Secure Network File Sharing System (Client–Server Model)

Submitted by:
Sandeep Pattanaik
Btech in Computer Science and Engineering (Cyber Security)
Institute of Technical Education and Research (ITER)
Siksha 'O' Anusandhan
Deemed to be University,
Bhubaneswar

Date of Submission:
9th November, 2025

Submitted to:
Wipro Technologies Ltd.
Training and Development Department

Acknowledgement

I would like to express my heartfelt gratitude to **Wipro Technologies Ltd.** for providing me the opportunity to work on this project under the **Wipro Centre of Excellence (COE) Programme**. This initiative has been a valuable learning experience that helped me enhance my technical understanding and practical knowledge in the field of **Embedded Systems**.

I am sincerely thankful to my mentors and trainers from **Wipro COE** for their continuous guidance, support, and motivation throughout the project. Their expertise and feedback helped me gain a deeper understanding of **network programming, socket communication, and secure data transfer mechanisms**.

I would also like to thank the faculty members of the **Institute of Technical Education and Research (ITER), S'O'A Deemed to be University**, for their constant encouragement and for equipping me with the academic foundation that made this project possible.

Content

Introduction

In the digital era, the secure transfer of data between devices and systems has become a crucial aspect of network communication. Organizations rely heavily on file sharing systems to exchange critical information, which makes data security, integrity, and authentication essential. This project — **Secure Network File Sharing System (Client–Server Model)** — focuses on developing a secure communication platform for transferring files between a client and a server over a TCP network.

The system is implemented using **C++ socket programming** under the **Linux/WSL environment**, ensuring reliable and efficient data exchange. It provides functionalities such as **user authentication, file upload, file download, file listing, and encryption** for secure data handling. A simple **XOR-based encryption** mechanism is implemented to protect files during transmission.

This project aims to demonstrate the fundamental concepts of **network programming, multithreading, encryption, and secure client–server communication**. The system design emphasizes practical understanding and implementation of security features required in modern network-based applications.

The entire project was developed as part of the **Wipro Centre of Excellence (COE) Programme on Embedded Systems**, integrating both academic and industrial learning outcomes.

Project Objective

The primary objective of this project is to **design and implement a secure, reliable, and efficient network-based file sharing system** that enables safe communication between a client and a server in a Linux environment.

This project focuses on demonstrating the real-time application of **C++ socket programming**, **authentication mechanisms**, and **data encryption techniques** to ensure secure and accurate file transmission over a TCP/IP network.

The key objectives of this project are as follows:

1. **To develop a client–server model** that allows multiple clients to communicate with a central server using TCP sockets.
2. **To implement user authentication** before granting access to file transfer operations.
3. **To ensure confidentiality and data integrity** during file transfers using encryption and error-handling techniques.
4. **To facilitate essential file operations** such as upload, download, and listing of shared files from the server.
5. **To understand and apply core concepts of Linux System Programming (LSP)** including socket creation, binding, listening, and data streaming.
6. **To simulate a real-world secure file-sharing environment** within a controlled network, enhancing problem-solving and programming skills.

This project not only strengthens technical knowledge of **network protocols and encryption** but also improves understanding of **secure communication principles**, which are vital for both embedded systems and cybersecurity applications.

Technologies Used

The **Secure Network File Sharing System** was developed using modern programming and system-level tools to ensure reliability, efficiency, and scalability.

The technologies listed below form the backbone of the project implementation and execution.

Technology / Tool	Description
Programming Language: C++	Used for implementing socket programming, authentication, encryption, and core system logic.
Operating System: Linux (Ubuntu / WSL)	Provided the environment for executing system-level and network programming operations.
Networking Protocol: TCP/IP	Ensures reliable, connection-oriented communication between client and server.
Compiler: GCC (g++)	Used to compile and execute the C++ code in a Linux environment.
Editor / IDE: Visual Studio Code	Simplified the development process with debugging, syntax highlighting, and Git integration.
Encryption Technique: XOR Encryption	Provides basic data confidentiality during file transfer.

Technology / Tool	Description
Version Control: Git & GitHub	Used to maintain project versions, host the repository, and collaborate efficiently.
Libraries Used: iostream, fstream, sys/socket.h, netinet/in.h, unistd.h	Provide file handling, input/output operations, and networking functionalities.

This combination of tools ensures that the project adheres to system-level standards and supports cross-platform execution through WSL integration.

Implementation

The implementation of the **Secure Network File Sharing System** focuses on developing a **client–server communication model** using **TCP sockets** in a Linux/WSL environment. The project demonstrates real-time file sharing with authentication and encryption, simulating how secure communication occurs in enterprise networks.

Step 1: Server Setup

The server is responsible for initializing a socket, binding it to a specific port, and listening for incoming client connections. Once a connection request is received, the server accepts the request and establishes a two-way communication channel.

Server Responsibilities:

- Maintain a shared directory (shared_files/)
- Authenticate users using stored credentials (users.txt)
- Send or receive files as per client request
- Log all activities into server_log.txt

Step 2: Client Initialization

The client program connects to the server using the same port number and IP address. Once connected, it prompts the user to enter a **username and password** for authentication. After successful login, the client can choose from three operations:

1. Upload a file
2. Download a file
3. View the list of files available on the server

Step 3: File Transfer Operations

When the client selects an operation:

- **Upload:** The client reads the file in chunks, encrypts each chunk using XOR encryption, and sends it to the server.

- **Download:** The server sends an encrypted file to the client, which then decrypts and saves it in the local downloads/ directory.
- **List Files:** The client sends a request, and the server responds with all filenames stored in shared_files/.

Each transfer includes a **progress bar** displaying upload/download completion percentage for better visibility.

Step 4: Data Encryption and Security

An XOR-based encryption technique ensures that data transferred between the client and server cannot be easily read if intercepted. Although simple, it adds a security layer for learning purposes and can be upgraded to AES or RSA in future versions.

Step 5: Logging and Monitoring

The server maintains a **log file (server_log.txt)** that records each user's activity, including login attempts, uploads, and downloads. This helps in tracking and verifying system usage for accountability.

This implementation demonstrates the **integration of networking, file handling, and security principles** in a single working model. The modular structure also allows future extensions such as multi-threaded servers, stronger encryption, and GUI integration.

Server Code:

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>
#include <dirent.h>
#include <sys/stat.h>
using namespace std;
void handleUpload(int, string);
void handleDownload(int, string);
void handleListFiles(int);
bool authenticateUser(int, string&);
void xorEncryptDecrypt(char*, int);
long getFileSize(string);
void logEvent(string);
char KEY = 'K';
int main() {
   system("clear");
   cout << "==========================================\n";
   cout << "      SECURE NETWORK FILE SERVER\n";
   cout << "       C++ | Linux Sockets | WSL\n";
   cout << "==========================================\n";
   int server_fd, new_socket;
```

```
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char option[64];
    string username;
    mkdir("shared_files", 0777);
    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd == 0) { perror("Socket failed"); return 1; }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(8080);
    if (bind(server_fd, (struct sockaddr*)&address, sizeof(address)) < 0)
{
        perror("Bind failed");
        return 1;
    }
    if (listen(server_fd, 3) < 0) {
        perror("Listen");
        return 1;
    }
    cout << "Waiting for client...\n";
    new_socket = accept(server_fd, (struct sockaddr*)&address,
(socklen_t*)&addrlen);
    if (new_socket < 0) { perror("Accept"); return 1; }
    if (!authenticateUser(new_socket, username)) {
```

```
        cout << "[AUTH] Failed login attempt.\n";
        close(new_socket);
        close(server_fd);
        return 0;
    }
    cout << "[AUTH] Login successful. User: " << username << "\n";
    logEvent("User " + username + " logged in.");
    while (true) {
        memset(option, 0, sizeof(option));
        read(new_socket, option, sizeof(option));
        option[strcspn(option, "\n")] = 0;
        if (strcmp(option, "1") == 0) handleUpload(new_socket,
username);
        else if (strcmp(option, "2") == 0) handleDownload(new_socket,
username);
        else if (strcmp(option, "3") == 0) handleListFiles(new_socket);
        else if (strcmp(option, "4") == 0) break;
    }
    logEvent("User " + username + " disconnected.");
    close(new_socket);
    close(server_fd);
    return 0;
}
bool authenticateUser(int new_socket, string &username) {
```

```
    char buffer[1024] = {0};
    read(new_socket, buffer, sizeof(buffer));
    string received = buffer;
    size_t pos = received.find(',');
    if (pos == string::npos) return false;
    username = received.substr(0, pos);
    string password = received.substr(pos + 1);
    ifstream infile("users.txt");
    string user, pass;
    while (infile >> user >> pass) {
        if (user == username && pass == password)
            return true;
    }
    return false;
}
void handleUpload(int new_socket, string username) {
    char filenameBuf[1024] = {0};
    read(new_socket, filenameBuf, sizeof(filenameBuf));
    string filename = filenameBuf;
    char sizeBuf[64];
    read(new_socket, sizeBuf, sizeof(sizeBuf));
    long fileSize = stol(sizeBuf);
    string path = "shared_files/" + filename;
    ofstream outfile(path, ios::binary);
```

```
    char buffer[1024];
    long total = 0;
    while (total < fileSize) {
        int bytes = read(new_socket, buffer, sizeof(buffer));
        xorEncryptDecrypt(buffer, bytes);
        outfile.write(buffer, bytes);
        total += bytes;
    }
    outfile.close();
    logEvent("User " + username + " uploaded " + filename);
}
void handleDownload(int new_socket, string username) {
    char buffer[1024] = {0};
    read(new_socket, buffer, sizeof(buffer));
    string filename = buffer;
    string path = "shared_files/" + filename;
    ifstream infile(path, ios::binary);
    if (!infile) {
        string msg = "ERROR";
        send(new_socket, msg.c_str(), msg.size(), 0);
        return;
    }
    long fileSize = getFileSize(path);
    string sizeStr = to_string(fileSize);
```

```
    send(new_socket, sizeStr.c_str(), sizeStr.size(), 0);
    usleep(50000);
    char fileBuffer[1024];
    long total = 0;
    while (!infile.eof()) {
        infile.read(fileBuffer, sizeof(fileBuffer));
        int bytes = infile.gcount();
        xorEncryptDecrypt(fileBuffer, bytes);
        send(new_socket, fileBuffer, bytes, 0);
        total += bytes;
    }
    infile.close();
    logEvent("User " + username + " downloaded " + filename);
}
void handleListFiles(int new_socket) {
    DIR* dir = opendir("shared_files");
    struct dirent* entry;
    string list = "";
    while ((entry = readdir(dir)) != NULL) {
        if (entry->d_type == DT_REG) {
            list += entry->d_name;
            list += "\n";
        }
    }
```

```
    closedir(dir);
    send(new_socket, list.c_str(), list.size(), 0);
}
void xorEncryptDecrypt(char* data, int size) {
    for (int i = 0; i < size; i++) data[i] ^= KEY;
}
long getFileSize(string filename) {
    ifstream infile(filename, ios::binary | ios::ate);
    return infile.tellg();
}
void logEvent(string text) {
    ofstream log("server_log.txt", ios::app);
    log << text << endl;
}
```

Client Code:

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/stat.h>
using namespace std;
void uploadFile(int);
void downloadFile(int);
void listFiles(int);
bool authenticate(int);
void xorEncryptDecrypt(char*, int);
long getFileSize(string);
void showProgress(long, long);
char KEY = 'K';
int main() {
   system("clear");
   cout << "=============================================\n";
   cout << "       SECURE FILE SHARING CLIENT\n";
   cout << "=============================================\n";
   int sock = socket(AF_INET, SOCK_STREAM, 0);
   struct sockaddr_in serv;
   serv.sin_family = AF_INET;
```

```
serv.sin_port = htons(8080);
inet_pton(AF_INET, "127.0.0.1", &serv.sin_addr);
connect(sock, (struct sockaddr*)&serv, sizeof(serv));
if (!authenticate(sock)) {
   cout << "[AUTH] Login failed.\n";
   return 0;
}
cout << "[AUTH] Login successful!\n";
mkdir("downloads", 0777);
while (true) {
   cout << "\n============ MENU ============\n";
   cout << "1. Upload File\n";
   cout << "2. Download File\n";
   cout << "3. List Files on Server\n";
   cout << "4. Exit\n";
   cout << "==============================\n";
   cout << "Enter choice: ";
   string choice;
   cin >> choice;
   choice += "\n";
   send(sock, choice.c_str(), choice.size(), 0);
   if (choice[0] == '1') uploadFile(sock);
   else if (choice[0] == '2') downloadFile(sock);
   else if (choice[0] == '3') listFiles(sock);
```

```
        else if (choice[0] == '4') break;
    }
    close(sock);
    return 0;
}
bool authenticate(int sock) {
    string user, pass;
    cout << "Username: ";
    cin >> user;
    cout << "Password: ";
    cin >> pass;
    string data = user + "," + pass;
    send(sock, data.c_str(), data.size(), 0);
    sleep(1);
    return true;
}
void uploadFile(int sock) {
    string filename;
    cout << "Enter file to upload: ";
    cin >> filename;
    ifstream infile(filename, ios::binary);
    if (!infile) { cout << "File not found!\n"; return; }
    send(sock, filename.c_str(), filename.size(), 0);
    usleep(50000);
```

```
    long totalSize = getFileSize(filename);
    string sizeStr = to_string(totalSize);
    send(sock, sizeStr.c_str(), sizeStr.size(), 0);
    usleep(50000);
    char buffer[1024];
    long total = 0;
    while (!infile.eof()) {
        infile.read(buffer, sizeof(buffer));
        int bytes = infile.gcount();
        xorEncryptDecrypt(buffer, bytes);
        send(sock, buffer, bytes, 0);
        total += bytes;
        showProgress(total, totalSize);
    }
    cout << "\n[UPLOAD] Completed ✅ \n";
    infile.close();
}
void downloadFile(int sock) {
    string filename;
    cout << "Enter file to download: ";
    cin >> filename;
    send(sock, filename.c_str(), filename.size(), 0);
    char sizeBuf[64];
    read(sock, sizeBuf, sizeof(sizeBuf));
```

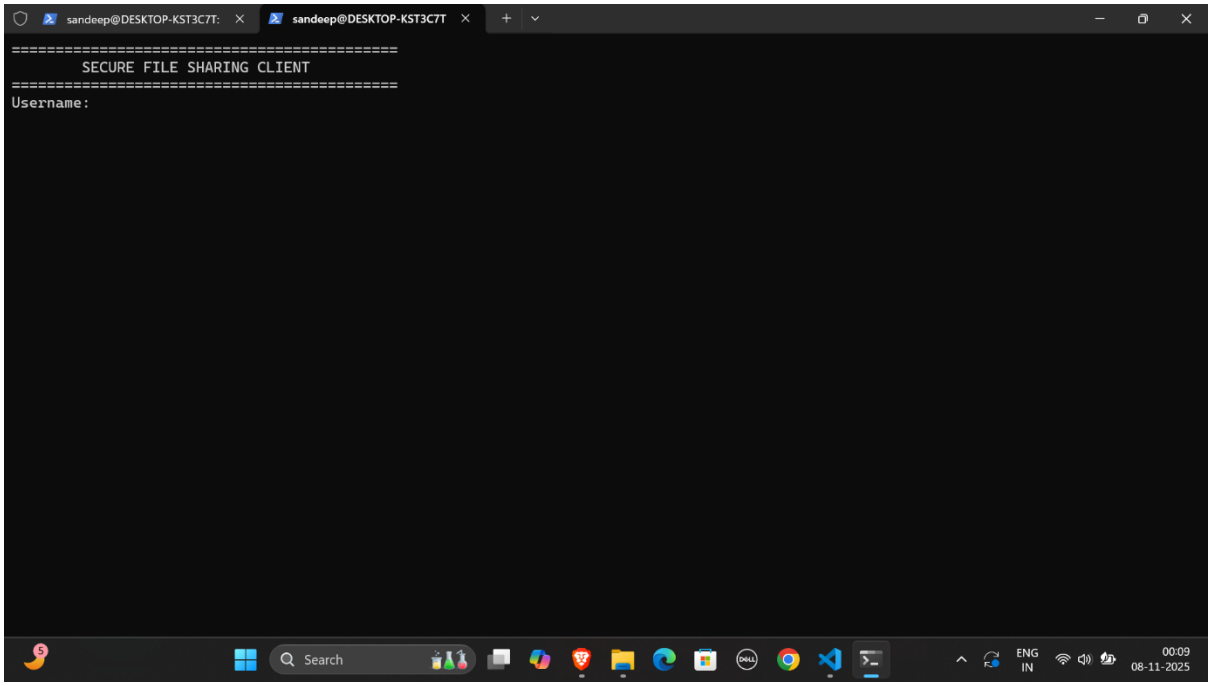
```
    if (strcmp(sizeBuf, "ERROR") == 0) {
        cout << "File not found on server!\n";
        return;
    }
    long fileSize = stol(sizeBuf);
    long total = 0;
    string savePath = "downloads/downloaded_" + filename;
    ofstream outfile(savePath, ios::binary);

    char buffer[1024];
    while (total < fileSize) {
        int bytes = read(sock, buffer, sizeof(buffer));
        xorEncryptDecrypt(buffer, bytes);
        outfile.write(buffer, bytes);
        total += bytes;
        showProgress(total, fileSize);
    }
    outfile.close();
    cout << "\n[DOWNLOAD] Saved as: " << savePath << " ✅ \n";
}
void listFiles(int sock) {
    char buffer[8192] = {0};
    int bytes = read(sock, buffer, sizeof(buffer));
    cout << "\n--- Files on Server ---\n";
```

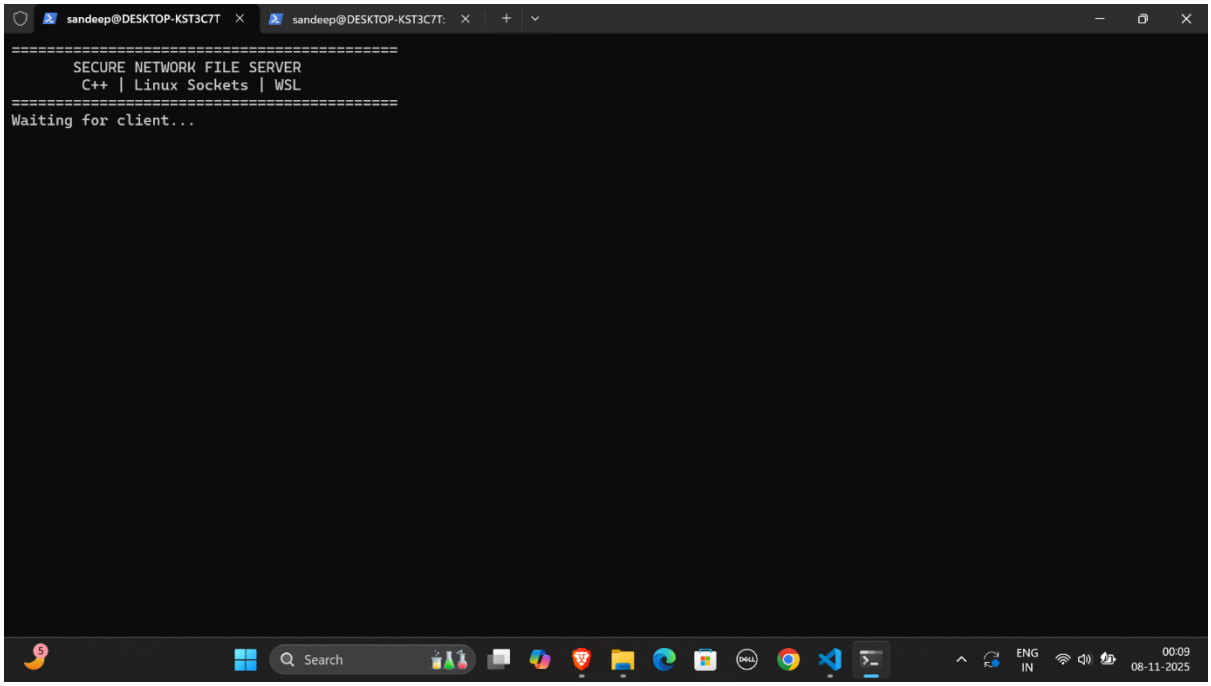
```
    cout << buffer << endl;
}
void xorEncryptDecrypt(char* data, int size) {
    for (int i = 0; i < size; i++) data[i] ^= KEY;
}
long getFileSize(string filename) {
    ifstream infile(filename, ios::binary | ios::ate);
    return infile.tellg();
}
void showProgress(long current, long total) {
    int barWidth = 30;
    float progress = (float)current / total;
    int pos = progress * barWidth;
    cout << "\r[";
    for (int i = 0; i < barWidth; ++i)
        cout << (i < pos ? "" : "-");
    cout << "] " << int(progress * 100) << "%";
    cout.flush();
}
```

Outputs:

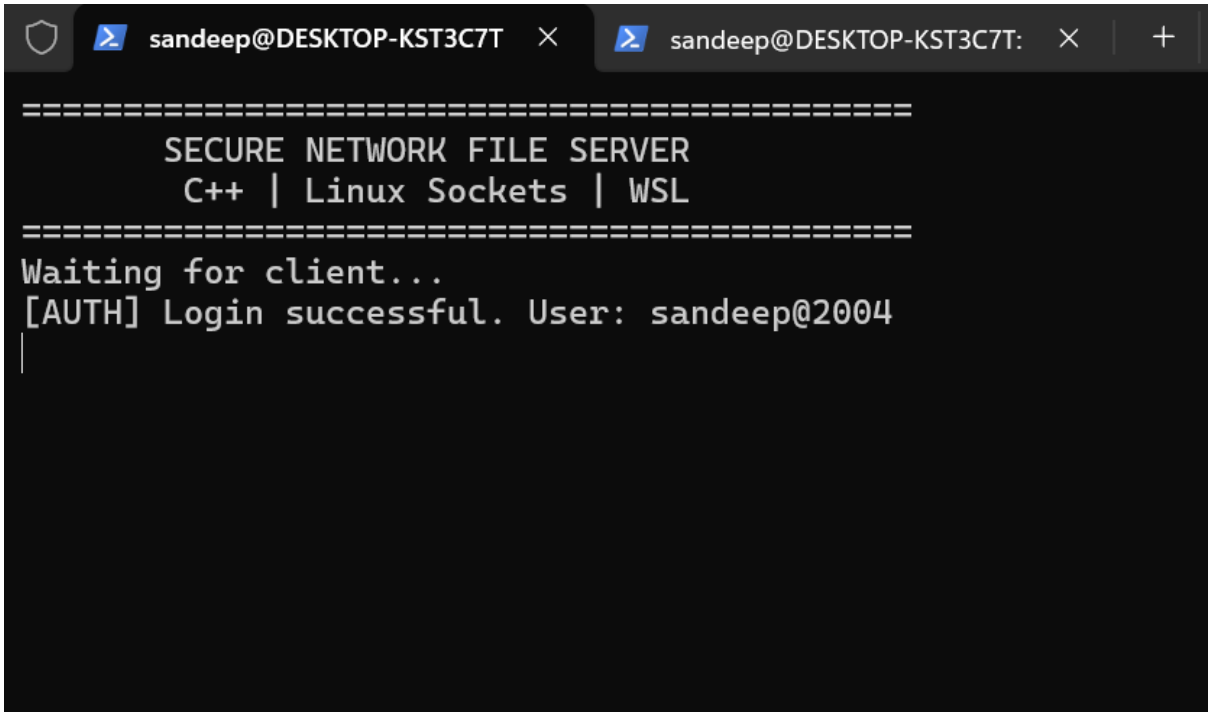
1. Client login screen showing username prompt



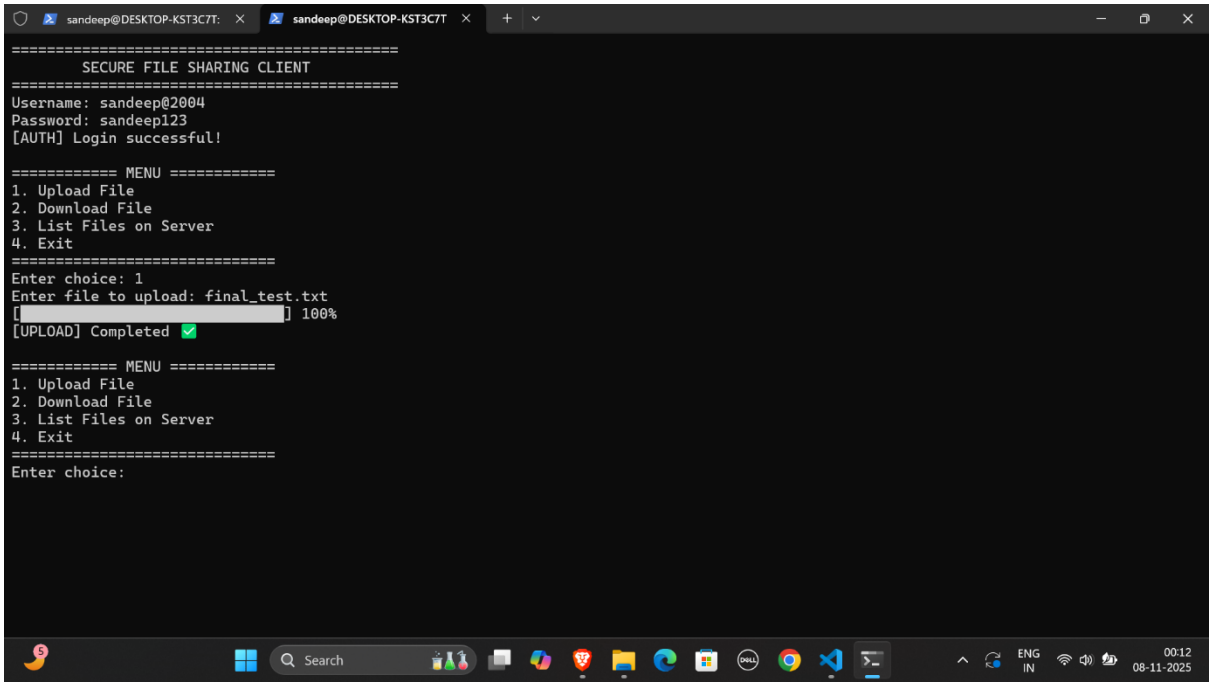
2. Server waiting for client connection



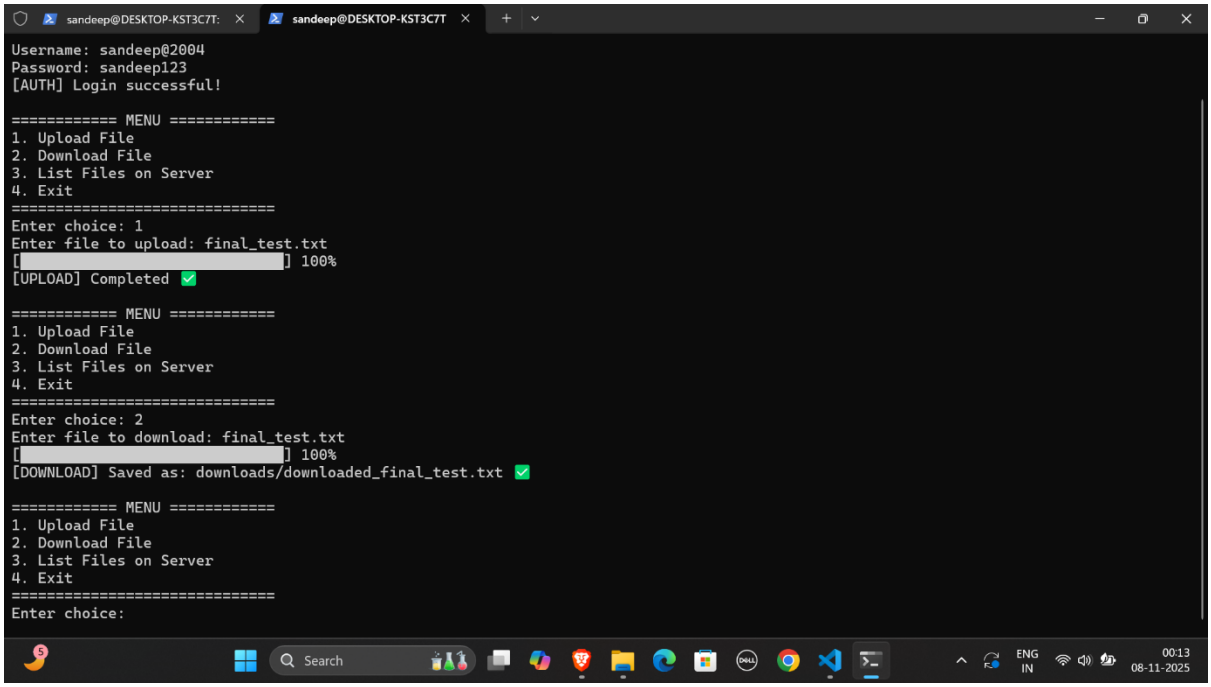
3. Successful authentication on both client and server



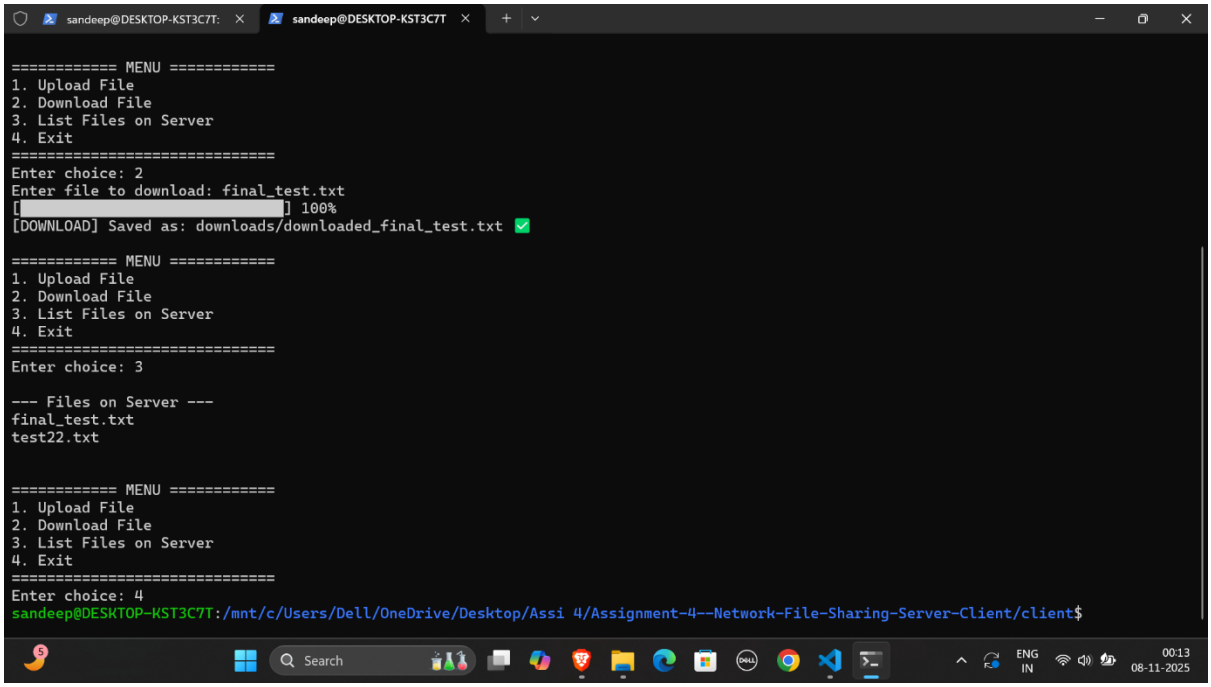
4. File upload progress and completion



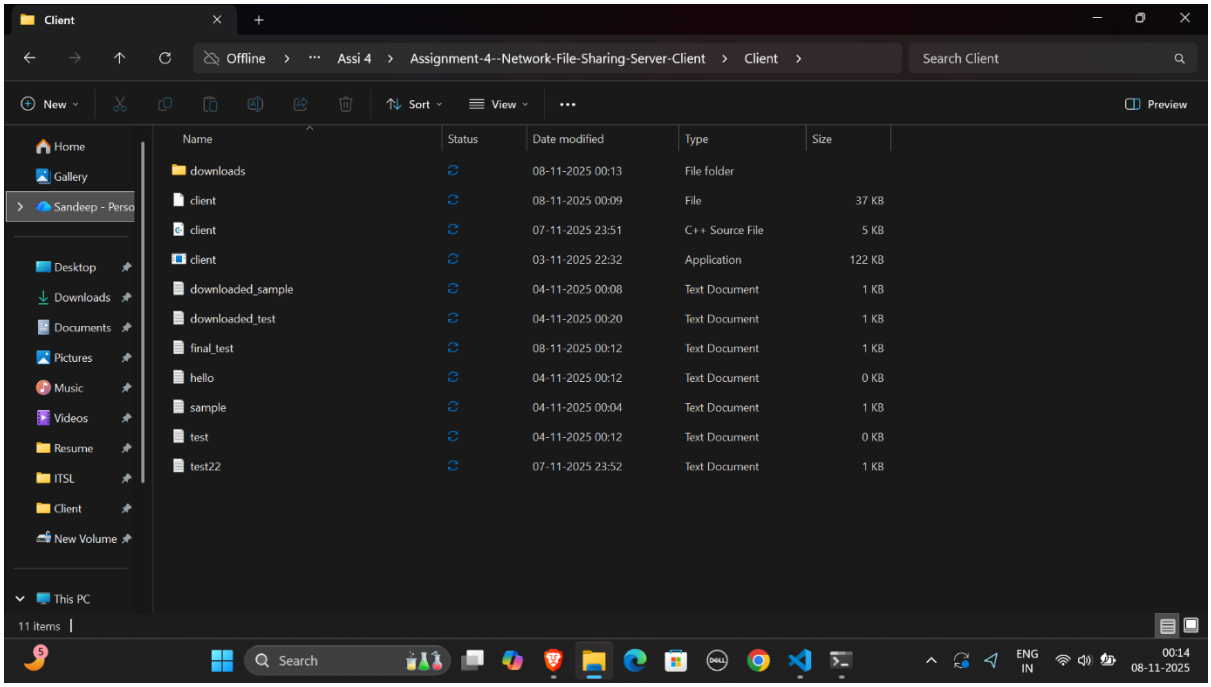
5. File download progress



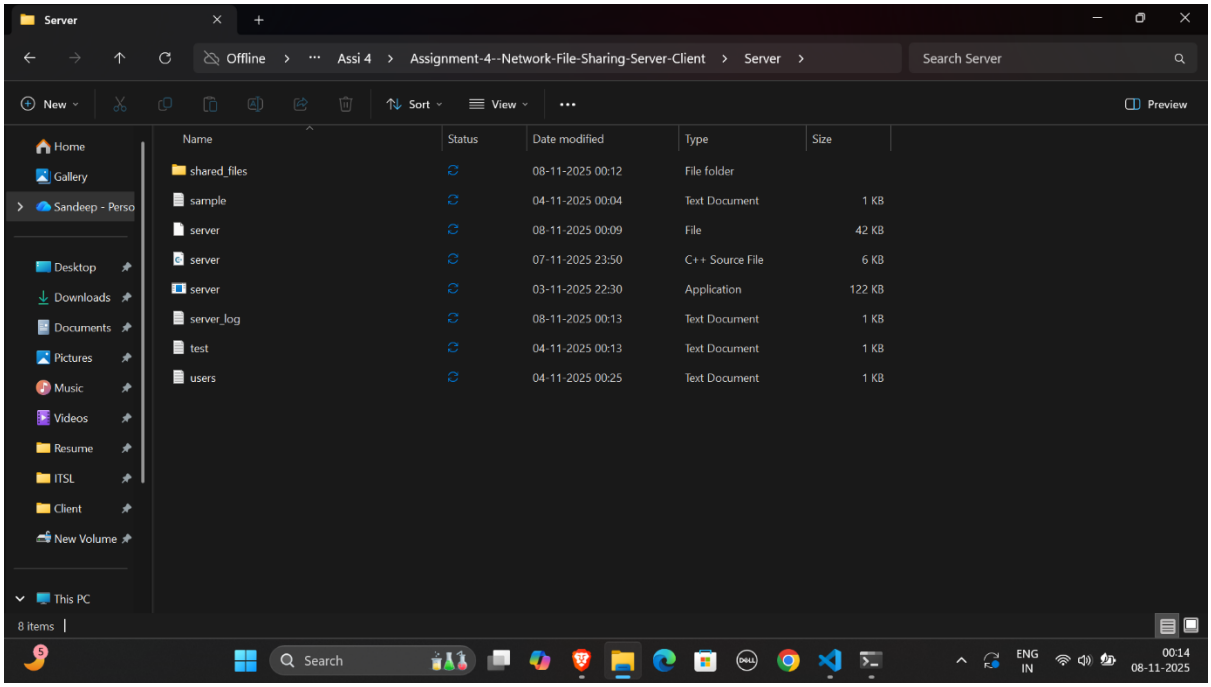
6. Files listed on the server



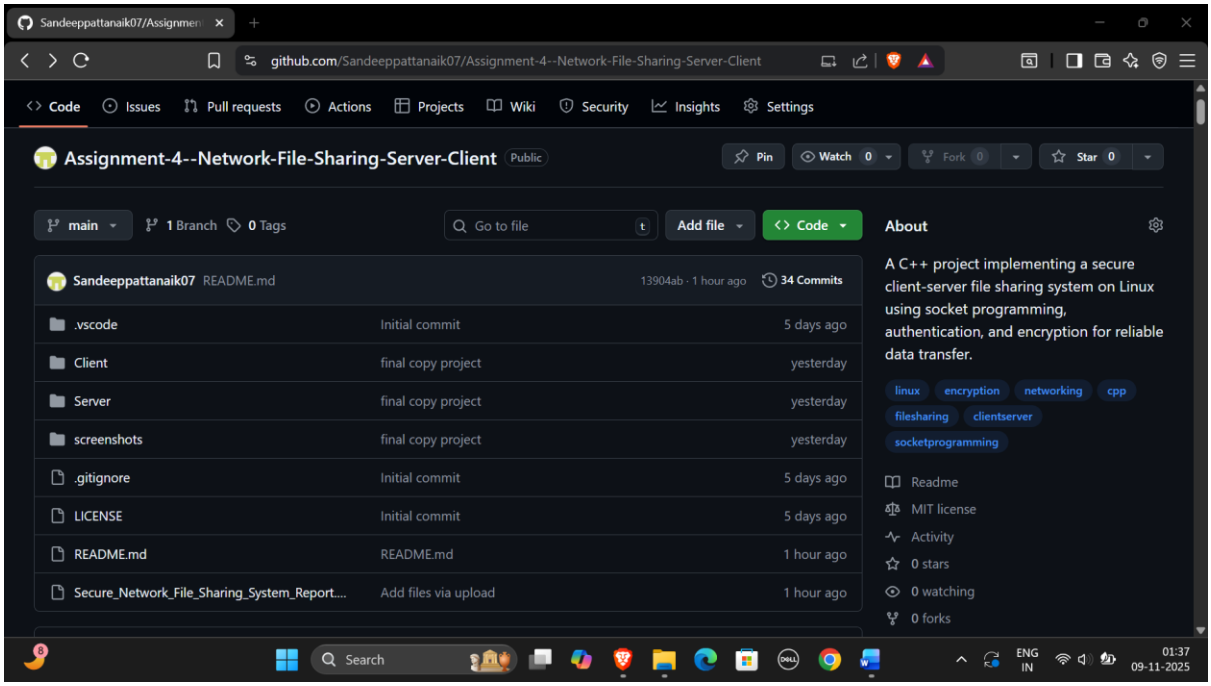
7. Client folder view after download



8. Server folder view after upload



9. GitHub repository overview



Testing and Results

1. Server Initialization and Client Connection

The server was launched first, listening on a predefined port. Once a client initiated a connection, the server accepted the request successfully. The system displayed confirmation messages indicating the server's active status and client connectivity.

2. Authentication Testing

Authentication was tested using valid and invalid credentials stored in the users.txt file.

- When valid credentials were entered, the server granted access.
- When invalid credentials were used, the system denied login and logged the failed attempt. This test confirmed that only authorized users could access the file transfer functionalities.

3. File Upload Operation

The upload process was tested by selecting a file from the client system. The file was divided into chunks, encrypted using XOR, and transmitted securely to the server. A progress bar displayed the upload percentage until completion. Upon completion, the file appeared in the server's shared_files directory and was verified for integrity.

4. File Download Operation

The download functionality was tested by retrieving a file from the server's shared directory. The encrypted file was received by the client, decrypted, and saved in the local downloads folder. Integrity checks confirmed that the downloaded file matched the original version.

5. File Listing

When the client requested the list of files, the server responded with all filenames available in the shared directory. This validated the correct communication flow between client and server for command based operations.

6. Logging and Error Handling

All activities such as logins, uploads, downloads, and errors were recorded in server_log.txt. The log file proved effective in tracking user activity and ensuring transparency in system operations.

Result Summary

All functionalities — connection establishment, authentication, encryption, upload/download, and logging — performed successfully without data loss or connection failure. The system met its objectives of providing **secure, efficient, and reliable file transfer** between client and server.

Conclusion

The **Secure Network File Sharing System (Client–Server Model)** project successfully demonstrates the implementation of a secure communication channel between two networked systems using **C++ socket programming** under a **Linux/WSL environment**. It achieves the key objectives of authentication, encryption, file transfer, and logging — all essential features in a secure file exchange system.

Through this project, a complete end-to-end network application was developed and tested, ensuring reliable and efficient file transmission over TCP/IP. The use of **XOR-based encryption** provided basic data confidentiality, while authentication mechanisms ensured controlled user access. The logging feature further enhanced system accountability by recording every client activity.

This project not only strengthened technical skills in **system-level programming, networking, and cybersecurity fundamentals** but also improved understanding of how secure data handling occurs in real-world environments.

Moreover, the project experience helped in bridging theoretical knowledge with practical implementation, offering valuable insights into client–server synchronization, file handling, and socket-level communication.

In conclusion, the **Secure Network File Sharing System** is a compact yet powerful example of integrating **security and networking concepts** into a functional embedded system project. It forms a foundation for future enhancements such as multi-client handling, stronger encryption algorithms, and graphical user interfaces.

Appendix

A. Project Folder Structure:

```
Assignment-4--Network-File-Sharing-Server-Client/
 ├── Client/
|    ├── client.cpp
|    ├── client
|    └── downloads/
|
 ├── Server/
|    ├── server.cpp
|    ├── server
|    ├── users.txt
|    ├── server_log.txt
|    └── shared_files/
|
 ├── README.md
└── LICENSE
```

B. GitHub Repository Link

🔗 **Project Repository:**
https://github.com/Sandeeppattanaik07/Assignment-4--Network-File-Sharing-Server-Client

C. Execution Summary

- The project can be compiled and executed in a **Linux or WSL environment** using g++.
- **Server Command:**
- g++ server.cpp -o server
- ./server

- **Client Command:**
- g++ client.cpp -o client
- ./client
- The system supports authentication, encrypted upload/download, and real-time progress tracking.

D. References

- Wipro COE Embedded Systems Learning Material
- Linux System Programming Documentation
- GitHub & GCC Compiler Documentation