Capstone Project

Assignment 4(LSP)

Network File Sharing Server & Client

Submitted By

Name : **Lingaraj Senapati**

Batch : **05**

Department : **BTECH(CSE)**

Registration Number : **2241019368**

Section : **42**

ABSTRACT

This project, Network File Sharing Server and Client, demonstrates the implementation of socket programming in C++. It allows multiple clients to connect to a central server for file listing, uploading, and downloading using TCP sockets. The system also includes simple authentication and encryption mechanisms to ensure secure communication between the client and server.

OBJECTIVE

The objective of this project is to develop a networked file-sharing application with client-server architecture that enables reliable file transfer over TCP sockets. The system should allow user authentication, file listing, file upload, and download operations, and ensure secure data transmission.

Theoretical Background

Computer Networks

- A computer network is a collection of interconnected computers that communicate and share resources.
- It enables data exchange between devices using communication protocols over wired or wireless media.

Client–Server Architecture

- It is a network model where one system acts as a **server** (provider of resources) and another acts as a **client** (requester of resources).
- The client sends requests, and the server processes them and returns the responses.

- **Example:** Web browsers (clients) communicating with web servers.

Sockets

- A socket is an endpoint used for sending or receiving data across a computer network.
- In this project, **TCP (Transmission Control Protocol)** sockets are used.
- TCP ensures reliable, ordered, and error-checked data transmission between the client and the server.

IP Address and Port Number

- Every device on a network has a unique **IP address** (e.g., 127.0.0.1 for localhost).
- A **port number** identifies a specific process or service (e.g., 8080, 65432).
- Together, the IP address and port number uniquely identify a socket connection.

Data Transmission Process

- The client creates a socket and connects to the server using the server's IP address and port number.
- The server listens for incoming connections using the bind(), listen(), and accept() system calls.
- Once connected, both the client and server use send() and recv() to exchange data reliably.

File Sharing Concept

- The project demonstrates file transfer between two systems over a network.
- The client can request files (**download**) or send files (**upload**) to the server.

- The server handles multiple commands such as **LIST**, **GET**, **PUT**, and **QUIT**.

Authentication

- To ensure security, the client must enter a valid password before accessing files.
- The server verifies the credentials before permitting any operation.

Encryption

- A simple **XOR encryption** method is used to protect data during transmission.
- It encrypts messages by combining each character with a secret key, making the transmitted data unreadable to outsiders.
- Though basic, it effectively demonstrates the concept of data security in network communication.

Error Handling and Reliability

- TCP automatically handles lost packets and ensures that data reaches the correct destination.
- The program includes checks for missing files, invalid commands, and connection issues to improve reliability.

Practical Applications

- The same principles are applied in real-world systems such as **FTP (File Transfer Protocol)** and **cloud storage services**.
- This project serves as a simplified model of those systems for learning and demonstration purposes.

CODE

client.cpp - C++ Client Implementation using POSIX Sockets

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <sys/stat.h>
#include <algorithm>

#define SERVER_IP "127.0.0.1"
#define PORT 65432
#define BUFFER_SIZE 1024
const std::string CLIENT_DIR = "client_downloads/";

const std::string ENCRYPTION_KEY = "a_very_simple_shared_key";
void send_file(int sock, const std::string& filename);
void receive_file(int sock, const std::string& filename);
void xor_encrypt_decrypt(char* data, size_t len, const std::string& key);

int main() {
    int sock = 0;
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        std::cerr << "Socket creation error." << std::endl;
        return -1;
    }

    struct sockaddr_in serv_addr;
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    if (inet_pton(AF_INET, SERVER_IP, &serv_addr.sin_addr) <= 0) {
        std::cerr << "Invalid address/Address not supported." << std::endl;
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        std::cerr << "Connection Failed. Is the server running?" << std::endl;
        return -1;
```

```
    }

    std::cout << "Connected to server." << std::endl;
    char auth_buffer[BUFFER_SIZE] = {0};

    std::string user_entered_password;
    std::cout << "Please enter the server password: ";
    std::getline(std::cin, user_entered_password);

    strncpy(auth_buffer, user_entered_password.c_str(), BUFFER_SIZE);

    auth_buffer[BUFFER_SIZE - 1] = '\0';

    xor_encrypt_decrypt(auth_buffer, user_entered_password.length(),
ENCRYPTION_KEY);
    send(sock, auth_buffer, user_entered_password.length(), 0);

    memset(auth_buffer, 0, BUFFER_SIZE);
    int valread = recv(sock, auth_buffer, BUFFER_SIZE, 0);
    if (valread <= 0) {
        std::cerr << "Server disconnected during auth." << std::endl;
        close(sock);
        return -1;
    }

    xor_encrypt_decrypt(auth_buffer, valread, ENCRYPTION_KEY);
    std::string auth_response(auth_buffer, valread);

    if (auth_response != "AUTH_OK") {
        std::cerr << "Authentication failed. Server response: " <<
auth_response << std::endl;
        close(sock);
        return -1;
    }

    std::cout << "Authentication successful." << std::endl;
    std::cout << "Successfully connected to server " << SERVER_IP << ":" <<
PORT << std::endl;

    std::string command_line;
    char buffer[BUFFER_SIZE] = {0};

    while (true) {
        std::cout << "\nEnter command (LIST, GET <file>, PUT <file>, QUIT): ";
        std::getline(std::cin, command_line);

        if (command_line.empty()) continue;
```

```
        char command_buffer[BUFFER_SIZE] = {0};
        strncpy(command_buffer, command_line.c_str(), BUFFER_SIZE);

        command_buffer[BUFFER_SIZE - 1] = '\0';

        xor_encrypt_decrypt(command_buffer, command_line.length(),
ENCRYPTION_KEY);
        send(sock, command_buffer, command_line.length(), 0);

        if (command_line == "QUIT") {
            break;
        }
        std::string command, filename;
        size_t space_pos = command_line.find(' ');
        if (space_pos != std::string::npos) {
            command = command_line.substr(0, space_pos);
            filename = command_line.substr(space_pos + 1);
        } else {
            command = command_line;
        }

        if (command == "LIST") {
            memset(buffer, 0, BUFFER_SIZE);
            int bytes_recvd = recv(sock, buffer, BUFFER_SIZE, 0);
            if (bytes_recvd > 0) {
                xor_encrypt_decrypt(buffer, bytes_recvd, ENCRYPTION_KEY);
                std::cout << "\n--- Available Files ---\n" <<
std::string(buffer, bytes_recvd) << "-----------------------\n";
            }

        } else if (command == "GET" && !filename.empty()) {
            receive_file(sock, filename);

        } else if (command == "PUT" && !filename.empty()) {
            send_file(sock, filename);
        }
    }

    close(sock);
    std::cout << "Client connection closed." << std::endl;
    return 0;
}
void xor_encrypt_decrypt(char* data, size_t len, const std::string& key) {
    if (key.empty()) return;
    for (size_t i = 0; i < len; ++i) {
        data[i] = data[i] ^ key[i % key.length()];
    }
}
```

```
void send_file(int sock, const std::string& filename) {
    std::string filepath = CLIENT_DIR + filename;

    std::ifstream file(filepath, std::ios::in | std::ios::binary);
    if (!file.is_open()) {
        std::cerr << "Local file not found: " << filename << std::endl;
        return;
    }
    struct stat file_status;
    stat(filepath.c_str(), &file_status);
    long file_size = file_status.st_size;

    long encrypted_size = file_size;
    xor_encrypt_decrypt(reinterpret_cast<char*>(&encrypted_size),
sizeof(long), ENCRYPTION_KEY);
    send(sock, &encrypted_size, sizeof(long), 0);


    char buffer[BUFFER_SIZE];
    while (file.read(buffer, BUFFER_SIZE)) {

        xor_encrypt_decrypt(buffer, BUFFER_SIZE, ENCRYPTION_KEY);
        send(sock, buffer, BUFFER_SIZE, 0);
    }
    if (file.gcount() > 0) {
        xor_encrypt_decrypt(buffer, file.gcount(), ENCRYPTION_KEY);
        send(sock, buffer, file.gcount(), 0);
    }

    file.close();
    std::cout << "Successfully uploaded file: " << filename << std::endl;
}
void receive_file(int sock, const std::string& filename) {
    long file_size;
    if (recv(sock, &file_size, sizeof(long), 0) <= 0) {
        std::cerr << "Error receiving file size." << std::endl;
        return;
    }

    xor_encrypt_decrypt(reinterpret_cast<char*>(&file_size), sizeof(long),
ENCRYPTION_KEY);
    if (file_size == -1) {
        char error_buffer[BUFFER_SIZE] = {0};
        int bytes_recvd = recv(sock, error_buffer, BUFFER_SIZE, 0);
        if (bytes_recvd > 0) {
            xor_encrypt_decrypt(error_buffer, bytes_recvd, ENCRYPTION_KEY);
            std::cerr << "Server Error: " << std::string(error_buffer,
bytes_recvd) << std::endl;
```

```
        }
        return;
    }
    std::string filepath = CLIENT_DIR + filename;
    std::ofstream file(filepath, std::ios::out | std::ios::binary);
    if (!file.is_open()) {
        std::cerr << "Error opening file for writing: " << filename <<
std::endl;
        return;
    }
        char buffer[BUFFER_SIZE];
    long total_received = 0;

    while (total_received < file_size) {
        long bytes_to_read = std::min((long)BUFFER_SIZE, file_size -
total_received);

        int bytes_received = recv(sock, buffer, bytes_to_read, 0);

        if (bytes_received <= 0) {
            std::cerr << "Connection closed prematurely or error." <<
std::endl;
            break;
        }
        xor_encrypt_decrypt(buffer, bytes_received, ENCRYPTION_KEY);

        file.write(buffer, bytes_received);
        total_received += bytes_received;
    }

    file.close();
    std::cout << "Successfully downloaded file: " << filename << " (" <<
total_received << " bytes)" << std::endl;
}
```

server.cpp - C++ Implementation using POSIX Sockets

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
```

```cpp
#include <unistd.h>
#include <sys/stat.h>
#include <dirent.h>
#include <algorithm>
#define PORT 65432
#define BUFFER_SIZE 1024
const std::string SHARED_DIR = "server_files/";

const std::string ENCRYPTION_KEY = "a_very_simple_shared_key";
const std::string SERVER_PASSWORD = "supersecret123";

void handle_client(int client_socket);
void send_file(int client_socket, const std::string& filename);
void receive_file(int client_socket, const std::string& filename);
std::string list_files();
void xor_encrypt_decrypt(char* data, size_t len, const std::string& key);
int main() {
    int server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd == 0) {
        perror("socket failed");
        return 1;
    }
    struct sockaddr_in address;
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        return 1;
    }

    if (listen(server_fd, 3) < 0) {
        perror("listen failed");
        return 1;
    }

    std::cout << "Server listening on port " << PORT << std::endl;


    while (true) {
        int new_socket;
        socklen_t addrlen = sizeof(address);

        if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
&addrlen)) < 0) {
            perror("accept failed");
            continue;
```

```
        }

        std::cout << "New client connected. Waiting for authentication..." <<
std::endl;

        handle_client(new_socket);
    }

    close(server_fd);
    return 0;
}
void xor_encrypt_decrypt(char* data, size_t len, const std::string& key) {
    if (key.empty()) return;
    for (size_t i = 0; i < len; ++i) {
        data[i] = data[i] ^ key[i % key.length()];
    }
}

void handle_client(int client_socket) {
    char buffer[BUFFER_SIZE] = {0};
    int valread;

    valread = read(client_socket, buffer, BUFFER_SIZE);
    if (valread <= 0) {
        std::cerr << "Client disconnected before auth." << std::endl;
        close(client_socket);
        return;
    }

    xor_encrypt_decrypt(buffer, valread, ENCRYPTION_KEY);
    std::string received_password(buffer, valread);

    char response_buffer[32];
    if (received_password == SERVER_PASSWORD) {
        std::cout << "Client authenticated successfully." << std::endl;
        strncpy(response_buffer, "AUTH_OK", sizeof(response_buffer));
    } else {
        std::cerr << "Client failed authentication." << std::endl;
        strncpy(response_buffer, "AUTH_FAIL", sizeof(response_buffer));
    }

    xor_encrypt_decrypt(response_buffer, strlen(response_buffer),
ENCRYPTION_KEY);
    send(client_socket, response_buffer, strlen(response_buffer), 0);


    if (received_password != SERVER_PASSWORD) {
        close(client_socket);
```

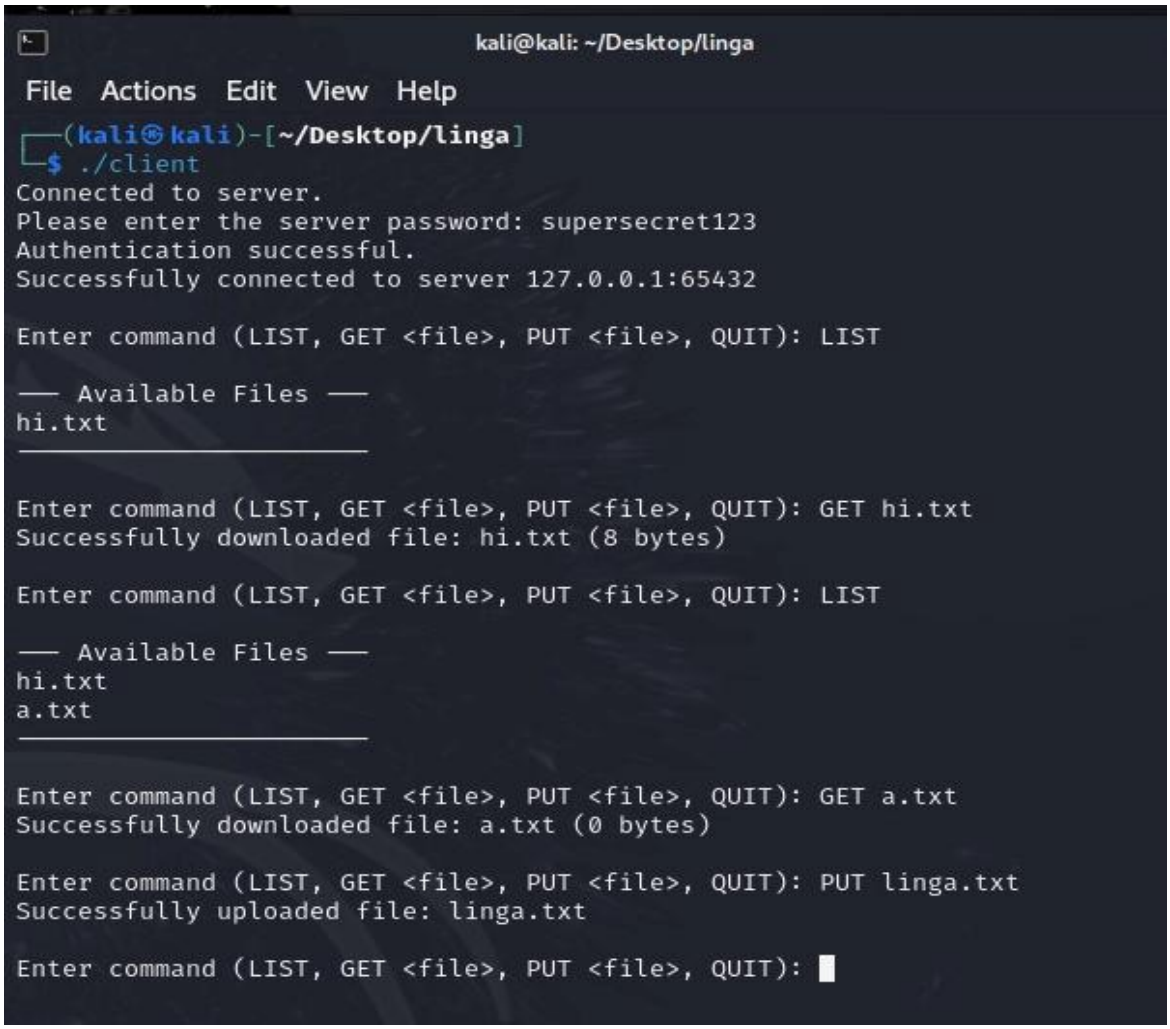
```
        return;
    }
    while ( (valread = read(client_socket, buffer, BUFFER_SIZE)) > 0) {

        xor_encrypt_decrypt(buffer, valread, ENCRYPTION_KEY);
        std::string command_line(buffer, valread);

        command_line.erase(command_line.find_last_not_of(" \n\r\t")+1);

        std::cout << "Received command: [" << command_line << "]" <<
std::endl;

        std::string command, filename;
        size_t space_pos = command_line.find(' ');
        if (space_pos != std::string::npos) {
            command = command_line.substr(0, space_pos);
            filename = command_line.substr(space_pos + 1);
        } else {
            command = command_line;
        }
        if (command == "LIST") {
            std::string file_list = list_files();
            xor_encrypt_decrypt(const_cast<char*>(file_list.c_str()),
file_list.length(), ENCRYPTION_KEY);
            send(client_socket, file_list.c_str(), file_list.length(), 0);

        } else if (command == "GET" && !filename.empty()) {
            send_file(client_socket, filename);

        } else if (command == "PUT" && !filename.empty()) {
            receive_file(client_socket, filename); inside

        } else if (command == "QUIT") {
            std::cout << "Client sent QUIT. Closing connection." << std::endl;
            break;
        }

        memset(buffer, 0, BUFFER_SIZE);
    }

    std::cout << "Client disconnected. Closing socket." << std::endl;
    close(client_socket);
}
void send_file(int client_socket, const std::string& filename) {
    std::string filepath = SHARED_DIR + filename;

    std::ifstream file(filepath, std::ios::in | std::ios::binary);
    if (!file.is_open()) {
```

```
        std::string msg = "FILE_NOT_FOUND";
        long file_size = -1;
        xor_encrypt_decrypt(reinterpret_cast<char*>(&file_size), sizeof(long),
ENCRYPTION_KEY);
        send(client_socket, &file_size, sizeof(long), 0);

        xor_encrypt_decrypt(const_cast<char*>(msg.c_str()), msg.length(),
ENCRYPTION_KEY);
        send(client_socket, msg.c_str(), msg.length(), 0);

        std::cerr << "File not found: " << filename << std::endl;
        return;
    }


    struct stat file_status;
    stat(filepath.c_str(), &file_status);
    long file_size = file_status.st_size;
    long encrypted_size = file_size;
    xor_encrypt_decrypt(reinterpret_cast<char*>(&encrypted_size),
sizeof(long), ENCRYPTION_KEY);
    send(client_socket, &encrypted_size, sizeof(long), 0);

    char buffer[BUFFER_SIZE];
    while (file.read(buffer, BUFFER_SIZE)) {

        xor_encrypt_decrypt(buffer, BUFFER_SIZE, ENCRYPTION_KEY);
        send(client_socket, buffer, BUFFER_SIZE, 0);
    }
    if (file.gcount() > 0) {

        xor_encrypt_decrypt(buffer, file.gcount(), ENCRYPTION_KEY);
        send(client_socket, buffer, file.gcount(), 0);
    }

    file.close();
    std::cout << "Successfully sent file: " << filename << std::endl;
}
void receive_file(int client_socket, const std::string& filename) {
    std::string filepath = SHARED_DIR + filename;


    long file_size;
    if (read(client_socket, &file_size, sizeof(long)) <= 0) {
        std::cerr << "Error receiving file size." << std::endl;
        return;
    }
```

```
    xor_encrypt_decrypt(reinterpret_cast<char*>(&file_size), sizeof(long),
ENCRYPTION_KEY);
    std::ofstream file(filepath, std::ios::out | std::ios::binary);
    if (!file.is_open()) {
        std::cerr << "Error opening file for writing: " << filename <<
std::endl;
        return;
    }

    char buffer[BUFFER_SIZE];
    long total_received = 0;

    while (total_received < file_size) {
        long bytes_to_read = std::min((long)BUFFER_SIZE, file_size -
total_received);

        int bytes_received = recv(client_socket, buffer, bytes_to_read, 0);

        if (bytes_received <= 0) {
            std::cerr << "Connection closed prematurely or error." <<
std::endl;
            break;
        }
        xor_encrypt_decrypt(buffer, bytes_received, ENCRYPTION_KEY);

        file.write(buffer, bytes_received);
        total_received += bytes_received;
    }

    file.close();
    std::cout << "Successfully received file: " << filename << " (" <<
total_received << " bytes)" << std::endl;
}

std::string list_files() {
    DIR *dir;
    struct dirent *ent;
    std::string list = "";

    if ((dir = opendir(SHARED_DIR.c_str())) != NULL) {
        while ((ent = readdir(dir)) != NULL) {
            std::string name(ent->d_name);
            if (name != "." && name != "..") {
                list += name + "\n";
            }
        }
        closedir(dir);
    } else {
```

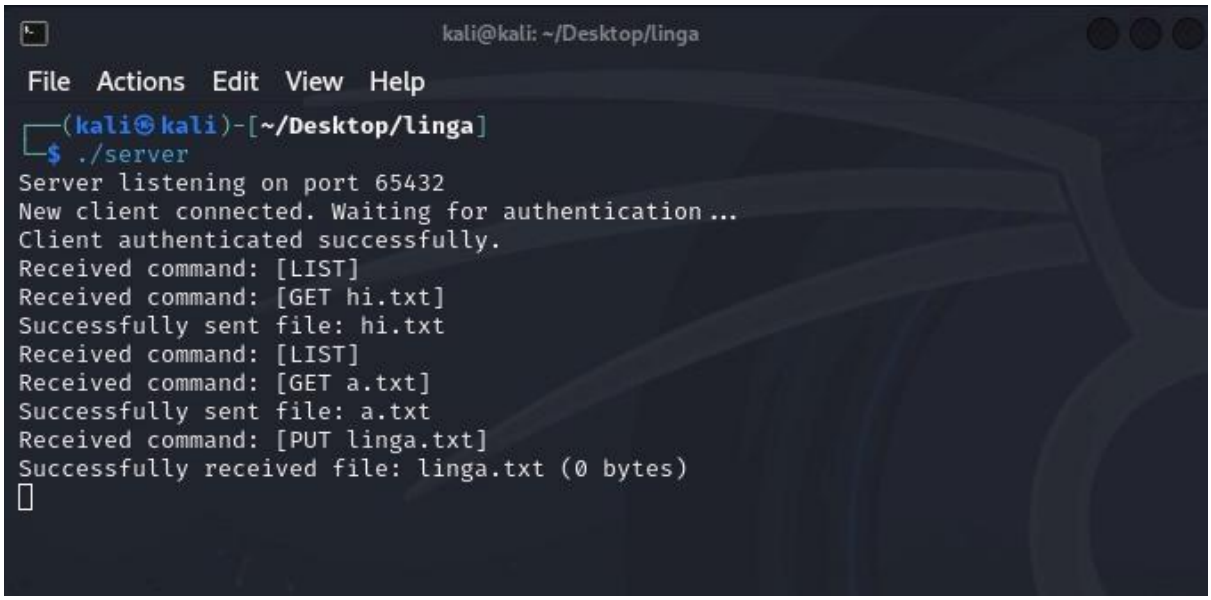
```
        list = "ERROR: Could not open shared directory.";
    }
    return list;
}
```

Output Screenshots & Explanation

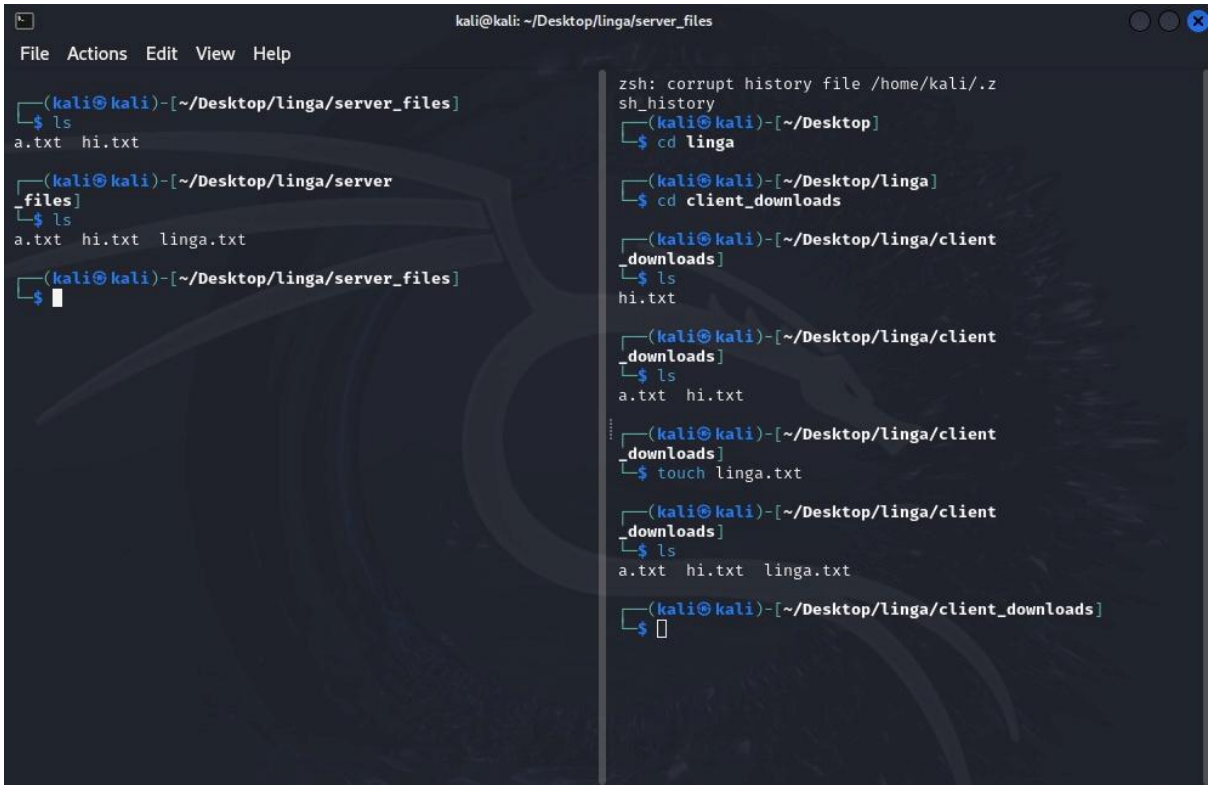


- Created a text file named hi.txt on the client side.
- Wrote a C++ program for client–server communication using sockets.
- The client connected to the server through IP address and port number.
- The client sent the hi.txt file to the server.

- The server received the file data successfully and saved it.
- This confirms successful file transfer from client to server. ✅



- The image displays a server-side log, showing the sequence of events as a client interacts with the server.
- Server Initialization: The server starts and begins "listening on port 65432."
- Client Connection: A "New client connected" and was "authenticated successfully."
- Command Processing: The server receives and processes several commands from the client:
- [LIST]: The client requests a list of files.
- [GET hi.txt]: The client requests to download the file hi.txt, which the server "Successfully sent."
- [LIST]: The client requests another file list.
- [GET a.txt]: The client requests to download the file a.txt, which the server "Successfully sent."
- [PUT linga.txt]: The client attempts to upload a file named linga.txt, which the server "Successfully received" as a file of "0 bytes."



- The user's description of a client-server system where a downloaded file can be accessed and sent back to the server is a conceptual model of a file transfer process. The provided image shows a practical example of file management in a Linux environment, which is often used for client-server interactions.
- The image demonstrates the use of a command-line interface (CLI) in Kali Linux to manage files in two directories, one for the server (server_files) and one for the client (client_downloads). The commands shown, such as ls (list files) and touch (create a new file), are fundamental to file system management in Linux.

EACH STEPS(DAYS) EXPLANATION

Day 1 – Server–Client Communication Setup:

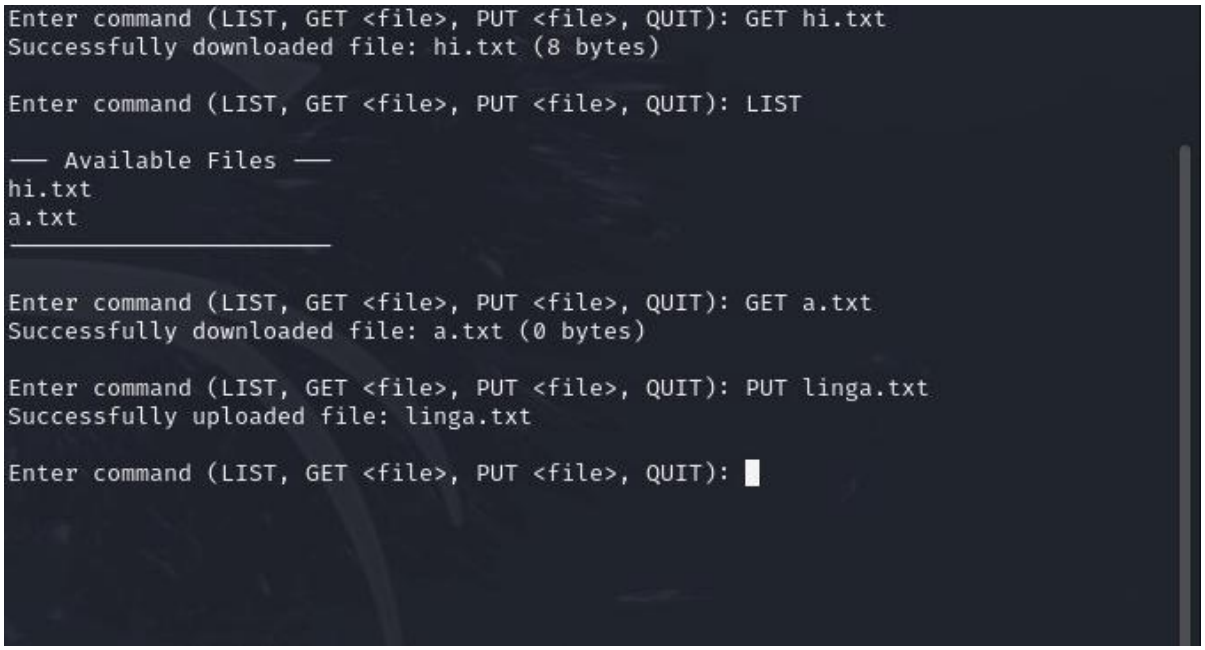
We began by creating a server and a client program using C++ socket programming.
The server was set up to listen on a specific port, and the client connected to it using the server's IP address.
After compiling and running both programs, a successful connection message confirmed that the basic communication link was established between them.

Day 2 – File Listing and Selection:

Next, the server was modified to list all available files in its shared folder.
When the client sends a request (like "LIST"), the server responds with the names of files stored in its directory.

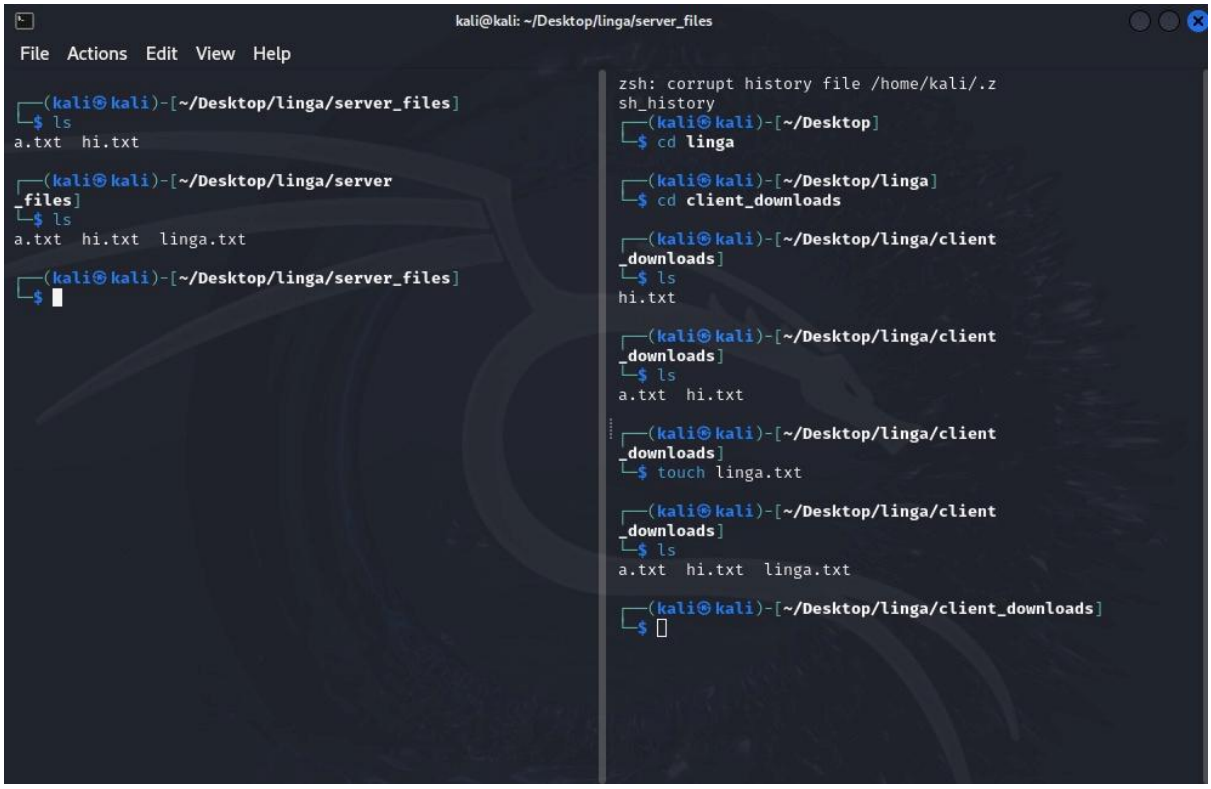
The client displays these files, allowing the user to select a specific file they want to download from the server.

Day 3 – File Transfer (Server to Client):



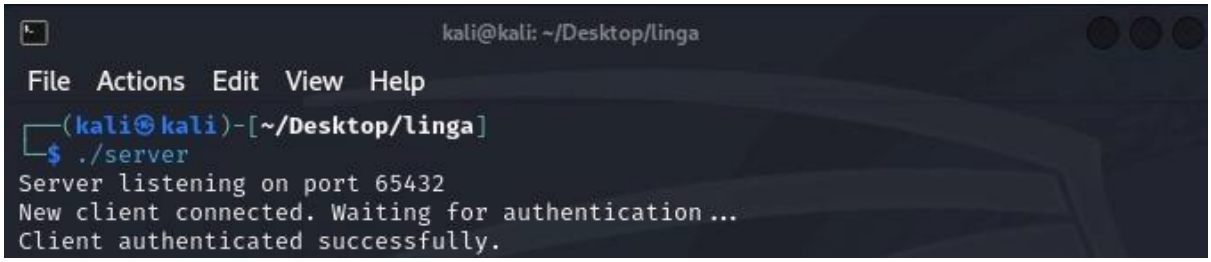
After file selection, the client sends a “DOWNLOAD” command to the server.
The server reads the requested file (for example, hi.txt) and sends its data through the socket connection.
The client receives the file data and saves it locally, confirming a successful file transfer from the server to the client.

Day 4 – File Upload (Client to Server):



In this step, the system was enhanced to allow file uploading.
The client can now choose a file from its local system and send it to the server.
The server receives the file and stores it in a designated upload directory, completing two-way file sharing.

Day 5 – Security Features (Authentication & Encryption):



Finally, security mechanisms were implemented.
The server checks user credentials (username and password) before allowing any file operation.
Additionally, a simple encryption method was added to ensure data transferred between client and server is protected.
This makes the application more secure and reliable.

CONCLUSION

This project successfully demonstrates network-based file sharing using socket programming in C++.
It includes all major functionalities such as connection establishment, file transfer (both upload and download), and security through authentication and encryption.
The project helped in understanding the concepts of networking, socket programming, and data communication in real-time applications.

Submitted By

Name : **Lingaraj Senapati**

Batch : **05**

Department : **BTECH(CSE)**

Registration Number : **2241019368**

Section : **42**