std::cerr << "Socket creation error\n";

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) {

std::cerr << "Invalid address/ Address not supported\n";

close(client\_socket);

return;

}

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

std::cerr << "Connection failed\n";

close(client\_socket);

return;

}

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

if (!input\_file) {

std::cerr << "Error opening file\n";

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();

std::cout << "File sent successfully\n";

close(client\_socket);

}

int main() {

std::string filename;

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

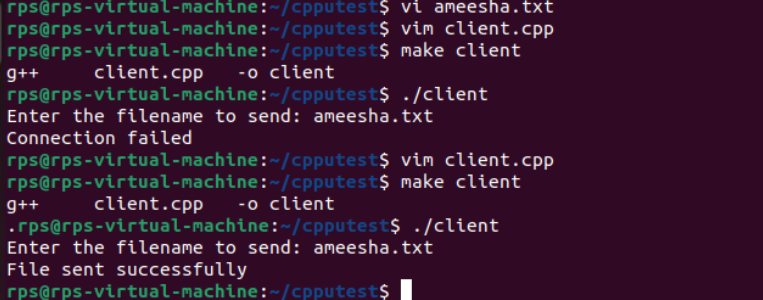
std::cin >> filename;

send\_file(filename, "172.20.0.13", 65432);

return 0;

}

**OUTPUT :-**

****

**PSEUDOCODES :-**

**PSEUDOCODE FOR SERVER :-**

START

INITIALIZE logger for logging

CREATE socket

IF socket creation fails THEN

LOG "Socket failed"

EXIT program

ENDIF

LOG "Socket created"

SET server address and port

BIND the socket to the server address and port

IF binding fails THEN

LOG "Bind failed"

EXIT program

ENDIF

LOG "Bind successful"

LISTEN for incoming connections

IF listening fails THEN

LOG "Listen failed"

EXIT program

ENDIF

LOG "Listening for connections..."

ACCEPT an incoming connection

IF accept fails THEN

LOG "Accept failed"

EXIT program

ENDIF

LOG "Connection accepted"

OPEN file to save received data

IF file fails to open THEN

LOG "Failed to open file for writing"

EXIT program

ENDIF

LOG "Receiving file..."

WHILE receiving data from the client

WRITE received data to the file

END WHILE

LOG "File received and saved"

CLOSE the connection

LOG "Connection closed"

END

**PSEUDOCODE FOR CLIENT :-**

START

DEFINE FUNCTION fileExists WITH PARAMETER fileName

TRY TO OPEN file with name fileName

IF file is successfully opened THEN

RETURN true

ELSE

RETURN false

ENDIF

END FUNCTION

INITIALIZE logger for logging

SET fileName to the name of the file to send (e.g., "file\_to\_send.txt")

CALL fileExists WITH fileName

IF file does not exist THEN

LOG "File does not exist"

EXIT program

ENDIF

CREATE socket

IF socket creation fails THEN

LOG "Socket creation error"

EXIT program

ENDIF

LOG "Socket created"

SET server address and port

TRY to convert server address to binary form

IF conversion fails THEN

LOG "Invalid address"

EXIT program

ENDIF

TRY to connect to the server

IF connection fails THEN

LOG "Connection failed"

EXIT program

ENDIF

LOG "Connected to server"

OPEN the file to send

IF file fails to open THEN

LOG "Failed to open file for reading"

EXIT program

ENDIF

LOG "Sending file..."

WHILE reading data from the file into the buffer

SEND data in the buffer to the server

END WHILE

LOG "File sent"

CLOSE the socket

LOG "Connection closed"

END

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

// logger.h

#ifndef LOGGER\_H

#define LOGGER\_H

#include <iostream>

#include <fstream>

#include <string>

#include <ctime>

class Logger {

public:

Logger(const std::string& logFile) : logStream(logFile, std::ios::app) {

if (!logStream.is\_open()) {

throw std::runtime\_error("Unable to open log file: " + logFile);

}

}

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

std::string timeStamp = getCurrentTime();

std::cout << timeStamp << " - " << message << std::endl;

logStream << timeStamp << " - " << message << std::endl;

}

private:

std::ofstream logStream;

std::string getCurrentTime() {

std::time\_t now = std::time(nullptr);

char buf[80];

std::strftime(buf, sizeof(buf), "%Y-%m-%d %H:%M:%S", std::localtime(&now));

return buf;

}

};

#endif // LOGGER\_H

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

// server.cpp

#include <iostream>

#include <fstream>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include "logger.h"

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

Logger logger("server\_log.txt");

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

// Creating socket file descriptor

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

logger.log("Socket failed");

exit(EXIT\_FAILURE);

}

logger.log("Socket created");

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

// Binding socket to the port

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

logger.log("Bind failed");

exit(EXIT\_FAILURE);

}

logger.log("Bind successful");

// Start listening for incoming connections

if (listen(server\_fd, 3) < 0) {

logger.log("Listen failed");

exit(EXIT\_FAILURE);

}

logger.log("Listening for connections...");

// Accept a connection

if ((new\_socket = accept(server\_fd, (struct sockaddr\*)&address, (socklen\_t\*)&addrlen)) < 0) {

logger.log("Accept failed");

exit(EXIT\_FAILURE);

}

logger.log("Connection accepted");

// Receive the file

std::ofstream outFile("received\_file.txt", std::ios::binary);

if (!outFile) {

logger.log("Failed to open file for writing");

exit(EXIT\_FAILURE);

}

logger.log("Receiving file...");

int bytesRead;

while ((bytesRead = read(new\_socket, buffer, BUFFER\_SIZE)) > 0) {

outFile.write(buffer, bytesRead);

}

outFile.close();

logger.log("File received and saved as 'received\_file.txt'");

// Close the socket

close(new\_socket);

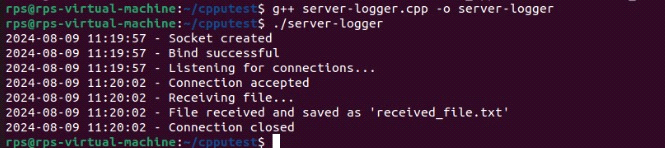
close(server\_fd);

logger.log("Connection closed");

return 0;

}

**OUTPUT FOR SERVER CODE :-**

****

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

// client.cpp

#include <iostream>

#include <fstream>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include "logger.h"

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

Logger logger("client\_log.txt");

int sock = 0;

struct sockaddr\_in serv\_addr;

char buffer[BUFFER\_SIZE] = {0};

// Create socket

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

logger.log("Socket creation error");

return -1;

}

logger.log("Socket created");

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

// Convert IPv4 and IPv6 addresses from text to binary form

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) {

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

return -1;

}

// Connect to server

if (connect(sock, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr)) < 0) {

logger.log("Connection failed");

return -1;

}

logger.log("Connected to server");

// Send the file

std::ifstream inFile("file\_to\_send.txt", std::ios::binary);

if (!inFile) {

logger.log("Failed to open file for reading");

return -1;

}

logger.log("Sending file...");

while (inFile.read(buffer, BUFFER\_SIZE)) {

send(sock, buffer, inFile.gcount(), 0);

}

if (inFile.gcount() > 0) {

send(sock, buffer, inFile.gcount(), 0);

}

logger.log("File sent");

// Close the socket

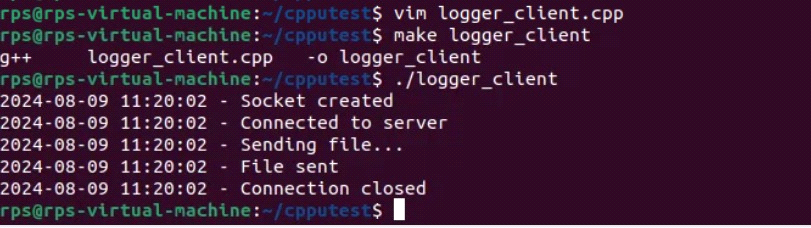
close(sock);

logger.log("Connection closed");

return 0;

}

**OUTPUT FOR CLIENT CODE :-**

****

**PASSWORD AUTHENTICATIONS**

**SERVER CODE FOR PASSWORD AUTHENTICATION :-**

#include <iostream>

#include <fstream>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include "logger.h"

#define PORT 8080

#define BUFFER\_SIZE 1024

#define PASSWORD "secret123" // Predefined password

int main() {

Logger logger("server\_log.txt");

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

// Creating socket file descriptor

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

logger.log("Socket failed");

exit(EXIT\_FAILURE);

}

logger.log("Socket created");

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

// Binding socket to the port

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

logger.log("Bind failed");

exit(EXIT\_FAILURE);

}

logger.log("Bind successful");

// Start listening for incoming connections

if (listen(server\_fd, 3) < 0) {

logger.log("Listen failed");

exit(EXIT\_FAILURE);

}

logger.log("Listening for connections...");

// Accept a connection

if ((new\_socket = accept(server\_fd, (struct sockaddr\*)&address, (socklen\_t\*)&addrlen)) < 0) {

logger.log("Accept failed");

exit(EXIT\_FAILURE);

}

logger.log("Connection accepted");

// Receive the password

int password\_len = read(new\_socket, buffer, BUFFER\_SIZE);

buffer[password\_len] = '\0';

std::string received\_password(buffer);

if (received\_password != PASSWORD) {

logger.log("Incorrect password");

close(new\_socket);

close(server\_fd);

return 0;

}

logger.log("Password accepted");

// Receive the file

std::ofstream outFile("received\_file.txt", std::ios::binary);

if (!outFile) {

logger.log("Failed to open file for writing");

exit(EXIT\_FAILURE);

}

logger.log("Receiving file...");

int bytesRead;

while ((bytesRead = read(new\_socket, buffer, BUFFER\_SIZE)) > 0) {

outFile.write(buffer, bytesRead);

}

outFile.close();

logger.log("File received and saved as 'received\_file.txt'");

// Close the socket

close(new\_socket);

close(server\_fd);

logger.log("Connection closed");

return 0;

}

**OUTPUT :-**

****

**CLIENT CODE FOR PASSWORD AUTHENTICATION :-**

#include <iostream>

#include <fstream>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include "logger.h"

#define PORT 8080

#define BUFFER\_SIZE 1024

bool fileExists(const std::string& fileName) {

std::ifstream infile(fileName);

return infile.good();

}

int main() {

Logger logger("client\_log.txt");

int sock = 0;

struct sockaddr\_in serv\_addr;

char buffer[BUFFER\_SIZE] = {0};

std::string fileName = "ameesha.txt";

if (!fileExists(fileName)) {

logger.log("File '" + fileName + "' does not exist.");

return -1;

}

// Create socket

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

logger.log("Socket creation error");

return -1;

}

logger.log("Socket created");

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

if (inet\_pton(AF\_INET, "172.20.0.13", &serv\_addr.sin\_addr) <= 0) {

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

return -1;

}

// Connect to server

if (connect(sock, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr)) < 0) {

logger.log("Connection failed");

return -1;

}

logger.log("Connected to server");

// Prompt for password

std::string password;

std::cout << "Enter password: ";

std::cin >> password;

// Send the password to the server

send(sock, password.c\_str(), password.size(), 0);

logger.log("Password sent");

// Send the file

std::ifstream inFile(fileName, std::ios::binary);

if (!inFile) {

logger.log("Failed to open file for reading");

return -1;

}

logger.log("Sending file...");

while (inFile.read(buffer, BUFFER\_SIZE)) {

send(sock, buffer, inFile.gcount(), 0);

}

if (inFile.gcount() > 0) {

send(sock, buffer, inFile.gcount(), 0);

}

logger.log("File sent");

// Close the socket

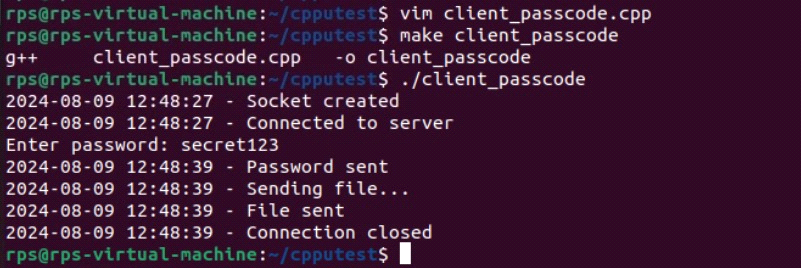
close(sock);

logger.log("Connection closed");

return 0;

}

**OUTPUT :-**

****

**SERVER CODE FOR ENCRYPTION AND PASSWORD :-**

#include <iostream>

#include <fstream>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include "logger.h"

#define PORT 8080

#define BUFFER\_SIZE 1024

#define PASSWORD "secret123" // Predefined password

#define XOR\_KEY 'K' // Key for XOR encryption

std::string xorEncryptDecrypt(const std::string &data, char key);

int main() {

Logger logger("server\_log.txt");

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

// Creating socket file descriptor

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

logger.log("Socket failed");

exit(EXIT\_FAILURE);

}

logger.log("Socket created");

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

// Binding socket to the port

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

logger.log("Bind failed");

exit(EXIT\_FAILURE);

}

logger.log("Bind successful");

// Start listening for incoming connections

if (listen(server\_fd, 3) < 0) {

logger.log("Listen failed");

exit(EXIT\_FAILURE);

}

logger.log("Listening for connections...");

// Accept a connection

if ((new\_socket = accept(server\_fd, (struct sockaddr\*)&address, (socklen\_t\*)&addrlen)) < 0) {

logger.log("Accept failed");

exit(EXIT\_FAILURE);

}

logger.log("Connection accepted");

// Receive the encrypted password

int password\_len = read(new\_socket, buffer, BUFFER\_SIZE);

buffer[password\_len] = '\0';

std::string received\_password\_encrypted(buffer);

// Decrypt the password

std::string received\_password = xorEncryptDecrypt(received\_password\_encrypted, XOR\_KEY);

if (received\_password != PASSWORD) {

logger.log("Incorrect password");

close(new\_socket);

close(server\_fd);

return 0;

}

logger.log("Password accepted");

// Receive the file

std::ofstream outFile("received\_file.txt", std::ios::binary);

if (!outFile) {

logger.log("Failed to open file for writing");

exit(EXIT\_FAILURE);

}

logger.log("Receiving file...");

int bytesRead;

while ((bytesRead = read(new\_socket, buffer, BUFFER\_SIZE)) > 0) {

outFile.write(buffer, bytesRead);

}

outFile.close();

logger.log("File received and saved as 'received\_file.txt'");

// Close the socket

close(new\_socket);

close(server\_fd);

logger.log("Connection closed");

return 0;

}

std::string xorEncryptDecrypt(const std::string &data, char key) {

std::string result = data;

for (size\_t i = 0; i < data.size(); ++i) {

result[i] = data[i] ^ key;

}

return result;

}

**OUTPUT :-**

****

**CLIENT CODE FOR ENCRYPTION AND PASSWORD :-**

#include <iostream>

#include <fstream>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include "logger.h"

#define PORT 8080

#define BUFFER\_SIZE 1024

#define XOR\_KEY 'K' // Key for XOR encryption

std::string xorEncryptDecrypt(const std::string &data, char key);

bool fileExists(const std::string& fileName) {

std::ifstream infile(fileName);

return infile.good();

}

int main() {

Logger logger("client\_log.txt");

int sock = 0;

struct sockaddr\_in serv\_addr;

char buffer[BUFFER\_SIZE] = {0};

std::string fileName = "ameesha.txt";

if (!fileExists(fileName)) {

logger.log("File '" + fileName + "' does not exist.");

return -1;

}

// Create socket

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

logger.log("Socket creation error");

return -1;

}

logger.log("Socket created");

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

if (inet\_pton(AF\_INET, "172.20.0.13", &serv\_addr.sin\_addr) <= 0) {

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

return -1;

}

// Connect to server

if (connect(sock, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr)) < 0) {

logger.log("Connection failed");

return -1;

}

logger.log("Connected to server");

// Prompt for password

std::string password;

std::cout << "Enter password: ";

std::cin >> password;

// Encrypt the password

std::string encrypted\_password = xorEncryptDecrypt(password, XOR\_KEY);

// Send the encrypted password to the server

send(sock, encrypted\_password.c\_str(), encrypted\_password.size(), 0);

logger.log("Encrypted password sent");

// Send the file

std::ifstream inFile(fileName, std::ios::binary);

if (!inFile) {

logger.log("Failed to open file for reading");

return -1;

}

logger.log("Sending file...");

while (inFile.read(buffer, BUFFER\_SIZE)) {

send(sock, buffer, inFile.gcount(), 0);

}

if (inFile.gcount() > 0) {

send(sock, buffer, inFile.gcount(), 0);

}

logger.log("File sent");

// Close the socket

close(sock);

logger.log("Connection closed");

return 0;

}

std::string xorEncryptDecrypt(const std::string &data, char key) {

std::string result = data;

for (size\_t i = 0; i < data.size(); ++i) {

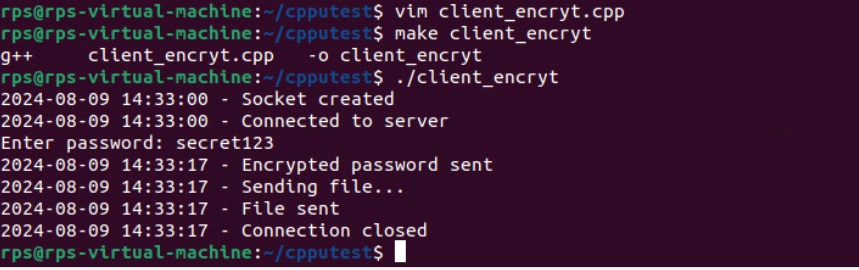
result[i] = data[i] ^ key;

}

return result;

}

**OUTPUT :-**

****